

Probabilistically Survivable MASs *

Sarit Kraus

Dept. of Computer Science
Bar-Ilan University
Ramat-Gan, 52900 Israel
sarit@cs.biu.ac.il

V.S. Subrahmanian and N. Cihan Tas

Dept. of Computer Science
University of Maryland
College Park, MD 20742
{vs,ctas}@cs.umd.edu

Abstract

Multiagent systems (MAS) can “go down” for a large number of reasons, ranging from system malfunctions and power failures to malicious attacks. The placement of agents on nodes is called a *deployment* of the MAS. We develop a probabilistic model of survivability of a deployed MAS and provide two algorithms to compute the probability of survival of a deployed MAS. Our probabilistic model does not make independence assumptions though such assumptions can be added if so desired. An optimal deployment of a MAS is one that maximizes its survival probability. We provide a mathematical answer to this question, an algorithm that computes an exact solution to this problem, as well as several algorithms that quickly compute approximate solutions to the problem. We have implemented our algorithms - our implementation demonstrates that computing deployments can be done scalably.

1 Introduction

As multiagent systems (MASs) are increasing used for critical applications, the ability of these MASs to survive intact when various external events occur (e.g. power failures, OS crashes, etc.) becomes increasingly important. However, one never knows when and if a system will crash or be compromised, and hence, any model of MAS survivability must take this uncertainty into account.

We provide for the first time, a formal model for reasoning about survivability of MASs which includes both a declarative theory of survivability, as well as implemented algorithms to compute optimal ways of deploying MASs across a network.

A MAS-deployment specifies a placement of agents on various network nodes. Based on probabilistic information about the survivability of a given node, we develop a formal

theory describing the probability that a given deployment will survive. This probability reflects the best guarantee we have of the MAS surviving. Our model does not assume that node failures are independent, though independence information can be easily added if so desired. The technical problem we need to grapple with is that of finding a MAS-deployment of the agents having the highest probability of survival. As we do not make unrealistic independence assumptions, this problem turns out to be intractable. As a consequence, heuristics are required to find a deployment (even if it is sub-optimal). We develop algorithms for the following tasks:

1. Given a MAS-deployment, how do we compute its probability of survival?
2. Find a MAS-deployment with the highest probability of survival - this algorithm is infeasible to implement in practice due to the above mentioned complexity results.
3. We develop a suite of heuristic algorithms to find (sub-optimal) MAS-deployments.

We have conducted detailed experiments with our algorithms - for space reasons, only some of them are described here. The experiments show that our heuristic algorithms can find deployments very fast.

2 Preliminaries

Agents. The only assumptions we make about agents is that they provide one or more services. We further assume that all host computers on which agents are located have a finite amount of memory resources, and that each (copy of an) agent requires some amount of memory, denoted by $\text{mem}(a)$. A *multiagent application* MAS is a finite set of agents - we make the assumption that all agents in a multiagent application are needed for it to function.

Networks. A *network* is a triple $(\mathcal{N}, \text{edges}, \text{mem})$ where $\mathcal{N}$ is a set of called *nodes*, $\text{edges} \subseteq \mathcal{N} \times \mathcal{N}$ specifies which nodes can communicate with which other nodes, and $\text{mem} : \mathcal{N} \rightarrow \mathbb{R}$ specifies the total memory¹ available at node n for use by agents situated at n . A network is *fully connected* iff $\text{edges} = \mathcal{N} \times \mathcal{N}$.

*The first author is also affiliated with UMIACS. This work was supported in part by the Army Research Lab under contract DAAL 0197K0135, the CTA on Advanced Decision Architectures, by ARO contract DAAD190010484, by NSF grant IIS0222914 and an NSF ITR award 0205489.

¹Note that the symbol mem is used to both denote the memory requirements of an agent, as well as the memory available at a node. It is easy to determine the intended meaning of this expression from context.

Definition 2.1 Suppose MAS is a multiagent application and $Ne = (\mathcal{N}, \text{edges}, \text{mem})$ is a network. A deployment for MAS on Ne is a mapping $\mu : \mathcal{N} \rightarrow 2^{\text{MAS}}$ specifying which agents are located at a given node. (As usual, if X is a set, 2^X is the power set of X). μ must satisfy the following condition:

- $(\forall a \in \text{MAS})(\exists n \in \mathcal{N}) a \in \mu(n)$. This condition says that every agent must be deployed somewhere.
- $(\forall n \in \mathcal{N}) \text{mem}(n) \geq \sum_{a \in \mu(n)} \text{mem}(a)$. This condition says that the agents deployed at a node cannot use more memory than that node makes available.

Intuitively, $\mu(n) = \{a_1, a_2\}$ says that agents a_1, a_2 are deployed at node n_1 .

Example 2.1 Suppose $\mathcal{N} = \{n_1, n_2, n_3, n_4\}$ and $\text{MAS} = \{a, b, c, d\}$. An example deployment is given by: $\mu(n_1) = \{a, b\}$, $\mu(n_2) = \{c, d\}$, $\mu(n_3) = \{a, b, c, d\}$ and $\mu(n_4) = \{d\}$. This example will be used throughout this paper.

3 Related Work

To our knowledge, there are no probabilistic models of survivability of a MAS. However, there are many works that are in related areas.

[Shehory *et al.*, 1998] use agent-cloning and agent-merging techniques to mitigate agent over-loading and promote system load balancing. Fan [Fan, 2001] proposes a BDI mechanism to formally model agent cloning to balance agent workload.

[Fedoruk and Deters, 2002] propose transparent agent replication technique - though an agent is represented by multiple copies, this is an internal detail hidden from other agents. Several other frameworks also support this kind of agent fault tolerance [Mishra, 2001].

[Marin *et al.*, 2001] develop adaptive fault tolerance techniques for MASs. They use simulations to assess migration and replication costs. However, [Marin *et al.*, 2001] concludes by saying that they do not address the questions of which of the agents to replicate, how many replicas should be made, where those replicas should be allocated. These questions are addressed in the current paper, but we do not propose a mechanism to synchronize agent replications.

[Kumar *et al.*, 2000] focus on the problem of broker agents that are inaccessible due to system failures. They use the theory of teamwork to specify robust brokered architectures that can recover from broker failure. We, on the other hand, consider the possible failure of any agent in the multi-agent systems.

The problem of network reliability has been studied extensively — [Gartner, 1999] provide an excellent survey. In this paper we build on top of these studies and assume, as discussed below, that there is a disconnect probability function for a network specifying the reliability of each node of the network.

The problem of fault-tolerant software systems has some similarities to our agent survivability problem. An extensive study was performed to solve this problem using the N-Version Problem (NVP) approach. The NVP is defined as the “independent generation of $N \geq 2$ functionally equivalent programs from the same initial specification” [Lyu and He, 1993]. In this approach, the reliability of a software system is

increased by developing several versions of special modules and incorporating them into a fault-tolerant system [Gutjahr, 1998]. However, these works (i) make unnecessary or unwarranted independence assumptions, (ii) provide only a measure of expected survivability rather than guaranteed survivability, (iii) do not consider replication.

4 A Probabilistic Model of Survivability

Multiagent applications can “go down” because nodes on which agents are located can crash. Alternatively, agents are on a mobile node (e.g. a vehicle) may wander beyond communications range, thus dropping out of the network.

Definition 4.1 A disconnect probability function for a network $(\mathcal{N}, \text{edges}, \text{mem})$ is a mapping $\text{dp} : \mathcal{N} \rightarrow C[0, 1]$ where $C[0, 1]$ is the set of all closed subintervals of $[0, 1]$.

Intuitively, if $\text{dp}(N) = [0.2, 0.3]$, then this says that there is a 20 – 30% probability that node N will get disconnected from the network. Note that this model supports the situation where we do not know the probability of node n getting disconnected - in this case, we can set $\text{dp}(N) = [0, 1]$. Likewise, if we know that a node will get disconnected with 80% probability with a $\pm 3\%$ margin of error, then we can set $\text{dp}(n) = [0.77, 0.83]$. One possibility to compute dp in a specific setting is by collecting statistical data on the past failures of each node. This would give us both a mean probability of failure for a given node as well as a standard deviation which would jointly result in a probability interval. In other applications (e.g. where statistics are not available) expert opinions can be used.

Given a network $(\mathcal{N}, \text{edges}, \text{mem})$ and a disconnect probability function dp , there is a space of possible networks that may arise in the future.

Definition 4.2 Suppose $(\mathcal{N}, \text{edges}, \text{mem})$ is a network and $\mathcal{N}' \subseteq \mathcal{N}$. Then $(\mathcal{N}', \text{edges}', \text{mem})$ is a possible future network where $\text{edges}' = \{(n_1, n_2) \mid (n_1, n_2) \in \text{edges} \text{ and } n_1, n_2 \in \mathcal{N}'\}$.

We use $\text{PFN}_{\text{dp}}(\mathcal{N}, \text{edges}, \text{mem})$ to denote the set of all possible future networks associated with a network $(\mathcal{N}, \text{edges}, \text{mem})$ and a disconnect probability function dp . Note that we can infer probabilities of possible future networks from such disconnect probabilities on nodes. Even though many future networks are possible at a given time t , only one of them will in fact occur at time t . So at time t , $\text{PFN}_{\text{dp}}(\mathcal{N}, \text{edges}, \text{mem})$ represents the space of possible network configurations. Given a network $Ne' = (\mathcal{N}', \text{edges}', \text{mem})$ we write $N \in Ne'$ iff $N \in \mathcal{N}'$. Furthermore, since in this paper we do not discuss the failure of edges, for space reasons, we will omit them from the networks in the rest of the paper.

Suppose $\text{prob}(Ne)$ denotes the probability of a possible future network Ne . For any $N \in \mathcal{N}$ we can write the constraint:

$$1 - \text{dp}(N) \cdot \text{UB} \leq \sum_{Ne' \in \text{PFN}_{\text{dp}}(\mathcal{N}, \text{mem}) \wedge N \in Ne'} \text{prob}(Ne') \leq 1 - \text{dp}(N) \cdot \text{LB}.$$

This constraint says that the sum of the probabilities of all future networks in which node N survives must be between $1 - \text{dp}(N) \cdot \text{UB}$ and $1 - \text{dp}(N) \cdot \text{LB}$. We take all such constraints

(one for each node) and add a constraint which says that the only possible future networks are those in $\text{PFN}_{\text{dp}}(\mathcal{N}, \text{mem})$. Last, but not least, we know that the probability of each future network is at least 0. This gives us:

$$\sum_{Ne' \in \text{PFN}_{\text{dp}}(\mathcal{N}, \text{mem})} \text{prob}(Ne') = 1.$$

For any $Ne' \in \text{PFN}_{\text{dp}}(\mathcal{N}, \text{mem})$, $\text{prob}(Ne') \geq 0$.

If $Ne = (\mathcal{N}, \text{mem})$, then $\text{CONS}(\text{dp}, Ne)$ denotes the set of all such constraints.² We can use $\text{CONS}(\text{dp}, Ne)$ to determine the survival probability of a given deployment.

Definition 4.3 Given a network Ne , a disconnect probability function dp , and a deployment μ , we say that the probability of survival of μ is given by the following linear program:

minimize $\sum_{Ne' \in \text{PFN}_{\text{dp}} \wedge \mu \text{ is a deployment w.r.t. } Ne'} \text{prob}(Ne')$
subject to $\text{CONS}(\text{dp}, Ne)$.

The solutions of $\text{CONS}(\text{dp}, Ne)$ are possible probabilities of possible future networks arising. Clearly, any of these probability assignments is possible. The objective function above adds the probabilities of all possible future networks where at least one copy of each agent in MAS survives. This expression must be minimized because different solutions of $\text{CONS}(\text{dp}, Ne)$ assign different values to this sum - as any of these solutions is possible, the only guarantee we can give about survivability of μ is that it exceeds the minimal such value.

Computing Optimal Deployment (COD) Problem. Given a network Ne , and a disconnect probability function dp , find a deployment whose probability of survival is maximal.

This is the key problem that we will solve.

5 Computing the Survival Probability of a Deployment

A naive way to find the probability of survival of a given μ is to solve the linear program of Definition 4.3 using classical linear programming algorithms [Hiller and Lieberman, 1974; Karmarkar, 1984]. However, the size of the linear program involved is enormous. Our *Compute Deployment Probability (CDP)* algorithm will avoid this problem. CDP uses a function called *Loc* which takes an agent a , a network $(\mathcal{N}, \text{mem})$, and a deployment μ as input, and returns the set of all nodes $N \in \mathcal{N}$ such that $a \in \mu(N)$ as output. One way of pruning the search is to use the following results.

Proposition 5.1 Suppose MAS is a multiagent application and Ne is a network and suppose there is at least one multiagent application deployment for MAS on Ne . Further suppose that for all agents a , $\text{mem}(a) > 0$. Then there exists an optimal multiagent deployment μ (i.e. μ has maximal probability of survival) such that for all agents a_1, a_2 , the set of locations of agent a_1 according to μ is not a strict subset of the set of locations of agent a_2 according to μ .

²It is important to note that if, for example, we know that the disconnect probabilities of n_1 and n_2 are independent, then we can expand $\text{CONS}(\text{dp}, Ne)$ to include the constraint $\text{prob}(\{n_1, n_2\}) = 1 - \text{dp}(n_1) * \text{dp}(n_2)$. For space reasons, we do not go into this in further detail.

The above result says that when trying to find an optimal multiagent deployment μ , we must ensure that no agent is located in a set of nodes that is a strict subset of the set of nodes that another agent is located in. As we shall see, this property allows us to prune our search a fair amount. Before describing our algorithm, we need to introduce some notation,

- An agent a is *relevant* w.r.t. μ and Ne if there is no other agent which is deployed at a strict subset of nodes at which a is deployed. $\text{ra}(Ne, \mu)$ denotes the set of relevant agents w.r.t. μ, Ne .
- The *necessary nodes* of Ne w.r.t. μ is $\text{nn}(Ne, \mu) = \{N \mid \exists a \in \text{ra}(Ne, \mu), N \in \text{Loc}(a, Ne, \mu)\}$. Nodes in which no relevant agents are deployed are not important.

The following theorem says that survivability is unaffected if we get rid of unnecessary nodes.

Theorem 5.1 Suppose MAS is a multiagent application, $Ne = (\mathcal{N}, \text{mem})$ is a network, dp is a disconnect probability function and μ is a feasible multiagent application deployment for MAS on Ne . Let $Ne' = (\mathcal{N}', \text{mem})$ where $\mathcal{N}' = \text{nn}(Ne, \mu)$. If μ' is the restriction of μ to Ne' , then $\text{surv}(\mu) = \text{surv}(\mu')$.

Proof Sketch. It is easy to see that it is enough to show the claim for the case that only one node is removed from $\mathcal{N}$ when constructing Ne' . Without loss of generality, let us assume that $\mathcal{N}' = \mathcal{N} \setminus \{N_1\}$. We make the following observations:

- It can be shown that $\mathcal{N}'' \subseteq \mathcal{N}'$ is feasible w.r.t. μ' iff $\mathcal{N}'' \cup \{N_1\}$ is also feasible w.r.t. μ .
- Let $n = |\mathcal{N}|$. $\text{CONS}(\text{dp}, Ne)$ consists of n equations, one for each node. $\text{CONS}(\text{dp}, Ne')$ does not include an equation for N_1 and thus includes $n - 1$ equations.
- We use prob (resp. prob') to denote the probability function in $\text{CONS}(\text{dp}, Ne)$ (resp. $\text{CONS}(\text{dp}, Ne')$). Consider the equations w.r.t. $N_i \in \mathcal{N}$, $N_i \neq N_1$. For both Ne and Ne' , both equations have the same left side, viz. $1 - \text{dp}(N_i)$. The right side of the relevant equation in $\text{CONS}(\text{dp}, Ne)$ consists of 2^{n-1} elements of the form $\text{prob}(\{N_i\} \cup \mathcal{N}''')$, $\mathcal{N}''' \subseteq \mathcal{N} \setminus \{N_i\}$. In $\text{CONS}(\text{dp}, Ne')$ the corresponding equation consists of 2^{n-2} elements of the form $\text{prob}'(\{N_i\} \cup \mathcal{N}''')$, $\mathcal{N}''' \subseteq \mathcal{N}' \setminus \{N_i\}$. Thus for each element in an equation of $\text{CONS}(\text{dp}, Ne')$ of the form $\text{prob}'(\mathcal{N}''')$ there are exactly two terms in the corresponding equation in $\text{CONS}(\text{dp}, Ne)$: one of the form $\text{prob}(\mathcal{N}''')$ and the other of the form $\text{prob}(\mathcal{N}''' \cup \{N_1\})$.
- Similarly, if the minimization expression with respect to Ne' is of length (i.e. number of terms) k , then the minimization expression with respect to Ne is $2k$. In particular, for each $\text{prob}'(\mathcal{N}''')$ there are two terms in the minimization expression of the form $\text{prob}'(\mathcal{N}''')$ and $\text{prob}'(\mathcal{N}''' \cup \{N_1\})$.

We are now ready to prove our claim. Suppose the minimization problem is solved with respect to Ne . We set $\text{prob}'(\mathcal{N}''') = \text{prob}(\mathcal{N}''') + \text{prob}(\mathcal{N}''' \cup \{N_1\})$. It is easy to see that based on our observations that $\text{CONS}(\text{dp}, Ne')$

will be satisfied and the values will minimize the relevant expression.

Suppose the minimization problem is solved with respect to Ne' . In this case, we add the following equations to $CONS(dp, Ne)$: (i) $\text{prob}'(\mathcal{N}''') = \text{prob}(\mathcal{N}''') + \text{prob}(\mathcal{N}'' \cup \{N_1\})$ (ii) We replace each expression in the minimization expression of the form $\text{prob}(\mathcal{N}''') + \text{prob}(\mathcal{N}'' \cup \{N_1\})$ by $\text{prob}'(\mathcal{N}''')$. It is easy to see that the minimization expression is identical to the one associated with Ne' and all the constraints of $CONS(dp, Ne)$ except the first one are identical to those of $CONS(dp, Ne')$ and are satisfied. Hence, it is left to show that the constraint associated with N_1 is satisfied.

This constraint is of the form: $1 - dp(N_1) = \text{prob}(\{N_1\}) + \text{prob}(\{N_1, N_2\}) + \dots + \text{prob}(\{N_1, \dots, N_n\})$. We replace any term $\text{prob}(\mathcal{N}'' \cup \{N_1\})$, $\mathcal{N}'' \subseteq \mathcal{N}'$ by $\text{prob}'(\mathcal{N}''') - \text{prob}(\mathcal{N}'')$. The sum of all the $\text{prob}'(\mathcal{N}''')$ from the last constraint of $CONS(dp, Ne')$ is equal to 1. Thus, we get that $\sum_{\mathcal{N}'' \subseteq \mathcal{N}'} \text{prob}(\mathcal{N}'') = dp(N_1)$ where $0 \leq \text{prob}(\mathcal{N}'') \leq 1$. As $0 \leq dp(N_1) \leq 1$, this equation is solvable. $\square$

The net impact of this theorem is that only necessary nodes need to be considered. We demonstrate this using Example 2.1.

Example 5.1 Consider the deployment of Example 2.1. Agent d is deployed at nodes $\{n_2, n_3, n_4\}$ and c is deployed at nodes $\{n_2, n_3\}$. Clearly, c is deployed at a strict subset of nodes at which d is deployed. In order for the deployment to survive in a given possible future network one of the nodes on which c is located, n_2 or n_3 must stay connected. But then d will also be deployed in the new network. However, if n_4 stays connected in a future network, but neither n_2 nor n_3 stay connected, the deployment will not survive. Thus, based on theorem 5.1 when computing the survivability of the deployment, there is no need to consider d and n_4 .

We are now ready to use the above theorem to formulate our algorithm *CDP* to compute probability of survival of a deployment. Our *CDP* algorithm will use the well known notion of a hitting set.

Definition 5.1 Suppose $S = \{S_1, \dots, S_n\}$ is a set of sets. A hitting set for S is any set $h \subseteq \bigcup_{i=1}^n S_i$ such that: (1) for all $1 \leq i \leq n$, $h \cap S_i \neq \emptyset$ and (2) there is no strict subset $h' \subset h$ satisfying condition (1) above. $\text{HitSet}(S)$ denotes the set of all hitting sets of S .

HitSet can be implemented in any number of standard ways prevalent in the literature. We will focus on the following hitting sets with respect to a given network.

Definition 5.2 Suppose MAS is a multiagent application, $Ne = (\mathcal{N}, \text{mem})$ is a network and μ is a feasible multiagent application deployment for MAS on Ne . The set of hitting sets with respect to MAS , Ne and μ is $\text{hs}(Ne, \mu, MAS) = \text{HitSet}(\{\text{Loc}(a, Ne, \mu) \mid a \in MAS\})$.

Intuitively, the hitting sets above describe minimal sets of nodes that must be present in a possible future network in order for the multiagent application to survive. We will use hitting sets to determine whether a deployment w.r.t. Ne can

be a deployment w.r.t. a possible future network. This intuition leads to the following algorithm *CDP*.

Algorithm 5.1 ($CDP(Ne, dp, MAS, \mu)$)

($\star$ **Input**): (1) A network $Ne = (\mathcal{N}, \text{mem})$ ($\star$)
 ($\star$ (2) a disconnect probability function dp , ($\star$)
 ($\star$ (3) a multiagent application MAS ($\star$)
 ($\star$ (4) a feasible deployment μ and ($\star$)
 ($\star$ **Output**: the survivability of μ . ($\star$)

1. $PossNe = \emptyset$; $MAS' = ra(Ne, \mu)$; $\mathcal{N}' = nn(Ne, \mu)$;
2. $H = \text{hs}(Ne', \mu', MAS')$;
3. **For any** $\mathcal{N}'' \in 2^{\mathcal{N}'}$ **do**
 - (a) $temp = H$; $flag = true$;
 - (b) **While** $temp \neq \emptyset$ **and** $flag$ **do**
 - i. $h = \text{headof}(temp)$; $temp = temp \setminus h$;
 - ii. **If** $h \subseteq \mathcal{N}''$ **then do**
 - A. $PossNe = PossNe \cup \{(\mathcal{N}'', \text{mem})\}$;
 - B. $flag = false$;
4. **Return the result of the following linear program:**
minimize $\sum_{Ne'' \in PossNe} \text{prob}(Ne'')$
subject to $Cons(dp, Ne')$.

CDP works by first focusing on the necessary nodes. Then, for each agent $a \in MAS'$, all nodes where that agent is located are identified. It then computes all hitting sets of these nodes. For any possible future network it checks whether one of the hitting sets is a subset of the nodes of the network. It is easy to see that *CDP* is exponential in the number of the necessary nodes. The following example illustrates the working of this algorithm.

Example 5.2 Consider the network, Ne and the deployment μ of example 2.1. Further, assume that $dp(n_1) = 0.1$, $dp(n_2) = 0.2$, $dp(n_3) = 0.3$ and $dp(n_4) = 0.4$. In the first step *CDP* sets $MAS' = \{a, c\}$, $\mathcal{N}' = \{n_1, n_2, n_3\}$.

There are 8 possible sets of nodes to be considered for future networks: $\mathcal{N}_1 = \{n_1\}$, $\mathcal{N}_2 = \{n_2\}$, $\mathcal{N}_3 = \{n_3\}$, $\mathcal{N}_4 = \{n_1, n_2\}$, $\mathcal{N}_5 = \{n_1, n_3\}$, $\mathcal{N}_6 = \{n_2, n_3\}$, $\mathcal{N}_7 = \{n_1, n_2, n_3\}$, $\mathcal{N}_8 = \emptyset$. We denote the associated networks by $Ne_1, Ne_2, \dots, Ne_8$.

In Step 2, *CDP* sets $H = \{h_1, h_2\}$ where $h_1 = \{n_1, n_2\}$ and $h_2 = \{n_3\}$. In step 3b it checks which of the node sets of possible future networks are supersets of a hitting set of H and will also set $PossNe = \{Ne_3, Ne_4, Ne_5, Ne_6, Ne_7\}$.

Denote $\text{prob}(Ne_i)$ with p_i . The linear program to be solved in step 4 is:

Minimize $p_3 + p_4 + p_5 + p_6 + p_7$ **subject to the constraints:**
 (1) $p_1 + p_4 + p_5 + p_7 = 0.9$
 (2) $p_2 + p_4 + p_6 + p_7 = 0.8$
 (3) $p_3 + p_5 + p_6 + p_7 = 0.7$
 (4) $p_1 + p_2 + p_3 + p_4 + p_5 + p_6 + p_7 + p_8 = 1$
 (5) $p_i \geq 0$.

Using the linear programming tool *Lindo*, we found the following results: $p_7 = 0.7$, $p_1 = 0.2$, $p_2 = 0.1$, $p_3 = p_4 = p_5 = p_6 = p_8 = 0$ which yields the minimum value of 0.7 for the objective function.

Example 5.3 In example 5.2, there were no constraints on the dependencies of the disconnect probabilities of the nodes. Suppose we know, in addition, that the probability that both

nodes n_1, n_3 get disconnected is 0.05, i.e. $dp(\{n_1, n_3\}) = 0.05$. In this case we should consider all possible future networks whose set of nodes is not a superset of $\{n_1, n_3\}$. Those sets are $\mathcal{N}_2$ and $\mathcal{N}_8$ of example 5.2. Thus, the new constraint is $p_2 + p_8 = 0.05$.

If we run the linear program of example 5.2 again with the additional constraint, the results are as follows: $p_4 = 0.05$, $p_6 = 0.05$, $p_7 = 0.65$, $p_1 = 0.2$, $p_2 = 0.05$, $p_5 = p_3 = p_8 = 0$ which yields the minimum value of 0.75 for the objective function.

6 Computing Optimal Deployments

We are now ready to develop algorithms to find an optimal multiagent deployment. We first present the COD algorithm to compute optimal deployments. We also present two heuristic algorithms, HAD1 and HAD2, which may find suboptimal deployments (but do so very fast).

6.1 The COD Algorithm

One may wonder if COD can be solved via a classical problem such as facility location problem (FLP) [Shmoys *et al.*, 1997]. In FLP, there are a set of facility locations and a set of consumers. The task is to determine which facilities should be used and which customers should be served from which facility so as to minimize some objective function (e.g. sum of travel times for customers to their assigned facility). One may think that we can directly use FLP algorithms to solve COD - unfortunately, this is not true.

Theorem 6.1 *The problem of checking if a MAD-deployment μ is optimal is NP^{NP} -hard.*

This theorem says that even if we have a polynomial oracle to solve NP-complete problems, checking if a MAD-deployment is optimal is still NP-hard! In fact, it is easy to reduce the facility location problem to that of finding an optimal deployment. Even if we have an oracle for facility location, the MAD-deployment problem is still NP-hard.

Computing an optimal MAD-deployment involves two sources of complexity. The first is the exponential space of possible deployments. The second is that even if we have a given deployment, finding its probability of survival is exponential.

Thus, to solve COD exactly, we do a state space search where the initial state places all agents in a multiagent application MAS on all nodes of the network Ne . If this placement is a deployment, then we are done. Otherwise, there are many ways of removing agents from nodes and each such way leads to a possible deployment. The value of a state is the survivability of the state, which can be computed using the CDP algorithm. As soon as a deployment is found, we can bound the search using the value of that deployment. The reason is that given any state in the search, all states obtained from that state by removing one or more agent has a lower survivability than the original state. Before presenting the COD algorithm, we first present the SEARCH routine used by it.

Algorithm 6.1 SEARCH ($Ne, dp, MAS, \mu, best_\mu, best_{val}$)

($\star$ Input: (1) A network $Ne = (\mathcal{N}, mem)$ $\star$)
 ($\star$ (2) a disconnect probability function, dp $\star$)

($\star$ (3) a multiagent application, MAS $\star$)
 ($\star$ (4) a deployment, μ $\star$)
 ($\star$ (5) best deployment found so far, $\star$)
 ($\star$ $best_\mu$ (global variable) $\star$)
 ($\star$ (6) best survivability value found thus far, $\star$)
 ($\star$ $best_{val}$ (global variable) $\star$)
 ($\star$ Output: The procedure changes the global variables $\star$)
 ($\star$ $best_\mu$ and $best_{val}$ $\star$)

```

1.  $prob(\mu) = CDP(Ne, dp, \mu, MAS);$ 
2. if ( $best_{val} < prob(\mu)$ ) then do
  (a) if  $\mu$  satisfies the mem constraints then do
    i.  $best_\mu = \mu$ 
    ii.  $best_{val} = prob(\mu)$ 
  (b) else do
    i. For any  $N \in \mathcal{N}$  do
      A. For any  $a \in \mu(N)$  do
        •  $temp_\mu = \mu;$ 
        •  $temp_\mu(N) = temp_\mu(N) \setminus \{a\}$ 
        • if  $temp_\mu$  is a deployment then
          search( $Ne, dp, MAS, temp_\mu, best_\mu, best_{val}$ )

```

We are now ready to present the COD algorithm.

Algorithm 6.2 COD(Ne, dp, MAS)

($\star$ Input: (1) A network $Ne = (\mathcal{N}, mem)$ $\star$)
 ($\star$ (2) a disconnect probability function dp , $\star$)
 ($\star$ (3) a multiagent application MAS $\star$)
 ($\star$ Output: an optimal deployment $\star$)

```

1.  $best_\mu = null; best_{val} = 0$ 
2. For any  $N \in \mathcal{N}$  do  $\mu(N) = MAS;$ 
3. search( $Ne, dp, MAS, \mu, best_\mu, best_{val}$ );
4. return  $best_\mu$ 

```

The correctness of COD depends on the correctness of bounding the search in step 2 of algorithm 6.1. We present the correctness result below.

Theorem 6.2 *Suppose MAS is a multiagent application, $Ne = (\mathcal{N}, mem)$ is a network and dp is a disconnect probability function. Then COD(Ne, dp, MAS) returns an optimal deployment of MAS on Ne .*

The astute reader may notice that CDP is computed for every placement. Many of these placements are very similar to each other. Hence, one may wonder whether it is possible to use the results of computing CDP applied to a previous placement to a placement that is very similar to the previous placement. The two propositions below show that this can be done.

Proposition 6.1 *Suppose MAS is a multiagent application, $Ne = (\mathcal{N}, mem)$ is a network and dp is a disconnect probability function. Suppose the placement μ' was obtained from the placement μ in step 2(b)iA of algorithm 6.1 and suppose $Ne' = (\mathcal{N}', mem)$, such that $\mathcal{N}' = nn(Ne, \mu')$. Then the set of hitting sets with respect to μ' and Ne' is a subset of the set of hitting sets with respect to μ and Ne . That is, $hs(Ne', \mu', MAS) \subseteq hs(Ne, \mu, MAS)$.*

The following example demonstrate a situation where $hs(Ne', \mu', MAS) \subset hs(Ne, \mu, MAS)$.

Example 6.1 Suppose $\mathcal{N} = \{n_1, n_2, n_3\}$, $MAS = \{a, b, c\}$ and the deployment μ is as follows: $\mu(n_1) = \{a, c\}$, $\mu(n_2) = \{a, b\}$ and $\mu(n_3) = \{b\}$. Thus, $Loc(a, Ne, \mu) = \{n_1, n_2\}$, $Loc(b, Ne, \mu) = \{n_2, n_3\}$, $Loc(c, Ne, \mu) = \{n_1\}$. The hitting sets are, $h_1 = \{n_1, n_2\}$, $h_2 = \{n_1, n_3\}$. If we remove agent a from node n_1 , then the set h_2 is no longer a hitting set.

Proposition 6.2 Suppose MAS is a multiagent application, $Ne = (\mathcal{N}, mem)$ is a network and dp is a disconnect probability function. Suppose the placement μ' was obtained from the placement μ in step 2(b)iA of algorithm 6.1 and suppose $Ne' = (\mathcal{N}', mem)$, such that $\mathcal{N}' = nn(Ne, \mu')$. Suppose there is $h \in hs(Ne, \mu, MAS)$ such that $N \in h$ but $N \notin \mathcal{N}'$. Then, $h \notin hs(Ne, \mu', MAS)$.

We use the above propositions to give a new version, CDP1, of the CDP algorithm.

Definition 6.1 Suppose MAS is a multiagent application, $Ne = (\mathcal{N}, mem)$ and μ is a deployment. A set $h \subseteq \mathcal{N}$ supports an agent $a \in MAS$ if there is $N \in h$ such that $a \in \mu(N)$.

CDP1 is now defined below.

Algorithm 6.3 CDP1($Ne, dp, MAS, \mu, H_p, a^*, N$)

(*** Input:** (1-4) as in CDP (algorithm 5.1.) *)
 (***** (5) a set of hitting sets H_p *)
 (***** (6) an agent $a^* \in MAS$ *)
 (***** (7) a node $N \in \mathcal{N}$ *)
 (*** Output:** (1) The survivability of μ . *)
 (***** (2) the set of hitting sets associated with μ' *)

Step 1 as in CDP.

2. If $H_p \neq \emptyset$ then do
 - (a) $H = \emptyset$;
 - (b) For any $h \in H_p$ do
 - If $N \notin h$ or $h \setminus \{N\}$ supports a^* then $H = H \cup (h \cap \mathcal{N}')$;
3. else $H = hs(Ne', \mu', MAS')$;
4. For any $\mathcal{N}'' \in 2^{\mathcal{N}}$ do
 - (a) $temp = H$; $flag = true$;
 - (b) While $temp \neq \emptyset$ and $flag$ do
 - i. $h = headof(temp)$; $temp = temp \setminus h$;
 - ii. If $h \subseteq \mathcal{N}''$ then do
 - A. $PossNe = PossNe \cup \{(\mathcal{N}'', mem)\}$;
 - B. $flag = false$;
5. Assign to p the result of the following linear program:

minimize $\sum_{Ne'' \in PossNe} prob(Ne'')$
 subject to $Cons(dp, Ne')$.
6. Return p and H ;

The SEARCH and COD algorithms need to be modified in a straightforward way to use CDP1 - we do not go through the details for space reasons.

6.2 Heuristic Algorithms

In this section, we describe two fast heuristic algorithms, HAD1, HAD2. HAD1 iteratively solves knapsack problems [Cormen *et al.*, 1990] by trying to pack nodes with low disconnect probability first.

Algorithm 6.4 HAD1 (Ne, dp, MAS)

(*** Input:** As in algorithm COD (6.2.) *)
 (*** Output:** a deployment *)

1. $\mu = \emptyset$; $flag = true$;
2. For all $N \in \mathcal{N}$ do $res_avail(N) = mem(N)$;
3. While $flag$ do
 - (a) $nodes = \mathcal{N}$; $agents = MAS$; $flag = false$;
 - (b) While ($nodes \neq \emptyset$) do
 - i. if $agents = \emptyset$ then $agents = MAS$;
 - ii. $N = argmin_{dp} nodes$;
 - iii. $agents_{dep} = knapsack(N, mem_avail(N), agents \setminus \mu(N))$;
 - iv. if $agents_{dep} \neq \emptyset$ then $flag = true$;
 - v. $\mu(N) = \mu(N) \cup agents_{dep}$;
 - vi. $res_avail(N) = res_avail(N) - \sum_{a \in agents_{dep}} mem(a)$;
 - vii. $agents = agents \setminus agents_{dep}$;
 - viii. $nodes = nodes \setminus \{N\}$;

The HAD2 algorithm is based on the intuition that we should first locate agents with high resource requirements, and then deal with agents with low resource requirements. Thus, we sort agents in ascending order according to resource requirements, place them, then go to agents with the second highest resource requirements, and so on. If at the end resources are still available, then we make replicas of the agents.

Algorithm 6.5 HAD2(Ne, dp, MAS)

(*** Input and Output** as in node-based-heuristic (HAD1) *)

1. $\mu = \emptyset$; $flag = true$;
2. For all $N \in \mathcal{N}$ $res_avail(N) = mem(N)$;
3. While $flag$ do
 - (a) $nodes = \mathcal{N}$; $agents = MAS$; $flag = false$;
 - (b) While ($agents \neq \emptyset$) do
 - i. $a = argmax_{mem} agents$
 - ii. $nodes_{poss} = \{N \mid mem(a) \leq res_avail(N)\}$;
 - iii. if $nodes_{poss} \neq \emptyset$ then do
 - A. $flag = true$;
 - B. $N = argmax_{dp} nodes_{poss}$;
 - C. $\mu(N) = \mu(N) \cup \{a\}$;
 - D. $res_avail(N) = res_avail(N) - mem(a)$
 - E. $agents = agents \setminus \{a\}$;

7 Experiments

We have implemented all the algorithms described in this paper. For space reasons, we only present experimental results on the heuristics for computing optimal deployments.

In our experiments, we varied the number of agents and the number of nodes. For each combination of agents and nodes, we ran several trials. In each trial, we randomly generated the memory available on each node, the node's disconnect probability, and the memory required for one copy of each agent. The experiments were conducted on a Linux box, using Red Hat 7.2 (Enigma). In all the experiments the number of nodes+ agents varied between 1 and 500. We randomly

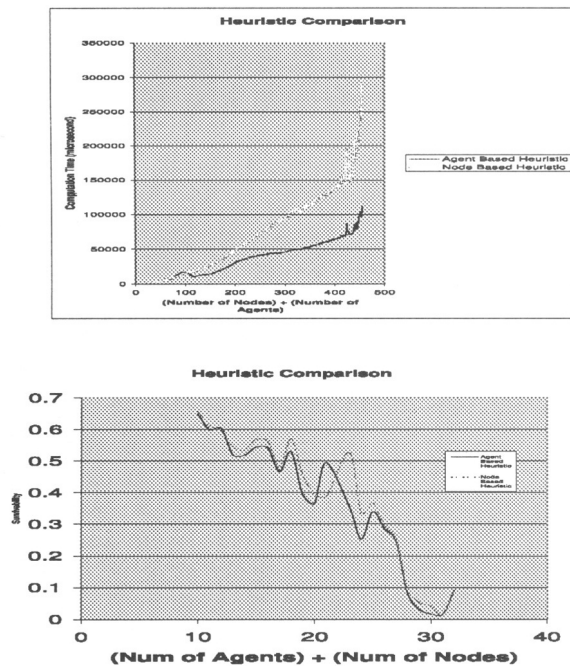


Figure 1: Heuristic comparison: Top figure: Computation time (in milliseconds) as the function of the sum of the number of nodes and agents. Bottom figure: survivability as a function of the sum of nodes and agents. The lighter line and the darker line refer to the nodes-based and agent-based heuristics, respectively.

generated numbers between 0 and 0.5 as the disconnect probability of each node. The sizes of the agents were uniformly distributed between 3 and 9 (units of memory) and the sizes of the nodes were uniformly distributed between 5 and 30 (units of memory).

The top graph of figure 7 demonstrates the time efficiency of both heuristics: they can find a deployment for 500 agents and sites in under a second.

When comparing HAD1 and HAD2, we noticed that (see bottom graph of figure 7): (1) as the sum of the number of agents and nodes increases, the survivability decreases. (2) The node based heuristic almost always finds better deployments than the agent-based heuristic. In addition, (3) When there are more agents than nodes, the node based heuristic will require less time, while when there are more nodes than agents the agent based heuristic will take less time.

The intuition behind observation (1) is as follows. As the number of agents increases, it becomes more difficult to maintain the feasibility of the system, and thus survivability decreases. In addition, when the number of nodes increases, there are more possible future networks and thus, the probability that there will be one network with a low probability increases. Since, the survivability depends on the worst case, its value decreases.

8 Conclusions

As more and more agents are deployed in mission critical commercial, telecommunications, business, and financial ap-

plications, there is a growing need for guarantees that such multiagent applications will survive various kinds of catastrophes. The scope of the problem is so vast that any one paper can only make a small dent in this very important problem.

In this paper, we have carved out such a small piece of the problem. Specifically, we study the problem of how to deploy multiple copies of agents in a MAS on nodes so that the probability of survivability of the MAS is maximized. We provide a formal, mathematical model for probabilistic MAS survivability, and develop an optimal algorithm for this purpose as well as some heuristic algorithms. We have conducted experiments showing the effectiveness of the approach.

References

- [Cormen *et al.*, 1990] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 1990.
- [Fan, 2001] X. Fan. On splitting and cloning agents, 2001. Turku Center for Computer Science, Tech. Reports 407.
- [Fedoruk and Deters, 2002] A. Fedoruk and R. Deters. Improving fault-tolerance by replicating agents. In *Proceedings AAMAS-02*, pages 737–744, Bologna, Italy, 2002.
- [Gartner, 1999] F. C. Gartner. Fundamentals of fault-tolerant distributed computing in asynchronous environments. *ACM Computing Surveys*, 31(1):1–26, 1999.
- [Gutjahr, 1998] W. J. Gutjahr. Reliability optimization of redundant software with correlated failures. In *The 9th Int. Symp. on Software Reliability Engineering*, Germany, 1998.
- [Hiller and Lieberman, 1974] F. S. Hiller and G. J. Lieberman. *Operations Research*. Holden-Day, San Francisco, 1974.
- [Karmarkar, 1984] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–396, 1984.
- [Kumar *et al.*, 2000] S. Kumar, P.R. Cohen, and H.J. Levesque. The adaptive agent architecture: achieving fault-tolerance using persistent broker teams. In *Proc. of ICMAS*, 2000.
- [Lyu and He, 1993] M. Lyu and Y. He. Improving the n-version programming process through the evolution of a design paradigm. *IEEE Trans. Reliability*, 42(2), 1993.
- [Marin *et al.*, 2001] O. Marin, P. Sens, J. Briot, and Z. Gues-soum. Towards adaptive fault tolerance for distributed multi-agent systems. In *Proceedings of ERSADS*, 2001.
- [Mishra, 2001] S. Mishra. Agent fault tolerance using group communication. In *Proc. of PDPTA-01*, NV, 2001.
- [Shehory *et al.*, 1998] O. Shehory, K. P. Sycara, P. Chalasani, and S. Jha. Increasing resource utilization and task performance by agent cloning. In *Proc. of ATAL-98*, pp 413–426, 1998.
- [Shmoys *et al.*, 1997] D. B. Shmoys, E. Tardos, and K. Aardal. Approximation algorithms for facility location problems. In *Proc. of STOC-97*, pages 265–274, 1997.

Minimally intrusive negotiating agents for resource sharing*

Fariba Sadri Francesca Toni

Department of Computing
Imperial College
180, Queens Gate
SW7 London, UK
{fs,ft}@doc.ic.ac.uk

Paolo Torroni

DEIS
University of Bologna
V.le Risorgimento, 2
40136 Bologna, Italy
ptorroni@deis.unibo.it

Abstract

We study the problem of agents negotiating periods of time during which they can have use of resources, thus allowing for the sharing of resources. We define a multi-stage negotiation framework where agents, in order to obtain resources, step through a sequence of stages, each characterised by an increased chance of a mutually agreeable deal but at the price of disclosing more and more information. In the sequence, the agents may agree to move to the next stage if the previous stage fails to produce a deal amongst them. In this paper, we concentrate on two early negotiation stages, characterised by minimal disclosure of information. Thus, the agents negotiating at these stages can be thought of as “minimally intrusive”.

1 Introduction

Negotiation of resources is an important research area in multi-agent systems. In general, agents may negotiate to obtain resources that they are missing but that are necessary to carry out their plans. In this paper, we assume that negotiation for resources takes place within the framework of [Sadri *et al.*, 2002], where the knowledge of the agents is represented as a tuple $\langle \mathcal{B}, \mathcal{R}, \mathcal{I}, \mathcal{D}, \mathcal{G} \rangle$ with $\mathcal{B}$: *beliefs* about the world, the self and the other agents, as well as the negotiation policies of the agent; $\mathcal{R}$: initially owned *resources*; $\mathcal{I}$: *intentions*, i.e., the plans¹ that the agent intends to carry out, in order to achieve its goals, together with the resources required for that plan; $\mathcal{D}$: store of past *dialogues*; and $\mathcal{G}$: agent *goals*.

We extend [Sadri *et al.*, 2002] to negotiate not just resources, but also time windows during which resources can be used by agents, thus allowing for sharing of (non-consumable) resources over time. The extended framework allows solutions for a wider range of resource re-allocation problems. In the extended framework, agent intentions are

sets of activities. Drawing inspiration from [El Sakkout and Wallace, 2000], we model an activity a (a is a unique identifier) as a tuple $\langle a, R_a, D_a, Ts_a, Te_a \rangle$ denoting that a requires resource R_a , has duration D_a , earliest start time Ts_a , latest end time Te_a . Note that, without loss of generality, we assume that each activity requires only one resource². In the extended framework, the knowledge of agents includes a (possibly empty) *concrete schedule* $\hat{\mathcal{I}}$ for activities, where if $\langle a, t_s, t_e \rangle \in \hat{\mathcal{I}}$ then $\langle a, R, D, Ts_a, Te_a \rangle \in \mathcal{I}$, such that $t_e - t_s = D$, $t_s \geq Ts_a$, and $t_e \leq Te_a$.

We develop negotiation protocols and policies that allow agents to exchange resources and strike deals for agreed time windows. We introduce a multi-stage process of negotiation, each characterised by a protocol and a policy. A higher stage involves more information passing between agents and more sophisticated negotiation, possibly including re-planning by the agents aimed at trying to help satisfy each other's constraints and requirements. The following example illustrates the sort of problems and solutions we propose in this paper.

Example 1 Let x and y be two agents. x 's intentions contain an activity a requiring a resource r , $\langle a, r, 3, 1, 5 \rangle \in \mathcal{I}_x$. y 's intentions contain an activity b also requiring r . x owns r from time 1 to 3 and from time 5 onwards, and y owns r from time 3 to 5. x needs r for three consecutive time slots between 1 and 5, but it currently owns r for only two time slots. Let us consider the three scenarios below.

1. $\langle b, r, 1, 4, 10 \rangle \in \mathcal{I}_y$. The problem is solved by y agreeing to give r to x for $[3, 4]$.
2. $\langle b, r, 1, 3, 10 \rangle \in \mathcal{I}_y$ and y has a concrete schedule $\hat{\mathcal{I}}_y = \langle b, 3, 4 \rangle$ for b to be carried out between 3 and 4. The problem is solved by y agreeing to postpone its schedule by 1 time slot and giving r to x for $[3, 4]$.
3. $\langle b, r, 2, 3, 10 \rangle \in \mathcal{I}_y$. y cannot give r away, because otherwise b becomes unfeasible. The problem can be solved by an exchange: y agrees to give r away to x for $[3, 4]$, in return for x giving r to y for $[5, 6]$. $\square$

2 Background and Preliminaries

In this section we review and adapt some concepts from background papers, needed in the rest of the paper.

²Indeed, it is possible to model any activity requiring multiple resources by a number of activities, one for each resource, with the same duration and times.

*This work is partially funded by the IST programme of the European Commission under the IST-2001-32530 SOCS project.

¹It is beyond the scope of this paper to provide a general and exhaustive representation of the agent knowledge. Instead, we concentrate on those elements relevant to the resource sharing problem. In particular, for simplicity, we identify here intentions with plans.

Definition 1 An agent system is a finite set $\mathcal{S}$, with at least two elements, where each $x \in \mathcal{S}$ is a ground term, representing the name of an agent. All elements of $\mathcal{S}$ are distinct (agent names must be uniquely identifiable). Each agent $x \in \mathcal{S}$ is equipped at each time $\tau \in \mathbb{N}$ with a knowledge base $\mathcal{K}_{x,\tau}$, namely a tuple $\langle \mathcal{B}_x, \mathcal{R}_x, \mathcal{I}_x, \mathcal{D}_{x,\tau}, \mathcal{G}_x, \hat{\mathcal{I}}_{x,\tau} \rangle$, as explained in the introduction.

Note that, in this paper, we assume that the only parts of the knowledge base of an agent that change over time are the dialogue store, which grows in time, and the concrete schedule. We will also assume that, in $\mathcal{I}_x$, all activities that require the same resource have disjoint time windows. In the sequel, when clear from the context, we will sometimes refer to the knowledge base of an agent x simply as $\mathcal{K}_x$ or $\mathcal{K}$.

Definition 2 Let $\mathcal{S}$ be an agent system. Let $\mathcal{R}_{es}$ be a set of resources.³ Let $\mathcal{R}_{es} \times \mathbb{N} \times \mathbb{N} \rightarrow \mathcal{S}$ be the resource allocation in the system at time τ (this can be determined, for example, from $\mathcal{R}$ and $\mathcal{D}$ of all agents at time τ). The allocation of resources is defined for time periods $[T_s, T_e]$, $T_s \geq 0, T_e \leq T_{max}$, where T_{max} is the maximum scheduling time, defined as follows:

$$T_{max} \stackrel{def}{=} \max_{x \in \mathcal{S}} \{TE \mid \langle -, -, -, -, TE \rangle \in \mathcal{I}_x\}.$$

The negotiation process we will define modifies the resource distribution in the agent system through time. The *Temporal Resource Reallocation Problem (T-RRP)* is the problem of answering to the following question: Does there exist a time τ during the negotiation process when the resource distribution is such that each agent has the resources it requires for time periods that would allow it to perform the activities in its intention, within their specified time windows?

The purpose of this work is to show how it is possible to find a solution to the T-RRP (when one exists) by using agents that negotiate by means of dialogues. We will assume that all agents in a system share the same communication language, in terms of syntax, semantics, ontology, and pragmatics. We refer to [Sadri et al., 2002] for a formal definition of a language for negotiation. In brief, a language defines the set of allowed dialogue moves. Each dialogue move is coded into a *tell* predicate, which has 5 arguments: sender, receiver, content, dialogue identifier, and time of the move. We will use the communication language $\mathcal{L}_{1-TW}$, defined below:

Definition 3 $\mathcal{L}_{1-TW} = \{$
 $\text{tell}(X, Y, \text{Content}, D, T) \mid \text{Content} \in$
 $\{$ (1) **request**(**give**($R, (Ts, Te)$)))
 (2) **accept**(**request**(**give**($R, (Ts, Te)$)))
 (3) **refuse**(**request**(**give**($R, (Ts, Te)$)))
 (4) **promise**($R, (Ts, Te), (Ts', Te')$)
 (5) **change**(**promise**($R, (Ts, Te), (Ts', Te')$)))
 (6) **accept**(**promise**($R, (Ts, Te), (Ts', Te')$))) $\}$ }

where Ts, Te are positive integers between 0 and T_{max} , for all dialogue moves, $Ts < Te$, and for (4)–(6), $Ts' < Te'$.

The first three moves are of intuitive meaning. (4) is used by X to propose a deal (*promise*): X will give R to Y for

³Without loss of generality, we assume that all resources in the system are non-consumable (for resource reallocation, consumable resources differ from non-consumable ones in that the former can be allocated only for maximal intervals $[0, T_{max}]$, and not for any sub-intervals, where T_{max} is given below in the definition).

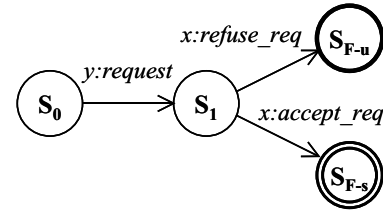


Figure 1: Protocol for Stage 1

the interval $[Ts', Te']$ if Y will give R to X for the interval $[Ts, Te]$. (5) is used to refuse a proposed deal and ask for a new one (there is no *refuse_promise* move which only terminates a protocol), (6) is used to accept a deal.

Given a language for negotiation $\mathcal{L}$, we define the set of *final moves* $\mathcal{F}(\mathcal{L})$. In particular, $\mathcal{F}(\mathcal{L}_{1-TW})$ is the subset of $\mathcal{L}_{1-TW}$ that contains all the moves whose content is (2), (3) or (6).

Dialogues can be generated by means of policies, held by the knowledge base of the agents.

Definition 4 Policies are expressed as *dialogue constraints* of the form $p_i \wedge C \Rightarrow p_{i+1}$, where p_i and p_{i+1} are moves. The conditions C are to be evaluated in the knowledge base of the agent or in extensions to it. The intended use of these policies is that if the agent receives a move p_i , and the conditions C are satisfied in its knowledge base, the agent generates p_{i+1} . An operational model for policies is defined in [Sadri et al., 2002].

Definition 5 Given an agent system $\mathcal{S}$ equipped with a language for negotiation $\mathcal{L}$ and two agents X and Y in $\mathcal{S}$ equipped with policy Pol , a *dialogue induced by Pol* between X and Y is a set of ground dialogue moves in $\mathcal{L}$, $\{p_0, p_1, p_2, \dots\}$, such that, for a given set of time lapses $0 \leq \tau_0 < \tau_1 < \tau_2 < \dots$:

1. $\forall i \geq 0$, p_i is uttered at time τ_i ;
2. $\forall i \geq 0$, if p_i is uttered by agent X (viz. Y), then p_{i+1} (if any) is uttered by agent Y (viz. X);
3. $\forall i > 0$, p_i can be uttered by agent $U \in \{X, Y\}$ only if there exists a (grounded) dialogue constraint in Pol , $p_{i-1} \wedge C \Rightarrow p_i$ s.t. $\mathcal{K}_{U, \tau_{i-1}} \wedge p_{i-1} \vdash C$;
4. there is an identifier D such that, $\forall i \geq 0$, the dialogue identifier of p_i is D ;
5. $\forall \tau, \tau_{i-1} < \tau < \tau_i, \forall i > 0$ s.t. p_i and p_{i-1} belong to the dialogue, there exist no moves at τ with either X or Y being either the receiver or the utterer.

A dialogue $\{p_0, p_1, \dots, p_m\}$, $m \geq 0$, is *terminated* if p_m is a ground final move, namely p_m is a ground instance of a move in $\mathcal{F}(\mathcal{L})$. We say that such a terminated dialogue *starts* at time τ_0 and *ends* at time τ_m .

Note that this definition prevents agents from being involved in more than one dialogue at a time (e.g., dialogues cannot be nested).

An important property of policies used to induce dialogues as in Definition 5 is *conformance* to protocols, known to all agents involved in the dialogue. A dialogue protocol can be defined as a set of states, representing the current state of dialogue, a set of allowed dialogue moves, and a set of transition rules that, given a state and a move, produce a state. A protocol is therefore defined as a finite state machine, consisting of states and arcs, which has among its states an initial state

S_0 , two final states, S_{F-s} (successful termination) and S_{F-u} (unsuccessful termination), and possibly a number of intermediate states S_i . The arcs can be viewed as allowed transitions mapping one state to another given a label. These labels correspond to the content of moves. An example of protocol is in Fig. 1. When we show the protocols, we use some abbreviations, such as for instance:

request for $request(give(R, (Ts, Te)))$,
refuse_req for $refuse(request(give(R, (Ts, Te))))$.

In order to define the concept of conformance, we define dialogues in relation with protocols.

Definition 6 Given an agent system $\mathcal{S}$ equipped with a language for negotiation $\mathcal{L}$, and a protocol $\mathcal{P}$, a *dialogue conforming to $\mathcal{P}$* , between two agents X and Y in $\mathcal{S}$, is a set of ground dialogue moves in $\mathcal{L}$, $\{p_0, p_1, p_2, \dots\}$, such that, for a given set of time lapses $0 \leq \tau_0 < \tau_1 < \tau_2 < \dots$:

1. 2. 4. 5. as for Definition 5;
3. the content of p_0 must label an arc from S_0 . $\forall i$, if the content of p_i is the label of an arc into a final state, then there is no p_{i+1} in the dialogue. $\forall i$, the contents of p_i and p_{i+1} , if they both exist, must be labels, respectively, of an arc going into a state, and an arc coming out of the same state.

A dialogue $\{p_0, p_1, \dots, p_m\}$, $m \geq 0$, is *terminated* if p_m is the label of an arc into a final state.

We are now ready to define the concept of conformance of policies to protocols.

Definition 7 Given a policy Pol and a protocol $\mathcal{P}$, Pol *conforms to $\mathcal{P}$* if every dialogue induced by Pol is a dialogue conforming to $\mathcal{P}$.

We define sequences of dialogues for a resource R and an activity A , between two agents X and Y .

Definition 8 Given an agent system $\mathcal{S}$, a *sequence of dialogues σ* between two agents X and Y in $\mathcal{S}$ for a resource R and an activity A is a set of terminated dialogues between X and Y , $\sigma = \{d_0, d_1, d_2, \dots\}$, where $d_j = \{p_{j0}, p_{j1}, \dots, p_{jn}\}$ for all j , such that, for a given set of time lapses $0 \leq \tau_0 < \tau_1 < \tau_2 < \dots$:

1. $\forall i \geq 0$, d_i is initiated by X at time τ_i , and it is terminated at a time $\tau'_i < \tau_{i+1}$;
2. $\nexists i, j, i \neq j$ such that $p_{i0} = p_{j0}$;
3. $\forall i$, if d_i and d_{i+1} are in σ , then $d_i = \{p_{i0}, \dots, p_{in}\}$ and $p_{in} = tell(Y, X, refuse(request(give(R, (Ts, Te))))), d_i, \tau)$ for some Ts, Te, τ ;
4. $\exists \langle A, R, D, TS, TE \rangle \in \mathcal{I}_X$ such that $\forall i$ $p_{i0} = tell(X, Y, request(give(R, (Ts, Te))), d_i, \tau_i)$ for some Ts, Te such that $Ts \geq TS$ and $Te \leq TE$;
5. If σ is finite, i.e. $\sigma = d_0, \dots, d_f$ for some $f \geq 0$, then $d_f = \{p_{f0}, \dots, p_{fn}\}$, the last dialogue in the sequence, is such that for some τ, Ts, Te, T'_s, T'_e , $p_{fn} = tell(Y, X, accept(request(give(R, (Ts, Te)))), d_f, \tau)$ or $p_{fn} = tell(Y, X, accept(promise(R, (Ts, Te), (T'_s, T'_e))), d_f, \tau)$.

We assume the atomicity of sequences of dialogues: agents will not react to any incoming request about a resource r if they are participating in an ongoing sequence of dialogues regarding r , and moreover they themselves will not make a request for r with respect to another activity while they are participating in an ongoing sequence of dialogues which they

initiated (an agent cycle very similar to the one in [Sadri *et al.*, 2002] will achieve this atomicity).

3 Negotiation Stages

We define two different stages of negotiation, each characterized by the degree of flexibility of the agents and the amount of information disclosed and used by them:

Stage 1: Request/flexible schedule

Stage 2: Blind deal

In this section, for each stage we define the *protocol*, the *policies* adopted by the agents, and the *properties* of the stage. The properties that we study are (i) conformance of the policy to the protocol, (ii) properties of single dialogues (termination and characterization of the class of problems that can be solved), (iii) properties of sequences of dialogues happening at that stage, and (iv) subsumption of earlier stages (in terms of solvable problems). In particular, for each stage we give an example of a problem that can be solved within it, and an example of a problem that cannot.

In defining the policies and in stating the results, we rely upon some predicates, whose formal definition is given in appendix. We use the notation $\mathcal{K}_{X,\tau} \vdash p$ to indicate that at time τ the knowledge of X entails a certain predicate p . The proofs of the results are omitted for lack of space.

3.1 Stage 1 - request/flexible schedule

Protocol. The protocol is given in Figure 1.

Policy. The policy is shown in Figure 2: an agent will *accept* a *request* for R if it can find a concrete schedule of its own activities that does not make use of R during the requested interval. It will *refuse* it otherwise, leading to an unsuccessful final state. Note that it is up to the requesting agent to find good heuristics to formulate a request or a series of requests which can be accepted.

Example 2 Let us consider the following example:

$$\begin{aligned} \mathcal{I}_y &= \{ \langle a, r, 5, 10, 20 \rangle \} & \mathcal{I}_x &= \{ \langle b, r, 5, 10, 20 \rangle \} \\ \hat{\mathcal{I}}_y &= \{ \langle a, 11, 16 \rangle \} & \hat{\mathcal{I}}_x &= \emptyset \\ \mathcal{R}_y &= \{ have(r, (10, 20)) \} & \mathcal{R}_x &= \emptyset \end{aligned}$$

There is no solution if y sticks to $\hat{\mathcal{I}}_y$: but if y was happy with a different schedule, e.g. $\hat{\mathcal{I}}'_y = \{ \langle a, 15, 20 \rangle \}$, then x and y could both do their activities by sharing resource r . Thus, this example can be solved by the following negotiation dialogue $d(1)$ occurring at Stage 1 (i.e., induced by the policy of Fig. 2):

$tell(x, y, request(give(r, (10, 15))), d(1), 1)$
 $tell(y, x, accept(request(give(r, (10, 15)))), d(1), 2)$ $\square$

Properties. Stage 1 is computationally demanding for the agent who is replying to a request. The problems that can be solved at this stage are all those that can be solved by means of a (possibly empty) modification in the agents' current concrete schedule.

(i): *protocol conformance*. It is possible to prove that the policy of Stage 1 is conforming to the protocol in Fig. 1.

(ii): *properties of single dialogues*. A request/flexible schedule interaction enjoys the property of termination. In fact, dialogues following this protocol have a fixed number of steps.

Theorem 1 Let us consider a system composed of two agents, x and y . Then, for all system resources r , all activities

(IC.1) $tell(X, y, \text{request}(\text{give}(R, (Ts, Te))), D, T) \wedge have(R, (Ts, Te), T) \wedge \neg need(R, (Ts, Te), T)$
 $\Rightarrow \exists T' \mid tell(y, X, \text{accept}(\text{request}(\text{give}(R, (Ts, Te)))), D, T') \wedge T < T'$

(IC.2) $tell(X, y, \text{request}(\text{give}(R, (Ts, Te))), D, T) \wedge have(R, (Ts, Te), T) \wedge need(R, (Ts, Te), T)$
 $\Rightarrow \exists T' \mid tell(y, X, \text{refuse}(\text{request}(\text{give}(R, (Ts, Te)))), D, T') \wedge T < T'$

(IC.3) $tell(X, y, \text{request}(\text{give}(R, (Ts, Te))), D, T) \wedge \neg have(R, (Ts, Te), T)$
 $\Rightarrow \exists T' \mid tell(y, X, \text{refuse}(\text{request}(\text{give}(R, (Ts, Te)))), D, T') \wedge T < T'$

Figure 2: Stage 1 policy for an agent y

a assigned to x , all times τ , and all intervals $[Ts, Te]$ s.t.

$\mathcal{K}_{x,\tau} \vdash miss(r, (Ts, Te), a) \wedge$

$\mathcal{K}_{y,\tau} \vdash avail(r, (Ts, Te)),$

there exists a dialogue d induced by the policy of Stage 1, starting at time τ , and ending at time τ' , such that

$\mathcal{K}_{x,\tau'} \vdash need(r, (Ts, Te), a, \tau') \wedge$

$\mathcal{K}_{y,\tau'} \vdash indiff(r, (Ts, Te)).$ Intuitively this theorem

states that if there exists a time window tw such that an agent x needs r in it and another agent has r available in tw , then there also exists a dialogue induced by the policy of Stage 1 which solves x 's reallocation problem about r . This intuitive understanding of the theorem is the result of the formal predicate definitions given in appendix.

(iii): *properties of sequences.* It is up to the agent who is missing a resource to find good heuristics to formulate the requests, which can lead to a successful dialogue sequence (provided it is possible to find a solution at this stage). Given an agent x , a resource r , and an activity a , one possibility could be a cycle starting at a time τ where x successively asks for intervals $[Ts, Te]$ such that $\mathcal{K}_{x,\tau} \vdash miss(r, (Ts, Te), a)$, until all such intervals are exhausted, or the request for one of the intervals is accepted. We call this *Strategy1*.

Theorem 2 Let us consider a system composed of two agents, x and y , negotiating at Stage 1. Then, for all resources r in the system, and all activities a assigned to x that require r , if x follows *Strategy1* in its attempt to acquire r from y , starting at τ with a request for r for an interval $[Ts, Te]$ such that $\mathcal{K}_{x,\tau} \vdash miss(r, (Ts, Te), a)$, then either the resulting sequence of dialogues will be finite and will terminate at a time τ' , and $\mathcal{K}_{x,\tau'} \vdash feas(a, (Ts^*, Te^*))$, for some Ts^*, Te^* , or $\nexists Ts', Te'$ such that

$\mathcal{K}_{x,\tau} \vdash miss(r, (Ts', Te'), a) \wedge$

$\mathcal{K}_{y,\tau} \vdash avail(r, (Ts', Te')).$

(iv): *subsumption of earlier stages.* Stage 1 subsumes our previous work done in [Sadri et al., 2002], in the sense that the resource reallocation problems it solves include those solved by [Sadri et al., 2002].

We now give a counterexample for Stage 1.

Example 3 Let us consider the following modification of Example 2, where there is a different initial resource assignment, and different time windows for the activity b (b must be completed by 15):

$\mathcal{I}_y = \{< a, r, 5, 10, 20 >\} \quad \mathcal{I}_x = \{< b, r, 5, 10, 15 >\}$
 $\mathcal{R}_y = \{have(r, (10, 15))\} \quad \mathcal{R}_x = \{have(r, (15, 20))\}$

We do not give any concrete schedules for the agents' activities since they play no role here. There is no solution to this problem that can be found at Stage 1. In fact, if y gives away r , it will not be possible for it to carry out a any more. $\square$

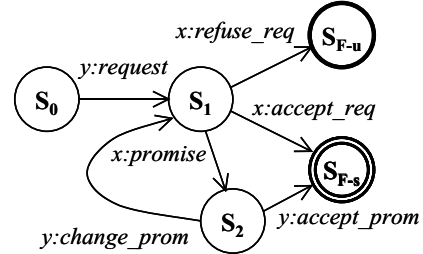


Figure 3: Protocol for Stage 2

3.2 Stage 2 - blind deal

Protocol. The protocol is shown in Fig. 3. After the initial move by y , $request(give(R, (Ts, Te)))$, which makes the dialogue reach state S_1 , the other agent x can either *accept* or *refuse*, as in Stage 1, or can propose a deal (*promise*). After a promise, the agent y who made the request can either accept the deal, causing a successful termination, or refuse it (*change_prom*), which brings back to S_1 .

Policy. The policy is shown in Fig. 4.⁴ An agent will *accept* a request about a resource R for the period $[Ts, Te]$ if it would do it at Stage 1, but it will refuse it *only if* it does not have any deal to propose (*promise*). In particular, an agent x will propose a deal, in reply to a request made by an agent y , if there exists an interval $[Ts', Te']$, disjoint from $[Ts, Te]$, which has the following property: Once y obtains R for $[Ts', Te']$, it will not need it anymore for $[Ts, Te]$. In that case, if y accepts to give R away for the interval $[Ts', Te']$, the negotiation process reaches a successful final state, otherwise x may continue proposing different deals (if they exist), until y accepts one (successful termination), or there exist no new ones to propose, in which case x will refuse the initial request, thus leading to an unsuccessful final state.

Since there might be several alternative proposals for a deal at a given time, but we want the agent's policies to be deterministic, we use in the definition of the policies of an agent x at Stage 2 a predicate $pick((Ts, Te), T)$, that at any given time T uniquely determines a time period, having made reference possibly to $\mathcal{D}_{x,T}$ and $\mathcal{R}_x$.

Example 3 can be solved by the following negotiation dialogue $d(2)$ occurring at Stage 2:

$tell(x, y, request(give(r, (10, 15))), d(2), 1)$
 $tell(y, x, promise(r, (15, 20), (10, 15)), d(2), 2)$
 $tell(x, y, accept(promise(r, (15, 20), (10, 15))), d(2), 3)$

Properties. At Stage 2 agents are more cooperative than at Stage 1. This is achieved by both agents – and in particu-

⁴We use the notation $\mathcal{K}_{X,\tau} \cup \Delta \vdash p$, where Δ is a set of atoms, to mean that $\mathcal{K}_{X,\tau}$, enlarged with Δ , entails p .

(IC.1) & (IC.3) from Stage 1

(IC.4) $\text{tell}(X, y, \text{request}(\text{give}(R, Ts, Te)), D, T) \wedge \text{have}(R, (Ts, Te), T) \wedge \text{need}(R, (Ts, Te), A, T)$
 $\wedge \nexists Ts', Te' \mid \neg \text{have}(R, (Ts', Te'), T) \wedge \text{tell}(y, X, \text{promise}(R, (Ts', Te'), (Ts, Te)), D, -) \notin \mathcal{D}_{y,T}$
 $\wedge \mathcal{K}_{y,T} \cup \{\text{obtain}(R, (Ts', Te')), \text{give_away}(R, (Ts, Te))\} \vdash \text{feas}(A, (Ts', Te'))$
 $\Rightarrow \exists T' \mid \text{tell}(y, X, \text{refuse}(\text{request}(\text{give}(R, Ts, Te))), D, T') \wedge T < T'$

(IC.5) $\text{tell}(X, y, \text{request}(\text{give}(R, Ts, Te)), D, T) \wedge \text{have}(R, (Ts, Te), T) \wedge \text{need}(R, (Ts, Te), A, T)$
 $\wedge \text{pick}((Ts', Te'), T) \wedge \neg \text{have}(R, (Ts', Te'), T) \wedge \text{tell}(y, X, \text{promise}(R, (Ts', Te'), (Ts, Te)), D, -) \notin \mathcal{D}_{y,T}$
 $\wedge \mathcal{K}_{y,T} \cup \{\text{obtain}(R, (Ts', Te')), \text{give_away}(R, (Ts, Te))\} \vdash \text{feas}(A, (Ts', Te'))$
 $\Rightarrow \exists T' \mid \text{tell}(y, X, \text{promise}(R, (Ts', Te'), (Ts, Te)), D, T') \wedge T < T'$

(IC.6) $\text{tell}(X, y, \text{change}(\text{promise}((- , -), (Ts, Te))), D, T) \wedge \text{have}(R, (Ts, Te), T) \wedge \text{need}(R, (Ts, Te), A, T)$
 $\wedge \text{pick}((Ts', Te'), T) \wedge \neg \text{have}(R, (Ts', Te'), T) \wedge \text{tell}(y, X, \text{promise}(R, (Ts', Te'), (Ts, Te)), D, -) \notin \mathcal{D}_{y,T}$
 $\wedge \mathcal{K}_{y,T} \cup \{\text{obtain}(R, (Ts', Te')), \text{give_away}(R, (Ts, Te))\} \vdash \text{feas}(A, (Ts', Te'))$
 $\Rightarrow \exists T' \mid \text{tell}(y, X, \text{promise}(R, (Ts', Te'), (Ts, Te)), D, T') \wedge T < T'$

(IC.7) $\text{tell}(X, y, \text{change}(\text{promise}(R, (Ts', Te'), (Ts, Te))), D, T) \wedge \text{have}(R, (Ts, Te), T) \wedge \text{need}(R, (Ts, Te), A, T)$
 $\wedge \nexists Ts', Te' \mid \neg \text{have}(R, (Ts', Te'), T) \wedge \text{tell}(y, X, \text{promise}(R, (Ts', Te'), (Ts, Te)), D, -) \notin \mathcal{D}_{y,T}$
 $\wedge \mathcal{K}_{y,T} \cup \{\text{obtain}(R, (Ts', Te')), \text{give_away}(R, (Ts, Te))\} \vdash \text{feas}(A, (Ts', Te'))$
 $\Rightarrow \exists T' \mid \text{tell}(y, X, \text{refuse}(\text{request}(\text{give}(R, Ts, Te))), D, T') \wedge T < T'$

(IC.8) $\text{tell}(X, y, \text{promise}(R, (Ts', Te'), (Ts, Te)), D, T) \wedge \text{miss}(R, (Ts, Te), A)$
 $\wedge \mathcal{K}_{y,T} \cup \{\text{obtain}(R, (Ts, Te)), \text{give_away}(R, (Ts', Te'))\} \vdash \text{feas}(A, (-, -))$
 $\Rightarrow \exists T' \mid \text{tell}(y, X, \text{accept}(\text{promise}(R, (Ts', Te'), (Ts, Te))), D, T') \wedge T < T'$

(IC.9) $\text{tell}(X, y, \text{promise}(R, (Ts', Te'), (Ts, Te)), D, T) \wedge \text{miss}(R, (Ts, Te), A)$
 $\wedge \mathcal{K}_{y,T} \cup \{\text{obtain}(R, (Ts, Te)), \text{give_away}(R, (Ts', Te'))\} \not\vdash \text{feas}(A, (-, -))$
 $\Rightarrow \exists T' \mid \text{tell}(y, X, \text{change}(\text{promise}(R, (Ts', Te'), (Ts, Te))), D, T') \wedge T < T'$

Figure 4: Stage 2 policy for an agent y

lar the agent to whom the request is addressed – to do more reasoning in order to be helpful in response to requests.

