In: Proceedings 4th International Conference on Logic Programming and Non-Monotonic Reasoning (LPNMR-97), LNCS 1265, pp. 198–217.

Computing Non-ground Representations of Stable Models

Thomas Eiter[1] and James Lu[2] and V.S. Subrahmanian[3]

[1] AG Informatik, Universität Giessen, Arndtstrasse 2, D-35392 Giessen, Germany. eiter@informatik.uni-giessen.de
[2] CS Dept, Bucknell University, Lewisburg, PA. lu@sol.cs.bucknell.edu
[3] Institute for Advanced Computer Studies, University of Maryland, College Park, Maryland 20742. vs@cs.umd.edu

Abstract. Turi [20] 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 show how non-ground representations of *both* the stable model and the well-founded semantics may be obtained through Turi's approach. As a practical consequence, 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. 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, non-ground representation, constraints, algorithms, complexity

1 Introduction

The declarative semantics of nonmonotonic logic programming has largely been based on methods that assume *propositional* programs. For example, the stable models [9] of a logic program are defined as certain minimal Herbrand models of the ground instantiation of the program. In addition, the Gelfond-Lifschitz transform that plays a key role in the definition of both the stable semantics and the well-founded semantics (WFS, for short) [21] for logic programming works only on ground instantiations [22, 1]. 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:

$$p(X,X) \leftarrow \quad q(X,Y) \leftarrow \mathbf{not}(p(X,Y)) \quad r(a,s(a)) \leftarrow$$

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 $r(a,s(a))$, 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*:

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

In contrast, were we to attempt to explicitly represent the stable model of this program using ground atoms, then an infinite set is required. □

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 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 if it is "equivalent" (in a sense made precise later) to the least constrained model (in the sense of Turi [20] and [7]) of $\mathbf{CT}(P, \mathcal{CI})$. We also illustrate how to define similarly a non-ground representation of the well-founded semantics. For the sake of effective computation, we will confine to finite CIs in our algorithms.

2 Preliminaries and Previous Results

We assume that L is an arbitrary, but fixed language[1] with equality generated by a finite signature Σ of constant symbols c, function symbols f, and predicate symbols p, as well as an infinite set of variable symbols. We assume that Σ has at least one constant. We often do not mention L explicitly. We use letters $a, b, c, \ldots$ for constants, $p, q, r, \ldots$ for predicates, and $X, Y, Z, \ldots$ for variable symbols. A bold face version of a symbol denotes a tuple of respective symbols, whose length is clear from the context. Throughout this paper, we focus on Herbrand models of L; $\mathcal{H}$ is the Herbrand pre-interpretation of L.

Elementary constraints with respect to L are all atoms $t_1 = t_2$, where t_1, t_2 are terms. A *constraint with respect to* L is any wff built from elementary constraints using &, $\vee$, $\neg$, $\forall$, and $\exists$. As usual, we write $t_1 \neq t_2$ for $\neg(t_1 = t_2)$.

Definition 1 *A* solution *to a constraint* $\mathcal{E}$ *is a ground substitution* σ *that binds all free variables in* $\mathcal{E}$ *(and possibly further) s.t.* $\mathcal{H} \models \mathcal{E}\sigma$*, i.e.,* $\mathcal{E}\sigma$ *is true in all Herbrand interpretations of* L*. A constraint is* solvable *if it has some solution* σ*.*

Example 2. Consider $\mathcal{E} = \forall X.(((X \neq f(Y))\&(Y \neq a)) \vee \exists Z(f(Z) = X)$. For $\sigma = \{Y/f(a)\}$, $\mathcal{E}$ is satisfiable in $\mathcal{H}$; σ is a solution to $\mathcal{E}$, which is solvable. □

[1] In general, L is fixed. When discussing complexity, it is useful to allow L to vary, with a program P and a query defining the nonlogical symbols of the language.

Definition 2 *Let p be an n-ary predicate symbol, $\mathbf{X} = X_1, \ldots, X_n$ be an n-tuple of variable symbols, 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 $[p(\mathbf{X}) \leftarrow \mathcal{E}]$ by $[p(\mathbf{X}) \leftarrow \mathcal{E}] = \{p(\mathbf{X})\sigma \mid \mathcal{H} \models \mathcal{E}\sigma,\ \sigma$ is a solution of $\mathcal{E}\,\}$.*

Example 3. Suppose our language contains three constant symbols a, b, c and no function symbols. Then $p(X,Y) \leftarrow X \neq Y$ is a constrained atom, and $[p(X,Y) \leftarrow X \neq Y] \ = \ \{\ p(a,b), p(a,c), p(b,c), p(b,a), p(c,a), p(c,b)\ \}$. □

For convenience, we omit the constraint part $\mathcal{E}$ if it is true on $\mathcal{H}$.

