SAT Modulo Well-Founded Semantics

Thomas Eiter
TU Wien, Austria

Tobias Nießen
TU Wien, Austria

Davide Soldà
TU Wien, Austria

Abstract

The well-founded semantics (WFS) for logic programs yields a unique three-valued model that serves as an efficient core for skeptical reasoning, but lacks built-in mechanisms for choice and case-based reasoning, limiting its expressiveness for problems such as decision making and planning. Propositional SAT solvers excel at combinatorial problems like the latter but, unlike WFS, do not naturally support reasoning under incomplete information or encoding transitive closure properties. We present an integration of a choice operator into WFS that preserves the suitability of the semantics for scalable, partial-information reasoning. From a propositional perspective, our semantics gracefully captures semantically unassigned atoms and constraints; we illustrate this approach in a setting for reasoning about actions under uncertainty. Furthermore, classical propositional satisfiability can not only be embedded into our framework, but now also be extended with reasoning over transitive closures. In terms of program evaluation, we show that the choice operator can be materialized by a SAT solver while propagating the consequences of choices through an extension of the alternating fixpoint algorithm for WFS with conflicts that are propagated back to the SAT solver. To further increase computational performance, we develop clause learning and syntactic decomposition techniques for logic programs with choices.

2012 ACM Subject Classification Theory of computation → Constraint and logic programming; Theory of computation → Automated reasoning; Mathematics of computing → Solvers; Computing methodologies → Nonmonotonic, default reasoning and belief revision; Computing methodologies → Logic programming and answer set programming; Computing methodologies → Reasoning about belief and knowledge; Computing methodologies → Planning under uncertainty

Keywords and phrases Well-Founded Semantics, SAT, Satisfiability, Logic Programming, Least Fixpoint Computation

Digital Object Identifier 10.4230/LIPIcs.SAT.2026.15

Supplementary Material *Software (Source Code)*: `https://doi.org/10.5281/zenodo.20122760` [29]

Funding This project has received funding from the European Research Council under the European Union's Horizon 2020 research and innovation programme. Grant agreement No. 101034440. Eiter and Soldà are partially supported by the MOSAIC project (EU H2020-MSCA-RISE-2020 Project 101007627) and by the WWTF project ICT22-023.

1 Introduction

The well-founded semantics (WFS) [36] provides a three-valued semantics for logic programs with negation that for ground (propositional) programs is computable in polynomial time, e.g., by an alternating fixpoint computation. However, its skeptical nature offers only limited support for decision making, as WFS does not explicitly branch reasoning into alternative models. A possible way to overcome this limitation is disjunction in rule heads; however, despite several proposed approaches [1, 9, 17, 57, 62], no broadly accepted solution has emerged. As an alternative that allows us to capture branching (i.e., nondeterminism)

in a semantically principled manner, we propose an integration of WFS with the *choice* construct [50], whose interpretation under well-founded semantics seems less controversial. Notably, choice is conceptually different from undefinedness: the former captures controllable nondeterminism, while the latter can be interpreted as epistemic uncertainty.

The integration can be dually viewed as a method for enriching SAT (and, more generally, SAT with first-order variables and Herbrand interpretations) with a principled form of least fixpoint reasoning in an SMT-like fashion. In particular, the well-founded semantics yields a three-valued approximation of inductive definitions such as reachability and transitive closure, which are notoriously difficult to express in classical SAT/SMT without resorting to ad hoc encodings, bounded unfolding, or intricate axiomatizations.

In this paper, we introduce *extended logic programs with choice* (ELP^{C}) by augmenting the syntax of extended logic programs (ELP) with an explicit choice operator. We define the semantics of such programs by extending WFSX [2], i.e., well-founded semantics with explicit negation. We show how the combination of the choice construct and explicit negation enables reasoning about actions in a partial information setting, where distinguishing choice and undefinedness reflects the classical separation of optional actions and incomplete states.

For program evaluation, we present an integration of choices into an WFSX evaluation algorithm via calls to a SAT solver, which paves the way to a notion of WFSX modulo theories. Our framework permits grounding *on demand* during the well-founded derivation, so no theoretical initial guessing or full instantiation of the Herbrand base is required. This contrasts with ASP, as stable-model semantics relies on early commitments through global assumptions on negation by default, even though lazy grounding techniques have been proposed [53]. Fortunately, WFS is known to behave favorably with respect to grounding, and is often employed to provide semantic guidance for constraining and simplifying the grounding process [42].

We further develop two optimization techniques for evaluating ELP^{C} programs, namely, *clause learning* and *program decomposition*. For the former, we present a procedure inspired by the SLX procedure [3] for a relaxation of WFSX in which contradictory conclusions are allowed, to learn from contradictions and violations of integrity constraints. For program decomposition, we build on program splitting [44] and stratified subprograms, structuring the input program into an alternation of stratified and non-stratified components, which can be exploited operationally. The stratified parts admit monotone fixpoint constructions which need no initial "optimistic" interpretation containing all potentially derivable atoms; as a result, the effective Herbrand base is reduced. By alternating the computation with the well-founded evaluation of the remaining components, substantial grounding can be avoided. Notably, there is a unique program splitting that maximizes the stratified layers and caters for more efficient fixpoint computation.

Finally, we show how to represent arbitrary SAT formulas as ELP^{C} programs and that the resulting semantics reconstructs exactly their intended models. We obtain in this way a *SAT-modulo-fixpoint* mechanism in which recursive definitions coexist with Boolean connectives in a uniform and semantically grounded manner. Viewed from this angle, our work contributes to the long-standing challenge of integrating inductive definitions and transitive closure into SMT solving, a problem that motivated several extensions such as μZ [40] and approaches based on Constrained Horn Clauses [6, 39], which however are two-valued and have different fixpoint semantics. In [48] an implementation of SAT(ID) was reported, where SAT was extended with inductive definitions, in the style of FO(ID) [22]. Furthermore, as our semantics assigns each classical literal a well-defined three-valued truth status, it naturally mirrors the partial assignments used internally by SAT/SMT solvers, enabling a model-construction process that integrates Boolean propagation coherently with fixpoint reasoning.

2 Related Work

The well-founded semantics is well-established in the areas of logic programming and knowledge representation. Starting from its introduction [36], it has been studied and extended in order to address different challenges, for example, through an integration with first-order theories by extending rule bodies with formulas of a given external theory [23], through an extension with monotone and antimonotone aggregates [4], or through a combination of the well-founded semantics with description logic [26, 43]. From a computational point of view, we recall the top-down XSB system [58], which is closely related to the top-down query-driven ASP system s(CASP) [7]. A noteworthy approach similar to ours is the one of LDL++ [64], which considers a syntactic, restricted notion of default negation and aggregates, along with certain notions of choices.

SAT solvers have been extensively used in the context of logic programming, in particular by exploiting the characterization of answer set in terms of *loop-formulas* and *completion* [46]. Tools that have been designed around this idea include the solvers *cmodels* [38], *sag* [47], *clasp* [34], *smodels*$_{cc}$ [63], and *wasp* [5]. This characterization has recently also been employed for certifying the decision of ASP solvers using a QBF encoding to accommodate disjunctive programs [20]. In our work, we propose a procedure for learning propositional formulas from conflicts encountered by the well-founded model construction that the SAT solver is fed with, which differs from the classical loop and completion formulas in ASP as it focuses on learning formulas that prevent choices that could lead to a contradiction. Notably, if our framework is applied exhaustively, we can compile the set of choices that generate well-founded models into a classical propositional formula.

A common argument for ASP solving is that SAT solving cannot naturally handle acyclicity/reachability properties well. Hence, there is a growing interest in developing techniques for handling such constraints within SAT. For example, approaches based on custom SAT constraint propagators for acyclicity constraints have beenproposed [32]. Other solving techniques for the same problem use a vertex elimination graph, and resort to an instance-dependent encoding into SAT [56]. These techniques have been fruitfully applied to encode of ASP into SAT modulo acyclicity [31]. A recent study [51] compares two possible implementations of assumption-based argumentation (ABA) reasoning, comparing the two techniques above for solving SAT under acyclicity constraints. In the context of planning, existing approaches combine MaxSAT and a cycle elimination technique [55]. Acyclicity constraints have been proven to be useful also in the context of analyzing the propagation of faults of components in the presence of circular dependencies [13].

When compared to ASP, a main difference of our approach is that it allows precise control over an ASP-style total choice versus leaving literals undefined in the fashion of the well-founded semantics. This corresponds to the *0-approximation* principle in reasoning about actions [8], where undefinedness represents explicitly the information that cannot be derived without case-based reasoning. More specifically, our framework can represent and reason about actions in settings with incomplete information. In particular, incomplete information is not necessarily modeled epistemically with a set of possible states, but in the style of [24, 60], where the values of some fluents, i.e., state variables, are considered to be unknown and no case-base reasoning is involved. Closely related to this setting are approaches that encode incomplete information into a classical planner [52].

3 Preliminaries

We recall relevant concepts and definitions from previous works on well-founded semantics.

3.1 Well-Founded Semantics with Explicit Negation

We recall the syntax of logic programs with explicit negation [37], followed by the definition of the well-founded semantics for such programs [2, 35, 54].

We assume a signature (set) $\mathcal{P}$ of atoms, and define *objective literals* as $\mathcal{L} = \{p, \neg p \mid p \in \mathcal{P}\}$ and *default literals* as $\mathit{df}\mathcal{L} = \{L, \mathrm{not}\, L \mid L \in \mathcal{L}\}$; here, "¬" stands for explicit (strong, classical) negation and "not" for default (weak) negation. These two kinds of negation are a well-known key feature in knowledge representation; for example, a rule of the form "if a driver does *not* have evidence that a train is *not* approaching, they should wait" can be captured by the rule $\texttt{wait} \leftarrow \mathrm{not}\, \neg\texttt{train}.$, which is more cautious than $\texttt{wait} \leftarrow \texttt{train}$.

▶ **Definition 1** (Extended Normal Logic Program [37]). *An* extended normal logic program π *(*ELP*, or simply a* logic program*) is a set of rules of the form:*

$$r \;:\; B_0 \leftarrow B_1, \ldots B_n, \mathrm{not}\, B_{n+1}, \ldots \mathrm{not}\, B_m. \qquad n, m \geq 0, \qquad B_j \in \mathcal{L} \;\; \textit{for}\; 0 \leq j \leq m.$$

We denote by $H(r) = B_0$ *the head and by* $B(r) = B^+(r) \cup B^-(r)$ *the body of* r*, where* $B^+(r) = \{B_1, \ldots, B_n\}$ *and* $B^-(r) = \{B_{n+1}, \ldots, B_m\}$*; if* $B(r) = \emptyset$*,* r *is a* fact*.*

We also consider logic programs with function-free first-order literals, where rules are treated as abbreviations for ground instances. As is common, we assume *safe negation*, i.e., each variable in a rule r must occur in $B^+(r)$ [25].

An *interpretation* for $\mathcal{P}$ is a set $I \subseteq \mathit{df}\mathcal{L}$ of default literals; I is consistent if $\{ L, \mathrm{not}\, L \} \not\subseteq I$ for every $L \in \mathcal{L}$ and if I satisfies the *coherence principle*, meaning for each objective literal L, $\neg L \in I$ entails $\mathrm{not}\, L \in I$, where $\neg\neg L$ stands for L. We call L and $\neg L$ complements to each other. An interpretation is *total*, if for each $p \in \mathcal{P}$, either $p \in I, \neg p \in I$, or $\{\mathrm{not}\, p, \mathrm{not}\, \neg p\} \subseteq I$.