(i): *protocol conformance*. It is possible to prove that the policy of Stage 2 is conforming to the protocol in Fig. 3.

(ii): *properties of single dialogues*. Stage 2 does not enjoy the fixed dialogue length property of Stage 1, but it terminates if we have a finite scheduling horizon T_{max} and we do not allow the same move twice at different times, in the same dialogue (which is a reasonable requirement). Stage 2 is computationally more demanding than Stage 1 for both agents.

Theorem 3 Let us consider a system composed of two agents, x and y , each having an initial resource assignment. Then, for all resources r in the system, all activities a and b , all times τ and all intervals $[Ts, Te]$, such that

$$\begin{aligned} \mathcal{K}_{x,\tau} &\vdash \text{miss}(r, (Ts, Te), a) \wedge \\ \mathcal{K}_{y,\tau} &\vdash \text{need}(r, (Ts, Te), b) \wedge \\ \exists [Ts', Te'] \text{ such that } \mathcal{K}_{y,\tau} \cup \{ &\text{give_away}(r, (Ts, Te)), \\ &\text{obtain}(r, (Ts', Te')) \} \vdash \text{feas}(b, (Ts'', Te'')) \end{aligned}$$

for some $[Ts'', Te'']$, there exists a dialogue d induced by the policy of Stage 2, starting at time τ , and ending at a time τ' , such that

$$\begin{aligned} \mathcal{K}_{x,\tau'} &\vdash \text{need}(r, (Ts, Te), a) \wedge \\ \mathcal{K}_{y,\tau'} &\vdash \text{indiff}(r, (Ts, Te)). \end{aligned}$$

(iii): *properties of sequences*.

Theorem 4 Let us consider a system composed of two agents, x and y , negotiating at Stage 2. Then, for all resources r in the system, and all activities a assigned to x which require r , if starting at time τ x follows *Strategy I* to request r from y , either the sequence of dialogues will terminate at τ' after a finite number of dialogues, and

$$\begin{aligned} \mathcal{K}_{x,\tau'} &\vdash \text{feas}(a, (Ts^*, Te^*)) \text{ for some } Ts^*, Te^*, \text{ or} \\ &\nexists Ts, Te, Ts', Te' \text{ such that} \\ &\mathcal{K}_{x,\tau} \vdash \text{miss}(r, (Ts, Te), a) \wedge \\ &\mathcal{K}_{y,\tau} \cup \{ \text{give_away}(r, (Ts, Te)), \\ &\quad \text{obtain}(r, (Ts', Te')) \} \vdash \text{indiff}(r, (Ts, Te)). \end{aligned}$$

(iv): *subsumption of earlier stages*. More problems can be solved by Stage 2 than by Stage 1, namely:

Theorem 5 Let us consider a system composed of two agents, x and y . For all resources r in the system, all activities a assigned to x , all times τ and intervals $[Ts, Te]$ s.t.

$$\begin{aligned} \mathcal{K}_{x,\tau} &\vdash \text{miss}(r, (Ts, Te), a) \wedge \\ \mathcal{K}_{y,\tau} &\vdash \text{avail}(r, (Ts, Te)), \end{aligned}$$

there exists a dialogue d induced by the policy of Stage 2, starting at time τ , and ending at time τ' , such that

$$\begin{aligned} \mathcal{K}_{x,\tau'} &\vdash \text{need}(r, (Ts, Te), a) \wedge \\ \mathcal{K}_{y,\tau'} &\vdash \text{indiff}(r, (Ts, Te)) \end{aligned}$$

Although this enlarges the set of problems that can be solved, it does not solve, for instance, the problems where more than one exchange is needed. We now give the following counterexample for Stage 2.

Example 4 Let us consider the following modification of Example 2, where there is a different resource and activity assignment:

$$\begin{aligned} \mathcal{I}_y &= \{ \langle a, r, 2, 10, 55 \rangle, & \mathcal{I}_x &= \{ \langle b, r, 5, 10, 20 \rangle \} \\ &\quad \langle c, r, 2, 15, 20 \rangle \}, & \mathcal{R}_x &= \{ \text{have}(r, (10, 13)), \\ \mathcal{R}_y &= \{ \text{have}(r, (13, 17)) \} & &\quad \text{have}(r, (17, 20)) \} \end{aligned}$$

There is no solution to this problem that can be found at Stage 2. In fact, the (minimal) requests that x will make to obtain r for 5 consecutive time periods are: $[13, 15]$ and $[15, 17]$. y may reply to the first one with a deal to obtain r for $[10, 12]$

or for [11, 13], neither of which makes x 's activity b feasible. The same will happen for the second request, which results in an unsuccessful sequence of negotiation dialogues for a problem that has a solution. $\square$

4 Discussion

This work benefits from a logic-based high level approach that facilitates specification and proof of formal properties and which has an operational model, that forms a bridge between the system description and implementation. This feature is difficult to find in most related work: as the negotiation process becomes more elaborated, it gets harder to find and prove formal properties as those identified here. In the following, we briefly survey similar approaches to agent negotiation or proposed solutions to the same problem.

Our modelling of constraints draws inspiration from work on constraint satisfaction for a monolithic system [El Sakkout and Wallace, 2000]. But our work differs from it both in context and approach. Our aim is to provide protocols and policies for a multi-stage process of negotiation in a collaborative multi-agent context, whereas their aim is the solution of T-RRP problems viewed as constraint satisfaction problems, while minimising changes to concrete schedules. [Harrison, 2000] extends such work to multi-agent systems, but the focus there is again that of minimising changes to existing schedules, and not on negotiation.

[Conry *et al.*, 1992] proposes an approach to negotiation based on multiple stages, focusing on coordination degrees. Progressive stages require that agents solve problems in a more coordinated way. In the first stage, the agents try to solve problems independently of other agents' constraints, while in the last one it is possible, e.g., to discover that the overall problem is over-constrained and thus a certain goal is unfeasible.

There are many issues regarding negotiation that our paper does not address. For example, we do not deal with the efficiency and timeliness of negotiation. These issues are addressed in [Kraus *et al.*, 1995] using utility functions.

We are currently working on extending our negotiation process to a number of further stages, whereby the agents disclose to each other more information about their constraints in order to be able to propose more informed deals for exchanges of resources. We would like to stress that we approach a resource sharing problem from a multi-agent perspective: Existing scheduling techniques are likely to outperform compared to the negotiation processes outlined in this work, but they do not generally allow for agent autonomy. In the final stage of such an approach, where all the constraints are known, the problem becomes one of distributed constraint satisfaction. [Yokoo and Hirayama, 2000] reviews a number of algorithms for such an application.

References

- [Conry *et al.*, 1992] S.E. Conry, K. Kuwabara, V.R. Lesser, and R.A. Meyer. Multistage negotiation in distributed constraint satisfaction. *IEEE TSMC*, 21(6), 1992.
- [El Sakkout and Wallace, 2000] H.H. El Sakkout and M.G. Wallace. Probe backtrack search for minimal perturbation in dynamic scheduling. *Constraints*, 5(4), 2000.

- [Harrison, 2000] C.A. Harrison. *Distributed Optimisation in a Brokered Architecture*. PhD diss., IC-PARC, Imperial College London, 2000.
- [Kraus *et al.*, 1995] S. Kraus, J. Wilkenfeld, and G. Zlotkin. Multi-agent negotiation under time constraints. *AIJ*, 75(2), 1995.
- [Sadri *et al.*, 2002] F. Sadri, F. Toni, and P. Torroni. An abductive logic programming architecture for negotiating agents. In *Proc. JELIA*, LNCS 2424. Springer Verlag, 2002.
- [Yokoo and Hirayama, 2000] M. Yokoo and K. Hirayama. Algorithms for distributed constraint satisfaction: A review. *AAMASJ*, 3(2), 2000.

Appendix A

- $\mathcal{K}_{X,\tau} \vdash \text{have}(\mathbf{R}, (\mathbf{T}s, \mathbf{T}e), \tau) \leftrightarrow$
 $(Te \text{ is } Ts + 1 \wedge \mathcal{K}_{X,\tau} \vdash \text{have}(R, Ts)) \vee$
 $(Tn \text{ is } Ts + 1 \wedge Tn < Te \wedge \mathcal{K}_{X,\tau} \vdash \text{have}(R, (Tn, Te), \tau) \wedge$
 $\mathcal{K}_{X,\tau} \vdash \text{have}(R, Ts))$
- $\mathcal{K}_{X,\tau} \vdash \text{have}(\mathbf{R}, \mathbf{T}) \leftrightarrow$
 $(\mathcal{K}_{X,\tau} \vdash \text{have_initially}(R, T) \wedge$
 $\nexists \tau' \leq \tau \mid \mathcal{K}_{X,\tau'} \vdash \text{give_away}(R, (Ta_{giv}, Tb_{giv}))) \wedge$
 $Ta_{giv} \leq T \wedge Tb_{giv} > T) \vee$
 $(\exists \tau' \leq \tau \mid \mathcal{K}_{X,\tau'} \vdash \text{obtain}(R, (Ta_{obt}, Tb_{obt}))) \wedge$
 $Ta_{obt} \leq T \wedge Tb_{obt} > T \wedge \nexists \tau'' \leq \tau \mid$
 $\mathcal{K}_{X,\tau''} \vdash \text{give_away}(R, (Ta_{giv}, Tb_{giv}))) \wedge$
 $Ta_{giv} \leq T \wedge Tb_{giv} > T \wedge \tau' \leq \tau''$
- $\mathcal{K}_{X,\tau} \vdash \text{have_initially}(\mathbf{R}, \mathbf{T}) \leftrightarrow$
 $\exists Ts, Te \mid \text{have}(R, (Ts, Te)) \in \mathcal{R}_X \wedge Ts \leq T \wedge Te > T$
- $\mathcal{K}_{X,\tau} \vdash \text{give_away}(R, (Ta_{giv}, Tb_{giv})) \leftrightarrow$
 $\exists \text{tell}(X, -, \text{Subject}, -, \tau) \in \mathcal{D}_{X,\tau} \mid$
 $\text{Subject} \in \{\text{accept}(\text{request}(\text{give}(R, (Ta_{obt}, Tb_{obt}))),$
 $\text{accept}(\text{promise}(R, (Ta_{obt}, Tb_{obt})), (-, -))\} \vee$
 $\exists \text{tell}(-, X, \text{Subject}, -, \tau) \in \mathcal{D}_{X,\tau} \mid$
 $\text{Subject} \in \{\text{accept}(\text{promise}(R, (-, -), (Ta_{obt}, Tb_{obt})))\}$
- $\mathcal{K}_{X,\tau} \vdash \text{obtain}(R, (Ta_{obt}, Tb_{obt})) \leftrightarrow$
 $\exists \text{tell}(-, X, \text{Subject}, -, \tau) \in \mathcal{D}_{X,\tau} \mid$
 $\text{Subject} \in \{\text{accept}(\text{request}(\text{give}(R, (Ta_{obt}, Tb_{obt}))),$
 $\text{accept}(\text{promise}(R, (Ta_{obt}, Tb_{obt})), (-, -))\} \vee$
 $\exists \text{tell}(X, -, \text{Subject}, -, \tau) \in \mathcal{D}_{X,\tau} \mid$
 $\text{Subject} \in \{\text{accept}(\text{promise}(R, (-, -), (Ta_{obt}, Tb_{obt})))\}$
- $\mathcal{K}_{X,\tau} \vdash \text{feas}(A, (Ts, Te)) \leftrightarrow$
 $\exists \langle A, R, D, Ts', Te' \rangle \in \mathcal{I}_X \mid Ts' \leq Ts \wedge Te' \geq Te \wedge$
 $Te - Ts = D \wedge \mathcal{K}_{X,\tau} \vdash \text{have}(R, (Ts, Te), \tau)$
- $\mathcal{K}_{X,\tau} \vdash \text{need}(R, (Ts, Te), \tau) \leftrightarrow$
 $\exists \langle A, R, -, -, - \rangle \in \mathcal{I}_X \mid \mathcal{K}_{X,\tau} \vdash \text{need}(R, (Ts, Te), A)$
- $\mathcal{K}_{X,\tau} \vdash \text{need}(R, (Ts, Te), A) \leftrightarrow$
 $\mathcal{K}_{X,\tau} \vdash \text{have}(R, (Ts, Te), \tau) \wedge$
 $\exists \langle A, R, D, Ts', Te' \rangle \in \mathcal{I}_X \wedge Ts \geq Ts' \wedge Te \leq Te' \wedge$
 $\nexists Ts'' \in [Ts', Te' - D] \mid$
 $\mathcal{K}_{X,\tau} \vdash \text{have}(R, (Ts'', Ts'' + D), \tau) \wedge$
 $(Ts'' \geq Te \vee Ts'' + D \leq Ts)$
- $\mathcal{K}_{X,\tau} \vdash \text{miss}(R, (Ts, Te)) \leftrightarrow$
 $\exists \langle A, R, -, -, - \rangle \in \mathcal{I}_X \mid \mathcal{K}_{X,\tau} \vdash \text{miss}(R, (Ts, Te), A)$
- $\mathcal{K}_{X,\tau} \vdash \text{miss}(R, (Ts, Te), A) \leftrightarrow$
 $\exists \langle A, R, -, -, - \rangle \in \mathcal{I}_X \mid \mathcal{K}_{X,\tau} \not\vdash \text{feas}(A, (-, -)) \wedge$
 $\mathcal{K}_{X,\tau} \cup \{\text{obtain}(R, (Ts, Te))\} \vdash \text{feas}(A, (-, -)) \wedge$
 $\nexists (Ts', Te') \subset (Ts, Te) \mid$
 $\mathcal{K}_{X,\tau} \cup \{\text{obtain}(R, (Ts', Te'))\} \vdash \text{feas}(A, (-, -))$
- $\mathcal{K}_{X,\tau} \vdash \text{indiff}(R, (Ts, Te)) \leftrightarrow$
 $\mathcal{K}_{X,\tau} \not\vdash \text{have}(R, (Ts, Te), \tau) \wedge$
 $\forall (Ts_j, Te_j) \mid \mathcal{K}_{X,\tau} \vdash \text{miss}(R, (Ts_j, Te_j)) \rightarrow$
 $Te \leq Ts_j \vee Ts \geq Te_j$
- $\mathcal{K}_{X,\tau} \vdash \text{avail}(R, (Ts, Te)) \leftrightarrow$
 $\mathcal{K}_{X,\tau} \vdash \text{have}(R, (Ts, Te), \tau) \wedge \mathcal{K}_{X,\tau} \not\vdash \text{need}(R, (Ts, Te), \tau)$

NATURAL LANGUAGE

Extended Gloss Overlaps as a Measure of Semantic Relatedness

Satanjeev Banerjee

Carnegie Mellon University
Pittsburgh, PA 15213
satanjeev.banerjee@cs.cmu.edu

Ted Pedersen

University of Minnesota
Duluth, MN, 55812
tpederse@umn.edu

Abstract

This paper presents a new measure of semantic relatedness between concepts that is based on the number of shared words (overlaps) in their definitions (glosses). This measure is unique in that it extends the glosses of the concepts under consideration to include the glosses of other concepts to which they are related according to a given concept hierarchy. We show that this new measure reasonably correlates to human judgments. We introduce a new method of word sense disambiguation based on extended gloss overlaps, and demonstrate that it fares well on the SENSEVAL-2 lexical sample data.

1 Introduction

Human beings have an innate ability to determine if two concepts are related. For example, most would agree that the automotive senses of *car* and *tire* are related while *car* and *tree* are not. However, assigning a value that quantifies the degree to which two concepts are related proves to be more difficult [Miller and Charles, 1991]. In part, this is because *relatedness* is a very broad notion. For example, two concepts can be related because one is a more general instance of the other (e.g., a *car* is a kind of *vehicle*) or because one is a part of another (e.g., a *tire* is a part of a *car*).

This paper introduces *extended gloss overlaps*, a measure of semantic relatedness that is based on information from a machine readable dictionary. In particular, this measure takes advantage of hierarchies or taxonomies of concepts as found in resources such as the lexical database WordNet [Fellbaum, 1998].

Concepts are commonly represented in dictionaries by word senses, each of which has a definition or gloss that briefly describes its meaning. Our measure determines how related two concepts are by counting the number of shared words (overlaps) in the word senses of the concepts, as well as in the glosses of words that are related to those concepts according to the dictionary. These related concepts are explicitly encoded in WordNet as relations, but can be found in any dictionary via synonyms, antonyms, or also-see references provided for a word sense. To our knowledge, this work represents the first attempt to define a quantitative measure of

relatedness between two concepts based on their dictionary definitions.

This paper begins with a brief description of WordNet, which was used in developing our measure. Then we introduce the extended gloss overlap measure, and present two distinct evaluations. First, we conduct a comparison to previous human studies of relatedness and find that our measure has a correlation of at least 0.6 with human judgments. Second, we introduce a word sense disambiguation algorithm that assigns the most appropriate sense to a target word in a given context based on the degree of relatedness between the target and its neighbors. We find that this technique is more accurate than all but one system that participated in the SENSEVAL-2 comparative word sense disambiguation exercise. Finally we present an extended analysis of our results and close with a brief discussion of related work.

2 WordNet

WordNet is a lexical database where each unique meaning of a word is represented by a synonym set or *synset*. Each synset has a gloss that defines the concept that it represents. For example the words *car*, *auto*, *automobile*, and *motorcar* constitute a single synset that has the following gloss: *four wheel motor vehicle, usually propelled by an internal combustion engine*. Many glosses have *examples* of usages associated with them, such as “*he needs a car to get to work*.”

Synsets are connected to each other through explicit semantic relations that are defined in WordNet. These relations only connect word senses that are used in the same part of speech. Noun synsets are connected to each other through *hypernym*, *hyponym*, *meronym*, and *holonym* relations.

If a noun synset *A* is connected to another noun synset *B* through the *is-a-kind-of* relation then *B* is said to be a *hypernym* of synset *B* and *B* a *hyponym* of *A*. For example the synset containing *car* is a hypernym of the synset containing *hatchback* and *hatchback* is a hyponym of *car*. If a noun synset *A* is connected to another noun synset *B* through the *is-a-part-of* relation then *A* is said to be a *meronym* of *B* and *B* a *holonym* of *A*. For example the synset containing *accelerator* is a meronym of *car* and *car* is a holonym of *accelerator*. Noun synset *A* is related to adjective synset *B* through the *attribute* relation when *B* is a *value* of *A*. For example the adjective synset *standard* is a value of the noun synset *measure*.

Taxonomic or *is-a* relations also exist for verb synsets. Verb synset *A* is a hypernym of verb synset *B* if *to B is one way to A*. Synset *B* is called a *troponym* of *A*. For example the verb synset containing the word *operate* is a hypernym of *drive* since to drive is one way to operate. Conversely *drive* is a troponym of *operate*. The troponym relation for verbs is analogous to the hyponym relation for nouns, and henceforth we shall use the term hyponym instead of the term troponym. Adjective synsets are related to each other through the *similar to* relation. For example the synset containing the adjective *last* is said to be similar to the synset containing the adjective *dying*. Verb and adjective synsets are also related to each other through cross-reference *also-see* links. For example, the adjectives *accessible* and *convenient* are related through *also-see* links.

While there are other relations in WordNet, those described above make up more than 93% of the total number of links in WordNet. These are the measures we have employed in the extended gloss overlap measure.

3 The Extended Gloss Overlap Measure

Gloss overlaps were introduced by [Lesk, 1986] to perform word sense disambiguation. The Lesk Algorithm assigns a sense to a target word in a given context by comparing the glosses of its various senses with those of the other words in the context. That sense of the target word whose gloss has the most words in common with the glosses of the neighboring words is chosen as its most appropriate sense.

For example, consider the glosses of *car* and *tire*: *four wheel motor vehicle usually propelled by an internal combustion engine* and *hoop that covers a wheel, usually made of rubber and filled with compressed air*. The relationship between these concepts is shown in that their glosses share the content word *wheel*. However, they share no content words with the gloss of *tree*: *a tall perennial woody plant having a main trunk and branches forming a distinct elevated crown*.

The original Lesk Algorithm only considers overlaps among the glosses of the target word and those that surround it in the given context. This is a significant limitation in that dictionary glosses tend to be fairly short and do not provide sufficient vocabulary to make fine grained distinctions in relatedness. As an example, the average length of a gloss in WordNet is just seven words. The extended gloss overlap measure expands the glosses of the words being compared to include glosses of concepts that are known to be related to the concepts being compared.

Our measure takes as input two concepts (represented by two WordNet synsets) and outputs a numeric value that quantifies their degree of semantic relatedness. In the sections that follow, we describe the foundations of the measure and how it is computed.

3.1 Using Glosses of Related Senses

There are two fundamental premises to the original Lesk Algorithm. First, words that appear together in a sentence will be used in related senses. Second, and most relevant to our measure, the degree to which senses are related can be identified by the number of overlaps in their glosses. In other

words, the more related two senses are, the more words their glosses will share.

WordNet provides *explicit* semantic relations between synsets, such as through the *is-a* or *has-part* links. However such links do not cover all possible relations between synsets. For example, WordNet encodes no direct link between the synsets *car* and *tire*, although they are clearly related. We observe however that the glosses of these two synsets have words in common. Similar to Lesk's premise, we assert that such overlaps provide evidence that there is an *implicit* relation between those synsets. Given such a relation, we further conclude that synsets explicitly related to *car* are thereby also related to synsets explicitly related to *tire*. For example, we conclude that the synset *vehicle* (which is the hypernym synset of *car*) is related to the synset *hoop* (which is the hypernym synset of *tire*). Thus, our measure combines the advantages of gloss overlaps with the structure of a concept hierarchy to create an extended view of relatedness between synsets.

We base our measure on the idea of an extended set of comparisons. When measuring the relatedness between two input synsets, we not only look for overlaps between the glosses of those synsets, but also between the glosses of the hypernym, hyponym, meronym, holonym and troponym synsets of the input synsets, as well as between synsets related to the input synsets through the relations of attribute, similar-to and also-see. Not all of these relations are equally helpful, and the optimum choice of relations to use for comparisons is possibly dependent on the application in which the overlaps-measure is being employed. Section 6 compares the relative efficacy of these relations when our measure of relatedness is applied to the task of word sense disambiguation.

3.2 Scoring Mechanism

We introduce a novel way of finding and scoring the overlaps between two glosses. The original Lesk Algorithm compares the glosses of a pair of concepts and computes a score by counting the number of words that are shared between them. This scoring mechanism does not differentiate between single word and phrasal overlaps and effectively treats each gloss as a "bag of words". For example, it assigns a score of 3 to the concepts *drawing paper* and *decal*, which have the glosses *paper that is specially prepared for use in drafting* and *the art of transferring designs from specially prepared paper to a wood or glass or metal surface*. There are three words that overlap, *paper* and the two-word phrase *specially prepared*.

There is a Zipfian relationship [Zipf, 1935] between the lengths of phrases and their frequencies in a large corpus of text. The longer the phrase, the less likely it is to occur multiple times in a given corpus. A phrasal *n*-word overlap is a much rarer occurrence than an single word overlap. Therefore, we assign an *n* word overlap the score of n^2 . This gives an *n*-word overlap a score that is greater than the sum of the scores assigned to those *n* words if they had occurred in two or more phrases, each less than *n* words long.

For the above gloss pair, we assign the overlap *paper* a score of 1 and *specially prepared* a score of 4, leading to a total score of 5. Note that if the overlap was the 3-word phrase *specially prepared paper*, then the score would have been 9.

Thus, our overlap detection and scoring mechanism can be formally defined as follows: When comparing two glosses, we define an *overlap* between them to be the longest sequence of one or more consecutive words that occurs in both glosses such that neither the first nor the last word is a function word, that is a pronoun, preposition, article or conjunction. If two or more such overlaps have the same longest length, then the overlap that occurs earliest in the first string being compared is reported. Given two strings, the longest overlap between them is detected, removed and in its place a unique marker is placed in each of the two input strings. The two strings thus obtained are then again checked for overlaps, and this process continues until there are no longer any overlaps between them. The sizes of the overlaps thus found are squared and added together to arrive at the score for the given pair of glosses.

3.3 Computing Relatedness

The extended gloss overlap measure computes the relatedness between two input synsets A and B by comparing the glosses of synsets that are related to A and B through explicit relations provided in WordNet.

We define RELS as a (non-empty) set of relations that consists of one or more of the relations described in Section 2. That is, $RELS \subset \{r \mid r \text{ is a relation defined in WordNet}\}$. Suppose each relation r ($r \in RELS$) has a function of the same name that accepts a synset as input and returns the gloss of the synset (or synsets) related to the input synset by the designated relation.

For example, assume r represents the hypernym relation. Then $r(A)$ returns the gloss of the hypernym synset of A . r can also represent the gloss “relation” such that $r(A)$ returns the gloss of synset A , and the example “relation” such that $r(A)$ returns the example string associated with synset A . If more than one synset is related to the input synset through the same relation, their glosses are concatenated and returned. We perform this concatenation because we do not wish to differentiate between the different synsets that are all related to the input synset through a particular relation, but instead are only interested in all their definitional glosses. If no synset is related to the input synset by the given relation then the null string is returned.

Next, form a non-empty set of *pairs* of relations from the set of relations above. The only constraint in forming such pairs is that if the pair (r_1, r_2) is chosen, ($r_1, r_2 \in RELS$), then the pair (r_2, r_1) must also be chosen so that the relatedness measure is reflexive. That is, $relatedness(A, B) = relatedness(B, A)$. Thus, we define the set RELPAIRS as follows:

$$RELPAIRS = \{(R_1, R_2) \mid R_1, R_2 \in RELS; \\ \text{if } (R_1, R_2) \in RELPAIRS, \text{ then } (R_2, R_1) \in RELPAIRS\}$$

Finally, assume that $score()$ is a function that accepts as input two glosses, finds the phrases that overlap between them and returns a score as described in the previous section. Given all of the above, the relatedness score between the input synsets A and B is computed as follows:

$$relatedness(A, B) = \sum_{\forall (R_1, R_2) \in RELPAIRS} score(R_1(A), R_2(B))$$

Our relatedness measure is based on the set of all possible pairs of relations from the list of relations described in section 3.1. For purposes of illustration, assume that our set of relations $RELS = \{\text{gloss, hype, hypo}\}$ (where hype and hypo are contractions of hypernym and hyponym respectively). Further assume that our set of relation pairs $RELPAIRS = \{(\text{gloss, gloss}), (\text{hype, hype}), (\text{hypo, hypo}), (\text{hype, gloss}), (\text{gloss, hype})\}$. Then the relatedness between synsets A and B is computed as follows:

$$relatedness(A, B) = score(gloss(A), gloss(B)) + \\ score(hype(A), hype(B)) + score(hypo(A), hypo(B)) + \\ score(hype(A), gloss(B)) + score(gloss(A), hype(B))$$

Observe that due to our pair selection constraint as described above, $relatedness(A, B)$ is indeed the same as $relatedness(B, A)$.

4 Comparison to Human Judgements

Our comparison to human judgments is based on three previous studies. [Rubenstein and Goodenough, 1965] presented human subjects with 65 noun pairs and asked them how similar they were on a scale from 0.0 to 4.0. [Miller and Charles, 1991] took a 30 pair subset of this data and repeated this experiment, and found results that were highly correlated (.97) to the previous study. The results from the 30 pair set common to both studies were used again by [Budanitsky and Hirst, 2001] in an evaluation of five automatic measures of semantic relatedness that will be mentioned in Section 7. They report that all of the measures fared relatively well, with the lowest correlation being .74 and the highest .85. When comparing our measure to these 30 words, we find that it has a correlation of .67 to the Miller and Charles human study, and one of .60 to the Rubenstein and Goodenough experiment.

We do not find it discouraging that the correlation of extended gloss overlaps is lower than those reported by Budanitsky and Hirst for other measures. In fact, given the complexity of the task, it is noteworthy that it demonstrates some correlation with human judgement. The fact that the test set contains only 30 word pairs is a drawback of human evaluation, where rigorous studies are by necessity limited to a small number of words. Automatic measures can be evaluated relative to very large numbers of words, and we believe such an evaluation is an important next step in order to establish where differences lie among such measures. As a final point of concern, concepts can be related in many ways, and it is possible that a human and an automatic measure could rely on different yet equally well motivated criteria to arrive at diverging judgements.

5 Application to WSD

We have developed an approach to word sense disambiguation based on the use of the extended gloss overlap measure.

In our approach, a window of context around the target word is selected, and a set of candidate senses is identified for each content word in the window. Assume that the window of context consists of $2n + 1$ words denoted by w_i , $-n \leq i \leq +n$, where the target word is w_0 . Further let $|w_i|$ denote the number of candidate senses of word w_i , and let these senses be denoted by $s_{i,j}$, $1 \leq j \leq |w_i|$.

Next we assign to each possible sense k of the target word a $SenseScore_k$ computed by adding together the relatedness scores obtained by comparing the sense of the target word in question with every sense of every non-target word in the window of context. The $SenseScore$ for sense $s_{0,k}$ is computed as follows:

$$SenseScore_k = \sum_{i=-n}^n \sum_{j=1}^{|w_i|} relatedness(s_{0,k}, s_{i,j}), i \neq 0$$

That sense with the highest $SenseScore$ is judged to be the most appropriate sense for the target word. If there are on average a senses per word and the window of context is N words long, there are $a^2 \times (N - 1)$ pairs of sets of synsets to be compared, which increases linearly with N .

5.1 Experimental Data

Our evaluation data is taken from the English *lexical sample* task of SENSEVAL-2 [Edmonds and Cotton, 2001]. This was a comparative evaluation of word sense disambiguation systems that resulted in a large set of results and data that are now freely available to the research community.

This data consists of 4,328 *instances* each of which contains a sentence with a single target word to be disambiguated, and one or two surrounding sentences that provide additional context. A human judge has labeled each target word with the most appropriate WordNet sense for that context. A word sense disambiguation system is given these same instances (minus the human assigned senses) and must output what it believes to be the most appropriate senses for each of the target words. There are 73 distinct target words: 29 nouns, 29 verbs, and 15 adjectives, and the part of speech of the target words is known to the systems.

5.2 Experimental Results

For every instance, function words are removed and then a window of words is defined such that the target word is at the center (if possible). Next, for every word in the window, candidate senses are picked by including the synsets in WordNet that the word belongs to, as well as those that an uninflected form of the word belong to (if any). Given these candidate senses, the algorithm described above finds the most appropriate sense of the target word.

It is possible that there be a tie among multiple senses for the highest score for a word. In this case, all those senses are reported as answers and partial credit is given if one of them prove to be correct. This would be appropriate if a word were truly ambiguous in a context, or if the meanings were very closely related and it was not possible to distinguish between them. It is also possible that no sense gets more than a score of 0 – in this case, no answer is reported since there is no evidence to choose one sense over another.

Given the answers generated by the algorithm, we compare them with the human decided answers and compute *precision* (the number of correct answers divided by the number of answers reported) and *recall* (the number of correct answers divided by the number of instances). These two values can be summarized by the *F-measure*, which is the harmonic mean

Table 1: WSD Evaluation Results

Baselines and Other Systems			
Algorithm	Prec.	Recall	F-Meas.
Sval-First	0.402	0.401	0.401
Overall*	0.351	0.342	0.346
Sval-Second	0.293	0.293	0.293
Sval-Third	0.247	0.244	0.245
Original Lesk	0.183	0.183	0.183
Random	0.141	0.141	0.141

Extended Gloss Overlaps w 3 word window			
POS	Prec.	Recall	F-Meas.
Noun	0.429	0.416	0.422
Adj.	0.367	0.346	0.356
Verb	0.270	0.266	0.268
Overall*	0.351	0.342	0.346

of the precision and recall:

$$F\text{-measure} = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

Table 1 lists the precision, recall and F-measure for all the SENSEVAL-2 words when disambiguated using a window size of 3. The overall results for our approach are shown as *Overall**, and these are also broken down based on the part of speech (POS) of the target word. This table also displays results from other baseline or representative systems. The *Original Lesk* results are based on utilizing the glosses of only the input synsets and nothing else. While this does not exactly replicate the original Lesk Algorithm it is quite similar. The *random* results reflect the accuracies obtained by simply selecting randomly from the candidate senses.

The *Sval-First*, *Sval-Second*, and *Sval-Third* results are from the top three most accurate fully automatic unsupervised systems in the SENSEVAL-2 exercise. This is the class of systems most directly comparable to our own, since they require no human intervention and do not use any manually created training examples. These results show that our approach was considerably more accurate than all but one of the participating systems.

These results are significant because they are based on a very simple algorithm that relies on assigning relatedness scores to the senses of a target word and the senses of its immediately adjacent neighbors. While the disambiguation results could be improved via the combination of various techniques, our focus is on developing the extended gloss overlap measure of relatedness as a general tool for Natural Language Processing and Artificial Intelligence.

6 Discussion

Table 1 shows that the disambiguation results obtained using the extended gloss overlap measure of semantic relatedness are significantly better than both the random and Original Lesk baselines. In the Original Lesk Algorithm, relatedness between two synsets is measured by considering overlaps between the glosses of the candidate senses of the target word

and its neighbors. By adding the glosses of related synsets, the results improve by 89% relative (16.3% absolute). This shows that overlaps between glosses of synsets explicitly related to the input synsets provide almost as much evidence about the implicit relation between the input synsets as do overlaps between the glosses of the input synsets themselves.

Table 1 also breaks down the precision, recall and F-measure according to the part of speech of the target word. Observe that the noun target words are the easiest to disambiguate, followed by the adjective target words. The verb target words prove to be the hardest to disambiguate. We attribute this to the fact that the number of senses per target word is much smaller for the nouns and adjectives than it is for the verbs. Nouns and adjective target words have less than 5 candidate senses each on average, whereas verbs have close to 16. Thus, when disambiguating verbs there are more choices to be made and more chances of errors.

The results in table 1 are based on a 3 word window of context. In other experiments we used window sizes of 5, 7, 9 and 11. Although this increase in window size provides more data to the disambiguation algorithm, our experiments show that this does not significantly improve disambiguation results. This suggests that words that are in the immediate vicinity of the target word are most useful for disambiguation, and that using larger context windows is either adding noise or redundant data. The fact that small windows are best corresponds with earlier studies on human subjects that showed that humans often only require a window of one or two surrounding words to disambiguate a target word [Choueka and Lusignan, 1985].

We also tried to normalize the overlap scores by the maximum score that two glosses can generate, but that did not help performance. We believe that the difference between the sizes of various glosses in terms of number of words is small enough to render normalization unnecessary.

6.1 Evaluating Individual Relation Pairs

Our measure of relatedness utilizes pairs of relations picked from the list of relations in section 3.1. In this section we attempt to quantify the relative effectiveness of these individual relation pairs. Specifically, given a set of relations RELS, we create all possible *minimal* relation pair sets, where a minimal relation pair set is defined as the set that contains either exactly one relation pair $\{(R_1, R_1)\}$ or exactly two relation pairs $\{(R_1, R_2), (R_2, R_1)\}$, where $R_1 \neq R_2$. For example $\{(gloss, gloss)\}$ and $\{(hype, gloss), (gloss, hype)\}$ are both minimal relation pair sets.

We evaluate each of these minimal relation pair sets by performing disambiguation using only the given minimal relation pair set and computing the resulting precision, recall and F-measure. The higher the F-measure, the “better” the quality of the evidence provided by gloss overlaps from that minimal relation pair set. In effect we are decomposing the extended gloss overlap measure into its individual pieces and assessing how each of those pieces perform individually.

Recall that each part of speech has a different set of relations associated with it. The difference in the numbers and types of relations available for the three parts of speech leads us to expect that the optimal minimal relation pair sets will

Table 2: Best Relation Pair Sets

Nouns			
Relation pair	Prec.	Recall	F-Meas.
hypo-mero	0.263	0.091	0.136
hypo-hypo	0.168	0.111	0.134
gloss-mero	0.272	0.087	0.132
gloss-gloss	0.161	0.108	0.129
example-mero	0.314	0.074	0.120

Adjectives			
Relation pair	Prec.	Recall	F-Meas.
also-gloss	0.220	0.084	0.122
attr-gloss	0.323	0.072	0.117
gloss-gloss	0.146	0.094	0.114
example-gloss	0.138	0.094	0.112
gloss-hype	0.164	0.083	0.110

Verbs			
Relation pair	Prec.	Recall	F-Meas.
example-example	0.061	0.048	0.053
example-hype	0.060	0.046	0.052
hypo-hypo	0.061	0.042	0.050
gloss-hypo	0.053	0.046	0.049
example-gloss	0.054	0.045	0.049

differ with the part of speech of the input synsets. Table 2 lists the top 5 minimal relation pair sets for target words belonging to the three parts of speech, where relation pair sets are ranked on the F-measure achieved by using them in disambiguation. Note that in this table, hypo, mero, also, attr, and hype stand for the relations hyponym, meronym, also-see, attribute, and hypernym respectively. Also in the table the relation pair r_1-r_2 refers to the minimal relation pair set $\{(r_1, r_2), (r_2, r_1)\}$ if $r_1 \neq r_2$ and $\{(r_1, r_1)\}$ otherwise.

Perhaps one of the most interesting observations is that no single minimal relation pair set achieves F-measure even close to that achieved using all the relation pairs (0.42, 0.35, and 0.26 for nouns, verbs, and adjectives respectively), suggesting that there is no single relation pair that generates a lot of evidence for the relatedness of two synsets. This finding also implies that the richer the set of explicit relations between synsets in WordNet, the more accurate the overlap based measure of semantic relatedness will be. This fact is borne out by the comparatively high accuracy attained by nouns which is the best developed portion of WordNet.

For nouns, Table 2 shows that comparisons between the glosses of the hyponyms and meronyms of the input synsets and also between the glosses of the input synsets are most informative about the relatedness of the synsets. Interestingly, although both hyponyms and hypernyms make up the is-a hierarchy, the hypernym relation does not provide an equivalent amount of information. In WordNet, a noun synset usually has a single hypernym (parent) but many hyponyms (children), which implies that the hyponym relation provides more definitional glosses to the algorithm than the hypernym re-

lation. This asymmetry also exists in the holonym–meronym pair of relations. Most noun synsets have less holonym (is–a–part–of) relations than meronyms (has–part) resulting in more glosses from the meronym relation. These further confirm that the accuracy of the relatedness measure depends at least partly on the number of glosses that we can access for a given pair of synsets.

This finding also applies to adjectives. The two most frequent relations, the also–see relation and the attribute relation, rank highest among the useful relations for adjectives. Similarly for verbs, the hyponym relation again appears to be extremely useful. Interestingly, for all three parts of speech, the example “relation” (which simply returns the example string associated with the input synset) seems to provide useful information. This is in keeping with the SENSEVAL–2 results where the addition of example strings to a Lesk–like baseline system improves recall from 16% to 23%.

7 Related Work

A number of measures of semantic relatedness have been proposed in recent years. Most of them rely on the noun taxonomy of the lexical database WordNet. [Resnik, 1995] augments each synset in WordNet with an information content value derived from a large corpus of text. The measure of relatedness between two concepts is taken to be the information content value of the most specific concept that the two concepts have in common. [Jiang and Conrath, 1997] and [Lin, 1997] extend Resnik’s measure by scaling the common information content values by those of the individual concepts. Our method of extended gloss overlaps is distinct in that it takes advantage of the information found in the glosses. The other measures rely on the structure of WordNet and corpus statistics. In addition, the measures above are all limited to relations between noun concepts, while extended gloss overlaps can find relations between adjectives and verbs as well.

8 Conclusions

We have presented a new measure of semantic relatedness based on gloss overlaps. A pair of concepts is assigned a value of relatedness based on the number of overlapping words in their respective glosses, as well as the overlaps found in the glosses of concepts they are related to in a given concept hierarchy. We have evaluated this measure relative to human judgements and found it to be reasonably correlated. We have carried out a word sense disambiguation experiment with the SENSEVAL–2 lexical sample data. We find that disambiguation accuracy based on extended gloss overlaps is more accurate than all but one of the participating SENSEVAL–2 systems.

Acknowledgements

Thanks to Jason Rennie for his WordNet::QueryData module, and to Siddharth Patwardhan for useful discussions, experimental help, and for integrating the extended gloss overlap measure into his WordNet::Similarity module. Both of these modules are freely available from the Comprehensive Perl Archive Network (search.cpan.org).

This work has been supported by a National Science Foundation Faculty Early CAREER Development award (#0092784) and NSF Grant no. REC–9979894. Any opinions, findings, conclusions, or recommendations expressed in this publications are those of the authors and do not necessarily reflect the views of the NSF or the official policies, either expressed or implied, of the sponsors or of the United States Government.

References

- [Budanitsky and Hirst, 2001] A. Budanitsky and G. Hirst. Semantic distance in WordNet: An experimental, application-oriented evaluation of five measures. In *Workshop on WordNet and Other Lexical Resources, Second meeting of the North American Chapter of the Association for Computational Linguistics*, Pittsburgh, June 2001.
- [Choueka and Lusinian, 1985] Y. Choueka and S. Lusinian. Disambiguation by short contexts. *Computers and the Humanities*, 19:147–157, 1985.
- [Edmonds and Cotton, 2001] P. Edmonds and S. Cotton, editors. *Proceedings of the Senseval–2 Workshop*. Association for Computational Linguistics, Toulouse, France, 2001.
- [Fellbaum, 1998] C. Fellbaum, editor. *WordNet: An electronic lexical database*. MIT Press, 1998.
- [Jiang and Conrath, 1997] J. Jiang and D. Conrath. Semantic similarity based on corpus statistics and lexical taxonomy. In *Proceedings on International Conference on Research in Computational Linguistics*, Taiwan, 1997.
- [Lesk, 1986] M. Lesk. Automatic sense disambiguation using machine readable dictionaries: How to tell a pine cone from a ice cream cone. In *Proceedings of SIGDOC ’86*, 1986.
- [Lin, 1997] D. Lin. Using syntactic dependency as a local context to resolve word sense ambiguity. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics*, pages 64–71, Madrid, July 1997.
- [Miller and Charles, 1991] G.A. Miller and W.G. Charles. Contextual correlates of semantic similarity. *Language and Cognitive Processes*, 6(1):1–28, 1991.
- [Resnik, 1995] P. Resnik. Using information content to evaluate semantic similarity in a taxonomy. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, Montreal, August 1995.
- [Rubenstein and Goodenough, 1965] H. Rubenstein and J.B. Goodenough. Contextual correlates of synonymy. *Computational Linguistics*, 8:627–633, 1965.
- [Zipf, 1935] G. Zipf. *The Psycho-Biology of Language*. Houghton Mifflin, Boston, MA, 1935.

Evaluating Coverage for Large Symbolic NLG Grammars

Charles B. Callaway

ITC-irst Istituto per la Ricerca Scientifica e Tecnologica
via Sommarive, 18 - Povo
38050 Trento, Italy
callaway@itc.it

Abstract

After many successes, statistical approaches that have been popular in the parsing community are now making headway into Natural Language Generation (NLG). These systems are aimed mainly at surface realization, and promise the same advantages that make statistics valuable for parsing: robustness, wide coverage and domain independence. A recent experiment aimed to empirically verify the linguistic coverage for such a statistical surface realization component by generating transformed sentences from the Penn TreeBank corpus. This article presents the empirical results of a similar experiment to evaluate the coverage of a purely symbolic surface realizer. We present the problems facing a symbolic approach on the same task, describe the results of its evaluation, and contrast them with the results of the statistical method to help quantitatively determine the level of coverage currently obtained by NLG surface realizers.

1 Introduction

Like parsing, text generation offers enormous potential benefits for more natural interaction with computers. Examples of applications which could be greatly improved include automatic technical documentation, intelligent tutoring systems, and machine translation, among many others. Historically, natural language generation (NLG) has focused on the study of symbolic pipelined architectures which receive knowledge structures and goals from knowledge-based applications and which proceed to progressively add linguistic information.

In the last few years, the same paradigm shift which occurred in the parsing community, the use of statistical/empirical methods, has begun to influence the NLG community as well. As with parsing, statistical generation promises benefits such as robustness in the face of bad data, wider coverage, domain and language independence, and less need for costly resources such as grammars. But unlike parsing, which starts with a very flat representation (text) which is easily accessible in large quantities to both statistical and symbolic methods, the semantic input for NLG is typically associated with large knowledge-based systems. The types of corresponding corpora which would be necessary for using

statistical processes, pairs of subgraphs of knowledge bases and their texts, do not currently exist in large quantities.

Because of this representation problem, most statistical systems have concentrated on replacing existing individual components in the standard NLG pipelined architecture [Reiter, 1994] without changing the remaining original symbolic modules. The most popular candidate has been the surface realization module [Elhadad, 1991; Bateman, 1995; Lavoie and Rambow, 1997; White and Caldwell, 1998], which is responsible for converting the syntactic representation of a sentence into the actual text seen by the user. Thus current statistical generators are still dependent on remaining architectural modules in a system to function and do not by themselves account for a large amount of linguistic phenomena: pronominalization, revision, definiteness, *etc.*

However, statistical surface realizers [Langkilde and Knight, 1998; Bangalore and Rambow, 2000; Ratnaparkhi, 2000; Langkilde-Geary, 2002] have focused attention on a number of problems facing standard, pipelined NLG that have until now been generally considered future work: large-scale, data-robust and language- and domain-independent generation. In addition, as Langkilde points out, empirical evaluation has not been standard practice in the NLG community, which has instead relied either on the software engineering practice of regression testing with a suite of examples or theoretical evaluations [Robin and McKeown, 1995].

This paper presents the analogue of this recent statistical experiment using a well-known off-the-shelf symbolic surface realizer, using an augmented generation grammar that includes support for dialogue and additional syntactic coverage. We first describe in the following section the representations and processes needed to understand its evaluation. We then detail our implemented system for converting sentences from a large corpus into a systemic functional notation, present an evaluation of that system and the grammar itself using Section 23 of the Penn TreeBank [Marcus *et al.*, 1993], and finally discuss the implications of that evaluation.

2 Sentence Representations

To undertake a large-scale evaluation of a symbolic surface realizer, we must first find a large quantity of sentence plans with which to produce text. However, most text planners cannot generate either the requisite syntactic variation or quantity of text, and we thus cannot turn to implemented gener-

```

(S (PP (IN Without)
      (NP (NNP GM)))
  (, .)
  (NP-SBJ
    (NP (JJ overall) (NNS sales))
    (PP (IN for)
      (NP (DT the) (JJ other)
        (NNP U.S.) (NNS automakers))))
  (VP (VBD were)
    (ADJP-PRD (RB roughly) (JJ flat)
      (PP (IN with)
        (NP (CD 1989) (NNS results ")))))))

((cat clause)
 (circum ((accompaniment ((cat pp) (position front) (accomp-polarity -)
  (np ((cat proper) (lex "GM"))))))))
 (process ((type ascriptive) (tense past)))
 (participants ((carrier ((cat common) (lex "sale") (number plural)
  (describer ((cat adj) (lex "overall"))))
  (qualifier ((cat pp) (prep ((lex "for"))
    (np ((cat common) (lex "automaker") (definite yes)
      (number plural) (status different)
      (classifier ((cat proper) (lex "U.S.")))))))))
  (attribute ((cat ap) (lex "flat") (modifier ((cat adv) (lex "roughly"))
    (qualifier ((cat pp) (prep ((lex "with"))
      (np ((cat common) (lex "result") (number plural)
        (classifier ((cat date) (year 1989)))))))))))

```

Figure 1: A Penn TreeBank Annotated Sentence and Corresponding FUF/SURGE Functional Description

ation systems as a source. To solve this problem, Langkilde trained a statistical algorithm [Langkilde-Geary, 2002] on a substitute set of sentence plans: the Penn TreeBank [Marcus *et al.*, 1993], a collection of sentences from newspapers such as the Wall Street Journal, which have been hand-annotated for syntax by linguists. An example sentence is shown on the left side of Figure 1. Hierarchical syntactic/semantic bracketing is provided along with the syntactic categories of lexemes and symbols in the newspaper texts.

Unfortunately, text planners currently in use do not generate representations of the form found in the Penn TreeBank, opting instead to use more fully-developed syntactic theories, such as HPSG [Pollard and Sag, 1994], from the linguistics community. Because annotated texts do not exist in this form, Langkilde created a pre-processing system to translate from the TreeBank annotation into the language accepted by the HALOGEN statistical surface realizer [Langkilde-Geary, 2002]. HALOGEN uses these translations to create a *forest lattice* whose paths from start to finish represent many possible versions of a single sentence. Separately, a larger corpus is processed to obtain bigram or trigram frequencies, which are then used to rank the possible sentence versions based on word adjacency. The highest ranked sentence is then presented as the final output of the system.

In contrast, most deep surface realizers are symbolic rather than statistical, and consist of components that check grammatical constraints, appropriately linearize constituents, and adjust for morphology and formatting. One such system in wide use, FUF/SURGE [Elhadad, 1991], combines ideas from systemic functional grammars and head-driven phrase structure grammars. An example of the FUF representation, known as a *functional description* is shown on the right side of Figure 1. SURGE is the largest generation grammar for English, and has the largest regression test suite available. But as Langkilde pointed out, 500 test examples are insufficient to empirically demonstrate the coverage of a grammar.

To arrive at a set of sentence plans which is representative of English, as well as to evaluate the coverage of the FUF/SURGE surface realizer in a way which can be directly compared to the HALOGEN evaluation, we likewise used the Penn TreeBank as a sentence source. Because our representations are also different, we (as Langkilde) needed a pre-processing system to convert from the TreeBank notation into the functional descriptions expected by the surface realizer.

Our pre-processor thus performs top-down structure traversal of a sentence annotated in Penn TreeBank format and

```

((S (-LRB- -LRB-)
  (NP-SBJ-1 (NP (NNP Sony) )
    (NP (PRP itself))))
  (VP (VBZ declines)
    (S (NP-SBJ (-NONE- *-1))
      (VP (TO to)
        (VP (VB comment))))
    (VP (VB comment))))
  (. .) (-RRB- -RRB-)))

(S (LRB LRB)
  (NP-SBJ-1 (NP (NNP "Sony"))
    (NP (PRP "itself"))))
  (VP (VBZ "declines")
    (S (NP-SBJ (NONE STAR-1))
      (VP (TO "to")
        (VP (VB "comment"))))
    (RRB RRB)))

```

Figure 2: Penn TreeBank Notation and Normalized Form

builds the corresponding functional description. The pre-processor is organized as a context-sensitive, proceduralized rewriting grammar which matches input symbols to output symbols. The resulting functional descriptions can then be given to the FUF/SURGE surface realizer, and the sentence string it produces can be lexically compared to the original sentence in various ways to determine how well the surface realizer performs at sentence generation.

3 Implementation

The implementation necessary for evaluating the coverage of FUF/SURGE comprised three processes: (1) normalizing the syntactic/semantic representations, (2) transforming the normalizations into functional descriptions, and (3) generating the sentence itself with a surface realizer. The normalizing phase is necessary to convert the original Penn TreeBank structures into a LISP-readable format (Figure 2), which was accomplished with a series of regular expression transformations on the original text file.

The most time-consuming aspect of the procedure was creating the transformation component, which was highly analogous to writing parsing rules by hand. The resulting component contained 4000 lines of code and approximately 900 rules, although most of the actual computational effort was spent instead in surface realization. Most of the problems encountered were the result of differences in the underlying grammars themselves. For example, the Penn TreeBank has a more hierarchical noun phrase structure than the flatter representation of SURGE's systemic functional grammar.

The final task involved changing the surface realization component (1) to add additional branches and surface forms to the grammar that were not originally present in order to produce surface forms not previously possible, (2) to add new punctuation and capitalization rules¹, and (3) to update irreg-

¹The Penn TreeBank, because it is a newspaper corpus, contains many newspaper headlines and stock quotes with domain-specific

Test Set:	Sentences	Average Time	Coverage	Close Match	Exact Match	Combined	NIST SSA
Callaway WSJ24	1346	4.15 sec.	87.8%	35.8%	37.8%	73.6%	0.8475
Callaway WSJ23	2416	3.48 sec.	88.7%	34.5%	38.4%	72.9%	0.8703
Langkilde WSJ23	2416	27.1–55.5 sec.	76.2–83.3%	N/A	5.2–57.5%	0.945%	N/A

Table 1: Comparison with HALOGEN [Langkilde, 2002]

ular morphology due to the vast number of words the system had not previously seen. The principal linguistic problems uncovered by this phase include:

- *Quotations*: Newspaper text generally contains large amounts of complex quotations, such as splitting a quoted phrase to insert the speaker in the middle, or merging a quote into an unquoted part of the sentence: “I have this feeling that it’s built on sand,” she says, that the market rises “but there’s no foundation to it.”
- *Punctuation scoping*: Problems related to the use of punctuation with tree structures [Doran, 1998]. For example, SURGE has a flat representation for noun phrases, causing difficulties with phrases such as *The major “circuit breakers”* where SURGE cannot insert punctuation between the adjective and nominal classifier.
- *Adverb and clause ordering*: Because satellite clauses in SURGE are placed using semantic information, they sometime appear in different (though still grammatically acceptable) positions than were specified in the original sentence [Elhadad *et al.*, 2001]. This can oftentimes cause a perfectly acceptable sentence to be produced, but highly skew automatic measurements of correctness such as tree edit distance (simple string accuracy). For example, contrast: “Exports fell 29% in the first few months” vs. “In the first few months, exports fell 29%.”
- *Semantic roles*: SURGE has a hybrid syntactic/semantic representation, whereas the Penn TreeBank is purely syntactic. Thus some guessing must be done to fill in semantic roles in the corresponding functional description. Wrong guesses can lead to incorrect surface forms even when the remainder of the transformation was accomplished successfully.

While the normalization process is slow (10–12 hours each for WSJ23 and WSJ24), it occurs offline and only once. The transformation process is quite fast, requiring on average 0.12 seconds per sentence. Meanwhile, the FUF/SURGE system is relatively slow, as it requires the use of functional unification, a task of inherent complexity due to backtracking. Section 23 needed 8,397.2 seconds to generate 2372 sentences (with a longest exact match of 48 words), while section 24 needed 5,590.4 seconds to generate 1,326 sentences (with a longest exact match of 44), for a combined average of 3.72 seconds per sentence.² This contrasts with statistical approaches like Langkilde’s, which require 27.1–55.5 seconds depending on

formatting not typically used in NLG systems, such as: “8 13/16% high, 8 1/2% low, 8 5/8% near closing bid, 8 3/4% offered.”

²Additionally, a compiled C version of FUF can produce sentences using the same grammar on the order of 0.1 seconds.

algorithm parameters, and gets exponentially worse if it uses trigrams or larger models in an attempt to improve quality.

4 Experiments and Results

In order to evaluate the coverage of the SURGE grammar, we used the standard train and test methodology. Unlike typical machine learning experiments, adjustments to the transformation rule set were done by hand, although the evaluation of the resulting sentences was performed automatically. Training took place over a period of several months, consisting of multiple iterations over Penn TreeBank Sections 0–22 and 24 to both improve the number of sentences which could be generated and to match as closely as possible the original sentences. These two goals were accomplished solely by adding rules to the transformation set and by updating SURGE grammar rules, notably aspects pertaining to the stock market domain, in addition to support for extended quotations which was added in previous work [Callaway and Lester, 2002].

We considered two types of string matches: exact matches that were identical character-by-character, and “close” matches which were two words or less longer or shorter than the original sentence. There were several motivations for this second choice: (1) many sentences were equivalent except for a minor missing/extra punctuation mark or wrongly capitalized word (especially with newspaper headlines); (2) as mentioned previously, movable clauses (so-called circumstantials in SURGE nomenclature) could be put in multiple acceptable locations; and (3) sentences with almost the exact number of words, especially sentences with more than 15 words, were much more likely to at least have all of the various phrases present when they were within two words or the original sentence’s length. We utilized the NIST Simple String Accuracy (SSA) as an automatic evaluation score (the same as used in Langkilde’s work), where the smallest number of Adds, Deletions, and Insertions were used to calculate accuracy: $1 - (A + D + T) / \#Characters$.

The only previous measure of generation coverage for Section 23 of the Penn TreeBank is that of [Langkilde-Geary, 2002], who defined coverage as the number of sentences for which the surface realizer produced strings. As seen in Table 1, our system achieved 87.8% on one of the training sets and 88.7% on the test set compared to between 76.2% and 83.3% for HALOGEN depending on its algorithm parameters.

A more detailed examination of the coverage and accuracy of the system is found in Table 2 for WSJ Section 24 and Table 3 for Section 23. Both tables are broken down by sentence length, which shows that the results are highly skewed towards sentence of smaller length, as would be expected in a test for exact matches. It should be noted, however, that surface realizers are rarely called upon to generate sentences

Length	# Sentences	Valid FD	Valid String	Close String	Exact Match	Combined	NIST SSA
36+	117	114 (97.4%)	87 (74.4%)	39 (33.3%)	7 (6.0%)	39.3%	0.3290
35–31	113	111 (98.2%)	95 (84.1%)	42 (38.1%)	14 (12.4%)	49.6%	0.7688
30–26	319	184 (97.9%)	156 (83.0%)	87 (46.8%)	31 (16.5%)	62.8%	0.8440
25–21	238	235 (98.7%)	207 (87.0%)	100 (40.3%)	65 (27.3%)	69.3%	0.8499
20–16	256	252 (98.8%)	227 (89.0%)	111 (43.5%)	94 (36.9%)	80.4%	0.8887
15–11	223	221 (98.7%)	208 (92.9%)	70 (31.3%)	132 (58.9%)	90.2%	0.9396
10– 6	152	150 (98.7%)	145 (95.4%)	21 (13.8%)	121 (79.6%)	93.4%	0.9704
5– 1	59	59 (100%)	57 (96.6%)	12 (20.3%)	45 (76.3%)	96.6%	0.9128
All	1346	1326 (98.5%)	1182 (87.8%)	482 (35.8%)	509 (37.8%)	73.6%	0.8475

Table 2: Sentence coverage/accuracy for the training WSJ24 sentences grouped by word length

Length	# Sentences	Valid FD	Valid String	Close String	Exact Match	Combined	NIST SSA
36+	159	158 (99.4%)	131 (82.4%)	64 (40.3%)	11 (6.9%)	47.2%	0.7525
35–31	202	200 (99.0%)	159 (78.7%)	74 (36.6%)	16 (7.9%)	44.6%	0.7654
30–26	319	315 (98.8%)	268 (84.0%)	136 (42.6%)	52 (16.3%)	58.9%	0.7999
25–21	409	399 (97.6%)	356 (87.0%)	184 (45.0%)	97 (23.7%)	68.7%	0.8501
20–16	468	458 (97.5%)	425 (90.4%)	203 (43.2%)	167 (35.5%)	78.7%	0.8754
15–11	440	428 (97.3%)	403 (92.6%)	117 (26.6%)	246 (55.9%)	82.5%	0.9088
10– 6	279	275 (98.6%)	268 (96.1%)	47 (16.9%)	214 (76.7%)	93.6%	0.9594
5– 1	141	139 (98.6%)	136 (96.5%)	10 (7.1%)	126 (89.4%)	96.5%	0.9929
All	2416	2372 (98.1%)	2146 (88.7%)	835 (34.5%)	929 (38.4%)	72.9%	0.8703

Table 3: Sentence coverage/accuracy for the unseen WSJ23 sentences grouped by word length

with the extended lengths and complexities found in highly educated newspaper text. Finally, the “valid FD” column indicates that the transformation program is very good at producing valid functional descriptions, even if they eventually are discarded by the grammar as being erroneous.

The test set had a slightly higher coverage than the training set shown above, as well as a higher number of perfect matches and a better score using the NIST SSA measure. Although we trained on other sections of the Penn TreeBank, time constraints due to the large amount of time required to generate all test sentences prevented us from having a fuller comparison set. Additionally, Section 24 was the first section we trained on, and it is likely that later sections were more similar to Section 23, or that the amount of domain-specific stock market constructions were imbalanced. Finally, the “close matches” column shows how many candidate sentences might be nearly exactly matching, and the sum of these two is reflected in the final category “combined.”

One interesting observation is that this evaluation (and correspondingly, the evaluation of HALOGEN) is not only an evaluation of the underlying surface realizer, but also of the accompanying transformation program that converts the Penn TreeBank notation into the specifications it expects. We thus set out to perform a minor, secondary evaluation to determine if it were possible to find a baseline metric for how many sentences could still be generated by the surface realizer even if the corresponding FDs could not be produced for them by the transformation program.

We thus randomly selected 25 sentences from each of the two sections which were not either perfect matches or “close”

matches, *i.e.*, they were not in the “combined” match category. While these sets included some of the problems listed in Section 3, 41 of the 50 were capable of being rendered by hand as FDs which produced exact matches without changes to the grammar, 6 required minor changes to the grammar which were quickly performed, and the remaining 3 sentences required major grammar changes which have still not yet been made. The latter were sentences that still do not have satisfactory linguistic analyses in the linguistic literature. We thus conclude that with better transformation rules, we could then obtain close to 95% coverage.

5 Discussion

Surface realization is probably the most understood and competent task in NLG today. There is a high possibility that surface realization can already be considered a solved problem, except with regard to problems introduced by new languages or highly specialized domains. However, there are two related unsolved problems inherent in the process described in this paper.

5.1 Automatic Evaluation of Output

Evaluation of NLG systems face the same problems as those that confront Machine Translation systems: Given a set of generated sentences, how do you tell how “good” they are in general, and how often you can produce good sentences in a given context or application. Work in machine translation has shifted to large-scale evaluations which require automatic evaluation techniques [Papineni *et al.*, 2001;

Doddington, 2002] because human graders cannot hope to examine all of the responses in a short enough period of time.

Yet current evaluation techniques are completely numeric/statistical in nature and do not attempt to measure semantic content (such as the well-known example of a missing “not” in a system’s output). Furthermore, these techniques are ill-equipped to evaluate the types of sentences produced by symbolic NLG systems. For example, by changing a feature specifying that, say, a particularly lengthy purpose clause should go at the beginning rather than the end of a sentence, a string edit distance metric will report a very large error when in terms of the system’s input, only one “error” has occurred.

As an example of this type of problem, consider that the string edit distance between the following original sentence and that produced by the transformation program and FUF/SURGE with input from the Penn TreeBank would be 83, resulting in a NIST SSA 50.4% match:

Freddie Mac said the principal-only securities were priced at 58 1/4 to yield 8.45%, **assuming an average life of eight years and a prepayment of 160% of the PSA model.**

Freddie Mac said **assuming an average life of eight years and a prepayment of 160% of the PSA model**, the principal-only securities were priced at 58 1/4 to yield 8.45%.

Obviously 50.4% is a poor score for such sentences, but many such examples were found in our corpus and their low scores were factored into the NIST Simple String Accuracy ratios in Tables 2 and 3.

5.2 Symbolic vs. Statistical Approaches

Almost all fully-developed NLG systems to-date operate on data specified in a knowledge base from some other system. The fact that this data has typically been represented as highly-structured data has been an impediment to traditional machine learning techniques which have previously operated mostly on flat, unstructured text data. It is also generally stated that statistical methods are more robust than their symbolic counterparts and more easily adapted to new data sets. The data in the previous section seems to indicate that HALOGEN, a statistical system, performs substantially better on longer sentences, even if it has lower overall coverage. But there are several advantages in favor of symbolic techniques.

First, the transformation program presented above can be tweaked to an arbitrary level of perfection by progressively adding more rules. Most statistical and machine learning systems however have eventually reached a boundary where progress becomes seemingly exponentially more difficult. Second, errors which are encountered during processing can be examined and fixed because the grammars and other resources are logically and semantically connected to the language being generated, rather than being a set of numbers. If however a statistical generator must create new output forms not contained in the initial corpus or model, it must be retrained from scratch.

Finally, symbolic surface realizers allow a wide range of optional operations which statistical programs currently can’t

offer, for example, the capability of adding formatting statements in HTML, modifying punctuation, generating dialect differences, adding prosody for TTS, *etc.* While this may be due to the multiple decades of history over which symbolic systems have been developed, it may also be due to the lack of annotated corpora that support statistical algorithms, or even potentially the impossibility of having a corpus at all, such as in large narratives [Callaway and Lester, 2002].

Additionally, there are some disadvantages to the current approaches undertaken in statistical NLG research. For instance, symbolic NL generation systems are already considered slow, and FUF/SURGE is generally considered to be the slowest in the NLG community. And yet the data from Table 3 shows that HALOGEN is anywhere from 6.5 to 16 times slower, and thus a 10-sentence paragraph might need 4 minutes or longer to be generated. Moreover, these approaches use techniques such as n -gram models, where n must be increased to improve quality, but results in even slower generation times and exponentially larger storage space.

5.3 Potential Applications

The transformation program presented here has additional side benefits besides helping calculate the coverage of a grammar. For example, in generation systems where non-linguists must maintain old data and add new data, such a program allows them to write sample sentences in the syntax-only TreeBank notation, which is much easier for non-linguists, and then convert those sentences directly into a more linguistically-manageable form for generation (*e.g.*, functional descriptions). Graphical editing tools for linguistic data such as GATE [Bontcheva *et al.*, 2002] or similar authoring tools could quicken the process even more.

Additionally, the transformation program can be used as an error-checker for the well-formedness of sentences contained in the TreeBank. Rules could be (and have been) added alongside the normal transformation rules that detect when errors are encountered, categorize them, and make them available to the corpus creator for correction. This extends not only to annotation errors by the corpus creator detectable at the syntax tree level, but even morphology errors such as incorrect verbs, typos, or British/American English differences by the original author of the text. Both of these tasks are much more difficult for a statistical system to accomplish, requiring separate retraining in the first case and locating or creating a corpus of possible mistakes in the second, much like what is done with tutoring systems where databases of potential student errors are painstakingly constructed.

6 Conclusions and Future Work

Recent years have seen the arrival of statistical approaches to the field of Natural Language Generation, much as was seen in parsing a decade ago. Of the many possible components in the standard NLG pipelined architecture, almost all of these statistical systems have focused on the surface realization component, offering the same robustness, wide coverage, and domain- and language-independence as for parsing.

Recent experiments with one statistically-based system, HALOGEN, showed that it could achieve respectable cover-

age without the typical development costs inherent in producing grammars for symbolic NLG. By taking as input sentences from the Penn TreeBank, HALOGEN was able to generate a substantial enough quantity of sentences to allow for an empirical analysis of grammatical coverage of English. This paper represents the analogous effort for a symbolic generation system using the FUF/SURGE systemic realization system, which includes the largest generation grammar.

We presented the results of a grammatical coverage evaluation experiment that showed the symbolic system had a higher level of coverage of English as represented by the Penn TreeBank. We also contrasted the statistical and symbolic methodologies and concluded with ideas for future applications for easier creation of NLG systems and automatic error-checking for large-scale corpora. We also plan to further develop our Italian generation grammar developed at ITC-irst [Novello and Callaway, 2003] with a similarly annotated corpus of Italian newspaper text.

7 Acknowledgements

This work was funded by the PEACH project, granted by the Autonomous Province of Trento in Italy.

References

- [Bangalore and Rambow, 2000] Srinivas Bangalore and Owen Rambow. Exploiting a probabilistic hierarchical model for generation. In *COLING-2000: Proceedings of the 18th International Conference on Computational Linguistics*, Saarbrücken, Germany, 2000.
- [Bateman, 1995] John A. Bateman. KPML: The KOMET-penman (multilingual) development environment. Technical Report Release 0.8, Institut für Integrierte Publikations- und Informationssysteme (IPSI), GMD, Darmstadt, 1995.
- [Bontcheva *et al.*, 2002] K. Bontcheva, H. Cunningham, V. Tablan, D. Maynard, and H. Saggion. Development of reusable and robust language processing components for information systems using gate. In *Proceedings of the 3rd International Workshop on Natural Language and Information Systems*, 2002.
- [Callaway and Lester, 2002] Charles B. Callaway and James C. Lester. Narrative prose generation. *Artificial Intelligence*, 139(2):213–252, 2002.
- [Doddington, 2002] George Doddington. Automatic evaluation of machine translation quality using n-gram co-occurrence statistics. In *Proceedings of the 2002 Conference on Human Language Technology*, San Diego, CA, March 2002.
- [Doran, 1998] Christine Doran. *Incorporating Punctuation into the Sentence Grammar: A Lexicalized Tree Adjoining Grammar Perspective*. PhD thesis, University of Pennsylvania, Philadelphia, PA, 1998.
- [Elhadad, 1991] Michael Elhadad. FUF: The universal unifier user manual version 5.0. Technical Report CUCS-038-91, Department of Computer Science, Columbia University, 1991.
- [Elhadad *et al.*, 2001] Michael Elhadad, Yael Netzer, Regina Barzilay, and Kathleen McKeown. Ordering circumstantial for multi-document summarization. In *Proceedings of the Bar-Ilan Symposium on the Foundations of Artificial Intelligence*, Jerusalem, Israel, June 2001.
- [Langkilde and Knight, 1998] Irene Langkilde and Kevin Knight. Generation that exploits corpus-based statistical knowledge. In *COLING-ACL-98: Proceedings of the Joint 36th Meeting of the Association for Computational Linguistics and the 17th International Conference on Computational Linguistics*, pages 704–710, Montréal, Canada, August 1998.
- [Langkilde-Geary, 2002] Irene Langkilde-Geary. An empirical verification of coverage and correctness for a general-purpose sentence generator. In *Second International Natural Language Generation Conference*, pages 17–24, Hariman, NY, July 2002.
- [Lavoie and Rambow, 1997] Benoit Lavoie and Owen Rambow. A fast and portable realizer for text generation systems. In *Proceedings of the 5th Conference on Applied Natural Language Processing*, 1997.
- [Marcus *et al.*, 1993] M. Marcus, B. Santorini, and M. Marcinkiewicz. Building a large annotated corpus of English: The PennTreeBank. *Computational Linguistics*, 26(2), 1993.
- [Novello and Callaway, 2003] Alessandra Novello and Charles Callaway. Porting to an Italian surface realizer: A case study. In *Proceedings of the 9th European Workshop on NLG*, Budapest, Hungary, April 2003.
- [Papineni *et al.*, 2001] K. Papineni, S. Roukos, T. Ward, and W. J. Zhu. BLEU: A method for automatic evaluation of mt. Technical Report RC22176, IBM Research Division, T. J. Watson Research Center, New York, September 2001.
- [Pollard and Sag, 1994] C. Pollard and I. Sag. *Head-Driven Phrase Structure Grammar*. The University of Chicago Press, Chicago, 1994.
- [Ratnaparkhi, 2000] Adwait Ratnaparkhi. Trainable methods for surface natural language generation. In *Proceedings of the First North American Conference of the ACL*, Seattle, WA, May 2000.
- [Reiter, 1994] Ehud Reiter. Has a consensus NL generation architecture appeared, and is it psycholinguistically plausible? In *Proceedings of the Seventh International Workshop on Natural Language Generation*, pages 163–170, Kennebunkport, ME, 1994.
- [Robin and McKeown, 1995] Jacques Robin and Kathy McKeown. Empirically designing and evaluating a new revision-based model for summary generation. *Artificial Intelligence*, 85(1–2), 1995.
- [White and Caldwell, 1998] Michael White and Ted Caldwell. EXEMPLARS: A practical, extensible framework for dynamic text generation. In *Proceedings of the Ninth International Workshop on NLG*, pages 266–275, Niagara-on-the-Lake, Ontario, August 1998.

