Declarative Solver Development: Case Studies

Bart Bogaerts and **Tomi Janhunen** and **Shahab Tasharrofi**
Helsinki Institute for Information Technology HIIT
Department of Computer Science
Aalto University, FI-00076 AALTO, Finland

Abstract

The formalisms for knowledge representation and reasoning (KR&R) typically have a variety of semantics, each one having its particular application scenarios. However, the KR&R community cannot readily benefit from such a variety due to a lack of efficient solver technology. This is partly caused by the fact that solver development is laborious and even accomplishing a working prototype can form a major effort.

In this paper, we introduce a new framework that enables us to *declaratively specify* a given semantics in second-order logic and to *automatically generate* a solver from that specification. Hence, KR&R researchers can rapidly develop a solver prototype for their new/existing semantics with a minimal effort. Technically, our framework builds on a recent approach for nesting SAT solvers based on lazy clause generation.

We evaluate our framework in the context of Dung's argumentation frameworks, logic programming, and propositional logic subject to standard and non-standard semantics. We show for each of those formalisms that one can easily specify its semantics using a few second-order sentences and that one can effectively obtain a solver for that semantics using our automated solver generation procedure.

For instance, in the case of argumentation frameworks, we obtain 16 different solvers, each solving one of four inference tasks for one of four major argumentation semantics and show that our solvers (slightly) outperform the best solver from the last system competition despite not being tuned for argumentation instances.

1 Introduction

Decades of research in Knowledge Representation (KR) have produced an immense amount of interesting KR languages that often come with a variety of semantics, each obeying different intuitions. Unfortunately, development of solver technology for these languages often lags behind, and for several languages the research remains purely theoretical. One possible explanation for this state of affairs is the fact that solver development is laborious and even accomplishing a working solver prototype can form a major implementation effort. Thus, the KR community often cannot readily benefit from the rich variety of languages in practice.

Luckily, both the *number* of available solvers and their *efficiency* has vastly improved over the last years. One of the most important boosts to solver development has been the addition of conflict-driven clause learning (CDCL) (Marques-Silva and Sakallah 1999) to SAT solvers, increasing both the *popularity* of SAT solvers for tackling real-life problems and their *efficiency*.

Many communities have warmly embraced CDCL. Researchers extended CDCL to create solvers for rich KR languages that go *beyond SAT* either because they try to tackle problems of a complexity higher than NP or because propositional logic is too limited to concisely express certain domain specific constraints, such as graph properties. To go beyond NP, one often either uses *incremental SAT solving* (Nadel and Ryvchin 2012) or extends the SAT language and solvers with, respectively, new *constraints* and *propagators* for those constraints. In the latter case, propagators often generate clauses to communicate with the underlying SAT solver. This approach has been followed in a number of fields. For instance, in the form of lazy clause generation (Ohrimenko, Stuckey, and Codish 2009) in constraint programming (Apt 2003), and DPLL(T) (Ganzinger et al. 2004) in SAT modulo theories (Barrett et al. 2009). An analogous architecture can be found inside modern answer-set solvers (Gebser, Kaufmann, and Schaub 2012; De Cat et al. 2013; Alviano et al. 2015) as well as in many other extensions to SAT solvers (Gebser, Janhunen, and Rintanen 2014; Bayless et al. 2015).

Even though one can often build on efficient, extensible SAT solvers (Eén and Sörensson 2004), extending a SAT solver tends to become a tedious and time-consuming programming exercise. The goal of this paper is to overcome this difficulty. To this end, we present a new methodology that enables *declarative specification* of a semantics of a KR formalism. We present tools that, given such a declarative specification, automatically produce a SAT-based solver for the specified semantics. This solver consists of a CDCL solver augmented with automatically generated propagators that perform lazy clause generation. The propagators themselves use CDCL under assumptions (Nadel and Ryvchin 2012) to generate small learned clauses.

We evaluate the proposed methodology from a KR perspective. We compactly describe various semantics of different formalisms in second-order logic and automatically obtain solvers from these specifications. In the context of *Dung's argumentation frameworks* (Dung 1995), discussed in Section 3, we present solvers for *finding an extension*, *enumerating extensions*, *cautious reasoning* and *credulous reasoning* for four main semantics, namely, *complete semantics*, *stable semantics*, *preferred semantics* and *grounded semantics*. For *logic programming*, in Section 4, we create solvers for the *stable model semantics* (Gelfond and Lifschitz 1988) and the *grounded model semantics* (Bogaerts, Vennekens, and Denecker 2015). In Section 5, we specify several semantics for propositional logic: classical models, HT-models (Heyting 1930), equilibrium models (Pearce 2000), supported models (Tasharrofi 2013), and parallel circumscription (Lifschitz 1985).

Since our methodology is purely declarative, it is also very flexible: small changes in semantics, or in the used reasoning task, translate to small changes in the second-order specifications. This flexibility shows up in the context of argumentation frameworks where large parts of the specifications are shared when changing semantics or reasoning task.

One might wonder whether this generic approach is efficient enough for practical use. We are confident that strongly tuned domain-specific solvers can always outperform such a solver. However, in many cases, obtaining the absolute best possible performance is not worth the development cost (for example when prototyping solvers for a certain semantics, writing checkers for competitions, or tackling applications that are not really pushing the boundaries of the current technology). In such cases, our methodology can be used (and in fact *should* be used given the low development cost). We experimentally evaluate performance in the context of argumentation frameworks. Here, we show that our solvers (slightly) outperform the state of the art solvers on the benchmarks used in the argumentation competition.

On the technical level, our approach is based on recent work by Janhunen, Tasharrofi, and Ternovska (2016). They developed a solver, called SAT-TO-SAT that integrates a CDCL solver with lazy clause generation propagators that internally use another CDCL solver as an oracle. Soon after this solver was presented, it was extended to allow for deeper *nesting* of solvers, essentially resulting in a solver for Quantified Boolean Formulas (QBFs) (Bogaerts, Janhunen, and Tasharrofi 2016). In Section 2.1 we recall the working of SAT-TO-SAT. Our methodology builds upon SAT-TO-SAT by adding a second-order modelling language and automatically generating the SAT-TO-SAT specifications based on a second-order specification.

This paper's main contribution is to show that, with the SAT-TO-SAT methodology, reasonably efficient solvers can be easily generated for a broad variety of logics. This paper's tools and examples are available at `http://research.ics.aalto.fi/software/sat/sat-to-sat/`.

2 SAT-TO-SAT and Second-Order Logic

2.1 Preliminaries

We assume familiarity with the basic concepts of propositional logic. A *vocabulary* is a set of symbols, also called *atoms*; we use σ, τ, ν to refer to vocabularies. A *literal* is an atom or its negation. Propositional formulas are defined in the standard way. A formula is in *conjunctive normal form (CNF)* if it is a conjunction of disjunctions of literals. If σ is a vocabulary, a (two-valued) σ-interpretation is a mapping $\sigma \rightarrow \{\mathbf{t}, \mathbf{f}\}$ where $\mathbf{t}$ denotes *true* and $\mathbf{f}$ *false*. A *partial σ-interpretation* is a mapping $\sigma \rightarrow \{\mathbf{t}, \mathbf{f}, \mathbf{u}\}$, where $\mathbf{u}$ denotes *unknown*. We call I more precise than J (notation $I \geq_p J$) if $I(p) = J(p)$ whenever $J(p) \neq \mathbf{u}$. The satisfaction relation $\models$ between interpretations and propositional theories is defined as usual. We call φ *I-satisfiable* if φ has a model more precise than I and *I-unsatisfiable* otherwise.

Traditionally, the task of a SAT solver is to check the satisfiability of a CNF φ over vocabulary ν. But many SAT solvers do more than that: they return a model of φ (hence, they perform the *model expansion task*) and they explain their answer (`SAT` and a model or `UNSAT`) in terms of a set of so-called *assumptions* (Nadel and Ryvchin 2012). In this paper, we assume that ν is the disjoint union of two vocabularies σ and τ, an *assumption vocabulary* σ and an *internal* vocabulary τ.

Definition 2.1 (Explaining Satisfiability)**.** Let φ be a formula over ν and $\nu = \sigma \cup \tau$ where σ and τ are disjoint. Let J be a partial σ-interpretation and M be a partial τ-interpretation. We say that (J, M) *explains φ's satisfiability* if all ν-interpretations more precise than J and more precise than M satisfy φ.

