Characterizing ASP Inferences by Unit Propagation

Martin Gebser and Torsten Schaub*

Institut für Informatik, Universität Potsdam, August-Bebel-Str. 89, D-14482 Potsdam
{gebser,torsten}@cs.uni-potsdam.de

Abstract. Computational approaches to Satisfiability Checking (SAT) and Answer Set Programming (ASP) have many aspects in common. In fact, the basic algorithms of ASP solvers are very similar to the Davis-Logemann-Loveland procedure (DLL) for SAT. The major difference lies in the inference rules, which are more complex in ASP. In this paper, we provide a generic framework, based on concepts from Constraint Processing (CSP), which allows us to view ASP inferences as forms of unit propagation. We develop declarative characterizations of ASP solvers *nomore++* and *smodels* in terms of constraints. By putting ASP solving into a common context with SAT and CSP, we shed new light on ASP solving techniques and their relationships to neighboring fields.

1 Introduction

Computational mechanisms of SAT and ASP solvers are closely related. This can, for instance, be seen on the success of SAT-based ASP solvers *assat* [1] and *cmodels* [2]. Clearly, SAT-based ASP solvers benefit from the availability of highly efficient SAT solvers, which they use as search engines. However, advanced techniques, such as clause learning and restarts [3], that are now standard in state-of-the-art SAT solvers are not yet exploited by genuine ASP solvers. The only exception to this, the *smodels*$_{cc}$ system [4], relies on a solitary and rather involved theoretical fundament.

We believe that the putative gap between SAT and ASP solvers originates from lacking declarativeness in the specification of ASP algorithms. In this paper, we address this deficiency and specify the constraints underlying ASP solvers *nomore++* [5] and *smodels* [6].[1] Our aims are twofold: First, we want to enhance the understanding of existing algorithms, second, we want to reveal their shortcomings and indicate subjects for possible improvements. Deeper insight into the nature of ASP solving, hopefully, is fruitful for the design of new algorithms and technology transfer from SAT or, more generally, CSP [8] to ASP.

Our goal is to compare computational approaches to SAT and ASP in terms of their inferences rather than as regards logical equivalence in terms of models. In particular, we are interested in the computational strength of different approaches, despite their logical similarity or even equivalence. To this end, we appeal to the concept of *nogoods*,

* Affiliated with the School of Computing Science at Simon Fraser University, Canada, and the Institute for Integrated and Intelligent Systems at Griffith University, Australia.

[1] Our choice is motivated by the fact that both systems primarily address normal logic programs. In contrast, *dlv* [7] is devised for dealing with disjunctive logic programs.

known from CSP, for capturing inferences in SAT as well as in different approaches to ASP solving. While nogoods are explicitly used as clauses in SAT solvers, they are only implicit in ASP solving. The usage of nogoods provides us with a uniform characterization of inferences; it allows us to show that SAT as well as ASP inferences constitute two different instances of the same CSP framework. Moreover, they make explicit which constraints underlie propagation in (different) ASP solvers.

2 Background

We deal with propositional expressions over an *alphabet* $\mathcal{A}$ of *atoms*, denoted by lowercase letters. We represent *interpretations* by subsets of $\mathcal{A}$, consisting of all atoms being true in the respective interpretation.

In the context of SAT, we are interested in finite *sets of clauses*, Γ, representing formulas in *conjunctive normal form* (CNF). Each clause $\gamma \in \Gamma$ is a set $\{p_1, \ldots, p_m, \neg p_{m+1}, \ldots, \neg p_n\}$ containing *propositional literals*, that is, atoms p_i or their negation $\neg p_i$ for $1 \leq i \leq n$. The set of atoms occurring in a clause set Γ is denoted by $atom(\Gamma)$.

Given a clause set Γ, the primary question is then whether it is *satisfiable*, that is, whether it has a model. As an example, consider $\Gamma_1 = \{\{x, \neg y\}, \{\neg x, z\}, \{y, z\}\}$. The sets $\{z\}$, $\{x, z\}$, and $\{x, y, z\}$ of true atoms are models of Γ_1; hence, Γ_1 is satisfiable. By adding clause $\{\neg z\}$ to Γ_1, we obtain an unsatisfiable set of clauses.

In ASP, we deal with *sets of rules*, Π, representing logic programs. Unlike clauses, rules, like π, can be regarded as ordered pairs [9] of the form

$$p_0 \leftarrow p_1, \ldots, p_m, not\ p_{m+1}, \ldots, not\ p_n\ , \tag{1}$$

where $head(\pi) = p_0$ is referred to as the *head* of π and the set $body(\pi) = \{p_1, \ldots, p_m, not\ p_{m+1}, \ldots, not\ p_n\}$ as the *body* of π. While $head(\pi)$ is simply an atom, $body(\pi)$ comprises atoms p_i or their default negation $not\ p_i$; simply referred to as *body literals*. Given a set X of body literals, let $X^+ = \{p \in \mathcal{A} \mid p \in X\}$ and $X^- = \{p \in \mathcal{A} \mid not\ p \in X\}$. For $body(\pi)$, we then get $body(\pi)^+ = \{p_1, \ldots, p_m\}$ and $body(\pi)^- = \{p_{m+1}, \ldots, p_n\}$. The set of atoms occurring in a set of rules Π is $atom(\Pi)$, the set of bodies is $body(\Pi) = \{body(\pi) \mid \pi \in \Pi\}$. For regrouping rule bodies sharing the same head p, define $body(p) = \{body(\pi) \mid \pi \in \Pi, head(\pi) = p\}$.

In ASP, we are interested in the *stable models* of a rule set Π. For coherence, we give a definition in terms of clauses. For π as in (1), define the associated clause as

$$\gamma(\pi) = \{p_0, \neg p_1, \ldots, \neg p_m, p_{m+1}, \ldots, p_n\}$$

and $\Gamma(\Pi) = \{\gamma(\pi) \mid \pi \in \Pi\}$. The term *stability* means that the model is stable under rule application. To this end, a rule set Π is reduced wrt an interpretation X:[2]

$$\Gamma(\Pi)^X = \{\gamma(\pi) \mid \pi \in \Pi, body(\pi)^+ \subseteq X, body(\pi)^- \cap X = \emptyset\}$$

Then, an interpretation X is a *stable model* of Π, if it is a $\subseteq$-minimal model of $\Gamma(\Pi)^X$. We say that X is a model of Π, if X is a model of $\Gamma(\Pi)$. In this way, every stable model is a also model of Π. Stable models are also called *answer sets*.

[2] This characterization traces back to [10]; meanwhile it has been adapted to ASP in [11, 12].

3 The Davis-Logemann-Loveland Procedure

Most complete state-of-the-art SAT solvers are (significantly improved) variants of the *Davis-Logemann-Loveland* procedure (DLL) [13]. The solving strategy of DLL is not restricted to SAT, but can be applied to other kinds of constraint satisfaction problems as well. In fact, even ASP solvers like *nomore++* and *smodels* follow the outline of DLL, though they rely on different propagation schemes. To compare the problem settings in SAT and ASP along with their solving algorithms, we first describe unit propagation and DLL via generic CSP concepts. Later on, particular settings found in DLL for SAT, *nomore++*, and *smodels* are mapped to these generic concepts. Hence, all problem solving approaches considered in this paper are specific DLL instances.

3.1 Unit Propagation

Abstracting from problem representations, we apply generic CSP concepts, specialized to variables over binary domains. In fact, a set Γ of clauses is a CSP: Its variables are the elements of $atom(\Gamma)$, the domain of every variable is $\{true, false\}$, and each clause $\gamma = \{p_1, \ldots, p_m, \neg p_{m+1}, \ldots, \neg p_n\}$ in Γ is a constraint, prohibiting that $p_1, \ldots, p_m$ are all assigned $false$ and $p_{m+1}, \ldots, p_n$ all $true$. While the embedding of clauses in CSP is straightforward, it is more complex for logic programs. Still, such an embedding is possible, and we will later on explain inferences in ASP solvers by their underlying constraints and unit propagation on them.