Hierarchical Semantic Classification: Word Sense Disambiguation with World Knowledge*

Massimiliano Ciaramita

Brown University
massi@brown.edu

Thomas Hofmann

Brown University
th@cs.brown.edu

Mark Johnson

Brown University
mark_johnson@brown.edu

Abstract

We present a learning architecture for lexical semantic classification problems that supplements task-specific training data with background data encoding general “world knowledge”. The model compiles knowledge contained in a dictionary-ontology into additional training data, and integrates task-specific and background data through a novel hierarchical learning architecture. Experiments on a word sense disambiguation task provide empirical evidence that this “hierarchical classifier” outperforms a state-of-the-art standard “flat” one.

1 Introduction

There is an increasing interest in natural language processing (NLP) and information retrieval (IR) for research on lexical semantics, in particular with respect to word sense disambiguation [Yoong and Hwee, 2002], information extraction [Riloff and Jones, 1999], named entity recognition [Collins, 2002], and automatic thesaurus extension [Hearst, 1992]. In general terms, the goal in these tasks is that of automatically associating words in text with semantic labels. In information extraction and named-entity recognition noun phrases or proper nouns are assigned to semantic categories such as “organization”, “person”, or “location”. In word sense disambiguation and thesaurus extension the goal is to assign words to finer-grained categories defined by existing dictionaries and ontologies.

Lexical semantic information can be useful in many NLP and IR applications such as text categorization, parsing, and language modeling for speech recognition. Furthermore it can be crucial for tasks that require complex inferences involving world knowledge, such as question answering.

One of the main difficulties in learning semantic annotations stems from the fact that training instances are often narrowly focused on very specific class labels and relatively few

in number. It thus seems intuitive to supplement task-specific training data, for example, sense-annotated training instances for a specific word, with background data encoding general “world knowledge”. The latter are typically available in sufficient quantities and need not to be generated separately for each classification task. To carry out this idea two crucial issues need to be addressed: How exactly can world knowledge be compiled into additional training data, and how can task-specific and background data be systematically integrated?

To address the first challenge, we propose to generate additional training data about broader semantic categories by extracting training sentences from a hierarchically structured ontology, WordNet¹ [Fellbaum, 1998]. We assumed that each example sentence associated with a lexical entry provides evidence for the kind of contexts in which that specific concept and all its ancestors in the hierarchy can appear. As far as the second challenge is concerned, we introduce a novel hierarchical learning architecture for semantic classification. More specifically, we present a simple and efficient on-line training algorithm generalizing the multiclass perceptron of [Crammer and Singer, 2002].

Finally, we carry out an experimental evaluation on a word sense disambiguation task, providing empirical evidence that the hierarchical classifier outperforms a state-of-the-art standard “flat” classifier for this task.

The paper is structured as follows. Section 2 introduces the main idea in more detail. In Section 3 we introduce WordNet and the simplified ontology derived from it that we used as the source of world knowledge. Section 4 deals with the basic multiclass perceptron and the proposed hierarchical multicomponent classifier. Finally, Sections 5 and 6 describe the data set used and the empirical results, respectively.

2 Word Sense Disambiguation and World Knowledge

Word sense disambiguation is the task of assigning to each occurrence of an ambiguous word in a text one of its possible senses. A dictionary is used to decide if a lexical entry is ambiguous or not, and to specify its set of possible senses. The most widely used lexical resource for this task is WordNet, which we describe in detail in the next section.

*We would like to thank our colleagues in the Information Retrieval and Machine Learning Group (IRML) and Brown Laboratory for Linguistic Information Processing (BLLIP), as well as Jesse Hochstadt for his editing advice. This material is based upon work supported by the National Science Foundation under Grant No. 0085940.

¹In this paper we always refer to WordNet version 1.71.

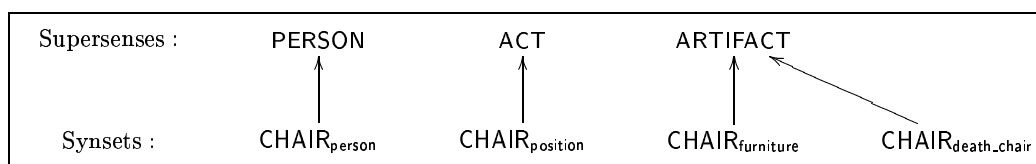


Figure 1. The simplified two-layer hierarchy for the noun *chair*.

As an illustration consider the noun “chair”, which according to WordNet is ambiguous. Two possible senses are explained in the following WordNet entries:

- *chair*₁ – a seat for one person, with a support for the back;
- *chair*₂ – (president, chairman, chairwoman, chair, chairperson), the officer who presides at the meetings of an organization;

Word sense disambiguation is often framed as a multi-class pattern classification task. Useful features include co-occurring words, word bigrams or trigrams, and properties of the syntactic context that contains the target word. Most commonly systems are trained on labeled data for a specific word, for each and tested on unseen items of the same word. The set of possible labels is the set of senses of the ambiguous word. One limitation of such a strategy is that the system bases its decision exclusively on what it has been able to learn about a few very specific concepts; e.g., *chair*₁ and *chair*₂. Furthermore, since manually sense-tagging words for the required training data is slow and expensive, the data is quite sparse.

A great deal of information about objects like “chairs” is indirect and can be derived from more general world knowledge through generalization and inference processes. Suppose that the task is to disambiguate between the two simple senses of *chair* in the following context:

1) “Here the quality of the finest *chair* components is merged with art.”

In this sentence *components* is a useful hint that we are dealing with the sense *chair*₁. Chairs are artifacts, and artifacts can have components. Conversely, even though in principle people could “have components” as well, this sounds a little odd. Intuitively, if a word sense disambiguation system had access to this type of information - that “chairs” are subordinates of broader concepts like “artifacts” and “people” - and some knowledge about these broader semantic categories, it might achieve a higher accuracy in disambiguating words. Notice that the system might never have previously observed any instance of the noun “chair”, in either sense, as “having components”.

The goal hence is to complement specific but limited knowledge about narrow classes with richer, if less specific, knowledge about more general classes. We can easily recover the fact that chairs are kinds of furniture or people from dictionaries and hierarchically organized ontologies like WordNet. Learning information about such general concepts, however, is complicated. One source of complication is the very problem we are trying to solve, lexical ambiguity. If we do

not know whether something is a person or an artifact we cannot learn reliable information about those more general concepts. One way of addressing these problems is offered by WordNet itself.

3 The Ontology

3.1 WordNet

WordNet is a broad-coverage, machine-readable dictionary widely used in NLP. The English version contains around 150,000 entries, mostly nouns, but also verbs, adjectives, and adverbs. WordNet is organized as a network of lexicalized concepts, called *synsets*, that comprise sets of synonyms. For example, the nouns {president, chairman, chairwoman, chair, chairperson} form a synset. A word that belongs to several synsets is *ambiguous*. Synsets are linked by semantic relations, the most important of which for nouns and verbs is the is-a relation, or *hyponymy*; e.g., “car” is a hyponym of “vehicle”. The verb and noun databases form is-a hierarchies with a few general concepts at the top and several thousand specific concepts at the leaf level.

The hierarchical structure of the database has aroused some interest in NLP, because it can support interesting computational language learning models, for example, in learning predicate selectional preferences [Light and Greif, 2002]. We aim to use the hierarchy to improve lexical classification methods. The model we present here can in principle make use of the full hierarchy. However, for the sake of simplicity we have focused on a less complex hierarchy, which has been derived from WordNet as described below.

3.2 A simple two-level hierarchy

WordNet was built, and is regularly updated, by lexicographers. Lexicographers group words together in synsets and individuate the relevant semantic relations between synsets. This process includes the classification of lexical entries into one of 26 broad semantic classes. In this paper we refer to these broad classes with the term *supersenses*. A few examples of supersense labels are person, animal, artifact, food, location, time, plant, process, attribute, substance, and relation. This set of labels is fairly general and therefore small. At the same time the labels are not too abstract. In other words, these classes seem natural and easily recognizable, and that is probably why lexicographers use them. In fact the level of generality is very close to that used in named-entity recognition (“location”, “person”, “organization”, etc.).

Each synset in WordNet is associated with one supersense label. As a result the database implicitly defines, in addition to the full hierarchy, a simpler two-layer hierarchy. Figure

1 above illustrates the synsets and supersenses *chair* belongs to.

3.3 The hierarchy as a source of world knowledge

For a few thousand concepts WordNet lists, among other types of semantic information, one or more example sentences. For the sense of chair above the example sentences are the following:

- *chair*₁ – “he put his coat over the back of the chair and sat down”
- *chair*₂ – “address your remarks to the chairperson”

Overall there are 9,258 of these sentences. Since each one is associated with one synset, that is in fact a sense-tagged instance of the word. In other words, WordNet provides a few thousand potential sense-tagged training instances.

Unfortunately, this additional data in itself would not be of much help: for most of the synsets there are no sentences², and typically the sentences are very short and do not provide much context. However, the situation appears in a different light if we take into account the hierarchy. Considering an example sentence for a synset also as an example sentence for its ancestors (synsets at higher levels in the hierarchy), the number of sentences grows larger at the superordinate levels. If we consider the supersense level, the set of example sentences constitutes in fact a small corpus of supersense-annotated data. Our hypothesis is that the several hundred sentences associated with each supersense can provide a useful source of general world knowledge. In the next section we describe a general multicomponent learning architecture that can be used to exploit this supplementary training data.

4 Multicomponent Learning Architecture

The idea of using the hierarchical structure of a domain to overcome sparseness problems has been explored in text categorization. These methods show improved accuracy and efficiency [Toutanova et al., 2001; Dumais and Chen, 2000]. In NLP the hierarchical structure of WordNet has been used to overcome sparseness data problems for estimating class distributions [Clark and Weir, 2002], and to exploit morphological information to improve lexical acquisition [Ciarmita, 2002].

4.1 Multiclass perceptron

The architecture we propose is a generalization of “ultra-conservative” on-line learning [Crammer and Singer, 2002], which is itself an extension of perceptron learning to the multiclass case. We describe this “flat” version of the classifier first. For each noun w we are given a training set $S = (x_i, y_i)_{i=1}^n$, where each instance $x_i \in \mathbb{R}^d$ and $y_i \in Y(w)$. $Y(w)$ is the set of synsets that WordNet assigns to w . Thus S summarizes n instances of noun w , where each instance i is represented as a vector of features x_i extracted from the context in which w occurred; d is the total number of features and y_i is the true label of x_i .³

²There are in total around 75,000 synsets in the noun database.

³Since some instances are labeled with multiple senses, in cases where the taggers were uncertain, y_i may actually be a set of labels.

Algorithm 1 Multiclass Perceptron

```

1: input training data  $(x_i, y_i)_{i=1}^n, \mathbf{V} = 0$ 
2: repeat
3:   for  $i = 1, \dots, n$  do
4:     if  $H_w(x_i; \mathbf{V}) \neq y_i$  then
5:        $v_{y_i} \leftarrow v_{y_i} + x_i$ 
6:        $E_i = \{y \in Y(w) : \langle v_y, x_i \rangle > \langle v_{y_i}, x_i \rangle\}$ 
7:       for  $y \in E_i$  do
8:          $v_y \leftarrow v_y - \frac{1}{|E_i|} x_i$ 
9:       end for
10:    end if
11:  end for
12: until no more mistakes

```

In general, a multiclass classifier for word w is a function $H_w : \mathbb{R}^n \rightarrow Y(w)$ that maps feature vectors x to one of the possible senses of w . In the multiclass perceptron, one introduces a weight vector $v_y \in \mathbb{R}^d$ for every $y \in Y(w)$ and defines H_w implicitly by the so-called winner-take-all rule:

$$H_w(x; \mathbf{V}) = \arg \max_{y \in Y(w)} \langle v_y, x \rangle. \quad (1)$$

Here $\mathbf{V} \in \mathbb{R}^{k \times d}$ refers to the matrix of weights, every column corresponding to one of the weight vectors v_y .

The learning algorithm works as follows: Training patterns are presented one at a time in the standard on-line learning setting. Whenever $H_w(x_i; \mathbf{V}) \neq y_i$ an update step is performed; otherwise the weight vectors remain unchanged. To perform the update, one first computes the error set E_i containing those class labels that have received a higher score than the correct class:

$$E_i = \{y \in Y(w) : \langle v_y, x_i \rangle > \langle v_{y_i}, x_i \rangle\} \quad (2)$$

An ultraconservative update scheme in its most general form is then defined as follows: Update $v_y \leftarrow v_y + \tau_y x_i$ with learning rates fulfilling the constraints $\tau_{y_i} = 1$, $\sum_{y \neq y_i} \tau_y = -1$, and $\tau_y = 0$ for $y \notin E_i \cup \{y_i\}$. Hence changes are limited to v_y for $y \in E_i \cup \{y_i\}$. The sum constraint ensures that the update is balanced, which is crucial to guaranteeing the convergence of the learning procedure (cf. [Crammer and Singer, 2002]). We have focused on the simplest case of uniform update weights, $\tau_y = -\frac{1}{|E_i|}$ for $y \in E_i$. The algorithm is summarized in Algorithm 1.

Notice that the presented multiclass perceptron algorithm learns all weight vectors in a coupled manner, in contrast to methods that perform multiclass classification by combining binary classifiers, for example, training a classifier for each class in a one-against-the-rest manner.

4.2 Hierarchical multiclass perceptron

The hierarchical multiclass perceptron is inspired by the framework for learning over structured output spaces introduced in [Hofmann et al., 2002]. The key idea is to introduce a weight vector not only for every (leaf-level) class, but also for every inner node in a given class taxonomy. In the current application to word sense disambiguation, the inner nodes correspond to the 26 supersenses s and we will hence

introduce additional weight vectors v_s for $s \in S(w)$, where $S(w)$ refers to the subset of supersenses induced by $Y(w)$. We will use the notation $s(y)$ to refer to the supersense corresponding to a synset y . Then discriminant functions $f(x, y)$ can be defined in an additive manner by

$$f(x, y; \mathbf{V}) = \langle v_y, x \rangle + \langle v_{s(y)}, x \rangle. \quad (3)$$

If one thinks of f in terms of a compatibility function between an observation vector x and a synset y , then the compatibility score is simply the sum of two independent contributions, one stemming from the supersense level and the other one coming from the more detailed synset level. The multiclass classifier is then again defined using the winner-take-all rule,

$$H_w(x; \mathbf{V}) = \arg \max_y f(x, y; \mathbf{V}), \quad (4)$$

Algorithm 2 Hierarchical Multiclass Perceptron

```

1: input training data  $(x_j, s_j)_{j=1}^m$  and  $(x_i, y_i)_{i=1}^n$ ,  $\mathbf{V} = 0$ 
2: repeat
3:   for  $j = 1, \dots, m$  do
4:      $E_j^S = \{s \in S(w) : \langle v_s, x_j \rangle > \langle v_{s_j}, x_j \rangle\}$ 
5:     if  $E_j^S \neq \emptyset$  then
6:        $v_{s_j} \leftarrow v_{s_j} + x_j$ 
7:       for  $s \in E_j^S$  do
8:          $v_s \leftarrow v_s - \frac{1}{|E_j^S|} x_j$ 
9:       end for
10:    end if
11:  end for
12:  for  $i = 1, \dots, n$  do
13:    if  $H_w(x_i; \mathbf{V}) \neq y_i$  then
14:       $v_{y_i} \leftarrow v_{y_i} + x_i$ 
15:       $E_i = \{y \in Y(w) : \langle v_y, x_i \rangle > \langle v_{y_i}, x_i \rangle\}$ 
16:      for  $y \in E_i$  do
17:         $v_y \leftarrow v_y - \frac{1}{|E_i|} x_i$ 
18:      end for
19:       $E_i^S = \{s \in S(w) : \langle v_s, x_i \rangle > \langle v_{s_i}, x_i \rangle\}$ 
20:      if  $E_i^S \neq \emptyset$  then
21:         $v_{s_i} \leftarrow v_{s_i} + x_i$ 
22:        for  $s \in E_i^S$  do
23:           $v_s \leftarrow v_s - \frac{1}{|E_i^S|} x_i$ 
24:        end for
25:      end if
26:    end if
27:  end for
28: until no more mistakes

```

The complete algorithm is summarized in Algorithm 2. The first part of the algorithm concerns the different nature of the two types of training data. As we explained in Section 3, the supplementary data derived from WordNet only provides annotations at the supersense level. We cannot use this information to perform updates for weight vectors v_y , but only to adjust the weights v_s . Hence for supersense-annotated training instances (x_j, s_j) we compute the error set on the supersense level as $E_j^S = \{s : \langle v_s, x_j \rangle > \langle v_{s_j}, x_j \rangle\}$ and perform the standard multiclass update step for all v_s with $s \in E_j^S \cup \{s_j\}$.

The second part concerns training on the task specific data (x_i, s_i) . If the classifier makes a mistake on pattern x_i error sets are computed for its individual components both at the synset and supersense levels (lines 15 and 19 above), which are updated according to the standard multiclass update rule.

As an example, suppose that given a pattern x_i of *chair* the synset error set is $E_i = \{\text{CHAIR}_{\text{furniture}}, \text{CHAIR}_{\text{death_chair}}\}$, while the correct label is $y_i = \text{CHAIR}_{\text{person}}$, and thus $s_i = \text{PERSON}$. The update vector $(\frac{1}{2})x_i$ is subtracted from the vectors relative to the labels in E_i , while x_i is added to v_{y_i} . If at the supersense level the error set $E_i^S = \{\text{ARTIFACT}\}$, x_i is subtracted from the vector for ARTIFACT and added to v_{s_i} . Therefore, through the supersense weight vectors, the background data affects classification at the synset level.

5 Data Set and Features

5.1 The Senseval data

We tested our system on a standard word sense disambiguation data set. The training and test data are those used in the last Senseval workshop (Senseval-2/ACL-01, 2001), which focused exclusively on word sense disambiguation. The training set consists of 8,611 paragraphs that contain an ambiguous word whose sense has been manually annotated. The inventory of senses is taken from WordNet. Similarly, the test set consists of 4,328 unlabeled pairs. We only ran experiments on the noun data, which consists of 3,512 training instances and 1,754 test instances. Each instance consists of a short passage taken from one of various sources: e.g., the Wall Street Journal, British National Corpus, and web pages. The task-specific training data, T_Y , is typically smaller than the general one, T_S . The average ratio $|T_S|/|T_Y|$ is equal to 20.3.

5.2 Features

We used the same feature set described in [Yoong and Hwee, 2002], which is compact but includes most of the features that have been found useful in this task: surrounding words, bigrams and trigrams, and syntactic information. Yoong and Hwee report results for several classifiers broken down by part of speech, which makes it possible to compare our system's performance with that of several others.

There are four types of features. The following sentence serves to illustrate them: "the dinner table and *chairs* are elegant yet comfortable". The feature set is described in greater details in [Yoong and Hwee, 2002]:

- part of speech of the neighboring words: $P_{-1} = \text{CC}$, $P_0 = \text{NNS}$, $P_{+1} = \text{AUX}$, ...
- single words in the surrounding context: $C = \text{elegant}$, $C = \text{dinner}$, $C = \text{table}$, $C = \text{the}$, ...
- bigrams and trigrams: $C_{-1,+1} = \text{and_are}$, $C_{-2,-2} = \text{table}$, $C_{+1,+2} = \text{are_elegant}$, ...
- head of the syntactic phrase that governs the target: $G = \text{are}$, $G_{\text{POS}} = \text{AUX}$, $G_{\text{VOICE}} = \text{active}$, $G_{\text{RELPOS}} = \text{left}$.

Syntactic features and part of speech tags were extracted from the syntactic parse trees of the Senseval-2 training and

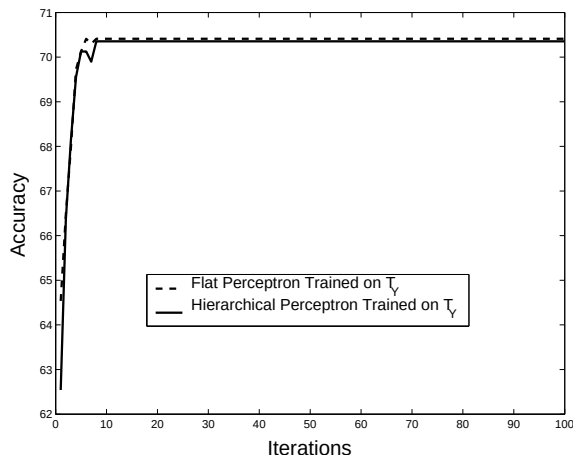


Figure 2. Test accuracy of the flat multiclass perceptron (dashed line) and the hierarchical multiclass perceptron (continuous line) on the word sense evaluation data set.

test data produced using Charniak’s parser [Charniak, 2000]. In this way we created the training data T_Y from the Senseval data. In exactly the same way we extracted features from the example sentences in WordNet to produce the additional training set for the supersense-level classes, T_S . Overall there are around 250,000 features.

6 Experiments

6.1 Experimental setup

We tested two models described in Section 4: the flat multiclass perceptron, trained and tested at the synset level, and the hierarchical one, trained on both the standard synset data and the training data for the supersenses extracted from WordNet. We also trained and tested a simple “flat” naive Bayes classifier. A different classifier was trained and tested for each word. We treated compounds such as *easy chair* and *chair* as different words.

All the results we report are given as accuracy:

$$\text{score} = \frac{\text{number of correct guesses}}{\text{number of test items}} \times 100$$

6.2 Results

Figure 2 shows the performance of the flat perceptron (dotted line) during each iteration. The perceptron in fact converges very quickly. This is probably due to the fact that there are relatively few training items: Normally the size of T_Y is between one and two hundred. To check whether an improvement was due to the algorithm alone and not to the combination of the algorithm and the additional, supersense data set, we also trained a hierarchical perceptron exclusively on the synset data. Figure 2 also plots the performance of the hierarchical perceptron trained only on T_Y . The two curves are virtually indistinguishable, meaning that without additional information not much can be gained from using the hierarchical classifier alone. In other words, with “flat” data a “flat” classifier is as good as a “hierarchical” one.

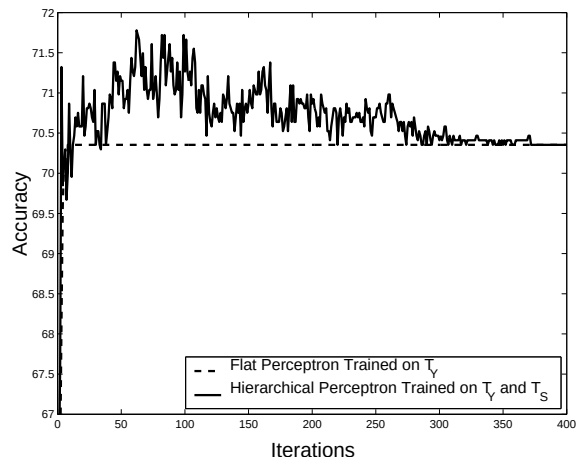


Figure 3. Test accuracy of the hierarchical (continuous line) vs. flat (dashed line) multiclass perceptron. The hierarchical multiclass perceptron was trained using supplementary supersense training data.

Figure 3 plots the performances of the flat and the hierarchical perceptron when also trained on T_S . The two patterns are very different. The hierarchical model converges only after more than 350 iterations.

Method	Score
Yoong’s AdaBoost	69.2
Best S2	69.5
Naive Bayes	68.0
Multiclass perceptron	70.4
Hierarchical multiclass perceptron	71.8

Table 1. Test accuracy on the Senseval-2 test data.

This might be due to several facts. First, the amount of data is much greater due to the addition of T_S , and it takes longer to learn. Second, the supersense data and the synset data are probably very different and noisy; as a consequence the weight vectors are continually readjusted, possibly along very different dimensions. The interesting thing, though, is that even in the midst of very wide oscillations there is a clear improvement, particularly between 50 and 100 iterations.

We present also a comparative table. Table 1 illustrates the results of our systems and other state-of-the-art word sense disambiguation ones. We set the number of iterations to a fixed number for all words equal to 100. Given that we set this value “knowing” that it is a good one for both our systems, our results and those of other systems are not really comparable. However, it is reasonable to expect that it is possible to set this stopping criterion well enough using held out data. Thus this comparison gives us an approximate idea of where our systems stand with respect to state-of-the-art ones in terms of performance. AdaBoost is the classifier that gave the best result on nouns in [Yoong and Hwee, 2002], Best S2 [Mihalcea and Moldovan, 2001] refers to the best-performing system on nouns among the Senseval-2 workshop systems. These results show that our systems’ performance is

noun	F	H	noun	F	H
stress	48.7	53.9	child	63.5	65.1
church	73.8	76.9	bar	73.4	69.1
mouth	65.5	69.0	day	68.4	69.1
sense	59.0	64.1	post	45.6	51.5
art	61.2	57.6	fatigue	81.0	88.1
chair	81.9	83.3	bum	73.3	80.0

Table 2. Example results on a few words. F = flat, H = hierarchical.

comparable to that of state-of-the-art ones and that our hierarchical model trained on background and specific data outperforms the flat one.

Results for a few individual words are presented in Table 2. They show that the improvements are not uniform, but vary from word to word. Overall we identified 105 nouns. The great majority of these are compounds that typically occur only once in the test data. Both systems achieve approximately the same score on these data. On the bulk of the test data, however, the systems perform differently. Of the 21 test words on which the classifiers achieve different scores, the hierarchical perceptron is more accurate than the flat one on 15 words, or 71.5% of the time.

This finding suggests a simple improvement for the hierarchical system. The contribution of the individual components of the classifier could be weighted setting the weights, after training, using held out data. In the simplest setting binary weights could be used; e.g., either the background information is used or not. Thus the background model would be used only when useful, otherwise its contributions would be ignored.

7 Conclusion

We have presented a learning architecture for lexical semantic classification that supplements task-specific training data with background data encoding general “world knowledge” extracted from a widely used broad-coverage, machine-readable dictionary. The model integrates task-specific and general information through a novel hierarchical learning architecture based on the multiclass perceptron. Experiments on a word sense disambiguation task showed that the hierarchical model achieves improved performance over a state-of-the-art standard “flat” system.

This new framework has a number of promising extensions. Additional accuracy gains are expected by using more sophisticated perceptron learning algorithms such as the voted perceptron [Freund and Schapire, 1998] and by using the dual perceptron with non-linear kernels. We have only made use of the simplest possible form of hierarchy (two-stage), in reality the hierarchical structure of WordNet is very complex and much more informative. The model presented here can be extended to include this type of structure as well as other sources of information. In addition, the two-layer model can be applied to all other open-class words in WordNet and a full-hierarchy-based model could be applied to verbs and nouns. There is also more information to extract from WordNet, for example, from the glosses, which can potentially be utilized as additional training data. Lastly, the

ideas we presented here might be used with other learning methods. We leave these topics for future research.

References

- [Charniak, 2000] E. Charniak. A Maximum-Entropy-Inspired Parser. In *Proceedings of the 1st Meeting of the North American Chapter of the Association for Computational Linguistics, NAACL-00*, 2000.
- [Ciaramita, 2002] M. Ciaramita. Boosting Automatic Lexical Acquisition with Morphological Information. In *Proceedings of the Workshop on Unsupervised Lexical Acquisition, ACL-02*, 2002.
- [Clark and Weir, 2002] S. Clark and D. Weir. *Class-Based Probability Estimation using a Semantic Hierarchy*. Computational Linguistics, 28, 2002.
- [Collins, 2002] M. Collins. Ranking Algorithms for Named-Entity Extraction: Boosting and the Voted Perceptron. In *Proceedings of the 40th Meeting of the Association for Computational Linguistics, ACL-02*, 2002.
- [Crammer and Singer, 2002] K. Crammer and Y. Singer. Ultraconservative On line Algorithms for Multiclass Problems. Technical Report [2001-18], School of Computer Science and Engineering, Hebrew University, Jerusalem, Israel, 2002.
- [Dumais and Chen, 2000] S. Dumais and H. Chen. Hierarchical Classification of Web Content. In *Proceedings of SIGIR-00, 23rd ACM International Conference on Research and Development in Information Retrieval*, 2000.
- [Fellbaum, 1998] C. Fellbaum. *WordNet: An Electronic Lexical Database*. MIT Press, Cambridge, Massachusetts, 1998.
- [Freund and Schapire, 1998] Y. Freund and R.E. Schapire. *Large Margin Classification Using the Perceptron Algorithm*. Machine Learning, 37, 1999.
- [Hearst, 1992] M. Hearst. Automatic Acquisition of Hyponyms from Large Text Corpora. In *Proceedings of the 14th International Conference on Computational Linguistics*, 1992.
- [Hofmann et al., 2002] T. Hofmann, I. Tsochantaridis and Y. Altun. Learning over Discrete Output Spaces via Joint Kernel Functions. In *Advances in Neural Information Processing Systems, Workshop on Kernel Methods*, 2002.
- [Light and Greif, 2002] M. Light and W. Greiff. *Statistical Models for the Induction and Use of Selectional Preferences*. In Cognitive Science, 87, 2002.
- [Mihalcea and Moldovan, 2001] R. Mihalcea and D.I. Moldovan. Pattern Learning and Automatic Feature Selection for Word Sense Disambiguation. In *Proceedings of the Second international Workshop on Evaluating Word Sense Disambiguation Systems (SENSEVAL-2)*, 2001.
- [Riloff and Jones, 1999] E. Riloff and R. Jones. Learning Dictionaries for Information Extraction by Multi-Level Bootstrapping. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, 1999.
- [Toutanova et al., 2001] K. Toutanova, F. Chen, K. Popat and T. Hofmann. Text Classification in a Hierarchical Mixture Model for Small Training Sets. In *Proceedings of the 10th International Conference on Information and Knowledge Management*, 2001.
- [Yoong and Hwee, 2002] K.L. Yoong and T.N. Hwee. An Empirical Evaluation of Knowledge Sources and Learning Algorithms for Word Sense Disambiguation. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP-2002)*, 2002.

GRAEL: an agent-based evolutionary computing approach for natural language grammar development

Guy De Pauw

CNTS - Language Technology Group

UIA - University of Antwerp

Antwerp - Belgium

guy.depauw@ua.ac.be

Abstract

This paper describes an agent-based evolutionary computing technique called GRAEL (Grammar Evolution), that is able to perform different natural language grammar optimization and induction tasks. Two different instantiations of the GRAEL-environment are described in this paper: in GRAEL-1 large annotated corpora are used to bootstrap grammatical structure in a society of agents, who engage in a series of communicative attempts, during which they redistribute grammatical information to reflect useful statistics for the task of parsing. In GRAEL-2, agents are allowed to mutate grammatical information, effectively implementing grammar rule discovery in a practical context. A combination of both GRAEL-1 and GRAEL-2 can be shown to provide an interesting all-round optimization for corpus-induced grammars.

1 Introduction

Evolutionary computing has seen many interesting applications on a broad range of research domains over the years. Its ability to overcome the problem of local maxima in finding a solution to a particular problem, by recombining and mutating individuals in a society of possible solutions, has made it an attractive technique for problems involving large, complicated and non-linearly divisible search spaces. The evolutionary computing paradigm has however always seemed reluctant to deal with natural language syntax, probably because it is essentially a recursive, non-propositional system, dealing with complex issues such as long-distance dependencies and constraints. This has made it difficult to incorporate it in typically propositional evolutionary systems such as genetic algorithms.

A limited amount of GA-related syntactic research has focused on linguistic data [Smith and Witten, 1996; Wyard, 1991; Antonisse, 1991; Araujo, 2002], but none of these systems are suited to a generic (treebank) grammar optimization task, mainly because the grammatical formalism and evolutionary processes underlying these systems are designed to fit a particular task, such as information retrieval [Losee, 1995]. Other work on syntax in the evolutionary computing paradigm has either been involved in studying the ori-

gins of grammar in a computational context [Batali, 2002; Kirby, 2001] or the co-ordinated co-evolution of grammatical principles [Briscoe, 1998]. Yet so far, little or no progress has been achieved in evaluating evolutionary computing as a tool for the induction or optimization of data-driven parsing techniques.

The GRAEL¹ environment provides a suitable framework for the induction and optimization of any type of grammar for natural language in an evolutionary setting. In this paper we hope to provide an overview of GRAEL as a grammar optimization and induction technique. We will first outline the basic architecture of the GRAEL environment in Section 2 on the basis of a toy example. Next, we introduce GRAEL-1 (Section 3) as a grammar optimization technique that can enhance corpus-induce grammars. By adding an element of mutation in GRAEL-2 we implement a method to extend the coverage of a corpus-induced grammar. We will also describe a combination of both GRAEL-1 and GRAEL-2 which can be shown to provide an interesting all-round optimization technique for corpus-induced grammars.

2 GRAEL - Grammar Evolution

A typical GRAEL society contains a population of agents in a virtual environment. Each of these agents holds a number of structures that allows it to produce sentences as well as induce a probabilistic grammar to analyze other agents' sentences. During an extended series of error-driven inter-agent interactions, these grammars are updated over time. While the evolutionary computing approach of GRAEL is able to define the quality of the grammars that are developed over time, the agent-based aspect of GRAEL ensures that the grammar optimization is grounded in the practical task of parsing itself. From an engineering point of view, GRAEL provides a general framework for grammar optimization and induction, but from a more theoretical point of view, GRAEL can also help us to understand the dynamics of grammar emergence and evolution over time.

In the data-driven GRAEL experiments described in this paper, the grammatical knowledge of the agents in the society is bootstrapped by using an annotated natural language corpus [Marcus *et al.*, 1993]. At the onset of such a data-driven

¹GRAMMAR EvoLution

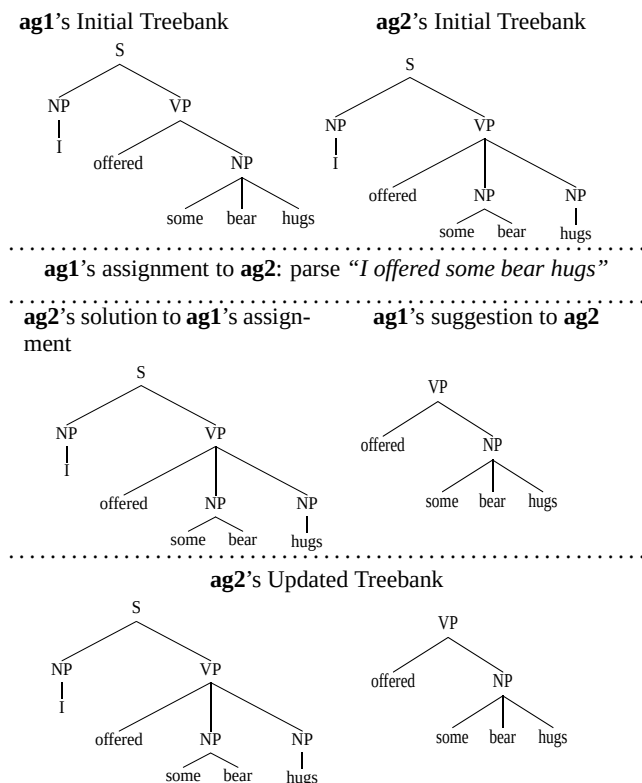


Figure 1: A GRAEL language game

GRAEL society, the syntactic structures of a treebank are randomly distributed over the agents, so that each agent holds a number of tree-structures in memory. These structures enable them to generate sentences, as well as provide grammars that allow them to analyze other agents' sentences.

The actual interaction between agents is implemented in *language games*: an agent (**ag1**) presents a sentence to another agent (**ag2**). If **ag2** is able to correctly parse **ag1**'s sentence, the communication is successful. If on the other hand, **ag2** is lacking the proper grammatical information to parse the sentence correctly, **ag1** shares the necessary information for **ag2** to arrive at the proper solution.

Figure 1 displays a toy example of such a language game. In this example, a "treebank" of two structures has been distributed over a society of two agents. The two agents engage in a language game, in which **ag1** presents the sentence "I offered some bear hugs" to **ag2** for parsing. At this point in time, **ag2**'s grammar does not contain the proper grammatical information to interpret this sentence the way **ag1** intended and so **ag2** will return an incorrect parse, even though it is consistent with its own grammar.

Consequently, **ag1** will try and help **ag2** out by revealing the minimal correct substructure of the correct parse that should enable **ag2** to arrive at a better solution. This information is incorporated in **ag2**'s grammar, who will try to parse the sentence again with the updated knowledge. When **ag2** is able to provide the correct analysis (or is not able to after a certain number of attempts) either **ag1**'s next sentence will

be parsed, or two other agents in the GRAEL society will be randomly selected to play a language game.

These interactions, which introduce a concept of error-driven knowledge sharing, extend the agents' grammars fast, so that the datasets can grow very large in a short period of time. A generation-based GRAEL society can be used to allow the society to purge itself of bad agents and build new generations of good parser agents, who contain a fortuitous distribution of grammatical knowledge. This involves the use of fitness functions that can distinguish good agents from bad ones. For a full overview of all the evolutionary parameters in the GRAEL environment, many of which have a significant impact on processing, we would like to refer to [De Pauw, 2002]. In this paper, we will describe the most relevant subset of experiments that allows us to evaluate GRAEL as a grammar induction and optimization technique.

3 GRAEL-1: Probabilistic Grammar Optimization

Historically, most syntactic parsers for natural language have made use of hand-written grammars, consisting of a laboriously crafted set of grammar rules. But in recent years, a lot of research efforts employ annotated corpora to automatically induce grammars [Bod, 1998; Collins, 1999; De Pauw, 2000]. Yet, data-analysis of the output generated by these parsers still brings to light fundamental limitations to these corpus-based methods [Klein and Manning, 2001]. Even though generally providing a much broader coverage than hand-built grammars, corpus-induced grammars may still not hold enough grammatical information to provide parses for a large number of sentences, as some rules that are needed to generate the correct tree-structures are not induced from the original corpus (cf. Section 4).

But even if there were such a thing as a full-coverage corpus-induced grammar, performance would still be limited by the probabilistic weights attributed to its rules. A typical data-driven parser provides a huge collection of possible parses for any given sentence. Fortunately, we can also induce useful statistics from the annotated corpus that provides a way to order these parse forests to express a preference for a particular parse. These statistics go a long way in providing well ordered parse forests, but in many other cases, it can be observed that the ranking of the parse forest is sometimes counter-intuitive in that correct constructs are often overtaken by obviously erroneous, but highly frequent structures.

This can easily be explained by the inherent nature of corpus-based grammars: the initial probabilistic values attached to the grammar rules induced from the annotated data are equal to their relative frequency in the corpus. It might be the case however that, even though they are directly induced from the annotated corpus, the probabilities of these rules are not suited to the disambiguation task as yet. It may therefore be useful to have the parser use the grammar to *practice* the task of parsing and adjust the probabilistic weights of particular structures according to these test cases. We then need to consider the initial grammar as basic raw material in need of optimization, as it is merely a reflection of the original data set and is not yet optimized for the task of parsing (unseen)

sentences itself.

Typical methods of probabilistic grammar optimization include, among others, bagging and boosting [Henderson and Brill, 2000; Collins, 2000], re-estimation of the constituents probabilities [Goodman, 1998; Charniak, 2000] and including extra information sources [Belz, 2001; Collins, 1999]. But we propose an agent-based evolutionary computing method to resolve this issue. Grammar optimization using a GRAEL environment is in this vein related to the aforementioned bagging approach to grammar optimization, albeit with some notable differences. By distributing the knowledge over a group of agents and having them interact with each other, we basically create a multiple-route model for probabilistic grammar optimization. Grammatical structures extracted from the training corpus, will be present in different quantities and variations throughout the GRAEL society. While the agents interact with each other and in effect practice the task on each other’s grammar, a varied range of probabilistic grammars are optimized in a situation that directly relates to the task at hand. The evolutionary aspects of the system make sure that, while marginally useful grammatical information is down-toned, common constructs are enforced, providing a better balanced model for statistical parsing.

The way GRAEL accomplishes a re-distribution of the original probabilistic values is by using the default GRAEL architecture described in Section 2. This type of error-driven learning makes sure that mistakes are being dealt with by transferring difficult grammatical constructs, thereby increasing their probabilistic value in the other agent’s grammar. This probabilistic adjustment will be taken into account during subsequent parsing attempts by this agent, hopefully triggering the correct grammatical structure in the future.

3.1 Experimental Setup

The overall setup of the GRAEL experiments is displayed in Figure 2. Baseline accuracy is measured by directly inducing a grammar from the training set to power Parser 1, which disambiguates the test set. This grammar takes on the form of a PMPG as outlined in [De Pauw, 2000] (cf. *infra*). The same training set is also randomly and equally distributed over a number of agents in the GRAEL society, who will consequently engage in a number of *language games*. At some point, established by the halting procedure (cf. *infra*), the society is halted and the fittest agent is selected from the society. This agent effectively constitutes a redistributed and probabilistically optimized grammar, which can be used to power Parser 2. GRAEL-1 accuracy is achieved by having this parser disambiguate the same test set.

We used two data sets from the Penn Treebank [Marcus *et al.*, 1993]. The main batch of experiments was conducted on the small, homogeneous ATIS-corpus, which consists of a collection of annotated sentences recorded by a spoken-dialogue system. The larger Wall Street Journal Corpus (henceforth WSJ), a collection of annotated newspaper articles, was used to test the system on a larger scale corpus. The common division between training set (Section 02-21) and test set (Section 23) was used. Semantically oriented flags and numeric flags indicating internal relations were removed to allow for more streamlined syntactic processing.

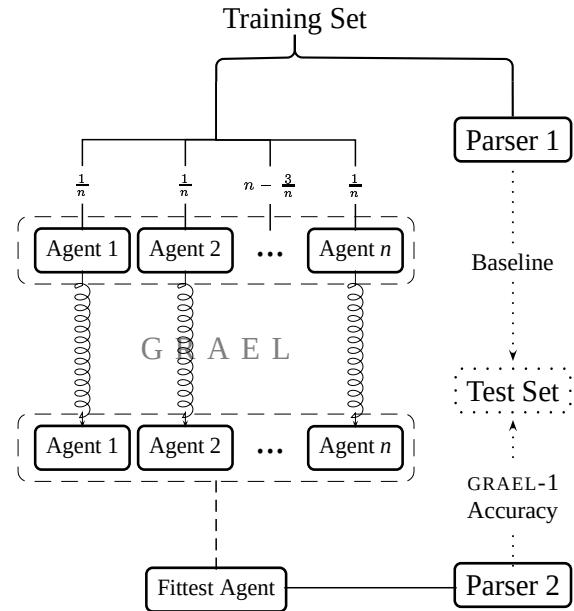


Figure 2: Comparing parsers: baseline parser and GRAEL

For syntactic processing, the agents use the parsing system PMPG described in [De Pauw, 2000], which integrates a CKY parser [Chappelier and Rajman, 1998] and a parse forest ranking scheme that employs probabilistic information as well as a memory-based operator to maximize for each parse the number of nodes that can be retrieved from memory. A PMPG takes the form of a simple PCFG-type grammar, enriched with numerical indices that encode contextual information previously observed in a treebank. This memory-based instantiation of Data-Oriented Parsing [Bod, 1998] ensures that larger syntactic structures are used as the basis for parsing, with a minimal loss of computational efficiency over regular PCFGs. The PMPG approach can therefore be considered to introduce a psycho-linguistically relevant memory-based operator in the parsing process.

The full experimental run varied society sizes, generation methods, fitness functions and halting procedures [De Pauw, 2002]. The subset of experiments described in this paper employed the sexual procreation method to introduce new generations by combining the grammars of two fit agents in the society to create new generations of parser agents. The fitness of an agent is defined by recording a weighted average of the F-score during inter-agent communication (also see Figure 3) and the F-score of the agent’s parser on a held-out validation set. This information was also used to try and halt the society at a global maximum and select the fittest agent from the society. For computational reasons, the experiments on the WSJ-corpus were limited to two different population sizes and used an approximation of GRAEL that can deal with large datasets in a reasonable amount of time. The test set was not used in any way during actual GRAEL-processing in agreement with blind-testing procedures.

	ATIS		WSJ	
	Exact Match	$F_{\beta=1}$ -score	Exact Match	$F_{\beta=1}$ -score
Baseline	70.7	89.3	16.0	80.5
GRAEL (5)	72.4	90.9	—	—
GRAEL (10)	77.6	92.1	—	—
GRAEL (20)	77.6	92.1	—	—
GRAEL (50)	75.9	92.2	22.2	80.7
GRAEL (100)	75.9	92.0	22.8	81.1
bagging (15)	75.9	91.8	20.0	81.4
boosting (15)	75.9	91.7	21.0	81.0

Table 1: Baseline vs GRAEL-1 results

3.2 Results

Table 1 displays the exact match accuracy and F-scores for the baseline model, a standard PMPG parser using a grammar induced from the training set (cf. Figure 2). It also displays scores of the GRAEL system for different population sizes. We notice a significant gain for all GRAEL models over the baseline model on the ATIS corpus, but increasing population size over 20 agents seems to decrease exact match accuracy on the ATIS corpus. Likewise, the small society of 5 agents achieves only a very limited improvement over the baseline method. Data analysis showed that the best moment to halt the society and select the fittest agent from the society, is a relatively brief period right before actual convergence sets and grammars throughout the society are starting to resemble each other more closely. The size of the the society seems to be the determining factor controlling the duration of this period.

Preliminary tests on a subset of the WSJ corpus had shown that society sizes of 20 agents and less to be unsuitable for a large-scale corpus, again ending up in a harmful premature convergence. The gain achieved by the GRAEL society is less spectacular than on the ATIS corpus, but it is still statistically significant. Larger society sizes and full GRAEL processing on the WSJ corpus should achieve a more significant gain.

The experiments do show however, that GRAEL-1 is indeed an interesting method for probabilistic grammar redistribution and optimization. Data analysis shows that many of the counter-intuitive parse forest orderings that were apparent in the baseline model, are being resolved after GRAEL-1 processing. It is also interesting to point out that we are achieving an error reduction rate of more than 26% over the baseline method, without introducing any new grammatical information in the society, but solely by redistributing what is already there. These experimental results indicate that annotated data can indeed be considered as raw material that can be optimized for the practical use of parsing unseen data.

[De Pauw, 2002] also describes experiments that directly compare GRAEL to the similar methods of bagging and boosting [Henderson and Brill, 2000], which are summarized in the bottom two lines of Table 1. Bagging and boosting were shown to obtain significantly lower accuracy figures on almost all accounts. Only the F-score for the bagging experiment exceeded that of the optimal GRAEL configuration, but this can be attributed to the fact that an approximation of GRAEL was used for full processing on the extensive WSJ

dataset.

4 GRAEL-2: Grammar Rule Discovery

The functionality of GRAEL-1 can be extended by only applying minor alterations to the GRAEL system. With GRAEL-2 we wish to provide a grammar rule discovery method which can deal with the problem of *grammar sparseness*. Hand-written and corpus-induced grammars alike have to deal with the fundamental issue of coverage. [Collins, 1999] for example reports that when using sections 2-21 of the WSJ-corpus as a training set and section 23 as a test set, 17.1% of the sentences in the test set require a rule not seen in the training set. Even for a large corpus such as the WSJ, sparse grammar is indeed a serious accuracy bottleneck.

It would therefore be useful to have a method that can take a corpus-induced grammar and extend it by generating new rules. But doing so in a blind manner, would provide huge, over-generating grammars, containing many nonsensical rules. The GRAEL-2 system described in this section, involves a *distributed* approach to this type of grammar rule discovery. The original (sparse) grammar is distributed among a group of agents, who can randomly *mutate* the grammatical structures they hold. The new grammatical information they create is tried and tested by interacting with each other. The neo-darwinist aspect of this evolutionary system will make sure that any useful mutated grammatical information is retained throughout the population, while noise is filtered out over time. This method provides a way to create new grammatical structures previously unavailable in the corpus, while at the same time evaluating them in a practical context, without the need for an external information source.

To accomplish this, we need to implement some minor alterations to the GRAEL-1 system. The most important adjustment occurs during the language games. We refer back to the toy example of the language game in Figure 1 to the point where **ag1** suggests the minimal correct substructure to **ag2**. In GRAEL-1 this step introduced a form of error-driven learning, making sure that the probabilistic value of this grammatical structure is increased. The functionality of GRAEL-2 however is different: in this step, we assume that there is a noisy channel between **ag1** and **ag2** which may cause **ag2** to misunderstand **ag1**'s structure. Small mutations on different levels of the substructure may occur, such as the deletion, addition and replacement of nodes. This effectively introduces previously unseen grammatical data in the GRAEL society, which

	ATIS		WSJ	
	$F_{\beta=1}$	Ex. Match	$F_{\beta=1}$	Ex. Match
Baseline	69.8	0	80.5	16
grael-1	73.8	0	81.4	22.8
grael-2	83.0	7.2	76.5	19.3
grael2+1	85.7	11.3	81.6	23.4

Table 2: Baseline vs GRAEL-1 vs GRAEL-2 vs GRAEL2+1 Results

will consequently be optimized over time.

Preliminary experiments however showed that this does not work as such, since the newly created structures were largely being ignored in favor of the gold-standard corpus structures. We therefore implemented another alteration to the GRAEL-1 system. Instead of just presenting the tree-structure originally assigned by the training set, we now require **ag1** to parse the string-only sentence using the grammar acquired during language games, replacing the tree-structure from the training set, with a possibly different tree-structure that incorporates some of the mutated information. This alteration makes sure that the mutated grammatical structures are actively being used, so that their usefulness as grammatical constructs can be measured in a practical context.

Experimental Setup and Results

The setup for the GRAEL-2 experiments is the same as for the GRAEL-1 experiments (cf. Figure 2). To test the grammar-rule discovery capabilities of GRAEL-2 we have compiled a special worst-case scenario test set for the ATIS corpus, consisting of the 97 sentences in the ATIS that require a grammar rule that cannot be induced from the training set. For the WSJ-experiment the normal test set was used. A 20-agent and a 100-agent society were respectively used for the ATIS and WSJ experiments.

The baseline and GRAEL-1 methods for the ATIS experiments trivially have an exact match accuracy of 0%, which also has a negative effect on the F-score (Table 2). GRAEL-2 is indeed able to improve on this significantly, proving that it is indeed an effective grammar rule discovery method. Data-analysis shows however that it has lost the beneficial probabilistic optimization effect of GRAEL-1.

We therefore performed another experiment, in which we turned the GRAEL-2 society into a GRAEL-1 society after the former’s halting point. In other words: we take a society of agents using mutated information and consequently apply GRAEL-1’s probabilistic redistribution properties on these grammars. Figure 3 shows the course of the GRAEL2+1 experiment. In this figure we see the F-scores recorded during inter-agent communicative attempts. After an almost linear increase during GRAEL-2 processing, the society is halted after an extended period of convergence. Next, GRAEL-1 processing resumes, which negatively affects F-scores for a brief period of time, until the society reconverges. Even though the F-scores do not seem to improve over those observed before the transition to GRAEL-1, Table 2 shows that the fittest agent selected from the society after the transition performs better on the held-out test set. The results on the WSJ-corpus are also

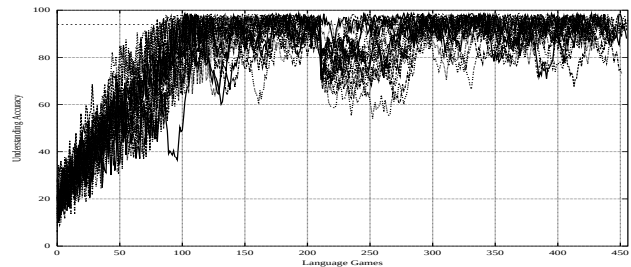


Figure 3: GRAEL2+1 Experiment - F-scores during language games

interesting in this respect. GRAEL-1 outperforms GRAEL-2 on this data set, but the combination of the two seems to be quite beneficial for parsing accuracy.

Evaluating a grammar rule discovery method poses an empirical problem in that it can never be clear what grammar rules are missing until we actually need them. The test set we compiled to perform the GRAEL-2 ATIS-experiments goes a long way in providing a touchstone to see how well GRAEL-2 performs as a supervised grammar induction method. And results indicate it performs quite well: mutated information becomes available that is able to create parses for difficult constructions, while the number of structures that constitute noise is limited and is attributed a small enough portion of the probability mass as not to stand in the way of actual useful mutated structures.

5 Concluding Remarks

This paper has presented one of the first research efforts that introduces agent-based evolutionary computing as a machine learning method for data-driven grammar optimization and induction. In recent years, many researchers have employed ensemble methods to overcome any negative bias their training data might impose on their classifiers. It is indeed important to view (annotated) data, not as an optimally distributed set of examples but as raw material that needs to be pre-processed before it can be used by a machine learning classifier. The bagging and boosting approach for instance, tries to create resamplings of the original data, to overcome the local maxima the data might restrict the classifier to, but we believe GRAEL adds an extra dimension to the task: by splicing the data and incorporating it in a society of communicating agents, we allow for the parallel development of several grammars at once, enhanced in a practical context that mirrors the goal itself: parsing unseen data.

We described two instantiations of the GRAEL environment. The basic GRAEL-1 system aims to provide a beneficial re-distribution of the probability mass of a probabilistic grammar. By using a form of error-driven learning in the course of language games between agents, probabilistic values are adjusted in a practical context. This optimizes the grammars for the task of parsing data, rather than reflecting the probability mass of the initial data set. This method favorably compares to established grammar optimization methods like bagging and boosting.

By adding an element of mutation to the concept of

GRAEL-1 we were able to extend its functionality and experiment on GRAEL-2 as a grammar rule discovery method. The results showed that GRAEL-2 could produce a broader-coverage grammar, but that GRAEL-1's ability to optimize the distribution of the probability mass of a grammar was counteracted. A grammar obtained from a GRAEL-2 society is therefore unsuited to be directly applied to parsing. But combining it again with a GRAEL-1 society however, goes a long way in resolving this issue, providing a grammar that has a broader coverage, as well as a better tuned probability mass distribution over the structures contained therein. We therefore believe the combination of GRAEL-2 and GRAEL-1 to be an interesting optimization toolkit for any given grammar, and corpus-induced grammars in particular. It can achieve a significant optimization over the baseline method, without using an external information source, simply on the basis of knowledge transfer and mutation in a practical and evolutionary context.

Acknowledgments

The research described in this paper was financed by the FWO (Fund for Scientific Research).

References

- [Antonisse, 1991] H. James Antonisse. A grammar-based genetic algorithm. In Gregory J. E. Rawlings, editor, *Foundations of genetic algorithms*, pages 193–204. Morgan Kaufmann, San Mateo, 1991.
- [Araujo, 2002] Lourdes Araujo. A parallel evolutionary algorithm for stochastic natural language parsing. In *Proceedings of The Seventh International Conference on Parallel Problem Solving From Nature*, pages 700–709, Granada, Spain, 2002.
- [Batali, 2002] J. Batali. The negotiation and acquisition of recursive grammars as a result of competition among exemplars. In Ted Briscoe, editor, *Linguistic Evolution through Language Acquisition: Formal and Computational Models*, chapter 5. Cambridge University Press, 2002.
- [Belz, 2001] A. Belz. Optimisation of corpus-derived probabilistic grammars. In *Proceedings of Corpus Linguistics 2001*, pages 46–57, Lancaster University, UK, 2001.
- [Bod, 1998] R. Bod. *Beyond Grammar—An Experience-Based Theory of Language*. Cambridge University Press, Cambridge, England, 1998.
- [Briscoe, 1998] E. Briscoe. Language as a complex adaptive system: co-evolution of language and of the language acquisition device. In *Proceedings of the 8th Meeting of Comp. Linguistics in the Netherlands*, pages pp. 3–40., Amsterdam, 1998. Rodopi.
- [Chappelier and Rajman, 1998] J.-C. Chappelier and M. Rajman. A generalized cyk algorithm for parsing stochastic cfg. In *Proceedings of Tabulation in Parsing and Deduction (TAPD'98)*, pages 133–137, Paris (FRANCE), 1998.
- [Charniak, 2000] Eugene Charniak. A maximum-entropy-inspired parser. In *Proceedings of NAACL'00*, pages 132–139, Seattle, USA, 2000.
- [Collins, 1999] M. Collins. *Head-driven Statistical Models for Natural Language Parsing*. PhD thesis, University of Pennsylvania, Pennsylvania, USA, 1999.
- [Collins, 2000] Michael Collins. Discriminative reranking for natural language parsing. In *Proc. 17th International Conf. on Machine Learning*, pages 175–182. Morgan Kaufmann, San Francisco, CA, 2000.
- [De Pauw, 2000] G. De Pauw. Aspects of pattern-matching in dop. In *Proceedings of the 18th International Conference on Computational Linguistics*, pages 236–242, 2000.
- [De Pauw, 2002] G. De Pauw. *An Agent-Based Evolutionary Computing Approach to Memory-Based Syntactic Parsing of Natural Language*. PhD thesis, University of Antwerp, Antwerp, Belgium, 2002.
- [Goodman, 1998] J. Goodman. *Parsing Inside-Out*. PhD thesis, Harvard University, Massachusetts, USA, 1998.
- [Henderson and Brill, 2000] J. Henderson and E. Brill. Bagging and boosting a treebank parser. In *Proceedings of the 1st Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL-2000)*, pages 34–41, 2000.
- [Kirby, 2001] S. Kirby. Spontaneous evolution of linguistic structure: an iterated learning model of the emergence of regularity and irregularity. *IEEE Transactions on Evolutionary Computation*, 5(2):102–110, 2001.
- [Klein and Manning, 2001] Dan Klein and Christopher D. Manning. Parsing with treebank grammars: Empirical bounds, theoretical models, and the structure of the penn treebank. In *Proceedings of ACL-EACL 2001*, pages 330–337, 2001.
- [Losee, 1995] R.M. Losee. Learning syntactic rules and tags with genetic algorithms for information retrieval and filtering: An empirical basis for grammatical rules. *Information Processing and Management*, 32(2):185–197, 1995.
- [Marcus et al., 1993] M. P. Marcus, B. Santorini, and M.A. Marcinkiewicz. Building a large annotated corpus of english: the penn treebank. *Computational linguistics*, 19:313–330, 1993. Reprinted in Susan Armstrong, ed. 1994, *Using large corpora*, Cambridge, MA: MIT Press, 273–290.
- [Smith and Witten, 1996] T. C. Smith and I. H. Witten. Learning language using genetic algorithms. In Stefan Wermter, Ellen Riloff, and Gabriele Scheler, editors, *Connectionist, Statistical, and Symbolic Approaches to Learning for Natural Language Processing*, volume 1040 of *LNAI*, pages 132–145. Springer Verlag, Berlin, 1996.
- [Wyard, 1991] P. Wyard. Context-free grammar induction using genetic algorithms. In R. Belew and L.B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 514–518, San Mateo, 1991. ICGA, Morgan Kaufmann.

NONMONOTONIC REASONING

NONMONOTONIC REASONING

DEFAULT LOGIC

Outlier Detection Using Default Logic

Angiulli Fabrizio
ICAR-CNR c/o DEIS
Univ. della Calabria
87030 Rende (CS), Italy
angiulli@icar.cnr.it

Rachel Ben-Eliyahu - Zohary
Comm. Systems Engineering Dept.
Ben-Gurion Univ. of the Negev
Beer-Sheva 84105, Israel
rachel@bgumail.bgu.ac.il

Luigi Palopoli
DIMET
Univ. di Reggio Calabria
Loc. Feo di Vito
89100 Reggio Calabria, Italy
palopoli@ing.unirc.it

Abstract

Default logic is used to describe regular behavior and normal properties. We suggest to exploit the framework of default logic for detecting *outliers* - individuals who behave in an unexpected way or feature abnormal properties. The ability to locate outliers can help to maintain knowledge-base integrity and to single out irregular individuals. We first formally define the notion of an *outlier* and an *outlier witness*. We then show that finding outliers is quite complex. Indeed, we show that several versions of the outlier detection problem lie over the second level of the polynomial hierarchy. For example, the question of establishing if at least one outlier can be detected in a given propositional default theory is Σ_3^P -complete. Although outlier detection involves heavy computation, the queries involved can frequently be executed off-line, thus somewhat alleviating the difficulty of the problem. In addition, we show that outlier detection can be done in polynomial time for both the class of acyclic normal unary defaults and the class of acyclic dual normal unary defaults.

1 Introduction

Default logics were developed as a tool for reasoning with incomplete knowledge. By using default rules, we can describe how things work in general and then make some assumptions about individuals and draw conclusions about their properties and behavior.

In this paper, we suggest a somewhat different usage of default logics. The basic idea is as follows. Since default rules are used for describing regular behavior, we can exploit them for detecting individuals or elements who *do not* behave normally according to the default theory at hand. We call such entities *outliers*. An outlier is an element that shows some properties that are contrary to those that can be logically justified.

Outlier detection can be useful in several application contexts, e.g., to single out exceptional behaving individuals or system components. Note that according to our approach, exceptions are not explicitly listed in the theory as “abnormals,” as is often done in logical-based abduction [Poole, 1989;

Console *et al.*, 1991; Eiter and Gottlob, 1995]. Rather, their “abnormality” is singled out exactly because some of the properties characterizing them do not have a justification within the theory at hand. For example, suppose that it usually takes about two seconds to download a one-megabyte file from some server. Then, one day, the system is slower - instead four seconds are needed to perform the same task. While four seconds may indicate a good performance it is helpful to find the source of the delay. Another example might be that someone’s car brakes are making a strange noise. Although they seem to be functioning properly, this is not normal behavior and the car should be serviced. In this case, the car brakes are outliers and the noise is their witness.

Outlier detection can also be used for examining database integrity. If an abnormal property is discovered in a database, the source who reported this observation would have to be double-checked.

Detecting abnormal properties, that is, detecting outliers, can also lead to an update of default rules. Suppose we have the rule that birds fly, and we observe a bird, say Tweety, that does not fly. We report this occurrence of an outlier in the theory to the knowledge engineer. The engineer investigates the case, finds out that Tweety is, for example, a penguin, and updates the knowledgebase with the default “penguins do not fly.”

In this paper, we formally state the ideas briefly sketched above within the context of Reiter’s default logic. For simplicity, we concentrate on the propositional fragment of default logic although the generalization of such ideas to the realm of first-order defaults also worth exploring. So, whenever we use a default theory with variables, as in some of the following examples, we relate to it as an abbreviation of its grounded version.

The rest of the paper is organized as follows. In Section 2, we give preliminary definitions as well as a formal definition of the concept of an outlier. In Section 3, we describe the complexity of finding outliers in propositional default logic. Section 4 analyzes the complexity of detecting outliers in disjunction-free propositional default logics, and section 5 describes some tractable cases. Related work is discussed in Section 6. Conclusions are given in Section 7.

Because of space limitations, throughout the paper proofs of results are sketched or omitted. Full proofs can be found in [Angiulli *et al.*, 2003].

2 Definitions

In this section we provide preliminary definitions for concepts we will be using throughout the paper.

2.1 Preliminaries

The following definitions will be assumed. Let T be a propositional theory. Then T^* denotes its logical closure. If S is a set of literals, then $\neg S$ denotes the set of all literals that are the negation of some literal in S .

Default logic was introduced by Reiter [Reiter, 1980]. A *propositional default theory* Δ is a pair (D, W) consisting of a set W of propositional formulas and a set D of default rules. A *default rule* δ has the form $\frac{\phi:\psi}{\chi}$ (or, equivalently, $\phi:\psi/\chi$), where ϕ , ψ and χ are propositional formulas, called, respectively, *prerequisite*, *justification*, and *consequent* of δ . The prerequisite could be omitted, though justification and consequent are required. If $\psi = \chi$, the default rule is called *normal*. The informal meaning of a default rule δ is the following: if ϕ is known, and if it is consistent to assume ψ , then we conclude χ . An *extension* is a maximal set of conclusions that can be drawn from a theory. An extension E of a propositional default theory $\Delta = (D, W)$ can be finitely characterized through the set D_E of *generating defaults* for E w.r.t. Δ , i.e., the set $D_E = \{\phi:\psi/\chi \in D \mid \phi \in E \wedge \neg\psi \notin E\}$. Indeed, $E = (W \cup \{\chi \mid \phi:\psi/\chi \in D_E\})^*$.

Let Δ be a default theory and l a literal. Then $\Delta \models l$ means that l belongs to every extension of Δ . Similarly, for a set of literals S , $\Delta \models S$ means that every literal $l \in S$ belongs to every extension of Δ . A default theory is *coherent* if it has at least one extension.

We review some basic definitions about complexity theory, particularly, the polynomial hierarchy. The reader is referred to [Garey and Johnson, 1979] for more on complexity theory. The classes Σ_k^P and Π_k^P are defined as follows: $\Sigma_0^P = \Pi_0^P = P$ and for all $k \geq 1$, $\Sigma_k^P = NP^{\Sigma_{k-1}^P}$, and $\Pi_k^P = co-\Sigma_k^P$. Σ_k^P models computability by a nondeterministic polynomial-time algorithm which may use an oracle, loosely speaking a subprogram that can be run with no computational cost, for solving a problem in Σ_{k-1}^P . The class D_k^P , $k \geq 1$, is defined as the class of problems that consists of the conjunction of two independent problems from Σ_k^P and Π_k^P , respectively. Note that for all $k \geq 1$, $\Sigma_k^P \subseteq D_k^P \subseteq \Sigma_{k+1}^P$. A problem A is *complete* for the class $\mathcal{C}$ iff A belongs to $\mathcal{C}$ and every problem in $\mathcal{C}$ is reducible to A by polynomial-time transformations. A well known Σ_k^P -complete problem is to decide the validity of a formula $QBE_{k,\exists}$, that is, a formula of the form $\exists X_1 \forall X_2 \dots Q X_k f(X_1, \dots, X_k)$, where Q is $\exists$ if k is odd and is $\forall$ if k is even, $X_1, \dots, X_k$ are disjoint set of variables, and $f(X_1, \dots, X_k)$ is a propositional formula in $X_1, \dots, X_k$. Analogously, the validity of a formula $QBE_{k,\forall}$, that is a formula of the form $\forall X_1 \exists X_2 \dots Q X_k f(X_1, \dots, X_k)$, where Q is $\forall$ if k is odd and is $\exists$ if k is even, is complete for Π_k^P . Deciding the conjunction $\Phi \wedge \Psi$, where Φ is a $QBE_{k,\exists}$ formula and Ψ is a $QBE_{k,\forall}$ formula, is complete for D_k^P .

2.2 Defining outliers

Next we formalize the notion of an outlier in default logic. In order to motivate the definition and make it easy to under-

stand, we first look at an example.

Example 2.1 Consider the following default theory which represents the knowledge that birds fly and penguins are birds that do not fly, and the observations that Tweety and Pini are birds and Tweety does not fly.

$$D = \left\{ \frac{Bird(x) : Fly(x)}{Fly(x)}, \frac{Penguin(x) : Bird(x)}{Bird(x)}, \frac{Penguin(x) : \neg Fly(x)}{\neg Fly(x)} \right\}$$

$$W = \{Bird(Tweety), Penguin(Pini), \neg Fly(Tweety)\}$$

This theory has two extensions. One extension is the logical closure of $W \cup \{Bird(Pini), \neg Fly(Pini)\}$ and the other is the logical closure of $W \cup \{Bird(Pini), Fly(Pini)\}$.

If we look carefully at the extensions, we note that Tweety not flying is quite strange, since we know that birds fly and Tweety is a bird. Therefore, there is no apparent justification for the fact that Tweety does not fly (other than the fact $\neg Fly(Tweety)$ belonging to W). Had we been told that Tweety is a penguin, we could have explained why Tweety does not fly. But, as the theory stands now, we are not able to explain why Tweety does not fly, and, thus, Tweety is an exception. Moreover, if we are trying to nail down what makes Tweety an exception, we notice that if we would have dropped the observation $\neg Fly(Tweety)$ from W , we would have concluded the exact opposite, namely, that Tweety does fly. Thus, $\neg Fly(Tweety)$ “induces” such an exceptionality (we will call *witness* a literal like $\neg Fly(Tweety)$). Furthermore, if we drop from W both $\neg Fly(Tweety)$ and $Bird(Tweety)$, we are no longer able to conclude that Tweety flies. This implies that $Fly(Tweety)$ is a consequence of the fact that Tweety is a bird, and thus $Bird(Tweety)$ is the property of Tweety that behaves exceptionally (or the *outlier*).

From the above example, one could be induced to define an outlier as an individual, i.e., a constant, in our case *Tweety*, that possesses an exceptional property, denoted by a literal having the individual as one of its arguments, in our case $Bird(Tweety)$. However, for a conceptual viewpoint, it is much more general and flexible to single out a property of an individual which is exceptional, rather than simply the individual. That assumed, we also note that within the propositional context we deal with here, we do not explicitly have individuals distinct from their properties and, therefore, the choice is immaterial.

Based on the example and considerations mentioned above, we can define the concept of an outlier as follows.

Definition 2.2 Let $\Delta = (D, W)$ be a propositional default theory such that W is consistent and $l \in W$ is a literal. If there exists a set of literals $S \subseteq W$ such that:

1. $(D, W_S) \models \neg S$, and
2. $(D, W_{S,l}) \not\models \neg S$.

where $W_S = W \setminus S$ and $W_{S,l} = W_S \setminus \{l\}$, then we say that l is an *outlier* in Δ and S is an *outlier witness set* for l in Δ .

According to this definition, a literal l is an outlier if and only if there is an exceptional property, denoted by a set of literals S , holding in *every* extension of the theory.

The exceptional property is the outlier witness for l . Thus, according to this definition, in the default theory of Example 2.1 above we should conclude that $Bird(Tweety)$ denotes an outlier and $\{\neg Fly(Tweety)\}$ is its witness. Note that we have defined an outlier witness to be a set, not necessarily a single literal since in some theories taking a single literal does not suffice to form a witness for a given outlier being that all witnesses of such an outlier have a cardinality strictly larger than one.

Example 2.3 Consider the default theory $\Delta = (D, W)$, where the set of default rules D conveys the following information about weather and traffic in a small town in southern California:

1. $\frac{July \wedge Weekend : \neg Traffic_Jam \wedge \neg Rain}{\neg Traffic_Jam \wedge \neg Rain}$ - that is, normally during a July weekend there are no traffic jams nor any rain.
2. $\frac{January : Rain}{Rain}, \frac{January : \neg Rain}{\neg Rain}$ - in January it sometimes rains and sometimes it doesn't rain.
3. $\frac{Weekend \wedge Traffic_Jam : Accident \vee Rain}{Accident \vee Rain}$ - If there is a traffic jam in the weekend then normally it must be raining or there would have been an accident.

Suppose also that $W = \{July, Weekend, Traffic_Jam, Rain\}$. Then, the set $S = \{Traffic_Jam, Rain\}$ is an outlier witness for both $Weekend$ and $July$. Moreover, S is a *minimal* outlier witness set for either $Weekend$ or $July$, since deleting one of the members from S will render S not being a witness set.

Here is another example.

Example 2.4 Consider the following default theory Δ :

$$D = \left\{ \frac{Income(x) \wedge Adult(x) : Works(x)}{Works(x)}, \frac{FlyingS(x) : InterestTakeOff(x)}{InterestTakeOff(x)}, \frac{FlyingS(x) : InterestNavigate(x)}{InterestNavigate(x)} \right\}$$

$$W = \{Income(Johnny), Adult(Johnny), \neg Works(Johnny), FlyingS(Johnny), \neg InterestTakeOff(Johnny)\}$$

This theory claims that normally adults who have a monthly income work, and students who take flying lessons are interested in learning how to take off and navigate. The observations are that Johnny is an adult who has a monthly income, but he does not work. He is also a student in a flying school but he is not interested in learning how to take-off. Based on the events of September 11, 2001, we'd like our system to conclude that Johnny is the argument of two outliers. Indeed, the reader can verify that the following facts are true:

1. $(D, W_{\neg Works(Johnny)}) \models Works(Johnny),$
2. $(D, W_{\neg InterestTakeOff(Johnny)}) \models InterestTakeOff(Johnny),$

$$3. (D, W_{\neg Works(Johnny), Adult(Johnny)}) \not\models Works(Johnny), \text{ and}$$

$$4. (D, W_{\neg InterestTakeOff(Johnny), FlyingS(Johnny)}) \not\models InterestTakeOff(Johnny)$$

Hence, both $\neg Works(Johnny)$ and $\neg InterestTakeOff(Johnny)$ are outlier witnesses, while $Adult(Johnny)$ and $FlyingS(Johnny)$ are outliers. Note that $Income(Johnny)$ is also an outlier, with the witness $\neg Works(Johnny)$.