For any not-free program π, we denote by $Cn(\pi)$ the $\subseteq$-minimal set of objective literals s.t., for each rule $r \in \pi$, if $B(r) \subseteq Cn(\pi)$, then $H(r) \in Cn(\pi)$. Note that $Cn(\pi)$ may contain p and $\neg p$, but we do *not* enforce explosion (i.e., $Cn(\pi) = \mathcal{L}$) in this case.

We follow the common definition of the semantics based on a program reduct and semi-normal programs to implement the coherence principle [2], but deviate insofar that we confine ourselves to consistent models. As an intuitive example consider the following program.

▶ **Example 2.** Consider $\pi = \{\texttt{p} \leftarrow \mathrm{not}\, \neg\texttt{p}. \quad \neg\texttt{p} \leftarrow \mathrm{not}\, \texttt{p}. \quad \texttt{u} \leftarrow \texttt{p}. \quad \texttt{u} \leftarrow \neg\texttt{p}.\}$. There is no clear evidence to derive either $\texttt{p}$ or $\neg\texttt{p}$, and the undefinedness of $\texttt{p}$ and $\neg\texttt{p}$ propagates to $\texttt{u}$. Hence, the well-founded model [2] is $S = \emptyset$. When adding $\texttt{p}$ as a fact, i.e., $\pi' = \pi \cup \{\, \texttt{p}. \,\}$, the well-founded model becomes $S' = \{\, \texttt{u}, \texttt{p}, \mathrm{not}\, \neg\texttt{p}, \mathrm{not}\, \neg\texttt{u} \,\}$.

We say that $I \not\models \bot$, $I \models \top$, $I \models l$ for $l \in \mathcal{L}$ if $p \in I$, and $I \models \mathrm{not}\, l$ for $l \in \mathcal{L}$ if $l \notin I$.

▶ **Definition 3** (GL reduct). *Let* π *be a logic program and* I *a consistent set of objective literals. The GL-reduct of* π *modulo* I *is* $\pi^I = \{H(r) \leftarrow B^+(r) \mid I \models \mathrm{not}\, b \;\textit{for}\; b \in B^-(r), r \in \pi\}$.

▶ **Definition 4** (Semi-normal Program). *The semi-normal version* π_s *of a program* π *is given by* $\pi_s = \{\, r_s : \; H(r) \;\leftarrow B(r), \mathrm{not}\, \neg H(r) \mid r \in \pi \,\}$.

We can now introduce the well-founded semantics as follows:

▶ **Definition 5** (WFSX, [3]). *For any logic program π, define the transfinite sequence $\{I_\alpha\}$ by (i) $I_0 = \emptyset$, (ii) $I_{\alpha+1} = Cn(\pi^{Cn(\pi')})$ with $\pi' = \pi_s^{I_\alpha}$, for each successor ordinal $\alpha + 1$, and (iii) $I_\delta = \bigcup_{\alpha<\delta} I_\alpha$ for each limit ordinal δ, and let I_γ be its least fixpoint. If $I_\gamma \subseteq Cn(\pi_s^{I_\gamma})$, then $\mathrm{WFMX}(\pi) = I_\gamma \cup \mathrm{not}(\mathcal{L} \setminus Cn(\pi_s^{I_\gamma}))$ is the well-founded model of π; otherwise π is contradictory and has no well-founded model.*

We refer to I_α in Definition 5 as *fixpoint interpretations*. Later, we will use a notion of *even* and *odd* interpretations I_i, where the even ones corresponds to the sequence of interpretations from Definition 5 and the odd ones are defined as the least fixpoint obtained after applying the reduct of the semi-normal version of the program with respect to the last even interpretation.

The condition $I_\gamma \subseteq Cn(\pi_s^{I_\gamma})$ is equivalent to $\{p, \neg p\} \not\subseteq I_\gamma$ for all $p \in \mathcal{P}$ [3, Theorem 5.3], as illustrated by a simple example.

▶ **Example 6.** The semi-normal version of $\pi = \{\, \mathtt{p}.,\ \neg\mathtt{p}. \,\}$ is $\pi_s = \{\, \mathtt{p} \leftarrow \mathrm{not}\ \neg\mathtt{p}., \neg\mathtt{p} \leftarrow \mathrm{not}\ \mathtt{p}. \,\}$. As $\pi^S = \pi$ for every set S, we have $I_\gamma = I_1 = Cn(\pi) = \{\, \mathtt{p},\ \neg\mathtt{p} \,\}$; this results in $Cn(\pi_s^{I_\gamma}) = \emptyset$, violating the condition $I_\gamma \subseteq Cn(\pi_s^{I_\gamma})$.

To further demonstrate the role of the semi-normal transformation in defining well-founded models, consider the following example:

▶ **Example 7.** The semi-normal version of $\pi = \{\, r0:\ \mathtt{q} \leftarrow \mathrm{not}\ \mathtt{p}.,\ r1:\ \neg\mathtt{q}.,\ r2:\ \mathtt{p} \leftarrow \mathrm{not}\ \mathtt{q}. \,\}$ is $\pi_s = \{\, r0_s:\ \mathtt{q} \leftarrow \mathrm{not}\ \mathtt{p}, \mathrm{not}\ \neg\mathtt{q}.,\ r1_s:\ \neg\mathtt{q} \leftarrow \mathrm{not}\ \mathtt{q}.,\ r2_s:\ \mathtt{p} \leftarrow \mathrm{not}\ \mathtt{q}, \mathrm{not}\ \neg\mathtt{p}. \,\}$. We find that $I_1 = \{\neg\mathtt{q}\}$ after the first iteration; however, computing the least fixpoint yields $I_2 = \{\neg\mathtt{q}, \mathtt{p}\}$ and, therefore, to $\mathrm{WFMX}(\pi) = \{\, \neg\mathtt{q}, \mathtt{p}, \mathrm{not}\ \mathtt{q}, \mathrm{not}\ \neg\mathtt{p} \,\}$. This model is preferred as it adheres to the *coherence principle*, which states that an explicit negation $\neg\mathtt{q}$ should entail the corresponding default negation $\mathrm{not}\ \mathtt{q}$.

We show next how a semi-normal program version can capture reductio ad absurdum [3].

▶ **Example 8.** Consider the program $\pi = \{r0 : \mathtt{p} \leftarrow \mathrm{not}\ \mathtt{p}.,\ r1 : \neg\mathtt{p}.\}$. As $\pi_s = \{r0_s : \mathtt{p} \leftarrow \mathrm{not}\ \mathtt{p}, \mathrm{not}\ \neg\mathtt{p}.,\ r1_s : \neg\mathtt{p} \leftarrow \mathrm{not}\ \mathtt{p}.\}$, we derive from $\pi_s^\emptyset$ both $\mathtt{p}$ and $\neg\mathtt{p}$, and then from $\pi^{Cn(\pi_s^\emptyset)}$ only $\neg\mathtt{p}$, thus $I_1 = \{\neg\mathtt{p}\}$. In the next iteration, we derive from $\pi_s^{I_1}$ only $\neg\mathtt{p}$ and then both $\mathtt{p}$ and $\neg\mathtt{p}$, which is a contradictory model. Intuitively, if we have an assumption $\mathrm{not}\ \mathtt{p}$ that would lead to a contradiction (having both $\mathtt{p}$ and $\neg\mathtt{p}$ in the model $I_2 = \{\mathtt{p}, \neg\mathtt{p}\}$), then we can reject it. As Example 6 shows, for a program that entails contradictory propositions that are not derived by rejectable assumptions, this is different.

Finally, we recall that WFSX conservatively extends WFS.

▶ **Theorem 9** (Relation to WFS [3]). *For explicit negation-free logic programs, WFSX coincides with WFS as defined in [36].*

3.2 SAT/SMT Paradigm

Despite the NP-hardness of propositional satisfiability solving, modern SAT solvers are capable of solving highly complex boolean formulas. Building upon the success of SAT solvers, Satisfiability Modulo Theories (SMT) solvers [10] find satisfying assignments for formulas over a first-order theory. Depending on the choice of the (fragment of the) theory, unlike boolean satisfiability checking, SMT checking is not necessarily decidable.

Unless a SAT/SMT formula is unsatisfiable, the SAT/SMT solver will produce a concrete assignment for all free variables. In other words, even if a formula can be falsified, the solver will always choose a satisfying assignment as long as one exists. Propositional formulas

generally are two-valued in SAT/SMT, and any subformula will be either true or false when interpreted under this concrete assignment. For example, a partially solved Sudoku puzzle can be naturally encoded in both WFS and SAT/SMT. However, a natural encoding in SAT/SMT will lead the solver to choose *some* valid solution, whereas a natural encoding in WFS will lead to a fixpoint that accurately represents uncertainty through undefined assignments. The semantics that we will introduce in Section 4 combines nondeterministic choices, such as those made by SMT solvers, with the notion of uncertainty in WFS.

4 SAT Modulo Well-Founded Semantics

In this section, we introduce a novel integration of nondeterministic choices into the Well-Founded Semantics (with explicit negation).

4.1 Syntax

We enrich the syntax of ELPs (see Definition 1) with a choice operator (ELP$^{\mathrm{C}}$) of the form

$$\ell\,\{p_1;\ldots;p_k\}\,u \tag{1}$$

where $p_1,\ldots,p_k \in \mathcal{P}$ and $\ell, u \in \mathbb{N}$ are natural numbers such that $0 \leq \ell \leq u \leq k$. Informally, (1) holds whenever (i) at least ℓ and at most u atoms p_i from the set $S = \{p_1,\ldots,p_k\}$ hold and (ii) the explicit negations $\neg p_j$ of all others atoms in S hold.

▶ **Definition 10** (ELP$^{\mathrm{C}}$ Programs). *An Extended Normal Logic Program with a Choice operator (ELP^C) π (or simply, a* logic program with choices*) is a set of rules of the form:*

$$r \;:\; B_0 \leftarrow B_1,\ldots B_n, \mathrm{not}\, B_{n+1},\ldots \mathrm{not}\, B_m. \qquad n,m \geq 0,$$

where B_0 is either an objective literal or a choice operator and $\{B_1,\ldots,B_m\} \subset \mathcal{L}$ are objective literals; $H(r)$, $B(r)$ etc. are as in Definition 1.

For convenience, we define $\bot \leftarrow \mathit{body}(c).$ as a shorthand for the two rules $p \leftarrow \mathit{body}(c).$ $\neg p \leftarrow \mathit{body}(c).$ so that whenever the body of the constraint c holds, we derive a contradiction.

