Advanced Preprocessing for Answer Set Solving

Martin Gebser and **Benjamin Kaufmann** and **André Neumann** and **Torsten Schaub**[1,2]

Abstract. We introduce the first substantial approach to preprocessing in the context of answer set solving. The idea is to simplify a logic program while identifying equivalences among its relevant constituents. These equivalences are then used for building a compact representation of the program (in terms of Boolean constraints). We implemented our approach as well as a SAT-based technique to reduce Boolean constraints. This allows us to empirically analyze both preprocessing types and to demonstrate their computational impact.

1 INTRODUCTION

Answer Set Programming (ASP; [3]) has become an attractive paradigm for declarative problem solving. This is partly due to the availability of efficient off-the-shelf ASP solvers [9, 19]. In fact, modern ASP solvers rely on Boolean constraint solving technology [1, 8, 7], leading to a similar performance as advanced SAT solvers [17]. On the other hand, the attractiveness of ASP stems from its rich modeling language, allowing for an easy and elaboration-tolerant handling of knowledge-intensive applications. In practice, an input program is usually run through multiple preprocessing steps. At first, a so-called grounder instantiates all variables, thus producing a ground logic program. Classical ASP solvers, such as *smodels* [19], more or less take the resulting program as is without doing further optimizations. In contrast, modern ASP solvers translate a ground program into a set of Boolean constraints (e.g., clauses) in order to exploit advanced SAT solving technology. Such translations necessitate the introduction of extra propositions (see below) in order to avoid an exponential blow-up. Also, this addition may result in exponentially smaller search spaces [16] and permits more succinct representations of loop constraints [14]. Nonetheless, the question arises in how far the introduced redundancy can be trimmed.

While ASP solvers still lack full-fledged preprocessing techniques, they already constitute an integral part of many SAT solvers [2, 20, 10]. There are two principal ways to address preprocessing in ASP solving: the external one, aiming at the reduction of a ground program, and the internal one, (recurrently) optimizing its inner representation. Within modern ASP solvers, the latter can be done by adapting corresponding techniques from SAT. Hence, we concentrate in the sequel on the former approach, being specific to ASP. Thereby, we build upon work on program transformations and equivalence [4, 5, 11]. To be precise, we develop preprocessing techniques for ground logic programs under answer set semantics. The idea is to transform a program into a simpler one, along with an assignment and a relation expressing equivalences among the assignable constituents of the program. These equivalences are subsequently exploited when transforming the resulting program into Boolean constraints, represented as clauses. We implemented both our external and a SAT-based internal reduction strategy within the ASP solver *clasp* [7]. This makes *clasp* the first ASP solver incorporating advanced preprocessing techniques. Furthermore, our implementation allows us to empirically assess both the external and the internal approach to preprocessing, thus demonstrating their computational impact.

[1] Affiliated with SFU, Canada, and Griffith University, Australia.
[2] Universität Potsdam, August-Bebel-Str. 89, D-14482 Potsdam, Germany

2 BACKGROUND

A (*normal*) *logic program* over an alphabet $\mathcal{A}$ is a finite multiset[3] of *rules* of the form $a \leftarrow b_1, \ldots, b_m, \sim c_{m+1}, \ldots, \sim c_n$, where $a, b_i, c_j \in \mathcal{A}$ are *atoms* for $0 < i \leq m, m < j \leq n$. A *literal* is an atom a or its (default) negation $\sim a$. Furthermore, let $\sim\mathcal{A} = \{\sim a \mid a \in \mathcal{A}\}$ and $\overline{\mathcal{A}} = \{\overline{a} \mid a \in \mathcal{A}\}$, where $\overline{a}$ is used for (classical) negation in propositional formulas. For a rule r, let $head(r) = a$ be the *head* of r and the multiset $body(r) = \{b_1, \ldots, b_m, \sim c_{m+1}, \ldots, \sim c_n\}$ be the *body* of r. Given a (multi)set B of literals, let $B^+ = \{a \in \mathcal{A} \mid a \in B\}$ and $B^- = \{a \in \mathcal{A} \mid \sim a \in B\}$. The set of atoms occurring in a logic program Π is denoted by $atom(\Pi)$ and $body(\Pi) = \{body(r) \mid r \in \Pi\}$. Also, we define $body(a) = \{body(r) \mid r \in \Pi, head(r) = a\}$.

Following [18], we characterize the answer sets of a logic program Π by the models of the completion [6] and loop formulas of Π. As mentioned above, in practice, this involves introducing extra propositions p_B for bodies B. Given a program Π over $\mathcal{A}$, its *completion formula* is then defined as follows:

$$CF(\Pi, \mathcal{A}) = \big\{a \leftrightarrow \big(\bigvee_{B \in body(a)} p_B\big) \mid a \in \mathcal{A}\big\} \cup \big\{p_B \leftrightarrow \big(\bigwedge_{b \in B^+} b \wedge \bigwedge_{c \in B^-} \overline{c}\big) \mid B \in body(\Pi)\big\} . \quad (1)$$

A *loop* is a (nonempty) set of atoms that circularly depend upon each other in a program's positive atom dependency graph [18]. The set of all loops of Π is denoted by $loop(\Pi)$. If $loop(\Pi) = \emptyset$, then Π is said to be *tight* [12]. The *loop formula* of some $L \in loop(\Pi)$ is

$$LF(\Pi, L) = \big(\bigvee_{a \in L} a\big) \rightarrow \big(\bigvee_{a \in L, B \in body(a), B^+ \cap L = \emptyset} p_B\big) ,$$

and $LF(\Pi) = \{LF(\Pi, L) \mid L \in loop(\Pi)\}$. The bodies contributing to the consequent of a loop formula provide *external support* for the antecedent's atoms. An atom is said to be *unfounded* if it belongs to the antecedent of a loop formula whose consequent is $\bot$, expressing the absence of external support.

We represent (classical) models by their set of entailed propositions, and let $\mathcal{M}(F)$ stand for the set of all models of F. For some alphabet $\mathcal{A}$, we define $\mathcal{M}(F)|_{\mathcal{A}} = \{M \cap \mathcal{A} \mid M \in \mathcal{M}(F)\}$. Then, a set $X \subseteq \mathcal{A}$ is an *answer set* of a logic program Π over $\mathcal{A}$ if $X \in \mathcal{M}(CF(\Pi, \mathcal{A}) \cup LF(\Pi))|_{\mathcal{A}}$. We let $AS(\Pi)$ denote the set of all answer sets of Π. Note that, whenever Π is tight, we have $X \in AS(\Pi)$ iff $X \in \mathcal{M}(CF(\Pi, \mathcal{A}))|_{\mathcal{A}}$.

Consider the following program Π over $\mathcal{A} = \{a, \ldots, f\}$:

$$\{a \leftarrow;\ b \leftarrow a, \sim c;\ c \leftarrow \sim b, \sim d;\ e \leftarrow \sim c;\ e \leftarrow f;\ f \leftarrow a, e\} .$$

We get the following completion formula, $CF(\Pi, \mathcal{A})$:

$$\{a \leftrightarrow p_0;\ b \leftrightarrow p_1;\ c \leftrightarrow p_2;\ d \leftrightarrow \bot;\ e \leftrightarrow p_3 \vee p_4;\ f \leftrightarrow p_5\} \cup \{p_0 \leftrightarrow \top;\ p_1 \leftrightarrow a \wedge \overline{c};\ p_2 \leftrightarrow \overline{b} \wedge \overline{d};\ p_3 \leftrightarrow \overline{c};\ p_4 \leftrightarrow f;\ p_5 \leftrightarrow a \wedge e\} .$$

[3] The usage of multisets is motivated by the syntactic nature of our approach and the fact that grounders produce duplicates. For simplicity, we keep standard set notation for multiset operations.

