A Finite Presentation Theorem for Approximating Logic Programs*

Nevin Heintze† and Joxan Jaffar‡

August 6, 1990

Abstract

In program analysis, a key notion used to approximate the meaning of a program is that of ignoring inter-variable dependencies. We formalize this notion in logic programming in order to define an approximation to the meaning of a program. The main result proves that this approximation is not only recursive, but that it can be finitely represented in the form of a cyclic term graph. This explicit representation can be used as a starting point for logic program analyzers.

*A preliminary version appears in the *Proceedings*, 17^{th} ACM Symposium on POPL.

†School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213-3890

‡IBM Thomas J. Watson Research Center, PO Box 218, Yorktown Heights, NY 10598

1 Introduction

The problem at hand is: given a logic program, obtain an approximation of its meaning, that is, obtain an approximation of its least model. The definition of the approximation should be declarative (so that results can be proved about soundness and relative completeness); the approximation should be accurate (so that it is meaningful in program analysis), and finally, the approximated meaning should be a recursive set.

The notion of approximation used in this paper is based upon ignoring dependencies between program variables. Such dependencies are represented in the standard meaning of a logic program. For example, suppose that the least model of a program implies that some atom, with variables X and Y, is *true* just in case (X, Y) is (a, b) or (b, a). Then there is an inter-variable dependency between X and Y because assigning a value to one restricts the possible assignments to the other. A contrasting situation is when the atom is *true* just in case (X, Y) is (a, a), (a, b), (b, a) or (b, b). Now, the standard meaning of a logic program is the least amongst the collection of all its models. Analogously, our approximate meaning of a program is the least model obtained by ignoring inter-variable dependencies in the program. Consider two independent approaches to formalize this definition. The first approach is based on an appropriate method of representing models; the second approach is based on an appropriate interpretation of rules in a program.

A model of a logic program can be characterized as the union of several sets, one corresponding to each rule, and each such set represents the contribution of the corresponding rule to the model. Specifically, the set $\mathcal{I}_R$ corresponding to a rule R is such that if $H \leftarrow Body$ is a ground instance of R where the atoms in $Body$ are in the model, then H is in $\mathcal{I}_R$. Traditionally, each such $\mathcal{I}_R$ is described by means of ground substitutions $\theta_1, \theta_2, \cdots$ so that $\mathcal{I}_R$ is $\{H\theta_1, H\theta_2, \cdots\}$ [1]. Now, if $\mathcal{I}_R$ does not contain information about inter-variable dependencies, it can be described in a simpler way: use just *one* mapping which takes each rule variable into a *set* of elements as opposed an individual element. Such a mapping, call it a *set-mapping*, induces a mapping of the head atom of the rule into a set of ground atoms in an obvious way. For example, if α is the mapping $[X \mapsto \{a, b\}, Y \mapsto \{c\}]$, then $p(X, Y)\alpha$ is $\{p(a, c), p(b, c)\}$[2]. Call the class of models characterizable in this manner the *set-based* models. We can now define that the approximate meaning of a logic program is simply its least set-based model.

[1] Strictly speaking, some models can not be represented in this way. However, they are not important here.

[2] For obtaining the empty set, there is a distinguished mapping α such that $t\alpha = \emptyset$ for any expression t.

Another approach to defining the approximate meaning of a program is to adapt the declarative reading of program rules. The standard reading of a rule $H \leftarrow Body$ can be phrased as:

> *if the Body atoms, in the context of mappings* θ_1, θ_2, $\cdots$, θ_n *are* true, *then the ground atoms* $H\theta_1$, $H\theta_2$, $\cdots$, $H\theta_n$ *are* true[3].

This reading embodies an inference which captures information about inter-variable dependencies; this information is represented in the mappings θ_1, θ_2, $\cdots$, θ_n. To ignore inter-variable dependencies, we collect together, for each variable X in the rule, the set S_X of possible values that X can take in the mappings. We then form a set-mapping α which maps each X into S_X, and apply this set-mapping to H. More formally, define α to be the set-mapping which maps X, for any variable X, into the set $\{X\theta_1, X\theta_2, \cdots, X\theta_n\}$. The declarative reading of a rule $H \leftarrow Body$ thus becomes:

> *if the Body atoms, in the context of mappings* θ_1, θ_2, $\cdots$, θ_n *are* true, *then the ground atoms in* $H\alpha$ *are* true.

It can be proved that the least interpretation of a program which is consistent with this reading is indeed a model. Hence this interpretation conservatively approximates the meaning of the program. It can also be proved that this approach of defining the approximate meaning of a program is equivalent to the previous approach. That is, for any program, the least interpretation consistent with the above declarative reading of its rules is precisely the least set-based model.

In the logic programming literature, the notion of program approximation appears in the area of types (where sets of elements are associated to program entities such as variables and arguments of predicate and function symbols) and in the area of program analysis (where information about both semantic and runtime properties is sought). See, for example, [10, 3, 1, 14, 13, 6]. Two main approaches are used. In one [10, 14], a closure operator similar to α above is used. The closure here is at the level of tuples of predicate and function symbol arguments. For example, applying this argument-wise closure to the atom $p(X, Y)$ in the context of the mappings $[X \mapsto f(a, b),\ Y \mapsto c]$ and $[X \mapsto f(b, a),\ Y \mapsto c]$, results in the set $\{p(f(a, b), c),\ p(f(b, a), c),\ p(f(a, a), c),\ p(f(b, b), c)\}$. Closure at the level of variables, on the other hand, gives $\{p(f(a, b), c), p(f(b, a), c)\}$ instead. It is shown

[3] In the literature, n is 1 rather than an arbitrary finite number.

in [6] that the approximate meaning of a program defined using argument-wise closure is not as accurate as that defined using variable-wise closure[4].

The other main approach to approximating the meaning of a program is based on abstract interpretation[5] which involves treating several values as one abstract value [3, 1, 13]. This abstraction is chosen so that an iterative fixpoint computation, with respect to the underlying abstract semantic function, terminates. In terms of accuracy, these kinds of approximations are not directly comparable to that defined in this paper. For example, an abstract interpretation algorithm can reason exactly on the class of DATALOG programs, whereas the least set-based model of a DATALOG program is, in general, an approximation to its meaning. Conversely, the least set-based model of a unary program is exactly its meaning, whereas no standard abstract interpretation algorithm can be exact over all unary programs. In other words, while abstract interpretation algorithms can capture some information about inter-variable dependency, they must embody another form of approximation[6].

In summary, the starting point is the notion of approximation which is based upon ignoring inter-variable dependencies. This basic notion is not only highly intuitive, but has been shown to be important for efficiency reasons [8]. For logic programs, we formalize this notion in the form of set-based models, or equivalently, in the form of models consistent with an appropriate limitation on the inferences that a program rule can make. Our main result proves that this single limitation is sufficient to guarantee that the approximate meaning of any program is a recursive set. An important corollary of this result is a new and significant subclass of logic programs whose least models are recursive. This subclass, in which our program approximation is exact, is the class of programs whose head atoms are linear (no two occurrences of the same variable), and whose body atoms can be partitioned into groups such that (a) no two groups share a variable and (b) each group mentions only one head variable.

The most important aspect of our result, however, is an algorithm which outputs an explicit representation of the approximate meaning. This representation can be useful for logic program analysis. For example, most analyzers need to reason about the success set of a program. Hence an explicit representation of this set can be used as on oracle. Another example is that the representation can be used to identify appropriate abstractions of the underlying domain. These abstractions can then be used by standard abstraction interpretation algorithms.

[4] Even so, it is open as to whether or not argument-wise closure gives rise to an approximate meaning which is recursive.

[5] Abstract interpretation is also widely used to infer *runtime* properties.

[6] For efficiency most of these algorithms, e.g. [2], also ignore inter-variable dependencies.

2 Preliminary Definitions

See [7] for standard terminology and definitions about pure logic programs P. Let Σ and Π denote the *functors*, that is, the (nonempty) alphabet of predicate and function symbols respectively; let HU and HB then denote the resulting Herbrand universe and base. Let VAR_t and VAR_a denote denumerable sets of variables (which pertain to terms and atoms respectively) and let VAR denote their union. A logic program P consists of a finite number of rules made up from the alphabet Σ, Π and VAR_t. Assume, without losing generality, that no two rules in P have the same variable. Given a logic program P, we write (a) Π_P, Σ_P and VAR_P to respectively denote the finite sets of predicate symbols, function symbols and variables in P, (b) HU_P and HB_P to respectively denote the Herbrand universe and base arising from Π_P and Σ_P, and (c) $P|_p$ denote those rules in P whose head atoms contain the predicate symbol p.

Where e is any expression, $var(e)$ denotes its collection of variables, and e is *ground* if $var(e) = \emptyset$. A *head-only* variable in a rule is one which does not appear in the body. Similarly for *body-only* variables.

A *substitution* θ is a mapping from VAR into atoms and terms such that $X\theta = X$ except for a finite number of variables. We write θ in the form $[X_1/t_1, X_2/t_2, \cdots, X_n/t_n]$, $n \geq 0$ to denote that θ maps X_i into t_i, $1 \leq i \leq n$, and that θ is the identity mapping elsewhere. A substitution can be extended to be a mapping from atoms and terms into atoms and terms in an obvious way, and we write $t\theta$ to denote the image of the atom or term t under the substitution θ.

A *set-substitution* α is a mapping from VAR into sets of ground atoms and ground terms such that $X\alpha = \emptyset$ except for a finite number of variables. We write α in the form $[X_1/S_1, X_2/S_2, \cdots\ X_n/S_n]$, $n \geq 0$ to denote that α maps X_i into the set S_i, $1 \leq i \leq n$, and that α maps all other variables X into $\emptyset$. A set-substitution α can also be extended to be a mapping from atoms and terms into sets of ground atoms and ground terms by means of the rule: for any functor f,

$$f(t_1, \cdots, t_n)\alpha \stackrel{\text{def}}{=} \{f(s_1, \cdots, s_n) : s_1 \in t_1\alpha, \cdots, s_n \in t_n\alpha\}\,.$$

An exception to this rule is a distinguished set-substitution $\mho$ which maps every atom or term, including ground atoms and terms, into $\emptyset$. (For example, if t is a ground term and α is a set substitution different from $\mho$, then by the above rules $t\alpha = \{t\}$; however $t\mho = \emptyset$.)

We use the symbol $\sim$ to group atoms or terms into a possibly empty sequence or

set, this choice being dictated by an obvious context. For example, we may write $A \longleftarrow \tilde{B}$ to denote a rule whose body atoms are represented by $\tilde{B}$. Writing $\tilde{B} \subseteq S$, on the other hand, means that each atom in $\tilde{B}$ appears in S.

Where f is any function from 2^{HB_P} into 2^{HB_P}, let $f{\uparrow}n$ denote $f^n(\emptyset)$ when $n \geq 0$, and let $f{\uparrow}\omega$ denote $\bigcup_{i \geq 0} f^i(\emptyset)$.

A *set expression* is constructed using given alphabets Π, Σ, VAR_t and VAR_a, and is defined to be either a *set expression over terms* or a *set expression over atoms*. A set expression over terms is one of the following:

- a special constant of the form $\bot$ or $\top$;
- a variable in VAR_t;
- $f(t_1, \ \ldots \ , \ t_n)$ where $f \in \Sigma$ is n-ary, $n \geq 0$, and $t_1, \cdots, t_n$ are set expressions over terms;
- $t \cup s$ or $t \cap s$ where t and s are set expressions over terms.

A set expression over atoms is one of the following:

- a variable in VAR_a;
- $p(t_1, \ \ldots \ , \ t_n)$ where $p \in \Pi$ is n-ary, $n \geq 0$, and $t_1, \cdots, t_n$ are set expressions over terms constructed using Σ and VAR_t;
- $t \cup s$ where t and s are set expressions over atoms.

A *set equation* is of the form $s = t$ where s and t are both set expressions over atoms, or they are both set expressions over terms. An *interpretation* is a mapping from VAR_a into 2^{HB}, and from VAR_t into 2^{HU}. An interpretation $\mathcal{I}$ induces, in an obvious way, a mapping from set expressions t into 2^{HB} or 2^{HU} by means of the rule: if t is:

- a variable, then $\mathcal{I}(t)$ is already defined;
- the special constant $\bot$, then $\mathcal{I}(t)$ is the empty set;
- the special constant $\top$ then $\mathcal{I}(t)$ is HU;

- a constant symbol c, then $\mathcal{I}(t)$ is $\{c\}$;
- $f(t_1, \ldots, t_n)$, where $n \geq 1$ and $f \in \Pi \cup \Sigma$, then $\mathcal{I}(t)$ is $\{f(s_1, \cdots, s_n) : s_1 \in \mathcal{I}(t_1), \cdots, s_n \in \mathcal{I}(t_n)\}$;
- $u \cup s$ or $u \cap s$, then $\mathcal{I}(t)$ is $\mathcal{I}(u) \cup \mathcal{I}(s)$ or $\mathcal{I}(u) \cap \mathcal{I}(s)$ respectively.

This mapping, in turn, induces a mapping from collections $\mathcal{C}$ of set equations into *true* or *false* in the obvious way. If $\mathcal{I}(\mathcal{C})$ is *true*, we say that $\mathcal{I}$ is a *model* of $\mathcal{C}$[7].

Given two interpretations $\mathcal{I}$ and $\mathcal{I}'$, the notation $\mathcal{I} \subseteq_{\mathcal{V}} \mathcal{I}'$ means $\mathcal{I}(X) \subseteq \mathcal{I}'(X)$ for all $X \in \mathcal{V}$, and the notation $\mathcal{I} \subseteq \mathcal{I}'$ abbreviates $\mathcal{I} \subseteq_{VAR} \mathcal{I}'$. We write $\mathcal{I} =_{\mathcal{V}} \mathcal{I}'$ to denote $\mathcal{I} \subseteq_{\mathcal{V}} \mathcal{I}'$ and $\mathcal{I}' \subseteq_{\mathcal{V}} \mathcal{I}$. Where $\mathcal{C}$ is a collection of set equations, $lm(\mathcal{C})$ denotes its least model, that is, $lm(\mathcal{C}) \subseteq \mathcal{I}$ for any model $\mathcal{I}$ of $\mathcal{C}$. Given two collections $\mathcal{C}$ and $\mathcal{C}'$ of set equations, the notation $\mathcal{C} \models \mathcal{C}'$ means that every model of $\mathcal{C}$ is a model of $\mathcal{C}'$. The notation $\mathcal{C} \underset{\mathcal{V}}{\Longleftrightarrow} \mathcal{C}'$ means that (a) for every model $\mathcal{I}$ of $\mathcal{C}$ there exists a model $\mathcal{I}'$ of $\mathcal{C}'$ such that $\mathcal{I} =_{\mathcal{V}} \mathcal{I}'$, and (b) for every model $\mathcal{I}'$ of $\mathcal{C}'$ there exists a model $\mathcal{I}$ of $\mathcal{C}$ such that $\mathcal{I} =_{\mathcal{V}} \mathcal{I}'$. Finally, we abbreviate $\underset{\mathcal{V}}{\Longleftrightarrow}$ by $\Longleftrightarrow$ when the set of variables in question is VAR.

3 The Approximate Meaning of a Logic Program

Define a *set-based interpretation* $\mathcal{I}$ of a logic program P to be a subset of HB_P which can be characterized as:

$$\bigcup_{1 \leq i \leq N} H_i \alpha_i$$

where each H_i, $1 \leq i \leq N$, is a head atom in P, and each α_i is a set-substitution (possibly $\mho$). $\mathcal{I}$ is a *set-based model* if for each ground instance $H \longleftarrow \tilde{B}$ of R_i, $1 \leq i \leq N$, H is in $H_i \alpha_i$ whenever $\tilde{B} \subseteq \mathcal{I}$.

Now, while the meaning of P is defined to be the smallest interpretation which models P, define the *approximate meaning* of P to be the smallest set-based model of P.

[7] The satisfiability problem for conjunctions of set equations is open. A decision procedure for a restricted class of equations appears in [5].

An equivalent definition of the exact meaning of P is that it is the least fixpoint of the following function which maps from and into interpretations of P:

$$T_P(\mathcal{I}) \stackrel{\text{def}}{=} \left\{ \begin{array}{ll} H\beta : & H \longleftarrow \tilde{B} \text{ is in } P, \text{ and} \\ & \beta \in \{\theta : \tilde{B}\theta \subseteq \mathcal{I}\} \end{array} \right\}$$

where θ ranges over ground substitutions on $var(P)$. Adapting this definition, one may obtain another definition of the approximate meaning of a program. Define an approximation operator $\mathcal{A}_{\mathcal{V}}$ which maps a nonempty collection Θ of ground substitutions on $\mathcal{V}$ into the set-substitution α where

$$X\alpha = \{X\theta : \theta \in \Theta\}$$

for all variables $X \in \mathcal{V}$. Further define that $\mathcal{A}_{\mathcal{V}}(\emptyset)$ is the distinguished set-substitution $\mho$ which maps every atom or term (including ground atoms and terms) into $\emptyset$. Now define

$$\tau_P(\mathcal{I}) \stackrel{\text{def}}{=} \left\{ a : a \in H\alpha \text{ where } \begin{array}{l} H \longleftarrow \tilde{B} \text{ is in } P, \text{ and} \\ \alpha = \mathcal{A}_{var(H)}(\{\theta : \tilde{B}\theta \subseteq \mathcal{I}\}) \end{array} \right\}$$

where θ ranges over ground substitutions on $var(P)$. Note that $\tau_P{\uparrow}\omega \supseteq T_P{\uparrow}\omega$ for any P, and so the approximation defined by τ_P is conservative.

Proposition 1 *For any logic program P, its least set-based model is $\tau_P{\uparrow}\omega$.* □

Proof: Let $R_1, \ldots, R_N$ comprise P, let R_i be of the form $H_i \longleftarrow \tilde{B}_i$, Let α_i, $1 \leq i \leq N$, denote set substitutions. Let $\mathcal{I} \subseteq HB_P$. The proposition follows from the fact that the least models of the following formulas are equivalent:

$$\exists \alpha_1 \ldots \alpha_N \left(\mathcal{I} = \bigcup_{1 \leq i \leq N} H_i \alpha_i \quad \text{and} \quad \bigwedge_{1 \leq i \leq N} H_i \alpha_i \supseteq \left\{ H_i \theta : \tilde{B}_i \theta \subseteq \mathcal{I} \right\} \right) \tag{3.1}$$

$$\exists \alpha_1 \ldots \alpha_N \left(\mathcal{I} = \bigcup_{1 \leq i \leq N} H_i \alpha_i \quad \text{and} \quad \bigwedge_{\substack{1 \leq i \leq N \\ X \in var(H_i)}} X \alpha_i \subseteq \left\{ X\theta : \tilde{B}_i \theta \subseteq \mathcal{I} \right\} \right) \tag{3.2}$$

$$\exists \alpha_1 \ldots \alpha_N \left(\mathcal{I} = \bigcup_{1 \leq i \leq N} H_i \alpha_i \quad \text{and} \quad \bigwedge_{\substack{1 \leq i \leq N \\ X \in var(H_i)}} X \alpha_i = \left\{ X\theta : \tilde{B}_i \theta \subseteq \mathcal{I} \right\} \right) \tag{3.3}$$

$$\exists \alpha_1 \ldots \alpha_N \left(\mathcal{I} = \bigcup_{1 \leq i \leq N} H_i \alpha_i \quad \text{and} \quad \bigwedge_{1 \leq i \leq N} \alpha_i = \mathcal{A}_{var(H_i)} \left(\left\{ \theta : \tilde{B}_i \theta \subseteq \mathcal{I} \right\} \right) \right) \tag{3.4}$$

$$\mathcal{I} = \tau_P(\mathcal{I}) \tag{3.5}$$

To show that (3.1) and (3.2) have the same least model, it suffices to prove that for any set substitution α, any set of substitutions Θ, and any atom or term t, that

$$t\alpha \supseteq \{ t\theta : \theta \in \Theta \} \text{ iff } \bigwedge_{X \in var(t)} X\alpha \supseteq \{ X\theta : \theta \in \Theta \}. \tag{3.6}$$

Clearly (3.6) holds in the case that Θ is empty; now suppose otherwise. Proceeding by induction on the structure of t, we first note that (3.6) holds trivially in case t is a constant or variable because both sides of the above equivalence reduce to *true*. If t is of the form $f(t_1, \ldots, t_n)$, then

$$\begin{aligned}
& f(t_1, \ldots, t_n)\alpha \supseteq \{ f(t_1, \ldots, t_n)\theta : \theta \in \Theta \} \\
\text{iff } & f(t_1\alpha, \ldots, t_n\alpha) \supseteq f(\{ t_1\theta : \theta \in \Theta \}, \ldots, \{ t_n\theta : \theta \in \Theta \}) \\
\text{iff } & t_j\alpha \supseteq \{ t_j\theta : \theta \in \Theta \}, \quad 1 \leq j \leq n \quad \text{(by definitions and } \Theta \neq \emptyset) \\
\text{iff } & \bigwedge_{X \in var(t_j)} X\alpha \supseteq \{ X\theta : \theta \in \Theta \}, \quad 1 \leq j \leq n \quad \text{(by induction hypothesis)} \\
\text{iff } & \bigwedge_{X \in var(t)} X\alpha \supseteq \{ X\theta : \theta \in \Theta \}
\end{aligned}$$

and so (3.6) is proved. This in turn proves that (3.1) and (3.2) have the same models, and hence they have the same least model.

To show that (3.2) and (3.3) have the same least model, it suffices to show that the least $\mathcal{I}$ satisfying (3.2) also satisfies (3.3). Let $\mathcal{I}$ be the least model of (3.2). Then there exist $\alpha_1, \ldots, \alpha_N$ such that $X\alpha_i \supseteq \left\{ X\theta : \tilde{B}_i \theta \subseteq \mathcal{I} \right\}$ $1 \leq i \leq N$,

$X \in var(H_i)$ and $\mathcal{I} = \bigcup_{1 \le i \le N} H_i\alpha_i$. Now define α_i', $1 \le i \le N$, by

$$X\alpha_i' = \left\{X\theta : \tilde{B}_i\theta \subseteq \mathcal{I}\right\} \; X \in var(H_i)$$

It follows that $H_i\alpha_i' \subseteq H_i\alpha_i$. Now define $\mathcal{I}'$ to be $\bigcup_{1 \le i \le N} H_i\alpha_i$. Clearly $\mathcal{I}' \subseteq \mathcal{I}$. Also $\mathcal{I}'$ satisfies (3.2), and since $\mathcal{I}$ is the least model of (3.2), this implies $\mathcal{I}' = \mathcal{I}$. This, in turn, implies that $\mathcal{I}$ satisfies (3.3), and hence that (3.2) and (3.3) have the same least model.

To show that (3.3) and (3.4) have the same least model, it suffices to prove the equivalence of the following for all $1 \le i \le N$ and all sets of substitutions Θ ground on $var(H_i)$:

$$\exists\alpha_i \left(\mathcal{I} \supseteq H_i\alpha_i \;\text{ and } \bigwedge_{X \in var(H_i)} X\alpha_i = \{X\theta : \theta \in \Theta\} \right);$$

$$\exists\alpha_i \left(\mathcal{I} \supseteq H_i\alpha_i \;\text{ and } \; \alpha_i = \mathcal{A}_{var(H_i)}(\Theta) \right).$$

In case Θ is empty, both these statements reduce to *true* (choose $\mho$ for α_i). Otherwise, $\alpha_i = \mathcal{A}_{var(H_i)}(\Theta)$ is, by definition of $\mathcal{A}_{var(H_i)}$, such that $X\alpha_i = \{X\theta : \theta \in \Theta\}$ for all $X \in var(H_i)$.

Finally, (3.4) and (3.5) are equivalent, by definition. Hence we have proved that (3.1), (3.2), (3.3), (3.4) and (3.5) all have the same least model. To complete the proof, note that the least set-based model of P is the least model of (3.1) be definition. This means that the least set-based model of P is the least fixpoint of $\mathcal{T}_P$, and since $\mathcal{T}_P$ is continuous (w.r.t. to $\subseteq$), this is $\mathcal{T}_P{\uparrow}\omega$. □

The approximation operator $\mathcal{A}$ is a closure operator over a collection of substitutions. In the literature, the notion of closure has been applied over sets of terms and atoms rather than sets of substitutions. This notion can be realized by the operator $\star$ defined as follows: for any set S of ground terms or atoms,

$$S^\star \stackrel{\text{def}}{=} \{c : c \text{ is a constant in } S\} \cup \{f(t_1, \ \ldots \ , \ t_n) : t_i \in (f_{(i)}^{-1}(S))^\star\}$$

where f ranges over the function and predicate symbols in S, and $f_{(i)}^{-1}(S) \stackrel{\text{def}}{=} \{t_i : f(t_1, \cdots, t_n) \in S\}$ denotes a projection of S. The work on types [14], for example, deals with the following function:

$$Y_P(\mathcal{I}) \stackrel{\text{def}}{=} \left\{ H\alpha : \begin{array}{l} H \longleftarrow \tilde{B} \text{ is in } P, \text{ and} \\ \alpha \in \{\theta : \tilde{B}\theta \subseteq \mathcal{I}\} \end{array} \right\}^\star$$

and so the approximate meaning of the program P is the least fixpoint $Y_P{\uparrow}\omega$ of Y_P. It is easy to see that Y_P is less accurate than $\mathcal{T}_P$. For example, the program P consisting of just two facts `p(a, b)` and `p(b, a)` is such that $Y_P{\uparrow}\omega = \{p(a,a), p(a,b), p(b,a), p(b,b)\} \supset T_P{\uparrow}\omega$ whereas $\mathcal{T}_P{\uparrow}\omega = T_P{\uparrow}\omega$. The decidability of $Y_P{\uparrow}\omega$ was left open.