▶ **Example 11** (Action choice). A typical use of the choice operator arises in action domains, where an agent must select exactly one action to execute at a time. For instance, the rule

$$1\{\,\mathtt{move};\mathtt{pick\_up};\mathtt{wait}\,\}1 \leftarrow \mathrm{not}\,\mathtt{goal}.$$

states that exactly one of `move`, `pick_up`, or `wait` must hold unless the `goal` has been achieved. Once an action is executed we can encode inertia in an ASP-like way such as

$$\mathtt{next}(\mathtt{fluent}(X)) \leftarrow \mathtt{fluent}(X) \wedge \mathrm{not}\, \neg\mathtt{next}(\mathtt{fluent}(X))$$
$$\neg\mathtt{next}(\mathtt{fluent}(X)) \leftarrow \mathtt{fluent}(X) \wedge \mathrm{not}\, \mathtt{next}(\mathtt{fluent}(X))$$

The applicability of each action can then be regulated by rules expressing its preconditions, as well as abnormality conditions that may prevent the action from being chosen. For example:

$$\neg\mathtt{move} \leftarrow \mathrm{not}\,\mathtt{pre\_move}. \qquad \neg\mathtt{move} \leftarrow \mathtt{ab\_move}.$$
$$\neg\mathtt{pick\_up} \leftarrow \mathrm{not}\,\mathtt{pre\_pick\_up}. \qquad \neg\mathtt{pick\_up} \leftarrow \mathtt{ab\_pick\_up}.$$
$$\neg\mathtt{wait} \leftarrow \mathrm{not}\,\mathtt{pre\_wait}. \qquad \neg\mathtt{wait} \leftarrow \mathtt{ab\_wait}.$$

Intuitively, rules of the form $\neg\texttt{a} \leftarrow \text{not}\,\texttt{pre\_a}.$ and $\neg\texttt{a} \leftarrow \texttt{ab\_a}$ prevent choosing action $\texttt{a}$ if either its preconditions $\texttt{pre\_a}$ are not fulfilled or abnormal conditions $\texttt{ab\_a}$ prevent it. Furthermore, we want to be able to express qualitative uncertainty about the environment similarly to other planning frameworks such as [24, 60]. Such uncertainty can be captured by

$$\texttt{fluent}(p) \leftarrow \text{not}\,\neg\texttt{fluent}(p). \qquad \neg\texttt{fluent}(p) \leftarrow \text{not}\,\texttt{fluent}(p).$$

which leads to an undefined truth value for both $\texttt{fluent}(p)$ and $\neg\texttt{fluent}(p)$. Such undefinedness is propagated by an encoding of the inertia rule, unless an action effects that $\texttt{fluent}(p)$ or $\neg\texttt{fluent}(p)$ is true. Together with the global choice $1\{\,\texttt{move};\texttt{pick\_up};\texttt{wait}\,\}1$, the program states that *exactly one admissible action must be selected unless* a goal condition is met, and that abnormality literals can dynamically block actions depending on the context.

We revisit Example 7.4 from [36] showing how the undefined truth value can be used to capture ill-defined relations. When relations are recursively defined, undefinedness faithfully propagates. As we have a choice operator, some factual input can be treated as optional, allowing us to test whether the relations are well-defined for each possible input; this can be seen as the computational problem of checking whether a set of properties or norms is well-defined for all possible input instances.

▶ **Example 12** (Handling cyclic precedence relations.)**.** In the following setting, we are provided with a set of properties identified by a property id and their content. For instance, $\texttt{prop}(44, \texttt{red})$ means that property 44 states $\texttt{red}$, while $\neg\texttt{prop}(32, \texttt{red})$ stands for property 32 states $\neg\texttt{red}$. We furthermore introduce a precedence relation over properties which can be used to prevent contradictory properties from being derived in case the $\texttt{prec}$ relation happens to be acyclic. However, if two properties are contradictory and there is a mutual precedence relation between them, then the following schema allows us to conclude $\texttt{holds}(P)$ and $\neg\texttt{holds}(P)$ to be undefined. If instead two contradictory properties do not have any precedence relation between them, then the program is contradictory.

$$\begin{aligned}
\texttt{holds}(P) &\leftarrow \texttt{prop}(\text{ID}, P), \text{not}\,\texttt{overwritten}(\text{ID}).\\
\neg\texttt{holds}(P) &\leftarrow \neg\texttt{prop}(\text{ID}, P), \text{not}\,\texttt{overwritten}(\text{ID}).\\
\neg\texttt{holds}(P) &\leftarrow \texttt{prop}(\text{ID}, P), \text{not}\,\neg\texttt{overwritten}(\text{ID}).\\
\texttt{holds}(P) &\leftarrow \neg\texttt{prop}(\text{ID}, P), \text{not}\,\neg\texttt{overwritten}(\text{ID}).\\
\texttt{overwritten}(\text{ID}) &\leftarrow \neg\texttt{holds}(P), \texttt{prop}(\text{ID}, P), \neg\texttt{prop}(\text{ID}', P), \texttt{prec}(ID, ID').\\
\neg\texttt{overwritten}(\text{ID}) &\leftarrow \texttt{holds}(P), \texttt{prop}(\text{ID}, P), \neg\texttt{prop}(\text{ID}', P), \texttt{prec}(\text{ID}', \text{ID}).\\
\texttt{prec}(\text{ID}, \text{ID}'') &\leftarrow \texttt{prec}(\text{ID}, \text{ID}'), \texttt{prec}(\text{ID}', \text{ID}'').
\end{aligned}$$

The following semantics for the choice construct satisfies the action choice intended behavior.

4.2 Semantics

In order to soon obtain a definition of well-founded models for ELP^{C} programs, we first introduce the notion of *choice-resolved* ELP^{C} programs.

▶ **Definition 13** (Choice-resolved ELP^{C}-program)**.** *Given an ELP^C-program π, for each rule $r \in \pi$ whose head is a choice of the form $\ell\{p_1; \ldots; p_k\}u$, let $C_r \subseteq \{p_1, \ldots, p_k\}$ be arbitrary with $\ell \leq |C_r| \leq u$, and replace r with the k many ELP rules*

$$\{p_i \leftarrow B(r). \mid p_i \in C_r\} \cup \{\neg p_j \leftarrow B(r). \mid p_j \in \{p_1, \ldots, p_k\} \setminus C_r\}.$$

The resulting program is a choice-resolved *program for π, denoted* CR-π.

By construction, *CR-*π is an ELP as per Definition 1, and the standard WFSX semantics (see Definition 5) can be applied to it; however, clearly, *CR-*π is not unique in general.

▶ **Definition 14** (WFSX$^{\mathrm{C}}$). *Given an ELP^C program π, the well-founded model* WFMX(π') *of any choice-resolved program π' of π is a well-founded model of π. We denote the set of all such models as* WFMX$^{\mathrm{C}}(\pi)$.

This construction provides a *decompositional* view of the choices in a program: each choice is resolved syntactically and contributes a deterministic component to the resulting *CR-*program. Such a decomposition is closely aligned with ordered disjunction–based extensions in which possible actions, world states, and desired outcomes induce a preference ordering over models via ordered disjunction [14]. The syntactic treatment of choices in our setting suggests a clear path to adapting similar preference mechanisms, in contrast to ASP's inherently model (assumption)-based treatment of choices through the reduct.

▶ **Example 15.** Consider the program $\pi = \{\, 1\{\mathtt{p};\, \mathtt{q}\}1 \leftarrow \mathrm{not}\, \mathtt{p}.\,\}$. Choice resolution (per Definition 13) permits two *decompositions* of π into deterministic programs:

- If we choose $\mathtt{p}$ (and thereby $\neg\mathtt{q}$), we obtain $\pi_1 = \{\mathtt{p} \leftarrow \mathrm{not}\, \mathtt{p}., \neg\mathtt{q} \leftarrow \mathrm{not}\, \mathtt{p}.\}$. The next fixpoint iteration step produces the empty set. Hence, both $\mathtt{p}$ and $\neg\mathtt{q}$ remain undefined in the well-founded model.
- If we choose $\mathtt{q}$ (and thereby $\neg\mathtt{p}$), we obtain $\pi_2 = \{\mathtt{q} \leftarrow \mathrm{not}\, \mathtt{p}., \neg\mathtt{p} \leftarrow \mathrm{not}\, \mathtt{p}.\}$. Subsequent fixpoint iterations yield $\neg\mathtt{p}$ and $\mathtt{q}$. The well-founded model is $\{\, \mathrm{not}\, \mathtt{p}, \neg\mathtt{p}, \mathtt{q}, \mathrm{not}\, \neg\mathtt{q}\,\}$.

It may be desirable to impose additional *integrity constraints* [3, Definition 7.1] on the well-founded model that otherwise cannot be expressed using regular rules. For instance, in planning problems, we may want to impose that a necessary precondition of an action must not be undefined if that action is chosen.

▶ **Definition 16** (Integrity constraints). *An* integrity constraint *is a statement of the form $c : L_1 \vee \cdots \vee L_k \leftarrow L_{k+1}, \ldots, L_n.$, where all L_i are default literals. A well-founded model W* satisfies *c if $\{\, L_1, \ldots, L_k \,\} \cap W \neq \emptyset$ or $\{\, L_{k+1}, \ldots, L_n \,\} \not\subseteq W$. A well-founded model W satisfies a set C of integrity constraints if it satisfies each integrity constraint $c \in C$.*

Integrity constraints, unlike regular rules within an ELP$^{\mathrm{C}}$ program, do not (directly) contribute to model construction; they merely serve to select a subset of the models in WFMX$^{\mathrm{C}}$. In particular, integrity constraints do not derive literals. For example, we may require that for each action $\mathtt{a}$ the integrity constraint $\mathtt{a} \vee \mathrm{not}\, \mathtt{a} \leftarrow \top$ holds, which rules out well-founded models in which whether or not $\mathtt{a}$ is chosen is undefined; however, this integrity constraint by itself does not derive either $\mathtt{a}$ or $\mathrm{not}\, \mathtt{a}$ during model construction.

We end this section with a useful property that we will take adantage of in order to simplify the evaluation of ELP$^{\mathrm{C}}$ programs.

▶ **Definition 17** (Choice-NF). *An ELP^C program π is in* Choice-Normal Form *(Choice-NF) if every rule $r \in \pi$ whose head is a choice is a fact (i.e., has an empty body).*

▶ **Theorem 18.** *Every ELP^C program π over atoms $\mathcal{P}_\pi \subset \mathcal{P}$ can be transformed, using a set $A \subset \mathcal{P}$ of auxiliary atoms with $A \cap \mathcal{P}_\pi = \emptyset$ and $|A| \leq |\mathcal{P}_\pi| * |\pi|$, into an ELP^C program π' in Choice-NF such that the models of π and π' coincide modulo A.*

4.3 Relation with ASP and SAT

The notion of choice was originally introduced with a stable model characterization [50], and naturally extends to an answer-set characterization that admits explicit negation. For completeness, we adapt the following definition and property from [3] to our setting.

Table 1 Recursive translation tr of propositional formulas into ELP$^{\mathrm{C}}$ rules.

Formula ψ	**Rules for** x_ψ **in** tr	**Formula** ψ	**Rules for** x_ψ **in** tr
$\psi = \psi_1 \wedge \cdots \wedge \psi_k$	$x_{\psi_i} \leftarrow x_\psi.\ (1 \leq i \leq k)$	$\psi_1 \vee \cdots \vee \psi_k$	$x_\psi \leftarrow x_{\psi_i}.\ (1 \leq i \leq k)$
	$x_\psi \leftarrow x_{\psi_1}, \ldots, x_{\psi_k}.$		$1\{x_{\psi_1}; \ldots; x_{\psi_k}\}k \leftarrow x_\psi.$
	$0\{x_{\psi_1}; \ldots; x_{\psi_k}\}(k-1) \leftarrow \neg x_\psi.$		$\neg x_\psi \leftarrow \neg x_{\psi_1}, \ldots, \neg x_{\psi_k}.$
	$\neg x_\psi \leftarrow \neg x_{\psi_i}.\ (1 \leq i \leq k)$		$\neg x_{\psi_i} \leftarrow \neg x_\psi.\ (1 \leq i \leq k)$

▶ **Definition 19** ((Partial) Stable Model). *Let π be an ELP^C program. A consistent set $S \subseteq df\mathcal{L}$ of default literals is a* partial stable model *of π if $S = Cn(\pi'^{S\cap\mathcal{L}}) \cup \mathrm{not}(\mathcal{L} \setminus Cn(\pi_s'^{S\cap\mathcal{L}}))$ for some CR-resolved program π' for π. Moreover, such S is an* answer set *or* stable model *of π if S is total, i.e., if for each literal $l \in \mathcal{L}$ either $l \in S$ or $\mathrm{not}\, l \in S$ holds.*

The well-founded semantics with choices is a *sound* approximation of the answer-set semantics.

▶ **Theorem 20.** *Let π' be any CR-resolved program for an ELP^{C}program π. If W is the well-founded model of π' and S an arbitrary answer set of π', then $W \subseteq S$.*

We moreover note that it is also possible to capture answer set semantics.