$CF(\Pi, \mathcal{A})$ has three models: $\{a, b, e, f, p_0, p_1, p_3, p_4, p_5\}$, $\{a, c, p_0, p_2\}$, and $\{a, c, e, f, p_0, p_2, p_4, p_5\}$. Furthermore, program Π has one loop, $\{e, f\}$, yielding $LF(\Pi) = \{e \vee f \rightarrow p_3\}$. This loop formula is falsified by $\{a, c, e, f, p_0, p_2, p_4, p_5\}$, thus $\{a, c, e, f\}$ is no answer set of Π. The other two models of $CF(\Pi, \mathcal{A})$ satisfy $LF(\Pi)$ and correspond to the answer sets $\{a, b, e, f\}$ and $\{a, c\}$ of Π.

Finally, a (*partial*) *Boolean assignment* A over $\mathcal{A} \cup 2^{\mathcal{A} \cup \sim\mathcal{A}}$ is a set of possibly negated elements of its domain. We define $\overline{A} = \{a \in \mathcal{A} \mid \overline{a} \in A\} \cup \{B \subseteq \mathcal{A} \cup \sim\mathcal{A} \mid \overline{B} \in A\}$. For instance, $A = \{a, \overline{d}, \overline{\{a, \sim c\}}\}$ assigns *true* to a and *false* to d as well as body $\{a, \sim c\}$, and $\overline{A} = \{d, \{a, \sim c\}\}$ contains all false elements of A.

3 PREPROCESSING

Our initial goal is to turn a given program Π over an alphabet $\mathcal{A}$ into a simplified program Π', a partial assignment A, and an equivalence relation $\mathcal{E}$ on the atoms and bodies in Π'. More formally, we transform a triple $(\Pi, \emptyset, \emptyset)$ into $(\Pi', A, \mathcal{E})$. Thereby, Π' is obtained from Π by program transformations, mainly involving rule eliminations and body modifications. The semantics of the original program Π is captured by Π' along with assignment A and $\mathcal{E}$, where the latter is also exploited to generate a compact representation of Π' in terms of Boolean constraints. Our transformation rules, shown in Table 1, are grouped into four building blocks: $s = \{(s_0), \ldots, (s_{15})\}$, $e = \{(e_{16}), \ldots, (e_{27})\}$, $a = \{(a_{28}), \ldots, (a_{35})\}$, and $u = \{(u_{36})\}$. (Note that many of them are subject to conditions, given in the rightmost column.) Roughly, the rules in s permit elementary simplifications, while e partitions atoms and bodies into equivalence classes. As a byproduct of this, all unclassified atoms are unfounded and set to false via (u_{36}). Finally, the rules in a substitute the atoms in an equivalence class by a unique representative for that class. Note that s, e, a, and u are intended to be applied till saturation before proceeding to another block of transformations. In what follows, we gradually explain the different transformations and also provide examples.

To begin with, rules (s_0) to (s_{10}) build upon well-known program transformations [4, 5, 11]. Let $T \mapsto^* T'$ represent the computation of a fixpoint T' by repeated applications of $\mapsto$ to T. Then, $\overset{s}{\mapsto}^*$ amounts to computing the fixpoint of Fitting's operator [13]. In addition, $\overset{s}{\mapsto}^*$ makes assignments to bodies and simplifies the program at hand. Finally, rules (s_{11}) to (s_{15}) preserve the correspondence between the program Π and its associated assignment A. For $\Pi_0 = \{a \leftarrow;\ b \leftarrow a, \sim c;\ c \leftarrow \sim b, \sim d\}$, we get $(\Pi_0, \emptyset, \emptyset) \overset{s}{\mapsto}^* (\Pi_1, A_1, \emptyset)$, where $\Pi_1 = \{b \leftarrow \sim c;\ c \leftarrow \sim b\}$ and $A_1 = \{a, \overline{d}\}$.

In general, a fixpoint of $\overset{s}{\mapsto}$ has the following syntactic properties.

Proposition 1 *Let* $(\Pi, \emptyset, \emptyset) \overset{s}{\mapsto}^* (\Pi', A, \emptyset)$*, for logic program* Π *over alphabet* $\mathcal{A}$*. Then, we have:*

1. $body(r) \neq \emptyset$*, for all* $r \in \Pi'$*;*
2. $body(a) \neq \emptyset$*, for all* $a \in atom(\Pi')$*;*
3. $(atom(\Pi') \cup body(\Pi')) \cap (A \cup \overline{A}) = \emptyset$*;*
4. $A \cap \overline{A} = \emptyset$*;*
5. $\{B \subseteq \mathcal{A} \cup \sim\mathcal{A} \mid B \in A \cup \overline{A}\} \subseteq \overline{A}$*;*
6. $\bigcup_{B \in \overline{A} \setminus \mathcal{A}} (B^+ \cup B^-) \subseteq atom(\Pi')$.

Using $BF(Y) = \{(\bigvee_{b \in B^+} \overline{b} \vee \bigvee_{c \in B^-} c) \mid B \in Y\}$, we can capture the relationship between the original program Π and the reduced program Π' along with assignment A as follows.

Proposition 2 *Let* $(\Pi, \emptyset, \emptyset) \overset{s}{\mapsto}^* (\Pi', A, \emptyset)$*, for logic program* Π *over alphabet* $\mathcal{A}$*. Then, we have*

$$AS(\Pi) = \mathcal{M}(CF(\Pi', \mathcal{A} \setminus A) \cup LF(\Pi') \cup (A \cap \mathcal{A}) \cup BF(\overline{A} \setminus \mathcal{A}))|_{\mathcal{A}}\ .$$

Rules (e_{16}) to (e_{27}) comprise the heart of our approach and build an equivalence relation on atoms and bodies. We represent *equivalence classes* as triples, viz., $E = [a, B, C]$, where a is an atom representative for E, B is a body (externally) supporting E, and C contains all atoms and bodies belonging to E. We denote the components of E by $a_E = a$, $B_E = B$, and $C_E = C$. Thereby, $\varnothing$ denotes a null value, where $a_E = \varnothing$ means that $C_E \cap \mathcal{A} = \emptyset$ and $B_E = \varnothing$ expresses that E is not (externally) supported. For a set $\mathcal{E}$ of equivalence classes, define:[4]

$$\mathcal{E}_C^s = \bigcup_{[a,B,C] \in \mathcal{E}, B \neq \varnothing} C \qquad \mathcal{E}_C = \bigcup_{[a,B,C] \in \mathcal{E}} C$$
$$\mathcal{E}_B^s = \bigcup_{[a,B,C] \in \mathcal{E}, B \neq \varnothing} B^+\ .$$

Some classes in $\mathcal{E}$ are defined as *dual* to each other (and are finally represented by complementary propositional literals). In Table 1, the rules (e_{16}) and (e_{17}) each introduce a new equivalence class E along with its dual class $\widetilde{E}$, and we assume both classes to be correlated via some unique name (e.g., $E_1, \widetilde{E}_1; E_2, \widetilde{E}_2; \ldots$). Finally, we use $\widetilde{E}$ to denote the dual class of E, and let $\widetilde{\widetilde{E}} = E$.

