ELIMINATING NEGATION FROM NORMAL LOGIC PROGRAMS

Kanchana Kanchanasut and Peter Stuckey

Key Center for Knowledge Based Systems,
Department of Computer Science, University of Melbourne,
Parkville 3052, Australia.

Abstract

In this paper, we propose a bottom-up partial evaluation of normal programs with a top-down expansion of negated atoms to obtain equivalent logic programs. A program $\mathbf{P}$ is transformed to $\mathbf{P}^\omega$ by a bottom-up computation on the positive component of $\mathbf{P}$ while the negative counterpart is left untouched. During this process, we collect all substitutions describing a partial answer set to all the positive atoms in the bodies of $\mathbf{P}$. The declarative semantics of $\mathbf{P}$ is given by the completion of $\mathbf{P}^\omega$. The completed predicate definitions in $\mathbf{P}^\omega$, if they do not contain local variables, can be used as a basis for expanding each negated atom in the bodies of $\mathbf{P}^\omega$. We show that for a class of programs where every negative subgoal can be expanded, the resultant program $\mathbf{P}'$ is a definite logic program with equality and disequality constraints. If the program falls outside this class, the resultant program may be executed using Chan's **SLD−CNF** resolution.

Our proposed scheme provides a sound and complete query answering system for a class of programs whose positive part has a finite $\mathbf{T}_\mathbf{P}^\omega$ and whose clauses satisfy the positive grounded property defined herein. With the bottom-up computation, all infinite positive loops are removed. With all the negated atoms expanded or eliminated at partial evaluation time, less work is required during the run-time query answering and problems with floundering are removed.

1 Introduction

In logic programming, when negation is introduced in queries, the meaning of a program is based upon the *Clark completion* [4] of the original program which basically turns implication signs into equivalence signs. **SLDNF** provides a sound and complete proof procedure for definite programs, but when negated atoms are allowed in the body of a program clause, the completeness of **SLDNF** is lost. In addition, for the case of 2-valued logic, the Clark completion **comp**($\mathbf{P}$) of a program $\mathbf{P}$ can be inconsistent even if $\mathbf{P}$ is consistent. **comp**($\mathbf{P}$) also has other drawbacks even without the presence of negated atoms in the program, as shown in ([14, 19]), where there are infinite looping **SLDNF−derivations** for $\mathbf{P}$.

Recent approaches to giving declarative semantics that capture the intended meaning of normal programs tend to adopt the *stable model semantics* [7] as the natural semantics for normal programs in the case of 2-valued logic. Other semantics with equivalent 2-valued models are the **fixcomp** model [6] and the **well−founded** model [19]. Ross [15] gave a procedural semantics for well-founded model semantics which is sound and complete for non-floundering programs in 3-valued logic.

We propose an operational model for a query-answering system for normal programs with respect to stable model semantics. It uses both the bottom-up and top-

down computational models; the bottom-up computation during a partial evaluation phase and a top-down **SLD–resolution** at run-time. The idea is to eliminate negated atoms so that **SLD** is sound and complete with improved run-time performance. We first apply a program transformation which evaluates all the positive atoms in the bodies of the program. Each positive atom in the body of the program gets expanded until there are no more positive atoms left on the right hand side. In other words, we generalize a $\mathbf{T_P}$ operator defined in [6] to non-ground programs. The transformed program $\mathbf{P}^\omega$ consists of clauses whose body consist only of equality constraints and negated atoms. We then take the Clark completion of $\mathbf{P}^\omega$ and use its negated predicate definitions to expand the negated subgoals in $\mathbf{P}^\omega$. There are negated subgoals whose negated definitions cannot be used for expansion due to the appearances of local variables. For cases where a program has all its negative atoms expanded, its transformed version will be in a form of a constraint logic program $\mathbf{P'}$ over the Herbrand universe with equality and disequality constraints and only positive atoms in the body.

Example 1.1 *Let* $\mathbf{P}$ *be*

$\mathbf{q(x)} \leftarrow \neg\mathbf{p(x)}$
$\mathbf{p(a)}$
$\mathbf{p(x)} \leftarrow \mathbf{s(y)} \wedge \neg\mathbf{r(y,x)}$
$\mathbf{s(b)}$
$\mathbf{r(b,c)}$

Given a query $\leftarrow\mathbf{q(x)}$*, Prolog will* **fail** *and* **SLDNF** *flounders.* $\mathbf{P}^\omega$ *is*

$\mathbf{q(x)} \leftarrow \neg\mathbf{p(x)}$
$\mathbf{p(a)}$
$\mathbf{p(x)} \leftarrow \neg\mathbf{r(b,x)}$
$\mathbf{s(b)}$
$\mathbf{r(b,c)}$

$\mathbf{comp}(\mathbf{P}^\omega)$ *is*

$\mathbf{q(x)} \leftrightarrow \neg\mathbf{p(x)}$
$\mathbf{p(x)} \leftrightarrow \mathbf{x} = \mathbf{a} \vee \neg\mathbf{r(b,x)}$
$\mathbf{s(x)} \leftrightarrow \mathbf{x} = \mathbf{b}.$
$\mathbf{r(z,x)} \leftrightarrow \mathbf{z} = \mathbf{b} \wedge \mathbf{x} = \mathbf{c}.$

Expanding all the negated atoms by their negated definitions, $\mathbf{P'}$ *becomes*

$\mathbf{q(x)} \leftarrow \mathbf{x} \neq \mathbf{a} \wedge \mathbf{r(b,x)}$
$\mathbf{p(a)}$
$\mathbf{p(x)} \leftarrow \mathbf{x} \neq \mathbf{c}$
$\mathbf{s(b)}$
$\mathbf{r(b,c)}$

Now if we ask $\leftarrow\mathbf{q(x)}$*, an* $\mathbf{SLD-derivation}$ *on* $\mathbf{P'}$ *will give us a* $'\mathbf{yes}'$ *answer with* $\mathbf{x} = \mathbf{c}$.

Our transformation preserves the stable model semantics of the program. Before describing the transformation, we give a brief review of the declarative semantics for normal programs in section 2. In section 3, we formally describe our transformation. We then describe a class of programs for which our system provides a sound and complete query evaluation procedure in section 4.

2 Semantics of Normal Programs

An early approach to understanding negation in normal logic programs is by the program completion introduced by Clark [4]. The meaning of a program is given by its completed definition which is simply a first order formula. The corresponding proof procedure for this scheme is the SLDNF-resolution which is SLD augmented by a non-monotonic rule called negation as failure. Problems related to this approach are: there are programs whose Clark completion is inconsistent and there are also programs with consistent Clark completion which have unintuitive models.

Another approach to the question of negation is to identify a model for which a program is "intended" to mean. This approach has lead to the introduction of classes of programs for which unique "intended" model exist, for example, *stratified* programs with unique iterated least models [1] and *locally stratified* programs with unique perfect models [13]. Stratified program are ones where recursion through negation is forbidden. Locally stratified programs are programs [13] whose stratification requirement is defined based on *priority relationship* on ground atoms instead of the predicate symbols as for the case of stratified programs. Przymusinski [13] has shown that by this generalization he has extended the class of stratified programs to include programs which are not stratified.

There are programs outside this class which have unique minimal models whose models can be obtained by the stable model semantics [7] and equivalently, the fixcomp semantics [6] and the 2-valued well-founded semantics [19]. A sufficient condition for programs to have unique stable model called *sufficiently stratified* is given in [6] which will be described in section 3.

In this paper we concentrate on programs with unique stable models.

3 Transformation of P to P′

Throughout this paper, a program means a *normal logic program* as defined in [10] where negative literals can occur in the body of any clause. We divide our tranformation into two phases: the bottom-up computation on positive atoms ($\mathbf{P}$ to $\mathbf{P}^\omega$) and the expansion of the negative atoms ($\mathbf{P}^\omega$ to $\mathbf{P}'$).

