A First-Order Representation of Stable Models*

Thomas Eiter, James Lu and V.S. Subrahmanian
Institut für Informatik, Universität Gießen, Arndtstraße 2, D-35392 Gießen, Germany. E-mail: eiter@informatik.uni-giessen.de

Department of Computer Science, Bucknell University, Lewisburg, PA. E-mail: lu@sol.cs.bucknell.edu

Institute for Advanced Computer Studies, Institute for Systems Research and Department of Computer Science, University of Maryland, College Park, Maryland 20742. E-mail: vs@cs.umd.edu

Turi (1991) introduced the important notion of a constrained atom: an atom with associated equality and disequality constraints on its arguments. A set of constrained atoms is a constrained interpretation. We investigate how non-ground representations of *both* the stable model semantics and the well-founded semantics may be obtained through Turi's approach. The practical implication of this is that the well-founded model (or the set of stable models) may be partially pre-computed at compile-time, resulting in the association of each predicate symbol in the program to a constrained atom. Algorithms to create such models are presented, both for the well founded case, and the case of stable models. Query processing reduces to checking whether each atom in the query is *true* in a stable model (resp. well-founded model). This amounts to showing the atom is an instance of one of some constrained atom whose associated constraint is solvable. Various related complexity results are explored, and the impacts of these results are discussed from the point of view of implementing systems that incorporate the stable and well-founded semantics.

Keywords: stable models, compact representation, automated reasoning

1. Introduction

Logic programming is commonly known as an alternative to the imperative programming paradigm. However, it has become clear more recently that it is also an important formalism for knowledge representation, cf. [1]. As it turned out, a definition of negation in logic programs is nontrivial, and has become a key issue of the research over the last decade. A large number of different semantics have been proposed in the past (see e.g. [7]), of which the stable model semantics [14] and the well-founded semantics (WFS, for short) [36] have received the main attention. The stable model semantics is closely related to well-known forms of nonmonotonic reasoning such as default logic [31] and circumscription [27, 21]. In fact, normal logic programs can be viewed as a fragment of Reiter's default logic.

The definition of most nonmonotonic logic programming semantics, including the stable semantics and the WFS, has been largely based on methods that assume *propositional* programs. For example, the stable models of a logic program are defined as certain minimal Herbrand models of the ground instantiation of the program. In addition, the Gelfond-Lifschitz transform, which plays a key role for both the stable semantics and the WFS, works only on ground programs [35, 2]. Clearly, when our interest is in the computer implementation of these semantics, the requirement of ground instantiations of programs can quickly encounter practical limitations. In addition, the representation of both stable and well-founded models using ground atoms is also highly impractical. To see this, consider the following simple example.

Example 1 Let P be the following logic program:

$$\begin{aligned} p(X,X) &\leftarrow \\ q(X,Y) &\leftarrow \mathbf{not}(p(X,Y)) \\ r(a,f(a)) &\leftarrow \end{aligned}$$

This program, which is stratified, has a unique stable model that is also the well-founded model. It satisfies all atoms of the form $p(X,X)$, the single atom

*A preliminary version of this paper has appeared in: Proc. 4th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR '97), J. Dix, U. Furbach, and A. Nerode (eds), LNCS 1265. Springer, 1997.

AI Communications 11 (1998) 53–73. PREPRINT
ISSN 0921-7126 / $ © 1998, IOS Press

$r(a, f(a))$, as well as all atoms $q(X, Y)$ where $X \neq Y$. Thus, a *non-ground* representation of this stable model contains the following three *constrained atoms*:

$$p(X, Y) \leftarrow X = Y,$$
$$q(X, Y) \leftarrow X \neq Y,$$
$$r(X, Y) \leftarrow X = a \;\&\; Y = f(a).$$

In contrast, were we to attempt to explicitly represent the stable model of this program using ground atoms, then an infinite set is required. Indeed, in order to even perform the Gelfond-Lifschitz transform, it is necessary to first "ground" out the above logic program, resulting in a propositional program containing infinitely many clauses. □

This example demonstrates, informally, that *constraints* may be used as the basis for compact representations of stable models. Such a non-ground representation of the semantics of logic programs was due to Giorgio Levi and his group's work on S-semantics at Pisa – in particular, Turi [34] used it to reason about Clark completions of logic programs. Following Turi's lead, in this paper, we develop a formal theory of stable and well-founded semantics based on non-ground computation and representation.

Our basic approach is as follows. A constrained interpretation $\mathcal{CI}$ will be a set of constrained atoms – atoms of the form $p(\mathbf{X}) \leftarrow \mathcal{E}$ where $\mathbf{X}$ is a vector of variables and $\mathcal{E}$ is a first-order constraint expression built from equality atoms. Given a normal logic program P and a constrained interpretation $\mathcal{CI}$, we will *translate* P into a *negation-free constraint logic program*, $\mathbf{CT}(P, \mathcal{CI})$, where the constraints are over the domain of terms. Then $\mathcal{CI}$ is a (non-ground) stable model of P just in the case that it is "equivalent" (in a sense made precise later in the paper) to the least constrained model (in the sense of Turi [34]) of $\mathbf{CT}(P, \mathcal{CI})$. We illustrate how the same technique can be used to define a non-ground representation of the well-founded semantics.

The rest of the paper is organized as follows. The next section gives some preliminaries on constrained interpretations, constraint logic programs, and the well-founded and stable semantics of normal logic programs. In Section 3, we present the constraint transformation and analyze its properties. After that, we discuss some complexity issues that come up in this context. In Section 4, we present the concepts of constrained non-ground stable and well-founded semantics, and then show that both of these "non-ground" representations faithfully capture the stable and well-founded model semantics of logic programs, respectively. Moreover, we discuss the complexity of reasoning from such non-ground representations. Section 5 tackles the problem of computing constrained non-ground stable models. It presents an algorithm which builds on a procedure for computing the constrained non-ground well-founded semantics. This algorithm is effective if the program is function-free; note that in presence of function symbols, existence of a stable model is known to be undecidable [24]. Section 6 discusses the effects of our approach to implementation. The final Section 7 reviews related work and concludes the paper.

2. Preliminaries and Previous Results

We assume familiarity with the basic concepts of logic programming and the standard notions of terms, atoms etc; for a background, see Lloyd [22].

In the following, we assume that $\mathcal{L}$ is an arbitrary, but fixed language[1] with equality (=) generated by a finite signature Σ of constant symbols $a, b, c, \ldots$, function symbols $f, g, \ldots$, and predicate symbols $p, q, \ldots$, as well as an infinite set *Var* of variable symbols $X, Y, Z, \ldots$. We assume that Σ contains at least one constant. We shall not mention $\mathcal{L}$ explicitly when it is clear from context. A bold face version of a symbol (e.g., $\mathbf{X}$) denotes a list of respective symbols, whose length is clear from the context.

As common in logic programming, we focus on Herbrand models of $\mathcal{L}$. Throughout the rest of this paper, all concepts (logical consequence, satisfiability etc) are based on Herbrand interpretations (for short, simply interpretations). We denote by $\mathcal{H}$ the Herbrand pre-interpretation of $\mathcal{L}$, i.e., the first-order structure where the universe is the Herbrand universe of $\mathcal{L}$, the predicate $=$ is interpreted as identity, the constants c are interpreted by themselves ($c^{\mathcal{H}} = c$), and every function f is interpreted by the function $f^{\mathcal{H}}$ defined by $f^{\mathcal{H}}(t_1, \ldots, t_n) = f(t_1^{\mathcal{H}}, \ldots, t_n^{\mathcal{H}})$.

[1] In general, $\mathcal{L}$ is assumed fixed. However, when discussing complexity results, it is often useful to allow $\mathcal{L}$ to vary, with a program P and a query defining the non-logical (i.e. constant, function and predicate) symbols of the language.

2.1. Well-Founded and Stable Semantics

A *normal logic program* (LP) is a set of clauses

$$p(\mathbf{t}) \leftarrow B_1 \& \ldots \& B_n \& \mathbf{not}(D_1) \& \ldots \& \mathbf{not}(D_m)$$

where $B_1, \ldots, B_n$, and $D_1, \ldots, D_m$ are atoms.[2]

Given a normal program P and an (Herbrand) interpretation I, the Gelfond-Lifschitz transformation, $\mathsf{GL}(P, I)$, of P with respect to I is the set of clauses

$$\begin{aligned}\mathsf{GL}(P, I) = \{A \leftarrow B_1 \& \ldots \& B_n \mid &\text{ the clause} \\ A \leftarrow B_1 \& \ldots \& B_n \& \mathbf{not}(D_1) \& \ldots \& \mathbf{not}(D_m) &\\ \text{is a ground instance of a clause in } P &\text{ such that} \\ \{D_1, \ldots, D_m\} \cap I = \emptyset\ \}.&\end{aligned}$$

Hence $\mathsf{GL}(P, I)$ is a logic program free of the non-monotonic negation **not**. Based on the Gelfond-Lifschitz transformation, we associate with a normal logic program P an operator F_P, mapping interpretations to interpretations, as follows:

$$F_P(I) = \mathsf{lfp}(T_{\mathsf{GL}(P,I)}),$$

where the T-operator is the usual immediate consequence operator associated with positive logic programs (cf. [22]). Then, I is a *stable model* of P [14] if and only if $F_P(I) = I$.

Baral and Subrahmanian [2] proved that F_P is an anti-monotone operator, i.e, $I_1 \subseteq I_2$ implies $F_P(I_2) \subseteq F_P(I_1)$, and that the *well-founded semantics* (WFS) of P [36] is captured by the least fix-point and the greatest fix-point of F_P^2 in the following way.

Proposition 1 ([2]) *Let P be a normal logic program and let A be a ground atom. Then,*

- *A is true in the WFS of P iff $A \in \mathsf{lfp}(F_P^2)$.*
- *A is false in the WFS of P iff $A \notin \mathsf{gfp}(F_P^2)$.* □

We will revisit these properties later on. Equipped with these preliminaries, we proceed in Section 3 to define a constraint-based version of the Gelfond-Lifschitz transform that works with normal constraint logic programs, which are introduced in the next subsection.

[2] We allow here programs to be infinite for technical reasons, since we consider (partial) groundings of such programs. The focus of our attention is on finite programs, however.

2.2. Preliminaries on Turi's work

The S-semantics is a family of "non-ground" semantics developed by the Pisa group [11]. This section outlines the essential ideas behind the non-ground semantics of Turi [34], which have been further developed by Gabbrielli and Levi [13].

2.2.1. constrained interpretations

Definition 1 An equational *constraint* (in $\mathcal{L}$) is any well-formed formula $\mathcal{E}$ built from atoms $t_1 = t_2$, where t_1 and t_2 are terms, using the logical connectives $\neg$, $\vee$, $\&$ and quantifiers $\forall$ and $\exists$. □

Free and bound occurrences of variables in a constraint $\mathcal{E}$ are defined as usual. We write $\mathcal{E}(\mathbf{X})$ to indicate that all free variables of $\mathcal{E}$ are among the variables in $\mathbf{X}$.

For notational convenience, we write $t_1 \neq t_2$ for $\neg(t_1 = t_2)$, and denote by $false$ resp. $true$ some propositional contradiction resp. tautology.

Definition 2 A *solution* for a constraint $\mathcal{E}(\mathbf{X})$ is a ground substitution σ to the variables in $\mathbf{X}$, such that $\mathcal{H} \models \mathcal{E}(\mathbf{X})\sigma$, i.e., $\mathcal{E}(\mathbf{X})\sigma$ is true in all Herbrand interpretations of $\mathcal{L}$. A constraint is *solvable*, if it has some solution σ. □

Example 2 Consider the constraint

$$\mathcal{E} : \forall X.(((X \neq f(Y)) \& (Y \neq a)) \vee \exists Z(f(Z) = X).$$

This constraint is satisfiable in $\mathcal{H}$ for $\sigma = \{Y/f(a)\}$. Therefore, σ is a solution for $\mathcal{E}$, and $\mathcal{E}$ is solvable. □

Definition 3 Let p be an n-ary predicate symbol, $\mathbf{X} = X_1, \ldots, X_n$ be an n-tuple of variables, and $\mathcal{E}$ be a constraint. Then $p(\mathbf{X}) \leftarrow \mathcal{E}$ is called a *constrained atom*, where $\mathcal{E}$ is the *constraint part*. We define $[p(\mathbf{X}) \leftarrow \mathcal{E}]$ as the following set of ground atoms:

$$\begin{aligned}[p(\mathbf{X}) \leftarrow \mathcal{E}] = \{p(\mathbf{X})\sigma \mid \mathcal{H} \models \mathcal{E}\sigma, \sigma \text{ is a solution} \\ \text{of } \mathcal{E}, \text{ and } (p(\mathbf{X}) \leftarrow \mathcal{E})\sigma \text{ is ground,}\}.\end{aligned}$$ □

Example 3 For a binary predicate p, the expression $p(X, Y) \leftarrow X \neq Y$ is a constrained atom A; assuming that $\mathcal{L}$ contains three constant symbols a, b, c and no function symbols, $[A] = \{\ p(a,b),\ p(a,c),\ p(b,c),\ p(b,a),\ p(c,a),\ p(c,b)\ \}$. □

For convenience, we omit a constraint part $\mathcal{E}$ if it is true on $\mathcal{H}$. E.g., we write $q(X) \leftarrow$ for $q(X) \leftarrow X = X$.

Definition 4 A *constrained interpretation* (*c-interpretation*) is a set $\mathcal{CI}$ of constrained atoms. For a given c-interpretation $\mathcal{CI}$, we let

$$[\mathcal{CI}] = \bigcup_{p(\mathbf{X}) \leftarrow \mathcal{E} \in \mathcal{CI}} [p(\mathbf{X}) \leftarrow \mathcal{E}],$$

which is the Herbrand interpretation naturally associated with $\mathcal{CI}$.

A constrained atom $p(\mathbf{X}) \leftarrow \mathcal{E}$ is true in a c-interpretation $\mathcal{CI}$ if $[p(\mathbf{X}) \leftarrow \mathcal{E}] \subseteq [\mathcal{CI}]$. □

Example 4 Let $\mathcal{CI}$ be the c-interpretation that contains the single constrained atom $p(X,Y) \leftarrow$. The constraint part of $p(X,Y) \leftarrow$ is empty; hence, any substitution is a solution. Therefore, for every language $\mathcal{L}$, $p(X,Y) \leftarrow X \neq Y$ is true in $\mathcal{CI}$, since $[p(X,Y) \leftarrow X \neq Y] \subseteq [\{p(X,Y) \leftarrow\}]$. □

C-interpretations are ordered by a relation $\leq$ as follows.

