A Decision Procedure for a Class of Set Constraints

Nevin Heintze[1] and Joxan Jaffar[2]

February 14, 1991
CMU-CS-91-110

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

A preliminary version of this paper appears in the *Proceedings*, 5^{th} IEEE Symposium on LICS.

[1]The first author is supported by an IBM Graduate Fellowship.
[2]IBM Thomas J. Watson Research Center, PO Box 218, Yorktown Heights, NY 10598

Keywords: set constraints, program analysis, decision procedures

Abstract

A set constraint is of the form $exp_1 \supseteq exp_2$ where exp_1 and exp_2 are set expressions constructed using variables, function symbols, projection symbols, and the set union, intersection and complement symbols. While the satisfiability problem for such constraints is open, restricted classes have been useful in program analysis. The main result herein is a decision procedure for *definite* set constraints which are of the restricted form $a \supseteq exp$ where a contains only constants, variables and function symbols, and exp is a positive set expression (that is, it does not contain the complement symbol). A conjunction of such constraints, whenever satisfiable, has a least model and the algorithm will output an explicit representation of this model.

1 Introduction

We consider a formalism for elementary set algebra which is useful for describing properties of programs whose underlying domain of computation is a Herbrand universe. The domain of discourse for this formalism is the powerset of this universe. Its functions include set union, intersection and complement, in addition to a family Σ of functions intended to be the set-based counterparts of the functions used in the program. More specifically, all functions f in Σ satisfy a distributive property such as $f(X \cup Y) = f(X) \cup f(Y)$. Finally, for each n-ary symbol f in Σ, we have a conjunction $f^{-1}_{(1)}, \cdots, f^{-1}_{(n)}$ representing the projections of f. A *set constraint* is of the form $sexp_1 \supseteq sexp_2$ where $sexp_1$ and $sexp_2$ are set expressions constructed using the above function symbols and variables. The satisfiability problem for a conjunction of set constraints is open; the rest of this section discusses important decidable sub-classes.

Set based reasoning can be expressed in the predicate calculus by regarding a unary predicate as the set of values on which it is true. There are decidability results which deal with cases where the symbols are unary; see [1] for example. (These results were in fact shown for the second-order calculus.) Rabin [8], using the second-order theory of k successors and a standard interpretation, showed that the sets definable are recursive, and furthermore, he showed that there are algorithms to decide questions such as emptiness and containment. These works, however, have limited application to reasoning about programs since they are restricted to unary symbols.

Decision procedures have been obtained for set constraints involving function symbols of arbitrary arity, but at the cost of restricting the use of quantifiers, logical connectives, complement and intersection. Such procedures have been useful in an area of program analysis concerned with approximating the set of values that a variable may take at runtime. These approximations can be described using set constraints by first associating a set variable with each program variable. Set constraints can then be used to impose relationships between the set variables so that any model of these constraints defines a superset of the actual run-time values. In summary, set constraints provide a methodology of program approximation which ignores dependencies between variables.

Reynolds [9] and Jones and Muchnick [5], for example, considered functional programs and used a class of set constraints whose satisfiability is guaranteed. The focus of these works was to provide an explicit representation of the model defined by the constraints. Mishra [7], in his framework for type inference in logic programs, used set constraints to model a superset of the success set of a program.

Though a full decision procedure was not presented, algorithms for subcases were investigated. Heintze and Jaffar [4] used set constraints to obtain a recursive and accurate approximation of the meaning of a logic program.

In this paper, a decision procedure is presented for the class of *definite* set constraints which have the form $a \supseteq sexp$ where a contains only constants, variables and function symbols, and $sexp$ is a positive set expression. Restriction to definite constraints is analogous to the restriction to definite clauses from general clauses in the sense that a definite set constraint has a least model whenever it has a model. This class of constraints is expressive enough to be useful for modeling program properties via constraints (for example, see the references cited above). An important aspect of the procedure presented is that if the input constraints are satisfiable, then the procedure constructs an explicit representation of the least model of the constraints.

The following section contains some initial definitions, and some basic properties of set expressions. Section 3 describes a grammar formalism based on set expressions which is a generalization of term grammars. In essence, the main algorithm of this paper is a transformation system based on set expression grammars. Section 4 deals with a subclass of definite set constraints which alway have least models. This section contains the core algorithm. Section 5 then applies this algorithm to obtain the main result. Finally, appendix I gives some example of executions of one of the main algorithms presented, and appendix II gives some examples of how definite set constraints can be used to analyze program properties.