We first introduce assignments over Boolean variables. An *assignment* A over a *domain*, $dom(A)$, is a sequence $(\sigma_1, \ldots, \sigma_n)$ of *signed literals*[3] σ_i $(1 \leq i \leq n)$ of form $\mathbf{T}p$ or $\mathbf{F}p$ for some $p \in dom(A)$; $\mathbf{T}p$ expresses that p is true and $\mathbf{F}p$ that it is false. We say that literals $\mathbf{T}p$ and $\mathbf{F}p$ are *complementary*, and denote the complement of a literal σ by $\overline{\sigma}$, that is, $\overline{\mathbf{T}p} = \mathbf{F}p$ and $\overline{\mathbf{F}p} = \mathbf{T}p$. Sometimes we abuse notation and write $\mathbf{T}p \in A$ or $\mathbf{F}p \in A$, although A is a sequence and not a set. Given this notation, we access true and false propositions in A via $A^{\mathbf{T}} = \{p \in dom(A) \mid \mathbf{T}p \in A\}$ and $A^{\mathbf{F}} = \{p \in dom(A) \mid \mathbf{F}p \in A\}$. Moreover, we write $A \subseteq B$ to express that any literal contained in A is also in B, where A and B are either assignments or sets of literals. An assignment A is *total*, if $A^{\mathbf{T}} \cup A^{\mathbf{F}} = dom(A)$, and *conflicting*, if it contains a pair of complementary literals, that is, $A^{\mathbf{T}} \cap A^{\mathbf{F}} \neq \emptyset$. We let $A \circ B$ denote the sequence obtained by concatenating assignments A and B. We denote the empty assignment, not containing any literals, by $\emptyset$. Finally, we drop parentheses and write simply σ for a singleton assignment (σ).

For a canonical representation of constraints, we use the generic concept of a *nogood*, known from CSP. In our setting, it is sufficient to consider nogoods over literals: We define a *nogood* as a set of signed literals $\{\sigma_1, \ldots, \sigma_n\}$. Given a set of nogoods Δ, a *solution* A is a non-conflicting total assignment such that $\delta \not\subseteq A$ for all nogoods $\delta \in \Delta$. For instance, given clause $\{x, \neg y\}$, the set $\{\mathbf{F}x, \mathbf{T}y\}$ is a nogood: Its literals violate the clause and cannot be jointly contained in a solution. For a nogood δ, a literal $\sigma \in \delta$, and an assignment A, we say that $\overline{\sigma}$ is *unit-resulting* for δ wrt A, if

1. $\delta \setminus \{\sigma\} \subseteq A$ and

[3] We omit the attribute *signed* whenever clear from the context.

```
1 function DLL(Δ, A)
2     B ← UP*(Δ, A)
3     if B^T ∩ B^F ≠ ∅ then return false
4     if B^T ∪ B^F = dom(B) then return true
5     σ ← Select(Δ, B)
6     return DLL(Δ, B ∘ σ) ∨ DLL(Δ, B ∘ σ̄)
```

Fig. 1. Generic DLL algorithm.

2. $\overline{\sigma} \notin A$.

By the first condition, all literals of δ other than σ are contained in A, hence, only $\overline{\sigma}$ can be used to avert δ. The second condition ensures that unit-resulting literals are not already contained in A. For illustration, consider the nogood $\{\mathbf{F}x, \mathbf{T}y\}$. Literal $\mathbf{F}y$ is unit-resulting wrt assignment $\mathbf{F}x$, but not wrt $(\mathbf{F}x, \mathbf{F}y)$. Wrt assignment $(\mathbf{F}x, \mathbf{T}y)$, both literals $\mathbf{T}x$ and $\mathbf{F}y$ are unit-resulting. For $0 \leq i < n$, a literal σ_{i+1} is *implied* by a set of nogoods Δ in an assignment $(\sigma_1, \ldots, \sigma_i, \sigma_{i+1}, \ldots, \sigma_n)$, if σ_{i+1} is unit-resulting for some $\delta \in \Delta$ wrt $(\sigma_1, \ldots, \sigma_i)$.

We now come to *unit propagation*. For a set Δ of nogoods and an assignment A, we define:

$$UP(\Delta, A) = \begin{cases} A \circ \sigma & \text{if there is a } \delta \in \Delta \text{ such that } \sigma \text{ is unit-resulting for } \delta \text{ wrt } A \\ A & \text{otherwise} \end{cases}$$

Note that σ is not necessarily unique, as there might be several nogoods with distinct unit-resulting literals in Δ. However, the order in which literals are assigned is immaterial when we consider fixpoints: By $UP^*(\Delta, A)$, we denote an assignment C such that $C = A \circ B$, every literal in B is implied by Δ in C, and $UP(\Delta, C) = C$. For all possible orders of assigning literals, $UP^*(\Delta, A)$ is conflicting or contains the same literals. For instance, consider $\Delta = \{\{\mathbf{T}x\}, \{\mathbf{F}x, \mathbf{T}y\}, \{\mathbf{F}x, \mathbf{F}z\}\}$. We obtain either $UP^*(\Delta, \emptyset) = (\mathbf{F}x, \mathbf{F}y, \mathbf{T}z)$ or $UP^*(\Delta, \emptyset) = (\mathbf{F}x, \mathbf{T}z, \mathbf{F}y)$. When we add $\{\mathbf{F}y, \mathbf{T}z\}$ to Δ, then $UP^*(\Delta, \emptyset)$ is conflicting.

3.2 DLL Procedure

With unit propagation as defined above, we can describe DLL in a general way, not restricted to SAT. To this end, we use nogoods instead of directly working on clauses. (However, clauses can be viewed as syntactic representations of nogoods.) Figure 1 shows a generic DLL algorithm for a set Δ of nogoods and an assignment A.

Applying generic DLL to SAT is straightforward. For a clause $\gamma = \{p_1, \ldots, p_m, \neg p_{m+1}, \ldots, \neg p_n\}$, we let $\delta(\gamma) = \{\mathbf{F}p_1, \ldots, \mathbf{F}p_m, \mathbf{T}p_{m+1}, \ldots, \mathbf{T}p_n\}$ be the corresponding nogood. Given a set Γ of clauses, we let $\Delta(\Gamma) = \{\delta(\gamma) \mid \gamma \in \Gamma\}$ be the corresponding set of nogoods, and let the domain of assignments be $atom(\Gamma)$. Then, Γ is satisfiable iff $DLL(\Delta(\Gamma), \emptyset)$ returns $true$. Apart from the constraint-based representation, the generic DLL algorithm in Figure 1 is in fact the same as the original DLL procedure for SAT [13].[4]

[4] Note that we omit the *pure literal* rule. Anyway, most SAT solvers do not apply it [3].

4 A Constraint Counterpart of *nomore++*

The *nomore++* approach [5] to ASP solving works on logic programs and assignments to programs' atoms and bodies. For capturing this, we need some additional definitions. In *nomore++*, the domain of an assignment A is for a program Π given by $dom(A) = atom(\Pi) \cup body(\Pi)$. When looking at corresponding interpretations or clause sets, we uniquely associate a body β with a new propositional atom p_β that does not occur in the underlying program, that is, $p_\beta \notin atom(\Pi)$. We identify β with p_β when comparing an assignment of *nomore++* with an assignment to propositional atoms of a clause set.

In what follows, we identify nogoods underlying inferences in *nomore++*. This allows us to view *nomore++*'s inferences as unit propagation, according to Section 3.1. The nogoods also yield a set of clauses that can be used for investigating the logical contents of the underlying inferences. To extract these clauses from nogoods, we define the following conversion: For a nogood $\delta = \{\mathbf{T}p_1, \ldots, \mathbf{T}p_m, \mathbf{F}p_{m+1}, \ldots, \mathbf{F}p_n\}$, define the corresponding clause as $\gamma(\delta) = \{\neg q_1, \ldots, \neg q_m, q_{m+1}, \ldots, q_n\}$ where $q_i = p_i$ if $p_i \in atom(\Pi)$ and $q_i = p_\beta$ if $p_i = \beta \in body(\Pi)$ for $1 \leq i \leq n$. For a set of nogoods Δ, let $\Gamma(\Delta) = \{\gamma(\delta) \mid \delta \in \Delta\}$.