Let us illustrate $\overset{e}{\mapsto}^*$ starting from $(\Pi_1, A_1, \emptyset)$:

$\overset{e}{\mapsto}$	Rule	$\mathcal{E}$
(e_{16})	$b \leftarrow \sim c$	$\mathcal{E}_1 = \{E_1 = [\varnothing, \{\sim c\}, \{\{\sim c\}\}], \widetilde{E}_1 = [\varnothing, \varnothing, \emptyset]\}$
(e_{17})	$b \leftarrow \sim c$	$\mathcal{E}_2 = \mathcal{E}_1 \cup \{E_2 = [b, \{\sim c\}, \{b\}], \widetilde{E}_2 = [\varnothing, \varnothing, \emptyset]\}$
(e_{18})	$b \leftarrow \sim c$	$\mathcal{E}_3 = \{E_1 = [b, \{\sim c\}, \{b, \{\sim c\}\}], \widetilde{E}_1, \widetilde{E}_2\}$
(e_{16})	$c \leftarrow \sim b$	$\mathcal{E}_4 = \mathcal{E}_3 \cup \{E_3 = [\varnothing, \{\sim b\}, \{\{\sim b\}\}], \widetilde{E}_3 = [\varnothing, \varnothing, \emptyset]\}$
(e_{20})		$\mathcal{E}_5 = \{\widetilde{E}_1 = [\varnothing, \{\sim b\}, \{\{\sim b\}\}], E_1, \widetilde{E}_2, \widetilde{E}_3\}$
(e_{17})	$c \leftarrow \sim b$	$\mathcal{E}_6 = \mathcal{E}_5 \cup \{E_4 = [c, \{\sim b\}, \{c\}], \widetilde{E}_4 = [\varnothing, \varnothing, \emptyset]\}$
(e_{18})	$c \leftarrow \sim b$	$\mathcal{E}_7 = \{\widetilde{E}_1 = [c, \{\sim b\}, \{c, \{\sim b\}\}], E_1 = [b, \{\sim c\}, \{b, \{\sim c\}\}], \widetilde{E}_2 = \widetilde{E}_3 = \widetilde{E}_4 = [\varnothing, \varnothing, \emptyset]\}$

We get two non-trivial, dual equivalence classes: E_1 and $\widetilde{E}_1$. Class E_1 is represented by b and supported by $\{\sim c\}$; $\widetilde{E}_1$ is represented by c and supported by $\{\sim b\}$. Observe that (e_{16}) and (e_{17}) introduce equivalence classes and their duals, while (e_{18}) and (e_{20}) merge different classes. (For simplicity, trivial dual classes are kept.)

The overall proceeding of $\overset{e}{\mapsto}^*$ is support-driven, that is, rules are only taken into account if their positive body atoms have been classified. Moreover, each (vital) class $[a, B, C]$ must be supported by some body $B \neq \varnothing$. To illustrate this, consider $\Pi_0 \cup \Pi_1^\star$, where

$$\Pi_1^\star = \{e \leftarrow \sim c;\ e \leftarrow f;\ f \leftarrow e;\ g \leftarrow e, \sim f;\ g \leftarrow h, \sim f;\ h \leftarrow f, g\}\ .$$

We get $(\Pi_0 \cup \Pi_1^\star, \emptyset, \emptyset) \overset{s}{\mapsto}^* (\Pi_1 \cup \Pi_1^\star, A_1, \emptyset)$ and continue by applying $\overset{e}{\mapsto}^*$ to $(\Pi_1 \cup \Pi_1^\star, A_1, \mathcal{E}_7)$:

$\overset{e}{\mapsto}$	Rule	$\mathcal{E}$
(e_{17})	$e \leftarrow \sim c$	$\mathcal{E}_1^\star = \mathcal{E}_7 \cup \{E_1^\star = [e, \{\sim c\}, \{e\}], \widetilde{E}_1^\star = [\varnothing, \varnothing, \emptyset]\}$
(e_{16})	$f \leftarrow e$	$\mathcal{E}_2^\star = \mathcal{E}_1^\star \cup \{E_2^\star = [\varnothing, \{e\}, \{\{e\}\}], \widetilde{E}_2^\star = [\varnothing, \varnothing, \emptyset]\}$
(e_{21})		$\mathcal{E}_3^\star = \mathcal{E}_7 \cup \{E_1^\star = [e, \{\sim c\}, \{e, \{e\}\}], \widetilde{E}_1^\star, \widetilde{E}_2^\star\}$
(e_{17})	$f \leftarrow e$	$\mathcal{E}_4^\star = \mathcal{E}_3^\star \cup \{E_3^\star = [f, \{e\}, \{f\}], \widetilde{E}_3^\star = [\varnothing, \varnothing, \emptyset]\}$
(e_{16})	$e \leftarrow f$	$\mathcal{E}_5^\star = \mathcal{E}_4^\star \cup \{E_4^\star = [\varnothing, \{f\}, \{\{f\}\}], \widetilde{E}_4^\star = [\varnothing, \varnothing, \emptyset]\}$
(e_{21})		$\mathcal{E}_6^\star = \mathcal{E}_3^\star \cup \{E_3^\star = [f, \{e\}, \{f, \{f\}\}], \widetilde{E}_3^\star, \widetilde{E}_4^\star\}$
(e_{19})	$f \leftarrow e$	$\mathcal{E}_7^\star = \mathcal{E}_7 \cup \{E_1^\star = [e, \{\sim c\}, \{e, \{e\}, f, \{f\}\}], \widetilde{E}_1^\star = \widetilde{E}_2^\star = \widetilde{E}_3^\star = \widetilde{E}_4^\star = [\varnothing, \varnothing, \emptyset]\}$
(e_{22})	~~$g \leftarrow e, \sim f$~~	$\mathcal{E}_7^\star$

We thus get $(\Pi_2^\star, A_1, \mathcal{E}_7^\star)$, where $\Pi_2^\star = \Pi_1 \cup (\Pi_1^\star \setminus \{g \leftarrow e, \sim f\})$. Set $\mathcal{E}_7^\star$ augments $\mathcal{E}_7$ with $E_1^\star$, revealing that e and f can be treated as equals. Note that the supporting body $\{\sim c\}$ does not belong to $C_{E_1^\star}$, given that bodies $\{e\}$ and $\{f\}$ in $C_{E_1^\star}$ are involved in loop $\{e, f\}$. Notably, the application of (e_{22}) to $g \leftarrow e, \sim f$ allows us to stop without classifying g and h, which are unfounded relative to $\Pi_2^\star$. However, by delaying the removal of $g \leftarrow e, \sim f$, an equivalence relation $\mathcal{E}_7^{\star\star}$ such that g and h belong to classes E satisfying $B_E = \varnothing$

[4] The superscript s indicates supporting bodies $B \neq \varnothing$.

(s_0) $(\Pi \cup \{r, r\}, A, \mathcal{E}) \overset{s}{\mapsto} (\Pi \cup \{r\}, A, \mathcal{E})$

(s_1) $(\Pi \cup \{a \leftarrow \ell, \ell, B\}, A, \mathcal{E}) \overset{s}{\mapsto} (\Pi \cup \{a \leftarrow \ell, B\}, A, \mathcal{E})$

(s_2) $(\Pi \cup \{a \leftarrow b, \sim b, B\}, A, \mathcal{E}) \overset{s}{\mapsto} (\Pi, A, \mathcal{E})$

(s_3) $(\Pi \cup \{a \leftarrow a, B\}, A, \mathcal{E}) \overset{s}{\mapsto} (\Pi, A, \mathcal{E})$

(s_4) $(\Pi \cup \{a \leftarrow\}, A, \mathcal{E}) \overset{s}{\mapsto} (\Pi, A \cup \{a\}, \mathcal{E})$

(s_5) $(\Pi, A, \mathcal{E}) \overset{s}{\mapsto} (\Pi, A \cup \{\overline{a}\}, \mathcal{E})$ $\quad (a \in \mathit{atom}(\Pi) \setminus (A \cup \overline{A}), \mathit{body}(a) = \emptyset)$

(s_6) $(\Pi \cup \{a \leftarrow \sim a, B\}, A, \mathcal{E}) \overset{s}{\mapsto} (\Pi, A \cup \{\overline{\{\sim a\} \cup B}\}, \mathcal{E})$