2 Preliminaries

Let $\mathcal{V}$ denote a denumerable set of variables, and let Σ denote a denumerable set of function and constant symbols; we assume that Σ contains at least one constant. Let $\tau(\Sigma)$ and $\tau(\Sigma \cup \mathcal{V})$ and denote the corresponding Herbrand universe (HU)[1] and Herbrand terms. Let $var(t)$, where t is any expression, denote the set of variables appearing in t. If $var(t)$ is the empty set, then t is said to be *ground*. Let $A, A_1, \cdots, A_n \subseteq \tau(\Sigma)$, $n \geq 1$, and let f be an n-ary symbol in Σ. Then $f(A_1, \cdots, A_n) \stackrel{\text{def}}{=} \{f(a_1, \cdots, a_n) : a_i \in A_i\}$, and $f^{-1}_{(i)}(A) \stackrel{\text{def}}{=} \{a_i : f(a_1, \cdots, a_n) \in A\}$.

A *set expression* is defined to be either a variable or constant, the special

[1] We assume throughout that HU contains at least two elements.

constant $\perp$ or $\top$, or of one of the forms $f(sexp_1, \cdots, sexp_n)$, $f_{(i)}^{-1}(sexp)$, $sexp \cup sexp'$, $sexp \cap sexp'$ and $\overline{sexp}$ where $1 \leq i \leq n$, $f \in \Sigma$ is n-ary, and $sexp$, $sexp'$, $sexp_1$, $\cdots$, $sexp_n$ are set expressions. A set expression is *atomic* if it does not contain any union, intersection, projection or complement symbols. An occurrence of an expression in a set expression is *nested* if it occurs within the scope of a function or projection symbol. A *set constraint* is of the form $sexp_1 \supseteq sexp_2$; if $sexp_1$ is atomic and $sexp_2$ does not contain the complement symbol, then we say that the set constraint is *definite*.

An *interpretation* is a map from $\mathcal{V} \longrightarrow 2^{HU}$. An interpretation $\mathcal{I}$, induces, in an obvious way, a mapping from set expressions $sexp$ into 2^{HU} as follows. $\mathcal{I}(\perp) = \emptyset$; $\mathcal{I}(\top) = HU$; $\mathcal{I}(c) = \{c\}$ where $c \in \Sigma$ is a constant; $\mathcal{I}(f_{(i)}^{-1}(sexp')) = f_{(i)}^{-1}(\mathcal{I}(sexp'))$ and $\mathcal{I}(f(sexp_1, \cdots, sexp_n)) = f(\mathcal{I}(sexp_1), \cdots, \mathcal{I}(sexp_n))$ where $f \in \Sigma$ is n-ary; and finally, $\mathcal{I}(sexp_1 \cup sexp_2) = \mathcal{I}(sexp_1) \cup \mathcal{I}(sexp_2)$ and $\mathcal{I}(sexp_1 \cap sexp_2) = \mathcal{I}(sexp_1) \cap \mathcal{I}(sexp_2)$ and $\mathcal{I}(\overline{sexp'}) = HU - \mathcal{I}(sexp')$. This mapping, in turn, induces a mapping from conjunctions of set constraints $\mathcal{C}$ into *true* or *false* in the obvious way. If $\mathcal{I}$ maps $\mathcal{C}$ into *true*, we say that $\mathcal{I}$ is a *model* of $\mathcal{C}$. We write $\mathcal{I} \supseteq \mathcal{I}'$ if $\mathcal{I}(X) \supseteq \mathcal{I}'(X)$ for all $X \in \mathcal{V}$. Where $\mathcal{C}$ and $\mathcal{C}'$ are conjunctions of set constraints and S is a set of variables, we write $\mathcal{C} \overset{}{\underset{S}{\Longleftrightarrow}} \mathcal{C}'$ if $\mathcal{C}$ and $\mathcal{C}'$ are equivalent in the sense that for any model $\mathcal{I}$ of $\mathcal{C}$, there is a model $\mathcal{I}'$ of $\mathcal{C}'$ such that $\mathcal{I}'(X) = \mathcal{I}(X)$ for all $X \in S$, and symmetrically, for any model $\mathcal{I}'$ of $\mathcal{C}'$, there is a model $\mathcal{I}$ of $\mathcal{C}$ such that $\mathcal{I}(X) = \mathcal{I}'(X)$ for all $X \in S$.