In contrast to using a single inference rule, like unit propagation, ASP solvers apply several ones, amounting to different semantic aspects of logic programs. For relating inferences in *nomore++* to unit propagation, we need to explicate the constraints underlying the different inference rules. We first concentrate on tight programs [14] and then extend the constraint characterization of *nomore++* to the non-tight case.

4.1 Tight Programs

The particular property of tight programs is that the *supported models* of a program's completion [15] coincide with the program's answer sets. The *completion* of a program is a propositional formula, asserting that an atom is true iff the atom has a true body. Tight programs can easily be reduced to SAT.

In [5], operators $\mathcal{P}_\Pi$ and $\mathcal{B}_\Pi$ are used for describing forward and backward propagation in *nomore++*. We consider here the combined effect of both operators. It turns out that one fraction of the inference rules applied by $\mathcal{P}_\Pi$ and $\mathcal{B}_\Pi$ allows for computing the models of a program, while another one enforces support for true program parts. Together, the inference rules allow for computing the supported models of a program.

Semantically, the body of a rule is a conjunction. Hence, it must be true if all its literals are true. Conversely, some of its literals must be false if the body is false. For a body $\beta = \{p_1, \ldots, p_m, not\ p_{m+1}, \ldots, not\ p_n\}$, the following nogood captures this:

$$\delta(\beta) = \{\mathbf{F}\beta, \mathbf{T}p_1, \ldots, \mathbf{T}p_m, \mathbf{F}p_{m+1}, \ldots, \mathbf{F}p_n\}$$

As an example, consider body $\{x, not\ y\}$: We obtain $\delta(\{x, not\ y\}) = \{\mathbf{F}\{x, not\ y\}, \mathbf{T}x, \mathbf{F}y\}$ and $\gamma(\delta(\{x, not\ y\})) = \{p_{\{x,not\ y\}}, \neg x, y\}$. Intuitively, $\delta(\beta)$ gives a constraint enforcing the truth of body β or the falsity of a contained literal.

Additionally, a body must be false if one of its literals is false. And conversely, all contained literals must be true if the body is true. For a body $\beta = \{p_1, \ldots, p_m, not\ p_{m+1}, \ldots, not\ p_n\}$, this is reflected by the following set of nogoods:

$$\Delta(\beta) = \{\, \{\mathbf{T}\beta, \mathbf{F}p_1\}, \ldots, \{\mathbf{T}\beta, \mathbf{F}p_m\}, \{\mathbf{T}\beta, \mathbf{T}p_{m+1}\}, \ldots, \{\mathbf{T}\beta, \mathbf{T}p_n\} \,\}$$

Taking again body $\{x, not\ y\}$, we obtain $\Delta(\{x, not\ y\}) = \{\{\mathbf{T}\{x, not\ y\}, \mathbf{F}x\}, \{\mathbf{T}\{x, not\ y\}, \mathbf{T}y\}\}$ and $\Gamma(\Delta(\{x, not\ y\})) = \{\{\neg p_{\{x,not\ y\}}, x\}, \{\neg p_{\{x,not\ y\}}, \neg y\}\}$.

For the bodies of a logic program Π, we obtain the following correspondence.

Proposition 1. *Let Π be a logic program.*

The set of clauses

$$\{\, \gamma(\delta(\beta)) \mid \beta \in \mathit{body}(\Pi) \,\} \cup \{\, \gamma \in \Gamma(\Delta(\beta)) \mid \beta \in \mathit{body}(\Pi) \,\}$$

and the propositional theory

$$\{\, p_\beta \equiv p_1 \wedge \cdots \wedge p_m \wedge \neg p_{m+1} \wedge \cdots \wedge \neg p_n \mid \beta \in \mathit{body}(\Pi), \beta = \{p_1, \ldots, p_m, not\ p_{m+1}, \ldots, not\ p_n\} \,\} \tag{2}$$

are logically equivalent.

This result captures the intuition that a body should be equivalent to the conjunction of its literals. However, the respective clauses are not only obtained when one does a propositional logic translation by hand. Below we show that one automatically ends up with these clauses when describing *nomore++*'s inferences by unit propagation.

We now come to inferences primarily aiming at atoms. An atom p must be true if some body in $\mathit{body}(p)$ is true. Conversely, all elements of $\mathit{body}(p)$ must be false if p is false. For $\mathit{body}(p) = \{\beta_1, \ldots, \beta_k\}$, we obtain the following set of nogoods:

$$\Delta(p) = \{\, \{\mathbf{F}p, \mathbf{T}\beta_1\}, \ldots, \{\mathbf{F}p, \mathbf{T}\beta_k\} \,\}$$

As an example, consider an atom x with $\mathit{body}(x) = \{\{y\}, \{not\ z\}\}$: We get $\Delta(x) = \{\{\mathbf{F}x, \mathbf{T}\{y\}\}, \{\mathbf{F}x, \mathbf{T}\{not\ z\}\}\}$ and $\Gamma(\Delta(x)) = \{\{x, \neg p_{\{y\}}\}, \{x, \neg p_{\{not\ z\}}\}\}$.

Finally, an atom p must be false if all elements of $\mathit{body}(p)$ are false. And conversely, some body in $\mathit{body}(p)$ must be true if p is true. For $\mathit{body}(p) = \{\beta_1, \ldots, \beta_k\}$, this is reflected by the following nogood:

$$\delta(p) = \{\mathbf{T}p, \mathbf{F}\beta_1, \ldots, \mathbf{F}\beta_k\}$$

For $\mathit{body}(x) = \{\{y\}, \{not\ z\}\}$, we obtain $\delta(x) = \{\mathbf{T}x, \mathbf{F}\{y\}, \mathbf{F}\{not\ z\}\}$ and $\gamma(\delta(x)) = \{\neg x, p_{\{y\}}, p_{\{not\ z\}}\}$.

Dually to Proposition 1, we get the following correspondence.

Proposition 2. *Let Π be a logic program.*

The set of clauses

$$\{\, \gamma \in \Gamma(\Delta(p)) \mid p \in \mathit{atom}(\Pi) \,\} \cup \{\, \gamma(\delta(p)) \mid p \in \mathit{atom}(\Pi) \,\}$$

and the propositional theory

$$\{\, p \equiv p_{\beta_1} \vee \cdots \vee p_{\beta_k} \mid p \in \mathit{atom}(\Pi), \mathit{body}(p) = \{\beta_1, \ldots, \beta_k\} \,\} \tag{3}$$

are logically equivalent.

Before we proceed with discussing further properties of the clauses associated with atoms and bodies, let us demonstrate that unit propagation on the respective nogoods truly corresponds to *nomore++*'s propagation operators $\mathcal{P}_\Pi$ and $\mathcal{B}_\Pi$. Denoting the fixpoint obtained by propagating an assignment A via $\mathcal{P}_\Pi$ and $\mathcal{B}_\Pi$ by $(\mathcal{P}_\Pi\mathcal{B}_\Pi)^*(A)$, we obtain the following one-to-one correspondence.

Theorem 1. *Let Π be a logic program and A be an assignment over $atom(\Pi) \cup body(\Pi)$. Let*

$$\begin{aligned}\Delta_\Pi = &\{\, \delta(\beta) \mid \beta \in body(\Pi) \,\} \cup \{\, \delta \in \Delta(\beta) \mid \beta \in body(\Pi) \,\} \cup \\ &\{\, \delta(p) \mid p \in atom(\Pi) \,\} \cup \{\, \delta \in \Delta(p) \mid p \in atom(\Pi) \,\}\,.\end{aligned}$$

For $B = UP^*(\Delta_\Pi, A)$ and $C = (\mathcal{P}_\Pi\mathcal{B}_\Pi)^*(A)$, we have the following:

1. *B is conflicting iff C is conflicting.*
2. *If neither B nor C is conflicting, then $(B^{\mathbf{T}}, B^{\mathbf{F}}) = (C^{\mathbf{T}}, C^{\mathbf{F}})$.*

The above result tells us that the same nogoods are obtained either by analyzing the behavior of $\mathcal{P}_\Pi$ and $\mathcal{B}_\Pi$ or by converting a program's completion to CNF, when abbreviating bodies by new symbols (cf. Proposition 1 and 2). The latter is in fact done by SAT-based ASP solvers, e.g., in *cmodels* [16], in order to keep the size of the clause set polynomial. It is thus justified to say that, for tight programs, propagation within SAT-based solvers and *nomore++* averts (almost) the same nogoods, either expressed by clauses or through connections between atoms and bodies in a program.