2.3 Defining outlier detection problems

In order to state the computational complexity of detecting outliers, in the rest of the work we refer to the following problems (also referred to as *queries*) defined for an input default theory $\Delta = (D, W)$:

$Q0$: Given Δ , does there exist an outlier in Δ ?

$Q1$: Given Δ and a literal $l \in W$, is there any outlier witness for l in Δ ?

$Q2$: Given Δ and a set of literals $S \subseteq W$, is S a witness for any outlier l in Δ ?

$Q3$: Given Δ , a set of literals $S \subseteq W$, and a literal $l \in W$, is S a witness for l in Δ ?

3 General complexity results

In this section we analyze the complexity associated with detecting outliers. First, we give some preliminary definitions involving notation.

Let L be a set of literals such that $\ell \in L$ implies that $\neg \ell \notin L$. Then we denote by $\mathcal{T}_L$ the truth assignment on the set of letters occurring in L such that, for each positive literal $p \in L$, $\mathcal{T}_L(p) = \mathbf{true}$, and for each negative literal $\neg p \in L$, $\mathcal{T}_L(p) = \mathbf{false}$.

Let T be a truth assignment on the set $x_1, \dots, x_n$ of letters. Then we denote by $Lit(T)$ the set of literals $\{\ell_1, \dots, \ell_n\}$, such that ℓ_i is x_i if $T(x_i) = \mathbf{true}$ and is $\neg x_i$ if $T(x_i) = \mathbf{false}$, for $i = 1, \dots, n$.

Theorem 3.1 $Q0$ is Σ_3^P -complete.

Proof: (Membership) Given a theory $\Delta = (D, W)$, we must show that there exists a literal l in W and a subset $S = \{s_1, \dots, s_n\}$ of W such that $(D, W_S) \models \neg s_1 \wedge \dots \wedge \neg s_n$ (query q') and $(D, W_{S,l}) \not\models \neg s_1 \wedge \dots \wedge \neg s_n$ (query q''). Query q' is Π_2^P -complete, while query q'' is Σ_2^P -complete [Gottlob, 1992; Stillman, 1992]. Thus, we can build a polynomial-time nondeterministic Turing machine with a Σ_2^P oracle, solving query $Q0$ as follows: the machine guesses both the literal l and the set S and then solves queries q' and q'' using two calls to the oracle.

(Hardness) Let $\Phi = \exists X \forall Y \exists Z f(X, Y, Z)$ be a quantified boolean formula, where $X = x_1, \dots, x_n$, $Y = y_1, \dots, y_m$, and Z are disjoint set of variables. We associate with Φ the default theory $\Delta(\Phi) = (D(\Phi), W(\Phi))$, where $W(\Phi)$ is the set $\{l, s_1, \bar{s}_1, \dots, s_n, \bar{s}_n\}$ consisting of new letters distinct from those occurring in Φ , and $D(\Phi) = D_1 \cup D_2 \cup D_3 \cup$

$D_4 \cup D_5$:

$$\begin{aligned} D_1 &= \left\{ \delta_{1,i} = \frac{\neg s_i \wedge x_i \wedge e_i}{x_i \wedge e_i}, \right. \\ &\quad \left. \bar{\delta}_{1,i} = \frac{\neg \bar{s}_i \wedge \neg x_i \wedge \neg e_i}{\neg x_i \wedge \neg e_i} \mid i = 1, \dots, n \right\} \\ D_2 &= \left\{ \delta_{2,i} = \frac{\neg s_i \wedge \neg \bar{s}_i \wedge \neg \alpha \wedge \beta}{\beta} \mid i = 1, \dots, n \right\} \cup \\ &\quad \cup \left\{ \delta_2 = \frac{\beta \wedge \alpha}{\alpha} \right\} \\ D_3 &= \left\{ \delta_{3,j} = \frac{y_j}{y_j}, \bar{\delta}_{3,j} = \frac{\neg y_j}{\neg y_j} \mid j = 1, \dots, m \right\} \\ D_4 &= \left\{ \delta_4 = \frac{l \wedge e_1 \wedge \dots \wedge e_n : f(X, Y, Z) \wedge g}{g} \right\} \\ D_5 &= \left\{ \delta_{5,i} = \frac{q : \neg s_i}{\neg s_i}, \bar{\delta}_{5,i} = \frac{q : \neg \bar{s}_i}{\neg \bar{s}_i} \mid i = 1, \dots, n \right\} \end{aligned}$$

where also $\alpha, \beta, g, e_1, \dots, e_n$ are new variables distinct from those occurring in Φ . Clearly, $W(\Phi)$ is consistent and $\Delta(\Phi)$ can be built in polynomial time. We next show that Φ is valid iff there exists an outlier in $\Delta(\Phi)$.

In the rest of the proof we denote by $\sigma(s_i)$ ($\hat{\sigma}(x_i)$ resp.) the literal x_i (s_i resp.) and by $\sigma(\bar{s}_i)$ ($\hat{\sigma}(\neg x_i)$ resp.) the literal $\neg x_i$ ($\bar{s}_i$ resp.), for $i = 1, \dots, n$. Letting S be a subset of $\{s_1, \bar{s}_1, \dots, s_n, \bar{s}_n\}$ ($\{x_1, \neg x_1, \dots, x_n, \neg x_n\}$ resp.), we denote by $\sigma(S)$ ($\hat{\sigma}(S)$ resp.) the set $\{\sigma(s) \mid s \in S\}$ ($\{\hat{\sigma}(s) \mid s \in S\}$ resp.).

($\Rightarrow$) Suppose that Φ is valid. Then we can show that l is an outlier in $\Delta(\Phi)$. As Φ is valid, then there exists a truth assignment T_X on the set X of variables such that T_X satisfies $\forall Y \exists Z f(X, Y, Z)$. Let $S = \hat{\sigma}(Lit(T_X))$. It can be shown that we can associate to each truth assignment T_Y on the set Y of variables, one and only one extension E_Y of $(D(\Phi), W(\Phi)_S)$. In particular, $E_Y \supseteq Lit(T_X) \cup Lit(T_Y)$. As Φ is valid, then $\neg f(X, Y, Z) \notin E_Y$ and $E_Y \models \neg S$. Furthermore, since there is no other extension of $(D(\Phi), W(\Phi)_S)$, then $(D(\Phi), W(\Phi)_S) \models \neg S$.

Consider now the theory $(D(\Phi), W(\Phi)_{S,l})$. We note that the literal l appears in the precondition of rule δ_4 , whose conclusion g represents, in turn, the precondition of the rules in the set D_5 , rules that allow to conclude $\neg S$, and that l does not appear in the conclusion of any rule of $D(\Phi)$. Thus $(D(\Phi), W(\Phi)_{S,l}) \not\models \neg S$. Hence l is an outlier in $\Delta(\Phi)$.

($\Leftarrow$) Suppose that there exists an outlier in $\Delta(\Phi)$. It can be shown that the outlier is l . Hence, there exists a nonempty set of literals $S \subseteq W(\Phi) - \{l\}$ such that S is an outlier witness for l in $\Delta(\Phi)$. It can then be shown that $S = \{s'_1, \dots, s'_n\}$, where s'_i is either s_i or $\bar{s}_i$, for $i = 1, \dots, n$. Now we show that $\mathcal{T}_{\sigma(S)}$ satisfies $\forall Y \exists Z f(X, Y, Z)$, i.e. that Φ is valid. For each set $L = \{\ell_1, \dots, \ell_m\}$, where ℓ_j is either y_j or $\neg y_j$, for $j = 1, \dots, m$, there exists one extension E_L of $(D(\Phi), W(\Phi) - S)$ such that $E_L \supseteq L$. We note also that $E_L \supseteq \sigma(S)$. Thus, in order for l to be an outlier in $\Delta(\Phi)$, it must be the case that for each set L , $\neg f(X, Y, Z) \notin E_L$ i.e., that $\mathcal{T}_{\sigma(S) \cup L}$ satisfies $f(X, Y, Z)$. Hence, we can conclude that Φ is valid. $\square$

Theorem 3.2 $Q1$ is Σ_3^P -complete.

Proof: The proof is analogous to that used in Theorem 3.1. $\square$

Theorem 3.3 $Q2$ is D_2^P -complete.

Proof: (Membership) Given a theory $\Delta = (D, W)$ and a subset $S = \{s_1, \dots, s_n\} \subseteq W$, we should verify that $(D, W_S) \models \neg s_1 \wedge \dots \wedge \neg s_n$ (statement q') and there exists a literal $l \in W$ such that $(D, W_{S,l}) \not\models \neg s_1 \wedge \dots \wedge \neg s_n$ (statement q''). Solving q' is in Π_2^P . As for statement q'' , it can be decided by a polynomial time nondeterministic Turing machine, with an oracle in NP, that (a) guesses both the literal $l \in W$ and the set $D_E \subseteq D$ of generating defaults of an extension E of $(D, W_{S,l})$ together with an order of these defaults; (b) checks the necessary and sufficient conditions that D_E must satisfy to be a set of generating defaults for E (see [Zhang and Marek, 1990] for a detailed description of these conditions), by multiple calls to the oracle; and (c) verifies that $\neg s_1 \wedge \dots \wedge \neg s_n \notin E$ by other calls to the oracle. It can be shown that the total number of calls to the oracle is polynomially bounded. Thus, $Q2$ is the conjunction of two independent problems, one in Π_2^P (q') and the other in Σ_2^P (q''), i.e. it is in D_2^P .

(Hardness) Let $\Delta_1 = (D_1, W_1)$ and $\Delta_2 = (D_2, W_2)$ be two propositional default theories such that W_1 and W_2 are consistent, let s_1, s_2 be two letters, and let q be the statement $(\Delta_1 \models s_1 \wedge \Delta_2 \models s_2)$. W.l.o.g. we can assume that Δ_1 and Δ_2 contain different letters, the letter s_1 occurs in D_1 but not in W_1 (and, from the previous condition, not in Δ_2), and the letter s_2 occurs in D_2 but not in W_2 (and hence not in Δ_1). We associate with q the default theory $\Delta(q) = (D(q), W(q))$ defined as follows. Let $D_1 = \{\frac{\alpha_i : \beta_i}{\gamma_i} \mid i = 1, \dots, n\}$ and let $L_1 = \{\ell_1, \dots, \ell_m\} \subseteq W_1$ be all the literals belonging to W_1 , then $D(q) = \{\frac{s_2 \wedge \alpha_i : \beta_i}{\gamma_i} \mid i = 1, \dots, n\} \cup \{\delta_j = \frac{\neg \ell_j \wedge \neg \mu \wedge \nu}{\nu} \mid j = 1, \dots, m\} \cup \{\delta_0 = \frac{\nu : \mu}{\mu}\} \cup D_2$, and $W(q) = W_1 \cup W_2 \cup \{\neg s_1, s_2\}$, where ν and μ are new letters distinct from those occurring in Δ_1 and Δ_2 , and from s_1 and s_2 . It can be shown that q is true iff $\{\neg s_1\}$ is a witness for some outlier in $\Delta(q)$. We note that q is the conjunction of a Π_2^P -hard and a Σ_2^P -hard independent problems, thus this proves the hardness part. $\square$

Theorem 3.4 $Q3$ is D_2^P -complete.

4 Disjunction-free theories

Disjunction-free theories form a significant subset of propositional default theories because they are equivalent to extended logic programs under stable model semantics [Gelfond and Lifschitz, 1991]. A finite propositional theory $\Delta = (D, W)$ is *disjunction-free* (DF in short), if W is a set of literals, and the precondition, justification and consequence of each default in D is a conjunction of literals. As we see below, outlier detection for DF theories is still quite complex.

Theorem 4.1 $Q0$ restricted to disjunction-free theories is Σ_2^P -complete.

Proof: (Membership) The membership proof is analogous to that of Theorem 3.1. We note that when disjunction-free theories are considered, q' and q'' are co-NP-complete and NP-complete, respectively.

(Hardness) Let $\Phi = \exists X \forall Y f(X, Y)$ be a quantified boolean formula, where $X = x_1, \dots, x_n$ and $Y =$

$y_1, \dots, y_m$ are disjoint set of variables, and $f(X, Y) = C_1 \wedge \dots \wedge C_r$, with $C_k = t_{k,1} \vee \dots \vee t_{k,u_k}$, and each $t_{k,1}, \dots, t_{k,u_k}$ is a literal, for $k = 1, \dots, r$. We associate to Φ the default theory $\Delta(\Phi) = (D(\Phi), W(\Phi))$, with $W(\Phi)$ the set $\{l, s_1, \bar{s}_1, \dots, s_n, \bar{s}_n\}$ of new letters distinct from those occurring in Φ , and $D(\Phi) = D_1 \cup D_2 \cup D_3 \cup D_5 \cup D'_1 \cup D'_2$, where D_1, D_2, D_3 and D_5 are the sets of defaults as in Theorem 3.1 and D'_1 and D'_2 are the following sets of defaults:

$$D'_1 = \left\{ \frac{t_{k,h}:c_k}{c_k} \mid k = 1, \dots, r; h = 1, \dots, u_k \right\}$$

$$D'_2 = \left\{ \frac{l \wedge e_1 \wedge \dots \wedge e_n \wedge c_1 \wedge \dots \wedge c_r: g}{g} \right\}$$

where also $\alpha, \beta, g, e_1, \dots, e_n, c_1, \dots, c_r$ are new variables distinct from those occurring in Φ . Clearly, $W(\Phi)$ is consistent and $\Delta(\Phi)$ can be built in polynomial time. The rest of the proof is similar to that of Theorem 3.1. $\square$

Theorem 4.2 *Q1 restricted to disjunction-free theories is Σ_2^P -complete.*

Theorem 4.3 *Q2 and Q3 restricted to disjunction-free theories are D^P -complete.*

5 Tractable Cases

In this section, we look for some classes of default theories for which outlier detection is computationally tractable.

Definition 5.1 A default theory is *normal mixed unary* (NMU in short) iff W is a set of literals and D is a set of defaults of the form $\frac{y:x}{x}$ where y is either missing or a literal and x is a literal.

Definition 5.2 An NMU default theory is *normal unary* (NU in short) iff the prerequisite of each default is either missing or positive. An NMU default theory is *dual normal unary* (DNU in short) iff the prerequisite of each default is either missing or negative.

Thus, NMU, NU, and DNU theories have a quite simple structure. In spite of that, the complexity of detecting outliers from these theories remain often quite high, as demonstrated by the following results (proofs are omitted for the sake of brevity).

Theorem 5.3 *The following hold over NMU default theories:*

- Q0 and Q1 are Σ_2^P -complete.
- Q2 and Q3 are D^P -complete.

Theorem 5.4 *The following hold over NU and DNU default theories:*

- Q0 and Q1 are NP-complete.
- Q2 and Q3 are in P.

Thus, restricting our attention to NMU, NU, or DNU theories does not suffice to attain tractability of the most general queries Q0 and Q1. Some further restriction is needed, which is considered next.

Theorem 5.5 ([Kautz and Selman, 1991] [Zohary, 2002]) *Suppose Δ is a normal (dual normal) unary default theory. We can decide whether a literal belongs to every extension of Δ in time $O(n^2)$, where n is the length of the theory.*

Definition 5.6 The *atomic dependency graph* of an NMU default theory Δ is a directed graph whose nodes are all atoms in the language of Δ , and such that there is an arc directed from p to q iff there is a default in Δ in which p or $\neg p$ is a prerequisite and q or $\neg q$ is a consequence.

Definition 5.7 A normal (dual normal) unary default theory is *acyclic* iff its atomic dependency graph is acyclic.

Theorem 5.8 *Queries Q0, Q1, Q2 and Q3, restricted to the class of acyclic NU or acyclic DNU default theories can be solved in polynomial time in the size of the input theory.*

Proof: It can be shown that for any acyclic NMU default theory $\Delta = (D, W)$ such that W is consistent and for any literal l in W , any minimal outlier witness set for l in Δ is at most 1 in size. Theorem's statement then follows from Theorem 5.5. $\square$

6 Related Work

The research on logical-based abduction [Poole, 1989; Console *et al.*, 1991; Eiter and Gottlob, 1995] is related to outlier detection. In the framework of logic-based abduction, the domain knowledge is described using a logical theory T . A subset X of hypotheses is an abduction explanation to a set of manifestations M if $T \cup X$ is a consistent theory that entails M . Abduction resembles outlier detection in that it deals with exceptional situations.

The work most relevant to our study is perhaps the paper by Eiter, Gottlob, and Leone on abduction from default theories [Eiter *et al.*, 1997]. There, the authors have presented a basic model of abduction from default logic and analyzed the complexity of the main abductive reasoning tasks. They presented two modes of abductions: one based on brave reasoning and the other on cautious reasoning. According to these authors, a default abduction problem (DAP) is a tuple $\langle H, M, W, D \rangle$ where H is a set of ground literals called *hypotheses*, M is a set of ground literals called *observations*, and (D, W) is a default theory. Their goal, in general, was to explain some observations from M by using various hypotheses in the context of the default theory (D, W) . They suggest the following definition for an explanation:

Definition 6.1 ([Eiter *et al.*, 1997]) *Let $P = \langle H, M, D, W \rangle$ be a DAP and let $E \subseteq H$. Then, E is a skeptical explanation for P iff*

1. $(D, W \cup E) \models M$, and
2. $(D, W \cup E)$ has a consistent extension.

There is a close relationship between outliers and skeptical explanations, as the following theorem states. The theorem also holds for ordered semi-normal default theories [Etherington, 1987].

Theorem 6.2 *Let $\Delta = (D, W)$ be a normal default theory, where W is consistent. Let $l \in W$ and $S \subseteq W$. S is an outlier witness set for l iff $\{l\}$ is a minimal skeptical explanation for $\neg S$ in the DAP $P = \langle \{l\}, \neg S, D, W_{S,l} \rangle$*

Hence, we can say that S is an outlier witness for l if $l \in W$, l is a skeptical explanation for S , but still $\neg S$ holds in every extension of the theory.

Theory \ Query	Q0	Q1	Q2	Q3
Propositional	Σ_3^P -c	Σ_3^P -c	D_2^P -c	D_2^P -c
DF, NMU	Σ_2^P -c	Σ_2^P -c	D^P -c	D^P -c
NU, DNU	NP-c	NP-c	P	P
Acy. NU, Acy. DNU	P	P	P	P

Table 1: Complexity results for outlier detection

Despite the close relationship between outlier detection and abduction demonstrated by the above theorem (especially for normal defaults) we believe that there is a significant difference between the two concepts. In abduction, we have to single out a set of manifestations and a set of potential explanations. Outlier detection, on the other hand, has much more to do with knowledge discovery. The task in outlier detection is to *learn* who the exceptionals (the outliers), or the suspects, if you wish, are, and to justify the suspicion (that is, list the outlier witnesses).

It also turns out that reducing outlier detection queries to abduction and vice versa is not straightforward, and therefore, when analyzing the computational complexities involved in answering outlier detection queries we have preferred to use the classical Boolean formula satisfiability problems.

7 Conclusion

Suppose you are walking down the street and you see a blind person walking in the opposite direction. You believe he is blind because he is feeling his way with a walking stick. Suddenly, something falls out of his bag, and to your surprise, he finds it immediately without probing about with his fingers, as you would expect for a blind person. This kind of behavior renders the “blind” person suspicious.

The purpose of this paper has been to formally mimic this type of reasoning using default logic. We have formally defined the notion of an outlier and an outlier witness, and analyzed the complexities involved, pointing out some non-trivial tractable subsets. The complexity results are summarized in Table 1, where $\mathcal{C}$ -c stands for $\mathcal{C}$ -complete. As explained in the introduction, outlier detection can also be used for maintaining database integrity and completeness.

This work can be extended in several ways. First, we can develop the concept of outliers in other frameworks of default databases, like System Z [Pearl, 1990] and Circumscription [McCarthy, 1980]. Second, we can look for intelligent heuristics that will enable us to perform the involved heavy computational task more efficiently. Third, we can study the problem from the perspective of default theories as a “semantic check toolkit” for relational databases.

Acknowledgements. The authors gratefully thank Michael Gelfond for some fruitful discussions on the subject of the paper.

References

[Angiulli *et al.*, 2003] Fabrizio Angiulli, Rachel Ben-Eliyahu-Zohary, and Luigi Palopoli. Outlier detection using default logic, 2003. Full manuscript.

- [Console *et al.*, 1991] Luca Console, Daniele Theseider Dupré, and Pietro Torasso. On the relationship between abduction and deduction. *Journal of Logic and Computation*, 1(5):661–690, 1991.
- [Eiter and Gottlob, 1995] Thomas Eiter and Georg Gottlob. The complexity of logic-based abduction. *JACM*, 42(1):3–42, 1995.
- [Eiter *et al.*, 1997] Thomas Eiter, Georg Gottlob, and Nicola Leone. Semantics and complexity of abduction from default theories. *Artificial Intelligence*, 90(1-2):177–223, 1997.
- [Etherington, 1987] David W. Etherington. Formalizing nonmonotonic reasoning systems. *Artificial Intelligence*, 31:41–85, 1987.
- [Garey and Johnson, 1979] M. R. Garey and D. S. Johnson. *Computers and intractability; a guide to the theory of NP-completeness*. W.H. Freeman, 1979.
- [Gelfond and Lifschitz, 1991] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.
- [Gottlob, 1992] George Gottlob. Complexity results for non-monotonic logics. *J. Logic Computat.*, 2(3):397–425, 1992.
- [Kautz and Selman, 1991] Henry A. Kautz and Bart Selman. Hard problems for simple default logics. *Artificial Intelligence*, 49:243–279, 1991.
- [McCarthy, 1980] John McCarthy. Circumscription - a form of non-monotonic reasoning. *Artificial Intelligence*, 13:27–39, 1980.
- [Pearl, 1990] Judea Pearl. System z: a natural ordering of defaults with tractable applications to nonmonotonic reasoning. In *TARK-90*, pages 121–135, Monterey, CA, 1990.
- [Poole, 1989] David Poole. Normality and faults in logic-based diagnosis. In *IJCAI-89*, pages 1304–1310, 1989.
- [Reiter, 1980] Raymond Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81–132, 1980.
- [Stillman, 1992] Jonathan Stillman. The complexity of propositional default logics. In *Proceedings of the 10th National Conference on Artificial Intelligence*, pages 794–800, San Jose, CA, July 1992.
- [Zhang and Marek, 1990] A. Zhang and W. Marek. On the classification and existence of structures in default logic. *Fundamenta Informaticae*, 13(4):485–499, 1990.
- [Zohary, 2002] Rachel Ben-Eliyahu Zohary. Yet some more complexity results for default logic. *Artificial Intelligence*, 139(1):1–20, 2002.

Ordering Default Theories

Chiaki Sakama

Department of Computer and Communication Sciences
Wakayama University
Sakaedani, Wakayama 640 8510, Japan
sakama@sys.wakayama-u.ac.jp

Abstract

In first-order logic, a theory T_1 is considered stronger than another theory T_2 if every formula derived from T_2 is also derived from T_1 . Such an order relation is useful to know relative value between different theories. In the context of default logic, a theory contains default information as well as definite information. To order default theories, it is necessary to assess the information content of a default theory. To this end, we introduce a multi-valued interpretation of default theories based on a nine-valued *bilattice*. It distinguishes definite and credulous/skeptical default information derived from a theory, and is used for ordering default theories based on their information contents. The technique is also applied to order nonmonotonic logic programs. The results of this paper provide a method for comparing different default theories and have important application to learning nonmonotonic theories.

1 Introduction

In knowledge representation based on logic, the relative value of a given theory is formally assessed by comparing the amount of information between theories. In first-order logic, a theory T_1 is considered *stronger* than another theory T_2 if every formula derived from T_2 is also derived from T_1 but not vice-versa (i.e., $T_1 \models T_2$ and $T_2 \not\models T_1$). For instance, the theory

$$T_1 = \{bird, bird \rightarrow flies\}$$

is stronger than the theory

$$T_2 = \{bird \vee flies\}.$$

In the context of *default logic* [Reiter, 1980], however, the problem is not so simple. For instance, consider the default theory:

$$T_3 = \{bird, \frac{bird : flies}{flies}\}.$$

In this case, both T_1 and T_3 imply *flies*, but the fact *flies* from T_1 is a conclusion from definite information, while the same fact from T_3 is a conclusion from default information. Introducing another fact $\neg flies$ to each theory, the

conclusion *flies* is still derived from the inconsistent theory $T_1 \cup \{\neg flies\}$, while it is withdrawn from $T_3 \cup \{\neg flies\}$. Thus, two theories T_1 and T_3 have the same extension, but conclusions derived from T_1 are stronger than those of T_3 .

To compare and order default theories, it is necessary to distinguish different sorts of information derived from a theory. Such consideration is meaningful and important with the following reasons.

- Studies in nonmonotonic logics have been centered on answering the question: “What information is concluded from a theory (with common-sense)?” On the other hand, few studies answer the question: “What sort of information is concluded from a theory?” Since default theories contain definite and default information, distinguishing different sorts of information is meaningful to assess the information content of a theory. Default theories contain *incomplete* information, so that the assessment provides a theoretical ground to measure the degree of “incompleteness” of a theory. These arguments are also effective in the field of *nonmonotonic logic programming* [Baral and Gelfond, 1994].
- It is important to know relative value between theories. A theory is considered more valuable than another theory if the former contains more information than the latter. Comparison of theories is especially important when there exist multiple sources of information as in *multi-agent systems*. In first-order logic, theories are ordered by logical entailment. In default logic, however, extensions of theories are not necessarily helpful for judging relative strength between theories (as presented above). To order default theories, it is necessary to provide a better ability of comparing default theories beyond their extensions. It should distinguish different sorts of information in a default theory, and order theories according to their information contents.
- In first-order logic, a theory is called *more general* than another theory if the former is stronger than the latter. Generality relations over first-order clauses have been extensively studied in the fields of *machine learning* and *inductive logic programming* [Nienhuys-Cheng and de Wolf, 1997]. In these fields, generalization is used as a basic operation for inductive learning, but it is unknown how to extend the notion to nonmonotonic the-

ories. To construct induction systems that learn non-monotonic theories, it is necessary to extend the generalization operation and to build a theory for ordering nonmonotonic theories. Ordering default theories thus has potential application to the theory of induction in nonmonotonic logics and nonmonotonic inductive logic programming.

With these background and motivation, this paper studies a method for ordering default theories. To this end, we first provide a multi-valued interpretation for default theories based on a nine-valued *bilattice*. It can distinguish different sorts of information derived from default theories. We then introduce ordering over default theories, which orders different default theories based on the multi-valued interpretations of formulas. The techniques are also applied to order nonmonotonic logic programs under the *answer set semantics*. The rest of this paper is organized as follows. Section 2 reviews the framework of default logic. Section 3 develops a theory of ordering default theories. Section 4 applies the technique to nonmonotonic logic programming. Section 5 discusses related issues and Section 6 summarizes the paper.

2 Default Logic

A *default theory* is defined as a pair $\Delta = (D, W)$ where D is a set of default rules and W is a set of first-order formulas (called *facts*). A default rule (or simply *default*) is of the form:

$$\frac{\alpha : \beta_1, \dots, \beta_n}{\gamma}$$

where $\alpha, \beta_1, \dots, \beta_n$ and γ are first-order formulas and respectively called the *prerequisite*, the *justifications* and the *consequent*. In this paper, any default is assumed to have at least one justification ($n \geq 1$). A default theory is called *super-normal* if every default is of the form $\frac{\alpha}{\gamma}$. As defaults and facts are syntactically distinguishable, we often write a default theory Δ as a set $W \cup D$ as far as no confusion arises. Any variable appearing in D and W is free and any default/fact with variables represents the set of its ground instances over the Herbrand universe of Δ . Throughout this paper we assume a default theory which is already ground-instantiated, i.e., for any default theory (D, W) , D and W contain no variable. Also, a formula means a propositional formula unless stated otherwise.

A set E of formulas is an *extension* of (D, W) if it coincides with the smallest deductively closed set E' of formulas satisfying the conditions: (i) $W \subseteq E'$, and (ii) for any ground default $\alpha : \beta_1, \dots, \beta_n / \gamma$ from D , $\alpha \in E'$ and $\neg \beta_i \notin E$ ($i = 1, \dots, n$) imply $\gamma \in E'$. A default theory may have none, one or multiple extensions in general. The set of all extensions of Δ is written as $\mathcal{EXT}(\Delta)$. Given a default theory Δ , a formula is a *credulous* conclusion of Δ if it belongs to some (but not all) extensions. By contrast, a formula is a *skeptical* conclusion of Δ if it belongs to all extensions. An extension E is *inconsistent* if it is the set of all formulas in the language.

Proposition 2.1 [Reiter, 1980]¹ A default theory $\Delta = (D, W)$ has the inconsistent extension iff W is inconsistent.

¹This property holds for defaults with non-empty justifications.

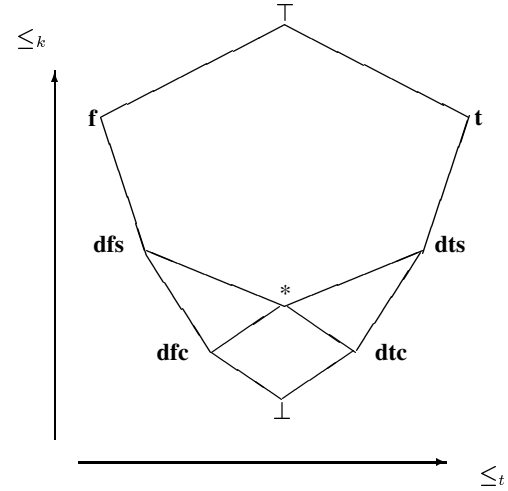


Figure 1: A lattice for logic IX

3 Ordering Default Theories

In classical logic, a formula F is interpreted true/false if $F/\neg F$ is a logical consequence of a theory; otherwise it is undefined. In default logic, on the other hand, a formula is either a definite consequence by W or a default consequence by D . Moreover, default consequences are brought by two different modes of inferences - skeptical or credulous reasoning. To characterize these different types of inferences, we first introduce a multi-valued logic for default reasoning.

Definition 3.1 The logic IX has the nine truth values $t, f, \top, \perp, dts, dfs, dtc, dfc, *$, which respectively mean *true*, *false*, *contradictory*, *undefined*, *skeptically true by default*, *skeptically false by default*, *credulously true by default*, *credulously false by default*, and *contradictory by default*.

The truth values of IX constitute a *bilattice* under the *knowledge ordering* $\leq_k$ and the *truth ordering* $\leq_t$ (Figure 1). Here, it holds that $\perp \leq_k dtc \leq_k * \leq_k dts \leq_k t \leq_k \top$; $\perp \leq_k dfc \leq_k * \leq_k dfs \leq_k f \leq_k \top$; $f \leq_t \top \leq_t t$; and $f \leq_t dfs \leq_t dfc \leq_t \{*, \perp\} \leq_t dtc \leq_t dts \leq_t t$.

In logic IX , negation $\neg$ is defined as: $\neg \top = \perp$, $\neg t = f$, $\neg f = t$, $\neg dts = dfs$, $\neg dfs = dts$, $\neg dtc = dfc$, $\neg dfc = dtc$, $\neg * = *$, and $\neg \perp = \perp$. On the other hand, the disjunction $\vee$ and the conjunction $\wedge$ are respectively defined by the join operation and the meet operation with respect to the truth ordering in the bilattice. That is, $t \vee x = t$ for $x \in IX$, $* \vee \perp = dtc$, $\top \wedge dfc = f$, and so on. It is easily verified that $\vee$ and $\wedge$ are associative, commutative, idempotent and absorptive.² Note that the truth-functional operations $\vee$ and $\wedge$ have their meaning supplied by the $\leq_t$ ordering, while we later order default theories by the $\leq_k$ ordering.

Under IX the interpretation of a formula in a default theory is defined as follows.

Definition 3.2 Given a default theory $\Delta = (D, W)$, the mapping ϕ_Δ associates a propositional formula F with a truth

²They are not distributive, e.g., $* \vee (\top \wedge \perp) \neq (* \vee \top) \wedge (* \vee \perp)$.

value of IX as follows:

$$\phi_{\Delta}(F) = \begin{cases} \mathbf{t} & \text{if } \mathcal{EXT}(\Delta) \neq \emptyset \text{ and } W \models F; \\ \mathbf{f} & \text{if } \mathcal{EXT}(\Delta) \neq \emptyset \text{ and } W \models \neg F; \\ \mathbf{dts} & \text{if } \mathcal{EXT}(\Delta) \neq \emptyset, W \not\models F \\ & \text{and } \forall E \in \mathcal{EXT}(\Delta) F \in E; \\ \mathbf{dfs} & \text{if } \mathcal{EXT}(\Delta) \neq \emptyset, W \not\models \neg F \\ & \text{and } \forall E \in \mathcal{EXT}(\Delta) \neg F \in E; \\ \mathbf{dte} & \text{if } \exists E \in \mathcal{EXT}(\Delta) \text{ s.t. } F \in E \\ & \text{and } \exists E' \in \mathcal{EXT}(\Delta) \text{ s.t. } F \notin E'; \\ \mathbf{dfe} & \text{if } \exists E \in \mathcal{EXT}(\Delta) \text{ s.t. } \neg F \in E \\ & \text{and } \exists E' \in \mathcal{EXT}(\Delta) \text{ s.t. } \neg F \notin E'; \\ \perp & \text{otherwise.} \end{cases}$$

In particular, we write

$$\begin{aligned} \phi_{\Delta}(F) &= \top \text{ if } \phi_{\Delta}(F) = \mathbf{t} \text{ and } \phi_{\Delta}(F) = \mathbf{f}; \\ \phi_{\Delta}(F) &= * \text{ if } \phi_{\Delta}(F) = \mathbf{dte} \text{ and } \phi_{\Delta}(F) = \mathbf{dfe}. \end{aligned}$$

Remark that formulas F and $\neg F$ are included in every extension of Δ iff W is inconsistent (Proposition 2.1). So it does not happen that both $\phi_{\Delta}(F) = \mathbf{dts}$ and $\phi_{\Delta}(F) = \mathbf{dfs}$ for any F .

The mapping ϕ_{Δ} provides multi-valued interpretations of formulas in a default theory. Intuitively, $\phi_{\Delta}(F) \in \{\mathbf{t}, \mathbf{f}\}$ means that F or $\neg F$ is a definite conclusion from W . When $\phi_{\Delta}(F) = \top$ for some formula F , $\phi_{\Delta}(G) = \top$ for any formula G . This is because in this case W is inconsistent and entails every formula.³ On the other hand, when $\phi_{\Delta}(F) \in \{\mathbf{dts}, \mathbf{dfs}\}$ (resp. $\phi_{\Delta}(F) \in \{\mathbf{dte}, \mathbf{dfe}\}$), F or $\neg F$ is a default conclusion inferred skeptically (resp. credulously) from Δ . When $\phi_{\Delta}(F) = *$, a formula F belongs to some extension and its negation $\neg F$ belongs to another extension. In this case, the truth value of F is contradictory by default. ϕ_{Δ} maps a formula into $\perp$ if it is included in no extension. In particular, every formula has the value $\perp$ if Δ has no extension.

Example 3.1 Let Δ be the theory:

$$\text{bird}, \frac{\text{bird} : \text{flies}}{\text{flies}},$$

which has the single extension $Th(\{\text{bird}, \text{flies}\})$. Then $\phi_{\Delta}(\text{bird}) = \mathbf{t}$, $\phi_{\Delta}(\text{flies}) = \mathbf{dts}$, and $\phi_{\Delta}(\text{bird} \rightarrow \text{flies}) = \mathbf{dts}$, etc.

Example 3.2 Let Δ be the theory:

$$\frac{\neg \text{rh-broken} \wedge \text{lh-broken}}{\text{lh-broken}}, \quad \frac{\neg \text{lh-broken} \wedge \text{rh-broken}}{\text{rh-broken}},$$

which has two extensions $Th(\{\text{lh-broken}\})$ and $Th(\{\text{rh-broken}\})$. Then $\phi_{\Delta}(\text{lh-broken}) = \phi_{\Delta}(\text{rh-broken}) = \mathbf{dte}$, $\phi_{\Delta}(\text{lh-broken} \vee \text{rh-broken}) = \mathbf{dts}$ and $\phi_{\Delta}(\text{lh-broken} \wedge \text{rh-broken}) = \perp$.

Example 3.3 Let Δ be the theory:

$$\frac{\text{quaker} \wedge \text{republican},}{\text{quaker} : \text{pacifist}}, \quad \frac{\text{republican} : \neg \text{pacifist}}{\neg \text{pacifist}},$$

which has two extensions $Th(\{\text{quaker} \wedge \text{republican}, \text{pacifist}\})$ and $Th(\{\text{quaker} \wedge \text{republican}, \neg \text{pacifist}\})$. Then $\phi_{\Delta}(\text{quaker} \wedge \text{republican}) = \mathbf{t}$ and $\phi_{\Delta}(\text{pacifist}) = *$.

³In this sense, our logic is not “paraconsistent”.

The followings are some properties of ϕ_{Δ} .

Proposition 3.1 Given two formulas F and G , $F \equiv G$ implies $\phi_{\Delta}(F) = \phi_{\Delta}(G)$.

Proposition 3.2 For formulas F and G ,

- $\phi_{\Delta}(\neg F) = \neg \phi_{\Delta}(F)$.
- $\phi_{\Delta}(F) \geq_k \phi_{\Delta}(G)$ iff $\phi_{\Delta}(\neg F) \geq_k \phi_{\Delta}(\neg G)$.
- $\phi_{\Delta}(F) \geq_t \phi_{\Delta}(G)$ iff $\phi_{\Delta}(\neg G) \geq_t \phi_{\Delta}(\neg F)$.
- $\neg(\phi_{\Delta}(F) \vee \phi_{\Delta}(G)) = \neg \phi_{\Delta}(F) \wedge \neg \phi_{\Delta}(G)$.
- $\neg(\phi_{\Delta}(F) \wedge \phi_{\Delta}(G)) = \neg \phi_{\Delta}(F) \vee \neg \phi_{\Delta}(G)$.

Proposition 3.3 For formulas F and G ,

- $\phi_{\Delta}(F \vee G) \geq_t \phi_{\Delta}(F) \vee \phi_{\Delta}(G)$.
- $\phi_{\Delta}(F \wedge G) \leq_t \phi_{\Delta}(F) \wedge \phi_{\Delta}(G)$.
- $\phi_{\Delta}(F \rightarrow G) \geq_t \phi_{\Delta}(\neg F) \vee \phi_{\Delta}(G)$.

Proof: By $\phi_{\Delta}(F \vee G) \geq_t \phi_{\Delta}(F)$ and $\phi_{\Delta}(F \vee G) \geq_t \phi_{\Delta}(G)$, $\phi_{\Delta}(F \vee G) \geq_t \phi_{\Delta}(F) \vee \phi_{\Delta}(G)$ holds. Also, $\phi_{\Delta}(F \wedge G) \leq_t \phi_{\Delta}(F)$ and $\phi_{\Delta}(F \wedge G) \leq_t \phi_{\Delta}(G)$ imply $\phi_{\Delta}(F \wedge G) \leq_t \phi_{\Delta}(F) \wedge \phi_{\Delta}(G)$. The result for implication follows by the relation $\phi_{\Delta}(F \rightarrow G) = \phi_{\Delta}(\neg F \vee G)$. $\square$

Example 3.4 In Example 3.2, $\phi_{\Delta}(\text{lh-broken} \vee \text{rh-broken}) \geq_t \phi_{\Delta}(\text{lh-broken}) \vee \phi_{\Delta}(\text{rh-broken})$ and $\phi_{\Delta}(\text{lh-broken} \wedge \text{rh-broken}) \leq_t \phi_{\Delta}(\text{lh-broken}) \wedge \phi_{\Delta}(\text{rh-broken})$.

Thus, the degree of truth of an internal disjunction (resp. conjunction) is generally higher (resp. lower) than that of an external one. By contrast, there are no general relations between $\phi_{\Delta}(F \vee G)$ and $\phi_{\Delta}(F) \vee \phi_{\Delta}(G)$; and $\phi_{\Delta}(F \wedge G)$ and $\phi_{\Delta}(F) \wedge \phi_{\Delta}(G)$ under the $\leq_k$ ordering.

Next we introduce ordering between default theories.

Definition 3.3 Let Δ_1 and Δ_2 be two default theories which have the same underlying language. Then, Δ_1 is *stronger* than Δ_2 (written as $\Delta_2 \leq_{DL} \Delta_1$) if $\phi_{\Delta_2}(F) \leq_k \phi_{\Delta_1}(F)$ for any formula F in the language. We write $\Delta_1 \simeq_{DL} \Delta_2$ if $\Delta_1 \leq_{DL} \Delta_2$ and $\Delta_2 \leq_{DL} \Delta_1$.

The relation $\leq_{DL}$ is a pre-order, i.e., a reflexive and transitive relation on the set of all default theories in the language. Throughout the paper, when we compare different default theories, we assume that they have the same underlying language.

Intuitively, a default theory Δ_1 is stronger than another default theory Δ_2 if Δ_1 entails more certain information than Δ_2 . In other words, when $\Delta_2 \leq_{DL} \Delta_1$, conclusions derived from Δ_1 are relatively more stable and reliable than those derived from Δ_2 . The ‘stronger’ relation reduces to the relation between (propositional) first-order theories when default theories have no defaults.

Proposition 3.4 Let $\Delta_1 = (\emptyset, W_1)$ and $\Delta_2 = (\emptyset, W_2)$ be two default theories. Then, $\Delta_2 \leq_{DL} \Delta_1$ iff $W_1 \models W_2$.

Thus, the relation $\leq_{DL}$ is a natural extension of the one for (propositional) first-order theories.

Proposition 3.5 Given two default theories Δ_1 and Δ_2 , $\Delta_1 \simeq_{DL} \Delta_2$ implies $\mathcal{EXT}(\Delta_1) = \mathcal{EXT}(\Delta_2)$, but not vice-versa.

The above proposition presents that the order-equivalence relation $\simeq_{DL}$ provides an equivalence relation which is stronger than the equivalence based on extensions.

Example 3.5 (introductory example) Let Δ_1 and Δ_2 be two default theories:

$$\begin{aligned}\Delta_1 : & \quad bird, \quad bird \rightarrow flies, \\ \Delta_2 : & \quad bird, \quad \frac{bird : flies}{flies},\end{aligned}$$

where $\phi_{\Delta_1}(bird) = \phi_{\Delta_2}(bird) = \mathbf{t}$, $\phi_{\Delta_1}(flies) = \mathbf{t}$, $\phi_{\Delta_2}(flies) = \mathbf{dts}$. Then, $\Delta_2 \leq_{DL} \Delta_1$.

On the other hand, given

$$\Delta_3 : \quad bird, \quad \frac{: bird \rightarrow flies}{bird \rightarrow flies},$$

$\Delta_2 \simeq_{DL} \Delta_3$. Note that all Δ_1 , Δ_2 and Δ_3 have the same extension $Th(\{bird, flies\})$.

The order $\leq_{DL}$ is nonmonotonic with respect to the increase of information.

Proposition 3.6 Let Δ_1 and Δ_2 be two default theories and F a formula. Then, $\Delta_1 \leq_{DL} \Delta_2$ implies neither $\Delta_1 \leq_{DL} \Delta_2 \cup \{F\}$ nor $\Delta_1 \cup \{F\} \leq_{DL} \Delta_2 \cup \{F\}$. In particular, $\Delta_1 \not\leq_{DL} \Delta_1 \cup \{F\}$ in general.

Example 3.6 Let Δ_1 and Δ_2 be two default theories:

$$\begin{aligned}\Delta_1 : & \quad \frac{: p \wedge \neg q}{p}, \\ \Delta_2 : & \quad \frac{: p \wedge \neg q}{p}, \quad r,\end{aligned}$$

where $\phi_{\Delta_1}(p) = \phi_{\Delta_2}(p) = \mathbf{dts}$, $\phi_{\Delta_1}(q) = \phi_{\Delta_2}(q) = \perp$, $\phi_{\Delta_1}(r) = \perp$ and $\phi_{\Delta_2}(r) = \mathbf{t}$. Then, $\Delta_1 \leq_{DL} \Delta_2$ holds. Let $F = (r \rightarrow q)$. Then, $\Delta_1 \not\leq_{DL} \Delta_2 \cup \{F\}$ and $\Delta_1 \cup \{F\} \not\leq_{DL} \Delta_2 \cup \{F\}$ by $\phi_{\Delta_1 \cup \{F\}}(p) = \mathbf{dts}$ and $\phi_{\Delta_2 \cup \{F\}}(p) = \perp$.

The introduction of new information may block the application of some default rules, which would cause the withdrawal of some default conclusions in a theory. This is a typical feature of default reasoning.

We finally provide a connection between the order relation $\leq_{DL}$ and default extensions.

Theorem 3.7 Let $\Delta_1 = (D_1, W_1)$ and $\Delta_2 = (D_2, W_2)$ be two default theories. Then, $\Delta_1 \leq_{DL} \Delta_2$ if the following conditions are satisfied:

1. $W_2 \models W_1$,
2. $\forall E_2 \in \mathcal{EXT}(\Delta_2) \exists E_1 \in \mathcal{EXT}(\Delta_1) \text{ s.t. } E_1 \subseteq E_2$,
3. $\forall E_1 \in \mathcal{EXT}(\Delta_1) \exists E_2 \in \mathcal{EXT}(\Delta_2) \text{ s.t. } E_1 \subseteq E_2$.

Proof: Let F be a formula. By $W_2 \models W_1$, $\mathbf{t} \leq_k \phi_{\Delta_1}(F)$ implies $\mathbf{t} \leq_k \phi_{\Delta_2}(F)$, and $\mathbf{f} \leq_k \phi_{\Delta_1}(F)$ implies $\mathbf{f} \leq_k \phi_{\Delta_2}(F)$. Suppose $\phi_{\Delta_1}(F) = \mathbf{dts}$. In this case, $\mathcal{EXT}(\Delta_1) \neq \emptyset$, $W \models F$ and F is included in every extension of Δ_1 . By the second condition, F is included in every extension of Δ_2 . Then, $\mathbf{dts} \leq_k \phi_{\Delta_2}(F)$. Similarly, it is shown that $\phi_{\Delta_1}(F) = \mathbf{dfs}$ implies $\mathbf{dfs} \leq_k \phi_{\Delta_2}(F)$. Next, suppose $\phi_{\Delta_1}(F) = \mathbf{dts}$. In this case, F is included in some (but not every) extension of Δ_1 . By the third condition, F is included

in some extension of Δ_2 . Then, $\mathbf{dts} \leq_k \phi_{\Delta_2}(F)$. Similarly, it is shown that $\phi_{\Delta_1}(F) = \mathbf{dfs}$ implies $\mathbf{dfs} \leq_k \phi_{\Delta_2}(F)$. Finally, suppose $\phi_{\Delta_1}(F) = \perp$. Since $\phi_{\Delta_2}(F) = x$ with $\perp \leq_k x$, $\perp \leq_k \phi_{\Delta_2}(F)$ holds. $\square$

Theorem 3.7 provides a sufficient condition to judge $\Delta_1 \leq_{DL} \Delta_2$ using extensions of default theories.

4 Ordering Nonmonotonic Logic Programs

In logic programming, default reasoning is realized by *negation as failure* (NAF). Logic programs containing NAF are called *nonmonotonic logic programs*.

Nonmonotonic logic programs considered in this paper are the class of *extended logic programs* (ELPs) [Gelfond and Lifschitz, 1991], which contain two kinds of negation; explicit (or classical) negation $\neg$ and NAF (or default negation) *not*. An extended logic program (or simply a program) is a set of *rules* of the form:

$$L_0 \leftarrow L_1, \dots, L_m, \text{not } L_{m+1}, \dots, \text{not } L_n \quad (n \geq m)$$

where each L_i ($0 \leq i \leq n$) is a positive/negative literal and *not* represents NAF. The literal L_0 is the *head* and the conjunction $L_1, \dots, L_m, \text{not } L_{m+1}, \dots, \text{not } L_n$ is the *body* of the rule. A rule or a program is called *not-free* if it contains no NAF (i.e., $m = n$). Given an ELP Π , the set of *not-free* rules from Π is denoted by Π^+ . A rule with the empty body $L \leftarrow$ is identified with the literal L . The head of any rule is non-empty.⁴ Like default theories, every variable in a program is interpreted as a free variable. A program Π is semantically identified with its ground instantiation, i.e., the set of ground rules obtained from Π by substituting variables with elements of the Herbrand universe of Π in every possible way. We handle ground programs throughout the paper.

The semantics of ELPs is given by the *answer set semantics* [Gelfond and Lifschitz, 1991]. Let *Lit* be the set of all ground literals in the language of a program (called the *literal base*). Suppose an ELP Π and a set of literals $S (\subseteq \text{Lit})$. Then, the *reduct* Π^S is the program which contains the ground rule $L_0 \leftarrow L_1, \dots, L_m$ iff there is a rule $L_0 \leftarrow L_1, \dots, L_m, \text{not } L_{m+1}, \dots, \text{not } L_n$ in the ground instantiation of Π such that $\{L_{m+1}, \dots, L_n\} \cap S = \emptyset$. Given a *not-free* ELP Π , $Cn(\Pi)$ denotes the smallest set of ground literals which is (i) *closed* under Π , i.e., for every ground rule $L_0 \leftarrow L_1, \dots, L_m$ from the ground instantiation of Π , $\{L_1, \dots, L_m\} \subseteq Cn(\Pi)$ implies $L_0 \in Cn(\Pi)$; and (ii) *logically closed*, i.e., it is either consistent or equal to *Lit*. Given an ELP Π and a set S of literals, S is an *answer set* of Π if $S = Cn(\Pi^S)$.

Answer sets represent possible beliefs of a program, and an ELP may have none, one, or multiple answer sets. In particular, every *not-free* ELP has the unique answer set. An answer set is *consistent* if it is not *Lit*. The set of all answer sets of an ELP Π is written as $\mathcal{AS}(\Pi)$.

Proposition 4.1 An ELP Π has the unique answer set *Lit* iff $Cn(\Pi^+) = \text{Lit}$.

⁴Under the answer set semantics which we consider in this paper, a rule with the empty head $\leftarrow F$ is expressed by the semantically equivalent rule $L \leftarrow F, \text{not } L$ with a literal L .

According to [Gelfond and Lifschitz, 1991], the rule $L_0 \leftarrow L_1, \dots, L_m, \text{not } L_{m+1}, \dots, \text{not } L_n$ is interpreted as the default rule:

$$\frac{L_1 \wedge \dots \wedge L_m : \neg L_{m+1}, \dots, \neg L_n}{L_0}$$

where $\neg \neg L = L$ for a positive literal L . In this case, there is a 1-1 correspondence between the answer sets of a program and the extensions of the corresponding default theory.⁵

Proposition 4.2 [Gelfond and Lifschitz, 1991] *Let Π be an ELP and Δ_Π its corresponding default theory. If S is an answer set of Π , then the deductive closure of S is an extension of Δ_Π . Conversely, every extension of Δ_Π is the deductive closure of exactly one answer set of Π .*

Using the correspondence, a multi-valued interpretation for ELPs is defined under the logic IX .

Definition 4.1 Given an ELP Π , the mapping ϕ_Π associates a positive literal $L \in \text{Lit}$ with a truth value of IX as follows:

$$\phi_\Pi(L) = \begin{cases} \mathbf{t} & \text{if } \mathcal{AS}(\Pi) \neq \emptyset \text{ and } L \in \text{Cn}(\Pi^+); \\ \mathbf{f} & \text{if } \mathcal{AS}(\Pi) \neq \emptyset \text{ and } \neg L \in \text{Cn}(\Pi^+); \\ \mathbf{dts} & \text{if } \mathcal{AS}(\Pi) \neq \emptyset, L \notin \text{Cn}(\Pi^+) \\ & \text{and } \forall S \in \mathcal{AS}(\Pi) L \in S; \\ \mathbf{dfs} & \text{if } \mathcal{AS}(\Pi) \neq \emptyset, \neg L \notin \text{Cn}(\Pi^+) \\ & \text{and } \forall S \in \mathcal{AS}(\Pi) \neg L \in S; \\ \mathbf{dte} & \text{if } \exists S \in \mathcal{AS}(\Pi) \text{ s.t. } L \in S \\ & \text{and } \exists T \in \mathcal{AS}(\Pi) \text{ s.t. } L \notin T; \\ \mathbf{dfe} & \text{if } \exists S \in \mathcal{AS}(\Pi) \text{ s.t. } \neg L \in S \\ & \text{and } \exists T \in \mathcal{AS}(\Pi) \text{ s.t. } \neg L \notin T; \\ \perp & \text{otherwise.} \end{cases}$$

In particular, we write

$$\begin{aligned} \phi_\Pi(L) &= \top \text{ if } \phi_\Pi(L) = \mathbf{t} \text{ and } \phi_\Pi(L) = \mathbf{f}; \\ \phi_\Pi(L) &= * \text{ if } \phi_\Pi(L) = \mathbf{dte} \text{ and } \phi_\Pi(L) = \mathbf{dfe}. \end{aligned}$$

Remark that literals L and $\neg L$ are included in every answer set of Π iff they are in $\text{Cn}(\Pi^+)$ (Proposition 4.1). So it does not happen that $\phi_\Pi(L)$ takes both $\mathbf{dts}$ and $\mathbf{dfs}$ for any L .

The intuitive meaning of ϕ_Π is analogous to that of ϕ_Δ .

Example 4.1 Let Π be the program:

$$p \leftarrow \text{not } q, \quad q \leftarrow \text{not } p, \quad r \leftarrow \text{not } \neg s,$$

which has two answer sets $\{p, r\}$ and $\{q, r\}$. Then $\phi_\Pi(p) = \phi_\Pi(q) = \mathbf{dte}$, $\phi_\Pi(r) = \mathbf{dts}$, and $\phi_\Pi(s) = \perp$.

Example 4.2 Let Π be the program:

$$p \leftarrow \text{not } \neg p, \quad \neg p \leftarrow \text{not } p,$$

which has two answer sets $\{p\}$ and $\{\neg p\}$. Then, $\phi_\Pi(p) = *$.

ϕ_Π has properties obtained from Proposition 3.2 by replacing ϕ_Δ with ϕ_Π and formulas with literals.

Ordering between ELPs is defined as follows.

⁵Precisely speaking, *not*-free rules in an ELP correspond to justification-free defaults. Although we supposed defaults with non-empty justifications in the previous sections, the following discussion is valid apart from the results of Section 3.

Definition 4.2 Let Π_1 and Π_2 be two ELPs which have the same literal base Lit . Then, Π_1 is *stronger* than Π_2 under the answer set semantics (written as $\Pi_2 \leq_{AS} \Pi_1$) if $\phi_{\Pi_2}(L) \leq_k \phi_{\Pi_1}(L)$ for any literal $L \in \text{Lit}$. We write $\Pi_1 \simeq_{AS} \Pi_2$ if $\Pi_1 \leq_{AS} \Pi_2$ and $\Pi_2 \leq_{AS} \Pi_1$.

The relation $\leq_{AS}$ is a pre-order on the set of all ELPs in the language. A program Π_1 is stronger than Π_2 if the answer sets of Π_1 entail more certain information than Π_2 . Different from the case of default logic, we compare programs in terms of literals included in answer sets. This is because in non-monotonic logic programs the meaning of a program is determined not by individual rules in a program, but by consequent literals included in selected models of a program. Thus, we capture the information content of a program as consequences brought by answer sets.

Proposition 4.3 Let Π_1 and Π_2 be two not-free ELPs which have the same literal base. Then, $\Pi_1 \leq_{AS} \Pi_2$ iff $\text{Cn}(\Pi_1) \subseteq \text{Cn}(\Pi_2)$.

Proposition 4.4 Given two ELPs Π_1 and Π_2 , $\Pi_1 \simeq_{AS} \Pi_2$ implies $\mathcal{AS}(\Pi_1) = \mathcal{AS}(\Pi_2)$, but not vice-versa.

Example 4.3 Let Π_1 and Π_2 be two programs:

$$\begin{aligned} \Pi_1 : \quad & p \leftarrow \neg q, \quad \neg q \leftarrow, \\ \Pi_2 : \quad & p \leftarrow \text{not } q, \quad \neg q \leftarrow \text{not } q, \end{aligned}$$

where Π_1 and Π_2 have the same answer set $\{p, \neg q\}$. Then, $\phi_{\Pi_1}(p) = \mathbf{t}$, $\phi_{\Pi_1}(q) = \mathbf{f}$, $\phi_{\Pi_2}(p) = \mathbf{dts}$, and $\phi_{\Pi_2}(q) = \mathbf{dfs}$. Thus, $\Pi_2 \leq_{AS} \Pi_1$.

It is easily verified that the order $\leq_{AS}$ has nonmonotonic properties which correspond to Proposition 3.6 with respect to the introduction of new rules to a program.

A connection between the order relation $\leq_{AS}$ and answer sets is given as follows (The proof is similar to Theorem 3.7).

Theorem 4.5 Let Π_1 and Π_2 be two ELPs which have the same literal base. Then, $\Pi_1 \leq_{AS} \Pi_2$ if the following conditions are satisfied:

1. $\text{Cn}(\Pi_1^+) \subseteq \text{Cn}(\Pi_2^+)$,
2. $\forall S \in \mathcal{AS}(\Pi_2) \exists T \in \mathcal{AS}(\Pi_1) \text{ s.t. } T \subseteq S$,
3. $\forall T \in \mathcal{AS}(\Pi_1) \exists S \in \mathcal{AS}(\Pi_2) \text{ s.t. } T \subseteq S$.

5 Discussion

In the context of multi-valued logics, Ginsberg [1988] firstly introduces a bilattice for default logic. He distinguishes definite and default conclusions obtained from a (super-normal) default theory using the bilattice of Figure 2. However, Ginsberg's bilattice is seven-valued and does not distinguish between skeptical and credulous default conclusions. For instance, suppose the super-normal default theory $\Delta = \{ \frac{p \wedge q}{p \wedge q}, \frac{\neg p}{\neg p} \}$ which has two default extensions $\text{Th}(\{p \wedge q\})$ and $\text{Th}(\{\neg p\})$. Then, $\phi_\Delta(p) = *$, $\phi_\Delta(q) = \mathbf{dte}$, and $\phi_\Delta(\neg p \vee q) = \mathbf{dts}$ in our framework, while Ginsberg interprets p as $*$ but handles both q and $\neg p \vee q$ as $\mathbf{dt}$. Thus, to distinguish skeptical/credulous default inference, additional truth values are necessary as introduced in this paper. Dionisio *et al.* [1998] distinguish skeptical/credulous default inference in super-normal default theories using modal logic.

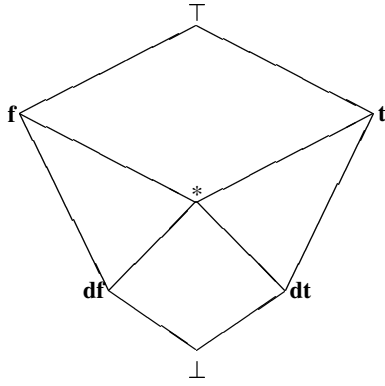


Figure 2: Ginsberg's bilattice for default logic

Their goal is reasoning about defaults and is not ordering default theories.

In logic programming, Fitting [1991] uses bilattices to characterize the semantics of normal logic programs. He uses a four-valued logic and does not distinguish definite and default information. Dix [1992] uses the knowledge ordering under a three-valued logic to compare information obtained from a single normal logic program under different semantics. This is in contrast to our approach in which we compare different programs under the single answer set semantics. Lattice-valued logics are also used for characterizing the “paraconsistent” semantics of logic programs [Damásio and Pereira, 1998]. To our best knowledge, however, the 9-valued bilattice used in this paper never appears in the literature. Moreover, existing studies all use multi-valued logics to provide a semantics of a single program, while we use them to compare information between different programs. The order-equivalence provides a stronger relation than the usual model-based equivalence (Proposition 4.4). On the other hand, it is known another *strong equivalence* relation between logic programs [Lifschitz *et al.*, 2001]. At the moment, we have an evidence that there is no stronger/weaker relation between the strong equivalence and the order-equivalence in general.

From the computational viewpoint, there is a difficulty for directly computing ϕ_Δ for an arbitrary formula F . This is due to the fact that the interpretation ϕ_Δ of a formula F is generally not constructive by those of the sub-formulas of F (Proposition 3.3). The same problem happens in the restricted class of super-normal default theories [Ginsberg, 1988]. For checking an order between default theories, however, Theorem 3.7 provides a sufficient condition to judge the relation $\Delta_1 \leq_{DL} \Delta_2$ using default extensions. In the context of logic programming, $\Pi_1 \leq_{AS} \Pi_2$ is checked by Theorem 4.5 using the existing procedures for computing answer sets.

In the fields of machine learning and inductive logic programming, a theory of generalization has been extensively studied in the context of first-order logic [Nienhuys-Cheng and de Wolf, 1997]. However, generalization under logical entailment $\models$ is not directly applicable to default theories and nonmonotonic logic programs. A default ordering introduced in this paper can order default theories and nonmonotonic logic programs, thereby could give a theoretical

ground for inductive generalization in nonmonotonic logic programs. For instance, for the programs $\Pi = \{flies(x) \leftarrow bird(x), bird(tweety) \leftarrow\}$ and $\Pi' = \{flies(x) \leftarrow bird(x), not abnormal(x), bird(tweety) \leftarrow\}$, the relation $\Pi' \leq_{AS} \Pi$ holds ($flies(tweety)$ has the value t in Π , while it has the value dfs in Π'). Thus, if we read the order $\leq_{AS}$ as “more general”, Π is considered a generalization of Π' . This coincides with the view in the ILP literature [Bain and Muggleton, 1992] in which Π' is a specialization of Π .

6 Conclusion

In this paper, we introduced a multi-valued interpretation of default theories, which can distinguish definite and skeptical/credulous default consequences. Based on this, we developed a theory for ordering default theories and applied the technique to ordering nonmonotonic logic programs. The 9-valued bilattice is used for characterizing other nonmonotonic formalisms which have the same inference modes as default logic. The results of this paper provide a method of comparing default theories or nonmonotonic logic programs in a manner different from the conventional extension-based or model-based standpoint. The proposed framework is considered to have potential application to inductive learning in nonmonotonic logics, which we will investigate in future study.

References

- [Bain and Muggleton, 1992] M. Bain and S. Muggleton. Non-monotonic learning. In: Muggleton (ed.), *Inductive Logic Programming*, Academic Press, pp. 145–161, 1992.
- [Baral and Gelfond, 1994] C. Baral and M. Gelfond. Logic programming and knowledge representation. *J. Logic Programming* 19/20:73–148, 1994.
- [Damásio and Pereira, 1998] C. V. Damásio and L. M. Pereira. A survey of paraconsistent semantics for logic programs. In: *Handbook of Defeasible Reasoning and Uncertainty Management Systems*, vol. 2, D. M. Gabbay and Ph. Smets (eds.), Kluwer Academic, pp. 241–320, 1998.
- [Dionísio *et al.*, 1998] F. M. Dionísio, S. Brass, M. Ryan, U. Lipeck. Hypothetical reasoning with defaults. In: *Proc. Workshop on Computational Aspects of NMR*, 1998.
- [Dix, 1992] J. Dix. A framework for representing and characterizing semantics of logic programs. In: *Proc. KR'92*, Morgan Kaufmann, pp. 591–602, 1992.
- [Fitting, 1991] M. Fitting. Bilattices and the semantics of logic programming. *J. Logic Program.* 11:91–116, 1991.
- [Gelfond and Lifschitz, 1991] M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing* 9:365–385, 1991.
- [Ginsberg, 1988] M. L. Ginsberg. Multi-valued logics: a uniform approach to inference in artificial intelligence. *Computational Intelligence* 4:265–316, 1988.
- [Lifschitz *et al.*, 2001] V. Lifschitz, D. Pearce, and A. Valverde. Strongly equivalent logic programs. *ACM Trans. Computational Logic* 2:526–541, 2001.
- [Nienhuys-Cheng and de Wolf, 1997] S.-H. Nienhuys-Cheng and R. de Wolf. *Foundations of inductive logic programming*. LNAI 1228, Springer, 1997.
- [Reiter, 1980] R. Reiter. A logic for default reasoning. *Artificial Intelligence* 13:81–132, 1980.

NONMONOTONIC REASONING

LOGIC PROGRAMMING

Aggregate Functions in Disjunctive Logic Programming: Semantics, Complexity, and Implementation in DLV*

Tina Dell’Armi
Dept. of Mathematics,
Univ. of Calabria
87030 Rende (CS), Italy
dellarmi@mat.unical.it

Wolfgang Faber
Inst. f. Informationssysteme 184/3,
TU Wien
A-1040 Wien, Austria
faber@kr.tuwien.ac.at

Giuseppe Ielpa
Dept. of Mathematics,
Univ. of Calabria
87030 Rende (CS), Italy
ielpa@mat.unical.it

Nicola Leone
Dept. of Mathematics, Univ. of Calabria
87030 Rende (CS), Italy
leone@unical.it

Gerald Pfeifer
Inst. f. Informationssysteme 184/2, TU Wien
A-1040 Wien, Austria
pfeifer@dbai.tuwien.ac.at

Abstract

Disjunctive Logic Programming (DLP) is a very expressive formalism: it allows to express every property of finite structures that is decidable in the complexity class Σ_2^P (NP^{NP}). Despite the high expressiveness of DLP, there are some simple properties, often arising in real-world applications, which cannot be encoded in a simple and natural manner. Among these, properties requiring to apply some arithmetic operators (like sum, times, count) on a set of elements satisfying some conditions, cannot be naturally expressed in DLP. To overcome this deficiency, in this paper we extend DLP by aggregate functions. We formally define the semantics of the new language, named DLP^A . We show the usefulness of the new constructs on relevant knowledge-based problems. We analyze the computational complexity of DLP^A , showing that the addition of aggregates does not bring a higher cost in that respect. We provide an implementation of the DLP^A language in DLV – the state-of-the-art DLP system – and report on experiments which confirm the usefulness of the proposed extension also for the efficiency of the computation.

1 Introduction

Expressiveness of DLP. Disjunctive Logic Programs (DLP) are logic programs where (nonmonotonic) negation may occur in the bodies, and disjunction may occur in the heads of rules. This language is very expressive in a precise mathematical sense: it allows to express every property of finite structures that is decidable in the complexity class Σ_2^P (NP^{NP}). Therefore, under widely believed assumptions,

*This work was supported by the European Commission under projects IST-2002-33570 INFOMIX, IST-2001-32429 ICONS, and FET-2001-37004 WASP.

DLP is strictly more expressive than *normal* (disjunction-free) logic programming, whose expressiveness is limited to properties decidable in NP. DLP can thus express problems which cannot be translated to Satisfiability of CNF formulas in polynomial time. Importantly, besides enlarging the class of applications which can be encoded in the language, disjunction often allows for representing problems of lower complexity in a simpler and more natural fashion (see [Eiter *et al.*, 2000]).

The problem. Despite this high expressiveness, there are some simple properties, often arising in real-world applications, which cannot be encoded in DLP in a simple and natural manner. Among these are properties requiring to apply some arithmetic operator (e.g., sum, times, count) on a set of elements satisfying some conditions. Suppose, for instance, that you want to know if the sum of the salaries of the employees working in a team exceeds a given budget (see **Team Building**, in Section 3). To this end, you should first order the employees defining a successor relation. You should then define a *sum* predicate, in a recursive way, which computes the sum of all salaries, and compare its result with the given budget. This approach has two drawbacks: (1) It is bad from the KR perspective, as the encoding is not natural at all; (2) it is inefficient, as the (instantiation of the) program is quadratic (in the cardinality of the input set of employees). Thus, there is a clear need to enrich DLP with suitable constructs for the natural representation and to provide means for an efficient evaluation of such properties.

Contribution. We overcome the above deficiency of DLP. Instead of inventing new constructs from scratch, we extend the language with a sort of aggregate functions, first studied in the context of deductive databases, and implement them in DLV [Eiter *et al.*, 2000] – the state-of-the-art Disjunctive Logic Programming system. The main contributions of this paper are the following.

- We extend Disjunctive Logic Programming by aggregate functions and formally define the semantics of the resulting

language, named DLP^A.

- We address knowledge representation issues, showing the impact of the new constructs on relevant problems.
- We analyze the computational complexity of DLP^A. Importantly, it turns out that the addition of aggregates does not increase the computational complexity, which remains the same as for reasoning on DLP programs.
- We provide an implementation of DLP^A in the DLV system, deriving new algorithms and optimization techniques for the efficient evaluation.
- We report on experimentation, evaluating the impact of the proposed language extension on efficiency. The experiments confirm that, besides providing relevant advantages from the knowledge representation point of view, aggregate functions can bring significant computational gains.
- We compare DLP^A with related work.

We present the most relevant aspects of DLP^A and of its implementation here, referring the interested reader to a technical report with all details [Dell’Armi *et al.*, 2003].

2 The DLP^A Language

In this section, we provide a formal definition of the syntax and semantics of the DLP^A language – an extension of DLP by set-oriented functions (also called aggregate functions). We assume that the reader is familiar with standard DLP; we refer to atoms, literals, rules, and programs of DLP, as *standard atoms*, *standard literals*, *standard rules*, and *standard programs*, respectively. For further background, see [Gelfond and Lifschitz, 1991; Eiter *et al.*, 2000].

2.1 Syntax

A (DLP^A) *set* is either a symbolic set or a ground set. A *symbolic set* is a pair $\{Vars : Conj\}$, where *Vars* is a list of variables and *Conj* is a conjunction of standard literals.¹ A *ground set* is a set of pairs of the form $\langle \bar{t} : Conj \rangle$, where $\bar{t}$ is a list of constants and *Conj* is a ground (variable free) conjunction of standard literals. An *aggregate function* is of the form $f(S)$, where S is a set, and f is a *function name* among $\#count$, $\#min$, $\#max$, $\#sum$, $\#times$. An *aggregate atom* is $Lg \prec_1 f(S) \prec_2 Rg$, where $f(S)$ is an aggregate function, $\prec_1, \prec_2 \in \{=, <, \leq, >, \geq\}$, and Lg and Rg (called *left guard*, and *right guard*, respectively) are terms. One of “ $Lg \prec_1$ ” and “ $\prec_2 Rg$ ” can be omitted. An *atom* is either a standard (DLP) atom or an aggregate atom. A *literal* L is an atom A or an atom A preceded by the default negation symbol *not*; if A is an aggregate atom, L is an *aggregate literal*.

A (DLP^A) *rule* r is a construct

$$a_1 \vee \dots \vee a_n :- b_1, \dots, b_k, \text{not } b_{k+1}, \dots, \text{not } b_m.$$

where $a_1, \dots, a_n$ are standard atoms, $b_1, \dots, b_m$ are atoms, and $n \geq 0, m \geq k \geq 0$. The disjunction $a_1 \vee \dots \vee a_n$ is the *head* of r , while the conjunction $b_1, \dots, b_k, \text{not } b_{k+1}, \dots, \text{not } b_m$ is the *body* of r . A (DLP^A) *program* is a set of DLP^A rules.

Syntactic Restrictions and Notation

¹Intuitively, a symbolic set $\{X : a(X, Y), p(Y)\}$ stands for the set of X -values making $a(X, Y), p(Y)$ true, i.e., $\{X : \exists Y s.t. a(X, Y), p(Y) \text{ is true}\}$. Note that also negative literals may occur in the conjunction *Conj* of a symbolic set.