The earlier work in [10] associates to a program, not a function, but a formula in which variables range over sets. The approximate meaning of a program is then given by a certain model of these formulas. It turns out that this model is essentially a fixpoint of Y_P. The accuracy of the program approximation that this model defines is therefore bounded by $Y_P{\uparrow}\omega$. In [10] and in a subsequent paper [11], algorithms were presented for solving subclasses of these set formulas. The solvability problem for the general class was left open.

We now conclude this section by illustrating the differences between Y_P, $\mathcal{T}_P$ and T_P on an example program P:

```
p(f(a, b)).
p(f(b, a)).
r(X) ⟵ p(f(X, X)).
s(f(Y, Z)) ⟵ p(f(Y, Z)).
```

Here $Y_P{\uparrow}\omega$ is $\left\{ \begin{array}{l} p(f(a,a)), p(f(a,b)), p(f(b,a)), p(f(b,b)) \\ r(a), r(b) \\ s(f(a,a)), s(f(a,b)), s(f(b,a)), s(f(b,b)) \end{array} \right\}$,

$\mathcal{T}_P{\uparrow}\omega$ is $\left\{ \begin{array}{l} p(f(a,b)), p(f(b,a)) \\ s(f(a,a)), s(f(a,b)), s(f(b,a)), s(f(b,b)) \end{array} \right\}$, and

$T_P{\uparrow}\omega$ is $\left\{ \begin{array}{l} p(f(a,b)), p(f(b,a)) \\ s(f(a,b)), s(f(b,a)) \end{array} \right\}$.

In general, we have $Y_P{\uparrow}\omega \supseteq \mathcal{T}_P{\uparrow}\omega \supseteq T_P{\uparrow}\omega$ for any program P.

In summary for this section, the approximate meaning of a program P is defined to be its least set-based model, or equivalently, the least fixpoint of $\mathcal{T}_P$. We argued that this formalizes the notion of ignoring inter-variable dependencies in logic programs, and further, that this approximation is more accurate than similar functions which have been used in logic program analysis. In what follows, the

fixpoint formulation of approximate meaning will be used as the basis for the main algorithm.

4 A Closed Form Representation using Set Formulas

Our first step toward a finite representation of $\mathcal{T}_P{\uparrow}\omega$ is a formula containing quantified set equations. This formula contains certain distinguished variables. An interpretation $\mathcal{I}$ of the formula can then be viewed as a subset of HB_P by taking the union of the $\mathcal{I}$-images of these distinguished variables. The essential property of the formula is that its models, thus viewed as subsets of HB_P, are precisely the fixpoints of $\mathcal{T}_P$.

This formula is comprised of two main parts. The first is a collection $\mathcal{C}_P$ of set equations, constructed from a given program P as follows. Let P contain the rules R_1, R_2, ..., R_N, $N \geq 1$. Associate to each predicate symbol $p \in \Pi_P$, a new variable Υ_p over atoms, and associate to each rule R_i a new variable ξ_i over atoms. $\mathcal{C}_P$ can now be defined as the union of equations of the form

$$\Upsilon_p = \begin{cases} \bigcup\limits_{i\,:\,R_i \,\in\, P|_p} \xi_i & \text{if } p \text{ is in a head atom in } P; \\ \bot & \text{otherwise;} \end{cases}$$

for all $p \in \Pi_P$. Let the distinguished variables of the form Υ_p be called the *type variables* corresponding to P.

The second part of the set formula makes a connection between (a) the variables ξ_i in $\mathcal{C}_P$ and the head atoms in P, and (b) the type variables in P and the body atoms in P. This set formula is a conjunction of several subformulas, one for each rule in P. Before introducing these subformulas, again let P be comprised of the rules R_1, ..., R_N, and let the i^{th} rule R_i be of the form $H \longleftarrow \tilde{B}$. Now associate to the rule R_i the following collection of pairs:

$$\begin{array}{l} \{(\xi_i, H)\} \ \bigcup \\ \left\{(p(\tilde{t}), \Upsilon_p) : p(\tilde{t}) \in \tilde{B}\right\} \ \bigcup \\ \{(X,\ \top_t) : X \text{ is a head-only variable in } R_i\}\,. \end{array}$$

The subformula corresponding to each rule R_i, call it $\mathcal{F}(\mathcal{D}_P^i)$, will be defined in terms of the pairs $\mathcal{D}_P^i$. Before we formally define $\mathcal{F}(\mathcal{D}_P^i)$, we consider some example

programs. Let P be

```
p(a).
p(b).
q(X) ←— p(X).
```

so that $\mathcal{C}_P$ is $\{\Upsilon_p = \xi_1 \cup \xi_2, \Upsilon_q = \xi_3\}$, $\mathcal{D}_P^1$ is $\{(\xi_1, p(a))\}$, $\mathcal{D}_P^2$ is $\{(\xi_2, p(b))\}$, and finally, $\mathcal{D}_P^3$ is $\{(\xi_3, q(X)), (p(X), \Upsilon_p)\}$. In this example, if it defined that a pair simply represents a set equation between its constituent parts, then

$$\mathcal{C}_P \wedge \mathcal{F}(\mathcal{D}_P^1) \wedge \mathcal{F}(\mathcal{D}_P^2) \wedge \mathcal{F}(\mathcal{D}_P^3)$$

represents the set equations

$$\begin{aligned} \Upsilon_p &= \xi_1 \cup \xi_2, \\ \Upsilon_q &= \xi_3, \\ \xi_1 &= p(a), \\ \xi_2 &= p(b), \\ \xi_3 &= q(X), \\ p(X) &= \Upsilon_p. \end{aligned}$$

These equations capture $\mathcal{T}_P{\uparrow}\omega$ in the sense that the images of the type variables Υ_p and Υ_q in the least model of the equations (in which $X = \{a, b\}$) sums to become $\mathcal{T}_P{\uparrow}\omega$. In general, however, obtaining a subformula $\mathcal{F}(\mathcal{D}_P^i)$ by simply by making the pairs $\mathcal{D}_P^i$ into set equations will not work. Suppose P were

```
p(a).
p(b).
q(a).
q(c).
r(X) ←— p(X), q(X).
```

Then $\mathcal{C}_P$ is $\{\Upsilon_p = \xi_1 \cup \xi_2, \Upsilon_q = \xi_3 \cup \xi_4, \Upsilon_r = \xi_5\}$,
$\mathcal{D}_P^1$ is $\{(\xi_1, p(a))\}$,
$\mathcal{D}_P^2$ is $\{(\xi_2, p(b))\}$,
$\mathcal{D}_P^3$ is $\{(\xi_3, q(a))\}$,
$\mathcal{D}_P^4$ is $\{(\xi_4, q(c))\}$, and finally,
$\mathcal{D}_P^5$ is $\{(\xi_5, r(X)), (p(X), \Upsilon_p), (q(X), \Upsilon_q)\}$.
Let the subformulas $\mathcal{F}(\mathcal{D}_P^1)$, $\mathcal{F}(\mathcal{D}_P^2)$, $\mathcal{F}(\mathcal{D}_P^3)$ and $\mathcal{F}(\mathcal{D}_P^4)$ be set equations as before. If $\mathcal{F}(\mathcal{D}_P^5)$ is also defined to be set equations, that is, $\{\xi_5 = r(X),\ p(X) = \Upsilon_p,\ q(X) = \Upsilon_q\}$, then the entire subformula is inconsistent. For this example, an appropriate formulation of $\mathcal{F}(\mathcal{D}_P^5)$ is

$$\begin{aligned}
\xi_5 &= \left\{ u : \exists x \left(\begin{array}{rcl} u & \in & r(X), \\ p(x) & \in & \Upsilon_p, \\ q(x) & \in & \Upsilon_q \end{array} \right) \right\} \\
X &= \left\{ x : \exists u \left(\begin{array}{rcl} u & \in & r(X), \\ p(x) & \in & \Upsilon_p, \\ q(x) & \in & \Upsilon_q \end{array} \right) \right\}
\end{aligned}$$

so that the least model of the subformula formula $\mathcal{C}_P \wedge \mathcal{F}(\mathcal{D}_P^1) \wedge \mathcal{F}(\mathcal{D}_P^2) \wedge \mathcal{F}(\mathcal{D}_P^3) \wedge \mathcal{F}(\mathcal{D}_P^4) \wedge \mathcal{F}(\mathcal{D}_P^5)$ (in which $X = \{a\}$) captures $\mathcal{T}_P{\uparrow}\omega$.

The above two examples only have one variable in the body atoms. Now suppose P were

```
p(a, b).
p(c, d).
q(X, Y) ⟵ p(X, Y).
```

then $\mathcal{C}_P$ is $\{\Upsilon_p = \xi_1 \cup \xi_2, \Upsilon_q = \xi_3\}$, $\mathcal{D}_P^1$ is $\{(\xi_1, p(a,b))\}$, $\mathcal{D}_P^2$ is $\{(\xi_2, p(c,d))\}$, and finally, $\mathcal{D}_P^3$ is $\{(\xi_3, q(X,Y)), (p(X,Y), \Upsilon_p)\}$. Let $\mathcal{F}(\mathcal{D}_P^1)$ and $\mathcal{F}(\mathcal{D}_P^2)$ be set equations as before. An appropriate formulation of $\mathcal{F}(\mathcal{D}_P^3)$ is a conjunction of three equations

$$\begin{aligned}
\xi_3 &= \left\{ u : \exists xy \left(\begin{array}{rcl} u & \in & q(X,Y), \\ p(x,y) & \in & \Upsilon_p \end{array} \right) \right\}, \\
X &= \left\{ x : \exists uy \left(\begin{array}{rcl} u & \in & q(X,Y), \\ p(x,y) & \in & \Upsilon_p \end{array} \right) \right\}, \\
Y &= \left\{ y : \exists ux \left(\begin{array}{rcl} u & \in & q(X,Y), \\ p(x,y) & \in & \Upsilon_p \end{array} \right) \right\}
\end{aligned}$$

so that the least model of our formula $\mathcal{C}_P \wedge \mathcal{F}(\mathcal{D}_P^1) \wedge \mathcal{F}(\mathcal{D}_P^2) \wedge \mathcal{F}(\mathcal{D}_P^3)$ (in which $X = \{a, c\}$ and $Y = \{b, d\}$) captures $\mathcal{T}_P{\uparrow}\omega$.

In general, the collection of pairs associated to a rule in the program is of the form $\{(a_1, sexp_1), \cdots, (a_m, sexp_m)\}$ where

- each a_j, for all $1 \leq j \leq m$, is an atom or term, and
- each $sexp_j$, for all $1 \leq j \leq m$, is a set expression.

Let $\mathcal{D}$ be such a collection of pairs $\{(a_1, sexp_1), \cdots, (a_m, sexp_m)\}$, and let $\mathcal{V} = \{X_1, \cdots, X_k\}$ denote the variables in the l.h.s. of $\mathcal{D}$. Now define $\mathcal{F}(\mathcal{D})$ to be

following conjunction of k equations:

$$\bigwedge_{1 \le j \le k} X_j = \left\{ x_j : \exists x_1 \cdots x_{j-1} x_{j+1} \cdots x_k \begin{pmatrix} a_1\mu & \in & sexp_1 \\ & \vdots & \\ a_m\mu & \in & sexp_m \end{pmatrix} \right\}$$

where $x_1, \ldots, x_k$ are new variables, and
μ denotes the renaming substitution $[X_1/x_1, \cdots, X_k/x_k]$.

We refer to equations appearing in $\mathcal{F}(\mathcal{D})$ as *quantified* equations. An *interpretation* and *model* of a collection of quantified equations are defined by extending, in the obvious way, the definitions given in section 2 for an interpretation and a model of set equations. Note that an interpretation of the r.h.s. of one of the above quantified equations is uniquely determined by an interpretation of the $sexp_i$, $1 \le i \le n$. In other words, where $\mathcal{D}$ is a collection of pairs as described above, an interpretation of the r.h.s. of $\mathcal{D}$ uniquely determines an interpretation of the r.h.s. of $\mathcal{F}(\mathcal{D})$.

Summarizing, we obtain from a program P a formula whose least model characterizes the desired set $\mathcal{T}_P{\uparrow}\omega$. Let $R_1, \ldots, R_N$ denote the rules in P. This formula is

$$\mathcal{C}_P \wedge \mathcal{F}(\mathcal{D}_P^1) \wedge \cdots \wedge \mathcal{F}(\mathcal{D}_P^N)$$

where $\mathcal{C}_P$ is

$$\bigwedge_{p \,\in\, \Pi_P} \Upsilon_p = \bigcup_{i : R_i \in P|_p} \xi_i$$

and each $\mathcal{D}_P^i$, $1 \le i \le N$, is a collection of pairs associated to the rule R_i. Let R_i be of the form $H_i \longleftarrow \tilde{B}_i$. Then $\mathcal{D}_P^i$ is

$$\{(\xi_i, H_i)\} \;\cup$$
$$\left\{(p(\tilde{t}), \Upsilon_p) : p(\tilde{t}) \in \tilde{B}_i\right\} \;\cup$$
$$\{(X,\ \top_t) : X \text{ is a head-only variable in } R_i\}.$$

For example, consider the program P:

```
p(a, b).
p(b, a).
q(X, W) ⟵ p(X, X).
r(Y, Z) ⟵ p(Y, Z).
```

The set formula corresponding to P is given by first obtaining $\mathcal{C}_P = \{\Upsilon_p = \xi_1 \cup \xi_2, \Upsilon_q = \xi_3, \Upsilon_r = \xi_4\}$, and then obtaining

$$\begin{aligned}
\mathcal{D}_P^1 &= \{(\xi_1, p(a,b))\}, \\
\mathcal{D}_P^2 &= \{(\xi_2, p(b,a))\}, \\
\mathcal{D}_P^3 &= \{(\xi_3, q(X,W)), (p(X,X), \Upsilon_p), (W, \top_a)\}, \\
\mathcal{D}_P^4 &= \{(\xi_4, r(Y,Z)), (p(Y,Z), \Upsilon_p)\}.
\end{aligned}$$

The set formula corresponding to P is $\mathcal{C}_P \wedge \mathcal{F}(\mathcal{D}_P^1) \wedge \ldots \wedge \mathcal{F}(\mathcal{D}_P^4)$, that is:

$$\begin{aligned}
&\Upsilon_p = \xi_1 \cup \xi_2, \\
&\Upsilon_q = \xi_3, \\
&\Upsilon_r = \xi_4, \\
&\xi_1 = \{u : u \in p(a,b)\}, \\
&\xi_2 = \{u : u \in p(b,a)\}, \\
&\xi_3 = \left\{ u : \exists xw \left(\begin{array}{rcl} u & \in & q(X,W), \\ p(x,x) & \in & \Upsilon_p \\ w & \in & \top_a \end{array} \right) \right\}, \\
&X = \left\{ x : \exists uw \left(\begin{array}{rcl} u & \in & q(X,W), \\ p(x,x) & \in & \Upsilon_p \\ w & \in & \top_a \end{array} \right) \right\}, \\
&W = \left\{ x : \exists ux \left(\begin{array}{rcl} u & \in & q(X,W), \\ p(x,x) & \in & \Upsilon_p \\ w & \in & \top_a \end{array} \right) \right\}, \\
&\xi_4 = \left\{ u : \exists yz \left(\begin{array}{rcl} u & \in & r(Y,Z), \\ p(y,z) & \in & \Upsilon_p \end{array} \right) \right\}, \\
&Y = \left\{ y : \exists uz \left(\begin{array}{rcl} u & \in & r(Y,Z), \\ p(y,z) & \in & \Upsilon_p \end{array} \right) \right\}, \\
&Z = \left\{ z : \exists uy \left(\begin{array}{rcl} u & \in & r(Y,Z), \\ p(y,z) & \in & \Upsilon_p \end{array} \right) \right\}.
\end{aligned}$$

The one and only model for this formula is given by

$$\begin{aligned}
\Upsilon_p &= \{p(a,b), p(b,a)\}, \\
\Upsilon_q &= \emptyset, \\
\Upsilon_r &= \{r(a,a), r(b,b), r(a,b), r(b,a)\}, \\
X &= \emptyset, \\
W &= \emptyset, \\
Y &= \{a,b\}, \\
Z &= \{a,b\}
\end{aligned}$$

and when this model is viewed as a subset of HB_P, we obtain

$$\{p(a,b), p(b,a), r(a,a), r(b,b), r(a,b), r(b,a)\}.$$

In what follows, we reserve the symbol N for the number of rules in the program P at hand, and we shall abbreviate the sequence $\mathcal{D}_P^1$, $\cdots$, $\mathcal{D}_P^N$ by $\tilde{\mathcal{D}}_P$, and the formula $\mathcal{F}(\mathcal{D}_P^1) \wedge \cdots \wedge \mathcal{F}(\mathcal{D}_P^N)$ by $\mathcal{F}(\tilde{\mathcal{D}}_P)$. We now prove that the models of the formula $\mathcal{C}_P \wedge \mathcal{F}(\tilde{\mathcal{D}}_P)$ correspond to the fixpoints of $\mathcal{T}_P$ in the following sense:

Main Lemma 1 *Let P be a logic program, and let $\mathcal{I} \subseteq HB_P$. Then $\mathcal{I}$ is a fixpoint of $\mathcal{T}_P$ iff there is model $\mathcal{I}'$ of $\mathcal{C}_P \wedge \mathcal{F}(\tilde{\mathcal{D}}_P)$ such that*

$$\mathcal{I} = \bigcup_{p \in \Pi_P} \mathcal{I}'(\Upsilon_p).$$

Proof: Let P contain the rules $R_1, R_2, \ldots, R_N$, and for each rule R_i, $1 \leq i \leq N$, let H_i denote its head atom and let $\tilde{B}_i$ denote its body atoms. In what follows, θ stands for a substitution which is ground on $var(P)$, and β stands for a substitution which is ground on $var(P) \cup \{\xi_1, \ldots, \xi_N\}$. Now consider the following chain of propositions:

1. $\mathcal{I} = \mathcal{T}_P(\mathcal{I})$
2. $\mathcal{I} = \left\{ a : a \in H\alpha \text{ where } \begin{array}{l} H \longleftarrow \tilde{B} \in P, \text{ and} \\ \alpha = \mathcal{A}_{var(H_i)}(\{\theta : \tilde{B}\theta \subseteq \mathcal{I}\}) \end{array} \right\}$
3. $\mathcal{I} = \bigcup_{1 \leq i \leq N} \left\{ a : a \in H_i\alpha \text{ where } \alpha = \mathcal{A}_{var(H_i)}(\{\theta : \tilde{B}_i\theta \subseteq \mathcal{I}\}) \right\}$
4. $\mathcal{I} = \bigcup_{1 \leq i \leq N} H_i(\mathcal{A}_{var(H_i)}(\{\theta : \tilde{B}_i\theta \subseteq \mathcal{I}\}))$
5. $\mathcal{I} = \bigcup_{1 \leq i \leq N} H_i\alpha_i$
   where α_i, $1 \leq i \leq N$, is $\mho$ in the case that $\neg\exists\theta \left(\tilde{B}_i\theta \subseteq \mathcal{I}\right)$; otherwise, α_i is the set-substitution such that $X\alpha_i = \{X\theta : \tilde{B}_i\theta \subseteq \mathcal{I}\}$ for all $X \in var(R_i)$.
6. $\bigwedge_{p \in \Pi_P} \mathcal{I}|_p = \bigcup_{R_i \in P|_p} H_i\alpha_i$
   where α_i, $1 \leq i \leq N$, is $\mho$ in the case that $\neg\exists\theta \left(\tilde{B}_i\theta \subseteq \mathcal{I}\right)$; otherwise, α_i is the set-substitution such that $X\alpha_i = \{X\theta : \tilde{B}_i\theta \subseteq \mathcal{I}\}$ for all $X \in var(R_i)$.

7.
$$\begin{cases} \bigwedge\limits_{p \in \Pi_P} & \Upsilon_p = \bigcup \left\{ H_i : R_i \in P|_p \text{ and } \exists\theta \left(\tilde{B}_i\theta \subseteq \bigcup\limits_{p \in \Pi_P} \Upsilon_p \right) \right\} \\ \bigwedge\limits_{\substack{1 \le i \le N \\ X \in var(R_i)}} & X = \left\{ X\theta : \tilde{B}_i\theta \subseteq \bigcup\limits_{p \in \Pi_P} \Upsilon_p \right\} \end{cases}$$

8.
$$\begin{cases} \bigwedge\limits_{p \in \Pi_P} & \Upsilon_p = \bigcup\limits_{i\,:\,R_i \in P|_p} \Upsilon_i \\ \bigwedge\limits_{1 \le i \le N} & \Upsilon_i = \left\{ t : t \in H_i \text{ and } \exists\theta \left(\tilde{B}_i\theta \subseteq \bigcup\limits_{p \in \Pi_P} \Upsilon_p \right) \right\} \\ \bigwedge\limits_{\substack{1 \le i \le N \\ X \in var(R_i)}} & X = \left\{ X\theta : \tilde{B}_i\theta \subseteq \bigcup\limits_{p \in \Pi_P} \Upsilon_p \right\} \end{cases}$$

9.
$$\begin{cases} \bigwedge\limits_{p \in \Pi_P} & \Upsilon_p = \bigcup\limits_{i\,:\,R_i \in P|_p} \Upsilon_i \\ \bigwedge\limits_{\substack{1 \le i \le N \\ W \in var(R_i) \cup \{\Upsilon_i\}}} & W = \left\{ W\beta : \begin{pmatrix} \Upsilon_i\beta & \in & H_i \\ \tilde{B}_i\beta & \subseteq & \bigcup\limits_{p \in \Pi_P} \Upsilon_p \end{pmatrix} \right\} \end{cases}$$

10.
$$\begin{cases} \bigwedge\limits_{p \in \Pi_P} & \Upsilon_p = \bigcup\limits_{i\,:\,R_i \in P|_p} \Upsilon_i \\ \bigwedge\limits_{\substack{1 \le i \le N \\ W \in var(R_i) \cup \{\Upsilon_i\}}} & W = \left\{ W\beta : \begin{pmatrix} \Upsilon_i\beta & \in & H_i \\ \bigwedge\limits_{q(\tilde{t}) \in \tilde{B}_i} q(\tilde{t})\beta & \in & \Upsilon_q \\ \bigwedge\limits_{Z \in HEADONLY_{R_i}} Z\beta & \in & \top_t \end{pmatrix} \right\} \end{cases}$$

11.
$$\begin{cases} \bigwedge\limits_{p \in \Pi_P} & \Upsilon_p = \bigcup\limits_{i\,:\,R_i \in P|_p} \Upsilon_i \\ \bigwedge\limits_{\substack{1 \le i \le N \\ W \in var(R_i) \cup \{\Upsilon_i\}}} & W = \left\{ W\beta : \bigwedge\limits_{(a,sexp) \in \mathcal{D}^i_P} a\beta \in sexp \right\} \end{cases}$$

12.
$$\begin{cases} \bigwedge\limits_{p \in \Pi_P} & \Upsilon_p = \bigcup\limits_{i\,:\,R_i \in P|_p} \Upsilon_i \\ \bigwedge\limits_{\substack{1 \le i \le N \\ W \in var(\tilde{R}_i) \cup \{\Upsilon_i\}}} & W = \left\{ x : \exists y_1 y_2 \cdots y_m \left(\bigwedge\limits_{(a,sexp) \in \mathcal{D}_P^i} a\mu \in sexp \right) \right\} \\ & \text{where } var(\tilde{R}_i) \cup \{\Upsilon_i\} = X, Y_1, \cdots, Y_m, \\ & x, y_1, \cdots, y_m \text{ are new variables, and} \\ & \mu \text{ denotes the renaming substitution} \\ & \quad [X/x, Y_1/y_1, \cdots, Y_m/y_m]. \end{cases}$$

13. $\mathcal{C}_P \wedge \mathcal{F}\left(\mathcal{D}_P^1\right) \wedge \cdots \wedge \mathcal{F}\left(\mathcal{D}_P^N\right)$.

The r.h.s. of proposition (2) is equal to the r.h.s. of (1) by the definition of $\mathcal{T}_P$. The r.h.s. of proposition (3) is a rewriting of the r.h.s. of (2) in such a way that the set generated by each individual rule is mentioned explicitly. That the r.h.s. of proposition (4) is equal to the r.h.s. of (3) is immediate. The r.h.s. of proposition (5) is equal to the r.h.s. of (4) because each set substitution α_i described in (5) is the same as the set substitution $\mathcal{A}_{var(H_i)}(\{\theta : \tilde{B}_i\theta \subseteq \mathcal{I}\})$ on the variables in R_i. Proposition (6) is merely a restatement of (5) in such a way that the sets associated to each predicate symbol in Π_P are mentioned individually. Combining these arguments, we have that propositions (1) through (6) are equivalent.

We now show that propositions (6) and (7) are equivalent in the sense that $\mathcal{I} \subseteq HB_P$ models (6) iff there is a model $\mathcal{I}'$ of (7) such that $\mathcal{I} = \bigcup\limits_{p \in \Pi_P} \mathcal{I}'(\Upsilon_p)$.

