Logic Based Knowledge Representation

Prof. Franz Baader
Chair for Automata Theory, TU Dresden
transcript by Andreas Witzel
revised by Jan Hladik

Contents

1 Motivation and Overview

The goal of artificial intelligence (AI) is to produce systems with an "intelligent behaviour". For humans, intelligent behaviour is strongly based on the use of acquired knowledge (e.g., intelligence tests also refer to your knowledge). Consequently, knowledge representation (KR) plays an important role in AI. In contrast to "classical" programming, the relevant knowledge is represented explicitly and symbolically in knowledge-based programming.

1.1 Classical programming vs. knowledge-based programming

In AI, "classical programming" is usually replaced by knowledge-based programming. In contrast to classical programming, in KB-programming the relevant information is represented explicitly and symbolically.

Classical programming is characterized by the following properties:

- A specialized program that is tailored to a specific application;
- knowledge about the problem domain and additional constraints are *implicit* in the structure of the program;
- therefore, knowledge is "introduced" by the programmer;
- if there are changes in the application, the program has to be rewritten.

Knowledge-based programming has the following properties:

- Use of general problem solvers that are independent of the specific domain;
- knowledge about the application domain is represented *explicitly* in a knowledge base;
- this knowledge can be acquired by a knowledge engineer, independent of the problem solvers;
- if there are changes in the application, only the knowledge base needs to be changed, but not the problem solver.

Example 1.1. We give examples for programs that determine whether two nodes of the following (acyclic) graph are connected:

$$
\begin{tikzcd}
 & & a \arrow[dl] \arrow[dr] & \\
 & b \arrow[dl] \arrow[dr] & & c \arrow[dl] \\
d & & e &
\end{tikzcd}
$$

A "classical" program might look like this:

```
procedure connected(x, y)
    if x = a then
        if y = b or y = c then
            return true
        else
            return (connected(b, y) or connected(c, y))
    else
        if x = b then
            if y = d or y = e then
                return true
            else
                return false
        else
            if x = c then
                if y = e then
                    return true
                else
                    return false
            else
                return false
```

In the following solution, the knowledge about the graph and the meaning of "connected" are represented explicitly:

Knowledge base:

Factual knowledge:

```
directly-connected(a, b)
directly-connected(a, c)
directly-connected(b, d)
directly-connected(b, e)
directly-connected(c, e)
```

Structural knowledge:

```
connected(x, y) if directly-connected(x, y)
connected(x, y) if directly-connected(x, z) and connected(z, y)
```

Query:

```
connected(a, e)?
```

The query is then answered by a general problem solver.

The distinction is in practice not as strict as in this example. "Classical" programs also represent parts of the application as explicit data structures, i.e.

the knowledge about the graph could be represented explicitly, whereas the meaning of "connected" would be encoded implicitly in the program.

On the other hand, many (knowledge-based) expert systems encode the knowledge about an application implicitly using (procedural) rules.

Prerequisites for knowledge-based programming. The relevant knowledge can in some way be represented *symbolically*, and thus stored in a computer.

1.2 Basic Tasks in KR

KR can be divided into three important subtasks:

1. Knowledge acquisition

2. Symbolic representation

3. Work with and apply the represented knowledge

ad 1) Knowledge acquisition

Acquire knowledge about the application domain that is relevant for solving the problem at hand, e.g. by interviewing experts.

This is a very complicated task which is similar to the problem of going from an informal task description to a formal specification of a program. We will not deal with this issue in the current course.

ad 2) Symbolic representation

What is the exact connection between the symbolic representation and the real world entities it is supposed to describe?

What formalism is used to represent the knowledge?

What is the meaning (the semantics) of the chosen formalism?

Expressive power

Is the expressive power of the chosen formalism adequate?

- Can all relevant knowledge be represented?
- Are all provided representational means really necessary?

Requirements on the semantics

- *no* procedural semantics, i.e. the semantics should *not* just be defined by what the programs working on it do with it;
- instead *declarative semantics*
  - maps the symbolic expressions to an abstraction of the world to be represented
  - notion of "truth" that allows us to determine whether a symbolic statement is true in the world under consideration.

ad 3) Work with and apply the represented knowledge

Similar to database systems, it must be possible to manipulate the stated knowledge (*knowledge management*):

- add new knowledge to already existing knowledge bases
- remove knowledge
- access the (explicitly) stored knowledge

An additional demand to KR systems is that one wants to access the knowledge that is only implicitly represented. For this purpose, KR systems need *inference mechanisms* to deduce implicit knowledge from the explicitly represented knowledge.

Example 1.1 (cont'd).

`connected`(a, c) is implicit knowledge while `directly-connected`(a, b) is explicitly stored.

1.3 Inference Mechanisms

Central requirement on the inference mechanisms:

- results should only depend on the semantics of the representation formalism,
- but *not* on the syntactic representation: semantically equivalent, but syntactically different knowledge bases yield the same inferences.

Example 1.1 (cont'd). The inference mechanism of Prolog does *not* satisfy this requirement if we use the semantics of first-order predicate logic (FOL) for Prolog programs, since the answer given depends on the syntax (more precisely, on the sequence of the clauses):

- Prolog gives correct positive and negative answers:
  - `connected(x,y)` **`if`** `directly-connected(x,y)`
  - `connected(x,y)` **`if`** `directly-connected(x,z)` **`and`** `connected(z,y)`
- Prolog sometimes gives the correct positive answer; it does not terminate if the nodes are not connected:
  - `connected(x,y)` **`if`** `directly-connected(x,y)`
  - `connected(x,y)` **`if`** `connected(z,y)` **`and`** `directly-connected(x,z)`
- Prolog does not terminate
  - `connected(x,y)` **`if`** `connected(z,y)` **`and`** `directly-connected(x,z)`
  - `connected(x,y)` **`if`** `directly-connected(x,y)`

Important properties for the evaluation of inference mechanisms:

1. Soundness: positive answers are always correct

   for example, answering the query connected(b, c) with "yes" is unsound (incorrect) for the graph from Example 1.1.

2. Completeness: positive answers are always found

   for example, if an inference mechanism does not answer the query connected(a, e) with "yes", then it is incomplete.

3. Decidability: is the inference problem decidable?

   i.e. is there an algorithm that terminates on any input and yields correct positive and negative answers?

   An inference problem can also be *partially decidable*: there is a procedure that always terminates answering "yes" for questions with positive answers; for questions with negative answers, the procedure need not terminate.

Side remark: In KR, one usually wants terminating inference algorithms (the user is not willing to wait forever for the answer). Therefore, in this lecture only decision procedures will be called sound and complete. In contrast, in FOL the inference problems are usually only partially decidable. Therefore, in that area also partial decision procedures are called sound and complete.

1.4 Complexity

How many computation steps does a decision procedure need to compute a result? How much memory does it consume?
Runtime and memory usage is described as a function of the size of the input.

- Worst case complexity

  How long (or how much memory) does it take to handle the hardest type of query and knowledge base?

  Are these cases relevant in practice?

- Average case complexity

  How much time (or space) is needed on the average?

  Average-case analysis is much harder than worst-case analysis. Where do suitable probability distributions come from?

In KR, researchers were initially looking for algorithms that need only polynomial time in the worst case (*tractable*). However, it has turned out that this poses too severe restrictions on expressive power (not even propositional logic is tractable).

There are two possible ways out:

(a) Drop completeness, i.e. the algorithm sometimes answers "no" although the correct answer is "yes".

(b) Use highly optimized implementations of (sound and complete) algorithms and live with high worst-case complexity.

Since (a) contradicts our requirement that the behaviour of the system should only depend on the semantics, we will pursue (b) in this lecture.

Summary. In this lecture, a KR system is determined by the following:

Syntax: formal language that describes what admissible entries of the knowledge base look like

Semantics: explains the meaning of syntax expressions; should be declarative

System services: inference mechanisms provided to the user (human or machine); should be defined (only!) via the semantics.

1.5 Possible Role of Logic

Usually logic in KR systems has a number of benefits:

- It provides us with a formal and declarative semantics;
- system services can be formally defined as logical reasoning problems;
- known methods from logic can be used when devising algorithms for system services.

When we talk of logic, we usually *do not* mean classical FOL since it is *not* directly suited as a KR mechanism:

- its syntax is not user-friendly;
- on the one hand, FOL is so expressive that inference problems are undecidable
  - validity and satisfiability of FOL-formulas are undecidable;
  - validity is partially decidable: there are algorithms that enumerate all valid formulas (theorem provers);
- on the other hand, FOL can only represent objective, static, and certain knowledge. In order to represent commonsense knowledge, one also needs to represent subjective, time-dependent, and uncertain knowledge.

In this lecture, three logic-based families of KR-formalisms are introduced that overcome the disadvantages of FOL:

1. Description Logics

- A family of KR-formalisms that allow the description of “notions” (*concepts*)
- Most DLs can be understood as fragments of FOL, but they have
  - a more user-friendly, variable-free syntax,
  - decidable inference problems,
  - less expressive power.

2. Modal Logics
   - A family of logics developed in philosophy for the representation of
     - subjective knowledge, e.g. “Bill believes that Monica knows that he loves her”;
     - time-dependent knowledge, e.g. “At some time in the future, Bill will love Monica.”
   - From a formal perspective, modal logics are closely related to description logics.
3. Non-monotonic logics
   - are used for representation and processing of uncertain and incomplete knowledge;
   - draw “plausible” conclusions that need not be correct: it may be necessary to retract them if new information becomes available (hence “non-monotonic”: implicit knowledge does *not* increase if explicit knowledge increases).

Example 1.2 (Closed-world assumption). By default, one often assumes that the given information is complete. In terms of Example 1.1, this means:

If `connected`(b, c) does not follow from the KB, then it is allowed to deduce $\neg$`connected`(b, c) – although this is not a logical consequence in the sense of FOL. Of course, $\neg$`connected`(b, c) is not a valid deduction any more if we add the knowledge `directly-connected`(b, c).

Thus making the closed-world assumption yields a non-monotonic formalism.

Note: The CWA is often made by humans, e.g. when reading a train connection table.

2 Early KR formalisms

2.1 Semantic networks

Semantic networks were developed by M.R. Quillian in 1968 for the representation of the contents of dictionaries. The name is due to the intended application of this formalism: describe the semantics of natural language. There existed a variety of implemented systems, mostly for natural language processing.

Semantic networks are graphs with labeled nodes and labeled edges. Figure 1 shows an example.

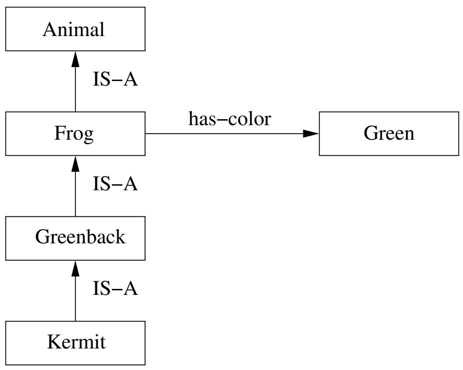


Figure 1: A semantic network

Nodes represent objects, concepts, and situations.

Edges represent relationships between nodes.

Property edges assign properties to objects, concepts, or situations, e.g. the property of "being green" is assigned to the concept "frog"

IS-A edges represent

- hierarchical organization of concepts, e.g. "greenback" is a subconcept of "frog";
- instantiation of concepts with objects, e.g. "Kermit" is an instance of the concept "greenback";
- inheritance of properties, e.g. the concept "greenback" inherits the property of being green from its superconcept "frog."

But: not all properties are necessarily inherited; e.g. the a grass frog does not inherit the property of being green.

"Semantics" of semantic networks.

- There exists no formal semantics.

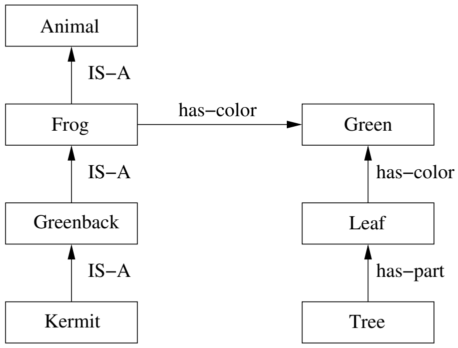


Figure 2: Spreading activation in a semantic network

- The meaning of semantic networks is defined only through algorithms operating on them.
- Identical networks can yield different results in different systems.

There is no common procedure for processing semantic networks. Frequently used techniques are "spreading activation" and "matching".

Spreading Activation. Purpose: detect relationships between pairs of concepts (see Figure 2)

- Beginning with the two nodes (e.g. "Greenback" and "Tree") to be compared, other nodes reachable via edges are "activated" in a wave-like fashion.
- If a node is activated by both "waves", then a relationship has been found ("Green").
- In Quillian's system, the paths relating the two original nodes to the intersection node were used to generate an output in "natural language." In our example: "A greenback is a frog, which has the color green. A tree has a part which is a leaf, which has the color green."

Matching. In order to answer a question to a semantic network, we construct a *query network* that describes the desired object or concept. This query network is then searched for as a sub-structure in the semantic network.

Example. (see Figure 3)

- Which color do frogs have?
  $\boxed{\text{Frog}} \xrightarrow{\texttt{has-color}} \boxed{?}$
- Which color do greenbacks have?
  $\boxed{\text{Greenback}} \xrightarrow{\texttt{has-color}} \boxed{?}$

To answer the query, a simple sub-graph matching does not suffice: due to inheritance, we must first modify the original network.

- Which color do grass frogs have?

  $\boxed{\text{Grass frog}} \xrightarrow{\texttt{has-color}} \boxed{?}$

  To answer this question, a conflict has to be solved since the color brown is stated explicitly, but the color green is inherited.

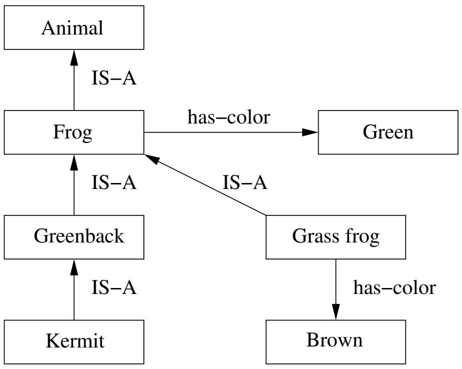


Figure 3: A semantic network with a conflict

Criticism: Due to the lack of formal semantics, the meaning of a given network is not clear.

- Meaning of nodes

  There is no clear distinction between concepts (classes of objects) and objects.

- Meaning of IS-A edges
  - They describe concept hierarchy as well as instantiation.
  - Are they denoting a strict subconcept-superconcept relationship, or are they a relation that holds “typically”, but with exceptions?

- Meaning of property edges

  What does $\boxed{\text{Frog}} \xrightarrow{\texttt{has-color}} \boxed{?}$ really mean?
  - A frog has only the color green (*value restrictions*).
  - A frog has at least the color green, but may have other colors as well (*existential restriction*).
  - A frog typically has the color green, but there may be exceptions (*default property*).

A partial logical reconstruction of the semantics of semantic networks has been given by Schubert (1979).

2.2 Frames

This notion has been introduced by Marvin Minsky as a name for data structures that represent prototypical objects such as "chair" and "door", and situations such as "entering a restaurant" or "buying a book".

Underlying idea: To interpret a situation and react in an appropriate way, humans use knowledge about similar situations that they have already experienced. From these experienced situations, they extract "standard expectations" for the current situation.

Intuitively, a frame collects all relevant knowledge about an object/situation. This is then used to "understand" new data, e.g. in image recognition.

Example 2.1. Even if we enter a specific restaurant for the first time, we have rather clear ideas about what to expect.

A frame has a name and consists of several *slots*, which are *filled* with *values*. Values can again be frames.

Generic Frames represent prototypical objects and situations. They have *generic slots* that may have several types such as:

- *specialization-of*: describes a hierarchical structure between frames and is used for the inheritance of properties;
- *fixed generic value* which cannot be overwritten;
- *default values* that are assumed unless we have more specific knowledge;
- *attached procedures* (if-needed): algorithms for computing values if they are needed.

Instance Frames represent actual situations and objects. Are linked to generic frames via the *specialization-of* slot.

Frames were originally not equipped with a formal semantics, which produced similar problems as for semantic networks. A (partial) logical reconstruction of the meaning of frame systems (without default slots and attached procedures) has been introduced by P. Hayes in 1977.

A generic frame for a restaurant is shown in Figure 4.

Inference Mechanisms for Frames. Selection of a frame that best fits a given situation: *matching.*

- The aim is to go from an instance frame that describes the current situation to a generic frame (to answer the question "In which situation are we?").
- For this purpose, values of the instance frame are filled in, using

```
Generic Restaurant Frame
    Specialization-of: Business-Establishment
       Types:       range:       (Cafeteria, Fast-Food, Seat-Yourself,
                                 Wait-to-be-Seated)
                    default:     Wait-to-be-Seated
                    if-needed:   IF plastic-orange-counter THEN Fast-Food
                                 IF stack-of-trays THEN Cafeteria
                                 IF wait-for-waitress-sign THEN Wait-to-be-seated
                                 OTHERWISE Seat-yourself
       Location:    range:       an ADDRESS
                    if-needed:   (Look at the Menu)
       Name         if-needed:   (Look at the Menu)
       Food-Style   range        (Burgers, Chinese, American, Seafood)
                    default:     American
```

Figure 4: A generic restaurant frame

  - values that are already known (observed in the current situation);
  - default values;
  - attached procedures;
  - values from matching generic frame.

- The matching process is guided by the "goal of the system".
- Frames may also contain information on what to do if matching fails.

2.3 Non-monotonic inheritance networks

This knowledge representation formalism goes back to S.E. Fahlmann (1979). An inheritance network consists of nodes, positive IS-A edges ($x \to y$), and negative IS-NOT-A edges ($x \not\to y$). There are no property edges.

The non-monotonism stems from the fact that edges are viewed as *defeasible*, i.e., they can be refuted by other information. Thus they mean "is usually a" and "is usually not a" rather than "is always (not) a".

Inferences are defined using the notion of *paths*:

- A path $x_0 \to x_1 \to \cdots \to x_{n-1} \to x_n$ of a network *supports* the conclusion $x_0 \to x_n$.
- A path $x_0 \to x_1 \to \cdots \to x_{n-1} \not\to x_n$ of a network *supports* the conclusion $x_0 \not\to x_n$.

Example 2.2. (see Figure 5, left)

- Kermit $\to$ Grass frog $\to$ Frog $\to$ Green being

  supports

  Kermit $\to$ Green being.

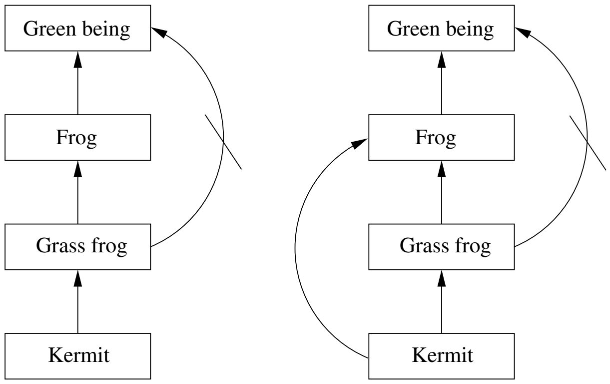


Figure 5: Examples for inheritance networks

- Kermit $\to$ Grass Frog $\not\to$ Green being

  supports

  Kermit $\not\to$ Green being.

But: what to do with such contradictory information?

A network contains a *contradiction* if there are two nodes x and y such that both $x \to y$ and $x \not\to y$ are supported.

Idea for resolving contradictions. *More specific* information is decisive. For Kermit, the information

Grass frog $\not\to$ Green being

is more specific than

Grass frog $\to$ Frog $\to$ Green being.

Fahlmann implemented this idea in his system NETL by searching for the shortest paths.

This formalism is non-monotonic: if the network is extended, previously supported consequences may become unsupported: The network

$\boxed{\text{Grass frog}} \to \boxed{\text{Frog}} \to \boxed{\text{Green being}}$

supports the conclusion "Grass frog $\to$ Green being", but when we add

$\boxed{\text{Grass frog}} \not\to \boxed{\text{Green being}}$

to the network, it does not support this conclusion any longer, since the new shorter path is decisive.

Remaining Contradictions: What do we do if the contradictory paths are of equal length?

Sceptical approach: If contradictions cannot be resolved, do not accept any of the two conclusions.

Credulous approach: View both conclusions as possible.

Problem. With the shortest path approach, redundant edges can have an impact on the set of accepted conclusions. In the right network in Figure 5, the path "Kermit $\to$ Grass frog $\not\to$ Green being" is not decisive anymore.

For this reason, Touretzky proposed in 1986 to make the following improvement to the "shortest path" approach:

Preemption of paths. The path $x \to \cdots \to y \to z$ is *preempted* by $x \to \cdots \to y' \not\to z$ if "y' is more specific information than y is", i.e. if there exists a path $y' \to \cdots \to y$ which is not preempted itself. Analogously, negative paths can be preempted by positive paths.

In the example, the path

Kermit $\to$ Frog $\to$ Green being

is preempted by the path

Kermit $\to$ Grass Frog $\not\to$ Green being

due to the presence of the path

Grass frog $\to$ Frog.

Criticism:

- The approach is still path-based, i.e. depends on the syntactic structure of the network.
- An exact definition of preemption is relatively complicated. In the literature, there is no agreement on what is the correct definition.
- For each of the definitions considered in the literature, there are examples for which these definitions do not yield the expected result.

Figure 6 shows an example of a contradiction which cannot be resolved by preemption.

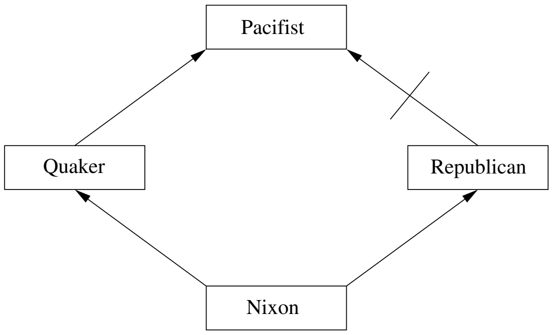


Figure 6: Nixon diamond

3 Description Logics

(Other names: terminological logics, concept description languages, KL-ONE-like languages.)

The purpose of description logics is to define the relevant notions/concepts (the terminology) of an application. This is done by

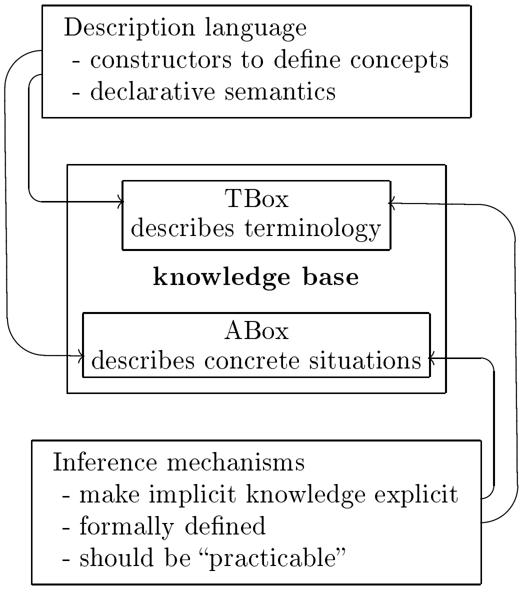
- describing the relationship between concepts;
- describing the relationship between objects and concepts;
- relating objects with objects.

The origin is a work of Ron Brachman (1978) about structured inheritance networks, which tried to overcome the problems caused by the lack of formal semantics of semantic networks and frames.

The first DL system was KL-ONE (Brachman 1979). There exist many successor systems based on the same ideas: **Back, Classic, Crack, Krypton, K-Rep, Kris, Loom**. The differences are in the expressive power of the languages and in the implementation of system services (more or less complete).

Fact (Horrocks) and **Racer** (Haarslev/Möller) are state-of-the-art systems.

Common to most DL systems is the following architecture:



3.1 Description Language

Different systems provide different constructors for concepts. As a prototypical example, we consider the language $\mathcal{ALCN}$ (=*a*ttributive *l*anguage with *c*omplement and *n*umber restrictions).

We start with introducing the syntax of $\mathcal{ALCN}$ concepts. *Concepts* are the formal objects that represent the informal notions from the application domain (concepts define them formally). They correspond to nodes in semantic networks or generic frames.

Additionally, we use binary relations called *roles* that correspond to property edges of semantic networks or slots in frames.

Definition 3.1 (Syntax of $\mathcal{ALCN}$ concepts). Let N_R and N_C be disjoint sets of *role names* and *concept names*. The set of *$\mathcal{ALCN}$-concept terms* is defined inductively as follows:

1. Each concept name $A \in N_C$ is an $\mathcal{ALCN}$-concept term;
2. If C, D are $\mathcal{ALCN}$-concept terms, $r \in N_R$, and $n \in \mathbb{N}$, then the following are also $\mathcal{ALCN}$-concept terms:
   - $\bot$, $\top$ (Bottom, Top)
   - $C \sqcap D$, $C \sqcup D$, $\neg C$ (Boolean Operators)
   - $\forall r.C$, $\exists r.C$ (value restriction and existential restriction)
   - $(\geq n\ r)$, $(\leq n\ r)$ (number restrictions; at-least-restriction, at-most-restriction)

For example, let `Human` and `Female` be concept names, and let `has-child` be a role name. Then

- `Human` $\sqcap \neg$`Female` describes the notion "male";
- $\exists$ `has-child.Human` describes the notion "parent";
- `Human` $\sqcap$ $\forall$`has-child.Female` describes "humans having only daughters (but possibly no children at all)";
- `Human` $\sqcap$ ($\geq$3 `has-child`) describes "humans having at least three children".

Formally, we define the semantics of the constructors using interpretations (which can be viewed as first-order structures with only unary and binary predicates).

Definition 3.2 (Semantics of $\mathcal{ALCN}$-concept terms). An *interpretation* $\mathcal{I}$ consists of a non-empty *interpretation domain* $\Delta^{\mathcal{I}}$ and an interpretation function $\cdot^{\mathcal{I}}$, which

- assigns to each $A \in N_C$ a subset $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$;
- assigns to each $r \in N_R$ a binary relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$.

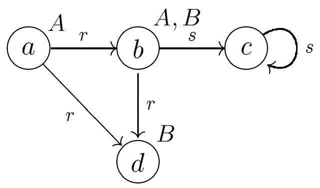
(Concept names are interpreted as sets of objects/individuals; role names are interpreted as binary relations.)

The interpretation function is inductively extended to (complex) $\mathcal{ALCN}$ concept terms as follows:

$$
\begin{aligned}
\bot^{\mathcal{I}} &:= \emptyset \\
\top^{\mathcal{I}} &:= \Delta^{\mathcal{I}} \\
(C \sqcap D)^{\mathcal{I}} &:= C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
(C \sqcup D)^{\mathcal{I}} &:= C^{\mathcal{I}} \cup D^{\mathcal{I}} \\
(\neg C)^{\mathcal{I}} &:= \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \\
(\forall r.C)^{\mathcal{I}} &:= \{d \in \Delta^{\mathcal{I}} \mid \forall e \in \Delta^{\mathcal{I}} : (d,e) \in r^{\mathcal{I}} \Rightarrow e \in C^{\mathcal{I}}\} \\
(\exists r.C)^{\mathcal{I}} &:= \{d \in \Delta^{\mathcal{I}} \mid \exists e \in \Delta^{\mathcal{I}} : (d,e) \in r^{\mathcal{I}} \wedge e \in C^{\mathcal{I}}\} \\
(\geq n\ r)^{\mathcal{I}} &:= \{d \in \Delta^{\mathcal{I}} \mid |\{e \in \Delta^{\mathcal{I}} \mid (d,e) \in r^{\mathcal{I}}\}| \geq n\} \\
(\leq n\ r)^{\mathcal{I}} &:= \{d \in \Delta^{\mathcal{I}} \mid |\{e \in \Delta^{\mathcal{I}} \mid (d,e) \in r^{\mathcal{I}}\}| \leq n\}
\end{aligned}
$$

Example 3.3. An interpretation $\mathcal{I}$ for concept names A, B and role names r, s.

$$
\begin{array}{rcl}
\Delta^{\mathcal{I}} & := & \{a,b,c,d\} \\
A^{\mathcal{I}} & := & \{a,b\} \\
B^{\mathcal{I}} & := & \{b,d\} \\
s^{\mathcal{I}} & := & \{(b,c),(c,c)\} \\
r^{\mathcal{I}} & := & \{(a,b),(a,d),(b,d)\}
\end{array}
$$



$$
\begin{array}{rcll}
(\neg A)^{\mathcal{I}} & = & \Delta^{\mathcal{I}} \setminus \underbrace{\{a,b\}}_{=A^{\mathcal{I}}} & = \{c,d\} \\
(\neg A \sqcap B)^{\mathcal{I}} & = & \{c,d\} \cap \{b,d\} & = \{d\} \\
(\forall r.B)^{\mathcal{I}} & = & \{a,b,c,d\} & \\
(\forall r.A)^{\mathcal{I}} & = & \{c,d\} & \\
(\exists s.\neg A)^{\mathcal{I}} & = & \{b,c\} & \\
(\exists s.A)^{\mathcal{I}} & = & \emptyset &
\end{array}
$$

3.2 The Terminology (TBox)

The TBox defines concept names as abbreviations for complex concept terms.

