

Logical Machinery of Heuristics (Preliminary Report)

Shahab Tasharrofi and Eugenia Ternovska
Simon Fraser University
{sta44,ter}@sfu.ca

Abstract

This paper is a preliminary report on a new declarative language that allows the programmer to specify heuristics (problem-specific inference methods) about the problem that is being solved. The heuristics that are defined in our language control and/or change the behavior of the underlying solver. In this way, we are able to attain problem-specific solvers that benefit from both the state-of-the-art general inference methods available in general-purpose solvers, and the problem-specific reasoning methods that are applicable only to this specific problem.

1 Introduction

Solving combinatorially complex problems is, understandably, hard. General tools (such as SAT, ASP, ILP solvers) are useful, yet they may not be very efficient for some problems as they may not be able to take advantage of the combinatorial structure of the problem. On the other hand, problem-specific software may be fast (for that problem), *if* it is written well and *if* it takes advantage of the structure of the problem. However, such software is costly to produce and it may not be efficient.

Our goal is to develop a method for automatically generating problem-specific solvers using general-purpose solvers. A problem-specific solver is different from a general solver in that it uses problem-specific inference mechanisms, in addition to the general ones. For example, in solving a Sudoku puzzle, there are three possibilities. A person may direct another by telling them, step by step, where to put each number. Alternatively, they can explain what constitutes a correct solution, and say nothing more. Finally, they can not only specify the properties of the correct solution, but also say what strategies they use when solving a puzzle. The first method is purely imperative and corresponds to problem-specific software. The second method corresponds to using a general-purpose solver for a declarative language. The third method is a counterpart of a problem-specific solver that we want to develop. In this paper, we use the Sudoku puzzle as our running example. Example 1 defines the Sudoku puzzle.

Example 1 (Sudoku Puzzle). *In Sudoku puzzle, you are given an $n^2 \times n^2$ table with some of its cells filled with numbers from 1 to n^2 . Then, you are asked*

			4	6				3
			3			4	1	
	1	4						
8		1	7		2		3	
	2		8		3	6		9
						9	2	
	8	3			6			
5				4	9			

Figure 1: An Instance of a 9×9 Sudoku Puzzle

to complete filling this table such that each row, each column and each small $n \times n$ sub-table has all the numbers $1, \dots, n^2$. What follows is a declarative axiomatization of the Sudoku puzzle in the language of first-order logic plus arithmetic:

$$\begin{aligned}
& \forall r \forall c \forall v (Filled(r, c, v) \supset Cell(r, c, v)), \\
& \forall r \forall c \forall v_1 \forall v_2 (Cell(r, c, v_1) \wedge Cell(r, c, v_2) \supset v_1 = v_2), \\
& \forall r \forall v \forall c_1 \forall c_2 (Cell(r, c_1, v) \wedge Cell(r, c_2, v) \supset c_1 = c_2), \\
& \forall c \forall v \forall r_1 \forall r_2 (Cell(r_1, c, v) \wedge Cell(r_2, c, v) \supset r_1 = r_2), \\
& \forall i \forall j \forall r_1 \forall c_1 \forall r_2 \forall c_2 \forall v ((i < n \wedge j < n \wedge r_1 < n \wedge c_1 < n \wedge r_2 < n \wedge c_2 < n \wedge \\
& \quad Cell(i * n + r_1, j * n + c_1, v) \wedge Cell(i * n + r_2, j * n + c_2, v)) \supset \\
& \quad (r_1 = r_2 \wedge c_1 = c_2)).
\end{aligned}$$

Figure 1 illustrates an example of a Sudoku instance with $n = 3$.

In this paper, we start developing a framework for generating problem-specific solvers. As always in our research, we rely on classic model theory. A structure for vocabulary σ is a domain (universe), together with an interpretation of relation and function symbols (including constants) over that domain. We represent a combinatorial problem as the logical problem of expanding a σ -structure $\mathcal{A}$ (that represents an instance of the combinatorial problem) to a structure $\mathcal{B}$ to satisfy a specification ϕ over a larger vocabulary $\sigma \cup \varepsilon$ in some logic $\mathcal{L}$ [10]. We call this logical problem a *model expansion (MX)* task. The expansion part of the structure $\mathcal{B}$ (the interpretation of the expansion vocabulary ε) constitutes a possible solution. Such a solution can be obtained by a general-purpose solver, such as, for example, SAT or an ASP solver. An input to the general solver is obtained through a procedure called *grounding*, where formula ϕ is combined with an instance structure $\mathcal{A}$ to generate a variable-free¹ (ground) formula $Gnd(\phi; \mathcal{A})$ over the expansion vocabulary (since the truth values of instance relations are given by structure $\mathcal{A}$, they are evaluated out). The general-purpose solver returns one of the possible interpretations of the expansion vocabulary that constitutes a solution to the combinatorial problem.

We view the work of a solver (either general-purpose or problem-specific) as a process of incremental construction of the expansion structure $\mathcal{B}$ that represents a possible solution. Initially, the interpretation of all the symbols in the expansion vocabulary ε is empty. Then the solver goes through a sequence of states

¹A constant symbol is introduced for each element of the domain of the instance structure.

$s_1, s_2, \dots$ (and backtracks if necessary). Each state s_i is a partial structure that expands instance structure $\mathcal{A}$ and has a partial information about the solution vocabulary ε , listing atoms that are considered true and those considered false at this stage of the computation.

Our general idea for constructing a problem-specific solver is to use a general-purpose solver together with *solving strategies (heuristics)* that are external to the general solver. Each heuristic is implied by axiomatization of the problem ϕ . Problem-specific inference rules are automatically extracted from the heuristics. These rules give both *reason* for abandoning a partially constructed solution and *advice* on how to proceed. Our work may be viewed as a generalization of clause learning in SAT. However, there are significant differences. Problem-specific inference rules (a counterpart of learned clauses) are obtained from the given heuristics during solving, by first solving an “internal” model expansion task, when the conditions are right, and then generating applicable domain-specific inference rules through grounding.