3.1 Positive Tranformation

Our tranformation from $\mathbf{P}$ to $\mathbf{P}^\omega$ is a generalization of a continuous operator $\mathbf{T_P}$ of [6] which operates on ground normal program clauses mapping one quasi-interpretation to another. The $\mathbf{T_P}$ operator in [6] is virtually identical to the $\mathbf{T_P}$ operator as defined for definite programs [18].

Instead of operating on ground clauses, our tranformation operates on the program clauses themselves. We will first reproduce the definitions of $\mathbf{T_P}$ from [6] together with its important properties.

Let $\mathbf{HB}(\mathbf{P})$ denote the Herbrand Base of program $\mathbf{P}$.

Definition. A *ground quasi-interpretation* [1] for $\mathbf{P}$ is a set of ground program clauses of the form $\mathbf{A} \leftarrow \neg\mathbf{B}_1, \ldots, \neg\mathbf{B_n}$, $\mathbf{n} \geq 0$, where $\mathbf{A}, \mathbf{B_i}$ are ground atoms in $\mathbf{HB}(\mathbf{P})$. ■

[1] In [6] a ground quasi-interpretation is called a quasi-interpretation.

The set of all ground quasi-interpretations for $\mathbf{P}$ is denoted by $\mathbf{GQI_P}$. It is clear that $\mathbf{GQI_P}$ is a complete lattice w.r.t. set inclusion.

Let $\mathbf{C}$ be the ground clause $\mathbf{A} \leftarrow \neg\mathbf{B}_1, \ldots, \neg\mathbf{B_n}, \mathbf{A}_1, \ldots, \mathbf{A_m}$ with $\mathbf{n} \geq 0, \mathbf{m} \geq 0$ and let $\mathbf{C_i}$ be ground clauses $\mathbf{A_i} \leftarrow \neg\mathbf{B}_{\mathbf{i}_1}, \ldots, \neg\mathbf{B}_{\mathbf{i_{n_i}}}$ with $1 \leq \mathbf{i} \leq \mathbf{m}$ and $\mathbf{n_i} \geq 0$. Then $\mathbf{T_C}(\mathbf{C}_1, \ldots, \mathbf{C_m})$ is the following clause

$$\mathbf{A} \leftarrow \neg\mathbf{B}_1, \ldots, \neg\mathbf{B_n}, \neg\mathbf{B}_{1_1}, \ldots, \neg\mathbf{B}_{1_{n_1}}, \ldots, \neg\mathbf{B}_{m_1}, \ldots, \neg\mathbf{B}_{m_{n_m}}.$$

We now introduce the transformation $\mathbf{T_P}$ on ground quasi-interpretations. Let $\mathbf{G_P}$ be the ground instances of clauses in $\mathbf{P}$.

$$\mathbf{T_P} : \mathbf{GQI_P} \rightarrow \mathbf{GQI_P}$$

$$\mathbf{T_P}(\mathbf{GQ}) = \{\mathbf{T_C}(\mathbf{C}_1, \ldots, \mathbf{C_m}) \mid \mathbf{C} \in \mathbf{G_P} \textbf{ and } \mathbf{C_i} \in \mathbf{GQ}, 1 \leq \mathbf{i} \leq \mathbf{m}\}$$

Theorem 3.1 $\mathbf{T_P}$ *is continuous [6].* ■

We define the *semantic kernel* [2] $\mathbf{SK}$ of $\mathbf{P}$ as follows:
Let $\mathbf{SK_n}(\mathbf{P}) = \mathbf{T_P^n}(\emptyset)$, and

$$\mathbf{SK}(\mathbf{P}) = \bigcup_{\mathbf{n} \geq 1} \mathbf{SK_n}(\mathbf{P}) \quad (\textbf{The least fixpoint of } \mathbf{T_P})$$

Let $\mathbf{p}$ be a predicate of $\mathbf{P}$ and $\{\mathbf{C}_1, \mathbf{C}_2, \ldots\}$ be the set of clauses in $\mathbf{SK}(\mathbf{P})$ whose heads are atoms with predicate symbol $\mathbf{p}$. $\mathbf{C_i}$ is a clause of the form $\mathbf{p}(\tilde{\mathbf{t}}) \leftarrow \neg\mathbf{B}_1, \ldots, \neg\mathbf{B_n}$. Then the *Clark completion of p* is

$$\forall(\mathbf{p}(\tilde{\mathbf{x}}) \leftrightarrow \mathbf{E}_1 \vee \ldots \vee \mathbf{E_m} \vee \ldots$$

where the right hand side is a (possibly infinite) disjunction. Every $\mathbf{E_i}$ is of the following form:

$$\tilde{\mathbf{x}} = \tilde{\mathbf{t}} \wedge \neg\mathbf{B}_1 \wedge \ldots \wedge \neg\mathbf{B_n}.$$

An infinite disjunction is **true** w.r.t. an interpretation if one or more of its elements is **true** w.r.t. this interpretation. Clark's completion of $\mathbf{SK}(\mathbf{P})$, which is called the *fixpoint completion* of $\mathbf{P}$, denoted by $\mathbf{fixcomp}(\mathbf{P})$, is a collection of the completed definitions of predicates of $\mathbf{P}$ together with Clark's equality theory.

Theorem 3.2 (Basic theorem) *[6]*

(a) *Every Herbrand model of* $\mathbf{P}$ *is a model of* $\mathbf{SK}(\mathbf{P})$.

(b) *Every Herbrand model of the fixpoint completion of* $\mathbf{P}$, $\mathbf{fixcomp}(\mathbf{P})$, *is a model of the Clark's completion of* $\mathbf{P}$, $\mathbf{comp}(\mathbf{P})$.

■

Note: In general, the reverse of part (b) of theorem 3.2, does not hold.

Other important properties of $\mathbf{SK}(\mathbf{P})$ that are discussed in [6] are the following:

[2] It is denoted as **LFP** in [6]

Definition. $\mathbf{P}$ is *sufficiently stratified* if the priority relation on the ground atoms of $\mathbf{SK}(\mathbf{P})$ is well-founded. That is to say $\mathbf{SK}(\mathbf{P})$ is locally stratified.

Theorem 3.3 *[6]*

(a) Every Herbrand model of $\mathbf{fixcomp}(\mathbf{P})$ *is a stable model of* $\mathbf{P}$ *and vice versa.*

(b) If $\mathbf{P}$ *is sufficiently stratified then there exists a unique stable model for* $\mathbf{P}$*.*

■

We now define a tranformation $\mathcal{T}_{\mathbf{P}}$ which is a generalization of $\mathbf{T_P}$ to non-ground atoms. This bears some similarity to the fixpoint operator of the non-ground semantics for definite programs [5]. We extend the definition of ground quasi-interpretations to a general case as follows.

Definition. A *negative clause NC* is a clause of the form

$$\mathbf{A} \leftarrow \neg\mathbf{B}_1, \ldots, \neg\mathbf{B_n}$$

Let $[\![\mathbf{NC}]\!]$ be the set of ground instances $\mathbf{NC}\,\theta$ of a negative clause $\mathbf{NC}$

$$\mathbf{A}\theta \leftarrow \neg\mathbf{B}_1\theta, \ldots, \neg\mathbf{B_n}\theta$$

■

We extend the $[\![\ ]\!]$ notation to sets of negative clauses in the obvious manner.