Definition 3.4 (Syntax).

1. If $A \in N_C$ and C is a concept term, then $A \doteq C$ is a concept definition.
2. A finite set $\mathcal{T}$ of concept definitions is called *TBox* if it does not contain

- multiple definitions or
- cyclic definitions.

Multiple definitions: There are $A \in N_C$ and distinct concept terms C, D such that $\{A \doteq C, A \doteq D\} \subseteq \mathcal{T}$.

Cyclic definitions: There are $n \geq 1$ and concept definitions $A_1 \doteq D_1, \ldots, A_n \doteq D_n$ in $\mathcal{T}$ such that

- D_i contains A_{i+1} for $1 \leq i < n$;
- D_n contains A_1.

The two limitations ensure that concept definitions are really just definitions for concept terms (i.e., macros that can easily be expanded, see Theorem 3.6).

Notation

- *Defined concept*: a concept whichoccurs on the left-hand side of a concept definition.
- *Primitive concept*: a concept which does not occur on the left-hand side of a concept definition.

Example.

$$
\begin{array}{rcl}
\mathtt{Male} & \doteq & \neg\mathtt{Female} \\
\mathtt{Woman} & \doteq & \mathtt{Human} \sqcap \mathtt{Female} \\
\mathtt{Man} & \doteq & \mathtt{Human} \sqcap \mathtt{Male} \\
\mathtt{Mother} & \doteq & \mathtt{Woman} \sqcap \exists \mathtt{has\text{-}child}.\mathtt{Human} \\
\mathtt{Father} & \doteq & \mathtt{Male} \sqcap \exists \mathtt{has\text{-}child}.\mathtt{Human}
\end{array}
$$

Definition 3.5 (Semantics). An interpretation $\mathcal{I}$ is a *model* of a TBox $\mathcal{T}$ iff we have $A^{\mathcal{I}} = C^{\mathcal{I}}$ for all $A \doteq C \in \mathcal{T}$.

Notation Two TBoxes $\mathcal{T}_1$ and $\mathcal{T}_2$ are *equivalent* iff they have the same models.

Theorem 3.6. For every TBox $\mathcal{T}$, there is an equivalent TBox $\hat{\mathcal{T}}$ such that only primitive concept names occur on the right-hand side of concept definitions in $\hat{\mathcal{T}}$.

Proof. Expansion of the definitions
$\mathcal{T}_0 := \mathcal{T}$ and $i := 0$
while a defined concept name occurs on the right-hand side of $\mathcal{T}_i$ **do**

1. choose an $A \doteq C \in \mathcal{T}_i$ such that a defined concept B occurs in C
2. let $B \doteq D$ be the definition of B in $\mathcal{T}_i$
3. replace all occurrences of B in C with D. Let $\hat{C}$ be the concept term obtained in this way

4. $\mathcal{T}_{i+1} := (\mathcal{T}_i \setminus \{A \doteq C\}) \cup \{A \doteq \hat{C}\}$; $i := i + 1$

od
$\hat{\mathcal{T}} := \mathcal{T}_i$

This expansion process always terminates (since $\mathcal{T}$ is acyclic) and yields an equivalent TBox (proof: exercise). □

Corollary 3.7. Let $\mathcal{T}$ be a TBox. A given interpretation of the primitive concept names and role names can be uniquely extended to a model of $\mathcal{T}$.

Proof. Construct the unfolded TBox $\hat{\mathcal{T}}$ that is equivalent to $\mathcal{T}$ (Theorem 3.6). Obviously, $\hat{\mathcal{T}}$ and $\mathcal{T}$ have the same primitive and defined concepts.

Let $\mathcal{I}$ be an interpretation that assigns, to each primitive $P \in N_C$, a set $P^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, and, to each $r \in N_R$, a relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. Extend $\mathcal{I}$ as follows: let $A \doteq \hat{C}$ be the definition of A in $\hat{\mathcal{T}}$. We set $A^{\mathcal{I}} = \hat{C}^{\mathcal{I}}$. This is well-defined since $\hat{C}$ contains only primitive concept names. By definition, $\mathcal{I}$ is a model of $\hat{\mathcal{T}}$. Since $\mathcal{T}$ and $\hat{\mathcal{T}}$ are equivalent, $\mathcal{I}$ is also a model of $\mathcal{T}$.

It thus remains to show the uniqueness of this extension. Let $\mathcal{J}$ be a model of $\mathcal{T}$ that coincides with $\mathcal{I}$ on the interpretation of primitive concept names and role names. Then $\mathcal{J}$ is also a model of $\hat{\mathcal{T}}$. Thus, for any defined concept name A, we have:

$$A^{\mathcal{J}} \quad = \quad \hat{C}^{\mathcal{J}} \underbrace{\quad = \quad}_{\text{since } \hat{C} \text{ contains only primitive concept names}} \hat{C}^{\mathcal{I}} \quad = \quad A^{\mathcal{I}}.$$

□

Example 3.8. In general, Corollary 3.7 does not hold for TBoxes with cycles:

$\mathcal{T} = \{\texttt{Momo} \doteq \texttt{Male} \sqcap \forall\texttt{has-child}.\texttt{Momo}\}$.

Intuition: Momo describes all *m*en having *o*nly *m*ale *o*ffspring (where "offspring" is the transitive closure of "`has-child`").

The interpretation $\mathcal{I}$ interprets only the primitive `Male` and the role `has-child`.

$$\text{Charles1} \xrightarrow{has-child} \text{Charles2} \xrightarrow{has-child} \text{Charles3} \xrightarrow{has-child} \cdots$$
(continues forever)

$$\text{James1} \xrightarrow{has-child} \text{James2} \xrightarrow{has-child} \ldots \xrightarrow{has-child} \text{JamesLast}$$

All Charles$i$ and James$i$ are `Male`.

Possible extensions of $\mathcal{I}$ to the defined concept `Momo`:

1. $\texttt{Momo}^{\mathcal{I}} = \{\text{James1}, \ldots, \text{JamesLast}\}$
2. $\texttt{Momo}^{\mathcal{I}} = \{\text{James1}, \ldots, \text{JamesLast}\} \cup \{\text{Charles}i \mid i \geq 1\}$

Note: JamesLast and thus *all* the James$i$ *must* belong to $\texttt{Momo}^{\mathcal{I}}$.

Exercise: Show that there are no other extensions that are models of $\mathcal{T}$.

Remarks:

1. This means that cyclic "definitions" are indeed no definitions since they do not characterize the concepts on the left-hand side *uniquely*.
2. This problem can be solved by using fixpoint semantics.

3.3 The Assertions (ABox)

Allow to introduce names for individuals, and to relate them with other individuals and with concepts.

Definition 3.9 (Syntax). Let N_I be a set of *individual names* that is disjoint to N_R and N_C.

1. *Assertions* are of the form
   - $a : C$ (concept assertion) and
   - $(a, b) : r$ (role assertion),

   where $a, b \in N_I$, $r \in N_R$, and C is a concept term.
2. A finite set of assertions is called *ABox*.

Example. Let `Bill` and `Monica` be individual names, and let `loves` and `has-child` be role names. Then

$\texttt{Bill} : (\geq 1\ \texttt{has-child})$

is a concept assertion, and

$(\texttt{Bill}, \texttt{Monica}) : \texttt{loves}$

is a role assertion.

Definition 3.10 (Semantics).

1. An interpretation $\mathcal{I}$ assigns, additionally, to each individual name $a \in N_I$, an element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$;

   This assignment is required to satisfy the *unique name assumption (UNA)*: for distinct individuals $a \neq b$, we have $a^{\mathcal{I}} \neq b^{\mathcal{I}}$.
2. An interpretation $\mathcal{I}$ is a *model* of the ABox $\mathcal{A}$ if
   - $a^{\mathcal{I}} \in C^{\mathcal{I}}$ for all $a : C \in \mathcal{A}$ and
   - $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$ for all $(a, b) : r \in \mathcal{A}$.

TBoxes and ABoxes can be combined straightforward: given a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$, we will often be interested in *common models* of $\mathcal{T}$ and $\mathcal{A}$.

Example 3.11. See Figure 7.

TBox:

$$
\begin{array}{rcl}
\texttt{Male} & \doteq & \neg\texttt{Female} \\
\texttt{Woman} & \doteq & \texttt{Human} \sqcap \texttt{Female} \\
\texttt{Man} & \doteq & \texttt{Human} \sqcap \texttt{Male} \\
\texttt{Mother} & \doteq & \texttt{Woman} \sqcap \exists\texttt{has-child}.\texttt{Human} \\
\texttt{Father} & \doteq & \texttt{Man} \sqcap \exists\texttt{has-child}.\texttt{Human} \\
\texttt{Parent} & \doteq & \texttt{Father} \sqcup \texttt{Mother} \\
\texttt{Grandmother} & \doteq & \texttt{Woman} \sqcap \exists\texttt{has-child}.\texttt{Parent} \\
\texttt{Fatherw/oSons} & \doteq & \texttt{Father} \sqcap \forall\texttt{has-child}.\texttt{Female} \\
\texttt{Motherw/oSons} & \doteq & \texttt{Mother} \sqcap \forall\texttt{has-child}.\texttt{Female} \\
\texttt{MotherOfOneChild} & \doteq & \texttt{Mother} \sqcap (\leq\ 1\ \texttt{has-child})
\end{array}
$$

ABox:

$$
\begin{array}{rcl}
\texttt{HILLARY} & : & \texttt{Woman} \\
\texttt{BILL} & : & \texttt{Man} \\
\texttt{HILLARY} & : & \texttt{MotherOfOneChild} \\
(\texttt{HILLARY}, \texttt{CHELSEA}) & : & \texttt{has-child} \\
(\texttt{BILL}, \texttt{CHELSEA}) & : & \texttt{has-child} \\
\texttt{MONIKA} & : & \texttt{Woman} \\
\texttt{CHELSEA} & : & \texttt{Woman}
\end{array}
$$

Figure 7: Examples for TBox and ABox

3.4 Inference Problems

Aims of the inference algorithms:

- Extract implicit knowledge from explicitly given one.
- Offer inference capabilities to the user as a system service.

In this section, we introduce the inference problems. Inference procedures for solving these problems are discussed in later chapters.

We distinguish *terminological* and *assertional* reasoning, the latter either *with* or *without* TBoxes. We will show that it suffices to develop an algorithm for assertional reasoning without TBoxes since all other tasks can be reduced to this one.

3.4.1 Terminological Reasoning

We consider the following problems:

- *Satisfiability* of a concept: is a concept free of contradictions?
- *Subsumption* of two concepts: is one concept a subconcept of the other one?

Definition 3.12. Let C and D be concepts, and $\mathcal{T}$ a TBox.

1. C is called *satisfiable* iff there is an interpretation $\mathcal{I}$ such that $C^{\mathcal{I}} \neq \emptyset$ (*satisfiability of concept terms*).
2. C is called *satisfiable with respect to* $\mathcal{T}$ iff there is a model $\mathcal{I}$ of $\mathcal{T}$ such that $C^{\mathcal{I}} \neq \emptyset$ (*satisfiability wrt. TBoxes*).
3. C *is subsumed by* D (written $C \sqsubseteq D$) iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for all interpretations $\mathcal{I}$ (*subsumption of concept terms*).
4. C *is subsumed by* D *with respect to* $\mathcal{T}$ (written $C \sqsubseteq_{\mathcal{T}} D$) iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for all models $\mathcal{I}$ of $\mathcal{T}$ (*subsumption wrt. TBoxes*).

Example. The concept

$$\texttt{Mother} \sqcap (\leq 0\ \texttt{has-child})$$

is satisfiable as a concept term, but not wrt. the TBox from Example 3.11.

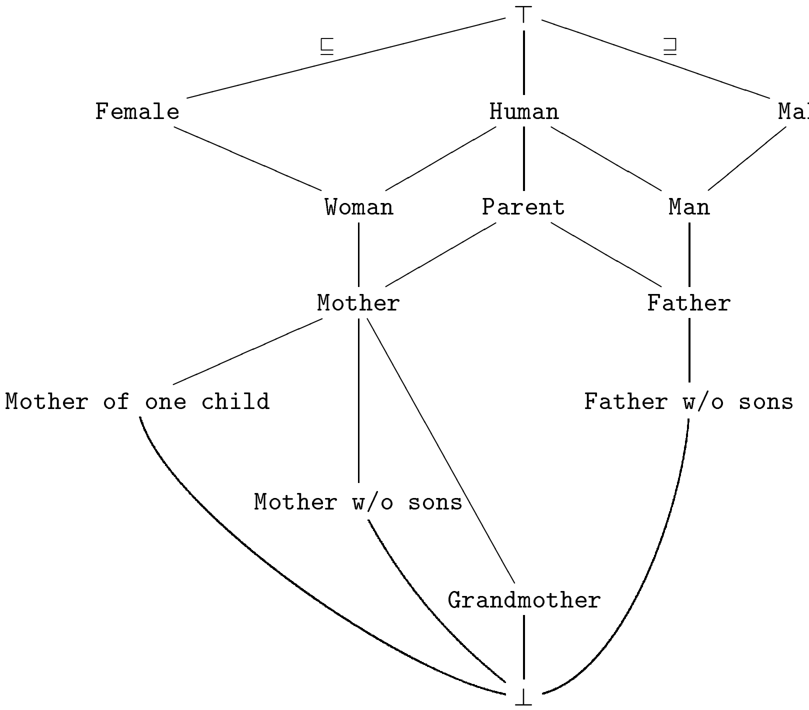
Theorem 3.13. The subsumption relation is reflexive and transitive, but in general not anti-symmetric, i.e. it is a quasi-ordering.

Proof. Exercise. □

We say that C is *equivalent* to D (wrt. $\mathcal{T}$) iff $C \sqsubseteq D$ and $D \sqsubseteq C$ (resp. $C \sqsubseteq_{\mathcal{T}} D$ and $D \sqsubseteq_{\mathcal{T}} C$).

DL systems offer the system service *TBox classification*, which computes the subsumption relationship between all concept names occurring in the TBox. The resulting concept hierarchy can then be displayed graphically.

In Example 3.11, we get:



All inference problems introduced in Definition 3.12 can be reduced to the satisfiability of concept terms (this holds for $\mathcal{ALCN}$ and other description languages, depending on the availability of certain operators).

Theorem 3.14.

1. Satisfiability and subsumption wrt. TBoxes can be reduced to satisfiability and subsumption of concept terms.
2. Satisfiability can be reduced to subsumption (in linear time) (if the concept language has $\bot$ or can express it, e.g. by $A \sqcap \neg A$).
3. Subsumption can (in linear time) be reduced to satisfiability (if the concept language has $\sqcap$ and $\neg$ or can express them).

Proof.

1. Let C and D be concept terms, $\mathcal{T}$ a TBox, and $\hat{\mathcal{T}}$ the expansion of $\mathcal{T}$ (as in Theorem 3.6).

   $\hat{C}$ and $\hat{D}$ are obtained from C and D by replacing all defined concept names with their definition from $\hat{\mathcal{T}}$. Then:

   (a) $\hat{C}$ is satisfiable iff C is satisfiable wrt. $\mathcal{T}$.

   (b) $\hat{C} \sqsubseteq \hat{D}$ iff $C \sqsubseteq_{\mathcal{T}} D$.

We only prove (1a) and leave (1b) as an exercise.

"$\Rightarrow$": Let $\mathcal{I}$ be an interpretation of the concept names and role names used in $\hat{C}$ such that $\hat{C}^{\mathcal{I}} \neq \emptyset$. By Corollary 3.7, $\mathcal{I}$ can be extended to a model $\hat{\mathcal{I}}$ of $\hat{\mathcal{T}}$, which is also a model of $\mathcal{T}$ by Theorem 3.6. We have

$$\emptyset \neq \hat{C}^{\mathcal{I}} \underbrace{=}_{\hat{\mathcal{I}} \text{ extends } \mathcal{I}} \hat{C}^{\hat{\mathcal{I}}} \underbrace{=}_{\hat{\mathcal{I}} \text{ is a model of } \hat{\mathcal{T}}} C^{\hat{\mathcal{I}}}$$

"$\Leftarrow$": Let $\mathcal{I}$ be a model of $\mathcal{T}$ with $C^{\mathcal{I}} \neq \emptyset$. Then $\mathcal{I}$ is also a model of $\hat{\mathcal{T}}$, and thus $\emptyset \neq C^{\mathcal{I}} = \hat{C}^{\mathcal{I}}$.

2. C is satisfiable iff $C \not\sqsubseteq \bot$, and C is satisfiable wrt. $\mathcal{T}$ iff $C \not\sqsubseteq_{\mathcal{T}} \bot$.
3. $C \sqsubseteq D$ iff $C \sqcap \neg D$ is unsatisfiable.

$\square$

Observation 3.15. The reduction described in the proof of Point 1 need not be polynomial. Define, for $n \geq 0$, the TBoxes

$$\begin{aligned}
\mathcal{T}_n \quad := \{ \quad & C_0 \doteq A \\
& C_1 \doteq \exists r_1.C_0 \sqcap \exists r_2.C_0 \\
& \dots \\
& C_n \doteq \exists r_1.C_{n-1} \sqcap \exists r_2.C_{n-1} \}.
\end{aligned}$$

The size of $\mathcal{T}_n$ is linear in n, but the size of $\hat{\mathcal{T}}_n$ is exponential in n.

3.4.2 Assertional Reasoning

We consider

- the *consistency problem of ABoxes*: is an ABox free of contradictions?
- the *instance problem*: is an individual in an ABox an instance of a concept?

Definition 3.16. Let C be a concept term, $a \in N_I$, $\mathcal{T}$ a TBox, and $\mathcal{A}$ an ABox.

1. The ABox $\mathcal{A}$ is *consistent* if it has a model.
2. The ABox $\mathcal{A}$ is *consistent wrt.* $\mathcal{T}$ if $\mathcal{A}$ and $\mathcal{T}$ have a common model.
3. The individual a *is an instance of* C *wrt.* $\mathcal{A}$ (written $\mathcal{A} \models a : C$) if $a^{\mathcal{I}} \in C^{\mathcal{I}}$ in all models $\mathcal{I}$ of $\mathcal{A}$.
4. The individual a *is an instance of* C *wrt.* $\mathcal{A}$ *and* $\mathcal{T}$ (written $\mathcal{A}, \mathcal{T} \models a : C$) if $a^{\mathcal{I}} \in C^{\mathcal{I}}$ in all common models $\mathcal{I}$ of $\mathcal{A}$ and $\mathcal{T}$.

In Example 3.11, we can conclude the following:

- $\mathcal{T}, \mathcal{A} \models \texttt{Bill} : \texttt{Father}$

- $\mathcal{T}, \mathcal{A} \models$ `Hillary` : `Mother w/o sons`
- $\mathcal{T}, \mathcal{A} \not\models$ `Bill` : `Father w/o sons`

Definition 3.17 (System Services). Let $\mathcal{T}$ be a TBox, $\mathcal{A}$ an ABox, $a \in N_I$, and C a concept term.

1. *Realization* of an individual a: what are the (minimal) concept names of which a is an instance?

   $I_{\mathcal{T},\mathcal{A}}(a) := \{A | A \text{ is a concept name in } \mathcal{T} \cup \mathcal{A} \text{ and } \mathcal{T}, \mathcal{A} \models a : A\}$

   $\mu I_{\mathcal{T},\mathcal{A}}(a) := \{A \in I_{\mathcal{T},\mathcal{A}}(a) | \text{there is no } B \in I_{\mathcal{T},\mathcal{A}}(a) \text{ with } B \sqsubset A\}$

   where $B \sqsubset A := B \sqsubseteq A \wedge B \not\sqsupseteq A$

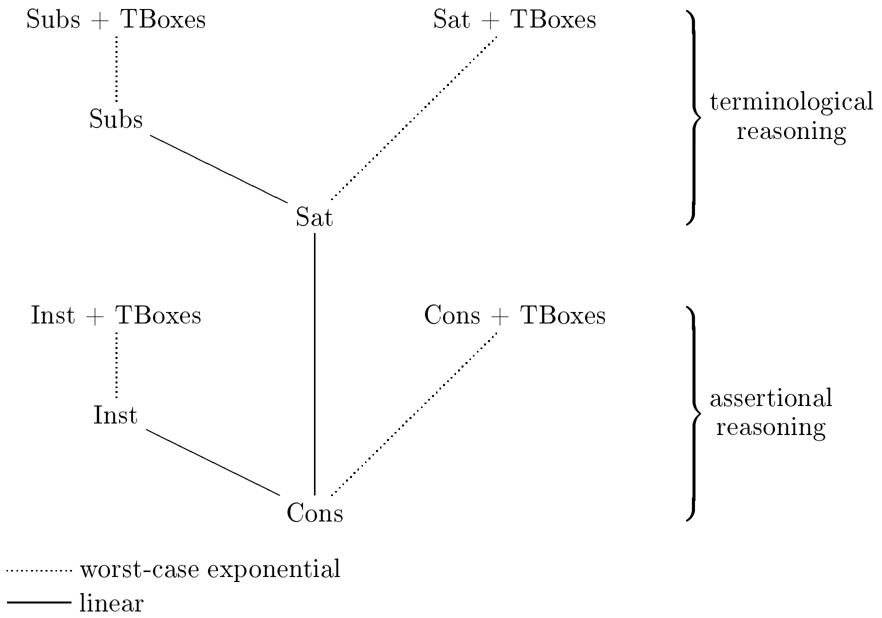
2. *Retrieval*: Which individuals are an instance of C?

   $R_{\mathcal{T},\mathcal{A}}(C) := \{a | a \text{ is individual name in } \mathcal{A} \text{ and } \mathcal{T}, \mathcal{A} \models a : C\}$

In Example 3.11, we have

$$
\begin{aligned}
I_{\mathcal{T},\mathcal{A}}(\texttt{Hillary}) &\doteq \{\texttt{Woman}, \texttt{Mother of one child}, \texttt{Female}, \texttt{Human}, \\
&\quad\ \texttt{Mother}, \texttt{Parent}, \texttt{Mother w/o sons}\} \\
\mu I_{\mathcal{T},\mathcal{A}}(\texttt{Hillary}) &\doteq \{\texttt{Mother of one child}, \texttt{Mother w/o sons}\}
\end{aligned}
$$

Note: It suffices to compute and store $\mu I_{\mathcal{T},\mathcal{A}}$ since from this, together with the concept hierarchy, $I_{\mathcal{T},\mathcal{A}}$ can easily be computed.



Theorem 3.18.

1. Satisfiability of concept terms (and therefore also subsumption) can be linearly reduced to the consistency of ABoxes.
2. Consistency of ABoxes wrt. TBoxes and the instance problem wrt. TBoxes can be reduced to ABox consistency and the instance problem, respectively (without TBoxes).
3. The instance problem can linearly be reduced to the consistency problem (if the concept language has $\neg$ or can express it).

Proof.

1. C is satisfiable iff $\{a : C\}$ is consistent.
2. Expand $\mathcal{T}$ to $\hat{\mathcal{T}}$ and replace defined names occurring in $\mathcal{A}$ with their definitions from $\hat{\mathcal{T}}$ (can result in exponential blowup).
3. $\mathcal{A} \models a : C$ iff $\mathcal{A} \cup \{a : \neg C\}$ is inconsistent.

$\square$

We say that an ABox $\mathcal{A}$ is *equivalent* to an ABox $\mathcal{B}$ if $\mathcal{A}$ and $\mathcal{B}$ have the same models.

4 A Tableau-based Consistency Algorithm for $\mathcal{ALCN}$

We use a specialization of the tableau proof method for first order logic to decide the consistency of ABoxes (without TBoxes, since they can be expanded in a preprocessing step). Our algorithm will always terminate and thus yield a decision procedure – in contrast to semi-decision procedures in the FOL case.

The algorithm works on *generalized ABoxes*: in contrast to the $\mathcal{ALCN}$-ABoxes used up to now, these have

1. no UNA for individual names and
2. inequality assertions of the form $a \neq b$, where $a, b \in N_I$. Semantics: $\mathcal{I}$ satisfies $a \neq b$ iff $a^{\mathcal{I}} \neq b^{\mathcal{I}}$.

Observe: Inequality assertions are considered to be symmetric, i.e. $a \neq b \in \mathcal{A}$ iff $b \neq a \in \mathcal{A}$ holds.

Lemma 4.1. For every $\mathcal{ALCN}$-ABox $\mathcal{A}$, there exists an equivalent generalized ABox $\mathcal{B}$.

Proof. Let $a_1, \ldots, a_n$ be the individual names in $\mathcal{A}$, and set

$$\mathcal{B} := \mathcal{A} \cup \{a_i \neq a_j \mid 1 \leq i, j \leq n, i \neq j\}$$

□

Thus, we can assume w.l.o.g. that we are given a generalized ABox as input.

Preprocessing

Definition 4.2. A concept term is in *negation normal form (NNF)* if negation occurs only in front of concept names, but not in front of complex concepts. A (generalized) ABox is in NNF if all concept terms occurring in it are in NNF.

A generalized ABox $\mathcal{A}$ can be transformed into NNF by exhaustively applying the following, equivalence-preserving rules to each concept term in $\mathcal{A}$:

$$\begin{array}{rcl}
\neg\neg C & \to & C \\
\neg(C \sqcup D) & \to & \neg C \sqcap \neg D \\
\neg(C \sqcap D) & \to & \neg C \sqcup \neg D \\
\neg(\forall r.C) & \to & \exists r.\neg C \\
\neg(\exists r.C) & \to & \forall r.\neg C \\
\neg(\leq n\ r) & \to & (\geq n+1\ r) \\
\neg(\geq n\ r) & \to & (\leq n-1\ r) \text{ if } n > 0 \\
\neg(\geq 0\ r) & \to & \bot
\end{array}$$

Exercise: By exhaustively applying these rules, a concept term C is transformed into an equivalent one in NNF (we have to prove termination and preservation of equivalence).

Thus, the obtained ABox is equivalent to the original one.

The consistency algorithm works on sets of generalized ABoxes. It starts with $\{\mathcal{A}_0\}$, where $\mathcal{A}_0$ is the generalized ABox in NNF given as input. The algorithm then applies transformation rules with the goal of finding obvious contradictions.

Definition 4.3. A generalized $\mathcal{ALCN}$-ABox $\mathcal{A}$ is called *closed* iff it contains one of the following obvious contradictions:

1. $\{a : A, a : \neg A\} \subseteq \mathcal{A}$ for an individual name a and a concept name A;
2. $a \neq a \in \mathcal{A}$ for an individual name a;
3. $\{a : (\leq n\ r), (a, b_1) : r, \ldots, (a, b_{n+1}) : r\} \cup \{b_i \neq b_j \mid 1 \leq i, j \leq n+1, i \neq j\} \subseteq \mathcal{A}$.

$\mathcal{A}$ is *open* if it is not closed.

Note: The second obvious contradiction cannot be produced by rules. It can thus only occur if it is already contained in the initial ABox.