Definition 3 *A* constrained interpretation *(c-interpretation, CI) is a set $\mathcal{CI}$ of constrained atoms. For a given CI $\mathcal{CI}$, 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 CI $\mathcal{CI}$ if $[p(\mathbf{X}) \leftarrow \mathcal{E}] \subseteq [\mathcal{CI}]$.

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

CIs are ordered by a relation $\leq$ as follows.

Definition 4 *For any CIs $\mathcal{CI}_1$ and $\mathcal{CI}_2$ define $\mathcal{CI}_1 \leq \mathcal{CI}_2$ iff $[\mathcal{CI}_1] \subseteq [\mathcal{CI}_2]$.*

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 CIs are treated semantically indiscernible. Therefore, we implicitly assume that constraints are standardized apart if needed.

Every finite CI is equivalent to a CI in which in all constraints are standardized apart and where addition each predicate p occurs in exactly one constrained atom; we call such CIs *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 CI can be easily transformed into normal form; thus, we often assume that finite CIs are in normal form. Moreover, we sometimes refer in the rest of this paper to a CI $\mathcal{CI}$ where, strictly speaking, the equivalence class $[\mathcal{CI}]_\sim$ of $\mathcal{CI}$ with respect to $\sim$ is meant.

A *constraint logic program* is a finite set of clauses

$$A \ \leftarrow \ \mathcal{E} \mid B_1 \& \cdots \& B_n \tag{1}$$

where $A, B_1, \ldots, B_n$ are atoms, and $A \leftarrow \mathcal{E}$ is a constrained atom.With each constraint logic program P, we associate a logic program P^* which contains all clauses $(A \leftarrow B_1 \& \cdots \& B_n)\sigma$ for each clause (1) from P where σ is a solution of

$\mathcal{E}$. A ground atom A is a logical consequence of P, if A is a logical consequence of the program P^*.

Given any constraint logic program P, we associate with P an operator W_P (usually also denoted S_P [7]) that maps CIs to CIs. In order to define this operator, we first define the notion of a *resolvent* of a clause w.r.t. a CI. 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 CI. 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 $1 \le i \le n$; if some p_i does not occur in $\mathcal{CI}$, then $res_P(C,\mathcal{CI})$ contains the single constraint atom $p(\mathbf{X}) \leftarrow$ *false* where *false* is any unsatisfiable constraint. Now W_P is defined as follows.

Definition 5 *For any constraint logic program P and CI $\mathcal{CI}$, let*
$$W_P(\mathcal{CI}) \;=\; \bigcup_{C\in P} res_P(C,\mathcal{CI}).$$

If P and $\mathcal{CI}$ 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 $\le$; hence, it has a least fixpoint, which we denote by $\mathsf{lfp}(W_P)$.

Proposition 1 *Let P be a constraint logic program. Then,*

1. *if $\mathcal{CI}_1 \le \mathcal{CI}_2$ then $W_P(\mathcal{CI}_1) \le W_P(\mathcal{CI}_2)$.*
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 $\le$, denoted by $\mathsf{lfp}(W_P)$.*
4. *a ground atom A is a logical consequence of P iff $A \in [\mathsf{lfp}(W_P)]$.* □

Example 5. Consider the program:

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

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

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

which has only linear size w.r.t. P. This representative is obtainable by applying the mentioned simplifications to W_P. However, as discussed later, we cannot always find a small (i.e., polynomial size) representative for $\mathsf{lfp}(W_P)$. □

Recall that a *normal logic program* 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] As the above clause may be rewritten into an equivalent constrained clause

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

where $\mathbf{X}$ are fresh variables, we sometimes assume the latter form when considering normal logic programs.

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

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

Hence $\mathsf{GL}(P, I)$ is a logic program free of **not**. Based $\mathsf{GL}(P, I)$, we associate with a normal logic program P an operator F_P as follows:

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

where the T-operator is the usual immediate consequence operator Then, I is a *stable model* of P [9] iff $F_P(I) = I$.

In [1] it was shown that F_P is anti-monotone, and that the *well-founded semantics* (WFS) of P [21] is captured as follows:

Proposition 2 [1] *Let P be a normal logic program, 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 $A \notin \mathsf{gfp}(F_P^2)$.* □

3 The Constraint-based Transformation

The new transformation, **CT**, takes as input a normal logic program P, a CI $\mathcal{CI}$, 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 $(A \leftarrow B_1 \,\&\, \ldots \,\&\, B_n)\sigma$.