Definition. A *quasi-interpretation* for $\mathbf{P}$ is a set of negative clauses over the alphabet of $\mathbf{P}$. ■

Let the set of all quasi-interpretations for $\mathbf{P}$ be denoted by $\mathbf{QI_P}$. We define the following relation $\preceq$ on $\mathbf{QI_P}$:

Definition. $\mathbf{A} \preceq \mathbf{B}$ where $\mathbf{A}$ and $\mathbf{B} \in \mathbf{QI_P}$ iff $[\![\mathbf{A}]\!] \subseteq [\![\mathbf{B}]\!]$. ■

Clearly, $\mathbf{A} \preceq \mathbf{B}$ and $\mathbf{B} \preceq \mathbf{A}$ iff $[\![\mathbf{A}]\!] = [\![\mathbf{B}]\!]$. When this is the case, we say that $\mathbf{A}$ and $\mathbf{B}$ belong to the same equivalence class. It is easy to show the following

Lemma 3.1 *If* $\tilde{\mathbf{X}}$ *is a directed subset of* $\mathbf{QI_P}/_{=}$ *(set of equivalence classes of* $\mathbf{QI_P}$*), then* $[\![\mathbf{lub}\,\tilde{\mathbf{X}}]\!] = \mathbf{lub}[\![\tilde{\mathbf{X}}]\!]$. ■

Definition. Let γ: $\mathbf{QI_P}/_{=} \rightarrow \mathbf{GQI_P}$ be defined by $\gamma\,(\mathbf{S}) = [\![\mathbf{S}]\!]$. ■

Lemma 3.2 γ *is a bijection.* ■

Corollary 3.1 $\mathbf{QI_P}/_{=}$ *and* $\preceq$ *is a complete lattice.* ■

Our operator $\mathcal{T}_{\mathbf{P}}$ is defined on $\mathbf{QI_P}$ in the same manner as $\mathbf{T_P}$ on $\mathbf{GQI_P}$. Let $\mathbf{C}$ be a clause of $\mathbf{P}$ of the following form

$$\mathbf{A} \leftarrow \neg\mathbf{B}_1, \ldots, \neg\mathbf{B_n}, \mathbf{A}_1, \ldots, \mathbf{A_m}$$

with $\mathbf{n}$ and $\mathbf{m} \geq 0$.
Let $\mathbf{NC_i}$ be negative clauses of the following form

$$\mathbf{A'_i} \leftarrow \neg\mathbf{B}_{\mathbf{i}_1}, \ldots, \neg\mathbf{B}_{\mathbf{i}_{\mathbf{n_i}}}$$

with $1 \leq \mathbf{i} \leq \mathbf{m}$ and $\mathbf{n_i} \geq 0$.
Let θ be a most general substitution such that $\mathbf{A}_1\theta = \mathbf{A}'_1\theta, \ldots, \mathbf{A_m}\theta = \mathbf{A}'_{\mathbf{m}}\theta$.
If no such θ exists then $\mathcal{T}_{\mathbf{C}}(\mathbf{NC}_1, \ldots, \mathbf{NC_m})$ is the empty clause, otherwise $\mathcal{T}_{\mathbf{C}}(\mathbf{NC}_1, \ldots, \mathbf{NC_m})$ is the following clause

$$\mathbf{A}\theta \leftarrow \neg\mathbf{B}_1\theta, \ldots, \neg\mathbf{B_n}\theta, \neg\mathbf{B}_{1_1}\theta, \ldots, \neg\mathbf{B}_{1_{\mathbf{n}_1}}\theta, \ldots, \neg\mathbf{B}_{\mathbf{m}_1}\theta, \ldots, \neg\mathbf{B}_{\mathbf{m}_{\mathbf{n_m}}}\theta$$

We now formally define the transformation $\mathcal{T}_{\mathbf{P}}$ on quasi-interpretations

Definition. $\quad \mathcal{T}_{\mathbf{P}} : \mathbf{QI_P} \rightarrow \mathbf{QI_P}$

$$\begin{aligned}\mathcal{T}_{\mathbf{P}}(\mathbf{Q}) = \; & \{\mathcal{T}_{\mathbf{C}}(\mathbf{NC}_1, \ldots, \mathbf{NC_m}) \mid \textbf{where } \mathbf{C} \in \mathbf{P} \textbf{ and} \\ & \mathbf{NC_i}, 1 \leq \mathbf{i} \leq \mathbf{m} \\ & \textbf{are renamed apart copies of elements of } \mathbf{Q}\}\end{aligned}$$

■

Lemma 3.3 *For any quasi-interpretation* $\mathbf{I}$, $\mathbf{T_P}([\![\mathbf{I}]\!]) = [\![\mathcal{T}_{\mathbf{P}}(\mathbf{I})]\!]$.

Proof: Clearly, $[\![\mathcal{T}_{\mathbf{C}}(\mathbf{NC}_1, \ldots, \mathbf{NC_m})]\!] = \bigcup \mathbf{T}_{\mathbf{C}'}([\![\mathbf{NC}_1]\!], \ldots, [\![\mathbf{NC_m}]\!])$, where $\mathbf{C}'$ ranges over ground instances of $\mathbf{C}$. ■

Lemma 3.4 $\mathcal{T}_{\mathbf{P}}$ *is continuous on* $\mathbf{QI_P}/_{=}$. ■

Proof: For any directed subset $\tilde{\mathbf{X}}$ of $\mathbf{QI_P}/_{=}$,

$$\begin{aligned}[\![\mathbf{lub}\mathcal{T}_{\mathbf{P}}(\tilde{\mathbf{X}})]\!] &= \mathbf{lub}[\![\mathcal{T}_{\mathbf{P}}(\tilde{\mathbf{X}})]\!] \; (\textbf{bylemma } 3.1) \\ &= \mathbf{lub}(\mathbf{T_P}([\![\tilde{\mathbf{X}}]\!])) \; (\textbf{bylemma } 3.3) \\ &= \mathbf{T_P}(\mathbf{lub}([\![\tilde{\mathbf{X}}]\!])) \\ &= \mathbf{T_P}([\![\mathbf{lub}(\tilde{\mathbf{X}})]\!]) \\ &= [\![\mathcal{T}_{\mathbf{P}}(\mathbf{lub}(\tilde{\mathbf{X}}))]\!] \; (\textbf{bylemma } 3.3).\end{aligned}$$

Since γ is a bijection, $\mathcal{T}_{\mathbf{P}}(\mathbf{lub}(\tilde{\mathbf{X}})) = \mathbf{lub}(\mathcal{T}_{\mathbf{P}}(\tilde{\mathbf{X}}))$. ■

Since $\mathcal{T}_{\mathbf{P}}$ is continuous $\mathcal{T}^{\mathbf{k}}_{\mathbf{P}}(\emptyset)$ reaches a fixpoint at or before ω steps. Denote the least fixpoint of $\mathcal{T}^{\mathbf{k}}_{\mathbf{P}}(\emptyset)$ as $\mathbf{P}^{\omega}$. In general, $\mathbf{P}^{\omega}$ will contain infinitely many clauses, but if $\mathcal{T}^{\mathbf{k}}_{\mathbf{P}}(\emptyset)$ reaches its fixpoint in a finite number of steps then $\mathbf{P}^{\omega}$ is finite, since each step only produces finitely many negative clauses. From now on, we shall assume $\mathbf{P}^{\omega}$ is finite. Obviously Datalog programs, where no function symbols occur, have finite $\mathbf{P}^{\omega}$.

Lemma 3.5 *Every model of* $\mathbf{comp}(\mathbf{P}^\omega)$ *is a model of* $\mathbf{comp}(\mathbf{P})$.

Proof: Let $\mathbf{head}(\mathbf{C})$ denote the head of a clause $\mathbf{C}$ and $\mathbf{body}(\mathbf{C})$ denote its body. Let $\mathbf{I}$ be a model of $\mathbf{comp}(\mathbf{P}^\omega)$.