Next we consider the semantics of the clauses corresponding to the identified nogoods. Models and supported models of a program can be characterized as follows.

Proposition 3. *Let Π be a logic program and*

$$\Gamma_\Pi = \{\, \gamma(\delta(\beta)) \mid \beta \in body(\Pi) \,\} \cup \{\, \gamma \in \Gamma(\Delta(p)) \mid p \in atom(\Pi) \,\}\,. \tag{4}$$

Then, $X \subseteq atom(\Pi)$ is a model of Π iff there is a model Y of Γ_Π such that $X = Y \cap atom(\Pi)$.

Note that, for a body $\beta = \{p_1, \ldots, p_m, not\ p_{m+1}, \ldots, not\ p_n\}$, clause $\gamma(\delta(\beta))$ is equivalent to $p_1 \wedge \cdots \wedge p_m \wedge \neg p_{m+1} \wedge \cdots \wedge \neg p_n \rightarrow p_\beta$. Similarly, for an atom p with $body(p) = \{\beta_1, \ldots, \beta_k\}$, the clauses in $\Gamma(\Delta(p))$ can be rewritten as $p_{\beta_1} \rightarrow p$, $\ldots$, $p_{\beta_k} \rightarrow p$. So the clauses in (4) enforce truth of bodies and atoms, while support for true bodies and atoms is not required. In order to also guarantee support, we can use the clauses obtained from nogoods in $\Delta(\beta)$ and $\delta(p)$.

Proposition 4. *Let Π be a logic program and*

$$\begin{aligned}\Gamma_\Pi = &\{\, \gamma(\delta(\beta)) \mid \beta \in body(\Pi) \,\} \cup \{\, \gamma \in \Gamma(\Delta(\beta)) \mid \beta \in body(\Pi) \,\} \cup \\ &\{\, \gamma(\delta(p)) \mid p \in atom(\Pi) \,\} \cup \{\, \gamma \in \Gamma(\Delta(p)) \mid p \in atom(\Pi) \,\}\,.\end{aligned}$$

Then, $X \subseteq atom(\Pi)$ is a supported model of Π iff there is a (unique) model Y of Γ_Π such that $Y \subseteq atom(\Gamma_\Pi)$ and $X = Y \cap atom(\Pi)$.

Given that for tight programs supported models and answer sets coincide [17], the following is an immediate consequence of Proposition 4.

Theorem 2. *Let Π be a tight logic program and Δ_Π as in Theorem 1.*

Then, $X \subseteq atom(\Pi)$ is an answer set of Π iff $X = A^{\mathbf{T}} \cap atom(\Pi)$ for a (unique) non-conflicting total assignment A over $atom(\Pi) \cup body(\Pi)$ such that, for any nogood $\delta \in \Delta_\Pi$, we have $\overline{\sigma} \in A$ for some $\sigma \in \delta$.

4.2 Restricted Strategies and Exponential Blow-Ups

Characterizations of answer sets in terms of either only atoms or rules are sufficient and produce equivalent results. However, we have shown in [18] that either restricted approach yields an exponentially weaker proof system than obtained when using both atoms and rule bodies. Looking at nogoods, we will see in Section 5 that even *smodels*, assigning only atoms, does not truly restrict its inferences to atoms (the same accounts for rules and *noMoRe* [19]). In this section, we describe the nogoods (or clauses, respectively) of truly atom- or body-centered approaches. Both restrictions yield exponentially more nogoods, which inevitably result in exponentially more branches that have to be explored by DLL.[5]

The nogoods of atom- and body-centered approaches can be obtained in a natural way using clauses and resolution. Let us recall the clauses reflecting the nogoods given in the previous section. For a body $\beta = \{p_1, \ldots, p_m, not\ p_{m+1}, \ldots, p_n\}$ and an atom p such that $body(p) = \{\beta_1, \ldots, \beta_k\}$, we have:

$$\begin{aligned}
\gamma(\delta(\beta)) &= \{p_\beta, \neg p_1, \ldots, \neg p_m, p_{m+1}, \ldots, p_n\} \\
\Gamma(\Delta(\beta)) &= \{\, \{\neg p_\beta, p_1\}, \ldots, \{\neg p_\beta, p_m\}, \{\neg p_\beta, \neg p_{m+1}\}, \ldots, \{\neg p_\beta, \neg p_n\} \,\} \\
\Gamma(\Delta(p)) &= \{\, \{p, \neg p_{\beta_1}\}, \ldots, \{p, \neg p_{\beta_k}\} \,\} \\
\gamma(\delta(p)) &= \{\neg p, p_{\beta_1}, \ldots, p_{\beta_k}\}
\end{aligned}$$

For obtaining the nogoods of a purely atom- or body-centered approach, we can resolve out either bodies or atoms from the above clauses. However, observe that resolving clause $\gamma(\delta(\beta))$ against any clause in $\Gamma(\Delta(\beta))$ yields a tautology. The same applies to clause $\gamma(\delta(p))$ and clauses in $\Gamma(\Delta(p))$. For resolving out bodies in a reasonable way, we thus have to resolve clauses in $\Gamma(\Delta(p))$ against clause $\gamma(\delta(\beta))$, and clause $\gamma(\delta(p))$ against clauses in $\Gamma(\Delta(\beta))$. Similarly, we can eliminate atoms by resolving clause $\gamma(\delta(\beta))$ against clauses in $\Gamma(\Delta(p))$ for positive body literals and against $\gamma(\delta(p))$ for negative body literals. With roles of positive and negative body literals interchanged, clauses in $\Gamma(\Delta(\beta))$ can be resolved against $\gamma(\delta(p))$ and $\Gamma(\Delta(p))$. This

[5] Note that an exponential number of nogoods does not oblige exponential space complexity, as nogoods need not necessarily be represented explicitly.

yields the following resolution transformations for eliminating either bodies or atoms:

$$\Gamma_\alpha(\Pi) = \{\, \{p, \neg p_1, \ldots, \neg p_m, p_{m+1}, \ldots, p_n\} \mid \pi \in \Pi, \\ head(\pi) = p, body(\pi) = \{p_1, \ldots, p_m, not\ p_{m+1}, \ldots, not\ p_n\} \,\} \quad (5)$$

$$\cup\ \{\, \{\neg p, q_1, \ldots, q_k\} \mid p \in atom(\Pi), body(p) = \{\beta_1, \ldots, \beta_k\}, \\ q_i = p_i \text{ if } p_i \in \beta_i^+ \text{ or } q_i = \neg p_i \text{ if } p_i \in \beta_i^- \text{ for } 1 \leq i \leq k \,\} \quad (6)$$

$$\Gamma_\beta(\Pi) = \{\, \{p_\beta, \neg p_{\beta_1}, \ldots, \neg p_{\beta_m}, p_{\beta_{1_{m+1}}}, \ldots, p_{\beta_{k_{m+1}}}, \ldots, p_{\beta_{1_n}}, \ldots, p_{\beta_{k_n}}\} \mid \\ (\beta = \{p_1, \ldots, p_m, not\ p_{m+1}, \ldots, not\ p_n\}) \in body(\Pi), \\ \beta_i \in body(p_i) \text{ for } 1 \leq i \leq m, \\ body(p_j) = \{\beta_{1_j}, \ldots, \beta_{k_j}\} \text{ for } m+1 \leq j \leq n \,\} \quad (7)$$

$$\cup\ \{\, \{\neg p_\beta, p_{\beta_1}, \ldots, p_{\beta_k}\} \mid \beta \in body(\Pi), p \in \beta^+, body(p) = \{\beta_1, \ldots, \beta_k\} \,\} \quad (8)$$

$$\cup\ \{\, \{\neg p_\beta, \neg p_{\beta'}\} \mid \beta \in body(\Pi), p \in \beta^-, \beta' \in body(p) \,\} \quad (9)$$