Thus in effect, $\mathbf{CT}(P, \mathcal{CI})$ is a "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).

[2] Programs are allowed to be infinite for technical reasons, since we consider (partial) groundings of such programs. We focus on finite programs, however.

Moving towards a definition of **CT**, we define the elimination of negated atoms from clauses with respect to a CI. 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 CI $\mathcal{CI}$ is the set of clauses

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

where $p_1(\mathbf{X}_1) \leftarrow \mathcal{E}_1$ is the unique constrained atom in $\mathcal{CI}$ with p_1 in the head, and whose free variables are $\mathbf{X}_1\mathbf{Y}_1$; note that the implicit existential quantifiers on $\mathbf{Y}_1$ in $\mathcal{E}_1$ are turned into universals 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 $\mathbf{not}(D_1)$,..., $\mathbf{not}(D_m)$.

Definition 6 *Let P be a normal logic program and $\mathcal{CI}$ be a normal CI. The* constraint-based transformation *of P with respect to the normal c-interpretation $\mathcal{CI}$, $\mathbf{CT}(P, \mathcal{CI})$, is the set of clauses $\{\mathsf{ELIM}(C, \mathcal{CI}) \mid C \in P\}$.*

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 CIs $\mathcal{CI}$, we first normalize $\mathcal{CI}$ to a CI $\mathcal{CI}^*$ 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 6. 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 CI

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

Eliminating $\mathbf{not}(q(X_1, Y_1))$ from C w.r.t. $\mathcal{CI}$ 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 $\mathsf{ELIM}(C, \mathcal{CI})$, which is

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

Pushing through negation and 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

$$p(X_1, Y_1) \leftarrow (X_1 = Y_1) \& (X_1 \neq a) \& (X_1 \neq b). \tag{2}$$

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, that are precisely the clauses $(p(X_1,Y_1) \leftarrow)\sigma$ where σ is a solution for the constraint part of rule 2. As $\mathbf{CT}(P,\mathcal{CI})$ amounts to rule 2, the desiderata are satisfied. $\square$

In the rest of this paper, we will focus only on finite CIs, as our main goal of computing the nonground well-founded/stable semantics, is only feasible in the finite case anyway.

Theorem 1. *Let P be a normal logic program, $\mathcal{CI}$ be a CI. Then*

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 $(A \leftarrow B_1 \,\&\cdots\&\, B_n)\sigma$.* $\square$

As $\mathbf{CT}(P,\mathcal{CI})$ is negation free, 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 7 *Let P be a normal logic program. The operator FNG_P, which maps CIs to CIs, is defined by*

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

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

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 $FNG_P(\mathcal{CI}_2) \leq FNG_P(\mathcal{CI}_1)$.*
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.* $\square$

3.1 A Discussion of 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 5 showed such a program. There, however, 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. [23]; for a background on complexity classes, consult [8]). 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 in presence of a powerful oracle for PSPACE-complete problems; indeed, such an algorithm renders the problem in $\mathrm{P}^{\mathrm{PSPACE}}$, which collapses with PSPACE. Therefore, by the results below, even if deciding truth of constrained atoms in CIs and deciding equivalence of CIs are for free, the computation of $\mathsf{lfp}(W_P)$ must take more than polynomial time in general (under the assumptions of non-collapsing complexity classes).

Consider now deciding whether a constrained atom is true in a CI. It is known that equational reasoning on $\mathcal{H}$ is decidable [3, 14]. Hence, it is decidable whether a given a constraint atom A (not necessarily ground) is true in a given finite CI $\mathcal{CI}$. However, the complexity is tremendous; recent results on equational reasoning immediately imply that it is nonelementary [**?**]. For the function-free case, we have much lower complexity.

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

1. *PSPACE-complete, if L 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 $k \geq 0$;*
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/coNP*-hard, if L is fixed*[3] *and all constraints are existential.* $\Box$

Here Π^P_j are the problems solvable in coNP, if an oracle for problems in Σ^P_{j-1} is available, where $\Sigma^P_1 = \mathrm{NP}$, $\Sigma^P_2 = \mathrm{NP}^{\mathrm{NP}}$, etc (see [8] for definitions).

The complexity of deciding equivalence of two CIs is similar. Since deciding truth of a constrained atom in a CI is decidable, clearly so is equivalence of two given finite CIs.