We now explain our general idea in more detail. When solving a problem, two types of inference rules can be used: generic inference rules that apply to all problems, and problem-specific inference rules that only apply to the current problem. Let ϕ be the axiomatization of a problem in a logic $\mathcal{L}$ with a classic model theory (such as first-order logic or a language of ASP). Rule $\frac{\alpha}{\beta}$ is called a *problem-specific inference rule for ϕ* if $\{\phi, \alpha\} \models_{\mathcal{L}} \beta$, i.e., all models that satisfy both ϕ and α also satisfy β . We call a collection I of inference rules for ϕ a *problem-specific inference method*. A problem ϕ may have many different inference methods $I_1, \dots, I_n$. For example, one can have several solving strategies for Sudoku, see Example 2. Each inference method can be very large (i.e., may contain many inference rules). Most of the rules will never be used during a computation. Adding them all as constraints, while logically sound, would not be a smart thing to do computationally. So, we are looking for a method to pick only those inference rules in a problem-specific inference method I that are *applicable at some point in the solving process*. This is done through an *inference procedure* P_I associated with inference method I . Procedure P_I , for each state s (a partial structure), returns a subset of $\{\frac{\alpha}{\beta} \in I \mid s \models \alpha\}$ (i.e., a subset of inference rules in I that are applicable in s). Note that P_I may be *incomplete*, i.e., there might be a rule $\frac{\alpha}{\beta} \in I$ that is never returned by P_I . In other words, for an incomplete P_I , we have:

$$\left(\bigcup_s P_I(s)\right) \subsetneq I$$

while, for a *complete* P_I , we have:

$$\left(\bigcup_s P_I(s)\right) = I$$

A problem-specific solver is a general solver that uses inference procedures $P_{I_1}, \dots, P_{I_n}$ for problem-specific inference methods $I_1, \dots, I_n$. When an inference procedure P_I is called, the rules produced by P_I are added, usually in the form of constraints, to the constraints the solver is already working on. Each procedure P_I is generated automatically from a formula H_I that defines the corresponding inference method I in a precise sense. We call those formulas *heuristic formulas*. Intuitively, H_I describes strategies for solving the general

task. For example, when solving a Sudoku puzzle, those strategies would be as follows:

Example 2 (Two Heuristics to Solve Sudoku Puzzles). *In a Sudoku puzzle, your guess would be wrong if it leads to a situation where, in some column c , a set S of cells in column c exists such that the number of values that can be assigned to cells in S is less than the number of cells in S .*

Also, in all situations, if a column c and a set of cells S in column c can be found such that the number of values that can be assigned to cells in S is equal to the number of cells in S , then the values that can be assigned to S cannot be assigned to cells in column c but outside S . This is because if some other cell in column c takes one of those values, we would end up in the previous situations of having more cells than values.

As we see from the above example, an “internal” MX-task needs to be solved to find the parameters that make α in a rule $\frac{\alpha}{\beta}$ true (so that it becomes applicable). In the above example, we need to find a set S of cells that satisfies the conditions described. In general, finding such a witness can be an NP-complete task. However, once this task is solved, it is clear what to do, and the directions will be of the form of ground inference rules $\frac{\alpha}{\beta}$ in I . The following example shows what happens when the “internal” MX task is solved.

Example 3 (Sudoku Heuristic in Action). *Suppose that we found a set S that contains cells number 2, 5 and 8 in the first column. Also, suppose that the only values that can be assigned to these cells is 1, 6 or 9. Then S satisfies the second heuristic of Example 2, and we have clear directions to follow: we deduct that no other cells except 2, 5 or 8 in the first column can be assigned values 1, 6 or 9. More formally, we will deduce the following logical formula:*

$$\left[\bigwedge_{\substack{r \in \{2, 5, 8\}, \\ v \in \{1, \dots, 9\} \setminus \{1, 6, 9\}}} \neg \text{cell}(r, 1, v) \right] \supset \neg \left[\bigvee_{\substack{r \in \{1, \dots, 9\} \setminus \{2, 5, 8\}, \\ v \in \{1, 6, 9\}}} \text{cell}(r, 1, v) \right]$$

2 Logical Machinery of Heuristics

In this section, we define the notion of heuristic for a declarative problem specification ϕ and describe how such heuristic can be used to construct a problem-specific solver for the problem specified by ϕ . Some of the material used in this section is defined in Appendix A. Such material includes three-valued translations, three-valued semantics and conditions that a logic $\mathcal{L}$ has to satisfy.

Definition 1 (Heuristic). *Let ϕ be a specification in a logic $\mathcal{L}$ over the vocabulary $\sigma \cup \varepsilon$. Then, $H(\bar{x})$ is said to be a heuristic for ϕ with parameters $\bar{x}$ if the following two conditions are simultaneously satisfied:*

1. *H is a pair $(\alpha_H(\bar{x}), \beta_H(\bar{x}))$ where $\alpha_H(\bar{x})$ and $\beta_H(\bar{x})$ are both formulas over the vocabulary $\sigma \cup \tau \cup \varepsilon$.*
2. *For all two-valued $(\sigma \cup \varepsilon \cup \tau)$ -structures $\mathcal{B}$ and for all assignments δ to variables in $\bar{x}$, if $\mathcal{B}$ satisfies ϕ and $\alpha_H(\bar{x})$ is satisfied under $\mathcal{B}$ and δ then $\beta_H(\bar{x})$ is also satisfied under $\mathcal{B}$ and δ .*

Sometimes we refer to $\alpha_H(\bar{x})$ and $\beta_H(\bar{x})$ as the condition and the effect of heuristic H respectively. Moreover, we may drop subscript H or variables $\bar{x}$ from $\alpha_H(\bar{x})$ and $\beta_H(\bar{x})$ if they are clear from the context. Also, in the rest of this paper, we use τ to represent heuristic's vocabulary.

The following example illustrates Definition 1 in the context of our running Sudoku example:

Example 4 (Sudoku: Heuristic). Consider our Sudoku heuristic from Example 2. This heuristic can be defined as the pair $H := (\alpha_H, \beta_H)$ where α_H and β_H are the following formulas in the language of first-order logic plus arithmetic:

$$\begin{aligned} \alpha_H &:= \begin{cases} \forall r \forall v ((\text{Row}(r) \wedge \text{Cell}(r, c, v)) \supset \text{Val}(v)) \wedge \\ |r : \text{Row}(r)| \geq |v : \text{Val}(v)|, \end{cases} \\ \beta_H &:= \begin{cases} |r : \text{Row}(r)| = |v : \text{Val}(v)| \wedge \\ \forall r \forall v ((\text{Vals}(v) \wedge \neg \text{Rows}(r)) \supset \neg \text{Cell}(r, c, v)). \end{cases} \end{aligned} \quad (1)$$

As Equation (1) shows, heuristic H for the Sudoku puzzle uses two new unary predicate symbols Row and Val as well as a free variable c . Therefore, in this example, heuristic's vocabulary is $\tau_H := \{\text{Row}/1, \text{Val}/1\}$ and heuristic's parameter is $\{c\}$.