Let us illustrate the above transformations on two examples. For Γ_α, consider an atom x being the head of rules $x \leftarrow y_1, not\ y_2$ and $x \leftarrow z_1, not\ z_2$. Applying (5) for both rules, we get two clauses: $\{x, \neg y_1, y_2\}$ and $\{x, \neg z_1, z_2\}$. Applying (6) for x, we obtain four clauses: $\{\neg x, y_1, z_1\}$, $\{\neg x, y_1, \neg z_2\}$, $\{\neg x, \neg y_2, z_1\}$, and $\{\neg x, \neg y_2, \neg z_2\}$. Together, the above six clauses are equivalent to: $x \equiv ((y_1 \wedge \neg y_2) \vee (z_1 \wedge \neg z_2))$. We now consider Γ_β. Let $\beta = \{x, not\ y\}$ be a body, $body(x) = \{\beta_1^x, \beta_2^x\}$, and $body(y) = \{\beta_1^y, \beta_2^y\}$. By (7), we get the following two clauses for β: $\{p_\beta, \neg p_{\beta_1^x}, p_{\beta_1^y}, p_{\beta_2^y}\}$ and $\{p_\beta, \neg p_{\beta_2^x}, p_{\beta_1^y}, p_{\beta_2^y}\}$. For $x \in \beta^+$, applying (8) gives clause: $\{\neg p_\beta, p_{\beta_1^x}, p_{\beta_2^x}\}$. Two more clauses are obtained by applying (9) for $y \in \beta^-$: $\{\neg p_\beta, \neg p_{\beta_1^y}\}$ and $\{\neg p_\beta, \neg p_{\beta_2^y}\}$. Together, the above five clauses are equivalent to: $p_\beta \equiv ((p_{\beta_1^x} \vee p_{\beta_2^x}) \wedge \neg(p_{\beta_1^y} \vee p_{\beta_2^y}))$.

For a program Π, $\Gamma_\alpha(\Pi)$ and $\Gamma_\beta(\Pi)$ correspond to the completion of Π, represented in CNF with propositional atoms for either only the atoms or the bodies of Π. Hence, the nogoods expressed by $\Gamma_\alpha(\Pi)$ and $\Gamma_\beta(\Pi)$ correspond to truly atom- or body-centered approaches. Note that (6) and (7) amount to distributivity. As a consequence, both $\Gamma_\alpha(\Pi)$ and $\Gamma_\beta(\Pi)$ can yield an exponential blow-up. As mentioned above, exponential space complexity can be omitted by implicitly constructing clauses in $\Gamma_\alpha(\Pi)$ and $\Gamma_\beta(\Pi)$ from Π during propagation, instead of storing them explicitly. The above transformations do thus correspond to restricted branching strategies, that is, restrictions of *Select* in Figure 1 to either only atoms or rule bodies. Such restrictions are sound and complete, given that the truth values of bodies can be determined from atoms' values, and vice versa (cf. Proposition 1 and 2). In fact, branching is restricted to atoms in *smodels*, and in *noMoRe* to rule bodies. Note that neither *smodels* nor *noMoRe* exploit their restricted branching strategies through corresponding constraints, such as $\Gamma_\alpha(\Pi)$ and $\Gamma_\beta(\Pi)$ from which literals that can only be implied are removed.

In what follows, we show that eliminating "redundant" program objects (either atoms or bodies) can lead to exponentially worse minimal run-times of computations. The logic program families witnessing this behavior are also used in [18] in the context of tableau proof systems. In contrast to [18], the results provided here are independent from specific computation patterns, that is, the inference rules applied by a "prover" are immaterial. In fact, inevitably exponential run-times result only from the analysis of nogoods (or clauses) that have to be considered in computations. For demonstrating

the exponential behavior, we use the following programs:

$$\Pi^n = \{\, x_1 \leftarrow not\ y_1 \quad y_1 \leftarrow not\ x_1 \quad \ldots \quad x_n \leftarrow not\ y_n \quad y_n \leftarrow not\ x_n \,\}$$
$$\Pi^n_\alpha = \{\, w \leftarrow not\ w \quad w \leftarrow not\ x_1, not\ y_1 \quad \ldots \quad w \leftarrow not\ x_n, not\ y_n \,\}$$
$$\Pi^n_\beta = \left\{ \begin{array}{l} v \leftarrow z_1, \ldots, z_n, not\ v \\ z_1 \leftarrow not\ x_1 \quad z_1 \leftarrow not\ y_1 \quad \ldots \quad z_n \leftarrow not\ x_n \quad z_n \leftarrow not\ y_n \end{array} \right\}$$

The programs of both families $\{\Pi^n \cup \Pi^n_\alpha\}$ and $\{\Pi^n \cup \Pi^n_\beta\}$ have no answer sets. Atoms x_i and y_i ($1 \leq i \leq n$) are mutually exclusive due to rules in Π^n. For Π^n_α, the body of any rule $w \leftarrow not\ x_i, not\ y_i$ can thus not be true, leaving only the self-contradictory rule $w \leftarrow not\ w$ for atom w. For Π^n_β, all atoms z_i have to be true, making rule $v \leftarrow z_1, \ldots, z_n, not\ v$ self-contradictory. Note that the nogoods associated with atoms and bodies of $\Pi^n \cup \Pi^n_\alpha$ and $\Pi^n \cup \Pi^n_\beta$ allow DLL to prove unsatisfiability with only a linear number of calls to $Select$: Unit propagation yields an immediate conflict when a literal of form $\mathbf{T}\{not\ x_i, not\ y_i\}$ is selected for $\Pi^n \cup \Pi^n_\alpha$ or of form $\mathbf{F}z_i$ for $\Pi^n \cup \Pi^n_\beta$. Considering $\Gamma_\alpha(\Pi^n \cup \Pi^n_\alpha)$ and $\Gamma_\beta(\Pi^n \cup \Pi^n_\beta)$, the obtained sets of clauses are isomorphic, that is, only the names of propositional atoms differ, while the structure is the same. Omitting a complete description, we however mention that $\Gamma_\alpha(\Pi^n \cup \Pi^n_\alpha)$ contains 2^n clauses of form $\{\neg w, \neg q_1, \ldots, \neg q_n\}$ such that $q_i \in \{x_i, y_i\}$ for $1 \leq i \leq n$, similarly, $\Gamma_\beta(\Pi^n \cup \Pi^n_\beta)$ contains 2^n clauses of form $\{p_{\{z_1,\ldots,z_n,not\ v\}}, \neg p_1, \ldots, \neg p_n\}$ such that $p_i \in \{p_{\{not\ x_i\}}, p_{\{not\ y_i\}}\}$ for $1 \leq i \leq n$. These clauses enable us to show exponential minimal run-times of DLL computations by a semantic argument, eliding the computation itself.

Proposition 5. *Let $\Pi^n \cup \Pi^n_\alpha$ and $\Pi^n \cup \Pi^n_\beta$ be as described above.*

1. *$\Gamma_\alpha(\Pi^n \cup \Pi^n_\alpha) \setminus \{\gamma\}$ is satisfiable for any clause $\gamma = \{\neg w, \neg q_1, \ldots, \neg q_n\}$ such that $q_i \in \{x_i, y_i\}$ for $1 \leq i \leq n$.*
2. *$\Gamma_\beta(\Pi^n \cup \Pi^n_\beta) \setminus \{\gamma\}$ is satisfiable for any clause $\gamma = \{p_{\{z_1,\ldots,z_n,not\ v\}}, \neg p_1, \ldots, \neg p_n\}$ such that $p_i \in \{p_{\{not\ x_i\}}, p_{\{not\ y_i\}}\}$ for $1 \leq i \leq n$.*

The result shows that, for programs $\Pi^n \cup \Pi^n_\alpha$ and $\Pi^n \cup \Pi^n_\beta$, the exponentially many nogoods associated with clauses in $\Gamma_\alpha(\Pi^n \cup \Pi^n_\alpha)$ and $\Gamma_\beta(\Pi^n \cup \Pi^n_\beta)$ must inevitably be considered by DLL in order to verify unsatisfiability. Recall that the reason for this is not an exponential space requirement. Rather, limited branching leads to exponentially many (symmetric) cases that have to be analyzed.