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

1. *PSPACE-complete, if L is 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 $k \geq 0$; and*

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

2. *solvable in polynomial time with few (a constant number) of* NP*-oracle calls and both* NP*/coNP-hard, if L is fixed and all constraints are existential.* □

The results in Theorems 3–4 imply that important problems on CIs 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.

4 Constrained Non-Ground Stable Models

Definition 8 *A CI $\mathcal{CI}$ is a* constrained non-ground *(CNG) stable model of P if and only if $\mathcal{CI} \sim \mathsf{lfp}(W_{\mathbf{CT}(P,\mathcal{CI})}) = FNG_P(\mathcal{CI})$.*

Example 7. Consider the program

$$\begin{array}{lll} p(X) \leftarrow \mathbf{not}(q(X)), & r(X) \leftarrow p(X) & w(0,s(0)) \leftarrow \\ q(X) \leftarrow \mathbf{not}(p(X)), & r(X) \leftarrow q(X) & \end{array}$$

This program has uncountably many ground stable models. Each of them is a CNG-stable model. The CI $\{p(X) \leftarrow,\ r(X) \leftarrow,\ w(X,Y) \leftarrow X = 0 \,\&\, Y = s(0)\}$, 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 $w(0, s(0))$. □

Theorem 5. *Suppose P is a normal logic program. Then:*

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

4.1 Constrained Non-Ground Well-Founded Semantics

Definition 9 *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 $\mathsf{lfp}(FNG^2_P)$.*

(ii) *A* is false according to the CNG-WFS of P *iff the set of formulas*

$$\{\neg p(\mathbf{t}), \forall \mathbf{X}_1\mathbf{Y}_1.(\mathcal{E}_1 \rightarrow p(\mathbf{X}_1)), \ldots, \forall \mathbf{X}_i\mathbf{Y}_i.(\mathcal{E}_i \rightarrow p(\mathbf{X}_i)), \ldots\} \qquad (3)$$

is unsatisfiable in $\mathcal{H}$, where the $p(\mathbf{X}_i) \leftarrow \mathcal{E}_i$ are all constrained atoms in $\mathsf{gfp}(FNG^2_P)$ with p in the head and $\mathbf{X}_i\mathbf{Y}_i$ their free variables.

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

The above formulation has important implications for query processing from a normal logic program under the WFS. It suggests that at compile-time, $\mathsf{lfp}(FNG^2_P)$ and $\mathsf{gfp}(FNG^2_P)$ may be *pre-computed* and *stored*. This is feasible if for finite CIs; they can be normalized and simplified by transformations preserving equivalence. When a query $A = p(\mathbf{t})$ is posed at run-time, *equational reasoning on* $\mathcal{H}$ can be performed to check which of the three conditions (i.e. true, false, unknown) is satisfied. The following theorem shows that this is possible.

Theorem 6. *Let P be a normal logic program. Then, a (possibly nonground) 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 well-founded semantics of P.* $\square$

The complexity of query answering under CNG-WFS is similar to Theorem 3. However, there is a small subtlety: While in Theorem 3 the CI is arbitrary, here it must represent $\mathsf{lfp}(FNG^2_P)$ resp. $\mathsf{gfp}(FNG^2_P)$ of some logic program P.

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

As before, if all variables in the constraint parts are from the variables in the head, then the complexity decreases. In particular, 2. and 3. are then polynomial. Thus, if we represent $\mathsf{lfp}(FNG^2_P)$ and $\mathsf{gfp}(FNG^2_P)$ by CIs of this form, 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.

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 [19]. However, this generalization is not immediate, and is far more complex than the algorithm for the ground case. Additional machinery is needed in order to make it work.

5.1 Computing the CNG Well-Founded Semantics

In this section, we will develop 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 [6] approximates the well-founded semantics (cf. [19]). In our first step, we use a nonground version Φ_P of Fitting's operator [6] that iteratively computes not only an interpretation, but also simplifies the program P. The least fixpoint of this operator, $\overline{\mathsf{lfp}(\Phi_P)} = (T_{\text{fix}}, F_{\text{fix}})$ satisfies $[T_{\text{fix}}] \subseteq [\mathsf{lfp}(FNG_P^2)]$ and $[F_{\text{fix}}] \subseteq [\overline{\mathsf{gfp}(FNG_P^2)}]$, i.e., approximates the CNG-WFS.
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 to compute CNG-stable models.

ALGORITHM Compute_CNGWFS(P)

Input: A normal logic program P.