$\mathcal{ALCN}$ **Tableau Algorithm**

Input: a generalized $\mathcal{ALCN}$-ABox $\mathcal{A}_0$ in NNF

Output:

"yes" if $\mathcal{A}_0$ is consistent

"no" otherwise

The Algorithm:

- $M := \{\mathcal{A}_0\}$
- As long as one of the following rules is applicable to an open ABox in M, apply such a rule.
- Let $\hat{M}$ be the set of ABoxes obtained after termination of this loop. Return "yes" if $\hat{M}$ contains an open ABox, and "no" if all ABoxes in $\hat{M}$ are closed.

The rules:

$\sqcap$-rule: Replace $\mathcal{A} \in M$ by $\mathcal{A}'$, if

$a : C \sqcap D \in \mathcal{A}$ and $\{a : C, a : D\} \not\subseteq \mathcal{A}$ and $\mathcal{A}' = \mathcal{A} \cup \{a : C, a : D\}$.

$\sqcup$-rule: Replace $\mathcal{A} \in M$ by $\mathcal{A}'$ and $\mathcal{A}''$, if

$a : C \sqcup D \in \mathcal{A}$ and $\{a : C, a : D\} \cap \mathcal{A} = \emptyset$ and $\mathcal{A}' = \mathcal{A} \cup \{a : C\}$ and $\mathcal{A}'' = \mathcal{A} \cup \{a : D\}$.

$\forall$-rule: Replace $\mathcal{A} \in M$ by $\mathcal{A}'$, if

$\{a : \forall r.C, (a,b) : r\} \subseteq \mathcal{A}$ and $b : C \notin \mathcal{A}$ and $\mathcal{A}' = \mathcal{A} \cup \{b : C\}$.

$\exists$-rule: Replace $\mathcal{A} \in M$ by $\mathcal{A}'$, if

$\{a : \exists r.C\} \subseteq \mathcal{A}$ and there is no individual name b with $\{(a,b) : r, b : C\} \subseteq \mathcal{A}$ and $\mathcal{A}' = \mathcal{A} \cup \{(a,c) : r, c : C\}$ for an individual name c new in $\mathcal{A}$.

$\geq$-rule: Replace $\mathcal{A} \in M$ by $\mathcal{A}'$, if

- $a : (\geq n\ r) \in \mathcal{A}$,
- there are no individual names $b_1 \ldots, b_n$ with $\mathcal{A} \supseteq \{(a,b_1) : r, \ldots, (a,b_n) : r\} \cup \{b_i \neq b_j \mid 1 \leq i < j \leq n\}$,
- $\mathcal{A}' = \mathcal{A} \cup \{(a,c_1) : r, \ldots, (a,c_n) : r\} \cup \{c_i \neq c_j \mid 1 \leq i < j \leq n\}$ for c_i new individual names in $\mathcal{A}$.

$\leq$-rule: Replace $\mathcal{A} \in M$ by at most $\binom{n+1}{2}$ ABoxes $\mathcal{A}_{i,j}$, if

- $a : (\leq n\ r) \in \mathcal{A}$,
- there are (at least) $n+1$ individual names $b_0, \ldots, b_n$ with $\{(a,b_0) : r, \ldots, (a,b_n) : r\} \subseteq \mathcal{A}$,
- there are $i \neq j$ with $0 \leq i, j \leq n$ and $b_i \neq b_j \notin \mathcal{A}$,
- for $i < j$ with $b_i \neq b_j \notin \mathcal{A}$, we define $\mathcal{A}_{i,j} := \mathcal{A}[b_i/b_j]$, i.e. $\mathcal{A}_{i,j}$ is obtained from $\mathcal{A}$ by replacing each occurrence of b_i with b_j.

Example. Consider the input

$$\{a : (\geq 1\ r), a : \forall r.(\exists s.\neg A \sqcap \exists s.\neg B), a : \exists r.\exists s.(A \sqcup B), a : (\leq 1\ r), a : \forall r.(\leq 1\ s)\}.$$

Then one branch of the computation yields these results:

$$\begin{array}{c}
a \\
\Big\downarrow r \\
b : \exists s.\neg A \sqcap \exists s.\neg B \quad b : \exists s.(A \sqcup B) \quad b : \exists s.\neg A \quad b : \exists s.\neg B \quad b : (\leq 1\ s) \\
\Big\downarrow s \\
c : \neg B \quad c : A \sqcup B \quad c : \neg A \quad c : A \quad \lightning
\end{array}$$

Definition 4.4. A generalized ABox is *complete* if no rule from the algorithm is applicable to it.

The algorithms searches for a complete and open ABox. This search can be visualized using a search tree (see Figure 8). Rules which lead to branching in the search tree (i.e. $\sqcup$-rule and $\leq$-rule) are called *non-deterministic*, the remaining ones are called *deterministic*.

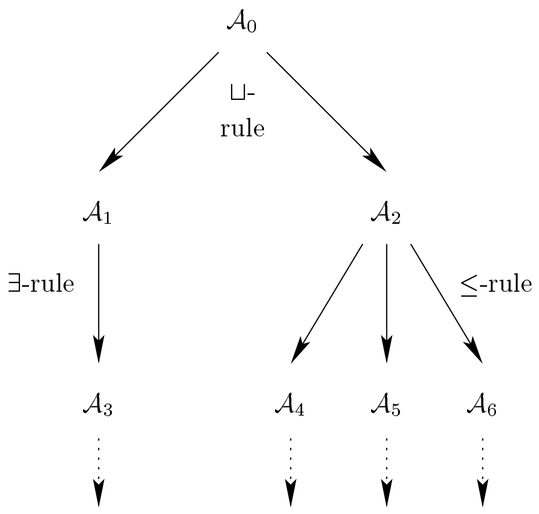


Figure 8: Search tree

Lemma 4.5.

1. A closed generalized ABox has no model.
2. If an ABox $\mathcal{A}$ is open and complete, then $\mathcal{A}$ has a model.

Proof.

1. This is trivial.
2. Let $\mathcal{A}$ be an open and complete ABox. We define the *canonical model* $\mathcal{I}$ of $\mathcal{A}$ as follows:
   - $\Delta^{\mathcal{I}}$ is the set of all individual names occurring in $\mathcal{A}$,
   - $A^{\mathcal{I}} := \{a \mid a : A \in \mathcal{A}\}$ for all concept names A,
   - $r^{\mathcal{I}} := \{(a, b) \mid (a, b) : r \in \mathcal{A}\}$ for all role names r,
   - $a^{\mathcal{I}} := a$ for each $a \in \Delta^{\mathcal{I}}$.

Claim: $\mathcal{I}$ is a model of $\mathcal{A}$.

a) Role assertions: $(a, b) : r \in \mathcal{A}$ yields $(a, b) \in r^{\mathcal{I}}$ by definition of $\mathcal{I}$.

b) Inequality assertions: $a \neq b \in \mathcal{A}$. Suppose $a^{\mathcal{I}} = b^{\mathcal{I}}$. Then $a = b$ by definition of $\mathcal{I}$, implying that $\mathcal{A}$ is closed.

c) Concept assertions: $a : C$. We prove this by induction on the structure of C.

   i) $a : A \in \mathcal{A}$. By definition of $\mathcal{I}$, we have $a \in \mathcal{A}^{\mathcal{I}}$.

ii) $a : \neg A \in \mathcal{A}$. Since $\mathcal{A}$ is open, we have $a : A \notin \mathcal{A}$ and thus $a \notin A^{\mathcal{I}}$, yielding $a \in (\neg A)^{\mathcal{I}}$.

iii) $a : C \sqcap D \in \mathcal{A}$. Since $\mathcal{A}$ is complete, the $\sqcap$-rule is not applicable implying $\{a : C, a : D\} \subseteq \mathcal{A}$. Induction gives $a \in C^{\mathcal{I}}$ and $a \in D^{\mathcal{I}}$ and thus $a \in C^{\mathcal{I}} \cap D^{\mathcal{I}} = (C \sqcap D)^{\mathcal{I}}$.

iv) $a : C \sqcup D \in \mathcal{A}$. Analogous to $\sqcap$.

v) $a : \exists r.C \in \mathcal{A}$. Since $\mathcal{A}$ is complete, there is a b such that $b : C \in \mathcal{A}$ and $(a, b) : r \in \mathcal{A}$. We know that $(a, b) \in r^{\mathcal{I}}$ and induction yields $b \in C^{\mathcal{I}}$. Hence, $a \in (\exists r.C)^{\mathcal{I}}$.

vi) $a : \forall r.C \in \mathcal{A}$ is similar.

vii) $a : (\geq n\ r) \in \mathcal{A}$. Since $\mathcal{A}$ is complete, there are $b_1, \dots, b_n$ with

$$\{(a, b_1) : r, \dots, (a, b_n) : r\} \cup \{b_i \neq b_j \mid 1 \leq i < j \leq n\} \subseteq \mathcal{A}.$$

We have $(a, b_1) \in r^{\mathcal{I}}, \dots, (a, b_n) \in r^{\mathcal{I}}$, and all b_i are distinct since the inequality assertions are satisfied by $\mathcal{I}$.

viii) $a : (\leq n\ r) \in \mathcal{A}$. Assume $a \notin (\leq n\ r)^{\mathcal{I}}$. Then there are distinct individuals $b_1, \dots, b_{n+1}$ such that $\{(a, b_1) : r, \dots, (a, b_{n+1}) : r\} \subseteq \mathcal{A}$. Since $\mathcal{A}$ is complete, the $\leq$-rule is not applicable. Then we have $b_i \neq b_j \in \mathcal{A}$ for all $i \neq j$. Thus $\mathcal{A}$ is closed, which yields a contradiction.

□

Lemma 4.6. Rule application preserves consistency, i.e. if $\mathcal{A}$ is replaced by $\mathcal{A}_1, \dots, \mathcal{A}_m$, then $\mathcal{A}$ is consistent if and only if one of the $\mathcal{A}_i$ is consistent.

Proof. We consider the $\sqcup$-rule, the $\exists$-rule, and the $\leq$-rule (we leave the remaining rules are an exercise).

$\sqcup$-rule: $a : C \sqcup D \in \mathcal{A}$, $\mathcal{A}' = \mathcal{A} \cup \{a : C\}$, and $\mathcal{A}'' = \mathcal{A} \cup \{a : D\}$

"$\Rightarrow$": Let $\mathcal{I}$ be a model of $\mathcal{A}$. Since $a : C \sqcup D \in \mathcal{A}$, we have $a^{\mathcal{I}} \in (C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$. Thus $a^{\mathcal{I}} \in C^{\mathcal{I}}$ or $a^{\mathcal{I}} \in D^{\mathcal{I}}$. In the first case, $\mathcal{I}$ is a model of $\mathcal{A}'$, and in the second case, $\mathcal{I}$ is a model of $\mathcal{A}''$.

"$\Leftarrow$": Let $\mathcal{I}$ be a model of $\mathcal{A}'$ or $\mathcal{A}''$. Then $\mathcal{A} \subseteq \mathcal{A}', \mathcal{A}''$ immediately yields that $\mathcal{I}$ is a model of $\mathcal{A}$.

$\exists$-rule: $a : \exists r.C \in \mathcal{A}$, and $\mathcal{A}' = \mathcal{A} \cup \{(a, c) : r, c : C\}$, where c does not appear in $\mathcal{A}$.

"$\Rightarrow$": Let $\mathcal{I}$ be a model of $\mathcal{A}$. Then $a^{\mathcal{I}} \in (\exists r.C)^{\mathcal{I}}$ and thus there is a $x \in \Delta^{\mathcal{I}}$ with $(a^{\mathcal{I}}, x) \in r^{\mathcal{I}}$ and $x \in C^{\mathcal{I}}$. We define a new interpretation $\mathcal{I}'$: it coincides with $\mathcal{I}$, except for the interpretation of c: $c^{\mathcal{I}} := x$.

Then $\mathcal{I}'$ satisfies the assertions $(a, c) : r$ and $c : C$. Since c does not occur in $\mathcal{A}$, $\mathcal{I}'$ is also a model of $\mathcal{A}$, thus of $\mathcal{A}'$.

"$\Leftarrow$": trivial since $\mathcal{A} \subseteq \mathcal{A}'$.

$\leq$-rule: $a : (\leq n\ r) \in \mathcal{A}$, and there are $b_1, \ldots, b_{n+1}$ in $\mathcal{A}$ with $(a, b_i) : r \in \mathcal{A}$.

> **"$\Rightarrow$":** Let $\mathcal{I}$ be a model of $\mathcal{A}$. Since $a^\mathcal{I} \in (\leq n\ r)^\mathcal{I}$ and $(a^\mathcal{I}, b_i^\mathcal{I}) \in r^\mathcal{I}$ for $i = 1, \ldots, n+1$, there are $i \neq j$ such that $b_i^\mathcal{I} = b_j^\mathcal{I}$. Therefore, $b_i \neq b_j \notin \mathcal{A}$ since $\mathcal{I}$ is a model of $\mathcal{A}$, and $\mathcal{A}_{i,j} = \mathcal{A}[b_i/b_j]$ is a successor ABox of $\mathcal{A}$ when applying the $\leq$-rule. Since $b_i^\mathcal{I} = b_j^\mathcal{I}$, $\mathcal{I}$ is also a model of $\mathcal{A}_{i,j}$.
>
> **"$\Leftarrow$":** Let $\mathcal{I}$ be a model of $\mathcal{A}_{i,j}$. Define $\mathcal{I}'$ as $\mathcal{I}$ with the only difference that $b_i^\mathcal{I} = b_j^\mathcal{I}$. Then $\mathcal{I}'$ is a model of $\mathcal{A}$.

$\square$

Theorem 4.7. Let $\hat{M}$ be the set of ABoxes obtained by the algorithm after the loop has terminated.

1. If $\hat{M}$ contains an open generalized ABox, then $\mathcal{A}_0$ is consistent (soundness).
2. If all elements of $\hat{M}$ are closed, then $\mathcal{A}_0$ is inconsistent (completeness).

Proof. Due to Lemma 4.6, $\mathcal{A}_0$ is consistent iff $\hat{M}$ contains a consistent ABox.

1. If $\hat{M}$ contains an open ABox $\hat{\mathcal{A}}$, then this ABox is also complete. With Lemma 4.5, $\hat{\mathcal{A}}$ has a model.
2. If all elements of $\hat{M}$ are closed, then by Lemma 4.5, they are inconsistent.

$\square$

Termination Without an additional strategy that guides the application of rules, termination is *not* guaranteed.

Example 4.8.

$$\begin{aligned}\mathcal{A}_0 \quad := \quad & \{a_0 : A, (a_0, a_0) : r, a_0 : (\leq 1\ r), a_0 : \exists r.A, a_0 : \forall r.((\leq 1\ r) \sqcap \exists r.A), (a_0, a_1) : r \\ & a_0 : (\leq 1\ r) \sqcap \exists r.A, a_1 : A\}\end{aligned}$$

Then we get:

$$\begin{aligned}\mathcal{A}_0 \quad & \xrightarrow{\forall\text{-rule}} \quad \mathcal{A}_1 = \mathcal{A}_0 \cup \{a_1 : (\leq 1\ r) \sqcap \exists r.A\} \\ & \xrightarrow{\sqcap\text{-rule}} \quad \mathcal{A}_2 = \mathcal{A}_1 \cup \{a_1 : (\leq 1\ r), a_1 : \exists r.A\} \\ & \xrightarrow{\exists\text{-rule}} \quad \mathcal{A}_3 = \mathcal{A}_2 \cup \{(a_1, a_2) : r, a_2 : A\} \\ & \xrightarrow{\leq\text{-rule}} \quad \mathcal{A}_4 = \mathcal{A}_3[a_1/a_0]\end{aligned}$$

We realize that $\mathcal{A}_4 = \mathcal{A}_0[a_1/a_2]$. Thus we can repeat rule application infinitely.

Sources of the problem:

1. The role cycle $(a_0, a_0) : r$.
2. The eager application of the $\exists$-rule. If the $\leq$-rule is applied before the $\exists$-rule, then the $\exists$-rule is not applicable any more since the ABox contains $(a_1, a_1) : r$ and $a_1 : A$.

Strategy: We must distinguish between old and new individuals:

- old individuals, which are already present in the input ABox $\mathcal{A}_0$;
- new individuals, which are introduced by $\exists$-rule or $\geq$-rule.

Note: Only old individuals may be involved in role cycles.

Refine the $\leq$-rule. Always replace new individuals with old ones, but not vice versa. This is needed to avoid losing the distinction between old and new individuals by applications of the $\leq$-rule.

Proceed in 2 phases.

Phase 1: Exhaustively apply rules to old individuals.

Phase 2: Apply rules arbitrarily.

Lemma 4.9. Phase 1 always terminates.

Proof. (Sketch)

1. The number of old individuals may never increase, only decrease (through applications of $\leq$-rule).
2. to each old individual, only finitely many rules may be applied:
   (a) All concept assertions are of the form $a : C$ with C subconcept of a concept appearing in $\mathcal{A}_0$.
   (b) For each individual a, the $\sqcap$-, $\sqcup$-, $\exists$-, and $\geq$-rule can be applied only finitely often: at most once for each assertion $a : C$, whose number is finite due to (2a).
   (c) For each assertion $a : (\leq n\ r)$ and $a : \forall r.C$, the number of applications of the $\leq$-rule and the $\forall$-rule is bounded by the number of r-successors of a in $\mathcal{A}$. This number is finite: each r-successor of a was either already an r-successor in $\mathcal{A}_0$, or introduced due to the application of the $\exists$- or $\geq$-rule. The number of successors of the first type is finite since $\mathcal{A}_0$ is finite. The number of successors of the second type is finite due to (2b).

$\square$

Lemma 4.10. In Phase 2, no rules are applied to old individuals.

Proof. (Sketch) Applicability of rules to old individuals depends on assertions involving old individuals and successors of old individuals.

- Due to the refinement of the $\leq$-rule, old individuals are never successors of new ones. Therefore, rule applications in Phase 2 can never add assertions involving old individuals.
- For successor individuals of old individuals, new assertions may be added, but existing ones not deleted. It follows that assertions which have been satisfied in Phase 1 remain satisfied.

□

Lemma 4.11. Phase 2 always terminates.

Proof. (Sketch) In contrast to old individuals, new individuals satisfy the following conditions:

1. Each new individual x has precisely one role-predecessor, i.e. there are exactly one $y \in N_I$ and $r \in N_R$ such that $(y, x) : r \in \mathcal{A}$.
2. If $(x, y) : r \in \mathcal{A}$ with x and y new, then

$$\{|C| \mid y : C \in \mathcal{A}\} > \{|D| \mid x : D \in \mathcal{A}\}$$

   where $|C|$ denotes the *length* of concept C, i.e. the number of symbols needed to write it.

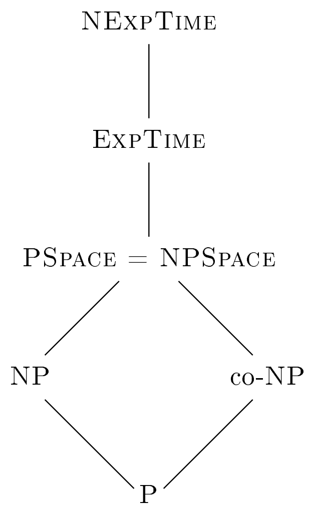
If we consider all role paths that emerge from a new individual y, then this yields (due to 1) a tree. The tree has a finite out-degree since we can apply the $\exists$- and $\geq$-rule at most finitely often to each individual. It has finite depth due to (2). □

Complexity of the Algorithm. We want to show that it is possible to decide the consistency of ABoxes using only polynomial space, i.e. the consistency problem for $\mathcal{ALCN}$-ABoxes is in the complexity class PSpace (which is considered to be worse than NP, see Figure 9). Due to Savitch's theorem, PSpace is the same as NPSpace, i.e. we can assume that the non-deterministic rules always consider only one alternative. Nevertheless, the presented algorithm needs an exponential amount of memory:

Example 4.12.

$$\begin{aligned} C_0 &:= B \\ C_{i+1} &:= \exists r.A \sqcap \exists r.\neg A \sqcap \forall r.C_i \end{aligned}$$

The ABox consistency algorithm applied to $\mathcal{A}_n := \{a : C_n\}$ generates a complete and open ABox containing $> 2^n$ individuals, i.e. requiring an amount of memory that is exponential in the size of the ABox.



	deterministic	non-deterministic
polynomial time	P	NP
polynomial space	PSpace = NPSpace	
exponential time	ExpTime	NExpTime

Figure 9: Complexity classes

Proof. We prove the claim by induction over the depth of the tree.

- $n = 0$. $\mathcal{A}_0 = \{a : B\}$ is complete and a tree of depth 0.
- $n \to n + 1$. We start with

$$\Big\{\mathcal{A}_{n+1} = \big\{a : \exists r.A \sqcap \exists r.\neg A \sqcap \forall r.C_n\big\}\Big\} \xrightarrow{\text{rule application}}$$
$$\Big\{\mathcal{A}_{n+1} \cup \big\{a_1 : A, \underline{a_1 : C_n}, a_2 : \neg A, \underline{a_2 : C_n}, (a, a_1) : r, (a, a_2) : r, \ldots\big\}\Big\}$$

Note: $\{a_1 : C_n\} \hat{=} \mathcal{A}_n$ and $\{a_2 : C_n\} \hat{=} \mathcal{A}_n$ (we generate two full binary trees of depth n with all leaves x having $x : B$ by the induction hypothesis.)

$\square$

This algorithm can be modified such that it only uses polynomial space.

Idea: We keep only parts of the ABox in memory, but never the whole ABox. This is possible only if the different parts are "independent".

Measure of the size of ABoxes: $|\mathcal{A}|$ is the number of symbols used to write $\mathcal{A}$. For numbers in $(\leq n\ r)$ and $(\geq n\ r)$ concepts, we assume *unary coding*, e.g. the size of 100 is 100, and not log(100).

Theorem 4.13. Consistency of $\mathcal{ALCN}$-ABoxes can be decided using only polynomial space.

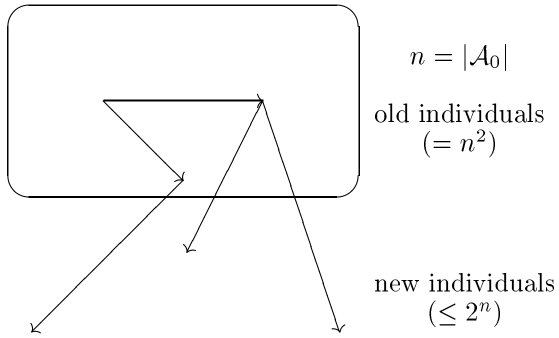
Proof. (Sketch)

Phase 1. This phase needs only polynomial space.

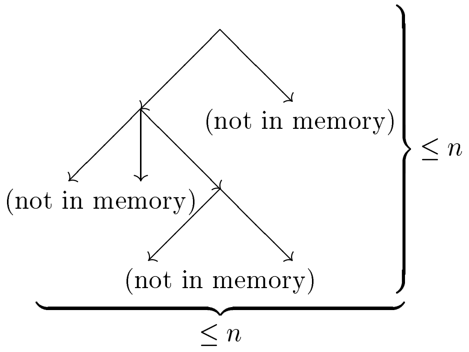
Reason:

1. A polynomial number of individuals may only be involved in a polynomial number of assertions.
   For each individual, we may only have linearly many assertions $a : C$ (since C appears as a subconcept in $\mathcal{A}_0$). For each pair of individuals, we may have only linearly many assertions $(a, b) : r$ and $a \neq b$ (since r appears in $\mathcal{A}_0$).
2. At most a linear number of new individuals is generated. For each old individual, we only generate a linear number of new individuals: one for each (sub)concept $\exists r.C$ in $\mathcal{A}_0$, and n individuals for each (sub)concept $(\geq n\ r)$ in $\mathcal{A}_0$ (note that n is linear since we assume unary coding of numbers).

Phase 2. Intuitively, this phase constructs, for each old individual, a tree with linear depth and linear out-degree (the nodes of the tree are new individuals and the edges are role assertions).



The idea is to avoid constructing the (exponentially large) trees in one piece; instead, we consider single branches in isolation (branches are of linear length).



Hence, the size of this (sub)tree is bounded by n^2 and thus polynomial. Underlying this construction are the ideas of searching for complete and open tree-shaped ABoxes by depth-first search, and the fact that openness of an ABox can be checked by considering each branch separately if all direct successors of nodes on the path have been generated.

□

Concerning unary coding of numbers: to avoid this restriction, we have to avoid the introduction of n successors when processing the concept $(\geq n\ r)$. The general idea is to introduce only a single representative for the n individuals (which makes identification more difficult). In this way, it can be shown that $\mathcal{ALCN}$-ABox consistency with binary coding of numbers is also in PSPACE.

The described PSPACE algorithm is optimal (wrt. worst case complexity) since the concept satisfiability problem for $\mathcal{ALC}$ (i.e. $\mathcal{ALCN}$ without number restrictions) is already PSPACE-hard.

We prove this formally by reduction of a known PSPACE-complete problem. The following problem is the "prototypical" PSPACE-complete problem (it plays a similar role for PSPACE as propositional satisfiability plays for NP).

Definition 4.14. A *quantified Boolean formula* (QBF) is of the form

$$Q_1 p_1 \dots Q_n p_n . \varphi(p_1, \dots, p_n)$$

where $Q_i \in \{\exists, \forall\}$ and $\varphi(p_1, \dots, p_n)$ is a propositional formula using only variables $p_1, \dots, p_n$.

Validity of a QBF is defined via induction on the length of the quantifier prefix. If φ is a propositional formula, then $\varphi[p/1]$ (resp. $\varphi[p/0]$) denotes φ, where p is replaced with 1 (resp. 0).

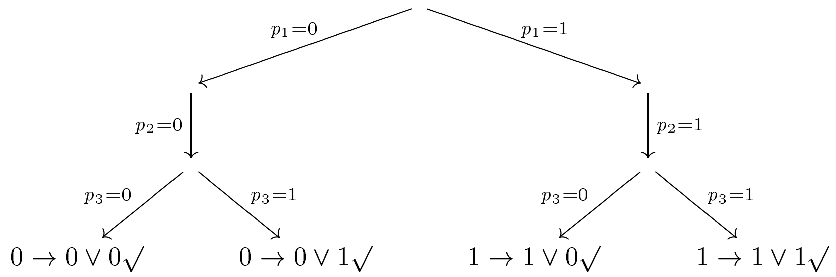
A QBF $Q_1 p_1 \dots Q_n p_n \varphi$ is valid iff

1. $Q_1 = \exists$: $Q_2 p_2 \dots Q_n p_n . \varphi[p_1/0]$ *or* $Q_2 p_2 \dots Q_n p_n [p_1/1]$ is valid;

2. $Q_1 = \forall$: $Q_2 p_2 \dots Q_n p_n . \varphi[p_1/0]$ *and* $Q_2 p_2 \dots Q_n p_n [p_1/1]$ are valid.

This way, we finally achieve a Boolean combination of truth values, which is valid iff it evaluates to 1.

Example 4.15. Consider the QBF $Q := \forall p_1 \exists p_2 \forall p_3 . p_1 \to (p_2 \vee p_3)$. Then Q is valid if and only if it has a quantifier tree:



The QBF $Q' = \forall p_1 \exists p_2 \forall p_3 . p_1 \to (p_2 \wedge p_3)$ is *not* valid.

Observe that:

- each level of the tree corresponds to one of the quantifiers of Q;
- each path from the root to a leaf has length n;
- in $\forall$-levels, each node has two successors: one for $p_i = 0$ and one for $p_i = 1$;
- in $\exists$-levels, each node has one successor: it suffices to explore only one of the possibilities $p_i = 0$ and $p_i = 1$;
- each branch corresponds to a truth assignment, and each such truth assignment makes φ true.

We will now describe the existence of a quantifier tree using $\mathcal{ALC}$-concepts.

Concept names:

$A_1, \ldots, A_n$ (corresponding to the propositional variables p_i), where n is the depth of the quantifier prefix;

$T_0, \ldots, T_{n+1}$ (T_i corresponds to the level $\geq i$ in the tree).