▶ **Theorem 21.** *For every ELP program π over $\mathcal{L}_\pi \subset \mathcal{L}$ there exists an ELP^C program π' over $\mathcal{L}_\pi \cup \mathcal{L}'_\pi$, where $\mathcal{L}'_\pi = \{L' \mid L \in \mathcal{L}_\pi\}$, such that the answer sets of π coincide with* WFMX$^{\mathrm{C}}(\pi)$ *modulo $\mathcal{L}'_\pi$.*

Proof. We can rewrite an ELP program π into π' as follows: for each literal $L \in \mathcal{L}_\pi$, we add a choice $0\{\, L' \,\}1$ for L'; this selects some S as in Definition 19. For each rule $r \in \pi$, we replace $B^-(r)$ with $B^-_{df}(r) = \{\neg L' \mid L \in B^-(r)\}$ to obtain a new rule $r' \ :\ H(r) \leftarrow B^+(r), B^-_{df}(r)$ without default negation. By adding constraints $\bot \leftarrow L, \neg L'.$ and $\bot \leftarrow \mathrm{not}\, L, L'.$ we can enforce that consequences of the reduct $\pi'^{S\cap\mathcal{L}}$ coincide with S. Thus, the answer sets of π coincide with those of π' modulo $\mathcal{L}'_\pi$. By Theorem 32 below, the answer sets of π' coincide with the well-founded models of π', which then yields the result. ◀

Considering SAT now instead of ASP, we note that in our semantics, we can express arbitrary propositional formulas under classical semantics. Given a propositional formula φ, a Tseitin-style translation tr, shown in Table 1, rewrites φ into an equisatisfiable set of ELP$^{\mathrm{C}}$ rules by introducing auxiliary atoms as needed.

▶ **Proposition 22.** *Given a formula ψ, the ELP^C program $tr(\psi) \cup \{\, x_\psi. \,\} \cup \bigcup_{p\in\mathcal{P}} 0\{p\}1$. is satisfiable if and only if ψ admits a classical model, where x_ψ denotes the propositional atom introduced by the tr translation to represent the formula ψ.*

The following example shows how ELP$^{\mathrm{C}}$ can combine classical constraints and reachability.

▶ **Example 23** (SAT guessing with fixpoint reachability). Let $V = \{1, \ldots, n\}$ be a finite set of nodes, that are expressed as facts $\mathtt{v}(1), \ldots, \mathtt{v}(n)$. Edges of a directed graph are guessed via choices $0\{\mathtt{e}(U, V)\}1 \leftarrow \mathtt{v}(U), \mathtt{v}(V)$. The guessed graph can be constrained by classical (non-recursive) SAT-style conditions; for instance, $\bot \leftarrow \mathtt{e}(U, V), \mathtt{e}(V, U), U \neq V$ prevents two-cycles. Reachability is then defined inductively:

$$\mathtt{reach}(X, X) \leftarrow \mathtt{v}(X). \qquad \mathtt{reach}(X, Z) \leftarrow \mathtt{e}(X, Y), \mathtt{reach}(Y, Z).$$

Given Proposition 22, we can express a SAT formula φ by encoding it into an ELP$^{\mathrm{C}}$ program, furthermore, we may use the predicate `reach` in φ, which is defined and governed by the well-founded semantics.

Algorithm 1 Computing WFSX$^{\mathrm{C}}$ for ELP$^{\mathrm{C}}$ programs in Choice-NF.

```
1:  Initialize the SAT solver
2:  for each choice ℓ { p_1, ..., p_k } u in the head of a rule r do
3:      for each 1 ≤ i ≤ k do
4:          introduce a SAT variable c_i^r for selecting p_i in r
5:      add the corresponding cardinality constraints ℓ ≤ (∑_{1≤i≤k} c_i^r) ≤ u      ▷ following [59]
6:  loop
7:      if the SAT solver returns UNSAT then
8:          return failure                                  ▷ no well-founded model exists
9:      W ← WFSX(SAT choice resolution)                     ▷ Definition 5
10:     if W = ⊥ then
11:         add clauses preventing the current conflict
12:     else if W violates the integrity constraints then   ▷ Definition 16
13:         add clauses preventing this integrity constraint violation
14:     else
15:         return W                                        ▷ Definition 14
```

5 Evaluating ELP$^{\mathrm{C}}$ Programs

We present an SMT-like solving procedure for ELP$^{\mathrm{C}}$ programs in Choice-NF that relies on a SAT solver. We use an encoding of cardinality constraints based on sequential counters, as presented by Sinz [59], that is not minimal in terms of the number of clauses or auxiliary variables but decidable by a CDCL solver almost entirely via unit propagation.

Algorithm 1 sets up the choice encoding (lines 2–5) and then selects in a loop a choice resolution (line 7) and checks whether the resulting model W is contradiction-free and satisfies all integrity constraints; if not, it adds clauses ruling out the current choice resolution (lines 10–13). The naïve version of this algorithm only adds a single clause at a time, ruling out only the current choice resolution. In the next section, we show how to optimize this exhaustive search by introducing *clause learning* for a more efficient handling of contradictions and integrity constraint violations. Then, in Subsection 5.2, we show how we can use a syntactic analysis on the program to decompose it into subprograms to speed-up the computation.

5.1 Clause Learning for Contradiction and Integrity Constraints

We present a SLX-inspired clause learning procedure, where SLX is a top-down procedure originally proposed in [3] for a relaxation of WFSX where contradictory conclusions are allowed, adapting similar techniques from related frameworks [12, 18, 19]. SLX consists of the sub-procedures SLX-T and SLX-TU that derive whether an objective literal l is *true* (SLX-T) or whether l is *true or undefined* (SLX-TU). Thus, checking whether not l holds amounts to checking whether the SLX-TU derivation for l fails.

We adapt SLX as the basis of our clause learning procedure: (i) we remove the infinite loop detection for the recursion through default negation, as we have a partial interpretation obtained from given fixpoint iteration steps, and (ii) we collect the choices that have been relevant to the current contradictory derivation.

Preliminaries: For readability, throughout this section, we let π be some fixed ELP$^{\mathrm{C}}$ program in Choice-NF. Let $C \subset \mathcal{L}$ be the set of objective literals such that their atoms appear in a choice head in π. Given this program, we define the following procedures.

▶ **Definition 24** (Block and Derive). *Let $i \geq 0$ and S be a set of visited literals. Then,*

- *the procedure* $\textsc{Block}(l, i, S)$ *produces a formula for literal $l \in \mathcal{L}$ as follows:*
  - *$\neg l$ if $l \in C$ and $i > 0$;*
  - *$\top$ if $l \notin C$, and either $l \in S$ or $i = 0$;*
  - *otherwise a conjunction of subformulas for all rules $r \in \pi$ if i is odd resp. $r \in \pi_s$ if i is even, such that $H(r) = l$, given by*

$$\bigvee_{b \in B^+(r)} \textsc{Block}(b, i, S \cup \{l\}) \;\vee\; \bigvee_{b \in B^-(r)} \textsc{Derive}(b, i-1, \emptyset).$$

- *the procedure* $\textsc{derive}(l, i, S)$ *produces a formula for literal $l \in \mathcal{L}$ as follows:*
  - *l if $l \in C$ and $i > 0$;*
  - *$\bot$ if $l \notin C$, and either $l \in S$ or $i = 0$;*
  - *a disjunction over the subformulas for each rule $r \in \pi$ if i is odd resp. $r \in \pi_s$ if i is even such that $H(r) = l$, given by*

$$\bigwedge_{b \in B^+(r)} \textsc{derive}(b, i, S \cup \{l\}) \;\wedge\; \bigwedge_{b \in B^-(r)} \textsc{block}(b, i-1, \emptyset).$$

Given a sequence $I_0, I_1, \ldots$ of fixpoint interpretations, as soon as we obtain an interpretation I_i with i even that contains contradictory literals p and $\neg p$, we generate a formula

$$\textsc{Consistent}(p, i) \coloneqq \textsc{Block}(p, i, \emptyset) \vee \textsc{Block}(\neg p, i, \emptyset)$$

that is added to the SAT solver context via a standard Tseitin transformation.

▶ **Example 25.** Consider the program

$$\pi = \{\neg \mathtt{p} \leftarrow \mathtt{p}.,\, \mathtt{p} \leftarrow \neg \mathtt{p}.,\, 0\,\{\mathtt{b}\}\,1.,\, \mathtt{p} \leftarrow \mathtt{b}, \mathrm{not}\ \mathtt{c}.,\, \neg \mathtt{p} \leftarrow \neg \mathtt{b}, \mathrm{not}\ \mathtt{c}.\}$$

and choose $\neg \mathtt{b}$ to be true. Then, fixpoint iteration yields $I_0 = \emptyset$, $I_1 = \{\mathtt{p}, \neg \mathtt{p}, \neg \mathtt{b}\}$, $I_2 = I_1$, which is contradictory. The naive version of Algorithm 1 would now block this particular choice, and the SAT solver would attempt choosing $\mathtt{b}$ next, which ultimately would lead to the same conflict. With our clause learning procedure, however, $\textsc{Consistent}(\mathtt{p}, 2)$ returns a formula equivalent to $\mathtt{b} \wedge \neg \mathtt{b}$ (i.e., $\bot$), which immediately makes the SAT context unsatisfiable and thereby a second model construction attempt unnecessary.

The generated formula is sound, meaning that it does not rule out any choice resolution that could lead to a consistent model. The following lemma is useful to prove it.

▶ **Lemma 26** (Effect of Block and Derive). *Let $J \colon C \to \{\top, \bot\}$. Then $J \models \textsc{Derive}(l, i, \emptyset)$ (respectively, $J \models \textsc{Block}(l, i, \emptyset)$) iff the set of choices induced by J entails that l appears (respectively, does not appear) in the i-th interpretation according to Definition 14.*

▶ **Theorem 27** (Effect of Consistent). *Let $J \colon C \to \{\top, \bot\}$ and I_i a fixpoint interpretation with i even, such that $p, \neg p \in I_i$ for some atom p. Then, $J \models \textsc{Consistent}(p, i)$ iff the set of choices induced by J entails that p and $\neg p$ do not both appear in the i-th interpretation.*

Given the tree-shaped recursive nature of the Block and Derive procedures, the size of the resulting formulas may be of concern. However, repeated invocations of these procedures with the same arguments lead to the same formulas, thereby allowing efficient caching. Through a combination of basic formula simplifications, formula reuse, and short circuiting, the number of recursive invocations and the size of the learned formula are both bounded.

Notably, we can further extend this technique to also generalize from a choice that leads to a particular integrity constraint violation (see Definition 16). Unlike conflicts during the computation of the well-founded model, integrity constraints generally can only safely be evaluated after the model has been fully constructed. Once we determine that the computed well-founded model does not satisfy a particular integrity constraint

$$c\colon L_1 \vee \cdots \vee L_k \leftarrow L_{k+1}, \ldots, L_n$$

with $L_1, \ldots, L_n \in \mathit{df}\,\mathcal{L}$, we therefore construct a formula $\textsc{Integrity}(c, i)$, where i is the last fixpoint interpretation index (and necessarily even):

$$\textsc{Integrity}(c, i) = \bigvee\nolimits_{1 \leq j \leq k} \textsc{InModel}(L_j, i) \vee \bigvee\nolimits_{k+1 \leq j \leq n} \textsc{NotInModel}(L_j, i),$$

where

$$\textsc{InModel}(L, i) = \begin{cases} \textsc{Derive}(l, i, \emptyset), & \text{for } l \in \mathcal{L}, L = l \\ \textsc{Block}(l, i-1, \emptyset), & \text{for } l \in \mathcal{L}, L = \mathrm{not}\, l \end{cases}$$

$$\textsc{NotInModel}(L, i) = \begin{cases} \textsc{Block}(l, i, \emptyset), & \text{for } l \in \mathcal{L}, L = l \\ \textsc{Derive}(l, i-1, \emptyset), & \text{for } l \in \mathcal{L}, L = \mathrm{not}\, l. \end{cases}$$

However, it may not be possible to satisfy the integrity constraint within the number of fixpoint iterations that were computed prior to reaching the alternating fixpoint. For instance, an integrity constraint ruling out undefinedness for a literal $\mathtt{c}$, written $\mathtt{c} \vee \mathrm{not}\, \mathtt{c} \leftarrow \top$, might be entirely unsatisfiable within k fixpoint interpretations, and may only become satisfiable in a future interpretation, which however was not computed as part of the model construction.