Output: T, F so that $[T] = [\mathsf{lfp}(FNG_P^2)]$, $[F] = [\overline{\mathsf{gfp}(FNG_P^2)}]$, simplification P^* of program P.

Step 1. Compute $\mathsf{lfp}(\Phi_P) = (T_{\text{fix}}, F_{\text{fix}})$ and set $(T_0, F_0) = (T_{\text{fix}}, F_{\text{fix}})$.
Step 2. $i = 0$; $P_0 = P$.
Step 3. For each rule R of the form $p(\mathbf{X}) \leftarrow \mathcal{E} \mid \mathit{Body}$ in P_i, do the following:

1. If $p(\mathbf{X}_1) \leftarrow \mathcal{E}_1 \in T_i$ and $\mathcal{H} \models \forall \mathbf{X}\mathbf{Y}\mathbf{X}_1 \exists \mathbf{Y}_1((\mathbf{X} = \mathbf{X}_1) \& \mathcal{E} \rightarrow \mathcal{E}_1)$, where $\mathbf{XY}$ resp. $\mathbf{X}_1\mathbf{Y}_1$ are the free variables in $\mathcal{E}$ resp. $\mathcal{E}_1$, then eliminate R from P_i.
2. If $p(\mathbf{X}_2) \leftarrow \mathcal{E}_2 \in F_i$ and $\mathcal{H} \models \forall \mathbf{X}\mathbf{Y}\mathbf{X}_2 \exists \mathbf{Y}_2((\mathbf{X} = \mathbf{X}_2) \& \mathcal{E} \rightarrow \mathcal{E}_2)$, where $\mathbf{XY}$ resp. $\mathbf{X}_2\mathbf{Y}_2$ are the free variables in $\mathcal{E}$ resp. $\mathcal{E}_2$, then eliminate R from P_i.
3. Otherwise, replace $\mathcal{E}$ in R by
$$\mathcal{E} \,\&\, ((\mathbf{X} = \mathbf{X}_1) \,\&\, \forall \mathbf{Y}_1 . \neg \mathcal{E}_1) \,\vee\, ((\mathbf{X} = \mathbf{X}_2) \,\&\, \forall \mathbf{Y}_2 . \neg \mathcal{E}_2).$$

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

Step 4.

$$\begin{aligned} F_{i+1} = F_i \,\cup\, &\{p(\mathbf{X}) \leftarrow \forall \mathbf{Y} \neg \mathcal{E} \mid p(\mathbf{X}) \leftarrow \mathcal{E} \in FNG_{P_{i+1}}(T_i)\} \\ \cup\, &\{p(\mathbf{X}) \leftarrow \mid p \text{ does not occur in } FNG_{P_{i+1}}(T_i)\,\} \end{aligned}$$

where $\mathbf{XY}$ are the free variables of $\mathcal{E}$ (and normalize F_{i+1}).

Step 5.

$$T_{i+1} = T_i \,\cup\, FNG_{P_{i+1}}(\{p(\mathbf{X}) \leftarrow \forall \mathbf{Y} \neg \mathcal{E} \mid p(\mathbf{X}) \leftarrow \mathcal{E} \in F_i\} \\ \cup \{p(\mathbf{X}) \leftarrow \mid p \text{ occurs not in } F_i\})$$

where $\mathbf{XY}$ are the free variables of $\mathcal{E}$ (and normalize T_{i+1}).

Step 6. If $[F_i] = [F_{i+1}]$ and $[T_i] = [T_{i+1}]$, then halt and return (T_i, 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, already 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 8. Consider the following logic program P, and assume a language $\mathcal{L}$ having two constants a, b:

$$\begin{array}{rll} p(a) \leftarrow & . & (1) \\ p(X) \leftarrow & p(Y) \,\&\, \mathbf{not}(q(X,Y)). & (2) \\ q(X,Y) \leftarrow & \mathbf{not}(p(X)). & (3) \end{array}$$

(b could be provided e.g. by an additional dummy rule, which we avoid for simplicity). Notice that P has two ground stable models: $M_1 = \{p(a), q(b,a), q(b,b)\}$ and $M_2 = \{p(a), p(b)\}$.