Role name: r

Auxiliary concepts:

1. `Depth`
   Nodes on level $\geq i$ are also on level $\geq i - 1$.
   $$\texttt{Depth} \doteq \prod_{i=1}^{n+1} (T_i \Rightarrow T_{i-1})$$
2. `Determined`
   From level i on, the value of p_i is fixed (i.e. it is distributed to "deeper" nodes).
   $$\texttt{Determined} \doteq \prod_{i=1}^{n} \Big(T_i \Rightarrow \big((A_i \Rightarrow \forall r . A_i) \sqcap (\neg A_i \Rightarrow \forall r . \neg A_i) \big) \Big)$$

3. $\texttt{Branching} \doteq$

$$\begin{aligned}
&\bigsqcap_{\substack{0\le i\le n-1\\ Q_{i+1}=\forall}} \overbrace{(T_i \sqcap \neg T_{i+1})}^{\text{exact depth } i} \Rightarrow \Big(\exists r.(T_{i+1} \sqcap \neg T_{i+2} \sqcap A_{i+1}) \\
&\qquad\qquad \sqcap \exists r.(T_{i+1} \sqcap \neg T_{i+2} \sqcap \neg A_{i+1})\Big) \\
\sqcap &\bigsqcap_{\substack{0\le i\le n-1\\ Q_{i+1}=\exists}} (T_i \sqcap \neg T_{i+1}) \Rightarrow \Big(\exists r.(T_{i+1} \sqcap \neg T_{i+2} \sqcap A_{i+1}) \\
&\qquad\qquad \sqcup \exists r.(T_{i+1} \sqcap \neg T_{i+2} \sqcap \neg A_{i+1})\Big)
\end{aligned}$$

4. Coding of φ
   C_φ is obtained from φ by replacing all p_i with A_i, $\wedge$ with $\sqcap$, and $\vee$ with $\sqcup$.

Reduction

$$\begin{aligned}
Q &= Q_1 p_1 \ldots Q_n p_n . \varphi \\
C_Q &\doteq T_0 \sqcap \neg T_1 \sqcap \bigsqcap_{i=0}^{n} \underbrace{\forall r \ldots \forall r}_{i \text{ times}} . \big(\texttt{Depth} \sqcap \texttt{Determined} \sqcap \texttt{Branching} \sqcap (T_n \Rightarrow C_\varphi)\big)
\end{aligned}$$

Theorem 4.16. The QBF Q is valid iff the $\mathcal{ALC}$-concept term C_Q is satisfiable.

Corollary 4.17. The following inference problems are PSpace-complete for $\mathcal{ALC}$ and $\mathcal{ALCN}$:

- satisfiability and subsumption of concept terms,
- ABox consistency and the instance problem.

Proof.

"In PSpace": This was shown for $\mathcal{ALCN}$-ABox consistency. Thus it also holds for the sublanguage $\mathcal{ALC}$ and for the other reasoning tasks by Theorem 3.18.

"PSpace-hard": This was shown for $\mathcal{ALC}$-concept satisfiability. Thus it also holds for the superlanguage $\mathcal{ALCN}$ and for the other reasoning tasks by Theorem 3.18.

□

Note: all these results also hold in the presence of TBoxes, although this is *not* an immediate consequence of the proved results.

5 A description logic with an undecidable satisfiability problem

In addition to *some* of the constructors introduced until now, the original KL-ONE system offered these constructors:

Role chains

Syntax: $r_1 \circ \cdots \circ r_n$ with $r_1, \ldots, r_n$ role names

Semantics:

$$\begin{aligned}(r_1 \circ \cdots \circ r_n)^{\mathcal{I}} &= r_1^{\mathcal{I}} \circ \cdots \circ r_n^{\mathcal{I}} \\ &= \{(a, c) \mid \exists b_0, \ldots, b_n . a = b_0, c = b_n, \\ &\quad\ (b_{i-1}, b_i) \in r_i^{\mathcal{I}} \text{ for } 1 \leq i \leq n\}\end{aligned}$$

Role-Value-Map (RVM)

Syntax: $(r \sqsubseteq s)$, where r and s are role chains

Semantics:

$$(r \sqsubseteq s)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} \mid r^{\mathcal{I}}(a) \subseteq s^{\mathcal{I}}(a)\}$$

where

$$\begin{aligned}r^{\mathcal{I}}(a) &= \{b \mid (a, b) \in r^{\mathcal{I}}\} \\ s^{\mathcal{I}}(a) &= \{c \mid (a, c) \in s^{\mathcal{I}}\}\end{aligned}$$

For example, $\exists r. \exists r. A \sqcap \forall s. \neg A \sqcap (r \circ r \sqsubseteq s)$ is unsatisfiable.

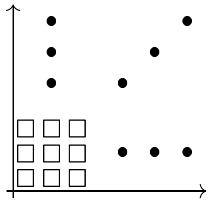
RVMs are very useful for knowledge representation:

- $(\texttt{child} \circ \texttt{child} \sqsubseteq \texttt{grandchild})$
- $(\texttt{nephew} \sqsubseteq \texttt{sibling} \circ \texttt{child})$

We want to show that $\mathcal{ALC}$ extended with RVMs has an undecidable concept satisfiability problem. For the proof, we reduce the undecidable domino problem (aka tiling problem).

The Domino Problem

Intuitively, we are given a finite set of *domino types*, i.e. squares with colored edges. Note: the orientation of tile types is fixed, i.e. tiles may not be rotated. The task is to tile the first quadrant of the plane with dominoes of the given types such that adjacent edges have the same color.



Question: is this possible with the available types, if there is an unlimited supply of dominoes of each type?

Formally, a domino system is defined as follows:

Definition 5.1. A *domino system* $\mathcal{D} = (D, H, V)$ consists of a finite set D of *domino types*, a horizontal matching relation $H \subseteq D \times D$, and a vertical matching relation $V \subseteq D \times D$.

A *solution* to the system (a tiling) is a map $\tau := \mathbb{N} \times \mathbb{N} \to D$ with

- if $\tau(x, y) = d$ and $\tau(x+1, y) = d'$, then $(d, d') \in H$;
- if $\tau(x, y) = d$ and $\tau(x, y+1) = d'$, then $(d, d') \in V$.

Intuition:

- $(d, d') \in H$ means: the right side of d and the left side of d' have the same color;
- $(d, d') \in V$ means: the upper side of d and the lower side of d' have the same color.

The following is a well-known result:

Theorem 5.2. The following problem is undecidable:
Given: a domino system $\mathcal{D}$
Question: does $\mathcal{D}$ have a solution?

A proof can be found e.g. in Börger, Grädel, Gurevitch: "The Classical Decision Problem".

We now construct an $\mathcal{ALC}$-concept term $C_{\mathcal{D}}$ and show:

$C_{\mathcal{D}}$ is satisfiable iff $\mathcal{D}$ has a solution.

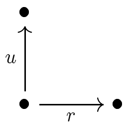
The concept term has to express the following:

- the grid, i.e. $\mathbb{N} \times \mathbb{N}$,
- the existence of a tiling that respects the horizontal and vertical matching conditions.

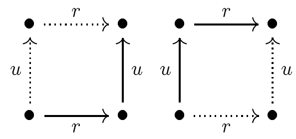
We first express this only locally, i.e. for a grid "cell" and its direct neighbors. To this end, we construct a concept term as follows:

Let $D = \{d_1, \ldots, d_n\}$. Then reserve concept names $D_1, \ldots, D_n$ (one for each domino type). Define $C \Rightarrow D$ as an abbreviation for $\neg C \sqcup D$.

$$D_{\mathcal{D}} := \exists r.\top \sqcap \exists u.\top$$



$$\sqcap\, (u \circ r \sqsubseteq r \circ u) \sqcap (r \circ u \sqsubseteq u \circ r)$$



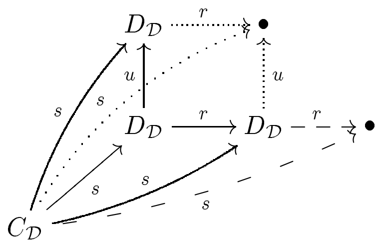
$$\sqcap\, (D_1 \sqcup \cdots \sqcup D_n) \sqcap \bigsqcap_{1 \leq i < j \leq n} \neg(D_i \sqcap D_j)$$ "• has exactly one type"

$$\sqcap \bigsqcap_{i=1}^{n} \Big(D_i \Rightarrow \bigsqcup_{(d_i,d_j) \in H} \forall r.D_j\Big)$$ "right neighbors have a compatible domino type"

$$\sqcap \bigsqcap_{i=1}^{n} \Big(D_i \Rightarrow \bigsqcup_{(d_i,d_j) \in V} \forall u.D_j\Big)$$ "upper neighbors have a compatible domino type"

It remains to ensure that $D_{\mathcal{D}}$ holds at *every* point of the grid.

$$C_{\mathcal{D}} := (s \circ u \sqsubseteq s) \sqcap (s \circ r \sqsubseteq s) \sqcap \exists s.\top \sqcap \forall s.D_{\mathcal{D}}$$



Lemma 5.3. $C_{\mathcal{D}}$ is satisfiable if $\mathcal{D}$ has a solution.

Proof. Let $\tau : \mathbb{N} \times \mathbb{N} \to D$ be a solution of $\mathcal{D}$. We define an interpretation $\mathcal{I}_\tau$ as follows:

$$\begin{array}{rcl}
\Delta^{\mathcal{I}_\tau} & := & \{a\} \cup \mathbb{N} \times \mathbb{N} \\
s^{\mathcal{I}_\tau} & := & \{(a, (i,j)) \mid (i,j) \in \mathbb{N} \times \mathbb{N}\} \\
u^{\mathcal{I}_\tau} & := & \{((i,j), (i, j+1)) \mid (i,j) \in \mathbb{N} \times \mathbb{N}\} \\
r^{\mathcal{I}_\tau} & := & \{((i,j), (i+1, j)) \mid (i,j) \in \mathbb{N} \times \mathbb{N}\} \\
D_k^{\mathcal{I}_\tau} & := & \{(i,j) \mid \tau(i,j) = d_k\} \text{ for } 1 \leq k \leq n
\end{array}$$

It is easy to see that $a \in C_{\mathcal{D}}^{\mathcal{I}_\tau}$. Thus, $C_{\mathcal{D}}$ is satisfiable. $\square$

The main problem in the other direction is that grid elements in the model can have more than a single r- and u-successor. It is crucial to "choose" the correct one for finding a solution.

Lemma 5.4. If $C_{\mathcal{D}}$ is satisfiable, then $\mathcal{D}$ has a solution.

Proof. Let $\mathcal{I}$ be an interpretation and $x \in \Delta^{\mathcal{I}}$ s.t. $x \in C_{\mathcal{D}}^{\mathcal{I}}$. We choose suitable elements of $\Delta^{\mathcal{I}}$ that can be used as grid elements. We start with a "stairway":

$$\begin{tikzcd}
 & & & {} \arrow[r, "r"] & \cdots \\
 & & {} \arrow[r, "r"] & {} \arrow[u, "u"'] & \\
 & D_{\mathcal{D}} \arrow[r, "r"] & {} \arrow[u, "u"'] & & \\
x \arrow[ur, bend left, "s"] \arrow[urr, bend right, "s"'] & & & &
\end{tikzcd}$$

Since each of the obtained grid points is connected to x via the roles, all the points belong to $D_{\mathcal{D}}$. The RVMs in $D_{\mathcal{D}}$ yield all the missing points.

Each grid point belongs to exactly one of $D_1^{\mathcal{I}}, \ldots, D_n^{\mathcal{I}}$. This yields the tiling

$$\tau_{\mathcal{I}} := \mathbb{N} \times \mathbb{N} \to \{d_1, \ldots, d_n\}.$$

The implications in $D_{\mathcal{D}}$ ensure that the horizontal and vertical matching conditions are satisfied. $\square$

Thus we have shown that $C_{\mathcal{D}}$ is satisfiable $\mathcal{D}$ has a solution. Together with Theorem 5.2, the following result follows immediately:

Theorem 5.5. In the description logic obtained by extending $\mathcal{ALC}$ with role-value-maps, the concept satisfiability problem is undecidable.

Remark. By reducing a different undecidable problem (the word problem of semi-groups), one can show that, already in the DL having only the constructors $\sqcap$, $\forall$, and RVMs, *subsumption* is undecidable. The original KL-ONE system had all of these constructors.

6 Structural Subsumption Algorithms

Due to its PSPACE-hardness, reasoning in $\mathcal{ALCN}$ takes exponential time in the worst case. If we want the complexity of reasoning services to be polynomial, we should use DLs that do *not* provide for all Boolean operators (since propositional logic is already NP-hard).

It is typical for such sub-Boolean DLs to provide for conjunction, but to prohibit disjunction and/or negation. Thus, the reduction of subsumption to satisfiability from Theorem 3.14 will not work anymore.

For logics without negation, concept satisfiability is often trivial (i.e., there are no unsatisfiable concepts). For deciding subsumption, so-called *structural algorithms* are used, which work in two phases:

1. normalize the concept terms, and
2. compare the syntactic structure of the normalized terms.

In this chapter, we consider two simple DLs for which structural subsumption algorithms exist:

- $\mathcal{FL}_0$: conjunction ($\sqcap$), value restriction ($\forall$), top concept ($\top$)
- $\mathcal{EL}$: conjunction ($\sqcap$), existential restriction ($\exists$), top concept ($\top$)

Then, we consider a so-called "non-standard" inference service that is important in the area of sub-Boolean DLs: the computation of the *least common subsumer*.

6.1 Subsumption in $\mathcal{FL}_0$

The *normalization* of $\mathcal{FL}_0$-concepts proceeds in two phases:

1. Distribution of value restriction over conjunction, and "treatment" of the $\top$ concept. To this end, we exhaustively apply the following, equivalence-preserving rules:

$$\begin{aligned} \top \sqcap C &\rightarrow C \\ C \sqcap \top &\rightarrow C \\ \forall r.\top &\rightarrow \top \\ \forall r.(C \sqcap D) &\rightarrow \forall r.C \sqcap \forall r.D \end{aligned}$$

This normalizes the input concept into the concept $\top$ or a conjunction of terms of the following form:

$$\forall r_1.\forall r_2 \ldots \forall r_k.A,$$

where A is a concept name and $r_1, \ldots, r_k$ are role names.

Abbreviation: we write $\forall r_1 \ldots r_k.A$ for this, where $r_1 \ldots r_k$ is viewed as a word over the alphabet Σ of role names (if $k = 0$, then this is the empty word ε).

Example. The concept

$$\forall r_1.\big(\forall r_2.A \sqcap \forall r_1.(A \sqcap B)\big) \sqcap A$$

is normalized to

$$\forall r_1 r_2.A \sqcap \forall r_1 r_1.A \sqcap \forall r_1 r_1.B \sqcap \forall \varepsilon.A$$

2. Grouping according to concept names. In place of $\forall w_1.A \sqcap \cdots \sqcap \forall w_l.A$, we write $\forall\{w_1, \ldots, w_l\}.A$, where $\{w_1, \ldots, w_l\}$ is a finite language over Σ (i.e., a finite subset of Σ^*).

   Example. Applied to the example concept, this step yields

$$\forall\{r_1 r_1, r_1 r_2, \varepsilon\}.A \sqcap \forall\{r_1 r_1\}.B.$$

In order to include the top concept in this normal form and to be able to compare concepts which involve different concept names, we also admit the empty language and interpret $\forall\emptyset.A$ as $\top$.

Example. In the presence of the concept names A and B, $\top$ has the normal form

$$\forall\emptyset.A \sqcap \forall\emptyset.B,$$

and $\forall r_1.(\forall r_2.(A \sqcap \forall r_1.A))$ has the normal form

$$\forall\{r_1 r_2, r_1 r_2 r_1\}.A \sqcap \forall\emptyset.B.$$

Given a finite set $\{A_1, \ldots, A_n\}$ of available concept names and an alphabet Σ of role names, each $\mathcal{FL}_0$-concept term C containing only these names is normalized to an expression

$$\mathrm{NF}(C) = \forall L_1.A_1 \sqcap \cdots \sqcap \forall L_n.A_n$$

with $L_1, \ldots, L_n$ finite languages over Σ.

Theorem 6.1. Let C, D be $\mathcal{FL}_0$-concept terms using only concept names from $\{A_1, \ldots, A_n\}$ such that

$$\begin{aligned}\mathrm{NF}(C) &= \forall L_1.A_1 \sqcap \cdots \sqcap \forall L_n.A_n\\ \mathrm{NF}(D) &= \forall M_1.A_1 \sqcap \cdots \sqcap \forall M_n.A_n.\end{aligned}$$

Then $C \sqsubseteq D$ iff $M_i \subseteq L_i$ for $i = 1, \ldots, n$.

Proof.

"$\Leftarrow$": This is trivial, since C (resp. its normal form) then contains all value restrictions that D contains (resp. its normal form).

"⇒": Assume $M_i \not\subseteq L_i$ for some $i \in \{1, \dots, n\}$, i.e. there is a $w = r_1 \dots r_k \in M_i \setminus L_i$. We construct an interpretation $\mathcal{I}$ containing an individual d_0 with $d_0 \in C^\mathcal{I} \setminus D^\mathcal{I}$, which proves $C \not\sqsubseteq D$.

$$\begin{array}{rcl} \Delta^\mathcal{I} & = & \{d_0, \dots, d_k\} \\ A_j^\mathcal{I} & = & \Delta^\mathcal{I} \quad \text{for } j \neq i \\ A_i^\mathcal{I} & = & \Delta^\mathcal{I} \setminus \{d_k\} \\ r_j^\mathcal{I} & = & \{(d_{j-1}, d_j)\} \quad \text{for } 1 \leq j \leq k \end{array}$$

Thus, $\mathcal{I}$ looks as follows:

$$d_0 \xrightarrow{r_1} d_1 \xrightarrow{r_2} \dots \xrightarrow{r_k} d_k$$

where $d_0, \dots, d_{k-1}$ are contained in all $A_j^\mathcal{I}$ and d_k is contained in all $A_j^\mathcal{I}$ $(j \neq i)$, *not* in $A_i^\mathcal{I}$.

We have the following:

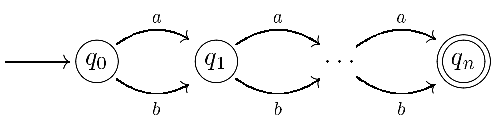
- $d_0 \notin D^\mathcal{I}$ since, for instances of D, the value restriction $\forall r_1 \dots r_k.A_i$ is required.
- $d_0 \in C^\mathcal{I}$ since, for instances of C, the restriction $\forall r_1 \dots r_k.A_i$ is *not* required. All other value restrictions $\forall w.A_j$ $(j \neq i$ or $w \neq r_1 \dots r_k)$ are satisfied:
  - for $j \neq i$: since $A_j^\mathcal{I} = \Delta^\mathcal{I}$
  - for $j = i$ and $w \neq r_1 \dots r_k$: since d_k is not reachable from d_0 with w.

□

Corollary 6.2. Subsumption of $\mathcal{FL}_0$-concept terms is decidable in polynomial time.

Proof. The transformation into normal form requires only polynomial time (exercise!). Checking the inclusion of finite languages can also be done in polynomial time: to decide whether $M_i \subseteq L_i$, in the worst case we have to check, for all words $w \in M_i$ and $v \in L_i$, whether $w = v$. Each such equality test can be done in time $\min(|w|, |v|)$, and $|M_i| \cdot |L_i|$ such tests have to be done. □

We have just shown that the inclusion of finite languages is polynomially decidable if these languages are given "directly" as sets of words. However, finite languages can also be given in the form of acyclic automata. For example,



has size n, but the accepted language contains all words of length n over the alphabet $\{a, b\}$, i.e. 2^n words.

This phenomenon is responsible for the following complexity of the inclusion problem:

Theorem 6.3. The inclusion problem for acyclic (non-deterministic) finite automata over alphabets of cardinality ≥ 2 is co-NP-complete, i.e. it is NP-complete[1] to decide whether the languages accepted by two acyclic automata are *not* contained in each other.

From this, we derive:

Corollary 6.4 (Nebel). Subsumption of $\mathcal{FL}_0$-concepts wrt. TBoxes is co-NP-hard.

Proof. We translate acyclic finite automata $A = (Q, \Sigma, I, F, \Delta)$ with $\Delta \subseteq Q \times \Sigma \times Q$ into a TBox $\mathcal{T}_A$ as follows:

W.l.o.g., we assume A to have exactly one initial state.

Concept names: all states in Q plus an additional name `Fin`

Role names: all symbols from A's alphabet Σ

For each $q \in Q$, $\mathcal{T}_A$ contains the following definition:

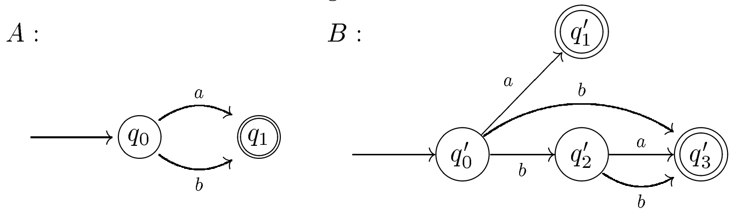
$$q \doteq \bigsqcap_{(q,r,q')\in\Delta} \forall r.q' \quad \sqcap \begin{cases} \texttt{Fin} & \text{if } q \text{ is final state} \\ \top & \text{otherwise} \end{cases}$$

Now let A and B be acyclic finite automata with initial states q_A and q_B, respectively. W.l.o.g., we assume the sets of states of A and B to be disjoint. Define

$$\mathcal{T} := \mathcal{T}_A \cup \mathcal{T}_B.$$

Then we have $q_B \sqsubseteq_{\mathcal{T}} q_A$ iff $L(A) \subseteq L(B)$: it is easily seen that, by expanding q_A (resp. q_B) wrt. $\mathcal{T}$, we obtain concept terms whose normal form is $\forall L(A).\texttt{Fin}$ (resp. $\forall L(B).\texttt{Fin}$). □

Example. Consider the following automata:



[1] For automata in general, this problem is PSPACE-complete.

We have $L(A) = \{a, b\}$, $L(B) = \{a, b, ba, bb\}$, and $\mathcal{T} =$

$$\begin{array}{rclcl}
\mathcal{T}_A & = & \{q_0 & \doteq & \forall a.q_1 \sqcap \forall b.q_1 \sqcap \top, \\
& & q_1 & \doteq & \texttt{Fin}\} \\
\cup & & & & \\
\mathcal{T}_B & = & \{q_0' & \doteq & \forall a.q_1' \sqcap \forall b.q_2' \sqcap \forall b.q_3' \sqcap \top, \\
& & q_1' & \doteq & \texttt{Fin}, \\
& & q_2' & \doteq & \forall a.q_3' \sqcap \forall b.q_3' \sqcap \top, \\
& & q_3' & \doteq & \texttt{Fin}\}
\end{array}$$

By expansion, we obtain:

$$\begin{array}{rcl}
\hat{q_0} & \doteq & \forall a.\texttt{Fin} \sqcap \forall b.\texttt{Fin} \sqcap \top \\
\hat{q_0'} & \doteq & \forall a.\texttt{Fin} \sqcap \forall b.(\forall a.\texttt{Fin} \sqcap \forall b.\texttt{Fin} \sqcap \top) \sqcap \forall b.\texttt{Fin} \sqcap \top
\end{array}$$

$$\begin{array}{rcl}
\mathrm{NF}(\hat{q_0}) & = & \forall\{a, b\}.\texttt{Fin} \\
\mathrm{NF}(\hat{q_0'}) & = & \forall\{a, b, ba, bb\}.\texttt{Fin}
\end{array}$$

Note: By reversing the reduction, i.e. reducing subsumption of $\mathcal{FL}_0$-concepts wrt. TBoxes to the inclusion problem of acyclic automata, we can even show that the subsumption problem is co-NP-complete (exercise).

6.2 Subsumption in $\mathcal{EL}$

In order to decide subsumption in $\mathcal{EL}$ (i.e., $\sqcap$ and $\exists$ are the only constructors), we view $\mathcal{EL}$-concept terms as finite trees, whose

- *nodes* are labeled with sets of concept names,
- *edges* are labeled with role names.

Such trees will be called $\mathcal{EL}$*-trees*.

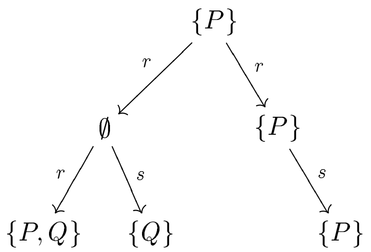
Example 6.5.

1. The $\mathcal{EL}$-concept term

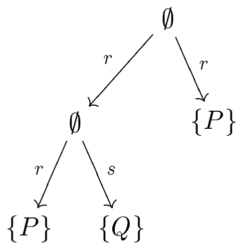
$$C := P \sqcap \exists r.(\exists r.(P \sqcap Q) \sqcap \exists s.Q) \sqcap \exists r.(P \sqcap \exists s.P)$$

corresponds to the tree

$\mathcal{T}_C$:



2. Conversely, the tree



   yields the $\mathcal{EL}$-concept $\exists r.(\exists r.P \sqcap \exists s.Q) \sqcap \exists r.P$.

Note: The label $\emptyset$ corresponds to the top concept, which is dropped if $\emptyset$ is not the label of a leaf, e.g.:

$$\begin{tikzcd} \emptyset \arrow[d, "r"] & \\ \emptyset & \end{tikzcd} \qquad \hat{=} \top \sqcap \exists r.\top \equiv \exists r.\top$$

The exact inductive definition of converting $\mathcal{EL}$-concept terms into $\mathcal{EL}$-trees and vice versa is omitted (exercise).

$\mathcal{EL}$-interpretations as graphs. Since concept names can be viewed as unary relations and role names as binary relations, we can view interpretations as (potentially infinite) graphs, whose

- *nodes* are labeled with sets of concept names,
- *edges* are labeled with role names.

We will call such graphs $\mathcal{EL}$-*graphs*.

Example 6.6.

$$\begin{array}{rcl}
\Delta^{\mathcal{I}} & = & \{a, b, c, d\} \\
P^{\mathcal{I}} & = & \{a, b, c, d\} \\
Q^{\mathcal{I}} & = & \{d\} \\
r^{\mathcal{I}} & = & \{(a, b), (a, c), (b, d)\} \\
s^{\mathcal{I}} & = & \{(b, d), (c, d)\}
\end{array}$$

This interpretation $\mathcal{I}$ corresponds to the graph $\mathcal{G}_\mathcal{I}$:

$$\begin{tikzcd}
& \overset{\{P\}}{a} \arrow[dl, "r"'] \arrow[dr, "r"] & \\
\overset{\{P\}}{b} \arrow[dr, "r"] \arrow[dr, bend right, "s"'] & & \overset{\{P\}}{c} \arrow[dl, "s"] \\
& \underset{\{P,Q\}}{d} &
\end{tikzcd}$$

It is easily checked that $a \in C^\mathcal{I}$ for the concept C from Example 6.5.

For an $\mathcal{EL}$-concept term C and an $\mathcal{EL}$-interpretation $\mathcal{I}$, the problem "$a \overset{?}{\in} C^\mathcal{I}$" can be characterized using homomorphisms.

Definition 6.7. Let N_R be a set of role names and N_C a set of concept names.

1. An $\mathcal{EL}$-*graph* is a triple $\mathcal{G} = (V, E, \ell)$ with
   - V is a set of nodes
   - $E \subseteq V \times N_R \times V$ is a set of labeled edges
   - $\ell : V \to \mathcal{P}(N_C)$ is a labeling function for nodes.

2. Let $\mathcal{G}_i := (V_i, E_i, \ell_i)$ be $\mathcal{EL}$-graphs for $i = 1, 2$. The mapping $h : V_1 \to V_2$ is a *homomorphism* from $\mathcal{G}_1$ to $\mathcal{G}_2$ if

   (h1) $(v, r, w) \in E_1 \Rightarrow (h(v), r, h(w)) \in E_2$ and

   (h2) $\ell(v) \subseteq \ell(h(v))$ for all $v \in V_1$.

Note: $\mathcal{EL}$-trees are special $\mathcal{EL}$-graphs.

Example 6.8. There exists a homomorphism from $\mathcal{T}_C$ in Example 6.5 to the graph $\mathcal{G}_\mathcal{I}$ in Example 6.6.