Definition 2.2 (SAT-solver)**.** Suppose $\nu = \sigma \cup \tau$. A *SAT-solver* is a procedure that takes as input a ν-CNF $\mathcal{T}$ and a two-valued σ-interpretation I.

- If $\mathcal{T}$ is I-satisfiable, it returns $(\texttt{SAT}, J, M)$ such that $J \leq_p I$ and (J, M) explains the satisfiability of φ.
- Otherwise, it returns $(\texttt{UNSAT}, J)$ where $J \leq_p I$ is such that $\mathcal{T}$ is J-unsatisfiable.

Hence SAT solvers solve the satisfiability problem of $\mathcal{T}$ under assumptions I. Note that in formalisations of SAT solvers, often a J that explains the answer of the solver is not present. However, J can always be taken equal to I. Modern SAT solvers, such as MiniSAT (Eén and Sörensson 2004), support smart reasoning methods to generate better (less precise) J.

In order to solve problems of form $\exists \nu : \mathcal{T}$, state-of-the-art SAT solvers use the conflict-driven clause learning (CDCL) algorithm (Silva, Lynce, and Malik 2009). The CDCL algorithm works by maintaining a state that represents a partial ν-interpretation. We use $\mathfrak{S}(S)$ to denote the state of a solver S. The algorithm uses operations of propagation, decision, backjumping, and restart to manipulate its state. Propagation takes a state $\mathfrak{S}(S)$ and either returns a (possibly) more precise state that is the consequence of its previous state, or returns a conflict clause showing no model can extend the current state. The decision operation takes a non-conflicting state $\mathfrak{S}(S)$ and branches the search on a variable v (decision variable) that is currently unassigned in $\mathfrak{S}(S)$. Backjumping takes a conflicting state $\mathfrak{S}(S)$, learns a clause from it and returns to a less precise non-conflicting state. The restart operation restarts the search while remembering learnt clauses.

Janhunen, Tasharrofi, and Ternovska (2016) recently introduced a framework for combining SAT solvers so that they solve $\exists\forall$QBF problems together. Essentially, this framework is based on lazy clause generation (Ohrimenko, Stuckey, and Codish 2009) where clauses are obtained from calls to another SAT solver. In this section, we present a formalisation of the working of SAT-TO-SAT in a slightly generalised setting, based on the work by Bogaerts, Janhunen, and Tasharrofi (2016).

The input for SAT-TO-SAT is an $\exists\forall$QBF of the form

$$\mathcal{T} = \exists\sigma : \varphi \wedge (\neg\exists\tau_1 : \psi_1) \wedge \cdots \wedge (\neg\exists\tau_n : \psi_n),$$

where φ is a σ-CNF and ψ_i's are ν_i-CNFs with $\nu_i = \sigma \cup \tau_i$. Without loss of generality, from now on, we assume that $n = 1$ and use τ for τ_1, ψ for ψ_1 and ν for $\sigma \cup \tau$. SAT-TO-SAT checks the validity of $\mathcal{T}$, i.e., it returns `SAT` iff there exists a σ-interpretation I that satisfies φ such that ψ is I-unsatisfiable and returns `UNSAT` otherwise. To explain how SAT-TO-SAT works, we need some terminology:

Definition 2.3 (Lowerbound/Upperbound Mapping)**.** The *LU-mapping of vocabulary* σ is $\sigma_{lu} = \{p_u \mid p \in \sigma\} \cup \{p_l \mid p \in \sigma\}$ with p_u (resp. p_l) representing upper- (resp. lower-) bound of p. The *LU-mapping* of a partial interpretation I, denoted as I_{lu}, is a 2-valued σ_{lu}-interpretation so that $I_{lu}(p_u) = \mathbf{t}$ if and only if $I(p) \neq \mathbf{f}$ and $I_{lu}(p_l) = \mathbf{t}$ if and only if $I(p) = \mathbf{t}$.

Note that for each atom p in the vocabulary σ, I_{lu} satisfies $I_{lu}(p_l) \leq_t I(p) \leq_t I_{lu}(p_u)$, i.e., p_l (respectively p_u) is a *lower-* (respectively *upper-*) bound on the truth value of p.

Definition 2.4 (σ-under-approximation)**.** Let ψ be a $\sigma \cup \tau$ formula. A *σ-under-approximation* of ψ is any $\sigma_{lu} \cup \tau$-formula η such that for all interpretations I:

(1) if I is a two-valued σ-interpretation, then $\exists\tau : \eta$ is satisfied in I_{lu} iff $\exists\tau : \psi$ is satisfied in I, and

(2) if I is a partial σ-interpretation, then $I_{lu} \models \exists\tau : \eta$ implies that every two-valued σ-interpretation more precise than I satisfies $\exists\tau : \psi$.

The first condition guarantees that in two-valued interpretations the approximation coincides with the original formula. The second states that if I_{lu} can be expanded to a model of η, then I can be expanded to a model of ψ.

Assuming a σ-under-approximation η for ψ, the SAT-TO-SAT algorithm instantiates two CDCL-solvers S_φ (tasked with solving φ) and S_η (tasked with solving η). After each unit propagation phase of S_φ, solver S_η is called with assumptions $\mathfrak{S}(S_\varphi)_{lu}$.

- If S_η returns $(\texttt{SAT}, J, M)$, it then follows from the fact that η is a σ-under-approximation of ψ that $\neg\exists\tau : \psi$ (and hence also $\mathcal{T}$) is I-unsatisfiable. In this case, J is used to create a clause that falsifies S_φ's current assignment; this clause is added to φ.
- If S_η returns $(\texttt{UNSAT}, J)$, nothing can be concluded. Literals in J are used as watched literals to avoid calling S_η again as long as $\mathfrak{S}(S_\varphi)_{lu}$ is more precise than J.

The use of the under-approximation has the effect that if $\mathfrak{S}(S_\varphi)$ is not exact, the call to the nested solver S_η remains sound in the sense that whenever S_η finds a model, a conflict clause can be added to φ. In case S_η is unsatisfiable (with the given assumption), nothing final can be concluded yet. In this case, the explanation of unsatisfiability can be used to avoid calling the nested solver too often.

The only thing that remains to be explained is how SAT-TO-SAT obtains a σ-approximation of ψ. This is done by the following syntactical transformation.

Lemma 2.5. *Let ψ be a $\sigma \cup \tau$-CNF. Let ψ_{lu} be the $\sigma_{lu} \cup \tau$-CNF obtained from ψ by*
1. replacing each literal $\neg p$ in ψ with $p \in \sigma$ by $\neg p_u$, and
2. replacing each literal p in ψ with $p \in \sigma$ by p_l.
Then, ψ_{lu} is a σ-under-approximation of ψ.

Bogaerts, Janhunen, and Tasharrofi (2016) extended these ideas to a richer setting by allowing a deeper nesting of solvers, hence essentially solving the validity problem of arbitrary Quantified Boolean Formulas (QBFs), where the structure of the quantification directly maps to the nesting structure of solvers. SAT-TO-SAT can generate all models of an input theory (instead of just one) using a standard wrapper which adds a model invalidating clause each time a solution is found and continues search afterwards.

2.2 Second-Order Modelling for SAT-TO-SAT

The input language for SAT-TO-SAT is an extension of the DIMACS format for specifying CNF theories. The SAT-TO-SAT format **(1)** allows for multiple CNF specifications and **(2)** adds a mechanism to link these specifications together, i.e., to specify the nesting structure of the solvers.

Since this is not a natural modelling language, we provide a high-level interface for our solver. We chose second-order logic since this is a well-known and widely understood language with a clear informal semantics. As discussed in the "Future Work" section, we do not consider second-order logic to be the final modelling language; we plan to extend our language with more features to obtain a higher degree of modelling flexibility.