(s_7) $(\Pi \cup \{a \leftarrow B\}, A \cup \{a\}, \mathcal{E}) \overset{s}{\mapsto} (\Pi, A \cup \{a\}, \mathcal{E})$

(s_8) $(\Pi \cup \{a \leftarrow B\}, A \cup \{\overline{B}\}, \mathcal{E}) \overset{s}{\mapsto} (\Pi, A \cup \{\overline{B}\}, \mathcal{E})$

(s_9) $(\Pi \cup \{a \leftarrow \ell, B\}, A \cup \{\ell\}, \mathcal{E}) \overset{s}{\mapsto} (\Pi \cup \{a \leftarrow B\}, A \cup \{\ell\}, \mathcal{E})$

(s_{10}) $(\Pi \cup \{a \leftarrow \sim \ell, B\}, A \cup \{\ell\}, \mathcal{E}) \overset{s}{\mapsto} (\Pi, A \cup \{\ell\}, \mathcal{E})$

(s_{11}) $(\Pi, A \cup \{\overline{\{\ell, \ell\} \cup B}\}, \mathcal{E}) \overset{s}{\mapsto} (\Pi, A \cup \{\overline{\{\ell\} \cup B}\}, \mathcal{E})$

(s_{12}) $(\Pi, A \cup \{\overline{\{b, \sim b\} \cup B}\}, \mathcal{E}) \overset{s}{\mapsto} (\Pi, A, \mathcal{E})$

(s_{13}) $(\Pi, A \cup \{\ell, \overline{\{\ell\} \cup B}\}, \mathcal{E}) \overset{s}{\mapsto} (\Pi, A \cup \{\ell, \overline{B}\}, \mathcal{E})$

(s_{14}) $(\Pi, A \cup \{\overline{\ell}, \overline{\{\ell\} \cup B}\}, \mathcal{E}) \overset{s}{\mapsto} (\Pi, A \cup \{\overline{\ell}\}, \mathcal{E})$

(s_{15}) $(\Pi, A \cup \{\overline{B}\}, \mathcal{E}) \overset{s}{\mapsto} (\Pi, A \cup \{\overline{a}, \overline{B}\}, \mathcal{E})$ $\quad (a \in (B^+ \cup B^-) \setminus (\mathit{atom}(\Pi) \cup A \cup \overline{A}))$

(e_{16}) $(\Pi \cup \{a \leftarrow B\}, A, \mathcal{E}) \overset{e}{\mapsto} (\Pi \cup \{a \leftarrow B\}, A, \mathcal{E} \cup \{E = [\varnothing, B, \{B\}], \widetilde{E} = [\varnothing, \varnothing, \emptyset]\})$ $\quad (B^+ \cup \mathcal{E}^{\mathrm{s}}_B \subseteq \mathcal{E}^{\mathrm{s}}_C, B \notin \mathcal{E}_C)$

(e_{17}) $(\Pi \cup \{a \leftarrow B\}, A, \mathcal{E}) \overset{e}{\mapsto} (\Pi \cup \{a \leftarrow B\}, A, \mathcal{E} \cup \{E = [a, B, \{a\}], \widetilde{E} = [\varnothing, \varnothing, \emptyset]\})$ $\quad (B^+ \cup \mathcal{E}^{\mathrm{s}}_B \subseteq \mathcal{E}^{\mathrm{s}}_C, a \notin \mathcal{E}_C)$

(e_{18}) $(\Pi \cup \{a \leftarrow B\}, A, \mathcal{E} \cup \{E, [a, B, C]\}) \overset{e}{\mapsto} (\Pi \cup \{a \leftarrow B\}, A, \mathcal{E} \cup \{E = [a, B, C \cup C_E]\})$ $\quad (\mathit{body}(a) \subseteq C_E, C_E \cap \mathit{atom}(\Pi) = \emptyset)$

(e_{19}) $(\Pi \cup \{a \leftarrow B\}, A, \mathcal{E} \cup \{E, [a, B, C]\}) \overset{e}{\mapsto} (\Pi \cup \{a \leftarrow B\}, A, \mathcal{E} \cup \{E = [a_E, B_E, C_E \cup C]\})$ $\quad (\mathit{body}(a) \subseteq C_E, C_E \cap \mathit{atom}(\Pi) \neq \emptyset)$

(e_{20}) $(\Pi, A, \mathcal{E} \cup \{E, \widetilde{E}, [a, B, C]\}) \overset{e}{\mapsto} (\Pi, A, \mathcal{E} \cup \{E = [a, B, C \cup C_E], \widetilde{E}\})$ $\quad (B \in C, B^+ = \emptyset, B^- \subseteq C_{\widetilde{E}}, C_E \cap \mathit{atom}(\Pi) = \emptyset)$

(e_{21}) $(\Pi, A, \mathcal{E} \cup \{E, \widetilde{E}, [a, B, C]\}) \overset{e}{\mapsto} (\Pi, A, \mathcal{E} \cup \{E = [a_E, B_E, C_E \cup C], \widetilde{E}\})$ $\quad (B \in C, B^+ \subseteq C_E, B^- \subseteq C_{\widetilde{E}}, C_E \cap \mathit{atom}(\Pi) \neq \emptyset)$

(e_{22}) $(\Pi \cup \{a \leftarrow B\}, A, \mathcal{E} \cup \{E, \widetilde{E}\}) \overset{e}{\mapsto} (\Pi, A, \mathcal{E} \cup \{E, \widetilde{E}\})$ $\quad ((B^+ \cap C_E) \cup (B^- \cap C_{\widetilde{E}}) \neq \emptyset, (B^+ \cap C_{\widetilde{E}}) \cup (B^- \cap C_E) \neq \emptyset)$

(e_{23}) $(\Pi, A, \mathcal{E} \cup \{[a, B, C]\}) \overset{e}{\mapsto} (\Pi, A, \mathcal{E} \cup \{[a, \varnothing, C]\})$ $\quad (B \neq \varnothing, B \notin \mathit{body}(\Pi))$

(e_{24}) $(\Pi, A, \mathcal{E} \cup \{[a, B, C]\}) \overset{e}{\mapsto} (\Pi, A, \mathcal{E} \cup \{[a, \varnothing, C]\})$ $\quad (B \neq \varnothing, B^+ \not\subseteq \mathcal{E}^{\mathrm{s}}_C)$

(e_{25}) $(\Pi \cup \{a \leftarrow B\}, A, \mathcal{E} \cup \{[a, \varnothing, C]\}) \overset{e}{\mapsto} (\Pi \cup \{a \leftarrow B\}, A, \mathcal{E} \cup \{[a, B, C]\})$ $\quad (B^+ \cup \mathcal{E}^{\mathrm{s}}_B \subseteq \mathcal{E}^{\mathrm{s}}_C)$

(e_{26}) $(\Pi \cup \{a \leftarrow B\}, A, \mathcal{E} \cup \{[a', \varnothing, C']\}) \overset{e}{\mapsto} (\Pi \cup \{a \leftarrow B\}, A, \mathcal{E} \cup \{[a, B, C]\})$ $\quad (\{a, a'\} \subseteq C', a \neq a', B^+ \cup \mathcal{E}^{\mathrm{s}}_B \subseteq \mathcal{E}^{\mathrm{s}}_C, C = (\{a, B\} \cap C') \cup (C' \setminus (\mathit{atom}(\Pi) \cup \mathit{body}(\Pi))))$

(e_{27}) $(\Pi \cup \{a \leftarrow B\}, A, \mathcal{E} \cup \{[\varnothing, \varnothing, C]\}) \overset{e}{\mapsto} (\Pi \cup \{a \leftarrow B\}, A, \mathcal{E} \cup \{[\varnothing, B, C]\})$ $\quad (B \in C, B^+ \cup \mathcal{E}^{\mathrm{s}}_B \subseteq \mathcal{E}^{\mathrm{s}}_C)$