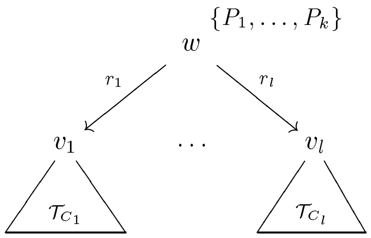
Theorem 6.9. Let C be an $\mathcal{EL}$-concept term and $\mathcal{I}$ an interpretation. Then, for all $a \in \Delta^\mathcal{I}$, the following properties are equivalent:

1. $a \in C^\mathcal{I}$;

2. there exists a homomorphism $h : \mathcal{T}_C \to \mathcal{G}_\mathcal{I}$ with $h(w) = a$, where w is the root of $\mathcal{T}_C$.

Proof. C is of the form $\underbrace{P_1 \sqcap \cdots \sqcap P_k}_{\text{concept names}} \sqcap \underbrace{\exists r_1.C_1 \sqcap \cdots \sqcap \exists r_l.C_l}_{C_i \text{ (possibly) complex concepts}}$.

Thus, the corresponding $\mathcal{EL}$-tree has this form:



"1 ⇒ 2": Let $a \in C^{\mathcal{I}}$. Then, in particular, we have $a \in P_j^{\mathcal{I}}$ for $j = 1, \ldots, k$. Therefore, a is labeled with a set $\ell(a)$ in $\mathcal{G}_{\mathcal{I}}$ that contains all the $P_1, \ldots, P_k$. Thus, (h2) is satisfied for the root w of $\mathcal{T}_C$ if we set $h(w) := a$.

From $a \in C^{\mathcal{I}}$, it also follows that there are $a_1, \ldots, a_\ell \in \Delta^{\mathcal{I}}$ such that $(a, a_j) \in r_j^{\mathcal{I}}$ and $a_j \in C_j^{\mathcal{I}}$. Thus, in $\mathcal{G}_{\mathcal{I}}$ there is an edge from a to a_j labeled with r_j. We set $h(v_j) := a_j$. For these nodes, (h1) is obviously satisfied. By IH, there are homomorphisms $h_j : \mathcal{T}_{C_j} \to \mathcal{G}_{\mathcal{I}}$ mapping v_j to a_j. These yield the remaining part of the homomorphism h from $\mathcal{T}_C$ to $\mathcal{G}_{\mathcal{I}}$.

"2 ⇒ 1": similar.

□

Using Theorem 6.9, we can now characterize subsumption in $\mathcal{EL}$ as follows:

Theorem 6.10. Let C, D be $\mathcal{EL}$-concept terms. Then the following propositions are equivalent:

1. $C \sqsubseteq D$
2. There exists a homomorphism $h : \mathcal{T}_D \to \mathcal{T}_C$ which maps the root v of $\mathcal{T}_D$ to the root w of $\mathcal{T}_C$.

Proof. **"2 ⇒ 1":** It suffices to show that, for each interpretation $\mathcal{I}$ and each $a \in \Delta^{\mathcal{I}}$, we have

$$a \in C^{\mathcal{I}} \Rightarrow a \in D^{\mathcal{I}}$$

If $a \in C^{\mathcal{I}}$, then Theorem 6.9 yields a homomorphism $g : \mathcal{T}_C \to \mathcal{G}_{\mathcal{I}}$ with $g(w) = a$. Then $g \circ h : \mathcal{T}_D \to \mathcal{G}_{\mathcal{I}}$: $(g \circ h)(u) := g(h(u))$ is a homomorphism (!) with $g(h(v)) = g(w) = a$. Again, with Theorem 6.9, we get $a \in D^{\mathcal{I}}$.

"1 ⇒ 2": Assume that there is no homomorphism from $\mathcal{T}_D$ to $\mathcal{T}_C$ such that v is mapped to w. The $\mathcal{EL}$-tree $\mathcal{T}_C$ can be viewed as an $\mathcal{EL}$-graph and thus it corresponds to an interpretation, i.e. there is an interpretation $\mathcal{I}$ with $\mathcal{G}_{\mathcal{I}} = \mathcal{T}_C$. For the element $w \in \Delta^{\mathcal{I}}$, we have:

- $w \in C^{\mathcal{I}}$, since the identity mapping is a homomorphism from $\mathcal{T}_C \to \mathcal{T}_C = \mathcal{G}_{\mathcal{I}}$ mapping w to w.
- $w \notin D^{\mathcal{I}}$, since there is no homomorphism from $\mathcal{T}_D$ to $\mathcal{T}_C$ mapping v to w.

This shows $C \not\sqsubseteq D$.

$\square$

Corollary 6.11. Subsumption of $\mathcal{EL}$-concept terms can be decided in polynomial time.

Proof. To decide $C \sqsubseteq D$, we test for the existence of a homomorphism from $\mathcal{T}_D \to \mathcal{T}_C$. To do this, we must *not* proceed top-down (from the root to the leaves), but rather bottom-up.

Idea: the nodes of $\mathcal{T}_D$ are, additionally, labeled with a set of nodes of $\mathcal{T}_C$. Intuitively, this labeling has the following meaning:

$w \in \ell'(v)$ iff there is a homomorphism

$$\underbrace{\mathcal{T}_D\big|_v}_{\text{subtree of } \mathcal{T}_D \text{ beginning at } v} \to \mathcal{T}_C\big|_w$$

mapping v to w.

Obviously, there is a suitable homomorphism from $\mathcal{T}_D$ to $\mathcal{T}_C$, if the labeling ℓ' of the root of $\mathcal{T}_D$ contains the root of $\mathcal{T}_C$.

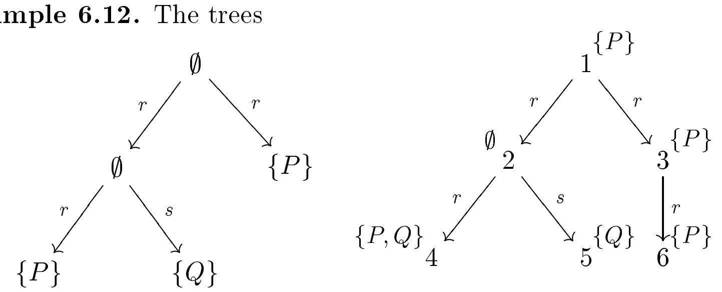
The labeling is constructed bottom-up: for a node v of $\mathcal{T}_D$, we check for all nodes w of $\mathcal{T}_C$, whether the following holds:

- $\ell(v) \subseteq \ell(w)$ and
- for each r-successor v' of v, there is an r-successor w' of w with $w' \in \ell'(v')$.

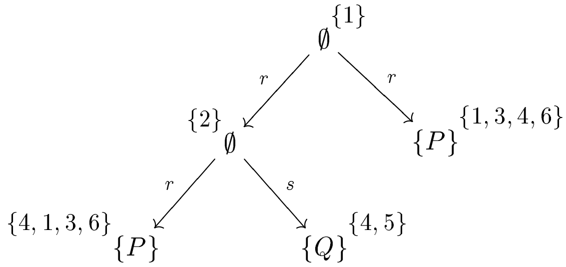
The label $\ell'(v)$ contains precisely those nodes passing this test.

Due to the bottom-up definition, the relevant sets $\ell'(v')$ have already been computed. It is easily seen that this algorithm needs only polynomial time. $\square$

Example 6.12. The trees



yield the left tree with additional labels:



Notes:

1. It is important here that we consider only trees (since these have leaves where we can start our construction). Indeed, in graphs the problem of the existence of a homomorphism is NP-complete.
2. In contrast to $\mathcal{FL}_0$, the addition of TBoxes to $\mathcal{EL}$ does not result in an increase in complexity. Indeed, subsumption of $\mathcal{EL}$-concepts wrt. TBoxes is still polynomial.

6.3 The least common subsumer in $\mathcal{FL}_0$ and $\mathcal{EL}$

For two concept terms C and D, the least common subsumer is intended to describe the *commonalities* of C and D.

Definition 6.13. Let $\mathcal{L}$ be a description logic and let C, D be $\mathcal{L}$-concept terms. The $\mathcal{L}$-concept term E is called *least common subsumer (LCS)* of C and D, if

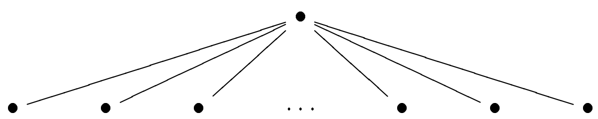
a) $C \sqsubseteq E$ and $D \sqsubseteq E$ and

b) if $C \sqsubseteq F$ and $D \sqsubseteq F$ for some $\mathcal{L}$-concept term F, then $E \sqsubseteq F$.

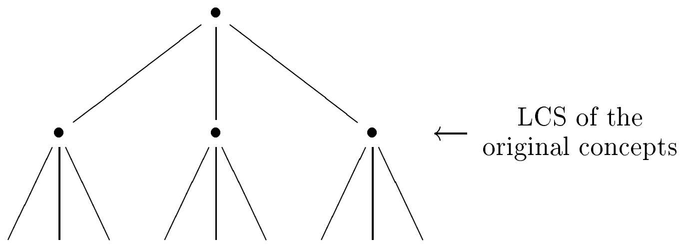
Notes:

1. Such an LCS does not always have to exist. If we consider the DL obtained from $\mathcal{FL}_0$ by removing the $\top$ concept, then the concepts $C := A$ and $D := B$, with A and B different concept names, have no LCS.
2. If an LCS exists, then it is unique up to equivalence, i.e. if E_1 and E_2 are LCS of C, D, then $E_1 \equiv E_2$. Therefore, we talk of *the* LCS of C and D and denote it with $\mathrm{lcs}(C, D)$.
3. For languages with disjunction, $\mathrm{lcs}(C, D) = C \sqcup D$. Therefore, the LCS is trivial in such languages (and not interesting since it does not extract the commonalities).

Application example. The lcs-operation can be used to add more structure to "flat" knowledge bases by introducing "meaningful" intermediate concepts into the concept hierarchy.



becomes



We will show that, in $\mathcal{FL}_0$ and $\mathcal{EL}$, the LCS always exists. For doing this, we will use the characterization of subsumption developed in Theorem 6.1 and Theorem 6.10.

6.3.1 The LCS in $\mathcal{FL}_0$

In $\mathcal{FL}_0$, the LCS operation corresponds to taking the intersection of the languages from the concepts' normal forms.

Theorem 6.14. Let C, D be $\mathcal{FL}_0$-concept terms, and

$$\begin{aligned} \mathrm{NF}(C) &= \forall L_1.A_1 \sqcap \cdots \sqcap \forall L_n.A_n \\ \mathrm{NF}(D) &= \forall M_1.A_1 \sqcap \cdots \sqcap \forall M_n.A_n. \end{aligned}$$

Then $\mathrm{lcs}(C, D)$ has the normal form

$$\forall (L_1 \cap M_1).A_1 \sqcap \cdots \sqcap \forall (L_n \cap M_n).A_n.$$

Proof. We will show that if E is a concept term with the stated NF, then E satisfies conditions a) and b) from Definition 6.13.

a) This follows from $L_i \cap M_i \subseteq L_i$ and $L_i \cap M_i \subseteq M_i$, and from Theorem 6.1.

b) Let F be a concept term with $C \sqsubseteq F$ and $D \sqsubseteq F$. We first show that the NF of F is of the form

$$\forall N_1.A_1 \sqcap \cdots \sqcap \forall N_n.A_n, \tag{$\star$}$$

i.e. it contains no additional concept names. Assume that the NF of F contains a name $B \notin \{A_1, \ldots, A_n\}$. Then $\mathrm{NF}(F)$ contains the conjunct $\forall N.B$ with $N \neq \emptyset$. Since B does not occur in $\mathrm{NF}(C)$ (i.e., only as $\forall \emptyset.B \equiv \top$), Theorem 6.1 yields $N \subseteq \emptyset$ since $C \sqsubseteq F$: contradiction.

Thus, F has the NF ($\star$) and, since $C \sqsubseteq F$ and $D \sqsubseteq F$, Theorem 6.1 yields $N_i \subseteq L_i$ and $N_i \subseteq M_i$, i.e. $N_i \subseteq L_i \cap M_i$. Again by Theorem 6.1, this immediately yields $E \sqsubseteq F$.

□

Corollary 6.15. In $\mathcal{FL}_0$, the LCS always exists and can be computed in polynomial time.

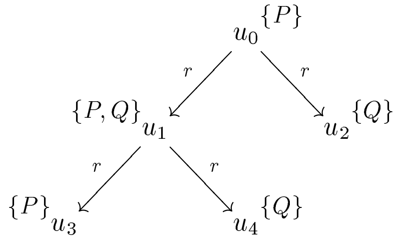
6.3.2 The LCS in $\mathcal{EL}$

The LCS operation in $\mathcal{EL}$ corresponds to taking the product of $\mathcal{EL}$-trees. For an $\mathcal{EL}$-tree $\mathcal{T}$ and a node v of $\mathcal{T}$, we write $\mathcal{T}\big|_v$ to denote the subtree of $\mathcal{T}$ with root v.

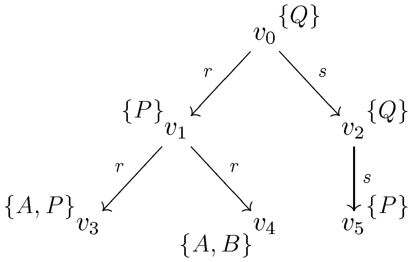
Definition 6.16. Let $\mathcal{T}_1 = (V_1, E_1, \ell_1)$ and $\mathcal{T}_2 = (V_2, E_2, \ell_2)$ be $\mathcal{EL}$-trees with roots u_0 and v_0. The *product* $\mathcal{T}_1 \times \mathcal{T}_2$ of $\mathcal{T}_1$ and $\mathcal{T}_2$ has the root (u_0, v_0) with labeling $\ell(u_0, v_0) = \ell(u_0) \cap \ell(v_0)$. For each pair of edges $(u_0, r, u) \in E_1$ and $(v_0, r, v) \in E_2$, in $\mathcal{T}_1 \times \mathcal{T}_2$ there is an edge $((u_0, v_0), r, (u, v))$, where (u, v) is the root of the product $\mathcal{T}_1\big|_u \times \mathcal{T}_2\big|_v$.

Example 6.17. We consider the following $\mathcal{EL}$-trees:

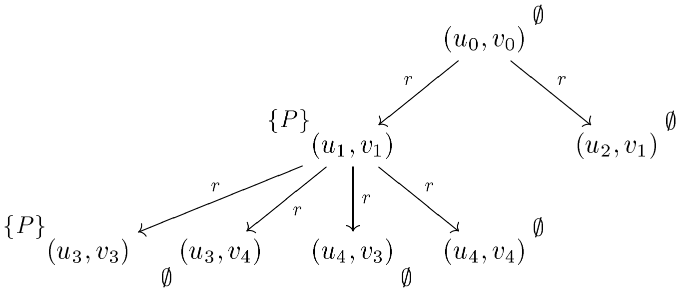
$\mathcal{T}_1$:



$\mathcal{T}_2$:



Then the product is



Theorem 6.18. Let C, D be $\mathcal{EL}$-concept terms with corresponding $\mathcal{EL}$-trees $\mathcal{T}_C$ and $\mathcal{T}_D$. Then the $\mathcal{EL}$-tree $\mathcal{T}_C \times \mathcal{T}_D$ corresponds to $\mathrm{lcs}(C, D)$.

Proof. We have to show that conditions a) and b) of Definition 6.13 are satisfied if E is an $\mathcal{EL}$-concept term with corresponding $\mathcal{EL}$-tree $\mathcal{T}_C \times \mathcal{T}_D$.

a) It is easily seen that the projections

$$\begin{aligned} \pi_1 : \mathcal{T}_C \times \mathcal{T}_D \to \mathcal{T}_C &\quad : \quad (u, v) \mapsto u \quad \text{and} \\ \pi_2 : \mathcal{T}_C \times \mathcal{T}_D \to \mathcal{T}_D &\quad : \quad (u, v) \mapsto v \end{aligned}$$

are homomorphisms from $\mathcal{T}_C \times \mathcal{T}_D$ to $\mathcal{T}_C$ and T_D. This yields $C \sqsubseteq E$ and $D \sqsubseteq E$.

b) Let F be an $\mathcal{EL}$-concept term with $C \sqsubseteq F$ and $D \sqsubseteq F$. Then there are homomorphisms $h_1 : \mathcal{T}_F \to \mathcal{T}_C$ and $h_2 : \mathcal{T}_F \to \mathcal{T}_D$ mapping the root of $\mathcal{T}_F$ to the roots of $\mathcal{T}_C$ and $\mathcal{T}_D$. By taking the product of these homomorphisms:

$$h(v) := (h_1(v), h_2(v))$$

we obtain a homomorphism $h : \mathcal{T}_F \to \mathcal{T}_C \times \mathcal{T}_D$ mapping the root of $\mathcal{T}_F$ to the root of $\mathcal{T}_C \times \mathcal{T}_D$. Thus, $E \sqsubseteq F$.

□

Example 6.17 (cont'd). The tree $\mathcal{T}_1$ corresponds to the concept term

$$C := P \sqcap \exists r.(P \sqcap Q \sqcap \exists r.P \sqcap \exists r.Q) \sqcap \exists r.Q$$

and the tree $\mathcal{T}_2$ corresponds to

$$D := Q \sqcap \exists r.\big(P \sqcap \exists r.(A \sqcap P) \sqcap \exists r.(A \sqcap B)\big) \sqcap \exists s.(Q \sqcap \exists s.P)$$

Then we get

$$\begin{aligned} \mathrm{lcs}(C, D) &= \exists r.(P \sqcap \exists r.P \sqcap \exists r.\top \sqcap \exists r.\top \sqcap \exists r.\top) \sqcap \exists r.\top \\ &= \exists r.(P \sqcap \exists r.P) \end{aligned}$$

Corollary 6.19. The LCS in $\mathcal{EL}$ does always exist and can be computed in polynomial time.

Proof. The product of two $\mathcal{EL}$-trees does always exist and can be computed in quadratic time. □

Note: By iterating the product construction, we can obtain an exponentially big concept term as LCS (LCS of n concepts instead of LCS of two concepts).

7 Modal Logics

If used in a natural way, description logics and first-order logic can only represent *objective* and *time-independent* knowledge.

Modal logics have been developed and used in mathematics, philosophy, computer science, and artificial intelligence. A major motivation for their use in AI is that they are well-suited for representing

- subjective knowledge, i.e. knowledge held by intelligent agents, belief of intelligent agents, knowledge about knowledge, etc. and
- time-dependent knowledge.

Example.

- John believes that Mary knows that he loves her.
- At some time in the future, Mary will love John.

We will only consider *propositional* modal logics (there are also first-order MLs). These logics extend the syntax of propositional logic with *pairs of unary operators* called Box $[m]\varphi$ and Diamond $\langle m\rangle\varphi$. Their *intuitive meaning* depends on the intended application, e.g.:

- knowledge and belief

$$\begin{array}{rcl} [\texttt{robo1}]\varphi & : & \text{“}\texttt{robo1}\ \textit{knows}\text{ that }\varphi\text{ is true”} \\ \langle\texttt{robo1}\rangle\varphi & : & \text{“}\texttt{robo1}\ \textit{believes}\text{ that }\varphi\text{ is true”} \end{array}$$

- time

$$\begin{array}{rcl} [\texttt{future}]\varphi & : & \text{“always in the }\texttt{future}, \varphi\text{ is true”} \\ \langle\texttt{future}\rangle\varphi & : & \text{“at some time in the }\texttt{future}, \varphi\text{ is true”} \end{array}$$

With different intuitive meanings, the expected “behavior” of modal operators changes. For example, an agent should know only true facts, i.e. $[\texttt{robo1}]\varphi$ should imply φ. In contrast, there is clearly no reason for $[\texttt{future}]\varphi$ to imply φ. Such different behavior is expressed by means of axioms or by restricting the class of semantic structures in which modal logics are interpreted.

First, we consider the *basic modal logic* K_n underlying most other modal logics (n stands for n pairs of modal operators; K_n is called a *multi*-modal logic if $n \geq 1$).

7.1 The modal logic K_n

Definition 7.1 (Syntax). Let A be a set of propositional letters, and $M_n = \{m_1, \ldots, m_n\}$ a set of *modal parameters*. The set of K_n-formulas is defined inductively:

- each propositional letter is a K_n-formula,
- if φ and ψ are K_n-formulas, then also $\varphi \wedge \psi$, $\varphi \vee \psi$, $\neg\varphi$,
- if φ is a K_n-formula and $m \in M_n$, then $[m]\varphi$ and $\langle m\rangle\varphi$ are K_n-formulas.

Note: Other Boolean operators ($\rightarrow$, $\leftrightarrow$, etc.) are introduced as abbreviations.

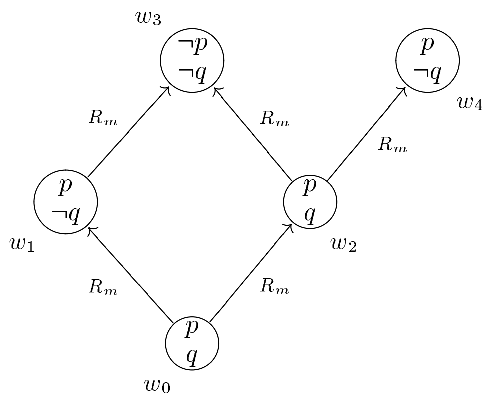
Example. $M_3 = \{\texttt{robo1}, \texttt{robo2}, \texttt{future}\}$
`robo1` knows that, at some time in the `future`, `robo2` believes that a is true:
$[\texttt{robo1}]\langle\texttt{future}\rangle\langle\texttt{robo2}\rangle a$

Definition 7.2 (Semantics).

1. Let A and M_n be given. A *Kripke structure* $\mathcal{K} = (\mathcal{W}, \pi, R)$ consists of
   - a set $\mathcal{W}$ of *possible worlds*,
   - a *valuation function* $\pi : \mathcal{W} \times A \rightarrow \{0, 1\}$;
     instead of $\pi(w, a) = b$, we write $\pi_w(a) = b$.
     Thus, for each world w, π_w is a valuation of propositional letters in the sense of propositional logic.
   - R contains, for each modal parameter m, an *accessibility relation* $R_m \subseteq \mathcal{W} \times \mathcal{W}$. We call $w' \in \mathcal{W}$ *m-accessible* from $w \in \mathcal{W}$ if $(w, w') \in R_m$.

   Intuition. For each world w, π_w describes what is true in this world.
   - The variable a holds always in the future if it holds in all `future`-accessible worlds.
   - An agent `robo1` knows a if a holds in all `robo1`-accessible worlds.
2. *Satisfaction* of a K_n-formula φ in a world w of a Kripke structure $\mathcal{K}$ is defined by structural induction:
   - $\mathcal{K}, w \models_{K_n} a$ iff $\pi_w(a) = 1$ for $a \in A$.
   - Booleans are interpreted as usual, e.g.
     $\mathcal{K}, w \models_{K_n} \varphi \wedge \psi$ iff $\mathcal{K}, w \models_{K_n} \varphi$ and $\mathcal{K}, w \models_{K_n} \psi$.
   - $\mathcal{K}, w \models_{K_n} [m]\varphi$ iff for all $w' \in \mathcal{W}$ with $(w, w') \in R_m$, we have $\mathcal{K}, w' \models_{K_n} \varphi$.
   - $\mathcal{K}, w \models_{K_n} \langle m\rangle\varphi$ iff there exists $w' \in \mathcal{W}$ with $(w, w') \in R_m$ and $\mathcal{K}, w' \models_{K_n} \varphi$.
3. A K_n-formula φ is *valid* ($\models_{K_n} \varphi$) if it is true in every world of every Kripke structure. It is *satisfiable*, if there is a Kripke structure $\mathcal{K}$ and a world w of $\mathcal{K}$ such that $\mathcal{K}, w \models_{K_n} \varphi$.

Example.



$$
\begin{array}{lll}
\mathcal{K}, w_0 & \models_{K_n} & p \wedge q \\
\mathcal{K}, w_0 & \models_{K_n} & \langle m \rangle q \\
\mathcal{K}, w_0 & \models_{K_n} & \langle m \rangle \neg q \\
\mathcal{K}, w_0 & \not\models_{K_n} & [m] \neg q \\
\mathcal{K}, w_0 & \models_{K_n} & [m] p \\
\mathcal{K}, w_0 & \models_{K_n} & \langle m \rangle [m](\neg p \wedge \neg q) \\
\mathcal{K}, w_2 & \not\models_{K_n} & \langle m \rangle q
\end{array}
$$

Lemma 7.3.

1. The formulas $\langle m \rangle \varphi$ and $\neg [m] \neg \varphi$ are equivalent, i.e. $\forall \mathcal{K}, w$:
$$\mathcal{K}, w \models \langle m \rangle \varphi \quad \text{iff} \quad \mathcal{K}, w \models \neg [m] \neg \varphi.$$
2. If φ is a propositional tautology, then $\models_{K_n} \varphi$.
3. If $\models_{K_n} \varphi$ and $\models_{K_n} \varphi \to \psi$, then $\models_{K_n} \psi$.
4. $\models_{K_n} \big([m]\varphi \wedge [m](\varphi \to \psi)\big) \to [m]\psi$.
5. If $\models_{K_n} \varphi$ then $\models_{K_n} [m]\varphi$.
6. $\not\models_{K_n} a \to [m]a$.

Proof.

1. This is trivial by the definition of the semantics.
2. Propositional tautologies are made true by every propositional valuation, thus also by every π_w.
3. Assume $\models_{K_n} \varphi$ and $\models_{K_n} \varphi \to \psi$. To show $\models_{K_n} \psi$, we consider a Kripke structure $\mathcal{K}$ and a world w. By assumption, $\mathcal{K}, w \models \varphi$ and $\mathcal{K}, w \models \varphi \to \psi$. Thus, the semantics yields $\mathcal{K}, w \models \psi$.

4. Let $\mathcal{K}$ be a Kripke structure and w be a world of $\mathcal{K}$. Assume $\mathcal{K}, w \models [m]\varphi \wedge [m](\varphi \rightarrow \psi)$. We have to show that $\mathcal{K}, w \models_{K_n} [m]\psi$. To this end, consider arbitrary w' with $(w, w') \in R_m$. Then we have $\mathcal{K}, w' \models \varphi$ and $\mathcal{K}, w' \models \varphi \rightarrow \psi$, yielding $\mathcal{K}, w' \models \psi$ as required.

5. If φ holds in every world, then in particular it holds in every world that is m-accessible from another world.

6. Counterexample: $\mathcal{K}$: $\underset{w_0}{a} \xrightarrow{R_m} \neg a$. Then $\mathcal{K}, w_0 \not\models a \rightarrow [m]a$.

□

7.2 Relationship to Description Logics

Considering the semantics of K_n, it is obvious that the model operators are similar to the value- and exists-restrictions of description logics.

$$\begin{aligned} \text{“in all accessible worlds”} &\ \hat{=}\ \text{“for all role successors”} \\ \text{“there is an accessible world”} &\ \hat{=}\ \text{“there is a role successor”} \end{aligned}$$

In 1991, K. Schild noticed that $\mathcal{ALC}$ is a syntactic variant of K_n:

$\mathcal{ALC}$	K_n
concept name A	propositional letter a
role name r	modal parameter m
$C \sqcap D$	$\varphi_C \wedge \varphi_D$
$C \sqcup D$	$\varphi_C \vee \varphi_D$
$\neg C$	$\neg \varphi_C$
$\forall r_m.C$	$[m]\varphi_C$
$\exists r_m.C$	$\langle m \rangle \varphi_C$
$\top$	$a \vee \neg a$
$\bot$	$a \wedge \neg a$

Example.

$$\begin{aligned} C \doteq A \sqcap \exists r_m.(B \sqcap \neg A) &\ \leadsto\ \varphi_C = a \wedge \langle m \rangle (b \wedge \neg a) \\ \varphi = [m]\langle m' \rangle (a \wedge \neg b) &\ \leadsto\ C_\varphi \doteq \forall r_m.\exists r_{m'}(A \sqcap \neg B) \end{aligned}$$

Relationship between the semantics

Interpretation $\mathcal{I}$	Kripke structure $\mathcal{K} = (\mathcal{W}, \pi, R)$
$\Delta^\mathcal{I}$ (individuals)	$\mathcal{W}$ (worlds)
$r_m^\mathcal{I} \subseteq \Delta^\mathcal{I} \times \Delta^\mathcal{I}$	$R_m \subseteq \mathcal{W} \times \mathcal{W}$
$A^\mathcal{I} \subseteq \Delta^\mathcal{I}$	$\{w \in \mathcal{W} \mid \pi_w(a) = 1\}$

Obviously, every interpretation $\mathcal{I}$ corresponds to exactly one Kripke structure $\mathcal{K}$ and vice versa.