We assume familiarity with the basics of second-order logic (SO). A vocabulary σ is a set of predicate and function symbols. Without loss of generality, in what follows we assume that σ is relational, i.e., σ has no function symbols. Terms, atoms, formulas, interpretations and the satisfaction relation are defined as usual. If I is an interpretation and p a predicate symbol, we use p^I to denote the interpretation of p in I. A σ-theory $\mathcal{T}$ is a set of second-order formulas with free symbols in σ. Suppose $\mathcal{T}$ is σ-theory and $p \in \sigma$ and $\sigma' = (\sigma \setminus \{p\}) \cup \{p'\}$; we use $\mathcal{T}[p/p']$ to denote the σ' theory obtained from $\mathcal{T}$ by replacing all occurrences of p by p'. Slightly abusing notation, we sometimes identify a finite theory $\mathcal{T}$ with the conjunction of all formulas in $\mathcal{T}$. If $\mathcal{T}$ is a theory, we sometimes use it in another theory as an abbreviation to put the formula it represents in that place.

For vocabularies σ and $\tau \subseteq \sigma$, the *model expansion task* takes a σ-theory $\mathcal{T}$ and a τ-interpretation I and returns a σ interpretation J such that $J \models \mathcal{T}$ and $J|_\tau = I$ if such J exists and returns `UNSAT` otherwise.

In order to translate a model expansion task for SO into a SAT-TO-SAT specification, we developed a tool, called SO2GROUNDER that takes as input a second-order theory $\mathcal{T}$ and a vocabulary τ and returns a *τ-grounder* for $\mathcal{T}$. That is, it returns a tool that takes as input a τ-interpretation I (in the form of a set of facts) and returns a SAT-TO-SAT-specification. The current implementation of

SO2GROUNDER is very basic and works by simple theory transformations as follows. **(1)** First, it transforms $\mathcal{T}$ into the form $\exists \nu_1 : \varphi_1 \wedge \neg \exists \nu_2 : \varphi_2$, where φ_1 is a theory without second-order quantifications and φ_2 is a theory of the same form as $\mathcal{T}$. This is done by pulling second-order quantifications as much outwards as possible using equivalences such as $(\exists p : \varphi) \wedge \psi \equiv \exists p : (\varphi \wedge \psi)$ for rewriting. **(2)** Next, it transforms φ_1 to a set of sentences of the form

$$\forall \overline{X} : \psi_0^\tau \Rightarrow (\exists \overline{Y}_1 : \psi_1^\tau \wedge \psi_1) \vee \cdots \vee (\exists \overline{Y}_n : \psi_n^\tau \wedge \psi_n), \quad (1)$$

where the ψ_i^τ are conjunctions of literals over τ and the ψ_i are disjunctions of literals over $\sigma^* \setminus \tau$, and σ^* extends σ with introduced Tseitin symbols. Intuitively, the ψ_i^τ are *guards* of the quantifications. They serve to limit the domain of variables that need to be instantiated. In order to obtain sentences of this form, φ_1 is first transformed into negation normal form by pushing negations inwards and, afterwards, Tseitin symbols are introduced where needed. **(3)** First-order sentences of the form (1) correspond exactly to rules in a GRINGO-SATGRND specification (Gebser et al. 2015). SO2GROUNDER creates a script that calls GRINGO-SATGRND to instantiate all first-order variables and create a CNF. A recursive call to SO2GROUNDER results in a script to ground $\exists \nu_2 : \varphi_2$. **(4)** Finally, it outputs a script that creates the two above groundings and links them together into a single SAT-TO-SAT specification.

The first step determines the *structure* of the resulting specification without instantiating any quantifications; all instantiations are done by GRINGO-SATGRND .

The current implementation of SO2GROUNDER has some limitations: (a) Step **(1)** above is not possible for all SO theories. For instance in the theory $\forall X : \exists p : p(X)$, the first-order variable X cannot be pushed inside the second-order quantification. Theories of this kind are currently not yet supported. (b) The current implementation using GRINGO-SATGRND is sometimes very slow. (c) The current implementation requires that each first-order quantification needs to be *bound* by some predicate in τ to ensure finite grounding (i.e., to ensure the *safety* of the rules in the GRINGO-SATGRND specification). For instance, a sentence of the form $\forall X : reach(X)$ is not allowed if $reach \notin \tau$, but a sentence $\forall X : in\_node(X) \Rightarrow reach(X)$ is allowed if $in\_node \in \tau$. Developing a dedicated grounder that does not rely on GRINGO-SATGRND is a topic for future work. In this grounder, we can overcome the limitation (a) by interleaving instantiation of first-order variables and detecting the structure of the SAT-TO-SAT specification.

The syntax used in our tool is an ASCII representation of second-order theories, similar to the IDP language (De Cat et al. 2014).

One might wonder whether it is a good idea to use the entire second-order modelling power, which essentially allows to encode problems arbitrary high in the polynomial hierarchy, to tackle (relatively) simple problems, such as for instance NP complete problems. Some remarks are in place here. First of all, in case $\mathcal{T}$ is an *existential second order* ($\exists$SO) theory, the SAT-TO-SAT instance generated by our grounder will simply be a CNF, without any nesting structure. In this case, SAT-TO-SAT is simply a SAT solver, which is the standard for tackling problems in NP. Secondly, if your theory is not an $\exists$SO theory, it might still be beneficial in practice to use the SAT-TO-SAT approach. For certain classes of theories (the exact classes have to be determined in future work, but one example includes the Hamiltonian path theory from Janhunen, Tasharrofi, and Ternovska (2016)), having a nested solver will not increase complexity since the nested solver only needs to perform unit-propagation (no search). Hence in this case, the overall solving algorithm remains in NP. In all other cases, a thorough experimental evaluation should determine whether this approach is beneficial on practical applications or not.

3 Dung's Argumentation Frameworks

Abstract argumentation frameworks (AFs) (Dung 1995) are simple and abstract systems to deal with contentious information and draw conclusions from it. In AFs, we are not interested in the actual content of arguments; this information is abstracted away. In spite of their conceptual simplicity, there exist many different semantics with different properties in terms of characterization, existence, and uniqueness.

An *abstract argumentation framework* Θ is a directed graph (A, R) in which the nodes A represent arguments and the edges in R represent attacks between arguments. We say that a *attacks* b if $(a, b) \in R$. A set $S \subseteq A$ *attacks* a if some $s \in S$ attacks a. A set $S \subseteq A$ *defends* a if it attacks all attackers of a. An *interpretation* of an AF $\Theta = (A, R)$ is a subset S of A. The intended meaning of such an interpretation is that all arguments in S are accepted (or believed) and all arguments not in S are rejected. Interpretations are ordered according to the acceptance relation: $S_1 \leq S_2$ iff $S_1 \subseteq S_2$, i.e., if S_2 accepts more arguments than S_1. There exist many different semantics of AFs that each define different sets of acceptable arguments according to different standards or intuitions. The major semantics for argumentation frameworks can be formulated using two operators: the *characteristic function* F_Θ mapping an interpretation S to

$$F_\Theta(S) = \{a \in A \mid S \text{ defends } a\}$$

and the operator U_Θ (U stands for unattacked) that maps an interpretation S to

$$U_\Theta(S) = \{a \in A \mid a \text{ is not attacked by } S\}.$$

An interpretation S is *conflict-free* if it is a postfixpoint of U_Θ ($S \leq U_\Theta(S)$), i.e., if no argument in S is attacked by S. The characteristic function is a monotone operator; its least fixpoint is called the *grounded extension* of Θ. The operator U_Θ is an anti-monotone operator; its fixpoints are called *stable extensions* of Θ. A *complete extension* is a conflict-free fixpoint of F_Θ. An interpretation is *admissible* if it is a conflict-free postfixpoint of F_Θ. A *preferred extension* is a $\leq$-maximal complete extension. Many more semantics, such as *semi-stable extensions*, *naive extensions*, and *stage extensions* can be characterized in a similar way (Dung 1995; Verheij 1996; Caminada, Carnielli, and Dunne 2012).

3.1 Finding Extensions for Various AF Semantics

We now describe four different semantics of argumentation frameworks in second order logic and hence obtain solvers that perform model expansion for these semantics. Afterwards, we show how *the same* second-order theories can be reused to create solvers that perform other types of inference for argumentation frameworks under the different semantics. Our particular choice of semantics and inference tasks were motivated by the 2015 International Competition on Computational Models of Argumentation (ICCMA 2015).

In this section, we assume that an argumentation framework is represented by two predicates: a unary predicate a, such that $a(X)$ holds iff X is a node and a binary predicate r such that $r(X,Y)$ holds if X attacks Y. We use a unary predicate s to describe the extensions in various different semantics. For instance, the theory $\mathcal{T}_{GR}$ below is such that for any a model I of $\mathcal{T}_{GR}$, the set s^I is a conflict-free extension of (a^I, r^I). This relation is formalized in Theorem 3.2.