For simplicity, and without loss of generality, we assume that the body of each rule contains at most one aggregate atom. A *global* variable of a rule r is a variable appearing in some standard atom of r ; a *local* variable of r is a variable appearing solely in an aggregate function in r .

Safety. A rule r is *safe* if the following conditions hold: (i) each global variable of r appears in a positive standard literal in the body of r ; (ii) each local variable of r appearing in a symbolic set $\{Vars : Conj\}$, also appears in a positive literal in *Conj*; (iii) each guard of an aggregate atom of r is either a constant or a global variable. A program is *safe* if all of its rules are *safe*.

Example 1 Consider the following rules:

$$\begin{aligned} p(X) &:- q(X, Y, V), Y < \#max\{Z : r(Z), \text{not } a(Z, V)\}. \\ p(X) &:- q(X, Y, V), Y < \#sum\{Z : \text{not } a(Z, S)\}. \\ p(X) &:- q(X, Y, V), T < \#min\{Z : r(Z), \text{not } a(Z, V)\}. \end{aligned}$$

The first rule is *safe*, while the second is not, since both local variables Z and S violate condition (ii). The third rule is not *safe* either, since the guard T is not a global variable, violating condition (iii).

Stratification. A DLP^A program $\mathcal{P}$ is *aggregate-stratified* if there exists a function $\| \cdot \|$, called *level mapping*, from the set of (standard) predicates of $\mathcal{P}$ to ordinals, such that for each pair a and b of (standard) predicates of $\mathcal{P}$, and for each rule $r \in \mathcal{P}$: (i) if a appears in the head of r , and b appears in an aggregate atom in the body of r , then $\|b\| < \|a\|$, and (ii) if a appears in the head of r , and b occurs in a standard atom in the body of r , then $\|b\| \leq \|a\|$.

Example 2 Consider the program consisting of a set of facts for predicates a and b , plus the following two rules:

$$\begin{aligned} q(X) &:- p(X), \#count\{Y : a(Y, X), b(X)\} \leq 2. \\ p(X) &:- q(X), b(X). \end{aligned}$$

The program is *aggregate-stratified*, as the level mapping $\|a\| = \|b\| = 1 \quad \|p\| = \|q\| = 2$ satisfies the required conditions. If we add the rule $b(X) :- p(X)$, then no legal level-mapping exists and the program becomes *aggregate-unstratified*.

Intuitively, *aggregate-stratification* forbids recursion through aggregates, which could cause an unclear semantics in some cases. Consider, for instance, the (aggregate-unstratified) program consisting only of rule $p(a) :- \#count\{X : p(X)\} = 0$. Neither $p(a)$ nor $\emptyset$ is an intuitive meaning for the program. We should probably assert that the above program does not have any answer set (defining a notion of “stability” for aggregates), but then positive programs would not always have an answer set if there is no integrity constraint. In the following we assume that DLP^A programs are *safe* and *aggregate-stratified*.

2.2 Semantics

Given a DLP^A program $\mathcal{P}$, let $U_{\mathcal{P}}$ denote the set of constants appearing in $\mathcal{P}$, $U_{\mathcal{P}}^N \subseteq U_{\mathcal{P}}$ the set of the natural numbers occurring in $U_{\mathcal{P}}$, and $B_{\mathcal{P}}$ the set of standard atoms constructible from the (standard) predicates of $\mathcal{P}$ with constants in $U_{\mathcal{P}}$. Furthermore, given a set X , $\bar{2}^X$ denotes the set of all multi-sets over elements from X . Let us now describe the domains and the meanings of the aggregate functions we consider: $\#count$: defined over $\bar{2}^{U_{\mathcal{P}}}$, the number of elements in the set.

$\#sum$: defined over $\overline{2}^{U_P}$, the sum of the numbers in the set.
 $\#times$: over $\overline{2}^{U_P}$, the product of the numbers in the set.²
 $\#min$, $\#max$: defined over $\overline{2}^{U_P} - \{\emptyset\}$, the minimum/maximum element in the set; if the set contains also strings, the lexicographic ordering is considered.³
 If the argument of an aggregate function does not belong to its domain, the aggregate evaluates to false, denoted as $\perp$ (and our implementation issues a warning in this case).

A *substitution* is a mapping from a set of variables to the set U_P of the constants appearing in the program P . A substitution from the set of global variables of a rule r (to U_P) is a *global substitution* for r ; a substitution from the set of local variables of a symbolic set S (to U_P) is a *local substitution* for S . Given a symbolic set without global variables $S = \{Vars : Conj\}$, the *instantiation of set S* is the following ground set of pairs $inst(S)$:

$\{\langle \gamma(Vars) : \gamma(Conj) \rangle \mid \gamma \text{ is a local substitution for } S\}$.⁴

A *ground instance* of a rule r is obtained in two steps: (1) a global substitution σ for r is first applied over r ; (2) every symbolic set S in $\sigma(r)$ is replaced by its instantiation $inst(S)$. The instantiation $Ground(P)$ of a program P is the set of all possible instances of the rules of P .

Example 3 Consider the following program P_1 :

$q(1) \vee p(2, 2). \quad q(2) \vee p(2, 1). \\ t(X) :- q(X), \#sum\{Y : p(X, Y)\} > 1.$

The instantiation $Ground(P_1)$ is the following:

$q(1) \vee p(2, 2). \quad q(2) \vee p(2, 1). \\ t(1) :- q(1), \#sum\{1 : p(1, 1), 2 : p(1, 2)\} > 1. \\ t(2) :- q(2), \#sum\{1 : p(2, 1), 2 : p(2, 2)\} > 1.$

An *interpretation* for a DLP^A program P is a set of standard ground atoms $I \subseteq B_P$. The truth valuation $I(A)$, where A is a standard ground literal or a standard ground conjunction, is defined in the usual way. Besides assigning truth values to the standard ground literals, an interpretation provides the meaning also to (ground) sets, aggregate functions and aggregate literals; the meaning of a set, an aggregate function, and an aggregate atom under an interpretation, is a multiset, a value, and a truth-value, respectively. Let $f(S)$ be an aggregate function. The valuation $I(S)$ of set S w.r.t. I is the multiset of the first constant of the first components of the elements in S whose conjunction is true w.r.t. I . More precisely, let $S_I = \{\langle t_1, \dots, t_n \rangle \mid \langle t_1, \dots, t_n : Conj \rangle \in S \wedge Conj \text{ is true w.r.t. } I\}$, then $I(S)$ is the multiset $[t_1 \mid \langle t_1, \dots, t_n \rangle \in S_I]$. The valuation $I(f(S))$ of an aggregate function $f(S)$ w.r.t. I is the result of the application of the function f on $I(S)$. (If the multiset $I(S)$ is not in the domain of f , $I(f(S)) = \perp$.)

An aggregate atom $A = Lg \prec_1 f(S) \prec_2 Rg$ is true w.r.t. I if: (i) $I(f(S)) \neq \perp$, and, (ii) the relationships $Lg \prec_1 I(f(S))$, and $I(f(S)) \prec_2 Rg$ hold whenever they are present; otherwise, A is false.

Using the above notion of truth valuation for aggregate atoms, the truth valuations of aggregate literals and rules, as

² $\#sum$ and $\#times$ applied over an empty set return 0 and 1, respectively.

³The latter is not yet supported in our first implementation.

⁴Given a substitution σ and a DLP^A object Obj (rule, conjunction, set, etc.), with a little abuse of notation, we denote by $\sigma(Obj)$ the object obtained by replacing each variable X in Obj by $\sigma(X)$.

well as the notion of model, minimal model, and answer set for DLP^A are an immediate extension of the corresponding notions in DLP [Gelfond and Lifschitz, 1991].

Example 4 Consider the aggregate atom $A = \#sum\{\langle 1 : p(2, 1) \rangle, \langle 2 : p(2, 2) \rangle\} > 1$ from Example 3. Let S be the ground set appearing in A . For interpretation $I = \{q(2), p(2, 2), t(2)\}$, $I(S) = [2]$, the application of $\#sum$ over $[2]$ yields 2, and A is therefore true w.r.t. I , since $2 > 1$. I is an answer set of the program of Example 3.

3 Knowledge Representation in DLP^A

In this section, we show how aggregate functions can be used to encode relevant problems.

Team Building. A project team has to be built from a set of employees according to the following specifications:

- (p_1) The team consists of a certain number of employees.
- (p_2) At least a given number of different skills must be present in the team.
- (p_3) The sum of the salaries of the employees working in the team must not exceed the given budget.
- (p_4) The salary of each individual employee is within a specified limit.
- (p_5) The number of women working in the team has to reach at least a given number.

Suppose that our employees are provided by a number of facts of the form $emp(Empld, Sex, Skill, Salary)$; the size of the team, the minimum number of different skills, the budget, the maximum salary, and the minimum number of women are specified by the facts $nEmp(N)$, $nSkill(N)$, $budget(B)$, $maxSal(M)$, and $women(W)$. We then encode each property p_i above by an aggregate atom A_i , and enforce it by an integrity constraint containing not A_i .

$in(I) \vee out(I) :- emp(I, Sx, Sk, Sa). \\ :- nEmp(N), not \#count\{I : in(I)\} = N. \\ :- nSkill(M), not \#count\{Sk : emp(I, Sx, Sk, Sa), in(I)\} \geq M. \\ :- budget(B), not \#sum\{Sa, I : emp(I, Sx, Sk, Sa), in(I)\} \leq B. \\ :- maxSal(M), not \#max\{Sa : emp(I, Sx, Sk, Sa), in(I)\} \leq M. \\ :- women(W), not \#count\{I : emp(I, f, Sk, Sa), in(I)\} \geq W.$

Intuitively, the disjunctive rule “guesses” whether an employee is included in the team or not, while the five constraints correspond one-to-one to the five requirements p_1 - p_5 . Thanks to the aggregates the translation of the specifications is surprisingly straightforward. The example highlights the usefulness of representing both sets and multisets in our language (a multiset can be obtained by specifying more than one variable in $Vars$ of a symbolic set $\{Vars : Conj\}$). For instance, the encoding of p_2 requires a set, as we want to count *different* skills; two employees in the team having the same skill, should count once w.r.t. p_2 . On the contrary, p_3 requires to sum the elements of a multiset; if two employees have the same salary, *both* salaries should be summed up for p_3 . This is obtained by adding the variable I to $Vars$. The valuation of $\{Sa, I : emp(I, Sx, Sk, Sa), in(I)\}$ yields the set $S' = \{\langle Sa, I \rangle : Sa \text{ is the salary of employee } I \text{ in the team}\}$. Then, the sum function is applied on the multiset of the first components Sa of the tuples $\langle Sa, I \rangle$ in S' (see Section 2.2).

Seating. We have to generate a sitting arrangement for a number of guests, with m tables and n chairs per table.

Guests who like each other should sit at the same table; guests who dislike each other should not sit at the same table.

Suppose that the number of chairs per table is specified by $nChairs(X)$ and that $person(P)$ and $table(T)$ represent the guests and the available tables, respectively. Then, we can generate a seating arrangement by the following program:

```
% Guess whether person P sits at table T or not.
at(P, T) ∨ not_at(P, T) :- person(P), table(T).
% The persons sitting at a table cannot exceed the chairs.
:- table(T), nChairs(C), not #count{P : at(P, T)} ≤ C.
% A person is seated at precisely one table; equivalent
% to :- person(P), at(P, T), at(P, U), T <> U.
:- person(P), not #count{T : at(P, T)} = 1.
% People who like each other should sit at the same table...
:- like(P1, P2), at(P1, T), not at(P2, T).
% ...while people who dislike each other should not.
:- dislike(P1, P2), at(P1, T), at(P2, T).
```

4 Computational Complexity of DLP^A

As for the classical nonmonotonic formalisms [Marek and Truszczyński, 1991], two important decision problems, corresponding to two different reasoning tasks, arise in DLP^A:

(Brave Reasoning) Given a DLP^A program $\mathcal{P}$ and a ground literal L , is L true in some answer set of $\mathcal{P}$?

(Cautious Reasoning) Given a DLP^A program $\mathcal{P}$ and a ground literal L , is L true in all answer sets of $\mathcal{P}$?

The following theorems report on the complexity of the above reasoning tasks for propositional (i.e., variable-free) DLP^A programs that respect the syntactic restrictions imposed in Section 2 (safety and aggregate-stratification). Importantly, it turns out that reasoning in DLP^A does not bring an increase in computational complexity, which remains exactly the same as for standard DLP. (See [Dell’Armi *et al.*, 2003] for the proofs.)

Theorem 5 *Brave Reasoning on ground DLP^A programs is Σ_2^P -complete.*

Theorem 6 *Cautious Reasoning on ground DLP^A programs is Π_2^P -complete.*

5 Implementation Issues

The implementation of DLP^A required changes to all modules of DLV. Apart from a preliminary standardization phase, most of the effort concentrated on the Intelligent Grounding and Model Generator modules.

Standardization. After parsing, each aggregate A is transformed such that both guards are present and both $\prec_1$ and $\prec_2$ are set to $\leq$. The conjunction $Conj$ of the symbolic set of A is replaced by a single, new atom Aux and a rule $Aux :- Conj$ is added to the program (the arguments of Aux being the distinct variables of $Conj$).

Instantiation. The goal of the instantiator is to generate a ground program which has precisely the same answer sets as the theoretical instantiation $Ground(\mathcal{P})$, but is sensibly smaller. The instantiation proceeds bottom-up following the dependencies induced by the rules, and, in particular, respecting the ordering imposed by the aggregate-stratification. Let “ $H :- B, aggr.$ ” be a rule, where H is the head of the rule, B is the conjunction of the standard body literals, and $aggr$

is an aggregate literal over a symbolic set $\{Vars : Aux\}$. First we compute an instantiation $\overline{B}$ for the literals in B ; this binds the global variables appearing in Aux . The (partially bound) atom Aux is then matched against its extension (since the bottom-up instantiation respects the stratification, the extension of Aux is already available), all matching facts are computed, and a set of pairs $\{\langle \theta_1(Vars) : \theta_1(\overline{Aux}) \rangle, \dots, \langle \theta_n(Vars) : \theta_n(\overline{Aux}) \rangle\}$ is generated, where θ_i is a substitution for the local variables in Aux such that $\theta_i(\overline{Aux})$ is an admissible instance of Aux (recall that DLV’s instantiator produces only those instances of a predicate which can potentially become true [Faber *et al.*, 1999a; Leone *et al.*, 2001])⁵. Also, we only store those elements of the symbolic set whose truth value cannot be determined yet and process the others dynamically, (partially) evaluating the aggregate already during instantiation. The same process is then repeated for all further instantiations of the literals in B .

Example 7 Consider the following rule r :

$$p(X) :- q(X), 1 < \#count\{Y : a(X, Y), \text{not } b(Y)\}.$$

The standardization rewrites r to:

$$p(X) :- q(X), 2 \leq \#count\{Y : aux(X, Y)\} \leq \infty.$$

$$aux(X, Y) :- a(X, Y), \text{not } b(Y).$$

Suppose that the instantiation of the rule for aux generates 3 potentially true facts for aux : $aux(1, a)$, $aux(1, b)$, and $aux(2, c)$. If the potentially true facts for q are $q(1)$ and $q(2)$, the following ground instances are generated:⁶

$$p(1) :- q(1), 2 \leq \#count\{a : aux(1, a)\}, \{b : aux(1, b)\} \leq \infty.$$

$$p(2) :- q(2), 2 \leq \#count\{c : aux(2, c)\} \leq \infty.$$

Duplicate Sets Recognition. To optimize the evaluation, we have designed a hashing technique which recognizes multiple occurrences of the same set in the program, even in different rules, and stores them only once. This saves memory (sets may be very large), and also implies a significant performance gain, especially in the model generation where sets are frequently manipulated during the backtracking process.

Example 8 Consider the following two constraints:

$$c_1 :- 10 \leq \#max\{V : d(V, X)\}.$$

$$c_2 :- \#min\{Y : d(Y, Z)\} \leq 5.$$

Our technique recognizes that the two sets are equal, and generates only one instance which is shared by c_1 and c_2 .

Now assume that both constraints additionally contain a standard literal $p(X)$. In this case, c_1 and c_2 have n instances each, where n is the number of facts for $p(X)$. By means of our technique, each pair of instances of c_1 and c_2 shares a common set, reducing the number of instantiated sets by half.

Model Generation. We have designed an extension of the Deterministic Consequences operator of the DLV system [Faber *et al.*, 1999b] for DLP^A programs. The new operator makes both forward and backward inferences on aggregate atoms, resulting in an effective pruning of the search space. We have then extended the Dowling and Gallier algorithm [Dowling and Gallier, 1984] to compute a fixpoint of

⁵A ground atom A can potentially become true only if we have generated a ground instance with A in the head.

⁶Note that a ground set contains only those aux atoms which are potentially true.

this operator in linear time using a multi-linked data structure of pointers. Given a ground set T , say, $\{\langle t_1^1, \dots, t_n^1 : Aux^1 \rangle, \dots, \langle t_1^m, \dots, t_n^m : Aux^m \rangle\}$, this structure allows us to access T in $O(1)$ whenever some Aux^i changes its truth value (supporting fast forward propagation); on the other hand, it provides direct access from T to each Aux^i atom (supporting fast backward propagation).

6 Experiments and Benchmarks

To assess the usefulness of the proposed DLP extension and evaluate its implementation, we compare the following two methods for solving a given problem:

- DLV^A . Encode the problem in DLP^A and solve it by using our extension of DLV with aggregates.
- DLV. Encode the problem in standard DLP and solve it by using standard DLV.

To generate DLP encodings from DLP^A encodings, suitable logic definitions of the aggregate functions are employed (which are recursive for $\#count$, $\#sum$, and $\#times$).

We compare these methods on two benchmark problems:

Time Tabling is a classical planning problem. In particular, we consider the problem of planning the timetable of lectures which some groups of students have to take. We consider a number of real-world instances at our University, where instance k deals with k groups.

Seating is the problem described in Section 3. We consider 4 (for small instances) or 5 (for larger instances) seats per table, with increasing numbers of tables and persons (with $numPersons = numSeats * numTables$). For each problem size (i.e., seats/tables configuration), we consider classes with different numbers of like resp. dislike constraints, where the percentages are relative to the maximum number of like resp. dislike constraints such that the problem is not over-constrained⁷. In particular, we consider the following classes: -) no like/dislike constraints at all; -) 25% like constraints; -) 25% like and 25% dislike constraints; -) 50% like constraints; -) 50% like and 50% dislike constraints. For each problem size, we have randomly generated 10 instances for each class above.

For Seating we use the DLP^A encoding reported in Section 3; all encodings and benchmark data are also available on the web at <http://www.dlvsystem.com/examples/ijcai03.zip>.

Number of Groups	Execution Time		Instantiation Size	
	DLV	DLV^A	DLV	DLV^A
1	10.95	0.55	91217	6972
2	36.79	2.05	178533	13986
3	79.84	4.68	264938	20888
4	147.53	7.86	367014	29029
5	224.46	12.30	436544	36043
6	321.85	17.18	518950	42767
7	437.94	25.36	606361	49993
8	618.23	37.78	761429	61916
9	-	57.00	-	74027

Figure 1: Experimental Results for Timetabling

⁷Beyond these maxima there is trivially no solution.

Number of Persons	Execution Time		Instantiation Size	
	DLV	DLV^A	DLV	DLV^A
8	0.010	0.010	320	101
12	0.034	0.010	996	248
16	26.872	0.011	2272	490
25	-	0.024	6643	1346
50	-	0.307	50029	7559
75	-	1.883	165442	22049
100	-	7.082	387886	47946
125	-	64.293	752769	88781
150	-	152.450	1294977	147567

Figure 2: Experimental Results for Seating

We ran the benchmarks on AMD Athlon 1.2 machines with 512MB of memory, using FreeBSD 4.7 and GCC 2.95. We have allowed a maximum running time of 1800 seconds per instance and a maximum memory usage of 256MB. Cumulated results are provided in Figures 1 and 2. In particular, for Timetabling we report the execution time and the size of the residual ground instantiation (the total number of atoms occurring in the instantiation, where multiple occurrences of the same atom are counted).⁸ For Seating, the execution time is the average running time over the instances of the same size. A “-” symbol in the tables indicates that the corresponding instance (some of the instances of that size, for Seating) was not solved within the allowed time and memory limits.

On both problems, DLV^A clearly outperforms DLV. On Timetabling, the execution time of DLV^A is one order of magnitude lower than that of DLV on all problem instances, and DLV could not solve the last instances within the allowed memory and time limits. On Seating, the difference becomes even more significant. DLV could solve only the instances of small size (up to 16 persons - 4 tables, 4 seats); while DLV^A could solve significantly larger instances in a reasonable time. The information about the instantiation sizes provides an explanation for such a big difference between the execution times of DLV and DLV^A . Thanks to the aggregates, the DLP^A encodings of Timetabling and Seating are more succinct than the corresponding encodings in standard DLP; this succinctness is also reflected in the ground instantiations of the programs. Since the evaluation algorithms are then exponential (in the worst case) in the size of the instantiation, the execution times of DLV^A turn out to be much shorter than the execution times of DLV.

7 Related Works

Aggregate functions in logic programming languages appeared already in the 80s, when their need emerged in deductive databases like LDL [Chimenti *et al.*, 1990] and were studied in detail, cf. [Ross and Sagiv, 1997; Kemp and Ramamohanarao, 1998]. However, the first implementation in Answer Set Programming, based on the Smodels system, is recent [Simons *et al.*, 2002].

Comparing DLP^A to the language of Smodels, we observe a strong similarity between cardinality constraints there and $\#count$. Also $\#sum$ and weight constraints in Smodels are

⁸Note that also atoms occurring in the sets of the aggregates are counted for the instantiation size.

similar in spirit. Indeed, the DLP^A encodings of both Team Building and Seating can be easily translated to Smodels' language. However, there are some relevant differences. For instance, in DLP^A aggregate atoms can be negated, while cardinality and weight constraint literals in Smodels cannot. Smodels, on the other hand, allows for weight constraints in the heads of rules, while DLP^A aggregates cannot occur in heads. (The presence of weight constraints in heads is a powerful KR feature; however, it causes the loss of some semantic property of nonmonotonic languages [Marek and Remmel, 2002].) Observe also that DLP^A aggregates like $\#min$, $\#max$, and $\#times$ do not have a counterpart in Smodels. Moreover, DLP^A provides a general framework where further aggregates can be easily accommodated (e.g., $\#any$ and $\#avg$ are already under development). Furthermore, note that symbolic sets of DLP^A directly represent pure (mathematical) sets, and can also represent multisets rather naturally (see the discussion on Team Building in Section 3). Smodels weight constraints, instead, work on multisets, and additional rules are needed to encode pure sets; for instance, Condition p_2 of Team Building cannot be encoded directly in a constraint, but needs the definition of an extra predicate. A positive aspect of Smodels is that, thanks to stricter safety conditions (all variables are to be restricted by *domain predicates*), it is able to deal with recursion through aggregates, which is forbidden in DLP^A . Finally, note that DLP^A deals with sets of terms, while Smodels deals with sets of atoms. As far as the implementation is concerned, also Smodels is endowed with advanced pruning operators for weight constraints, which are efficiently implemented; we are not aware, though, of techniques for the automatic recognition of duplicate sets in Smodels.

DLP^A also seems to be very similar to a special case of the semantics for aggregates discussed in [Gelfond, 2002], which we are currently investigating.

Another interesting research line uses 4-valued logics and approximating operators to define the semantics of aggregate functions in logic-based languages [Denecker et al., 2001; 2002; Pelov, 2002]. These approaches are founded on very solid theoretical grounds, and appear very promising, as they could provide a clean formalization of a very general framework for arbitrary aggregates in logic programming and nonmonotonic reasoning, where aggregate atoms can also “produce” new values (currently, in both DLP^A and Smodels the guards of the aggregates need to be bound to some value). However, these approaches sometimes amount to a higher computational complexity [Pelov, 2002], and there is no implementation available so far.

8 Conclusion

We have proposed DLP^A , an extension of DLP by aggregate functions, and have implemented DLP^A in the DLV system. On the one hand, we have demonstrated that the aggregate functions increase the knowledge modeling power of DLP, supporting a more natural and concise knowledge representation. On the other hand, we have shown that aggregate functions do not increase the complexity of the main reasoning tasks. Moreover, the experiments have confirmed that

the succinctness of the encodings employing aggregates has a strong positive impact on the efficiency of the computation.

Future work will concern the introduction of further aggregate operators, the relaxation of the syntactic restrictions of DLP^A , and the design of further optimization techniques and heuristics to improve the efficiency of the computation.

We thank the anonymous reviewers for their thoughtful comments and suggestions for improvements of this paper.

References

- [Chimenti et al., 1990] D. Chimenti, R. Gamboa, R. Krishnamurthy, S. Naqvi, S. Tsur, and C. Zaniolo. The LDL System Prototype. *IEEE TKDE*, 2(1), 1990.
- [Dell'Armi et al., 2003] T. Dell'Armi, W. Faber, G. Ielpa, N. Leone, and G. Pfeifer. Semantics and Computation of Aggregate Functions in Disjunctive Logic Programming. Technical Report INFSYS RR-1843-03-07, TU Wien, April 2003.
- [Denecker et al., 2001] M. Denecker, N. Pelov, and M. Bruynooghe. Well-Founded and Stable Model Semantics for Logic Programs with Aggregates. *ICLP-2001*, pp. 212–226. Springer, 2001.
- [Denecker et al., 2002] M. Denecker, V. Marek, and M. Truszczyński. Ultimate Approximations in Monotonic Knowledge Representation Systems. *KR-2002*, pp. 177–188.
- [Dowling and Gallier, 1984] W. F. Dowling and J. H. Gallier. Linear-time Algorithms for Testing the Satisfiability of Propositional Horn Formulae. *JLP*, 3:267–284, 1984.
- [Eiter et al., 2000] T. Eiter, W. Faber, N. Leone, and G. Pfeifer. Declarative Problem-Solving Using the DLV System. *Logic-Based Artificial Intelligence*, pp. 79–103. Kluwer, 2000.
- [Faber et al., 1999a] W. Faber, N. Leone, C. Mateis, and G. Pfeifer. Using Database Optimization Techniques for Nonmonotonic Reasoning. *DDL'99*, pp. 135–139.
- [Faber et al., 1999b] W. Faber, N. Leone, and G. Pfeifer. Pushing Goal Derivation in DLP Computations. *LPNMR'99*, pp. 177–191. Springer.
- [Gelfond and Lifschitz, 1991] M. Gelfond and V. Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 9:365–385, 1991.
- [Gelfond, 2002] M. Gelfond. Representing Knowledge in A-Prolog. *Computational Logic. Logic Programming and Beyond*, number 2408 in LNCS, pp. 413–451. Springer, 2002.
- [Kemp and Ramamohanarao, 1998] D. B. Kemp and K. Ramamohanarao. Efficient Recursive Aggregation and Negation in Deductive Databases. *IEEE TKDE*, 10:727–745, 1998.
- [Leone et al., 2001] N. Leone, S. Perri, and F. Scarcello. Improving ASP Instantiators by Join-Ordering Methods. *LPNMR'01*, LNAI 2173. Springer, September 2001.
- [Marek and Remmel, 2002] V.W. Marek and J.B. Remmel. On Logic Programs with Cardinality Constraints. *NMR'2002*, pp. 219–228, April 2002.
- [Marek and Truszczyński, 1991] V.W. Marek and M. Truszczyński. Autoepistemic Logic. *JACM*, 38(3):588–619, 1991.
- [Pelov, 2002] N. Pelov. Non-monotone Semantics for Logic Programs with Aggregates. <http://www.cs.kuleuven.ac.be/~pelov/papers/nma.ps.gz>, October 2002.
- [Ross and Sagiv, 1997] K. A. Ross and Y. Sagiv. Monotonic Aggregation in Deductive Databases. *Journal of Computer and System Sciences*, 54(1):79–97, February 1997.
- [Simons et al., 2002] P. Simons, I. Niemelä, and T. Soininen. Extending and Implementing the Stable Model Semantics. *Artificial Intelligence*, 138:181–234, June 2002.

On Tight Logic Programs and Yet Another Translation from Normal Logic Programs to Propositional Logic

Fangzhen Lin and Jicheng Zhao*

Department of Computer Science

Hong Kong University of Science and Technology

Clear Water Bay, Kowloon, Hong Kong

{flin,jczhao}@cs.ust.hk

Abstract

Fages showed that if a program is tight, then every propositional model of its completion is also its stable model. Recently, Babovich, Erdem, and Lifschitz generalized Fages' result, and showed that this is also true if the program is tight *on the given model* of the completion. As it turned out, this is quite a general result. Among the commonly known benchmark domains, only Niemelä's normal logic program encoding of the Hamiltonian Circuit (HC) problem does not have this property. In this paper, we propose a new normal logic program for solving the HC problem, and show that the program is tight on every model of its completion. Experimental results showed that for many graphs, this new encoding improves the performance of both SMOLELS and ASSAT(Chaff2), especially of the latter system which is based on the SAT solver Chaff2. We also propose a notion of inherently tight logic programs and show that for any program, it is inherently tight iff all its completion models are stable models. We then propose a polynomial transformation from a logic programs to one that is inherently tight, thus providing a reduction of stable model semantics to program completion semantics and SAT.

1 Introduction

It is well-known that a stable model of a logic program is also a logical model of the completion of the logic program, but the converse is not true in general. However, Fages [1994] showed that if the program is tight (i.e. it has no positive loops), then the converse is also true. Recently, Babovich, Erdem, and Lifschitz [2000] generalized Fages' result, and showed that the converse is also true if the program is tight *on the given model* of the completion. As it turned out, this is quite a general result as many interesting logic programs are indeed tight on every model of their completions. In fact, among the commonly known benchmark domains, only Niemelä's normal logic program encoding of the Hamiltonian Circuit (HC) problem [Niemelä,

1999] is not tight on its completion models. This leads naturally to the question: is there a normal logic program encoding of the HC problem that is tight on every model of its completion? At first glance, a positive answer seems easily followed from some known results. For instance, using any of the known transformation from logic programs to propositional theories (e.g. [Ben-Eliyahu and Dechter, 1996; Lin and Zhao, 2002]), one can translate a program to a set of clauses, and then back to a logic program that is tight (on every model of the program's completion). While this is certainly true, it is not really what we wanted. Given a graph, a logic program for solving the HC problem for the graph normally should have two parts - a set of facts that defines the graph in terms of vertices and edges, and a set of general rules. While the above reasoning shows that it follows from the known results that for any given graph, a tight logic program can be found for solving the HC problem on the graph, it does not follow, although it is true as we shall show in this paper, that there is a set of general rules such that for any given graph, the logic program consisting of the set of general rules and the set of facts encoding the graph is always tight.

By modifying Niemelä's encoding, we propose a new normal logic program for solving the HC problem, and show that the program is tight on every model of its completion. Compared with Niemelä's encoding, our new encoding yields much larger programs - about twice more atoms and rules. Surprisingly, despite its larger size, for both SMOLELS [Simons, 2003] and ASSAT [Lin and Zhao, 2002], the new encoding performs better¹ on many randomly generated graphs. For ASSAT, it also performs better on complete graphs, which are hard for SMOLELS and ASSAT partly because of the sizes of the programs corresponding to these graphs. However, our new encoding is slower than Niemelä's on some of the hand-coded hard graphs in [Lin and Zhao, 2002], which are constructed by taking a few copies of a graph and connecting these copies by some arcs. While these experimental results are mixed, they do suggest that it is worthwhile to try to encode a problem as a logic program whose stable models are the same as its completion models.

This motivated us to investigate a sufficient and necessary

*Corresponding author

¹All performances in this paper are about computing *one* stable model of a program.

condition for a completion model to be a stable model. To this end, we generalize the notion of tight on a set of atoms into that of *inherently tight* on a set of atoms. Specifically, we call a program P *inherently tight* on a set S of atoms if there is a subprogram $Q \subseteq P$ such that S is a stable model of Q and Q is tight on S . We show that for any program, a completion model of this program is also its stable model iff the program is inherently tight on the model.

We then propose a polynomial transformation from any logic program to one that is inherently tight on all its completion models. This provides a reduction of stable model semantics to completion semantics and SAT. Compared to the one in [Ben-Eliyahu and Dechter, 1996] which needs n^2 extra atoms and n^3 new rules, and the one in [Lin and Zhao, 2002] which does not introduce any new atom but in the worst case may add an exponential number of new rules (clauses), our transformation introduces $O(n^2 + m)$ extra atoms and $O(n \times m)$ extra rules in the worst case, where n is the number of atoms in the original program and m the number of rules. We have observed that in all of the benchmark logic programs, the number of rules in a program is smaller than n^2 . One useful feature of our transformation is that it is modular w.r.t adding new facts.

This paper is organized as follows. Section 2 introduces some logical preliminaries. Section 3 studies an encoding of the HC problem that is tight on every completion model of the program, and reports some experimental results using SMOELS and ASSAT(Chaff2). Section 4 proposes a notion of inherent tightness and a translation from a logic program to one that is inherently tight. Section 5 concludes this paper. Due to space limitations, proofs, if given, are only sketched.

2 Logical Preliminaries

Logic programs A normal logic program is a set of rules of the following form:

$$a \leftarrow b_1, \dots, b_m, \text{not } c_1, \dots, \text{not } c_n, \quad (1)$$

where a is either an atom or empty, b 's, and c 's are atoms. When a is empty, then the rule is a constraint, and when $m = n = 0$ and a is an atom, then the rule is a fact, asserting that a is true. Given a rule r , in the following, we denote by $Head(r)$ its head atom, and $Pos(r)$ and $Neg(r)$ the set of positive body literals and the set of atoms that occur under not in the body of rule r , respectively. For any set X of atoms, we denote the set $\{\text{not } L : L \in X\}$ by $not(X)$. So a rule r can be represented as

$$Head(r) \leftarrow Pos(r) \cup not(Neg(r)). \quad (2)$$

Given a program P , in the following, we let $Atom(P)$ be the set of atoms in P .

Program completion Given a logic program P , its completion, $Comp(P)$ is the union of the constraints in P and the Clark completion [Clark,] of P with constraints deleted. In other words, it consists of the following sentences:

- For any atom a , if $a \leftarrow G_1, \dots, a \leftarrow G_n$ are all the rules about a in P , then $a \equiv G_1 \vee \dots \vee G_n$ is in $Comp(P)$. Here we abuse the notation and write G_i both as the

body of a rule and a disjunct in a formula. Its intended meaning is that when we write a body G in a formula, it stands for *true* if G is empty, it is the conjunction of literals in G with *not* replaced by $\neg$ otherwise. Notice here that when $n = 0$, the equivalence for a becomes $a \equiv \text{false}$;

- If $\leftarrow G$ is a constraint in P , then $\neg G$ is in $Comp(P)$.

The stable model semantics Given a logic program P without constraints, and a set S of atoms, the Gelfond-Lifschitz transformation of P on S , written as P_S , is obtained from P as follows:

- For each atom q and any rule r in P , if $q \in Neg(r)$ and $q \notin S$, then delete the literal *not* q from the body of rule r .
- In the resulted set of rules, delete all those rules that still contain a negative literal in their bodies, i.e., for any rule $r \in P$, if there is a $q \in Neg(r)$ such that $q \in S$, then delete this rule r .

It is clear that for any set of atoms S , P_S is a set of rules without any negative literals. Thus P_S has a unique minimal model, which is the same as the set of atoms that can be derived from the program by resolution when rules are interpreted as implications. We denote this set as $Cons(P_S)$. A set S is a *stable model* [Gelfond and Lifschitz, 1988] of P if and only if $S = Cons(P_S)$.

Now let P be a program that may have constraints, and P' the result of deleting all constraints in P . Then a set of atoms is a stable model of P iff it is a stable model of P' and satisfies all the constraints in P .

Dependency graph The *predicate dependency graph* [Apt *et al.*,] of a logic program P is a directed graph with signed edges. The vertices are atoms mentioned in P . There is a directed positive (resp. negative) edge from vertex p to q if there is a rule r in P such that $Head(r) = q$ and $p \in Pos(r)$ (resp. $Head(r) = q$ and $p \in Neg(r)$). Informally, a positive (resp. negative) edge from p to q means that q depends positively (resp. negatively) on p .

For any two atoms α and β in P , there is a positive path from α to β if and only if there is a path from vertex α to vertex β in the dependence graph G_P and the path has no negative edges.

3 Tight logic programs and Hamiltonian Circuit problem

A program P is *tight* [Lifschitz, 1996] if there exists a level mapping λ such that, for every rule

$$a \leftarrow b_1, \dots, b_m, \text{not } c_1, \dots, \text{not } c_n \quad (3)$$

in P ,

$$\lambda(b_1), \dots, \lambda(b_m) < \lambda(a). \quad (4)$$

Here a *level mapping* is a function from literals to ordinals.

It is not hard to see that a program is tight iff its dependency graph has no positive cycles. For these programs, the program completion semantics under classical logic coincides with the stable model semantics. That is, if a program is tight, then a set of atoms is its stable model iff it is a classical propositional

model of its completion [Fages, 1994]. Recently, Babovich, Erdem, & Lifschitz [2000] generalized this result to “tight on a set of literals” by modifying the mapping λ to a *partial level mapping* with respect to a set of literals. Formally, a program P is tight on a set S of atoms if there exists a partial level mapping λ with the domain S such that, for every rule (3) in P , if a and $b_1, \dots, b_k \in S$, then

$$\lambda(b_1), \dots, \lambda(b_k) < \lambda(a). \quad (5)$$

They showed that if a normal logic program is tight on a model of its completion, then this model is also a stable model. As a result, if the logic program is tight or tight on all models of its completion, to find stable models of this logic program, we can reduce it to SAT by computing its completion. As it turned out, this is quite a general result as many interesting logic programs, including all constraint satisfaction problems and planning problems are indeed tight on every model of their completions. In fact, among the commonly known benchmark domains, only Niemelä’s normal logic program encoding of the Hamiltonian Circuit (HC) problem [Niemelä, 1999] is not always tight on its completion models.

As the experimental results reported in [Babovich *et al.*, 2000; Huang *et al.*, 2002; Lin and Zhao, 2002] showed, for problems like graph coloring and blocks world planning whose encodings in logic programs are either tight or tight on their completion models, computing a stable model of these programs using the state of art SAT solvers on their completions is a lot faster than SMOBELS, which is currently the state of art specialized stable model generator for normal logic programs. For HC problems, while ASSAT still outperforms SMOBELS, the improvements are not as great as in other problems. One of the reasons is that when a completion model may not be a stable model, ASSAT needs to repeatedly call a SAT solver, and the total computation time is directly proportional to the number of times a SAT solver is called. So to see if this was due to the inherent difficulty of the HC problem or because of the particularity of Niemelä’s encoding [Niemelä, 1999], we came up with an encoding of the HC problem that is always tight on the completion models.

To motivate our new encoding, let’s first look at Niemelä’s given below:

$$hc(V, U) \leftarrow arc(V, U), \text{not } otherroute(V, U). \quad (6)$$

$$otherroute(V, U) \leftarrow arc(V, U), arc(V, W), \quad (7)$$

$$hc(V, W), U \neq W.$$

$$otherroute(V, U) \leftarrow arc(V, U), arc(W, U), \quad (8)$$

$$hc(W, U), V \neq W.$$

$$reached(U) \leftarrow arc(V, U), hc(V, U), reached(V), \quad (9)$$

$$\text{not } initialnode(V).$$

$$reached(U) \leftarrow arc(V, U), hc(V, U), initialnode(V). \quad (10)$$

$$\leftarrow vertex(V), \text{not } reached(V). \quad (11)$$

As one can see, rules (6) - (8) guess some paths with the property that for every vertex X , if X has at least one outgoing arcs, then exactly one of these arcs is chosen to be in the paths, and similarly if X has at least one incoming arcs, then exactly one is chosen to be in the paths. The

rest of the rules then make sure, especially by asserting that $reached(X)$ must be true for every vertex X , that these paths are in fact a single simple cycle that covers the whole graph. When grounded, this program is not tight because rule (9) would create cycles among $reached(X)$ atoms.

Another way of solving the problem is to first guess some paths that must go through every vertex exactly once, and then make sure that when we do not count the outgoing arc from the starting vertex, these paths are in fact a single path that does not contain any cycles. This is the idea behind our program in Figure 1 for solving the HC problem:

$$outgoing(V) \leftarrow arc(V, U), hc(V, U). \quad (12)$$

$$incoming(U) \leftarrow arc(V, U), hc(V, U). \quad (13)$$

$$\leftarrow vertex(V), \text{not } outgoing(V). \quad (14)$$

$$\leftarrow vertex(V), \text{not } incoming(V). \quad (15)$$

$$hc(V, U) \leftarrow arc(V, U), \text{not } otherroute(V, U). \quad (16)$$

$$otherroute(V, U) \leftarrow arc(V, U), arc(V, W), \quad (17)$$

$$hc(V, W), U \neq W.$$

$$otherroute(V, U) \leftarrow arc(V, U), arc(W, U), \quad (18)$$

$$hc(W, U), V \neq W.$$

$$reached(V, U) \leftarrow arc(V, U), hc(V, U), \quad (19)$$

$$\text{not } initialnode(V).$$

$$reached(V, U) \leftarrow arc(W, U), hc(W, U), vertex(V), \quad (20)$$

$$\text{not } initialnode(W), reached(V, W).$$

$$\leftarrow vertex(V), reached(V, V). \quad (21)$$

Figure 1: An encoding of the HC problem that is always tight on every model of its completion

Notice that rules (16) - (18) are the same as (6) - (8); rules (12) - (15) are needed to make sure that the chosen path must go through each vertex exactly once for the cases when a vertex has no outgoing (incoming) arcs in the graph; rules (19) - (21) use the binary predicate $reached(X, Y)$ to make sure that when we delete the chosen outgoing arc from the starting vertex, the chosen path has no cycles, thus must be a single path. Formally, it can be shown that the program correctly solves the HC problem and is always tight on every completion model.

Proposition 1 *Let $G = (V, E)$ be a graph. Let P be the union of the rules in Figure 1 and facts represented by the graph in terms of $vertex(X)$ and $arc(X, Y)$. If M is a stable model of P , then atoms of the form $hc(u, v)$ in M correspond to an HC of G . Furthermore, P is tight on every model of its completion.*

Given a graph, compared with Niemelä’s encoding, ours yields a larger program because $reached$ in our program is a binary predicate. To evaluate the performance of this program, we tried SMOBELS (version 2.27) and ASSAT(Chaff2) (version 1.3 with its simplification feature activated) on the same set of graphs² used in [Lin and Zhao,

²These graphs can be downloaded from ASSAT’s web site <http://www.cs.ust.hk/assat>.

2002], which divided it into three categories: randomly generated graphs, hand-coded hard instances, and complete graphs. Our experiments were done on Sun Ultra 5 machines with 256M memory running Solaris. The times given below are in CPU seconds as reported by Unix “time” command, and include the times for running lparse for grounding the input programs. For randomly generated graphs (see Table 1), ASSAT clearly performs much better using our new encoding; SMOBELS also seemed to behave better using the new encoding as the run times are more uniform now. However, for hand-coded graphs (see Table 2), which are formed by first taking several copies of a small graph and then selectively adding some arcs to connect these components, the old encoding was clearly better for SMOBELS. For ASSAT, some of the graphs became easier with the new encoding, but many became harder. Finally, for complete graphs, which are hard partly because they yield huge programs both for the new and old encodings, the new encoding turned out to be better for ASSAT but a little worse for SMOBELS. For instance, for the complete graph with 80 vertices, using the old encoding, ASSAT returned with a solution in 3297 seconds, and SMOBELS in 5072 seconds, but with the new encoding, ASSAT took only 170 seconds, but SMOBELS needed 12485 seconds.

From these mixed results, it seems that both encodings have merits, especially if one uses SMOBELS. But the experiments do suggest that the new encoding, being tight on every completion model, is better for SAT-based stable model generators such as ASSAT and Cmodels³. We would not be surprised that with better and faster SAT solvers, the new encoding would outperform in all cases.

4 Inherently tight programs

Given the potential benefit of tight logic programs for SAT-based stable model generators, we now investigate a more general translation from an arbitrary logic program to one whose completion models are always stable models. To this end, we first define a notion of *inherently tight on a set of atoms*, and show that it is a sufficient and necessary condition for a program’s completion model to be a stable model.

Definition 1 A program P is *inherently tight on a set of atoms S* if there is a program Q such that $Q \subseteq P$, S is a stable model of Q and Q is tight on S .

Proposition 2 A normal logic program is *inherently tight on all of its stable models*.

Proof: Suppose S is a stable model of P . Notice first that each rule in P_S has a corresponding one in P under Gelfond-Lifschitz transformation. Now define W , a set of rules, as follows:

1. W is initially empty, and T initially P_S ;

³We also ran experiments for Cmodels (<http://www.cs.utexas.edu/users/tag/cmodels.html>). The results were very similar to the ones for ASSAT. We did not include its data here because it is not guaranteed to work when the input program is not tight, like Niemelä’s encoding of the HC problem here.

Problem	SMODELS		ASSAT(Chaff2)	
	old	new	old	new
nv50a380	1.1	30.2	7.1	3.3
nv50a400	1.2	33.5	29.2	3.4
nv50a420	>2h	24.3	20.5	3.5
nv50a440	1.2	29.9	1.9	3.8
nv50a460	>2h	33.2	13.5	4.1
nv50a480	1.8	35.1	18	4.2
nv50a500	2.0	34.9	40.3	4.2
nv50a520	2.1	41.0	1.3	4.6
nv50a540	2.4	41.2	2.3	4.7
nv50a560	2.8	48.3	3.6	4.9
nv50a580	2.9	45.2	78.6	5
nv60a320	>2h	35.1	0.5	3.4
nv60a360	>2h	33.0	10.3	3.7
nv60a420	1.5	63.1	169.4	4.4
nv60a440	22.5	61.5	41.1	4.8
nv60a460	1.5	70.1	203.9	4.6
nv60a480	1.4	63.6	42	4.7
nv60a500	1.9	73.1	1.2	5.2
nv60a520	>2h	73.1	4.4	4
nv60a540	2.1	79.8	3664.8	5.6
nv60a560	1779.6	86.0	3.8	5.8
nv60a580	2.3	89.5	9.7	6.2
nv70a300	0.4	37.5	20.2	3.4
nv70a320	0.6	40.3	26.6	4
nv70a340	0.8	56.1	103.4	4.3
nv70a360	0.9	69.2	3.7	4.4
nv70a380	1.0	76.4	32.1	4.6
nv70a400	1.3	63.8	5.5	5.1
nv70a420	1.4	73.9	12.7	5.3
nv70a440	265.8	77.9	20.3	5.5
nv70a460	>2h	104	19.6	5.8
nv70a480	1.7	102.4	102.7	6.2
nv70a500	14.3	115.2	0.8	6.2
nv70a520	>2h	97.2	232.1	6.5
nv70a540	2.2	110.5	18.8	7
nv70a560	2.2	456.9	87.9	7.1
nv70a580	>2h	146.1	211.4	7.2

Table 1: Randomly generated graphs: nvXaY is a graph with X vertices and Y arcs.

Problem	SMODELS		ASSAT(Chaff2)	
	old	new	old	new
2xp30	0.1	>2h	0	>2h
2xp30.2	0.9	>2h	205.5	19.3
2xp30.2	>2h	>2h	155.9	12.9
2xp30.3	>2h	>2h	159.8	12.8
2xp30.4	>2h	>2h	3824.1	>2h
4xp20	0.1	>2h	0	38.4
4xp20.1	>2h	>2h	13.1	101.9
4xp20.2	1.3	>2h	8.2	8.6
4xp20.3	0.1	>2h	4.8	107.2

Table 2: Hand-coded hard graphs: NxG.M is a graph with N copies of graph G and some arcs connecting the copies.

2. Choose a fact $\alpha \leftarrow$ in T . Suppose r is the corresponding rule in P . Now let W be $W \cup r$, and remove all rules about α from T , and delete α in the bodies of all other rules in T ;
3. Go back to Step 2 until T is empty.

It is clear that W is tight and S is one of its stable models. Thus P is inherently tight on S . ■

Theorem 1 *Let P be a normal logic program, and S a set of atoms. Then S is a stable model of P if and only if S is a model of the completion of P and P is inherently tight on S .*

Proof: $\Rightarrow$: By Proposition 2 and the fact that a stable model is also a model of a program.

$\Leftarrow$: Since S be a model of the completion of P , so S is closed under P . Since P is inherently tight on S , there is a $Q \subseteq P$ such that S is a stable model of Q and Q is tight on S . Thus S is supported by Q , hence also supported by P . Now S is supported by and closed under P , thus a stable model of P (by a theorem in [Babovich *et al.*, 2000]). ■

We say that a program is *inherently tight* if it is inherently tight on all models of its completion. Clearly, a program that is tight on all its completion models is inherently tight. By Theorem 1, if all models of its completion are also its stable model, then the program is inherently tight, and vice versa.

We now give a translation from an arbitrary logic program to one that is inherently tight. In the following, a set S of atoms in a program P is called a *strongly connected component* (SCC) if there is a path in G_P from u to v for any $u, v \in S$, and S is not a subset of any other such set, where G_P is the dependency graph of P .

Algorithm 1 *Let P be a normal logic program. the following algorithm transforms it to an inherently tight program Q . Let Q be empty initially.*

1. For any rule $r \in P$ of the form

$$a \leftarrow b_1, \dots, b_n, d_1, \dots, d_m, \text{not } c_1, \dots, \text{not } c_s \quad (22)$$

b_i ($1 \leq i \leq n$) and atom a are in the same SCC while d_j ($1 \leq j \leq m$) and a are not. We add the following rules into Q :

$$a_r \leftarrow \text{not } \text{right}(a, b_1), \dots, \text{not } \text{right}(a, b_n), \quad (23)$$

$$b_1, \dots, b_n, d_1, \dots, d_m, \text{not } c_1, \dots, \text{not } c_s \quad (24)$$

$$\text{right}(b_i, a) \leftarrow a_r \quad (25)$$

$$\text{right}(X, a) \leftarrow a_r, \text{right}(X, b_i) \quad (26)$$

In these rules, $1 \leq i \leq n$ and variable X will be instantiated to all atoms α if α and a are in the same SCC;

2. If $r \in P$ is a constraint, add rule r it into Q .

Intuitively, if a set M is a stable model of Q , then $a_r \in M$ iff $a \in M$ and the rule r in P ‘contributes’ to the calculation of $\text{Cons}(Q_M)$, meaning that rule r is actually used to derive a . $\text{right}(u, v)$ is used to record that atom u is used to prove v and this is needed to prevent loops. The following example illustrates the idea.

Example 1 *Consider the program $P = \{a \leftarrow b, b \leftarrow a, a \leftarrow \text{not } c, c \leftarrow c\}$. It has two completion models $\{a, b\}$ and $\{c\}$, but only the first one is a stable model. For this program, there are two SCCs: $\{a, b\}$ and $\{c\}$. Algorithm 1 transforms it into the following program Q :*

$$\begin{aligned} a_1 &\leftarrow \text{not } \text{right}(a, b), b. \\ a &\leftarrow a_1. \\ \text{right}(b, a) &\leftarrow a_1. \\ \text{right}(a, a) &\leftarrow a_1, \text{right}(a, b). \\ \text{right}(b, a) &\leftarrow a_1, \text{right}(b, b). \\ a_2 &\leftarrow \text{not } c. \\ a &\leftarrow a_2. \\ b_1 &\leftarrow \text{not } \text{right}(b, a), a. \\ b &\leftarrow b_1. \\ \text{right}(a, b) &\leftarrow b_1. \\ \text{right}(a, b) &\leftarrow b_1, \text{right}(a, a). \\ \text{right}(b, b) &\leftarrow b_1, \text{right}(b, a). \\ c_1 &\leftarrow \text{not } \text{right}(c, c), c. \\ c &\leftarrow c_1. \\ \text{right}(c, c) &\leftarrow c_1. \\ \text{right}(c, c) &\leftarrow c_1, \text{right}(c, c). \end{aligned}$$

The completion of the program contains the following equivalences $c \equiv c_1$, $c_1 \equiv \neg \text{right}(c, c) \wedge c$, and $\text{right}(c, c) \equiv c_1$. Thus it entails $\neg c \wedge \neg c_1 \wedge \neg \text{right}(c, c)$. So there is only one model of the completion where a and b are true and c is false, which is also the only stable model of Q . Notice that in the stable model of P , a is established by the second rule about it. Correspondingly, in the stable model of Q , a_2 is true while a_1 is not, and $\text{right}(a, b)$ is true while $\text{right}(b, a)$ is not (a is established first, and then b is derived from a).

Theorem 2 *For any normal logic program P , the program Q output by Algorithm 1 is inherently tight. Furthermore P and Q are equivalent on $\text{Atom}(P)$, that is, for any subset S of $\text{Atom}(P)$, S is a stable model of P iff there is exactly one stable model S' of Q such that $S = S' \cap \text{Atom}(P)$.*

Proof: We show only that Q is inherently tight. The second part of the theorem is easier.

We only need to prove that all models of $\text{Comp}(Q)$ are its stable models. Let N be a model of the completion of Q . We show that N satisfies every loop formula of Q , thus it must be a stable model of Q (Theorem 1 in [Lin and Zhao, 2002]). Suppose L is a positive loop in Q . According to Algorithm 1, either L is a loop that does not contain any atom of the form $\text{right}(u, v)$ or L is a set of atoms of the form $\text{right}(u, v)$.

If L is a loop that does not contain any atom of the form $\text{right}(u, v)$, suppose $v_1, \dots, v_s$ are all atoms in L . L has even number of atoms. Atoms in $\text{Atom}(P)$ and newly added atoms occur along the loop alternatively. Suppose the loop is: $x'_1, x_1, x'_2, x_2, \dots, x'_t, x_t$ where x'_i are newly added atoms and x_i are atoms in $\text{Atom}(P)$ ($1 \leq i \leq t$). If all v_j ($1 \leq j \leq s$) are in N , it is known that $\text{right}(x_i, x_{i+1})$ are in N by rule 25. Further considering rule 26, we have $\text{right}(x_i, x_{i-1}) \in N$ for some i , thus rule 23 can not be satisfied for some x'_i

in the loop. So not all atoms in the loop are in N . Now that N is a model of the completion, so the loop formula [Lin and Zhao, 2002] for loop L is satisfied.

If L is a loop and it contains a set of atoms $right(a_1, a_2), right(a_2, a_3), \dots, right(a_v, a_1)$. Suppose $right(a, b)$ is an atom in L and it is deduced by the rule $right(a, b) \leftarrow b_i, right(a, c)$. Suppose there is a rule r such that $Pos(r) \cap L = \emptyset$, if $Body(r)$ is satisfied by N , the loop formula of L is satisfied. If there is no such a r , then the rule $right(a, b) \leftarrow b_i, right(a, c)$ must be satisfied. Thus $b_i \in N$. But there is only one rule about b_i which requires $right(\alpha, \beta) \notin N$ for some $\alpha, \beta \in L$ for it to be applicable. However from our assumption that $L \subseteq N$, and L is a loop, it must be the case that $right(\alpha, \beta) \in N$ for all $\alpha, \beta \in L$. This is a contradiction, thus there must be an r as above. ■

Algorithm 1 effectively provides yet another reduction of answer set semantics to SAT by calculating the completion of the translated program. Compared with the mapping in [Ben-Eliyahu and Dechter, 1996], which always adds n^2 extra atoms and n^3 extra rules, the number of extra atoms and rules needed for our transformation depends on the number of rules whose heads and some of positive literals in their bodies belong to the same SCC. In the worst case, it needs $O(n^2 + m)$ extra atoms and $O(n \times m)$ extra rules, where n is the number of atoms and m the number of rules in the input program. Usually, m is a lot less than n^2 . For instance, for Niemelä's encoding of the HC problem, given a graph with k vertices, there are at most $O(k^2)$ atoms and $O(k^3)$ rules.

One interesting feature of Algorithm 1 is that it is modular w.r.t. new facts. That is, if P' is a set of facts, then for any program P , the program returned by the algorithm on $P \cup P'$ is the union of the programs returned by it on P and P' . This feature will be handy when we extend the algorithm to programs with variables as the instantiation of a program with variables can be thought of as the union of the program and a set of ground facts.

We have implemented the algorithm and tried it on Niemelä's encoding of the HC problem, and found that for SMODELs, the translated program is still better than the original one on randomly generated graphs. But on other types of graphs, the translated program is a lot worse. But for ASSAT, the translated program was slower even on randomly generated programs. Apparently, being (inherently) tight does not always guarantee better performance even for SAT-based stable model generators.

5 Conclusions and future work

We have proposed a new solution to the Hamiltonian Circuit problem. For any given graph, the solution yields a program that is tight on every model of its completion. Compared with Niemelä's encoding of the same problem, our new encoding performs better on both randomly generated graphs and complete graphs, but is slightly worse on a set of hand-coded graphs. While these results hold for both SMODELs, which is a specialized stable model generator, and ASSAT, which makes use of SAT solver Chaff2, the performance gain seems greater for ASSAT than for SMODELs.

We also defined a notion of inherent tightness that captures a sufficient and necessary condition for a program completion model to be its stable model, and proposed an algorithm for translating an arbitrary logic program into an inherently tight one, which provides a reduction of stable model semantics to completion semantics and SAT.

In terms of future work, we are interested in finding out more about the relationships between stable model semantics and completion semantics, especially the kinds of features of an inherently tight logic program that would be particularly good for SAT-based stable model generators like ASSAT.

Acknowledgements

We thank Yuting Zhao for useful discussions related to the topics of this paper, especially for his helps in using ASSAT, and for providing some experimental data on ASSAT.

This work was supported in part by the Research Grants Council of Hong Kong under Competitive Earmarked Research Grant HKUST6205/02E.

References

- [Apt *et al.*,] K. R. Apt, H. A. Blair, and A. Walker. In Jack Minker, editor, *Foundations of deductive databases and logic programming*, pages 89–148.
- [Babovich *et al.*, 2000] Y. Babovich, E. Erdem, and V. Lifschitz. Fages' theorem and answer set programming. In *NMR-2000*, 2000.
- [Ben-Eliyahu and Dechter, 1996] R. Ben-Eliyahu and R. Dechter. Propositional semantics for disjunctive logic programs. *Annals of Mathematics and Artificial Intelligence*, 12:53–87, 1996.
- [Clark,] K. L. Clark. In H. Gallaire and J. Minker, editors, *Logics and Databases*, pages 293–322.
- [Fages, 1994] F. Fages. Consistency of clark's completion and existence of stable models. In *Journal of Methods of Logic in Computer Science*, volume 1, pages 51–60, 1994.
- [Gelfond and Lifschitz, 1988] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *ICLP'88*, August 1988.
- [Huang *et al.*, 2002] G.-S. Huang, X. Jia, C.-J. Liao, and J.-H. You. Two-literal logic programs and satisfiability representation of stable models: A comparison. In *Proc. 15th Canadian Conference on AI, LNCS, Springer*, 2002.
- [Lifschitz, 1996] V. Lifschitz. Foundations of logic programming. In *Principles of Knowledge Representation*, 1996.
- [Lin and Zhao, 2002] F. Lin and Y. Zhao. ASSAT: Computing answer sets of a logic program by sat solvers. In *AAAI'02*, 2002.
- [Niemelä, 1999] I. Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25(3-4):241–273, 1999.
- [Simons, 2003] P. Simons. *Smodels: a system for computing the stable models of logic programs, version 2.27*, 2003.

On the Equivalence between Answer Sets and Models of Completion for Nested Logic Programs

Jia-Huai You and Li-Yan Yuan
Department of Computing Science
University of Alberta, Canada

Mingyi Zhang
Guizhou Academy of Sciences
P.R. China

Abstract

We present a sufficient as well as a necessary condition for the equivalence between answer sets and models of completion for logic programs with nested expressions in the bodies of rules. This condition is the weakest among all that we are aware of even for normal logic programs. To obtain this result, we present a polynomial time reduction from this class of nested logic programs to extended programs. Consequently, answer sets for these nested programs can be computed by an answer set generator for extended programs on the one hand, and characterized in terms of models of completion on the other.

1 Introduction

We consider logic programs with nested expressions in the bodies of rules, called *nested logic programs* in this paper. Nested expressions are formulas built from literals (atoms with or without classic negation), connectives such as negation as failure *not*, disjunction “;”, and conjunction “&”. This is a proper subset of the class of programs considered in [Lifschitz *et al.*, 1999], where the head of a rule can also be an arbitrary formula. Lloyd and Topor [Lloyd and Topor, 1984] defined nested logic programs without negation as failure, and argued for the higher expressive power of the extended language, while Lifschitz *et al.* further showed the role of negation as failure in nested logic programs.

Our goal in this paper is to provide a characterization of answer sets for nested logic programs in terms of *tightness* on the one hand, and to use an answer set generator, such as Smodels or DLV to compute answer sets for nested logic programs on the other. The studies on characterizations of answer sets have proved useful in relating the completion semantics [Clark, 1978] with the answer set semantics [Gelfond and Lifschitz, 1988]. Fages [Fages, 1994] defined a syntactic condition on logic programs, called *positive-order-consistent*, which guarantees the equivalence of the two semantics. Positive-order-consistent requires a program to have no positive loops. For example, the program $\{a \leftarrow a\}$ is not positive-order-consistent. Babovich *et al.* [Babovich *et al.*, 2000] on the other hand formalized the notion, *tight on a set*, which says that an extended logic program Π is *tight on X* if

there exists a level mapping λ from X to ordinals, such that for any rule

$$h \leftarrow l_1, \dots, l_m, \text{not } l_{m+1}, \dots, \text{not } l_n$$

in Π , if $\{h\} \cup \{l_1, \dots, l_m\} \subseteq X$ then, for each $i \in [1..m]$,

$$\lambda(h) > \lambda(l_i).$$

Erdem and Lifschitz [Erdem and Lifschitz, to appear] extended the work to nested logic programs, and weakened the condition so that the positive literals are required to be ordered by λ only for applicable rules w.r.t. X .

As shown in [Babovich *et al.*, 2000; Erdem and Lifschitz, to appear], that a program is tight on S is a sufficient condition for the equivalence of S being an answer set and S satisfying the completion of the program. However, there are simple programs that are not tight even on their answer sets. For example, suppose we want to express that two or more propositions are equivalent, and they are true if certain conditions are satisfied. For instance, consider

$$\Pi = \{a \leftarrow b. \ b \leftarrow a. \ b \leftarrow \text{not } c.\}$$

Clearly, Π is not tight on its unique answer set $\{a, b\}$. Any level mapping λ that maps the domain $\{a, b\}$ to non-negative integers will have to satisfy $\lambda(a) > \lambda(b)$, as well as $\lambda(b) > \lambda(a)$, which is impossible.

Even for logic programs without negation as failure, tightness may fail to apply, e.g., with $\Pi = \{a \leftarrow . \ a \leftarrow a.\}$, Π is not tight on its unique answer set $\{a\}$.

In this paper, we define the notion of *weak tightness*, which is a sufficient as well as a necessary condition for a model of completion to be an answer set. Consequently, an answer set is characterized by three independent properties: *weak tightness*, *supportedness*, and *closedness*. This characterization of answer sets can be generalized to nested programs via a linear time transformation to extended programs. This yields a new characterization of answer sets for nested programs. Although the transformation uses a linear number of extra variables, we will see that these extra variables are *non-split*, in the sense that in a Davis-Putnam procedure [Davis and Putnam, 1960], they need not create choice points during search for answer sets, and thus they need not enlarge the search space.

Our results have some practical implications. First, they widen the application range of systems like Cmodels

[Babovich, 2002], a system implemented for the computation of answer sets for tight logic programs. Secondly, the transformation from nested programs to extended programs allows answer sets for nested programs to be computed by an answer set generator for extended programs, or indirectly via the models of their completed programs.

The next section introduces logic programs with nested expressions. In Section 3 we define weak tightness and show that answer sets are characterized by this condition along with models of completion. In Section 4 we present a polynomial time transformation from nested programs to extended programs. Section 5 remarks on related work.

2 Nested Logic Programs

We consider a propositional language L . A *literal* is an atom, or an atom with the sign $\neg$ in front, called the *classical negation*. *Elementary formulas* are literals and constants $\top$ (“true”) and $\perp$ (“false”). Formulas are built from elementary formulas using the unary connective *not* and the binary connectives $\wedge$ (conjunction) and $\vee$ (disjunction). A nested logic program (or, just program if not otherwise specified) is a set of rules of the form

$$H \leftarrow G$$

where H is a literal or $\perp$, and G is a formula. H is called the *head* of the rule and G the *body*. If $G = \top$, we will drop it from the body; rules with the body $\top$ are called *facts*.

Here are some special cases. A formula of the form *not* ϕ , where ϕ is a literal, is called a *default negation*. When G is a conjunction of literals and default negations, the rule is called an *extended rule*. In an extended rule, the literals that do not appear in the scope of *not* are called *positive literals*. The programs that consist of only extended rules are called *extended programs*; and extended logic programs without classical negation are called *normal programs*.

An extended rule may be denoted by $H \leftarrow B^+, B^-$, where B^+ is the set of positive literals and B^- the set of default negations.

We denote by $\text{lit}(\Phi)$ the set of literals occurring in Φ , where Φ is any syntactic entity, such as a formula or a program. Let X be a set of literals. $X|_V$ denotes the subset of X , restricted to the literals in V , i.e., $X|_V = X \cap V$. A set of elementary formulas X is *consistent* if $\perp \notin X$ and it contains no complementary literals l and $\neg l$.

That a consistent set of literals X satisfies a formula F , denoted $X \models F$, is defined recursively as follows:

- for elementary formula F , $X \models F$ if $F \in X$ or $F = \top$
- $X \models \text{not } F$ if $X \not\models F$
- $X \models (F, G)$ if $X \models F$ and $X \models G$
- $X \models (F; G)$ if $X \models F$ or $X \models G$

A consistent set X of literals is *closed under* a program Π if, for every rule $H \leftarrow G$ in Π , $H \in X$ whenever $X \models G$. X is *supported by* Π if, for any $l \in X$ there is a rule $l \leftarrow G$ such that $X \models G$.

We now define answer sets for a program. For a program Π without negation as failure, a consistent set of literals X is an answer set for Π if and only if X is *minimal* among the consistent sets of literals closed under Π .

A consistent set of literals X is an answer set of program Π if and only if X is *minimal* among the consistent sets of literals closed under Π^X , where the *reduct* Π^X is obtained from Π by replacing every *maximal* occurrence of a formula *not* F by $\perp$ if $X \models F$ and $\top$ otherwise.

We define the completion semantics for finite nested programs without the classic negation. Let Π be such a program. The *completion* of Π , denoted $\text{Comp}(\Pi)$, is the following set of propositional formulas: for each atom ϕ which is either an atom in L , or $\perp$,

- if ϕ does not appear as the head of any rule in Π , then $\phi \equiv \perp \in \text{Comp}(\Pi)$;
- otherwise, $\phi \equiv \text{Body}_1 \vee \dots \vee \text{Body}_n \in \text{Comp}(\Pi)$ (with each default negation *not* ξ replaced by negative literal $\neg \xi$), if there are exactly n rules $\phi \leftarrow \text{Body}_i \in \Pi$ with ϕ as the head.

Finally, given a nested program Π without negation as failure, to construct the unique minimal set among the consistent sets of literals closed under Π , we define the familiar T_Π operator as

$$T_\Pi(S) = \{l \mid l \leftarrow G \in \Pi \text{ and } S \models G\}$$

It is clear that T_Π is monotonic over the domain of sets of literals. The least fixpoint can then be computed by

$$\begin{aligned} T_\Pi \uparrow 0 &= \emptyset \\ T_\Pi \uparrow i + 1 &= T_\Pi(T_\Pi \uparrow i), \quad \text{for all } i \geq 0 \\ T_\Pi \uparrow \infty &= \bigcup_{i=0}^{\infty} T_\Pi \uparrow i \end{aligned}$$

The following proposition is needed later in this paper.

Proposition 2.1 *Let Π be a program without negation as failure. Then, X is an answer set for Π iff X is consistent and $X = T_\Pi \uparrow \infty$.*

3 Weakly Tight Logic Programs

Let Π be an extended program, X a set of literals, and h a literal in X . We define the set of *applicable rules* of Π , with respect to h and X , as follows;

$$\Pi(h, X) = \{h \leftarrow \text{Body}^+, \text{Body}^- \in \Pi \mid h \in X, \text{Body}^+ \subseteq X, \text{ and } \forall \text{not } \phi \in \text{Body}^-, \phi \notin X\}$$

A *level mapping* λ is a partial function from literals to natural numbers such that the set $\mathcal{N}$ of natural numbers mapped from literals forms a partial order $\langle \mathcal{N}, > \rangle$.¹

Definition 3.1 *Let X be a set of literals and Π an extended program. Π is said to be weakly tight on X if there exists a level mapping λ with the domain X such that for any nonempty $\Pi(h, X)$ where $h \in X$, there is a rule $h \leftarrow \text{Body}^+, \text{Body}^-$ in $\Pi(h, X)$ such that for each $b \in \text{Body}^+$, $\lambda(h) > \lambda(b)$.*

Example 3.2 *Let program $\Pi = \{a \leftarrow b, b \leftarrow a, b \leftarrow c, \text{not } d, c \leftarrow \cdot\}$. Then, $S = \{a, b, c\}$ is the only answer set for Π . Though Π is not tight on S , Π is weakly tight on S . For example, a level mapping λ satisfying the weak tightness condition can be: $\lambda(a) = 3, \lambda(b) = 2, \lambda(c) = 1$.*

¹By a partial order, we mean a relation that is transitive and irreflexive.

Lemma 3.3 *Let Π be an extended program, and S an answer set for Π . Then Π is weakly tight on S .*

Proof. Since S is an answer set for Π , $S = T_{\Pi^S} \uparrow \infty$. It can be seen that the following mapping λ exists: for each literal q such that $q \in T_{\Pi^S} \uparrow n$ but $q \notin T_{\Pi^S} \uparrow n - 1$, let $\lambda(q) = n$. Clearly, λ satisfies the condition for weak tightness. $\square$

Theorem 3.4 *For any extended program Π and any consistent set of literals S , S is an answer set for Π iff Π is weakly tight on S , and S is closed under and supported by Π .*

Proof. ($\Rightarrow$) Let S be an answer set for Π . Then $T_{\Pi^S} \uparrow \infty = S$. It's obvious that S is closed under and supported by Π . From Lemma 3.3 we know that Π is also weakly tight on S .

($\Leftarrow$) Assume Π is weakly tight on S , and S is closed under and supported by Π . We show that S is an answer set for Π . Let $M = T_{\Pi^S} \uparrow \infty$. The condition that S is closed implies $M \subseteq S$. We are done if $M = S$. Otherwise we have $M \subset S$. Let $q \in S \setminus M$. Since S is supported by Π , there is a rule $r : q \leftarrow \text{Body}^+, \text{Body}^-$ in Π such that $q \leftarrow \text{Body}^+ \in \Pi^S$ and $\text{Body}^+ \subseteq S$. Thus, the same rule r is in $\Pi(q, S)$, which is nonempty. As Π is weakly tight on S , there is a mapping λ and a rule $q \leftarrow \text{Body}'^+, \text{Body}'^-$ in $\Pi(q, S)$ such that $q \leftarrow \text{Body}'^+ \in \Pi^S$ and $\lambda(q) > \lambda(b)$ for each $b \in \text{Body}'^+$. Since M is also closed, there is $b \in \text{Body}'^+$ such that $b \in S \setminus M$. Then, from the definition of λ , there is a literal $p \in S \setminus M$ such that $\lambda(p)$ is minimal, hence $p \in T_{\Pi^S} \uparrow 1$. This contradicts the assumption that M is the least fixpoint of the operator T_{Π^S} . $\square$

Note that neither supportedness nor closedness can be removed from the theorem for it still to hold. E.g. the program $\Pi = \{a \leftarrow b\}$ is tight as well as weakly tight on $\{a, b\}$, which is not supported by Π . For closedness, we note that any program is tight as well as weakly tight on the empty set.