In the remainder of the paper, we consider only set expressions which do not contain the complement symbol, and adopt the convention that, unless otherwise specified, a set expression $sexp$ does not contain this symbol. Such expressions satisfy some basic monotonicity properties, which can be proved using simple structural induction arguments.

Proposition 1 (Monotonicity) *If $\mathcal{I}_1 \subseteq \mathcal{I}_2$ then $\mathcal{I}_1(sexp) \subseteq \mathcal{I}_2(sexp)$.*

Proposition 2 (Substitition) *Let $sexp'$ be the result of replacing an occurrence of a sub-expression t in $sexp$ by t'. Then $\mathcal{I}(t') \subseteq \mathcal{I}(t)$ implies $\mathcal{I}(sexp') \subseteq \mathcal{I}(sexp)$.*

The following are some basic properties about interpretations on ground set expressions $sexp$ and $sexp'$: (a) $\mathcal{I}(sexp)$ denotes a unique set regardless of the choice of $\mathcal{I}$ – denote this by $[\![sexp]\!]$; (b) if $sexp$ does not contain any union symbols, then there is a unique ground atomic set expression a such that $[\![sexp]\!] = [\![a]\!]$; (c) if $sexp$ is atomic and different from the symbols $\perp$ and $\top$, then $[\![sexp]\!] = \{sexp\}$.

Consider next the problem of explicitly representing a given interpretation. What is required is a representation which can be used to verify whether or not an arbitrary set constraint is *true* in the interpretation. Toward this aim, define that a *term grammar* ([3], p.67) is a grammar $\mathcal{G}$ whose nonterminals are variables in $\mathcal{V}$, and each rule is of the form $X \Longrightarrow t$ where X is a variable and t is an atomic set expression. Derivability $=_{\mathcal{G}}\Rightarrow$ is defined in an obvious way: $sexp =_{\mathcal{G}}\Rightarrow sexp'$ if $sexp'$ is the result of replacing *one* occurrence of a variable X in $sexp$ by t, and $X \Longrightarrow t$ is a rule in $\mathcal{G}$. A term grammar $\mathcal{G}$ then defines an interpretation $\mathcal{I}_{\mathcal{G}}$ since it associates to each variable a set of ground terms as follows:

$$\mathcal{I}_{\mathcal{G}}(X) = \bigcup_{X =_{\mathcal{G}}\Rightarrow\star\, t} [\![t]\!]$$

where t ranges over ground terms and $=_{\mathcal{G}}\Rightarrow\star$ is the reflexive and transitive closure of $=_{\mathcal{G}}\Rightarrow$ [2].

We note that a term grammar is essentially a parenthesis grammars [6]. The class of languages definable by term grammars is exactly the class accepted by tree automata[3]. Moreover a term grammar can be viewed as a description of a non-deterministic finite tree automata and vice-versa. Hence there are algorithms for combining term grammars over the boolean and projection operations, and for testing emptyness and containment (see [10], [2]; also see [3] for a survey of relevant results). This means that, given a representation of an interpretation $\mathcal{I}$ in the form of a term grammar, standard algorithms from the literature can be used to check whether a given set constraint, $sexp_1 \supseteq sexp_2$, is *true* in $\mathcal{I}$.

The main result of this paper can now be stated as follows. Given a conjunction of definite set constraints, the algorithm presented in this paper:
(a) determines if the conjunction is satisfiable, and
(b) if so, outputs a term grammar which represents its least model.
Now, since any term grammar for a language L can be used to build a conjunction of definite set constraints whose least model is L, it follows that the sets definable as the least model of definite set constraints are exactly those definable by regular term grammars.

[2] In what follows, we may abbreviate $=_{\mathcal{G}}\Rightarrow\star$ by $\stackrel{\star}{\Longrightarrow}$ whenever there is only one grammar $\mathcal{G}$ in context.
[3] In terms of later definitions (e.g. [3]), these automata are the frontier-to-root tree automata.