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

1. In Step 1 of the algorithm, $(T_{\text{fix}}, F_{\text{fix}})$ are computed to be:

$$T_{\text{fix}} = \{p(X) \leftarrow X = a, \quad q(X,Y) \leftarrow \mathit{false}\}.$$
$$F_{\text{fix}} = \{p(X) \leftarrow \mathit{false}, \quad q(X,Y) \leftarrow X = a\}.$$

   We set $T_0 = T_{\text{fix}}$ and $F_0 = F_{\text{fix}}$ 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, F_1 is set to F_0.
6. In Step 5, T_1 is set to T_0.
7. The algorithm halts in Step 6 because $T_1 = T_0$ and $F_1 = F_0$. It returns: $T_1 = \{p(X) \leftarrow X = a\}$ and $F_1 = \{q(X,Y) \leftarrow X = a\}$ and

$$P_1 = \quad \begin{array}{l} p(X) \leftarrow X \neq a \mid p(Y) \,\&\, \mathbf{not}(q(X,Y)).\ (2') \\ q(X,Y) \leftarrow X \neq a \mid \mathbf{not}(p(X))(3') \end{array}$$

□

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

5.2 Computing CNG-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 **Compute_CNGWFS** algorithm given above. This algorithm then outputs a triple (T_i, F_i, P_{i+1}). All atoms in $[T_i]$ (resp. $[F_i]$) are true (resp. false) in all stable models of P. We proceed by looking at atoms that are not in $[T_i] \cup [F_i]$ and adopt a branch and bound procedure that searches an (abstract) tree called BB-tree(P). Before defining how BB-tree(P) is constructed, we need some simple definitions.

Definition 10 *Let (T, F) be a consistent pair of CIs. Then the* unknown set *$U(T, F)$ generated by (T, F) is*

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

where $\mathbf{X}_1\mathbf{Y}_1$ (resp., $\mathbf{X}_2\mathbf{Y}_2$) are the free variables of $\mathcal{E}_1$ (resp., $\mathcal{E}_2$).

Example 9. Suppose we return to the program P of Example 8 and consider the sets T_1, F_1 returned as a consequence of the **Compute_CNGWFS** algorithm. In that case, after simplification, we have:

$$U(T_1, F_1) = \{p(X) \leftarrow X \neq a;\ q(X,Y) \leftarrow X \neq a\}. \qquad \square$$

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 11 *Let $C = A \leftarrow \mathcal{F} \mid Body$ be a clause, and let $p(\mathbf{X}) \leftarrow \mathcal{E}$ be a constraint atom with free variables $\mathbf{XY}$. 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 10. Suppose P is as discussed in Examples 8 and 9. Then the modification of clause (2') by assuming the constrained atom $CA = p(X') \leftarrow X' \neq a$ to be true yields

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

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 12 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 $A \leftarrow \Xi$ is a constraint, and $\Xi = \Xi_1 \vee \Xi_2$, then this constraint atom may be split into: $A \leftarrow \Xi_1 \,\&\, \Xi_2$ and $A \leftarrow \Xi_1 \,\&\, \neg\Xi_2$.

2. If $A \leftarrow \Xi$ is a constraint, and σ is a solution of Ξ, then $A \leftarrow \Xi \,\&\, \neg\sigma$ and $A \leftarrow \sigma$ represents a splitting.

Example 11. Suppose we return to our program P of examples 8 and 9, and consider the constrained atom $CA = p(X) \leftarrow X \neq a$ contained in $U(T_1, F_1)$. 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 (CA, CA). If there were additional constants, one possible way to split CA, following the second strategy from above, is into the following two constraints:

$$p(X) \leftarrow X = b.$$
$$p(X) \leftarrow X \neq a \,\&\, X \neq b.$$

Definition 13 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 14 *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, T, F, U), where P is a program, T and F are CIs describing true and false atoms, respectively, and $U = U(T, F)$ is the unknown set generated by them. N is a success node *if $[T] \cup [F] = B_L$, is a* failure node *if $[T] \cap [F] \neq \emptyset$, and an* open node *otherwise.*

1. *The root N_0 of BB-tree(P) is labeled with (P_0, T_0, F_0, U_0) where (T_0, F_0, P_0) is the result of* **Compute_CNGWFS**(P) *and $U_0 = U(T_0, F_0)$.*
2. *For each open node $N = (P_N, T_N, F_N, U_N)$, nondeterministically pick a constrained atom $A \leftarrow \mathcal{E}$ in U_N. The node N has children, $N_1, N_2, \ldots$ as follows:*

   Each child N_i has label (P_i, T_i, F_i, U_i), where P_i is a program and T_i, F_i, and 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 T_N, and let (T_i^*, F_i^*, P_i^*) be the result of **Compute_CNGWFS**(P_i^+). *Then,*

   - $P_i = P_i^*$;
   - $T_i = T_N \cup T_i^*$;
   - $F_i = F_N \cup F_i^*$;
   - $U_i = U(T_i, F_i)$.