Lemma 7.4. Let $\mathcal{I}$ be an interpretation and $\mathcal{K}$ the corresponding Kripke structure. Then, for each $\mathcal{ALC}$-concept term C, we have

$$C^{\mathcal{I}} = \{w \in \mathcal{W} \mid \mathcal{K}, w \models_{K_n} \varphi_C\}.$$

Proof. By structural induction. □

Theorem 7.5. An $\mathcal{ALC}$-concept term C is satisfiable iff the K_n-formula φ_C is satisfiable.

Corollary 7.6. Satisfiability of $\mathcal{ALC}$-concept terms can be reduced in linear time to the satisfiability of K_n-formulas and vice versa.

Note:

1. It is an immediate consequence of the corollary that satisfiability of K_n-formulas is PSPACE-complete (see Corollary 4.17). This was, however, previously known for modal logic. Tableau algorithms similar to those for DLs had also been developed.
2. The correspondence allows to transfer results and methods from modal logics, which has been investigated for a longer time and more intensely, to description logics and vice versa (e.g. the finite model property, see below).
3. Some concept constructors have no counterpart in modal logic (e.g. RVMs), and some have only recently been considered in ML (e.g. number restrictions, which are called "graded modalities" in ML terminology).
4. As $\mathcal{ALC}$, K_n can thus be viewed as a fragment of FOL.

7.3 Finite model property

A modal or description logic is said to have the *finite model* property (FMP) if every satisfiable formula is satisfiable in a Kripke structure $\mathcal{K} = (\mathcal{W}, \pi, R)$ with a finite set $\mathcal{W}$. The finite model property is very useful since it has important consequences regarding (partial) decidability:

- If a modal/description logic has the FMP then satisfiability is partially decidable (by enumeration of finite models).
- If additionally the modal/description logic is a fragment of FOL, then satisfiability and validity are even decidable (by simultaneous enumeration of the finite models and valid formulas).

Since the tableau algorithm for $\mathcal{ALCN}$, applied to $\{a : C\}$, yields a finite canonical model for every satisfiable concept C, the correspondence between $\mathcal{ALC}$ and K_n shows that K_n enjoys the FMP.

Alternatively it is possible to show the FMP of K_n (and thus also for $\mathcal{ALC}$) by using the *filtration technique* that is also applicable to many other modal and description logics.

For a K_n-formula φ, we will use $\mathrm{sub}(\varphi)$ to denote the set of all subformulas of φ, i.e.

- $\mathrm{sub}(a) = \{a\}$ for propositional letters a,
- $\mathrm{sub}(\varphi \wedge \psi) = \{\varphi \wedge \psi\} \cup \mathrm{sub}(\varphi) \cup \mathrm{sub}(\psi)$,
- $\mathrm{sub}(\varphi \vee \psi)$ analogously,
- $\mathrm{sub}(\neg\varphi) = \{\neg\varphi\} \cup \mathrm{sub}(\varphi)$,
- $\mathrm{sub}(\langle m\rangle\varphi) = \{\langle m\rangle\varphi\} \cup \mathrm{sub}(\varphi)$,
- $\mathrm{sub}([m]\varphi)$ analogously.

Obviously, $|\mathrm{sub}(\varphi)| \leq |\varphi|$, i.e. the number of subformulas of φ is linearly bounded by the size of φ (the number of symbols needed to represent it).

Theorem 7.7. Let φ be a K_n-formula of length m. Then φ is satisfiable iff there is a Kripke structure $\hat{\mathcal{K}} = (\hat{\mathcal{W}}, \hat{\pi}, \hat{R})$ with $|\hat{\mathcal{W}}| \leq 2^m$ and $\hat{\mathcal{K}}, \hat{w}_0 \models_{K_n} \varphi$ for some $w_0 \in \hat{\mathcal{W}}$.

Note: This is even a *bounded model property.*

Proof.

"⇐": This is trivial.

"⇒": Let $\mathcal{K} = (\mathcal{W}, \pi, R)$ be a Kripke structure with $\mathcal{K}, w_0 \models_{K_n} \varphi$ for some $w_0 \in \mathcal{W}$. We define a *relation* $\sim$ on $\mathcal{W}$:

$$w \sim w' \text{ iff for all } \psi \in \mathrm{sub}(\varphi)\text{: } \mathcal{K}, w \models \psi \text{ iff } \mathcal{K}, w' \models \psi.$$

It is easily shown that $\sim$ is an equivalence relation with at most 2^m equivalence classes (since $|\mathrm{sub}(\varphi)| \leq m$). For $w \in \mathcal{W}$, let $\hat{w}$ denote the equivalence class of w wrt. $\sim$, i.e.

$$\hat{w} := \{w' \in \mathcal{W} \mid w \sim w'\}.$$

We now define:

- $\hat{\mathcal{W}} := \{\hat{w} \mid w \in \mathcal{W}\}$.
- $\hat{\pi}_{\hat{w}}(a) := \pi_w(a)$ for all propositional letters in $\mathrm{sub}(\varphi)$. This is well-defined by the definition of $\sim$ and since $a \in \mathrm{sub}(\varphi)$.
- $\hat{R} : (\hat{w}_1, \hat{w}_2) \in \hat{R}_m$ iff there are $w_1' \in \hat{w}_1$ and $w_2' \in \hat{w}_2$ such that $(w_1', w_2') \in R_m$.
- $\hat{\mathcal{K}} := (\hat{\mathcal{W}}, \hat{\pi}, \hat{R})$.

Claim: For all $\psi \in \mathrm{sub}(\varphi)$ and $w \in \mathcal{W}$, we have $\mathcal{K}, w \models \psi$ iff $\hat{\mathcal{K}}, \hat{w} \models \psi$.

Proof: Structural induction:

- Propositional letters: this is trivial by definition of $\hat{\pi}$.
- Conjunction:

$$\begin{array}{lll} \mathcal{K}, w \models \psi_1 \wedge \psi_2 & \text{iff} & \mathcal{K}, w \models \psi_1 \text{ and } \mathcal{K}, w \models \psi_2 \\ & \overset{\text{by I.H.}}{\text{iff}} & \hat{\mathcal{K}}, \hat{w} \models \psi_1 \text{ and } \hat{\mathcal{K}}, \hat{w} \models \psi_2 \\ & \text{iff} & \hat{\mathcal{K}}, \hat{w} \models \psi_1 \wedge \psi_2 \end{array}$$

- Disjunction and negation: this is similar to conjunction.
- Box operator:
  i) $\mathcal{K}, w \models [m]\varphi_1$. We show $\hat{\mathcal{K}}, \hat{w} \models [m]\varphi_1$. Let $\hat{w}_1 \in \hat{\mathcal{W}}$ with $(\hat{w}, \hat{w}_1) \in \hat{R}_m$.
  We show: $\hat{\mathcal{K}}, \hat{w}_1 \models \varphi_1$.
  From $(\hat{w}, \hat{w}_1) \in \hat{R}_m$, it follows that there are $w' \in \hat{w}$ and $w_1' \in \hat{w}_1$ such that $(w', w_1') \in R_m$. Since $w' \sim w$ and $\mathcal{K}, w \models [m]\varphi_1$, we have $\mathcal{K}, w' \models [m]\varphi_1$ and thus $\mathcal{K}, w_1' \models \varphi_1$ since $(w', w_1') \in R_m$. Since $w_1' \sim w_1$, we have $\mathcal{K}, w_1 \models \varphi_1$ yielding $\hat{\mathcal{K}}, \hat{w}_1 \models \varphi_1$.

  ii) Assume $\hat{\mathcal{K}}, \hat{w} \models [m]\psi_1$.
  We show: $\mathcal{K}, w \models [m]\psi_1$.
  Fix a $w_1 \in \mathcal{W}$ such that $(w, w_1) \in R_m$. By definition of $\hat{R}_m$, we have $(\hat{w}, \hat{w}_1) \in \hat{R}_m$. Thus, $\hat{\mathcal{K}}, \hat{w}_1 \models \psi_1$ by the semantics. The induction hypothesis yields $\mathcal{K}, w_1 \models \psi_1$. This shows $\mathcal{K}, w \models [m]\psi_1$.
- Diamond operator: this is similar to the box operator.

This concludes the proof of the claim.

Since $\varphi \in \mathrm{sub}(\varphi)$ and $w_0 \in \mathcal{W}$, the claim yields that $\mathcal{K}, w_0 \models \varphi$ implies $\hat{\mathcal{K}}, \hat{w}_0 \models \varphi$.

$\square$

7.4 An Axiomatization for K_n

Goal:

$$\left.\begin{array}{c} \text{Axiom schemata} \\ \\ \text{Inference rules} \end{array}\right\} \text{derive exactly the valid } K_n\text{-formulas}$$

By adding more axiom schemata, we can describe the behaviour of modal operators for knowledge, belief, time, etc. For example, if we want to use [ ] for talking about knowledge, then $[\,]\varphi \to \varphi$ is a natural axiom schema. We see that such additions correspond to conditions on accessibility relations in Kripke frames (such as reflexivity, symmetry, transitivity, ...).

W.l.o.g., in the following we assume that formulas contain only $[m]$-modalities, but no $\langle m\rangle$-modalities (cf Lemma 7.3, 1).

Axiom schemata

(Taut) all instances of propositional tautologies

$$\underbrace{p \vee \neg p}_{\text{propositional tautology}} \quad\rightsquigarrow\quad \underbrace{[m]a \vee \neg[m]a}_{\text{ML axiom}}$$

(K) $\big([m]\varphi \wedge [m](\varphi \to \psi)\big) \to [m]\psi$

A formula φ is *deducible/provable* in K_n (written $\vdash_{K_n} \varphi$) if it can be obtained from axioms (i.e. instances of (Taut) or (K)) by applying inference rules a finite number of times. In this case, φ is called a *theorem* of K_n.

Inference rules

(MP) (Modus Ponens) If $\vdash \varphi$ and $\vdash \varphi \to \psi$, then $\vdash \psi$

(GE) (Generalization) If $\vdash \varphi$, then $\vdash [\ ]\varphi$

Example 7.8. For arbitrary formulas φ and ψ, we have:

$$\vdash_{K_n} [m](\varphi \to \psi) \to ([m]\varphi \to [m]\psi)$$

Proof.

1. The formula $\big((a \wedge b) \to c\big) \to \big(b \to (a \to c)\big)$ is a propositional tautology.
2. Thus (Taut) yields:

$$\vdash_{K_n} \big((\underbrace{[m]\varphi}_{a} \wedge \underbrace{[m](\varphi \to \psi)}_{b}) \to \underbrace{[m]\psi}_{c}\big) \to \big((\underbrace{[m](\varphi \to \psi)}_{b}) \to (\underbrace{[m]\varphi}_{a} \to \underbrace{[m]\psi}_{c})\big)$$

3. The left-hand side of 2. is an instance of (K), thus:

$$\vdash_{K_n} \big([m]\varphi \wedge [m](\varphi \to \psi)\big) \to [m]\psi$$

4. Then (MP) yields the right-hand side of 2.:

$$\vdash_{K_n} [m](\varphi \to \psi) \to ([m]\varphi \to [m]\psi)$$

□

The argument used in this example can be generalized as follows:

Lemma 7.9. Let φ and ψ be formulas. If $\varphi \to \psi$ is instance of a propositional tautology, then:

1. $\vdash_{K_n} \varphi$ implies $\vdash_{K_n} \psi$;
2. $\vdash_{K_n} [m]\varphi$ implies $\vdash_{K_n} [m]\psi$.

Proof.

1. This follows as in example 7.8.
2.

$\vdash_{K_n} \varphi \to \psi$	(Taut)
$\vdash_{K_n} [m](\varphi \to \psi)$	(GE)
$\vdash_{K_n} [m]\varphi$	assumption
$\vdash_{K_n} [m](\varphi \to \psi) \to ([m]\varphi \to [m]\psi)$	Example 7.8
$\vdash_{K_n} [m]\psi$	2× (MP)

□

Note: In particular, this implies that propositionally equivalent formulas can be substituted for one another.

Lemma 7.10.

$$\begin{aligned}\vdash_{K_n} \quad & [m]\big(\varphi_1 \to (\varphi_2 \to \ldots(\varphi_k \to \psi)\ldots)\big) \\ & \to \big([m]\varphi_1 \to ([m]\varphi_2 \to \ldots([m]\varphi_k \to [m]\psi)\ldots)\big)\end{aligned}$$

Proof. Induction on k.

$k = 1$**:** See Example 7.8.

$k - 1 \to k$**:**

1. With induction hypothesis, we have:

$$\vdash_{K_n} \underbrace{[m]\,\overbrace{\big(\varphi_2 \to \ldots(\varphi_k \to \psi)\ldots\big)}^{\varphi}}_{c} \to \underbrace{\big([m]\varphi_2 \to \ldots([m]\varphi_k \to [m]\psi)\ldots\big)}_{d}$$

2. Example 7.8 yields:

$$\vdash_{K_n} \underbrace{[m](\varphi_1 \to \varphi)}_{a} \to \big(\underbrace{[m]\varphi_1}_{b} \to \underbrace{[m]\varphi}_{c}\big)$$

3. The following formula is a propositional tautology:

$$\vdash_{K_n} \big(a \to (b \to c)\big) \to \big((c \to d) \to (a \to (b \to d))\big)$$

4. After applying (MP) twice, we obtain $\vdash_{K_n} \big(a \rightarrow (b \rightarrow d)\big)$, i.e.:

$$\begin{aligned}\vdash_{K_n} \quad & [m]\Big(\varphi_1 \rightarrow \big(\varphi_2 \rightarrow \ldots (\varphi_k \rightarrow \psi) \ldots\big)\Big) \\ & \rightarrow \Big([m]\varphi_1 \rightarrow \big([m]\varphi_2 \rightarrow \ldots ([m]\varphi_k \rightarrow [m]\psi) \ldots\big)\Big)\end{aligned}$$

□

We now want to show:

Theorem 7.11.

$$\vdash_{K_n} \varphi \quad \text{iff} \quad \models_{K_n} \varphi.$$

The direction "⇒" (Soundness) is an easy consequence of Lemma 7.3:

(2)+(4) show that all axioms are valid

(3)+(5) show that inference rules preserve validity

The direction "⇐" (Completeness) is more involved. We start with showing that any consistent set of formulas can be extended to a maximally consistent set. Here, consistency is a syntactic notion that is defined wrt. our axiomatization.

Definition 7.12.

1. A formula φ is called *(K_n-) consistent*[2] if $\neg\varphi$ is not a theorem, i.e. $\not\vdash_{K_n} \neg\varphi$.
2. The set $\{\varphi_1, \ldots, \varphi_n\}$ is *consistent* if $\varphi_1 \wedge \cdots \wedge \varphi_n$ is consistent.
3. An infinite set of formulas is *consistent* if every finite subset is consistent.
4. A set Γ of formulas is *maximally consistent* if it is consistent and, for each formula $\varphi \notin \Gamma$, $\Gamma \cup \{\varphi\}$ is inconsistent.

Lemma 7.13.

1. Each consistent set of formulas can be extended into a maximally consistent set.
2. If Γ is a maximally consistent set, then
   (a) for each formula φ, either $\varphi \in \Gamma$ or $\neg\varphi \in \Gamma$, but not both;
   (b) $\varphi \wedge \psi \in \Gamma$ iff $\varphi \in \Gamma$ and $\psi \in \Gamma$;
   (c) if $\varphi \in \Gamma$ and $\varphi \rightarrow \psi \in \Gamma$, then $\psi \in \Gamma$;
   (d) if $\vdash_{K_n} \varphi$, then $\varphi \in \Gamma$.

[2] In the following, we will simply use the term consistent and omit "K_n". Note that the meaning still depends on the K_n axioms and rules.

Proof. We start with showing that, for any consistent set Γ and formula φ, we have

$$\Gamma \cup \{\varphi\} \text{ is consistent or } \Gamma \cup \{\neg\varphi\} \text{ is consistent}$$

Assume that both are inconsistent. Then, there are $\varphi_1, \ldots, \varphi_n \in \Gamma$ and $\psi_1, \ldots, \psi_\ell \in \Gamma$ such that

$$\begin{aligned} &\vdash_{K_n} \neg(\underbrace{\varphi_1 \wedge \cdots \wedge \varphi_n}_{a} \wedge \underbrace{\varphi}_{b}) \\ \text{and} \quad &\vdash_{K_n} \neg(\underbrace{\psi_1 \wedge \cdots \wedge \psi_\ell}_{a'} \wedge \underbrace{\neg\varphi}_{\neg b}) \end{aligned}$$

Note: since Γ is consistent, φ resp. $\neg\varphi$ must be contained in the inconsistent finite subset of $\Gamma \cup \{\varphi\}$ resp. $\Gamma \cup \{\neg\varphi\}$.

The following is a propositional tautology:

$$\neg(a \wedge b) \to \big(\neg(a' \wedge \neg b) \to \neg(a \wedge a')\big)$$

$2\times$ (MP) yields:

$$\vdash_{K_n} \neg(\varphi_1 \wedge \cdots \wedge \varphi_n \wedge \psi_1 \wedge \cdots \wedge \psi_\ell)$$

This is a contradiction to the consistency of Γ.

ad 1: Let $\varphi_0, \varphi_1, \ldots$ be an enumeration of all K_n-formulas. For a given consistent Γ, we define:

$$\begin{aligned} \Gamma_0 &:= \Gamma \\ \Gamma_{i+1} &:= \begin{cases} \Gamma_i \cup \{\varphi_i\} & \text{if this is consistent} \\ \Gamma_i \cup \{\neg\varphi_i\} & \text{otherwise} \end{cases} \\ \hat{\Gamma} &:= \bigcup_{i \geq 0} \Gamma_i \end{aligned}$$

i) Clearly $\Gamma \subseteq \hat{\Gamma}$, i.e. $\hat{\Gamma}$ extends Γ.

ii) $\hat{\Gamma}$ is consistent:
By definition, all Γ_i are consistent (note: if $\Gamma_i \cup \{\varphi\}$ is inconsistent, then $\Gamma_i \cup \{\neg\varphi\}$ is consistent).
If $\hat{\Gamma}$ was inconsistent, then at least one finite subset of it would be inconsistent as well. However, each such finite subset is clearly contained in a set Γ_i.

iii) $\hat{\Gamma}$ is maximal:
Suppose $\varphi \notin \hat{\Gamma}$. Let $\varphi = \varphi_i$. Since $\varphi_i \notin \hat{\Gamma} \supseteq \Gamma_{i+1}$, $\Gamma_i \cup \{\varphi_i\}$ was inconsistent. Thus, $\hat{\Gamma} \cup \{\varphi_i\}$ is also inconsistent.

ad 2: Let Γ be maximally consistent.

(a) Since, for each φ, we have that $\Gamma \cup \{\varphi\}$ or $\Gamma \cup \{\neg\varphi\}$ is consistent, maximality of Γ yields $\varphi \in \Gamma$ or $\neg\varphi \in \Gamma$. Assume $\varphi \in \Gamma$ and $\neg\varphi \in \Gamma$. Since $\neg(\varphi \wedge \neg\varphi)$ is an instance of a propositional tautology, $\vdash_{K_n} \neg(\varphi \wedge \neg\varphi)$ and thus Γ is inconsistent. $\lightning$

(b) **"$\Rightarrow$":** Let $\varphi \wedge \psi \in \Gamma$. Assume $\varphi \notin \Gamma$. Then $\neg\varphi \in \Gamma$. Now, $\neg((\varphi \wedge \psi) \wedge \neg\varphi)$ is an instance of a propositional tautology, i.e. $\vdash_{K_n} \neg((\varphi \wedge \psi) \wedge \neg\varphi)$, which shows that Γ is inconsistent. $\lightning$ (The proof for ψ is analogous.)

"$\Leftarrow$": Let $\varphi \in \Gamma$ and $\psi \in \Gamma$. Assume $\varphi \wedge \psi \notin \Gamma$. Then $\neg(\varphi \wedge \psi) \in \Gamma$. Now, $\neg(\varphi \wedge \psi \wedge \neg(\varphi \wedge \psi))$ is instance of a propositional tautology, showing that Γ is inconsistent. $\lightning$

(c) Assume $\varphi \in \Gamma$, $\varphi \to \psi \in \Gamma$, but $\psi \notin \Gamma$. Then $\neg\psi \in \Gamma$. Now, $\neg(\varphi \wedge (\varphi \to \psi) \wedge \neg\psi)$ is instance of a propositional tautology implying that Γ is inconsistent. $\lightning$

(d) Assume $\vdash_{K_n} \varphi$ and $\varphi \notin \Gamma$. Then $\neg\varphi \in \Gamma$. Since $\vdash_{K_n} \varphi$ and $\varphi \to \neg\neg\varphi$ is instance of a propositional tautology, we have $\vdash_{K_n} \neg\neg\varphi$. Since $\neg\varphi \in \Gamma$, Γ is thus inconsistent. $\lightning$

$\square$

Proof of completeness ("$\Leftarrow$" of Theorem 7.11)
We show the following:

$$\text{Each } K_n\text{-consistent formula is satisfiable in a Kripke structure.} \qquad (\star)$$

This implies completeness as follows:

Let $\models_{K_n} \varphi$. Assume $\not\vdash_{K_n} \varphi$. Then $\not\vdash_{K_n} \neg\neg\varphi$, i.e. $\neg\varphi$ is consistent. By $(\star)$, $\neg\varphi$ is satisfiable, i.e. there is a Kripke structure $\mathcal{K}$ and a world w such that $\mathcal{K}, w \models \neg\varphi$. This is a contradiction to the validity of φ.

In order to show $(\star)$, we construct the *canonical Kripke structure* in which *every* K_n-consistent formula is satisfied. $\mathcal{K}^{K_n} := (\mathcal{W}^{K_n}, \pi^{K_n}, R^{K_n})$ with the following properties:

- $\mathcal{W}^{K_n} := \{w_\Gamma \mid \Gamma \text{ maximal consistent set}\}$, i.e. for each maximal consistent set, there is exactly one world in $\mathcal{W}^{K_n}$.
- For $a \in A$: $\pi^{K_n}_{w_\Gamma}(a) := \begin{cases} 1 & a \in \Gamma \\ 0 & a \notin \Gamma \end{cases}$
- For $m \in M_n$ and a set of formulas Γ, we define $\Gamma/m := \{\varphi \mid [m]\varphi \in \Gamma\}$ and $R_m := \{(w_\Gamma, w_\Delta) \mid \Gamma/m \subseteq \Delta\}$.

Theorem 7.14. For all maximal consistent sets Γ and all formulas φ, we have

$$\mathcal{K}^{K_n}, w_\Gamma \models \varphi \text{ iff } \varphi \in \Gamma.$$

Proof. By structural induction:

i) $\varphi = a$ $(a \in A)$:
Then, $\mathcal{K}^{K_n}, w_\Gamma \models a$ iff $\pi_{w_\Gamma}^{K_n}(a) = 1$ iff $a \in \Gamma$ by definition of $\pi_{w_\Gamma}^{K_n}$.

ii) $\varphi = \psi_1 \wedge \psi_2$:

$$\begin{array}{rcl} \mathcal{K}^{K_n}, w_\Gamma \models \psi_1 \wedge \psi_2 & \text{iff} & \mathcal{K}^{K_n}, w_\Gamma \models \psi_1 \text{ and } \mathcal{K}^{K_n}, w_\Gamma \models \psi_2 \\ & \overset{\text{by Induction}}{\text{iff}} & \psi_1 \in \Gamma \text{ and } \psi_2 \in \Gamma \\ & \overset{\text{by Lemma 7.13 2b)}}{\text{iff}} & \psi_1 \wedge \psi_2 \in \Gamma \end{array}$$

iii) $\varphi = \neg\psi$:

$$\begin{array}{rcl} \mathcal{K}^{K_n}, w_\Gamma \models \neg\psi & \text{iff} & \mathcal{K}^{K_n}, w_\Gamma \not\models \psi \\ & \overset{\text{Ind.}}{\text{iff}} & \psi \notin \Gamma \\ & \overset{\text{Lemma 7.13 2a)}}{\text{iff}} & \neg\psi \in \Gamma \end{array}$$

iv) $\varphi = [m]\psi$:

"$\Leftarrow$": Let $[m]\psi \in \Gamma$. Then $\psi \in \Gamma/m$. Therefore, for each Δ with $(w_\Gamma, w_\Delta) \in R_m$, we have $\psi \in \Delta$. Induction yields $\mathcal{K}^{K_n}, w_\Delta \models \psi$ and thus $\mathcal{K}^{K_n}, w_\Gamma \models [m]\psi$.

"$\Rightarrow$": Let $\mathcal{K}^{K_n}, w_\Gamma \models [m]\psi$. We have to show that $[m]\psi \in \Gamma$.

Claim: Then $\Gamma/m \cup \{\neg\psi\}$ is inconsistent.

Proof: Assume that $\Gamma/m \cup \{\neg\psi\}$ is consistent. Then there is a maximal consistent set Δ with $\Gamma/m \cup \{\neg\psi\} \subseteq \Delta$. Now $\Gamma/m \subseteq \Delta$ yields $(w_\Gamma, w_\Delta) \in R_m$. Since $\neg\psi \in \Delta$, we get $\psi \notin \Delta$ by Lemma 7.13 2a). Induction yields $\mathcal{K}^{K_n}, w_\Delta \not\models \psi$. Together with $(w_\Gamma, w_\Delta) \in R_m$, we get $\mathcal{K}^{K_n}, w_\Gamma \not\models [m]\psi$. $\lightning$

Since $\Gamma/m \cup \{\neg\psi\}$ is inconsistent, there are $\varphi_1, \ldots, \varphi_k \in \Gamma/m$ such that

$$\vdash_{K_n} \neg(\varphi_1 \wedge \cdots \wedge \varphi_k \wedge \neg\psi) \qquad (+)$$

Note: We can assume w.l.o.g. that $\neg\psi$ appears here since $\vdash_{K_n} \neg\vartheta$ implies $\vdash_{K_n} \neg(\vartheta \wedge \sigma)$ since $\neg a \to \neg(a \wedge b)$ is a propositional tautology.

Using a suitable tautology, we derive from (+):

$$\vdash_{K_n} \varphi_1 \to (\varphi_2 \to \ldots (\varphi_k \to \psi) \ldots)$$

(GE) yields:

$$\vdash_{K_n} [m](\varphi_1 \to (\varphi_2 \to \ldots (\varphi_k \to \psi) \ldots)) \qquad (++)$$

Lemma 7.10 gives us:

$$\vdash_{K_n} \quad [m]\big(\varphi_1 \to (\varphi_2 \to \dots (\varphi_k \to \psi) \dots)\big)$$
$$\to \big([m]\varphi_1 \to ([m]\varphi_2 \to \dots ([m]\varphi_k \to [m]\psi) \dots)\big)$$

And thus (MP) yields:

$$\vdash_{K_n} [m]\varphi_1 \to ([m]\varphi_2 \to \dots ([m]\varphi_k \to [m]\psi) \dots)$$

Again using a suitable tautology, we get:

$$\vdash_{K_n} \neg\big([m]\varphi_1 \wedge \dots \wedge [m]\varphi_k \wedge \neg[m]\psi\big)$$

Thus the formula $[m]\varphi_1 \wedge \dots \wedge [m]\varphi_k \wedge \neg[m]\psi$ is inconsistent.
Since $\varphi_1, \dots, \varphi_k \in \Gamma/m$, $[m]\varphi_1, \dots, [m]\varphi_k \in \Gamma$. Since Γ is consistent, it follows that $\neg[m]\psi \notin \Gamma$, i.e. $[m]\psi \in \Gamma$ as required.

□

Corollary 7.15. If φ is consistent, then there is a world w_Γ in $\mathcal{K}^{K_n}$ such that $\mathcal{K}^{K_n}, w_\Gamma \models \varphi$.