Consider a clause $\mathbf{C}' \in \mathbf{P}^\omega$ of the form

$$\mathbf{A}\theta \leftarrow \neg\mathbf{B}_1\theta, \ldots, \neg\mathbf{B_n}\theta, \neg\mathbf{B}'_{1_1}\theta, \ldots, \neg\mathbf{B}'_{1_{n_1}}\theta, \ldots, \neg\mathbf{B}'_{m_1}\theta, \ldots, \neg\mathbf{B}'_{m_{nm}}\theta,$$

where $\mathbf{C}' \in \mathcal{T}_\mathbf{C}(\mathbf{C}_1, \ldots, \mathbf{C_m})$ where $\mathbf{C} \in \mathbf{P}$ is of the form

$$\mathbf{A} \leftarrow \neg\mathbf{B}_1, \ldots, \neg\mathbf{B_n}, \mathbf{A}_1, \ldots, \mathbf{A_m},$$

and $\mathbf{C_i} \in \mathbf{P}^\omega$ is of the form,

$$\mathbf{A}'_\mathbf{i} \leftarrow \neg\mathbf{B}'_{1_i}, \ldots, \mathbf{B}'_{i_{n_i}}$$

and θ is the **mgu** of $\mathbf{A}_1 = \mathbf{A}'_1 \wedge \ldots \wedge \mathbf{A_m} = \mathbf{A}'_\mathbf{m}$.

Let α and γ be assignments of elements in the domain of $\mathbf{I}$ to variables.

(a) First we show that $\mathbf{I}$ models $\mathbf{P}$. Suppose $\mathbf{I} \models \mathbf{body}(\mathbf{C})\gamma$ then $\mathbf{I} \models \mathbf{A_i}\gamma$ and since $\mathbf{I} \models \mathbf{comp}(\mathbf{P}^\omega)$ there exists a clause $\mathbf{C_i} \in \mathbf{P}^\omega$ such that $\mathbf{I} \models \neg\mathbf{B_i}\gamma, \ldots, \neg\mathbf{B}_{i_{n_i}}\gamma$. Thus

$$\mathbf{I} \models \neg\mathbf{B}_1\gamma, \ldots, \neg\mathbf{B_n}\gamma, \neg\mathbf{B}'_{1_1}\gamma, \ldots, \neg\mathbf{B}'_{1_{n_1}}\gamma, \ldots, \neg\mathbf{B}'_{m_1}\gamma, \ldots, \neg\mathbf{B}'_{m_{nm}}\gamma.$$

Since $\mathbf{I}$ models Clarks axioms, $\mathbf{A}_1 = \mathbf{A}'_1 \wedge \ldots \wedge \mathbf{A_m} = \mathbf{A}'_\mathbf{m}$ is unifiable, thus there exists $\mathbf{C}' \in \mathbf{T}_\mathbf{C}(\mathbf{C}_1, \ldots, \mathbf{C_m})$ as above and its instance α such that $\mathbf{body}(\mathbf{C})\gamma = \mathbf{body}(\mathbf{C}')\alpha$. Hence $\mathbf{I} \models \mathbf{A}\gamma = \mathbf{A}\theta\alpha$.

(b) Suppose $\mathbf{I} \models \mathbf{A}\gamma$ then since $\mathbf{I} \models \mathbf{comp}(\mathbf{P}^\omega)$ there exists $\mathbf{C}' \in \mathbf{P}^\omega$ as above and its instance α such that $\mathbf{A}\gamma = \mathbf{A}\theta\alpha$. Now $\mathbf{I} \models \mathbf{body}(\mathbf{C}')\alpha$ and since $\mathbf{I} \models \mathbf{C_i}$ and $\mathbf{I} \models (\neg\mathbf{B}_{i_1}\theta, \ldots, \neg\mathbf{B}_{i_{n_i}}\theta)\alpha$ then $\mathbf{I} \models \mathbf{A_i}\theta\alpha$. Hence $\mathbf{I} \models \mathbf{body}(\mathbf{C})\theta\alpha$ i.e. $\mathbf{I} \models \mathbf{body}(\mathbf{C})\gamma$.

■

$\mathbf{P}^\omega$ is merely a finite representation of $\mathbf{SK}(\mathbf{P})$, since $[\![\mathbf{P}^\omega]\!] = \mathbf{SK}(\mathbf{P})$, and hence its Clark completion yields the same set of Herbrand models as $\mathbf{SK}(\mathbf{P})$. It follows from theorem 3.3(a) that the Herbrand models of $\mathbf{comp}(\mathbf{P}^\omega)$ are equivalent to the stable models [7] of $\mathbf{P}$.

Lemma 3.6 *Every Herbrand model of* $\mathbf{comp}(\mathbf{P}^\omega)$ *is a stable model of* $\mathbf{P}$. ■

3.2 Negative Expansion

$\mathbf{P}^\omega$ consists of clauses which only have negated atoms in the body. With the aim of obtaining a definite program, we expand the negated atoms as follows.

Removing Redundant Atoms If the predicate symbol of a negated atom never occurs as the head of a clause in $\mathbf{P}^\omega$, then remove the negated atom from the clause in $\mathbf{P}^\omega$ in which it appears.

Completed Definitions We now take the Clark completion of $\mathbf{P}^\omega$. The completion of each predicate contains a set of equality constraints together with a set of conjunctive negative atoms. Let $\tilde{\mathbf{x}}$ be $\mathbf{n}$ new variables, where $\mathbf{n}$ is the arity of $\mathbf{p}$. For each clause $\mathbf{p}(\tilde{\mathbf{s}}_\mathbf{i}) \leftarrow \mathbf{B_i}$ in $\mathbf{P}^\omega$ let $\mathbf{E_i} \equiv \tilde{\mathbf{x}} = \tilde{\mathbf{s}}_\mathbf{i} \wedge \mathbf{B_i}$. For example, $\mathbf{p}(\mathbf{f}(\mathbf{f}(\mathbf{z}))) \leftarrow \neg\mathbf{q}(\mathbf{f}(\mathbf{z}))$ becomes $\mathbf{p}(\mathbf{x}) \leftarrow \mathbf{x} = \mathbf{f}(\mathbf{f}(\mathbf{z})) \wedge \neg\mathbf{q}(\mathbf{f}(\mathbf{z}))$.

The completed definition of $\mathbf{p}$ is then

$$\mathbf{p}(\tilde{\mathbf{x}}) \leftrightarrow \exists\, \mathbf{Y}_1\mathbf{E}_1 \vee \exists\, \mathbf{Y}_2\mathbf{E}_2 \vee \ldots \vee \exists\, \mathbf{Y_n}\mathbf{E_n}$$

where $\mathbf{Y_i}$ are the variables in $\tilde{\mathbf{x}} = \tilde{\mathbf{s}}_\mathbf{i} \wedge \mathbf{B_i}$ not in $\tilde{\mathbf{x}}$.

We call a variable $\mathbf{y} \in \mathbf{Y_i}$ a *local variable* [12] if it does not occur in $\tilde{\mathbf{x}} = \tilde{\mathbf{s}}_\mathbf{i}$. If no variables in each $\mathbf{E_i}$ of $\mathbf{p}$ are local, we say $\mathbf{p}$ has an *eligible definition* .