3 Set Expression Grammars

We now define a class of grammars, generalizing term grammars, intended to provide a representation of an interpretation which is more explicit than the conjunction of set constraints at hand, but not necessarily as explicit as a term grammar. That is, we cannot, in general, directly determine whether or not a given set constraint is *true* in the interpretation defined by a set expression grammar. (This is in contrast to a term grammar representation of an interpretation.) A key result of this paper, presented in the next section, is that a set expression grammar can be transformed into an equivalent term grammar.

A *set expression grammar* $\mathcal{G}$ is defined to be like a term grammar except that each rule $X \Longrightarrow sexp$ in $\mathcal{G}$ is such that $sexp$ may contain intersection and projection symbols[4]. As with term grammars, $\mathcal{G}$ defines an interpretation, call it $\mathcal{I}_{\mathcal{G}}$,

$$\mathcal{I}_{\mathcal{G}}(X) \stackrel{\text{def}}{=} \bigcup_{X \stackrel{\star}{\Longrightarrow} sexp} [\![sexp]\!]$$

where $X \in VAR$ and $sexp$ is a ground set expression. Consider for example the set expression grammar $\mathcal{G}$ consisting of the two rules $X \Longrightarrow f(X) \cap f(a)$ and $X \Longrightarrow a$. Here the only ground set expressions derivable from X are $f(a) \cap f(a)$ and a, and so $\mathcal{I}_{\mathcal{G}}$ maps X to the set $\{f(a), a\}$ and all other variables to $\emptyset$. We conclude with three properties of a set expression grammar $\mathcal{G}$.

Proposition 3 *Let ψ be a function symbol, a projection symbol or an intersection symbol. Let the arity of ψ be n and let $t_1, \cdots, t_n$, $t'_1, \cdots, t'_n$ be set expressions. Then*

$$\psi(t_1, \cdots, t_n) \stackrel{\star}{\Longrightarrow} \psi(t'_1, \cdots, t'_n) \quad \textit{iff} \quad t_1 \stackrel{\star}{\Longrightarrow} t'_1,\ \cdots,\ t_n \stackrel{\star}{\Longrightarrow} t'_n. \qquad \Box$$

Proof: One direction is clear: the derivations $t_1 \stackrel{\star}{\Longrightarrow} t'_1, \cdots, t_n \stackrel{\star}{\Longrightarrow} t'_n$ can be immediately combined to yield a derivation $\psi(t_1, \cdots, t_n) \stackrel{\star}{\Longrightarrow} \psi(t'_1, \cdots, t'_n)$. Conversely, suppose that $\psi(t_1, \cdots, t_n) \stackrel{\star}{\Longrightarrow} \psi(t'_1, \cdots, t'_n)$. Then there must be a derivation of the form

$$\psi(t_1, \cdots, t_n) \Longrightarrow \psi(t_1^{(1)}, \cdots, t_n^{(1)}) \Longrightarrow \psi(t_1^{(2)}, \cdots, t_n^{(2)}) \Longrightarrow \ldots \Longrightarrow \psi(t_1^{(k)}, \cdots, t_n^{(k)})$$

where $\psi(t_1^{(k)}, \cdots, t_n^{(k)})$ is $\psi(t'_1, \cdots, t'_n)$ and $k \geq 0$. Then, for each $1 \leq i \leq n$, this defines a sequence $t_i, t_i^{(1)}, t_i^{(2)}, \ldots, t_i^{(k)}$ such that if s is immediately after some t in this sequence then either s is equal to t or $t \Longrightarrow s$. It then follows that we can construct a derivation $t_i \Longrightarrow \ldots \Longrightarrow t'_i$, and so $t_i \stackrel{\star}{\Longrightarrow} t'_i$, as required. $\Box$

[4] The union and complement symbols are still omitted.

Note that the next property may not hold if $sexp$ contains the union symbol; for example, if $\mathcal{G}$ consists of the single rule $X \Longrightarrow a$ then $\mathcal{I}_{\mathcal{G}}(X \cup Z)$ is $\{a\}$, but there are no ground expressions derivable from $X \cup Z$ and so $\bigcup_{X \cup Z \overset{\star}{\Longrightarrow} sexp'} [\![sexp']\!]$ is the empty set.