   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* $SSS(k)(A \leftarrow \mathcal{E}) = (A \leftarrow \mathcal{E}_k, A \leftarrow F_k)$*, 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* $SSS(k)(A \leftarrow \mathcal{E}) = (A \leftarrow \mathcal{E}_k, A \leftarrow F_k)$. □

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 12. Suppose we return to program P of examples 8–11. The root of any BB-tree(P, SSS) is labeled with the quadruple $N_1 = (P_1, T_1, F_1, U_1)$ where P_1, T_1, F_1 are as described in the preceding examples, and $U_1 = U(T_1, F_1)$ as described in Example 9. Suppose we pick the constraint $p(X) \leftarrow X \neq a$ from 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. The program P_1^+ is after simplifications:

$$\begin{aligned} P_1^+ = \quad & p(X) \leftarrow X \neq a \,\&\, Y = a \mid p(Y) \,\&\, \mathbf{not}(q(X,Y)). \\ & q(X,Y) \leftarrow X \neq a \mid \mathbf{not}(p(X)). \\ & q(X,Y) \leftarrow X \neq a. \\ & p(X) \leftarrow X = a. \end{aligned}$$

(The last clause is from T_1). **Compute_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 $U_1' = \emptyset$; hence, N_1' is a success node.

The second child, N_2', is generated by assuming $p(X') \leftarrow afteX \neq a$ is true. Program P_2^+ is after simplifications (cf. Example 11):

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

After computing **Comp_CNGWFS**(P_2^+), we get $T_2 = \{p(X) \leftarrow true\}$ and $F_2 = \{q(X,Y) \leftarrow true\}$. 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 STABLE_COMP(P):

Input: A logic program P.

Output: The set S of all[4] CNG-stable models of P.

Step 1. Construct the root $N_0 = (P_0, T_0, F_0, U_0)$.
If $U_0 = emptyet$, then set $S = T_0$ and halt; otherwise, initialize list *Active* to N_0.
Step 2. $S = \emptyset$. (* Solution collection now empty *)
Step 3. Pick a node $N = (P_N, T_N, F_N, U_N)$ in list *Active*;
Step 4. Choose a constraint-atom $p(\mathbf{X}) \leftarrow \mathcal{E}$ in U_N;
Step 5. Execute **children**(N, CA, k), and let $N_1 = (P_1, T_1, F_1, U_1)$, $N_2 = (P_2, T_2, F_2, U_2)$ 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 $[T_j] \cup [F_j] = B_L$ and $[T_j] \cap [F_j] = \emptyset$, then label N_j as a *success* node and insert T_j into S.
2. If $[T_j] \cap [F_j] \neq \emptyset$, 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 non-deterministic, 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 CIs. Different such strategies are selectable. Notice that in a previous version of this paper, a *fixed* randomized constraint splitting strategy was applied.

Example 13. 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

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

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 $\mathcal{I}_2 = \{p(X) \leftarrow \mathit{true}\}$. 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,* **STABLE_COMP**(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 **STABLE_COMP**(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 [19] 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 $N = (P, T, F, U)$ and $N' = (P', T', F', U')$ satisfy $[T] \subseteq [T']$ and $[F] \subseteq [F']$, then only node N needs to be considered further; if $[T] \supset [\mathcal{CI}]$ (resp. $[F] \supset [\mathcal{CI}]$) 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. Table 1 summarizes this discussion, which focuses on stable models.

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.

Full Ground Pre-Computation. All stable models (or as many as are desired) are pre-computed at compile time, and stored as sets of *ground* atoms.

Pre-Computations of NG-stable Models. In [11], 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.

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

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

Table 1. Approaches to computing stable modelsx

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.* [11]. Though [11] 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. The work in [12] 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 [5].

Other related work is by McCain and Turner [16], 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. Work by Marek, Nerode and Remmel [15] considers constraint models that are related to CNG-stable models. However, the framework is different, and algorithms are not addressed.

Acknowledgment. The authors are obliged to Georg Gottlob for many useful discussions, which strongly influenced this paper. They further thank the referees for pointing out improvements and their constructive criticism. In partiuclar, a referee pointed out that related work on computing non-ground well-founded semantics has been done by C. Damásio in his PhD thesis (1996, unpublished).
Subrahmanian's work was supported in part by the Army Research Office under Grants DAAH-04-95-10174 and DAAH-04-96-10297, by ARPA/Rome Labs contract F30602-93-C-0241 (ARPA Order Nr. A716), by an NSF Young Investigator award IRI-93-57756, NSF Grant No. IRI-9423967 and by the Army Research Laboratory under Cooperative Agreement No. DAAL01-96-2-0002.