Definition 5 For any c-interpretations $\mathcal{CI}_1$ and $\mathcal{CI}_2$ define $\mathcal{CI}_1 \leq \mathcal{CI}_2$ if and only if $[\mathcal{CI}_1] \subseteq [\mathcal{CI}_2]$. □

Note that $\leq$ is reflexive and transitive, but not necessarily anti-symmetric. It induces an equivalence relation $\sim$, where $\mathcal{CI}_1 \sim \mathcal{CI}_2$ iff $\mathcal{CI}_1 \leq \mathcal{CI}_2$ and $\mathcal{CI}_2 \leq \mathcal{CI}_1$. It is easily verified that $\mathcal{CI}_1 \sim \mathcal{CI}_2$ iff $[\mathcal{CI}_1] = [\mathcal{CI}_2]$.

Equivalent c-interpretations are treated semantically indiscernible. Therefore, we implicitly assume that constraints are standardized apart whenever needed.

Every finite c-interpretation is equivalent to a c-interpretation in which in all constraints are standardized apart and where in addition each predicate p occurs in exactly one constrained atom; we call such c-interpretations *normal*. Indeed, two constrained atoms A_1, A_2 with the same predicate p in the head can be replaced by an equivalent single constrained atom. Therefore, every finite c-interpretation can be easily transformed into normal form; thus, we often assume that finite c-interpretations are in normal form. Moreover, we sometimes refer in the rest of this paper to a c-interpretation $\mathcal{CI}$ where, strictly speaking, the equivalence class $[\mathcal{CI}]_\sim$ of $\mathcal{CI}$ with respect to $\sim$ is meant.

2.2.2. constraint logic programs

A *normal constraint logic program* (CLP) P is a set of clauses of the form

$$A \leftarrow \mathcal{E} \,|\, B_1 \& \cdots \& B_n \,\&\, \mathbf{not}(D_1) \,\&\, \cdots \,\&\, \mathbf{not}(D_m)$$

where A and all B_i, D_j are atoms, and $A \leftarrow \mathcal{E}$ is a constrained atom. If P is free of the operator **not**, then P is *positive*. Typically, we deal with finite CLPs.

There is a natural correspondence between normal standard LPs and normal CLPs. Every clause of a normal LP can be rewritten into an equivalent constrained clause of the form

$$p(\mathbf{X}) \leftarrow \mathbf{X} = \mathbf{t} \mid B_1 \& \ldots \& B_n \,\&\, \mathbf{not}(D_1) \,\&\, \ldots \,\&\, \mathbf{not}(D_m)$$

where $\mathbf{X}$ are fresh variables; therefore, in abuse of notation, we sometimes view an normal LP P as the normal CLP obtained by rewriting all rules.

Vice versa, with each normal CLP P, we associate a normal LP P^* which contains for each clause as above from P all clauses $(A \leftarrow B_1 \& \cdots \& B_n)\sigma$ where σ is a solution of $\mathcal{E}$.

Thus, the CLP P can be views as a compact representation of the normal LP P^*. In this line, we say that a ground atom A is a logical consequence of P, if and only if A is a logical consequence of P^*.

Given any positive constraint logic program P, we associate with P an operator W_P (usually also denoted S_P) that maps c-interpretations to c-interpretations. In order to define this operator, we first define the notion of a *resolvent* of a clause w.r.t. a c-interpretation.

Suppose

$$C = p(\mathbf{X}) \leftarrow \mathcal{E}_0 \,|\, p_1(\mathbf{t}_1) \& \cdots \& p_n(\mathbf{t}_n)$$

is a clause and $\mathcal{CI}$ is a c-interpretation. Without loss of generality, we assume that all clauses in $\mathcal{CI} \cup \{C\}$ are (mutually) standardized apart. Then the *resolvents of* C with respect to $\mathcal{CI}$, denoted $res_P(C, \mathcal{CI})$, is the set of all constraint atoms

$$p(\mathbf{X}) \leftarrow \mathcal{E}_0 \,\&\, (\mathbf{X}_1 = \mathbf{t}_1) \& \mathcal{E}_1 \,\&\, \cdots \,\&\, (\mathbf{X}_n = \mathbf{t}_n) \& \mathcal{E}_n$$

where $p_i(\mathbf{X}_i) \leftarrow \mathcal{E}_i$ is in $\mathcal{CI}$ for each $1 \leq i \leq n$; if some p_i does not occur in $\mathcal{CI}$, then $res_P(C, \mathcal{CI})$ contains the single constraint atom $p(\mathbf{X}) \leftarrow \mathit{false}$ where false is any unsatisfiable constraint.

Now W_P is defined as follows.

Definition 6 For any positive constraint logic program P and c-interpretation $\mathcal{CI}$, let

$$W_P(\mathcal{CI}) \;=\; \bigcup_{C \in P} res_P(C, \mathcal{CI}).$$ □

If $\mathcal{CI}$ and P are finite, then $W_P(\mathcal{CI})$ is finite as well and can be normalized.

It is easy to see that W_P is monotone with respect to the ordering $\leq$; hence, it has a least fixpoint, which we denote by $\mathsf{lfp}(W_P)$.

Proposition 2 *Let P be a positive constraint logic program. Then,*

1. *if $\mathcal{CI}_1 \leq \mathcal{CI}_2$ then $W_P(\mathcal{CI}_1) \leq W_P(\mathcal{CI}_2)$.*
2. *if $\mathcal{CI}_1 \sim \mathcal{CI}_2$, then $W_P(\mathcal{CI}_1) \sim W_P(\mathcal{CI}_2)$.*
3. *W_P has a least fixpoint with respect to $\leq$, denoted by $\mathsf{lfp}(W_P)$.*
4. *a ground atom A is a logical consequence of P if and only if $A \in [\mathsf{lfp}(W_P)]$.* □

Proof. The proof of 1. is straightforward, and 2. is immediate by 1. part Part 3. is a simple consequence of 1. and the well-known Knaster-Tarski Theorem that every monotone operator on a complete lattice has a least fixpoint. Part 4. follows from the fact that the i-th stage of W_P, W_P^i, is equivalent with the i-th stage $T_{P^*}^i$ of the immediate consequence operator T_{P^*} (cf. [22]), i.e., $[W_P^i] = T_{P^*}^i$, which is shown by an easy induction. □

Example 5 Consider the following program that represents the classical transitive closure computation on a simple graph:

$$\begin{aligned} P = \{\ & e(X,Y) \leftarrow Y = a, \\ & t(X,Y) \leftarrow e(X,Y), \\ & t(X,Y) \leftarrow t(X,Z)\,\&\,t(Z,Y)\ \}. \end{aligned}$$

The iterative applications of the W_P operator are shown below (here, the result of W_P is normalized):

$$\begin{aligned} W_P^1(\emptyset) = \{ & e(X,Y) \leftarrow Y = a\} \\ W_P^2(\emptyset) = \{ & e(X,Y) \leftarrow Y = a, \\ & t(X,Y) \leftarrow (X = X_1)\&(Y = Y_1)\&(Y_1 = a)\} \\ W_P^3(\emptyset) = \{ & e(X,Y) \leftarrow Y = a, \\ & t(X,Y) \leftarrow ((X = X_1)\&(Y = Y_1)\&(Y_1 = a)) \\ & \qquad \vee \\ & \qquad ((X = X_2)\&(Z = Y_2)\&(Y_2 = a)\& \\ & \qquad (Z = X_3)\&(Y = Y_3)\&(Y_3 = a))\ \} \\ \cdots & \end{aligned}$$

It is easy to verify that $[W_P^3(\emptyset)]$ is equivalent to $[W_P^2(\emptyset)]$. Thus, $W_P^2(\emptyset)$ describes $\mathsf{lfp}(W_P)$. By pushing through equalities and straightforward simplification, we obtain that

$$\mathsf{lfp}(W_P) \sim \{e(X,Y) \leftarrow Y = a,\ t(X,Y) \leftarrow Y = a\}.$$

□

An important issue that comes along with the use of constraints, as shown in the previous example, is simplification of constraint expressions. A number of algebraic simplification rules for constraints in the equality theory (i.e., term algebra) we use here are applicable, e.g., substitution of equals for equals, pushing negations through conjunction and disjunctions, and, in case of a finite Herbrand universe, constructive negation. More generally, any set of sound algebraic simplification rules may be used. By using such methods, constraint parts in $W_P(\mathcal{CI})$ as well as in arbitrary c-interpretations can be simplified such that they occupy less space and are easier to read.

It is clear that sophisticated and highly compressive simplification of constraints requires some computational effort, and that with respect to our as well as other applications, the trade-off between the gain in compression and the computation time and space spent for simplification has to be carefully deliberated. The issue of proper constraint simplification is per se a highly relevant problem, and a thorough discussion would lead for beyond the scope of this paper. However, we note that the elementary rules mentioned above (substitution of equals for equals, pushing trough negation etc –which are computationally rather cheap) may help to reduce the size of the W_P^i's substantially, and sometimes, even result in an exponential saving. To see this, consider the example below.

Example 6 Consider the program P:

$$\begin{aligned} & p_1(X,Y) \leftarrow (X = c_1)\&(Y = c_2),\ \ c_1, c_2 \in \{a, b\}, \\ & p_i(X,Y) \leftarrow p_{i-1}(X,Z)\&p_{i-1}(Z,Y),\ \ 2 \leq i \leq n. \end{aligned}$$

Here we have that $W_P^{n+1}(\emptyset) = W_P^n(\emptyset)$, and that the size of $W_P^{n+1}(\emptyset)$ is exponential in n. Note that

$$\begin{aligned} \mathsf{lfp}(W_P) \sim \{\, & p_i(X,Y) \leftarrow (X = c_1)\&(Y = c_2)\ \mid \\ & c_1, c_2 \in \{a, b\},\ 1 \leq i \leq n\,\}, \end{aligned}$$

which has only linear size with respect to P. This representative can be obtained by applying the mentioned simplifications to W_P. However, as discussed later, not always a small (i.e., polynomial size) representative for $\mathsf{lfp}(W_P)$ can be found. □

3. The Constraint-based Transformation

In this section, we present a non-ground version of the Gelfond-Lifschitz transformation which we use for the definition of constrained non-ground stable models and well-founded semantics. The new transformation, $\mathbf{CT}$, takes as input a normal logic program P, a c-interpretation $\mathcal{CI}$, and returns a *constraint* logic program, $\mathbf{CT}(P, \mathcal{CI})$. The postulates for this program are the following properties.

(**Desideratum 1**) If $C = A \leftarrow \mathcal{E}(\mid B_1 \,\&\, \ldots \,\&\, B_n$ is a clause in $\mathbf{CT}(P, \mathcal{CI})$ and σ is a solution of $\mathcal{E}$, then all ground instances of $(A \leftarrow B_1 \,\&\, \ldots \,\&\, B_n)\sigma$ are contained in $\mathsf{GL}(P, [\mathcal{CI}])$.

(**Desideratum 2**) If C is a clause in $\mathsf{GL}(P, [\mathcal{CI}])$, then there is a clause $C' \in \mathbf{CT}(P, \mathcal{CI})$ of the form $A \leftarrow \mathcal{E} \mid B_1 \,\&\, \ldots \,\&\, B_n$ such that for some solution σ of $\mathcal{E}$, C is a ground instance of $(A \leftarrow B_1 \,\&\, \ldots \,\&\, B_n)\sigma$.

Thus in effect, $\mathbf{CT}(P, \mathcal{CI})$ is a constraint non-ground version of the Gelfond-Lifschitz transformation; the grounding of the associated logic program $\mathbf{CT}(P, \mathcal{CI})^*$ coincides with $\mathsf{GL}(P[\mathcal{CI}])$. $\mathbf{CT}(P, \mathcal{CI})$ can be seen as partial deduction (unfolding).

Towards a definition of $\mathbf{CT}$, we define the elimination of negated atoms from clauses with respect to a c-interpretation.

Let C be the clause

$$A \leftarrow \mathcal{E} \mid B_1 \,\&\, \cdots \,\&\, B_n \,\&\, \mathbf{not}(D_1) \,\&\, \ldots \,\&\, \mathbf{not}(D_m)$$

where D_1 has the form $p_1(\mathbf{t}_1)$. The *elimination* of $\mathbf{not}(D_1)$ from C with respect to a normal c-interpretation $\mathcal{CI}$ is the set of clauses

$$A \leftarrow \mathcal{E} \,\&\, (\mathbf{X}_1 = \mathbf{t}_1) \,\&\, \forall \mathbf{Y}_1 \neg \mathcal{E}_1(\mathbf{X}_1 \mathbf{Y}_1) \;\mid\; B_1 \,\&\, \ldots \,\&\, B_n \,\&\, \mathbf{not}(D_2) \,\&\, \ldots \,\&\, \mathbf{not}(D_m).$$

where $p_1(\mathbf{X}_1) \leftarrow \mathcal{E}_1(\mathbf{X}_1 \mathbf{Y}_1)$ is the unique constrained atom in $\mathcal{CI}$ with p_1 in the head; note that the implicit existential quantifiers on $\mathbf{Y}_1$ in $\mathcal{E}_1$ are turned into universal quantifiers.

The *elimination of all negated literals from C with respect to $\mathcal{CI}$*, denoted $\mathsf{ELIM}(C, \mathcal{CI})$, is obtained by iteratively eliminating $\mathbf{not}(D_1), \ldots, \mathbf{not}(D_m)$ one after another.

Definition 7 Let P be a normal logic program and $\mathcal{CI}$ be a normal c-interpretation. The *constraint-based transformation* of P with respect to the normal c-interpretation $\mathcal{CI}$, $\mathbf{CT}(P, \mathcal{CI})$, is the collection of clauses $\mathbf{CT}(P, \mathcal{CI}) = \{\mathsf{ELIM}(C, \mathcal{CI}) \mid C \in P\}$. □

In other words, we apply the negation elimination process to each clause in P, and place the resulting clauses in $\mathbf{CT}(P, \mathcal{CI})$. Thus, $\mathbf{CT}(P, \mathcal{CI})$ is a negation-free *constraint logic program*.

In case of arbitrary finite c-interpretations $\mathcal{CI}$, we first normalize $\mathcal{CI}$ to a c-interpretation $\mathcal{CI}^*$ and then apply the transformation $\mathbf{CT}(P, \mathcal{CI}^*)$. Note that this does not yield a unique constraint program as normalization is not unique; however, all resulting constraint programs are equivalent.