First, we provide some second-order theories defining useful concepts in the context of argumentation. Afterwards, we give the theories describing the four semantics. Conflict-freeness of s is expressed by the following theory.

$$\mathcal{T}_{CF} = \{\neg\exists N, M : r(N,M) \land s(N) \land s(M).\}$$

Indeed, this theory simply states that there can be no two nodes in s such that one of them attacks the other. Two useful notions, the nodes attacked and defended by s (denoted *att* and *def*) are defined by the following theory.

$$\mathcal{T}_{AD} = \left\{ \begin{array}{l} \forall N : att(N) \Leftrightarrow (a(N) \land \exists M : r(M,N) \land s(M)). \\ \forall N : def(N) \Leftrightarrow (a(N) \land \forall M : r(M,N) \Rightarrow att(M)). \end{array} \right\}$$

This theory defines att as the set of arguments N such that there is some M in s that attacks M and defines def as the set of arguments N such that s attacks all attackers of N. These two definitions closely follow the informal description of attack and defense as defined earlier. Fixpoints of F_Θ are described by the following theory.

$$\mathcal{T}_{FP} = \{\mathcal{T}_{AD}. \quad \forall N : s(N) \Leftrightarrow def(N).\}$$

This theory indeed states that s equals the set of nodes that are defended by s, i.e., that it is a fixpoint of F_Θ. Recall that we use $\mathcal{T}_{AD}$ as an abbreviation to insert the contents of $\mathcal{T}_{AD}$.

The following theory represents the *grounded semantics*.

$$\mathcal{T}_{GR} = \left\{ \begin{array}{l} \mathcal{T}_{FP}. \\ \neg\exists s', att', def' : \\ \qquad \mathcal{T}_{FP}[s/s', def/def', att/att'] \land \\ \qquad (\forall N : s'(N) \Rightarrow s(N)) \land \\ \qquad (\exists N : s(N) \land \neg s'(N)). \end{array} \right\}$$

In $\mathcal{T}_{GR}$, we demand that s is a fixpoint of F_Θ by including $\mathcal{T}_{FP}$. The expression $\neg\exists s', att', def' \ldots$ expresses that there exists no smaller fixpoint (there is no s' that is a strict subset of s and that is a fixpoint of F_Θ as well).

The *stable semantics* is expressed by the following theory.

$$\mathcal{T}_{ST} = \{ \ \mathcal{T}_{AD}. \quad \forall N : a(N) \Rightarrow (s(N) \Leftrightarrow \neg att(N)). \ \}$$

This theory contains the definitions of att and def and the condition $\forall N : a(N) \Rightarrow (s(N) \Leftrightarrow \neg att(N))$, which expresses that s consists exactly of those nodes not attacked by s, i.e., that s is a fixpoint of U_Θ.

The following theory expresses the *complete semantics*.

$$\mathcal{T}_{CO} = \{\mathcal{T}_{FP}. \quad \mathcal{T}_{CF}.\}$$

By including $\mathcal{T}_{CF}$ and $\mathcal{T}_{FP}$, this theory expresses that s is a conflict-free fixpoint of F_Θ, i.e., a complete extension of Θ.

Finally, the *preferred semantics* is described by the following theory.

$$\mathcal{T}_{PR} = \left\{ \begin{array}{l} \mathcal{T}_{CO}. \\ \neg\exists s', att', def' : \\ \qquad \mathcal{T}_{CO}[s/s', def/def', att/att'] \land \\ \qquad (\forall N : s(N) \Rightarrow s'(N)) \land \\ \qquad \exists N : s'(N) \land \neg s(N). \end{array} \right\}$$

This theory expresses that s is a complete extension and that it is subset-maximal: the formula $\neg\exists s', \ldots$ states that there can be no s' that is a strict superset of s and that is also a complete extension.

Definition 3.1. Let $\mathcal{T}$ be second-order theory with free symbols a, r, and s and let ς be an argumentation semantics.

We say that $\mathcal{T}$ *describes* ς if for each AF (A,R), a structure I with $a^I = A$ and $r^I = R$ is a model of $\mathcal{T}$ if and only if s^I is an extension (according to ς) of (A,R).

Theorem 3.2. *The following hold:*

- $\exists att, def : \mathcal{T}_{GR}$ *describes the grounded semantics,*
- $\exists att, def : \mathcal{T}_{ST}$ *describes the stable semantics,*
- $\exists att, def : \mathcal{T}_{CO}$ *describes the complete semantics, and*
- $\exists att, def : \mathcal{T}_{PR}$ *describes the preferred semantics of AFs.*

3.2 Other Inference Methods for AFs

In Section 2, we discussed how SAT-TO-SAT can perform model expansion to find one (or more) models of a second-order theory. As shown in Theorem 3.2, this boils down to generating extensions for argumentation frameworks if we instantiate SAT-TO-SAT with the above theories. Often, other inference methods are considered for AFs. In this paper, we show how to declaratively build solvers that (in one of the above semantics) perform the following inference tasks: **(1)** finding some extension (SE); **(2)** enumerating all extensions (EE); **(3)** credulous inference, i.e., deciding if an argument x holds in some extension (DC); and **(4)** skeptical inference, i.e., deciding if an argument x holds in all extensions (DS).

If a theory $\mathcal{T}_\varsigma$ that describes some semantics ς is given, we already described how to automatically build solvers that perform inference tasks SE and EE. For DC and DS, we simply note that credulous and skeptical reasoning are respectively equivalent to model expansion for the following theories $\mathcal{T}_\varsigma^{cred}$ and $\mathcal{T}_\varsigma^{skep}$:

$$\mathcal{T}_\varsigma^{cred} = \{\exists s : \mathcal{T}_\varsigma \land s(x).\} \quad \mathcal{T}_\varsigma^{skep} = \{\forall s : \mathcal{T}_\varsigma \Rightarrow s(x).\}$$

Theorem 3.3. *Let ς be an argumentation semantics and $\mathcal{T}_\varsigma$ be a theory describing ς. Let (A,R) be an argumentation framework and $x \in A$ an argument. The argument x is credulously (respectively skeptically) inferred by (A,R) (under semantics ς) if and only if there exists a model of $\mathcal{T}_\varsigma^{cred}$ (respectively $\mathcal{T}_\varsigma^{skep}$) that interprets a as A and r as R.*

Theorem 3.3 shows that both credulous and skeptical reasoning tasks can be easily accomplished in our framework.

SAT-TO-SAT– CoQuiAAS	Some Extension	Enumerate Extensions	Credulous Inference	Skeptical Inference
Complete Semantics	192 – 192	191 – 191	**576** – 575	576 – 576
Preferred Semantics	**192** – 191	**190** – 189	576 – 576	576 – 576
Grounded Semantics	192 – 192	192 – 192	576 – 576	576 – 576
Stable Semantics	192 – 192	**192** – 191	576 – 576	576 – 576

Table 1: Experimental evaluation of SAT-TO-SAT vs CoQuiAAS on all 16 tracks of the latest argumentation competition. The first number is the number of instances solved by SAT-TO-SAT; the second the number of instances solved by CoQuiAAS.

3.3 Experimental Evaluation

We equipped SAT-TO-SAT with 16 ASCII representations of the above theories, resulting in 16 different argumentation solvers (four reasoning tasks for four semantics). We evaluated this approach on all benchmarks used in the latest argumentation competition (ICCMA 2015). This benchmark set contains tests for four semantics: **(1)** complete semantics, **(2)** preferred semantics, **(3)** grounded semantics, and **(4)** stable semantics. For each semantic, the benchmarks contain:

- 192 instances with inference task SE,
- 192 instances with inference task EE,
- 576 instances with inference task DC, and
- 576 instances with inference task DS.

We compared performance of our approach with CoQuiAAS (Lagniez, Lonca, and Mailly 2015), the winner of last years competition. All tests were run on an Intel©Xeon©E5-4652 CPU clocked at 2.70GHz with 260GB of RAM, running Ubuntu 14.04LTS. Tests were run with a time limit of 600 seconds (the same limit as in the competition). Table 1 compares the number of instances solved by SAT-TO-SAT versus CoQuiAAS for the various combination of inference task and semantics.