Example 3.1 *In* $\mathbf{P}^\omega$ *resulting from the following program,* $\mathbf{p}(\mathbf{x})$ *is not eligible.*

$$\mathbf{p}(\mathbf{x}) \leftarrow \mathbf{e}(\mathbf{x},\mathbf{y}) \wedge \neg\mathbf{q}(\mathbf{y})$$
$$\mathbf{e}(\mathbf{x},\mathbf{y}).$$

However, if the fact $\mathbf{e}(\mathbf{x},\mathbf{y})$ *is replaced by* $\mathbf{e}(\mathbf{x},\mathbf{f}(\mathbf{x}))$ *or* $\mathbf{e}(\mathbf{f}(\mathbf{x}),\mathbf{x})$*, for example, then the definition of* $\mathbf{p}$ *in* $\mathbf{comp}(\mathbf{P}^\omega)$ *becomes eligible.*

Expansion For each negated atom in $\mathbf{P}^\omega$ whose definition in $\mathbf{comp}(\mathbf{P}^\omega)$ is eligible, replace it by its negated definition. We describe how to negate a completed eligible definition in the following section. The remaining negated atoms whose definitions are not eligible are left to be expanded during run-time by constructive negation.

Redundancy Elimination After the transformation, clauses in the final program may be redundant such as $\mathbf{p}(\mathbf{x}) \leftarrow \mathbf{x} = \mathbf{b} \wedge \mathbf{x} \neq \mathbf{b}$ or they may have redundant constraints such as $\mathbf{x} = \mathbf{f}(\mathbf{y}) \wedge \forall\,\mathbf{z}\,\tilde{\mathbf{x}} \neq \mathbf{g}(\mathbf{z}) \wedge \forall\,\mathbf{t}\,\tilde{\mathbf{x}} \neq \mathbf{h}(\mathbf{t},\mathbf{t})$. These redundancies can be removed without affecting the semantics of the program.

3.3 Negating Predicate Definition

Sato and Tamaki [16] proposed a negation technique for tranforming definite programs to their dual programs. Their technique of taking the negation of a completed definition can be applied to programs with negated atoms (as Chan and Wallace did in [3]). For the following discussion, we assume that the clauses are eligible.

Suppose we have the following predicate definition where for

$$\begin{array}{lcl} \forall\,\tilde{\mathbf{x}}(\mathbf{p}(\tilde{\mathbf{x}}) & \leftrightarrow & \exists\,\tilde{\mathbf{y}_1}\tilde{\mathbf{x}} = \mathbf{t}(\tilde{\mathbf{y}_1}) \wedge \mathbf{Q}_1[\tilde{\mathbf{x}},\tilde{\mathbf{y}_1}] \\ & \vee & \ldots \\ & \vee & \exists\,\tilde{\mathbf{y}_n}\tilde{\mathbf{x}} = \mathbf{t}(\tilde{\mathbf{y}_n}) \wedge \mathbf{Q_n}[\tilde{\mathbf{x}},\tilde{\mathbf{y}_n}]) \end{array}$$

and $\mathbf{Q_i}[\tilde{\mathbf{x}},\tilde{\mathbf{y}_i}]$ is a conjunction of negated atoms containing variables in $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{y}_i}$. Then we have a special property of the Herbrand Universe following from the equality theory that

$$\neg(\exists\,\mathbf{y_i}\ \tilde{\mathbf{x}} = \mathbf{t}(\tilde{\mathbf{y}_i}) \wedge \mathbf{Q_i}[\tilde{\mathbf{x}},\tilde{\mathbf{y}_i}]) \leftrightarrow (\forall\,\mathbf{y_i}\ \tilde{\mathbf{x}} \neq \mathbf{t}(\tilde{\mathbf{y}_i})) \vee (\exists\,\tilde{\mathbf{y}_i}\ \tilde{\mathbf{x}} = \mathbf{t}(\tilde{\mathbf{y}_i}) \wedge \neg\mathbf{Q}[\tilde{\mathbf{x}},\tilde{\mathbf{y}_i}])$$

Applying the above property to the definition of $\neg\mathbf{p}(\tilde{\mathbf{x}})$,

$$\begin{array}{lcl} \forall\,\tilde{\mathbf{x}}(\neg\mathbf{p}(\tilde{\mathbf{x}}) & \leftrightarrow & (\forall\,\tilde{\mathbf{y}}, \tilde{\mathbf{x}} \neq \mathbf{t}(\tilde{\mathbf{y_1}})) \vee (\exists\,\tilde{\mathbf{y_1}}, \tilde{\mathbf{x}} = \mathbf{t}(\tilde{\mathbf{y_1}}) \wedge \neg\mathbf{Q_1}[\tilde{\mathbf{x}}, \tilde{\mathbf{y_1}}]) \\ & \wedge & \ldots \\ & \wedge & (\forall\,\tilde{\mathbf{y_n}}, \tilde{\mathbf{x}} \neq \mathbf{t}(\tilde{\mathbf{y_n}}) \vee (\exists\,\tilde{\mathbf{y_n}}, \tilde{\mathbf{x}} = \mathbf{t}(\tilde{\mathbf{y_n}}) \wedge \neg\mathbf{Q_n}[\tilde{\mathbf{x}}, \tilde{\mathbf{y_n}}])). \end{array}$$

We can expand all negated atoms using their negated definition above and reorganize the result of the expansion into clausal form again.

Example 3.2 *Let a completed definition of* $\mathbf{p}(\tilde{\mathbf{x}})$ *be*

$$\begin{array}{lcl} \mathbf{p}(\mathbf{x}) & \leftrightarrow & \mathbf{x} = \mathbf{f}(\mathbf{a}, \mathbf{a}) \wedge \neg\mathbf{e}(\mathbf{x}) \\ & \vee & \mathbf{x} = \mathbf{f}(\mathbf{a}, \mathbf{y}) \wedge \neg\mathbf{f}(\mathbf{y}) \\ & \vee & \mathbf{x} = \mathbf{a} \wedge \neg\mathbf{g}(\mathbf{x}) \end{array}$$

Its negated definition is

$$\begin{array}{lcl} \neg\mathbf{p}(\mathbf{x}) & \leftrightarrow & \mathbf{x} \neq \mathbf{f}(\mathbf{a}, \mathbf{a}) \vee (\mathbf{x} = \mathbf{f}(\mathbf{a}, \mathbf{a}) \wedge \mathbf{e}(\mathbf{x})) \\ & \wedge & \forall\,\mathbf{y}\,\mathbf{x} \neq \mathbf{f}(\mathbf{a}, \mathbf{y}) \vee \exists\,\mathbf{y}\,(\mathbf{x} = \mathbf{f}(\mathbf{a}, \mathbf{y}) \wedge \mathbf{f}(\mathbf{y})) \\ & \wedge & \mathbf{x} \neq \mathbf{a} \vee (\mathbf{x} = \mathbf{a} \wedge \mathbf{g}(\mathbf{x})) \end{array}$$

We show by example that when there exists a local variable in any of the negated predicates in $\mathbf{P}^\omega$, we cannot apply the above negation technique without losing clausal form.

Example 3.3 *Let* $\mathbf{P}^\omega$ *be*

$$\mathbf{p}(\mathbf{x}) \leftarrow \neg\mathbf{r}(\mathbf{x}, \mathbf{y}) \wedge \neg\mathbf{s}(\mathbf{x}, \mathbf{y})$$

Negation of definition of $\mathbf{p}$

$$\neg\mathbf{p}(\mathbf{x}) \leftrightarrow \forall\,\mathbf{y}\,(\mathbf{r}(\mathbf{x}, \mathbf{y}) \vee \mathbf{s}(\mathbf{x}, \mathbf{y}))$$

which means a non-clausal form will result from any negative expansion of $\neg\mathbf{p}(\mathbf{x})$.

If we allow our negated atoms to be expanded by ineligible definitions and use the transformation of Lloyd and Topor [11] for removing quantifiers from the final programs, we obtain clauses similar to those from which we started. Hence we leave negated atoms whose definition in $\mathbf{comp}(\mathbf{P}^\omega)$ is not eligible to be expanded during run-time by constructive negation.