Example 7 Consider $\mathcal{CI} = \{p(X_1, Y_1) \leftarrow X_1 \neq Y_1\}$ and suppose P is the clause $q(X) \leftarrow \mathbf{not}(p(X, Y))$. Then the elimination of $\mathbf{not}(p(X, Y))$ from this clause yields

$$q(X) \leftarrow (X_1 = X) \& (Y_1 = Y) \& \neg(X_1 \neq Y_1),$$

which simplifies to

$$q(X) \leftarrow X = Y. \tag{1}$$

Let us check that this rule as $\mathbf{CT}(P, \mathcal{CI})$ accurately captures the desiderata described earlier. The predicate p consists of all pairs (t_1, t_2) of elements in the Herbrand universe where $t_1 \neq t_2$; thus, q consists of the entire Herbrand universe, i.e., $\mathsf{GL}(P, [\mathcal{CI}])$ consists of all rules $q(t) \leftarrow$ where t is a ground term. This is precisely captured by Rule 1, and the desiderata are thus clearly satisfied. □

Example 8 Consider the program P consisting of the following clause C:

$$p(X_1, Y_1) \leftarrow \mathbf{not}(q(X_1, Z_1)) \,\&\, \mathbf{not}(r(Z_1))$$

and the normal c-interpretation

$$\mathcal{CI} = \{p(X_0, Y_0) \leftarrow\ , \quad r(X_3) \leftarrow X_3 = b \,\&\, X_3 \neq Y_3, \quad q(X_2, Y_2) \leftarrow (X_2 \neq Y_2) \vee (X_2 = a)\}.$$

The elimination of $\mathbf{not}(q(X_1, Y_1))$ from C with respect to $\mathcal{CI}$ yields the constrained clause

$$p(X_1, Y_1) \leftarrow (X_1 = X_2) \,\&\, (Y_1 = Y_2) \,\&\, \neg((X_2 \neq Y_2) \vee (X_2 = a)) \mid \mathbf{not}(r(Y_1)).$$

Subsequent elimination of $\mathbf{not}(r(Y_1))$ leads to the $\mathbf{not}$-free constrained clause $\mathsf{ELIM}(C, \mathcal{CI})$

$$p(X_1, Y_1) \leftarrow (X_1 = X_2) \,\&\, (Z_1 = Y_2) \,\&\, \neg((X_2 \neq Y_2) \vee (X_2 = a)) \,\&\, (Z_1 = X_3) \,\&\, \forall Y_3 \neg((X_3 = b) \,\&\, X_3 = Y_3).$$

Pushing through negation and the resulting equalities $X_1 = X_2 = Y_2 = Z_1 = X_3$, we get

$$p(X_1, Y_1) \leftarrow (X_1 = Y_1) \,\&\, (X_1 \neq a) \,\&\, \forall Y_3((X_1 \neq b) \vee (X_1 = Y_3)),$$

which simplifies to

$$p(X_1, Y_1) \leftarrow (X_1 = Y_1) \,\&\, (X_1 \neq a) \,\&\, (X_1 \neq b). \quad (*)$$

The pairs that are not in q are all (t,t) where $t \neq a$; furthermore, r does not contain the ground terms different from b, and b only if it is not the single term in the Herbrand universe (which is true). Thus, $\mathsf{GL}(P,[\mathcal{CI}])$ consists of all clauses $p(t,t) \leftarrow$, where t is any ground term different from a and b. However, this are precisely the clauses $(p(X_1,Y_1) \leftarrow)\sigma$ where σ is a solution for the constraint part of rule (*). As $\mathbf{CT}(P,\mathcal{CI})$ amounts to rule (*), the desiderata are clearly satisfied. □

In general, arbitrary c-interpretations can not be rewritten as normal c-interpretations. This may be respected by generalizing the elimination of a negative goal $\mathbf{not}(D_1)$ from the body of clause C with respect to $\mathcal{CI}$ as follows. If the predicate p_1 occurs in finitely many constrained atoms of $\mathcal{CI}$, then replace them by an equivalent single constraint atom $p_1(\mathbf{X}_1) \leftarrow \mathcal{E}_1$ as obvious and eliminate $\mathbf{not}(D_1)$ as usual. In case that infinitely many constrained atoms $p_1(\mathbf{X}_i) \leftarrow \mathcal{E}_i$, $i = 1,\ldots,i,\ldots$ occur in $\mathcal{CI}$, the elimination of $\mathbf{not}(D_1)$ yields all clauses

$$A \leftarrow \mathcal{E} \,\&\, (\mathbf{X}_1 = \mathbf{t}_1) \,\&\, (\mathbf{X}_1 = \mathbf{t}_1^*) \;\mid\; B_1 \& \ldots \& B_n \,\&\, \mathbf{not}(D_2) \,\&\, \ldots \,\&\, \mathbf{not}(D_m)$$

where $\mathbf{t}_1^*$ is variable-free and the set of formulas

$$\{\neg p_1(\mathbf{t}_1^*), \forall \mathbf{X}_1\mathbf{Y}_1(\mathcal{E}_1(\mathbf{X}_1\mathbf{Y}_1) \rightarrow p_1(\mathbf{X}_1)), \ldots, \forall \mathbf{X}_i\mathbf{Y}_i(\mathcal{E}_i(\mathbf{X}_i\mathbf{Y}_i) \rightarrow p_1(\mathbf{X}_i)), \ldots \} \quad (**)$$

is satisfiable in $\mathcal{H}$. The elimination $\mathsf{ELIM}(C,\mathcal{CI})$ of all negative literals is then an infinite collection of rules; thus, the program $\mathbf{CT}(P,\mathcal{CI})$ may contain infinitely many rules if $\mathcal{CI}$ is an arbitrary c-interpretation.

In the remainder of this paper, we will focus for applications on finite c-interpretations, even if definitions and results are formulated for general c-interpretations. Our main goal of computing the non-ground well-founded/stable semantics, is only feasible in the finite case anyway. In particular, we can assume that all c-interpretations are finite if the language has no function symbols, i.e., in case of function-free programs. As discussed earlier, compact non-ground representations in his case are still highly relevant.

The next theorem states that $\mathbf{CT}(P,\mathcal{CI})$ meets the two desiderata required of the transformed program.

Theorem 1 *Let P be a normal LP and $\mathcal{CI}$ be a c-interpretation. Then, $\mathbf{CT}(P,\mathcal{CI})$ satisfies the desiderata 1 and 2, i.e., the following hold:*

1. *If $C = A \leftarrow \mathcal{E} \mid B_1 \& \ldots \& B_n$ is a clause in $\mathbf{CT}(P,\mathcal{CI})$ and σ is a solution of $\mathcal{E}$, then all ground instances of $(A \leftarrow B_1 \& \ldots \& B_n)\sigma$ are contained in $\mathsf{GL}(P,[\mathcal{CI}])$.*
2. *If C is a clause in $\mathsf{GL}(P,[\mathcal{CI}])$, then there is a clause $C' \in \mathbf{CT}(P,\mathcal{CI})$ of the form $A \leftarrow \mathcal{E} \mid B_1 \& \ldots \& B_n$ such that for some solution σ of $\mathcal{E}$, C is a ground instance of $(A \leftarrow B_1 \,\&\, \cdots \,\&\, B_n)\sigma$.* □

Proof. For part 1, let $C = A \leftarrow \mathcal{E} \mid B_1 \& \ldots \& B_n$ be a clause from $\mathbf{CT}(P,\mathcal{CI})$ and let σ be a solution of $\mathcal{E}$. Then, there exists a clause C' of form

$$A \leftarrow B_1 \,\&\, \cdots \,\&\, B_n \,\&\mathbf{not}(D_1) \,\&\, \cdots \,\&\, \mathbf{not}(D_m)$$

in P such that $C = \mathsf{ELIM}(C',\mathcal{CI})$ (resp., C is among the clauses $\mathsf{ELIM}(C',\mathcal{CI})$). Consider the definition of $\mathcal{E}$ and let $\mathcal{E}_i' = ((\mathbf{X}_i = \mathbf{t}_i)\&\forall \mathbf{Y}_i \neg \mathcal{E}_i(\mathbf{X}_i\mathbf{Y}_i)\sigma$ (resp., $\mathcal{E}_i' = ((\mathbf{X}_i = \mathbf{t}_i)\&(\mathbf{X}_i = \mathbf{t}_i^*))\sigma$). Since $\mathcal{H} \models \mathcal{E}\sigma$, clearly $\mathcal{H} \models F_i$. Hence, $\mathcal{H} \models \forall \mathbf{Y}_i \neg \mathcal{E}_i(\mathbf{X}_i\mathbf{Y}_i)\sigma$ (resp., $\mathcal{H} \models (\mathbf{X}_i = \mathbf{t}_i^*)\sigma$). Consequently, the constraint $\mathcal{E}_i(\mathbf{X}_i\mathbf{Y}_i)\sigma$ has no solution, and therefore the ground atom $p_i(\mathbf{t}_i)\sigma$ (which is an instance of B_i) is false in $\mathcal{CI}$ (resp., the ground atom $p_i(\mathbf{t}_i)\sigma = p_i(\mathbf{t}_i^*)$ is false in $\mathcal{CI}$ by (**)). Hence, every ground clause $(A \leftarrow B_1 \,\&\, \cdots \,\&\, B_n \,\&\, \mathbf{not}(D_1) \,\&\, \cdots \,\&\, \mathbf{not}(D_m))\tau$, where τ is any ground substitution that extends σ, has all its negative literals false in $[\mathcal{CI}]$. Therefore, the clause $(A \leftarrow B_1 \,\&\, \cdots \,\&\, B_n)\tau$ is in $\mathsf{GL}(P,[\mathcal{CI}])$. Consequently, every ground instance of the clause $(A \leftarrow B_1 \,\&\, \cdots \,\&\, B_n)\sigma$ is in $\mathsf{GL}(P,[\mathcal{CI}])$.

For part 2, let C be a clause in $\mathsf{GL}(P,\mathcal{CI})$. Then there is a clause C_0 in P of the form $A \leftarrow B_1 \,\&\, \cdots \,\&\, B_n \,\&\, \mathbf{not}(D_1) \,\&\, \cdots \,\&\, \mathbf{not}(D_m)$ and a ground substitution τ for the variable in C_0 such that $D_i\tau \notin [\mathcal{CI}]$, $i = 1,\ldots,m$ and $C = (A \leftarrow B_1 \,\&\, \cdots \,\&\, B_n)\tau$. Consider the clause $C' = A \leftarrow \mathcal{E} \mid B_1 \& \ldots \& B_n$ from the collection $\mathsf{ELIM}(C_0,\mathcal{CI})$, where the constraint $\mathcal{E}$ has form

$$(\mathbf{X} = \mathbf{t})\&\mathcal{E}_1' \& \cdots \& \mathcal{E}_m' \qquad (2)$$

and, by the elimination of $\mathbf{not}(D_i)$, either $\mathcal{E}_i' = \forall \mathbf{Y}_i \neg \mathcal{E}_i(\mathbf{X}_i\mathbf{Y}_i)$, or $\mathcal{E}_i' = (\mathbf{X}_i = t_i^*)$ where $p_i(\mathbf{t}_i^*) = D_i\tau$. Since $D_i\tau \notin [\mathcal{CI}]$, we have $\mathcal{H} \models (\mathbf{X}_i = \mathbf{t}_i\tau) \rightarrow \forall \mathbf{Y}_i \neg \mathcal{E}_i(\mathbf{X}_i\mathbf{Y}_i)$ or $\mathcal{H} \models (\mathbf{X}_i = \mathbf{t}_i\tau) \rightarrow (\mathbf{X}_i = \mathbf{t}_i^*)$, respectively. By the form of $\mathcal{E}$, the constraint $\mathcal{E}\tau$ is clearly solvable; note that the free variables in $\mathcal{E}\tau$ are $\mathbf{X}\mathbf{X}_1 \cdots \mathbf{X}_n$. Let θ be any solution of $\mathcal{E}\tau$, and let σ be the restriction of $\tau \cup \theta$ to the free variables in $\mathcal{E}$. Then, σ is a solution of $\mathcal{E}$, and clearly C is a ground instance of the clause $(A \leftarrow B_1 \,\&\, \cdots \,\&\, B_n)\sigma$ (apply $(\tau \cup \theta) \setminus \sigma$). This proves part 2. □

As $\mathbf{CT}(P,\mathcal{CI})$ is negation free, it follows that the operator W_P is applicable; in particular, $W_{\mathbf{CT}(P,\mathcal{CI})}$ has a least fixpoint w.r.t. $\leq$. This is captured via the operator FNG_P.

Definition 8 Let P be a normal logic program. The operator FNG_P, which maps c-interpretations to c-interpretations, is defined by

$$FNG_P(\mathcal{CI}) = \mathsf{lfp}(W_{\mathbf{CT}(P,\mathcal{CI})}). \qquad \square$$

As expected, FNG_P possesses all the nice properties of the ground F_P operator.

Lemma 1 *Let P be a normal logic program, and let $\mathcal{CI}$ be any c-interpretation. Then, $[FNG_P(\mathcal{CI})] = F_P([\mathcal{CI}])$.*

Proof. Theorem 1 implies that the ground instantiation of $\mathbf{CT}(P,\mathcal{CI})^*$ coincides with the program $\mathsf{GL}(P,[\mathcal{CI}])$. Let A be any ground atom. Then, A is true in $FNG_P(\mathcal{CI})$, iff A is true in $\mathsf{lfp}(W_{\mathbf{CT}(P,\mathcal{CI})})$, iff A is a logical consequence of $\mathbf{CT}(P,\mathcal{CI})^*$, iff A is a logical consequence of $\mathsf{GL}(P,\mathcal{CI})$, iff $A \in F_P([\mathcal{CI}])$. $\square$

Theorem 2 *Let P be a normal logic program. Then the following holds:*

1. *FNG_P is anti-monotone, i.e. $\mathcal{CI}_1 \leq \mathcal{CI}_2$ implies that $FNG_P(\mathcal{CI}_2) \leq FNG_P(\mathcal{CI}_1)$.*
2. *If $\mathcal{CI}_1 \sim \mathcal{CI}_2$, then $FNG_P(\mathcal{CI}_1) \sim FNG_P(\mathcal{CI}_2)$.*
3. *FNG_P^2 has a least and a greatest fixpoint.* $\square$

Proof. Since F_P is anti-monotone [2], $\mathcal{CI}_1 \leq \mathcal{CI}_2$ implies by Lemma 1 that $FNG_P(\mathcal{CI}_2) \sim F_P([\mathcal{CI}_2]) \leq F_P([\mathcal{CI}_1]) \sim FNG_P(\mathcal{CI}_1)$. This verifies part 1. Part 2 follows immediately from part 1. Since FNG_P is anti-monotone, FNG_P^2 is monotone. By the well-known Knaster-Tarski Theorem, every monotone operator on a complete lattice has a least and greatest fixpoint. $\square$

3.1. A discussion of complexity issues

Before defining constrained non-ground stable models, we make some brief remarks on complexity issues.