(a_{28}) $(\Pi \cup \{a \leftarrow B\}, A, \mathcal{E} \cup \{E, \widetilde{E}\}) \overset{a}{\mapsto} (\Pi, A, \mathcal{E} \cup \{E, \widetilde{E}\})$ $\quad (a \in C_E \setminus \{a_E\}, \{(a' \leftarrow B') \in \Pi \cup \{a \leftarrow B\} \mid a' \in C_E \setminus \{a_E\}, B'^+ = \emptyset, a' \in \bigcup_{r \in \Pi \cup \{a \leftarrow B\}} \mathit{body}(r)^+\} = \emptyset)$

(a_{29}) $(\Pi \cup \{a \leftarrow b, B\}, A, \mathcal{E} \cup \{E, \widetilde{E}\}) \overset{a}{\mapsto} (\Pi \cup \{a \leftarrow a_E, B\}, A, \mathcal{E} \cup \{E, \widetilde{E}\})$ $\quad (b \in C_E \setminus \{a_E\}, \{(a' \leftarrow B') \in \Pi \mid a' \in C_E \setminus \{a_E\}, B'^+ = \emptyset, a' \in \bigcup_{r \in \Pi \cup \{a \leftarrow b, B\}} \mathit{body}(r)^+\} = \emptyset)$

(a_{30}) $(\Pi \cup \{a \leftarrow b, B\}, A, \mathcal{E} \cup \{E, \widetilde{E}\}) \overset{a}{\mapsto} (\Pi \cup \{a \leftarrow \sim a_{\widetilde{E}}, B\}, A, \mathcal{E} \cup \{E, \widetilde{E}\})$ $\quad (b \in C_E \setminus \{a_E\}, (b \leftarrow B') \in \Pi, B'^+ = \emptyset)$

(a_{31}) $(\Pi \cup \{a \leftarrow \sim c, B\}, A, \mathcal{E} \cup \{E, \widetilde{E}\}) \overset{a}{\mapsto} (\Pi \cup \{a \leftarrow B\}, A, \mathcal{E} \cup \{E, \widetilde{E}\})$ $\quad (c \in C_E, B^+ \cap C_{\widetilde{E}} \neq \emptyset)$

(a_{32}) $(\Pi \cup \{a \leftarrow \sim c, B\}, A, \mathcal{E} \cup \{E, \widetilde{E}\}) \overset{a}{\mapsto} (\Pi \cup \{a \leftarrow \sim a_E, B\}, A, \mathcal{E} \cup \{E, \widetilde{E}\})$ $\quad (c \in C_E \setminus \{a_E\}, B^+ \cap C_{\widetilde{E}} = \emptyset)$

(a_{33}) $(\Pi, A \cup \{\overline{B}\}, \mathcal{E} \cup \{E, \widetilde{E}\}) \overset{a}{\mapsto} (\Pi, A, \mathcal{E} \cup \{E, \widetilde{E}\})$ $\quad ((B^+ \cap C_E) \cup (B^- \cap C_{\widetilde{E}}) \neq \emptyset, (B^+ \cap C_{\widetilde{E}}) \cup (B^- \cap C_E) \neq \emptyset)$

(a_{34}) $(\Pi, A \cup \{\overline{\{b\} \cup B}\}, \mathcal{E} \cup \{E\}) \overset{a}{\mapsto} (\Pi, A \cup \{\overline{\{a_E\} \cup B}\}, \mathcal{E} \cup \{E\})$ $\quad (b \in C_E \setminus \{a_E\})$

(a_{35}) $(\Pi, A \cup \{\overline{\{\sim c\} \cup B}\}, \mathcal{E} \cup \{E\}) \overset{a}{\mapsto} (\Pi, A \cup \{\overline{\{\sim a_E\} \cup B}\}, \mathcal{E} \cup \{E\})$ $\quad (c \in C_E \setminus \{a_E\})$

(u_{36}) $(\Pi, A, \mathcal{E}) \overset{u}{\mapsto} (\Pi, A \cup \{\overline{a}\}, \mathcal{E})$ $\quad (a \in \mathit{atom}(\Pi) \setminus (\mathcal{E}^{\mathrm{s}}_C \cup \overline{A}))$

Table 1. Transformation rules for preprocessing (where $\ell \in \mathcal{A} \cup \overline{\mathcal{A}}$, $\sim a = \overline{a}$, $\sim \overline{a} = a$, and $\overline{\overline{a}} = a$).

could have been obtained as well. The latter again signals that g and h are unfounded, as in the case that they remain unclassified.

The next results shed some light on the syntactic properties of the consecutive application of $\overset{s}{\mapsto}^*$ and $\overset{e}{\mapsto}^*$, abbreviated by $\overset{s}{\mapsto}^* \overset{e}{\mapsto}^*$.

Proposition 3 *Let* $(\Pi, \emptyset, \emptyset) \overset{s}{\mapsto}^* \overset{e}{\mapsto}^* (\Pi', A, \mathcal{E})$*, for logic program* Π *over alphabet* $\mathcal{A}$*. Then, we have:*

1. $\mathcal{E}^{\mathrm{s}}_B \subseteq \mathcal{E}^{\mathrm{s}}_C \subseteq \mathit{atom}(\Pi') \cup \mathit{body}(\Pi')$*;*
2. $\mathcal{E}_C \cap (A \cup \overline{A}) = \emptyset$*;*
3. $C_E \cap C_{E'} = \emptyset$*, for all* $E, E' \in \mathcal{E}$ *such that* $E \neq E'$*;*
4. $(a_E \leftarrow B_E) \in \Pi'$*, for all* $E \in \mathcal{E}$ *such that* $a_E \neq \varnothing, B_E \neq \varnothing$*;*
5. $\mathit{body}(r)^+ \not\subseteq \mathcal{E}^{\mathrm{s}}_C$*, for all* $r \in \Pi'$ *such that* $\mathit{head}(r) \notin \mathcal{E}^{\mathrm{s}}_C$*.*

We next show that our transformations preserve answer sets and that duality among equivalence classes carries forward to answer sets.

Proposition 4 *Let* $(\Pi, \emptyset, \emptyset) \overset{s}{\mapsto}^* \overset{e}{\mapsto}^* (\Pi', A, \mathcal{E})$*, for logic program* Π *over alphabet* $\mathcal{A}$*, and let* $X \in AS(\Pi)$*. Then, we have:*

1. $A \cap \mathcal{A} \subseteq X \subseteq (A \cap \mathcal{A}) \cup \mathcal{E}^{\mathrm{s}}_C$*;*
2. $C_E \cap \mathcal{A} \subseteq X$ *and* $C_{\widetilde{E}} \cap X = \emptyset$ *or* $C_{\widetilde{E}} \cap \mathcal{A} \subseteq X$ *and* $C_E \cap X = \emptyset$*, for all* $\{E, \widetilde{E}\} \subseteq \mathcal{E}$*.*

Equivalences and implicit or explicit unfoundedness of atoms (cf. $\mathcal{E}^{\star}_7$ and $\mathcal{E}^{\star\star}_7$ above) are exploited by the remaining transformations: (a_{28}) to (a_{35}) substitute equivalent atoms by the representative a_E (or $\sim a_{\widetilde{E}}$ via rule (a_{30})) for their class E, while (u_{36}) assigns false to unfounded atoms.