Intuitively, Definition 1 says that a heuristics is a formula of the form "If condition (α_H) then effect (β_H).". Abusing notations from first-order logic, this intuitive view can be described by the following validity in logic $\mathcal{L}$:

$$\models_{\mathcal{L}} [\phi \supset \forall \bar{x} (\alpha_H(\bar{x}) \supset \beta_H(\bar{x}))]. \quad (2)$$

Note that, in Equation (2), ϕ does not depend on vocabulary symbols in τ while α_H and β_H use vocabulary symbols in τ . Effectively, it means that, for all interpretations of τ , Equation (2) holds. Thus, for all $\sigma \cup \tau$ -structures $\mathcal{A}$ and for all assignments δ to variables in $\bar{x}$ we have that:

$$\text{Gnd}_L(\phi; \mathcal{A}), \text{Gnd}_L(\alpha_H(\bar{x} \setminus \delta(\bar{x})); \mathcal{A}) \models_{\mathcal{L}} \text{Gnd}_L(\beta_H(\bar{x} \setminus \delta(\bar{x})); \mathcal{A})$$

That is, if $\mathcal{L}$ has an implication operator $\supset$ satisfying the deduction theorem, then theory $\{\text{Gnd}_L(\phi; \mathcal{A})\}$ is equivalent to theory $\{\text{Gnd}_L(\phi; \mathcal{A}), \text{Gnd}_L(\alpha_H \supset \beta_H[\bar{x} \setminus \delta(\bar{x})]; \mathcal{A})\}$ for all $\sigma \cup \tau$ -structures $\mathcal{A}$. Moreover, since $\text{Gnd}_L(\phi; \mathcal{A})$ is independent of $\mathcal{A}$'s interpretation of τ and since the previous statement holds for all interpretations of τ , we have the following logical equivalence between theories: for all σ -structures $\mathcal{A}$,

$$\{\text{Gnd}_L(\phi; \mathcal{A})\} \Leftrightarrow \{\text{Gnd}_L(\phi; \mathcal{A})\} \cup \bigcup_{\substack{(\sigma \cup \tau)\text{-structures } \mathcal{B}, \\ \mathcal{B} \text{ expands } \mathcal{A}, \\ \delta : \bar{x} \mapsto \text{dom}(\mathcal{A})}} \{\text{Gnd}_L((\alpha_H \supset \beta_H)[\bar{x} \setminus \delta(\bar{x})]; \mathcal{B})\}$$

That is, the solutions to the problem specified by ϕ for instance structure $\mathcal{A}$ remains the same even if we add all the formulas of form $\text{Gnd}_L(\alpha_H \supset \beta_H; \mathcal{B})$ (for all possible interpretations of τ) to the result of grounding ϕ . As stated in the introduction, the goal of this paper is to intelligently add these formulas to the underlying solver (as constraints) so as to help the underlying solver in its search for a solution to $\text{Gnd}_L(\phi; \mathcal{A})$. One of the most important criteria that we will consider for intelligently adding such formulas is their applicability as defined below:

Definition 2 (Heuristic's Universal Applicability). *Let $H(\bar{x}) := (\alpha_H(\bar{x}), \beta_H(\bar{x}))$ be a heuristic for a problem specified in logic $\mathcal{L}$. Also, let $\mathcal{A}$ be a $(\sigma \cup \varepsilon_l \cup \varepsilon_u)$ -structure that denotes the current state of the solving process. Then we say that heuristic H is universally applicable in state $\mathcal{A}$ if a $(\sigma \cup \varepsilon_l \cup \varepsilon_u \cup \tau)$ -structure $\mathcal{B}$ and assignment $\delta : \bar{x} \mapsto \text{dom}(\mathcal{A})$ exists such that:*

1. $\mathcal{B}$ expands structure $\mathcal{A}$ to vocabulary τ , i.e., $\text{vocab}(\mathcal{B}) = \text{vocab}(\mathcal{A}) \cup \tau$ and $\mathcal{B}|_{\text{vocab}(\mathcal{A})} = \mathcal{A}$, and,
2. for all expansions $\mathcal{B}'$ of $\mathcal{B}$ to vocabulary ε , if $R_l^{\mathcal{B}'} \subseteq R^{\mathcal{B}'} \subseteq R_u^{\mathcal{B}'}$ (for all $R \in \varepsilon$) then $\mathcal{B}', \delta \models_{\mathcal{L}} \alpha_H(\bar{x})$.

If so, $\mathcal{B}$ and δ are said to witness H 's universal applicability in state $\mathcal{A}$.

Intuitively, Definition 2 tells that, in order to intelligently add constraints to the underlying solver, the first and foremost criteria is to add only those constraints that are relevant to the current state of the solving process. The general applicability criterion given by Definition 2 guarantees the relevancy of a heuristic to the current state by guaranteeing that $\text{Gnd}_{\mathcal{L}}(\alpha_H(\bar{x} \setminus \delta(\bar{x})); \mathcal{B})$ (the condition of heuristic H) is satisfied in all possible two-valued extensions of the current state. Therefore, $\text{Gnd}_H(\beta_H(\bar{x} \setminus \delta(\bar{x})); \mathcal{B})$ (i.e., the effect described by heuristic H) is a new constraint that the underlying solver has to satisfy in its current state.

However, even when our instance structure is finite and our heuristic is fixed, in order to check the general applicability condition as in Definition 2, one has to solve a Σ_2^P (possibly Σ_2^P -complete) task (relative to model checking in $\mathcal{L}$) because a second-order existential quantifier (to find τ 's interpretation) is followed by a second-order universal quantifier (to check all possible interpretations of ε). Clearly, in practice, performing such a task is not feasible. Thus, in order to avoid such a high computational complexity, we have modified the applicability condition of a heuristic as follows:

Definition 3 (Heuristic's Applicability). *Let $H(\bar{x}) := (\alpha_H(\bar{x}), \beta_H(\bar{x}))$ be a heuristic for a problem specified in logic $\mathcal{L}$. Also, let $\mathcal{A}$ be a $(\sigma \cup \varepsilon_l \cup \varepsilon_u)$ -structure that denotes the current state of the solving process. Then we say that heuristic H is applicable in state $\mathcal{A}$ if a $(\sigma \cup \varepsilon_l \cup \varepsilon_u \cup \tau)$ -structure $\mathcal{B}$ and assignment $\delta : \bar{x} \mapsto \text{dom}(\mathcal{A})$ exists such that:*

1. $\mathcal{B}$ expands structure $\mathcal{A}$ to vocabulary τ , i.e., $\text{vocab}(\mathcal{B}) = \text{vocab}(\mathcal{A}) \cup \tau$ and $\mathcal{B}|_{\text{vocab}(\mathcal{A})} = \mathcal{A}$, and,
2. $\mathcal{B}, \delta \models_{\mathcal{L}} \alpha_H^+(\bar{x})$ (for positive three-valued translation $\alpha_H^+(\bar{x})$ of $\alpha_H(\bar{x})$).