Given P and $\mathcal{CI}$, $\mathbf{CT}(P,\mathcal{CI})$ can be computed in polynomial time. This is not possible for $FNG_P(\mathcal{CI})$ in general, since $\mathsf{lfp}(W_P)$ can be exponential even for a simple function-free program P. Example 6 contained an example of such a program. In that particular case, though, it was possible to use constraint simplification in order to obtain a small (polynomial-sized) representative for $\mathsf{lfp}(W_P)$.

However, a small representative for $\mathsf{lfp}(W_P)$ cannot always be constructed efficiently. The reason is that deciding whether a ground atom $p(a)$ is derivable from P, may be reduced to deciding whether $p(a)$ is in $[\mathsf{lfp}(W_P)]$, i.e., in $[\mathcal{CI}]$, which is in PSPACE in general (if the constraint part of the constrained atom $p(X) \leftarrow \mathcal{E}$ in $\mathcal{CI}$ is known to be quantifier-free, then it is even in NP), as shown in the next theorem.

Furthermore, it is well-known in the folklore that the problem of deciding whether a ground atom follows from a datalog program P is complete in EXPTIME (cf. [37]; for a background on complexity classes, consult [16]). Thus, the computation of $\mathsf{lfp}(W_P)$ must take more than polynomial time in general; otherwise, we could solve an EXPTIME-complete problem in PSPACE (resp. in NP), which is strongly hypothesized to be impossible. This remains valid even if a powerful oracle is available that solves PSPACE-complete problems; indeed, an algorithm of this kind renders the problem in the complexity class $\mathrm{P}^{\mathrm{PSPACE}}$, which collapses with PSPACE. Therefore, by the results below, even if deciding truth of constrained atoms in c-interpretations and deciding equivalence of c-interpretations are for free, the computation of $\mathsf{lfp}(W_P)$ must take more than polynomial time in general (under the assumptions of noncollapsing complexity classes).

Consider now the problem of deciding whether a constrained atom is true in a c-interpretation.

It is known that equational reasoning on $\mathcal{H}$ is decidable [5, 23]. As a consequence, it is decidable whether a given constraint atom A (not necessarily ground) is true in a given finite c-interpretation $\mathcal{CI}$. However, the complexity is tremendous; recent results on equational reasoning immediately imply that it is non-elementary [38, 28],[3] and thus among the hardest computable problems. Already in the function-free case, the test is expensive, as follows from the next theorem.

Theorem 3 *Suppose we assume that the language $\mathcal{L}$ is function-free. Given a constrained atom $A = p(\mathbf{X}) \leftarrow \mathcal{E}$ and a finite c-interpretation $\mathcal{CI}$, deciding whether A is true in $\mathcal{CI}$ is*

1. *PSPACE-complete, if the language is the one generated by A and $\mathcal{CI}$, and Π^p_{k+2}-complete if in addition the quantifier depth in constraint parts is bounded by the constant $k \geq 0$;*

[3] The results revise the belief in a previously announced result [18] that this problem is PSPACE-complete.

2. NP-*complete, if A is a* ground atom $p(\mathbf{X}) \leftarrow \mathbf{X} = \mathbf{t}$ *and all constraint parts in $\mathcal{CI}$ are quantifier-free* (*i.e., have implicit existential quantifiers*)*;*
3. *solvable in polynomial time with a small* (*constant*) *number of* NP*-oracle calls and both* NP *and* coNP*-hard, if the language is fixed*[4] *and all constraint parts are existential.* □

Here Π_j^P are the problems solvable in coNP, if an oracle for problems in Σ_{j-1}^P is available, where $\Sigma_1^P = \mathrm{NP}$, $\Sigma_2^P = \mathrm{NP}^{\mathrm{NP}}$, etc (see [16] for definitions and discussion).

Proof. In order to test whether A is true in $\mathcal{CI}$, we have to check whether for every ground substitution σ for the free variables $\mathbf{XY}$ of $\mathcal{E}$ such that $\mathcal{H} \models \mathcal{E}\sigma$, there is a constrained atom $p(\mathbf{X}') \leftarrow \mathcal{E}'$ in $\mathcal{CI}$ such that $\mathcal{H} \models ((\mathbf{X} = \mathbf{X}') \rightarrow \exists \mathbf{Y}' \mathcal{E}')\sigma$;[5] i.e., we have to check

$$\mathcal{H} \models \forall \mathbf{X}\mathbf{Y}\mathbf{X}' \exists \mathbf{Y}'.((\mathbf{X} = \mathbf{X}') \& \mathcal{E} \rightarrow \mathcal{E}'). \qquad (3)$$

It is well-known that evaluation of a given first-order formula on a given finite structure $\mathcal{A}$ is complete in PSPACE (which is known as the combined complexity of first-order logic as a database query language [37]), and PSPACE hard even if the signature contains only equality and the structure (i.e., the domain) is fixed.Therefore, it follows immediately that our problem is complete in PSPACE if the language is the one generated by A and $\mathcal{CI}$.

For the case where the quantifier depth is bounded by the constant $k \geq 0$, we note that the formula in (3) has no free variables and its quantifier depth is bounded by $k + 2$, where the leading quantifier is universal; it is easy to show by induction on the quantifier depth that evaluating such a formula on $\mathcal{H}$ lies in Π_{k+2}^P, which verifies the membership part. Moreover, since evaluating first-order sentences from the first-order prenex class Π_k^0 over equality, $k \geq 1$, on an even fixed finite structure $\mathcal{A}$ is complete for Π_k^P [37], it suffices for the hardness part to observe that for any such sentence ϕ from Π_{k+2}^0, a predicate p and formulas $\mathcal{E}$ and $\mathcal{E}'$ of quantifier depth k can be easily constructed such that (3) amounts to $\mathcal{A} \models \phi$. This proves part 1.

For part 2, we observe that the formula in (3) obviously reduces to the purely existential first-order formula $\mathcal{E}'[\mathbf{X}'/\mathbf{t}]$, for which the test is NP-complete.

To verify part 3, we note that if the language is fixed, the universal quantifiers $\forall \mathbf{X}\mathbf{X}'$ in (3) amount to a constant number of ground substitutions σ, for which, as by the hypothesis $\mathbf{Y}$ and $\mathbf{Y}'$ are void, we have to check $\mathcal{H} \models \neg\mathcal{E}\sigma \vee \mathcal{E}'\sigma$; the latter holds if and only if either $\mathcal{H} \models \neg\mathcal{E}\sigma$ or $\mathcal{H} \models \mathcal{E}'\sigma$. Each such check is possible with an NP oracle; hence, overall a constant number of NP oracle calls suffices. Clearly, for proper $\mathcal{E}$ resp. $\mathcal{E}'$, it follows that the problem is hard for coNP resp. NP. □

In part 2 of the theorem, it is even possible to allow function symbols in the language, without increasing the complexity. This follows from the (as far as we know unpublished) result that the existential fragment of the theory of term algebras is NP-complete. This result and recent algorithms [39] can be exploited to decide the truth a ground atom in a finite c-interpretation $\mathcal{CI}$ in which no quantifiers occur (except for implicit quantifiers) by a nondeterministic algorithm in polynomial time, even if function symbols are present. Thus, compared to the function-free case, there is somewhat surprisingly no increase in complexity. A similar remark applies on other results below.

The complexity of deciding equivalence of two c-interpretations is similar. Since truth of a constrained atom in a c-interpretation is decidable, clearly also the complexity of deciding equivalence of two given finite c-interpretations is decidable. The complexity increase over the former problem is negligible; in the function-free case, the complexity is of the same order.

Theorem 4 *Suppose the language is function-free, and $\mathcal{CI}_1$, $\mathcal{CI}_2$ are two finite c-interpretations. Deciding whether $\mathcal{CI}_1 \sim \mathcal{CI}_2$ is*

1. *PSPACE-complete, if the language is the one generated by A and $\mathcal{CI}$, and Π_{k+2}^p-complete if in addition the quantifier depth in constraint parts is bounded by the constant $k \geq 0$; and*
2. *solvable in polynomial time with a constant number of* NP*-oracle calls and* NP*/coNP-hard, if the language is fixed and all constraint parts are existential.* □

Proof. Clearly, $\mathcal{CI}_1 \sim \mathcal{CI}_2$ holds if and only if every constrained atom A in $\mathcal{CI}_1$ (resp., $\mathcal{CI}_2$) is true in $\mathcal{C}_2$ (resp., $\mathcal{CI}_1$). Hence this problem reduces to a linear number of tests ($|\mathcal{CI}_1|$ + $|\mathcal{CI}_2|$ many) whether a constrained atom A is true in a c-interpretation $\mathcal{CI}$. From Theorem 3 and the fact that PSPACE and Π_{k+1}^P are closed under conjunction, the membership part of 1. follows. For the membership part of 2., observe that

[4] Fixed language means here and in other complexity results that the predicates and constants are from fixed finite sets.

[5] Note that the free variables in the constraint part which appear not in the head have an implicit existential quantifier. If such variables are excluded, the complexity decreases by one level to Π_{k+1}^P.

$\mathcal{CI}_1$, $\mathcal{CI}_2$ can be easily normalized, and then only a constant number of tests as described remains.

For the hardness parts, observe that in the hardness proofs in Theorem 3, the formula $\mathcal{E}$ in 1. and 2. resp. $\mathcal{E}/\mathcal{E}'$ in 3. can be chosen to be a tautology resp. a tautology/falsity, such that $[p(\mathbf{X}) \leftarrow \mathcal{E}] = \{p(\mathbf{t}) \mid \mathbf{t} \in \mathcal{H}\}$ resp. $[p(\mathbf{X}) \leftarrow \mathcal{E}] = \{p(\mathbf{t}) \mid \mathbf{t} \in \mathcal{H}\}$ / $[p(\mathbf{X}) \leftarrow \mathcal{E}'] = \{p(\mathbf{t}) \mid \mathbf{t} \in \mathcal{H}\}$. Thus, for $\mathcal{CI}_1 = [p(\mathbf{X}) \leftarrow \mathcal{E}]$ and $\mathcal{CI}_2 = [p(\mathbf{X}) \leftarrow \mathcal{E}']$, one of the inclusions $\mathcal{CI}_1 \leq \mathcal{CI}_2$ or $\mathcal{CI}_2 \leq \mathcal{CI}_1$ is satisfied, and truth of $p(\mathbf{X}) \leftarrow \mathcal{E}$ in $\mathcal{CI}_2$ amounts to $\mathcal{CI}_1 \sim \mathcal{CI}_2$. □

The results in Theorems 3–4 imply that important problems on c-interpretations are intractable even if $k = 0$, i.e., no quantifiers occur in constraint parts $\mathcal{E}$; here, however, still implicit existential quantifiers are possible.

For the important case that all variables in the constraint part $\mathcal{E}$ are from the head, the complexities of the problems are not that drastic. In particular, the complexity results in parts 2. and 3. are lowered to polynomial time, i.e., the problems are efficiently solvable.

If we contrast the above complexity results with the constraint based transformation, we can see that intuitively each level of negation in the original program P adds one quantifier alternation to constraint parts and thus one level of complexity in the polynomial hierarchy; if the program is free of nonmonotonic negation **not**, then the complexity is at the first level (Π_1^P); if there is one level of negation, it is at the second (Π_2^P); and so on. In fact, this intuition is supported by the proof of Theorem 7 below.

Intuitively, a c-interpretation for representing a set of ground atoms can be more compact (i.e., requires less space) if quantifiers and variables in the constraint parts that do not occur in the heads are allowed. On the other hand, a more compact representation may need more time (measured in the size of the representation) for eliciting the represented ground facts. Thus, constrained interpretations/models save space (and can often represent infinite sets of ground atoms in a finite way) but it may be slower to answer queries when compared to a completely instantiated ground stable model (though the latter may be infinite and hence such a storage scheme would be un-implementable – or, even in the function-free case, it may require so much storage so as to exhaust available memory). Section 6 analyses the trade-offs in different representations of stable models.

4. Constrained Non-Ground Stable Models

In this section, we present the concept of non-ground stable model. It relies on the constraint-based transformation of logic programs.

Definition 9 A c-interpretation $\mathcal{CI}$ is a *constrained non-ground* (*CNG*) *stable model* of P if and only if $\mathcal{CI} \sim \mathsf{lfp}(W_{\mathbf{CT}(P,\mathcal{CI})}) = FNG_P(\mathcal{CI})$. □

Two simple examples of non-ground stable models are given below. Henceforth, we mean by "stable models" the ground stable models of Gelfond and Lifschitz. The phrase "constrained non-ground stable model" then refers to c-interpretations that are stable in the above sense.

Example 9 For the program of Example 1, the c-interpretation

$$\begin{aligned}\mathcal{CI} = \{&p(X,Y) \leftarrow X = Y,\\ &q(X,Y) \leftarrow X \neq Y,\\ &r(X,Y) \leftarrow X = a \,\&\, Y = f(a)\,\}\end{aligned}$$

is a CNG-stable model. □

Example 10 Consider the program P:

$$\begin{aligned}&p(X) \leftarrow \mathbf{not}(q(X)), \quad q(X) \leftarrow \mathbf{not}(p(X)),\\ &r(X) \leftarrow p(X), \quad r(X) \leftarrow q(X), \quad s(0, f(0)) \leftarrow .\end{aligned}$$

This program has uncountably many ground stable models. Each of them is a CNG-stable model. The c-interpretation $\mathcal{CI} = \{p(X) \leftarrow \ , \ r(X) \leftarrow \ , \ s(X,Y) \leftarrow X = 0 \,\&\, Y = f(0)\ \}$, however, is a CNG-stable model which is a compact representation of the ground stable model that makes all atoms for p and r true, as well as $s(0, f(0))$. □

The above example brings up an interesting question about representations: Given two different c-interpretations $\mathcal{CI}_1$ and $\mathcal{CI}_2$ such that $\mathcal{CI}_1 \sim \mathcal{CI}_2$, is it possible that one of these could be CNG-stable, while the other is not? The following result, which follows from Theorem 2, shows that this cannot occur.

Proposition 3 *Let P be a logic program and $\mathcal{CI}_1$, $\mathcal{CI}_2$ c-interpretations such that $\mathcal{CI}_1 \sim \mathcal{CI}_2$. Then, $\mathcal{CI}_1$ is a CNG-stable model of P iff $\mathcal{CI}_2$ is a CNG-stable model of P.*

As a consequence, the notion of CNG-stable models generalizes the notion of a ground stable model.