4.3 Non-Tight Programs

For treating non-tight programs, we need to identify atoms lacking a non-circular support: For a program Π and a subprogram Π' of Π, we define the *greatest unfounded set* [20], denoted $GUS_\Pi(\Pi')$, as the $\subseteq$-maximal subset U of $atom(\Pi)$ such that, for any rule $\pi \in \Pi'$, we have either $head(\pi) \notin U$ or $body(\pi) \cap U \neq \emptyset$.[6] Note that, for any $\Pi' \subseteq \Pi$, $GUS_\Pi(\Pi')$ exists and is unique.

[6] For a partial interpretation (X, Y) such that X contains the atoms that are true and Y the atoms that are false, the original definition from [20] is obtained when we let $\Pi' = \{\pi \in \Pi \mid body(\pi)^+ \cap Y = \emptyset, body(\pi)^- \cap X = \emptyset\}$. We however leave the choice of Π' open here.

Given a non-tight program Π and an assignment A, *nomore++*'s operator $\mathcal{U}_\Pi$ [5] handles greatest unfounded sets. That is, it falsifies all atoms in $GUS_\Pi(\{\pi \in \Pi \mid body(\pi) \notin A^{\mathbf{F}}\})$. Given that in principle any subset of $body(\Pi)$ may induce an unfounded set, we need nogoods for every possible subset Θ of $body(\Pi)$. Any such Θ might form the set $body(\Pi) \cap A^{\mathbf{F}}$ of false bodies wrt some assignment A. The respective nogoods will contain $\mathbf{F}\beta$ for every body $\beta \in \Theta$ and $\mathbf{T}p$ for every atom $p \in GUS_\Pi(\{\pi \in \Pi \mid body(\pi) \notin \Theta\})$. However, a nogood of form $\{\mathbf{T}p, \mathbf{F}\beta_1, \ldots, \mathbf{F}\beta_k\}$, where $\Theta = \{\beta_1, \ldots, \beta_k\}$, would allow unit propagation to work in backward direction as well. That is, it would allow for deriving $\mathbf{T}\beta_i$ if A contains $\mathbf{T}p$ and $\mathbf{F}\beta_j$ for $1 \leq j \leq k, j \neq i$. Although this is logically correct, it is not done by operator $\mathcal{U}_\Pi$. Hence, for exactly characterizing the inference scheme of *nomore++*, we need a mechanism for preventing backward propagation. The solution is to use extra propositions, μ_Θ and ν_Θ, that are uniquely associated with a set Θ of bodies. For Π and a set $\Theta = \{\beta_1, \ldots, \beta_k\} \subseteq body(\Pi)$, we then get the following nogoods:

$$\Delta(\Theta) = \{\, \{\mathbf{T}p, \mathbf{T}\mu_\Theta, \mathbf{T}\nu_\Theta\}, \{\mathbf{F}\mu_\Theta, \mathbf{F}\beta_1, \ldots, \mathbf{F}\beta_k\}, \{\mathbf{F}\nu_\Theta, \mathbf{F}\beta_1, \ldots, \mathbf{F}\beta_k\} \mid p \in GUS_\Pi(\{\pi \in \Pi \mid body(\pi) \notin \Theta\}) \,\}$$

This set of nogoods captures all possible inferences of $\mathcal{U}_\Pi$ falsifying unfounded sets.

Theorem 3. *Let Π be a logic program and A be an assignment over $atom(\Pi) \cup body(\Pi)$. Let $\Delta_U = \{\, \delta \in \Delta(\Theta) \mid \Theta \subseteq body(\Pi) \,\}$.*

For $B = UP^*(\Delta_U, A)$ and $C = \mathcal{U}_\Pi(A)$, we have the following:

1. *B is conflicting iff C is conflicting.*
2. *If neither B nor C is conflicting, then $B^{\mathbf{F}} = C^{\mathbf{F}}$.*

Of course, Δ_U contains exponentially many nogoods since there are exponentially many subsets Θ of $body(\Pi)$. However, the nogoods need not be represented explicitly. In fact, any genuine ASP solver propagates them by relying on their implicit representation by Π. The necessity to include additional propositions μ_Θ and ν_Θ in nogoods reflects the restriction of unfounded set handling to forward propagation, cutting off logically valid consequences. This is an obvious subject to future improvement.

Finally, we combine the results for tight and non-tight programs, for showing that the provided nogoods do in fact characterize answer sets by unit propagation and DLL.

Theorem 4. *Let Π be a logic program, Δ_Π as in Theorem 1, and Δ_U as in Theorem 3.*

Then, $X \subseteq atom(\Pi)$ is an answer set of Π iff $X = A^{\mathbf{T}} \cap atom(\Pi)$ for a (unique) non-conflicting total assignment A over $atom(\Pi) \cup body(\Pi) \cup \{\mu_\Theta, \nu_\Theta \mid \Theta \subseteq body(\Pi)\}$ such that, for any $\Theta \subseteq body(\Pi)$ such that $\Theta \not\subseteq A^{\mathbf{F}}$, we have $\{\mu_\Theta, \nu_\Theta\} \subseteq A^{\mathbf{F}}$, and, for any nogood $\delta \in \Delta_\Pi \cup \Delta_U$, we have $\overline{\sigma} \in A$ for some $\sigma \in \delta$.

5 A Constraint Counterpart of *smodels*

The nogoods underlying *smodels* are very similar to the ones of *nomore++*, except that bodies are considered rule-wise. This introduces redundancy because more than one

proposition might refer to the same body. For tight programs, similar results as ours have been obtained in [21] in terms of clauses. We here only briefly provide the nogoods describing propagation by *smodels*' functions $atleast$ and $atmost$ [6]. As in [21], we uniquely associate a new proposition β_π with each rule π. Such a proposition captures the notion of an "(in)active rule" used in [22] for describing *smodels*' implementation.

For a program Π and a rule $\pi \in \Pi$ with $head(\pi) = p$ and $body(\pi) = \{p_1, \ldots, p_m, not\ p_{m+1}, \ldots, not\ p_n\}$, let:

$$\Delta(\pi) = \{\, \{\mathbf{F}p, \mathbf{T}\beta_\pi\}, \{\mathbf{F}\beta_\pi, \mathbf{T}p_1, \ldots, \mathbf{T}p_m, \mathbf{F}p_{m+1}, \ldots, \mathbf{F}p_n\} \,\}$$

For an atom p such that $\{\pi \in \Pi \mid head(\pi) = p\} = \{\pi_1, \ldots, \pi_k\}$, let:

$$\begin{aligned}\Delta(p) = {} & \{\, \{\mathbf{T}p, \mathbf{F}\beta_{\pi_1}, \ldots, \mathbf{F}\beta_{\pi_k}\} \,\} \cup \\ & \textstyle\bigcup_{1 \leq i \leq k} \big(\{\, \{\mathbf{T}\beta_{\pi_i}, \mathbf{F}p'\} \mid p' \in body(\pi_i)^+ \,\} \cup \{\, \{\mathbf{T}\beta_{\pi_i}, \mathbf{T}p'\} \mid p' \in body(\pi_i)^- \,\} \big)\end{aligned}$$

Observe that the above nogoods are quite similar to those of *nomore++* in the tight case. The difference is that propositions for bodies are introduced rule-wise. In fact, as also observed in [21], the exclusion of body representatives in the corresponding nogoods fails to capture *smodels*' $atleast$. To see this, consider $\Pi = \{a \leftarrow not\ a\}$ being logically equivalent to clause $\{a\}$. Still, $atleast$ does not derive $\mathbf{T}a$ from the empty assignment. This shows that *smodels*' propagation does in fact rely on the truth status of rule bodies and does not go "through" them. By including rule bodies in nogoods, we obtain the following correspondence to $atleast$.