3.4 Correctness

As $\mathbf{P}^\omega$ is a non-ground semantic kernel of $\mathbf{P}$, its Clark completion gives the declarative semantics for $\mathbf{P}$. Since the negative expansion simply replaces atoms by there equivalents in the Clark completion of $\mathbf{P}^\omega$, we have the following theorem.

Theorem 3.4 *If the Clark completion of* $\mathbf{P}^\omega$ *is consistent, the Herbrand models of* $\mathbf{comp}(\mathbf{P}'), \mathbf{comp}(\mathbf{P}^\omega)$ *and* $\mathbf{fixcomp}(\mathbf{P})$ *are identical.*

$$\mathbf{comp}(\mathbf{P}') =_{\mathbf{H}} \mathbf{comp}(\mathbf{P}^\omega) =_{\mathbf{H}} \mathbf{fixcomp}(\mathbf{P}).$$

■

Corollary 3.2 *If the Clark completion of* $\mathbf{P}^\omega$ *is consistent, the Herbrand models of* $\mathbf{comp}(\mathbf{P}')$ *are the stable models of* $\mathbf{P}$. ■

For definite programs $\mathbf{P}'$, SLD-resolution (for constraint logic programs involving equalities and disequalities over the Herbrand Universe) is sound and complete with respect to $\mathbf{comp}(\mathbf{P}')$ for success and finite failure due to the following result of Jaffar and Lassez.

Theorem 3.5 *[8]*
$\mathbf{comp}(\mathbf{P}') \models \mathbf{G}$ *iff* $\mathbf{G} \in \mathbf{SS}(\mathbf{P}')$
$\mathbf{comp}(\mathbf{P}') \models \neg\mathbf{G}$ *iff* $\mathbf{G} \in \mathbf{FF}(\mathbf{P}')$
■

Unfortunately if we just consider the Herbrand models of $\mathbf{comp}(\mathbf{P}')$ SLD-resolution is, in general, not complete for finite failure, just as for definite logic programs [10]. The *canonical* programs [9] are those programs where the greatest fixpoint of the familiar $\mathbf{T_P}$ operator of Van Emden and Kowalski [18] is $\mathbf{T_P} \downarrow \omega$. Let $\mathbf{A} \models_\mathbf{H} \mathbf{B}$ be true if every *Herbrand* model of $\mathbf{A}$ models $\mathbf{B}$, then we have the following result from Jaffar and Lassez.

Theorem 3.6 *[8]*
$\mathbf{comp}(\mathbf{P}') \models_\mathbf{H} \mathbf{G}$ *iff* $\mathbf{G} \in \mathbf{SS}(\mathbf{P}')$
If $\mathbf{P}'$ *is canonical* $\mathbf{comp}(\mathbf{P}') \models_\mathbf{H} \neg\mathbf{G}$ *iff* $\mathbf{G} \in \mathbf{FF}(\mathbf{P})$. ■

Hence the canonical programs are exactly those programs for which SLD-resolution is complete with respect to finite failure. All practical programs are canonical, and it seems unlikely that $\mathbf{P}'$ will not be canonical. We can, however, construct examples where $\mathbf{P}'$ is not canonical.

Example 3.4 *The program* $\mathbf{P} = \mathbf{P}^\omega$ *is ineligible due to its first clause.*

$$\mathbf{p(a)} \leftarrow \neg\mathbf{q(x)}$$
$$\mathbf{q(x)} \leftarrow \neg\mathbf{r(x)}$$
$$\mathbf{r(f(x))} \leftarrow \neg\mathbf{q(x)}$$

$\mathbf{P}'$ *is*

$$\mathbf{p(a)} \leftarrow \mathbf{r(x)}$$
$$\mathbf{r(f(x))} \leftarrow \mathbf{r(x)}$$
$$\mathbf{q(x)} \leftarrow \forall\, \mathbf{z}\ \mathbf{x} \neq \mathbf{f(z)}$$
$$\mathbf{q(x)} \leftarrow \mathbf{x} = \mathbf{f(z)} \wedge \mathbf{q(z)}$$

The inelegibility of $\mathbf{P}$ *is unimportant since the first clause is not required to be negated. Now* $\mathbf{p(a)}$ *does not finitely fail, but there are no Herbrand models of* $\mathbf{P}'$ *that make* $\mathbf{p(a)}$ *true, hence* $\mathbf{P}'$ *is not canonical.*

We now show that for the class of programs which we are principally interested in our transformation always produces canonical programs.

Lemma 3.7 *If* $\mathbf{P}^\omega$ *is completely eligible then* $\mathbf{P}'$ *is canonical.*

Proof: If $\mathbf{P}^\omega$ is completely eligible then every variable in the body appears in the head. Replacing equality constraints in $\mathbf{P'}$ by substituting with there mgus leaves $\mathbf{P'}$ such that every (free) variable appearing in the body appears in the head. Let $\mathbf{GFF}$ be the set of atoms all of whose ground derivations are finitely failed. Now from [8] $\overline{\mathbf{GFF}} = \mathbf{gfp}(\mathbf{T_{P'}})$. We show that $\mathbf{FF} = \mathbf{GFF}$ for the programs under consideration.

Any derivation of a ground goal $\mathbf{G}$ in $\mathbf{P'}$ contains only ground goals because every variable in the body appears in the head. Hence any atom $\mathbf{A} \in \mathbf{FF}$ whose derivations are all finitely failed is also in $\mathbf{GFF}$ and vice versa.

Hence $\mathbf{gfp}(\mathbf{T_{P'}}) = \overline{\mathbf{GFF}} = \overline{\mathbf{FF}} = \mathbf{T_{P'}} \downarrow \omega$. Thus $\mathbf{P'}$ is canonical. ■

Theorem 3.7
For sufficiently stratified $\mathbf{P}$ *with finite and completely eligible* $\mathbf{P}^\omega$*,*

(a) SLD-resolution on $\mathbf{P'}$ is sound and complete for success.

(b) SLD-resolution on $\mathbf{P'}$ is sound and complete for finite failure,

with respect to the unique stable model of $\mathbf{P}$*.*

Proof: By Corollary 3.2 the unique Herbrand model of $\mathbf{comp}(\mathbf{P'})$ is the unique stable model of $\mathbf{P}$. Now $\mathbf{P'}$ contains only positive atoms and equality and disequality constraints, and is canonical by Lemma 3.7. Hence by Theorem 3.6 the result follows. ■

4 Restrictions

In this section we discuss conditions under which our approach faces difficulties and suggest how to overcome these problems.

4.1 Sufficiently Stratification

We wanted to consider only programs with unique stable model. A sufficient condition for this class of programs is the "sufficiently stratified" condition as defined by [6] which says that the priority relation on ground atoms of $\mathbf{P}^\omega$ is well-founded. This is a larger class than locally stratified as shown in example 4.1.

4.2 Eligibility

We can characterise those programs $\mathbf{P}$ whose positive expanded version $\mathbf{P}^\omega$ will contain only eligible clauses as follows.

Definition. Let $\mathbf{P}^+$ be the program $\mathbf{P}$ with negative literals removed. A program $\mathbf{P}$ is *positive grounded* if for each clause, $\mathbf{C}$ in $\mathbf{P}$

$$\mathbf{A} \leftarrow \mathbf{B_1}, \ldots, \mathbf{B_n}, \neg\mathbf{B_{n+1}}, \ldots, \neg\mathbf{B_m}$$

and each successful derivation $\leftarrow\mathbf{B_1}, \ldots, \mathbf{B_n}$ from $\mathbf{P}^+$ with answer substitution θ, then all variables in $\mathbf{C}\theta$ appear in $\mathbf{A}\theta$. ■

Positive groundness is not a syntactic condition. It is clear that if a program is positive ground it is *admissible* (see [10]). Sufficient conditions for positive groundness include *allowedness* (see [10]) and *mode correctness* [17].