Let $\mathcal{I} \subseteq HB_P$ satisfy (6), and define an interpretation $\mathcal{I}'$ of (7) by first specifying that $\mathcal{I}'(\Upsilon_p) \stackrel{\text{def}}{=} \mathcal{I}|_p$ for all $p \in \Pi_P$, and then specifying that

$$\mathcal{I}'(X) \stackrel{\text{def}}{=} \left\{ X\theta : \tilde{B}_i\theta \subseteq \bigcup_{p \in \Pi_P} \mathcal{I}|_p \right\},\ 1 \le i \le N,\ X \in var(R_i). \tag{4.7}$$

Thus $\mathcal{I}'$ models the second conjunction in (7) by design. To see that $\mathcal{I}'$ also models the first conjunction in (7), consider

$$\begin{aligned}
\mathcal{I}'(\Upsilon_p) &= \mathcal{I}|_p \\
&= \bigcup_{R_i \in H|_p} H_i\alpha_i \\
&= \bigcup \left\{ \mathcal{I}'(H_i) : R_i \in P|_p \text{ and } \exists\theta \left(\tilde{B}_i\theta \subseteq \mathcal{I} \right) \right\} \\
&= \mathcal{I}' \left(\bigcup \left\{ H_i : R_i \in P|_p \text{ and } \exists\theta \left(\tilde{B}_i\theta \subseteq \mathcal{I} \right) \right\} \right). \\
&= \mathcal{I}' \left(\bigcup \left\{ H_i : R_i \in P|_p \text{ and } \exists\theta \left(\tilde{B}_i\theta \subseteq \bigcup_{p \in \Pi_P} \Upsilon_p \right) \right\} \right).
\end{aligned}$$

The first equation follows by design. The second equation follows from the fact that $\mathcal{I}$ satisfies (6). To prove the third equation, fix an i, $1 \leq i \leq N$. In the case that $\exists\theta \left(\tilde{B}_i\theta \subseteq \mathcal{I} \right)$, we have $X\alpha_i = \mathcal{I}'(X)$ for all $X \in var(R_i)$, and that α_i is different from $\mho$. Now, if a set substitution different from $\mho$ agrees with an interpretation on a set of variables, they both extend to the same mapping on atoms involving only these variables. Hence $H_i\alpha_i = \mathcal{I}'(H_i)$. In the case that $\neg\exists\theta \left(\tilde{B}_i\theta \subseteq \mathcal{I} \right)$, α_i is $\mho$, and there is no set corresponding to H_i in the r.h.s. In short, for every i, either there are equivalent sets in both sides of the third equation, or else $H_i\alpha_i$ is empty in the l.h.s. and there is no set for i in the r.h.s. The fourth equality follows from the general property that $\mathcal{I}(t_1) \cup \mathcal{I}(t_2) = \mathcal{I}(t_1 \cup t_2)$ for any interpretation $\mathcal{I}$ and any atoms t_1 and t_2. Finally, the fifth equation follows from the fact that

$$\mathcal{I}' \left(\bigcup_{p \in \Pi_P} \Upsilon_p \right) = \mathcal{I}.$$

Thus $\mathcal{I}'$ models the first conjunction in (7) and so $\mathcal{I}'$ models all of (7).

Conversely, suppose that $\mathcal{I}'$ is a model of (7). Let $\mathcal{I} \subseteq HB_P$ be defined as the union of $\mathcal{I}'(\Upsilon_p)$ over all $p \in \Pi_P$. Arguing like above, it can proved that

$$\begin{aligned}
\mathcal{I}|_p &= \mathcal{I}'(\Upsilon_p) \\
&= \mathcal{I}' \left(\bigcup \left\{ H_i : R_i \in P|_p \text{ and } \exists\theta \left(\tilde{B}_i\theta \subseteq \bigcup_{p \in \Pi_P} \Upsilon_p \right) \right\} \right) \\
&= \bigcup \left\{ \mathcal{I}'(H_i) : R_i \in P|_p \text{ and } \exists\theta \left(\tilde{B}_i\theta \subseteq \mathcal{I} \right) \right\} \\
&= \bigcup_{1 \leq i \leq N} H_i\alpha_i.
\end{aligned}$$

Thus proposition (7) is equivalent to (6) in the sense that a subset $\mathcal{I}$ of HB_P satisfies (6) iff there is a model $\mathcal{I}'$ of (7) such that $\mathcal{I}'(\Upsilon_p) = \mathcal{I}|_p$ for all $p \in \Pi_P$.

We now prove that proposition (7) and (8) are equivalent in the sense that $\mathcal{I}$ models (7) iff there is an adaptation $\mathcal{I}'$ of $\mathcal{I}$ (to the variables Υ_i) such that $\mathcal{I}'$

models (8). Let $\mathcal{I}$ be a model of (7). Construct $\mathcal{I}'$ by

$$\mathcal{I}'(\Upsilon_i) \stackrel{\text{def}}{=} \begin{cases} \mathcal{I}(H_i), & \text{if } \exists\theta \left(\tilde{B}_i\theta \subseteq \bigcup\limits_{p\in\Pi_P} \mathcal{I}(\Upsilon_p) \right) \\ \emptyset, & \text{otherwise.} \end{cases} \tag{4.8}$$

$\mathcal{I}'$ models the third conjunction in (8), which represent the second conjunction in (7), because it agrees with $\mathcal{I}$ on the variables in these equations. To show that $\mathcal{I}'$ models the second conjunction in (8), consider the i^{th} equation therein. If $\neg\exists\theta \left(\tilde{B}_i\theta \subseteq \mathcal{I} \right)$ then $\mathcal{I}'$ maps the r.h.s. of this equation into $\emptyset$. In the alternative case, $\mathcal{I}'$ maps the r.h.s. of the equation into $\mathcal{I}'(\{t : t \in H_i\})$, that is, $\mathcal{I}(H_i)$. Thus, in both cases, $\mathcal{I}'$ maps the r.h.s. of the equation into $\mathcal{I}'(\Upsilon_i)$. Finally, that $\mathcal{I}'$ models the first conjunction of (8) can be shown by first fixing the value of p and then arguing as follows:

$$\begin{aligned} \mathcal{I}'(\Upsilon_p) &= \mathcal{I}(\Upsilon_p) \\ &= \mathcal{I}\left(\bigcup \left\{ H_i : R_i \in P|_p \text{ and } \exists\theta \left(\tilde{B}_i\theta \subseteq \bigcup_{p\in\Pi_P} \Upsilon_p \right) \right\} \right) \\ &= \bigcup \left\{ \mathcal{I}\left(H_i\right) : R_i \in P|_p \text{ and } \exists\theta \left(\tilde{B}_i\theta \subseteq \bigcup_{p\in\Pi_P} \mathcal{I}(\Upsilon_p) \right) \right\} \\ &= \mathcal{I}'\left(\bigcup_{R_i\in P|_p} \Upsilon_i \right) \end{aligned}$$

The first equation follows the definition of $\mathcal{I}'$. The second equation holds because $\mathcal{I}$ is a model of (7). The third equation merely involves a distribution of $\mathcal{I}$. The final equation follows from the definition (4.8). Hence any model $\mathcal{I}$ of (7) can be adapted to become a model $\mathcal{I}'$ of (8) such that both models agree on the variables Υ_p.

The converse argument, that any model of (8) is a model of (7), can be established by similar arguments. Thus proposition (8) is equivalent to (7) in the sense that an interpretation $\mathcal{I}$ is a model of (7) iff there is a model $\mathcal{I}'$ of (8) such that $\mathcal{I}'(\Upsilon_p) = \mathcal{I}(\Upsilon_p)$ for all $p \in \Pi_P$.

The difference between propositions (8) and (9) is that the second and third conjunctions in (8) are collapsed into one conjunction in (9). Specifically, for each i, $1 \leq i \leq N$, the equations

$$\Upsilon_i = \left\{ t : t \in H_i \text{ and } \exists\theta \left(\tilde{B}_i\theta \subseteq \bigcup_{p \in \Pi_P} \Upsilon_p \right) \right\} \tag{4.9}$$

$$\bigwedge_{X \in var(R_i)} X = \left\{ X\theta : \tilde{B}_i\theta \subseteq \bigcup_{p \in \Pi_P} \Upsilon_p \right\} \tag{4.10}$$

in the second and third conjunctions in (8) are replaced by

$$\Upsilon_i = \left\{ \Upsilon_i\beta : \begin{pmatrix} \Upsilon_i\beta & \in & H_i \\ \tilde{B}_i\beta & \subseteq & \bigcup_{p \in \Pi_P} \Upsilon_p \end{pmatrix} \right\} \tag{4.11}$$

$$\bigwedge_{W \in var(R_i)} W = \left\{ W\beta : \begin{pmatrix} \Upsilon_i\beta & \in & H_i \\ \tilde{B}_i\beta & \subseteq & \bigcup_{p \in \Pi_P} \Upsilon_p \end{pmatrix} \right\} \tag{4.12}$$

in (9). Let $\mathcal{I}$ be a model of (8). We now show that $\mathcal{I}$ models (9), and we proceed by case analysis.

First assume that $\exists\theta \left(\tilde{B}_i\theta \subseteq \bigcup_{p \in \Pi_P} \mathcal{I}(\Upsilon_p) \right)$. In this case, the r.h.s. of (4.9) and (4.11) respectively reduce to $\{t : t \in H_i\}$ and $\{\Upsilon_i\beta : (\Upsilon_i\beta \in H_i)\}$ when interpreted by $\mathcal{I}$. Thus $\mathcal{I}$ maps both the r.h.s. of (4.9) and (4.11) into $\mathcal{I}(H_i)$. Before considering (4.10) and (4.12), we note that since $\mathcal{I}$ models (4.10), it maps $var(R_i)$ into nonempty sets and this, in turn, implies that $\mathcal{I}(H_i)$ is nonempty. Now consider the equation in (4.10) corresponding to a variable $V \in var(R_i)$, and consider all the substitutions θ satisfying the condition in the r.h.s. of the equation. Since $\Upsilon_i \notin var(R_i)$, we can define a substitution β corresponding to each substitution θ by extending θ so that $\Upsilon_i\beta \in \mathcal{I}(H_i)$. Thus β satisfies the condition in the r.h.s. of the equation corresponding to V in (4.12). Conversely, any β satisfying (4.12) can be used as a θ in (4.10). It then easily follows that $\mathcal{I}$ maps the r.h.s of the equation for V in both (4.10) and (4.12) to the same set. Hence $\mathcal{I}$ models (4.11) and (4.12).

The alternative case, that $\neg\exists\theta \left(\tilde{B}_i\theta \subseteq \bigcup_{p \in \Pi_P} \mathcal{I}(\Upsilon_p) \right)$, is easy. The r.h.s. of (4.9), (4.10), (4.11) and (4.12) are each mapped into $\emptyset$, and so $\mathcal{I}$ models (4.11) and (4.12).

Thus a model of (8) is also a model of (9). The converse proof is essentially similar and hence omitted. Hence the models of (8) are precisely the models of (9).

Proposition (10) is essentially a restatement of (9) in such a way that the atoms in $\tilde{B}_i$ are mentioned explicitly. Note that the extra condition $Z\beta \in \top_t$ is vacuous in the sense that it satisfied by any β. Proposition (11) is simply a restatement of (10) using the notation $\mathcal{D}_P^i$ introduced above for a set of pairs. Thus propositions (9) through (11) are mutually equivalent.

To see that proposition (12) is equivalent to (11), we note that the only difference between them is that in each equation in (11), the expression of the form

$$\left\{ W\theta : \bigwedge_{(a,sexp)\in\mathcal{D}_P^i} a\theta \in sexp \right\} \tag{4.13}$$

is replaced by the expression in (12):

$$\left\{ x : \exists y_1 y_2 \cdots y_m \left(\bigwedge_{(a,sexp)\in\mathcal{D}_P^i} a\mu \in sexp \right) \right\} \tag{4.14}$$

where $var(\tilde{R}_i) = \{W, Y_1, \cdots, Y_m\}$, $x, y_1, \cdots, y_m$ are new variables, and μ denotes the renaming substitution $[W/x, Y_1/y_1, \cdots, Y_m/y_m]$. Let $\mathcal{I}'$ be an interpretation of (4.13) and let g be an arbitrary element in $\mathcal{I}'(4.13)$. Therefore $g = W\theta$, for some substitution $\theta = [W/g, Y_1/t_1, \cdots, Y_m/t_m, \cdots]$ which grounds $\{W, Y_1, \cdots, Y_m\}$, and $a\theta \in \mathcal{I}(sexp)$ for each $(a, sexp) \in \mathcal{D}_P^i$. Clearly g is in $\mathcal{I}'(4.14)$ because the assignment of the values $g, t_1, \cdots, t_m$ to the variables $x, y_1 y_2 \cdots y_m$ satisfies $a\mu \in \mathcal{I}(sexp)$, for each $(a, sexp) \in \mathcal{D}_P^i$. The converse case is proved similarly by constructing an appropriate θ given an assignment of values to the variables $x, y_1, \cdots, y_m$. Hence proposition (12) is equivalent to (11).

The formula (13) is, by definition, the formula in (12). Summarizing all the above arguments, we have

- propositions (1) through (6) are mutually equivalent;
- a subset $\mathcal{I}$ of HB_P satisfies (6) iff there is model $\mathcal{I}'$ of (7) such that $\mathcal{I}'(\Upsilon_p) = \mathcal{I}|_p$ for all $p \in \Pi_P$;
- $\mathcal{I}'$ models (7) iff there is a model $\mathcal{I}''$ of (8) such that $\mathcal{I}''(\Upsilon_p) = \mathcal{I}'(\Upsilon_p)$ for all $p \in \Pi_P$;
- the models of (8) are precisely the models of (9); and

- propositions (9) through (13) are essentially restatements of one another;

and so Main lemma 1 is proved. □

In what follows, the problem at hand is an algorithm to construct the least model of the set formula $\mathcal{C}_P \wedge \mathcal{F}(\tilde{\mathcal{D}}_P)$ corresponding to a program P. The following corollary of the above lemma, whose proof is immediate, guarantees that this least model is, in essence, $\mathcal{T}_P{\uparrow}\omega$.

Corollary 1 *Let P be a logic program, and let $\mathcal{I}$ be the least model of the set formula $\mathcal{C}_P \wedge \mathcal{F}(\tilde{\mathcal{D}}_P)$ corresponding to P. Then*

$$\bigcup_{p \in \Pi_P} \mathcal{I}(\Upsilon_p) = \mathcal{T}_P{\uparrow}\omega. \quad \square$$

5 A Simplification Algorithm for Set Equations

We shall deal with formulas of the form $\mathcal{F}\left(\tilde{\mathcal{D}}\right)$ in the next section. In this section, we present an algorithm which deals with collections of set equations $\mathcal{C}$. This algorithm implements the function $SIMPLIFY(\mathcal{C})$ which transforms $\mathcal{C}$ into an equivalent collection written in a canonical "solved" form. The most important property of this form is that properties of its least model can be easily verified.

5.1 Normal Form, Solved Form and Explicit Form

This subsection contains the definitions of the input and output forms for the $SIMPLIFY$ algorithm. We begin with some preliminary notations and definitions.

Let $\mathcal{C}$ be the collection[8] of equations $\{X_1 = exp_1, \cdots, X_n = exp_n\}$ such that $exp_1, \cdots, exp_n$ are arbitrary expressions and $X_1, \cdots, X_n$ are distinct variables called the *solved variables*. We write $lhs(\mathcal{C})$ to denote the set of variables in the left-hand-sides of equations, that is the solved variables. We also write $rhs(\mathcal{C})|_X$ to denote

- the set expression on the right-hand-side of the equation for X, if X is a solved variable in $\mathcal{C}$;

[8] For convenience, we shall sometimes treat conjunctions such as $\mathcal{C}_P$ and $\mathcal{F}(\tilde{\mathcal{D}}_P)$ as sets of equations.

- X itself, if X is not a solved variable in $\mathcal{C}$.

An important part of the simplification algorithm described in this section involves the use of restricted forms of set equations. One restriction is on the depth of the nesting[9] of set expressions. If t is a subexpression of a set expression $sexp$, then we say that this occurrence of t is nested if it occurs within the scope of a function or predicate symbol, and otherwise the occurrence is un-nested . For example in $X \cup p(X, Y)$ the first occurrence of X is un-nested, but the second is nested. We say that a set expression, set equation or a collection of set equations is *deeply nested* if it contains a nested occurrence of a non-variable term. For example $f(U \cap V)$ and $g(U, a)$ are deeply nested, but $X \cap g(U, X)$ is not. In this section, no set equations will be deeply nested.

Another restriction relates to the occurrence of $\cup$ symbols inside the scope of $\cap$ symbols. We define

Definition 1 (DNF) *An* primitive *set expression is either a constant, a variable, or of the form $f(\tilde{t})$ where $f \in \Pi_P \cup \Sigma_P$ is n-ary and $\tilde{t}$ is a sequence of n primitive set expressions. A* conjunction *of set expressions, or simply* conjunct*, is of the form $a_1 \cap a_2 \cap \cdots \cap a_n$ where each $a_i, 1 \leq i \leq n$, is an primitive set expression. A set expression is in* disjunctive normal form *(or just DNF) if it is of the form $sexp_1 \cup sexp_2 \cup \cdots \cup sexp_n$ where each $sexp_i, 1 \leq i \leq n$, is a conjunction of set expressions.* □

We note that when a set expression is not deeply nested (and this is the case in this section), it can be transformed using the usual distributive operation on $\cap$ over $\cup$ to obtain an equivalent expression in DNF. For a set expression $sexp$, we write $DNF(sexp)$ to denote the corresponding DNF expression thus obtained. Clearly such a function is easily implementable. We note that this function may change the number of symbols in $sexp$ however it preserves primitive subexpressions: an primitive expression a is a subexpression of $sexp$ if and only if it is a subexpression of $DNF(sexp)$. Furthermore, the DNF function does not change the meaning of an expression, that is, for any $sexp$ and interpretation $\mathcal{I}$, $\mathcal{I}(sexp) = \mathcal{I}(DNF(sexp))$. We are now in a position to define the form of the input to the $SIMPLIFY$ algorithm.

Definition 2 (Normal Form) *A collection of set equations $\mathcal{C}$ is in* normal form *if it is of the form $X_1 = sexp_1$, $X_2 = sexp_2$, $\cdots$, $X_n = sexp_n$ where*

[9] The term nesting is used throughout to mean encapsulation within the scope of a functor.

(a) the X_i, $1 \leq i \leq n$, are distinct variables, and (b) the $sexp_i$, $1 \leq i \leq n$, are set expressions which are in DNF and are not deeply nested. □

In what follows, we shall tacitly assume that the symbols $\cup$ and $\cap$ are associative, commutative and idempotent. Thus, for example, we may regard $s \cap (t \cap u)$ and $(s \cap t) \cap u$ as identical expressions.

The equations output by the $SIMPLIFY$ function are a restricted kind of normal form equations defined as follows:

Definition 3 (Solved Form) *A collection of set equations $\mathcal{C}$ is in* solved form *if it is in normal form, and every conjunct in $rhs(\mathcal{C})$ is either (a) a constant (including $\top$ and $\bot$), or (b) of the form $f(Y_1, \cdots, Y_n)$ where f is an n-ary functor and Y_1, $\cdots$, Y_n are variables, or (c) a non-solved variable, or (d) of the form $X \cap sexp$ where X is a non-solved variable.* □

Where $X = s_1 \cup \cdots \cup s_n$ is an equation in a solved form collection $\mathcal{C}$, we write $crhs(\mathcal{C})$ to denote the collection of the conjuncts in $rhs(\mathcal{C})$ which fall into cases (a) and (b) above. Similarly, we define $crhs(\mathcal{C})|_X$ to be the collection of the conjuncts in $rhs(\mathcal{C})|_X$ which fall into cases (a) and (b) above.

Despite the above seemingly complicated definition of solved form, the main property that is required from equations $\mathcal{C}$ in solved form is that $\mathcal{C}$ can be written into a form such that the least model of $\mathcal{C}$ is explicitly represented:

Definition 4 (Explicit Form) *A collection of set equations $\mathcal{C}$ is in* explicit form *if it is in solved form, and every conjunct in $rhs(\mathcal{C})$ is primitive and contains no non-solved variables.* □

Before giving the details of how a solved form collection can be rewritten into explicit form, consider some equivalent formulations of this notion of explicit form. First observe that an explicit form collection $\mathcal{C}$ can be considered as a cyclic term-graph representation of its least model. This graph can be informally described as follows:

- each node is labelled with a constant, functor, the symbol $\cup$, or a distinct variable;

- each constant has no out-going edges, each n-ary functor has n out-going edges to the nodes corresponding to its arguments, and similarly $\cup$ has two out-going edges; each variable has one out-going edge to the node corresponding to $rhs(\mathcal{C})|_X$.

Such a graph describes a model in an obvious way: every node in the graph can be interpreted as the set of ground terms which can be constructed by following paths from that node. Thus there is a set of ground terms associated to each solved variable.

Alternatively, an explicit form collection of equations can be viewed as a term grammar as follows:

for each equation $X = t_1 \cup \ldots \cup t_n$ in $\mathcal{C}$, construct the productions
$X \Longrightarrow t_1$, ..., $X \Longrightarrow t_n$.

Such productions, in turn, can be viewed as a description of a non-deterministic tree automaton (see, for example, [4]). Hence standard algorithms can be applied to determine questions such as membership, inclusion and emptyness.

It can easily be proved that given an explicit form collection $\mathcal{C}$, we can construct a term graph, a term grammar and a tree automaton, all of which describe the least model of $\mathcal{C}$. It can also be proved that $\mathcal{C}$ has only one model. Summarizing, the essential property of an explicit form collection is that it is an unambiguous and explicit description of a model, that is:

- $\mathcal{C}$ has exactly one model with respect to $var(\mathcal{C})$;
- this model can be viewed as a regular set [4] of terms; thus there are algorithms to decide questions about emptiness and containment.

We conclude this subsection by showing that a solved form collection of equations can be transformed into explicit form. Formally:

Proposition 2 *A collection $\mathcal{C}$ of set equations in solved form has a least model. Furthermore, $\mathcal{C}$ can rewritten into an explicit form collection such that the least model is preserved.*

Proof: Define that a set expression is *non-empty with respect to* a set of variables $\mathcal{V}$ if it is either (a) a constant different from $\perp$, or (b) of the form $f(X_1, \cdots, X_n)$

where each X_i, $1 \leq i \leq n$, is in $\mathcal{V}$, or (c) of the form $t_1 \cup \cdots \cup t_k$ where some t_i, $1 \leq i \leq k$, is non-empty w.r.t. $\mathcal{V}$. Otherwise a set expression is *empty* w.r.t $\mathcal{V}$. Thus, if a $sexp$ is non-empty w.r.t. $\mathcal{V}$ then $\mathcal{I}(sexp) \neq \emptyset$ whenever $\mathcal{I}(X) \neq \emptyset$ for all $X \in \mathcal{V}$. Finally, define that a solved variable X in a solved form $\mathcal{C}$ is *non-empty* (respectively *empty*) *w.r.t.* $\mathcal{V}$ if $rhs(\mathcal{C})|_X$ is non-empty (respectively empty) w.r.t. $\mathcal{V}$.

Now consider the following marking algorithm whose purpose is to identify those variables in $var(\mathcal{C})$ which are empty in $lm(\mathcal{C})$.

let all variables in $\mathcal{C}$ be unmarked;
while there is a solved variable X in $\mathcal{C}$ such that
    X is unmarked, and
    $rhs(\mathcal{C})|_X$ is non-empty w.r.t. the marked variables
do
    mark X;
endwhile

Clearly the algorithm terminates, returning two sets M and U of marked and unmarked variables respectively. It is easy to show by induction that every variable in M is indeed non-empty in every model of $\mathcal{C}$. Using the set M, we now construct an explicit form collection $\mathcal{C}_e$ whose model is $lm(\mathcal{C})$.

Construct $\mathcal{C}_e$ from $\mathcal{C}$ as follows. For each equation $X = sexp$ in $\mathcal{C}$, delete every conjunction in $sexp$ which is empty with respect to M. In case $sexp$ becomes empty as a result of such deletion, the equation becomes $X = \perp$. The resulting collection $\mathcal{C}_e$ is in explicit form because every conjunction in $rhs(\mathcal{C}_e)$ is non-empty w.r.t. M and hence is primitive and contains no non-solved variables.

Let $\mathcal{I}_e$ denote $lm(\mathcal{C}_e)$, and we prove that $\mathcal{I}_e$ is a model of $\mathcal{C}$. Corresponding to each equation $X = sexp$ in $\mathcal{C}_e$ is an equation in $\mathcal{C}$ of the form $X = sexp \cup sexp'$; and vice versa. Now, for any variable $Y \in U$, either Y does not appear in $\mathcal{C}$ or else $Y = \perp$ appears in $\mathcal{C}_e$; thus $\mathcal{I}_e(Y) = \emptyset$. Next, each conjunction in $sexp'$ is empty w.r.t. M, and so the conjunction is either $\perp$ or of the form $f(Y_1, \cdots, Y_k)$ where one of the variables $Y_1, \cdots, Y_k$ is in U. Thus $\mathcal{I}_e$ maps the conjunction into $\emptyset$. This, in turn, implies that $\mathcal{I}_e(sexp') = \emptyset$. Since $\mathcal{I}_e$ is also a model of $X = sexp$ in $\mathcal{C}_e$, $\mathcal{I}_e$ models the equation $X = sexp \cup sexp'$ in $\mathcal{C}$. Thus $\mathcal{I}_e$ models $\mathcal{C}$.

To complete the proof, it remains to show that $\mathcal{I}_e = lm(\mathcal{C})$. Observe that for each variable X, if $X \in lhs(\mathcal{C}_e)$ then $rhs(\mathcal{C}_e)|_X \subseteq rhs(\mathcal{C})|_X$. We now use this fact

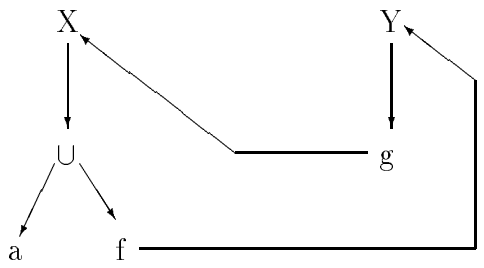
to show by structural induction that for all terms t, for all models $\mathcal{I}$ of $\mathcal{C}$ and for all variables X, $t \in \mathcal{I}_e(X)$ implies $t \in \mathcal{I}(X)$. If t is a constant, say c, then

$c \in \mathcal{I}_e(X)$ implies
c is a conjunct in $rhs(\mathcal{C}_e)|_X$ implies
c is a conjunct in $rhs(\mathcal{C})|_X$ implies
$c \in \mathcal{I}(X)$.