Proof. The set $\{\varphi\}$ can be extended to a maximally consistent set Γ. Since $\varphi \in \Gamma$, Theorem 7.14 yields $\mathcal{K}^{K_n}, w_\Gamma \models \varphi$. □

7.5 Axioms for knowledge

In this section, $[m]\varphi$ should be read as "agent m *knows* φ". (The names of the axioms have historical reasons.)

Axiom T. "An agent does not have wrong knowledge."

This axiom is intended to distinguish "belief" from "knowledge". It is possible to *believe* things that are not true, but one cannot *know* them.

(T) $[m]\varphi \to \varphi$

Positive Introspection (Axiom 4) "If an agent knows something, then he knows that he knows it."

(4) $[m]\varphi \to [m][m]\varphi$

Negative Introspection (Axiom 5) "If an agent does not know something, then he knows that he does not know it."

(5) $\neg[m]\varphi \to [m]\neg[m]\varphi$

Whereas (T) and (4) are generally accepted as properties of knowledge, (5) is disputed. Note that (5) implies that an agent can assess his own competence, e.g. he knows when his knowledge is not sufficient. Obviously, this is not always a reasonable assumption.

By combining these axioms, we obtain different modal logics of knowledge and belief, all extending K_n.

T_n: K_n + (T);

i.e. axiom schemas: (Taut), (K), (T);

rules of inference: (MP), (GE).

We write $\vdash_T \varphi$ if φ is derivable in this system

$S4_n$: T_n + (4);

i.e. axiom schemas: (Taut), (K), (T), (4);

rules of inference: (MP), (GE).

$S5_n$: $S4_n$ + (5);

i.e. axiom schemas: (Taut), (K), (T), (4), (5);

rules of inference: (MP), (GE).

Usually, $S4_n$ and $S5_n$ are used as *logics of knowledge*. By dropping (T), we obtain a *logic of belief*. To prevent that agents believe obvious nonsense, one usually adds axiom (D):

(D) $\neg[m]\texttt{false}$;

i.e. an agent does not believe obviously contradictory formulas.

$KD45_n$: K_n + (D) + (4) + (5);

i.e. axiom schemas: (Taut), (K), (D), (4), (5);

rules of inference: (MP), (GE).

How can we decide which formulas are theorems of these logics and which are not?

- Obviously, all derivable formulas (i.e. all theorems) can be recursively enumerated (exercise);
- but how can we check that a formula will *never* occur in this enumeration, i.e. that it is *not* derivable?

For K_n, the decidability of the problem "$\vdash_{K_n} \varphi$?" is a consequence of the soundness and completeness of the axiomatization together with the decidability of "$\models_{K_n} \varphi$?", e.g. by the $\mathcal{ALCN}$ tableau algorithm.

Therefore, we proceed in two steps:

1. Search for a suitable *semantic characterization* of the logics in question;

2. use these to modify the tableau algorithm in an appropriate way.

It turns out that the additional axioms correspond to certain properties of the accessibility relations.

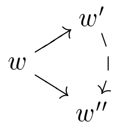
The following properties occur:

reflexivity: $(w, w) \in R_m$ for all $w \in \mathcal{W}$;

transitivity: $(w, w') \in R_m$ and $(w', w'') \in R_m$ implies $(w, w'') \in R_m$;

symmetry: $(w, w') \in R_m$ implies $(w', w) \in R_m$;

euclidianness: $(w, w') \in R_m$ and $(w, w'') \in R_m$ implies $(w', w'') \in R_m$;



seriality: $\forall w \in \mathcal{W}. \exists w' \in \mathcal{W}. (w, w') \in R_m$.

Lemma 7.16.

1. If R_m is symmetric and transitive, then it is Euclidian.

2. The following conditions are equivalent:

   (a) R_m is symmetric, transitive, and serial;

   (b) R_m is reflexive and Euclidian;

   (c) R_m is an equivalence relation, i.e. reflexive, symmetric, and transitive.

Proof. Exercise. □

Theorem 7.17. Let $\mathcal{K}$ be a Kripke structure and φ a formula.

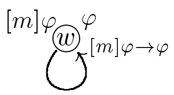
1. If R_m is reflexive, then (T):$[m]\varphi \to \varphi$ holds in $\mathcal{K}$ (i.e. $\mathcal{K}, w \models [m]\varphi \to \varphi$ for all worlds w of $\mathcal{K}$).

2. If R_m is transitive, then (4):$[m]\varphi \to [m][m]\varphi$ holds in $\mathcal{K}$.

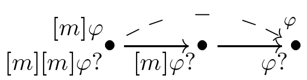
3. If R_m is Euclidean, then (5):$\neg[m]\varphi \to [m]\neg[m]\varphi$ holds in $\mathcal{K}$.

4. If R_m is serial, then (D):$\neg[m]$`false` holds in $\mathcal{K}$.

Proof.

1. 

Since R_m is reflexive, $[m]\varphi$ implies φ.

2. 

   Since R_m is transitive, every indirect successor is also a direct successor, thus $[m]\varphi$ implies $[m][m]\varphi$.

3. $$\begin{tikzcd} & v^{\neg[m]\varphi} \arrow[d, dashed] \\ {\neg[m]\varphi \atop [m]\neg[m]\varphi?}\; w \arrow[ur, bend left] \arrow[r] & u_{\neg\varphi} \end{tikzcd}$$

   If $\neg[m]\varphi$ holds at node w, there exists a successor u of w in which $\neg\varphi$ holds. Since R_m is Euclidian, u is also a successor of every successor v of w, thus $\neg[m]\varphi$ holds at every v, and thus $[m]\neg[m]\varphi$ at w.

4. $$\neg[m]\texttt{false}\ \bullet \longrightarrow \bullet\ \neg\texttt{false}$$

   Every node has a successor (in which trivially $\neg$`false` holds), thus $[m]$`false` holds nowhere.

$\square$

Conversely, by adding the axiom schemas to K_n, we can derive all formulas that hold in Kripke structures with the corresponding relational property.

Theorem 7.18. Let φ be a formula.

1. If φ holds in all Kripke structures with *reflexive* accessibility relations, then φ can be derived in K_n + (T) = T_n.
2. If φ holds in all Kripke structures with *transitive* accessibility relations, then φ can be derived in K_n + (4) = $K4_n$.
3. If φ holds in all Kripke structures with *Euclidean* accessibility relations, then φ can be derived in K_n + (5).
4. If φ holds in all Kripke structures with *serial* accessibility relations, then φ can be derived in K_n + (D).

Proof.

1. As in the proof of Theorem 7.11, we show that each T_n-consistent formula φ is satisfiable in the T_n-canonical Kripke structure. Here, the term T_n-*consistent* is defined analogous to K_n-consistency, we only use $\vdash_{T_n}$ instead of $\vdash_{K_n}$. The statements of Lemma 7.13 also hold for T_n-consistency (since all K_n axioms are still present in T_n). The *T_n-canonical Kripke structure* $\mathcal{K}^{T_n}$ is based on maximally T_n-consistent sets instead of maximally K_n-consistent sets.

   As in Theorem 7.14, we can show that

   $$\mathcal{K}^{T_n}, w_\Gamma \models \varphi \text{ iff } \varphi \in \Gamma.$$

   It thus suffices to show the following:

   Claim: The accessibility relations of $\mathcal{K}^{T_n}$ are reflexive.

Proof:

- For $w_\Gamma, w_\Delta \in \mathcal{W}^{T_n}$, we have $(w_\Gamma, w_\Delta) \in R_m$ iff $\Gamma/m = \{\varphi \mid [m]\varphi \in \Gamma\} \subseteq \Delta$.
- Γ contains $[m]\varphi \to \varphi$ for all φ, since otherwise $\neg([m]\varphi \to \varphi) \in \Gamma$ in contradiction to the facts that Γ is consistent and $\vdash_{T_n} [m]\varphi \to \varphi$.
- Therefore, if Γ contains $[m]\varphi$, then it also contains φ (Lemma 7.13 2c).

Thus $\Gamma/m \subseteq \Gamma$, i.e. $(w_\Gamma, w_\Gamma) \in R_m$.

2. Consider $(K_n + 4)$-consistency and the corresponding canonical Kripke structure. We show that the relations in this structure are transitive:

   Let $(w_{\Gamma_1}, w_{\Gamma_2}) \in R_m$ and $(w_{\Gamma_2}, w_{\Gamma_3}) \in R_m$, i.e. $\Gamma_1/m \subseteq \Gamma_2$ and $\Gamma_2/m \subseteq \Gamma_3$.

   - Γ_1 contains all instances of (4), i.e.

   $$[m]\varphi \to [m][m]\varphi \in \Gamma_1 \text{ for all } \varphi.$$

   - We want to show: $\Gamma_1/m \subseteq \Gamma_3$.

   Let $\varphi \in \Gamma_1/m$, i.e. $[m]\varphi \in \Gamma_1$.

   Therefore, $[m][m]\varphi \in \Gamma_1$ by (4) and (MP).

   Thus $[m]\varphi \in \Gamma_1/m \subseteq \Gamma_2$, and thus $\varphi \in \Gamma_2/m \subseteq \Gamma_3$.

3. Let $(w_\Gamma, w_{\Gamma_1}) \in R_m$ and $(w_\Gamma, w_{\Gamma_2}) \in R_m$. We have to show: $(w_{\Gamma_1}, w_{\Gamma_2}) \in R_m$.

   - We have: (i) $\Gamma/m \subseteq \Gamma_1$ and (ii) $\Gamma/m \subseteq \Gamma_2$.
   - We show: $\Gamma_1/m \subseteq \Gamma_2$.

   Let $\varphi \in \Gamma_1/m$, i.e. $[m]\varphi \in \Gamma_1$.

   Assume $\varphi \notin \Gamma_2$. Then, by (ii), $[m]\varphi \notin \Gamma$.

   Then $\neg[m]\varphi \in \Gamma$.

   Since all instances of (5) are in Γ, this implies that $[m]\neg[m]\varphi \in \Gamma$, and thus by (i), $\neg[m]\varphi \in \Gamma_1$. This is a contradiction to $[m]\varphi \in \Gamma_1$.

4. For a fixed w_Γ, we show that $(w_\Gamma, w_\Delta) \in R_m$ for some Δ.

   We know that $\neg[m]\texttt{false} \in \Gamma$.

   We now consider Γ/m and show that it is consistent:

Assume that Γ/m is inconsistent. Then there are formulas $\varphi_1, \ldots, \varphi_k \in \Gamma/m$ with

$$\begin{array}{rl}
 & \vdash_{K_n + (D)} \neg(\varphi_1 \wedge \cdots \wedge \varphi_k) \\
\overset{\text{tautology: } \neg a \to (a \to \texttt{false})}{\Rightarrow} & \vdash_{K_n + (D)} \varphi_1 \wedge \cdots \wedge \varphi_k \to \texttt{false} \\
\overset{\text{tautology}}{\Rightarrow} & \vdash_{K_n + (D)} \varphi_1 \to (\varphi_2 \to \ldots (\varphi_k \to \texttt{false}) \ldots) \\
\overset{\text{as in the proof of 7.11}}{\Rightarrow} & \vdash_{K_n + (D)} [m]\varphi_1 \to \ldots ([m]\varphi_k \to [m]\texttt{false}) \ldots
\end{array}$$

Thus, the set $\Theta = \{[m]\varphi_1, \ldots, [m]\varphi_k, \neg[m]\texttt{false}\}$ is inconsistent. Since $\Theta \subseteq \Gamma$ (because $[m]\varphi_i \in \Gamma$ since $\varphi_i \in \Gamma/m$), this is a contradiction to the consistency of Γ.

Thus: Γ/m is consistent. Therefore, there is a maximally consistent set Δ with $\Delta \supseteq \Gamma/m$, i.e. we have a pair $(w_\Gamma, w_\Delta) \in R_m$.

□

Corollary 7.19. Let φ be a formula.

1. The formula φ holds in all Kripke structures with reflexive and transitive accessibility relations iff φ can be derived in $S4_n$.
2. The formula φ holds in all Kripke structures whose accessibility relations are equivalence relations iff φ is derivable in $S5_n$.
3. The formula φ holds in all Kripke structures with Euclidean, transitive, and serial accessibility relations iff φ is derivable in $KD45_n$.

Proof.

"$\Leftarrow$": This follows immediately from Theorem 7.17 and Lemma 7.16.

"$\Rightarrow$": This can be shown as in Theorem 7.18, i.e. by defining the appropriate notion of consistency and showing that the corresponding canonical Kripke structure satisfies the required properties.

□

7.6 Tableau Algorithms for T_n and $S4_n$

The tableau algorithm for *satisfiability in* K_n is identical to the one for $\mathcal{ALC}$. In order to show satisfiability of φ_0, we start with $\{\{x_0 : \varphi_0\}\}$, where x_0 is a world variable, φ_0 is a formula in NNF, and the set $\{x_0 : \varphi_0\}$ is called a *system* (instead of ABox). We then apply rules until no rule is applicable.

In the following, we omit the obvious non-applicability conditions.

$\wedge$-rule: $\{\ldots, x : \varphi_1 \wedge \varphi_2, \ldots\} \Rightarrow \{\ldots, x : \varphi_1 \wedge \varphi_2, x : \varphi_1, x : \varphi_2, \ldots\}$

$\vee$-rule: $\{\ldots, x : \varphi_1 \vee \varphi_2, \ldots\} \Rightarrow \left\{ \begin{array}{l} \{\ldots, x : \varphi_1 \vee \varphi_2, x : \varphi_1, \ldots\} \\ \{\ldots, x : \varphi_1 \vee \varphi_2, x : \varphi_2, \ldots\} \end{array} \right\}$

$\langle\ \rangle$-rule: $\{\dots, x : \langle m\rangle\varphi, \dots\} \Rightarrow \{\dots, x : \langle m\rangle\varphi, y : \varphi, (x,y) : R_m, \dots\}$, with a new world variable y.

[ ]-rule: $\{\dots, x : [m]\varphi, (x,y) : R_m, \dots\} \Rightarrow \{\dots, x : [m]\varphi, (x,y) : R_m, y : \varphi, \dots\}$

Obvious contradiction (closed system): presence of $x : a, x : \neg a$ for some propositional letter a.

We have shown (for the more general $\mathcal{ALCN}$-algorithm):

1. Soundness: the rules preserve (in)consistency, and closed systems are inconsistent.
2. Completeness: a complete and open system has a canonical model.
3. Termination.

A Tableau Algorithm for T_n. We consider only structures with reflexive accessibility relations: each system *implicitly* contains $(x,x) : R_m$ for every world variable x and every $m \in M_n$. Instead of adding these edges explicitly, we realize their effect (interaction with the [ ]-rule) through a new rule:

T-rule: $\{\dots, x : [m]\varphi, \dots\} \Rightarrow \{\dots, x : [m]\varphi, x : \varphi, \dots\}$ if $x : \varphi$ is not already contained in the system.

We have to show:

1. Soundness: Let $\mathcal{K}$ be a Kripke structure with reflexive accessibility relations *satisfying* a system Γ, i.e. there is a mapping $x \mapsto w_x$ of world variables in Γ to worlds of $\mathcal{K}$ such that

   - $x : \varphi \in \Gamma \Rightarrow \mathcal{K}, w_x \models \varphi$ and
   - $(x,y) : R_m \in \Gamma \Rightarrow (w_x, w_y) \in R_m$.

   Let $x : [m]\varphi \in \Gamma$. Then $\mathcal{K}, w_x \models [m]\varphi$. Since R_m is reflexive, the semantics yields $\mathcal{K}, w_x \models \varphi$. Therefore, $\mathcal{K}$ also satisfies $\Gamma \cup \{x : \varphi\}$.

   The other direction is trivial since Γ is strictly extended.

2. Completeness for T_n: We define, for an open and complete system Γ, the canonical model as for $\mathcal{ALC}$, with only one difference:

   - we additionally put $(x,x) \in R_m$ for all x and all m.

   In the proof that the canonical model satisfies all assertions in Γ, this new definition needs only be taken into account for assertions of the form $x : [m]\varphi$: We (additionally) have to show that $\mathcal{K}, x \models \varphi$ in the canonical model $\mathcal{K}$. But since the T-rule is not applicable, we know that $x : \varphi \in \Gamma$ and thus $\mathcal{K}, x \models \varphi$ by induction.

3. Termination for T_n: Since we start with $\{\{x_0 : \varphi\}\}$, there is only Phase 2. The termination argument works exactly as for Phase 2 of the $\mathcal{ALC}$-algorithm.

   Note:

   - Like the other rules, the T-rule generates only assertions of the kind $x : \varphi$ where φ is a subformula of φ_0.
   - It is applicable at most once for each world variable and each subformula $[m]\varphi$ of φ_0.
   - The maximum size of formulas decreases when going to successor worlds.

A Tableau Algorithm for $S4_n$. Here, each system containing $(x, y) : R_m$ and $(y, z) : R_m$ contains $(x, z) \in R_m$ *implicitly.* Again, this *additional link* only concerns the [ ]-rule:

if $\{x : [m]\varphi, (x, y) : R_m, (y, z) : R_m\} \subseteq \Gamma$, then we also need $z : \varphi$ to be in Γ.

We achieve this by adding a new rule (to the T_n rules):

4-rule: $\{\ldots, x : [m]\varphi, (x, y) : R_m, \ldots\}$
$\Rightarrow \{\ldots, x : [m]\varphi, (x, y) : R_m, y : [m]\varphi \ldots\}$
if $y : [m]\varphi$ is not already present.

1. Soundness of the 4-rule:

   Let $\mathcal{K}$ be a Kripke structure with transitive accessibility relations satisfying a system Γ. Let $x : [m]\varphi$, $(x, y) : R_m \in \Gamma$. Then $\mathcal{K}, w_x \models [m]\varphi$ and $(w_x, w_y) \in R_m$.

   We show: $\mathcal{K}, w_y \models [m]\varphi$.

   To this end, we take a world w with $(w_y, w) \in R_m$. Since R_m is transitive, we have $(w_x, w) \in R_m$ and thus $\mathcal{K}, w_x \models [m]\varphi$ yields $\mathcal{K}, w \models \varphi$.

   Therefore, $\mathcal{K}, w_y \models [m]\varphi$.

2. Completeness for $S4_n$:

   We construct the canonical model as for T_n, but with the accessibility relations defined as follows:

   $$\begin{array}{rcl} R_m & := & \{(x,x) \mid x \text{ world variable in } \Gamma\} \cup \\ & & \{(x_0, x_n) \mid (x_0, x_1) : R_m, \ldots, (x_{n-1}, x_n) : R_m \in \Gamma \text{ for } n \geq 1\} \end{array}$$

   Again this only affects the $x : [m]\varphi$ case:

   Let $\{(x_0, x_1) : R_m, \ldots, (x_{n-1}, x_n) : R_m, x_0 : [m]\varphi\} \subseteq \Gamma$.

   We show: $\mathcal{K}, x_n \models \varphi$.

Since the 4-rule is not applicable, we obtain

$$x_1 : [m]\varphi, x_2 : [m]\varphi, \ldots, x_{n-1} : [m]\varphi \in \Gamma,$$

and due to $(x_{n-1}, x_n) : R_m \in \Gamma$ and non-applicability of the [ ]-rule, we get $x_n : \varphi \in \Gamma$, and induction yields $\mathcal{K}, x_n \models \varphi$.

3. Termination for $S4_n$:

   Does *not* hold: e.g. $\varphi_0 = [m]\langle m \rangle a$

$$\begin{array}{c}
x_0 : [m]\langle m\rangle a, \underbrace{\langle m\rangle a}_{\text{T-rule}} \\
\Big\downarrow \langle\ \rangle\text{-rule} \\
x_1 : a, \underbrace{\langle m\rangle a}_{[\ ]\text{-rule}}, \underbrace{[m]\langle m\rangle a}_{\text{4-rule}} \\
\Big\downarrow \\
x_2 : a, \langle m\rangle a, [m]\langle m\rangle a \\
\Big\downarrow \\
\ldots
\end{array}$$

Solution:

(a) apply $\langle\ \rangle$-rule only if none of the other rules is applicable;

(b) the $\langle\ \rangle$-rule is *not* applied if there already exists an ancestor world that has all the assertions that the new world would receive.

In the example: The world x_2 would not be generated since x_1 already has all assertions that x_2 would receive.

Formally: Consider $x : \langle m\rangle\varphi \in \Gamma$ to which the $\langle\ \rangle$-rule would be applicable.

- The node x' is *m-ancestor* of x if there are $x_0, \ldots, x_n$ ($n \geq 0$) such that $x' = x_0$, $\{(x_0, x_1) : R_m, \ldots, (x_{n-1}, x_n) : R_m\} \subseteq \Gamma$, and $x_n = x$. (Note: x is an ancestor of itself.)
- Define $\Gamma_{x'} := \{\psi \mid x' : \psi \in \Gamma\}$. (These are the formulas in the label of x'.)
- Define $\Gamma_{x,\langle m\rangle\varphi} := \{\varphi\} \cup \{\psi \mid x : [m]\psi \in \Gamma\} \cup \{[m]\psi \mid x : [m]\psi \in \Gamma\}$. (These are the formulas that *an m-successor* of x' would be labeled with.)

The new $\langle\ \rangle_4$-rule is *not* applicable to $x : \langle m\rangle\varphi$ if there is an m-ancestor x' of x with $\Gamma_{x,\langle m\rangle\varphi} \subseteq \Gamma_{x'}$.

We say: x' *blocks* x *for* $\langle m\rangle\varphi$.

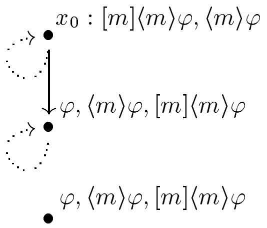
Completeness. We modify the definition of the canonical model. Define the accessibility relation as follows:

$$\begin{aligned} R_m \quad := \quad & \big\{(x,x) \mid x \text{ world variable in } \Gamma\big\} \cup \\ & \big\{(x,x') \mid (x,x') : R_m \in \Gamma \text{ or } x' \text{ blocks } x \text{ for some } \langle m\rangle\varphi\big\}^*, \end{aligned}$$

where * is the transitive closure.

If x' blocks x for $\langle m\rangle\varphi$ then x' has all the assertions that a newly introduced successor for $x : \langle m\rangle\varphi$ would receive. Thus it is easy to show that this canonical model satisfies all assertions in Γ.

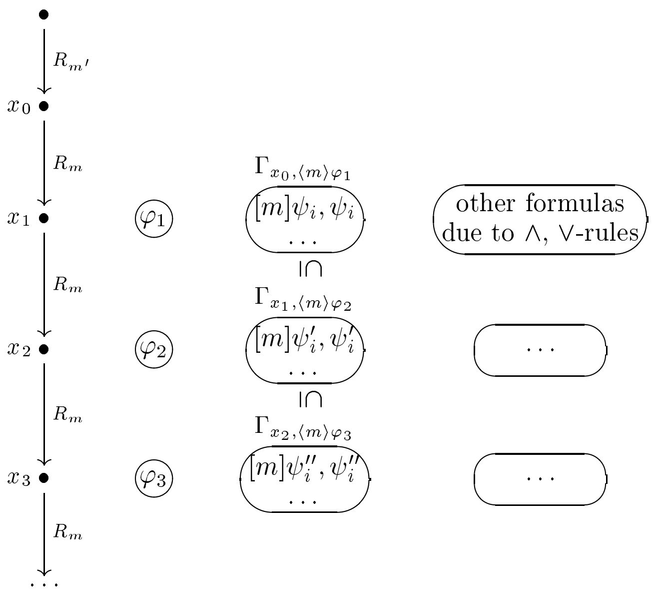
$$\varphi_0 = [m]\langle m\rangle\varphi$$



Termination. We also show that the algorithm needs only polynomial space. As for $\mathcal{ALC}$, a tree is generated:

$$\begin{tikzcd} & & x_0 \arrow[dl, "R_m"'] \arrow[dr, "R_{m'}"] & \\ & x_1 \arrow[dl] \arrow[dr] & & x_2 \\ x_3 & & x_4 & \end{tikzcd}$$

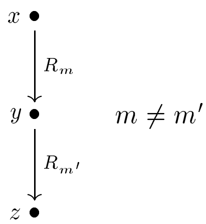
In this tree, paths of consecutive edges all labeled with the same modal parameter R_m cannot be longer than n^2, with n denoting the length of the input formula.



Each node label is a subset of the predecessor's label: the $[m]\psi_i$-formulas are pushed along the path by the 4-rule and then the ψ_i-formulas are generated by the T-rule. Since all occurring formulas are subformulas of the input formula, there are only n different φ_i and also only n different sets $\Gamma_{x_{i-1},\langle m\rangle\varphi}$ (due to the set inclusion).

Thus, if the path was longer than n^2, then there would be an $i < j$ with $\varphi_i = \varphi_j$ and $\Gamma_{x_{i-1},\langle m\rangle\varphi_i} = \Gamma_{x_{j-1},\langle m\rangle\varphi_j}$. At the time when x_j was generated, $\Gamma_{x_{j-1},\langle m\rangle\varphi_j}$ was already contained in Γ_{x_i} since the $\langle\ \rangle_4$-rule is only applied if no other rule is applicable. Therefore, x_{j-1} was blocked by x_i for $\langle m\rangle\varphi_j$, which is a contradiction to the fact that x_j was generated.

When changing the modal parameter



as in the case of $\mathcal{ALC}$, the maximum length of formulas decreases (e.g. when going from x to z in the above example). Summing up, the depth of our trees is bounded by n^3. As in the case of $\mathcal{ALC}$, we can then show that the algorithm can be executed in polynomial space (by performing a depth-first search keeping only branches of the tree in memory). The proof of PSPACE-hardness is also similar to the proof for $\mathcal{ALC}$.

Theorem 7.20. For T_n, $S4_n$, and $KD45_n$, satisfiability is PSPACE-complete. For $S5_n$, this depends on n:

If $n = 1$**:** satisfiability is NP-complete.

If $n > 1$**:** satisfiability is PSPACE-complete.

Proof sketch.

T_n**:** This is very similar to $\mathcal{ALC}/K_n$.

$S4_n$**:** We have proved the upper bound explicitly. The lower bound can be proved by a reduction of the validity problem of QBFs.

$KD45_n$**:** This is similar to $S4_n$.

$S5_n$**:** For $n = 1$: The lower bound follows directly from the NP-hardness of the satisfiability problem for propositional logic. To prove the upper bound, we show that every satisfiable formula is satisfiable in a model of polynomial size. Then a (non-deterministic) TM "guesses" a model.

For $n > 1$: The lower bound can be proved by a reduction from T_1 to $S5_2$: the relation $(R_m \circ R_{m'})$ of $S5_2$ behaves as a single relation of T_1. The upper bound can be shown via a =tableau algorithm.

$\square$

8 Default-rules

In semantic networks and frames, there are so-called "defaults" which are only applied if they do not contradict existing knowledge. However, in these early approaches, defaults lack a proper semantics.

Example. Rules of the kind:
"Frogs are usually green."
"Birds are usually able to fly."

If we encounter a grass frog (or penguin) that is not green (or able to fly), we should not obtain a contradiction.

Idea. In addition to positive applicability conditions (preconditions, e.g. `frog`, `bird`), we also have negative applicability conditions (justifications, e.g. ¬`grass frog` and ¬`penguin`), which must not be in contradiction to the existing knowledge in order for the rule to be applicable.

This idea was formalized by Reiter (1980) as follows:

Definition 8.1. A *default rule* consists of a *precondition* α, a *justification* β, and a *postcondition* γ (where α, β, γ are first order formulas).

Such a rule is written as

$$\frac{\alpha : \beta}{\gamma}.$$

A default rule is called *closed* if α, β, and γ are closed (i.e. contain no free variables). Otherwise, the rule is called *open*.

Intuitive meaning: The default rule $\frac{\alpha:\beta}{\gamma}$ is applicable if

- α follows from the current knowledge and
- β is consistent with the current knowledge.

The application of the rule adds the postcondition γ to the current knowledge.

Definition 8.2. A *default theory* is a pair (D, W) where

- D is a set of default rules and
- W is a set of closed FO-formulas (the world description).

It is called *closed* if all members of D are closed.

As a special case, we will consider default theories in which all formulas stem from description logic concepts and assertions.