If so, $\mathcal{B}$ and δ are said to witness H 's applicability in state $\mathcal{A}$.

Following example demonstrates applicability constraint in the context of our running example.

Example 5 (Sudoku: Heuristic's Applicability). *Consider heuristic H from Example 4. In order to check the applicability of H , according to Definition 3, we obtain formula α_H^+ as follows:*

$$\alpha_H^+ := \left\{ \begin{array}{l} \forall r \forall v ((\text{Row}(r) \wedge \text{Cell}_u(r, c, v)) \supset \text{Val}(v)) \wedge \\ |r : \text{Row}(r)| \geq |v : \text{Val}(v)|. \end{array} \right.$$

Note that, in α_H^+ , predicate symbol Cell has been replaced by Cell_u because Cell appears in a negative position. Now, having α_H^+ , we want to check applicability of

H in Example 3. In Example 3, state $\mathcal{A}$ is such that $(r, 1, v) \notin \text{Cell}_u^{\mathcal{A}}$ whenever $r \in \{1, 6, 9\}$ and $v \notin \{2, 5, 8\}$. Thus, we consider following structure $\mathcal{B}$ and assignment δ :

$$\delta := \{c \mapsto 1\}; \mathcal{B}|_{\sigma \cup \varepsilon^+ \cup \varepsilon^-} := \mathcal{A}; \text{Row}^{\mathcal{B}} := \{1, 6, 9\}; \text{Val}^{\mathcal{B}} := \{2, 5, 8\}.$$

Clearly, α_H^+ evaluates to true under structure $\mathcal{B}$ and assignment δ because, according to $\mathcal{B}$, rows 1, 6 and 9 in the first column cannot take any value except 2, 5 or 8. Thus, $\mathcal{B}$ and δ witness the applicability of H in state $\mathcal{A}$.

Note that, if the heuristic H in Definition 3 is fixed, the task of checking H 's applicability is doable in NP (relative to model-checking for $\mathcal{L}$). This is because, given state $\mathcal{A}$, we can non-deterministically guess structure $\mathcal{B}$ expanding $\mathcal{A}$ and accept if our guess satisfies $\alpha_H'(\bar{x})$.

In order to clarify the relationship between Definition 3 and Definition 2, we give Theorem 1 to show that a heuristic is universally applicable whenever it is applicable. Therefore, applicability of a heuristic is a special case of its universal applicability.

Theorem 1 (Applicability $\subseteq$ Universal Applicability). *Let $H(\bar{x}) := (\alpha_H(\bar{x}), \beta_H(\bar{x}))$ be a heuristic for a specification in logic $\mathcal{L}$ and $\mathcal{A}$ be a $\sigma \cup \varepsilon_l \cup \varepsilon_u$ -structure that denote the current state of solving process. Also, let $\mathcal{B}$ and δ witness H 's applicability in state $\mathcal{A}$. Then, $\mathcal{B}$ and δ also witness H 's universal applicability in state $\mathcal{A}$.*

Note that, the converse of Theorem 1 does not hold. As an easy example, consider $\alpha_H(x) := (R(x) \vee \neg R(x))$ (for some $R \in \varepsilon$). Clearly, $\alpha_H(x)$ is valid in all two-valued structures interpreting ε . However, this is not the case for $\alpha_H^+(x) := (R_l(x) \vee \neg R_u(x))$ (for example when $R_l^{\mathcal{B}} = \emptyset$ and $R_u^{\mathcal{B}} = \text{dom}(\mathcal{B})$). Hence, Definition 3 is strictly weaker than Definition 2. That is, there are cases where a heuristic is universally applicable according to Definition 2 but not applicable according to Definition 3.

3 Generating Problem Specific Solvers

Having introduced heuristics in general and having seen sample heuristics, we are now ready to describe how problem specific solvers can be generated by combining the specification of a task with some heuristics that are specifically designed to help solving that particular task. As stated before, a problem-specific solver is a general-purpose solver that is equipped with new inference methods. Therefore, in order to understand how a problem-specific solver works, we should first know its underlying general-purpose solver (henceforth simply called the underlying solver).

Figure 2 shows a rough description of general-purpose solvers and contrasts them against problem-specific solvers. As depicted in Figure (2a), a general-purpose solver moves between three actions of deciding, propagating and back-jumping until it either finds a model or it can establish that no model exists.

However, problem-specific solvers, in addition to propagation mechanisms of underlying solvers, can also obtain knowledge by applying problem-specific heuristics. Thus, as Figure (2b) depicts, problem-specific solvers simply extend

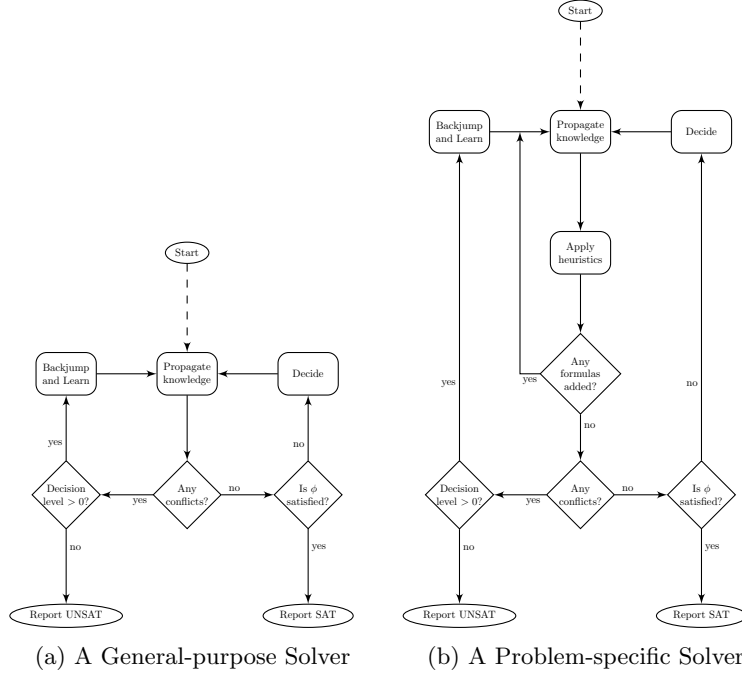


Figure 2: Flow-charts of General-purpose and Problem-specific Solvers

underlying solvers with applications of heuristics in order to obtain new information. Hence, in order to generate problem-specific solvers, we only need to construct procedures that, given the current state of the solver, can produce relevant inference rules.