If t is of the form $f(t_1, \cdots, t_n)$ then

$f(t_1, \cdots, t_n) \in \mathcal{I}_e(X)$ implies
$rhs(\mathcal{C}_e)|_X$ has a conjunct $f(X_1, \cdots, X_n)$ s.t. $t_i \in \mathcal{I}_e(X), 1 \leq i \leq n$ implies
$rhs(\mathcal{C})|_X$ has a conjunct $f(X_1, \cdots, X_n)$ s.t. $t_i \in \mathcal{I}(X), 1 \leq i \leq n$ implies
$f(t_1, \cdots, t_n) \in \mathcal{I}(X)$ □

Example 1 *If* $\mathcal{C}$ *were* $\begin{pmatrix} X = f(Y) \cup a \\ Y = g(X) \cup (h(X) \cap Z) \end{pmatrix}$ *then* $\mathcal{C}_e$ *is* $\begin{pmatrix} X = f(Y) \cup a \\ Y = g(X) \end{pmatrix}$.
A term graph corresponding to $\mathcal{C}_e$ *can be constructed as follows:*



We can also construct a term grammar and a frontier to root tree automaton (with start state q_0*) corresponding to* $\mathcal{C}_e$ *as follows:*

$$\begin{array}{lcl} X & \Longrightarrow & f(Y) \\ X & \Longrightarrow & a \\ Y & \Longrightarrow & g(X) \end{array} \qquad \begin{array}{lcl} q_{\mathrm{Y}} & \xrightarrow{f} & q_{\mathrm{X}} \\ q_0 & \xrightarrow{a} & q_{\mathrm{X}} \\ q_{\mathrm{X}} & \xrightarrow{g} & q_{\mathrm{Y}} \end{array}$$

The least model of $\mathcal{C}$ *described by all the above is:*

$$\begin{array}{lcl} X & = & \{a, fg(a), \cdots (fg)^n(a), \cdots\}. \\ Y & = & \{g(a), gfg(a), \cdots (gf)^n g(a), \cdots\}. \end{array}$$ □

5.2 SIMPLIFY

Consider the following group of five transformation steps on a normal form collection $\mathcal{C}$ of set equations.

Transformation 1 (C-Substitute) *If $\mathcal{C}$ contains two distinct equations $X = sexp$ and $Y = t$ such that $sexp$ contains an un-nested occurrence of the variable Y, then replace the first equation by $X = DNF(sexp')$ where $sexp'$ is the result of replacing the occurrence of Y in $sexp$ by t. We say that a C-substitution transformation* using *Y was applied to $\mathcal{C}$ to obtain $\mathcal{C}'$.*

Transformation 2 (Cycle-Removal) *If $\mathcal{C}$ contains the equation $X = sexp$, such that $sexp$ contains an un-nested occurrence of the variable X, then replace that occurrence of X by a new variable.*

Transformation 3 (Intersection Clash) *If $\mathcal{C}$ contains an occurrence of the expression $t \cap s$ where t and s have distinct outermost functors, replace that occurrence by $\bot$.*

Transformation 4 (Intersection Simplification) *If $\mathcal{C}$ contains an occurrence of the expression $t \cap \bot$ or $t \cap \top$, then replace that occurrence by $\bot$ or t respectively.*

Transformation 5 (Intersection Distribution) *If $\mathcal{C}$ contains an occurrence of the expression $f(X_1, \cdots, X_n) \cap f(Y_1, \cdots, Y_n)$, and if there exist variables Z_i, $1 \leq i \leq n$, such that $\mathcal{C} \models Z_i = X_i \cap Y_i$, then replace that occurrence by $f(Z_1, \cdots, Z_n)$.*

The first two transformations implement a restricted form of substitution whose purpose is to remove all un-nested occurrences of solved variables on the right-hand-sides of equations. The remaining three transformations perform obvious simplifications involving the $\cap$ symbol. We note that all five of the transformations preserve normal form, that is, the application of a transformation to a collection in normal form results in another collection in normal form. We now prove that the above transformations are correct in the sense that they preserve equivalence between the input and output collections of set equations. These proofs are straightforward. The exception is the proof for cycle-removal transformation where the transformation itself is somewhat surprising.

Lemma 1 (Correctness of Transformations 1 thru 5)
Let $\mathcal{C}'$ be obtained from $\mathcal{C}$ by using transformation 1, 2, 3, 4 or 5. Then

$$\mathcal{C} \underset{var(\mathcal{C})}{\Longleftrightarrow} \mathcal{C}'.$$

Proof for Transformations 1. Straightforward and hence omitted.

Proof for Transformation 2. The only difference between $\mathcal{C}$ and $\mathcal{C}'$ is that the former contains an equation of the form $X = (X \cap sexp_1) \cup sexp_2$ while the latter contains $X = (Z \cap sexp_1) \cup sexp_2$ where Z is a new variable.

Now, given any model $\mathcal{I}$ of $\mathcal{C}$, we can modify $\mathcal{I}$ to obtain a model $\mathcal{I}'$ of $\mathcal{C}'$ by changing the assignment for the variable Z so that $\mathcal{I}'(Z) = \mathcal{I}(X)$. Clearly $\mathcal{I}$ and $\mathcal{I}'$ agree on $var(\mathcal{C})$. Hence we have proved that if $\mathcal{I}$ is a model of $\mathcal{C}$ then there exists a model $\mathcal{I}'$ of $\mathcal{C}'$ such that $\mathcal{I} =_{var(\mathcal{C})} \mathcal{I}'$.

To prove the converse (and complete the proof from transformation 2), we show that any model of $\mathcal{C}'$ satisfies $\mathcal{C}$. Let $\mathcal{I}'$ be a model of $\mathcal{C}'$. Hence $\mathcal{I}'$ satisfies

$$\mathcal{I}'(X) = (\mathcal{I}'(Z) \cap \mathcal{I}'(sexp_1)) \cup \mathcal{I}'(sexp_2)$$

To show that $\mathcal{I}'$ satisfies $\mathcal{C}$, it suffices to show that $\mathcal{I}'$ satisfies $X = (X \cap sexp_1) \cup sexp_2$, or equivalently, that $\mathcal{I}'$ satisfies

$$\mathcal{I}'(X) = (\mathcal{I}'(X) \cap \mathcal{I}'(sexp_1)) \cup \mathcal{I}'(sexp_2)$$

To show this, first choose an arbitrary $g \in \mathcal{I}'(X)$, and consider the following chain of reasoning:

$g \in \mathcal{I}'(X)$ implies
$g \in (\mathcal{I}'(Z) \cap \mathcal{I}'(sexp_1)) \cup \mathcal{I}'(sexp_2)$ and $g \in \mathcal{I}'(X)$ implies
$g \in \mathcal{I}'(sexp_1) \cup \mathcal{I}'(sexp_2)$ and $g \in \mathcal{I}'(X)$ implies
$g \in (\mathcal{I}'(X) \cap \mathcal{I}'(sexp_1)) \cup \mathcal{I}'(sexp_2)$.

This proves that $\mathcal{I}'(X) \subseteq (\mathcal{I}'(X) \cap \mathcal{I}'(sexp_1)) \cup \mathcal{I}'(sexp_2)$. To show the converse, choose an arbitrary $g \in (\mathcal{I}'(X) \cap \mathcal{I}'(sexp_1)) \cup \mathcal{I}'(sexp_2)$ and we have:

$g \in (\mathcal{I}'(X) \cap \mathcal{I}'(sexp_1)) \cup \mathcal{I}'(sexp_2)$ implies
$g \in \mathcal{I}'(X)$ or $g \in \mathcal{I}'(sexp_2)$ implies
$g \in (\mathcal{I}'(Z) \cap \mathcal{I}'(sexp_1)) \cup \mathcal{I}'(sexp_2)$ or $g \in \mathcal{I}'(sexp_2)$ implies

$$g \in (\mathcal{I}'(Z) \cap \mathcal{I}'(sexp_1)) \cup \mathcal{I}'(sexp_2) \quad \text{implies}$$
$$g \in \mathcal{I}'(X).$$

Hence $\mathcal{I}'$ is a model of $\mathcal{C}$ and the proof is finished.

Proof for Transformations 3 and 4. The proof is straightforward from the observation that for any interpretation $\mathcal{I}$ of $\mathcal{C}$, and for any set expressions t and s, (a) $\mathcal{I}(t \cap s) = \emptyset$ if t and s have different outermost functors, (b) $\mathcal{I}(t \cap \bot) = \emptyset$, and finally, (c) $\mathcal{I}(t \cap \top) = \mathcal{I}(t)$.

Proof for Transformation 5. The proof is straightforward from the observation that for any model $\mathcal{I}$ of $\mathcal{C}$, $\mathcal{I}(f(X_1, \cdots, X_n) \cap f(Y_1, \cdots, Y_n))$ is equal to $\mathcal{I}(f(X_1 \cap Y_1, \cdots, X_n \cap Y_n))$ and by the hypothesis $\mathcal{C} \models Z_i = X_i \cap Y_i$, this is equal to $\mathcal{I}(f(Z_1, \cdots, Z_n))$. □

We present the *SIMPLIFY* algorithm in figure 1. It is essentially an exhaustive application of the above transformations. Just above, we have argued that the transformations are correct in the sense that they preserve equivalence between the input and output collections of equations. To ensure that the final output is solved form, however, we require that transformation 5 above is, in fact, applicable whenever there exists a subexpression of the form $f(X_1, \cdots, X_n) \cap f(Y_1, \cdots, Y_n)$ in $\mathcal{C}$. Toward this end, new variables may have to be introduced before transformation 5 is applied. However, the number of new variables introduced can be bounded by $2^{|\mathcal{V}|}$ where $\mathcal{V}$ is the set of variables in the collection of set equations $\mathcal{C}$ just after the execution of line 3. This is because, after execution of line 3, the algorithm will only be dealing with variables of the form V_S where $S \subseteq \mathcal{V}$. Intuitively, we start off by having a variable $V_{\{X\}}$ corresponding to each variable X in the input equations. Whenever new variables are introduced, they will be of the form $V_{\{X_1, \cdots, X_n\}}$ and their intended meaning is $\cap V_{\{X_i\}}$ over $1 \leq i \leq n$. The essential benefit in such a variable naming scheme is that the one variable can capture the meaning of several expressions. For example, $V_{\{X,Y,Z\}}$ represents not just $V_{\{X\}} \cap V_{\{Y\}} \cap V_{\{Z\}}$, but also $V_{\{X,Y\}} \cap V_{\{Z\}}$ and $V_{\{X\}} \cap V_{\{Y,Z\}} \cap V_{\{Y\}} \cap V_{\{X\}}$ etc.

Lemma 2 *Let $\mathcal{C}$ denote a normal form collection of set equations. Then the SIMPLIFY algorithm, when input with $\mathcal{C}$,*

- *terminates, and outputs a collection $\mathcal{C}_s$ of set equations;*
- *$\mathcal{C}_s$ is in solved form, and*
- $\mathcal{C} \underset{var(\mathcal{C})}{\Longleftrightarrow} \mathcal{C}_s$.

input a collection $\mathcal{C}$ of set equations in normal form;
1. **for** each solved variable X in $\mathcal{C}$ **do**
2. **exhaustively-apply** transformation 2 to $\mathcal{C}$;
3. **exhaustively-apply** transformation 1 using X to $\mathcal{C}$;

endfor
4. **rename** each variable X in $\mathcal{C}$ into $V_{\{X\}}$;
5. **repeat**
6. **exhaustively-apply** transformations 3 and 4 to $\mathcal{C}$;
7. **if** $\mathcal{C}$ contains a subexpression of the form $f(V_{S_1}, V_{S_2}, \cdots, V_{S_n}) \cap f(V_{T_1}, V_{T_2}, \cdots, V_{T_n})$
8. **then** replace the above subexpression by $f(V_{S_1 \cup T_1}, \cdots, V_{S_n \cup T_n})$;
9. **for** each $i : 1 \leq i \leq n$ **do**
10. **if** $V_{S_i \cup T_i}$ is a non-solved variable **then**
11. add to $\mathcal{C}$ the set equation $V_{S_i \cup T_i} = DNF(rhs(\mathcal{C})|_{V_{S_i}} \cap rhs(\mathcal{C})|_{V_{T_i}})$;

endif

endfor

endif
12. **until** none of transformations 3, 4 and 5 is applicable to $\mathcal{C}$;

rename each variable $V_{\{X\}}$ in $\mathcal{C}$ into X;
output $\mathcal{C}$;

Figure 1: **SIMPLIFY**

Proof: We first prove termination. The termination of the for-loop at line 1 is proven as follows. Firstly, transformation 2 cannot be repeated infinitely because each application removes one occurrence of a solved variable from $\mathcal{C}$ and does not introduce new solved variables. Next, each application of transformation 1 using a solved variable X removes one un-nested occurrence of X from the right-hand-sides in $\mathcal{C}$. (This is guaranteed from the fact that transformation 2 has already been applied to $\mathcal{C}$ and so $rhs(\mathcal{C})|_X$ does not contain any un-nested occurrences of X.) Therefore, the exhaustive application of transformation 1 using X terminates. Finally, that the for-loop on line 1 terminates, follows from the fact that neither transformation 1 nor 2 introduce new solved variables.

That line 4 terminates is clear. We now show that the repeat-loop at line 5 terminates.

- The step at line 6 clearly terminates because both transformations 3 and 4 diminish the size of expressions in $\mathcal{C}$.
- The step at line 7 may enlarge the collection $\mathcal{C}$ with new equations. However, the total number of equations which can be added (line 11) is bounded by 2^k, where k is the number of variables in $\mathcal{C}$ on entry to the repeat-loop at line 5. Since transformation 5 diminishes the size of expressions in $\mathcal{C}$, the loop at line 5 terminates.

Next we observe that the following loop invariant holds: at the end of each iteration of the repeat-loop, (a) $\mathcal{C}$ is in normal form and (b) all solved variables X appearing on the right-hand-sides of $\mathcal{C}$ are nested.

Clearly (a) holds just before execution of the for-loop at line 2, and it continues to hold throughout execution of this loop. Consider then property (b). Each iteration of the for-loop with respect to a solved variable X concludes with an exhaustive C-substitution for un-nested occurrences of X. Therefore, there are no un-nested occurrences of X in $rhs(\mathcal{C})$ after this iteration. This, in turn, implies that further applications of transformations 1 or 2 (in other iterations of the for-loop) will not introduce any un-nested occurrences of X in $rhs(\mathcal{C})$. Since the for-loop iterates over all the solved variables in $\mathcal{C}$ (and these variables are fixed throughout the for-loop), (b) holds after execution of the for-loop. Therefore, both properties (a) and (b) hold just before execution of the repeat-loop.

For the inductive step, we now show that these properties hold after an iteration of the repeat-loop. The step at line 6 does not violate the properties because transformations 3 and 4 only serve to delete subexpressions. The step at line 7

first applies transformation 5, and then may make $V_{S_i \cup T_i}$ into a solved variable by the addition of a new equation $V_{S_i \cup T_i} = DNF(rhs(\mathcal{C})|_{V_{S_i}} \cap rhs(\mathcal{C})|_{V_{T_i}})$. That (a) is not violated by this addition is clear. That (b) is not violated by this addition is an easy consequence of the fact that both $rhs(\mathcal{C})|_{V_{S_i}}$ and $rhs(\mathcal{C})|_{V_{T_i}}$ satisfy (b), and that the DNF operation does not alter nesting of terms (it only deals with distributing $\cap$ symbols over $\cup$ symbols). Thus the invariant holds at the end of each iteration of the repeat-loop.

That $\mathcal{C}_s$ is in solved form is now an easy consequence of the above invariant and the fact that none of transformations 3, 4 and 5 are applicable to $\mathcal{C}_s$.

The remainder of the proof, that $\mathcal{C} \underset{var(\mathcal{C})}{\Longleftrightarrow} \mathcal{C}_s$, essentially follows from lemma 1. The only operation within $SIMPLIFY$ which is not directly stated as a transformation application is the **if** statement at line 7. This statement is in essence an application of transformation 5. To show that lemma 1 can be used, we now prove another invariant of the main loop: at the end of each iteration of the repeat-loop, each variable $V_{\{X_1,\cdots,X_n\}}$ in $\mathcal{C}$, $n \geq 1$, is such that $V_S = V_{\{X_1\}} \cap \cdots \cap V_{\{X_n\}}$ holds in all models of $\mathcal{C}$. This invariant clearly holds initially because all variables in $\mathcal{C}$ are of the form $V_{\{X\}}$ where $X \in \mathcal{V}$.

Now consider the inductive step. Clearly step 10 is the only step which may affect the invariant since it is the only step which introduces a new solved variable $V_{S_i \cup T_i}$ to $\mathcal{C}$. Let S_i be $\{U_1, \cdots, U_k\}$ and let T_i be $\{W_1, \cdots, W_l\}$. Since the invariant holds before the introduction of $V_{S_i \cup T_i}$, $V_{S_i} = V_{\{U_1\}} \cap \cdots \cap V_{\{U_k\}}$ and $V_{T_i} = V_{\{W_1\}} \cap \cdots \cap V_{\{W_l\}}$ are satisfied in all models of $\mathcal{C}$. Hence we have, in all models of $\mathcal{C}$, that

$$\begin{aligned}
V_{S_i \cup T_i} &= DNF(rhs(\mathcal{C})|_{V_{S_i}} \cap rhs(\mathcal{C})_{V_{T_i}}), \quad \text{by construction;} \\
&= rhs(\mathcal{C})|_{V_{S_i}} \cap rhs(\mathcal{C})|_{V_{T_i}}, \quad \text{since } DNF \text{ preserves equivalence;} \\
&= V_{S_i} \cap V_{T_i}, \\
&= V_{\{U_1\}} \cap \cdots \cap V_{\{U_k\}} \cap V_{\{W_1\}} \cap \cdots \cap V_{\{W_l\}}, \quad \text{by induction hypothesis;} \\
&= V_{\{X_1\}} \cap \cdots \cap V_{\{X_n\}}
\end{aligned}$$

where $\{X_1, \cdots, X_n\}$ is $\{U_1, \cdots, U_k\} \cup \{W_1, \cdots, W_l\}$. Thus the invariant holds.

Upon termination of the algorithm, this invariant and lemma 1 easily imply the desired result that $\mathcal{C} \underset{var(\mathcal{C})}{\Longleftrightarrow} \mathcal{C}_s$. □

Corollary 2 *Let $\mathcal{C}$ denote a normal form collection of set equations input to the $SIMPLIFY$ algorithm, and let $SIMPLIFY(\mathcal{C})$ denote the output. Then*

- $lhs(SIMPLIFY(\mathcal{C})) \supseteq lhs(\mathcal{C})$ *and*

- *any model of* $SIMPLIFY(\mathcal{C})$ *is a model of* $\mathcal{C}$. □

6 The Main Result

Recall that in section 4, we defined, corresponding to a given program P, a formula consisting of solved form collection $\mathcal{C}_P$ of set equations, and a formula $\mathcal{F}(\tilde{\mathcal{D}}_P)$ where $\tilde{\mathcal{D}}_P$ is a family of collections of pairs. We now present the algorithm which inputs such $\mathcal{C}_P$ and $\tilde{\mathcal{D}}_P$, preprocesses these expressions to obtain $\mathcal{C}^{(0)}$ and $\tilde{\mathcal{D}}^{(0)}$, and then performs a finite number of reduction steps

$$\mathcal{C}^{(0)},\ \tilde{\mathcal{D}}^{(0)} \Longrightarrow \mathcal{C}^{(1)},\ \tilde{\mathcal{D}}^{(1)} \Longrightarrow \cdots \Longrightarrow \mathcal{C}^{(i)},\ \tilde{\mathcal{D}}^{(i)} \Longrightarrow \cdots \Longrightarrow \mathcal{C}^{(n)},\ \tilde{\mathcal{D}}^{(n)} \qquad (6.15)$$

to finally obtain a collection of solved form set equations $\mathcal{C}^{(n)}$ such that the least model of $\mathcal{C}^{(n)}$ is the same as the least model of the formula $\mathcal{C}_P \wedge \mathcal{F}(\tilde{\mathcal{D}}_P)$. The essence of the algorithm is that it progressively transfers information about the least model we seek from $\tilde{\mathcal{D}}^{(i)}$ to $\mathcal{C}^{(i)}$. In the final step, therefore, $\mathcal{C}^{(n)} \wedge \mathcal{F}(\tilde{\mathcal{D}}^{(n)})$ will be such that all the desired information will be in $\mathcal{C}^{(n)}$.

The subsections below are organized as follows. In section 6.1, the main body of the algorithm which performs the reductions (6.15) is defined. Termination and correctness properties of the reductions are then established in sections 6.2 and 6.3 respectively. Finally, in section 6.4, we present a preprocessing algorithm which, when given $\mathcal{C}_P$ and $\tilde{\mathcal{D}}_P$ corresponding to a logic program P, outputs equivalent expressions $\mathcal{C}^{(0)}$ and $\tilde{\mathcal{D}}^{(0)}$ which may then be input to the main algorithm.

6.1 Description of Main Algorithm

We now describe the body of the main algorithm which starts with a collection of set equations $\mathcal{C}$ and a family of collections of pairs $\tilde{\mathcal{D}}$ which satisfy certain initial conditions. Before formally stating these initial conditions, define that a collection of quantified equations $\mathcal{F}(\tilde{\mathcal{D}})$ is in *normal form* if it is of the form $X_1 = qexp_1$, $\cdots$, $X_n = qexp_n$ where the X_i, $1 \leq i \leq n$, are distinct variables, and the r.h.s. of each pair in $\tilde{\mathcal{D}}$ is not deeply nested. We also define that a collection $\mathcal{E}$ of set *and* quantified equations is in normal form if the collection of set equations in $\mathcal{E}$ is in normal form, the collection of quantified equations in $\mathcal{E}$ is in normal form, and no two equations in $\mathcal{E}$ have the same variable in the l.h.s. The initial conditions on $\mathcal{C}$ and $\tilde{\mathcal{D}}$ can now be stated as:

Main Algorithm Precondition:
• $\mathcal{C}$ *is in solved form, and*
• $\mathcal{C} \cup \mathcal{F}(\tilde{\mathcal{D}})$ *is in normal form.*

The basis of this algorithm, as for the $SIMPLIFY$ algorithm described previously, is a group of transformations. Each transformation takes as input expressions $\mathcal{C}$ and $\tilde{\mathcal{D}}$ described above, and outputs two items: (a) a new family $\tilde{\mathcal{D}}'$ of collections of pairs, and (b) a collection δ of set equations in normal form. If a transformation is applied on $\mathcal{C}$ and $\tilde{\mathcal{D}}$ to obtain δ and $\tilde{\mathcal{D}}'$, we write

$$\mathcal{C}, \tilde{\mathcal{D}} \Longrightarrow \tilde{\mathcal{D}}', \delta$$

and we say that a transformation is *applied* on $\tilde{\mathcal{D}}$ in the *context* of $\mathcal{C}$ to obtain the the new family $\tilde{\mathcal{D}}'$ and an *extracted collection* δ of set equations.

Transformation 6 (D-Substitution) *If $\tilde{\mathcal{D}}$ contains an un-nested occurrence of a $\mathcal{C}$-solved variable Y in the r.h.s. of a pair, then replace the occurrence by $rhs(\mathcal{C})|_Y$. Output the resulting $\tilde{\mathcal{D}}$, and $\delta = \emptyset$.* □

Transformation 7 (Clash) *If $\mathcal{D}_j \in \tilde{\mathcal{D}}$ contains a pair $(f(s_1, \cdots, s_n), g(t_1, \cdots, t_m))$, where $f \neq g$, or a pair $(s, \perp)$, then output $\tilde{\mathcal{D}} - \{\mathcal{D}_j\}$ and $\delta = \{Y_1 = \perp, \cdots, Y_k = \perp\}$ where $Y_1, \cdots, Y_k$ are the variables in the left-hand-sides of $\mathcal{D}_j$.* □

Transformation 8 (Cancellation) *If $\tilde{\mathcal{D}}$ contains a pair $(f(s_1, \cdots, s_n), f(t_1, \cdots, t_n))$, then replace the pair by the pairs $(s_1, t_1), \cdots, (s_n, t_n)$. Output the resulting $\tilde{\mathcal{D}}$ and $\delta = \emptyset$.* □

The next transformation step is conditional on a relation $NONEMPTY$ which is defined as follows. Where $\mathcal{C}$ is a solved form collection of set equations and $sexp$ is a set expression, $NONEMPTY(\mathcal{C}, sexp)$ is *true* iff $\mathcal{C} \models sexp \neq \emptyset$. Clearly $NONEMPTY(\mathcal{C}, sexp)$ is equivalent to

$$NONEMPTY(SIMPLIFY(\mathcal{C} \cup \{X = sexp\}), X)$$

where X is a new variable. Thus the problem of implementing the function $NONEMPTY$ can be reduced to the problem of determining if a given variable is always non-empty in a given solved form. The proof of proposition 2 contains such an implementation: $NONEMPTY(\mathcal{C}, X)$ iff X is marked non-empty after applying the marking algorithm to the collection $SIMPLIFY(\mathcal{C} \cup \{X = sexp\})$.