As Table 1 shows, in 12 out of the 16 settings, SAT-TO-SAT and CoQuiAAS both solve the same number of instances while, in the rest, SAT-TO-SAT solves more instances than the winner of last year's competition. Even though these results suggest that SAT-TO-SAT outperforms CoQuiAAS, we must be careful before making a conclusion for two reasons. First of all, in many settings, both SAT-TO-SAT and CoQuiAAS simply solve all instances, meaning that the test set cannot truly reveal the performance difference between these two solvers. Secondly, when compared to time limits used in other domains such as SAT competitions, the 10 minute time limit is very short and cannot reliably distinguish between the performance of different solvers. We plan to extend our benchmarks to harder instances in the future.

It is worth stressing that SAT-TO-SAT is not designed as a solver for argumentation. With only a few lines of code (an ASCII-representation of theories $\mathcal{T}_{CO}, \mathcal{T}_{PR}, \mathcal{T}_{GR}$, and $\mathcal{T}_{ST}$) we turned it into a solver for argumentation that performs on par with the winner of last year's competition (on the benchmarks used in that competition). This technique can be extended to handle more semantics from abstract argumentation theory and can in principle be used in any domain with a semantics that can be described using some second-order formula. In the future, we plan to also apply this technique to richer formalisms from the field of abstract argumentation, such as Abstract Dialectical Frameworks (Brewka and Woltran 2010).

4 Logic Programming

In this section, we discuss how our results apply to logic programming. We restrict our attention to propositional logic programs. However, the ideas presented here can easily be generalized to richer settings. We show how to construct a stable model generator and a grounded model generator.

Let Σ be a propositional alphabet, i.e., a collection of symbols which are called *atoms*. A *literal* is an atom p or the negation $\neg q$ of an atom q. A logic program $\mathcal{P}$ is a set of *rules* r of the form $h_1 \vee \cdots \vee h_l \leftarrow a_1 \wedge \cdots \wedge a_n \wedge \neg b_1 \wedge \cdots \wedge \neg b_m$, where the h_i's, a_i's, and b_i's are atoms. The formula $h_1 \vee \cdots \vee h_l$ is called the *head* of r, denoted by $head(r)$, and the formula $a_1 \wedge \cdots \wedge a_n \wedge \neg b_1 \wedge \cdots \wedge \neg b_m$ the *body* of r, denoted by $body(r)$. A program is called *normal* if $l = 1$ for all rules in $\mathcal{P}$. An interpretation I of the alphabet Σ is an element of 2^{Σ}, i.e., a subset of Σ.

4.1 Stable Models

An interpretation is a *model* of $\mathcal{P}$ if for each rule r in $\mathcal{P}$, whenever $body(r)$ is satisfied by I, so is $head(r)$. The *reduct* of $\mathcal{P}$ with respect to I, denoted as $\mathcal{P}^I$, is the logic program:

$$\begin{aligned}&\{h_1 \vee \cdots \vee h_l \leftarrow a_1 \wedge \cdots \wedge a_n \mid \\ &\quad h_1 \vee \cdots \vee h_l \leftarrow a_1 \wedge \cdots \wedge a_n \wedge \neg b_1 \wedge \cdots \wedge \neg b_m \in \mathcal{P} \\ &\qquad \wedge\, b_i \notin I \text{ for all } 1 \leq i \leq m\}.\end{aligned}$$

An interpretation I is a *stable model* of $\mathcal{P}$ if it is a minimal model of $\mathcal{P}^I$. If $\tau \subseteq \sigma$ and $\mathcal{P}$ is a logic program such that no atom $a \in \tau$ occurs in the head of a rule in $\mathcal{P}$, we call I a *parameterized stable model* of $\mathcal{P}$ with respect to parameters τ if I is a stable model of $\mathcal{P} \cup (I \cap \tau)$ (Denecker et al. 2012). This semantics generalizes the stable semantics by introducing a set of *parameters* that can have arbitrary values; they are sometimes also called *external atoms* or *open atoms*.

We can describe a logic program by means of predicates r, a, h, pb, and nb with intended interpretation that $r(R)$ holds for all rules with identifier R, $a(A)$ holds for all atoms A, $h(R, H)$ means that H is an atom in the head of rule R, $pb(R, A)$ that A is a positive literal in the body of R and $nb(R, B)$ that B is the atom of some negative literal in the body of R. This reified meta-representation of logic programs is analogous to the one used for example by Gebser et al. (2008). With this vocabulary, augmented with predicates p and i with intended meaning that $p(A)$ holds for all parameters and $i(A)$ holds for all atoms true in some interpretation, we can describe the parameterized stable semantics for (disjunctive) logic programs as follows.

$$\mathcal{T}_{SM} = \left\{ \begin{array}{l} \forall A : i(A) \Rightarrow a(A). \\ \forall R : r(R) \Rightarrow \\ \quad (\forall A : pb(R,A) \Rightarrow i(A)) \\ \quad \wedge (\forall B : nb(R,B) \Rightarrow \neg i(B)) \\ \qquad \Rightarrow (\exists H : h(R,H) \wedge i(H)). \\ \neg\exists i' : \\ \quad (\forall A : i'(A) \Rightarrow i(A)) \wedge \\ \quad (\forall A : p(A) \Rightarrow (i'(A) \Leftrightarrow i(A))) \wedge \\ \quad (\exists A : i(A) \wedge \neg i'(A)) \wedge \\ \quad \forall R : r(R) \Rightarrow \\ \qquad (\forall A : pb(R,A) \Rightarrow i'(A)) \\ \qquad \wedge (\forall B : nb(R,B) \Rightarrow \neg i(B)) \\ \qquad\quad \Rightarrow (\exists H : h(R,H) \wedge i'(H)). \end{array} \right\}$$

The first part of this theory expresses that i is interpreted as a model of $\mathcal{P}$: the constraint $i(A) \Rightarrow a(A)$ expresses that the interpretation is a subset of the vocabulary and the second constraint expresses that whenever the body of a rule is satisfied in i, so is at least one of its head atoms. The constraint $\neg\exists i' \dots$ expresses that i is $\subseteq$-minimal: there cannot be an $i' \subsetneq i$ (that agrees with i on the parameters) that is a model of the reduct of $\mathcal{P}$ with respect to i. In other words, whenever i' satisfies all positive literals in the body of a rule R and i satisfies all negative literals in the body of R, i' must also satisfy some atom in the head of R.

Theorem 4.1. *Let $\mathcal{P}$ be a (disjunctive) logic program and I an interpretation that interprets $\{a, r, pb, nb, h\}$ according to $\mathcal{P}$. Then, $I \models \mathcal{T}_{SM}$ if and only if i^I is a parameterized stable model of $\mathcal{P}$.*

Theorem 4.2 shows that feeding $\mathcal{T}_{SM}$ to SO2GROUNDER results in an answer-set solver for disjunctive logic programs that uses SAT-TO-SAT in the background. The same theory also works for normal logic programs. Many ASP solvers already exist (Syrjänen and Niemelä 2001; Dell'Armi et al. 2001; Gebser, Kaufmann, and Schaub 2012; De Cat et al. 2013; Alviano et al. 2015). Comparing the performance of our solver with existing normal/disjunctive logic programming solvers is a topic for future work.