Algorithms in Figure 3 demonstrate a concrete (and yet high-level) view of a problem-specific solver. As it can be seen in Figure 3, these algorithms use methods for two logical fragments $\mathcal{L}$ and $\mathcal{L}'$. These logical fragments can be the same (e.g., $\mathcal{L}$ and $\mathcal{L}'$ can both be answer set programs) or be different (e.g., $\mathcal{L}$ be first-order logic and $\mathcal{L}'$ be the language of SAT solvers, i.e., CNF formulas).

Our implementation of a problem-specific solver, as in Figure 3, uses methods that are already implemented for logic $\mathcal{L}$, logic $\mathcal{L}'$ or a translation from $\mathcal{L}$ to $\mathcal{L}'$. Examples of such methods are $\text{Gnd}_{\mathcal{L}}$, $\text{Propagate}_{\mathcal{L}'}$, $\text{Decide}_{\mathcal{L}'}$ and $\text{Translate}_{\mathcal{L} \rightarrow \mathcal{L}'}$. Therefore, all the research that is already gone into optimized solving of specifications in a logical fragment can be readily adopted in our framework of developing problem-specific solvers. For example, many brilliant pieces of software has already been produced to solve answer set programs or first-order specifications. In our research direction, we strive to use all such efficient implementations to devise an even more powerful framework for solving model expansion tasks.

4 Conclusion and Future Works

In this paper, we introduced the novel concept of heuristics and described how heuristics can be combined with general-purpose solvers in order to construct

Function $\text{MX}_\phi(\mathcal{A}) : \text{UNSAT} \parallel (\text{SAT}, \mathcal{B})$

```

/*  $\mathcal{A}$  is a  $\sigma$ -structure specifying input to MX task */
/* Specification  $\phi$  (in language  $\mathcal{L}$ ) is fixed */
/* Heuristics  $H_1, \dots, H_n$  for  $\phi$  are also fixed */
 $\psi \leftarrow \text{Gnd}_{\mathcal{L}}(\phi; \mathcal{A});$ 
 $C \leftarrow \text{Translate}_{\mathcal{L} \mapsto \mathcal{L}'}(\psi; \mathcal{A});$ 
return  $\text{Solve}(C, \mathcal{A});$ 

```

Function $\text{Apply}_H(\mathcal{B}') : C'$

```

/* This procedure is for fixed heuristic  $H$ , with fixed  $\alpha_H$ ,
    $\beta_H$  and  $\alpha_H^+$  (positive three-valued translation of  $\alpha_H$ ) */
/*  $\mathcal{B}'$  is a  $(\sigma \cup \varepsilon_l \cup \varepsilon_u)$ -structure specifying solver's state */
 $M \leftarrow \text{MX}_{\alpha_H}(\mathcal{B}')$ ;
if  $M = (\text{SAT}, \mathcal{B}'')$  then
   $\mathcal{A}' \leftarrow \mathcal{B}''|_{\sigma \cup \tau};$ 
  return  $\text{Translate}_{\mathcal{L} \mapsto \mathcal{L}'}(\text{Gnd}_{\mathcal{L}}(\alpha_H \supset \beta_H; \mathcal{A}'), \mathcal{A}')$ 
else return  $\{\};$ 

```

Function $\text{Solve}(C, \mathcal{A}) : \text{UNSAT} \parallel (\text{SAT}, \mathcal{B})$

```

/*  $C$  is a set of ground constraints in language  $\mathcal{L}'$  */
/*  $\mathcal{A}$  is a  $\sigma$ -structure that denotes instance */
/* Heuristics  $H_1, \dots, H_n$  for  $\phi$  are also fixed */
/* For  $i \in \{1, \dots, n\}$ , procedure  $\text{Apply}_{H_i}$  implements heuristic  $H_i$  */
 $\mathcal{B}' \leftarrow \text{Empty expansion of } \mathcal{A} \text{ to } \varepsilon_l \cup \varepsilon_u;$ 
while true do
  repeat
     $\text{Changed} \leftarrow \text{false};$ 
     $\text{Propagate}_{\mathcal{L}'}(C, \mathcal{B}')$ ;
    foreach  $i \in \{1, \dots, n\}$  do
       $C' \leftarrow \text{Apply}_{H_i}(\mathcal{B}')$ ; /*  $C'$ : A set of constraints that  $H_i$  produces on state  $\mathcal{B}'$  */
      if  $C' \not\subseteq C$  then
         $C \leftarrow C \cup C'$ ;
         $\text{Changed} \leftarrow \text{true};$ 
      end
    end
  until  $\neg \text{Changed};$ 
  if  $\mathcal{B}'$  has a conflict then
    if  $\text{Decision level} > 0$  then  $\text{BackjumpAndLearn}_{\mathcal{L}'}(C, \mathcal{B}')$ ;
    else return  $\text{UNSAT};$ 
  else
    if  $\mathcal{B}'$  satisfies  $\phi$  then return  $(\text{SAT}, (\mathcal{B}'|_{\sigma \cup \varepsilon_l})[\varepsilon_l \setminus \varepsilon])$ ;
    else  $\text{Decide}_{\mathcal{L}'}(C, \mathcal{B}')$ ;
  end
end

```

Figure 3: High-level Implementation of a Problem-specific Solver

problem-specific solvers. Our construction of problem-specific solvers is different from many other similar ideas such as soft constraints because of its latent and on-demand generation of inference rules. Moreover, we guarantee that our generated inference rules are relevant to the current state of the solving process by guaranteeing the applicability of a heuristic.

Similarly, our treatment of heuristics is different from incremental grounding techniques [4, 9]. This is because, a heuristic is essentially a different problem that is somehow relevant to the main problem while incremental grounding studies methods to avoid grounding the whole specification. That is, in our situation, we have already grounded the whole specification ϕ and we are interested in methods to expedite the solving process for ϕ .