Although $\overset{a}{\mapsto}$ and $\overset{u}{\mapsto}$ leave program Π_1 unchanged, they allow for further reducing $\Pi^{\star}_2$ in view of the obtained equivalence classes. We obtain $(\Pi^{\star}_2, A_1, \mathcal{E}^{\star}_7) \overset{a}{\mapsto}^* (\Pi^{\star}_3, A_1, \mathcal{E}^{\star}_7) \overset{u}{\mapsto}^* (\Pi^{\star}_3, A_2, \mathcal{E}^{\star}_7)$, where

$$\Pi^{\star}_3 = \Pi_1 \cup \{e \leftarrow \sim c;\ e \leftarrow e;\ g \leftarrow h, \sim e;\ h \leftarrow e, g\}$$

and $A_2 = A_1 \cup \{\overline{g}, \overline{h}\} = \{a, \overline{d}, \overline{g}, \overline{h}\}$.

Using $\mathcal{E}[X] = \bigcup_{[a,B,C] \in \mathcal{E}, C \cap X \neq \emptyset} (C \cap \mathcal{A})$ for accumulating all atoms equivalent to members of X, we obtain the following result.

Proposition 5 *Let* $(\Pi, \emptyset, \emptyset) \overset{s}{\mapsto}^* \overset{e}{\mapsto}^* \overset{a}{\mapsto}^* \overset{u}{\mapsto}^* (\Pi', A, \mathcal{E})$*, for logic program* Π *over alphabet* $\mathcal{A}$*. Then, we have*

$$AS(\Pi) = \{X \cup \mathcal{E}[X] \cup (A \cap \mathcal{A}) \mid X \in AS(\Pi') \cap \mathcal{M}(BF(\overline{A} \setminus \mathcal{A}))\}\,.$$

Finally, we consider the saturated result of preprocessing, where $\Pi \overset{*}{\mapsto} (\Pi', A, \mathcal{E})$ stands for $(\Pi, \emptyset, \emptyset)$ $(\overset{s*}{\mapsto} \overset{e*}{\mapsto} \overset{a*}{\mapsto} \overset{u*}{\mapsto})^*$ $(\Pi', A, \mathcal{E})$. Let $\sigma = \{y_1/y_1', \ldots, y_n/y_n'\}$ denote a substitution, and let $Y\sigma$ be Y with every occurrence of y_i replaced by y_i' for $1 \leq i \leq n$. This allows us to formulate the following termination and confluence result.

Theorem 6 *Let Π be a logic program over $\mathcal{A}$. Then, we have:*

1. *Every derivation $\overset{*}{\mapsto}$ from Π terminates with some $(\Pi', A, \mathcal{E})$ such that no transformation rule in Table 1 is applicable to $(\Pi', A, \mathcal{E})$;*
2. *If $\Pi \overset{*}{\mapsto} (\Pi_1, A_1, \mathcal{E}_1)$ and $\Pi \overset{*}{\mapsto} (\Pi_2, A_2, \mathcal{E}_2)$, then $(A_1 \cap \mathcal{A}) \cup \mathcal{E}[A_1] = (A_2 \cap \mathcal{A}) \cup \mathcal{E}[A_2]$, $\Pi_1\sigma = \Pi_2$, and $(\overline{A}_1 \setminus \mathcal{A})\sigma = \overline{A}_2 \setminus \mathcal{A}$, where $\sigma = \{a/a_E \mid E \in \mathcal{E}_2, a \in C_E \cap \mathcal{A}\}$;*
3. *If $\Pi \overset{*}{\mapsto} (\Pi_1, A_1, \mathcal{E}_1)$, $\Pi \overset{*}{\mapsto} (\Pi_2, A_2, \mathcal{E}_2)$, and $\{E_1, \widetilde{E}_1\} \subseteq \mathcal{E}_1$ such that $B_{E_1} \neq \emptyset$, then $\{E_2, \widetilde{E}_2\} \subseteq \mathcal{E}_2$ such that $B_{E_2} \neq \emptyset$, $C_{E_1}\sigma = C_{E_2}\sigma$, and $C_{\widetilde{E}_1}\sigma = C_{\widetilde{E}_2}\sigma$, where $\sigma = \{a/a_E \mid E \in \mathcal{E}_2, a \in C_E \cap \mathcal{A}\}$.*

Reconsidering $\Pi_0 \cup \Pi_1^\star$, we get $(\Pi_0 \cup \Pi_1^\star) \overset{*}{\mapsto} (\Pi_1, A_2, \mathcal{E}^\star)$, where $\mathcal{E}^\star$ contains two vital classes, viz., $E^\star = [b, \{\sim c\}, \{b, \{\sim c\}, e, \{e\}, f, \{f\}\}]$ and $\widetilde{E}^\star = [c, \{\sim b\}, \{c, \{\sim b\}\}]$, while all other classes $E \in \mathcal{E}^\star$ are such that $B_E = \emptyset$. This outcome is independent from the order in which transformations are applied. Also note that all six rules of $\Pi_1^\star$ are removed by preprocessing, thus transforming non-tight program $\Pi_0 \cup \Pi_1^\star$ into tight program Π_1.

Notably, the result of our transformations goes beyond the *well-founded model* [21] of a logic program.

Proposition 7 *Let $\Pi \overset{*}{\mapsto} (\Pi', A, \mathcal{E})$, for logic program Π over $\mathcal{A}$, and let $I \subseteq \mathcal{A} \cup \overline{\mathcal{A}}$ be the well-founded model of Π. Then, we have $I \cap \mathcal{A} \subseteq (A \cap \mathcal{A}) \cup \mathcal{E}[A]$ and $I \cap \overline{\mathcal{A}} \subseteq (\mathcal{A} \setminus (A \cup \mathcal{E}[A \cup atom(\Pi')]))$.*

Similar to the known algorithms for computing a program's well-founded model, $\overset{*}{\mapsto}$ can be computed in quadratic time. In fact, if no program rule is removed (via rules other than (a_{28})) after the initial application of $\overset{s*}{\mapsto}$, a linear pass of $\overset{s*}{\mapsto} \overset{e*}{\mapsto} \overset{a*}{\mapsto} \overset{u*}{\mapsto}$ suffices to compute $\overset{*}{\mapsto}$, while iteration, viz., $(\overset{s*}{\mapsto} \overset{e*}{\mapsto} \overset{a*}{\mapsto} \overset{u*}{\mapsto})^*$, is needed otherwise.

We now take advantage of the result of our initial preprocessing phase, $(\Pi', A, \mathcal{E})$, for obtaining a compact completion formula. To this end, we use $\mathcal{E}$ to induce a *variable mapping* $\nu : atom(\Pi') \cup \{p_B \mid B \in body(\Pi')\} \to \mathcal{V} \cup \overline{\mathcal{V}}$, where $\mathcal{V}$ is an alphabet of variable names. For each $\{E, \widetilde{E}\} \subseteq \mathcal{E}$ such that $B_E \neq \emptyset$, we select a unique $v \in \mathcal{V}$ and map the elements of E and $\widetilde{E}$ as follows:

1. $\nu(y) = v$ iff $y \in (C_E \cap atom(\Pi')) \cup \{p_B \mid B \in C_E \cap body(\Pi')\}$;
2. $\nu(y) = \overline{v}$ iff $y \in (C_{\widetilde{E}} \cap atom(\Pi')) \cup \{p_B \mid B \in C_{\widetilde{E}} \cap body(\Pi')\}$.

Practically, ν amounts to an abstraction of the original program, as used for the internal representation within ASP solvers. We then use ν for inducing a substitution $\sigma_\nu = \{y/\nu(y) \mid y \in atom(\Pi') \cup \{p_B \mid B \in body(\Pi')\}\}$. For $(\Pi_1, A_2, \mathcal{E}^\star)$, we get mapping $\nu_1 = \{b \mapsto v;\ c \mapsto \overline{v};\ p_{\{\sim c\}} \mapsto v;\ p_{\{\sim b\}} \mapsto \overline{v}\}$, using only one variable v.