Transformation 9 (Compaction) *If* $\mathcal{D}_j \in \tilde{\mathcal{D}}$ *is of the form* $(\tilde{X}, \tilde{\Phi})$ *and* $NONEMPTY(\mathcal{C}, \cap(\tilde{\Phi}|_{Y_1})) \wedge \cdots \wedge NONEMPTY(\mathcal{C}, \cap(\tilde{\Phi}|_{Y_k}))$ *where*

- *the notation* $\tilde{X}$ *denotes a sequence of variables (that is, the l.h.s. of each pair in* $\mathcal{D}_j$ *is a variable), and the variables in this sequence are* $Y_1, \cdots, Y_k$*;*
- *the notation* $\cap(\tilde{\Phi}|_{Y_l})$, $1 \leq l \leq k$, *denotes the set expression* $\phi_1 \cap \cdots \cap \phi_m$ *such that* $(Y_l, \phi_1), \cdots, (Y_l, \phi_m)$ *are the pairs in* $\mathcal{D}_j$ *of the form* $(Y, _l\, \phi)$

then output $\tilde{\mathcal{D}} - \{\mathcal{D}_j\}$ *and* $\delta = \{Y_1 = DNF(\cap(\tilde{\Phi}|_{Y_1})), \cdots, Y_k = DNF(\cap(\tilde{\Phi}|_{Y_k}))\}$. □

Example 2 *Let* $\mathcal{C}$ *contain* $U = a$ *and* $V = a$*, and let* $\mathcal{D}_j$ *be* $\{(X, f(U)), (Y, g(V)), (X, f(V))\}$*. Since* $NONEMPTY(\mathcal{C}, f(U) \cap f(V)) \wedge NONEMPTY(\mathcal{C}, g(V))$ *is* true*, the compaction transformation deletes* $\mathcal{D}_j$ *and outputs* $\delta = \{X = f(U) \cap f(V), Y = g(V)\}$*. Note that* $\mathcal{F}(\mathcal{D}_j)$*:*

$$X = \left\{ x : \exists y \left(\begin{array}{lll} x & \in & f(U) \\ y & \in & g(V) \\ x & \in & f(V) \end{array} \right) \right\}$$

$$Y = \left\{ y : \exists x \left(\begin{array}{lll} x & \in & f(U) \\ y & \in & g(V) \\ x & \in & f(V) \end{array} \right) \right\}$$

is equivalent to δ *in all interpretations where* $f(U) \cap f(V)$ *and* $g(V)$ *are nonempty (as is the case here in all models of* $\mathcal{C}$*).* □

Transformation 10 (Union) *If* $\mathcal{D}_j \in \tilde{\mathcal{D}}$ *is of the form* $\{(s, t_1 \cup t_2), (\tilde{\Psi}, \tilde{\Phi})\}$ *then*

- *let* $Y_1, \cdots, Y_k$ *be the variables in* s *and* $\tilde{\Psi}$*;*
- *let* $Y_1', \cdots, Y_k', Y_1'', \cdots, Y_k''$ *be distinct new variables and let* η_1 *and* η_2 *denote the renaming substitutions* $[Y_1/Y_1', \cdots, Y_k/Y_k']$ *and* $[Y_1/Y_1'', \cdots, Y_k/Y_k'']$*;*
- *let* $\tilde{\mathcal{D}}'$ *be* $\tilde{\mathcal{D}} - \{\mathcal{D}_j\} \cup \{\mathcal{D}_j', \mathcal{D}_j''\}$ *where*
  $\mathcal{D}_j' = \{(s\eta_1,\ t_1), (\tilde{\Psi}\eta_1,\ \tilde{\Phi})\}$ *and* $\mathcal{D}_j'' = \{(s\eta_2,\ t_2), (\tilde{\Psi}\eta_2,\ \tilde{\Phi})\}$*, and*
- *let* δ *be* $\{Y_1 = Y_1' \cup Y_1'', \cdots, Y_k = Y_k' \cup Y_k''\}$*;*

and output $\tilde{\mathcal{D}}'$ *and* δ*.* □

Example 3 *Suppose that $\mathcal{D}_j$ consists of the single pair $(p(X),\ p(U) \cup q(V))$. Then the union transformation replaces $\mathcal{D}_j$ by two collections $\mathcal{D}'_j = \{(p(X'), p(U))\}$ and $\mathcal{D}''_j = \{(p(X''), q(V))\}$, where X' and X'' are new variables. The equation in δ is $X = X' \cup X''$. Note that $\mathcal{F}(\mathcal{D}_j)$:*

$$X = \{x : p(x) \in p(U) \cup q(V)\}$$

is essentially equivalent to $\mathcal{F}(\mathcal{D}'_j) \wedge \mathcal{F}(\mathcal{D}''_j) \wedge \delta$:

$$\begin{array}{rcl} X & = & X' \cup X'' \\ X' & = & \{x' : p(x') \in p(U)\} \\ X'' & = & \{x'' : p(x'') \in p(U)\} \end{array}$$ □

Transformation 11 (Intersection) *If $\mathcal{D}_j \in \tilde{\mathcal{D}}$ contains a pair $(s, t_1 \cap t_2)$, then replace that pair by the two pairs (s, t_1), (s, t_2). Output the resulting $\tilde{\mathcal{D}}$ and $\delta = \emptyset$.* □

Whenever one of the above transformations is applied on $\tilde{\mathcal{D}}$ in the context of $\mathcal{C}$ to obtain δ and $\tilde{\mathcal{D}}'$, we may write $\mathcal{C}, \tilde{\mathcal{D}} \overset{tf}{\Longrightarrow} \tilde{\mathcal{D}}, \delta$ to indicate by tf the particular transformation used. We say that $\tilde{\mathcal{D}}$ is *closed* under a transformation if that transformation is not applicable to $\tilde{\mathcal{D}}$.

We are now in a position to define, in figure 2, the reduction step of our algorithm. This step is described in the form of a function whose essential operation is an exhaustive application of the above transformations in the context of a fixed collection $\mathcal{C}$ of set constraints. For the sake of simpler proofs, however, the order in which transformations may be applied is restricted.

Our main algorithm can now be described in figure 3. We conclude this subsection by proving two properties of the algorithm. The first ensures that the $SIMPLIFY$ function is only applied to normal form set equations, and the second implies that a key correctness condition specified later in this section is well defined.

Proposition 3 (Main Loop Invariant) *Let $\mathcal{C}^{(0)}$ and $\tilde{\mathcal{D}}^{(0)}$, satisfying the main algorithm precondition, be input to the main algorithm. Then:*

(a) just before every execution of $SIMPLIFY$, $\mathcal{C}^{(i)} \cup \Delta^{(i+1)} \cup \mathcal{F}(\tilde{\mathcal{D}}^{(i+1)})$ is in normal form, and

function REDUCE($\mathcal{C}$, $\tilde{\mathcal{D}}$) {
  $\Delta := \emptyset$;
1. **repeat** {
2. **while** (the D-substitution transformation is applicable to $\mathcal{C}, \tilde{\mathcal{D}}$) **do** {
  $\mathcal{C}, \tilde{\mathcal{D}} \underset{subs}{\Longrightarrow} \tilde{\mathcal{D}}, \delta$
  $\Delta := \Delta \cup \delta$;
  }
3. **while** (the union or intersection transformation is applicable to $\mathcal{C}, \tilde{\mathcal{D}}$) **do** {
  $\mathcal{C}, \tilde{\mathcal{D}} \underset{union}{\Longrightarrow} \tilde{\mathcal{D}}, \delta$ or
  $\mathcal{C}, \tilde{\mathcal{D}} \underset{intersect}{\Longrightarrow} \tilde{\mathcal{D}}, \delta$ accordingly;
  $\Delta := \Delta \cup \delta$;
  }
4. **while** (the cancel or clash transformation is applicable to $\mathcal{C}, \tilde{\mathcal{D}}$) **do** {
  $\mathcal{C}, \tilde{\mathcal{D}} \underset{cancel}{\Longrightarrow} \tilde{\mathcal{D}}, \delta$ or
  $\mathcal{C}, \tilde{\mathcal{D}} \underset{clash}{\Longrightarrow} \tilde{\mathcal{D}}, \delta$ accordingly;
  $\Delta := \Delta \cup \delta$;
  }
5. **while** (the compaction transformation is applicable to $\mathcal{C}, \tilde{\mathcal{D}}$) **do** {
  $\mathcal{C}, \tilde{\mathcal{D}} \underset{compact}{\Longrightarrow} \tilde{\mathcal{D}}, \delta$
  $\Delta := \Delta \cup \delta$;
  }
  } **until** (none of transformations 6–11 is applicable to $\mathcal{C}, \tilde{\mathcal{D}}$);
6. **return** $\left(\tilde{\mathcal{D}}, \Delta\right)$;
}

Figure 2: **REDUCE**

input $\mathcal{C}^{(0)}$ and $\tilde{\mathcal{D}}^{(0)}$;
1. $\mathcal{V}^\star := var(\mathcal{C}^{(0)}) \cup \{V_{\{X_1,\cdots,X_k\}} : X_i \in var(\mathcal{C}^{(0)}), 1 \leq i \leq k, k \geq 2\}$[a];
   $i := 0$;
2. **repeat** {
3. $\quad (\tilde{\mathcal{D}}^{(i+1)},\ \Delta^{(i+1)}) := REDUCE(\mathcal{C}^{(i)},\ \tilde{\mathcal{D}}^{(i)})$;
4. $\quad \mathcal{C}^{(i+1)} := SIMPLIFY(\mathcal{C}^{(i)} \cup \Delta^{(i+1)})$;
5. $\quad i := i + 1$;
6. } **until** $(crhs(\mathcal{C}^{(i+1)})|_X = crhs(\mathcal{C}^{(i)})|_X$ for all $X \in \mathcal{V}^\star)$;
   output $\mathcal{C}^{(i)}$;

Figure 3: **Main Algorithm**

[a] These are variables that $SIMPLIFY$ may introduce.

(b) just after every execution of the main loop, $\Delta^{(1)} \cup \ldots \cup \Delta^{(i)} \cup \mathcal{F}(\tilde{\mathcal{D}}^{(i)})$ *is in normal form.*

Proof: The essence of the proof is in the following property about a single application of one of the transformations that make up $REDUCE$:

if $\mathcal{C} \cup \mathcal{F}(\tilde{D})$ is in normal form and $\mathcal{C}, \tilde{\mathcal{D}} \underset{\text{tf}}{\Longrightarrow} \tilde{\mathcal{D}}', \delta$ then

- $\mathcal{C} \cup \delta \cup \mathcal{F}(\tilde{D}')$ is in normal form;
- $lhs(\delta) \subseteq lhs(\mathcal{F}(\tilde{D}))$, and
- for all $X \in lhs(\mathcal{F}(\tilde{D}'))$, either $X \in lhs(\mathcal{F}(\tilde{D}))$ or X is a new variable.

An easy induction argument using the above property then proves the following property about $REDUCE$ as a whole:

if $\mathcal{C} \cup \mathcal{F}(\tilde{\mathcal{D}})$ is in normal form and $(\tilde{\mathcal{D}}',\ \delta) = REDUCE(\mathcal{C},\ \tilde{\mathcal{D}})$
then $\mathcal{C} \cup \Delta \cup \mathcal{F}(\tilde{\mathcal{D}}')$ is in normal form.

We now prove prove (a), that is the following holds just after the execution of line 3 in any iteration:

$$\mathcal{C}^{(i)} \cup \Delta^{(i+1)} \cup \mathcal{F}(\tilde{D}^{(i+1)}) \text{ is in normal form.} \tag{6.16}$$

In the first iteration, $\mathcal{C}^{(0)} \cup \mathcal{F}(\tilde{D}^{(0)})$ is in normal form just before the loop is executed because of the main algorithm precondition. By the above property of $REDUCE$, $\mathcal{C}^{(0)} \cup \Delta^{(1)} \cup \mathcal{F}(\tilde{D}^{(1)})$ is in normal form just after line 3 is executed.

Now consider the $n+1^{th}$ iteration; thus i is n. Assume that (6.16) holds just after line 3 is executed. Thus after line 4 is executed, $\mathcal{C}^{(i+1)} \cup \mathcal{F}(\tilde{D}^{(i+1)})$ is in normal form because $\mathcal{C}^{(i+1)}$ is in solved form and any variable in the l.h.s. of $SIMPLIFY(\mathcal{C}^{(i)} \cup \Delta^{(i+1)})$ is either a variable from $lhs(\mathcal{C}^{(i)} \cup \Delta^{(i+1)})$ or a new variable. Thus after line 5 is executed, $\mathcal{C}^{(i)} \cup \mathcal{F}(\tilde{D}^{(i)})$ is in normal form. This means that if there is an $n+2^{th}$ iteration, we can use the above property of $REDUCE$ to show that (6.16) holds just after line 3 is executed again. Thus (a) is proved.

(b) Since (6.16) holds just after line 3 is executed, the following holds just after the subsequent execution of line 5:

$$\mathcal{C}^{(i-1)} \cup \Delta^{(i)} \cup \mathcal{F}(\tilde{D}^{(i)}) \text{ is in normal form.} \tag{6.17}$$

Now, by corollary 2, $SIMPLIFY$ preserves solved variables in the sense that $lhs(\mathcal{C}) \subseteq lhs(\mathcal{C}')$ if $\mathcal{C}' = SIMPLIFY(\mathcal{C})$. Since $\mathcal{C}^{(1)}$ is built from $\mathcal{C}^{(0)}$ and $\Delta^{(1)}$, $\mathcal{C}^{(2)}$ is built from $\mathcal{C}^{(1)}$ and $\Delta^{(2)}$, ... , it follows that $lhs(\Delta^{(1)}) \cup \ldots \cup lhs(\Delta^{(i)}) \subseteq lhs(\mathcal{C}^{(i)})$ and $lhs(\Delta^{(1)})$, ... , $lhs(\Delta^{(i)})$ are mutually disjoint. This, (6.17), and the previously proven fact (a) that $\Delta^{(1)}$, ... , $\Delta^{(i)}$ are each in normal form, proves (b). □

6.2 Termination

We prove, in two main parts, that $REDUCE(\mathcal{C}, \tilde{\mathcal{D}})$ terminates.

Lemma 3 (Termination of $REDUCE$)
(a) Each of the four while-loops terminates.
(b) The main repeat-loop terminates.

Proof: First consider the while-loop at line 2 which consists of applications of the D-substitution transformation. Since $\mathcal{C}$ is in solved form, each application of this transformation reduces the number of occurrences of un-nested $\mathcal{C}$-solved variables in $\tilde{\mathcal{D}}$. Thus this loop terminates.

Second, consider the while-loop at line 3 which consists of applications of the union and intersection transformations. Define $\#(\mathcal{D}_j)$ where $\mathcal{D}_j$ is a collection of pairs, to be 3^d where d is the number of $\cup$ and $\cap$ symbols in $\mathcal{D}_j$. Define $\nabla(\tilde{\mathcal{D}})$ to be

$$\sum_{\mathcal{D}_j \in \tilde{\mathcal{D}}} \#(\mathcal{D}_j).$$

In the case that an intersection transformation is applied, the only change is that a collection of pairs $\mathcal{D}_j$ is replaced by another collection $\mathcal{D}'_j$ which has one less $\cap$ symbol. It follows that $\#(\mathcal{D}'_j) < \#(\mathcal{D}_j)$. In the case than a union transformation is applied, the only change is that a collection of pairs $\mathcal{D}_j$, with say d $\cup$ and $\cap$ symbols, is replaced by two collections $\mathcal{D}'_j$ and $\mathcal{D}''_j$, each of which has $d-1$ $\cup$ and $\cap$ symbols. We thus have

$$\#(\mathcal{D}'_j) + \#(\mathcal{D}''_j) = 3^{d-1} + 3^{d-1} < 3^d = \#(\mathcal{D}_j)$$

and so, in the case of either transformation, $\nabla(\tilde{D})$ strictly decreases. Termination of the step at line 3 is proved.

Third, consider the while-loop at line 4 which consists of applications of the clash and cancel transformations. Consider the total number of symbols appearing on the left hand sides of pairs in $\tilde{\mathcal{D}}$. Since both the clash and cancel transformations strictly reduce this number, the termination of the step at line 4 is immediate.

Fourth, consider the while-loop at line 5 which consists of applications of the compaction transformation. Since each such application reduces the number of collections in $\tilde{\mathcal{D}}$ by one, the termination of the step at line 5 is immediate.

Finally, we consider part (b) and prove that the main loop in step 1 terminates. Let $depth(t)$ denote the maximum level of nesting within the term t; let $depth(p)$, where p is a pair $(a, sexp)$, be

$$\begin{cases} 0 & \text{if } sexp \text{ is a variable, but not a } \mathcal{C}\text{-solved variable;} \\ depth(a) & \text{otherwise.} \end{cases}$$

Let the metric $\nabla(\tilde{\mathcal{D}})$ be the largest depth of a pair in $\tilde{\mathcal{D}}$. Suppose that one iteration of the main repeat-loop is executed upon the family $\tilde{\mathcal{D}}$. Let $\tilde{\mathcal{D}}_2$ denote the result of applying the step at line 2 to $\tilde{\mathcal{D}}$; let $\tilde{\mathcal{D}}_3$ denote the result of applying the step at line 3 to $\tilde{\mathcal{D}}_3$; similarly define $\tilde{\mathcal{D}}_4$ and $\tilde{\mathcal{D}}_5$.

That no transformation increases the metric of the family to which it is applied can be easily verified. This is essentially because no transformation increases the

depth of the l.h.s. of any pair, and no transformation is applicable to a pair if its r.h.s. is a non-$\mathcal{C}$-solved variable. Thus:

$$\nabla(\tilde{\mathcal{D}}) \geq \nabla(\tilde{\mathcal{D}}_2) \geq \nabla(\tilde{\mathcal{D}}_3) \geq \nabla(\tilde{\mathcal{D}}_4) \geq \nabla(\tilde{\mathcal{D}}_5).$$

By design, $\tilde{\mathcal{D}}_2$ is closed under D-substitution. That is, the only variables appearing un-nested in the r.h.s. of pairs in $\tilde{\mathcal{D}}_2$ are non-$\mathcal{C}$-solved variables. In fact, $\tilde{\mathcal{D}}_3$ is also closed under D-substitution. This is because in a single application of a union or intersection transformation, the only variables appearing un-nested after the transformation are those which appear un-nested before the transformation. Thus $\tilde{\mathcal{D}}_3$ is closed under all three of the transformations D-substitution, union and intersection. We now consider $\tilde{\mathcal{D}}_3$ in two main cases.

(a) Suppose that $\tilde{\mathcal{D}}_3$ is closed under both the clash and cancel transformations. Then the body of the loop at line 4 will not be executed, and so $\tilde{\mathcal{D}}_4$ is $\tilde{\mathcal{D}}_3$. Thus $\tilde{\mathcal{D}}_4$ is closed under all but the compaction transformation. Now $\tilde{\mathcal{D}}_5$ is closed under the compaction transformation by design. Since $\tilde{\mathcal{D}}_5$ is obtained from $\tilde{\mathcal{D}}_4$ simply by the deletion of collections of pairs from the latter, $\tilde{\mathcal{D}}_5$ remains closed under the other transformations. Thus $\tilde{\mathcal{D}}_5$ is closed under *all* transformations, and the whole algorithm terminates.

(b) Now consider the alternative case in which a clash or cancel transformation is applicable to $\tilde{\mathcal{D}}_3$. Thus $\nabla(\tilde{\mathcal{D}}_3)$, call it k, is positive. Let $(\psi_1, \phi_1), \cdots, (\psi_m, \phi_m)$, $m \geq 1$, be those pairs in $\tilde{\mathcal{D}}_3$ whose depth is k (these are the pairs of maximal depth in $\tilde{\mathcal{D}}_3$). Now, for all $1 \leq i \leq n$:

- because $k \geq 1$, ψ_i is of the form $f(\tilde{t})$;
- because $k \geq 1$ and $\tilde{\mathcal{D}}_3$ is closed under the D-substitution, ϕ_i is not a variable; because $\tilde{\mathcal{D}}_3$ is also closed under the union and intersection transformations, ϕ_i is primitive; thus ϕ_i is of the form $g(\tilde{s})$.

It follows that, for all $1 \leq i \leq n$, there is an application of the clash or cancel transformation on $\tilde{\mathcal{D}}_3$ which removes the pair (ψ_i, ϕ_i), and replaces it with zero or more pairs of strictly smaller depth. Thus $\nabla(\tilde{\mathcal{D}}_4) < \nabla(\tilde{\mathcal{D}}_3)$.

In summary, if an iteration of the main repeat loop falls into the first case (a), then this is in fact the last iteration. If the iteration falls into the latter case (b) the value of $\nabla(\tilde{\mathcal{D}})$ strictly decreases. Thus the algorithm terminates. □

We now consider the termination of the main algorithm. Crucial to our proof is the following property which must hold at the end of every iteration of the main

loop:

Proposition 4 *Each nested variable appearing in the r.h.s. of a pair in $\tilde{\mathcal{D}}^{(i)}$, or in the r.h.s. of an equation in $\mathcal{C}^{(i)}$, is a variable from $\mathcal{V}^\star$.*

Proof: We proceed by induction. For the base case, it is immediate that the invariant is satisfied by $\mathcal{C}^{(0)}$ and $\tilde{\mathcal{D}}^{(0)}$.

For the inductive step, we first note that the only transformations used in the main algorithm where nested variables may be introduced are (a) the $\tilde{\mathcal{D}}$-substitute transformation, (b) the $\mathcal{C}$-substitute transformation of the $SIMPLIFY$ routine, and (c) the $\cap$-simplification transformation of the $SIMPLIFY$ routine.

Suppose that the invariant is satisfied by $\mathcal{C}$ and $\tilde{\mathcal{D}}$. Consider the application of the transformations noted in (a), (b) and (c) above to $\mathcal{C}$ and $\tilde{\mathcal{D}}$. Suppose that a $\tilde{\mathcal{D}}$-substitute transformation is applied. The effect of this transformation is to replace an un-nested variable with the r.h.s. of the corresponding equation in $\mathcal{C}$. Since the only nested variables in this expression are from $\mathcal{V}^\star$, the invariant continues to hold after this transformation. The case of the application of a $\mathcal{C}$-substitute transformation can similarly be argued. Finally, consider an application of the $\cap$-simplification transformation. Here, an expression of the form $f(X_1, \cdots, X_n) \cap f(X'_1, \cdots, X'_n)$, where $X_i, X'_i \in \mathcal{V}^\star$, $1 \leq i \leq n$, is replaced by an expression of the form $f(Y_1, \cdots, Y_n)$. Further, some equations may be added to $\mathcal{C}$ (line 11 of $SIMPLIFY$). We note that (i) by the construction of the transformation, the variables $Y_1, \cdots, Y_n$ are chosen to be from $\mathcal{V}^\star$, and (ii) if some equations are added to $\mathcal{C}$, then the r.h.s. of each new equation satisfies the invariant by the induction hypothesis. Thus proposition 4 is proved. □

Main Lemma 2 *The main algorithm terminates.*

Proof: Clearly it suffices to prove that the main loop at line 2 terminates. Toward this aim, we first note that every $\mathcal{C}^{(i)}$, $i \geq 0$, constructed by the algorithm, is in solved form. This is an immediate from lemma 2 on the correctness of $SIMPLIFY$. By proposition 4, any expression t in $crhs(\mathcal{C}^{(i)})|_X$ is of the form $f(\tilde{Y})$ where $\tilde{Y} \subseteq \mathcal{V}^\star$. The termination of the main algorithm now easily follows from the two facts:

- There are a finite number of distinct expressions of the form $f(\tilde{Y})$ where $\tilde{Y} \subseteq \mathcal{V}^\star$, and therefore the number of possibilities for the sets $crhs(\mathcal{C}^{(i)})|_X$, $X \in \mathcal{V}^\star$ is bounded.

- $crhs(\mathcal{C}^{(i)})|_X \subseteq crhs(\mathcal{C}^{(i+1)})|_X$. This is because any difference between $rhs(\mathcal{C}^{(i+1)})|_X$ and $rhs(\mathcal{C}^{(i)})|_X$ is due to the $SIMPLIFY$ routine, and it easy to verify that $SIMPLIFY$ does not delete or rewrite any conjunct of the form $f(\tilde{Y})$ where $\tilde{Y}$ are variables. □

6.3 Correctness

We next prove a series of basic facts about the transformations 6-11. In the following propositions, we will use the notation and terminology established in the definitions of the transformations. Now suppose that

$$\mathcal{C}, \tilde{\mathcal{D}} \Longrightarrow \tilde{\mathcal{D}}', \delta$$

that is, a transformation is applied in the context of $\mathcal{C}$ to $\tilde{\mathcal{D}}$ in order to obtain a new family $\tilde{\mathcal{D}}'$ and the extracted equations δ. Then:

Proposition 5 (D-Substitution) *If the D-substitution transformation is used, then the only difference between $\mathcal{F}(\tilde{\mathcal{D}})$ and $\mathcal{F}(\tilde{\mathcal{D}}')$ is that some equations $Y_1 = exp_1$, $\cdots$, $Y_k = exp_k$ in $\mathcal{F}(\tilde{\mathcal{D}})$ are replaced by some equations $Y_1 = exp'_1$, $\cdots$, $Y_k = exp'_k$ in $\mathcal{F}(\tilde{\mathcal{D}}')$ such that*

for all models $\mathcal{I}$ of $\mathcal{C}$, $\mathcal{I}(exp_i) = \mathcal{I}(exp'_i)$, $1 \leq i \leq k$.

Proof: Straightforward and hence omitted. □

Proposition 6 (Clash) *If the clash transformation is used, then the only difference between $\mathcal{F}(\tilde{\mathcal{D}})$ and $\mathcal{F}(\tilde{\mathcal{D}}')$ is that some equations $Y_1 = exp_1$, $\cdots$, $Y_k = exp_k$ in $\mathcal{F}(\tilde{\mathcal{D}})$ do not appear in $\mathcal{F}(\tilde{\mathcal{D}}')$. Furthermore*

for all interpretations $\mathcal{I}$, $\mathcal{I}(exp_i) = \emptyset$, $1 \leq i \leq k$.