Corollary 3.5 *Let Π be a finite normal program and S a set of atoms satisfying $\text{Comp}(\Pi)$. Then, Π is weakly tight on S iff S is an answer set for Π .*

To compare two conditions that guarantee a model of completion to be an answer set, we have the following definition.

Definition 3.6 *Let $\text{Cond1}(X, Y)$ and $\text{Cond2}(X, Y)$ be conditions where X ranges over the set of all extended programs and Y over the set of all sets of literals. $\text{Cond1}(X, Y)$ is said to be weaker than $\text{Cond2}(X, Y)$ if*

- *for all X and Y , $\text{Cond2}(X, Y)$ implies $\text{Cond1}(X, Y)$, and*
- *there exist some X and Y such that $\text{Cond1}(X, Y)$ does not imply $\text{Cond2}(X, Y)$.*

Proposition 3.7 *For any extended program Π and set of literals X , if Π is tight on X then it is weakly tight on X . But the converse is not true.*

In Section 5, we will see that weak tightness is also weaker than other conditions used to characterize answer sets in the literature. Then, the question arises as whether the weak tightness condition is the weakest. Technically, the answer is *no*, since the weak tightness condition can be made weaker in trivial ways, e.g., one can make it weaker by not requiring

a level mapping to be applied to a rule with a body literal that does not appear in the head of any rule.

4 Transformation

When a program Π is translated to another program Π' , new propositional symbols may be used. Thus, the language for Π' is enlarged. Below, we define a notion of equivalence between a program and its translation. Throughout this section, a program means a finite nested program.

Definition 4.1 *Let Π be a program in language L . Suppose the language L' for Π' is a superset of L . Π' is equivalent to Π , denoted $\Pi \sqsubseteq \Pi'$, if and only if both of the following statements hold.*

- *For any answer set S for Π , there is exactly one answer set S' for Π' such that $S'_L = S$.*
- *For any answer set S' for Π' , S'_L is an answer set for Π .*

Lemma 4.2 *The relation $\sqsubseteq$ is reflexive and transitive.*

Our transformation makes use of some transformations in [Lifschitz *et al.*, 1999] that preserve strong equivalence. Two programs Π_1 and Π_2 are said to be *strongly equivalent* if for any program P , $\Pi_1 \cup P$ and $\Pi_2 \cup P$ have the same set of answer sets. Strong equivalence entails our notion of equivalence.

Proposition 4.3 *If two programs Π_1 and Π_2 are strongly equivalent, then they are equivalent to each other; i.e., $\Pi_1 \sqsubseteq \Pi_2$ and $\Pi_2 \sqsubseteq \Pi_1$.*

4.1 Transformation

In the following, we say that a formula is *basic* if it is built from “,” “,” literals, constants, and *not* L and *not not* L where L is a literal or constant. A program is said to be *basic* if it consists of rules whose body formulas are basic. A rule

$$H \leftarrow G_1; \dots; G_n.$$

is said to be *flat* if every G_i is a conjunction of literals and default negations. A program is *flat* if every rule in it is flat.

Let Π be a nested program. Our transformation consists of three steps, briefly described below.

- Π is transformed to a basic program Π' by some simplification rules.
- Π' is transformed to a flat program Π'' by transformation rules that name some subexpressions by new atoms.
- Π'' is then transformed to Π''' by splitting each flat rule with disjunctive body into several extended rules with the same head.

We now give the details. Below, $F \Rightarrow R$ denotes a rewrite rule.

Step 1. Simplification

1. $\text{not } (F; G) \Rightarrow \text{not } F; \text{not } G$
2. $\text{not } (F, G) \Rightarrow \text{not } F; \text{not } G$
3. $\text{not not not } F \Rightarrow \text{not } F$

4. $F, \top \Rightarrow F$ (and its symmetric case, $\top, F \Rightarrow F$; the same for all rewrite rules below)
5. $F; \top \Rightarrow \top$
6. $F, \perp \Rightarrow \perp$
7. $F; \perp \Rightarrow F$
8. $p, \neg p \Rightarrow \perp$
9. $\text{not } p; \text{not } \neg p \Rightarrow \top$

where Rule 3 has higher priority to be applied than Rules 1 and 2. $\square$

The simplification rules given here are essentially their corresponding equivalences given in [Lifschitz *et al.*, 1999]. The only difference is that, for the purpose of termination, we cannot directly adopt an equivalence for symmetry, such as $F, G \Leftrightarrow G, F$, as a rewrite rule. We therefore have to represent the symmetric cases by rules.

The simplification rules are similar to the ones found in propositional logic, except perhaps Rule 3, which simplifies negation as failure only when it is nested at least three levels deep. Among these rules, only 1-3 are essential for the purpose of producing a basic program. The rest are used to simplify basic formulas further.

That a rule has a higher priority to be applied than another rule means that when both rules are applicable, the former should be applied and the latter should not. The only intention here for Rule 3 over Rules 1 and 2 is to prevent transforming an expression of the form

$$\underbrace{\text{not} \dots \text{not}}_n F$$

where $n > 2$, by distributing each *not* over F (which would result in higher complexity).

Step 2. Naming subexpressions

- (a) $F, (G; H) \Rightarrow F, \phi$
and add the rule $\phi \leftarrow G; H$ where ϕ is a new atom.²
- (b) $(G; H), F \Rightarrow \phi, F$
and add the rule $\phi \leftarrow G; H$ where ϕ is a new atom.
- (c) $\text{not not } p \Rightarrow \text{not } \phi$
and add the rule $\phi \leftarrow \text{not } p$ where ϕ is a new atom. $\square$

Step 3. Splitting

A flat rule of form $H \leftarrow G_1; \dots; G_n$ is replaced by rules

$$\begin{aligned} H &\leftarrow G_1. \\ &\dots\dots \\ H &\leftarrow G_n. \end{aligned}$$

The set of all transformation rules in Steps 1, 2 and 3 define a binary relation over programs. We will use $\Pi \succ_i \Pi'$ to denote a single transformation step that transforms program Π to program Π' by an application of a transformation rule in Step i where $i \in [1..3]$. By a *rewrite sequence*, we mean a sequence of zero or more rewrite steps which takes the general form

$$\Pi_0 \succ_{i_0} \dots \succ_{i_{n-1}} \Pi_n \succ_{i_n} \dots$$

²Apparently, in this case all the occurrences of $(G; H)$ in the given program can be replaced by ϕ simultaneously.

A program Π is called an ω_i -normal form if Π cannot be reduced further by transformation rules in Step i where $i \in [1..3]$. In addition, we denote by

$$\Pi \succ_i^* \Pi'$$

a rewrite sequence generated only by the transformation rules in Step i .

Example 4.4 Suppose Π contains a single rule

$$a \leftarrow ((\text{not not } b); c), \text{not } d.$$

Π is an ω_1 -normal form. We can have $\Pi \succ_2 \Pi'$ where $\Pi' = \{a \leftarrow \xi, \text{not } d. \xi \leftarrow \text{not not } b; c.\}$, and $\Pi' \succ_2 \Pi''$ where $\Pi'' = \{a \leftarrow \xi, \text{not } d. \xi \leftarrow \text{not } \eta; c. \eta \leftarrow \text{not } b.\}$. Finally, $\Pi'' \succ_3 \Pi'''$ where $\Pi''' = \{a \leftarrow \xi, \text{not } d. \xi \leftarrow \text{not } \eta. \xi \leftarrow c. \eta \leftarrow \text{not } b.\}$.

Lemma 4.5 There is no infinite sequence

$$\Pi_0 \succ_i \dots \succ_i \Pi_n \succ_i \dots$$

where Π_0 is a nested program when $i = 1$, Π_0 is an ω_1 -normal form when $i = 2$, and Π_0 is an ω_2 -normal form when $i = 3$,

Lemma 4.6 Let n be the size of the program Π_0 mentioned below.

1. For any nested program Π_0 , and any rewrite sequence $\Pi_0 \succ_1 \dots \succ_1 \Pi_m$ where Π_m is an ω_1 -normal form, Π_m is a basic program, and m is bounded by $O(n)$.
2. For any ω_1 -normal form Π_0 , and any rewrite sequence $\Pi_0 \succ_2 \dots \succ_2 \Pi_m$ where Π_m is an ω_2 -normal form, Π_m is a flat program, and m is bounded by $O(n)$.
3. For any ω_2 -normal form Π_0 , and any rewrite sequence $\Pi_0 \succ_3 \dots \succ_3 \Pi_m$ where Π_m is an ω_3 -normal form, Π_m is an extended program, and m is bounded by $O(n)$.

We can show that every transformation in Step 2 preserves equivalence.

Lemma 4.7 If $\Pi \succ_2 \Pi'$ then $\Pi \sqsubseteq \Pi'$.

For any program Π , Lemmas 4.5 and 4.6 together guarantee the existence of a finite rewrite sequence

$$\Pi \succ_1^* \Pi' \succ_2^* \Pi'' \succ_3^* \Pi'''$$

where Π' is an ω_1 -normal form, Π'' is an ω_2 -normal form, and Π''' is an extended program. Let us call such a Π''' the *transformation of Π* , denoted by $Tr(\Pi)$.³

We are ready to show the main theorem of this section.

Theorem 4.8 For any program Π , $\Pi \sqsubseteq Tr(\Pi)$.

Proof. Assume $\Pi \succ_1^* \Pi' \succ_2^* \Pi'' \succ_3^* \Pi''' = Tr(\Pi)$. From [Lifschitz *et al.*, 1999], Π and Π' are strongly equivalent, so are Π'' and Π''' . It follows that $\Pi \sqsubseteq \Pi'$ and $\Pi'' \sqsubseteq \Pi'''$. Lemma 4.7 shows that every individual rewrite step in $\Pi' \succ_2^* \Pi''$ preserves equivalence. By transitivity (Lemma 4.2) we get $\Pi' \sqsubseteq \Pi''$, and therefore $\Pi \sqsubseteq Tr(\Pi)$. $\square$

³Although transformation rules can be applied nondeterministically, it is easy to show that the rewrite relation $\succ_i^*$ for any $i \in [1..3]$ is *confluent* [Dershowitz and Jouannaud, 1990] so that the final extended program is unique, up to atom renaming.

Corollary 4.9 For any programs Π and P ,

1. $\Pi \cup P \subseteq Tr(\Pi) \cup P$
2. $\Pi \cup P \subseteq Tr(\Pi) \cup Tr(P)$, if the sets of new atoms, $lit(Tr(\Pi)) - lit(\Pi)$ and $lit(Tr(P)) - lit(P)$, are disjoint.

The claim in 1 above is considered by some authors as a form of *strong equivalence* within the same language.

It follows from Lemma 4.6 that

Theorem 4.10 For any program Π , there is a linear time reduction of Π to an extended program Π' so that the answer sets for Π can be identified in linear time from those for Π' .

We remark that the extra atoms introduced in our transformation are *non-split*. They are just “connecting atoms” serving the purpose of propagating values from subexpressions. Their values can be determined solely by constraint propagation from the values of the original atoms.

Non-split variables can be accommodated easily in systems like Smodels by applying the procedure *Heuristic* only to the set of original variables, so that non-split variables will not be picked up for guessing a value. In this way, non-split variables do not create *choice points* during the search for answer sets.

4.2 Weakly tight nested logic programs

We give a characterization of answer sets for nested programs, in terms of their transformations, and show the counterpart of Theorem 3.4 for nested programs.

Definition 4.11 Let Π be a nested program and $Tr(\Pi)$ its transformation. Let $X \subseteq lit(\Pi)$ and V satisfy

$$V = \{\phi \in lit(Tr(\Pi)) \mid \phi \notin lit(\Pi), \phi \in T_{Tr(\Pi)}^{x \cup V} \uparrow \infty\}.$$

Then, Π is said to be *weakly tight* on X if and only if $Tr(\Pi)$ is weakly tight on $X \cup V$.

Theorem 4.12 Let Π be a nested program, and X a consistent set of literals. X is an answer set for Π iff Π is weakly tight on X , and X is closed under and supported by Π .

Corollary 4.13 Let Π be a finite nested program without the classic negation and S a set of atoms satisfying $Comp(\Pi)$. Then, Π is weakly tight on S iff S is an answer set for Π .

5 Related Work

Well-Supported Model

Definition 5.1 [Fages, 1991] Let Π be a normal program. A set of atoms S is *well-supported* iff there exists a level mapping λ with S as the domain such that for any atom $a \in S$ there exists a rule $a \leftarrow Body^+, Body^- \in \Pi$, which is applicable w.r.t. S , and for each $p \in Body^+$, $\lambda(a) > \lambda(p)$.

Ben-Eliyahu and Dechter’s characterization [Ben-Eliyahu and Dechter, 1994] of answer sets for head cycle-free disjunctive programs reduces to that of Fages for normal programs.

Theorem 5.2 [Fages, 1991] For any normal program Π , the well-supported models of Π are exactly the answer sets for Π .

The level mapping in Fages’ definition depends on the supportedness: one needs to find a rule that supports an atom, and then determines if a level mapping applies to the same rule. The two conditions in our case are independent.

Example 5.3 Let $\Pi = \{a \leftarrow b; b \leftarrow c\}$, and $S = \{a, b, c\}$. Note that S is a model of Π . Π is weakly tight (as well as tight) on S , since we can assign a level mapping as: $\lambda(a) = 2$, $\lambda(b) = 1$, and $\lambda(c) = 0$. But S is not well-supported by Π .

To summarize the relationships among tightness, weak tightness and well-supportedness for normal programs, we have, for any normal program Π and set of atoms S ,

- If S is well-supported, then Π is weakly tight on S ; but the converse is not true.
- If Π is tight on S , then Π is weakly tight on S ; but the converse is not true.
- That Π is tight on S is not a sufficient condition for S to be well-supported; that S is well-supported is not a sufficient condition for Π to be tight on S .

Therefore, the weak tightness is the weakest among all three, bridging the gap between tightness and well-supportedness.

Strong Compatibility of Logic Programs

In [Zhang, 1992], Zhang showed a characterization of default extensions. Here, we re-formulate it for logic programs.

For an extended logic program Π , let us define

$$\begin{aligned} Con(\Pi) &= \{H \mid H \leftarrow B^+, B^- \in \Pi\} \\ Pre(\Pi) &= \{L_i \mid 1 \leq i \leq m, \\ &\quad H \leftarrow L_1, \dots, L_m, not L_{m+1}, \dots, not L_n \in \Pi\} \\ Just(\Pi) &= \{\neg L_i \mid m+1 \leq i \leq n, \\ &\quad H \leftarrow L_1, \dots, L_m, not L_{m+1}, \dots, not L_n \in \Pi\} \end{aligned}$$

A subprogram $\Pi' \subseteq \Pi$ is said to be *compatible* (w.r.t. Π) if $Con(\Pi') \cup Just(\Pi')$ is consistent and $\perp \notin Con(\Pi')$.

For any $\Pi' \subseteq \Pi$, we let $\Lambda(\Pi') = \bigcup_{i=0}^{\infty} R_i(\Pi')$, where

$$\begin{aligned} R_0(\Pi') &= \{H \leftarrow not L_1, \dots, not L_n \in \Pi' \mid n \geq 0\}, \\ &\text{and for all } k \geq 0, \\ R_k(\Pi') &= \{H \leftarrow B^+, B^- \in \Pi' \mid B^+ \subseteq \bigcup_{i=0}^{k-1} Con(R_i(\Pi'))\} \end{aligned}$$

The characterization of default extensions as given in [Zhang, 1992], when applied to logic programs, can be stated as follows.

Theorem 5.4 Let Π be an extended logic program. Π has an answer set iff there is a subprogram $\Pi' \subseteq \Pi$ such that

- P1. Π' is compatible,
- P2. $\Lambda(\Pi') = \Pi'$, and
- P3. for any rule $r \in \Pi - \Pi'$, either $\Lambda(\Pi' \cup \{r\}) \neq \Pi' \cup \{r\}$, or $\Pi' \cup \{r\}$ is not compatible.

A subprogram $\Pi' \subseteq \Pi$ is said to be *strongly compatible* if it is compatible and $\Lambda(\Pi') = \Pi'$. We can show the following relationship.

Proposition 5.5 Let Π be an extended program. For any $\Pi' \subseteq \Pi$,

1. if Π' is strongly compatible, then Π is weakly tight on $Con(\Pi')$;
2. Π is weakly tight on $Con(\Pi')$, and $Con(\Pi')$ is closed under and supported by Π , iff Π' is strongly compatible and satisfies P3 as well.

Since no proper subset of $\{P1, P2, P3\}$ corresponds to the satisfaction of completion, to see whether a model of completion is also an answer set one needs all three conditions. The key lies in condition P2 which embodies both supportedness and an order of derivation. Thus, Zhang's characterization does not provide a nontrivial condition for a model of completion to be an answer set.

Note that P2 alone does not imply weak tightness. For example, with $\Pi = \{a \leftarrow a; a \leftarrow not a\}$, we have $\Lambda(\Pi) = \Pi$. But Π is not weakly tight on $Con(\Pi) = \{a\}$, due to the fact that $a \leftarrow a$ is the only applicable rule w.r.t. $\{a\}$.

ASSAT

In the system ASSAT [Lin and Zhou, 2002], the completion of a normal program is executed by a SAT solver, possibly repeatedly. When a model M of the completion from program Π is generated, if $M \neq \Pi^M$ then M is not an answer set for Π . ASSAT relies on $M - \Pi^M$ to compute loop formulas which say that the atoms in $M - \Pi^M$ that are involved in a (maximal) loop should be false if they cannot be derived otherwise. However, $M - \Pi^M$ does not provide a complete picture of the loops involved. There could be loops whose interpretation in M is faked due to the loops on $M - \Pi^M$. For example, let $\Pi = \{c \leftarrow c. c \leftarrow not b. b \leftarrow not a. a \leftarrow a\}$, and let the computed model of completion be $M = \{a, c\}$. In this case, $M - \Pi^M = \{a\}$. Though the loop on a is captured correctly, the loop on c is captured incorrectly. It remains to be seen whether stronger loop formulas can be computed, based on the idea of level mapping.

Transformations of Nested Programs

The transformation given in [Lifschitz *et al.*, 1999] deals with arbitrary nested programs where rules may have arbitrary formulas in the heads. The transformation preserves strong equivalence but is not polynomial.

Independently in [Pearce *et al.*, 2002], arbitrary nested programs are translated to disjunctive programs where extra variables are used to avoid an exponential blow-up, an idea similar to the one adopted in this paper. Our transformation however departs from that of [Lifschitz *et al.*, 1999] in a minimal way. By partitioning the transformation into three stages (cf. Theorem 4.8), we are able to identify exactly where extra variables are required. This is only when dealing with a disjunction or an expression like *not not L* where L is a literal. In [Pearce *et al.*, 2002], other forms of subformulas, such as the whole body of a rule and a conjunction, can also be labeled by extra variables. Our transformation is thus simpler and more compact. While our transformation takes linear time, the one in [Pearce *et al.*, 2002] is only claimed to be polynomial time. Finally, the extra variables introduced in our case are non-split. It is not clear whether this is possible for the class of all nested programs where the heads of rules are arbitrary formulas. It is interesting to see whether our approach, combined with some of the ideas in [Pearce *et*

al., 2002], can be extended in order to be able to handle all nested programs.

References

- [Babovich *et al.*, 2000] Y. Babovich, E. Erdem, and V. Lifschitz. Fages' theorem and answer set programming. In *Proc. 8th International Workshop on Non-Monotonic Reasoning*, 2000.
- [Babovich, 2002] Y. Babovich. Cmodels home page at www.cs.utexas.edu/users/tag/cmodels.html. 2002.
- [Ben-Eliyahu and Dechter, 1994] R. Ben-Eliyahu and R. Dechter. Propositional semantics for disjunctive logic programs. *Annals of Math. and Artificial Intelligence*, 14:53–87, 1994.
- [Clark, 1978] K.L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, 1978.
- [Davis and Putnam, 1960] M. Davis and H. Putnam. A computing procedure for quantification theory. *JACM*, 7(3):201–215, 1960.
- [Dershowitz and Jouannaud, 1990] N. Dershowitz and P. Jouannaud. Rewrite systems. In *Handbook of Theoretical Computer Science, Vol B: Formal Methods and Semantics*, pages 243–320. North-Holland, 1990.
- [Erdem and Lifschitz, to appear] E. Erdem and V. Lifschitz. Tight logic programs. *Theory and Practice of Logic Programming*, to appear.
- [Fages, 1991] F. Fages. A fixpoint semantics for general logic programs compared with the well-supported and stable model semantics. *New Generation Computing*, 9:425–443, 1991.
- [Fages, 1994] F. Fages. Consistency of Clark's completion and existence of stable models. *Journal of Methods of Logic In Computer Science*, 1:51–60, 1994.
- [Gelfond and Lifschitz, 1988] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proc. 5th ICLP*, pages 1070–1080. MIT Press, 1988.
- [Lifschitz *et al.*, 1999] V. Lifschitz, L. Tang, and H. Turner. Nested expressions in logic programs. *Annals of Mathematics and Artificial Intelligence*, 25:369–389, 1999.
- [Lin and Zhou, 2002] F. Lin and Y. Zhou. ASSAT, computing answer set of a logic program by SAT solvers. In *Proc. AAAI '02*, pages 112–118, 2002.
- [Lloyd and Topor, 1984] J. Lloyd and R. Topor. Making Prolog more expressive. *J. Logic Programming*, 1(3):225–240, 1984.
- [Pearce *et al.*, 2002] D. Pearce, V. Sarsakov, T. Schaub, H. Tompits, and S. Woltran. A polynomial translation of logic programs with nested expressions into disjunctive logic programs (preliminary report). In *Proc. 19th ICLP*, 2002.
- [Zhang, 1992] M. Zhang. A characterization of extensions of general default theories. In *Proc. 9th Canadian Artificial Intelligence Conference*, pages 134–139, 1992.

NONMONOTONIC REASONING

NONMONOTONIC REASONING

Answer Set Optimization*

Gerhard Brewka

Comp. Sci. Institute
University of Leipzig
Leipzig, Germany

brewka@informatik.uni-leipzig.de

Ilkka Niemelä

Dept. of Comp. Sci. and Eng.
Helsinki University of Technology
Helsinki, Finland
Ilkka.Niemela@hut.fi

Mirosław Truszczyński

Dept. of Comp. Sci.
University of Kentucky
Lexington, KY 40506-0046, USA
mirek@cs.uky.edu

Abstract

We investigate the combination of answer set programming and qualitative optimization techniques. Answer set optimization programs (*ASO* programs) have two parts. The generating program P_{gen} produces answer sets representing possible solutions. The preference program P_{pref} expresses user preferences. It induces a preference relation on the answer sets of P_{gen} based on the degree to which rules are satisfied.

We discuss possible applications of *ASO* programming, give complexity results and propose implementation techniques. We also analyze the relationship between *ASO* programs and *CP*-networks.

1 Introduction

Answer set semantics [Gelfond and Lifschitz, 1991] describes the meaning of a logic program P in terms of sets of literals. The exact definition of answer sets depends on the kind of rules used in P , yet two properties are always required. Answer sets are closed under the rules of P , and they are grounded in P : each literal has a derivation using “applicable” rules from P . Answer set programming has become a popular knowledge representation tool. There are several reasons for this:

1. Logic programs are expressive enough to model many typical knowledge representation problems in AI. In particular, the availability of default negation in the body of rules makes it possible to represent defeasible information.
2. Many problems in reasoning about actions, planning, diagnosis, belief revision and product configuration have elegant formulations as logic programs so that models of programs, rather than proofs of queries, describe problem solutions [Lifschitz, 2002; Soininen, 2000; Baral, 2003].
3. The semantics of answer sets is intuitive and avoids the pitfalls of resolution-based systems like Prolog. For instance, it is independent of the order in which rules are written and correctly handles loops.

*The authors acknowledge the support of DFG grant Computationale Dialektik BR 1817/1-5, Academy of Finland grant 53695 and NSF grant IIS-0097278, respectively.

4. In the same time, the syntax of logic programs is restrictive enough to allow for fast implementations and several highly efficient answer-set provers have been developed. Most advanced among them are *Smodels* [Niemelä and Simons, 1997] and *dlv* [Eiter *et al.*, 1998].

To increase the ease of use of logic programs in knowledge representation researchers have suggested and investigated several extensions to the basic formalism. Well-known examples of such extensions include disjunctive logic programs and programs with cardinality and weight constraints [Simons *et al.*, 2002].

An important issue for many applications is the representation of preferences and reasoning about them. Researchers have investigated preferences among program rules [Schaub and Wang, 2001], among program literals [Sakama and Inoue, 2000], and context-dependent preferences among literals through the use of ordered disjunction [Brewka, 2002].

Representing and handling preferences in the formalism of logic programs is also the main topic of this paper. However, our approach differs from existing ones in an important aspect. Rather than specifying a preference relation among the rules or literals in a single logic program, we use two different programs. The first program, P_{gen} , is used to *generate* answer sets, that is, define the space of *acceptable* solutions. Context-dependent preferences are described in a second program, the preference program P_{pref} . These preferences are used to *compare* answer sets of P_{gen} , that is, to form a preference ordering of acceptable solutions. Intuitively, the rules of P_{gen} are hard constraints which specify conditions an answer set *must* satisfy; the rules of P_{pref} are soft constraints describing conditions under which one answer set is to be considered better than another.

The decoupling of answer-set generation and answer-set comparison has at least two advantages:

1. The method for comparing answer sets is independent of the type of the generating program P_{gen} . It may be any type of a logic program (for instance, normal, extended, disjunctive, involving cardinality or weight atoms), as long as it has a well-defined semantics given by a collection of sets of literals.
2. Preferences in P_{pref} (soft constraints) can be specified independently of P_{gen} (hard constraints). This makes preference elicitation easier since the task is broken into sepa-

rate and smaller subtasks. Moreover, it makes the overall setting better aligned with practical applications. Indeed, often what is possible is determined by external factors (available resources, constraints of the environment, design and engineering constraints for products) while what is preferred is described independently and by different agents (users, customers, groups of customers).

The rest of the paper is organized as follows. The next section introduces a new formalism, answer-set optimization programs, and defines their syntax and semantics. The subsequent section provides examples and additional observations on the basic formalism. In Section 4, we show how meta-preferences, that is, preferences on the preference rules, can be introduced and dealt with. Next, we discuss complexity of computational problems arising in the context of our formalism and propose implementation techniques. Our work is related to the approach proposed in [Boutilier *et al.*, 1999]. We investigate this relationship in Section 6. We conclude with additional discussion of other related work and possible extensions to the formalism presented here.

2 Optimization programs

We use two separate programs to describe the space of answer sets and preferences among them.

Definition 1 Let A be set of atoms. A preference program over A is a finite set of rules of the form

$$C_1 > \dots > C_k \leftarrow a_1, \dots, a_n, \text{not } b_1, \dots, \text{not } b_m \quad (1)$$

where the a_i s and b_j s are literals (expressions x and $\neg x$, where x is an atom in A), and the C_i s are boolean combinations over A (to be defined below).

The rule intuitively reads: if an answer set S contains $a_1, \dots, a_n$ and does not contain any of the literals $b_1, \dots, b_m$ then C_1 is preferred over C_2 , C_2 over C_3 , etc. (we will give a precise semantics later in this section).

A *boolean combination* over A is a formula built of atoms in A by means of disjunction, conjunction, strong ($\neg$) and default (not) negation, with the restriction that strong negation is allowed to appear only in front of atoms, and default negation only in front of literals. For example, $a \wedge (b \vee \text{not } \neg c)$ is a boolean combination, whereas $\neg(a \vee b)$ is not. Using boolean combinations rather than, say, literals in the heads of preference rules gives us additional expressiveness. Using conjunction we can express that certain combinations of properties are preferred over other combinations. Disjunction allows us to express that certain options are equally preferred. For instance, the rule $a > (b \vee c) > d \leftarrow f$ says that in case of f the best option is a ; b and c are equally preferred second best options, and d is the least preferred option. Finally, we can use expressions like $a > b > c > (\text{not } a \wedge \text{not } b \wedge \text{not } c)$ if we prefer to have one of properties a, b, c over not having any of them.

Definition 2 Let S be a set of literals. Satisfaction of a boolean combination C in S (denoted $S \models C$) is defined as:

$$\begin{aligned} S \models l \text{ (l literal)} & \quad \text{iff} \quad l \in S \\ S \models \text{not } l \text{ (l literal)} & \quad \text{iff} \quad l \notin S \end{aligned}$$

$$\begin{aligned} S \models C_1 \vee C_2 & \quad \text{iff} \quad S \models C_1 \text{ or } S \models C_2 \\ S \models C_1 \wedge C_2 & \quad \text{iff} \quad S \models C_1 \text{ and } S \models C_2. \end{aligned}$$

We next define the notion of an optimization program.

Definition 3 An answer-set optimization (ASO) program is a pair (P_{gen}, P_{pref}) , where P_{gen} is a logic program called the generating program, and P_{pref} is a preference program.

As we already mentioned earlier, the program P_{gen} used for generating answer-sets can be of *any type*. We only require that the semantics be given in terms of sets of literals, or *answer sets*, that are associated with programs.

The key question is: how does the preference program P_{pref} determine a preference ordering on the answer sets described by the generating program P_{gen} ? Let us consider an answer set S and a rule of the form (1). Given a set S of literals, three different situations are possible:

1. the body of r is not satisfied in S , that is, $a_i \notin S$ for some $i \in \{1, \dots, n\}$, or some $b_j \in S$, for $j \in \{1, \dots, m\}$
2. the body of r is satisfied in S and none of the C_i s is satisfied in S
3. the body of r is satisfied in S and at least one C_i is satisfied in S .

In the case (1), the rule r is *irrelevant* to S because the rule does not apply. The case (2) is more subtle: the rule lists preferences among several options, yet none of the options holds. We consider this as *another* kind of irrelevance. Let us assume that a rule states that *red* is better than *green*, and S contains *blue* and no other color. In this case the preference of *red* over *green* appears irrelevant to S since S does not mention these two colors at all. In the case (3), the preference expressed in the rule is satisfied to some degree (as at least one C_i holds in S). Thus, we define the *satisfaction degree* of r in S , $v_S(r)$, by setting $v_S(r) = I$, if r is irrelevant to S , and $v_S(r) = \min\{i: S \models C_i\}$, otherwise.

Concerning the relationship between I and the other satisfaction degrees, two viable options seem to exist. We can consider I as *incomparable* to other values, based on the view that “irrelevance” cannot be compared to *proper* satisfaction degrees. According to this view, selecting *blue* in the example above is neither better nor worse than selecting *green* or *red*. But one can also argue that *green* violates preferences whereas *blue* does not, and that *blue* is thus preferable to *green*. We will adopt this latter view here and use the following *preorder* $\geq$ to compare satisfaction degrees (Fig. 1):

$$\begin{array}{c} 1, I \\ | \\ 2 \\ | \\ \dots \end{array}$$

Fig.1: The preorder on satisfaction degrees

The values I and 1 are regarded *equally good* ($1 \geq I$ and $I \geq 1$) and better than all others. In addition, for each x , we have $x \geq x$. We write $x > y$ if x is *strictly better* than y .

Definition 4 Let $P_{pref} = \{r_1, \dots, r_n\}$ be a preference program and let S be an answer set. We say that S induces a satisfaction vector $V_S = (v_S(r_1), \dots, v_S(r_n))$.

We extend the preorder on satisfaction degrees to preorders on satisfaction vectors and answer sets as follows:

Definition 5 Let S_1 and S_2 be answer sets. We write $V_{S_1} \geq V_{S_2}$ if $v_{S_1}(r_i) \geq v_{S_2}(r_i)$, for every $i \in \{1, \dots, n\}$. We write $V_{S_1} > V_{S_2}$ if $V_{S_1} \geq V_{S_2}$ and for some $i \in \{1, \dots, n\}$, $v_{S_1}(r_i) > v_{S_2}(r_i)$. In these cases we also write $S_1 \geq S_2$ and $S_1 > S_2$, respectively.

We refer to this ordering as the *preference satisfaction ordering* (or *ps-ordering*, for short).

Definition 6 A set of literals S is an optimal model of an ASO program (P_{gen}, P_{pref}) if S is an answer set of P_{gen} and there is no answer set S' of P_{gen} such that $S' > S$.

3 Examples

The prototypical application for ASO programs are configuration problems where P_{gen} describes possible configurations and P_{pref} preferences among them. We use a dinner example similar to the one discussed in [Brewka, 2002]. It is convenient to use programs with cardinality constraints to generate answer sets. Such programs allow for the use of special atoms of the form $n\{a_1, \dots, a_k\}m$, where the a_i are literals, to represent: at least n and at most m of the a_i are true. Although rules built from cardinality constraints can, in principle, be replaced by sets of rules without such constraints, they make problem specifications much more concise and readable. For the precise definitions we refer the reader to [Simons *et al.*, 2002].

Let us assume that P_{gen} consists of the rules:

1{*soup, salad*}1
1{*beef, fish*}1
1{*pie, ice-cream*}1
1{*red, white, beer*}1.

Each of these constraints enforces the selection of exactly one of the items it lists into an answer set. Thus, P_{gen} generates the space of $2 \times 2 \times 2 \times 3 = 24$ answer sets. Let us assume that P_{pref} is:

white > *red* > *beer* $\leftarrow$ *fish*
red $\vee$ *beer* > *white* $\leftarrow$ *beef*
pie > *ice-cream* $\leftarrow$ *beer*.

This preference program designates as non-preferred all answer sets containing *fish* but not *white*, all answer sets containing *beef* but not *red* or *beer*, and all answer sets containing *beer* and not *pie*. We note that the objective is not to have these answer sets *eliminated*, which could be accomplished simply by adding the three constraints

$\leftarrow$ *fish*, not *white*
 $\leftarrow$ *beef*, not *red*, not *beer*
 $\leftarrow$ *beer*, not *pie*.

to the original program. The role of the preference program is to define *soft constraints* and, moreover, to do so independently of the generating program. For instance, if we later learn that constraints $F = \{\leftarrow \text{red}; \leftarrow \text{white}; \leftarrow \text{pie}\}$ need to be included in P_{gen} , the additional constraints would lead to inconsistency. This does not happen in our approach. The answer sets generated by the extended program are:

$S_1 = \{\text{ice-cream}, \text{beer}, \text{beef}, \text{soup}\}$
 $S_2 = \{\text{ice-cream}, \text{beer}, \text{beef}, \text{salad}\}$
 $S_3 = \{\text{ice-cream}, \text{beer}, \text{fish}, \text{soup}\}$
 $S_4 = \{\text{ice-cream}, \text{beer}, \text{fish}, \text{salad}\}$

Their satisfaction vectors are $V_1 = (I, 1, 2)$, $V_2 = (I, 1, 2)$, $V_3 = (3, I, 2)$, and $V_4 = (3, I, 2)$, respectively. Thus, S_1 and S_2 are equally good and are maximally preferred in the presence of F to S_3 and S_4 , the latter two answer sets being also equally good.

4 Meta-preferences

The notion of optimality underlying our approach is somewhat weak. In general, many optimal answer sets may exist, and one often wants additional means to express that one preference (that is, one rule in the program P_{pref}) is more important than another. Here is a generalization of ASO programs where it is possible to express such meta-preferences:

Definition 7 A ranked ASO program P is a sequence

$$(P_{gen}, P_{pref}^1, \dots, P_{pref}^n)$$

consisting of a generating program P_{gen} and a sequence of pairwise disjoint preference programs P_{pref}^i .

The rank of a rule $r \in P_{pref}^1 \cup \dots \cup P_{pref}^n$, denoted $\text{rank}(r)$, is the unique integer i for which $r \in P_{pref}^i$.

Intuitively, preference rules with lower rank are preferred over preference rules with higher rank. We can now modify the definition of preference among answer sets by taking preferences among rules into account:

Definition 8 Let $P = (P_{gen}, P_{pref}^1, \dots, P_{pref}^n)$ be a ranked ASO program. Let S_1 and S_2 be answer sets of P_{gen} . We define $S_1 \geq_{\text{rank}} S_2$ if for every preference rule r' such that $v_{S_1}(r') \geq v_{S_2}(r')$ does not hold, there is a rule r'' such that $\text{rank}(r'') < \text{rank}(r')$ and $v_{S_1}(r'') > v_{S_2}(r'')$.

Clearly, the preorder $\geq_{\text{rank}}$ extends (is stronger than) the preorder $\geq$. It is also easy to see that $\geq_{\text{rank}}$ -optimal answer sets can be obtained in the following way: select all answer sets optimal wrt P_1 , among those pick the ones optimal wrt P_2 and so on.

A further generalization of ranked to partially ordered ASO programs is straightforward and not presented here for lack of space.¹

In some cases a natural ordering of the preference rules can be derived from the structure of the preference program. For each preference program P we define its dependency graph $G(P)$ as follows. The atoms appearing in P form the vertex set of $G(P)$. There is a directed edge from a vertex b to a vertex a in $G(P)$ if there is a rule r in P such that a appears in the head of r and b appears in the body of r .

If the graph $G(P)$ is acyclic, there is a natural ranking of its atoms. Namely, we define the rank of an atom a , $\text{rank}(a)$, recursively as follows: $\text{rank}(a) = 0$ for every atom a that has no predecessors in $G(P)$; otherwise, we define $\text{rank}(a)$

¹Another possible extension concerns the setting of weighted preference rules. To compare answer sets one could use weighted sums of the violation degrees of preference rules.

as the maximum of the ranks of all predecessors of a in $G(P)$ incremented by 1.

The ranking of atoms implies the ranking of rules. Namely, we define the rank of a preference rule r , $rank(r)$, as the maximum rank of an atom appearing in the head of r .

We call preference programs with acyclic dependency graphs *acyclic preference programs*. They are important for two reasons:

1. They commonly appear in practice as preferences are often described according to some partial order on features defining answer sets, with some features being more important than others. For instance, in the dinner example, most users will start with the preferences concerning the main course. They could then condition their preferences concerning the appetizer and the beverage on the choice of the main course. And, finally, they may describe their preferences concerning the dessert based on earlier choices of the main course, appetizer and beverage.
2. As we just demonstrated, in the case when $G(P)$ is acyclic, there exists a natural ranking on the rules. This ranking is implied by the program itself and allows us to strengthen our ordering relation on answer sets, as described above in Definition 8. We call the resulting ordering determined by an acyclic preference program — the *canonical ps-ordering*.

5 Complexity and implementation

The complexity of *ASO* programs depends on the class of generating programs. To simplify the treatment we consider here only generating programs where deciding existence of an answer set is **NP**-complete. This class of programs includes ground normal programs (possibly extended with strong negation or weight and cardinality constraints) [Simons *et al.*, 2002].

The following two results indicate that allowing preferences adds an extra layer of complexity.

Theorem 1 *Let $P = (P_{gen}, P_{pref})$ be an ASO program and S an answer set of P_{gen} . Then deciding whether S is optimal is **coNP**-complete.*

Theorem 2 *Given an ASO program P and a literal l deciding whether there is an optimal answer set S such that $l \in S$ is Σ_2^P -complete.*

The complexity results imply that (unless the polynomial hierarchy collapses) preference rules cannot be translated to generating rules in polynomial time, i.e., the problem of finding an optimal answer set for an *ASO* program cannot be mapped in polynomial time to a problem of finding an answer set of a program obtained by translating the *ASO* program to a set of generating rules only.

However, in [Brewka *et al.*, 2002] an implementation technique for computing optimal answer sets for logic programs with ordered disjunction on top of a standard answer set prover has been developed. A similar technique has earlier been used in [Janhunen *et al.*, 2000] to compute stable models of disjunctive logic programs using *Smodels*. The computation is based on a tester program that takes as input an answer set and generates a strictly better one if such an answer set

exists. The computation starts with an arbitrary answer set generated by the generator. This answer set is given to the tester program. If the tester fails to generate a strictly better one, we have found an optimal answer set and we are done. If a strictly better answer set is discovered, the tester is run with the new answer set as input. This continues until an optimal answer set is reached.

This technique can be adapted for computing optimal answer sets of an *ASO* program (P_{gen}, P_{pref}) by choosing a suitable tester program P_T for a given answer set S_0 . The tester program P_T is constructed by adding the following rules to the generator program P_{gen} :²

1. For each preference $r \in P_{pref}$ include a fact $v_0(r, d) \leftarrow$ where d is the satisfaction degree of r in S_0 .
2. Include rules

$$\begin{aligned} & \leftarrow \text{not better} \\ \text{better} & \leftarrow v_0(R, V_0), v_1(R, V_1), \\ & \quad \text{geq}(V_1, V_0), \text{not geq}(V_0, V_1) \\ & \leftarrow v_0(R, V_0), v_1(R, V_1), \text{not geq}(V_1, V_0). \end{aligned}$$
3. Add facts $\text{geq}(d_1, d_2) \leftarrow$ giving the preorder $\geq$ on the set $\{irr, 1, 2, \dots\}$.
4. For each Boolean combination C_i in P_{pref} (and its non-atomic subexpressions) introduce a new atom c_i and add rules capturing the conditions under which the expression is satisfied. For example, if C_i is a disjunction $C_h \vee C_l$, then add rules $c_i \leftarrow c_h$ and $c_i \leftarrow c_l$.
5. For each preference $r \in P_{pref}$ of the form (1) add rules

$$\begin{aligned} \text{body}(r) & \leftarrow a_1, \dots, a_n, \text{not } b_1, \dots, \text{not } b_m \\ \text{heads}(r) & \leftarrow c_i \quad (\text{for each } C_i) \\ v_1(r, irr) & \leftarrow \text{not body}(r) \\ v_1(r, irr) & \leftarrow \text{not heads}(r), \text{body}(r) \\ v_1(r, 1) & \leftarrow c_1, \text{body}(r) \\ v_1(r, 2) & \leftarrow \text{not } v_1(r, 1), c_2, \text{body}(r) \\ & \dots \\ v_1(r, k) & \leftarrow \text{not } v_1(r, 1), \dots, \text{not } v_1(r, k-1), \\ & \quad c_k, \text{body}(r). \end{aligned}$$

If P_T has an answer set S , then S restricted to the original language of P_{gen} is an answer set for P_{gen} that is strictly preferred to S_0 , and if P_T has no answer set, no such answer set exists.

6 Relationship to CP-networks

An important approach to the problem of eliciting and approximating preferences is that of *CP*-networks proposed and developed in [Boutilier *et al.*, 1999]. We will now review this approach and show how it relates to our work.

The approach of *CP*-networks is concerned with comparing vectors of feature values that we call *configurations*. Let $\mathcal{A} = \{A_1, \dots, A_k\}$ be a set of features (attributes). For each feature A_i , let D_i be its domain, that is, a finite and non-empty set of *values* or *selections* for A_i . We assume that all domains are pairwise disjoint. A *configuration* is a k -tuple $(v_1, \dots, v_k)$ such that $v_i \in D_i, 1 \leq i \leq k$.

²We follow the Prolog convention that terms starting with capital letters are variables and write *irr* rather than I to avoid confusion.

The user prefers some configurations to others. Since the number of configurations is, in general, exponential in $|\cup_{i=1}^k D_i|$, it may be impractical to directly elicit and store the user's preference ordering on the set of all configurations. Thus, the task is to identify partial (and, in some sense, basic) information about the user's preferences, and develop ways to approximate the preference ordering implied by this partial information. The formalism of *CP*-networks [Boutilier *et al.*, 1999] is an approach to accomplish that.

A *CP*-network over a set of features $\mathcal{A}$ is a pair $(G, \mathcal{P})$, where G is a directed graph whose vertices are features from $\mathcal{A}$. The edges of the graph G determine dependencies among features: preferences for a value for a feature A depend only on values selected for the parents of A in the network. Thus, for each feature A and for each selection of values for the parent features for A , a *CP*-network specifies a total ordering relation on the domain of A . All these total orderings form the component $\mathcal{P}$ of a *CP*-network³.

The information contained in a *CP*-network implies preferences among configurations. A configuration V is *one-step preferred* to a configuration W if for some feature A (given a configuration U , by $U(A)$ we denote the value from the domain of A selected to U):

1. $V(B) = W(B)$ for every feature $B \in \mathcal{A} \setminus \{A\}$ (that is, for every feature other than A), and
2. $V(A)$ is strictly preferred to $W(A)$ in the ordering of the domain D of A specified by the network for the selection for the parents of A as given by V (or, equivalently, by W).

The configuration V is *CP-preferred* to W if V can be obtained from W by a sequence of one-step improvements. It is easy to see that for acyclic *CP*-networks (acyclic networks seem to arise in most situations occurring in practice) this ordering is indeed a partial ordering; we call it a *CP*-ordering⁴.

We will now show that the information represented by a *CP*-network N (with features $\mathcal{A} = \{A_1, \dots, A_k\}$) can be represented by means of *ASO* programs. First, we specify the space of all configurations as answer sets to the program $P_{gen}(N)$ that, for each feature $A_i \in \mathcal{A}$ ($1 \leq i \leq k$) contains the following rule (we assume here that $D_i = \{d_1, \dots, d_n\}$):

$$1\{d_1, \dots, d_n\}1.$$

As already discussed in Section 3, such rule enforces the selection of exactly one of the listed values to an answer set. Thus, answer sets of the program $P_{gen}(N)$ are precisely the configurations of N .

To specify preferences, we proceed as follows. Let A be a feature and let $\mathcal{B} = \{B_1, \dots, B_r\}$ be the set of its parents in the *CP*-network. For every selection $(v_1, \dots, v_r)$ of values of features $B_1, \dots, B_r$, respectively, the network specifies an ordering, say $d_1 > \dots > d_n$, on the domain

$D = \{d_1, \dots, d_n\}$ of A . We represent that fact by including a rule

$$d_1 > \dots > d_n \leftarrow v_1, \dots, v_r$$

in the preference program $P_{pref}(N)$.

Thus, at the level of syntax, our approach extends that of *CP*-nets. In particular, [Boutilier *et al.*, 1999] is concerned only with one fixed space of all configurations that contain for every feature exactly one value from its domain. In contrast, in our approach we have a substantial flexibility in defining answer sets by varying the generator program.

In addition, our approach is more robust when user preferences are inconsistent, the situation that often occurs in practice. For instance, the user may specify a preference rule $a > b \leftarrow body_1$ and $b > a \leftarrow body_2$ unaware of situations where both $body_1$ and $body_2$ are satisfied. Such inconsistencies cannot be modeled by *CP*-networks. In our approach they can with the effect that one of the preference rules will be violated whenever both bodies are true.

The key question is that of the relationship between the semantics of the two approaches. First, we observe that they are different. Let us consider the following example from [Boutilier *et al.*, 1999]. There are two features A and B with values a, a' , and b, b' , respectively. Feature B depends on feature A and the preferences are specified as follows (we give them in the notation of *ASO* programs):

$$\begin{aligned} a &> a' \\ b &> b' \leftarrow a \\ b' &> b \leftarrow a'. \end{aligned}$$

In the *CP*-ordering, $S_1 = \{a, b'\}$ is preferred over $S_2 = \{a', b'\}$ as there is a one-step improvement leading from S_2 to S_1 . On the other hand, none of S_1 and S_2 is strictly preferred over the other in the ps-ordering (the corresponding satisfaction-degree vectors are: $(1, 2, I)$ and $(2, I, 1)$, respectively). The reason is that the meaning of the rules in the two approaches is slightly different. In the *CP*-net approach, $x > x' \leftarrow body$ means (x and x' form the domain of a feature X): among all answer sets satisfying $body$, answer set S_1 is better than S_2 if both agree on all features except X and S_1 makes x true and S_2 makes x' true. In our approach $x > x' \leftarrow body$ is more like a soft constraint expressing: whenever $body$ is true there is a reason to prefer x over x' . Here, among the answer sets satisfying $body$, an answer set S_1 is preferred over S_2 if $x \in S_1$, $x' \in S_2$, and the other rules of the program are satisfied at least as well in S_1 as in S_2 . In a sense, what is different is the interpretation of the *ceteris paribus* — other things being equal — phrase, more precisely, the interpretation of what is meant by the “other things” that have to be equal. In the *CP*-network approach the “other things” are the selections of values for other features which have to be the same. In our approach it is the degree of satisfaction of the other rules in the program.

In general, our basic approach yields a weaker ordering. Namely, we have the following theorem.

Theorem 3 *Let N be an acyclic *CP*-network and let V and W be two configurations. If V is strictly preferred to W under the ps-ordering determined by the *ASO* program $(P_{gen}(N), P_{pref}(N))$, then V is strictly preferred to W under the *CP*-ordering implied by N .*

³The restriction that domains be totally ordered can be dropped. We adopt it, following [Boutilier *et al.*, 1999] to keep the discussion simple.

⁴This ordering captures the intuition *ceteris paribus* or *all other things being equal*, the aspect that is discussed in detail in [Boutilier *et al.*, 1999].

It is easy to see that the representation of an acyclic *CP*-network in our syntax results in an acyclic preference program. In this case, we have the following result.

Theorem 4 *Let N be an acyclic *CP*-network and let V and W be two configurations. If V is strictly preferred to W under the *CP*-ordering determined by the *CP*-net N , then V is strictly preferred to W under the canonical *ps*-ordering determined by the *ASO* program $(P_{gen}(N), P_{pref}(N))$.*

These two theorems show that we can approximate the *CP*-ordering by means of orderings implied by our approach. It is important as the relationship between two answer sets with respect to (ranked) *ps*-ordering can be verified in polynomial time, while it is not known whether polynomial time algorithms exist in the case of the *CP*-ordering (in fact, there are examples of configurations that require an exponentially long chain of one-step improvements to demonstrate that one is preferred to the other).

7 Further related work and conclusions

In this paper we have introduced *ASO* programs which combine a generating program with a preference program. The combination allows us to specify possible solutions of a problem (answer sets) together with preferences among specific aspects of solutions. The preference program orders the answer sets according to the satisfaction degrees of its preference rules.

Numerous papers which introduce preferences to logic programming exist in the literature. For an overview of some of them see for instance [Schaub and Wang, 2001]. Closest to ours is the proposal in [Brewka, 2002; Brewka *et al.*, 2002]. Here, preferences are expressed through ordered disjunction. Ordered disjunction is a non-commutative kind of disjunction which gives preference to the first disjunct over the second. The approach presented here differs from ordered disjunction in the following respects:

1. generation and comparison of answer sets are separated,
2. preference handling works independently of the type of program used for answer set generation,
3. more general preferences can be stated due to our use of boolean combinations.

In fact, it is not difficult to show that logic programs with ordered disjunction (*LPODs*) are a special case of our approach. Let P be an *LPOD*, P_{gen} a logic program possessing the same answer sets as P , and let

$$P_{pref} = \{c_1 > \dots > c_n \leftarrow body : c_1 \times \dots \times c_n \leftarrow body \in P\}$$

The preferred answer sets of P under the Pareto criterion (cf. [Brewka *et al.*, 2002]) and those of the corresponding *ASO* program coincide. We can also capture the inclusion based preference criterion from [Brewka *et al.*, 2002] using appropriate ranked *ASO* programs. For each rule $c_1 \times \dots \times c_n \leftarrow body \in P$ and for each $i \leq n$ we have to include $c_1 \vee \dots \vee c_i > \top \leftarrow body$ in the preference program P_{pref}^i . This gives us exactly the inclusion-preferred answer sets of P .

In future work we plan to investigate further generalizations of our approach where rule heads may contain arbitrary

partial orders on boolean combinations. We intend also to study in depth the distinction between options being “equally good” and “incomparable”. Finally, we will study extensions of the one-step improvement concept from *CP*-networks to the setting of general *ASO* programs.

References

- [Baral, 2003] C. Baral. *Knowledge representation, reasoning and declarative problem solving*. Cambridge University Press, 2003. ISBN 0521818028.
- [Boutilier *et al.*, 1999] C. Boutilier, R.I. Brafman, H.H. Hoos, and D. Poole. Reasoning with conditional ceteris paribus preference statements. In *Proc. UAI-99*, 1999.
- [Brewka *et al.*, 2002] G. Brewka, I. Niemelä, and T. Syrjänen. Implementing ordered disjunction using answer set solvers for normal programs. In *Proc. JELIA 2002*. Springer Verlag, 2002.
- [Brewka, 2002] G. Brewka. Logic programming with ordered disjunction. In *Proc. AAAI-02*. Morgan Kaufmann, 2002.
- [Eiter *et al.*, 1998] T. Eiter, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello. The KR system dlw: Progress report, comparisons and benchmarks. In *Proc. Principles of Knowledge Representation and Reasoning, KR-98*. Morgan Kaufmann, 1998.
- [Gelfond and Lifschitz, 1991] M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.
- [Janhunen *et al.*, 2000] T. Janhunen, I. Niemelä, P. Simons, and J.-H. You. Unfolding partiality and disjunctions in stable model semantics. In *Proc. Principles of Knowledge Representation and Reasoning, KR-00*, pages 411–419. Morgan Kaufmann, 2000.
- [Lifschitz, 2002] V. Lifschitz. Answer set programming and plan generation. *Artificial Intelligence Journal*, 138(1-2):39–54, 2002.
- [Niemelä and Simons, 1997] I. Niemelä and P. Simons. Efficient implementation of the stable model and well-founded semantics for normal logic programs. In *Proc. 4th Intl. Conference on Logic Programming and Nonmonotonic Reasoning*. Springer Verlag, 1997.
- [Sakama and Inoue, 2000] C. Sakama and K. Inoue. Prioritized logic programming and its application to common-sense reasoning. *Artificial Intelligence*, 123(1-2):185–222, 2000.
- [Schaub and Wang, 2001] T. Schaub and K. Wang. A comparative study of logic programs with preference. In *Proc. IJCAI-01*, 2001.
- [Simons *et al.*, 2002] P. Simons, I. Niemelä, and T. Soinen. Extending and implementing the stable model semantics. *Artificial Intelligence*, 138(1-2):181–234, 2002.
- [Soininen, 2000] T. Soinen. *An Approach to Knowledge Representation and Reasoning for Product Configuration Tasks*. PhD thesis, Helsinki University of Technology, Finland, 2000.

Weak Conditional Logics of Normality

James P. Delgrande

School of Computing Science,
Simon Fraser University,
Burnaby, B.C., Canada V5A 1S6
jim@cs.sfu.ca

Abstract

A default conditional $\alpha \rightarrow \beta$ has most often been informally interpreted as a defeasible version of a classical conditional, usually the material conditional. That is, the intuition is that a default should behave (implicitly or explicitly) as its (say) material counterpart “by default” or unless explicitly overridden. In this paper, we develop an alternative interpretation, in which a default is regarded more like a rule, leading from premises to conclusion. To this end, a general semantic framework under a “rule-based” interpretation is developed, and a family of weak conditional logics is specified, along with associated proof theories. Nonmonotonic inference is defined very easily in these logics. One obtains a rich set of nonmonotonic inferences concerning the incorporation of irrelevant properties and of property inheritance. Moreover, this interpretation resolves problems that have been associated with previous approaches.

1 Introduction

A major approach in nonmonotonic reasoning has been to represent a default as an object that one can reason about, either as a conditional as part of some object language, or as a nonmonotonic consequence operator. Thus for example “an adult is (typically or normally) employed” might be represented $a \rightarrow e$, where $\rightarrow$ represents a default conditional, distinct from the material conditional $\supset$. Given a suitable proof theory and semantics, one can derive other defaults from a given set of defaults. There has been widespread agreement concerning just what principles should constitute a minimal logic, a suggested “conservative core,” of defaults. However, the resulting conditional is quite weak, at least compared with the material conditional, in that it does not (in fact, *should not*) fully support principles such as strengthening of the antecedent, transitivity, and modus ponens.

Since one would want to obtain these latter properties “by default,” such logics are extended nonmonotonically by a “closure” operation or step. This closure operation has, for example, been defined by selecting a (preferred) subset of

the models of a theory; in the resulting subset of the models one obtains strengthening of the antecedent, transitivity, or (effectively) modus ponens, wherever feasible. Essentially then, there are two components to default reasoning within such a system. First, there is a standard, monotonic logic of conditionals that expresses relations among defaults that are deemed to always hold. Second, there is a nonmonotonic mechanism for obtaining defaults (and default consequences) where justified. Representative (but not even remotely exhaustive) work in this area includes [Geffner and Pearl, 1992; Pearl, 1990; Kraus *et al.*, 1990]. In essence, these approaches treat the default conditional like its classical counterpart, the material conditional, where “feasible” or “by default”.

While this work captures an important notion of default entailment – perhaps *the* most important notion – it is not without difficulties. As described in the next section, some principles of the “core” logic are not uncontentious; as well, there are examples of default reasoning in which one obtains undesired results. Lastly, there are more recent approaches, notably addressing causality, in which one requires a weaker notion of default inference, rejecting, for example, contrapositive default inferences. In response to these points, we suggest that there is a second, distinct, interpretation of default conditionals, in which a default is regarded more like a rule, with properties more in line with a rule of inference, than a weakened classical conditional.

In the following sections we describe our proposed approach informally and formally. We begin by proposing an exceptionally weak logic of conditionals; from this basis a family of conditional logics is defined. Given a default conditional $\alpha \rightarrow \beta$, the underlying intuition that is formalised is that α supplies a context in which, all other things being equal, β normally holds or, more precisely, in the context of α , $\alpha \wedge \beta$ is more “normal” than $\alpha \wedge \neg\beta$. Notably, all of the logics that we consider are weaker than the aforementioned “conservative core”. It proves to be the case however that a nonmonotonic operation is very easily defined; this nonmonotonic step essentially specifies that a property is irrelevant with respect to a default unless it is known to be relevant. This nonmonotonic step easily admits inferences that in other approaches have taken significant formal machinery to obtain. As well, we show that the aforementioned difficulties that arise in interpreting a default as a weak classical conditional do not arise here.

This distinction between treating a default as a conditional or as a rule has been noted previously; see for example [Geffner and Pearl, 1992]. As well, work on inheritance networks [Horty, 1994] can be viewed as investigating proof theories for the latter interpretation; and work on causality such as [McCain and Turner, 1997] falls in the rule-based framework. However a logic (that is, with both semantics and proof theory) capturing this interpretation has not (to our knowledge) been investigated previously, nor has a fully general nonmonotonic closure operator been developed under this interpretation. Last, we suggest in the conclusion that this alternative interpretation may be widely applicable, extending to areas such as counterfactual reasoning, generally treated via the stronger interpretation.

2 Background

In recent years, much attention has been paid to conditional systems of default reasoning. Such systems deal with defeasible conditionals based on notions of preference among worlds or interpretations. Thus, the default that a bird normally flies can be represented propositionally as $b \rightarrow f$.¹ These approaches are typically expressed using a modal logic in which the connective $\rightarrow$ is a binary modal operator. The intended meaning of $\alpha \rightarrow \beta$ is approximately “in the least worlds (or most preferred worlds) in which α is true, β is also true”. Possible worlds (or, again, interpretations) are arranged in at least a partial preorder, reflecting a metric of “normality” or “preferredness” on the worlds. Given a set of defaults Γ , default entailment with respect to Γ , $\vdash_{\Gamma}$, can be defined via:

$$\text{If } \Gamma \vdash \alpha \rightarrow \beta \text{ then } \alpha \vdash_{\Gamma} \beta.$$

There has been a remarkable convergence or agreement on what inferences ought to be common to all nonmonotonic systems; space considerations preclude a full listing of approaches and references. The resulting set of principles has been called the *conservative core* in [Pearl, 1989]. It was originally considered in [Adams, 1975], and has been studied extensively, as the system **P**, in [Kraus *et al.*, 1990]. One expression of the logic of conditionals is as follows. The logic includes classical propositional logic and the following rules and axioms:²

RCEA/LLE: From $\vdash \alpha \equiv \beta$ infer $\vdash (\alpha \rightarrow \gamma) \equiv (\beta \rightarrow \gamma)$.

RCM/RW: From $\vdash \beta \supset \gamma$ infer $\vdash (\alpha \rightarrow \beta) \supset (\alpha \rightarrow \gamma)$

ID/Ref: $\alpha \rightarrow \alpha$

RT/Cut: $((\alpha \rightarrow \beta) \wedge (\alpha \wedge \beta \rightarrow \gamma)) \supset (\alpha \rightarrow \gamma)$

ASC/CM: $((\alpha \rightarrow \beta) \wedge (\alpha \rightarrow \gamma)) \supset (\alpha \wedge \beta \rightarrow \gamma)$

¹An alternative is to treat the conditional as a nonmonotonic inference operator, $b \vdash f$. In a certain sense these approaches can be considered equivalent; here, for simplicity, we remain within the conditional logic framework.

²Two systems of nomenclature have arisen, one associated with conditional logic and one with nonmonotonic consequence operators. We list both (when both exist) when first presenting an axiom or rule; for example the conditional logic rule for substitution of logical equivalents in the antecedent is called **RCEA**; its nonmonotonic consequence operator, Left Logical Equivalence is abbreviated **LLE**. Hence we first list the rule as **RCEA/LLE**.

CA/Or: $((\alpha \rightarrow \gamma) \wedge (\beta \rightarrow \gamma)) \supset (\alpha \vee \beta \rightarrow \gamma)$

These principles are not uncontentious; for example, [Poole, 1991] can be viewed as arguing against a derived principle **CC/And** (viz. $((\alpha \rightarrow \beta) \wedge (\alpha \rightarrow \gamma)) \supset (\alpha \rightarrow \beta \wedge \gamma)$ obtained via **RT** and **ASC**). Likewise, [Neufeld, 1989] suggests against **CA** in some cases.

Nonetheless, the resulting logic (and proposed strengthenings of the logic) are weak. A central difficulty is that seemingly *irrelevant* properties will block a desired inference. For example, given that a bird is asserted to fly by default, one cannot thereby conclude that a green bird flies by default. The problem, essentially, is that there is nothing requiring preferred worlds in which birds fly to include among them green-bird worlds. Consequently, various means of strengthening the logic to incorporate irrelevant properties in a principled fashion have been proposed. Thus for “birds fly,” since being green is presumably irrelevant with respect to flight, one would want to have, among the preferred worlds in which birds fly, a (preferred) subset in which there are green birds.

Rational closure [Lehmann and Magidor, 1992], for example, assumes that a world is ranked as unexceptional as possible. In *conditional entailment* [Geffner and Pearl, 1992], a partial order on possible worlds is determined by ranking worlds based on the highest-ranked default that distinguishes between the worlds. In both cases, defaults are evaluated with respect to the resulting ranking. Thus, since there is no reason to suppose that greenness has any bearing on flight, one assumes that greenness has no effect on flight.

However there are difficulties with both approaches. Rational closure employs a very strong minimization criterion; see [Geffner and Pearl, 1992] for a number of problematic examples. As well, consider an elaboration of an example given by John Horty:

$$\top \rightarrow \neg f, a \rightarrow f, \top \rightarrow n, o \rightarrow \neg n. \quad (1)$$

(Normally one does not eat with the fingers (f), but one does when eating asparagus (a); normally one uses a napkin (n), but not when one is out of napkins (o).) The rational closure of these conditionals gives that, if one is not out of napkins ($\neg o$), one is not eating asparagus ($\neg a$). Clearly this interaction between unrelated defaults is undesirable. As well, in neither conditional entailment nor rational closure does one obtain inheritance of properties (however see [Benferhat *et al.*, 1993]). In addition, consider the following example [Geffner and Pearl, 1992]:

$$a \rightarrow e, u \rightarrow a, u \rightarrow \neg e, f \rightarrow a. \quad (2)$$

(That is, adults are normally employed, university students are normally adults but are not employed, and Frank Sinatra fans are normally adults.) In both conditional entailment and rational closure we obtain the default inference that Frank Sinatra fans are not university students. But this is a curious inference, since there is nothing in the example that would seem to relate Frank Sinatra fans to university students.

We suggest that at least some of these examples do not necessarily reflect a problem with the approaches per se. Rather, our thesis is that there are (at least) two distinct interpretations that can be given to a default. First, there is the intuition

that a default is essentially a weak version of the material conditional (or, in more recent approaches, necessary entailment), and should behave as such a conditional, except that it is defeasible. This intuition is seen most clearly in the expression of defaults by circumscriptive abnormality theories [McCarthy, 1986]. In this case a default $\alpha \rightarrow \beta$ is represented as the formula $\alpha \wedge \neg Ab_i \supset \beta$. The circumscription of Ab_i asserts that Ab_i is false (roughly) if consistently possible. Obviously, if Ab_i is asserted to be false, the result is exactly the material conditional. As well, conditional entailment adopts this intuition: the default $\alpha \rightarrow \beta$ is basically the same as $\top \rightarrow (\alpha \supset \beta)$ (and so $\top \rightarrow (\neg\beta \supset \neg\alpha)$), together with specificity information implicit in α [Geffner and Pearl, 1992, p. 232]. This interpretation also underlies approaches that assume that defaults are founded (formally or informally) on notions of probability.³

The second interpretation regards a default more as an (object-level) *rule*, whose properties are closer to those of a rule of inference. Thus, given a conditional $\alpha \rightarrow \beta$, if the antecedent α happens to be true, we conclude β by default. Given $\neg\beta$ we specifically do not want to conclude $\neg\alpha$. This latter interpretation covers an important class of defaults that have not been addressed as a logic of conditionals per se (although this distinction has been noted in [Geffner and Pearl, 1992]; and inheritance networks [Horty, 1994] can be viewed as proof theoretic accounts of this interpretation). A further motivation for exploring this interpretation is that there has been recent interest in conditional accounts of causality (for example [McCain and Turner, 1997]), in which reasoning via a default contrapositive is explicitly rejected. Thus, from “ a causes b ” we don’t want to conclude “ $\neg b$ causes $\neg a$.”

3 Defaults as Rules

The general approach is the same as those described in the previous section: we begin by specifying a logic of defaults and subsequently provide a principled, nonmonotonic, means to extend the logic to account for irrelevant properties. Our point of departure is that we informally treat defaults more like rules of inference; in particular, defaults are intended to be applied in a “forward” direction only. Our interpretation, roughly, is that the antecedent of a default establishes a context in which the consequent (normally) holds, or holds all other things being equal. Thus, for default $\alpha \rightarrow \beta$, our interpretation is roughly that, in the context established by α , it is the case that β is more normal (typical, etc.) than $\neg\beta$. We express this semantically by

$$\|\alpha \wedge \neg\beta\|^M < \|\alpha \wedge \beta\|^M, \quad (3)$$

that is, the proposition (see the next section) $\|\alpha \wedge \beta\|^M$ is more normal (typical, etc.) than $\|\alpha \wedge \neg\beta\|^M$. It seems reasonable that our binary relation of relative normality $<$ be asymmetric and transitive, and so we generally assume that these conditions hold. We note that the form of (3) has appeared regularly in the literature, going back at least to [Lewis, 1973].⁴

³We note in passing that there are examples of defaults that have nothing to do with probability, for example (1).

⁴We use $<$ in the opposite sense of Lewis and many other authors. For our interpretation, it seems to make more sense to express

The difference is that usually the interpretation of (3) is along the lines of “the *least worlds* where $\alpha \wedge \neg\beta$ is true are less normal than the least $\alpha \wedge \beta$ worlds”; our interpretation refers to the *proposition* expressed by these formulas. Consequently, our relation $<$ is not an accessibility relation in the normal sense, since it is a relation on *sets* of worlds.

Filling in the (formal) details yields a weak logic of conditionals, significantly weaker than the so-called “conservative core”. We also consider various strengthenings of the logic, but these strengthenings are still weaker than this “core” set of defaults. We subsequently define a notion of nonmonotonic inference with respect to these logics. It proves to be the case that this is very easy to do in our approach. Basically, the (semantic) relation $X < Y$ asserts that in the “context” (set of possible worlds) $X \cup Y$, partitioned by X, Y , we have that Y is more normal than X . Our nonmonotonic assumption is that this obtains in all “feasible” subcontexts. That is, for proposition Z , unless there is reason to conclude otherwise, we assert that $Z \cap X < Z \cap Y$. The next section develops the formal details.

4 The Approach

4.1 The Base Logic

Let $\mathcal{L}_{PC}$ be the language of propositional logic defined, for simplicity, over a finite alphabet $\mathbf{P} = \{a, b, c, \dots\}$ of *propositional letters* or *atomic propositions*, and employing the logical symbols $\neg, \vee, \wedge, \supset$, and $\equiv$. The symbol $\top$ is taken to be some propositional tautology, and $\perp$ is defined as $\neg\top$. The language $\mathcal{L}$ is $\mathcal{L}_{PC}$ extended with the binary operator $\rightarrow$ as a *weak conditional*. For convenience, arguments of $\rightarrow$ are members of $\mathcal{L}_{PC}$; hence, we do not allow nested occurrences of $\rightarrow$. Formulas are denoted by the Greek letters $\alpha, \beta, \dots$ and sets of formulas by upper case Greek letters $\Gamma, \Delta, \dots$. Sentences are interpreted with respect to a *comparative conditional model* $M = \langle W, <, P \rangle$ where:

1. W is a set (of states or possible worlds);
2. $< \subseteq W \times 2^W \times 2^W$ with properties described below;
3. $P : \mathbf{P} \mapsto 2^W$.

We use the upper case letters $X, X_1, Y, \dots$ to denote sets of possible worlds. P maps atomic sentences onto sets of worlds, being those worlds at which the sentence is true. The relation $<$ associates with each world $w \in W$ a binary notion of *relative normality* between propositions; we write $X <_w Y$ to assert informally that, according to world w , proposition Y is more normal than X . That is, given a partition $\{X, Y\}$ of a context $X \cup Y$, the relation $X < Y$ asserts that Y is more normal (unexceptional, etc.) than X . We assume that $<$ is a strict partial ordering on its last two arguments, that is for $w \in W$, $<_w$ is asymmetric and transitive. As well, we assume that the incoherent proposition is maximally abnormal:

$$X \neq \emptyset \quad \text{iff} \quad \emptyset <_w X. \quad (4)$$

Truth of a formula at a world in a model is as for propositional logic, with an addition for $\rightarrow$:

“ Y is more normal than X ” by $X < Y$ rather than by $Y < X$.

Definition 4.1

1. $\models_w^M p$ for $p \in \mathbf{P}$ iff $w \in P(p)$.
2. $\models_w^M \alpha \wedge \beta$ iff $\models_w^M \alpha$ and $\models_w^M \beta$.
3. $\models_w^M \neg \alpha$ iff $\not\models_w^M \alpha$.
4. $\models_w^M \alpha \rightarrow \beta$ iff $\|\alpha \wedge \neg \beta\|^M <_w \|\alpha \wedge \beta\|^M$ or $\|\alpha\|^M = \emptyset$.

A formula α is *valid*, written $\models \alpha$, just if it is true at every world in every model. We identify the proposition expressed by a sentence α with the set of worlds in which α is true, denoted $\|\alpha\|^M$, that is,

$$\|\alpha\|^M = \{w \mid \models_w^M \alpha\}.$$

Thus $\alpha \rightarrow \beta$ is true just if the proposition expressed by $\alpha \wedge \beta$ is more normal than that expressed by $\alpha \wedge \neg \beta$. α is necessarily true, $\Box \alpha$, is defined as $\neg \alpha \rightarrow \alpha$.