4.2 Grounded Models

Our methodology can also be used to create solvers for semantics for which no solvers exist yet. One such example is the *grounded model semantics* from Bogaerts, Vennekens, and Denecker (2015). The *immediate consequence operator* $T_{\mathcal{P}}$ of a normal logic program $\mathcal{P}$ (van Emden and Kowalski 1976) maps interpretations to interpretations as follows:

$$T_{\mathcal{P}}(I) = \{p \mid \exists r \in \mathcal{P} : head(r) = p \wedge body(r)^I = \mathbf{t}\}.$$

For normal logic programs, Bogaerts, Vennekens, and Denecker (2015) defined that a set $U \subseteq \Sigma$ is a *2-unfounded set* of $\mathcal{P}$ with respect to interpretation I if $T_{\mathcal{P}}(I \setminus U) \cap U = \emptyset$. Intuitively, a 2-unfounded set is a set of atoms such that if you make them false (starting from I), they stay false (when applying $T_{\mathcal{P}}$). This definition is a slight variant of the notion of unfounded set from Van Gelder, Ross, and Schlipf (1991). Furthermore, they defined a new semantics based on 2-unfounded sets. An interpretation I is a *grounded model* of $\mathcal{P}$ if $T_{\mathcal{P}}(I) = I$ and I does not contain any atoms that belong to a 2-unfounded set U of P with respect to I.[1] Until now, no solvers for grounded model semantics exist. By describing this semantics as the second-order theory $\mathcal{T}_{GM}$ below, we provide the first such solver. We use the same predicates as in our encoding of the stable semantics. We use auxiliary predicates u to encode an unfounded set, tb with intended interpretation that $tb(R)$ holds if R is a rule with body true in i and tb_u with intended interpretation that $tb_u(R)$ holds if R is a rule with body true in $i \setminus u$.

$$\mathcal{T}_{GM} = \left\{ \begin{array}{l} \forall A : i(A) \Rightarrow a(A). \\ \forall R : tb(R) \Leftrightarrow r(R) \wedge \exists B : \\ \qquad (pb(R,B) \wedge i(B)) \vee (nb(R,B) \wedge \neg i(B)). \\ \forall A : i(A) \Leftrightarrow \exists R : tb(R) \wedge h(R,A). \\ \neg\exists u, tb_u : \\ \quad (\exists A : i(A) \wedge u(A)) \wedge \\ \quad (\forall R : tb_u(R) \Leftrightarrow r(R) \wedge \\ \qquad \exists B : (pb(R,B) \wedge i(B) \wedge \neg u(B)) \vee \\ \qquad\qquad (nb(R,B) \wedge (\neg i(B) \vee u(B)))) \wedge \\ \quad \neg\exists A : (\exists R : tb_u(R) \wedge h(R,A)) \wedge u(A). \end{array} \right\}$$

The second sentence in this theory defines tb. The third enforces that i is a fixpoint of $T_{\mathcal{P}}$; the last sentence states that there cannot exist a non-trivial an unfounded set u of $\mathcal{P}$.

Theorem 4.2. *Let $\mathcal{P}$ be a normal logic program and I an interpretation that interprets $\{a, r, pb, nb, h\}$ according to $\mathcal{P}$. Then, $I \models \mathcal{T}_{GM}$ iff i^I is a grounded model of $\mathcal{P}$.*

5 Propositional Logics

In this section, we study logics (both monotonic and non-monotonic) that use propositional logic as their syntax. We describe their semantics in second-order logic, and thus, obtain solvers for all these formalisms.

5.1 Classical Propositional logic

A formula φ over vocabulary σ is represented using an interpretation of a vocabulary τ_m containing predicates $atom$, and, or, imp, neg, and $sentence$. The domain of this interpretation consists of all atoms in σ and one identifier for each subformula of φ. The intended interpretation of the predicates is as follows **(1)** $atom$ is a unary predicate that holds for all propositional atoms; **(2)** and (respectively or, imp, and neg) is a predicate such that $and(F, G_1, G_2)$ (resp. $or(F, G_1, G_2)$, $imp(F, G_1, G_2)$, and $neg(F, G)$) means that F is the identifier for conjunctive formula $G_1 \wedge G_2$ (resp. disjunctive formula $G_1 \vee G_2$, implicative formula $G_1 \Rightarrow G_2$ and negative formula $\neg G$) and, **(3)** finally, the identifier for the sentence φ is singled out by a unary predicate $sentence$.

Example 5.1. Let φ be the formula $(\neg p \rightarrow q) \wedge (\neg q \rightarrow p)$. The reified form of φ is the interpretation I with domain $\{p, q, 1, 2, 3, 4, 5\}$ satisfying

[1]To be precise, they defined grounded models as an instantiation of the abstract algebraical theory of *grounded fixpoints*. The characterization of grounded models we use here is based on Corollary 5.6 from Bogaerts, Vennekens, and Denecker (2015).

$atom^I = \{p, q\}$, $neg^I = \{(1, p), (3, q)\}$, $imp^I = \{(2, 1, q), (4, 3, p)\}$, $and^I = \{(5, 2, 4)\}$, and $sentence^I = \{5\}$.

The (classical) satisfaction relation $\models$ between interpretations and formulas is defined as usual. It can be represented by the following theory over vocabulary $\tau_c := \tau_m \cup \{i, s\}$, where i is a predicate that encodes an interpretation and s holds for all subformulas ψ of φ such that $i \models \psi$.

$$\mathcal{T}_C = \left\{ \begin{array}{l} \forall A : atom(A) \Rightarrow (s(A) \Leftrightarrow i(A)). \\ \forall F, G : neg(F, G) \Rightarrow (s(F) \Leftrightarrow \neg s(G)). \\ \forall F, G_1, G_2 : and(F, G_1, G_2) \Rightarrow \\ \quad (s(F) \Leftrightarrow s(G_1) \wedge s(G_2)). \\ \forall F, G_1, G_2 : or(F, G_1, G_2) \Rightarrow \\ \quad (s(F) \Leftrightarrow s(G_1) \vee s(G_2)). \\ \forall F, G_1, G_2 : imp(F, G_1, G_2) \Rightarrow \\ \quad (s(F) \Leftrightarrow (s(G_1) \Rightarrow t(G_2))). \\ \forall F : sentence(F) \Rightarrow s(F). \end{array} \right\}$$

This theory simply defines satisfaction (the relation s) by the standard recursive rules of propositional logic.

Theorem 5.2. *Let φ be a propositional formula over σ and I_φ its reified representation. Let J be a $\tau_m \cup \{i\}$-structure that equals I_φ on all symbols except i. Then J is a model of $\exists s : \mathcal{T}_C$ if and only if i^J is a model of φ.*

5.2 The Logic of Here and There

The logic of here and there is a non-classical, intuitionistic logic, first defined by Heyting (1930). In this logic, the semantics of a propositional formula φ over σ is given by two sets of atoms H and T (respectively called "here" and "there") with $H \subseteq T \subseteq \sigma$. HT-satisfiability, denoted by $\langle H, T\rangle \models_{HT} \varphi$, is defined recursively as follows:

- $\langle H, T\rangle \models_{HT} a$ if $a \in H$;
- $\langle H, T\rangle \models_{HT} F \wedge G$ (resp. $\langle H, T\rangle \models_{HT} F \vee G$) if $\langle H, T\rangle \models_{HT} F$ and (resp. or) $\langle H, T\rangle \models_{HT} G$;
- $\langle H, T\rangle \models_{HT} \neg F$ if $T \not\models F$; and,
- $\langle H, T\rangle \models_{HT} F \Rightarrow G$ if **(1)** $T \models F \Rightarrow G$ and **(2)** either $\langle H, T\rangle \not\models_{HT} F$ or $\langle H, T\rangle \models_{HT} G$.

The following first-order theory describes HT-satisfiability for propositional theories over vocabulary $\tau_{ht} := \tau_m \cup \{h, t\}$. It makes use of two predicates h and t to encode the pair $\langle H, T\rangle$, a predicate s to encode classical satisfiability in the interpretation T and a predicate ht to encode all formulas satisfied in the pair $\langle H, T\rangle$.

$$\mathcal{T}_{HT} = \left\{ \begin{array}{l} \mathcal{T}_C[i/t]. \\ \forall A : atom(A) \wedge h(A) \Rightarrow t(A). \\ \forall A : atom(A) \Rightarrow (ht(A) \Leftrightarrow \neg s(A)). \\ \forall F, G_1, G_2 : and(F, G_1, G_2) \Rightarrow \\ \quad (ht(F) \Leftrightarrow ht(G_1) \wedge ht(G_2)). \\ \forall F, G_1, G_2 : or(F, G_1, G_2) \Rightarrow \\ \quad (ht(F) \Leftrightarrow ht(G_1) \vee ht(G_2)). \\ \forall F, G : neg(F, G) \Rightarrow (ht(F) \Leftrightarrow \neg s(F)). \\ \forall F, G_1, G_2 : imp(F, G_1, G_2) \Rightarrow \\ \quad (ht(F) \Leftrightarrow s(F) \wedge (ht(G_1) \Rightarrow ht(G_2))). \\ \forall F : sentence(F) \Rightarrow ht(F). \end{array} \right\}$$

This theory contains the definition of classical satisfaction and the condition that $h \subseteq t$. The sentences that follow are straightforward translations of the recursive rules in the definition of the HT-semantics.