Furthermore, heuristics are also different from lazy solving techniques [6, 2, 5] used in solving satisfiability modulo theories (SMT) or DPLL(T). This is because, again, the underlying solver in a problem-specific solver already has access to the whole specification while, in SMT, some of the specification is hidden away in the theory solver. In effect, it means that, in our case, a model is accepted if it satisfies all the constraints that the underlying solver has but, in SMT, a model is accepted only if both the underlying solver and all the theory solvers accept the model. This difference allows us to control the expressiveness of our framework in a more fine-grained fashion.

In future, we envisage the development of theoretical underpinnings of our framework in more details. For example, we are interested in devising fast and effective implementations for combining heuristics with the underlying solver. In this direction, we have conceptualized notions of *triggers* and *utility constraints* that enable us to efficiently use heuristics. We have not discussed these notions in the current paper due to size and time constraints.

More importantly, we have embarked upon the project of developing a problem-specific solver generator based on our current framework. Currently, this piece of software is under development and we hope to have demo version available by the time the LaSh workshop starts.

References

- [1] Amir Aavani, Xiongnan (Newman) Wu, Shahab Tasharrofi, Eugenia Ternovska, and David G. Mitchell. Enfragmo: A system for modelling and solving search problems with logic. In *18th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, pages 15–22, 2012.
- [2] Roberto Bruttomesso, Alessandro Cimatti, Anders Franzén, Alberto Griggio, Ziyad Hanna, Alexander Nadel, Amit Palti, and Roberto Sebastiani. A lazy and layered SMT(BV) solver for hard industrial verification problems. In Werner Damm and Holger Hermanns, editors, *Computer Aided Verification*, volume 4590 of *Lecture Notes in Computer Science*, pages 547–560. Springer Berlin Heidelberg, 2007.
- [3] F. Calimeri, W. Faber, M. Gebser, G. Ianni, R. Kaminski, T. Krennwallner, N. Leone, F. Ricca, and T. Schaub. ASP-Core-2 input language format, 2012.

- [4] Broes de Cat, Marc Denecker, and Peter J. Stuckey. Lazy model expansion by incremental grounding. In Agostino Dovier and Vítor Santos Costa, editors, *Technical Communications of the 28th International Conference on Logic Programming, ICLP 2012, September 4-8, 2012, Budapest, Hungary*, volume 17 of *LIPICs*, pages 201–211. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.
- [5] Broes de Cat, Marc Denecker, Peter J. Stuckey, and Maurice Bruynooghe. Lazy model expansion: Interleaving grounding with search. *CoRR*, abs/1402.6889, 2014.
- [6] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS’08/ETAPS’08*, pages 337–340, Berlin, Heidelberg, 2008. Springer-Verlag.
- [7] Marc Denecker and Eugenia Ternovska. Inductive situation calculus. *Artificial Intelligence*, 2007.
- [8] M. Gebser, R. Kaminski, A. König, and T. Schaub. Advances in *gringo* series 3. In J. Delgrande and W. Faber, editors, *Proceedings of the Eleventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR’11)*, volume 6645 of *Lecture Notes in Artificial Intelligence*, pages 345–351. Springer-Verlag, 2011.
- [9] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Max Ostrowski, Torsten Schaub, and Sven Thiele. Engineering an incremental asp solver. In *Proceedings of the 24th International Conference on Logic Programming, ICLP ’08*, pages 190–205, Berlin, Heidelberg, 2008. Springer-Verlag.
- [10] D. G. Mitchell and E. Ternovska. A framework for representing and solving NP search problems. In *Proc. AAAI’05*, pages 430–435, 2005.
- [11] Johan Wittocx, Maarten Mariën, and Marc Denecker. Grounding FO and FO(ID) with bounds. *Journal of Artificial Intelligence Research (JAIR)*, 38(1):223–269, May 2010.

A Three-valued Interpretations and Grounding

In this appendix, we describe the preliminaries required to formally develop the main contributions of this paper. As usual, σ represents instance vocabulary, ϵ represents expansion vocabulary, and, τ represents heuristic vocabulary.

In this paper, we assume that our logic $\mathcal{L}$ has a structural model theory and a corresponding evaluation mechanism. That is, every logic $\mathcal{L}$ considered here is associated with a poset $(L, \leq_L)$, a set of operators $\mathcal{O}$ on elements in L , and an evaluation mechanism $\llbracket \cdot \rrbracket_{\mathcal{L}}^{\mathcal{B}, \delta}$ such that:

1. There are two distinct elements $0_L, 1_L \in L$ with 0_L representing false and 1_L representing true (we may drop subscripts in case of no ambiguity).
2. For all σ -structures $\mathcal{B}$, for all atomic formulas $R(\bar{t})$ (over vocabulary σ), and for all variable assignments δ (to at least free variables of $R(\bar{t})$), we have $\llbracket R(\bar{t}) \rrbracket_{\mathcal{L}}^{\mathcal{B}, \delta} = 1$ if $\mathcal{B}, \delta \models_{\mathcal{L}} R(\bar{t})$, and $\llbracket R(\bar{t}) \rrbracket_{\mathcal{L}}^{\mathcal{B}, \delta} = 0$ otherwise.

3. Each n -ary connective $\circ$ in $\mathcal{L}$ is associated with an n -ary algebraic operator $\circ' \in \mathcal{O}$ such that, for all σ -structures $\mathcal{B}$ and assignments δ to variables $\bar{x}$, and for all formulas $\circ(\phi_1, \dots, \phi_n) \in \mathcal{L}$ (over vocabulary σ and with free variables in $\bar{x}$) we have $\llbracket \circ(\phi_1, \dots, \phi_n) \rrbracket_{\mathcal{L}}^{\mathcal{B}, \delta} = \circ'(\llbracket \phi_1 \rrbracket_{\mathcal{L}}^{\mathcal{B}, \delta}, \dots, \llbracket \phi_n \rrbracket_{\mathcal{L}}^{\mathcal{B}, \delta})$.
4. Finally, for all σ -structures $\mathcal{B}$ and assignments δ to variables $\bar{x}$, and for all formulas $\phi \in \mathcal{L}$ (over vocabulary σ and with free variables in $\bar{x}$) we have that $\mathcal{B}, \delta \models_{\mathcal{L}} \phi$ if and only if $\llbracket \phi \rrbracket_{\mathcal{L}}^{\mathcal{B}, \delta} = 1$.