Consider the logic closed under classical propositional logic along with the following rules of inference and axioms:

RCEA/LE: From $\vdash \alpha \equiv \alpha'$ infer $\vdash (\alpha \rightarrow \beta) \equiv (\alpha' \rightarrow \beta)$

RCECA:

$$\text{From } \vdash \alpha \supset (\beta \equiv \beta') \text{ infer } \vdash (\alpha \rightarrow \beta) \equiv (\alpha \rightarrow \beta')$$

RI/SupraCI: From $\vdash \alpha \supset \beta$ infer $\vdash \alpha \rightarrow \beta$

WeakCEM: $\neg(\alpha \rightarrow \perp) \supset ((\alpha \rightarrow \beta) \supset \neg(\alpha \rightarrow \neg \beta))$

Trans: $((\alpha \vee \beta \rightarrow \gamma) \wedge (\beta \vee \gamma \rightarrow \gamma)) \supset (\alpha \vee \gamma \rightarrow \gamma)$

We call the smallest logic based on the above axiomatisation **C**. Theoremhood of a formula α is denoted, as usual, by $\vdash \alpha$. This system is quite weak; however in the full paper we consider the even weaker logic **C⁻** consisting of propositional logic and **RCEA**, **RCECA**, and **RI**. **RCEA** asserts that conditionals with the same consequent and equivalent (in propositional logic) antecedents are equivalent. **RCECA** asserts the same thing with respect to consequents, but is somewhat more general, in that the consequents need be equivalent just in the “context” given by the antecedent. **RI** asserts that if β is logically implied by α , then it is also normally implied. **WeakCEM** gives a weak version of the excluded middle for a weak conditional; in the semantics this is reflected by asymmetry of $<$. Similarly **Trans** reflects transitivity of $<$ in the semantics. We obtain the following basic results:

Theorem 4.1

1. $\alpha \rightarrow \alpha$
2. $(\alpha \rightarrow \beta) \supset (\alpha \rightarrow (\alpha \wedge \beta))$
3. $(\alpha \rightarrow \beta) \supset (\alpha \rightarrow (\alpha \supset \beta))$
4. if $\vdash \beta$ then $\vdash \alpha \rightarrow \beta$

As well, we obtain:

Theorem 4.2 α is valid in the class of comparative conditional models iff $\vdash \alpha$ in **C**.

For those familiar with conditional logic, the formulas **RCM**, **RT**, **ASC**, **CC**, and **CA** (and their nonmonotonic consequence operator counterparts: Right Weakening, Cut, Cautious Monotony, And, and Or) are not valid in **C**. Nonetheless, despite its (monotonic) inferential weakness, the logic already allows a rich set of nonmonotonic inferences, as covered in the next section. However, before proceeding we first consider various extensions to the logic.

4.2 Extensions to the Logic

In the logic **C** most properties of the relation $<_w$ stem from its being a strict partial order (viz. asymmetric and transitive). We also look at strengthening $<_w$ by considering properties that seem reasonable for a notion of normality. Consider the following:

CD: If $X <_w Y$ then $X \setminus Z <_w Y$.

CU: If $X <_w Y$ then $X <_w Y \cup Z$ provided $X \cap Z = \emptyset$.

UD: If $X <_w Y$ then $X \setminus Z <_w Y \cup Z$.

WDjU: If $X_1 <_w Y_1$, $X_2 <_w Y_2$, $Y_1 \cap Y_2 = \emptyset$ and $(X_1 \cup X_2) \cap (Y_1 \cup Y_2) = \emptyset$ then $X_1 \cup X_2 <_w Y_1 \cup Y_2$.

For **CD**, if X is less normal than Y , then a stronger proposition than X (viz. $X \setminus Z$) is also less normal than Y . **CU** is a dual: if X is less normal than Y , then a weaker proposition than Y (viz. $Y \cup Z$) is also more normal than X . **UD** combines these conditions, and **WDjU** is a weak version of disjoint unions, described below. Interestingly, the first two of these conditions have appeared in the belief revision literature. Our relation $<$ is what [Alchourrón and Makinson, 1985] call a (*transitive*) *hierarchy*; while $<$ with **CD** and **CU** is a *regular hierarchy*. Their interpretation of $<$ echoes ours for $X < Y$, that “ X is less secure or reliable or plausible ... than Y ” [Gärdenfors and Rott, 1995, p. 75].

Consider next the following rules and formulas:

WeakRCM: If $\vdash \beta \supset \gamma$ then $\vdash (\alpha \rightarrow \beta) \supset (\alpha \wedge \gamma \rightarrow \beta)$

CM/Cond: $(\alpha \wedge \beta \rightarrow \gamma) \supset (\alpha \rightarrow (\beta \supset \gamma))$

RCM: If $\vdash \beta \supset \gamma$ then $\vdash (\alpha \rightarrow \beta) \supset (\alpha \rightarrow \gamma)$

D: $((\alpha \wedge \beta \rightarrow \gamma) \wedge (\alpha \wedge \neg \beta \rightarrow \gamma)) \supset (\alpha \rightarrow \gamma)$.

Space considerations preclude a full discussion of these conditions. **WeakRCM** is a weaker version of both the rule **RCM** and the formula **ASC** in conditional logic. **CM** (“Conditionalisation” in nonmonotonic consequence relations) gives a conditional version of one half of the deduction theorem. Combining **WeakRCM** and **CM** yields the rule **RCM**, allowing weakening of the consequent of a conditional. **D** supplies a certain “reasoning by cases” for the conditional.

We obtain the following correspondence between semantic conditions and the axiomatisation:

Theorem 4.3 **C** + **CD** (**CU**, **UD**, **WDjU**) is complete with respect to the class of comparative conditional models closed under **WeakRCM** (**CM**, **RCM**, **D**).

We could go on and add other conditions in the semantics. Again space considerations dictate against a lengthy discussion, but two conditions are worth noting here:

Disjoint Union: If $(X \cup Y) \cap Z = \emptyset$ then:

$$X <_w Y \text{ iff } X \cup Z <_w Y \cup Z.$$

Connectivity: For $X, Y \subseteq W$, either $Y <_w X$ or $X <_w Y$.

Disjoint union has appeared frequently in the literature, for example [Savage, 1972; Fine, 1973; Dubois *et al.*, 1994]. The addition of disjoint union requires that the notion of a model be altered slightly (from a relation $<$ to $\leq$); the resultant semantic framework would correspond to the basic definition

of a *plausibility structure* [Friedman and Halpern, 2001]. The addition of connectivity would make $<_w$ a *qualitative probability* in the terminology of [Savage, 1972].

5 Nonmonotonic Reasoning

We claimed at the outset that the logic **C** and its strengthenings would allow a simple approach to nonmonotonic inference, having just the “right” properties for a rule-based interpretation of a conditional. For the logics, the central idea was that, given a partition $\{X, Y\}$ of a context $X \cup Y \subseteq W$, the relation $X < Y$ asserts that Y is more normal (unexceptional, etc.) than X . To obtain nonmonotonic inference, we simply assume that this relation holds in any subcontext, that is $X \cap Z < Y \cap Z$ wherever “reasonable”. More formally, we have the following:

Definition 5.1 Let $M = \langle W, <, P \rangle$ be a comparative conditional model in **C**.

Define $M^* = \langle W, <^*, P \rangle$, the augmentation of M , by:

$X <^*_w Y$ iff there are $X' \supseteq X$, $Y' \supseteq Y$ such that

1. $X' <_w Y'$ and
2. for every X'', Y'' where
 $X \subseteq X''$, $Y \subseteq Y''$ and $Y'' <_w X''$

we have:

$$X' \subseteq X'', Y' \subseteq Y''.$$

In the augmentation of a model, we will have $X <^*_w Y$ if $X <_w Y$. Otherwise, $X <^*_w Y$ if there are X', Y' where $X' <_w Y'$ for $X \subseteq X'$ and $Y \subseteq Y'$, and there is no X'', Y'' where $X \subseteq X'' \subseteq X'$ and $Y \subseteq Y'' \subseteq Y'$ and $Y'' <_w X''$ in the original model.

Theorem 5.1 If $M = \langle W, <, P \rangle$ is a comparative conditional model then so is $M^* = \langle W, <^*, P \rangle$.

We define $\models^*$ as validity in the class of augmented comparative conditional models; that is $\models^* \alpha$ iff α is true at every world in every augmented comparative conditional model. Nonmonotonic inference is defined as follows:

Definition 5.2 Let $\Gamma \subseteq \{\alpha \rightarrow \beta \mid \alpha, \beta \in \mathcal{L}_{PC}\}$.

$\alpha \sim_{\Gamma} \beta$ iff $\models^* \Gamma \supset (\alpha \rightarrow \beta)$.

We say that β is a nonmonotonic inference from α with respect to Γ , or just β is a nonmonotonic inference from α if the set Γ is clear from the context of discussion.

We illustrate nonmonotonic inference first by a familiar example:

$$b \rightarrow f, \tag{5}$$

$$b \rightarrow w, \tag{6}$$

$$\Box(p \supset b), \tag{7}$$

$$p \rightarrow \neg f. \tag{8}$$

Thus birds fly and have wings, and penguins are (necessarily) birds that do not fly. We obtain the following:

$$\begin{array}{ll} b \wedge w \sim f, & p \wedge w \sim \neg f, \\ b \wedge \neg w \sim f, & p \wedge b \sim \neg f, \\ b \wedge \neg p \sim f, & p \wedge b \wedge w \sim \neg f. \end{array}$$

We also obtain $b \wedge x \wedge y \wedge z \sim w$ for $x \in \{\top, g, \neg g\}$, $y \in \{\top, p, \neg p\}$, $z \in \{\top, f, \neg f\}$. Thus green (g) birds have wings,

as do non-green flying penguins. As well $p \sim w$, and so penguins inherit the property of having wings by virtue of necessarily being birds. Note that if we replaced (7) by $p \rightarrow b$, we would no longer obtain $p \sim w$; however we would obtain the weaker $b \wedge p \sim w$. We justify this by noting that a normality conditional $\alpha \rightarrow \beta$ does not imply a strict specificity relation between α and β whereas $\Box(\alpha \supset \beta)$ does.

The next example further illustrates reasoning in the presence of exceptions.

$$q \rightarrow p, \quad r \rightarrow \neg p, \quad q \rightarrow g \tag{9}$$

So Quakers are pacifists while Republicans are not, and Quakers are generous. We obtain $q \wedge \neg r \sim p$ and $q \wedge r \sim g$. Thus in the last case, while Quakers that are Republican are, informally, exceptional Quakers, they are nonetheless still generous by default.

Concerning our original motivating examples, in (1) we do not obtain the undesirable inference $\neg o \sim \neg a$, and in (2) we do not obtain $f \sim \neg u$. Last, we note that while we obtain full incorporation of irrelevant properties, we do not obtain full default transitivity. Thus

$$a \rightarrow b, \quad b \rightarrow c$$

does not yield $a \sim c$ (nor, incidentally, do we obtain $\neg b \sim \neg a$). However we do get $a \wedge b \sim c$. If we replaced $a \rightarrow b$ with $\Box(a \supset b)$ we would get $a \sim c$. If we replaced $b \rightarrow c$ with $\Box(b \supset c)$ we would again get $a \sim c$, in **C** (in fact we could derive $a \rightarrow c$ in those logics containing **RCM**, as given in Theorem 4.3).

6 Discussion

We have argued that there are two interpretations of a default conditional: as a weak (typically material) implication, or as something akin to a rule of inference. The former interpretation is explicit in, for example, circumscriptive abnormality theories, and implicit in an approach such as conditional entailment. It is clear that there are many, and varied, applications in which the first interpretation is appropriate. However we have also noted that there are various reasons to suppose that this is not the only such interpretation: First, work such as [Poole, 1991] and [Neufeld, 1989] can be viewed as arguing against principles of the “core” logic underlying this first interpretation (the former arguing against the principle **CC/And** and the latter against **CA/Or**). Second, there are examples of inferences in approaches such as rational closure or in conditional entailment that are either too weak or too strong. Last, there are emerging areas (such as causal reasoning) in which a “weak material implication” interpretation is not appropriate. While this distinction has been recognized previously, what is new here is the development of a family of logics, with a novel semantic theory and proof theory, along with a specification of nonmonotonic inference, for the “rule-based” interpretation.

All of the logics presented here are quite weak, at least compared to the “conservative core” or, equivalently, the system **P** of [Kraus et al., 1990]. We argue however that such lack of inferential capability is characteristic of a “rule-based” interpretation of a conditional. Moreover it proves to

be the case that nonmonotonic reasoning is defined very easily in these logics, and allows a rich set of inferences concerning the incorporation of irrelevant properties and of property inheritance.

An open question concerns how informal, commonsense defaults should be classified – whether as a defeasible classical conditional or as a rule. Certainly past work has favoured the “defeasible classical conditional” interpretation. However, a case can be made that many examples formerly interpreted as belonging to the first category are better interpreted as belonging to the “rule” category. Consider Lewis’ approach to counterfactuals [Lewis, 1973] in which the following example, concerning a past party, is given: “If John had gone it would have been a good party” and “If John and Mary had gone it would have not been a good party”. From this we deduce that “if John had gone, Mary would not have gone”. This, to most readers, is a strange result: John’s going and Mary’s going are (presumably) independent events. Arguably this result ought not to obtain, and so perhaps counterfactuals, as previously modelled by Lewis’ sphere semantics, may be better interpreted via the “rule” interpretation.

Acknowledgements

I thank the reviewers for their useful suggestions. This work was supported by a Natural Sciences and Engineering Council of Canada Discovery Grant.

References

- [Adams, 1975] E. Adams. *The Logic of Conditionals*. D. Reidel Publishing Co., Dordrecht, Holland, 1975.
- [Alchourrón and Makinson, 1985] C.E. Alchourrón and D. Makinson. On the logic of theory change: Safe contraction. *Studia Logica*, 44:405–422, 1985.
- [Benferhat *et al.*, 1993] S. Benferhat, C. Cayrol, D. Dubois, J. Lang, and H. Prade. Inconsistency management and prioritized syntax-based entailment. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 640–645, Chambéry, Fr., 1993.
- [Dubois *et al.*, 1994] D. Dubois, J. Lang, and H. Prade. Possibilistic logic. In Dov M. Gabbay, C. J. Hogger, and J. A. Robinson, editors, *Nonmonotonic Reasoning and Uncertain Reasoning*, volume 3 of *Handbook of Logic in Artificial Intelligence and Logic Programming*, pages 439–513. Oxford, 1994.
- [Fine, 1973] T.L. Fine. *Theories of Probability: An Examination of Foundations*. Academic Press, 1973.
- [Friedman and Halpern, 2001] N. Friedman and J. Halpern. Plausibility measures and default reasoning. *JACM*, 48(4):649–685, 2001.
- [Gärdenfors and Rott, 1995] P. Gärdenfors and H. Rott. Handbook of logic in artificial intelligence and logic programming, volume 4 of *Epistemic and Temporal Reasoning*. Oxford Science Publications, 1995.
- [Geffner and Pearl, 1992] H. Geffner and J. Pearl. Conditional entailment: Bridging two approaches to default reasoning. *Artificial Intelligence*, 53(2-3):209–244, 1992.
- [Horty, 1994] J.F. Horty. Some direct theories of nonmonotonic inheritance. In D.M. Gabbay, C.J. Hogger, and J.A. Robinson, editors, *Nonmonotonic Reasoning and Uncertain Reasoning*, volume 3 of *Handbook of Logic in Artificial Intelligence and Logic Programming*, pages 111–187. Oxford: Clarendon Press, 1994.
- [Kraus *et al.*, 1990] S. Kraus, D. Lehmann, and M. Magidor. Nonmonotonic reasoning, preferential models and cumulative logics. *Artificial Intelligence*, 44(1-2):167–207, 1990.
- [Lehmann and Magidor, 1992] D. Lehmann and M. Magidor. What does a conditional knowledge base entail? *Artificial Intelligence*, 55(1):1–60, 1992.
- [Lewis, 1973] D. Lewis. *Counterfactuals*. Harvard University Press, 1973.
- [McCain and Turner, 1997] N. McCain and H. Turner. Causal theories of action and change. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, pages 460–465, Providence, RI, 1997.
- [McCarthy, 1986] J. McCarthy. Applications of circumscription to formalizing common-sense knowledge. *Artificial Intelligence*, 28:89–116, 1986.
- [Neufeld, 1989] E. Neufeld. Defaults and probabilities; extensions and coherence. In *Proceedings of the First International Conference on the Principles of Knowledge Representation and Reasoning*, pages 312–323, Toronto, Canada, 1989.
- [Pearl, 1989] J. Pearl. Probabilistic semantics for nonmonotonic reasoning: A survey. In *Proceedings of the First International Conference on the Principles of Knowledge Representation and Reasoning*, pages 505–516, Toronto, May 1989. Morgan Kaufman.
- [Pearl, 1990] J. Pearl. System Z: A natural ordering of defaults with tractable applications to nonmonotonic reasoning. In R. Parikh, editor, *Proc. of the Third Conference on Theoretical Aspects of Reasoning About Knowledge*, pages 121–135, Pacific Grove, Ca., 1990. Morgan Kaufmann Publishers.
- [Poole, 1991] D.L. Poole. The effect of knowledge on belief: Conditioning, specificity, and the lottery paradox in default reasoning. *Artificial Intelligence*, 49(1-3):281–307, 1991.
- [Savage, 1972] L.J. Savage. *The Foundations of Statistics*. Dover Publications Inc., New York, second edition, 1972.

Recycling Computed Answers in Rewrite Systems for Abduction

Fangzhen Lin*

<http://www.cs.ust.hk/~flin>

Department of Computer Science
Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong

Jia-Huai You

<http://www.cs.ualberta.ca/~you>

Department of Computer Science
University of Alberta
Edmonton, Alberta, Canada

Abstract

In rule-based systems, goal-oriented computations correspond naturally to the possible ways that an observation may be explained. In some applications, we need to compute explanations for a series of observations with the same domain. The question whether previously computed answers can be recycled arises. A yes answer could result in substantial savings of repeated computations. For systems based on classic logic, the answer is *yes*. For nonmonotonic systems however, one tends to believe that the answer should be *no*, since recycling is a form of adding information. In this paper, we show that computed answers can always be recycled, in a nontrivial way, for the class of rewrite procedures proposed earlier in [Lin and You, 2001] for logic programs with negation. We present some experimental results on an encoding of a logistics domain.

1 Introduction

Logic programming with answer sets [Gelfond and Lifschitz, 1988] or partial stable models [Przymusiński, 1990] has been adopted as a framework for abduction, and a number of formalisms and top-down query answering procedures have been proposed [Dung, 1991; Eshghi and Kowalski, 1989; Kakas and Mancarella, 1990; Kakas *et al.*, 2000; Lin and You, 2001; Satoh and Iwayama, 1992].

The question we shall address in this paper is the following. With a sound and complete procedure for abduction, suppose we have computed explanations (conveniently represented as a disjunction) $Es = E_1 \vee \dots \vee E_n$ for observation q . Suppose also that in the course of computing explanations for another observation p , we run into q again. Now, we may use the proofs Es for q without actually proving q again. The question is this: will the use of the proofs Es for q in the proof for p preserve the soundness and completeness of the procedure?

In this paper, we answer this question positively, but in a nontrivial way, for the class of rewrite procedures proposed

in [Lin and You, 2001]. The main result is a theorem (Theorem 4.6) that says recycling preserves the soundness and completeness.

The class of rewrite procedures proposed in [Lin and You, 2001] is based on the idea of *abduction as confluent and terminating rewriting*. These systems are called *canonical systems* in the literature of rewrite systems [Dershowitz and Jouannaud, 1990]. The confluence and termination properties guarantee that rewriting terminates at a unique normal form independent of the order of rewriting. Thus, each particular strategy of rewriting yields a rewrite procedure.

The general idea of recycling is not new. Recycling in systems based on classic logic is always possible, since inferences in these systems can be viewed as transforming a logic theory to a logically equivalent one. In dynamic programming, it is the use of the answers for previously computed subgoals that reduces the computational complexity. In some game playing programs, for example in the world champion checker program *Shinook* (www.cs.ualberta.ca/~chinook), the endgame database stores the computed results for endgame situations which can be referenced in real-time efficiently.

However, the problem of recycling in a nonmonotonic proof system has rarely been investigated. We note that recycling is to use previous proofs. This differs from adding consequences. For example, it is known that the semantics based on answer sets or (maximal) partial stable models do not possess the *cautious nonmonotonicity* property. That is, adding a consequence of a program could gain additional models thus losing some consequences. The following example is due to Dix [1991]:

$$P = \{a \leftarrow \text{not } b. \quad b \leftarrow c, \text{not } a. \quad c \leftarrow a.\} \quad (1)$$

P has only one answer set, $\{a, c\}$. Thus, c is a consequence. When augmented with the rule $c \leftarrow$, the program gains a second answer set, $\{b, c\}$, and loses a as a consequence.

The next section defines logic program semantics. Section 3 reviews the rewriting framework. Then in Section 4 we formulate rewrite systems with computed rules and prove that recycling preserves soundness and completeness. Section 5 extends this result to rewrite systems with abduction, and Section 6 reports some experimental results.

*This author's work was supported in part by HK RGC CERG HKUST6205/02E.

2 Logic Program Semantics

A rule is of the form

$$a \leftarrow b_1, \dots, b_m, \text{not } c_1, \dots, \text{not } c_n.$$

where a , b_i and c_i are atoms of the underlying propositional language $\mathcal{L}$. $\text{not } c_i$ are called *default negations*. A *literal* is an atom ϕ or its negation $\neg\phi$. A (*normal*) *program* is a finite set of rules. The *completion* of a program P , denoted $\text{Comp}(P)$, is a set of equivalences: for each atom $\phi \in \mathcal{L}$, if ϕ does not appear as the head of any rule in P , $\phi \leftrightarrow F \in \text{Comp}(P)$; otherwise, $\phi \leftrightarrow B_1 \vee \dots \vee B_n \in \text{Comp}(P)$ (with default negations replaced by the corresponding negative literals) if there are exactly n rules $\phi \leftarrow B_i \in P$ with ϕ as the head. We write T for B_i if B_i is empty.

The rewriting system of [Lin and You, 2001] is sound and complete w.r.t. the partial stable model semantics [Przymusiński, 1990]. A simple way to define partial stable models without even introducing 3-valued logic is by the so called *alternating fixpoints* [You and Yuan, 1995]. Let P be a program and S a set of default negations. Define a function over sets S of default negations: $F_P(S) = \{\text{not } a \mid P \cup S \not\models a\}$. The relation $\vdash$ is the standard propositional derivation relation with each default negation $\text{not } \phi$ being treated as a named atom $\text{not } \phi$.

A *partial stable model* M is defined by a fixpoint of the function that applies F_P twice, $F_P^2(S) = S$, while satisfying $S \subseteq F_P(S)$, in the following way: for any atom ξ , $\neg\xi \in M$ if $\text{not } \xi \in S$, $\xi \in M$ if $P \cup S \vdash \xi$, and ξ is *undefined* otherwise. An *answer set* E is defined by a fixpoint S such that $F_P(S) = S$ and $E = \{\xi \in \mathcal{L} \mid P \cup S \vdash \xi\}$.

3 Goal Rewrite Systems

We introduce goal rewrite systems as formulated in [Lin and You, 2001].

A goal rewrite system is a rewrite system that consists of three types of rewrite rules: (1) Program rules from $\text{Comp}(P)$ for literal rewriting; (2) Simplification rules to transform and simplify goals; and (3) Loop rules for handling loops.

A *program rule* is a completed definition $\phi \leftrightarrow B_1 \vee \dots \vee B_n \in \text{Comp}(P)$ used from left to right: ϕ can be rewritten to $B_1 \vee \dots \vee B_n$, and $\neg\phi$ to $\neg B_1 \wedge \dots \wedge \neg B_n$. These are called *literal rewriting*.

A *goal*, also called a *goal formula*, is a formula which may involve $\neg$, $\vee$ and $\wedge$. A goal resulted from a literal rewriting from another goal is called a *derived goal*. Like a formula, a goal may be transformed to another goal without changing its semantics. This is carried out by simplification rules.

We assume that in all goals negation appears only in front of a literal. This can be achieved by simple transformations using the following rules: for any formulas Φ and Ψ ,

$$\neg\neg\Phi \rightarrow \Phi; \neg(\Phi \vee \Psi) \rightarrow \neg\Phi \wedge \neg\Psi; \neg(\Phi \wedge \Psi) \rightarrow \neg\Phi \vee \neg\Psi$$

3.1 Simplification rules

The simplification rules constitute a nondeterministic transformation system formulated with a mechanism of loop handling in mind, which requires keeping track of literal sequences $g_0, \dots, g_n$ where each g_i , $0 < i \leq n$, is in the goal

formula resulted from rewriting g_{i-1} . Two central mechanisms in formalizing goal rewrite systems are *rewrite chains* and *contexts*.

- **Rewrite Chain:** Suppose a literal l is written by its definition $\phi \leftrightarrow \Phi$ where $l = \phi$ or $l = \neg\phi$. Then, each literal l' in the derived goal is generated in order to prove l . This ancestor-descendant relation is denoted $l \prec l'$. A sequence $l_1 \prec \dots \prec l_n$ is then called a *rewrite chain*, abbreviated as $l_1 \prec^+ l_n$.
- **Context:** A rewrite chain $g = g_0 \prec g_1 \prec \dots \prec g_n = T$ records a set of literals $C = \{g_0, \dots, g_{n-1}\}$ for proving g . We will write $T(\{g_0, \dots, g_{n-1}\})$ and call C a *context*. A context will also be used to maintain consistency: if g can be proved via a conjunction, all of the conjuncts need be proved with contexts that are non-conflicting with each other. For simplicity, we assume that whenever $\neg F$ is generated, it is automatically replaced by $T(C)$, where C is the set of literals on the corresponding rewrite chain, and $\neg T$ is automatically replaced by F .

Note that for any literal in a derived goal, the rewrite chain leading to it from a literal in the given goal is uniquely determined. As an example, suppose the completion of a program has the definitions: $a \leftrightarrow \neg b \wedge \neg c$ and $b \leftrightarrow q \vee \neg p$. Then, we get a rewrite sequence, $a \rightarrow \neg b \wedge \neg c \rightarrow \neg q \vee p \wedge \neg c$. For the three literals in the last goal, we have rewrite chains from a : $a \prec \neg b \prec \neg q$; $a \prec \neg b \prec p$; and $a \prec \neg c$.

Simplification Rules: Let Φ and Φ_i be goal formulas, C be a context, and l a literal.

- SR1. $F \vee \Phi \rightarrow \Phi$ SR1'. $\Phi \vee F \rightarrow \Phi$
- SR2. $F \wedge \Phi \rightarrow F$ SR2'. $\Phi \wedge F \rightarrow F$
- SR3. $T(C_1) \wedge T(C_2) \rightarrow T(C_1 \cup C_2)$ if $C_1 \cup C_2$ is consistent
- SR4. $T(C_1) \wedge T(C_2) \rightarrow F$ if $C_1 \cup C_2$ is inconsistent
- SR5. $\Phi_1 \wedge (\Phi_2 \vee \Phi_3) \rightarrow (\Phi_1 \wedge \Phi_2) \vee (\Phi_1 \wedge \Phi_3)$
- SR5'. $(\Phi_1 \vee \Phi_2) \wedge \Phi_3 \rightarrow (\Phi_1 \wedge \Phi_3) \vee (\Phi_2 \wedge \Phi_3)$ $\square$

SR3 merges two contexts if they contain no complementary literals, otherwise SR4 makes it a failure to prove. Repeated applications of SR5 and SR5' can transform any goal formula to a disjunctive normal form (DNF).

3.2 Loop rules

After a literal l is rewritten, it is possible that at some later stage either l or $\neg l$ appears again in a goal on the same rewrite chain. Two rewrite rules are formulated to handle loops.

Definition 3.1 Let $S = l_1 \prec^+ l_n$ be a rewrite chain.

- If $\neg l_1 = l_n$ or $l_1 = \neg l_n$, then S is called an *odd loop*.
- If $l_1 = l_n$, then
 - S is called a *positive loop* if l_1 and l_n are both atoms and each literal on $l_1 \prec^+ l_n$ is also an atom;
 - S is called a *negative loop* if l_1 and l_n are both negative literals and each literal on $l_1 \prec^+ l_n$ is also negative;
 - Otherwise, S is called an *even loop*.

In all the cases above, l_n is called a loop literal.

Loop Rules: Let $g_1 \prec^+ g_n$ be a rewrite chain.

- LR1. $g_n \rightarrow F$
if $g_i \prec^+ g_n$, for some $1 \leq i < n$, is a positive loop or an odd loop.
- LR2. $g_n \rightarrow T(\{g_1, \dots, g_n\})$
if $g_i \prec^+ g_n$, for some $1 \leq i < n$, is a negative loop or an even loop. $\square$

A *rewrite sequence* is a sequence of zero or more rewrite steps $Q_0 \rightarrow \dots \rightarrow Q_k$, denoted $Q_0 \rightarrow^* Q_k$, such that Q_0 is an initial goal, and for each $0 \leq i < k$, Q_{i+1} is obtained from Q_i by

- literal rewriting at a non-loop literal in Q_i , or
- applying a simplification rule to a subformula of Q_i , or
- applying a loop rule to a loop literal in Q_i .

Example 3.2 For the program given in the Introduction, $P_0 = \{a \leftarrow \text{not } b. \ b \leftarrow c, \text{not } a. \ c \leftarrow a.\}$, a is proved but b is not. This is shown by the following rewrite sequences:

$$\begin{aligned} a &\rightarrow \neg b \rightarrow \neg c \vee a \rightarrow \neg a \vee a \rightarrow F \vee a \rightarrow a \rightarrow T(\{a, \neg b\}) \\ b &\rightarrow c \wedge \neg a \rightarrow a \wedge \neg a \rightarrow \neg b \wedge \neg a \rightarrow F \wedge \neg a \rightarrow F \end{aligned}$$

Let $P_1 = \{b \leftarrow \text{not } c. \ c \leftarrow c.\}$. b is proved and $\neg b$ is not.

$$b \rightarrow \neg c \rightarrow \neg c \rightarrow T(\{\neg b, \neg c\}); \quad \neg b \rightarrow c \rightarrow c \rightarrow F$$

3.3 Previous results

In [Lin and You, 2001], it is shown that (1) a goal rewrite system is a canonical system, i.e., it confluent and terminating; (2) any goal rewrite system is sound and complete w.r.t. the partial stable model semantics; and (3) the rewriting framework can be extended for abduction in a relatively straightforward manner.

4 Goal Rewrite Systems with Computed Rules

We first use two examples to illustrate the main technical results of this section.

Example 4.1 Given a rewrite system R^0 , suppose we have a rewrite sequence $\neg q \rightarrow a \rightarrow a \rightarrow F$. The failure is due to a positive loop on a . We may recycle the computed answer by replacing the rewrite rule for $\neg q$ by the new rule, $\neg q \rightarrow F$. We thus get a new system, say R^1 . Suppose in trying to prove g we have

$$g \rightarrow a \rightarrow \neg q \rightarrow F$$

where the last step makes use of the computed answer for $\neg q$. The question arises as whether this way of using previously computed results guarantees the soundness and completeness. Theorem 4.6 to be proved later in this paper answers this question positively. To see it for this example, assume we have the following, successful proof in R^0

$$g \rightarrow a \rightarrow \neg q \rightarrow a \rightarrow T(\{g, a, \neg q\})$$

where the termination is due the even loop on a . Had such a sequence existed, recycling would have produced a wrong

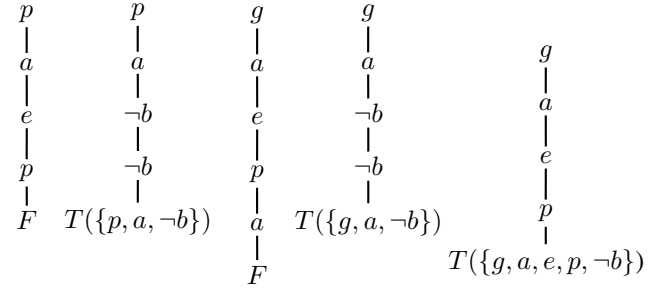


Figure 1: Recycling may generate extra proofs

result. However, one can see that the existence of the rewrite step $a \rightarrow \neg q$ implies the existence of a different way to prove $\neg q$:

$$\neg q \rightarrow a \rightarrow \neg q \vee \dots \rightarrow T(\{\neg q, a\}) \vee \dots$$

contradicting that $\neg q$ was rewritten to F in R^0 .

Before giving the next example, we introduce a different way to understand rewrite sequences. Since any goal formula can always be transformed to a DNF using the distributive rules SR5 and SR5', and the order of rewriting does not matter, we can view rewriting as generating a sequence of DNFs. Thus, a rewrite sequence in DNF from an initial goal g ,

$$g \rightarrow^* N_1 \vee \dots \vee N_n$$

can be conveniently represented by *derivation trees*, or *d-trees*, one for each N_i representing one possible way of proving g . For any i , the d-tree for N_i has g as its root node, wherein a branch from g to a leaf node corresponds to a rewrite chain from g that eventually ends with an F or some $T(C)$. As such a disjunct is a conjunction, a successful proof requires each branch to succeed and the union of all resulting contexts to be consistent.

The next example is carefully constructed to illustrate that recycling may not yield the same answers as if no recycling were carried out. In particular, one can sometimes get additional answers.

Example 4.2 Consider the program:

$$\begin{aligned} g &\leftarrow a. \quad a \leftarrow \text{not } b. \quad a \leftarrow e. \\ b &\leftarrow b. \quad e \leftarrow p. \quad p \leftarrow a. \end{aligned}$$

In Fig. 1, each d-tree consists of a single branch. The left two d-trees are expanded from goal p corresponding to the following rewrite sequence:

$$\begin{aligned} p &\rightarrow a \rightarrow e \vee \neg b \rightarrow p \vee \neg b \rightarrow F \vee \neg b \\ &\rightarrow \neg b \rightarrow \neg b \rightarrow T(\{p, a, \neg b\}) \end{aligned}$$

The next two d-trees are for goal g , corresponding to the rewrite sequence:

$$\begin{aligned} g &\rightarrow a \rightarrow e \vee \neg b \rightarrow p \vee \neg b \rightarrow a \vee \neg b \\ &\rightarrow F \vee \neg b \rightarrow \neg b \rightarrow \neg b \rightarrow T(\{g, a, \neg b\}) \end{aligned}$$

Now, we recycle the proof for p in the proof for g and compare it with the one without recycling. Clearly, the successful

d -tree for g (the fourth from the left) will still succeed as it doesn't involve any p . The focus is then on the d -tree in the middle, in particular, the node p in it; this d -tree fails when no recycling was performed.

Since p is previously proved with context $\{g, a, \neg b\}$, recycling of this proof amounts to terminating p with a context which is the union of this context with the rewrite chain leading to p (see the d -tree on the right). But this results in a successful proof that fails without recycling.

Though recycling appears to have generated a wrong result, one can verify that both generated contexts, $\{g, a, \neg b\}$ and $\{g, a, e, p, \neg b\}$, belong to the same partial stable model. Thus, recycling in this example didn't lead to an incorrect answer but generated a redundant one. Theorem 4.6 shows that this is not incidental. Indeed, if p is true in a partial stable model, by derivation (look at the d -tree in the middle), so must be e , a , and g .

4.1 Rewrite systems with computed rules

Given a goal rewrite system R , we may denote a rewrite sequence from a literal g by $g \rightarrow_R E$.

Definition 4.3 (Computed rule)

Let R be a goal rewrite system in which literal p is rewritten to its normal form. The computed rule for p is defined as: If $p \rightarrow_R F$, the computed rule for p is the rewrite rule $p \rightarrow F$; if $p \rightarrow_R T(C_1) \vee \dots \vee T(C_n)$, then the computed rule for p is the rewrite rule $p \rightarrow T(C_1) \vee \dots \vee T(C_n)$.

For the purpose of recycling, a computed rule $p \rightarrow E$ is meant to replace the existing literal rewrite rule for p . If a computed rule is of the form $p \rightarrow F$ representing a failed derivation, it can be used directly as the literal rewrite rule for p . Otherwise, we must combine the contexts in E with the rewrite chain leading to p , and keep only consistent ones.

Recycling Rule:

Let $g_1 \prec^+ g_n$ be a rewrite chain where g_n is a non-loop literal. Let $G = \{g_1, \dots, g_n\}$, and $g_n \rightarrow T(D_1) \vee \dots \vee T(D_k)$ be the computed rule for g_n . Further, let $\{D'_1, \dots, D'_{k'}\}$ be the subset of $\{D_1, \dots, D_k\}$ containing any D_i such that $D_i \cup G$ is consistent. Then, the recycling rule for g_n is defined as:

RC. $g_n \rightarrow T(G \cup D'_1) \vee \dots \vee T(G \cup D'_{k'})$ $\square$

In the sequel, a rewrite system includes the recycling rule as well as zero or more computed rules. We note that the termination and confluence properties remain to hold for the extended systems.

We are interested in the soundness and completeness of a series of rewrite systems, each of which recycles computed answers generated on the previous one. For this purpose, given a program P we use R_P^0 to denote the original goal rewrite system where literal rewrite rules are defined by the Clark completion of P . For all $i \geq 0$, R_P^{i+1} is defined in terms of R_P^i as follows: Let Δ_i be the set of computed rules (generated) on R_P^i for the set of literals $\mathcal{L}_{\Delta_i}$. Then, R_P^{i+1} is the rewrite system obtained from R_P^i by replacing the rewrite rules for the literals in $\mathcal{L}_{\Delta_i}$ by those in Δ_i . In the rest of this section, we will always refer to a fixed program P . Thus we may drop the subscript P and write R^i .

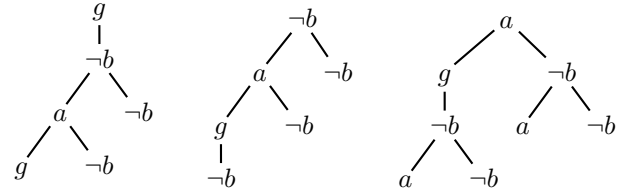


Figure 2: Loop rotation

Definition 4.4 A rewrite system R^i is sound iff, for any literal g and rewrite sequence $g \rightarrow_{R^i} T(C_1) \vee \dots \vee T(C_n)$, and for each C_j , $j \in [1..n]$, there exists a partial stable model M of P such that $g \in C_j \subseteq M$. R^i is complete iff, for any literal g such that $g \in M$ for some partial stable model M of P , there is a rewrite sequence $g \rightarrow_{R^i} T(C_1) \vee \dots \vee T(C_n)$ such that for some C_j , $j \in [1..n]$, $g \in C_j \subseteq M$.

An important property of provability by rewriting is the so-called *loop rotation*, which is needed in order to prove the completeness of recycling; namely, a proof (a successful branch in a d -tree) terminated by a loop rule can be captured in rotated forms.

To describe this property, we need the following notation about rewrite chains: Any direct dependency relation $l \prec l'$ may be denoted by $l \cdot l'$, and we allow a segment (which may be empty) of a rewrite chain to be denoted by a Greek letter such as δ , θ , and ξ . Thus, we may write $x \cdot \delta \cdot y$ to denote a rewrite chain from x to y via δ , or $x \cdot \delta$ to mean a rewrite chain that begins with x followed by the segment denoted by δ . A rewrite chain may also be used to denote the set of the literals on it.

Lemma 4.5 (loop rotation)

Let R^0 be a rewrite system without computed rules. Let Tr be a d -tree for literal g that succeeds with context C . Suppose a branch of Tr ends with a loop, $g \cdot \theta \cdot g$, for some θ . Then, for any literal $l \in \theta$, there is a proof of l that succeeds with the same context C .

Proof. A loop, $\pi = g \cdot l_1 \cdot l_2 \cdot \dots \cdot l_n \cdot g$, where g and l_i are literals, can always be rotated as

$$l_1 \cdot l_2 \cdot \dots \cdot l_n \cdot g \cdot l_1, \quad l_2 \cdot \dots \cdot l_n \cdot g \cdot l_1 \cdot l_2, \quad \dots$$

and so on, so that if π is a negative loop (or an even loop, resp.) so is its rotated loop. Rotation over a d -tree can be performed as follows: remove the top node n , and for any link from the top node, $n \cdot q$, attach the link $n \cdot q$ to any occurrence of n . The assumption of the existence of loop $g \cdot \theta \cdot g$ ensures that in every round of rotation there is at least one occurrence of the top node. (See Fig 2 for an illustration where rotation proceeds from left to right.) It can be seen that the type of a loop is always preserved and the set of literals on the tree remains unchanged. $\square$

4.2 Soundness and completeness of recycling

Theorem 4.6 For any $i \geq 0$, R^i is sound and complete.

We sketch how this can be proved. We can prove the claim by induction on i . R^0 , the system without computed rules, is sound and complete [Lin and You, 2001]. Now assume for all

j with $0 \leq j \leq i$, R^j are sound and complete, and show that R^{i+1} is also sound and complete.

We only need to consider the situations where rewriting in R^{i+1} differs from that of R^i . Let $\mathcal{L}_{\Delta_i}$ be the set of literals whose computed rules are generated on R^i . We can first carry out rewriting without rewriting the literals that are in $\mathcal{L}_{\Delta_i}$. In this case, rewriting from g in both R^i and R^{i+1} terminate at the same expression, which is either F or a DNF, say $N_1 \vee \dots \vee N_m$. Each N_i can be represented by a d-tree.

For the soundness, we assume $g \rightarrow_{R^{i+1}} T(D_1) \vee \dots \vee T(D_s)$. For any $D \in \{D_1, \dots, D_s\}$, we need to show that there is a partial stable model M such that $D \subseteq M$. Consider the d-tree Tr that generates D and suppose g is its root node. We can show inductively in a bottom-up fashion that all the literals on Tr belong to the same partial stable model.

For the completeness, we can show that for any context generated in R^i , the same context will be generated in R^{i+1} . Then, R^{i+1} is complete simply because R^i is complete.

Let $p \in \mathcal{L}_{\Delta_i}$, and consider a proof of g in R^i that goes through p . In particular, consider a d-tree of this proof that contains p . Since each branch of this d-tree can be expanded and eventually terminated independent of others, for simplicity, we consider how a branch $g \cdot \xi \cdot p$ in the d-tree is extended. In R^{i+1} the computed rule for p is used while in R^i it is not. We only need to consider two cases in R^i : either g is proved via p and a previously computed rule, or the branch is terminated due to a loop. Here, let us consider the latter only. In expanding the rewrite chain $g \cdot \xi \cdot p$ in R^i , we may form a loop, say $g \cdot \xi \cdot p \cdot \xi'$. If the loop is in ξ' , exactly the same loop occurs in rewriting p as the top goal in R^i , so it is part of the computed rule for p . Otherwise it is a loop that *crosses over* p , in the general form

$$\pi = g \cdot \theta_1 \cdot l \cdot \theta_2 \cdot p \cdot \theta_3 \cdot l$$

where l is the loop literal. As a special case of loop rotation over a branch (cf. Lemma 4.1), the same way of terminating a rewrite chain presents itself in proving p as the top goal in R^i , which is

$$\pi' = p \cdot \theta_3 \cdot l \cdot \theta_2 \cdot p.$$

If the loop on π is a negative loop (or an even loop, resp.), so is π' . Thus the same context will be generated in R^{i+1} .

As given in the corollary below, if we only recycle failed proofs then exactly the same contexts will be generated.

Corollary 4.7 *Let R^i be a rewrite system where each computed rule is of the form $p \rightarrow F$. Let g be a literal and E be a normal form. Then, for any $i \geq 0$, $g \rightarrow_{R^0} E$ iff $g \rightarrow_{R^i} E$.*

5 Recycling in Abductive Rewrite Systems

As shown in [Lin and You, 2001], the rewriting framework can be extended to abduction in a straightforward way: the only difference in the extended framework is that we do not apply the Clark completion to abducibles. That is, once an abducible appears in a goal, it will remain there unless it is eliminated by the simplification rule $SR2$ or $SR2'$. In a similar way, the goal rewrite systems with computed rules in the previous section can be extended to abduction as well. Once a

goal is rewritten to a disjunction of conjunctions of abducible literals and T :

$$p \rightarrow_R [l_{11}(C_{11}) \wedge \dots \wedge l_{1k_1}(C_{1k_1})] \vee \dots \vee [l_{m1}(C_{m1}) \wedge \dots \wedge l_{mk_m}(C_{mk_m})]$$

where each l_{ij} is either T or an abducible literal, and $C_{i1} \cup \dots \cup C_{ik_i}$ is consistent for each i , then this result can be recycled.

6 Experiments

We have implemented a depth-first search rewrite procedure with branch and bound. The procedure can be used to compute explanations using a nonground program, under the condition that in each rule a variable that appears in the body must also appear in the head. When this condition is not satisfied, one only needs to instantiate those variables that only appear in the body of a rule. This is a significant departure from the approaches that are based on ground computation where a function-free program is first instantiated to a ground program with which the intended models are then computed.

To check the effectiveness of the idea of recycling, we tested our system on the logistics problem in [Lin and You, 2001]. This is a domain in which there is a truck and a package. A package can be in or outside a truck, and a truck can be moved from one location to another. The problem is that given state constraints such as that the truck and the package can each be at only one location at any given time, and that if the package is in the truck, then when the truck moves to a new location, so does the package, how we can derive a complete specification of the effects of the action of moving a truck from one location to another. Suppose that we have the following propositions: $ta(x)$ ($pa(x)$) – the truck (package) is at location x initially; in – the package is in the truck initially; $ta(x, y, z)$ ($pa(x, y, z)$) – the truck (package) is at location x after performing the action of moving it from y to z ; $in(y, z)$ – the package is in the truck after performing the action of moving the truck from y to z . Then in [Lin and You, 2001], the problem is solved by computing the abduction of successor state propositions $\{ta(x, y, z), pa(x, y, z), in(y, z)\}$ in terms of initial state propositions $\{ta(x), pa(x), in\}$ (abducibles) using the following logic program (see [Lin and You, 2001] for more details):

$$ta(X, X1, X). \quad (2)$$

$$pa(X, X1, X2) \leftarrow ta(X, X1, X2), in(X1, X2). \quad (3)$$

$$ta(X, X1, X2) \leftarrow X \neq X2, ta(X), \text{not } taol(X, X1, X2). \quad (4)$$

$$taol(X, X1, X2) \leftarrow Y \neq X, ta(Y, X1, X2). \quad (5)$$

$$pa(X, X1, X2) \leftarrow pa(X), \text{not } paol(X, X1, X2). \quad (6)$$

$$paol(X, X1, X2) \leftarrow Y \neq X, pa(Y, X1, X2). \quad (7)$$

$$in(X, Y) \leftarrow in. \quad (8)$$

Here the variables are to be instantiated over a domain of locations. For instance, given query $pa(3, 2, 3)$, our system would compute its abduction as $pa(3) \vee in$, meaning that for it to be true, either the package was initially at 3 or it was inside the truck.

Which subgoals to compute first? If we want to compute

the abduction of all propositions, without the framework of recycling introduced here, the only way is to compute them one by one independently. With the idea of recycling, we can try to recycle previously computed results. The question is then which goals to compute first. This question arises even if we just want to compute the abduction of a single goal: instead of computing it using the original program, it may sometimes be better if we first compute the abduction of some other goals and recycle the result.

A simple solution is to find out the dependency relations among the propositions: a proposition p depends on q if either q is in the body of a rule whose head is p or inductively, there is another proposition r such that p depends on r , and r depends on q .

For our program above, it is easy to see that among the propositions that we want to compute their abduction of, those of the form $\text{in}(X, Y)$ and those of the form $\text{ta}(X, Y, Z)$ are independent of each other, but those of the form $\text{pa}(X, Y, Z)$ are depended on propositions of the forms $\text{in}(X, Y)$ and $\text{ta}(X, Y, Z)$. So we should compute first the abduction of $\text{in}(X, Y)$ and $\text{ta}(X, Y, Z)$. Now $\text{in}(X, Y)$ is solved by rule (8), $\text{ta}(X, Y, X)$ by rule (2), and as it turned out, when $X \neq Z$, $\text{ta}(X, Y, Z)$ is always false, and its computation is relatively easy. For instance, for the domain with 9 locations, query $\text{ta}(7, 1, 6)$ took only 2.6 seconds. In comparison, query $\text{pa}(7, 1, 7)$ took more than 7000 seconds without recycling.

Table 1 contains run time data for some representative queries.¹ For comparison purpose, each query is given two entries: the one under “NR” refers to regular rewriting system without using recycling, and the one under “WR” refers to rewriting system using computed rules about $\text{ta}(X, Y, Z)$. As one can see, especially for hard queries like $\text{pa}(7, 1, 7)$, recycling in this case significantly speeds up the computation.

7 Concluding remarks

We have considered the problem of how to reuse previously computed results for answering other queries in the abductive rewriting system of Lin and You [2001] for logic programs with negation, and showed that this can indeed be done. We have also described a methodology of using the recycling system in practice by analysing the dependency relationship among propositions in a logic programs. We applied this methodology to the problem of computing the effect of actions in a logistics domain, the same one considered in [Lin and You, 2001], and our experimental results showed that recycling in this domain can indeed result in good performance gain. For future work we want to implement a system that can automatically analyse a program and decide how best to recycle previous computations.

¹Our implementation was written in Sicstus Prolog, and the experiments were done on a PIII 1GHz notebook with 512 MB memory. For generating explanations for regular rewriting system, our implementation is a significant improvement over the one in [Lin and You, 2001]. For instance, for a domain with 7 locations query $\text{pa}(3, 2, 3)$ took more than 20 minutes for the implementation reported in [Lin and You, 2001], but required less than 1 second under our implementation running on a comparable machine.

Query	9 locations		10 locations	
	NR	WR	NR	WR
$\text{pa}(1, 2, 3)$	0.71	0.41	1.50	0.89
$-\text{pa}(1, 2, 3)$	75.89	2.28	342.96	5.65
$\text{pa}(3, 2, 3)$	137.05	0.89	630.69	1.98
$-\text{pa}(3, 2, 3)$	2.97	1.98	7.64	5.03
$\text{pa}(1, 5, 7)$	122.87	0.75	278.07	1.31
$-\text{pa}(1, 5, 7)$	727.6	7.07	2534.09	19.08
$\text{pa}(7, 5, 1)$	108.66	17.82	188.50	30.72
$-\text{pa}(7, 5, 1)$	74.43	2.26	340.51	5.64
$\text{pa}(7, 1, 7)$	7619.72	20.78	29140.69	35.65
$-\text{pa}(7, 1, 7)$	2.98	2.01	7.71	5.05

Table 1: Recycling in logistics domain. Legends: NR - no recycling; WR - recycling $\text{ta}(X, Y, Z)$ goals. All times are in CPU seconds.

References

- [Dershowitz and Jouannaud, 1990] N. Dershowitz and P. Jouannaud. Rewrite systems. In *Handbook of Theoretical Comp. Sci., Vol B: Formal Methods and Semantics*. North-Holland, 1990.
- [Dix, 1991] J. Dix. Classifying semantics of logic programs. In *Proc. First Workshop on LPNMR*, pages 167–180, 1991.
- [Dung, 1991] P. Dung. Negations as hypothesis: An abductive foundation for logic programming. In Koichi Furukawa, editor, *8th ICLP*. MIT press, 1991.
- [Eshghi and Kowalski, 1989] K. Eshghi and R.A. Kowalski. Abduction compared with negation by failure. In *Proc. 6th ICLP*, pages 234–254. MIT Press, 1989.
- [Gelfond and Lifschitz, 1988] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proc. 5th Int’l Conference on Logic Programming*, pages 1070–1080. MIT Press, 1988.
- [Kakas and Mancarella, 1990] A. Kakas and P. Mancarella. Generalized stable models: a semantics for abduction. In *Proc. 9th European Conference on Artificial Intelligence*, pages 285–291, 1990.
- [Kakas et al., 2000] A. Kakas, A. Michael, and C. Mourlas. ACLP: abductive constraint logic programming. *J. Logic Programming*, 44(1-3):129–178, 2000.
- [Lin and You, 2001] F. Lin and J. You. Abduction in logic programming: a new definition and an abductive procedure based on rewriting. In *Proc. IJCAI’01*, pages 655–661, 2001. The full paper appears in *Artificial Intelligence* 140(1/2): 175-205 (2002).
- [Przymusinski, 1990] T.C. Przymusinski. Extended stable semantics for normal and disjunctive logic programs. In *Proc. 7th ICLP*, pages 459–477. MIT Press, 1990.
- [Satoh and Iwayama, 1992] K. Satoh and R. Iwayama. A query evaluation method for abductive logic programming. In *Proc. JICSLP’92*, pages 671–685, 1992.
- [You and Yuan, 1995] J. You and L. Yuan. On the equivalence of semantics for normal logic programs. *JLP*, 22:212–221, 1995.

ONTOLOGIES AND FOUNDATIONS

What is Artificial Intelligence? Psychometric AI as an Answer

Selmer Bringsjord & Bettina Schimanski

selmer@rpi.edu • schimb@rpi.edu

Department of Computer Science

Department of Cognitive Science

Rensselaer AI & Reasoning (RAIR) Lab

Rensselaer Polytechnic Institute (RPI)

Troy NY 12180 USA

Abstract

We propose an answer to the “What is AI?” question, namely, that AI is really (or at least really ought in significant part to be) *Psychometric AI* (PAI). Along the way, we: set out and rebut five objections to PAI; describe PERI, a robot in our lab who exemplifies PAI; and briefly treat the future of Psychometric AI, first by pointing toward some promising PAI-based applications, and then by raising some of the “big” philosophical questions the success of Psychometric AI will raise.

1 Introduction

What exactly is AI? We’d be willing to wager that many of you have been asked this question — by colleagues, reporters, friends and family, and others. Even if by some fluke you’ve dodged the question, perhaps you’ve asked it yourself, maybe even perhaps (in secret moments, if you’re a practitioner) to yourself, without an immediate answer coming to mind. At any rate, AI *itself* repeatedly asks the question — as the first chapter of many AI textbooks reveals. In this paper we want to propose an answer, namely, that AI is really (or at least really ought in significant part to be) *Psychometric AI* (sometimes just ‘PAI’ (rhymes with π) for short). We also want to tell you something about both PERI, a robot in our lab who exemplifies PAI, and the future we envision for PAI.

Our plan herein is as follows. In the next section, 2, we answer the “What is AI?” question from the standpoint of Psychometric AI, and introduce some of the tests at the heart of this brand of AI. In section 3 we rebut some objections that will inevitably be brought against Psychometric AI. The rebuttal to the first of these objections will reveal the foundation for PAI: the Turing Test (TT) and its more demanding cousin, the Total TT (TTT). In section 4 we introduce you to PERI. Our penultimate section briefly treats the future of Psychometric AI, first by pointing toward some promising PAI-based applications, and then by raising some of the “big” philosophical questions the success of Psychometric AI will raise. We end by addressing a second round of objections, formulated by those who read earlier versions of the present paper.

2 What is AI? Psychometric AI as an Answer

Presumably the ‘A’ part of ‘AI’ isn’t the challenge: We seem to have a fairly good handle on what it means to say that something is an artifact, or artificial. (We can ignore here conundrums arising from self-reproducing systems, systems that evolve without human oversight, etc.) It’s the ‘I’ part that seems to throw us for a bit of a loop. What’s intelligence? *This* is the big, and hard, question. Innumerable answers have been given, but most thinkers seem to forget that there is a particularly clear and straightforward answer available, courtesy of the field that has sought to operationalize the concept in question; that field is psychometrics. Psychometrics is devoted to systematically measuring psychological properties, usually via tests. These properties include the one most important in the present context: intelligence. In a nutshell, then, the initial version of our account of intelligence is this: *Some agent is intelligent if and only if it excels at all established, validated tests of intelligence.* (This account is inadequate, for reasons we explain below before supplanting it with a more sophisticated one.) AI then reduces to Psychometric AI: the field devoted to building a computational system able to score well on such tests. This may strike you as a preposterously narrow definition of AI. The first step (in a series taken as this paper unfolds) in diffusing this attitude is to take a look at some intelligence tests, some of which, we surmise, are a good deal richer than you might at present think.

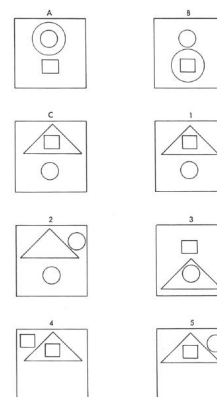


Figure 1: Sample Problem Solved by Evan’s (1968) ANALOGY Program. *A is to B as C is to ...?*

In the early days of AI, Psychometric AI was at least implicitly entertained. After all, in the mid 1960s, the largest Lisp program on earth was Evans' 1968 ANALOGY program, which could solve problems like those shown in Figure 1. Evans himself predicted that systems able to solve such problems would "be of great practical importance in the near future," and he pointed out that performance on such tests is often regarded to be the "touchstone" of human intelligence. Unfortunately, ANALOGY simply hasn't turned out to be the first system in a longstanding, comprehensive research program: after all, we find ourselves, at present, trying to start that very program. What went wrong? Well, certainly Psychometric AI would be patently untenable if the tests upon which it is based consist solely of geometric analogies. This point is entailed by such observations as this one from Fischler & Firschein 1987:

If one were offered a machine purported to be intelligent, what would be an appropriate method of evaluating this claim? The most obvious approach might be to give the machine an IQ test. . . . However, [good performance on tasks seen in IQ tests would not] be completely satisfactory because the machine would have to be specially prepared for any specific task that it was asked to perform. The task could not be described to the machine in a normal conversation (verbal or written) if the specific nature of the task was not already programmed into the machine. Such considerations led many people to believe that the ability to communicate freely using some form of natural language is an essential attribute of an intelligent entity. (Fischler & Firschein, 1987], p. 12)

Unfortunately, while this quote helps explain why ANALOGY in and of itself didn't ignite a research program to drive AI, Fischler & Firschein apparently are familiar with only what we call **narrow**, as opposed to **broad**, intelligence tests. Arguably, this distinction goes back to Descartes' (descartes.haldane.ross.vol1, p. 116) claim that while a machine could in the future pass any test for a particular mental power, no machine could pass a test for any mental power whatsoever. This rather speculative claim can be seen to be cashed out in two different and longstanding views of intelligence within psychology: Thurstone's 1938 and Spearman's 1927. In Thurstone's view (put barbarically), intelligence consists in the capacity to solve a *broad* range of problems, e.g., verbal analogies, geometric analogies, digit recall, story understanding, commonsense reasoning, arithmetical calculation, and so on. In Spearman's view (again, put roughly), intelligence is a specific, *narrow*, underlying capacity (notoriously) referred to as *g*, summoned up to the highest degree when solving highly focused and abstract problems like those ANALOGY solved. The most famous set of "*g*-relevant" problems is the tightly guarded and much-used Raven's 1962 Progressive Matrices, or just 'RPM.' An example of a problem from RPM is shown in Figure 2, which is taken from [Carpenter *et al.*, 1990]. As part of PERI, we have built theorem prover-based agents able to infallibly crack not only geometric analogies, but RPM items they have never seen before (Figure 2 shows part of an OTTER [Wos *et al.*, 1992] proof that serves to identify the solution). (The algorithms deployed by these agents were devised as part of contracted research for The Educational Testing Service, or ETS.) It is much harder

to build agents able to solve broad tests of intelligence ETS tests that include sub-tasks demanding the kinds of communicative capacities Fischler & Firschein have in mind.

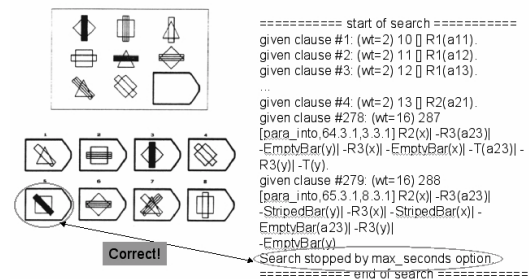


Figure 2: Simple RPM Problem "Cracked" by RAIR Lab's PERI

Psychological Corporation's popular WAIS (Wechsler Adult Intelligent Scale) is a paradigmatic example of a broad intelligence test that includes the full array of "Thurstonean" sub-tests (the complete array is enumerated in Baron 2000). While we don't have enough space to explain all these sub-tests, we point out that Fischler & Firschein's criticism of simplistic versions of Psychometric AI certainly evaporates in the face of the WAIS. That this is so follows from the sub-test on the WAIS known as "Comprehension," in which, in ordinary conversation, subjects are asked fiendishly tricky "general knowledge" questions. For example, examinees might be asked to explain why the tires on automobiles are made of rubber, rather than, say, plastic. Perhaps you'll agree that such a question would make a nice challenge to any system purported to have commonsense intelligence. Were CYCORP (<http://www.cyc.com>) to herald a system having general, common-sense intelligence, the WAIS would be a vehicle for verification. There are other sub-tests on the WAIS that are just as challenging.

With help from additional researchers in the RAIR Lab, we are in the process of "cracking" the WAIS, by way of the design and construction of PERI. The sub-test we have cracked most recently is "Block Design." PERI, when given any configuration of blocks in the space of all possible ones, deduces the solution in under a second of CPU time, and proceeds to assemble this solution with its manipulator/gripper. In section 4 we provide some details about PERI's exploits. (Readers wanting to peek ahead should see Figures 4, 5, 6, and 7.)

At present we are tackling the much more difficult sub-test "Picture Arrangement," which requires of examinees that they arrange jumbled snapshots to form coherent stories. (For readers wanting to look ahead, a home-grown example is shown in Figure 3. For legal reasons, actual WAIS examples cannot be shown.) Here our research attempts to make use of prior work in story generation (e.g., Bringsjord & Ferrucci 2000).

We anticipate that some will insist that intelligence tests, even broad ones, are still just too narrow, when put in the context of the full array of cognitive capacities seen in *homo sapiens*. Well, we agree! But we are understanding intelligence, from the standpoint of psychometrics, to include many varied

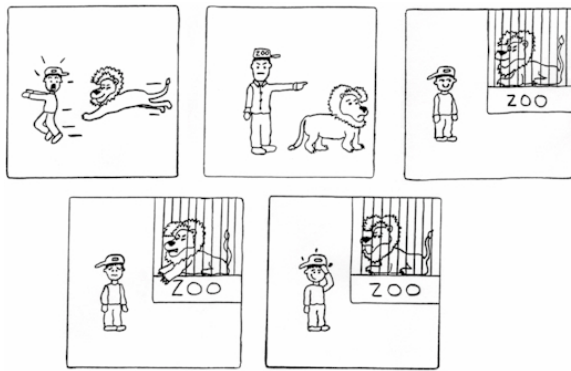


Figure 3: Examinees must arrange to make a coherent story.

tests of intellectual ability. Accordingly, we now move to a less naive definition of PAI:

Psychometric AI is the field devoted to building information-processing entities capable of at least solid performance on all established, validated tests of intelligence and mental ability, a class of tests that includes not just the rather restrictive IQ tests, but also tests of artistic and literary creativity, mechanical ability, and so on.

This definition replaces the old, provisional one: the new definition, when referring to tests of mental ability, is referring to much more than IQ tests. For example, following Sternberg, someone with much musical aptitude would count as brilliant even if their scores on tests of “academic” aptitude (e.g., on the SAT, GRE, LSAT, etc.) were low. But specifically what sorts of additional tests would be involved? We don’t have space to canvass the myriad tests that psychometricians have validated. To give a quick sense of how latitudinarian (and therefore challenging) Psychometric AI is intended to be, we mention The Torrance Tests of Creative Thinking [Torrance, 1990; 1988]. This test comes in both “visual” and “verbal” forms. In the visual form, test takers are asked to draw pictures (often by enriching existing sketches); in the verbal form, test takers are asked to write — creatively. For example, one of the activities subjects engage in on the verbal test is the following.

Most people throw their tin cans away, but they have thousands of interesting and unusual uses. In the spaces below and on the next page, list as many of these interesting and unusual uses as you can think of. Do not limit yourself to any one size of can. You may use as many cans as you like. Do not limit yourself to the uses you have seen or heard about; think about as many possible new uses as you can. (From the verbal version of [Torrance, 1990].)

Believe it or not, after the Torrance Test is administered, one can send it out to be professionally judged by fixed and reliable criteria. Building an intelligent agent capable of scoring well on the Torrance Test is a tall order, but some researchers hitherto unaware of Psychometric AI are apparently in the process of working specifically toward this aim, and variants thereof (e.g. see Bringsjord & Ferrucci 2000 and Bringsjord 1998).

3 Objections

Objection 1: “Can you be serious? PAI is so idiosyncratic!” Recall that we mentioned at the outset that AI textbooks tend to be self-reflective. Let’s look a bit at some of these volumes. Doing so will reveal that Psychometric AI is far from idiosyncratic, because it is (at least arguably) a generalization of a longstanding answer to the “What is AI?” question, namely, the answer that appeals to the Turing Test (TT) and its relatives. To see this answer in action, let’s turn to *AIMA* [Russell & Norvig, 1994], which tells us that there are four general, different ways to define intelligence (pp. 4–8): we can say that an entity is intelligent if and only if it “thinks like humans”, “acts like humans”, “thinks rationally”, or “acts rationally.”

Russell & Norvig 1994 opt for the fourth route, but we want to draw your attention to the first and third ones, which don’t seem exactly promising, because ‘thinking’ is probably no clearer than ‘intelligence.’ However, Turing came up with the test that now bears his name precisely because he found the concept of thinking hopelessly vague. As Russell and Norvig point out, TT and other more stringent tests, e.g., Stevan Harnad’s 1991 *Total Turing Test* (in which a passing robot must display not only human-level linguistic performance, but sensorimotor performance at this level as well), provide a way to clarify the first route in the quartet. Specifically, it can be said that AI is the field devoted to building artificial entities (or systems) able to pass TTT.

We could go on to present case after case in which TT or variants are used to define AI (e.g., see Ginsberg’s 1993 introductory AI text), but perhaps you will agree that whether or not you affirm the TT-based answer to the “What is AI?” question, you have to admit that Turing, by the lights of a good many, is definitely on to something. But what, exactly? Well, no doubt there is more than one reason for the apparent immortality of the TT. But surely one reason for its longevity is simply that it’s a clean, crisp *test*. Tests are attractive in no small part because they have determinate starts and ends, and yield concrete verdicts that can silence unproductive debate. (Of course, computability theory relies heavily on tests. E.g., when we say that a set A is **decidable**, we are among other things saying that we can successfully apply a test to some object o in order to determine whether or not $o \in A$.) Turing’s 1950 goal in his seminal “Computing Machinery and Intelligence” was indeed to supplant the maddeningly vague “Can a computing machine think?” with “Can a computing machine pass the Turing test?” We’re not concerned here with whether he reached his goal. Rather, the idea is that PAI extends and clarifies Turing’s approach.

Objection 2: “But Don’t TT and TTT Subsume Psychometric AI?” We offer a three-part rebuttal: (1) In an attempt to build a robot able to pass for a human, certainly “divide and conquer” is a prudent strategy, and Psychometric AI automatically enforces that methodology: tests are crafted to check for different capacities in focused fashion. Since all topics are fair game in TT and TTT, they have much less value as engineering goals. (2) There is another reason why PAI can be viewed as a helpful generalization of the Turing/Harnad approach. This reason is that, let’s face it, neither TT nor TTT is currently a meaningful objective: they are both gi-

gantically ambitious goals, so much so that no one has really made any progress toward reaching them. (At *Turing 2000*, the conference held at Dartmouth to commemorate both Turing's 1950 prediction that a TT-passing machine would be created before the new millennium and AI's inaugural 1956 conference at Dartmouth, no system better at conversation than a toddler appeared.) Psychometrics offers us an abundance of tests, many of which are *reasonable* challenges for an artificial agent. Psychometric AI appropriates these tests. (3) The tests in question haven't been picked out of thin air. These tests allow us to look into the heart of mind/brain. That's the beauty and power of tests, when they have been empirically and statistically validated. Tests have a gem-like character, and PAI piggybacks on this. Given this, if we build an agent able to pass established tests, we can be fairly confident that as a welcome side-effect we will have an agent capable of many significant higher-level feats.

Objection 3: "But AI has applications that need to be built!" Of course. And Turing wasn't saying all that anybody would work on was passing the TT. After all, few work directly on building an agent able to pass it. An agent able to pass the tests in question will have the capacity to provide the desired applications. Momentarily we present some applications we are working toward that stem directly from PAI.

Objection 4: "But PAI will only tap logicist AI!" Actually, the tasks in question will *unite* logicist and sub-symbolic approaches. For example, "Block Design" on the WAIS requires robotic manipulation, and therefore cognitive processing that is rapid and reactive. This processing is not reasoning-based in PERI. The same can be said for PERI's vision and speech systems.

Objection 5: "But plenty of AI researchers don't do PAI!" This objection can be fleshed out as follows: "I fail to see how you can be seriously proposing a foundation for AI, given that plenty of AI researchers don't deal with tests of any kind. Would we be wrong in regarding such people to be doing AI?"

This is an important objection. It raises the general question of whether we are promoting Psychometric AI as **descriptive** or **prescriptive**. Do we mean to suggest that AI is *in fact* PAI? Or do we mean to maintain that AI *ought* to be, or at least be viewed as, PAI?

The answer is this: Our overall claim about Psychometric AI is that it's *both* descriptive *and* prescriptive. If an AI application is sufficiently ambitious, we hold that Psychometric AI will automatically kick in: the theory is descriptive, in that the developers will inevitably modularize the challenge, and set about building sub-agents that successfully negotiate the tests associated with these modules. On the other hand, if the AI application is a "humble" one, it probably itself constitutes a miniature test: our claim is once again that PAI is descriptive. But how is it then that the theory is also prescriptive? Well, again, we submit that R&D dedicated to building test-taking agents will produce building blocks for accomplishing some very helpful systems. Some examples of such systems are presented in section 5. But first, as planned, we say a few things about PERI.

4 The Robot PERI

PERI, whose name stands for "Psychometric Experimental Robotic Intelligence," is a system capable of logic/reasoning, vision, physical manipulation, speech, and hearing. It is important to note that PERI was not designed to simulate how a human thinks. Our work in connection with this robot is AI, not cognitive modeling.

PERI interacts with the environment via its five-degree-of-freedom vertically articulated mechanical arm, a SCORBOT-ER IX model from the Intelitek Corporation, and a pneumatic two-finger parallel gripper which can either be completely open or closed around an object. Its vision is based on the output of a Sony Black-and-White XC55 Video Camera and Cognex MVS-8100M frame grabber. PERI's speech is transmitted through computer speakers and it hears through a microphone attached to the speaker's head while using the Dragon Naturally-Speaking Professional Solutions Version 6.0 software. At the core of PERI resides its brain and nervous system — a complex Lisp program and an associated Scrobot Advanced Control Language Library. Due to the lack of space, this is as much technical detail as we can prudently reveal about PERI.

The rest of this section will focus on the Block Design task and PERI's success with it. PERI can not only solve the particular Block Design problems in the WAIS, but any Block Design puzzle given to it. For legal reasons we are unable to disclose the Block Design task from the WAIS, therefore we discuss another similar yet even more challenging block puzzle (courtesy of the Binary Arts Corporation). (In the interests of space, and in keeping with the "Philosophical Foundations" category under which this paper falls, we leave aside the mathematization of the WAIS block puzzle and the harder one from Binary Arts. For ease of exposition, we refer to the space in question as $\mathcal{S}$.)

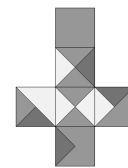


Figure 4: One Puzzle Block Folded Out

In this particular puzzle there are a total of four blocks, each of which is different. There are only three colors (pink, purple, and yellow) used to make the design on each side; that design is either a combination of up to four triangles or one solid color. This is done merely to give a specific color to each edge of a block. In fact, all the sides of all four cubes are different from one another. This means there are a total of 24 unique sides. Refer to Figures 4 and 5 for a closer look.

The task is, after having been presented with the cubes for the first time, to place them together so that every edge that touches another is the same color. All cubes must be used, but obviously there are quite a few different solutions. One solution is shown in Figure 6. Does the task sound easy to you? If so, you are supremely confident. While PERI solves the hard-

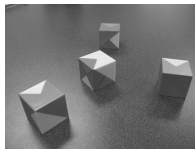


Figure 5: Blocks (from Binary Arts' # 5766) Scattered

est configuration in a matter of seconds, after having visually examined the blocks, in our experience it can take a clever human several long minutes, if not a half hour, to conquer the entire task. Figure 7 shows PERI assembling a solution.