Proof: Recall that D_j is the collection of pairs removed by the transformation, and let it be denoted by $(\tilde{\Psi}, \tilde{\Phi})$. $\mathcal{F}(\tilde{\mathcal{D}}_j)$ is therefore

$$\bigwedge_{1 \leq i \leq k} Y_i = \{y_i : \exists y_1 \cdots y_{i-1} y_{i+1} \cdots y_k \left(\tilde{\Psi}\mu \in \tilde{\Phi}\right)\}$$

where μ is the renaming substitution $[Y_1/y_1, \cdots, Y_k/y_k]$. We now show that, in all interpretations $\mathcal{I}$,

$$\mathcal{I}(\{y_i : \exists y_1 \cdots y_{i-1} y_{i+1} \cdots y_k \left(\tilde{\Psi}\mu \in \tilde{\Phi}\right)\}) = \emptyset, \quad 1 \leq i \leq k.$$

Suppose otherwise; then there must exist a ground substitution on the variables $y_1, \cdots, y_k$, call it θ, such that $\tilde{\Psi}\mu\theta \in \mathcal{I}(\tilde{\Phi})$ is *true*. Since $(f(t_1, \cdots, t_n), g(s_1, \cdots, s_m))$ is a pair in $(\tilde{\Psi}, \tilde{\Phi})$, we have that $f(t_1, \cdots, t_n)\mu\theta \in \mathcal{I}(g(s_1, \cdots, s_m))$. This is clearly absurd. □

Proposition 7 (Cancellation) *If the cancellation transformation is used, then the only difference between $\mathcal{F}(\tilde{\mathcal{D}})$ and $\mathcal{F}(\tilde{\mathcal{D}}')$ is that some equations $Y_1 = exp_1, \cdots, Y_k = exp_k$ in $\mathcal{F}(\tilde{\mathcal{D}})$ are replaced by some equations $Y_1 = exp'_1, \cdots, Y_k = exp'_k$ in $\mathcal{F}(\tilde{\mathcal{D}}')$ such that*

for all interpretations $\mathcal{I}$, $\mathcal{I}(exp_i) = \mathcal{I}(exp'_i)$, $1 \leq i \leq k$.

Proof: Let $\mathcal{D}_j$ be the collection in $\tilde{\mathcal{D}}$ which contained the replaced pair $(f(\tilde{s}), f(\tilde{t}))$ where $\tilde{s}$ and $\tilde{t}$ respectively denote $s_1, \cdots, s_n$ and $t_1, \cdots, t_n$. Thus the variables in the left-hand-sides of $\mathcal{D}_j$ are the abovementioned variables Y_1, $\cdots$, Y_k. Let the remaining pairs in $\mathcal{D}_j$ be denoted by $(\tilde{\Psi}, \tilde{\Phi})$. Now, the only difference between $\mathcal{F}(\tilde{\mathcal{D}})$ and $\mathcal{F}(\tilde{\mathcal{D}}')$ is that

$$\bigwedge_{1 \leq i \leq k} Y_i = \left\{ y_i : \exists y_1 \cdots y_{i-1} y_{i+1} \cdots y_k \begin{pmatrix} f(\tilde{s})\mu & \in & f(\tilde{t}) \\ \tilde{\Psi}\mu & \in & \tilde{\Phi} \end{pmatrix} \right\} \tag{6.18}$$

where $y_1, \cdots, y_k$ are new variables, and
μ denotes $[Y_1/y_1, \cdots, Y_k/y_k]$;

appears in $\mathcal{F}(\tilde{\mathcal{D}})$, whereas

$$\bigwedge_{1 \leq i \leq k} Y_i = \left\{ y_i : \exists y_1 \cdots y_{i-1} y_{i+1} \cdots y_k \begin{pmatrix} s_1\mu & \in & t_1 \\ & \cdots & \\ s_n\mu & \in & t_n \\ \tilde{\Psi}\mu & \in & \tilde{\Phi} \end{pmatrix} \right\} \tag{6.19}$$

where $y_1, \cdots, y_k$ are new variables, and
μ denotes $[Y_1/y_1, \cdots, Y_k/y_k]$;

appears in $\mathcal{F}(\tilde{\mathcal{D}}')$. Let $\mathcal{I}$ be any interpretation and fix i, $1 \leq i \leq k$. The r.h.s. of the i^{th} equations in (6.18) and (6.19), when interpreted by $\mathcal{I}$, respectively become

$$S_1 = \left\{ Y_i\theta : \begin{pmatrix} f(\tilde{s})\theta & \in & \mathcal{I}(f(\tilde{t})) \\ \tilde{\Psi}\theta & \in & \mathcal{I}(\tilde{\Phi}) \end{pmatrix} \right\},$$

$$S_2 = \left\{ Y_i\theta : \begin{pmatrix} s_1\theta & \in & \mathcal{I}(t_1) \\ & \cdots & \\ s_n\theta & \in & \mathcal{I}(t_n) \\ \tilde{\Psi}\theta & \in & \mathcal{I}(\tilde{\Phi}) \end{pmatrix} \right\}.$$

The remainder of the proof is simply that $S_1 = S_2$, an obvious fact. □

Proposition 8 (Compaction) *If the compaction transformation is used, then the only difference between $\mathcal{F}(\tilde{\mathcal{D}})$ and $\mathcal{F}(\tilde{\mathcal{D}}')$ is that some equations $Y_1 = exp_1$, $\cdots$, $Y_k = exp_k$ in $\mathcal{F}(\tilde{\mathcal{D}})$ do not appear in $\mathcal{F}(\tilde{\mathcal{D}}')$. Furthermore, δ is of the form $\{Y_1 = exp'_1, \cdots, Y_k = exp'_k\}$ where*

for all models $\mathcal{I}$ of $\mathcal{C}$, $\mathcal{I}(exp_i) = \mathcal{I}(exp'_i)$, $1 \leq i \leq k$.

Proof: Recall that we denote $\mathcal{D}_j$ by $(\tilde{X}, \tilde{\Phi})$; thus $var(\tilde{X})$ is $\{Y_1, \cdots Y_k\}$. Now, the only difference between $\mathcal{F}(\tilde{\mathcal{D}})$ and $\mathcal{F}(\tilde{\mathcal{D}}')$ is that

$$\bigwedge_{1 \leq i \leq k} Y_i = \left\{ y_i : \exists y_1 y_2 \cdots y_{i-1} y_{i+1} \cdots y_k \begin{pmatrix} \tilde{X}\mu & \in & \tilde{\Phi} \end{pmatrix} \right\} \tag{6.20}$$

where $y_1, y_2, \cdots, y_k$ are new variables, and
μ denotes $[Y_1/y_1, \cdots, Y_k/y_k]$;

appears in $\mathcal{F}(\tilde{\mathcal{D}})$ but not in $\mathcal{F}(\tilde{\mathcal{D}}')$. It is thus sufficient to prove that the conjunction of the extracted equations in δ

$$\bigwedge_{1 \leq i \leq k} \left(Y_i = DNF \left(\bigcap_{(Y_i,\phi) \in \mathcal{D}_j} \phi \right) \right) \tag{6.21}$$

is equivalent to the conjunction (6.20). Toward this aim, first rearrange the equations in (6.20) to obtain

$$\bigwedge_{1 \leq i \leq k} Y_i = \left\{ y_i : \exists y_1 y_2 \cdots y_{i-1} y_{i+1} \cdots y_k \begin{pmatrix} \tilde{Y_1}\mu & \in & \tilde{\Phi}|_{Y_1} \\ & \cdots & \\ \tilde{Y_k}\mu & \in & \tilde{\Phi}|_{Y_k} \end{pmatrix} \right\} \tag{6.22}$$

where $y_1, y_2, \cdots, y_k$ are new variables, and
μ denotes $[Y_1/y_1, \cdots, Y_k/y_k]$;

where $(\tilde{Y}_i, \tilde{\Phi}|_{Y_i})$, $1 \leq i \leq k$, denote the pairs in $\mathcal{D}_j$ whose l.h.s. is Y_i. We can further rewrite (6.22) into

$$\bigwedge_{1 \leq i \leq k} Y_i \;=\; \left\{ y_i : \left(\begin{array}{rcl} \exists y_1 & \in & \cap\left(\tilde{\Phi}|_{Y_1}\right) \\ & \cdots & \\ \exists y_{i-1} & \in & \cap\left(\tilde{\Phi}|_{Y_{i-1}}\right) \\ y_i & \in & \cap\left(\tilde{\Phi}|_{Y_i}\right) \\ \exists y_{i+1} & \in & \cap\left(\tilde{\Phi}|_{Y_{i+1}}\right) \\ & \cdots & \\ \exists y_k & \in & \cap\left(\tilde{\Phi}|_{Y_k}\right) \end{array} \right) \right\} \tag{6.23}$$

by (a) a suitable inward movement of quantifiers, and for all $1 \leq l \leq k$, (b) replacing $\tilde{Y}_l\mu \in \tilde{\Phi}|_{Y_l}$ by $Y_l\mu \in \cap\left(\tilde{\Phi}|_{Y_l}\right)$, and (c) replacing $Y_l\mu$ by y_l.

Since $NONEMPTY(\mathcal{C}, \cap(\tilde{\Phi}|_{Y_1})) \wedge \cdots \wedge NONEMPTY(\mathcal{C}, \cap(\tilde{\Phi}|_{Y_k}))$ is *true*, $\exists y_l \in \mathcal{I}\left(\cap\left(\tilde{\Phi}|_{Y_l}\right)\right)$, $1 \leq l \leq k$, is *true* in all models $\mathcal{I}$ of $\mathcal{C}$. Thus (6.23) reduces to

$$\bigwedge_{1 \leq i \leq k} Y_i \;=\; \left\{ y_i : \left(\begin{array}{rcl} y_i & \in & \cap\left(\tilde{\Phi}|_{Y_i}\right) \end{array} \right) \right\}$$

in all models of $\mathcal{C}$. It is now immediate that this formula is equivalent to (6.21). □

Proposition 9 (Union) *If the union transformation is used, then the only difference between $\mathcal{F}(\tilde{\mathcal{D}})$ and $\mathcal{F}(\tilde{\mathcal{D}}')$ is that the collection of equations $\mathcal{F}(\mathcal{D}_j)$ is replaced in $\mathcal{F}(\tilde{\mathcal{D}})$ by two collections $\mathcal{F}(\mathcal{D}'_j)$ and $\mathcal{F}(\mathcal{D}''_j)$ in $\mathcal{F}(\tilde{\mathcal{D}}')$. Furthermore, where $\mathcal{F}(\mathcal{D}_j)$ is of the form $Y_1 = exp_1, \cdots, Y_k = exp_k$,*

$\mathcal{F}(\mathcal{D}'_j)$ is of the form $Y'_1 = exp'_1, \cdots, Y'_k = exp'_k$;
$\mathcal{F}(\mathcal{D}''_j)$ is of the form $Y''_1 = exp''_1, \cdots, Y''_k = exp''_k$, and
for all interpretations $\mathcal{I}$, $\mathcal{I}(exp_i) = \mathcal{I}(exp'_i \cup exp''_i)$, $1 \leq i \leq k$.

Proof: The only difference between $\mathcal{F}(\tilde{\mathcal{D}})$ and $\mathcal{F}(\tilde{\mathcal{D}}')$ is that the formula

$$\bigwedge_{1 \leq i \leq k} Y_i \;=\; \left\{ y_i : \exists y_1 \cdots y_{i-1} y_{i+1} \cdots y_k \left(\begin{array}{rcl} s\mu & \in & t_1 \cup t_2 \\ \tilde{\Psi}\mu & \in & \tilde{\Phi} \end{array} \right) \right\} \tag{6.24}$$

where $y_1, y_2, \cdots, y_k$ are new variables, and
μ denotes $[Y_1/y_1, Y_2/y_2, \cdots\ Y_k/y_k]$;

appears in $\mathcal{F}(\tilde{\mathcal{D}})$, whereas the following two formulas

$$\bigwedge_{1 \leq i \leq k} Y_i' = \left\{ y_i' : \exists y_1' \cdots y_{i-1}' y_{i+1}' \cdots y_k' \left(\begin{array}{lcl} s\eta_1\mu_1 & \in & t_1 \\ \tilde{\Psi}\eta_1\mu_1 & \in & \tilde{\Phi} \end{array} \right) \right\} \tag{6.25}$$

where $y_1', y_2', \cdots, y_k'$ are new variables, and
μ_1 denotes $[Y_1'/y_1', Y_2'/y_2', \cdots Y_k'/y_k']$;

$$\bigwedge_{1 \leq i \leq k} Y_i'' = \left\{ y_i'' : \exists y_1'' \cdots y_{i-1}'' y_{i+1}'' \cdots y_k'' \left(\begin{array}{lcl} s\eta_2\mu_2 & \in & t_2 \\ \tilde{\Psi}\eta_2\mu_2 & \in & \tilde{\Phi} \end{array} \right) \right\} \tag{6.26}$$

where $y_1'', y_2'', \cdots, y_k''$ are new variables, and
μ_2 denotes $[Y_1''/y_1'', Y_2''/y_2'', \cdots Y_k''/y_k'']$;

appear in $\mathcal{F}(\tilde{\mathcal{D}}')$.

Fix an i, $1 \leq i \leq k$. Let rhs, rhs_1 and rhs_2 denote the r.h.s. of the i^{th} equations of the conjunctions (6.24), (6.25) and (6.26) respectively. To prove the lemma, it suffices to prove for all interpretations $\mathcal{I}$ that $\mathcal{I}(rhs) = \mathcal{I}(rhs_1) \cup \mathcal{I}(rhs_2)$.

Now let $\mathcal{I}$ be any interpretation; thus $\mathcal{I}(rhs)$ is

$$\left\{ Y_i\theta : \left(\begin{array}{lcl} s\theta & \in & \mathcal{I}(t_1 \cup t_2) \\ \tilde{\Psi}\theta & \in & \mathcal{I}(\tilde{\Phi}) \end{array} \right) \right\} \tag{6.27}$$

where θ ranges over ground substitutions on $Y_1, Y_2, \cdots, Y_k$. Similarly $\mathcal{I}(rhs_1)$ and $\mathcal{I}(rhs_2)$ respectively are

$$\left\{ Y_i'\theta_1 : \left(\begin{array}{lcl} s\eta_1\theta_1 & \in & \mathcal{I}(t_1) \\ \tilde{\Psi}\eta_1\theta_1 & \in & \mathcal{I}(\tilde{\Phi}) \end{array} \right) \right\} \tag{6.28}$$

$$\left\{ Y_i''\theta_2 : \left(\begin{array}{lcl} s\eta_2\theta_2 & \in & \mathcal{I}(t_2) \\ \tilde{\Psi}\eta_2\theta_2 & \in & \mathcal{I}(\tilde{\Phi}) \end{array} \right) \right\} \tag{6.29}$$

where η_1 is the renaming $[Y_1/Y_1', \cdots, Y_k/Y_k']$, η_2 is $[Y_1/Y_1'', \cdots, Y_k/Y_k'']$, θ_1 ranges over ground substitutions on $Y_1', \cdots, Y_k'$, and finally, θ_2 ranges over ground substitutions on $Y_1'', \cdots, Y_k''$.

Since η_1 is a renaming, $\eta_1\theta_1$ ranges over all ground substitutions on $Y_1, \cdots, Y_k$. Thus (6.28) can be written in the form

$$\left\{ Y_i\theta : \left(\begin{array}{lcl} s\theta & \in & \mathcal{I}(t_1) \\ \tilde{\Psi}\theta & \in & \mathcal{I}(\tilde{\Phi}) \end{array} \right) \right\}. \tag{6.30}$$

Similarly (6.29) can be written in the form

$$\left\{ Y_i\theta : \begin{pmatrix} s\theta & \in & \mathcal{I}(t_2) \\ \tilde{\Psi}\theta & \in & \mathcal{I}(\tilde{\Phi}) \end{pmatrix} \right\}. \tag{6.31}$$

Clearly (6.27) equals the union of (6.30) and (6.31), and so the proposition follows. □

Proposition 10 (Intersection) *If the intersection transformation is used, then the only difference between* $\mathcal{F}(\tilde{\mathcal{D}})$ *and* $\mathcal{F}(\tilde{\mathcal{D}}')$ *is that the collection of equations* $\mathcal{F}(\tilde{\mathcal{D}}) = \{Y_1 = exp_1, \cdots, Y_k = exp_k\}$ *is replaced by* $\mathcal{F}(\tilde{\mathcal{D}}') = \{Y_1 = exp'_1, \cdots, Y_k = exp'_k\}$. *Furthermore,*

for all interpretations $\mathcal{I}$, $\mathcal{I}(exp_i) = \mathcal{I}(exp'_i)$, $1 \leq i \leq k$.

Proof: Let $\mathcal{D}_j$ be of the form $\{(s, t_1 \cap t_2), (\tilde{\Psi}, \tilde{\Phi})\}$ so that this collection is replaced by $\{(s, t_1), (s, t_2), (\tilde{\Psi}, \tilde{\Phi})\}$ in $\tilde{\mathcal{D}}'$. Thus $var(s) \cup var(\tilde{\Psi})$ is $\{Y_1, \cdots, Y_k\}$, and the only difference between $\mathcal{F}(\tilde{\mathcal{D}})$ and $\mathcal{F}(\tilde{\mathcal{D}}')$ is that

$$\begin{aligned} \bigwedge_{1 \leq i \leq k} Y_i \quad = \quad & \left\{ y_i : \exists y_1 \cdots y_{i-1} y_{i+1} \cdots y_k \begin{pmatrix} s\mu & \in & t_1 \cap t_2 \\ \tilde{\Psi}\mu & \in & \tilde{\Phi} \end{pmatrix} \right\} \\ & \text{where } y_1, \cdots, y_k \text{ are new variables, and} \\ & \mu \text{ denotes } [Y_1/y_1, \cdots, Y_k/y_k]; \end{aligned} \tag{6.32}$$

appears in $\mathcal{F}(\tilde{\mathcal{D}})$, whereas the formula

$$\begin{aligned} \bigwedge_{1 \leq i \leq k} Y_i \quad = \quad & \left\{ y_i : \exists y_1 \cdots y_{i-1} y_{i+1} \cdots y_k \begin{pmatrix} s\mu & \in & t_1 \\ s\mu & \in & t_2 \\ \tilde{\Psi}\mu & \in & \tilde{\Phi} \end{pmatrix} \right\} \\ & \text{where } y_1, \cdots, y_k \text{ are new variables, and} \\ & \mu \text{ denotes } [Y_1/y_1, \cdots, Y_k/y_k]; \end{aligned} \tag{6.33}$$

appears in $\mathcal{F}(\tilde{\mathcal{D}}')$.

Fix an i, $1 \leq i \leq k$. Let rhs_1 and rhs_2 denote the r.h.s. of the i^{th} equations of the conjunctions (6.32) and (6.33) respectively. To prove the lemma it suffices to prove, for all interpretations $\mathcal{I}$, that $\mathcal{I}(rhs_1) = \mathcal{I}(rhs_2)$. Toward this aim, we rewrite $\mathcal{I}(rhs_1)$ and $\mathcal{I}(rhs_2)$ respectively into:

$$\left\{ Y_i\theta : \left(\begin{array}{rcl} s\theta & \in & \mathcal{I}(t_1 \cap t_2) \\ \tilde{\Psi}\theta & \in & \mathcal{I}(\tilde{\Phi}) \end{array} \right) \right\} \tag{6.34}$$

$$\left\{ Y_i\theta : \left(\begin{array}{rcl} s\theta & \in & \mathcal{I}(t_1) \\ s\theta & \in & \mathcal{I}(t_2) \\ \tilde{\Psi}\theta & \in & \mathcal{I}(\tilde{\Phi}) \end{array} \right) \right\} \tag{6.35}$$

where θ ranges over ground substitutions on the variables $Y_1, \cdots, Y_k$. That (6.34) and (6.35) are equivalent is evident. □

An important part of the proof of correctness involves treating set equations, such as $\mathcal{C}$ and δ, and quantified equations, such as $\mathcal{F}(\tilde{\mathcal{D}})$, as mappings from interpretations into interpretations. To define this mapping, let S be a collection of set and quantified equations, and let S be in normal form. Let $\mathcal{I}$ be an arbitrary interpretation. Now define that $[\![S]\!](\mathcal{I}) = \mathcal{I}'$ where

$$\mathcal{I}'(X) \stackrel{\text{def}}{=} \begin{cases} \mathcal{I}(exp) & \text{if } X = exp \text{ is an equation in } S; \\ \mathcal{I}(X) & \text{if } X \notin lhs(S). \end{cases}$$

Recalling the definition of $rhs(C)|_X$, the above definition can be more simply stated as $\left([\![S]\!](\mathcal{I})\right)(X) \stackrel{\text{def}}{=} \mathcal{I}(rhs(S)|_X)$. We write $[\![S]\!] = [\![S']\!]$ to denote that, for all interpretations $\mathcal{I}$, $[\![S]\!](\mathcal{I}) = [\![S']\!](\mathcal{I})$

We now consider some basic properties of the $[\![\ ]\!]$ mapping. In the following propositions, let T, S, S_0, S_1, ..., be collections of equations in normal form. Let $\mathcal{I}$, $\mathcal{I}_1$ and $\mathcal{I}_2$ denote interpretations. Our first proposition requires no proof:

Proposition 11 *$\mathcal{I}$ is a model of S iff $[\![S]\!](\mathcal{I}) = \mathcal{I}$.* □

Proposition 12 (Monotonicity of $[\![\ ]\!]$) *$\mathcal{I}_1 \subseteq \mathcal{I}_2$ implies $[\![S]\!](\mathcal{I}_1) \subseteq [\![S]\!](\mathcal{I}_2)$.*

Proof: Assume that $\mathcal{I}_1 \subseteq \mathcal{I}_2$. Observe that

(a) if exp is a set expression, then $\mathcal{I}_1(exp) \subseteq \mathcal{I}_2(exp)$;

(b) if exp is the expression $\left\{ y_i : \exists y_1 \ldots y_{i-1} y_{i+1} \ldots y_m \left(\tilde{\Psi}\mu \in \tilde{\Phi} \right) \right\}$ where $\{Y_1, \cdots, Y_m\}$ denotes the variables in $\tilde{\Psi}$, $y_1, y_2, \cdots, y_m$ are new variables, and μ denotes $[Y_1/y_1, Y_2/y_2, \cdots\ Y_m/y_m]$, then we first note that $\mathcal{I}_1(\tilde{\Phi}) \subseteq \mathcal{I}_2(\tilde{\Phi})$ by (a). It is now easy to see that $\mathcal{I}_1(exp) \subseteq \mathcal{I}_2(exp)$.

Thus $\mathcal{I}_1(exp) \subseteq \mathcal{I}_2(exp)$ in both the above cases. Hence, if $X \in lhs(S)$, that is $X = exp$ appears in S for some expression exp, then $\left(\llbracket S \rrbracket(\mathcal{I}_1)\right)(X) = \mathcal{I}_1(exp) \subseteq \mathcal{I}_2(exp) = \left(\llbracket S \rrbracket(\mathcal{I}_2)\right)(X)$. If, on the other hand, $X \notin lhs(X)$, then $\left(\llbracket S \rrbracket(\mathcal{I}_1)\right)(X) = \mathcal{I}_1(X) \subseteq \mathcal{I}_2(X) = \left(\llbracket S \rrbracket(\mathcal{I}_2)\right)(X)$. Hence $\llbracket S \rrbracket(\mathcal{I}_1) \subseteq \llbracket S \rrbracket(\mathcal{I}_2)$. □

Proposition 13 (Algebraic properties of $\llbracket\ \rrbracket$)
Let $lhs(S_0)$, $lhs(S_1)$, $lhs(S_2)$, ... be mutually disjoint.

(a) *Let $\llbracket T \rrbracket(\mathcal{I}) =_{\mathcal{V}} \llbracket S_1 \rrbracket \llbracket S_2 \rrbracket(\mathcal{I})$ for some $\mathcal{V}$.*
If $lhs(T)$ is disjoint from $lhs(S_0)$, and $var(rhs(S_1))$ is disjoint from $lhs(S_0)$,
then $\llbracket S_0 \cup T \rrbracket(\mathcal{I}) =_{\mathcal{V}} \llbracket S_1 \rrbracket \llbracket S_0 \cup S_2 \rrbracket(\mathcal{I})$

(b) *Let $\llbracket T \rrbracket(\mathcal{I}) = \llbracket S_1 \cup S_2 \rrbracket(\mathcal{I})$.*
If $lhs(T)$ is disjoint from $lhs(S_0)$,
then $\llbracket S_0 \cup T \rrbracket(\mathcal{I}) = \llbracket S_0 \cup S_1 \cup S_2 \rrbracket(\mathcal{I})$

(c) *Let $T \subseteq S_1 \cup \cdots \cup S_n$.*
If $\mathcal{I}$ is a model of T, and $\mathcal{I}(X) = \emptyset$ for all $X \in lhs(S_1 \cup \cdots \cup S_n) - lhs(T)$,
then $\mathcal{I} \subseteq \llbracket S_1 \rrbracket \llbracket S_2 \rrbracket \cdots \llbracket S_n \rrbracket(\mathcal{I})$

Proof:

(a) Let $\mathcal{I}$ be an interpretation and let X be any variable in $\mathcal{V}$. Consider now the exhaustive cases:

- $X \in lhs(S_0)$:
  Now $\left(\llbracket S_0 \cup T \rrbracket(\mathcal{I})\right)(X)$ is $\mathcal{I}(rhs(S_0 \cup T)|_X)$ by definition. Since $X \notin lhs(T)$, this reduces to $\mathcal{I}\left(rhs(S_0)|_X\right)$. Similarly, $\left(\llbracket S_1 \rrbracket \llbracket S_0 \cup S_2 \rrbracket(\mathcal{I})\right)(X)$ is $\left(\llbracket S_0 \cup S_2 \rrbracket(\mathcal{I})\right)(rhs(S_1)|_X)$ by definition. Since $X \notin lhs(S_1)$, this reduces to $\left(\llbracket S_0 \cup S_2 \rrbracket(\mathcal{I})\right)(X)$ which is $\mathcal{I}(rhs(S_0 \cup S_2)|_X)$ by definition. Since $X \notin lhs(S_2)$, this in turn reduces to $\mathcal{I}\left(rhs(S_0)|_X\right)$.