▶ **Example 28.** Consider the program

$$\pi = \{\, 0\{\,\mathtt{p}\,\}1.,\ \mathtt{q} \leftarrow \mathrm{not}\, \mathtt{p}.,\ \mathtt{r} \leftarrow \mathrm{not}\, \mathtt{s}.,\ \mathtt{s} \leftarrow \mathrm{not}\, \mathtt{t}.,\ \mathtt{t} \leftarrow \mathtt{q}, \mathrm{not}\, \mathtt{r}.,\ \mathtt{c} \leftarrow \mathrm{not}\, \mathtt{s}.\,\}$$

and the integrity constraint $\mathtt{c} \vee \mathrm{not}\, \mathtt{c} \leftarrow \top$. There are two possible choice resolutions. If we choose $\neg\mathtt{p}$, then we obtain the following sequence of interpretations:

$$I_0 = \emptyset,\ I_1 = \{\mathtt{q}, \mathtt{s}, \mathtt{r}, \mathtt{c}, \mathtt{t}, \neg\mathtt{p}\},\ I_2 = \{\mathtt{q}, \neg\mathtt{p}\},\ I_3 = \{\mathtt{q}, \mathtt{c}, \mathtt{r}, \mathtt{t}, \mathtt{s}, \neg\mathtt{p}\},\ I_4 = \{\mathtt{q}, \neg\mathtt{p}\}.$$

Such a computation leads to the well founded model

$$W_{\neg\mathtt{p}} = \{\neg\mathtt{p}, \mathtt{q}, \mathrm{not}\, \neg\mathtt{c}, \mathrm{not}\, \neg\mathtt{q}, \mathrm{not}\, \neg\mathtt{r}, \mathrm{not}\, \neg\mathtt{s}, \mathrm{not}\, \neg\mathtt{t}, \mathrm{not}\, \mathtt{p}\},$$

which means that the set $\{\, \mathtt{c}, \mathtt{r}, \mathtt{s}, \mathtt{t}\,\}$ of literals is undefined, hence, $W_{\neg\mathtt{p}}$ violates the integrity constraint. If we instead choose $\mathtt{p}$, the computed sequence of fixpoint interpretations is

$$I_0 = \emptyset,\ I_1 = \{\mathtt{r}, \mathtt{q}, \mathtt{t}, \mathtt{c}, \mathtt{p}, \mathtt{s}\},\ I_2 = \{\mathtt{p}\},\ I_3 = \{\mathtt{p}, \mathtt{r}, \mathtt{c}, \mathtt{s}\},\ I_4 = \{\mathtt{s}, \mathtt{p}\},$$
$$I_5 = \{\mathtt{s}, \mathtt{p}\},\ I_6 = \{\mathtt{s}, \mathtt{p}\},$$

which translates into the well-founded model

$$W_{\mathtt{p}} = \{\mathtt{p}, \mathtt{s}, \mathrm{not}\, \mathtt{c}, \mathrm{not}\, \neg\mathtt{c}, \mathrm{not}\, \neg\mathtt{p}, \mathrm{not}\, \neg\mathtt{q}, \mathrm{not}\, \neg\mathtt{r}, \mathrm{not}\, \neg\mathtt{s}, \mathrm{not}\, \neg\mathtt{t}, \mathrm{not}\, \mathtt{q}, \mathrm{not}\, \mathtt{r}, \mathrm{not}\, \mathtt{t}.\},$$

which satisfies the integrity constraint ($\mathrm{not}\, \mathtt{c} \in W_{\mathtt{p}}$). However, the number of alternations increased with respect to the computation of $W_{\neg\mathtt{p}}$. Therefore, any attempt of satisfying the integrity constraint within the original number of alternations would have failed.

In other words, adding $\textsc{Integrity}(c, i)$ to the SAT context might make it unsatisfiable, even if the integrity constraint could perhaps be satisfied for a different choice resolution that would have increased the number of fixpoint iterations. We solve this issue by weakening the formula before adding it to the SAT context in the following way:

$$\textsc{WeakIntegrity}(c, i) \coloneqq \textsc{Integrity}(c, i) \vee \textsc{FutureFixpoint}(i), \text{ where}$$
$$\textsc{FutureFixpoint}(i) \coloneqq \bigvee\nolimits_{l \in \mathcal{L}} (\textsc{Block}(l, i-2, \emptyset) \wedge \textsc{Derive}(l, i, \emptyset)) .$$

The resulting formula produced by $\textsc{WeakIntegrity}(c, i)$ indeed achieves that any choice assignment satisfying the formula satisfies the integrity constraint c at the i-th fixpoint interpretation or leads to a longer sequence of fixpoint interpretations. We add this formula to the SAT context through a standard Tseitin translation as well.

Conveniently, all of these procedures produce formulas in Negation Normal Form (NNF), which means that weakening any subformula may weaken the formula as a whole, but can never strengthen it. We can take advantage of this property as follows.

- For even $i > 2$, if $l \in I_{i-2}$, then $\textsc{Block}(l, i) \models \textsc{Block}(l, i-2)$. Because $l \in I_{i-2}$, the formula $\textsc{Block}(l, i-2)$ will still rule out the current choice assignment (Lemma 26). Therefore, in this case, we can compute and use $\textsc{Block}(l, i-2)$ instead of $\textsc{Block}(l, i)$.
- Similarly, for odd $i > 1$, if $l \notin I_{i-2}$, then $\textsc{Derive}(l, i) \models \textsc{Derive}(l, i-2)$, and we can compute and use $\textsc{Derive}(l, i-2)$ instead of $\textsc{Derive}(l, i)$.

These shortcuts can significantly reduce recursion depth, especially when they can be applied repeatedly. The formulas produced by Consistent (in case of a conflict) and WeakIntegrity (in case of an integrity constraint violation) may now be weaker (but not stronger), however, they still – at least – rule out the current choice assignment and therefore are sufficient for guiding the SAT solver toward a satisfying assignment, if one exists.

5.2 Program Decomposition

Resorting to standard analysis on syntactic dependencies of a logic program [28], we adapt two such techniques with beneficial implications in the solving phase. First, we make use of *splitting* [45], which allows to divide a program into a bottom and a top subprogram. The well-founded model of the entire program can then be obtained by aggregating the well-founded models of these subprograms. Second, we show how a program with *stratified* use of default negation [36] allows computing the well-founded model via a single fixpoint procedure that also behaves favorably in terms of grounding. We later combine the two techniques to structure the program into stratified and unstratified subprograms. We start by introducing a notion of splitting sets.

▶ **Definition 29** (Splitting Set). *A set $U \subseteq \mathcal{L}$ is a* splitting set *for an ELP^C program π if (i) for every rule $r \in \pi$, if $H(r) \cap U \neq \emptyset$, then $(B(r) \cup H(r)) \subseteq U$, and (ii) $\neg l \in U$ for $l \in U$.*

The set of rules $r \in \pi$ satisfying $(B(r) \cup H(r)) \subseteq U$ is denoted as $b_U(\pi)$ and called *bottom* of π with respect to U. This partition allows one to induce sets of rules (subprograms) that are *closed-by-definition*, meaning that the truth values of atoms they derive cannot be altered by upper subprograms, as they do not syntactically depend on them.

Given a consistent set $S \subseteq \mathit{df}\mathcal{L}$ of default literals and a splitting set U for a program π, we construct the *top* program $e_U(\pi, S)$ as follows: we include all rules r' resulting from some $r \in \pi \setminus b_U(\pi)$ by replacing each literal $l \in U$ in $B^+(r)$ with (i) $\top$ if $l \in S$, (ii) $\bot$ if not $l \in S$,

and (iii) a fresh literal $\mathbf{u}$ otherwise[1], and replacing $\operatorname{not} l$ in $B(r)$ with $l \in U$ with (i) $\bot$ if $l \in S$, (ii) $\top$ if $\operatorname{not} l \in S$, and (iii) $\mathbf{u}$ otherwise. If $\mathbf{u}$ was used, we add the rule $\mathbf{u} \leftarrow \operatorname{not} \mathbf{u}$, which derives $\mathbf{u}$ in the first step of fixpoint iteration but not in the second etc.; this captures exactly the undefinedness truth value. Our main result on program splitting is then as follows.

▶ **Theorem 30** (Splitting theorem). *Let U be a splitting set for an ELP^C program π. Then S is a partial stable model of π iff both (i) S_1 is a partial stable model of $b_U(\pi)$ and (ii) S_2 is a partial stable model of $e_U(\pi, S_1)$, where $S_1 = S \cap (U \cup \operatorname{not} U)$ and $S_2 = S \setminus (U \cup \operatorname{not} U)$.*

The proof of this theorem follows [45], adapted to our three-valued setting. Note that we introduced condition (ii) in Definition 29 as, when applying the semi-normal transformation of the program in the computation of the well-founded model, each rule with head p contains $\operatorname{not} \neg p$ in the body. For example, given $\pi = \{r_0\colon \mathtt{p} \leftarrow \operatorname{not} \mathtt{q}.,\ r_1\colon \mathtt{q} \leftarrow \operatorname{not} \mathtt{p}.,\ r_2\colon \neg\mathtt{p}\}$, if $\{\mathtt{p}, \mathtt{q}\}$ were a splitting set, then the well-founded model of the bottom subprogram $\{r_0, r_1\}$ would leave both $\mathtt{p}$ and $\mathtt{q}$ undefined, but we would later find $\neg\mathtt{p}$ to be true. However, the well-founded model of the whole program π is $\{\neg\mathtt{p}, \mathtt{q}, \operatorname{not} \mathtt{p}, \operatorname{not} \neg\mathtt{q}\}$.

We repeatedly apply splitting to partition the program into smaller subprograms and combine the resulting partial stable models according to Theorem 30. Within each subprogram, we exploit another decomposition technique based on *stratification*. To this end, we introduce dependency graphs for ELP$^{\mathrm{C}}$ programs in Choice-NF.

▶ **Definition 31** (Dependency Graph). *The* dependency graph *G_π of an ELP^C program π in Choice-NF is the directed graph whose vertices are all objective literals occurring in π, and whose edges represent dependencies induced by rule bodies as follows: for each rule $r \in \pi$,*

- *add for each objective literal l in $H(r)$ and each literal $l' \in B^+(r)$ a positive edge $l \to^+ l'$;*
- *add for each objective literal l in $H(r)$ and each literal $l' \in B^-(r)$ a negative edge $l \to^- l'$.*

If there are no cycles in the dependency graph G_π of a program π passing through a negative arc, we say that the program π is *stratified*. This allows us to further partition $\mathcal{L}$ into *strata* $P_0, \ldots, P_k$ that permit an iterative fixpoint computation. We obtain this partitioning by computing the strongly connected components of G_π and sorting them in some topological order. Fix some CR-version π' of π. Let $\pi'_i \subset \pi'$ be the set of rules whose head atoms belong to P_i. We compute S_0 as $Cn(\pi'_0)$. For $i > 0$, given the interpretation $S_{<i} = \bigcup_{j<i} S_j$ computed at lower strata, we evaluate π'_i as $S_i = Cn(\pi_i'^{S_{<i}})$, where $\pi_i'^{S_{<i}}$ is the reduct as per Definition 3. The interpretation of the entire program π is then $S = \bigcup_{i=0}^{k} S_i$. If S is consistent, let $W_{\pi'}^{strat} = S \cup \{\operatorname{not} l \mid l \in \mathcal{L}_\pi \setminus S\}$ be the stratified model of π'.