Example 6 (Evaluation Mechanisms). *The requirement of having an evaluation mechanism for a logic is already satisfied in many logical fragments such as classical logic and intuitionistic logic. As an example, boolean algebras define such an evaluation mechanism for propositional classical logic and a free heyting algebra gives such an evaluation mechanism for propositional intuitionistic logic.*

In addition to the above conditions on a logic $\mathcal{L}$, we also require each n -ary operator $\circ' \in \mathcal{O}$ to be either monotone or anti-monotone with respect to each of its arguments. That is, for all $1 \leq i \leq n$, one of the following cases hold:

- Either, for all $a_1, \dots, a_n \in L$ and for all $a'_i \in L$ with $a_i \leq_L a'_i$ we have $\circ'(a_1, \dots, a_{i-1}, a_i, a_{i+1}, \dots, a_n) \leq_L \circ'(a_1, \dots, a_{i-1}, a'_i, a_{i+1}, \dots, a_n)$,
- Or, for all $a_1, \dots, a_n \in L$ and for all $a'_i \in L$ with $a_i \leq_L a'_i$ we have $\circ'(a_1, \dots, a_{i-1}, a'_i, a_{i+1}, \dots, a_n) \leq_L \circ'(a_1, \dots, a_{i-1}, a_i, a_{i+1}, \dots, a_n)$.

In the former case, $\circ'$ is *monotone with respect to its i -th argument* and, in the latter case, $\circ'$ is *anti-monotone with respect to its i -th argument*. We also extend these notions of monotone and anti-monotone (with respect to argument positions) to each connective $\circ$ of logic $\mathcal{L}$ that corresponds to operator $\circ' \in \mathcal{O}$.

Example 7 (Monotonicity and Anti-monotonicity w.r.t. Arguments). *Connectives (and their corresponding operators) in many logical fragments (such as classical logic, intuitionistic logic and logic of here-and-there) satisfy monotonicity and anti-monotonicity conditions with respect to their arguments. For example, both binary conjunction and binary disjunction are monotone with respect to both of their arguments while negation is anti-monotone with respect to its only argument. Moreover, implication is anti-monotone with respect to its antecedent and monotone with respect to its consequent.*

Using the notions of monotone and anti-monotone connective positions, we define positive and negative subformulas for logic $\mathcal{L}$ as follows:

Definition 4. *Let ϕ be a formula in logic $\mathcal{L}$ and ψ be a subformula of ϕ . Then, ψ is said to be a positive subformula of ϕ if it appears inside an even number of anti-monotone connective positions and it is said to be a negative subformula of ϕ if it appears inside an odd number of connective positions.*

Definition 4 corresponds to ordinary definitions of positive/negative subformulas in many logical frameworks such as classical, intuitionistic or HT-logic. Using Definition 4, we define the three-valued semantics for a logic $\mathcal{L}$ as follows:

Definition 5 (Three-valued Semantics). *For logic $\mathcal{L}$ with $0_L \leq_L 1_L$, define:*

1. A partial σ -structure $\mathcal{A}$ is a structure in which, for all n -ary predicate symbol $R \in \sigma$ and for all n -ary tuples $t \in [\text{dom}(\mathcal{A})]^n$, the query $t \in R^{\mathcal{A}}$ can have one of the following three results: yes, no or unknown.

2. For partial σ -structure $\mathcal{A}$, total vocabulary of $\mathcal{A}$ is $\sigma' \subseteq \sigma$ such that $\mathcal{A}|_{\sigma'}$ is a total structure (i.e., has no unknown values). Also, partial vocabulary of $\mathcal{A}$ is equivalent to $\sigma \setminus \sigma'$.
3. For a partial σ -structures $\mathcal{A}$ and $\sigma' \subseteq \sigma$ (with σ' containing at least the partial vocabulary of $\mathcal{A}$), three-valued translation of $\mathcal{A}$ w.r.t. σ' is a $((\sigma \setminus \sigma') \cup \sigma'_l \cup \sigma'_u)$ -structure $\mathcal{A}'$ such that (1) σ'_l (respectively, σ'_u) contains new predicate symbol R_l (respectively, R_u) for each predicate symbol $R \in \sigma'$, (2) $R_l^{\mathcal{A}'}$ (respectively, $R_u^{\mathcal{A}'}$) is the set of tuples t for which query $t \in R^{\mathcal{A}}$ is true (respectively, query $t \in R^{\mathcal{A}}$ is not false), and, (3) $\mathcal{A}'|_{\sigma} = \mathcal{A}|_{\sigma}$.
4. For formula $\phi \in \mathcal{L}$ and vocabulary σ , positive (respectively, negative) three-valued translation of ϕ w.r.t. σ , denoted by ϕ^+ (respectively, denoted by ϕ^-), is the formula obtained from ϕ by replacing positive occurrences of atomic subformulas $R(\bar{t})$ with $R_l(\bar{t})$ (respectively, with $R_u(\bar{t})$) and negative occurrences of atomic formula $R(\bar{t})$ with $R_u(\bar{t})$ (respectively, with $R_l(\bar{t})$) whenever $R \in \sigma$.
5. For partial σ -structure $\mathcal{A}$, vocabulary σ' containing at least the partial vocabulary of $\mathcal{A}$, variable assignment δ , and formula $\phi \in \mathcal{L}$, we say $(\mathcal{A}, \delta)$ satisfies ϕ , denoted by $\mathcal{A}, \delta \models_{\mathcal{L}-3} \phi$, if $\mathcal{A}, \delta \models_L \phi^+$ and $(\mathcal{A}, \delta)$ falsifies ϕ , denoted by $\mathcal{A}', \delta \models_{\mathcal{L}-3} \phi$, if $\mathcal{A}', \delta \not\models_{\mathcal{L}} \phi^-$ where $\mathcal{A}'$ is the three-valued translation of $\mathcal{A}$ w.r.t. σ' .