Proposition 4 *Let sexp be a set expression which does not contain the union symbol*[5]*. Then, where sexp′ ranges over ground set expressions,*

$$\mathcal{I}_{\mathcal{G}}(sexp) = \bigcup_{sexp \overset{\star}{\Longrightarrow} sexp'} [\![sexp']\!] \quad \Box$$

Proof: The proof is by structural induction on $sexp$. The base cases are as follows: if $sexp$ is a variable, then the proposition follows immediately from the definition of $\mathcal{I}_{\mathcal{G}}$; if $sexp$ is a constant, then $sexp \overset{\star}{\Longrightarrow} sexp'$ implies that $sexp' = sexp$, and so the result follows from the fact that $\mathcal{I}_{\mathcal{G}}(sexp) = [\![sexp]\!]$.

To prove the induction case, suppose that $sexp$ is the set expression $\psi(t_1, \ldots, t_n)$ where the proposition holds for $t_1, \cdots, t_n$, and let ψ be a function, projection or intersection symbol of arity n. Then:

$$\begin{aligned}
\mathcal{I}_{\mathcal{G}}(\psi(t_1, \cdots, t_n)) &= \psi(\mathcal{I}_{\mathcal{G}}(t_1), \cdots, \mathcal{I}_{\mathcal{G}}(t_n)) \\
&= \psi(\bigcup_{t_1 \overset{\star}{\Longrightarrow} t'_1} [\![t'_1]\!], \cdots, \bigcup_{t_n \overset{\star}{\Longrightarrow} t'_n} [\![t'_n]\!]) \\
&= \bigcup_{\psi(t_1, \cdots, t_n) \overset{\star}{\Longrightarrow} \psi(t'_1, \cdots, t'_n)} [\![\psi(t'_1, \cdots, t'_n)]\!]
\end{aligned}$$

where $t'_1, \ldots, t'_n$ range over ground set expressions. The first equality is just the definition of interpretation. The second equality follows from the induction assumption. To prove the third and last equality, we show that any element of the left hand side (l.h.s.) must be an element of the right hand side (r.h.s.) and vice-versa. First, let g be an element of the l.h.s. Since ψ is either a function, projection or intersection symbol, it is easy to verify that there must exist $t'_1, \cdots, t'_n$ such that $t_i \overset{\star}{\Longrightarrow} t'_i$, $1 \le i \le n$, and $g \in \psi([\![t'_1]\!], \ldots, [\![t'_n]\!])$. From proposition 3, $\psi(t_1, \ldots, t_n) \overset{\star}{\Longrightarrow} \psi(t'_1, \ldots, t'_n)$. That g is an element of the r.h.s. then follows from the equality $\psi([\![t'_1]\!], \ldots, [\![t'_n]\!]) = [\![\psi(t'_1, \ldots, t'_n)]\!]$. Now let g be an element of the r.h.s. Then there must exist $\psi(t'_1, \cdots, t'_n)$ such that $\psi(t_1, \cdots, t_n) \overset{\star}{\Longrightarrow} \psi(t'_1, \cdots, t'_n)$ and $g \in [\![\psi(t'_1, \ldots, t'_n)]\!]$. Again, using the equality $\psi([\![t'_1]\!], \ldots, [\![t'_n]\!]) = [\![\psi(t'_1, \ldots, t'_n)]\!]$

[5] By our convention, $sexp$ also does not contain the complement symbol.

and proposition 3, proves that $t_i \stackrel{\star}{\Longrightarrow} t'_i$, $1 \leq i \leq n$ and that $g \in \psi([\![t'_1]\!], \ldots, [\![t'_n]\!])$. It is easy to verify that this implies that g is an element of the l.h.s. of the final equality. ∎

Proposition 5 *If* $sexp_1 \stackrel{\star}{\Longrightarrow} sexp_2$ *then* $\mathcal{I}_{\mathcal{G}}(sexp_1) \supseteq \mathcal{I}_{\mathcal{G}}(sexp_2)$. ∎