- $X \in lhs(S_1)$:
  Now $\left(\llbracket S_0 \cup T \rrbracket(\mathcal{I})\right)(X)$ is $\mathcal{I}(rhs(S_0 \cup T)|_X)$ by definition. Since $X \notin lhs(S_0)$, this reduces to $\mathcal{I}(rhs(T)|_X)$ which is $\left(\llbracket T \rrbracket(\mathcal{I})\right)(X)$ by definition. By assumption this is equal to $\left(\llbracket S_1 \rrbracket \llbracket S_2 \rrbracket(\mathcal{I})\right)(X)$ which can be rewritten in the form

$\mathcal{I}'(rhs(S_1)|_X)$ where $\mathcal{I}'$ is $[\![S_2]\!](\mathcal{I})$. Similarly, $\left([\![S_1]\!][\![S_0 \cup S_2]\!](\mathcal{I})\right)(X)$ can be rewritten in the form $\mathcal{I}''(rhs(S_1)|_X)$ where $\mathcal{I}''$ is $[\![S_0 \cup S_2]\!](\mathcal{I})$. The remainder of the proof is that $\mathcal{I}'(rhs(S_1)|_X) = \mathcal{I}''(rhs(S_1)|_X)$, and this follows from the fact that $\mathcal{I}'$ and $\mathcal{I}''$ differ only on $lhs(S_0)$.

- $X \notin lhs(S_0)$ and $X \notin lhs(S_1)$:
  Above, we have shown that if $X \notin lhs(S_0)$, then $\left([\![S_0 \cup T]\!](\mathcal{I})\right)(X)$ is $\left([\![S_1]\!][\![S_2]\!](\mathcal{I})\right)(X)$. This is $\left([\![S_2]\!](\mathcal{I})\right)(rhs(S_1)|_X)$ by definition. Since $X \notin lhs(S_1)$, this reduces to $\left([\![S_2]\!](\mathcal{I})\right)(X)$. Similarly, we have shown above that if $X \notin lhs(S_1)$, $\left([\![S_1]\!][\![S_0 \cup S_2]\!](\mathcal{I})\right)(X)$ reduces to $\left([\![S_0 \cup S_2]\!](\mathcal{I})\right)(X)$, which is $\mathcal{I}(rhs(S_0 \cup S_2)|_X)$ by definition. Since $X \notin lhs(S_0)$, this reduces to $\mathcal{I}(rhs(S_2)|_X)$, which is $\left([\![S_2]\!](\mathcal{I})\right)(X)$ by definition.

Thus for all $X \in \mathcal{V}$, $\left([\![S_0 \cup T]\!](\mathcal{I})\right)(X) = \left([\![S_1]\!][\![S_0 \cup S_2]\!](\mathcal{I})\right)(X)$ and so (a) is proved.

(b) Let $\mathcal{I}$ be an interpretation and let X be any variable. Suppose that $X \in lhs(S_0)$. Now $\left([\![S_0 \cup T]\!](\mathcal{I})\right)(X)$ is $\mathcal{I}\left(rhs(S_0 \cup T)|_X)\right)$ by definition. Since $X \notin lhs(T)$, this reduces to $\mathcal{I}\left(rhs(S_0)|_X)\right)$. Similarly, $\left([\![S_0 \cup S_1 \cup S_2]\!](\mathcal{I})\right)(X)$ is $\mathcal{I}\left(rhs(S_0 \cup S_1 \cup S_2)|_X)\right)$ by definition. Since $X \notin lhs(S_1)$ and $X \notin lhs(S_2)$, this reduces to $\mathcal{I}\left(rhs(S_0)|_X)\right)$, and the proof for this case is finished.

Suppose that $X \notin lhs(S_0)$. Arguing as above, we can show that $\left([\![S_0 \cup T]\!](\mathcal{I})\right)(X)$ reduces to $\left([\![T]\!](\mathcal{I})\right)(X)$, and similarly that $\left([\![S_0 \cup S_1 \cup S_2]\!](\mathcal{I})\right)(X)$ reduces to $\left([\![S_1 \cup S_2]\!](\mathcal{I})\right)(X)$. Since $\left([\![T]\!](\mathcal{I})\right)(X) = \left([\![S_1 \cup S_2]\!](\mathcal{I})\right)(X)$ by assumption, we are done.

(c) We first show that $[\![S_i]\!](\mathcal{I}) \supseteq \mathcal{I}$ for all $1 \leq i \leq n$. Let X be any variable. If $X \notin lhs(S_i)$, then $\left([\![S_i]\!](\mathcal{I})\right)(X)$ reduces to $\mathcal{I}(X)$. If $X \in lhs(S_i)$ but $X \notin lhs(T)$, then $\mathcal{I}(X) = \emptyset$ by assumption. The remaining alternative is $X \in lhs(S_i)$ and $X \in lhs(T)$. Now $\left([\![S_i]\!](\mathcal{I})\right)(X)$ is $\mathcal{I}(rhs(S_i)|_X)$ by definition. This is equal to $\mathcal{I}(X)$ because the equation $X = rhs(S_i)|_X$ appears in T and $\mathcal{I}$ is a model of T. Thus $[\![S_i]\!](\mathcal{I}) \supseteq \mathcal{I}$ for all $1 \leq i \leq n$.

Using the monotonicity of $[\![S_1]\!], \cdots, [\![S_{i-1}]\!]$, we have

$$[\![S_1]\!] \cdots [\![S_{i-1}]\!][\![S_i]\!](\mathcal{I}) \supseteq [\![S_1]\!] \cdots [\![S_{i-1}]\!](\mathcal{I})$$

for all $1 \leq i \leq n$. This, in turn, can be repeatedly used in the following chain of

reasoning:

$$\begin{array}{ll} & [\![S_1]\!] \cdots [\![S_{n-1}]\!][\![S_n]\!](\mathcal{I}) \\ \supseteq & [\![S_1]\!] \cdots [\![S_{n-1}]\!](\mathcal{I}) \\ & \vdots \\ \supseteq & [\![S_1]\!](\mathcal{I}) \\ \supseteq & \mathcal{I}. \end{array}$$

and the proof of (c) is finished. □

In what follows, the $[\![\ ]\!]$ mapping will be used to relate two formulas where one is obtained from the other via a number of applications of the transformations. For example, let the result of $REDUCE(\mathcal{C}, \tilde{\mathcal{D}})$ be $(\tilde{\mathcal{D}}', \Delta)$. Consider the sequence $\delta_1, \cdots, \delta_n$ of extracted collections, in chronological order, which constitute Δ. Let $\delta_1^u, \cdots, \delta_k^u$ be subsequence of $\delta_1, \cdots, \delta_n$ containing just those δ's extracted by the union transformation; similarly define $\delta_1^c, \cdots, \delta_m^c$ to be the subsequence extracted by the clash and compaction transformations. The relationship between the original formula and its $REDUCE$'d counterpart is essentially that

$$[\![\mathcal{C}]\!][\![\mathcal{F}(\mathcal{D})]\!] = [\![\mathcal{C}]\!][\![\delta_1^u]\!] \cdots [\![\delta_k^u]\!][\![\delta_1^c \cup \cdots \cup \delta_m^c \cup \mathcal{F}(\mathcal{D}')]\!].$$

We now formally prove that the transformations are correct in two senses: one, that they preserve models, and two, that they preserve the mapping $[\![\ ]\!]$ as explained above. The next lemma deals with the base case in which just one transformation is applied.

Lemma 4 (Correctness of One Application of a Transformation) *Let* $\mathcal{C}, \tilde{\mathcal{D}} \underset{tf}{\Longrightarrow} \tilde{\mathcal{D}}', \delta$. *Let* $\mathcal{V} = VAR - (var(\tilde{\mathcal{D}} \cup \delta) - var(\tilde{\mathcal{D}}))$, *that is,* $\mathcal{V}$ *contains all variables except the new variables introduced by this transformation. Then for all models* $\mathcal{I}$ *of* $\mathcal{C}$*:*

(a) $\mathcal{C} \wedge \mathcal{F}(\tilde{\mathcal{D}}) \underset{\mathcal{V}}{\Longleftrightarrow} \mathcal{C} \wedge \delta \wedge \mathcal{F}(\tilde{\mathcal{D}}')$;

(b) $$[\![\mathcal{F}(\tilde{\mathcal{D}})]\!](\mathcal{I}) =_{\mathcal{V}} \begin{cases} [\![\delta]\!][\![\mathcal{F}(\tilde{\mathcal{D}}')]\!](\mathcal{I}) & \text{if } tf \text{ is the union transformation;} \\ [\![\delta \cup \mathcal{F}(\tilde{\mathcal{D}}')]\!](\mathcal{I}) & \text{if } tf \text{ is clash or compaction;} \\ [\![\mathcal{F}(\tilde{\mathcal{D}}')]\!](\mathcal{I}) & \text{if } tf \text{ is D-substitution, cancel or intersection.} \end{cases}$$

Proof: First, suppose that the transformation used is either D-substitution, cancellation or intersection. Then $\delta = \emptyset$ and by propositions 5, 7 and 10, the only difference between $\mathcal{F}(\tilde{\mathcal{D}}')$ and $\mathcal{F}(\tilde{\mathcal{D}})$ is that some equations $Y_1 = exp_1, \cdots, Y_k = exp_k$

in $\mathcal{F}(\tilde{\mathcal{D}})$ are replaced by some equations $Y_1 = exp'_1, \cdots, Y_k = exp'_k$ in $\mathcal{F}(\tilde{\mathcal{D}}')$ such that for all models $\mathcal{I}$ of $\mathcal{C}$, $\mathcal{I}(exp_i) = \mathcal{I}(exp'_i)$, $1 \leq i \leq k$. It follows that:

$$\mathcal{C} \wedge \mathcal{F}(\tilde{\mathcal{D}}) \iff \mathcal{C} \wedge \mathcal{F}(\tilde{\mathcal{D}}')$$

and so (a) is proved. Consider now $[\![\mathcal{F}(\tilde{\mathcal{D}})]\!]$ and $[\![\mathcal{F}(\tilde{\mathcal{D}}')]\!]$, and let $\mathcal{I}$ be a model of $\mathcal{C}$. If Y is one of $Y_1, \cdots, Y_k$, say Y_i, then

$$\left([\![\mathcal{F}(\tilde{\mathcal{D}})]\!](\mathcal{I})\right)(Y) = \mathcal{I}(exp_i) = \mathcal{I}(exp'_i) = \left([\![\mathcal{F}(\tilde{\mathcal{D}}')]\!](\mathcal{I})\right)(Y).$$

If, however, Y is different from all of $Y_1, \cdots, Y_k$, then it follows immediately that

$$\left([\![\mathcal{F}(\tilde{\mathcal{D}})]\!](\mathcal{I})\right)(Y) = \left([\![\mathcal{F}(\tilde{\mathcal{D}}')]\!](\mathcal{I})\right)(Y)$$

because $Y = exp$ appearing in $\mathcal{F}(\tilde{\mathcal{D}})$ implies that $Y = exp$ appears also in $\mathcal{F}(\tilde{\mathcal{D}}')$, and vice-versa. Hence $[\![\mathcal{F}(\tilde{\mathcal{D}})]\!](\mathcal{I}) = [\![\mathcal{F}(\tilde{\mathcal{D}}')]\!](\mathcal{I})$ and so (b) is proved. This completes the proof of the lemma in the case of the D-substitution, cancellation or intersection transformation.

Suppose now that the clash or compaction transformation is applied. Let δ be $\{Y_1 = exp'_1, \cdots, Y_k = exp'_k\}$, and let $\{Y_1 = exp_1, \cdots, Y_k = exp_k\}$ be the equations whose l.h.s. are $Y_1, \cdots, Y_k$ in $\mathcal{F}(\tilde{\mathcal{D}})$. (Thus, in the case of clash, the exp'_i are $\perp$, and in the case of compaction, the exp'_i are conjunctions.) By the clash and compaction propositions 6 and 8, we have that:

$$\begin{array}{ll} & \mathcal{C} \wedge \mathcal{F}(\tilde{\mathcal{D}}) \\ \iff & \mathcal{C} \wedge \{Y_1 = exp_1, \cdots, Y_k = exp_k\} \wedge \mathcal{F}(\tilde{\mathcal{D}}') \\ \iff & \mathcal{C} \wedge \{Y_1 = exp'_1, \cdots, Y_k = exp'_k\} \wedge \mathcal{F}(\tilde{\mathcal{D}}') \\ \iff & \mathcal{C} \wedge \delta \wedge \mathcal{F}(\tilde{\mathcal{D}}'). \end{array}$$

Condition (a) is thus proved. To show (b), let $\mathcal{I}$ be model of $\mathcal{C}$. If Y is one of $Y_1, \cdots, Y_k$, say Y_i, then

$$\left([\![\mathcal{F}(\tilde{\mathcal{D}})]\!](\mathcal{I})\right)(Y) = \mathcal{I}(exp'_i) = \mathcal{I}(exp_i) = \left([\![\delta \cup \mathcal{F}(\tilde{\mathcal{D}})]\!](\mathcal{I})\right)(Y)$$

If Y is not one of $Y_1, \cdots, Y_k$, then it is immediate that $\left([\![\mathcal{F}(\tilde{\mathcal{D}})]\!](\mathcal{I})\right)(Y) = \left([\![\delta \cup \mathcal{F}(\tilde{\mathcal{D}})]\!](\mathcal{I})\right)(Y)$. This proves (b), and hence finishes the proof of the lemma for in the case of the clash and compaction transformations.

Suppose now that the union transformation is applied. From the definition of this transformation and from proposition 9 we have that:

- δ is $\{Y_1 = Y_1' \cup Y_1'', \cdots, Y_k = Y_k' \cup Y_k''\}$;
- $\mathcal{F}(\tilde{\mathcal{D}})$ is $\{Y_1 = exp_1, \cdots, Y_k = exp_k\} \cup \mathcal{F}(\tilde{\mathcal{D}}'')$;
- $\mathcal{F}(\tilde{\mathcal{D}}')$ is $\{Y_1' = exp_1', Y_1'' = exp_1'', \cdots, Y_k' = exp_k', Y_k'' = exp_k''\} \cup \mathcal{F}(\tilde{\mathcal{D}}'')$; and
- $\mathcal{I}(exp_i) = \mathcal{I}(exp_i') \cup \mathcal{I}(exp_i'')$, $1 \leq i \leq k$, for any model $\mathcal{I}$ of $\mathcal{C}$.

where $\tilde{\mathcal{D}}''$ is $\tilde{\mathcal{D}} - \{\mathcal{D}_j\}$. Now let $\mathcal{I}$ be a model of $\mathcal{C} \wedge \mathcal{F}(\tilde{D})$. Adapt $\mathcal{I}$ to $\mathcal{I}'$ by defining that $\mathcal{I}'(Y_i') = \mathcal{I}(exp_i')$ and $\mathcal{I}'(Y_i'') = \mathcal{I}(exp_i'')$, $1 \leq i \leq k$. Clearly $\mathcal{I}'$ satisfies

$$\begin{array}{lll} \mathcal{C} & \wedge & \{Y_1 = Y_1' \cup Y_1'', \cdots, Y_k = Y_k' \cup Y_k''\} \\ & \wedge & \{Y_1' = exp_1', Y_1'' = exp_1'', \cdots, Y_k' = exp_k', Y_k'' = exp_k''\} \wedge \mathcal{F}(\tilde{\mathcal{D}}'') \end{array}$$

and so one direction of (a) is proved. The other direction can be shown as follows. Let $\mathcal{I}$ be model of the formula immediately above. Clearly $\mathcal{I}$ models $\{Y_1 = exp_1, \cdots, Y_k = exp_k\}$ because

$$\mathcal{I}(Y_i) = \mathcal{I}(Y_i') \cup \mathcal{I}(Y_i'') = \mathcal{I}(exp_i') \cup \mathcal{I}(exp_i'') = \mathcal{I}(exp_i).$$

We now show that (b) holds in the case of the union transformation. If Y is one of $Y_1, \cdots, Y_k$, say Y_i, then

$$\begin{array}{ll} & \left(\left[\!\left[\mathcal{F}(\tilde{\mathcal{D}})\right]\!\right](\mathcal{I})\right)(Y) \\ = & \mathcal{I}(exp_i) \\ = & \mathcal{I}(exp_i') \cup \mathcal{I}(exp_i'') \\ = & \left(\left[\!\left[\mathcal{F}(\tilde{\mathcal{D}}')\right]\!\right](\mathcal{I})\right)(Y_i' \cup Y_i'') \\ = & \left(\left[\!\left[\mathcal{F}(\tilde{\mathcal{D}}')\right]\!\right](\mathcal{I})\right)(rhs(\delta)|_Y) \\ = & \left(\left[\!\left[\delta\right]\!\right]\left[\!\left[\mathcal{F}(\tilde{\mathcal{D}}')\right]\!\right](\mathcal{I})\right)(Y). \end{array}$$

If $Y \in lhs(\mathcal{F}(\tilde{\mathcal{D}})) - \{Y_1, \cdots, Y_k\}$, then it is immediate that $\left(\left[\!\left[\mathcal{F}(\tilde{\mathcal{D}})\right]\!\right](\mathcal{I})\right)(Y) = \left(\left[\!\left[\delta\right]\!\right]\left[\!\left[\mathcal{F}(\tilde{\mathcal{D}}')\right]\!\right](\mathcal{I})\right)(Y)$ because the equation for Y in $\mathcal{F}(\tilde{\mathcal{D}})$ and $\mathcal{F}(\tilde{\mathcal{D}}')$ is the same, and there is no equation for Y in δ. The remaining case, that of

$Y \in \mathcal{V} - lhs(\mathcal{F}(\tilde{\mathcal{D}}))$, is also immediate because there is no equation for Y anywhere. This proves (b), and hence the lemma is proved. □

The above lemma proved equivalences of $[\![\,]\!]$ mappings for a single application of a transformation. The equivalences were defined in three different ways depending on which transformation was used. The following lemma combines these equivalences over single applications into an equivalence over the sequence of applications generated by the entire algorithm.

Lemma 5 (Correctness of a Series of Transformations) *Let $\mathcal{C}^{(0)}$ and $\tilde{\mathcal{D}}^{(0)}$, satisfying the main algorithm precondition, be input to the main algorithm, and let the algorithm terminate after $n+1$ iterations, $n \geq 0$. Let δ_1^u, δ_2^u, $\cdots$, δ_k^u denote the sequence of equations extracted by applications of the union transformation, and similarly, let δ_1^c, δ_2^c, $\cdots$, δ_m^c denote the sequence of equations extracted by applications of the clash and compaction transformations. Then, for all models $\mathcal{I}$ of $\mathcal{C}^{(n)}$:*

$$[\![\mathcal{F}(\tilde{\mathcal{D}}^{(0)})]\!](\mathcal{I}) =_{var(\mathcal{C}^{(0)},\tilde{\mathcal{D}}^{(0)})} [\![\delta_1^u]\!] \cdots [\![\delta_k^u]\!][\![\delta_1^c \cup \cdots \cup \delta_m^c \cup \mathcal{F}(\tilde{\mathcal{D}}^{(n+1)})]\!](\mathcal{I}). \quad (6.36)$$

Proof: First, note that the main loop invariant (proposition 3) implies that the $k+1$ collections δ_1^u, ..., δ_k^u, and $\delta_1^c \cup \cdots \cup \delta_m^c \cup \mathcal{F}(\tilde{\mathcal{D}}_p)$ are in normal form. This proves that all of the $[\![\ ]\!]$ expressions in equation (6.36) are well defined.

Before proving (6.36) we establish some notation. Let $\mathcal{C}_j, \tilde{\mathcal{D}}_j \Longrightarrow \tilde{\mathcal{D}_{j+1}}, \delta_{j+1}$, $0 \leq j \leq p-1$, describe the sequence of transformations applied during the execution of the algorithm. Thus, by appropriate grouping, the sequence $\mathcal{C}_0$, $\mathcal{C}_1$, $\cdots$ $\mathcal{C}_{p-1}$ may be written as

$$\begin{array}{llll}
\mathcal{C}^{(0)}, & \mathcal{C}^{(0)}, & \cdots \ \mathcal{C}^{(0)}, & (\Pi_1 \textit{ copies, during first invocation of } REDUCE) \\
\mathcal{C}^{(1)}, & \mathcal{C}^{(1)}, & \cdots \ \mathcal{C}^{(1)}, & (\Pi_2 \textit{ copies, during second invocation of } REDUCE) \\
 & \vdots & & \\
\mathcal{C}^{(n)}, & \mathcal{C}^{(n)}, & \cdots \ \mathcal{C}^{(n)}, & (\Pi_{n+1} \textit{ copies, during } (n+1)^{th} \textit{ invocation of } REDUCE)
\end{array}$$

where $p = \Pi_1 + \cdots + \Pi_{n+1}$, and each Π_i denotes the number of transformations applied during the i^{th} invocation of $REDUCE$. We also have that

$$\begin{array}{rcl}
\Delta^{(1)} & = & \delta_1 \cup \cdots \cup \delta_{\Pi_1}, \\
\mathcal{C}^{(1)} & = & SIMPLIFY(\mathcal{C}^{(0)} \cup \delta_1 \cup \cdots \cup \delta_{\Pi_1}), \\
\Delta^{(2)} & = & \delta_{\Pi_1+1} \cup \cdots \cup \delta_{\Pi_2} \\
\mathcal{C}^{(2)} & = & SIMPLIFY(\mathcal{C}^{(1)} \cup \delta_{\Pi_1+1} \cup \cdots \cup \delta_{\Pi_2}), \\
 & \cdots & \\
\Delta^{(n+1)} & = & \delta_{\Pi_n+1} \cup \cdots \cup \delta_{\Pi_{n+1}}, \\
\mathcal{C}^{(n+1)} & = & SIMPLIFY(\mathcal{C}^{(n)} \cup \delta_{\Pi_n+1} \cup \cdots \cup \delta_{\Pi_{n+1}}),
\end{array}$$

Now, from corollary 2, if $\mathcal{C}' = SIMPLIFY(\mathcal{C})$ then any model of $\mathcal{C}'$ is a model of $\mathcal{C}$. Repeated application of this fact proves that $\mathcal{I}$ is not only a model of $\mathcal{C}^{(n)}$, but also a model of $\mathcal{C}^{(0)}$, ..., $\mathcal{C}^{(n-1)}$. Hence $\mathcal{I}$ is a model of $\mathcal{C}_0$, ..., $\mathcal{C}_{p-1}$.

To prove equation (6.36) we prove by induction that for all $0 \leq j \leq p$,

$$[\![\mathcal{F}(\tilde{\mathcal{D}}_0)]\!](\mathcal{I}) =_{var(\mathcal{C}^{(0)}, \tilde{\mathcal{D}}^{(0)})} [\![\delta_1^u]\!] \cdots [\![\delta_{k'}^u]\!] [\![\delta_1^c \cup \cdots \cup \delta_{m'}^c \cup \mathcal{F}(\tilde{\mathcal{D}}_j)]\!](\mathcal{I})$$

where δ_1^u, δ_2^u, $\cdots$, $\delta_{k'}^u$, $1 \leq k' \leq k$, is the subsequence of δ_1, δ_2, $\cdots$, δ_j corresponding to applications of the union transformation and similarly δ_1^c, δ_2^c, $\cdots$, $\delta_{m'}^c$, $1 \leq m' \leq m$, is the subsequence corresponding to applications of the clash and compaction transformations. The lemma is then immediate from the case of $j = p$.

The base case of $j = 0$ is trivially true. Assume now that the lemma holds for a given j. The $(j+1)^{th}$ transformation is either (a) a union transformation, (b) a clash or compaction transformation, or (c) one of the remaining transformations. Let δ_{j+1} be the equations extracted by this transformation, and consider these cases in turn:

(a) Let $\mathcal{V}_{new}$ be the new variables introduced by this application of the union transformation, that is, the variables $\Big(var(\tilde{\mathcal{D}}_{j+1}) \cup var(\delta_{j+1})\Big) - var(\tilde{\mathcal{D}}_j)$. Let $\mathcal{V}_{old}$ denote $VAR - \mathcal{V}_{new}$. By lemma 4, we then have

$$[\![\mathcal{F}(\tilde{\mathcal{D}}_j)]\!](\mathcal{I}) =_{\mathcal{V}_{old}} [\![\delta_{j+1}]\!] [\![\mathcal{F}(\tilde{\mathcal{D}}_{j+1})]\!](\mathcal{I}).$$

By inspection of the union transformation, we have that $rhs(\delta_{j+1})$ contains only new variables, that is, $var(rhs(\delta_{j+1})) \subseteq \mathcal{V}_{new}$. This means that $var(rhs(\delta_{j+1})$ and $lhs(\delta_1^c \cup \cdots \cup \delta_{m'}^c)$ are disjoint. We can therefore apply proposition 13(a) (by substituting $\mathcal{F}(\tilde{\mathcal{D}}_j)$, $\delta_1^c \cup \cdots \cup \delta_{m'}^c$, δ_{j+1} and $\mathcal{F}(\tilde{\mathcal{D}}_{j+1})$ for T, S_0, S_1 and S_2 in the proposition) to obtain

$$[\![\delta_1^c \cup \cdots \cup \delta_{m'}^c \cup \mathcal{F}(\tilde{\mathcal{D}}_j)]\!](\mathcal{I}) =_{\mathcal{V}_{old}} [\![\delta_{j+1}]\!] [\![\delta_1^c \cup \cdots \cup \delta_{m'}^c \cup \mathcal{F}(\tilde{\mathcal{D}}_{j+1})]\!](\mathcal{I}).$$

Now since $[\![\delta_1^u]\!][\![\delta_2^u]\!]\cdots[\![\delta_{k'}^u]\!]$ contains only variables from $\mathcal{V}_{old}$, we have

$$\begin{array}{ll} & [\![\delta_1^u]\!]\cdots[\![\delta_{k'}^u]\!][\![\delta_1^c \cup \cdots \cup \delta_{m'}^c \cup \mathcal{F}(\tilde{\mathcal{D}}_j)]\!](\mathcal{I}) \\ =_{\mathcal{V}_{old}} & [\![\delta_1^u]\!]\cdots[\![\delta_{k'}^u]\!][\![\delta_{j+1}]\!][\![\delta_1^c \cup \cdots \cup \delta_{m'}^c \cup \mathcal{F}(\tilde{\mathcal{D}}_{j+1})]\!](\mathcal{I}). \end{array}$$

On renaming δ_{j+1} into $\delta_{k'+1}^u$, and noting that $var(\mathcal{C}^{(0)}, \tilde{\mathcal{D}}^{(0)}) \subseteq \mathcal{V}_{old}$, the hypothesis for $j+1$ follows directly from the hypothesis for j.