Theorem 5. *Let Π be a logic program and A be an assignment over $atom(\Pi)$. Let*

$$\Delta_\Pi = \{\, \delta \in \Delta(\pi) \mid \pi \in \Pi \,\} \cup \{\, \delta \in \Delta(p) \mid p \in atom(\Pi) \,\}\,.$$

For $B = UP^*(\Delta_\Pi, A)$ and $C = atleast(\Pi, A)$, we have the following:

1. *B is conflicting iff C is conflicting.*
2. *If neither B nor C is conflicting, then $(B^\mathbf{T} \cap atom(\Pi), B^\mathbf{F} \cap atom(\Pi)) = (C^\mathbf{T}, C^\mathbf{F})$.*

For a non-tight program Π and an assignment A over $atom(\Pi)$, let $\Pi_A = \{\pi \in \Pi \mid body(\pi)^+ \cap A^\mathbf{F} = \emptyset, body(\pi)^- \cap A^\mathbf{T} = \emptyset\}$. That is, Π_A contains all rules whose bodies are not already determined to be false by the literals in A. Then *smodels*' function $atmost$ determines the set $atom(\Pi) \backslash (GUS_\Pi(\Pi_A) \cup A^\mathbf{F})$, containing all atoms that can potentially be true in an extension of A. All other atoms must be false, so *smodels* adds $\mathbf{F}p$ to A for all atoms $p \in GUS_\Pi(\Pi_A) \backslash A^\mathbf{F}$. In this respect, also unfounded set handling of *smodels* and *nomore++* is very similar. In contrast to *nomore++* where bodies can be assigned $\mathbf{F}$, *smodels* however determines inapplicable rules with false bodies from the literals in A. This gives us the choice to include either program atoms or additional propositions β_π for rule bodies in nogoods reflecting unfounded set inference. Note that a false rule body is sufficient to know that the rule is inapplicable and cannot support the head atom; this is independent from which literal in the body is false. Thus, it makes sense to reuse propositions β_π. For a program Π and a subset $\Xi = \{\pi_1, \ldots, \pi_k\}$ of Π, we then get similar nogoods as with *nomore++*:

$$\begin{aligned}\Delta(\Xi) = \{\, & \{\mathbf{T}p, \mathbf{T}\mu_\Xi, \mathbf{T}\nu_\Xi\}, \{\mathbf{F}\mu_\Xi, \mathbf{F}\beta_{\pi_1}, \ldots, \mathbf{F}\beta_{\pi_k}\}, \\ & \{\mathbf{F}\nu_\Xi, \mathbf{F}\beta_{\pi_1}, \ldots, \mathbf{F}\beta_{\pi_k}\} \mid p \in GUS_\Pi(\Pi \setminus \Xi) \,\}\end{aligned}$$

Again μ_Ξ and ν_Ξ are new propositions uniquely associated with set Ξ for inhibiting backward propagation, which is not done by *smodels* via function $atmost$.

Finally, *smodels*' function $expand$ iterates propagation via $atleast$ and $atmost$ until a fixpoint is reached. Combining the nogoods that describe $atleast$ and $atmost$ by unit propagation, we have the following correspondence.

Theorem 6. *Let Π be a logic program and A be an assignment over $atom(\Pi)$. Let Δ_Π as in Theorem 5 and $\Delta_U = \{\, \delta \in \Delta(\Xi) \mid \Xi \subseteq \Pi \,\}$.*

For $B = UP^*(\Delta_\Pi \cup \Delta_U, A)$ and $C = expand(\Pi, A)$, we have the following:

1. *B is conflicting iff C is conflicting.*
2. *If neither B nor C is conflicting, then $(B^\mathbf{T} \cap atom(\Pi), B^\mathbf{F} \cap atom(\Pi)) = (C^\mathbf{T}, C^\mathbf{F})$.*

Note that, regardless of the different use of propositions for bodies, all results obtained in Section 4 carry over from *nomore++* to *smodels*. While inferences in *nomore++* and *smodels* are semantically equivalent, both solvers differ in their selection of branching literals, and in their representation of bodies: explicit in *nomore++*, while implicit and attached to rules in *smodels*.

6 Discussion

By characterizing inferences of ASP solvers *nomore++* and *smodels* in terms of unit propagation, we have not only revealed their structural similarity, but we have seen that both ASP solvers inherently incorporate program atoms and rule bodies (rules must be true anyway) into their solving process. The results in Section 4.2 and in [18] show that this is a sensible design: Restricting attention completely to either only atoms or bodies eliminates redundancy, but is payed with exponentially many cases to be analyzed inevitably in the worst case. In this respect, *nomore++*'s approach to identify both atoms and bodies of a program as equitable propositions adopts redundancy for having at least the chance to select the "right" branching literals. Notably, incorporating bodies as unique entities yields a stronger unit propagation than found in *smodels*, where bodies are attached to rules so that duplicate occurrences are not detected. To see this, consider the rules $x \leftarrow y, z$ and $w \leftarrow y, z$ sharing body $\{y, z\}$. If x is false in an assignment, then rule $x \leftarrow y, z$ must not be applied. Even when y and z are yet undefined, body $\{y, z\}$ must be false. However, *smodels* is unable to detect this, so it does not recognize that rule $w \leftarrow y, z$ is inapplicable as well. In contrast, *nomore++* assigns false to body $\{y, z\}$, and if $\{y, z\}$ is the only body with head atom w, then, unlike *smodels*, *nomore++* derives that w must be false. In fact, propagation of *smodels* is strictly weaker than the one of *nomore++* without being "cheaper" to compute; as shown in Section 5, we have to incorporate rule bodies in nogoods for an exact characterization of *smodels*' propagation. So the only real effect of excluding rule bodies from assignments is that only atoms are branched upon. As we discussed before, this risks exponentially worse minimal run-times compared to unrestricted branching.

Nogoods can be expressed as clauses in a straightforward manner, so it is interesting to compare genuine ASP solvers with SAT-based ones, doing an explicit CNF conversion. In order to avoid an exponential blow-up, *cmodels* introduces auxiliary propositional atoms abbreviating bodies [16]. Such auxiliary atoms are introduced rule-wise,

similar to the respective propositions in *smodels*' nogoods (cf. Section 5). As shown above, duplicated bodies might thus not be detected by *cmodels*, while *nomore++* might be able to apply unit propagation. However, *cmodels*' CNF conversion tries to avoid introducing auxiliary atoms if possible: The strategy is to only introduce an auxiliary atom if a program atom is the head of more than one rule with more than one literal in the body. When we consider an atom x being the head of the rules $x \leftarrow y, z$ and $x \leftarrow y$, then *cmodels* does not introduce auxiliary atoms and derives, from x being true in an assignment, that y must be true as well. In contrast, *nomore++* is not able to apply unit propagation, as there are two bodies, $\{y, z\}$ and $\{y\}$, for head atom x.[7]

In summary, atoms and bodies as explicit propositions are useful: They prevent an exponential blow-up in CNF conversions and also avoid (unnecessary) exponential minimal run-times of DLL, as shown in Section 4.2. Assuming that neither atoms nor bodies are eliminated, the propagation obtained for purely atom- or body-centered approaches amounts to inferences going beyond unit propagation. Regarding nogoods and their corresponding clauses, considering exclusively atoms or bodies can be seen as a kind of one-step resolution, resolving out program objects not under consideration. Such kind of inference is possible, even without an exponential blow-up. It however means that an ASP solver using such inference is not a DLL-like solver, as inferences go beyond unit propagation and rather amount to a restricted form of resolution. It remains a subject for future research whether such extended inference patterns pay off. Finally, using atoms and bodies in nogoods greatly facilitates the description of solvers and their proving strategies. Instead of multiple inference rules, we can use a uniform framework based on unit propagation. Nogoods can easily be converted into clauses, which can be used for analyzing the logical contents of inferences; e.g., Propositions 1 and 2 show that bodies correspond to conjunctions (of body literals) and atoms to disjunctions (of bodies). The one-to-one correspondence between our framework and propositional logic might enhance the understanding of ASP and corresponding solvers, and allow for fruitful knowledge transfer. As direct applications, we can see the integration of SAT solver techniques, like clause learning, into ASP solvers, and the emergence of "serious" proof-theoretical investigations of ASP, similar to resolution for SAT.