Theorem 5 *Let P be a normal logic program. Then:*

1. If I is a ground stable model, then the set $\{p(\mathbf{X}) \leftarrow \mathbf{X}=\mathbf{t} \mid p(\mathbf{t}) \in I\}$ is a CNG-stable model.
2. If $\mathcal{CI}$ is a CNG-stable model, then $[\mathcal{CI}]$ is a ground stable model.

Proof. Part 1: Since I is a stable model, it holds $I = F_P(I)$. By Lemma 1, we have $F_P(I) = [FNG_P(\mathcal{CI})]$ for any $\mathcal{CI}$ such that $[\mathcal{CI}] = I$. Hence, $\mathcal{CI} \sim FNG_P(\mathcal{CI})$. This means I (seen as a CNG interpretation) is a CNG stable model of P.

Part 2: If $\mathcal{CI} \sim FNG_P(\mathcal{CI})$, then $[\mathcal{CI}] = FNG_P(\mathcal{CI}) = F_P([\mathcal{CI}])$ by Lemma 1. Hence, $[\mathcal{CI}]$ is a ground stable model of P. □

4.1. Constrained Non-Ground Well-Founded Semantics

In this section, we show how the operator FNG_P may be used to define a non-ground version of the well-founded semantics of P. Specifically, such a semantics can be characterized by $\mathsf{lfp}(FNG_P^2)$ and $\mathsf{gfp}(FNG_P^2)$.

We proceed to define the *constrained non-ground well-founded semantics* (CNG-WFS) as follows.

Definition 10 Let $A = p(\mathbf{t})$ be an atom (not necessarily ground) and let P be a normal logic program. Then,

(i) A is *true according to the CNG-WFS of* P, iff $p(\mathbf{X}) \leftarrow \mathbf{X}=\mathbf{t}$ is true in $\mathsf{lfp}(FNG_P^2)$.

(ii) A *is false according to the CNG-WFS of* P, iff $p(\mathbf{X}) \leftarrow \mathbf{X}=\mathbf{t}$ is true in $\overline{\mathsf{gfp}(FNG_P^2)}$ (i.e., each ground atom A' in $[p(\mathbf{X}) \leftarrow \mathbf{X}=\mathbf{t}]$ is false in $\mathsf{gfp}(FNG_P^2)$); equivalently, each constrained atom $p(\mathbf{X}) \leftarrow \mathcal{E}$ in $\mathsf{gfp}(FNG_P^2)$ with atom p in the head is true in the c-interpretation $[p(\mathbf{X}) \leftarrow \forall\mathbf{Y}\neg(\mathbf{X}=\mathbf{t})]$, where $\mathbf{XY}$ are the free variables of $\mathbf{X}=\mathbf{t}$.

(iii) A *is* unknown *according to the CNG-WFS of* P iff A is neither true nor false according to the CNG-WFS of P. □

Note that in the case where $\mathsf{lfp}(FNG_P^2)$ is normal, truth of A according to CNG-WFS amounts to

$$\mathcal{H} \models \forall\mathbf{X}\mathbf{Y}\mathbf{X}_1\exists\mathbf{Y}_1.((\mathbf{X}=\mathbf{t})\&(\mathbf{X}=\mathbf{X}_1) \rightarrow \mathcal{E}_1),$$

where A is rewritten to $p(\mathbf{X}) \leftarrow \mathbf{X}=\mathbf{t}$ and $\mathbf{XY}$ are the free variables of $\mathbf{X}=\mathbf{t}$, and $p(\mathbf{X}_1) \leftarrow \mathcal{E}_1(\mathbf{X}_1, \mathbf{Y}_1)$ is the unique constrained atom in $\mathsf{lfp}(FNG_P^2)$ with p in the head (if no such atom exists, A is clearly not true according to CNG-WFS). Similarly, falsity of A according to CNG-WFS amounts to

$$\mathcal{H} \models \forall\mathbf{X}\mathbf{Y}\mathbf{X}_i\mathbf{Y}_i.((\mathbf{X}=\mathbf{t})\&(\mathbf{X}=\mathbf{X}_i) \rightarrow \neg\mathcal{E}_i),$$

for all constraint atoms $p(\mathbf{X}) \leftarrow \mathcal{E}_i(\mathbf{X}_i, \mathbf{Y}_i)$ in $\mathsf{gfp}(FNG_P^2)$ with p in the head (if no such atom exists, then A is clearly false according to the CNG-WFS).

The above formulation possesses important implications for query processing from a normal logic program P under the well-founded semantics. It suggests the following procedure. At compile-time, $\mathsf{lfp}(FNG_P^2)$ and $\mathsf{gfp}(FNG_P^2)$ (or $\overline{\mathsf{gfp}(FNG_P^2)}$) may be *pre-computed* and *stored*. This is in particular feasible if these c-interpretations are finite (e.g., in case of a function-free program P); they can be normalized and simplified by equivalence preserving transformations. Then at run-time, when a query $A = p(\mathbf{t})$ is posed, *equational reasoning on* $\mathcal{H}$ can be performed for checking which of the three conditions (i.e. true, false, unknown) is satisfied.

The following theorem shows that this procedure is actually sound, by stating that the CNG-WFS is an accurate non-ground representation of the well-founded semantics.

Theorem 6 *Let P be a normal logic program. Then, a (possibly non-ground) atom A is true (resp. false) in the CNG-WFS of P iff all ground instances of A are true (resp. false) in the ground WFS of P.* □

Proof. Let $\mathcal{CI}$ be any c-interpretation. An easy induction on i using Lemma 1 shows that for every $i \geq 1$, $[FNG^i(\mathcal{CI})] = F_P^i([\mathcal{CI}])$.

As a consequence, $[\mathsf{lfp}(FNG_P^2)] = \mathsf{lfp}(F_P^2)$. Therefore, A is true according to the CNG well-founded semantics of P, iff each ground instance of A is true in $\mathsf{lfp}(FNG_P^2)$, iff each ground instance of A is true according to the well-founded semantics of P.

Similarly, it follows that $[\mathsf{gfp}(FNG_P^2)] = \mathsf{gfp}(F_P^2)$. Note that $\mathsf{gfp}(FNG_P^2)$ is the fixpoint of the possibly transfinite sequence $G_0 = \Omega$, $G_\eta = FNG_P^2(\bigcup_{\zeta<\eta} G_\zeta)$, for every ordinal $\eta > 0$, where Ω is the set of all ground atoms of the language.

Therefore, an atom A of form $p(\mathbf{t})$ is false according to the CNG well-founded semantics of P, iff $p(\mathbf{X}) \leftarrow \mathbf{X}=\mathbf{t}$ is true in $\overline{\mathsf{gfp}(FNG_P^2)}$, iff each ground atom in $[p(\mathbf{X}) \leftarrow \mathbf{X}=\mathbf{t}]$ is false in $\mathsf{gfp}(FNG_P^2)$, iff for each ground instance A' of A, $A' \notin \mathsf{gfp}(F_P^2)$. □

The following theorem tells us the complexity of answering a query using the CNG-WFS model of a

function-free logic program. The results are similar to the ones for deciding truth of a constraint atom in a c-interpretation (Theorem 3). However, there is a small subtlety which must not be overlooked. While in Theorem 3 the c-interpretation is arbitrary, in the present theorem it must represent $\mathsf{lfp}(FNG^2_P)$ resp. $\mathsf{gfp}(FNG^2_P)$ of some logic program P. In fact, we assume in the hardness parts that $\mathsf{lfp}(FNG^2_P)$ and $\mathsf{gfp}(FNG^2_P)$ are given by representatives that can be obtained by following the standard fixpoint construction. Thus, in effect, this result strengthens Theorem 3.

Theorem 7 *Suppose we are given as input* $\mathsf{lfp}(FNG^2_P)$, $\mathsf{gfp}(FNG^2_P)$ *for a function-free program* P *and an atom* A (*not necessarily ground*). *Then, determining whether* A *is true* (*resp. false*) *in the CNG-WFS of* P *is*

1. *PSPACE-complete, if the language* $\mathcal{L}$ *is given by* P;
2. NP-*complete* (*resp.* coNP-*complete*), *if* A *is ground and all constraint parts in* $\mathsf{lfp}(FNG^2_P)$ ($\mathsf{gfp}(FNG^2_P)$) *are existential, regardless of a fixed language* $\mathcal{L}$;
3. *solvable in polynomial time with constantly many* NP-*oracle calls and both* NP *and* coNP-*hard, if the language* $\mathcal{L}$ *is fixed and all constraint parts in* $\mathsf{lfp}(FNG^2_P)$ ($\mathsf{gfp}(FNG^2_P)$) *are existential.*

Proof. See [10]. □

As before, if all variables in the constraint parts are from the variables in the head, then the complexity decreases. In particular, the problems 2. and 3. are polynomial in this case. Thus, if we represent $\mathsf{lfp}(FNG^2_P)$ and $\mathsf{gfp}(FNG^2_P)$ by c-interpretations of this form (which are computed in the precompilation phase), all ground queries in CNG-WFS can be answered efficiently. As in the case of CNG-stable models, using this form we can gain an exponential reduction in the space needed for storing the well-founded model while we stay within the same order of time for query answering.

We remark that we get the same complexity results for determining truth resp. falsity of an atom in a given CNG stable model, if we assume that CNG stable models are computed by first computing the CNG-well-founded semantics, i.e., $\mathsf{lfp}(FNG^2_P)$ and $\mathsf{gfp}(FNG^2_P)$ and then extending this "partial" model if needed to a "total" CNG stable model of the program. This is an easy consequence of the fact that the program of the reduction in the proof of the previous theorem is stratified, and hence its well-founded model coincides with its unique total stable model, which means that $\mathsf{lfp}(FNG^2_P)$ is a CNG-stable model.

5. Algorithms

In this section, we present an algorithm for computing the CNG-stable models of a normal logic program. The algorithm, which builds on the CNG well-founded semantics, is effective for function-free programs. It is a generalization of an algorithm for computing the stable models of a ground logic program [33]. However, this generalization is not immediate, and is more complex than the algorithm for the ground case. Additional machinery is needed to make it work.

5.1. Computing the CNG Well-Founded Semantics

In this section, we present a procedure that has the following steps:

1. Pruned Non-Ground Fitting Semantics Computation: It is well known that Fitting's Kripke-Kleene semantics for logic programs [12] approximates the well-founded semantics (cf. [33]). In our first step, we use a non-ground version Φ_P of Fitting's operator [12] that iteratively computes not only an interpretation, but also simplifies the program P. The least fixpoint of this operator, $\mathsf{lfp}(\Phi_P) = (\mathcal{T}_{\text{fix}}, \mathcal{F}_{\text{fix}})$ satisfies $[\mathcal{T}_{\text{fix}}] \subseteq [\mathsf{lfp}(FNG^2_P)]$ and $[\mathcal{F}_{\text{fix}}] \subseteq \overline{[\mathsf{gfp}(FNG^2_P)]}$, i.e., approximates the CNG-WFS. The technical definition of this operator is a bit involved, and we refer the interested reader to [10] for the details.
2. After that, we are left with a set of rules that only involve atoms which are "unknown" according to Fitting's semantics. We will then compute the alternating fixpoint associated with these atoms, again in an incremental way; as in the preceding step, the program will continue to be pruned.
3. The end result of the above two phases has two parts – the first part is the well-founded semantics, while the second is a logic program all of whose atoms evaluate to unknown according to the WFS. This latter logic program will later be used in Section 5.2 to compute CNG-stable models.

Using the non-ground version of Fitting's operator, the constrained non-ground well-founded semantics may now be computed in the following way.

ALGORITHM CNGWFS(P)

Input: A normal logic program P.
Output: $\mathcal{T}$, $\mathcal{F}$ such that $[\mathcal{T}] = [\mathsf{lfp}(FNG^2_P)]$, $[\mathcal{F}] = \overline{[\mathsf{gfp}(FNG^2_P)]}$, simplification P^* of program P.

Step 1. Compute $\mathsf{lfp}(\Phi_P) = (\mathcal{T}_{\text{fix}}, \mathcal{F}_{\text{fix}})$ and set $(\mathcal{T}_0, \mathcal{F}_0) = (\mathcal{T}_{\text{fix}}, \mathcal{F}_{\text{fix}})$.
Step 2. $i = 0$; $P_0 = P$.
Step 3. For each rule R of the form $p(\mathbf{X}) \leftarrow \mathcal{E} \,|\, Body$ in P_i, do the following:

1. If $p(\mathbf{X}_1) \leftarrow \mathcal{E}_1(\mathbf{X}_1, \mathbf{Y}_1) \in \mathcal{T}_i$ and $\mathcal{H} \models \forall \mathbf{X}\mathbf{Y}\mathbf{X}_1 \exists \mathbf{Y}_1((\mathbf{X} = \mathbf{X}_1) \& \mathcal{E}(\mathbf{X}, \mathbf{Y}) \rightarrow \mathcal{E}_1(\mathbf{X}_1, \mathbf{Y}_1))$, then eliminate R from P_i.
2. If $p(\mathbf{X}_2) \leftarrow \mathcal{E}_2(\mathbf{X}_2, \mathbf{Y}_2) \in \mathcal{F}_i$ and $\mathcal{H} \models \forall \mathbf{X}\mathbf{Y}\mathbf{X}_2 \exists \mathbf{Y}_2((\mathbf{X} = \mathbf{X}_2) \& \mathcal{E}(\mathbf{X}, \mathbf{Y}) \rightarrow \mathcal{E}_2(\mathbf{X}_2, \mathbf{Y}_2))$, then eliminate R from P_i.
3. Otherwise, replace $\mathcal{E}$ in R by $\mathcal{E} \& ((\mathbf{X} = \mathbf{X}_1) \& \forall \mathbf{Y}_1 . \neg \mathcal{E}_1) \& ((\mathbf{X} = \mathbf{X}_2) \& \forall \mathbf{Y}_2 . \neg \mathcal{E}_2)$.

Let P_{i+1} denote the resulting program.