(b) By lemma 4, we have that

$$[\![\mathcal{F}(\tilde{\mathcal{D}}_j)]\!](\mathcal{I}) = [\![\delta_{j+1} \cup \mathcal{F}(\tilde{\mathcal{D}}_{j+1})]\!](\mathcal{I}),$$

and so by proposition 13(b) (in which we substitute $\mathcal{F}(\tilde{\mathcal{D}}_j)$ for T, δ_{j+1} for S_1, $\mathcal{F}(\tilde{\mathcal{D}}_{j+1})$ for S_2 and $\delta_1^c \cup \cdots \cup \delta_{m'}^c$ for S_0)

$$[\![\delta_1^c \cup \cdots \cup \delta_{m'}^c \cup \mathcal{F}(\tilde{\mathcal{D}}_j)]\!](\mathcal{I}) = [\![\delta_1^c \cup \cdots \cup \delta_{m'}^c \cup \delta_{j+1} \cup \mathcal{F}(\tilde{\mathcal{D}}_{j+1})]\!](\mathcal{I}).$$

On applying $[\![\delta_1^u]\!][\![\delta_2^u]\!]\cdots[\![\delta_{k'}^u]\!]$ to both sides of this equation and renaming δ_{j+1} into $\delta_{m'+1}^u$, the hypothesis for $j+1$ follows directly from the hypothesis for j.

(c) In the case that the transformation is not union, clash or compaction, lemma 4 implies that $[\![\mathcal{F}(\tilde{\mathcal{D}}_{j+1})]\!](\mathcal{I}) = [\![\mathcal{F}(\tilde{\mathcal{D}}_j)]\!](\mathcal{I})$. By proposition 13(b) (in which S_1 is substituted by $\emptyset$) we have

$$[\![\delta_1^c \cup \cdots \cup \delta_{m'}^c \cup \mathcal{F}(\tilde{\mathcal{D}}_j)]\!](\mathcal{I}) = [\![\delta_1^c \cup \cdots \cup \delta_{m'}^c \cup \mathcal{F}(\tilde{\mathcal{D}}_{j+1})]\!](\mathcal{I}).$$

Once again, on applying $[\![\delta_1^u]\!][\![\delta_2^u]\!]\cdots[\![\delta_{k'}^u]\!]$ to both sides of this equation, the hypothesis for $j+1$ follows directly from the hypothesis for j. □

Lemma 6 (Termination Property of Transformations)
If no transformations are applicable on $\tilde{\mathcal{D}}$ in the context of $\mathcal{C}$ then $[\![\mathcal{F}(\tilde{\mathcal{D}})]\!](lm(\mathcal{C}))$ assigns $\emptyset$ to all variables in $lhs(\mathcal{F}(\tilde{\mathcal{D}}))$.

Proof: Let $Y_i = exp$ be an equation in $\mathcal{F}(\tilde{\mathcal{D}})$. Then exp is of the form

$$\{y_i : \exists y_1 \cdots y_{i-1} y_{i+1} \cdots y_k \quad \left(\tilde{\Psi}\mu, \tilde{\Phi} \right) \}$$

where $var(\tilde{\Psi}) = \{Y_1, \cdots, Y_k\}$,
$y_1, \cdots, y_k$ are new variables, and
μ denotes the renaming substitution
$[Y_1/y_1, \cdots\ Y_k/y_k]$.

Since no transformation is applicable to $(\tilde{\Psi}, \tilde{\Phi})$ in the context of $\mathcal{C}$, then $(\tilde{\Psi}, \tilde{\Phi})$ cannot contain any pairs of the form $(t, s \cup s')$, $(t, s \cap s')$, $(f(\tilde{t}), f'(\tilde{s}))$, or (t, V) where V is a solved variable in $\mathcal{C}$. In fact, either:

- $(\tilde{\Psi}, \tilde{\Phi})$ contains a pair of the form (t, V), for some variable V, and V is not a solved variable in $\mathcal{C}$; or
- $(\tilde{\Psi}, \tilde{\Phi})$ is of the form $(\tilde{X}, \tilde{\Phi})$ and for some variable Y in $\tilde{X}$, it is not the case that

  $NONEMPTY(\mathcal{C}, \bigcap_{(Y,\phi)\in(\tilde{\Psi},\tilde{\Phi})} \phi)$.

In either case, we have that there are no values for the variables in $\tilde{\Psi}$ such that $\tilde{\Psi} \in (lm(\mathcal{C}))(\tilde{\Phi}))$. Hence $lm(\mathcal{C})(exp)$ is empty, and the result follows. □

Using this notation, the next proposition states an obvious property of the collections $\mathcal{C}^{(i)}$.

Main Lemma 3 (Correctness of Main Algorithm) *Let $\mathcal{C}^{(0)}$ and $\tilde{\mathcal{D}}^{(0)}$ satisfy the main algorithm precondition. Let $\mathcal{C}^{(0)}$ and $\tilde{\mathcal{D}}^{(0)}$ be input to the main algorithm, and let $\mathcal{C}$ be output. Then,*

- $lm(\mathcal{C}) =_{var(\mathcal{C}^{(0)}, \tilde{\mathcal{D}}^{(0)})} lm(\mathcal{C}^{(0)} \wedge \mathcal{F}(\tilde{\mathcal{D}}^{(0)}))$, *and*
- $\mathcal{C}$ *is in solved form*

Proof: We first dispense with proving that $\mathcal{C}$ is in solved form. This follows directly from the fact that $SIMPLIFY$ returns equations in solved form. The remainder of this proof is that $lm(\mathcal{C}) =_{var(\mathcal{C}^{(0)}, \tilde{\mathcal{D}}^{(0)})} lm(\mathcal{C}^{(0)} \wedge \mathcal{F}(\tilde{\mathcal{D}}^{(0)}))$.

Suppose that the algorithm terminates after $n + 1$ iterations. Let δ_1, δ_2, $\cdots$, δ_p denote the sequence of extracted collections obtained during execution of the algorithm. Let δ^u_1, δ^u_2, $\cdots$, δ^u_k be the subsequence of this consisting of just those extracted collections obtained via the union transformation. Similarly, let δ^c_1, δ^c_2, $\cdots$, δ^c_m be corresponding subsequence with respect to the clash and compaction transformations.

We now show that $lm(\mathcal{C}^{(n)})$ is a model of $\mathcal{C}^{(0)} \wedge \mathcal{F}(\tilde{\mathcal{D}}^{(0)})$. That $lm(\mathcal{C}^{(n)})$ is a model of $\mathcal{C}^{(0)}$ follows from repeated application of corollary 2. We now show

that $lm(\mathcal{C}^{(n)})$ is a model of $\mathcal{F}(\tilde{\mathcal{D}}^{(0)})$. Let $\mathcal{I}_n$ and $\mathcal{I}_{n+1}$ abbreviate $lm(\mathcal{C}^{(n)})$ and $lm(\mathcal{C}^{(n+1)})$ respectively, and let $\mathcal{V}_0$ abbreviate $var(\mathcal{C}^{(0)}, \tilde{\mathcal{D}}^{(0)})$. We proceed in three parts. First we show that

$$\mathcal{I}_n =_{\mathcal{V}_0} \mathcal{I}_{n+1}. \tag{6.37}$$

Obtain from $\mathcal{C}^{(n)}$ a collection $\hat{\mathcal{C}}^{(n)}$ by truncating each equation $X = sexp$ in $\mathcal{C}^{(n)}$ into $X = crhs(\mathcal{C}^{(n)})|_X$. Partition $\hat{\mathcal{C}}^{(n)}$ to obtain $\overline{\mathcal{C}}^{(n)}$ and $\overline{\overline{\mathcal{C}}}^{(n)}$ where the first partition $\overline{\mathcal{C}}^{(n)}$ contains all the equations whose l.h.s. is a variable from $\mathcal{V}^\star$. Similarly obtain the sets $\hat{\mathcal{C}}^{(n+1)}$, $\overline{\mathcal{C}}^{(n+1)}$ and $\overline{\overline{\mathcal{C}}}^{(n+1)}$. Now, by the construction of the least model given in proposition 2 $lm(\hat{\mathcal{C}}^{(n)}) = lm(\mathcal{C}^{(n)})$ (the deleted expressions are empty in the least model). Similarly, we have $lm(\hat{\mathcal{C}}^{(n+1)}) = lm(\mathcal{C}^{(n+1)})$. Now by proposition 4, all of the variables in $\overline{\mathcal{C}}^{(n)}$ are in $\mathcal{V}^\star$. Thus the least model of $\hat{\mathcal{C}}^{(n)}$ restricted to $\mathcal{V}^\star$ is defined by $\overline{\mathcal{C}}^{(n)}$ alone. That is, $lm(\hat{\mathcal{C}}^{(n)}) =_{\mathcal{V}^\star} lm(\overline{\mathcal{C}}^{(n)})$. Similarly, we have $lm(\hat{\mathcal{C}}^{(n+1)}) =_{\mathcal{V}^\star} lm(\overline{\mathcal{C}}^{(n+1)})$. Finally, by the termination condition of the main algorithm, ($crhs(\mathcal{C}^{(n+1)})|_X = crhs(\mathcal{C}^{(n)})|_X$ for all $X \in \mathcal{V}^\star$), which implies that $\overline{\mathcal{C}}^{(n)} = \overline{\mathcal{C}}^{(n+1)}$. Thus $\mathcal{I}_{n+1} =_{\mathcal{V}^\star} \mathcal{I}_n$, and since $\mathcal{V}_0 \subseteq \mathcal{V}^\star$, this implies (6.37).

Second, we use (6.37) to show that

$$\mathcal{I}_n =_{\mathcal{V}_0} [\![\delta_1^u]\!] \cdots [\![\delta_k^u]\!][\![\delta_1^c \cup \cdots \cup \delta_m^c]\!](\mathcal{I}_n) \tag{6.38}$$

Consider the following chain of inclusions.

$$\begin{array}{ll} & \mathcal{I}_n \\ \subseteq_{\mathcal{V}_0} & [\![\delta_1^u]\!] \cdots [\![\delta_k^u]\!][\![\delta_1^c \cup \cdots \cup \delta_m^c]\!](\mathcal{I}_n) \\ \subseteq_{\mathcal{V}_0} & [\![\delta_1^u]\!] \cdots [\![\delta_k^u]\!][\![\delta_1^c \cup \cdots \cup \delta_m^c]\!](\mathcal{I}_{n+1}) \\ \subseteq_{\mathcal{V}_0} & \mathcal{I}_{n+1} \\ \subseteq_{\mathcal{V}_0} & \mathcal{I}_n \end{array} \tag{6.39}$$

To prove the first inclusion, let $\delta_1, \cdots, \delta_{p'}$, $p' \leq p$, be the equations extracted in all *but the last* iteration of the main loop. By repeated application of corollary 2, $\mathcal{I}_n$ is a model of $\delta_1 \cup \cdots \cup \delta_{p'}$. Also, for $p' < l \leq p$, none of the variables in $lhs(\delta_i)$ appear in $\mathcal{C}_n$. Hence $\mathcal{I}_n$ maps all such variables to $\emptyset$. By proposition 13(c) (in which $\delta_1 \cup \cdots \cup \delta_{p'}$ is substituted for S), we obtain the first inclusion in (6.39).

To prove the second inclusion of (6.39), note that from corollary 2, $\mathcal{C}^{(n+1)} \models \mathcal{C}^{(n)}$. Hence $\mathcal{I}_n \subseteq \mathcal{I}_{n+1}$. The inclusion then follows from easily from proposition 12 (monotonicity of $[\![\ ]\!]$).

To prove the third inclusion of (6.39), note that $\mathcal{I}_{n+1}$ is a model of $\delta_1, \cdots, \delta_p$,

and hence the required inclusion follows from repeated application of proposition 11.

The final inclusion of (6.39) follows immediately from (6.37). Hence, we have proved each of the inclusions in (6.39). But this means that each inclusion in (6.39) is in fact an equality, and this therefore proves (6.38).

The final part of the proof that $\mathcal{I}_n$ is a model of $\mathcal{F}(\tilde{\mathcal{D}}^{(0)})$ uses (6.38) in the following chain of reasoning:

$$\begin{aligned}
[\![\mathcal{F}(\tilde{\mathcal{D}}^{(0)})]\!](\mathcal{I}_n) &=_{\mathcal{V}_0} [\![\delta_1^u]\!] \cdots [\![\delta_k^u]\!][\![\delta_1^c \cup \cdots \cup \delta_m^c \cup \mathcal{F}(\tilde{\mathcal{D}}^{(n+1)})]\!](\mathcal{I}_n) && (6.40)\\
&=_{\mathcal{V}_0} [\![\delta_1^u]\!] \cdots [\![\delta_k^u]\!][\![\delta_1^c \cup \cdots \cup \delta_m^c]\!](\mathcal{I}_n) && (6.41)\\
&=_{\mathcal{V}_0} \mathcal{I}_n. && (6.42)
\end{aligned}$$

The equality (6.40) follows directly from lemma 5. We now prove the equality (6.41). By lemma 6, we have, for all equations $X = exp$ in $\mathcal{F}(\tilde{\mathcal{D}}^{(n+1)})$, that $\mathcal{I}_n(exp) = \emptyset$. By the main loop invariant (proposition 3), X does not appear in $lhs(\mathcal{C}^{(n)})$ and so $\mathcal{I}_n(X) = \emptyset$. Thus $\mathcal{F}(\tilde{\mathcal{D}}^{(n+1)})$ can be removed from the r.h.s. of (6.40), and (6.41) follows. The final equality, (6.42) has already been proved in (6.38). Since $lhs(\mathcal{F}(\tilde{\mathcal{D}}^{(0)})) \subseteq \mathcal{V}_0$, the above equalities imply $[\![\mathcal{F}(\tilde{\mathcal{D}}^{(0)})]\!](\mathcal{I}_n) = \mathcal{I}_n$. By proposition 11, $\mathcal{I}_n$ is a model of $\mathcal{F}(\tilde{\mathcal{D}}^{(0)})$. Hence $\mathcal{I}_n$ is a model of $\mathcal{C}^{(0)} \wedge \mathcal{F}(\tilde{\mathcal{D}}^{(0)})$.

It remains to show that $\mathcal{I}_n$, is not just a model, but the least model of $\mathcal{C}^{(0)} \wedge \mathcal{F}(\tilde{\mathcal{D}}^{(0)})$. Now, by lemma 4,

$$\mathcal{C}^{(0)} \wedge \mathcal{F}(\tilde{\mathcal{D}}^{(0)}) \underset{\mathcal{V}_0}{\Longleftrightarrow} \mathcal{C}^{(n)} \wedge \mathcal{F}(\tilde{\mathcal{D}}_n).$$

which means that for each model of $\mathcal{C}^{(0)} \wedge \mathcal{F}(\tilde{\mathcal{D}}^{(0)})$, there is a model of $\mathcal{C}^{(n)} \wedge \mathcal{F}(\tilde{\mathcal{D}}_n)$ such that the two models agree on $\mathcal{V}_0$. Hence corresponding to the least model $\mathcal{I}_0$ of $\mathcal{C}^{(0)} \wedge \mathcal{F}(\tilde{\mathcal{D}}^{(0)})$, is a model $\mathcal{I}_0'$ of $\mathcal{C}^{(n)} \wedge \mathcal{F}(\tilde{\mathcal{D}}_n)$ such that $\mathcal{I}_0' =_{\mathcal{V}_0} \mathcal{I}_0$. Since $\mathcal{I}_n$ is the least model of $\mathcal{C}^{(n)} \wedge \mathcal{F}(\tilde{\mathcal{D}}_n)$, $\mathcal{I}_n \subseteq \mathcal{I}_0'$ which in turn implies $\mathcal{I}_n \subseteq_{\mathcal{V}_0} \mathcal{I}_0$. Now, it is already proved above that $\mathcal{I}_n$ is a model of $\mathcal{C}^{(0)} \wedge \mathcal{F}(\tilde{\mathcal{D}}^{(0)})$, and so $\mathcal{I}_0 \subseteq_{\mathcal{V}_0} \mathcal{I}_n$. Thus $\mathcal{I}_n =_{\mathcal{V}_0} \mathcal{I}_0$. □

6.4 The Final Algorithm

Given a program P, let $\mathcal{C}_P \wedge \mathcal{F}(\tilde{\mathcal{D}}_P)$ denote the associated set formula. The main algorithm cannot, in general, be directly applied to the expressions $\mathcal{C}_P$ and $\tilde{\mathcal{D}}_P$

input an arbitrary logic program P;
1. obtain the set formula $\mathcal{C}_P \wedge \mathcal{F}(\tilde{\mathcal{D}}_P)$ associated with P;
2. apply the preprocessing algorithm to $\mathcal{C}_P$ and $\tilde{\mathcal{D}}_P$ to obtain $\mathcal{C}^{(0)}$ and $\tilde{\mathcal{D}}^{(0)}$;
3. apply the main algorithm to $\mathcal{C}^{(0)}$ and $\tilde{\mathcal{D}}^{(0)}$ to obtain $\mathcal{C}$;

output $\mathcal{C}$;

Figure 4: **Final Algorithm**

because the main algorithm precondition may not be satisfied. More precisely, this precondition may be violated only because the r.h.s. of a pair in $\tilde{\mathcal{D}}_P$ may be deeply nested. Hence the following preprocessing algorithm which transforms $\mathcal{C}_P$ and $\tilde{\mathcal{D}}_P$ into equivalent expressions $\mathcal{C}^{(0)}$ and $\tilde{\mathcal{D}}^{(0)}$ which satisfy the precondition.

Replace each pair (ψ, ϕ) in $\tilde{\mathcal{D}}_P$ by the pair (ψ, Z) where Z is a new variable, and add to $\mathcal{C}_P$ the equation $Z = \phi$. Next, exhaustively apply the following transformation: if $\mathcal{C}_P$ contains a subexpression $f(t_1, \cdots, t_n)$ where some t_i, $1 \leq i \leq n$, is not a variable, then replace that occurrence of t_i by a new variable X and add the equation $X = t_i$. Next, rewrite the r.h.s. of each equation in $\mathcal{C}_P$ into DNF. Finally, apply $SIMPLIFY$ to $\mathcal{C}_P$. Let the expressions resulting from all these steps be denoted $\mathcal{C}^{(0)}$ and $\tilde{\mathcal{D}}^{(0)}$. It is easy to prove

Proposition 14
(a) $var(\mathcal{C}_P, \tilde{\mathcal{D}}_P) \subseteq var(\mathcal{C}^{(0)}, \tilde{\mathcal{D}}^{(0)})$;
(b) $lm\left(\mathcal{C}^{(0)} \wedge \mathcal{F}(\tilde{\mathcal{D}}^{(0)})\right) =_{var(\mathcal{C}_P, \tilde{\mathcal{D}}_P)} lm\left(\mathcal{C}_P \wedge \mathcal{F}(\tilde{\mathcal{D}}_P)\right)$, *and*
(c) $\mathcal{C}^{(0)}$ *and* $\tilde{\mathcal{D}}^{(0)}$ *satisfy the main algorithm precondition.* □

The algorithm is presented in its entirety in figure 4. We now conclude this section with the main result of this paper.

Theorem:

Let P be an arbitrary logic program input to the final algorithm. Then
(i) the final algorithm terminates, returning a formula $\mathcal{C}$;

(ii) $\mathcal{C}$ is a solved form collection of set equations, and

(iii) $\bigcup_{p \in \Pi_P} (lm(\mathcal{C}))(\Upsilon_p) = \mathcal{T}_P \uparrow \omega$.

Proof: First note that by proposition 14(c) the main algorithm is applied appropriately. Now, the proof of termination follows essentially from main lemma 2. That $\mathcal{C}$ is a solved form collection of set equations follows from two facts: (a) $SIMPLIFY$ is only applied to a normal form collection (and this is guaranteed by the main loop invariant), and (b) lemma 2, which states the output of $SIMPLIFY$ is a solved form collection of set equations. Thus only (iii) remains to be proved. Now,

$$\begin{array}{lll} lm(\mathcal{C}) & =_{var(\mathcal{C}_P,\tilde{\mathcal{D}}_P)} & lm\left(\mathcal{C}^{(0)} \wedge \mathcal{F}(\tilde{\mathcal{D}}^{(0)})\right), \quad \text{by lemma 3 and prop}^n \text{ 14(a);} \\ & =_{var(\mathcal{C}_P,\tilde{\mathcal{D}}_P)} & lm\left(\mathcal{C}_P \wedge \mathcal{F}(\tilde{\mathcal{D}}_P)\right), \quad \text{by prop}^n \text{ 14(b);} \end{array}$$

and by main lemma 1, we have

$$\bigcup_{p \in \Pi_P} lm(\mathcal{C}_P \wedge \mathcal{F}(\tilde{\mathcal{D}}_P))(\Upsilon_p) = \mathcal{T}_P{\uparrow}\omega.$$

Since $\Upsilon_p \in var(\mathcal{C}_P, \tilde{\mathcal{D}}_P)$ for all $p \in \Pi_P$, the two interpretations $lm(\mathcal{C})$ and $lm(\mathcal{C}_P \wedge \mathcal{F}(\tilde{\mathcal{D}}_P))$ agree on all such Υ_p. Thus (iii) follows. □

7 Conclusion

In program analysis, a key notion used to approximate the meaning of a program is that of ignoring inter-variable dependencies. We formalized this notion for logic programs using an approximation operator on term substitutions. We argued that this formalization produced more accurate approximations than similar formalizations in the literature. We then proved that the approximate meaning of a logic program can be equivalently formalized as the least model of a set formula associated with the program. The main result proved that such least models are not only recursive sets, but they are recognizable by tree automata. This result is essentially an algorithm consisting of two main parts.

The first part of our algorithm, called SIMPLIFY, was a decision procedure for a class of set equations, that is, set equations in normal form. Earlier related work is that of Reynolds [12] and [9], the latter being an independent treatise of work similar to the former. In these works, the set expressions were allowed to contain projection functions, but they were not allowed to contain the intersection function (these works were not concerned with problem of shared variables inherent in logic programs). Thus their transformations mainly served to eliminate occurrences of projection functions while ours mainly served to eliminate occurrences of the

intersection function[10].

The second and main part of our algorithm is also based upon a group of transformations. When used directly, these transformations do not always lead to a terminating computation. The key aspect of the main algorithm, therefore, is in its controlled use of the transformations. This is realized by the function REDUCE each of whose invocations performed transformations local to a fixed collection of set equations. The main loop in the main algorithm then ensured that REDUCE is invoked sufficiently often.

We finally note that two important practical issues have yet to be addressed: the first is efficiency; second, our algorithm does not, in general, return a *minimal* representation of $\mathcal{T}_P\uparrow\omega$.

References

[1] M. Bruynooghe, "A Practical Framework for the Abstract Interpretation of Logic Programs", revised version of report CW-62, Dept. Comp. Sci., K.U. Leuven, 33 pages, October 1987.

[2] M. Bruynooghe and G. Janssens, "An Instance of Abstract Interpretation integrating Type and Mode Inference", *Proceedings* 5^{th} *Int. Conf. on Logic Programming*, K.A. Bowen and R.A. Kowalski (eds), MIT Press, pp 669 - 683, August 1988.

[3] S.K. Debray, "Global Optimization of Logic Programs", Ph.D. Thesis, Department of Computer Science, SUNY at Stony Brook, 233 pages, December 1986.

[4] F. Gècseg and M. Steinby, "Tree Automata", Akadèmiai Kiadò, Budapest, 1984.

[5] N.C. Heintze and J. Jaffar, "A Decision Procedure for a Class of Herbrand Set Constraints", draft manuscript (extended abstract to appear in *Proceedings* 5^{th} *IEEE Symposium on Logic in Computer Science* June 1990).

[6] N.C. Heintze and J. Jaffar, "Semantic Types for Logic Programs", draft manuscript (submitted for publication), March 1990.

[7] J.W. Lloyd, "Foundations of Logic Programming", Second Edition, Springer-Verlag Series on Symbolic Computation, 1987.

[8] N.D. Jones and S.S. Muchnick, "Complexity of Flow Analysis, Inductive Assertion Synthesis, and a Language due to Dijkstra", *Proceedings* 20^{th} *Conference on*

[10] In [5], we present a decision procedure which deals with *both* projection and intersection functions.

Foundations of Computer Science, pp 185 - 190, also in, *Program Flow Analysis: Theory and Applications*, N.D. Jones and S.S. Muchnick (eds), Prentice-Hall, 1981.

[9] N.D. Jones and S.S. Muchnick, "Flow Analysis and Optimization of LISP-like Structures", *Proceedings* 6^{th} *ACM Symposium on Principles of Programming Languages*, pp 244 - 256, 1979.

[10] P. Mishra, "Toward a Theory of Types in PROLOG", *Proceedings* 1^{st} *IEEE Symposium on Logic Programming*, Atlantic City, pp 289 - 298, 1984.

[11] P. Mishra and U.S. Reddy, "Declaration-free Type Checking", *Proceedings* 12^{th} *ACM Symposium on Principles of Programming Languages*, New Orleans, pp 7 - 21, 1985.

[12] J.C. Reynolds, "Automatic Computation of Data Set Definitions", *Information Processing 68*, North-Holland, pp 456 - 461, 1969.

[13] J. Xu and D.S. Warren, "A Type Inference System for PROLOG", *Proceedings* 5^{th} *Intl. Conf. and Symp. on Logic Programming*, MIT Press, pp 604 - 619. July 1988.

[14] E. Yardeni and E.Y. Shapiro, "A Type System for Logic Programs", in *Concurrent PROLOG: Collected Papers*, Vol. 2, MIT Press, pp 211 - 244, 1987.