References

1. C. Baral and V.S. Subrahmanian. *Dualities Between Alternative Semantics for Logic Programming and Non-Monotonic Reasoning*, *J. Automated Reasoning*, 10:399–420, 1993.
2. C. Bell, A. Nerode, R. Ng and V.S. Subrahmanian. (1994) *Computation and Implementation of Nonmonotonic Deductive Databases*, *JACM*, 41(6):1178–1215, 1994.
3. H. Comon, P. Lescanne. *Equational Problems and Disunification*, *J. Symbolic Computation*, 7:371–425, 1989.
4. J. Dix and M. Müller. *Implementing Semantics of Disjunctive Logic Programs Using Fringes and Abstract Properties*, Proc. LPNMR '93, (eds. L.-M. Pereira and A. Nerode), pp 43–59, 1993.
5. 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, eds, *Proc. WS Non-Monotonic Extensions of Logic Programming (at JICSLP '96)*, pp 143–160, 1996. CS-Report 17/96, Univ. Koblenz.
6. M.C. Fitting. *A Kripke–Kleene Semantics for Logic Programming*, *J. Logic Programming*, 4:295–312, 1985.
7. M. Gabbrielli, G. Levi. *Modeling Answer Constraints in Constraint Logic Programs*, Proc. ICLP, 1991, pp.238–251.
8. D. Johnson, *A Catalogue of Complexity Classes.* In: Handbook of TCS, 1990.
9. M. Gelfond and V. Lifschitz. *The Stable Model Semantics for Logic Programming*, in: Proc. 5th JICSLP, pp 1070–1080, 1998.
10. M. Gelfond and V. Lifschitz. *Classical Negation in Logic Programs and Disjunctive Databases*, *New Generation Computing*, 9:365–385, 1991.
11. G. Gottlob, S. Marcus, A. Nerode, G. Salzer, and V.S. Subrahmanian. *A Non-Ground Realization of the Stable and Well-Founded Semantics*, *Theoretical Computer Science*, 166:221–262, 1996.
12. V. Kagan, A. Nerode, and V. Subrahmanian. Computing Minimal Models by Partial Instantiation. *Theoretical Computer Science*, 155:15–177, 1996.
13. J.W. Lloyd. *Foundations of Logic Programming*, Springer Verlag, 1987.
14. M. Maher. *Complete Axiomatization of the algebra of finite, rational and infinite trees*, in Proc. 3rd IEEE LICS, 1988.
15. W. Marek, A. Nerode, J. Remmel. *On Logical Constraints in Logic Programming*, Proc. LPNMR '95 (eds. W. Marek, A. Nerode, and M. Truszczyński), LNCS 928, pp 44–56, 1995.
16. N. McCain and H. Turner. *Language Independence and Language Tolerance in Logic Programs*, Proc. ICLP, 1994.
17. T. Sato and F. Motoyoshi. *A Complete Top-down Interpreter for First Order Programs*, Proc. ILPS '91, pp 37–53. MIT Press, 1991.
18. P. Stuckey. *Constructive Negation for Constraint Logic Programming*, Proc. LICS '91, pp 328–339. IEEE Computer Science Press, 1991.
19. V.S. Subrahmanian, D. Nau and C. Vago. WFS + Branch and Bound = Stable Models, *IEEE TDKE*, 7(3):362–377, 1995.
20. D. Turi. *Extending S-Models to Logic Programs with Negation*, Proc. ICLP '91, pp 397–411, 1991.
21. A. van Gelder, K. Ross and J. Schlipf. *Well-founded Semantics for General Logic Programs*, *JACM*, 38(3):620–650, 1991.

22. A. van Gelder. *The Alternating Fixpoint of Logic Programs with Negation*, Proc. 8th ACM Symp. on Principles of Database Systems, pp 1–10.
23. M. Vardi. *The On the Complexity of Bounded-Variable Queries*, Proc. 14th ACM Symp. on Theory of Computing, San Francisco, pp. 137–146, 1982.
24. S. Vorobyov. An Improved Lower Bound for the Elementary Theories of Trees. In J. K. S. M. A. McRobbie, ed, *Proc. 13th Conference on Automated Deduction (CADE '96)*, LNCS 1104, pp. 275–287, 1996.
25. S. Vorobyov. Existential Theory of Term Algebras is in Quasi-Linear Non-Deterministic Time. Manuscript, February 1997.