Step 4. $\mathcal{F}_{i+1} = \mathcal{F}_i \cup$
$\{p(\mathbf{X}) \leftarrow \forall \mathbf{Y} \neg \mathcal{E}(\mathbf{X}, \mathbf{Y}) \mid p(\mathbf{X}) \leftarrow \mathcal{E}(\mathbf{X}, \mathbf{Y}) \in FNG_{P_{i+1}}(\mathcal{T}_i)\} \cup$
$\{p(\mathbf{X}) \leftarrow \mid p \text{ does not occur in } FNG_{P_{i+1}}(\mathcal{T}_i)\}$
and normalize $\mathcal{F}_{i+1}$.
Step 5. $\mathcal{T}_{i+1} = \mathcal{T}_i \cup$
$FNG_{P_{i+1}}(\{p(\mathbf{X}) \leftarrow \forall \mathbf{Y} \neg \mathcal{E}(\mathbf{X}, \mathbf{Y}) \mid p(\mathbf{X}) \leftarrow \mathcal{E}(\mathbf{X}, \mathbf{Y}) \in \mathcal{F}_i\} \cup \{p(\mathbf{X}) \leftarrow \mid p \text{ occurs not in } \mathcal{F}_i\})$
and normalize $\mathcal{T}_{i+1}$.
Step 6. If $[\mathcal{F}_i] = [\mathcal{F}_{i+1}]$ and $[\mathcal{T}_i] = [\mathcal{T}_{i+1}]$, then halt and return $(\mathcal{T}_i, \mathcal{F}_i, P_{i+1})$. Otherwise, $i := i + 1$. Goto Step 3. □

We note that this procedure does not terminate on every input. In particular, Step 1. may involve an infinite computation. However, it will always terminate if the program P is function-free. As we focus on this case, the algorithm is thus effective for our purposes.

Example 11 Consider the following logic program P:

$$p(a) \leftarrow \quad , \qquad (1)$$
$$p(X) \leftarrow p(Y) \,\&\, \mathbf{not}(q(X, Y)), \qquad (2)$$
$$q(X, Y) \leftarrow \mathbf{not}(p(X)). \qquad (3)$$

Let us assume that the language $\mathcal{L}$ has two constants a, b (b could be provided e.g. by an additional dummy rule, which we avoid for simplicity).

The working of the algorithm CNGWFS is now as follows (some of the constraints are simplified in the presentation below):

1. In Step 1 of the algorithm, $(\mathcal{T}_{\text{fix}}, \mathcal{F}_{\text{fix}})$ are computed to be:

$$\mathcal{T}_{\text{fix}} = \{p(X) \leftarrow X = a, \;\; q(X, Y) \leftarrow \mathit{false}\};$$
$$\mathcal{F}_{\text{fix}} = \{p(X) \leftarrow \mathit{false}, \;\; q(X, Y) \leftarrow X = a\}.$$

We set $\mathcal{T}_0 = \mathcal{T}_{\text{fix}}$ and $\mathcal{F}_0 = \mathcal{F}_{\text{fix}}$ and $P_0 = P$.

2. Rule (1): Part (1) of Step 3 applies and we eliminate rule (1) from P.
3. Rule (2): Part (3) of Step 3 applies, and the rule is replaced by the rule

$$p(X) \leftarrow X \neq a \mid p(Y) \& \mathbf{not}(q(X, Y)).$$

4. Rule (3): Again Part (3) of Step 3 applies, and the rule is replaced by the new rule:

$$q(X, Y) \leftarrow X \neq a \mid \mathbf{not}(p(X)).$$

5. In Step 4, $\mathcal{F}_1$ is set to $\mathcal{F}_0$.
6. In Step 5, $\mathcal{T}_1$ is set to $\mathcal{T}_0$.
7. The algorithm halts in Step 6 as $\mathcal{T}_1 = \mathcal{T}_0$ and $\mathcal{F}_1 = \mathcal{F}_0$. It returns: $\mathcal{T}_1 = \{p(X) \leftarrow X = a\}$ and $\mathcal{F}_1 = \{q(X, Y) \leftarrow X = a\}$ and P_1 contains

$$p(X) \leftarrow X \neq a \mid p(Y) \,\&\, \mathbf{not}(q(X, Y)), \; (2')$$
$$q(X, Y) \leftarrow X \neq a \mid \mathbf{not}(p(X)). \qquad (3')$$

□

Theorem 8 *Let P be any function-free logic program. Then, the call of* CNGWFS(P) *effectively computes a triple $(\mathcal{T}, \mathcal{F}, P^*)$ such that $[\mathcal{T}] = [\mathsf{lfp}(FNG^2_P)]$, $[\mathcal{F}] = \overline{[\mathsf{gfp}(FNG^2_P)]}$, and the programs $P^* \cup T$ and P have the same ground stable models.* □

5.2. Computing constrained non-ground stable models

In this section, we describe how to compute all stable models of a normal logic program using the constraint based approach. The basic idea is to first compute the well-founded semantics using the CNGWFS algorithm given above. This algorithm then outputs a triple $(\mathcal{T}_i, \mathcal{F}_i, P_{i+1})$. All atoms in $[\mathcal{T}_i]$ (resp. $[\mathcal{F}_i]$) are true (resp. false) in all stable models of P. We proceed by looking at atoms that are not in $[\mathcal{T}_i] \cup [\mathcal{F}_i]$ and adopt a branch and bound procedure that searches an (abstract) tree called BB-tree(P). Prior to describing BB-tree(P), we need some definitions.

Definition 11 Let $(\mathcal{T}, \mathcal{F})$ be a consistent pair of c-interpretations. Then the *unknown set* $U(T, F)$ generated by (T, F) is

$$U(T, F) = \{\, p(\mathbf{X}) \leftarrow (\mathbf{X} = \mathbf{X}_1) \,\&\, \forall \mathbf{Y}_1 \neg \mathcal{E}_1(\mathbf{X}_1 \mathbf{Y}_1) \,\&\, (\mathbf{X} = \mathbf{X}_2) \,\&\, \forall \mathbf{Y}_2 \neg \mathcal{E}_2(\mathbf{X}_2 \mathbf{Y}_2) \mid p(\mathbf{X}_1) \leftarrow \mathcal{E}_1 \in \mathcal{T}, \; p(\mathbf{X}_2) \leftarrow \mathcal{E}_2 \in \mathcal{F}\}.$$

□

Example 12 Suppose we return to the program P of Example 11 and consider the sets $\mathcal{T}_1, \mathcal{F}_1$ returned as a consequence of the CNGWFS algorithm. In that case, after simplification, we have $U(\mathcal{T}_1, \mathcal{F}_1) = \{p(X) \leftarrow X \neq a;\; q(X, Y) \leftarrow X \neq a\}$. □

We next define the concept of modification of a program by an assumption $CA = p(\mathbf{X}) \leftarrow \mathcal{E}$. Intuitively, if CA is assumed true, then in the ground instantiation of the program P, all literals involving a ground atom $p(\mathbf{t})$ such that $p(\mathbf{t})$ is contained in $[CA]$ are eliminated from rule bodies, by deleting every positive occurrence of such a literal and by replacing every negative occurrence with *false*. The modification by assuming CA is false is symmetric.

Definition 12 Let $C = A \leftarrow \mathcal{F}|Body$ be a clause, and let $p(\mathbf{X}) \leftarrow \mathcal{E}(\mathbf{XY})$ be a constraint atom. Then the *modification of C assuming $p(\mathbf{X}) \leftarrow \mathcal{E}$ is true is the following set of clauses $\Delta^+(C, p(\mathbf{X}) \leftarrow \mathcal{E})$. Suppose the positive atoms in $Body$ involving p are $A_i = p(\mathbf{t}_i)$, $i = 1, \ldots, m$, and the negative atoms involving p are $B_j = \mathbf{not}(p(\mathbf{t}'_j))$, $j = 1, \ldots, n$. Set $\Delta_0^+ = \{C\}$, and execute the following two steps:*

Step 1. *For each positive $p(\mathbf{t}_i)$, $i = 1, \ldots, m$ construct Δ_{i+1}^+ from Δ_i^+ by adding for each clause $C' = A' \leftarrow \mathcal{F}' \mid Body'$ in Δ_i^+ the clause*

$$A' \leftarrow \mathcal{F}' \,\&\, (\mathbf{t}_i = \mathbf{X}) \,\&\, \mathcal{E} \mid Body' \setminus p(\mathbf{t}_i),$$

where $Body' \setminus p(\mathbf{t}_i)$ is $Body'$ with $p(\mathbf{t}_i)$ removed;

Step 2. *For each negative literal $\mathbf{not}(p(\mathbf{t}'_j))$, $j = 1, \ldots, n$ construct Δ_{m+j}^+ from Δ_{m+j-1}^+ by replacing the constraint of every clause $C' = A' \leftarrow \mathcal{F}' \mid Body'$ in Δ_{m+j-1}^+ with the constraint $\mathcal{F}' \,\&\, \forall \mathbf{XY}[(\mathbf{t}'_j = \mathbf{X}) \supset \neg \mathcal{E}(\mathbf{X}, \mathbf{Y})]$*

Then, let $\Delta^+(C, p(\mathbf{X}) \leftarrow \mathcal{E}) = \Delta_{m+n}^+$.

For any normal logic program P, the modification *of P* assuming $p(\mathbf{X}) \leftarrow \mathcal{E}$ is true*, denoted by $\Delta^+(P, p(\mathbf{X}) \leftarrow \mathcal{E})$, is*

$$\Delta^+(P, p(\mathbf{X}) \leftarrow \mathcal{E}) = \bigcup_{C \in P} \Delta^+(C, p(\mathbf{X}) \leftarrow \mathcal{E}).$$

The modification of a clause C (resp. program P) by assuming $p(\mathbf{X}) \leftarrow \mathcal{E}$ is false, denoted $\Delta^-(C, p(\mathbf{X}) \leftarrow \mathcal{E})$ (resp. $\Delta^-(P, p(\mathbf{X}) \leftarrow \mathcal{E})$), is symmetric. □

Remark. A more restrict definition of the modification of a clause, which strengthens the constraints of the clauses from Δ_i^+ included in Δ_{i+1}^+, is possible, but omitted for simplicity.

Example 13 Suppose P is as discussed in Examples 11 and 12. Then the modification of clause (2') by assuming the constrained atom $CA = p(X') \leftarrow X' \neq a$ to be true yields

$$\begin{aligned}\Delta^+(2', CA) = \\ \{\, & p(X) \leftarrow X \neq a \mid p(Y) \,\&\, \mathbf{not}(q(X,Y)), \\ & p(X) \leftarrow X \neq a \,\&\, (Y = X') \,\&\, \\ & \qquad (X' \neq a) \mid \mathbf{not}(q(X,Y)) \qquad \}.\end{aligned}$$

for the clause $(3') = q(X, Y) \leftarrow X \neq a \mid \mathbf{not}(p(X))$, we obtain

$$\begin{aligned}\Delta^+(3', CA) = \\ \{q(X,Y) \leftarrow & X \neq a \,\&\, \forall X'.(X = X' \supset \\ & \neg(X' \neq a)) \mid \mathbf{not}(p(X))\} \\ = \{\,\}.\end{aligned}$$

□

Definition 13 A *constraint splitting strategy*, CS, is any mapping from satisfiable constrained atoms to pairs of satisfiable constrained atoms such that

$$\mathsf{CS}(p(\mathbf{X}) \leftarrow \mathcal{E}) = (p(\mathbf{X}) \leftarrow \mathcal{E}_1, p(\mathbf{X}) \leftarrow \mathcal{E}_2)),$$

where $\mathcal{E}_1 = \mathcal{E}_2 = \mathcal{E}$ if $[A \leftarrow \mathcal{E}]$ contains a single ground atom, and otherwise $G_1 = [p(\mathbf{X}) \leftarrow \mathcal{E}_1]$ and $G_2 = [p(\mathbf{X}) \leftarrow \mathcal{E}_2]$ satisfy (i) $G_1 \cap G_2 = \emptyset$, and (ii) $G_1 \cup G_2 = [p(\mathbf{X}) \leftarrow \mathcal{E}]$. □

In general, there are many different constraint splitting strategies that may be applied. Two examples are (others exist, but are not discussed here).

1. If $A \leftarrow \mathcal{E}$ is a constraint, and $\mathcal{E} = \mathcal{E}_1 \vee \mathcal{E}_2$ where $\mathcal{E}_1$ and $\mathcal{E}_2$ are satisfiable but incomparable, then this constrained atom may be split into: $A \leftarrow \mathcal{E}_2$ and $A \leftarrow \mathcal{E}_1 \,\&\, \neg\mathcal{E}_2$.
2. If $A \leftarrow \mathcal{E}$ is a satisfiable constraint such that $[A \leftarrow \mathcal{E}]$ contains more than one atom, and σ is a solution of $\mathcal{E}$, then, in abuse of notation, $A \leftarrow \mathcal{E} \,\&\, \neg\sigma$ and $A \leftarrow \sigma$ represents a splitting.

Example 14 Let P be the program of Example 12, and consider the constrained atom $A = p(X) \leftarrow X \neq a$ contained in $U(\mathcal{T}_1, \mathcal{F}_1)$. We notice that $\sigma = \{X = b\}$ is the only solution of the constraint $X \neq a$ over the Herbrand universe consisting just of the constants a, b. Thus, any CS returns (A, A). If there were additional constants, one possible way to split A, following the second strategy from above, is into the following two constraints:

$$p(X) \leftarrow X = b \;\text{ and }\; p(X) \leftarrow X \neq a \,\&\, X \neq b.$$ □

Definition 14 A *split selection strategy*, SSS is a mapping from the natural numbers to constraint splitting strategies, (i.e. $SSS(i)$ is a constraint splitting strategy) which satisfies the following property: Suppose $A \leftarrow \mathcal{E}$ is any constrained atom, where $\mathcal{E}$ is

solvable. Then, there exists an integer i, such that $SSS(i)(A \leftarrow \mathcal{E}) = (A \leftarrow \mathcal{E}_1, A \leftarrow \mathcal{E}_2)$, and $[A \leftarrow \mathcal{E}_1]$ contains a single ground atom; such an i is called *basic* for $A \leftarrow \mathcal{E}$. □

What a split selection strategy does is to look at a counter i and choose a constraint splitting strategy. However, split selection strategies are "converging" in the sense that eventually, a single ground atom is chosen. Notice that if we choose the second splitting strategy from above, then $i = 1$ can be basic for every proper $A \leftarrow \mathcal{E}$.

Definition 15 Associated with any logic program P, and any split selection strategy SSS, is a tree, called BB-*tree*(P, SSS) constructively defined as follows:

Each node N is labeled with a quadruple $(P, \mathcal{T}, \mathcal{F}, U)$, where P is a program, T and F are c-interpretations describing true and false atoms, respectively, and $\mathcal{U} = U(\mathcal{T}, \mathcal{F})$ is the unknown set generated by them. N is a *success node* if $[\mathcal{T}] \cup [\mathcal{F}] = B_L$, is a *failure node* if $[\mathcal{T}] \cap [\mathcal{F}] \neq \emptyset$, and an *open node* otherwise.