Having mapping ν induced by $(\Pi', A, \mathcal{E})$, we express the completion and loop formulas of Π' using the variables in $\mathcal{V}$:

$$VF_\nu(\Pi', A, \mathcal{E}) = \big(LF(\Pi') \cup BF(\overline{A} \setminus \mathcal{A}) \cup CF(\Pi', atom(\Pi') \cup (\mathcal{A} \setminus (A \cup \mathcal{E}[A \cup atom(\Pi')])))\big)\sigma_\nu\ .$$

Note that applying σ_ν leaves the introduction of body propositions (cf. (1)) implicit. In our example, we get

$$VF_{\nu_1}(\Pi_1, A_2, \mathcal{E}^\star) = CF(\Pi_1, \{b, c, d, g, h\})\sigma_{\nu_1} = \{v \leftrightarrow v;\ \overline{v} \leftrightarrow \overline{v};\ d \leftrightarrow \bot;\ g \leftrightarrow \bot;\ h \leftrightarrow \bot\}\ .$$

Note that $LF(\Pi_1)$ is empty (since Π_1 is tight), and so is $BF(\overline{A}_2 \setminus \mathcal{A})$. Clearly, $CF(\Pi_1, \{b, c, d, g, h\})\sigma_{\nu_1}$ possesses the models $\emptyset$ and $\{v\}$. Such models are linked to the atoms in an original program Π by appeal to $EF_\nu(\mathcal{E}) = \{a \leftrightarrow \nu(a_E) \mid E \in \mathcal{E}, B_E \neq \emptyset, a \in C_E \cap \mathcal{A}\}$; e.g., $EF_{\nu_1}(\mathcal{E}^\star) = \{b \leftrightarrow v;\ e \leftrightarrow v;\ f \leftrightarrow v;\ c \leftrightarrow \overline{v}\}$.

Formally, we have the following result.

Theorem 8 *Let $\Pi \overset{*}{\mapsto} (\Pi', A, \mathcal{E})$, for logic program Π over $\mathcal{A}$, and let ν be a variable mapping induced by $(\Pi', A, \mathcal{E})$. Then, we have*

$$AS(\Pi) = \mathcal{M}((A \cap \mathcal{A}) \cup \mathcal{E}[A] \cup VF_\nu(\Pi', A, \mathcal{E}) \cup EF_\nu(\mathcal{E}))|_{\mathcal{A}}\ .$$

For instance, for $(\Pi_1, A_2, \mathcal{E}^\star)$, ν_1, and $\mathcal{A} = \{a, \ldots, h\}$, we obtain $\mathcal{M}(\{a\} \cup \emptyset \cup VF_{\nu_1}(\Pi_1, A_2, \mathcal{E}^\star) \cup EF_{\nu_1}(\mathcal{E}^\star))|_{\mathcal{A}} = \{\{a, b, e, f\}, \{a, c\}\}$, which are the two answer sets of $\Pi_0 \cup \Pi_1^\star$. Finally, note that our implementation within *clasp* takes advantage of the preprocessing result only for the initial construction of a compact completion formula, while loop formulas are not computed a priori, but only if they are used for propagation or conflict analysis.

4 EXPERIMENTS

We conducted systematic experiments on the benchmark sets used in the categories *SCore* and *SLparse* of the ASP competition [15]. Our comparison considers the ASP solver *clasp* in four modes: (1) no elaborated preprocessing, only elementary simplifications as in (s_0) to (s_{15}); (2) external program reduction (as described in Section 3); (3) internal reduction, extending *SatELite*-like techniques [10];[5] and (4) both types of preprocessing. Table 2 summarizes results in seconds (t), indicating the number of timeouts via a superscript. Each line averages over n runs on $n/3$ instances, each shuffled three times. Furthermore, $|r|$, $|a|$, and $|b|$ give the average number of rules, atoms, and bodies, respectively, in the original programs of each class; $|v|$ and $|c|$ give the average number of variables and Boolean constraints in the internal representation. The number of variables $|v|$ is the same for variant (1) and (3) as well as for (2) and (4), respectively, and thus not duplicated in Table 2. At the bottom of Table 2, all individual runs are summed up, not taking averages. Full details are provided at [7].

In total, we see that variant (4) performs best, even though *SatELite*-like techniques are currently not applied to so-called extended rules (allowed within *SLparse* instances, shown in the second part of Table 2), while we have generalized external program reduction to work on such rules too. Furthermore, *SatELite*-like techniques work best on tight examples, being released from unfounded set checking. (Note that 2/3 of the benchmark classes are tight.) Unlike this, the approach in Section 3 is advantageous on non-tight programs due to its support-driven strategy. Another factor is the size of input programs. While our external technique $(\overset{s*}{\mapsto} \overset{e*}{\mapsto} \overset{a*}{\mapsto} \overset{u*}{\mapsto})$ is implemented in a linear fashion, *SatELite*-like techniques involve subsumption tests yielding a quadratic worst case behavior. Regarding the number of variables, one has to compare $|a|+|b|$ with $|v|$. In the worst case, both would be equal. However, we sometimes see significant reductions of more than one order of magnitude. Given that the elementary simplifications already cut down the number of variables, the speed-ups of version (2) over (1) are mainly due to the reduced completion formula (reflected by $|c|$). Also, the number $|c|$ of constraints is often much smaller than the original number $|r|$ of rules.

5 DISCUSSION

We provided the first ASP-specific approach to preprocessing logic programs, aiming at reducing an input program as well as the number of variables in its internal representation. The latter goal is also pursued by *smodels* [19], where choices rely on atoms occurring negatively in bodies, and by *cmodels* [8], where heuristics are used to

[5] Note that a straightforward application of *SatELite*-like techniques is insufficient since it interferes with unfounded set detection.