Theorem 5.3. *Let φ be a propositional formula over σ and I_φ its reified representation. Let J be a τ_{ht}-structure that equals I_φ on all symbols except h and t. Then J is a model of $\exists s, ht : \mathcal{T}_{HT}$ if and only if $\langle h^J, t^J\rangle \models_{HT} \varphi$.*

5.3 Equilibrium Logic

Equilibrium logic (Pearce 2000) is a semantics for propositional theories that extends the stable model semantics for logic programs. The equilibrium models of a propositional theory $\mathcal{T}$ are defined as those interpretations I so that **(1)** I is a classical model of $\mathcal{T}$ and **(2)** there does not exist any proper subset I' of I so that $\langle I', I\rangle$ is an HT-model of $\mathcal{T}$. Equilibrium logic is described by the following τ_c-theory.

$$\mathcal{T}_{EQ} = \left\{ \begin{array}{l} \mathcal{T}_C. \\ \neg\exists h : \mathcal{T}_{HT}[t/i] \wedge \\ \qquad (\exists A : atom(A) \wedge \neg h(A) \wedge i(A)). \end{array} \right\}$$

In $\mathcal{T}_{EQ}$, we assert that i satisfies φ classically and that there cannot exist an HT-model $\langle h, i\rangle$ of φ in which h is a proper subset of i. As such, with this theory, we obtain a solver for equilibrium logic. Contrary to earlier work (Cabalar, Pearce, and Valverde 2005; Pearce, Tompits, and Woltran 2009), our solver is obtained purely by a declarative description of the equilibrium model semantics.

Theorem 5.4. *Let φ be a propositional formula over σ and I_φ be φ's reified representation. Let J be a $\tau_m \cup \{i\}$-structure that equals I_φ except on i. Then, $J \models \exists s : \mathcal{T}_{EQ}$ if and only if i^J is an equilibrium model of φ.*

5.4 Supported Models of Propositional Formulas

Tasharrofi (2013) presented a semantics known as *supported semantics* for propositional formulas. Despite what its name suggests, this semantics is also an extension of the stable semantics of logic programs, in the sense that it respects the principles of minimality and well-justifiability from Gelfond and Lifschitz (1990). Informally, this extension defines S to be a *supported model* of propositional formula φ if φ combined with the negation of all atoms false in S entails each atom in S using the rules of intuitionistic logic. As a consequence of this definition, true atoms are *well-justified* because they are provable independently from other true atoms in S.

Tasharrofi (2013) also characterized supported models using HT-models as follows: an interpretation I is a supported model of propositional formula φ if and only if $\langle I, I\rangle$ is the unique HT-model of formula $\bigwedge\{\neg x \mid x \text{ is an atom and } x \notin I\} \wedge \varphi$. This is equivalent to $\langle I, I\rangle$ being the unique HT-model of ϕ with $T \subseteq I$. Using such a characterization of supported models, supported model semantics is easily formulated in second order logic by the following τ_c-theory.

$$\mathcal{T}_{Sup} = \left\{ \begin{array}{l} \mathcal{T}_C. \\ \neg\exists h, t : \mathcal{T}_{HT} \wedge \\ \quad (\forall A : atom(A) \Rightarrow (t(A) \Rightarrow i(A))) \wedge \\ \quad (\exists A : atom(A) \wedge \neg h(A) \wedge i(A)). \end{array} \right\}$$

In this theory, we require that **(1)** s is a classical model of φ (hence, $\langle s, s\rangle$ is an HT-model of φ), and **(2)** no other HT-model $\langle h, t\rangle$ of φ with $t \subsetneq i$ exists.

Using this theory, we automatically obtain the first solver for propositional formulas under supported model semantics. It is also noteworthy that $\mathcal{T}_{EQ}$ and $\mathcal{T}_{Sup}$ are strikingly similar. The only difference is that in $\mathcal{T}_{Sup}$, t can be a proper subset of i while, in $\mathcal{T}_{EQ}$, t equals i). This similarity works as further evidence that our method for automatic generation of solvers is the right way to produce new solvers. Intuitively, solvers for two slightly different semantics (both are generalisations of the stable semantics to propositional logic) should not be that different from each other.

Theorem 5.5. *Let φ be a propositional formula over σ and I_φ be φ's reified representation. Let J be a $\tau_m \cup \{i\}$-structure that equals I_φ except on i and s. Then, $J \models \exists s : \mathcal{T}_{Sup}$ if and only if i^J is a supported model of φ.*

Using Theorem 5.5, we immediately obtain the first solver for supported model semantics.

5.5 Parallel Circumscription

Lifschitz (1985) defined theories subject to parallel circumscription as triples $\langle P, F, T\rangle$ in which T is a propositional theory and P and F are two disjoint sets of atoms. A structure I is a *$\langle P, F\rangle$-minimal model* of T if **(1)** I is a classical model of T and **(2)** there exists no $I' \subseteq I$ such that I' is a classical model of T, $I' \cap P = I \cap P$, and I' assigns less atoms in P to true than I does, i.e., $I' \cap P \subsetneq I \cap P$. Note that the symbols *not* in $P \cup F$ can *vary* freely.

Assuming that, T is represented using the reification method as before, and predicates p and f are used to denote sets P and F of atoms, the semantics of parallel circumscription is easily described in second-order logic as follows.

$$\mathcal{T}_{PC} = \left\{ \begin{array}{l} \mathcal{T}_C. \\ \neg\exists i', s' : \\ \quad (\forall A : f(A) \Rightarrow (i'(A) \Leftrightarrow i(A)))\wedge \\ \quad (\forall A : p(A) \Rightarrow (i'(A) \Rightarrow i(A)))\wedge \\ \quad (\exists A : p(A) \wedge \neg i'(A) \wedge i(A))\wedge \\ \quad \mathcal{T}_C[i/i', s/s']. \end{array} \right\}$$

In this theory; we assert indeed that i is a classical model of T, while being minimal in the sense of circumscription, i.e., there exists no i' so that i and i' coincide on f and i contains more true atoms from p than i' does. From $\mathcal{T}_{PC}$, we automatically obtain a solver for parallel circumscription (cfr. Theorem 5.6 below). This is not the first such solver, e.g., Janhunen and Oikarinen (2004) have created a solver by translating parallel circumscription into disjunctive logic programs.

Theorem 5.6. *Let $\langle P, F, T\rangle$ be a theory subject to circumscription and I its reified representation. Let J be a $\tau_m \cup \{i, p, f\}$-structure that equals I_φ except on i. Then, $J \models \mathcal{T}_{PC}$ if and only if i^J is a $\langle P, F\rangle$-minimal model of T.*

6 Conclusion and Future Work

In this paper, we presented a generic methodology for declarative solver development, where solvers for a logical formalism are generated solely based on a second-order specification of the semantics. We showed empirically that this methodology is applicable in a wide range of logics by **(1)** presenting solvers for formalisms for which solving technology was already available, such as argumentation frameworks, answer set programming, and various semantics for propositional theories, **(2)** presenting solvers for formalisms for which no solving technology existed yet, such as the grounded model semantics for logic programming and the supported model semantics for propositional logic. We showed that this methodology is very flexible: small changes in semantics or inference task to perform, result in small changes in the declarative specification. We showed that the ease with which solvers can be developed in this methodology is not always at the expense of efficiency: in the context of argumentation frameworks, we obtained a level of performance comparable to the state of the art.

We identify several major directions for future work. **(1)** We will search for more *application domains*. We presume this formalism to be applicable to a wide range of logics, and intend to declaratively implement solvers for those logics. **(2)** We will extend the *modelling language* of our solver with useful concepts. For example, to naturally model semantics of logic programs with aggregates (Ferraris 2005; Son, Pontelli, and Elkabani 2006; Pelov, Denecker, and Bruynooghe 2007; Faber, Pfeifer, and Leone 2011; Gelfond and Zhang 2014), a notion of *aggregate* in the language would be needed. In order to integrate semantics from *constraint programming*, integer functions and arithmetic would be key concepts to add to our logic. **(3)** We will *benchmark* our solver on more application domains to test performance and if possible improve *efficiency* of SAT-TO-SAT. **(4)** The methodology we present here is purely declarative. Even though in some cases efficient solvers are generated, we cannot expect this to always be the case. In such situations, a developer should not fall back to the imperative methods immediately. In the future, we plan, besides the declarative specifications, to allow the user to specify certain notions of *control*. For example, Janhunen, Tasharrofi, and Ternovska (2016) already explained how one can specify which clauses a nested SAT solver should learn. Additionally, one might want to control different options, in the different nested solvers, or the used search strategy. In the long term, our goal is to provide a methodology that captures an entire spectrum between *declarative* and *imperative* solver development methods, i.e., where a developer chooses at which level he/she wants to control the execution of the solver and where the implementation of other parts is generated automatically from the declarative specification of the semantics. **(5)** We will implement a native version of SO2GROUNDER that does not rely on tools such as GRINGO-SATGRND .