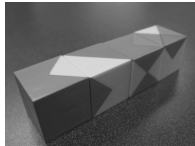


Figure 6: A Solution to Binary Arts Corp.'s Puzzle 5766

The same basic algorithm (described below) that PERI uses for the WAIS Block Design task can be used to solve any such puzzle in the overall mathematical space $\mathcal{S}$ ($\approx$ 3D regular solid with each side having a characteristic capturable in extensional first-order fashion). The first step is to encode the pieces as declarative information about geometric shapes in PERI's "mind." Before the test is administered to a human participant, he is given a chance to examine the blocks (or other shapes), as would PERI. What follows is a general algorithm which PERI can apply to any 3D physical shapes within a limit of size (i.e., which its gripper can properly hold and manipulate).

General Algorithm for "Cracking" any 3D Block Design in $\mathcal{S}$

1. Document original pieces by color, dimension, characteristics on each side, and total number.
2. Input goal configuration (a picture that will need to be deciphered).
3. Partition the goal into distinguishable pieces that match similar aspects of those that are available pieces in the original. Start first with the entire goal as one piece. Some aspects of the pieces may be ignored at this stage.
4. Once the goal has been partitioned, determine if original puzzle pieces match the partitioned ones. If not, go back to step 3 and partition it into two pieces, three pieces, etc. (An exceedingly large cutoff is imposed to handle cases where no partitioning is valid, otherwise non-halting is possible.) If there are matching original puzzle pieces to the goal partitioning, go on to step 5.
5. Start with a goal piece and match it to an original piece that has not yet been used. There will be a finite search for each matching piece since step 4 has been passed, indicating the goal is known to be solvable. When a match is found, the original piece is physically added to the solution "arena" by changing the $\langle x, y \rangle$ positioning of the original piece as well as the angle, side, or any other necessary aspect. Continue the present step until no more pieces in the goal exist that need a match.

In the case of the puzzle from the Binary Arts Corporation, the goal configuration is not specified ahead of time (as it is in the WAIS). Therefore, we assume that the goal is given ahead of time and are then able to use the above general algorithm without any modification. PERI can solve the original version of Binary Arts' puzzle; however, the original version doesn't correspond to the WAIS Block Design task, the cracking of which was our goal. The next challenge PERI faces is the more

difficult "Picture Arrangement" subtask of the WAIS, which is discussed in the next section.

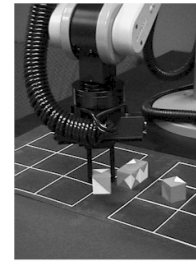


Figure 7: PERI Solving a Block Design Puzzle

5 The Future of Psychometric AI

We very briefly offer some thoughts on the future of PAI, from the standpoints of both applications and philosophical foundations.

Much of our prior AI work in automated test generation has been supported by ETS, in connection with wide-scale "high stakes" tests. Our efforts to crack the WAIS should lead to systems able to generate both new items for other established tests, and new tests. In general, it seems to us that there is a growing symbiosis between AI and the test industry. It isn't just the generation side that holds promise: essays written by examinees taking the Educational Testing Service's GMAT (Graduate Management Admission Test) are now read and graded by machine (though there is human input as well), and this trend promises to accelerate. Another application is the use of tests as security devices. For example, our work in Psychometric AI yields a fairly rigorous account (presentation of which is beyond the scope of the present paper) of what tests a machine can solve, versus what tests positively stump machines, but are solvable by humans. This account can be used, for instance, to devise tests designed to weed out troublesome softbots from human users in situations where hackers attempt to build bots in order to overwhelm online systems (e.g., polling systems).

We believe that the "Picture Arrangement" task (see again Figure 3) provides a helpful microworld for another application: the problem of threat anticipation in the intelligence community. Currently, ETS is supporting research in our lab devoted to augmenting the predictive power of intelligence analysts, as that power is described, for example, in [Heuer, 1999]. However, we intend to pursue a new dimension in this work, one based on the notion that predicting what (say) a terrorist will do is in significant measure the completion of a story based on "snapshots" earlier in the "narrative" in which the terrorist is an actor. Our attempt to enable PERI to crack "Picture Arrangement" is the first step in the exploration of this new dimension. In our initial work on this problem, once a snapshot in a group like that shown in Figure 3 is selected, a search for a consistency proof that a particular successor is possible under narrative constraints is fired. If a proof is found, the successor is selected, and the process iterates. Narrative constraints are declarative formalizations of plots, themes, and characters.

We end by briefly discussing two philosophical issues raised by Psychometric AI.

It's perhaps not uninteresting to ponder the philosophical consequences of a future PERI able to excel on *all* established tests. Would we declare PERI in this case to be a bona fide genius? Or would we instead infer that since a "mere" machine is displaying such mastery, human intelligence must go well beyond what can be tested for? We don't have the answers to such questions, but we will be doing our best, through concrete engineering, to raise these questions. We suspect that the degree to which these questions are debated will relate directly to the degree to which our engineering (and those joining us) succeeds.

Finally, we point out that Psychometric AI seems to clarify the dividing line between so-called "Strong" and "Weak" AI. This is so because PAI is fundamentally driven by how things *appear*, not by whether or not some invisible property is in play. "Weak" AI strives for artifacts which, as a matter of directly observable fact, display certain behaviors. "Strong" AI, on the other hand, aims for machines having such mysterious properties as consciousness and qualia. The distinction can be traced back to the "Objection from Consciousness" in Turing's 1950 defense of TT. The objection is famously encapsulated in Professor Jefferson's Lister Oration of 1949:

Not until a machine can write a sonnet or compose a concerto because of thoughts and emotions felt, and not by the chance fall of symbols, could we agree that machine equals brain — that is, not only write it, but know that it had written it. No mechanism could feel (and not merely artificially signal, an easy contrivance) pleasure at its successes, grief when its valves fuse, be warmed by flattery, be made miserable by mistakes, be charmed by sex, be angry or depressed when it cannot get what it wants. (p. 17 of Turing 1950)

Turing responds: "This argument appears to be a denial of the validity of our test." Precisely! He goes on to stick to his guns: he proclaims that if the computer produces the kind of language normally taken to justify ascriptions of consciousness (and he gives examples of such language in connection with a deep understanding of Shakespeare), we ought to go ahead and make those ascriptions: we ought to hold that the computer is conscious. Psychometric AI and "Weak" AI are firmly in this camp. If there is some sensible, established test for consciousness, if *appearance* is both demanded and supplied, then members of this camp confidently declare the underlying properties to be in place. In light of this, we readily admit that some proponents of "Strong" AI will not be fans of Psychometric AI. Our readers must judge whether or not that is a virtue or a vice for the form of AI we have defended.

6 Objections — Second Round

Objection 6: "You argue that your approach is much in line with Turing's TT and Harnad's TTT, yet those two are concerned with machines behaving like humans, whereas Psychometric AI is concerned with doing *well* on IQ tests (and the like), which many humans obviously do not necessarily do! One crucial question which surprisingly you don't address at all is: What are the cognitive limitations that stop us from doing much better on such tests? Should similar limitations

be put into PAI systems just to make their performance more human-like and thus give them a chance to behave human-like (and pass TT and TTT), or should PAI systems be built to perform as well as possible (but without a chance of passing TT and TTT)? Should PAI, for example, be concerned with building mediocre chess players with human-like limitations instead of really good ones (assuming that chess could count as some kind of test). And if so, wouldn't PAI have to be cognitive modeling (including modeling of limitations), not "just" AI, contrary to what you say (at least about your work with PERI)?"

Rest assured that we are doing AI first and foremost; cognitive modeling is subsidiary, and can be derived from success on the AI front. This objection is plagued by a *non sequitur*: it doesn't follow from the fact that PAI aims at perfect-performing artificial test-takers that there is there no chance of such systems passing TT and TTT. As an engineering technique, it is wise to "crack" tests via algorithms that can enable us to build artificial agents able to perform flawlessly on them. Once that is accomplished, it will be easy to "de-smart" these agents to produce ones able to match any level of poor performance seen in the human sphere. As Turing (1950) pointed out, if a robotic interlocutor in the TT is sufficiently smart, a query to it like "What is $13,567,890,399 \times 23,456,899,221$?" will return a suitably incompetent (= humanesque) response (e.g., "I would need my trusty calculator for that!"), despite the fact that the machine has the power to produce the correct answer instantly.

Objection 7: "I think you are making a mistake in the end by identifying your approach with 'Weak AI', which according to Searle is just the claim that computers are (merely) useful tools in the study of mind. The position of 'Strong AI,' on the other hand, claims that computers (or computer programs) actually can have human-like mental states (if running the right program). You argue that ascriptions of consciousness (etc.) to computers (or computer-controlled robots) could and should be made, and that in my opinion puts you in the 'Strong' AI camp."

Unfortunately, our critic misunderstands the distinction between "Strong" and "Weak" AI. "Strong" AI is indeed the view that cognition (including subjective awareness or phenomenal consciousness) is computation, and that an appropriately programmed information processing machine operating at or below the Turing Limit can literally *be* subjectively aware. "Weak" AI holds that though the external behavior of such machines might convince external observers to ascribe subjective awareness to them, such machines can't possibly have the relevant mental states. Some opponents of "Strong" AI, such as Bringsjord himself, are actively seeking to engineer artifacts (such as PERI himself!) that would convince everyone (or at least *nearly* everyone) that such artificial creatures are literally subjectively aware! — so the objection obviously rests on a confusion (see Bringsjord 1992, Bringsjord & Ferrucci 2000, and the just-published Bringsjord & Zenzen 2003).

Some additional objections are inevitable. We have space here to but rebut some of them in rapid-fire fashion. More thorough rebuttals are of course forthcoming, in a more capacious venue.

- “The statement that PAI requires a system to perform well in ‘any possibly existing’ psychometric tests makes PAI ill-defined.”
 - Not at all. The great challenge we’re willing to face up to is to engineer PERI with the capacity to crack tests he has never seen before. We ask only that the test be validated via the ordinary statistical-mathematical standards used in psychometrics.
- “Your presented example for PERI to tackle — block design — is too simple even in the general case to justify your deep claims.”
 - With all due respect: Get serious. Herbert Simon, at the 1956 inception of AI, proclaimed that thinking computers were just around the corner, on the strength of LOGIC THEORIST’s ability to prove such marvelously subtle theorems as that $\neg q \rightarrow \neg p$ can be derived from $p \rightarrow q$. (For the verbatim sanguine proclamation, see Russell & Norvig 1994.) Today, AI has yet to produce a machine with the conversational power of a sharp toddler. PERI is a research program moving now onto tests that no current AI technology can crack, and we will avail ourselves not only of logic-based formalisms, tools, and techniques that are at the heart of the RAIR Lab, but also — to give PERI robust perception/action capability — ammunition from cognitive robotics that may well be outside the logicist paradigm.
 - Turing (1950) certainly made some rather deep claims (e.g., that all of human cognition is at bottom computational, and that by the year 2000 not only the conversational power of a toddler would be matched by machines, but that of a novelist would be), and as far as we can tell, the only implementations he could point to for justification were painfully primitive.

Acknowledgments

We are indebted to the Educational Testing Service for sponsoring applied AI in the intersection of AI and testing (the eWRITER Project), that led us to ponder the philosophical foundations of AI in connection with this applied work.

References

- [Baron & Kalsher, 2000] R. Baron and M. Kalsher. *Psychology*. Allyn and Bacon, Boston, Massachusetts, 2000.
- [Bringsjord, 1992] S. Bringsjord *What Robots Can and Can’t Be*. Kluwer, Dordrecht, The Netherlands, 1992.
- [Bringsjord, 1998] S. Bringsjord. Chess is Too Easy. *Technology Review*, 1012:23–28, 1998.
- [Bringsjord & Ferrucci, 2000] S. Bringsjord and D. Ferrucci. *Artificial Intelligence and Literary Creativity: Inside the Mind of Brutus, a Storytelling Machine*. Lawrence Erlbaum, Mahwah, New Jersey, 2000.
- [Bringsjord & Zenzen, 2003] S. Bringsjord and M. Zenzen. *Superminds: People Harness Hypercomputation, and More*. Kluwer, Dordrecht, The Netherlands, 2003.
- [Carpenter *et al.*, 1990] P. Carpenter, M. Just, and P. Shell. What One Intelligence Test Measures: A Theoretical Account of the Processing in the Raven Progressive Matrices Test. *Psychological Review*, 97:404–431, 1990.
- [Descartes, 1911] R. Descartes. *The Philosophical Works of Descartes, Volume 1*. Translated by Elizabeth S. Haldane and G.R.T. Ross. Cambridge University Press, Cambridge, United Kingdom, 1911.
- [Evans, 1968] G. Evans. A Program for the Solution of a Class of Geometric-Analogy Intelligence-Test Questions. In Minsky, M., ed. *Semantic Information Processing*, pages 271–353, Cambridge, Massachusetts, 1968. MIT Press
- [Fischler & Firschein, 1987] M. Fischler and O. Firschein. *Intelligence: The Eye, the Brain, and the Computer*. Addison-Wesley, Reading, Massachusetts, 1987.
- [Ginsberg, 1993] M. Ginsberg. *Essentials of Artificial Intelligence*. Morgan Kaufmann, New York, New York, 1993.
- [Harnad, 1991] S. Harnad. Other Bodies, Other Minds: A Machine Incarnation of an Old Philosophical Problem. *Minds and Machines*, 1(1):43–54, 1991.
- [Heuer, 1999] R. J. Heuer. *Psychology of Intelligence Analysis*. Central Intelligence Agency, Center for the Study of Intelligence, 1999.
- [Jensen, 1998] A. Jensen. *The g Factor: The Science of Mental Ability*. Praeger, New York, New York, 1998.
- [Raven, 1962] J. C. Raven. *Advanced Progressive Matrices Set II*. H. K. Lewis, London, United Kingdom, 1962.
- [Russell & Norvig, 1994] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Saddle River, New Jersey, 1994.
- [Spearman, 1927] C. Spearman. *The Abilities of Man*. MacMillan, New York, New York, 1927.
- [Sternberg, 1988] R. Sternberg. *The Triarchic Mind: A New Theory of Human Intelligence*. Viking, New York, New York, 1988.
- [Thurstone, 1938] L. Thurstone. *Primary Mental Abilities*. University of Chicago Press, Chicago, Illinois, 1938.
- [Torrance, 1988] E. P. Torrance. The Nature of Creativity as Manifest in its Testing. In Sternberg, R., ed. *The Nature of Creativity*, pages 43–75, Cambridge, United Kingdom, 1988. Cambridge University Press
- [Torrance, 1990] E. P. Torrance. *The Torrance Tests of Creative Thinking*. Scholastic Testing Service, Bensenville, Illinois, 1990.
- [Turing, 1950] A. Turing. Computing Machinery and Intelligence. *Mind*, LIX(236):433–460, 1950.
- [Wos *et al.*, 1992] L. Wos, R. Overbeek, E. Lusk, and J. Boyle. *Automated Reasoning: Introduction and Applications*. McGraw Hill, New York, New York, 1992.

Tucking RCC in Cyc's Ontological Bed

Pierre Grenon

Institute for Formal Ontology and Medical Information Science
University of Leipzig
Haertelstrasse 16-18, D-04107, Leipzig, Germany
pgrenon@ifomis.uni-leipzig.de

Abstract

Formalisms and axiomatic theories are designed to support reasoning, they are often intended with a preferred interpretation and a targeted ontology. Questions of proper interpretations and of the possible challenge of an intended interpretation arise when integrating a particular theory in pre-existing formal and ontological settings. This paper reports on an instance of this general problem of ontological engineering. The case study is that of the integration of the Region Connection Calculus for spatial reasoning in the Cyc knowledge base. We show that given the assumptions on the Cyc ontology, RCC had to be interpreted within a substantialist metaphysics of space as a Boolean algebra of spatial regions which are distinct from their occupants. The RCC literature suggests such an intended interpretation, and this paper intends to show that this was a necessary condition of integration in Cyc's ontology. This led to the enrichment of the Cyc knowledge base, rather than to a radical modification of the upper-level ontology.

1 Introduction

Our standpoint is that of ontological engineers attempting to integrate a comparatively small and allegedly domain specific theory within a larger, multipurpose formal framework. Bluntly put, our ontological interest is concerned with types of entities admitted by the respective theories (and their domains of variables), the intended interpretations of these theories, and issues of relative coherence and soundness. We consider that ontological engineering in this sense is distinct from a work of implementation. Indeed, the power of information science ontologies lies in their semantic aspects and their ontological foundations. These elements will guide, but ought to remain independent from, a work of implementation within a peculiar computing system. In this context, it clearly appears that in order to use an existing axiomatic theory, it is important to know which are its possible interpretations and, especially, if one is the intended one. It is indeed crucial to evaluate how the theory has to be interpreted in order for it to fit within an existing ontology.

Two main questions arise: (i) does the formal theory support an interpretation compatible with that intended for the embedding framework? (ii) is it possible to bend the theory and its intended interpretation such as to allow maximal integration? This may lead either to non-trivial changes in the embedding ontology or to the choice of a peculiar interpretation for the theory to be integrated. In turn, it becomes necessary to check whether the chosen interpretation is really allowed by the integrated theory (given its axioms) and that it does not generate ontological monsters, so to speak, according to the standards of the larger ontology. When starting a work of integration, one tends to be maximally conservative from the standpoint of the embedding framework (this is, at least, our experience), for the obvious reason that, *a priori*, it might be simpler and more efficient to adapt a specific theory to an existing larger ontology rather than the converse. Changes in the embedding ontology are always perilous and their consequences are not always easily anticipated. Moreover, from the standpoint of an allegedly total or universal ontology, it is very tempting when integrating presumably specific theories to generalize their relevance by extending their domain of interpretation, and even relaxing not only the intended interpretation, but also the axiomatization. The present paper can be taken as a reminder to be cautious when making a methodological maxim of this tendency.

The axiomatic theory of interest is the so-called Region Connection Calculus (RCC as put forward in [Randell *et al.*, 1992]), a theory prototypically applied to spatial reasoning. The embedding ontology in which we introduced RCC is that of the Cyc knowledge base (Cyc KB), as described by [Lenat and Guha, 1990], and which stands as a larger theory of reality. In part 2, we give some elements regarding the background of Cyc's spatial ontology. In part 3, we present RCC's underlying metaphysics as informally rendered in the most significant sources. Part 4 exposes how we were led to question RCC's ontological significance in our work of integration. We present in part 5 the resulting ontological settings and some extensions of the basic theory that we were able to introduce in a Cyc framework.

2 Cyc's original spatial ontology

The Cyc KB is intended to serve as an encyclopedic repository of all human knowledge, primarily common-sense knowledge. It purports to provide a medium for the representation of facts and the inscription of rules about all existing and imaginable things. Cyc's knowledge is represented in a language called CycL, described in [Cycorp, 2002], which here can be seen as a first order language. The Cyc KB is structured and compartmented into theories (called microtheories), additional theories can be introduced *a volo* in order to account for the specificity of a given domain or context. Since our objective was to produce an integration of RCC supporting spatial reasoning in the Cyc system, we will focus on Cyc spatial ontology.

The relevant fragment of the KB is nonetheless extremely broad. Cyc's declared goal of achieving common-sense reasoning has, as an objective, carved up the ontology in a metaphysically determinate way. In particular, "spatial reasoning" has been understood primarily as a type of reasoning about objects in space, and not primarily about space itself. Roughly, this view can be outlined by the position that in the world there are objects only, in particular physical ones, delimitating their 'place'. In this context, spatial reasoning as such refers to the relations that exist between such objects, say, in virtue of being in such and such relative positioning. We shall refer hereafter to this understanding as the 'relational' approach. Cyc's spatial ontology was primarily relational in this sense. Its ontology's top spatial category, `#$SpatialThing`, is defined in [Cycorp, 1997] as that of entities "with a spatial extent or location relative to some other [instance of `#$SpatialThing`]". A large part of the hierarchy under `#$SpatialThing` was covered by the category so-called `#$PartiallyTangible` which is, in a word, that of objects with a material constituent, concrete things. In addition, `#$SpatialThing` also includes certain events. But many specializations were geometrical or otherwise abstract in some sense (so-called `#$Intangible`). Meanwhile, Cyc had already a large body of spatial and topological relations, most of them over `#$PartiallyTangible`. These were typically documented as relations holding between objects in virtue of the relations between these objects' spatial extents. Indeed, many of the spatial entities in Cyc could be handled as spatial extents, metonymically, but were not properly speaking identical with their extents. This is notably the case with geopolitical entities (on the other hand, Cyc has a microtheory in which geopolitical entities are geographical regions, not merely located at such a region), but also with persons (which are examples of composite entities, with a tangible part, a body, and an intangible part), or even books (composed of a material support and an informational content). However, spatial extents (intuitively, spatial regions) were not 'reified' nor mentioned in assertions. Their ontological status was, so to speak, implicit and mostly intuitive. On the other hand, still in [Cycorp, 1997], Cyc had a notion of (pure) spatial region used essentially to define contextually empty space. But then, for an entity, its being an instance of this collection was overtly and essentially context dependent. The documentation discusses as an example a

piece of atmosphere which can be seen as empty space in a particular context, not in another. A substantialist view of space considers space and regions of space as substances independent for their existence of what occupies them. Under this view, it is not contextual for an entity to be a region of space, it is essential. In other words, regions of space as portions of an existentially independent (spatial) substance did not exist in Cyc. We will not delve further into the Cyc ontology. For our purpose, it suffices to observe that attempting to integrate RCC in Cyc meant starting to look at RCC from an essentially relational spatial ontology.

3 RCC and its canonical interpretation

Was this approach allowed by RCC itself? RCC is technically referred to as a mereotopology, that is, a theory with both a mereological fragment (a theory of part and whole) and a topological fragment articulated together. The basic part of the theory, so-called RCC-8, has only one topological primitive: Connects-With (or C). We will assume familiarity with the basics of RCC which can be gathered in [Randell *et al.*, 1992] and, informally, in the introduction of [Cohn *et al.*, 1997]. The calculus can be assimilated to a Boolean algebra, which, as we shall see, is not without ontological consequences. Our interest here is mostly in the ontological significance of the theory. RCC is said to support both a spatial and a temporal interpretation. Here, only a putative spatial interpretation is of interest, so regions will be spatial regions. RCC is, with respect to spatial reasoning, a theory of regions of space and of their relations, not a theory of physical objects nor of their relations to regions. It is already an other form of spatial reasoning than the one we encountered in Cyc. Characteristically, regions are considered as extended (RCC is originally introduced in analogy to Allen's interval logic). In purely model-theoretic terms, intended models are regular open sets of R^n . For common-sense reasoning, n is equal to either 1, 2 or 3. Going further in the specification of regions would be mostly irrelevant, as specializations derive straightforwardly from the theory. However, what is arguably not trivial is precisely what regions are, or, in other words, what their ontological status is: are they dependent or independent ontologically, that is, for their existence, on other entities which are not themselves spatial regions? In fact, depending on whether the term 'region' refers to a part of a substantial space or to a portion of a relational space, the answer and the ensuing interpretation of the theory would be significantly different ontologically speaking. In other words, the alleged reality that the theory intends to capture is not that evident.

It is useful to remark that the relations in RCC are required to hold between regions of equal dimensionality. This indicates the possibility of treating uni-dimensional and bi-dimensional regions of space, even if the respective domains are independent. This means an acceptance of objects of lower dimensionality than that of physical objects (which are uncontroversially tri-dimensional). Thus, if RCC is a fragment of a theory of reality, its underlying ontology probably is not restricted to physical objects and assimilated things, or, in Cyc's terms, `#$PartiallyTangible`. However, so

far it remains open whether one can conceive in a RCC-like framework objects of lower dimensions that are not spatial regions or, alternatively, only regions of lower dimensions at which material objects are somehow located. The rejection of boundaries from RCC after [Randell and Cohn, 1989] is yet another evidence that the theory is about relations among regions of a given non-null dimension.

More generally, in RCC, the distinction between regions and their putative occupants is almost systematic in the informal literature, especially in [Randell and Cohn, 1989; Randell *et al.*, 1992; Cohn *et al.*, 97]. These sources suggest the acceptance of a substantialist view of space; that is, the view that space consists in a substance in its own rights distinct from the objects that occupy portions of it. Formally, the issue is scarcely addressed before [Bennett, 2001] which represents a dramatic broadening of the scope of application of the theory (in addition to the stronger geometrical approach it presents). In multi-sortal axiomatizations, the domain is always restricted to the sort REGION (and secondarily to the sort NULL), a sort PhysObj for occupants of regions is mentioned only to limit the scope of the theory. The first order axiomatizations are seen as fragments of more complete theories. It is not clear how much this is, or is not, an artifact of the sortal approach which is itself an artefact of a putative implementation. Presumably, however, first-order axioms should incorporate quasi-sortal restrictions implicit in the multi-sortal settings.

4 Path to Integration

Cyc's general maxim for ontological integration is generalization. On the one hand, the foregoing considerations on the use of sorts in RCC provide a starting point for expanding the RCC schema by generalizing over sortal restrictions. Overlooking any sortal restriction, on the other hand, is tantamount to claiming generalization of RCC to putatively all entities, namely, #SpatialThing in Cyc. This is clearly not desirable: the scope has to be at least restricted to spatial things. For Cyc, objects in space (and physical events) are primitives. If RCC is generalizable to #SpatialThing and if regions (#SpaceRegion) are but a subtype of spatial things, it would be redundant and sub-optimal to develop the theory on regions only. Yet, RCC takes regions as primitives. Pushed to the extreme, this latter position leads to defining spatial objects in terms of regions. There would be only one substance, space. An object would be no more than a qualitative singularity in space. Such an eliminativist view (doing away with the primitivity of entities in space) is not conceivable for Cyc's upper-level ontology. The compromise still appears to be straightforward generalization of RCC's notions at the level of #SpatialThing in Cyc. After all, there seems to be no *prima facie* objection to this in the RCC literature. Indeed, the intended interpretation for RCC is as a calculus of 'spatial regions'. But the sources are all starting from the introduction of regions as primitives. There seems to be no claim in the other sense, namely that RCC means, specifically and necessarily, spatial regions, and not, more generally, spatial things in the sense of the Cyc KB.

Let us assume that generalization is possible. C is then to be interpreted as a relation between objects such that these spatially connect along the RCC lines ([Cohn and Varzi, 1996] situates RCC's C among a variety of interpretations). Modulo the issues with openness of regions in RCC, Cyc had a relation of touching among #SpatiallyTangible which intuitively was close enough to C to suggest the possibility of a sound generalization at the level of #SpatialThing. Most RCC-8 relations (the basic set of binary relations in RCC) had proxies in Cyc (with exception of inverses for implementation reasons). More precisely, there were relations at the level of #SpatiallyTangible which were holding true in virtue of their relata's putative spatial extents in RCC-like relations themselves. In particular, Cyc already had a relation of co-spatiality, a relation holding true of things with identical spatial extents. Meanwhile, [Randell *et al.*, 1992] defines = as identity between regions. However, in Cyc, co-spatiality was not identity. There were conceivable cases of entities with the same (spatial or geographical) extent and which were not identical as alluded to in Part 2. The question of whether a generalization of RCC's relations was possible could be rephrased as whether = in RCC was indeed identity or could be interpreted as a co-spatiality relation on #SpatialThing. If the former, we would have applied RCC straightforwardly to the Cyc ontology. We will see that the status of = in the RCC literature is something of a puzzle.

In [Randell and Cohn, 1989; Randell *et al.*, 92], we find the following axiom:

$$(1) x = y \equiv (P(x,y) \wedge P(y,x)),$$

where P is "Part-Of". That is, x is identical to y if and only if x is a part of y and y is a part of x. (1) simply states the anti-symmetry of P, i.e., if two objects of the domain are part of the other, they are actually one and the same. Observe that given the definition of P:

$$(2) P(x,y) \equiv \forall z (C(z,x) \rightarrow C(z,y)),$$

that is, x is a part of y if and only if everything that connects with x connects with y, we can infer the extensionality of C:

$$(3) x = y \leftrightarrow \forall z (C(z,x) \leftrightarrow C(z,y)).$$

There is however a progressive terminological and ideological shift in the RCC literature concerning the status of =. In [Gotts, 1994], 'EQ' is the privileged notation for the relation of identity and '=' is only mentioned as an alternative notation. With [Bennett, 1995], the matter becomes allegedly more than a notational issue. [Bennett, 1995] argues in section 3 that an axiom of extensionality for C (and thus the anti-symmetry of P) is optional in RCC. The choice depends on the way we construe (1), either as an axiom or as a definition. A similar claim is made with the note 2 of [Bennett, 1996]:

"[Randell *et al.*, 1992] define[s] the relation EQ(x,y) as equivalent to $P(x,y) \wedge P(y,x)$. If this definition is regarded as an axiom rather than a definition and if EQ is treated as logical equality then this axiom is equivalent to Cext."

Cext is the right to left implication in (3), the use of 'EQ' for '=' is [Bennett, 1995]'s import. Using '=' is just a convenient abuse of notation, this symbol is not really that for equality. In order to follow the rationale, let us adopt momentarily this shift of notation, and reserve '=' as a symbol for equality. The thesis is radically defended by [Stell, 2000] that because EQ is allegedly defined and not simply introduced as a logical primitive, EQ is not equality. EQ would simply be an equivalence relation. [Stell, 2000] mentions the note 11 in [Cohn *et al.*, 1997]:

“For notational convenience we will sometimes write $x = y$ rather than $EQ(x, y)$; technically the latter is preferable, since EQ is a relation defined in terms of C rather than true logical equality.”

[Stell, 2000] claims that this note provides evidence supporting intuitively his claim. In our opinion, this is a misdiagnosis and the note warrants a better interpretation. Namely, it is an indication that the introduction of EQ (originally, =) by way of definition in RCC reflects a desire of reducing the number of primitive relations in the theory. The equality relation is defined via P, P is defined via C; thus far, there is only one primitive in the theory, C, instead of two, C and =. Interestingly, although not addressing this point of detail, [Smith, 1996] stigmatizes this general tendency in RCC as inherited from the classical schools of mereology.

There is a more serious rationale than such footnotes exegesis for not endorsing the [Bennett, 1995; Stell, 2000] alternative. It seems now that the contentious part of the argument rests on the claim that EQ is capable of taking different interpretations according to the way (1) is conceived. Let us assume that EQ is not = (logical identity) and rewrite (1), (2) and (3) with EQ in place of = (call the resulting propositions (1'), (2') and (3')). Let us see the possible consequences for the ontology. This choice apparently enables the generalization of the intended domain of interpretation from the domain of pure space to that of objects in space. EQ is merely an equivalence relation, "EQ(x,y)" means that x and y have the same region, the same spatial extent. This corresponds (again, abstracting away fine details of the topology) to the co-spatiality relation in Cyc. Note how accommodating this interpretation might be. If we can still include in our ontology regions of space and preserve the ability to interpret on this restricted domain EQ as =, EQ gives the identity criterion for space regions (= entails EQ and, on regions, EQ entails =). So, it seems that choosing a weak interpretation is rather positive and optimal for the resulting ontology.

But there is a drawback. If EQ is but an equivalence relation over a broader domain, then P, which is now a relation on objects (such that the extent of one is part of the other's extent), is no longer anti-symmetric. This means that RCC no longer warrants that two objects with the same (spatial) extents are one and the same thing. In particular, there could be several distinct entities which are all the sums of a given set of objects, say three bricks. Without (1) and only (1'), the sum of brick 1 and of the sum of brick 2 and 3 is distinguishable from the sum of brick 3 and of the sum of brick 1

and 2. What is crucial in deciding of the possibility of generalizing the full RCC calculus (RCC-8 and additional Boolean operators) is the behavior of operators in the absence of (1) or equivalently the extensionality of C. The operator for sum, for instance, is defined by [Randell *et al.*, 1992] by:

(4) $\text{sum}(x,y) = \text{def } \iota w (\forall z (C(z,w) \leftrightarrow (C(z,x) \vee C(z,y))))$.

RCC's literature generates a new puzzle for interpreting (4). [Randell and Cohn, 1989; Randell *et al.* 1992] suggest that while (4) uses the iota operator of definite description, its method of elimination is not necessarily, although possibly, to follow Russell's. Rather, it is intended to be eliminated such as to produce the following assertion:

(5) $\forall x \forall y \forall z (C(z, \text{sum}(x,y)) \leftrightarrow (C(z,x) \vee C(z,y)))$.

This has the effect of virtually allowing plural reference, if EQ is to be interpreted otherwise than as equality (since without (1), (1') is not sufficient to rule out that there may be more than one entity substitutable to the sum of two entities in (5) and thus more than one candidate for being such a sum). The trouble with operators which may lose their intended functionality, goes seemingly unnoticed. Moreover, (5) does not even secure the existence of a value. By suggesting this transcription, the authors merely seek to ensure that the Boolean operators are totally defined on their domain. For this purpose, providing for a conventional value in cases where the operators do not properly denote, the introduction of the NULL sort, or, in other words, a notion of null region, is required. It is clear, even for these authors, that this move is not ontologically defensible (see also page 13 of [Simons, 1987] for its vehement dismissal). Rather, this is an algebraic desiderata and an implementational requirement (in short, codes have to return a value). In the context of a putative generalization, this is even more puzzling. What could possibly be the mereological product of two people? A null person (or null person's part)? Such a question leads to nonsense from Cyc's common-sense standpoint. Cyc's uneasiness with the ontological prodigality of RCC's Boolean operators goes further. If Boolean operators denote, they denote objects in space. The sum of this cup and the spill of coffee on this desk is constructible in the theory as a spatial thing. Both the cup and the puddle are in space, that is consensual. The question is whether it is desirable (and needed) to consider that there are sums of such arbitrarily united objects. Ontologically speaking, there were no incentives to go after a notion of null object and totally defined operators, although even in Cyc, the implementation would eventually require an apparatus mimicking the function of the null region.

So, conversely, what was the meaning of a Russellian elimination of definite description? This would result in the following transcription of (4):

(6) $\exists w (\forall z (C(z,w) \leftrightarrow (C(z,x) \vee C(z,y))))$

$\wedge \forall w' (\forall z (C(z,w') \leftrightarrow (C(z,x) \vee C(z,y))) \rightarrow w = w')$.

Observe that (5) is indeed entailed by (6). But there is something new in (6). Namely, = is indubitably logical identity and we verify that (6) entails the uniqueness of sum. If EQ was not =, there could be two objects, a and b, in the domain

such that $EQ(a,b)$ but not $a=b$. However, by (3'), these two objects would connect the same objects. Since trivially any object is a sum, any object in the EQ relation with it connects its parts (by (3')) and the first conjunct of (6)). But then a and b are identical (by the second conjunct of (6)). EQ is therefore $=$. This, in turn, entails the extensionality of C .

So, if the iota operator is eliminated à la Russell, C is indubitably extensional, P is anti-symmetric, and EQ is $=$. We took this to be a *reductio ad absurdum* of the [Bennett, 1995; Stell, 2000] thesis, and, in fine, as an indication that Cyc's co-spatiality was not an RCC relation. In other words, we had to interpret RCC as a calculus of regions of a substantial space at which entities of various kinds could be located, and not as a theory of these entities. The alternative would have been either going forward with a non extensional mereo(topo)logy of objects-in-space departing significantly from both RCC and Cyc or going back precisely to the more restricted domain of space regions.

5 Extending Cyc's spatial ontology

We thus chose to go for an interpretation according to which RCC was a theory of regions of substantial space. We therefore developed an ontology of space as an entity with an independent existence, choosing to rework and have `#$SpatialRegion` for domain of RCC relations. In order to bridge the gap between spatial things and spatial regions, we developed a locative apparatus inspired largely by [Casati and Varzi, 1996]. This enabled us to formalize the meaning of object level spatial categories and relations, and make explicit their metonymical character: spatial relations among objects were holding in virtue of similar relations between these objects' spatial extents. Originally, only spatial objects were in the ontology. We introduced (pure) spatial regions, RCC relations among some of these regions, and relations of location between spatial objects and regions. We used these in order to explicitly define spatial relations. This resulted in a spatial reasoning language with greater expressive power, clearer semantics, and an enriched ontology. It also facilitated spatial reasoning with access to a greater variety of data. We could use spatial regions for instance as handles when manipulating datasets for GIS based reasoning. There is unfortunately not enough room here to give specifics.

In a dedicated microtheory, we put some basic constraints on our calculus following RCC's intended interpretation, these motivated in particular further categorial distinctions. To begin with, RCC is region based: relationships are understood as holding among extended regions, i.e., neither points or scattered sets of points are in the domain. We interpreted the regularity of the open sets in the alleged models of RCC as founding yet another category of spatial regions, namely regions homogeneous in dimensionality. We wanted to apply RCC to regions of the three basic spatial dimensionalities (namely, one, two and three dimensional regions), consistently with Cyc's common-sense reasoning leanings. We chose to introduce dimensionality as primitive notions (at the topological level) and as instances of `#$AttributeValue` (properties of entities as abstract par-

ticulars). This afforded us to define three specializations of extended spatial regions which were homogeneous in dimensionality, i.e., regular in the intended sense. We were only requiring our relations to take as arguments homogeneous extended regions and were axiomatically enforcing constraints on co-dimensionality of the relata. So, in the end, we had a single set of RCC relations and operators which were applicable within each dimension independently of other dimensions.

Anecdotally, we carried on in a specific microtheory an atomic extension of RCC along the lines suggested in [Randell *et al.*, 1992] by imposing categorial restrictions on the relevant axioms. Despite RCC being a region based calculus, a larger ontology such as Cyc could make use of point-based reasoning and zero dimensional elements. We developed a variant of RCC in a specific microtheory introducing the category of `#$SpatialPoint` as a primitive and a relation of incidence of points in a region as suggested by [Randell *et al.*, 1992]. We refined this elementary treatment with more topologically oriented primitives of interior and boundary point incidence. Additionally, we developed a set of trans-dimensional relations among entities of other dimensions. If material objects are always located at a three dimensional region, they can be geographically located at a given parallel for instance. Moreover, their footprints (geographical projection) are two-dimensional extents with similar locational properties. So, on the one hand, we allowed partial location of entities at regions of lower dimensionality, possibly at points. On the other hand, we designed projective locations, typically intended to account for geographical positioning. More generally, this allowed approaching regional boundaries in terms of incidence.

6 Concluding Remarks

There exist alternatives in the metaphysics of space. The main distinction being between substantial and relational space. This paper intended to provide evidence that the RCC contributors are 'right' in putting forward certain ontological basics as a frame for using their theory, namely a substantial view of space. The reason why it could not be relational is that the distinction between co-spatial and identical objects would be lost, while co-spatiality is not an identity criterion of objects in space. Once the ontological independence of spatial regions is endorsed, however, two solutions exist. On the one hand, objects are themselves independent from the regions they occupy or they are not. If they are, as in Cyc, RCC has to be interpreted as we put forward here. However, there might be a way in which we could mimic if not secure a relational formalism. RCC predicates could be predicated of objects and events in space, nonetheless, the denotations of operators such as sum, for instance, provided they exist, would be spatial extents. In that context, the sum of two entities in the RCC sense would be the sum of their extents. As the extensionality of C would be preserved by this functionality of sum, this would be identical to the extent of any of their putative sums (in a purely mereological sense). One can then endorse the thesis that such extents are

parts of some kind (namely, spatial) of entities in space. Co-spatiality becomes a form of overlapping, it is identity of spatial parts. However, this solution requires a broad and strong mereological treatment at the upper-level of the ontology which goes far beyond spatiality. Such a solution which was not affordable in the terms of Cyc's ontology suggests the possibility of alternative interpretations for RCC provided it is to be embedded in an ontology with a more general and primitive mereological framework.

Acknowledgements

The redaction of this paper was supported by the Alexander von Humboldt Foundation under the auspices of its Wolfgang Paul Program. This paper stems from research done while working at Cycorp. Such work, in particular that alluded to in section 5, could never have been accomplished without the participation of James Masters, many thanks go to him for the many discussions and debates we had there. Additional thanks go to James Zaiss, Pierluigi Miraglia, Michael Pool, Fritz Lehmann and William Jarrold for helpful comments and discussions during this period. Thanks go to Brandon Bennett for discussions during his visit at IFOMIS, and to finish with to the anonymous reviewers of IJCAI for their insightful comments. They are not responsible for any errors which remain.

References

- [Bennett, 1995] Brandon Bennett. Carving up Space: existential axioms for logical theories of spatial regions. In *Proceedings of the 1995 International Joint Conference on Artificial Intelligence workshop on Spatial and Temporal Reasoning*, Montreal, Canada, 1995.
- [Bennett, 1996] Brandon Bennett. Carving up Space: steps towards construction of an absolutely complete theory of spatial regions. In *Proceedings of JELIA '96*, pages 337-353, 1996.
- [Bennett, 2001] Brandon Bennett. Space, Time, Matter and Things, *Proceedings of the Second International Conference on Formal Ontology in Information Systems (FOIS'01)*, pages 105-116, 2001.
- [Casati and Varzi, 1996] Roberto Casati and Achille C. Varzi. The structure of Spatial Localization. *Philosophical Studies*, 82, pages 205-239.
- [Cohn and Varzi, 1996] Anthony G. Cohn and Achille C. Varzi. Modes of Connection. In *Spatial Information Theory - Cognitive and Computational Foundations of Geographic Information Science* (Lecture Notes in Computer Science 1661), pages 299-314, Springer-Verlag, 1999.
- [Cohn *et al.*, 1997] Anthony Cohn, Brandon Bennett, J. Gooday, and N. M. Gotts. RCC: a calculus for Region based Qualitative Spatial Reasoning. *GeoInformatica*, 1:275-316, 1997.
- [Cycorp, 1997] <http://www.cyc.com/cyc-2-1/toc.html>, Cycorp, Inc.
- [Cycorp, 2002] <http://www.cyc.com/cycl.html>, Cycorp, Inc.
- [Gotts, 1994] N. M. Gotts. How far Can We 'C'? Defining a 'Doughnut' Using Connection Alone. In *Proceedings of the Fourth International Conference on Knowledge Representation*, Morgan Kaufmann, pages 246-257, 1994.
- [Lenat and Guha, 1990] Douglas B. Lenat and R. V. Guha. *Building Large Knowledge Based Systems*. Reading, Massachusetts: Addison Wesley, 1990.
- [Randell and Cohn, 1989] David A. Randell and Anthony G. Cohn. Modelling Topological and Metrical Properties in Physical Processes. *Proceedings of the First International Conference on the Principles of Knowledge Representation and Reasoning*, pages 55-66, Morgan Kaufmann, Los Altos, 1989.
- [Randell *et al.*, 1992] David A. Randell, Zhan Cui, and Anthony G. Cohn. A Spatial Logic Based on Regions and Connection. *Proceedings of the Third International Conference on Knowledge Representation and Reasoning*, pages 165-176, Morgan Kaufmann, San Mateo, 1992.
- [Simons, 1987] Peter Simons. *Parts: a study in ontology*. Oxford University Press, 1987.
- [Smith, 1996] Barry Smith. Mereotopology: A theory of parts and boundaries. In *Data & Knowledge Engineering*, 20:287-303, 1996.
- [Stell, 2000] John G. Stell. Boolean Connection Algebras: A New Approach to the Region-Connection Calculus. *Artificial Intelligence*, 122:11-136, 2000.

Integrity and Change in Modular Ontologies

Heiner Stuckenschmidt and Michel Klein

Vrije Universiteit Amsterdam

de Boelelaan 1081a, 1081HV Amsterdam

{heiner, michel.klein}@cs.vu.nl

Abstract

The benefits of modular representations are well known from many areas of computer science. In this paper, we concentrate on the benefits of modular ontologies with respect to local containment of terminological reasoning. We define an architecture for modular ontologies that supports local reasoning by compiling implied subsumption relations. We further address the problem of guaranteeing the integrity of a modular ontology in the presence of local changes. We propose a strategy for analyzing changes and guiding the process of updating compiled information.

1 Introduction

Currently, research in the area of the semantic web is in a state where ontologies are ready to be applied in real applications such as semantic web portals, information retrieval or information integration. In order to lower the effort of building ontology-based applications, there is a clear need for a representational and computational infrastructure in terms of general purpose tools for building, storing and accessing ontologies. A number of such tools have been developed, i.e. ontology editors, reasoning systems and more recently storage and query systems.¹ Most of these tools, however, treat ontologies as monolithic entities and provide little support for specifying, storing and accessing ontologies in a modular manner.

1.1 Why Modularization ?

There are many reasons for thinking about ontology modularization. Our work is mainly driven by three arguments. These also bias the solution we propose, as it is aimed at improving the current situation with respect to the following aspects.

Distributed Systems: In highly distributed systems such as the semantic web, modularity naturally exists in terms of physical location. Providing interfaces and mechanisms

for connecting these natural modules is a prerequisite for easy maintenance [Heflin and Hendler, 2000].

Large Ontologies: Modularization also helps to manage very large ontologies we find for example in medicine or biology. Here modularity helps to maintain and reuse parts of the ontology as smaller modules are easier to handle than the complete ontology [Rector, 2003].

Efficient Reasoning: A specific problem that occurs in the case of distributed and large models is the problem of efficient reasoning. The introduction of modules with local semantics and clear interfaces will help to develop efficient reasoning methods [McIlraith and Amir, 2001].

1.2 Requirements

There are a couple of requirements a modular ontology architecture has to fulfill in order to improve ontology maintenance and reasoning in the way suggested above. The requirements will be the main guidelines for the design of our solution proposed in this work.

Loose Coupling: In general, we cannot assume that two ontology modules have anything in common. This refers to the conceptualization as well as the specific logical language used for the interpretation of objects, concepts or relations.

Self-Containment: In order to facilitate the reuse of individual modules we have to make sure that modules are self-contained. In especially, the result of certain reasoning tasks such as subsumption or query answering within a single module should be possible without having to access other modules.

Integrity: Having self-contained ontology modules may lead to inconsistencies that arise from changes in other ontology modules. We have to provide mechanisms for checking whether relevant knowledge in other systems has changed and for updating our modules accordingly.

1.3 Our Approach

In the following, we describe our approach to ontology modularization on an abstract level. We emphasize the main design decisions and motivate them on the basis of the requirements defined above. The technical details of the approach will be given in the following sections.

¹An extensive overview is provided in the OntoWeb deliverable, available at http://www.ontoweb.org/download/deliverables/D13_v1-0.zip.

View-Based Mappings: We adopt the approach of view-based information integration. In particular, ontology modules are connected by conjunctive queries. This way of connecting modules is more expressive than simple one-to-one mappings between concept names but less expressive than the logical language used to describe concepts. We decide to sacrifice a higher expressiveness for the sake of conceptual simplicity and desirable semantic properties such as independence of the ontology language used.

Compilation of Implied Knowledge: In order to make local reasoning independent from other modules, we use a knowledge compilation approach. The idea is to compute the result of each mapping query off-line and add the result as an axiom to the ontology module using the result. During reasoning, these axioms replace the query thus enabling local reasoning.

Change Detection and Automatic Update: Once a query has been compiled, the correctness of reasoning can only be guaranteed as long as the concept hierarchy of the queried ontology module does not change. In order to decide whether the compiled axiom is still valid, we propose a change detection mechanism that is based on a taxonomy of ontological changes and their impact of the concept hierarchy.

The rest of the work is organized as follows. In section 2 we provide a definition of ontology modules based on a minimal notion of ontologies that fixes important properties we will use later on while leaving as much freedom for specific implementations as possible. Section 3 introduces our approach to self-containment in terms of compiled knowledge. The remainder of the paper is devoted to the problem of detecting changes and preserving integrity amongst modules in a system.

2 Modular Ontologies

Before we start investigating the problem of change and integrity, we define the notion of modular ontology we will use as a basis for our technical results.

2.1 Modules and Queries

A number of languages for encoding ontologies on the Web have been proposed (see [Gomez-Perez and Corcho, 2002] for an overview). In order to get a general notion of ontological knowledge, we define the general structure of an ontological module and its instantiation independent of a concrete language.

Definition 1 (Ontology Module) A module is a triple $M = \langle \mathcal{C}, \mathcal{R}, \mathcal{O} \rangle$ where $\mathcal{C}$ is a set of concept definitions, $\mathcal{R}$ is a set of relation definitions and $\mathcal{O}$ is a set of object definitions. Further, we define the signature of a module $\langle \mathcal{C}, \mathcal{R}, \mathcal{O} \rangle$ to be a triple $\langle \mathcal{CN}, \mathcal{RN}, \mathcal{ON} \rangle$, where $\mathcal{CN}$ is the set of all names of concepts defined in $\mathcal{C}$, $\mathcal{RN}$ the set of all relation names in $\mathcal{R}$ and $\mathcal{ON}$ the set of all object names occurring in $\mathcal{O}$.

Queries over ontological knowledge are defined as conjunctive queries, where the conjuncts are predicates that correspond to concepts and relations of an ontology. Further,

variables in a query may only be instantiated objects in that ontology.

Definition 2 (Ontology-Based Queries) Let V be a set of variables disjoint from $\mathcal{ON}$ then an ontology-based query Q over a module $M = \langle \mathcal{C}, \mathcal{R}, \mathcal{O} \rangle$ is an expressions of the form $q_1 \wedge \dots \wedge q_m$ where q_i are query terms of the form $C(x)$ or $R(x, y)$ such that $x, y \in V \cup \mathcal{ON}$, $c \in \mathcal{CN}$ and $r \in \mathcal{RN}$ or are of the form $x = o$ where $x \in V$ and $o \in \mathcal{ON}^2$.

The fact that all conjuncts relate to elements of the ontology allows us to determine the answer to ontology-based queries in terms of instantiations of the query that are logical consequences of the knowledge base.

2.2 Internal and External Definitions

The notion of module and query given above is a quite standard ones. What makes up a modular ontology now, is the possibility to use ontology-based queries in order to define concepts in one module in terms of a query over another module. For this purpose, we divide the set of concepts in a module into internally defined concepts $\mathcal{C}_I$ and externally defined concepts $\mathcal{C}_E$ resulting into the following definition of $\mathcal{C}$:

$$\mathcal{C} = \mathcal{C}_I \cup \mathcal{C}_E, \mathcal{C}_I \cap \mathcal{C}_E = \emptyset \quad (1)$$

Internally defined concepts are specified by using concept expressions in the spirit of description logics [Baader *et al.*, 2003]. We do not require a particular logic to be used.

Definition 3 (Internal Concept Definition) An internal concept definition is an axiom of one of the following forms $C \sqsubseteq D, C \equiv D$ where $C \in \mathcal{CN}$ and D is a concept expression of the form $f(t_1, \dots, t_n)$ where the terms t_i are either concept names or concept expressions and f is an n -ary concept building operator.

Besides this standard way of defining concepts, we consider externally defined concepts that are assumed to be equivalent to the result of a query posed to another module in the modular ontology. This way of connecting modules is very much in spirit of view-based information integration which is standard technique in the area of database systems [Halevy, 2001]. The choice of conjunctive queries for connecting different modules is motivated by the trade-off between expressiveness of the mapping and conceptual as well as computational simplicity. Our approach is more expressive than simple one-to-one mappings; having more complex mappings would contradict the principle of loose coupling of different modules.

Definition 4 (External Concept Definition) An external concept definition is an axiom of the form: $C \equiv M : Q$ Where M is a module and Q is an ontology-based query over the signature of M .

A modular ontology is now simply defined as a set of modules that are connected by external concept definitions. In particular we require that all external definitions are contained in the modular system.

²Note that this may include data-type expressions as the type itself is can be considered to be a concept, the actual value a member of that concept and the comparison operator a special relation.

Definition 5 (Modular Ontology) A modular ontology $\mathcal{M} = \{M_1, \dots, M_m\}$ is a set of modules such that for each externally defined concept $C \equiv M_i : Q$, M_i is also a member of $\mathcal{M}$.

We will use this notion of a modular ontology in the following to investigate the problem of integrity of logical reasoning across modules.

2.3 Semantics and Logical Consequence

We define a model-based semantics for modular ontologies using the notion of a distributed interpretation proposed by [Borgida and Serafini, 2002] in the context of distributed description logics:

Definition 6 (Distributed Interpretation) A distributed interpretation $\mathfrak{S} = \langle \{\mathfrak{S}_i\}_{i \in \text{Index}}, r \rangle$ of a modular ontology $\mathcal{M}$ consists of interpretations $\mathfrak{S}_i$ for the individual module M_i over domains Δ_i , such that:

- $C_i^{\mathfrak{S}} \subseteq \Delta_i$ for all concept definitions $C \in \mathcal{C}_i$
- $R_i^{\mathfrak{S}} \subseteq \Delta_i \times \Delta_i$ for all relation definition $R \in \mathcal{R}_i$
- $O_i^{\mathfrak{S}} \in \Delta_i$ for all object definitions $O \in \mathcal{O}_i$

and a function r associating to each pair of indices i, j a binary relation $r_{ij} \subseteq \Delta_i \times \Delta_j$. $r_{ij}(d)$ denotes the set $\{d' \in \Delta_j \mid (d, d') \in r_{ij}\}$; for every $D \subseteq \Delta_i$ $r_{ij}(D)$ denotes $\bigcup_{d \in D} r_{ij}(d)$.

The assumption of disjoint interpretation domains again reflects the principle of loose coupling underlying our approach. Based on the notion of a distributed interpretation we can define a model of a modular ontology as an interpretation that satisfies the constraints imposed by internal and external concept definitions. In contrast to [Borgida and Serafini, 2002], we do not introduce special operators for defining the relations between different domains, we rather interpret external concept definitions as constraints on the relation between the domains:

Definition 7 (Logical Consequence) A distributed interpretation $\mathfrak{S}$ is a model for a modular ontology $\mathcal{M}$ if for every module M_i we have $\mathfrak{S} \models C$ for every concept definition C in M_i where $\models$ is defined as follows.

- $\mathfrak{S} \models C \sqsubseteq D$, iff $C_i^{\mathfrak{S}} \subseteq D_i^{\mathfrak{S}}$
- $\mathfrak{S} \models C \equiv D$, iff $C_i^{\mathfrak{S}} = D_i^{\mathfrak{S}}$
- $\mathfrak{S} \models C \equiv M_j : Q$, iff $C_i^{\mathfrak{S}} = r_{ji}(Q^{\mathfrak{S}_j})$.

Here $Q^{\mathfrak{S}_j}$ denotes the interpretation of the set of answers to query Q . An axiom A logically follows from a set of axioms S if $\mathfrak{S} \models S$ implies $\mathfrak{S} \models A$ for every model $\mathfrak{S}$. We denote this fact by $S \models A$.

The actual definitions of concepts impose further constraints on the interpretation of a modular ontology. For the case of internally defined concepts, these constraints are provided by the definition of concept building operators of description logics. For the case of externally defined concepts, the situation is more complicated and will be discussed in more details in the next section.

3 Compilation and Local Reasoning

Using the notion of logical consequence defined above, we now turn our attention to the issue of reasoning in modular ontologies. For the sake of simplicity, we only consider the interaction between two modules in order to clarify the basic principles. Further, we assume that only one of the two modules contains externally defined concepts in terms of queries to the other module.

3.1 Implied Subsumption

As mentioned in the introduction, we are interested in the possibility of performing local reasoning. For the case of ontological reasoning, we focus on the task of deriving implied subsumption relations between concepts within a single module. For the case of internally defined concepts this can be done using well established reasoning methods [Donini et al., 1996]. Externally defined concepts, however, cause problems: being defined in terms of a query to the other module, a local reasoning procedure will often fail to recognize an implied subsumption relation between these concepts. Consequently, subsumption between externally defined concepts requires reasoning in the external module as the following theorem shows.

Theorem 1 (Implied Subsumption) Let E_1 and E_2 be two concepts in module M_i that are externally defined in module M_j by queries Q_1 and Q_2 , then $\mathfrak{S} \models E_1 \sqsubseteq E_2$ if $\mathfrak{S}_j \models Q_1 \sqsubseteq Q_2$.

Proof 1 $[\mathfrak{S}_j \models Q_1 \sqsubseteq Q_2] \Rightarrow [Q_1^{\mathfrak{S}_j} \subseteq Q_2^{\mathfrak{S}_j}] \Rightarrow [r_{ji}(Q_1^{\mathfrak{S}_j}) \subseteq r_{ji}(Q_2^{\mathfrak{S}_j})] \Rightarrow [E_1^{\mathfrak{S}_i} \subseteq E_2^{\mathfrak{S}_i}] \Rightarrow [\mathfrak{S} \models E_1 \sqsubseteq E_2]$

The result presented above implies the necessity to decide subsumption between conjunctive queries in order to identify implied subsumption relations between externally defined concepts. In order to decide subsumption between queries, we translate them into internally defined concepts in the module they refer to. A corresponding sound and complete translation is described in [Horrocks and Tessaris, 2000]. Using the resulting concept definition, to which we refer as *query concepts*, we can decide subsumption between externally defined concepts by local reasoning in the external ontology.

3.2 Compilation and Integrity

We can avoid the need to perform reasoning in external modules each time we perform reasoning in a local module using the idea of knowledge compilation [Cadoli and Donini, 1997]. The idea of compilation is to perform the external reasoning once and add the derived subsumption relations as axioms to the local module. These new axioms can then be used for reasoning instead of the external definitions of concepts. This set of additional axioms can be computed using Algorithm 1.

If we want to use the compiled axioms instead of external definitions, we have to make sure that this will not invalidate the correctness of reasoning results. We call this situation, where the compiled results are correct as integrity. We formally define integrity as follows:

Algorithm 1 Compile

Require: the module $M = \langle C_I \cup C_E, R, O \rangle$
Require: the external module $M_j = \langle C_j, R_j, O_j \rangle$
for all $E \equiv M_j : Q \in C_E$ **do**
 $C'_E := C'_E \cup \{E \sqsubseteq C \mid C \in C_j, \mathfrak{S}_j \models E \sqsubseteq Q\}$
end for
return C'_E

Definition 8 (Integrity) We consider integrity of two ontology modules M, M_j to be present if $M, M_j \models M^c$ where M^c is the result of replacing the set of external concept definitions in M by $\text{compile}(M, M_j)$.

At the time of applying the compilation this is guaranteed by theorem 1, however, integrity cannot be guaranteed over the complete life-cycle of the modular ontology. The problem is, that changes to the external ontology module can invalidate the compiled subsumption relationships. In this case, we have to perform an update of the compiled knowledge.

4 Change Robustness

In principle, testing integrity might be very costly as it requires reasoning within the external ontology. In order to avoid this, we propose a heuristic change detection procedure that analyzes changes with respect to their impact on compiled subsumption relations. Work on determining the impact of changes on a whole ontology is reported in [Heflin and Hendler, 2000]. As our goal is to determine whether changes in the external ontology invalidates compiled knowledge, we have to analyze the actual impact of changes on individual concept definitions. We want to classify these changes as either *harmless* or *harmful* with respect to compiled knowledge.

4.1 Determining Harmless Changes

As compiled knowledge reflects subsumption relations between query concepts, a harmless change is a set of modifications to an ontology that does not change these subsumption relations. Finding harmless changes is therefore a matter of deciding whether the modifications affect the subsumption relation between query concepts. We first look at the effect of a set of modifications on individual concepts:

Assuming that C represents the concept under consideration before and C' the concept after the change there are four ways in which the old version C may relate to the new version C' :

1. the meaning of concept is not changed: $C \equiv C'$ (e.g. because the change was in another part of the ontology, or because it was only syntactical);
2. the meaning of a concept is changed in such a way that concept becomes more general: $C \sqsubseteq C'$
3. the meaning of a concept is changed in such a way that concept becomes more specific: $C' \sqsubseteq C$
4. the meaning of a concept is changed in such a way that there is no subsumption relationship between C and C' .

The same observations can be made for a relation before and after a change, denoted as R and R' respectively. The next question is how these different types of changes influences the interpretation of query concepts. We take advantage of the fact that there is a very tight relation between changes in concepts of the external ontology and implied changes to the query concepts using these concepts:

Lemma 1 (Monotonicity of Effect) Let $c(Q)$ be the set of all concept names and $r(Q)$ the set of all relation names occurring in query Q , let further $C \in c(Q)$ and $R \in r(Q)$ then changing C has the same impact on the interpretation of Q as it has on the interpretation of C , in particular, we have $C \sqsubseteq C' \implies Q \sqsubseteq Q'$ and $C' \sqsubseteq C \implies Q' \sqsubseteq Q$ where Q' is the query as being interpreted after changing C . Analogously, a change of R has the same effect on the complete query.

Proof 2 (Sketch) The idea of the proof is the following: Queries contain conjuncts of the form $C(x)$ or $R(x, y)$. Conjuncts of the first form are interpreted as $\{x \mid x \in C^{\mathfrak{S}}\}$. It directly follows that changing the interpretation of the concept C referred to in a conjunct of this type leads to the same change on the interpretation of the conjunct and because conjunction is interpreted as set intersection the whole query. Conjuncts of the second type are interpreted as $\{x \mid \exists y : (x, y) \in R^{\mathfrak{S}}\}$. The variable y can be further constraint by a conjunct of the first type. Again changes in the interpretation of the concept that further restricts y have the same effect on possible interpretations of y and therefore also on the interpretation of conjuncts of the second type. Using the same argument, we see that making R more general/specific (allowing more/less tuples in the relation) makes conjuncts of the second form more general/specific. Using these basic conclusions, we can proof the lemma by induction over the lengths of the path in the dependency graph of the query where nodes represent conjuncts and arcs co-occurrence of variables.

We can exploit this relation between the interpretation of concepts and queries in order to identify the effect of changes in the external ontology on the subsumption relations between different query concepts. First of all the above result directly generalizes to multiple changes with the same effect, i.e. a query Q becomes more general(specific) or stays the same if none of the elements in $c(Q) \cup r(Q)$ become more specific(general). Further, the subsumption relation between two query concepts does not change if the more general(specific) query becomes even more general(specific) or stay the same. Combining these two observations, we derive the following characterization of harmless change.

Theorem 2 (Harmless Change) A change is harmless with respect to compiled knowledge (i.e. $Q_1 \sqsubseteq Q_2 \implies Q'_1 \sqsubseteq Q'_2$) if for all compiled subsumption relations $C_1 \sqsubseteq C_2$ where C_i is defined by query Q_i we have:

- $X' \sqsubseteq X$ for all $X \in c(Q_1) \cup r(Q_1)$
- $X \sqsubseteq X'$ for all $X \in c(Q_2) \cup r(Q_2)$

Proof 3 We assume that $X' \sqsubseteq X$ for all $X \in c(Q_1) \cup r(Q_1)$. Applying lemma 1 with respect to all $X \in c(Q_1) \cup r(Q_1)$ we

derive $Q'_1 \sqsubseteq Q_1$. We further assume that $X \sqsubseteq X'$ for all $X \in c(Q_2) \cup r(Q_2)$. Using lemma 1 we get $Q_2 \sqsubseteq Q'_2$. This leads us to $Q'_1 \sqsubseteq Q_1 \sqsubseteq Q_2 \sqsubseteq Q'_2$. Theorem 2 is established by transitivity of the subsumption relation.

The theorem provides us with a correct but incomplete method for deciding whether a change is harmless. This basic method can be refined by analyzing the overlap of $c(Q_1)$ and $c(Q_2)$ in combination with the relations they restrict. This more accurate method is not topic of this paper, but it relies on the same idea as the theorem given above.

4.2 Characterizing Changes

Now we are able to determine the consequence of changes in the concept hierarchy on the integrity of the mapping, we still need to know what the effect of specific modifications on the interpretation of a concepts is (i.e. whether it becomes more general or more specific). As our goal is to determine the integrity of mappings without having to do classification, we describe what theoretically could happen to a concept as result of a modification in the ontology. To to so, we have listed all possible change operations to an ontology according to the OWL-lite³ knowledge model in the same style as done in [Banerjee *et al.*, 1987]. The list of operations is extendable to other knowledge models; we have chosen the OWL-lite model because of its simplicity and its expected important role on the Semantic Web. Apart from *atomic change operations* to an ontology — like add range restriction or delete subclass relation — the list also contains some *complex change operations*, which consist of multiple atomic operations and/or incorporate some additional knowledge. The complex changes are often more useful to specify effects than the basic changes. For example, for operations like concept moved up, or domain enlarged, we can specify the effect more accurately than for the atomic operations subclass relation changed and domain modified⁴. Atomic changes can be detected without using the knowledge in the ontology itself, only using the knowledge of the knowledge model, i.e. the language. These changes are detected at a structural level. To identify complex changes, we also need to use the content of the ontology itself. We are currently working on rules and heuristics to distill complex changes from sets of atomic changes [Klein and Noy, 2003]. Table 1 contains some examples of operations and their effect on the classification of concepts. The table only shows a few examples, although our full ontology of change operations contains around 120 operations. This number is still growing as new complex changes are defined. A snapshot of the change ontology can be found online.⁵ The specification of effects is not complete, in the sense that it describes “worst case” scenario’s, and that for some operations the effect is “unknown” (i.e. unpredictable). In contrast to [Franconi *et al.*, 2000] who provides complete semantics of changes we prefer to use heuristics in order to avoid expensive reasoning about the impact of changes.

³See <http://www.w3.org/TR/owl-features/>.

⁴For a complete list, see <http://wonderweb.man.ac.uk/deliverables/D20.shtml>.

⁵<http://ontoview.org/changes/1/3/>

Operation	Effect
Attach a relation to concept C	C : Specialized
Complex: Change the superclass of concept C to a concept lower in the hierarchy	C : Specialized
Complex: Restrict the range of a relation R (effect on all C that have a restriction on R)	R : Specialized, C : Specialized
Remove a superclass relation of a concept C	C : Generalized
Change the concept definition of C from primitive to defined	C : Generalized
Add a concept definition A	C : Unknown
Complex: Add a (not further specified) subclass A of C	C : No effect
Define a relation R as functional	R : Specialized

Table 1: Some modification to an ontology and their effects on the classification of concepts in the hierarchy.

4.3 Update Management

With the elements that we described in this section, we now have a complete procedure to determine whether compiled knowledge in other modules is still valid when the external ontology is changed. The complete procedure is as follows:

1. create a list of concepts and relations that are part of the “subsuming” query of any compiled axiom;
2. create another list of concepts and relations that are part of the “subsumed” query of any compiled axiom;
3. achieve the modifications that are performed in the external ontology;
4. use the modifications to determine the effect on the interpretation of the concept and relations.
5. check whether there are concepts or relations in the first, “subsuming”, list that became more specific, or concepts or relations in the second, “subsumed”, list that became more general, or concepts or relations in any of the lists with an unknown effect; if not, the integrity of the mapping is preserved.

Algorithm 2 Update

Require: Ontology Module M

Require: Ontology Module M_j

```

for all compiled axioms  $C_1 \sqsubseteq C_2$  in  $M^c$  do
  for all  $X \in c(Q_1) \cup r(Q_1)$  do
    if effect on  $C$  is ‘generalized’ or ‘unknown’ then
       $M^c := Compile(M, M_j)$ 
    end if
  end for
  for all  $X \in c(Q_2) \cup r(Q_2)$  do
    if effect on  $X$  is ‘specialized’ or ‘unknown’ then
       $M^c := Compile(M, M_j)$ 
    end if
  end for
end for

```

We describe the procedure in a more structured way in Algorithm 2. The algorithm triggers a (re-)compilation step only if it is require in order to resume integrity. Otherwise

no action is taken, because the previously compiled knowledge is still valid. All the steps can be automated. A tool that helps to automate step 3 and 4 is described in [Klein *et al.*, 2002]. This tool will compare two versions of an ontology and derive the list of change operations that is necessary to transform the one into the other. It will also be able to detect some of the *complex* operations. The tool will also annotate the definitions in an ontology with the effect that the change has on its place in the hierarchy.

5 Conclusions

There is a growing need for applying the principle of modularity to representations of ontological knowledge in order to facilitate the creation, maintenance and re-use of knowledge. This paper contributes to the development of a theory of modular ontologies, focussing on the issue of reasoning in modular ontologies that change over time. The contributions of this paper is three-fold:

1. We propose an architecture for modular ontologies and analyze the role of mappings in logical reasoning across modules.
2. We describe a knowledge compilation approach that makes local reasoning within modules possible and define the notion of integrity.
3. We develop an update strategy that preserves integrity by identifying changes in ontology modules and deciding whether the compiled knowledge has to be updated or not.

We think that the approach described meets the practical needs of creating and using ontologies without missing a formal underpinning. It uses well-established representations of ontological knowledge and a rather simple and intuitive representation of mappings. Further, all of the supporting methods described can be automated in order to assist ontology engineers and developers of ontology-based systems. We deliberately chose to make some simplifications in order to be able to develop concise methods. First of all, these simplifications concern the restriction to a system of only two modules and the use of a rather weak heuristic for determining the effect of changes on compiled knowledge. In future work, we will investigate the impact of these simplifications and try to develop a more complete theory of the interaction in complex systems of modules and of the impact of changes on derived knowledge. Further one can as well imagine an external definition of relations using conjunctive queries with more than one free variable and reducing implied subsumption to the general problem of query containment under constraints [Calvanese *et al.*, 1998].

References

- [Baader *et al.*, 2003] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic Handbook - Theory, Implementation and Applications*. Cambridge University Press, 2003.
- [Banerjee *et al.*, 1987] Jay Banerjee, Won Kim, Hyoung-Joo Kim, and Henry F. Korth. Semantics and Implementation of Schema Evolution in Object-Oriented Databases. *SIGMOD Record (Proc. Conf. on Management of Data)*, 16(3):311–322, May 1987.
- [Borgida and Serafini, 2002] A. Borgida and L. Serafini. Distributed description logics: Directed domain correspondences in federated information sources. In *Proceedings of the International Conference on Cooperative Information Systems*, 2002.
- [Cadoli and Donini, 1997] M. Cadoli and F.M. Donini. A survey on knowledge compilation. *AI Communications*, 10(3-4):137–150, 1997.
- [Calvanese *et al.*, 1998] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. On the decidability of query containment under constraints. In *Proc. of the 17th ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS'98)*, pages 149–158, 1998.
- [Donini *et al.*, 1996] F.M. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. Reasoning in description logics. In Gerhard Brewka, editor, *Principles of Knowledge Representation*, pages 193–238. CSLI Publications, 1996.
- [Franconi *et al.*, 2000] Enrico Franconi, Fabio Grandi, and Federica Mandreoli. A semantic approach for schema evolution and versioning in object-oriented databases. In *Computational Logic 2000*, number 1861 in Lecture Notes in Computer Science, pages 1048 – 1062, 2000.
- [Gomez-Perez and Corcho, 2002] A. Gomez-Perez and O. Corcho. Ontology languages for the semantic web. *IEEE Intelligent Systems*, January/February:54–60, 2002.
- [Halevy, 2001] A.Y. Halevy. Answering queries using views: A survey. *The VLDB Journal*, 10(4):270–294, 2001.
- [Heflin and Hendler, 2000] Jeff Heflin and James Hendler. Dynamic ontologies on the web. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000)*, pages 443–449. AAAI/MIT Press, Menlo Park, CA, 2000.
- [Horrocks and Tessaris, 2000] I. Horrocks and S. Tessaris. A conjunctive query language for description logic aboxes. In *AAAI/IAAI*, pages 399–404, 2000.
- [Klein and Noy, 2003] Michel Klein and Natalya F. Noy. A component-based framework for ontology evolution. Technical Report IR-504, Department of Computer Science, Vrije Universiteit Amsterdam, March 2003.
- [Klein *et al.*, 2002] Michel Klein, Atanas Kiryakov, Damyan Ognyanov, and Dieter Fensel. Ontology versioning and change detection on the web. In *13th International Conference on Knowledge Engineering and Knowledge Management (EKAW02)*, Sigüenza, Spain, October 1–4, 2002.
- [McIlraith and Amir, 2001] S. McIlraith and E. Amir. Theorem proving with structured theories. In B. Nebel, editor, *Proceedings of IJCAI'01*, pages 624–634, San Mateo, August 2001. Morgan Kaufmann.
- [Rector, 2003] A. Rector. Modularisation of domain ontologies implemented in description logics and related formalisms including OWL. In *Proceedings of the 16th International FLAIRS Conference*. AAAI, 2003.