▶ **Theorem 32** (Stratified Fixpoint). *Let π be a stratified program in Choice-NF and π' any* CR*-version of π. Then the stratified model $W_{\pi'}^{strat}$ coincides with both the well-founded model of π' and every answer set of π'. In particular, π' has a unique answer set, and this answer set is total and equal to the well-founded model of π'.*

We can combine Theorem 30 and Theorem 32 to compute the stable model of a program π as follows. Let $\pi_0 = \pi$ and $W_0 = \emptyset$. Starting at $i = 0$, if π_i is empty, then $W_\pi = W_i$ is a stable model of π and the final element of the sequence. Otherwise, let U_i be a splitting set for π_i. If the bottom slice $b_{U_i}(\pi_i)$ is stratified, apply the stratified fixpoint construction to obtain S_i per Theorem 32; otherwise, use the regular alternating fixpoint computation to obtain S_i. Let π_{i+1} be the top slice $e_{U_i}(\pi_i, S_1)$ and let $W_{i+1} = W_i \cup S_i$.

[1] Replacing a positive literal with $\mathbf{u}$ can be performed on grounded programs; if the grounding occurs on the fly, then this replacement should not occur and we would add a rule $l \leftarrow \operatorname{not} l$ for such a literal.

This procedure *localizes* those portions of the program that require a full alternating fixpoint construction.

▶ **Example 33** (Stack of subprograms). Let $\pi = \pi_1 \cup \pi_2 \cup \pi_3$, where $\pi_1 = \{\mathtt{p} \leftarrow \mathrm{not}\,\mathtt{p}, \mathtt{p}.\}$, $\pi_2 = \{\mathtt{s} \leftarrow \mathrm{not}\,\mathtt{p}.,\ \mathtt{r} \leftarrow \mathtt{z}, \mathrm{not}\,\mathtt{s}.,\ \mathtt{z} \leftarrow \mathtt{r}., \mathtt{q} \leftarrow \neg\mathtt{p}\}$, and $\pi_3 = \{\bot \leftarrow \mathrm{not}\,\mathtt{m}, \mathtt{r}\}$. Here, π_1 is the bottom of π w.r.t. the splitting set $\{\mathtt{p}, \neg\mathtt{p}\}$ and π_2 the bottom of the remainder $\pi \setminus \pi_1$ w.r.t. the splitting set $\{\mathtt{r}, \neg\mathtt{r}, \mathtt{s}, \neg\mathtt{s}, \mathtt{z}, \neg\mathtt{z}\}$. Thus, π_1, π_2, π_3 form a bottom-up evaluation stack of subprograms of π. While π_1 is unstratified, π_2 is stratified and maximal, i.e., cannot be enlarged while remaining stratified; we thus obtain the following partitioning of literals: $P_{\pi_1} = \{p, \neg p\}$, $P^1_{\pi_2} = \{\mathtt{s}, \mathtt{q}, \neg\mathtt{s}, \neg\mathtt{q}\}$, $P^2_{\pi_2} = \{\mathtt{r}, \mathtt{z}, \neg\mathtt{r}, \neg\mathtt{z}\}$, where $P^1_{\pi_2}$, $P^2_{\pi_2}$ are *strata* of π_2.

Similarly to the computation of stratification, we can efficiently compute splitting sets for the whole program π by computing the strongly connected components of G_π while *ignoring* edge polarity and ensuring condition (ii) of Definition 29 by *adding* edges in both directions between p and $\neg p$ for each atom p. We can now compute a topological order of these strongly connected components. Any prefix of this sequence of components forms a splitting set of π.

6 Implementation and Experiments

We have implemented the algorithm described in the previous sections in C++, using CaDiCal [11] as the SAT solver. Our source code is publicly available: see [29].

Due to the novelty of the semantics, there is no established approach that we can directly compare this implementation against. Nevertheless, the WFSX$^{\mathrm{C}}$ semantics can be encoded in a natural and straightforward manner for an ASP solver. We experimentally test the performance of our SAT-based algorithm against an ASP encoding of the WFSX$^{\mathrm{C}}$ semantics (also available in [29]), on a graph problem which requires a three-valued solution.

Namely, we revisit the graph-winning game [35, 61], which is played by two players, A and B, on a directed graph $G = \langle V, E\rangle$ who alternate on *moves*, which must transit a token from a node to a neighbor. The *winning positions* are the nodes where no move of the token is possible. It is well known that WFS captures the problem with a single rule:

$$\mathtt{w}(X) \leftarrow \mathtt{arc}(X, Y), \mathrm{not}\,\mathtt{w}(Y).$$

where `arc` is a binary relation provided as a set of facts.

We extend the game with sets of arcs that we can add to or remove from the graph, represented by binary predicates `add` and `remove`, and a set of states that we want to be *winning states*. For each such state X, we add the rules

$$\mathtt{c} \leftarrow \mathrm{not}\,\mathtt{w}(X)., \quad \neg\mathtt{c} \leftarrow \mathtt{c}.,$$

which cause a conflict if X is not a winning state, and the integrity constraint

$$\mathtt{c} \vee \mathrm{not}\,\mathtt{c} \leftarrow \top.,$$

which rules out models in which $\mathtt{w}(X)$ and hence c is undefined.

▶ **Example 34** (Graph winning with choices). Consider the instance of extended graph-winning where (i) the set of arcs is $\{\mathtt{arc}(a,b), \mathtt{arc}(b,c), \mathtt{arc}(a,d), \mathtt{arc}(d,b)\}$, (ii) the possible additions are described by rules $\mathtt{arc}(d,e) \leftarrow \mathtt{add}(d,e).$, $\mathtt{arc}(e,d) \leftarrow \mathtt{add}(e,d).$ and choice facts $0\{\mathtt{add}(e,d)\}1.$, $0\{\mathtt{add}(d,e)\}1.$, (iii) no removals are possible, and (iv) the only required winning state is b, expressed by the rules $\mathtt{c} \leftarrow b.$ and $\neg\mathtt{c} \leftarrow \mathtt{c}.$ plus the integrity constraint $\mathtt{c} \vee \mathrm{not}\,\mathtt{c}$. Figure 1 shows three possible labelings of the nodes according to the following sets of choices (left to right): $\{\neg\mathtt{add}(e,d), \neg\mathtt{add}(d,e)\}$, $\{\mathtt{add}(e,d), \mathtt{add}(d,e)\}$, and $\{\neg\mathtt{add}(e,d), \mathtt{add}(d,e)\}$. Only the last one results in a model that satisfies the integrity constraint.

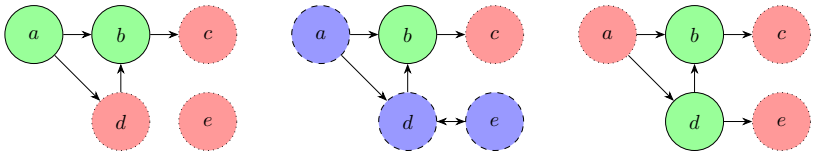

Figure 1 Different choice outcomes for Example 34. Dashed blue nodes mark undefined nodes, dotted red nodes mark losing nodes, and solid green nodes mark winning nodes.

Experimental setup

We generated random graphs with a parametric number of nodes, with edges chosen at random, and a subset of the edges was selected at random to be removable (by choice rules of the form $0\{\,\texttt{remove}(u,v)\,\}1.$) while other edges were randomly selected to be addable (by $0\{\,\texttt{add}(u,v)\,\}1.$). A small number of nodes was selected as winning states. We generated 50 graphs for each parameter set; all of the instances are also publicly available [29]. The parameters are as follows:

- 10 nodes, 30 edges (22 fixed, 4 addable, 4 removable), two fixed winning nodes,
- 15 nodes, 50 edges (40 fixed, 5 addable, 5 removable), two fixed winning nodes,
- 50 nodes, 580 edges (500 fixed, 40 addable, 40 removable), four fixed winning nodes,
- 150 nodes, 1360 edges (1200 fixed, 80 addable, 80 removable), five fixed winning nodes,
- 200 nodes, 2200 edges (2000 fixed, 100 addable, 100 removable), five fixed winning nodes,
- 250 nodes, 2740 edges (2500 fixed, 120 addable, 120 removable), five fixed winning nodes.

We ran all four evaluation approaches, i.e., (1) the ASP encoding (as solved by clingo 5.8.0 [33]), (2) the naive version of Algorithm 1, (3) its variant with Clause Learning (CL), and (4) the further extension with splitting via dependency graph and splitting sets, on all graphs. We conducted the experiments on a Linux 6.12 system with an Intel i7-1185G7 processor and 16 GiB of memory and with a timeout of 90 seconds per graph.

Experimental results

Table 2 show the resulting execution times (in seconds) and the number of timeouts. We observe significant differences between the four methods. Unsurprisingly, the exponential nature of the exhaustive search performed by Algorithm 1 without clause learning leads to timeouts except for the smallest numbers of nodes. Adding clause learning is crucial for timely termination, and while then the runtime still grows with the size of the instances, even the biggest instances were solved within seconds. We further see that splitting indeed succeeds in decomposing complex instances into smaller sub-problems that can be solved more easily, resulting in another significant reduction of runtime, especially for large number of nodes. The ASP encoding also performs well and, while our approach with clause learning (and splitting) generally appears faster, its runtime does not nearly increase as quickly as the one of the naive algorithm.

7 Conclusion and Future Work

We have presented a novel semantics for logic programs that augments Extended Normal Logic Programs with an explicit choice operator, thereby allowing case-based reasoning and non-deterministic choices while maintaining the scalability and skeptical reasoning capabilities of the Well Founded Semantics. To showcase the benefits for modeling problem,

Table 2 Performance comparison of evaluation methods on random instances of extended graph-winning, with all runtimes in seconds.

	Random graphs, 10 nodes			
Method	Best	Mean	Variance	Timeouts
ASP encoding	0.024	0.064	0.001	–
Naive	0.000	0.021	0.000	–
Clause learning (CL)	0.000	0.001	0.000	–
CL and splitting	0.000	0.001	0.000	–

	Random graphs, 15 nodes			
Method	Best	Mean	Variance	Timeouts
ASP encoding	0.058	0.140	0.003	–
Naive	0.000	0.152	0.002	–
Clause learning (CL)	0.000	0.001	0.000	–
CL and splitting	0.000	0.001	0.000	–

	Random graphs, 50 nodes			
Method	Best	Mean	Variance	Timeouts
ASP encoding	1.144	1.524	0.140	–
Naive	–	–	–	50
Clause learning (CL)	0.029	0.031	0.000	–
CL and splitting	0.008	0.009	0.000	–

	Random graphs, 150 nodes			
Method	Best	Mean	Variance	Timeouts
ASP encoding	13.190	16.676	8.850	–
Naive	–	–	–	50
Clause learning (CL)	1.906	2.534	0.667	–
CL and splitting	0.367	0.501	0.064	–

	Random graphs, 200 nodes			
Method	Best	Mean	Variance	Timeouts
ASP encoding	29.398	34.450	13.538	–
Naive	–	–	–	50
Clause learning (CL)	6.049	7.793	2.746	–
CL and splitting	1.188	1.586	0.519	–

	Random graphs, 250 nodes			
Method	Best	Mean	Variance	Timeouts
ASP encoding	59.663	66.626	38.103	5
Naive	–	–	–	50
Clause learning (CL)	17.576	22.890	19.923	–
CL and splitting	3.056	3.882	0.756	–

we have considered an application for reasoning about actions and planning. Furthermore, we have proposed an algorithm for evaluating logic programs according to the ELP^{C} semantics. Through the integration of a SAT solver for making choices, the efficient encoding of cardinality constraints, and – most importantly – the development of a clause learning procedure to propagate information about contradictory fixpoint results back to the SAT solver, this algorithm is capable of handling complex logic programs. In our experiments, it generally outperforms an ASP encoding of the same semantics, solved by a state-of-the-art ASP engine. We further have extended existing splitting techniques to ELP^{C} semantics and thereby enabled the decomposition of logic programs into stratified and unstratified components to aid in solving; our experiments confirmed a beneficial effect on the performance of evaluation.