Problem				clasp (1)			clasp (2)			clasp (3)		clasp (4)	
Name (n)	$\|r\|$	$\|a\|$	$\|b\|$	$\|v\|$	$\|c\|$	t	$\|v\|$	$\|c\|$	t	$\|c\|$	t	$\|c\|$	t
15-Puzzle (30)	17203	5161	13029	3100	24348	0.3	2930	23942	0.3	13497	0.3	13296	0.3
BlockedN-Queens (42)	308796	5503	155646	53716	69281	^{18}285.8	50613	2988	^{16}254.5	2720	^{18}265.1	2720	^{18}265.7
EqTest (15)	6901	434	2996	1143	12338	16.0	999	11514	14.4	9866	16.4	9419	14.7
Factoring (15)	6974	4965	6782	3637	13407	5.6	2244	9524	3.9	3791	1.8	3765	1.9
HamiltonianPath (42)	4228	1533	2542	1358	5533	0.1	748	2987	0.1	2974	0.1	1277	0.1
RLP-150 (42)	728	151	715	288	3002	0.3	286	2992	0.3	2994	0.3	2986	0.3
RLP-200 (42)	1184	201	1165	455	4850	0.9	453	4838	0.9	4835	1.0	4826	0.9
RandomNonTight (42)	839	55	806	287	5286	32.3	283	5267	32.8	5286	31.3	5252	33.4
SchurNumbers (15)	12014	736	4391	1005	4862	2.3	829	3971	1.4	2451	2.6	1602	1.0
15-Puzzle (15)	38250	11385	37498	15694	116321	^{1}213.2	15298	115173	96.3	79624	104.1	79624	112.8
BlockedN-Queens (15)	5024	4699	2726	2472	331	17.1	894	331	9.1	331	9.5	331	13.5
BoundedSpanningTree (15)	206557	2359	203226	68524	201427	3.7	67796	198432	3.7	190486	16.5	190486	16.8
CarSequencing (15)	1582	2303	1263	1189	630	^{15}600.0	695	630	^{15}600.0	630	^{15}600.0	630	^{13}566.3
Factoring (12)	7685	5470	7472	4006	14803	8.6	2473	10525	4.1	4196	2.2	4170	2.1
HamiltonianCycle (15)	10502	7003	4955	3986	12236	0.3	1925	7916	0.2	4676	1.4	4641	1.3
HamiltonianPath (15)	4924	1623	2920	1514	6102	0.1	864	3387	0.1	3364	0.1	1560	0.1
Hashiwokakero (12)	738726	149926	717900	227596	2163406	^{3}125.2	217954	1912400	^{3}125.2	1915809	^{3}125.4	1912400	^{3}125.3
KnightsTour (15)	58062	10968	37996	14866	16518	0.5	11383	10559	0.5	5317	0.7	3402	0.7
RLP-150 (15)	735	151	721	290	3030	0.4	288	3019	0.3	3023	0.4	3014	0.3
RLP-200 (15)	793	199	781	326	3309	1.1	319	3269	1.0	3276	1.0	3244	1.0
RandomNonTight (15)	848	55	816	290	5380	9.0	287	5361	5.8	5380	9.0	5347	5.5
SchurNumbers (15)	85319	1713	43097	7570	11438	^{2}129.3	7307	11438	^{1}164.0	10705	^{1}129.0	10705	^{1}97.8
SearchTest-plain (15)	690808	4339	522045	34753	160494	^{3}122.9	31869	148922	^{2}114.1	114633	^{3}124.4	105102	^{1}81.5
SearchTest-verbose (15)	802803	4959	606804	40320	165791	12.3	36964	152633	13.8	97379	37.5	88708	34.9
SocialGolfer (15)	31506	11269	31108	12500	119754	^{3}120.6	11857	119754	^{3}121.3	108148	^{3}124.4	108148	^{3}124.2
SolitaireBackward (15)	20508	8381	9305	5473	39345	1.9	2545	18017	1.1	13980	1.7	11740	0.7
SolitaireBackward2 (15)	27435	4397	25517	8713	14323	^{4}260.4	8366	14323	^{6}312.8	10008	^{4}179.1	10009	^{3}177.7
SolitaireForward (15)	19606	8020	8858	5153	29835	^{3}120.3	3602	23819	^{3}120.3	18448	^{2}90.3	15253	^{3}120.2
Su-Doku (9)	1003593	17053	502502	173185	12772	7.1	165897	12772	7.9	12772	11.0	12772	11.3
TowersOfHanoi (15)	18340	7215	15028	7294	15903	24.1	5500	13527	24.4	8665	24.7	8664	16.0
TravelingSalesperson (15)	3825	3065	1588	1448	3588	0.4	583	2356	0.2	2356	0.3	2339	1.5
VerifyTest-variableSearchSpace (15)	12914	2296	9134	1061	4285	0.1	608	3088	0.1	1273	0.1	806	0.1
WeightBoundedDominatingSet (15)	3163	2879	798	1187	2048	^{6}245.9	264	910	^{4}165.1	453	^{3}128.2	453	^{2}105.4
WeightedLatinSquare (15)	997	770	446	405	222	0.0	146	222	0.0	222	0.0	222	0.0
WeightedSpanningTree (15)	112034	2185	108934	36998	81210	2.3	36294	78426	2.2	78052	4.5	78052	4.4
Total		time/timeouts			44116.9/58			40774.2/53		38641.0/52		37139.0/47	
		variables/constraints			10954406/46339719			10172081/39117132		-/35997972		-/35438242	

Table 2. Experiments with *clasp* (1.0.5) on a 2.2GHz PC under Linux; each run restricted to 600s time and 1GB RAM.

eliminate body variables. However, up to now *clasp* is the only ASP solver integrating advanced preprocessing techniques. Neither ASP-specific (external) nor *SatELite*-like (internal) preprocessing have yet been implemented elsewhere in the context of ASP. Our experiments show that investments in preprocessing are well spent. In fact, the best results are obtained when combining ASP-specific with *SatELite*-like preprocessing. Instead of integrating preprocessing into *clasp*, it could be performed by a dedicated front-end, beneficial also to other solvers. The development of such a tool is left as a future issue.

REFERENCES

[1] `http://assat.cs.ust.hk`.

[2] F. Bacchus, 'Enhancing Davis Putnam with extended binary clause reasoning', in *Proceedings AAAI'02*, pp. 613–619. AAAI Press, (2002).

[3] C. Baral, *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, (2003).

[4] S. Brass and J. Dix, 'Semantics of (disjunctive) logic programs based on partial evaluation', *Journal of Logic Programming*, **40**(1), 1–46, (1999).

[5] S. Brass, J. Dix, B. Freitag, and U. Zukowski, 'Transformation-based bottom-up computation of the well-founded model', *Theory and Practice of Logic Programming*, **1**(5), 497–538, (2001).

[6] K. Clark, 'Negation as failure', in *Logic and Data Bases*, eds., H. Gallaire and J. Minker, pp. 293–322. Plenum Press, (1978).

[7] `http://www.cs.uni-potsdam.de/clasp`.

[8] `http://www.cs.utexas.edu/users/tag/cmodels`.

[9] `http://www.dlvsystem.com`.

[10] N. Eén and A. Biere, 'Effective preprocessing in SAT through variable and clause elimination', in *Proceedings SAT'05*, eds., F. Bacchus and T. Walsh, pp. 61–75. Springer, (2005).

[11] T. Eiter, M. Fink, H. Tompits, and S. Woltran, 'Simplifying logic programs under uniform and strong equivalence', in *Proceedings LPNMR'04*, eds., V. Lifschitz and I. Niemelä, pp. 87–99. Springer, (2004).

[12] F. Fages, 'Consistency of Clark's completion and the existence of stable models', *J. of Methods of Logic in Computer Science*, **1**, 51–60, (1994).

[13] M. Fitting, 'Tableaux for logic programming', *Journal of Automated Reasoning*, **13**(2), 175–188, (1994).

[14] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub, 'Conflict-driven answer set solving', in *Proceedings IJCAI'07*, ed., M. Veloso, pp. 386–392. AAAI Press/MIT Press, (2007).

[15] M. Gebser, L. Liu, G. Namasivayam, A. Neumann, T. Schaub, and M. Truszczyński, 'The first answer set programming system competition', in *Proceedings LPNMR'07*, eds., C. Baral, G. Brewka, and J. Schlipf, pp. 3–17. Springer, (2007).

[16] M. Gebser and T. Schaub, 'Tableau calculi for answer set programming', in *Proceedings ICLP'06*, eds., S. Etalle and M. Truszczyński, pp. 11–25. Springer, (2006).

[17] C. Gomes, H. Kautz, A. Sabharwal, and B. Selman, 'Satisfiability solvers', in *Handbook of Knowledge Representation*, eds., V. Lifschitz, F. van Hermelen, and B. Porter. Elsevier, (2008).

[18] F. Lin and Y. Zhao, 'ASSAT: computing answer sets of a logic program by SAT solvers', *Artificial Intelligence*, **157**(1-2), 115–137, (2004).

[19] `http://www.tcs.hut.fi/Software/smodels`.

[20] S. Subbarayan and D. Pradhan, 'NiVER: Non increasing variable elimination resolution for preprocessing SAT instances', in *Proceedings SAT'04*, eds., H. Hoos and D. Mitchell, pp. 276–291. Springer, (2005).

[21] A. Van Gelder, K. Ross, and J. Schlipf, 'The well-founded semantics for general logic programs', *Journal of the ACM*, **38**(3), 620–650, (1991).