1. The root N_0 of BB-tree(P) is labeled with $(P_0, \mathcal{T}_0, \mathcal{F}_0, \mathcal{U}_0)$ where $(\mathcal{T}_0, \mathcal{F}_0, P_0)$ is the result of $\mathrm{CNGWFS}(P)$ and $\mathcal{U}_0 = U(\mathcal{T}_0, \mathcal{F}_0)$.
2. For each open node $N = (P_N, \mathcal{T}_N, \mathcal{F}_N, \mathcal{U}_N)$, nondeterministically pick a constrained atom $A \leftarrow \mathcal{E}$ in $\mathcal{U}_N$. The node N has children, $N_1, N_2, \ldots$ as follows:

Each child N_i has label $(P_i, \mathcal{T}_i, \mathcal{F}_i, \mathcal{U}_i)$, where P_i is a program and $\mathcal{T}_i, \mathcal{F}_i$, and $\mathcal{U}_i$ are c-interpretations obtained as follows. Let P_i^+ be the program P_N modified by adopting the assumption $\mathbf{A}_i$ (described below) and augmented by $\mathcal{T}_N$, and let $(\mathcal{T}_i^*, \mathcal{F}_i^*, P_i^*)$ be the result of $\mathrm{CNGWFS}(P_i^+)$. Then,

- $P_i = P_i^*$;
- $\mathcal{T}_i = \mathcal{T}_N \cup \mathcal{T}_i^*$;
- $\mathcal{F}_i = \mathcal{F}_N \cup \mathcal{F}_i^*$;
- $\mathcal{U}_i = U(\mathcal{T}_i, \mathcal{F}_i)$.

The assumptions $\mathbf{A}_i$ are as follows:

- $\mathbf{A}_1$ is that $A \leftarrow \mathcal{E}$ is false;
- $\mathbf{A}_2$ is that $A \leftarrow \mathcal{E}$ is true;
- for odd integers $i = 2k + 1 > 2$, $\mathbf{A}_i$ is that $A \leftarrow \mathcal{E}_k$ is false, where $\mathcal{E}_k$ appears in $SSS(k)(A \leftarrow \mathcal{E}) = (A \leftarrow \mathcal{E}_k, A \leftarrow \mathcal{F}_k)$, and
- for even integers $i = 2k + 2 > 2$, $\mathbf{A}_i$ is that $A \leftarrow \mathcal{E}_k$ is true, where $\mathcal{E}_k$ appears in $SSS(k)(A \leftarrow \mathcal{E}) = (A \leftarrow \mathcal{E}_k, A \leftarrow \mathcal{F}_k)$. □

Note that this tree is potentially infinitely branching. However, due to the assumption on the split selection strategy, in the actual procedure for computing the non-ground stable models, finite branching suffices (exploiting that at some point, $SSS(k)(A \leftarrow \mathcal{E})$ must yield a single ground atom).

Before proceeding any further, let us take a quick example to see what a BB-tree may look like.

Example 15 Suppose we return to program P of Examples 11–14. For the Herbrand universe of two constants a and b, P has two ground stable models: $M_1 = \{p(a), q(b,a), q(b,b)\}$ and $M_2 = \{p(a), p(b)\}$.

The root of any BB-tree(P, SSS) is labeled with the quadruple $N_1 = (P_1, \mathcal{T}_1, \mathcal{F}_1, \mathcal{U}_1)$ where $P_1, \mathcal{T}_1, \mathcal{F}_1$ are as described in the preceding examples, and $\mathcal{U}_1 = U(\mathcal{T}_1, \mathcal{F}_1)$ as described in Example 12. Suppose we pick the constraint $p(X) \leftarrow X \neq a$ from $\mathcal{U}_1$ and generate the children $N_1', N_2', \ldots$ of N_1. The first child, N_1', is generated according to the assumption that $p(X') \leftarrow X' \neq a$ is false. After simplifications, the program P_1^+ is:

$$
\begin{aligned}
p(X) &\leftarrow X \neq a \,\&\, Y = a \mid p(Y) \,\&\, \mathbf{not}(q(X,Y)).\\
q(X,Y) &\leftarrow X \neq a \mid \mathbf{not}(p(X)).\\
q(X,Y) &\leftarrow X \neq a.\\
p(X) &\leftarrow X = a.
\end{aligned}
$$

(The last clause is from $\mathcal{T}_1$). $\mathrm{CNGWFS}(P_1^+)$ returns $(P_1^*, \mathcal{T}_2^*, \mathcal{F}_2^*)$ where $\mathcal{T}_1^* = \{p(X) \leftarrow X = a, q(X,Y) \leftarrow X \neq a\}$ and $\mathcal{F}_1^* = \{p(X) \leftarrow X \neq a, q(X,Y) \leftarrow X = a\}$. Notice that $\mathcal{T}_1' = \mathcal{T}_1^*$, $\mathcal{F}_1' = \mathcal{F}_1^*$, and $\mathcal{U}_1' = \emptyset$; hence, N_1' is a success node.

The second child, N_2', is generated by assuming $p(X') \leftarrow X \neq a$ is true. After simplifications, program P_2^+ is:

$$
\begin{aligned}
p(X) &\leftarrow X \neq a \,\&\, \mid p(Y) \,\&\, \mathbf{not}(q(X,Y)).\\
p(X) &\leftarrow X \neq a \,\&\, Y \neq a \mid \mathbf{not}(q(X,Y)).\\
p(X) &\leftarrow X = a.
\end{aligned}
$$

Applying $\mathrm{CNGWFS}(P_2^+)$, we get $\mathcal{T}_2 = \{p(X) \leftarrow true\}$ and $\mathcal{F}_2 = \{q(X,Y) \leftarrow true\}$. Also N_2' is a success node. □

Our basic algorithm builds the tree BB-tree(P, SSS), and uses the following simple operation **children**(N, CA, k) for generating the children of a node N:

children(N, CA, k)**:** Given a node N, a constrained atom $CA = A \leftarrow \mathcal{E}$, and and integer $k \geq 0$, the children N_{2k+1} and N_{2k+2} of the node N as above are generated and returned. Moreover, a flag *basic* is set to true if $[CA]$ contains a single atom or k is basic for CA.

Notice that the split selection strategy SSS is here implicit, and that **children** must report *basic* true after a finite number of calls **children**(N, CA, k), $k = 0, 1, \ldots$

An algorithm for stable model computation will now work as follows:

ALGORITHM CNGSTABLE(P):

Input: A logic program P.

Output: Collection S of all[6] CNG-stable models of P.

Step 1. Construct the root $N_0 = (P_0, \mathcal{T}_0, \mathcal{F}_0, \mathcal{U}_0)$.
If $\mathcal{U}_0 = \emptyset$, then set $S = \mathcal{T}_0$ and halt; otherwise, initialize list $Active$ to N_0.

Step 2. Set $S = \emptyset$.

Step 3. Pick a node $N = (P_N, \mathcal{T}_N, \mathcal{F}_N, \mathcal{U}_N)$ in list *Active*;

Step 4. if N has no pair $\ell_N = (CA, k)$ of an constrained atom CA and an integer k attached, then set $k = 0$, select a constrained atom $CA = p(\mathbf{X}) \leftarrow \mathcal{E}$ from $\mathcal{U}_N$, and attach $\ell_N = (CA, k)$ to N, else update k to $k + 1$.

Step 5. Execute **children**(N, CA, k), and let $N_1 = (P_1, \mathcal{T}_1, \mathcal{F}_1, \mathcal{U}_1)$ and $N_2 = (P_2, \mathcal{T}_2, \mathcal{F}_2, \mathcal{U}_2)$ be the nodes that are returned.
If *basic* = true, then remove N from list *Active*.

Step 6. For N_j where $j = 1, 2$ do the following:

1. If $[\mathcal{T}_j], \cup [\mathcal{F}_j] = B_L$ and $[\mathcal{T}_j] \cap [\mathcal{F}_j] = \emptyset$, then label N_j as a *success* node and insert $\mathcal{T}_j$ into S.
2. If $[\mathcal{T}_j] \cap [\mathcal{F}_j] \neq \emptyset$, then label N_j as a *failure* node.
3. If neither of the previous two cases applies, then N_j is labeled open and is added to *Active*.

Step 7. If $Active \neq \emptyset$, then goto Step 3. □

In the above algorithm, Steps 3 and 4 are nondeterministic, and various heuristics may be used to choose N and $p(\mathbf{X}) \leftarrow \mathcal{E}$.

For Step 3, we might use e.g. a depth-first or breadth-first strategy, or based on some weighting function, a greedy strategy; for Step 4, one could implement exhaustion of a predicate p_1, followed by exhaustion of p_2 etc, where the order of processing is determined by some criterion, possibly based on a heuristics for the "intricacy" of a predicate estimated from the structure of the constraints in the c-interpretations. Different such strategies are imaginable. Notice that in a previous version of this paper, a *fixed* randomized constraint splitting strategy was applied.

[6] The algorithm may be easily modified to compute a single CNG-stable model of P, instead of all by halting the first time an insertion into S is made.

Example 16 We continue the example from above. In Step 1, the list *Active* is initialized with the node N_1 from Example 12, which is chosen in Step 3. Suppose the constrained atom $CA = p(X') \leftarrow X' \neq a$ is chosen in Step 4; then, $(CA, 0)$ is attached to N_1, and in Step 5 the children N_1' and N_2' of N_1 as in Example 12 are generated. If the Herbrand universe consists of a and b, then *basic* is set true, and thus node N_1 is removed from *Active*.

In Step 6, condition 1. applies to both N_1' and N_2', which are labeled as success nodes and inserted into S. Since in Step 7 the list *Active* is empty, the algorithm terminates and outputs S with two CNG-stable models: $\mathcal{I}_1 = \{p(X) \leftarrow X = a,\ q(X, Y) \leftarrow X \neq a\}$ and $\mathcal{I}_2 = \{p(X) \leftarrow true\}$. They amount to the two ground stable models of the program P. □

Theorem 9 *Let P be a function-free logic program. Then, assuming a proper split selection strategy, algorithm* CNGSTABLE(P) *computes all stable models of P and halts in finite time.*

In order to see that the claimed results holds, observe that the tree which is generated by CNGSTABLE(P) is finitely branching. Moreover, by the properties of a splitting strategy, the set of undefined atoms at node N is always a proper superset of the set of undefined atoms at any of its children; hence, every branch in the tree is finite, and thus, by König's Lemma, the tree is finite. It remains thus to argue that the output is indeed a collection of all stable models of P. This can be established by a generalization of the arguments in [33] for the ground case, taking into account that each node N has two children which correspond to the assumption that a ground atom $p(\mathbf{t})$ is true and false, respectively (which guarantees completeness), and that the generation of an open or success node corresponds to a contracted sequence of respective generations in the ground case.

Notice that, in general, S contains several equivalent CNG-stable models. The above algorithm can be enhanced by pruning techniques, in which the generation of subtrees which do not contribute any CNG stable model or not one inequivalent from already computed ones is reduced. For example, if two nodes

$N = (P, \mathcal{T}, \mathcal{F}, \mathcal{U})$ and $N' = (P', \mathcal{T}', \mathcal{F}', \mathcal{U}')$ satisfy $[\mathcal{T}] \subseteq [\mathcal{T}']$ and $[\ F] \subseteq [\mathcal{F}']$, then only node N needs to be considered further; if $[\mathcal{T}] \supset [\mathcal{CI}]$ (resp. $[\mathcal{F}] \supset [\mathcal{CI}]$) for some CNG-stable model $\mathcal{CI}$ in S, then node N need not be considered further.

6. Effect on Actual Implementations

The techniques presented here give us now *four* possible approaches to computing the stable model and the well-founded semantics of logic programs. We analyze the pros and cons of these approaches and discuss under which conditions a given approach is more appropriate. The discussion focuses on stable models, but with the understanding that similar comments apply to well-founded semantics.

The Classical Approach. Here, elementary syntax checking is performed at compile-time, and all computations are performed during run-time using any (sound and complete) deduction technique such as resolution.

Advantages. The advantage is, clearly, that compilation is very fast, as no precomputation is required. In addition, no space is utilized in storing models.

Disadvantages. Currently, there exist almost no procedures for run-time query evaluation under stable semantics (the best known ones work in the case of locally stratified programs). Even if such a procedure were to exist, they will necessarily be very slow.

Full Ground Pre-Computation. In this approach, all stable models (or as many as are desired) are precomputed at compile time, and stored as sets of *ground* atoms.

Advantages. Run-time query evaluation using this approach is very, very fast, and boils down to simple retrieval from a relational DBMS. Moreover, it facilitates executing queries such as aggregates, that can be difficult to execute under the classical approach.

Disadvantages. Compilation takes longer than in the classical case, and in many cases, it may be infeasible to store ground stable models – even in the Datalog case, the stable models may be too large to fit into memory.

Pre-Computations of NG-stable Models. In [15], Gottlob *et al.* showed how sets of *atoms* (non-ground but without constrains) can be used to represent stable models. These are called NG-stable models and can be pre-computed and stored, just as in the case of ground stable models.

Relative Advantages. The advantage of this approach is that run-time query evaluation is *faster* than in the classical approach, where all deduction is performed during run-time, but is *slower* than the Full Ground Pre-Computation approach. This is because checking whether an atom A is true in an NG-stable model requires determining whether an atom exists in that model that subsumes A; subsumptions being more expensive than membership checks, hence the Full Ground Pre-Computation approach is more advantageous from this point of view. This point is particularly relevant when indexing techniques are considered, which enable rapid location of "candidate" atoms.

Relative Disadvantages. With respect to compilation, NG-stable models are obviously more efficiently stored than completely ground stable models. In the latter case, even when dealing with Datalog programs, just the process of grounding out the program before the start of the full ground pre-computation may cause us to run out of memory. Of course, the classical approach is still better from the point of view of compilation time.

Pre-Computations of CNG-stable Models. Finally, the CNG-stable models approach described in this paper may be precomputed and stored.

Relative Advantages. From the point of time requirements for compilation, the new technique is more feasible than either the Full Ground Pre-Computation approach (due to memory limitation), or the Pre-Computation of NG-stable models approach. The CNG-stable model transform, $\mathbf{CT}(P, \mathcal{CI})$, is often faster than the NG-stable model transform because (1) sets of ground atoms can always be represented more compactly using constraints than just using atoms (after all, in the worst case, atoms may be viewed as constrained atoms with an empty/nonempty constraint). Often the constraint representation is much more compact (cf. Example 17). (2) Computing $\mathbf{CT}(P, \mathcal{CI})$ is quite easy – the time taken for the computation is proportional to the product of the number of negated atoms occurring in P, and the number of constrained atoms in $\mathcal{CI}$ – in contrast, the NG-stable models approach constructs a complex tree and performs an expensive operation called *anticover* computation (a detailed discussion of this is omitted for space reasons). Furthermore, CNG-stable models computed by Algorithm CNGSTABLE always require less (or, in the worst case equal space), than both NG-stable models as well as fully grounded stable models. Hence the demand for storage using CNG-stable models is usually more moderate than the NG-stable model or fully grounded approaches.