Theorem 4.1 $\mathbf{P}$ *is positive grounded iff all clauses of* $\mathbf{P}^\omega$ *are eligible.*

Proof: The proof proceeds by induction. We show $\mathbf{P}$ is positive grounded for each $\mathbf{BF-derivation}$ [20] of length $\leq \mathbf{n}$ in $\mathbf{P}^+$ iff all clauses in $\mathcal{T}_{\mathbf{P}}^{\mathbf{n}+1}(\emptyset)$ are eligible. Concurrently we show $\mathbf{A} \in \mathcal{T}_{\mathbf{P}^+}^{\mathbf{n}}(\emptyset)$ for iff $\mathbf{A} \leftarrow \neg\tilde{\mathbf{B}} \in \mathcal{T}_{\mathbf{P}}^{\mathbf{n}}(\emptyset)$.

Base Case $\mathbf{n} = 0$. Trivial.

Induction step Take a clause, $\mathbf{C}$,

$$\mathbf{A} \leftarrow \mathbf{B}_1, \ldots, \mathbf{B_n}, \neg\mathbf{B}_{\mathbf{n}+1}, \ldots, \neg\mathbf{B_m}$$

in $\mathbf{P}$, the corresponding clause in $\mathbf{P}^+$ is

$$\mathbf{A} \leftarrow \mathbf{B}_1, \ldots, \mathbf{B_n}.$$

Let $\mathbf{B'_i} \in \mathcal{T}_{\mathbf{P}^+}^{\mathbf{n}}(\emptyset)$ and $\mathbf{B'_i} \leftarrow \tilde{\mathbf{B}}_\mathbf{i}$ be corresponding elements of $\mathcal{T}_{\mathbf{P}}^{\mathbf{n}}(\emptyset)$ and let $\mathbf{B_i}\theta = \mathbf{B'_i}\theta$ for $1 \leq \mathbf{i} \leq \mathbf{n}$. Now by induction hypothesis each $\mathbf{B'_i} \leftarrow \neg\tilde{\mathbf{B}}_\mathbf{i}$ is eligible hence all variables in $\neg\tilde{\mathbf{B}}_\mathbf{i}\theta$ appear in $\mathbf{B'_i}\theta = \mathbf{B_i}\theta$. Then $\mathbf{A}\theta \in \mathcal{T}_{\mathbf{P}^+}^{\mathbf{n}+1}(\emptyset)$ and similarly the clause $\mathbf{C'} \in \mathcal{T}_{\mathbf{C}}(\mathbf{B'}_1 \leftarrow \neg\tilde{\mathbf{B}}_1, \ldots, \mathbf{B_n} \leftarrow \neg\tilde{\mathbf{B}}_\mathbf{n})$,

$$\mathbf{A}\theta \leftarrow \neg\tilde{\mathbf{B}}_1\theta, \ldots, \neg\tilde{\mathbf{B}}_\mathbf{n}\theta, \neg\mathbf{B}_{\mathbf{n}+1}\theta, \ldots, \neg\mathbf{B_m}\theta$$

is in $\mathcal{T}_{\mathbf{P}}^{\mathbf{n}+1}(\emptyset)$. Let θ be an answer substitution for a BF-derivation of $\mathbf{B}_1, \ldots, \mathbf{B_n}$ of length $\leq \mathbf{n}$. θ is positive grounding for $\mathbf{C}$ iff,

- all variables in $\mathbf{C}\theta$ are in $\mathbf{A}\theta$, iff
- all variables in $\mathbf{B_i}\theta$ and $\neg\mathbf{B_i}\theta$ are in $\mathbf{A}\theta$, iff
- all variables in $\neg\tilde{\mathbf{B}}_\mathbf{i}\theta$ and $\neg\mathbf{B_i}\theta$ are in in $\mathbf{A}\theta$, iff
- $\mathbf{C'}$ is eligible.

■

Example 4.1 *The following program ,*$\mathbf{P}$*, [7] is not locally stratified but is sufficiently stratified and positive grounded.*

$\mathbf{p}(\mathbf{a}, \mathbf{b})$.
$\mathbf{q}(\mathbf{x}) \leftarrow \mathbf{p}(\mathbf{x}, \mathbf{y}) \wedge \neg\mathbf{q}(\mathbf{y})$.

$\mathbf{P}^\omega$ *becomes*

$\mathbf{p}(\mathbf{a}, \mathbf{b})$
$\mathbf{q}(\mathbf{a}) \leftarrow \neg\mathbf{q}(\mathbf{b})$.

Then $\mathbf{P'}$ *becomes*

$\mathbf{p}(\mathbf{a}, \mathbf{b})$
$\mathbf{q}(\mathbf{x}) \leftarrow \mathbf{x} = \mathbf{a} \wedge \mathbf{b} \neq \mathbf{a}$
$\mathbf{q}(\mathbf{x}) \leftarrow \mathbf{x} = \mathbf{a} \wedge \mathbf{q}(\mathbf{b})$.

For a program with ineligible clauses, it is still possible to eliminate all negated atoms if it satisfies the following property: $\mathbf{P}^\omega$ is finite and stratified. We can replace each remaining negated atom by its negated set of answer constraints. Note that if $\mathbf{P}$ is stratified we do not loose stratification in the positive transformation e.g. $\mathbf{p} \leftarrow \mathbf{q}, \mathbf{q} \leftarrow \neg\mathbf{r}$ becomes $\mathbf{p} \leftarrow \neg\mathbf{r}$.

Lemma 4.1 *If* $\mathbf{P}$ *is stratified,* $\mathbf{P}^\omega$ *is stratified.* ■

In fact, the stratification requirement for all of $\mathbf{P}^\omega$ is too strong. It is sufficient to impose the stratification condition on a module of $\mathbf{P}^\omega$ which is needed for a derivation of the negated atoms containing local variables only.

Example 4.2 *The first clause of the following program is ineligible but* $\mathbf{q}(\mathbf{x}, \mathbf{y})$ *has a finite set of answers.*

$\mathbf{p}(\mathbf{x}) \leftarrow \neg\mathbf{q}(\mathbf{x}, \mathbf{y})$
$\mathbf{q}(\mathbf{a}, \mathbf{a})$
$\mathbf{q}(\mathbf{a}, \mathbf{f}(\mathbf{x}))$

can be transformed to

$\mathbf{p}(\mathbf{x}) \leftarrow \mathbf{x} \neq \mathbf{a} \wedge \mathbf{y} \neq \mathbf{a}$
$\mathbf{p}(\mathbf{x}) \leftarrow \mathbf{x} \neq \mathbf{a} \wedge \forall\, \mathbf{z}\ \mathbf{y} \neq \mathbf{f}(\mathbf{z})$

That is $\mathbf{p}(\mathbf{x}) \leftrightarrow \mathbf{x} \neq \mathbf{a}$.

4.3 Finite Representation

We need to keep our intermediate program $\mathbf{P}^\omega$ and subsequently $\mathbf{P}'$ finite which implies a finite $\mathcal{T}_\mathbf{P}^\omega$. $\mathbf{P}^\omega$ may be infinite when we have positive recursion involving function symbols. However this includes most practical programs. In these cases, we can omit the bottom-up expansion of some positive literals to ensure a finite fixpoint of $\mathcal{T}_\mathbf{P}$. If $\mathbf{R}$ are the positive recursively defined predicates, then we treat $\mathbf{R}-$**atoms** exactly like negative literals in the bottom-up computation.