Definition 8.3. Let $\mathcal{L}$ be a description logics (e.g. $\mathcal{ALC}$). The pair (D, W) is an $\mathcal{L}$*-default theory* if

- the elements of D are of the form

$$\frac{C_1(x) : C_2(x)}{C_3(x)},$$

where C_1, C_2 and C_3 are $\mathcal{L}$-concepts, which are regarded as formulas with one free variable;

- $W = \mathcal{T} \cup \mathcal{A}$, where $\mathcal{T}$ is an $\mathcal{L}$-TBox and $\mathcal{A}$ an $\mathcal{L}$-ABox.

Note: $\mathcal{L}$-defaults are open defaults.

Example 8.4. $W = \mathcal{T}_0 \cup \mathcal{A}_0$

$$\begin{array}{rcl}
\mathcal{T}_0 & : & \texttt{Penguin} \doteq \texttt{Bird} \sqcap \forall \texttt{lives-in.Antarctica} \sqcap \texttt{SpecialPenguinProperty} \\
& & \texttt{Flies} \doteq \exists \texttt{motion.Fly} \\
\mathcal{A}_0 & : & \texttt{Penguin}(\texttt{Tweety}) \\
D_0 & : & \dfrac{\texttt{Bird}(x) : \texttt{Flies}(x)}{\texttt{Flies}(x)} \\
& & \dfrac{\texttt{Penguin}(x) : \neg\texttt{Flies}(x)}{\neg\texttt{Flies}(x)}
\end{array}$$

Does `Tweety` fly?

8.1 Semantics for closed defaults

Reiter (1980) starts with giving a semantics for *closed* default theories. This is non-trivial since the result of rule application depends on the order in which rules are applied.

Example 8.4 (cont'd). In W, both defaults from D_0 are applicable to Tweety. If we apply the first one, $\mathcal{A}_0$ is extended to $\mathcal{A}_1 = \mathcal{A}_0 \cup \{\texttt{Flies}(\texttt{Tweety})\}$. Now, the second rule is not applicable any more. If we start with applying the second rule, we get $\mathcal{A}_1' = \mathcal{A}_0 \cup \{\neg\texttt{Flies}(\texttt{Tweety})\}$. We should not try to apply both rules simultaneously since then we get an inconsistent extension.

Reiter's solution: default theories can have multiple extensions.

Another problem is that rules that have already been applied can be "made non-applicable" by later rule applications. Thus, we would have to retract consequences that have already been added.

Example 8.5.

$$\begin{array}{rcl}
D & = & \left\{ \dfrac{a : b}{c}, \dfrac{c : \neg b}{\neg b} \right\} \quad \text{where } a,\, b,\, c \text{ are closed atomic formulas} \\
W_0 & = & \{a\}
\end{array}$$

Initially, only the first default is applicable: $W_1 = \{a, c\}$. Now the second rule has become applicable: $W_2 = \{a, c, \neg b\}$. But then, $\neg b$ contradicts the justification for applying the first default.

Reiter's solution: "guess" the set of formulas that is the result of exhaustive rule application and check the justifications against that set.

In the following, let (D, W) be a *closed* default theory.

Definition 8.6. A set E of closed formulas is called *extension of* (D, W) if the following holds:

1. $E = \mathrm{Th}(E)$, i.e. E is closed under logical consequence.
   $(\mathrm{Th}(E) = \{\alpha \mid E \underbrace{\models}_{\text{in FOL}} \alpha\})$
2. $E = \bigcup_{i=0}^{\infty} \mathrm{Th}(E_i)$, where

$$\begin{aligned} E_0 &:= W, \\ E_{i+1} &:= E_i \cup \left\{ \gamma \mid \frac{\alpha : \beta}{\gamma} \in D, E_i \models \alpha, E \not\models \neg\beta \right\}. \end{aligned}$$

Note that E has an influence on the definition of E_i.

Theorem 8.7. If E is an extension of (D, W), then there is a subset Γ_E of $\mathrm{Con}(D) := \{\gamma \mid \frac{\alpha:\beta}{\gamma} \in D\}$ such that $E = \mathrm{Th}(W \cup \Gamma_E)$.

Proof. Obviously, there are sets $\Gamma_1 \subseteq \Gamma_2 \subseteq \Gamma_3 \subseteq \ldots$ of $\mathrm{Con}(D)$ with

- $E_i = W \cup \Gamma_i$,
- $E = \bigcup_{i=0}^{\infty} \mathrm{Th}(W \cup \Gamma_i)$.

The compactness theorem for predicate logic says that a formula follows from an infinite set Θ iff it follows from a finite subset of Θ. Thus, $\Gamma_1 \subseteq \Gamma_2 \subseteq \Gamma_3 \subseteq \ldots$ yields

$$\bigcup_{i=0}^{\infty} \mathrm{Th}(W \cup \Gamma_i) = \mathrm{Th}\left(W \cup \bigcup_{i=0}^{\infty} \Gamma_i \right).$$

Thus, we can choose $\Gamma_E := \bigcup_{i=0}^{\infty} \Gamma_i$. □

Corollary 8.8. If D is finite, then there are only finitely many extensions of (D, W) for any W.

Example 8.9. There are default theories with more than one extension:

$$\begin{aligned} D &= \left\{ \frac{b : f}{f}, \frac{p : \neg f}{\neg f} \right\} \\ W &= \{p \to b, p\} \end{aligned}$$

$E^{(1)} = \mathrm{Th}(W \cup \{f\})$ and $E^{(2)} = \mathrm{Th}(W \cup \{\neg f\})$ are extensions.

$$\begin{aligned} E_0^{(1)} &= W & \qquad E_0^{(2)} &= W \\ E_1^{(1)} &= E_0^{(1)} \cup \{f\} & \qquad E_1^{(2)} &= E_0^{(2)} \cup \{\neg f\} \\ E_2^{(1)} &= E_1^{(1)} & \qquad E_2^{(2)} &= E_1^{(2)} \end{aligned}$$

Exercise: Show that these are the only extensions.

Example 8.10. There are default theories that have no extension:

$$W = \{a\} \qquad\qquad D = \left\{ \frac{a : \neg b}{b} \right\}$$

There are only two potential extensions: $E^{(1)} = \mathrm{Th}(\{a\})$ and $E^{(2)} = \mathrm{Th}(\{a, b\})$.

$$\begin{array}{cc} E_0^{(1)} = \{a\} & E_0^{(2)} = \{a\} \\ E_1^{(1)} = E_0^{(1)} \cup \{b\} & E_1^{(2)} = E_0^{(2)} \\ \text{too large} & \text{too small} \end{array}$$

Exercise: Show that Example 8.5 has no extension.

Note: having no extension is *not* the same as being inconsistent, which means having no *model*.

Theorem 8.11. Let (D, W) be a closed default theory.

1. If W is inconsistent, then $\mathrm{Th}(W)$ is the only extension.
2. If W is consistent, then all extensions of (D, W), for any D, are consistent.

Proof.

1. Let W be inconsistent, i.e. $\mathrm{Th}(W)$ is the set of all formulas.

   (a) $E := \mathrm{Th}(W)$ is an extension:

   $$\begin{aligned} E_0 &:= W \\ E_1 &:= E_0 \cup \left\{ \gamma \mid \frac{\alpha : \beta}{\gamma} \in D, E_0 \models \alpha, \underbrace{E \not\models \neg\beta}_{\substack{\text{holds for no } \neg\beta \text{ since} \\ \text{all } \neg\beta \in E = \mathrm{Th}(W)}} \right\} = E_0 \end{aligned}$$

   Thus, $\bigcup_{i=1}^{\infty} \mathrm{Th}(E_i) = \mathrm{Th}(W)$.

   (b) $\mathrm{Th}(W)$ is the only extension since each extension contains W and $\mathrm{Th}(W)$ is already the set of all formulas.

2. Let $E = \mathrm{Th}(E)$ be an inconsistent extension of (D, W). Then no default rule is applicable (as in 1a above). Thus $E_1 = E_0 = W$. Since W is consistent, we have $\bigcup_{i=0}^{\infty} \mathrm{Th}(E_i) = \mathrm{Th}(W) \neq E$. Thus, E is not an extension.

□

Still, default theories (D, W) with a consistent world description W may have no extension (see Example 8.10). However, there is a natural subclass of default theories that always have at least one extension.

Definition 8.12. A default is called *normal* if it is of the form $\frac{\alpha:\beta}{\beta}$, i.e. if justification and consequence coincide. A default theory (D, W) is called *normal* if all defaults in D are normal.

Theorem 8.13. If (D, W) is a normal default theory, then it has at least one extension.

Proof. We assume w.l.o.g. that W is consistent. We construct an extension of (D, W) by using a greedy approach as follows (note that there can be other extensions):

$$F_0 := W.$$

Assume that F_i is already defined. We define

$$D_{i+1} := \left\{ \frac{\alpha : \beta}{\beta} \in D \mid F_i \models \alpha, F_i \not\models \neg\beta \right\}.$$

Let $\hat{D}_{i+1}$ be a maximal subset (note: there may be several possibilities) of D_{i+1} such that

$$F_i \cup \left\{ \beta \mid \frac{\alpha : \beta}{\beta} \in \hat{D}_{i+1} \right\}$$

is consistent. Then set

$$F_{i+1} := F_i \cup \left\{ \beta \mid \frac{\alpha : \beta}{\beta} \in \hat{D}_{i+1} \right\}.$$

Finally, define

$$E := \bigcup_{i \geq 0} \mathrm{Th}(F_i)$$

Since $F_0 \subseteq F_1 \subseteq \dots$ and the F_i are consistent by construction, E is consistent by the compactness theorem.

Claim: E is an extension.
Consider the sequence

$$E_0 := W;$$
$$E_{i+1} := E_i \cup \left\{ \beta \mid \frac{\alpha : \beta}{\beta} \in D, E_i \models \alpha, E \not\models \neg\beta \right\}.$$

We show:

$$\text{for all } i \geq 0, \quad E_i = F_i$$

by induction on i.

$i = 0$: This is obvious, since $E_0 = W = F_0$.

$i \to i+1$:

a) Proof that $E_{i+1} \subseteq F_{i+1}$:
Since $E_i \subseteq F_i \subseteq F_{i+1}$, it suffices to consider $\beta \in E_{i+1} \setminus E_i$.
Let $\frac{\alpha:\beta}{\beta} \in D$, $E_i \models \alpha$, $E \not\models \neg\beta$.

- $\frac{\alpha:\beta}{\beta} \in D_{i+1}$ since
  1) $E_i \models \alpha$ and $E_i = F_i$ by IH (thus $F_i \models \alpha$);
  2) $E \not\models \neg\beta$ and $F_i \subseteq E = \bigcup_{i \geq 0} \mathrm{Th}(F_i)$ (thus $F_i \not\models \neg\beta$).
- $\frac{\alpha:\beta}{\beta} \in \hat{D}_{i+1}$: due to maximality of $\hat{D}_{i+1}$, the contrary implies $F_i \cup \left\{\beta' \mid \frac{\alpha':\beta'}{\beta'} \in \hat{D}_{i+1}\right\} \cup \{\beta\}$ is inconsistent, i.e.

$$F_{i+1} = F_i \cup \left\{\beta' \mid \frac{\alpha' : \beta'}{\beta'} \in \hat{D}_{i+1}\right\} \models \neg\beta;$$

  in contradiction to $E = \bigcup \mathrm{Th}(F_i) \not\models \neg\beta$.

Thus, $\beta \in F_{i+1}$ as required.

b) Proof that $F_{i+1} \subseteq E_{i+1}$:
Again, it suffices to consider $\beta \in F_{i+1} \setminus F_i$. Then there is an $\frac{\alpha:\beta}{\beta} \in \hat{D}_{i+1}$ with $F_i \models \alpha$ and $F_i \not\models \neg\beta$ (remember $\hat{D}_{i+1} \subseteq D_{i+1} = \{\frac{\alpha':\beta'}{\beta'} \in D \mid F_i \models \alpha', F_i \not\models \neg\beta'\}$).

- $E_i \models \alpha$ since $E_i = F_i \models \alpha$.
- $E \not\models \neg\beta$ since $\beta \in F_{i+1} \subseteq E$ and E is consistent.

Thus $\beta \in E_{i+1}$.

□

Note: Normal default theories may have more than one extension. Which inference relation is induced by the notion of extensions? Intuitively, the extensions describe different possible situations.

Definition 8.14. Let (D, W) be a closed default theory and α a closed formula.

1. The formula α is a *skeptical consequence* of (D, W) if α appears in all extensions (α holds in all possible situations).

2. The formula α is a *credulous consequence* if α appears in at least one extension (α is possible).

8.2 Semantics of description logic default theories

For a description logic $\mathcal{L}$, $\mathcal{L}$-defaults are open by definition. We obtain a closed default theory by instantiating all the default rules with all the individuals occurring in the ABox.

Definition 8.15. Let (D, W) be an $\mathcal{L}$-default theory with $W = \mathcal{T} \cup \mathcal{A}$ for an $\mathcal{L}$-TBox $\mathcal{T}$ and an $\mathcal{L}$-ABox $\mathcal{A}$. We define

$$\hat{D} = \left\{ \frac{A(a) : B(a)}{C(a)} \mid \frac{A(x) : B(x)}{C(x)} \in D \text{ and } a \text{ is an individual name occurring in } \mathcal{A} \right\}.$$

The *extensions* of the open default theory (D, W) are the extensions of the corresponding closed default theory $(\hat{D}, W)$.

Note: by restricting the attention to individuals appearing *explicitly* in the ABox, we lose some default consequences.

Example 8.16.

$$\mathcal{T} : \texttt{Father-of-physician} = \texttt{Man} \sqcap \exists\texttt{has-child}.\texttt{Physician}$$
$$\mathcal{A} : \texttt{Tom} : \texttt{Father-of-physician}$$
$$D : \frac{\texttt{Physician}(x) : \texttt{Rich}(x)}{\texttt{Rich}(x)}$$

Instantiate:

$$D' : \frac{\texttt{Physician(Tom)} : \texttt{Rich(Tom)}}{\texttt{Rich(Tom)}}$$

Question: Is $\texttt{Tom} : \exists\texttt{has-child}.\texttt{Rich}$ a default consequence of (D, W)?

Answer: No, since `Tom`'s child does not appear explicitly (as a name) in the ABox.

We can avoid this problem by introducing a name for each implicit individual, in this way making it an explicit one. In the example, we would just introduce one new name (say, `Mary`) and extend $\mathcal{A}$ with the following assertions:

$$(\texttt{Tom}, \texttt{Mary}) : \texttt{has-child}$$
$$\texttt{Mary} : \texttt{Physician}$$

In principle, this is what Reiter proposed to deal with open defaults. The following example shows that, using this approach, logically equivalent world descriptions can lead to different default consequences.

Example 8.17.

$$D := \left\{ \frac{A(x) : \neg B(x)}{\neg B(x)} \right\}$$
$$\mathcal{T} := \emptyset$$
$$\mathcal{A}_1 := \{a : \exists r.(A \sqcap B)\} \qquad \mathcal{A}_2 := \{a : \exists r.(A \sqcap B), a : \exists r.A\}$$

Note: $\mathcal{A}_1$ and $\mathcal{A}_2$ are logically equivalent.

Introducing explicit individuals yields:

$\hat{\mathcal{A}}_1 = \mathcal{A} \cup \{(a,b) : r, b : A \sqcap B\}$

$$\hat{D}_1 = \left\{ \frac{A(a) : \neg B(a)}{\neg B(a)}, \frac{A(b) : \neg B(b)}{\neg B(b)} \right\}$$

No default is applicable.

$\hat{\mathcal{A}}_2 = \mathcal{A} \cup \{(a,b) : r, b : A \sqcap B, (a,c) : r, c : A\}$

$$\hat{D}_2 = \left\{ \frac{A(a) : \neg B(a)}{\neg B(a)}, \frac{A(b) : \neg B(b)}{\neg B(b)}, \frac{A(c) : \neg B(c)}{\neg B(c)} \right\}$$

The third default yields $\neg B(c)$.

Now, $a : \exists r.\neg B$

is not a default consequence. *is* a default consequence.

To avoid such problems, we only apply default rules to explicit individuals, without introducing implicit ones (Definition 8.15).

8.3 Inference procedures

Let (D, W) be a description logic default theory with finite sets D and W. Then, the corresponding closed default theory $(\hat{D}, W)$ is also finite. Thus, we are only interested in reasoning with finite, closed default theories.

By Corollary 8.8, there are only finitely many extensions of $(\hat{D}, W)$. By Theorem 8.7, these are of the form $\mathrm{Th}(W \cup \Gamma)$ with $\Gamma \subseteq \mathrm{Con}(\hat{D})$. If the instance problem of the underlying description logic $\mathcal{L}$ is decidable, then we can effectively check, for each extension E and assertion $a : C$, whether $W \cup \Gamma \models a : C$ holds.

Therefore, if *all* the extensions E are given in the form $\mathrm{Th}(W \cup \Gamma)$, then we can compute the skeptical consequences (for all $E = \mathrm{Th}(W \cup \Gamma)$, we have $W \cup \Gamma \models a : C$) and also the credulous consequences (there is an $E = \mathrm{Th}(W \cup \Gamma)$ with $W \cup \Gamma \models a : C$).

But how do we compute all extensions?

First Solution. By Theorem 8.7, each extension E of $(\hat{D}, W)$ is of the form $\mathrm{Th}(W \cup \Gamma_E)$ for some subset Γ_E of $\mathrm{Con}(\hat{D}) = \{\gamma \mid \frac{\alpha:\beta}{\gamma} \in \hat{D}\}$. Since $\hat{D}$ is finite, there are only finitely many subsets $\Gamma \subseteq \mathrm{Con}(\hat{D})$. Let $\Gamma \subseteq \mathrm{Con}(\hat{D})$. To check whether $E := \mathrm{Th}(W \cup \Gamma)$ is an extension, we put Definition 8.6 to work:

$$E_0 := W$$

$$E_{i+1} := E_i \cup \left\{ \gamma \mid \frac{\alpha : \beta}{\gamma} \in \hat{D}, \underbrace{E_i \models \alpha, W \cup \Gamma \not\models \neg\beta}_{\text{decidable}} \right\}$$

Each single step is effective and, since $\hat{D}$ is finite, the iteration becomes stable after finitely many steps, i.e. there is an n with $E_{n+1} = E_n$. It remains to check whether $\mathrm{Th}(E_n) = \mathrm{Th}(W \cup \Gamma)$, which can also be done effectively.

Theorem 8.21. Let (D, W) be a finite $\mathcal{L}$-default theory for a DL $\mathcal{L}$ with a decidable instance problem. Then both credulous and skeptical consequence are decidable.

Problem: This approach to computing extensions is very costly, namely best-case exponential (in the number of closed defaults, which is the product of the number of defaults and the number of individual names).

Second Solution. Characterize the subsets of $\hat{D}$ that generate extensions.

Definition 8.22. $D' \subseteq \hat{D}$ is a *set of generating defaults* for an extension E iff

$$D' = \left\{ \frac{\alpha : \beta}{\gamma} \mid E \models \alpha \text{ and } E \not\models \neg\beta \right\}.$$

Lemma 8.23. If $D' \subseteq \hat{D}$ is a set of generating defaults for the extension E, then $E = \mathrm{Th}(W \cup \mathrm{Con}(D'))$.

Proof. Let the sets E_i be defined as usual.

"$\subseteq$": We show $E_i \subseteq \mathrm{Th}(W \cup \mathrm{Con}(D'))$ for all $i \geq 0$. This implies $E \subseteq \mathrm{Th}(W \cup \mathrm{Con}(D'))$.

$i = 0$**:** $E_0 = W \subseteq \mathrm{Th}(W \cup \mathrm{Con}(D'))$.

$i \to i + 1$**:** Let $\gamma \in E_{i+1} \setminus E_i$, i.e. $\frac{\alpha:\beta}{\gamma} \in \hat{D}$, $E_i \models \alpha$ and $E \not\models \neg\beta$. Then $E \models \alpha$ and $E \not\models \neg\beta$, thus $\frac{\alpha:\beta}{\gamma} \in D'$. Hence, $\gamma \in \mathrm{Con}(D')$.

"$\supseteq$": It suffices to show $W \cup \mathrm{Con}(D') \subseteq E$ since $E = \mathrm{Th}(E)$. Obviously, $W \subseteq E$. Let $\frac{\alpha:\beta}{\gamma} \in D'$, i.e. $E \models \alpha$ and $E \not\models \neg\beta$. Since $E = \bigcup_{i \geq 0} \mathrm{Th}(E_i)$, there is an i with $E_i \models \alpha$. Thus, $\gamma \in E_{i+1} \subseteq E$.

$\square$

Definition 8.24. Let $D' \subseteq \hat{D}$. We define $D_0 := \emptyset$ and, for $i \geq 0$,

$$D_{i+1} := D_i \cup \left\{ d = \frac{\alpha : \beta}{\gamma} \mid d \in D' \text{ and } W \cup \mathrm{Con}(D_i) \models \alpha \right\}.$$

D' is called *grounded in* W if $D' = \bigcup_{i \geq 0} D_i$.

With this notion, we can now characterize sets of generating defaults. Since every extension E is of the form $E = \mathrm{Th}(W \cup \mathrm{Con}(D'))$ for a set D' of generating defaults, it suffices to consider sets of generating defaults when computing extensions.

Theorem 8.25 (Schwind, Risch). Let $(\hat{D}, W)$ be a closed default theory and $D' \subseteq \hat{D}$. Then D' is a set of generating defaults for some extension E iff the following conditions are satisfied:

1. D' is grounded in W,

2. For all $d = \frac{\alpha:\beta}{\gamma} \in \hat{D}$, it holds that

$$d \in D' \text{ iff } W \cup \mathrm{Con}(D') \models \alpha \text{ and } W \cup \mathrm{Con}(D') \not\models \neg\beta.$$

Proof.

"$\Rightarrow$": Assume that D' is a set of generating defaults for some extension E. With Lemma 8.23, we have $E = \mathrm{Th}(W \cup \mathrm{Con}(D'))$. By Definition 8.22, $D' = \{d = \frac{\alpha:\beta}{\gamma} \mid E \models \alpha \wedge E \not\models \neg\beta\}$. Let the sets E_i be defined as usual. We define

$$D'_0 := \emptyset;$$
$$D'_{i+1} := \left\{ d = \frac{\alpha:\beta}{\gamma} \mid E_i \models \alpha \text{ and } E \not\models \neg\beta \right\}.$$

By definition of E_i and D'_i, $E_i = W \cup \mathrm{Con}(D'_i)$ for all $i \geq 0$. We define another sequence:

$$D_0 := \emptyset;$$
$$D_{i+1} := \left\{ d = \frac{\alpha:\beta}{\gamma} \mid W \cup \mathrm{Con}(D_i) \models \alpha \text{ and } d \in D' \right\}.$$

Using an easy induction, it is not hard to show that $D_i = D'_i$ for $i \geq 0$. Since $E = \bigcup_{i\geq 0} \mathrm{Th}(E_i)$, we have $E \models \alpha$ iff $\exists i.E_i \models \alpha$. Thus $D' = \bigcup_{i\geq 0} D'_i = \bigcup_{i\geq 0} D_i$.
Hence, D' is grounded in W, i.e. point 1 is satisfied. Concerning 2: By definition of D', we have $d = \frac{\alpha:\beta}{\gamma} \in D'$ iff $E \models \alpha$ and $E \not\models \neg\beta$. Since $E = \mathrm{Th}(W \cup \mathrm{Con}(D'))$, this is just point 2.

"$\Leftarrow$": Let D' be a set of defaults satisfying 1 and 2. We define $E := \mathrm{Th}(W \cup \mathrm{Con}(D'))$ and consider the sequences

$$E_0 := W;$$
$$E_{i+1} := E_i \cup \left\{ \gamma \mid d = \frac{\alpha:\beta}{\gamma} \in \hat{D}, E_i \models \alpha, E \not\models \neg\beta \right\};$$

and

$$D_0 := \emptyset;$$
$$D_{i+1} := D_i \cup \left\{ d = \frac{\alpha:\beta}{\gamma} \mid d \in D' \text{ and } W \cup \mathrm{Con}(D_i) \models \alpha \right\}.$$

Claim: $W \cup \mathrm{Con}(D_i) = E_i$ for $i \geq 0$.

$i = 0$: This is obvious since $W \cup \emptyset = W = E_0$.

$i \to i+1$:

"$\subseteq$": The formula γ is contained in $\mathrm{Con}(D_{i+1}) \setminus \mathrm{Con}(D_i)$, i.e. $\frac{\alpha:\beta}{\gamma} \in D'$ and $W \cup \mathrm{Con}(D_i) \models \alpha$. From $\frac{\alpha:\beta}{\gamma} \in D'$ and point 2, we get $W \cup \mathrm{Con}(D') \not\models \neg\beta$. Therefore $E \not\models \neg\beta$. Since $W \cup \mathrm{Con}(D_i) = E_i$, we get $\gamma \in E_{i+1}$.

"$\supseteq$": The formula γ is contained in $E_{i+1} \setminus E_i$, i.e. $\frac{\alpha:\beta}{\gamma} \in \hat{D}$ and $E_i \models \alpha$ and $E \not\models \neg\beta$. It follows $E \models \alpha$ and, since $E = \mathrm{Th}(W \cup \mathrm{Con}(D'))$, we obtain $\frac{\alpha:\beta}{\gamma} \in D'$ by point 2. By induction, $E_i = W \cup \mathrm{Con}(D_i)$ yielding $\frac{\alpha:\beta}{\gamma} \in D_{i+1}$.

This concludes the proof of the claim.

Now we have

$$\begin{aligned}
\bigcup_{i \geq 0} \mathrm{Th}(E_i) &= \bigcup_{i \geq 0} \mathrm{Th}(W \cup \mathrm{Con}(D_i)) && \text{by the claim} \\
&= \mathrm{Th}(W \cup \mathrm{Con}(D')) && \text{since } \bigcup_{i \geq 0} D_i = D' \text{ by point 1} \\
&= E
\end{aligned}$$

□

This theorem can be used to devise an optimized algorithm for computing all extensions of a default theory. A sketch of this algorithm is presented in Figure 10. Note that this algorithm requires that $\widehat{\mathcal{D}}$ and $\mathcal{W}$ are finite and that entailment in the base language is decidable. (Proof of correctness: Baader & Hollunder, JAR 14, 1995).

Compute-All-Extensions$(\widehat{\mathcal{D}}, \mathcal{W})$
begin
(1) if $\mathcal{W}$ is inconsistent
(2) then print "Inconsistent world description"
(3) else for all maximal subsets $\mathcal{D}'$ of $\widehat{\mathcal{D}}$ such that
$\mathcal{W} \cup Con(\mathcal{D}')$ is consistent
(4) do *Remove-Defaults*$(\mathcal{W}, \widehat{\mathcal{D}}, \mathcal{D}')$;
end

Remove-Defaults$(\mathcal{W}, \widehat{\mathcal{D}}, \mathcal{D}')$
begin
(1) let $\mathcal{D}_0$ be the largest subset of $\mathcal{D}'$ that is grounded in $\mathcal{W}$;
(2) if $\mathcal{W} \cup Con(\mathcal{D}_0) \models \neg\beta$ for some justification $\beta \in Jus(\mathcal{D}_0)$
(3) then let $d = \frac{\alpha:\beta}{\gamma}$ be the corresponding default;
(4) *Remove-Defaults*$(\mathcal{W}, \widehat{\mathcal{D}}, \mathcal{D}_0 \setminus \{d\})$;
(5) for all maximal subsets $\mathcal{D}''$ of $\mathcal{D}_0$ such that
$d \in \mathcal{D}''$ and $\mathcal{W} \cup Con(\mathcal{D}'') \not\models \neg\beta$
(6) do *Remove-Defaults*$(\mathcal{W}, \widehat{\mathcal{D}}, \mathcal{D}'')$;
(7) else if for each $\frac{\alpha:\beta}{\gamma} \in \widehat{\mathcal{D}} \setminus \mathcal{D}_0$ either $\mathcal{W} \cup Con(\mathcal{D}_0) \not\models \alpha$
(8) or $\mathcal{W} \cup Con(\mathcal{D}_0) \models \neg\beta$
(9) then add $\mathcal{D}_0$ to the list of sets of generating defaults;
end

Figure 10: Algorithm for computing all extensions of a default theory $(\widehat{\mathcal{D}}, \mathcal{W})$