Our work gives rise to several promising research directions. Recently, there has been an increasing interest in the integration of first-order theories into logic programs, with semantics being developed to facilitate such an integration [15, 16, 27, 41], as different semantics capture different intuitive behavior and use cases. A prominent example is *difference logic* [10, 21], which is a tractable fragment of linear arithmetic that plays a central role in planning [49] with durative action and scheduling [41] under time constraints, while having partial information on the truth value for the fluents. The ELP^{C} semantics that we presented is well-suited for an SMT-like integration of first-order theories.

Lastly, while the algorithm presented in this work directly interfaces a SAT solver, it would likely benefit from an even tighter integration, e.g., through the recently proposed *Satisfiability Modulo User Propagators* framework [30], in which case conflicts could be derived and clauses could be learned from partial SAT assignments.

References

1 João F. L. Alcântara, Carlos Viegas Damásio, and Luís Moniz Pereira. A well-founded semantics with disjunction. In Maurizio Gabbrielli and Gopal Gupta, editors, *Logic Programming, 21st International Conference, ICLP 2005, Sitges, Spain, October 2-5, 2005, Proceedings*, volume 3668 of *Lecture Notes in Computer Science*, pages 341–355. Springer, Springer, 2005. doi:10.1007/11562931_26.

2 José Júlio Alferes. *Semantics of logic programs with explicit negation*. PhD thesis, Universidade Nova de Lisboa, Lisbon, Portugal, 1993. URL: http://hdl.handle.net/10362/1056.

3 José Júlio Alferes, Carlos Viegas Damásio, and Luís Moniz Pereira. A logic programming system for nonmonotonic reasoning. *Journal of Automated Reasoning*, 14:93–147, 1995. doi:10.1007/BF00883931.

4 Mario Alviano, Francesco Calimeri, Wolfgang Faber, Nicola Leone, and Simona Perri. Unfounded sets and well-founded semantics of answer set programs with aggregates. *Journal of Artificial Intelligence Research*, 42:487–527, 2011. doi:10.1613/jair.3432.

5 Mario Alviano, Carmine Dodaro, Wolfgang Faber, Nicola Leone, and Francesco Ricca. WASP: A native ASP solver based on constraint learning. In Pedro Cabalar and Tran Cao Son, editors, *Logic Programming and Nonmonotonic Reasoning*, volume 8148 of *Lecture Notes in Computer Science*, pages 54–66. Springer, Springer, 2013. doi:10.1007/978-3-642-40564-8_6.

6 Daneshvar Amrollahi, Hossein Hojjat, and Philipp Rümmer. An encoding for CLP problems in SMT-LIB. In Temur Kutsia, Daniel Ventura, David Monniaux, and José F. Morales, editors, *Proceedings 18th International Workshop on Logical and Semantic Frameworks, with Applications and 10th Workshop on Horn Clauses for Verification and Synthesis, LSFA/HCVS 2023, and 10th Workshop on Horn Clauses for Verification and Synthesis, Rome, Italy & Paris, France, July 1-2, 2023 & April 23, 2023*, volume 402, pages 118–130. Open Publishing Association, April 2023. doi:10.4204/eptcs.402.12.

7 Joaquín Arias, Manuel Carro, Elmer Salazar, Kyle Marple, and Gopal Gupta. Constraint answer set programming without grounding. *Theory and Practice of Logic Programming*, 18(3-4):337–354, 2018. doi:10.1017/S1471068418000285.

8 Chitta Baral, Vladik Kreinovich, and Raul Trejo. Computational complexity of planning and approximate planning in the presence of incompleteness. *Artificial Intelligence*, 122(1-2):241–267, 2000. doi:10.1016/S0004-3702(00)00043-6.

9 Chitta Baral, Jorge Lobo, and Jack Minker. Generalized disjunctive well-founded semantics for logic programs. *Annals of Mathematics and Artificial Intelligence*, 5(2):89–131, 1992. doi:10.1007/BF01543473.

10 Clark W. Barrett and Cesare Tinelli. Satisfiability modulo theories. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*, pages 305–343. Springer, 2018. doi:10.1007/978-3-319-10575-8_11.

11 Armin Biere, Tobias Faller, Katalin Fazekas, Mathias Fleury, Nils Froleyks, and Florian Pollitt. CaDiCaL 2.0. In Arie Gurfinkel and Vijay Ganesh, editors, *Computer Aided Verification - 36th International Conference, CAV 2024, Montreal, QC, Canada, July 24-27, 2024, Proceedings, Part I*, volume 14681 of *Lecture Notes in Computer Science*, pages 133–152. Springer, Springer, 2024. doi:10.1007/978-3-031-65627-9_7.

12 Roland N. Bol and Lars Degerstedt. Tabulated resolution for the well-founded semantics. *The Journal of Logic Programming*, 34(2):67–109, 1998. doi:10.1016/S0743-1066(97)00073-3.

13 Marco Bozzano, Alessandro Cimatti, Alberto Griggio, Martin Jonás, and Greg Kimberly. Analysis of cyclic fault propagation via ASP. In Georg Gottlob, Daniela Inclezan, and Marco Maratea, editors, *Logic Programming and Nonmonotonic Reasoning - 16th International Conference, LPNMR 2022, Genova, Italy, September 5-9, 2022, Proceedings*, volume 13416 of *Lecture Notes in Computer Science*, pages 470–483. Springer, Springer, 2022. doi:10.1007/978-3-031-15707-3_36.

14 Gerhard Brewka, Ilkka Niemelä, and Tommi Syrjänen. Logic programs with ordered disjunction. *Computational Intelligence*, 20(2):335–357, 2004. doi:10.1111/j.0824-7935.2004.00241.x.

15 Pedro Cabalar, Jorge Fandinno, Torsten Schaub, and Sebastian Schellhorn. Lower bound founded logic of here-and-there. In Francesco Calimeri, Nicola Leone, and Marco Manna, editors, *Logics in Artificial Intelligence - 16th European Conference, JELIA 2019, Rende, Italy, May 7-11, 2019, Proceedings*, volume 11468 of *Lecture Notes in Computer Science*, pages 509–525. Springer, Springer, 2019. doi:10.1007/978-3-030-19570-0_34.

16 Pedro Cabalar, Roland Kaminski, Max Ostrowski, and Torsten Schaub. An ASP semantics for default reasoning with constraints. In Subbarao Kambhampati, editor, *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 1015–1021. IJCAI/AAAI Press, 2016. URL: http://www.ijcai.org/Abstract/16/148.

17 Pedro Cabalar, Sergei P. Odintsov, and David Pearce. Logical foundations of well-founded semantics. In Patrick Doherty, John Mylopoulos, and Christopher A. Welty, editors, *Proceedings, Tenth International Conference on Principles of Knowledge Representation and Reasoning, Lake District of the United Kingdom, June 2-5, 2006*, volume 6, pages 25–35. AAAI Press, 2006. URL: https://aaai.org/papers/KR06-006-logical-foundations-of-well-founded-semantics/.

18 Weidong Chen and David Scott Warren. A goal-oriented approach to computing the well-founded semantics. *The Journal of Logic Programming*, 17(2):279–300, 1993. Special Issue: Non-Monotonic Reasoning and Logic Programming. doi:10.1016/0743-1066(93)90034-E.

19 Weidong Chen and David Scott Warren. Query evaluation under the well founded semantics. In Catriel Beeri, editor, *Proceedings of the Twelfth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, PODS '93, pages 168–179, New York, NY, USA, 1993. Association for Computing Machinery. doi:10.1145/153850.153865.

20 Leroy Chew, Alexis de Colnet, and Stefan Szeider. ASP-QRAT: A conditionally optimal dual proof system for ASP. In Pierre Marquis, Magdalena Ortiz, and Maurice Pagnucco, editors, *Proceedings of the 21st International Conference on Principles of Knowledge Representation and Reasoning, KR 2024, Hanoi, Vietnam. November 2-8, 2024*, 2024. doi:10.24963/kr.2024/24.

21 Scott Cotton and Oded Maler. Fast and flexible difference constraint propagation for DPLL(T). In Armin Biere and Carla P. Gomes, editors, *Theory and Applications of Satisfiability Testing - SAT 2006, 9th International Conference, Seattle, WA, USA, August 12-15, 2006, Proceedings*, volume 4121 of *Lecture Notes in Computer Science*, pages 170–183. Springer, Springer, 2006. doi:10.1007/11814948_19.

22 Marc Denecker and Eugenia Ternovska. A logic of nonmonotone inductive definitions. *ACM transactions on computational logic (TOCL)*, 9(2):1–52, 2008. doi:10.1145/1342991.1342998.

23 Wlodzimierz Drabent and Jan Maluszynski. Hybrid rules with well-founded semantics. *Knowledge and information systems*, 25(1):137–168, 2010. doi:10.1007/s10115-010-0300-5.

24 Thomas Eiter, Wolfgang Faber, Nicola Leone, Gerald Pfeifer, and Axel Polleres. A logic programming approach to knowledge-state planning, II: the dlv$^{\mathrm{k}}$ system. *Artificial Intelligence*, 144(1-2):157–211, 2003. doi:10.1016/S0004-3702(02)00367-3.

25 Thomas Eiter, Giovambattista Ianni, and Thomas Krennwallner. Answer Set Programming: A primer. In Sergio Tessaris, Enrico Franconi, Thomas Eiter, Claudio Gutierrez, Siegfried Handschuh, Marie-Christine Rousset, and Renate A. Schmidt, editors, *Reasoning Web. Semantic Technologies for Information Systems, 5th International Summer School 2009, Brixen-Bressanone, Italy, August 30 - September 4, 2009, Tutorial Lectures*, volume 5689 of *Lecture Notes in Computer Science*, pages 40–110. Springer, 2009. doi:10.1007/978-3-642-03754-2_2.

26 Thomas Eiter, Giovambattista Ianni, Thomas Lukasiewicz, and Roman Schindlauer. Well-founded semantics for description logic programs in the semantic web. *ACM Transactions on Computational Logic (TOCL)*, 12(2):11:1–11:41, 2011. doi:10.1145/1877714.1877717.

27 Thomas Eiter and Rafael Kiesel. ASP ($\mathcal{AC}$): Answer set programming with algebraic constraints. *Theory and Practice of Logic Programming*, 20(6):895–910, 2020. doi:10.1017/S1471068420000393.

28 Thomas Eiter, Nicola Leone, and Domenico Sacca. On the partial semantics for disjunctive deductive databases. *Annals of Mathematics and Artificial Intelligence*, 19(1):59–96, 1997. doi:10.1023/A:1018947420290.