In contrast to propositional CNF formulas containing an explicit clause for every (known) nogood, atoms and rule bodies in logic programs are subject to rather implicit nogoods, induced by the more intricate answer set semantics. Of course, these nogoods might also be explicated as a CNF formula, but the logic program format is more succinct, similar to Boolean circuits compared to plain CNF. For tight programs, the CNF conversion however is straightforward and reasonably economical regarding space, provided that extra propositional symbols are introduced for abbreviating rule bodies. With non-tight programs, the situation is different, and the number of nogoods is inherently exponential [24]. Thus, SAT-based solvers add respective clauses incrementally whenever a non-stable supported model needs to be eliminated, but not a priori. Genuine ASP solvers also do not represent all nogoods explicitly, rather, they incorporate unfounded set checks into their propagation. This takes advantage of the fact that the exponen-

[7] However, a redundancy like a singleton body being contained in a non-singleton one could be eliminated in preprocessing, using the NONMIN reduction [23].

tially many nogoods dealing with unfounded sets can be checked in polynomial (even in linear) time.

However, the implicit nogoods underlying unit propagation for unfounded sets are somewhat peculiar (cf. Section 4.3 and 5). This is because unfounded set inference only works in forward direction, falsifying unfounded atoms. True atoms are however not protected from becoming unfounded, so conflicts arising from such situations are not prevented. In this respect, unfounded set propagation within genuine ASP solvers is asymmetric. Such a kind of directional propagation does not exist in the context of general CSP and constraint propagation. Thus, we require additional "intermediate" propositions to disable backward propagation when we characterize unfounded set inference in terms of nogoods and unit propagation. Clearly, doing "real" unit propagation also for unfounded sets is a subject to future research.

Finally, let us comment on the logical fundament of unfounded set inference. The nogoods describing the falsification of unfounded sets essentially result from *loop formulas* [1, 25]. Greatest unfounded sets are not necessarily *loops* in the sense of [1], still loop formulas can be constructed. As shown in [25], answer sets are exactly the models of a program that satisfy the loop formulas of all nonempty subsets of the program's atoms. Hence, all loop formulas are implicitly checked by ASP solvers and must be satisfied by solutions. For the characterization of answer sets, it suffices to concentrate on loop formulas of crucial sets of atoms, namely loops. From this perspective, greatest unfounded set falsification bears redundancy. On the implementation level, ASP solvers partially address this redundancy [6, 26], but on the logical side inferences still amount to greatest unfounded set falsification. Recent results show that even loops do not yet render relevant unfounded sets precisely: Concentrating on *elementary sets* [27], which form a subset of a program's loops, is enough. Such advanced concepts are not yet exploited by genuine ASP solvers and are another open issue for future research.

Acknowledgments. This work was supported by DFG (SCHA 550/6-4). We are grateful to Ilkka Niemela and the anonymous referee for their helpful suggestions.

References

1. Lin, F., Zhao, Y.: ASSAT: computing answer sets of a logic program by SAT solvers. Artificial Intelligence **157**(1-2) (2004) 115–137
2. Giunchiglia, E., Lierler, Y., Maratea, M.: A SAT-based polynomial space algorithm for answer set programming. In Delgrande, J., Schaub, T., eds.: Proceedings of the Tenth International Workshop on Non-Monotonic Reasoning (NMR'04). (2004) 189–196
3. Mitchell, D.: A SAT solver primer. Bulletin of the European Association for Theoretical Computer Science **85** (2005) 112–133
4. Ward, J., Schlipf, J.: Answer set programming with clause learning. In Lifschitz, V., Niemelä, I., eds.: Proceedings of the Seventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'04). Springer (2004) 302–313
5. Anger, C., Gebser, M., Linke, T., Neumann, A., Schaub, T.: The `nomore++` approach to answer set solving. In Sutcliffe, G., Voronkov, A., eds.: Proceedings of the Twelfth International Conference on Logic for Programming Artificial Intelligence and Reasoning (LPAR'05). Springer (2005) 95–109

6. Simons, P., Niemelä, I., Soininen, T.: Extending and implementing the stable model semantics. Artificial Intelligence **138**(1-2) (2002) 181–234
7. Leone, N., Faber, W., Pfeifer, G., Eiter, T., Gottlob, G., Koch, C., Mateis, C., Perri, S., Scarcello, F.: The DLV system for knowledge representation and reasoning. ACM Transactions on Computational Logic (2006) To appear.
8. Dechter, R.: Constraint Processing. Morgan Kaufmann (2003)
9. Lifschitz, V.: Foundations of logic programming. In Brewka, G., ed.: Principles of Knowledge Representation. CSLI Publications (1996) 69–127
10. Reiter, R.: A logic for default reasoning. Artificial Intelligence **13**(1-2) (1980) 81–132
11. Linke, T.: Graph theoretical characterization and computation of answer sets. In Nebel, B., ed.: Proceedings of the International Joint Conference on Artificial Intelligence, Morgan Kaufmann (2001) 641–645
12. Faber, W., Leone, N., Pfeifer, G.: Recursive aggregates in disjunctive logic programs: Semantics and complexity. In Alferes, J., Leite, J., eds.: Proceedings of the Nineth European Conference of Logics in Artificial Intelligence. Springer (2004) 200–212
13. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem-proving. Communications of the ACM **5** (1962) 394–397
14. Erdem, E., Lifschitz, V.: Tight logic programs. Theory and Practice of Logic Programming **3**(4-5) (2003) 499–518
15. Clark, K.: Negation as failure. In Gallaire, H., Minker, J., eds.: Logic and Data Bases. Plenum Press (1978) 293–322
16. Babovich, Y., Lifschitz, V.: Computing answer sets using program completion. Unpublished draft (2003)
17. Fages, F.: Consistency of Clark's completion and the existence of stable models. Journal of Methods of Logic in Computer Science **1** (1994) 51–60
18. Gebser, M., Schaub, T.: Tableau calculi for answer set programming. In Etalle, S., Truszczyński, M., eds.: Proceedings of the Twenty-second International Conference on Logic Programming (ICLP'06). Springer (2006) To appear.
19. Konczak, K., Linke, T., Schaub, T.: Graphs and colorings for answer set programming. Theory and Practice of Logic Programming **6**(1-2) (2006) 61–106
20. van Gelder, A., Ross, K., Schlipf, J.: The well-founded semantics for general logic programs. Journal of the ACM **38**(3) (1991) 620–650
21. Giunchiglia, E., Maratea, M.: On the relation between answer set and SAT procedures (or, between cmodels and smodels). In Gabbrielli, M., Gupta, G., eds.: Proceedings of the Twenty-first International Conference on Logic Programming (ICLP'05). Springer (2005) 37–51
22. Simons, P.: Extending and Implementing the Stable Model Semantics. Dissertation, Helsinki University of Technology (2000)
23. Brass, S., Dix, J.: Semantics of (disjunctive) logic programs based on partial evaluation. Journal of Logic Programming **40**(1) (1999) 1–46
24. Lifschitz, V., Razborov, A.: Why are there so many loop formulas? ACM Transactions on Computational Logic (2006) To appear.
25. Lee, J.: A model-theoretic counterpart of loop formulas. In Kaelbling, L., Saffiotti, A., eds.: Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI'05), Professional Book Center (2005) 503–508
26. Calimeri, F., Faber, W., Leone, N., Pfeifer, G.: Pruning operators for answer set programming systems. Technical Report INFSYS RR-1843-01-07, Technische Universität Wien (2001)
27. Gebser, M., Lee, J., Lierler, Y.: Elementary sets for logic programs. In Dix, J., Hunter, A., eds.: Proceedings of the Eleventh International Workshop on Nonmonotonic Reasoning (NMR'06). IFI-06-04 Clausthal University of Technology (2006) 68–75