Definition. Let $\mathbf{P}^\mathbf{R}$ be the program obtained by removing negative literals and $\mathbf{R}-$**atoms** from the body of clauses in $\mathbf{P}$. $\mathbf{P}$ is positive ground w.r.t. $\mathbf{R}$ if for each clause $\mathbf{C}$ in $\mathbf{P}$

$$\mathbf{A} \leftarrow \mathbf{B}_1, \ldots, \mathbf{B_n}, \mathbf{R}_1, \ldots, \mathbf{R_m}, \neg\mathbf{B}_1, \ldots, \neg\mathbf{B_p}.$$

and every successful derivation of $\leftarrow\mathbf{B}_1, \ldots, \mathbf{B_n}$ in $\mathbf{P}^\mathbf{R}$ with answer substitutions θ, all variables in $\mathbf{C}\theta$ appear in $\mathbf{A}\theta$. ■

Since there is no recursion $\mathcal{T}_\mathbf{P}^\mathbf{k}(\emptyset)$ has a finite fixpoint. It follows from Thorem 4.1 that the resulting program $\mathbf{P}^\omega$ is completely eligible iff $\mathbf{P}$ is positive grounded w.r.t. $\mathbf{R}$.

Example 4.3 *The following program is positive grounded w.r.t. 'even(x)'.*

$\mathbf{p(x)} \leftarrow \mathbf{even(x)} \wedge \neg\mathbf{q(x)}$
$\mathbf{q(x)} \leftarrow \mathbf{t(x)} \wedge \neg\mathbf{r(x)}$
$\mathbf{t}(\mathbf{s}^4(\mathbf{y}))$
$\mathbf{r}(\mathbf{s}^6(0))$
$\mathbf{even}(0)$
$\mathbf{even}(\mathbf{s}^2(\mathbf{x})) \leftarrow \mathbf{even(x)}.$

$\mathbf{P}'$ *is*

$\mathbf{p(x)} \leftarrow \mathbf{even(x)} \wedge \forall\mathbf{y}\ \mathbf{x} \neq \mathbf{s}^4(\mathbf{y})$
$\mathbf{p(x)} \leftarrow \mathbf{even(x)} \wedge \mathbf{x} = \mathbf{s}^4(\mathbf{y}) \wedge \mathbf{r}(\mathbf{s}^4(\mathbf{y}))$
$\mathbf{q(x)} \leftarrow \mathbf{x} = \mathbf{s}^4(\mathbf{y}) \wedge \mathbf{s}^4(\mathbf{y}) \neq \mathbf{s}^6(0)$
$\mathbf{t}(\mathbf{s}^4(\mathbf{y}))$
$\mathbf{r}(\mathbf{s}^6(0))$
$\mathbf{even}(0)$
$\mathbf{even}(\mathbf{s}^2(\mathbf{x})) \leftarrow \mathbf{even(x)}.$

5 Discussion

5.1 Related Work

Chan and Wallace [3] proposed a treatment of negation during partial evaluation time by expanding negated subgoals or eliminating them in order to improve run-time efficiency. They applied Sato and Tamaki 's [16] negation technique and Chan [2] negated answers to eliminate negated atoms whenever possible. The remaining negated atoms are treated by constructive negation [2] technique during run-time. Both schemes use the Sato and Tamaki transformation which is restricted to eligible clauses. Because Chan and Wallace apply the transformation to the original program while we apply apply the transformation to $\mathbf{P}^\omega$, they are unable to eliminate all negations from positive grounded programs.

5.2 Conclusion

We have proposed a scheme which partially evaluates normal programs to obtain their semantic kernels. From this point, for positive grounded programs or programs with finite and stratified $\mathbf{P}^\omega$, we can expand the negated atoms to obtain a program which is free from negation. For programs which fall outside this class, we propose the application of a constructive negation technique for evaluation. The scheme can readily be extended to include disequality constraints in the original programs.

Acknowledgement : We would like to thank Phan Minh Dung, James Harland, Kotagiri Ramamohanarao and Harald Sondergaard for useful comments and discussions on earlier drafts.

References

[1] K. Apt, H. Blair and A. Walker. Towards a Theory of Declarative Knowledge. In J. Minker, editor, *Foundation of Deductive Database and Logic Programming*, pages 89-148. Morgan Kaufmann, 1988.

[2] D. Chan. Constructive Negation Based on the Completed Database. In R. D. Kowalski and K. Bowen, editors, *Proc Fifth Intl. Conference and Symposium on Logic Programming* , pages 111-125. MIT Press, 1988.

[3] D. Chan and M. Wallace. A Treatment of Negation During Partial Evaluation. In H.D. Abramson and M.H. Rogers, editors, *Meta-Programming in Logic Programming* , pages 299-318, MIT Press, 1989. Proceeding of the Meta88 Workshop, June 1988.

[4] K. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Databases*, pages 293–322. Plenum Press, 1978.

[5] M. Falaschi, G. Levi, M. Martelli and C. Palamidessi A New Declarative Semantics for Logic Languages. In R. D. Kowalski and K. Bowen, editors, *Proc Fifth Intl. Conference and Symposuim on Logic Programming* , pages 993 - 1005. MIT Press, 1988.

[6] P. M. Dung and K. Kanchanasut. A Fixpoint Approach to Declarative Semantics of Logic Programs. In E. L. Lusk and R. A. Overbeek, editors, *Proc of the North American Conference on Logic Programming* , pages 604-625. MIT Press, 1989.

[7] M. Gelfond and V. Lifschitz. The Stable Model Semantics for Logic Programs. In R. Kowalski and K. Bowen, editors, *Proceeding of the Fifth International Conference on Logic Programming*, pages 1070-1079. Seattle, Washington, 1988.

[8] J. Jaffar and J-L. Lassez. Constraint Logic Programming. In *Proceedings Fourteenth ACM Symposium on the Principles of Programming Languages* . pages 111-119. Jan 1987.

[9] J. Jaffar and P. Stuckey. Canonical Logic Programs. *Journal of Logic Programming* **3**. pages 142-155. 1986.

[10] J. W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, second edition 1987.

[11] J. W. Lloyd and R. W. Topor. Making Prolog More Expressive, *Journal of Logic Programming* **3**, 1984.

[12] L. Naish. Negation and Quatifiers in NU-Prolog. In *Proceedings Third International Conference on Logic Programming* , pages 73-77. July, 1985.

[13] T. C. Przymusinski. On the declarative semantics of deductive database and logic programming. In J. Minker, editor, *Foundation of Deductive Database and Logic Programming*, pages 193-216. Morgan Kaufmann, 1988.

[14] T. C. Przymusinski. Non-monotonic formalisms and logic programming. In G. Levi and M. Martelli, editors, *Proceeding of the Sixth Logic Programming Symposium* , pages 655-674. MIT Press, Mass, 1989.

[15] K. Ross. A Procedural Semantics for Well Founded Negation in Logic Programs. In *Proceeding of the Eighth ACM SICACT-SIGMOD-SIGART Symposium on Principles of Database Systems* , pages 22 - 33, 1989.

[16] T. Sato and H. Tamaki. Transformational Logic Program Synthesis. In *Proceedings of the International Conference on Fifth Generation Computer Systems* , pages 195-201. ICOT. 1984.

[17] Z. Somogyi. A System of Precise Modes for Logic Programs. In J-L Lassez, editor, *Proceeding of the Fourth International Conference in Logic Programming* , pages 769-785. MIT Press, 1987.

[18] M. van Emden and R. Kowalski. The semantics of logic as a programming language. *Journal of the ACM* **23** : 733–742, 1976.

[19] A. van Gelder, K. Ross and J.S. Schipf. Unfounded sets and well-founded semantics for general logic programs. In *Proceedings of the Seventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 221-230, 1988.

[20] D.A. Wolfram, M. J. Maher and J-L Lassez. A Unified Treatment of Resolution Strategies for Logic Programs. In S-A Tarnlund, editor, *Proceedings of the Second International Conference in Logic Programming* , pages 263-273, 1984.