29 Thomas Eiter, Tobias Nießen, and Davide Soldà. SAT modulo well-founded semantics (SAT 2026 Software Implementation), May 2026. doi:10.5281/zenodo.20122760.

30 Katalin Fazekas, Aina Niemetz, Mathias Preiner, Markus Kirchweger, Stefan Szeider, and Armin Biere. Satisfiability modulo user propagators. *Journal of Artificial Intelligence Research*, 81:989–1017, December 2024. doi:10.1613/jair.1.16163.

31 Martin Gebser, Tomi Janhunen, and Jussi Rintanen. Answer set programming as SAT modulo acyclicity. In Torsten Schaub, Gerhard Friedrich, and Barry O'Sullivan, editors, *ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014)*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, pages 351–356. IOS Press, IOS Press, 2014. doi:10.3233/978-1-61499-419-0-351.

32 Martin Gebser, Tomi Janhunen, and Jussi Rintanen. Sat modulo graphs: Acyclicity. In *European Workshop on Logics in Artificial Intelligence*, pages 137–151. Springer, 2014. doi:10.1007/978-3-319-11558-0_10.

33 Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. Multi-shot ASP solving with clingo. *Theory and Practice of Logic Programming*, 19(1):27–82, 2019. doi:10.1017/S1471068418000054.

34 Martin Gebser, Benjamin Kaufmann, and Torsten Schaub. Conflict-driven answer set solving: From theory to practice. *Artificial Intelligence*, 187:52–89, 2012. doi:10.1016/j.artint.2012.04.001.

35 Allen Van Gelder, Kenneth A. Ross, and John S. Schlipf. Unfounded sets and well-founded semantics for general logic programs. In Chris Edmondson-Yurkanan and Mihalis Yannakakis, editors, *Proceedings of the Seventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, March 21-23, 1988, Austin, Texas, USA*, pages 221–230. ACM, 1988. doi:10.1145/308386.308444.

36 Allen Van Gelder, Kenneth A. Ross, and John S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM (JACM)*, 38(3):620–650, 1991. doi:10.1145/116825.116838.

37 Michael Gelfond and Vladimir Lifschitz. Logic programs with classical negation. In David H. D. Warren and Péter Szeredi, editors, *Logic Programming, Proceedings of the Seventh International Conference*, pages 579–597, Jerusalem, Israel, 1990. MIT Press.

38 Enrico Giunchiglia, Yuliya Lierler, and Marco Maratea. Answer set programming based on propositional satisfiability. *Journal of Automated reasoning*, 36(4):345–377, 2006. doi:10.1007/s10817-006-9033-2.

39 Arie Gurfinkel. Program verification with constrained horn clauses (invited paper). In Sharon Shoham and Yakir Vizel, editors, *Computer Aided Verification - 34th International Conference, CAV 2022, Haifa, Israel, August 7-10, 2022, Proceedings, Part I*, volume 13371 of *Lecture Notes in Computer Science*, pages 19–29. Springer, 2022. doi:10.1007/978-3-031-13185-1_2.

40 Krystof Hoder, Nikolaj S. Bjørner, and Leonardo Mendonça de Moura. μZ- an efficient engine for fixed points with constraints. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, volume 6806 of *Lecture Notes in Computer Science*, pages 457–462, Berlin, Heidelberg, 2011. Springer. doi:10.1007/978-3-642-22110-1_36.

41 Tomi Janhunen, Roland Kaminski, Max Ostrowski, Sebastian Schellhorn, Philipp Wanko, and Torsten Schaub. Clingo goes linear constraints over reals and integers. *Theory and Practice of Logic Programming*, 17(5-6):872–888, 2017. doi:10.1017/S1471068417000242.

42 Roland Kaminski and Torsten Schaub. On the foundations of grounding in answer set programming. *Theory and Practice of Logic Programming*, 23(6):1138–1197, 2023. doi:10.1017/s1471068422000308.

43 Matthias Knorr, José Júlio Alferes, and Pascal Hitzler. A well-founded semantics for hybrid MKNF knowledge bases. In Diego Calvanese, Enrico Franconi, Volker Haarslev, Domenico Lembo, Boris Motik, Anni-Yasmin Turhan, and Sergio Tessaris, editors, *Proceedings of the 2007 International Workshop on Description Logics (DL2007), Brixen-Bressanone, near Bozen-Bolzano, Italy, 8-10 June, 2007*, volume 250 of *CEUR Workshop Proceedings*, 2007. URL: `https://ceur-ws.org/Vol-250/paper_54.pdf`.

44 Vladimir Lifschitz and Hudson Turner. Splitting a logic program. In Pascal Van Hentenryck, editor, *Proceedings of the Eleventh International Conference on Logic Programming*, volume 94, pages 23–37. MIT Press, 1994.

45 Vladimir Lifschitz and Hudson Turner. Splitting a logic program. In Pascal Van Hentenryck, editor, *Proceedings of the Eleventh International Conference on Logic Programming*, pages 23–37, Santa Margherita Ligure, Italy, 1994. MIT Press.

46 Fangzhen Lin and Yuting Zhao. ASSAT: computing answer sets of a logic program by SAT solvers. *Artificial Intelligence*, 157(1-2):115–137, 2004. `doi:10.1016/j.artint.2004.04.004`.

47 Zhijun Lin, Yuanlin Zhang, and Hector Hernandez. Fast SAT-based Answer Set Solver. In *Proceedings, The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference, July 16-20, 2006, Boston, Massachusetts, USA*, volume 21, pages 92–97. Menlo Park, CA; Cambridge, MA; London; AAAI Press, AAAI Press, 2006. URL: `https://aaai.org/papers/00092-AAAI06-015-fast-sat-based-answer-set-solver/`.

48 Maarten Mariën, Johan Wittocx, and Marc Denecker. Integrating inductive definitions in sat. In *International Conference on Logic for Programming Artificial Intelligence and Reasoning*, pages 378–392. Springer, 2007. `doi:10.1007/978-3-540-75560-9_28`.

49 Veena S Mellarkod and Michael Gelfond. Enhancing ASP systems for planning with temporal constraints. In *International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 309–314. Springer, 2007. `doi:10.1007/978-3-540-72200-7_31`.

50 Ilkka Niemelä, Patrik Simons, and Timo Soininen. Stable model semantics of weight constraint rules. In Michael Gelfond, Nicola Leone, and Gerald Pfeifer, editors, *Logic Programming and Nonmonotonic Reasoning, 5th International Conference, LPNMR'99, El Paso, Texas, USA, December 2-4, 1999, Proceedings*, volume 1730 of *Lecture Notes in Computer Science*, pages 317–331. Springer, Springer, 1999. `doi:10.1007/3-540-46767-X_23`.

51 Andreas Niskanen, Masood Feyzbakhsh Rankooh, Tuomo Lehtonen, and Matti Järvisalo. Reasoning in assumption-based argumentation via SAT. In Magdalena Ortiz, Renata Wassermann, and Torsten Schaub, editors, *Proceedings of the 22nd International Conference on Principles of Knowledge Representation and Reasoning, KR 2025, Melbourne, Australia, November 1-17, 2025*, volume 22, pages 707–717, 2025. `doi:10.24963/kr.2025/68`.

52 Héctor Palacios and Hector Geffner. Compiling uncertainty away in conformant planning problems with bounded width. *Journal of Artificial Intelligence Research*, 35:623–675, 2009. `doi:10.1613/jair.2708`.

53 Alessandro Dal Palù, Agostino Dovier, Enrico Pontelli, and Gianfranco Rossi. GASP: answer set programming with lazy grounding. *Fundamenta Informaticae*, 96(3):297–322, 2009. `doi:10.3233/FI-2009-180`.

54 Luís Moniz Pereira and José Júlio Alferes. Well founded semantics for logic programs with explicit negation. In Bernd Neumann, editor, *Proceedings of the 10th European Conference on Artificial Intelligence*, ECAI '92, pages 102–106, USA, 1992. John Wiley and Sons.

55 Masood Feyzbakhsh Rankooh, Andreas Niskanen, and Matti Järvisalo. Cost-optimal delete-free classical planning via maximum satisfiability. In Magdalena Ortiz, Renata Wassermann, and Torsten Schaub, editors, *Proceedings of the 22nd International Conference on Principles of Knowledge Representation and Reasoning, KR 2025, Melbourne, Australia, November 1-17, 2025*, volume 22, pages 783–793, 2025. `doi:10.24963/kr.2025/75`.

56 Masood Feyzbakhsh Rankooh and Jussi Rintanen. Propositional encodings of acyclicity and reachability by using vertex elimination. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 5861–5868, 2022. doi:10.1609/aaai.v36i5.20530.

57 Kenneth A. Ross. The well founded semantics for disjunctive logic programs. In Won Kim, Jean-Marie Nicolas, and Shojiro Nishio, editors, *Deductive and Object-Oriented Databases, Proceedings of the First International Conference on Deductive and Object-Oriented Databases (DOOD'89), Kyoto Research Park, Kyoto, Japan, 4-6 December, 1989*, pages 385–402. North-Holland/Elsevier Science Publishers, 1989.

58 Konstantinos Sagonas, Terrance Swift, and David Scott Warren. XSB as an efficient deductive database engine. In Richard T. Snodgrass and Marianne Winslett, editors, *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data, Minneapolis, Minnesota, USA, May 24-27, 1994*, pages 442–453. ACM New York, NY, USA, 1994. doi:10.1145/191839.191927.

59 Carsten Sinz. Towards an optimal CNF encoding of boolean cardinality constraints. In Peter van Beek, editor, *Principles and Practice of Constraint Programming - CP 2005, 11th International Conference, CP 2005, Sitges, Spain, October 1-5, 2005, Proceedings*, volume 3709 of *Lecture Notes in Computer Science*, pages 827–831, Berlin, Heidelberg, 2005. Springer. doi:10.1007/11564751_73.

60 Davide Soldà and Thomas Eiter. deon-B: A language for well-founded deontic planning. In Giovanni Casini, Besik Dundua, and Temur Kutsia, editors, *Logics in Artificial Intelligence - 19th European Conference, JELIA 2025, Kutaisi, Georgia, September 1-4, 2025, Proceedings, Part I*, volume 16093 of *Lecture Notes in Computer Science*, pages 187–204. Springer, Springer, 2025. doi:10.1007/978-3-032-04587-4_12.

61 M. H. van Emden and K. L. Clark. The logic of two-person games. In K. L. Clark and F. G. McCabe, editors, *Micro-PROLOG: Programming in Logic*, pages 320–340. Prentice-Hall International, 1984.

62 Kewen Wang and Lizhu Zhou. Comparisons and computation of well-founded semantics for disjunctive logic programs. *ACM Transactions on Computational Logic (TOCL)*, 6(2):295–327, 2005. doi:10.1145/1055686.1055690.

63 Jeffrey Ward and John S. Schlipf. Answer set programming with clause learning. In Vladimir Lifschitz and Ilkka Niemelä, editors, *Logic Programming and Nonmonotonic Reasoning, 7th International Conference, LPNMR 2004, Fort Lauderdale, FL, USA, January 6-8, 2004, Proceedings*, volume 2923 of *Lecture Notes in Computer Science*, pages 302–313. Springer, Springer, 2004. doi:10.1007/978-3-540-24609-1_26.

64 Carlo Zaniolo, Natraj Arni, and KayLiang Ong. Negation and aggregates in recursive rules: the LDL++ approach. In Stefano Ceri, Katsumi Tanaka, and Shalom Tsur, editors, *Deductive and Object-Oriented Databases, Third International Conference, DOOD'93, Phoenix, Arizona, USA, December 6-8, 1993, Proceedings*, volume 760 of *Lecture Notes in Computer Science*, pages 204–221. Springer, Springer, 1993. doi:10.1007/3-540-57530-8_13.