Acknowledgments

The support from the Finnish Center of Excellence in Computational Inference Research (COIN) funded by the Academy of Finland (under grant #251170) is gratefully acknowledged.

References

Alviano, M.; Dodaro, C.; Leone, N.; and Ricca, F. 2015. Advances in WASP. In *Proceedings of LPNMR*, volume 9345 of *LNCS*, 40–54.

Apt, K. R. 2003. *Principles of Constraint Programming*. Cambridge University Press.

Barrett, C. W.; Sebastiani, R.; Seshia, S. A.; and Tinelli, C. 2009. Satisfiability modulo theories. In *Handbook of Satisfiability*, volume 185 of *FAIA*. IOS Press. 825–885.

Bayless, S.; Bayless, N.; Hoos, H. H.; and Hu, A. J. 2015. SAT modulo monotonic theories. In *Proceedings of AAAI*, 3702–3709.

Bogaerts, B.; Janhunen, T.; and Tasharrofi, S. 2016. Solving QBF instances with nested SAT solvers. In *Proceedings of Beyond NP*.

Bogaerts, B.; Vennekens, J.; and Denecker, M. 2015. Grounded fixpoints and their applications in knowledge representation. *Artif. Intell.* 224:51–71.

Brewka, G., and Woltran, S. 2010. Abstract dialectical frameworks. In *Proceedings of KR*, 102–111.

Cabalar, P.; Pearce, D.; and Valverde, A. 2005. Reducing propositional theories in equilibrium logic to logic programs. In *Proceedings of EPIA*, volume 3808 of *LNCS*, 4–17.

Caminada, M. W. A.; Carnielli, W. A.; and Dunne, P. E. 2012. Semi-stable semantics. *J. Log. Comput.* 22(5):1207–1254.

De Cat, B.; Bogaerts, B.; Devriendt, J.; and Denecker, M. 2013. Model expansion in the presence of function symbols using constraint programming. In *Proceedings of ICTAI*, 1068–1075.

De Cat, B.; Bogaerts, B.; Bruynooghe, M.; and Denecker, M. 2014. Predicate logic as a modelling language: The IDP system. *CoRR* abs/1401.6312.

Dell'Armi, T.; Faber, W.; Ielpa, G.; Koch, C.; Leone, N.; Perri, S.; and Pfeifer, G. 2001. System description: Dlv. In *Proceedings of LPNMR*, volume 2173 of *LNCS*, 424–428.

Denecker, M.; Lierler, Y.; Truszczyński, M.; and Vennekens, J. 2012. A Tarskian informal semantics for answer set programming. In *Proceedings of ICLP*, volume 17 of *LIPIcs*, 277–289.

Dung, P. M. 1995. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artif. Intell.* 77(2):321 – 357.

Eén, N., and Sörensson, N. 2004. An extensible SAT-solver. In *Proceedings of SAT*, volume 2919 of *LNCS*, 502–518.

Faber, W.; Pfeifer, G.; and Leone, N. 2011. Semantics and complexity of recursive aggregates in answer set programming. *Artif. Intell.* 175(1):278–298.

Ferraris, P. 2005. Answer sets for propositional theories. In *Proceedings of LPNMR*, 119–131.

Ganzinger, H.; Hagen, G.; Nieuwenhuis, R.; Oliveras, A.; and Tinelli, C. 2004. DPLL(*T*): Fast decision procedures. In *Proceedings of CAV*, volume 3114 of *LNCS*, 175–188.

Gebser, M.; Pührer, J.; Schaub, T.; and Tompits, H. 2008. A meta-programming technique for debugging answer-set programs. In *Proceedings of AAAI*, 448–453.

Gebser, M.; Janhunen, T.; Kaminski, R.; Schaub, T.; and Tasharrofi, S. 2015. Writing declarative specifications for clauses. In *Proceedings of GTTV*.

Gebser, M.; Janhunen, T.; and Rintanen, J. 2014. SAT modulo graphs: Acyclicity. In *Proceedings of JELIA*, volume 8761 of *LNCS*, 137–151.

Gebser, M.; Kaufmann, B.; and Schaub, T. 2012. Conflict-driven answer set solving: From theory to practice. *Artif. Intell.* 187:52–89.

Gelfond, M., and Lifschitz, V. 1988. The stable model semantics for logic programming. In *Proceedings of ICLP*, 1070–1080.

Gelfond, M., and Lifschitz, V. 1990. Logic programs with classical negation. In *Proceedings of ICLP*, 579–597.

Gelfond, M., and Zhang, Y. 2014. Vicious circle principle and logic programs with aggregates. *TPLP* 14(4-5):587–601.

Heyting, A. 1930. Die formalen regeln der intuitionistischen logik. *Sitzungsberichte der Preussischen Akademie der Wissenschaften (16/1/1930)* 42–56.

ICCMA. 2015. International competition on computational models of argumentation. `http://argumentationcompetition.org/2015/index.html`.

Janhunen, T., and Oikarinen, E. 2004. Capturing parallel circumscription with disjunctive logic programs. In *Proceedings of JELIA*, volume 3229 of *LNCS*, 134–146.

Janhunen, T.; Tasharrofi, S.; and Ternovska, E. 2016. SAT-TO-SAT: Declarative extension of SAT solvers with new propagators. In *Proceedings of AAAI*.

Lagniez, J.; Lonca, E.; and Mailly, J. 2015. Coquiaas: A constraint-based quick abstract argumentation solver. In *Proceedings of ICTAI*, 928–935.

Lifschitz, V. 1985. Computing circumscription. In *Proceedings of IJCAI*, 121–127.

Marques-Silva, J. P., and Sakallah, K. A. 1999. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers* 48(5):506–521.

Nadel, A., and Ryvchin, V. 2012. Efficient SAT solving under assumptions. In *Proceedings of SAT*, volume 7317 of *LNCS*, 242–255.

Ohrimenko, O.; Stuckey, P. J.; and Codish, M. 2009. Propagation via lazy clause generation. *Constraints* 14(3):357–391.

Pearce, D.; Tompits, H.; and Woltran, S. 2009. Characterising equilibrium logic and nested logic programs: Reductions and complexity' . *TPLP* 9(5):565–616.

Pearce, D. 2000. Equilibrium logic: An extension of answer set programming for nonmonotonic reasoning. In *Proceedings of WLP*, 17.

Pelov, N.; Denecker, M.; and Bruynooghe, M. 2007. Well-founded and stable semantics of logic programs with aggregates. *TPLP* 7(3):301–353.

Silva, J. P. M.; Lynce, I.; and Malik, S. 2009. Conflict-driven clause learning SAT solvers. In *Handbook of Satisfiability*, volume 185 of *FAIA*. IOS Press. 131–153.

Son, T. C.; Pontelli, E.; and Elkabani, I. 2006. An unfolding-based semantics for logic programming with aggregates. *CoRR* abs/cs/0605038.

Syrjänen, T., and Niemelä, I. 2001. The smodels system. In *Proceedings of LPNMR*, volume 2173 of *LNCS*, 434–438.

Tasharrofi, S. 2013. A rational extension of stable model semantics to the full propositional language. In *Proceedings of IJCAI*.

van Emden, M. H., and Kowalski, R. A. 1976. The semantics of predicate logic as a programming language. *J. ACM* 23(4):733–742.

Van Gelder, A.; Ross, K. A.; and Schlipf, J. S. 1991. The well-founded semantics for general logic programs. *J. ACM* 38(3):620–650.

Verheij, B. 1996. Two approaches to dialectical argumentation: Admissible sets and argumentation stages. In *Proceedings of FAPR*, 357–368.