Definition 5 is used throughout this paper. Thus, it is essential to show its naturalness. We achieve this goal through a series of propositions about this definition. The first such proposition is as follows:

Proposition 1 (Three-valued Semantics is Well-defined). *Let $\mathcal{L}$ be a logic as in Definition 5. Then, for all partial σ -structures $\mathcal{A}$, for all variable assignments δ to variables $\bar{x}$, and for all $\phi \in \mathcal{L}$ (over vocabulary σ and with free variables in $\bar{x}$), it cannot be the case that $(\mathcal{A}, \delta)$ simultaneously satisfies and falsifies ϕ .*

Proposition 1 says that our definition of three-valued semantics is well-defined because it can never be the case that a formula is both satisfied and falsified (simultaneously) under a partial structure. Note that one can easily construct cases where $(\mathcal{A}, \delta)$ neither satisfies nor falsifies ϕ . In fact, this is exactly why we call it a three-valued semantics. Because, for all formulas ϕ over vocabulary σ and with free variables $\bar{x}$, for all partial structures $\mathcal{A}$ interpreting σ , and for all variable assignments δ to $\bar{x}$, ϕ evaluates to exactly one of the following three values: (1) ϕ evaluates to true when $(\mathcal{A}, \delta)$ satisfies it (2) ϕ evaluates to false when $(\mathcal{A}, \delta)$ falsifies it, and, (3) ϕ evaluates to unknown when it is neither satisfied nor falsified by $(\mathcal{A}, \delta)$.

In Definition 5, partial structures can be thought of as approximations of total structures (structures without unknown values). In order to formalize this notion, we use the precision ordering $\leq_p$ between partial σ -structures as follows: For two partial σ -structure $\mathcal{A}_1$ and $\mathcal{A}_2$, we say $\mathcal{A}_2$ is more precise than $\mathcal{A}_1$, denoted by $\mathcal{A}_1 \leq_p \mathcal{A}_2$ if, for all $R \in \sigma$, we have $R_l^{\mathcal{A}_1} \subseteq R_l^{\mathcal{A}_2} \subseteq R_u^{\mathcal{A}_2} \subseteq R_u^{\mathcal{A}_1}$ where $\mathcal{A}_1'$ and $\mathcal{A}_2'$ are three-valued representations of $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively. Clearly, according to the precision ordering, total structures are maximal elements, i.e., the most precise it can get. The next proposition gives us an understanding of three-valued semantics' behavior for such most precise partial structures.

Proposition 2 (Three-valued Semantics under Total Structures). *For logic $\mathcal{L}$, σ -structure $\mathcal{A}$, formula $\phi \in \mathcal{L}$, and, variable assignment δ , we have (1) $\mathcal{A}, \delta \models_{\mathcal{L}} \phi$ if and only if $\mathcal{A}', \delta \models_{\mathcal{L}-3} \phi$, and, (2) $\mathcal{A}, \delta \not\models_{\mathcal{L}} \phi$ if and only if $\mathcal{A}', \delta \not\models_{\mathcal{L}-3} \phi$.*

Proposition 2 says that, as expected, three-valued semantics of $\mathcal{L}$ coincides with its ordinary two-valued model-theoretical semantics if $\mathcal{A}$ is total (i.e., $\mathcal{A}$ has no unknown values). While Proposition 2 is indeed a property always expected from a reasonable three-valued semantics, it does not yet show us the full picture about three-valued semantics. The following proposition completes this picture by relating precision of partial structures to three-valued semantics.

Proposition 3 (Progression in Three-valued Semantics). *Consider logic $\mathcal{L}$ and partial σ -structures $\mathcal{A}_1$ and $\mathcal{A}_2$ with $\mathcal{A}_1 \leq_p \mathcal{A}_2$. Then, for all formulas $\phi \in \mathcal{L}$ and for all variable assignments δ , we have (1) $\llbracket \phi^+ \rrbracket_{\mathcal{L}}^{\mathcal{A}_1, \delta} \leq_L \llbracket \phi^+ \rrbracket_{\mathcal{L}}^{\mathcal{A}_2, \delta}$, and, (2) $\llbracket \phi^- \rrbracket_{\mathcal{L}}^{\mathcal{A}_2, \delta} \leq_L \llbracket \phi^- \rrbracket_{\mathcal{L}}^{\mathcal{A}_1, \delta}$.*

Proposition 3 shows that the more precise partial structures get, the further away they get from an unknown value. Indeed, as Proposition 2 shows, the most precise partial structures (i.e., total structures) cannot evaluate to unknown in their three-valued semantics. We thus have the following corollary:

Corollary 1. *For logic $\mathcal{L}$, partial σ -structures $\mathcal{A}_1$ and $\mathcal{A}_2$ with $\mathcal{A}_1 \leq_p \mathcal{A}_2$, variable assignment δ , and formula $\phi \in \mathcal{L}$, we have:*

- *If $(\mathcal{A}_1, \delta)$ satisfies ϕ then $(\mathcal{A}_2, \delta)$ also satisfies ϕ , and,*
- *If $(\mathcal{A}_1, \delta)$ falsifies ϕ then $(\mathcal{A}_2, \delta)$ also falsifies ϕ .*

Corollary 1 says that, once, under a partial structure $\mathcal{A}$, a formula ϕ evaluates to something other than unknown in the three-valued semantics of $\mathcal{L}$, then it will continue to have that value in all structures that are more precise than $\mathcal{A}$. This property is essential if a three-valued semantics is going to be used in a solver. For instance, this property is used in a solver so as to either justify solver's backtracking/backjumping once the current partial structure falsifies a constraints, or, justify solver's reporting of a model once a partial structure satisfies all constraints.

Apart from all the previous conditions we have put on our choice of logic $\mathcal{L}$, we also require $\mathcal{L}$ to have a grounding mechanism. That is, a grounding procedure $Gnd_{\mathcal{L}}$ exists such that, for all sentences ϕ in $\mathcal{L}$ over vocabulary $\sigma \cup \varepsilon$ and for all two-valued σ -structures $\mathcal{A}$, $Gnd_{\mathcal{L}}(\phi; \mathcal{A})$ returns a variable-free formula ψ in $\mathcal{L}$ over vocabulary $\varepsilon \cup \tilde{A}$ such that, for all two-valued $(\sigma \cup \varepsilon \cup \tilde{A})$ -structure $\mathcal{B}$ expanding $\mathcal{A}$ and interpreting $\tilde{A}$ as required, we have that $\mathcal{B}$ satisfies ϕ if and only if $\mathcal{B}$ satisfies ψ .

Example 8 (Grounding). *Similar to three-valued semantics and positive/negative subformulas, grounding is also a well-studied (although more recent) subject that is already available for many logical fragments. For example, Enfragmo system [1] implements intelligent grounding for an extension of first-order logic, IDP system [11] does grounding for the language of FO(ID) [7], and, Gringo [8] is a grounder for a framework of answer set programming [3].*