Relative Disadvantages. As for run-time query processing, this approach is slower than the NG-stable models because it may be necessary to check for solvability of very large and complicated constraints. With the NG-stable models, the only check required is subsumption.

Table 1 summarizes this discussion. If one examines this table, then we notice that the Classical Approach and the Full Grounding represent *extremes* – they are best for some things, and truly awful in others. In contrast, CNG (and NG) stable models represent *intermediate* approaches that adopt a "middle ground". The relative advantages of CNG vs. NG are listed above.

There are (at least) four groups – Constantini [6], Leone *et al.* [20, 19], Subrahmanian *et al.* [33], and Sacca and Zaniolo [32], who have studied methods for computing stable models based on first computing the well-founded semantics. In all these approaches, the well-founded model is first computed by a ground fixpoint computation – using the techniques of this paper, this can be replaced by the *non-ground* fixpoint computation using the operator FNG_P^2 that characterizes WFS. Furthermore, all the above approaches then compute stable models using some sort of non-deterministic search – e.g. Sacca and Zaniolo use explicit "choice" operators in clause bodies, while the others use a search strategy, e.g. branch and bound to "guess" the truth values of atoms that are assigned "unknown" in the well-founded model, and then search to see if this guess can be "verified". Non-ground versions of such non-deterministic search operators are defined by Constantini [6] as well as by Leone et al. [20, 19] and they can be used as the basis for a non-ground computation as well.

7. Related Work and Conclusion

To our knowledge, the only technique that currently presents a non-ground representation of the stable and well-founded semantics is the work of Gottlob *et al.* [15]. The work in [17] is related, but addresses only definite logic programs. Dix and Stolzenburg have recently presented work on non-ground representation of disjunctive logic programs a well, focusing on the D-WFS semantics [9].

Though [15] and this paper have the same goal, viz. that of developing non-ground representations of stable and well-founded semantics, they achieve these goals in quite different ways. There are two key differences: First, [15] considers the so-called S-interpretations, due to Falaschi et al. [11] – these are sets of atoms that are not necessarily ground. Thus, sets of atoms are stable models, not sets of constrained atoms. Secondly, [15] presents a way of constructing a set of clauses based on transforming the program P with respect to a set of atoms. The resulting set of clauses generated by this transformation may be *significantly larger* than P itself, and indeed, much larger than the CNG-stable models proposed here. To see this, consider the following example.

Example 17 Let P be the logic program:

$$p(X, X)$$

$$q(X, Y) \leftarrow \mathbf{not}(p(X, Y))$$

$$r(a, f(a))$$

In the [15] framework, we need to find a way of representing the set of all atoms $q(X, Y)$ with $X \neq Y$ without using constraints, a non-obvious task. In contrast, the c-interpretation

$$\{\, p(X, Y) \leftarrow X = Y;\ \ q(X, Y) \leftarrow X \neq Y;\ \ r(X, Y) \leftarrow X = a \,\&\, Y = f(a)\}$$

is a CNG-stable model that uses a compact representation. □

It turns out that each NG-stable model can be viewed as a CNG-stable model since an atom $p(\mathbf{t})$ can be written equivalently as the constrained atom $p(\mathbf{X}) \leftarrow \mathbf{X} = \mathbf{t}$. Given any NG-stable model SI, the construction in Algorithm 1 gives us a CNG-stable model that is equivalent, but more succinctly represented.

Other related work includes the work of McCain and Turner [26] who study how stable model semantics changes when the underlying language changes. This has a surface similarity to our work, but they do not attempt to develop non-ground representations of stable and well-founded semantics.

There is now a growing body of literature on computing stable models, but they are restricted mostly to the propositional (or ground) case [33, 3, 6, 8, 20, 19, 32]. The methods in these papers develop *representations* of non-ground stable models that can be computed by some procedure, though such a procedure has not been explicitly described.

The S-semantics group in Pisa has studied non-ground representations of logic programming semantics [11, 13, 34]. Fortunately for us, they did not develop non-ground versions of the stable and well-

Table 1
Comparison of approaches to computing stable models

Criterion	Best	2nd Best	3rd Best	Worst
Ease of Compilation	Classical	CNG Stable Mod.	NG-Stable Mod.	Full Ground
Storage Requirements	Classical	CNG Stable Mod.	NG-Stable Mod.	Full Ground
Run-Time Query Execution	Full Ground	NG-Stable Mod.	CNG Stable Mod.	Classical

founded semantics, which is what we do in this paper. Work by Marek, Nerode and Remmel [25] considers constraint models that are related to CNG-stable models. However, the framework is different, and algorithms are not addressed. Pollett and Remmel [29] consider logic programs with quantified Boolean formulas constraints, aiming at a more compact programming language rather than model representation. Query evaluation under der WFS using SLG resolution [4], implemented in the XSB system [30], is somewhat less related to our work, as it is a top down approach and, moreover, aims at answering a goal rather than representing the well-founded model. It can be used to compute a residual program for query evaluation under stable model semantics [30].

Almost all works on the study of the stable and well-founded semantics assume that these models are composed of *ground* atoms, and that programs are "grounded out" when attempting to perform the Gelfond-Lifschitz transform. This assumption is very useful for articulating the declarative semantics of logic programs with negation, and in explaining the intuitions behind the transformations and stability criteria. However, in practice, the approach of grounding programs is prohibitively expensive – when function symbols are present, it may be impossible, and even in the Datalog case, memory requirements may make it infeasible. For instance, to instantiate a clause containing 4 variables in say a personnel database describing 10,000 employees of a company, instantiation of this clause alone would lead to $(10000^4) = 10^{16}$ instances.

In this paper, we have proposed a method based on *constrained interpretations* to represent the stable and well-founded semantics in a non-ground manner. Constrained interpretations can often capture infinite sets of ground atoms in a finite way. We have described a non-ground version of the Gelfond-Lifschitz transform that utilizes these constraints to avoid grounding the program, even partially. Thus, the use of constraints facilitates: (1) representing in a succinct, non-ground way, a ground stable model (or models) and (2) performing the Gelfond-Lifschitz transform in a sound, complete, and non-ground way, thus avoiding the expensive grounding step in entirety.

In addition, we have derived complexity results on the efficiency of this approach, and have discussed how, and under what circumstances, four possible approaches to computing stable models may shape up relative to one another.

Several problems remain for further work. One concerns the structure of c-interpretations that are obtained as non-ground representatives of the stable models resp. the well-founded model of a normal logic program. In this context, it is an interesting issue to investigate classes of programs which under a predetermined set of simplification methods for constraints give rise to c-interpretation where the quantifier-depth is below a given limit. Another issue is using a more expressive logic than first-order logic for non-ground representation. In particular, an enrichment by generalized quantifiers such as a (possibly restricted) fixpoint operator might be useful for allowing to express predicate extensions which are not finitely representable in first-order logic. This requires further investigation.

Acknowledgements. The authors would like to thank H. Comon for clarifying remarks on the complexity of equational reasoning, and S. Vorobyov and P Mielniczuk for supplying papers on this subject. Moreover, they acknowledge the review comments on previous versions of this paper.

References

[1] C. Baral and M. Gelfond. Logic Programming and Knowledge Representation. *Journal of Logic Programming*, 19/20:73–148, 1994.

[2] C. Baral and V. Subrahmanian. Dualities Between Alternative Semantics for Logic Programming and Nonmonotonic Reasoning. *Journal of Automated Reasoning*, 10:399–420, 1993.

[3] C. Bell, A. Nerode, R. Ng, and V. Subrahmanian. Mixed Integer Programming Methods for Computing Non-Monotonic Deductive Databases. *Journal of the ACM*, 41(6):1178–1215, November 1994.

[4] W. Chen, T. Swift, and D. Warren. Efficient Top-Down Computation of Queries under the Well-Founded Semantics. *Journal of Logic Programming*, 24:161–199, 1995.

[5] H. Comon and P. Lescanne. Equational Problems and Disunification. *Journal of Symbolic Computation*, 7:371–425, 1989.

[6] S. Constantini. Contributions to the Stable Model Semantics of Logic Programs with Negation. *Theoretical Computer Science*, 150:231–255, 1993.

[7] J. Dix. Semantics of Logic Programs: Their Intuitions and Formal Properties. An Overview. In A. Fuhrmann and H. Rott, editors, *Logic, Action and Information. Proc. of the Konstanz Colloquium in Logic and Information* (*LogIn'92*), pages 241–329. DeGruyter, 1995.

[8] J. Dix and M. Müller. Implementing Semantics of Disjunctive Logic Programs Using Fringes and Abstract Properties. In L.-M. Pereira and A. Nerode, editors, *Proceedings of the Second International Workshop on Logic Programming and Nonmonotonic Reasoning* (*LPNMR-93*), pages 43–59, Lisbon, Portugal, July 1993. MIT Press.

[9] J. Dix and F. Stolzenburg. Computation of Non-Ground Disjunctive Well-Founded Semantics with Constraint Logic Programming. In J. Dix, L. M. Pereira, and T. C. Przymusinski, editors, *Proc. Workshop Non-Monotonic Extensions of Logic Programming*, pages 143–160. DeGruyter, 1996.

[10] T. Eiter, J. Lu, and V. Subrahmanian. A First-Order Representation of Stable Models. Technical Report 9805, Institut für Informatik, Universität Gießen, Germany, 1998.

[11] M. Falaschi, G. Levi, M. Martelli, and C. Palamidessi. A New Declarative Semantics for Logic Languages. In *Proceedings Fifth ICLP-88*, pages 993–1005, 1988.

[12] M. Fitting. A Kripke-Kleene Semantics for Logic Programs. *Journal of Logic Programming*, 2(4):295–312, 1985.

[13] M. Gabbrielli and G. Levi. Modeling Answer Constraints in Constraint Logic Programs. In *Proceedings ICLP-91*, pages 238–251. MIT Press, 1991.

[14] M. Gelfond and V. Lifschitz. The Stable Model Semantics for Logic Programming. In *Logic Programming: Proceedings Fifth Intl Conference and Symposium*, pages 1070–1080, Cambridge, Mass., 1988. MIT Press.

[15] G. Gottlob, S. Marcus, A. Nerode, G. Salzer, and V. Subrahmanian. A Non-Ground Realization of the Stable and Well-Founded Semantics. *Theoretical Computer Science*, 166:221–262, 1996.

[16] D. S. Johnson. A Catalog of Complexity Classes. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, chapter 2. Elsevier Science Publishers B.V. (North-Holland), 1990.

[17] V. Kagan, A. Nerode, and V. Subrahmanian. Computing Minimal Models by Partial Instantiation. *Theoretical Computer Science*, 155:157–177, 1996.

[18] K. Kunen. Answer Sets and Negation as Failure. In *Proceedings ICLP '87*, pages 219–228. MIT-Press, 1987.

[19] N. Leone, M. Romeo, P. Rullo, and D. Sacca. Effective Implementation of Negation in Database Logic Query Languages. In *LOGIDATA+: Deductive Databases with Complex Objects*, number 701 in LNCS, pages 159–175. Springer, 1993.

[20] N. Leone and P. Rullo. Safe Computation of the Well-Founded Semantics of DATALOG Queries. *Information Systems*, 17(1):17–31, 1992.

[21] V. Lifschitz. Computing Circumscription. In *Proceedings International Joint Conference on Artificial Intelligence IJCAI-85*, pages 121–127, 1985.

[22] J. Lloyd. *Foundations of Logic Programming*. Springer, Berlin, 1984, 1987.

[23] M. Maher. Complete Axiomatization of the Algebra of Finite, Rational and Infinite Trees. In *Proceedings IEEE LICS-88*, pages 348–357. IEEE Computer Science Press, 1988.

[24] W. Marek, A. Nerode, and J. Remmel. The Stable Models of a Predicate Logic Program. *Journal of Logic Programming*, 21(3):129–153, 1994.

[25] W. Marek, M. Truszczyński, and A. Rajasekar. Complexity of Extended Disjunctive Logic Programs. *Annals of Mathematics and Artificial Intelligence*, 15(3/4), 1995.

[26] N. McCain and H. Turner. Language Independence and Language Tolerance in Logic Programs. In *Proceedings ICLP-94*, pages 38–57, Santa Margherita Ligure, Italy, June 1994. MIT-Press.

[27] J. McCarthy. Applications of Circumscription to Formalizing Common-Sense Knowledge. *Artificial Intelligence*, 28:89–116, 1986.

[28] P. Mielniczuk. Basic Theory of Feature Trees. manuscript, 1997.

[29] C. Pollett and J. Remmel. Nonmonotonic Reasoning with Quantified Boolean Constraints. In *Proceedings of the 4th International Conference on Logic Programming and Nonmonotonic Reasoning* (*LPNMR-97*), number 1265 in LNCS, pages 18–39. Springer, 1997.

[30] P. Rao, K. Sagonas, T. Swift, D. Warren, and J. Freire. XSB: A System for Efficiently Computing Well-Founded Semantics. In *Proceedings of the 4th International Conference on Logic Programming and Nonmonotonic Reasoning* (*LPNMR-97*), number 1265 in LNCS, pages 430–440. Springer, 1997.

[31] R. Reiter. A Logic for Default Reasoning. *Artificial Intelligence*, 13:81–132, 1980.

[32] D. Saccà and C. Zaniolo. Stable Models and Non-Determinism in Logic Programs with Negation. In *Proceedings PODS-90*, pages 205–218, 1990.

[33] V. Subrahmanian, D. Nau, and C. Vago. WFS + Branch and Bound = Stable Models. *IEEE Transactions on Knowledge and Data Engineering*, 7:362–377, 1996.

[34] D. Turi. Extending S-Models to Logic Programs with Negation. In *Proceedings ICLP-91*, pages 397–411. MIT Press, 1991.

[35] A. van Gelder. The Alternating Fixpoint of Logic Programs With Negation. In *Proceedings PODS-89*, pages 1–10, 1989.

[36] 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.

[37] M. Vardi. On the Complexity of Bounded-Variable Queries. In *Proceedings PODS-95*, pages 266–276, 1995.

[38] S. Vorobyov. An improved lower bound for the elementary theories of trees. In J. K. S. M. A. McRobbie, editor, *Proceedings of the 13th Conference on Automated Deduction* (*CADE '96*), volume 1104 of *Lecture Notes in Artificial Intelligence*, pages 275–287. Springer, 1996.

[39] S. Vorobyov. Existential Theory of Term Algebras is in Quasi-Linear Non-Deterministic Time. Manuscript, February 1997.