Proof: We first prove the proposition in the case that $sexp_1$ is a variable, say X. In this case, neither $sexp_1$ nor $sexp_2$ contain the union symbol, and so proposition 4 may be applied to obtain:

$$\begin{aligned} \mathcal{I}_{\mathcal{G}}(X) &= \bigcup_{X \stackrel{\star}{\Longrightarrow} sexp'} [\![sexp']\!] \\ \mathcal{I}_{\mathcal{G}}(sexp_2) &= \bigcup_{sexp_2 \stackrel{\star}{\Longrightarrow} sexp'} [\![sexp']\!] \end{aligned}$$

Now, since $sexp_2 \stackrel{\star}{\Longrightarrow} sexp'$ implies $X \stackrel{\star}{\Longrightarrow} sexp'$, it follows that $\mathcal{I}_{\mathcal{G}}(X) \supseteq \mathcal{I}_{\mathcal{G}}(sexp_2)$.

To complete the proof, let $V_1, \ldots, V_n$ denote the occurrences of variables in $sexp_1$. Then, the derivation $sexp_1 \stackrel{\star}{\Longrightarrow} sexp_2$ may be decomposed into applications of rules corresponding to each V_i. These define, for each $1 \leq i \leq n$ a derivation $V_i \stackrel{\star}{\Longrightarrow} t_i$ such that the result of replacing each occurrence V_i in $sexp_1$ by t_i results in the expression $sexp_2$. Now, applying the part of the current proposition which has already been completed (that is the case where $sexp_1$ is a variable), proves that $\mathcal{I}_{\mathcal{G}}(V_i) \supseteq \mathcal{I}_{\mathcal{G}}(t_i)$. Hence the substitution proposition may be applied to $sexp_1$ and $sexp_2$ to prove that $\mathcal{I}_{\mathcal{G}}(sexp_1) \supseteq \mathcal{I}_{\mathcal{G}}(sexp_2)$, and so the entire proof is complete. ∎

4 Reduced Form Set Constraints

A conjunction $\mathcal{C}$ of set constraints is in *reduced form* if each constraint is of the form $X \supseteq sexp$ where X is a variable and $sexp$ is a set expression. Reduced form constraints always have at least one model since the interpretation which maps each variable into HU is trivially a model of any constraint of the form $X \supseteq sexp$. In fact they always have a least model (we delay the proof of this until proposition 13, where we prove the more general property that a collection of definite constraints has a least model whenever it has a model). The problem at hand is: find an explicit representation of the least model of a reduced form conjunction[6] $\mathcal{C}$, hereafter denoted $lm_{\mathcal{C}}$, or simply $lm_{\mathcal{C}}$.

In case $\mathcal{C}$ does not contain any intersection or projection symbols, then $\mathcal{C}$ itself is an explicit representation of $lm_{\mathcal{C}}$. In case $\mathcal{C}$ does not contain any intersection symbols (but may contain projection symbols), early work by Reynolds [9], and later by Jones and Muchnick [5], gave algorithms for the explicit representation of $lm_{\mathcal{C}}$. The representations used in these papers are equivalent to term grammars. If, on the other hand, $\mathcal{C}$ does not contain any projection symbols (but may contain intersection), then a simplification algorithm contained in [4] solves the problem for $lm_{\mathcal{C}}$. Once again, the explicit representation used there is the same as a term grammar.

However, these results cannot be combined to deal with the case where $\mathcal{C}$ may contain both kinds of symbols because of the interaction between intersection and projection. The main contribution of this section is an algorithm to deal with this combination.

The section consists of two parts. The first part involves constructing a set expression grammar which defines $lm_{\mathcal{C}}$. The second part (and the bulk of the section) then deals with the problem of converting this set expression grammar into an equivalent term grammar.

We begin by transforming $\mathcal{C}$ into *grammar form*; this form is defined as a conjunction, each of whose constraints is of the form $X \supseteq sexp$, where $sexp$ does not contain the union symbol. This is achieved by the algorithm in figure 1. The conjunction $\mathcal{C}$ obtained by this algorithm has essentially the same model as the original conjunction. Specifically:

Proposition 6 *Let $\mathcal{C}$ be an reduced form conjunction of set constraints input to the grammar form algorithm. Then the algorithm terminates and outputs a grammar form conjunction $\mathcal{C}'$ such that $lm_{\mathcal{C}'}(U) = lm_{\mathcal{C}}(U)$ for all $U \in var(\mathcal{C})$.* $\square$

Proof: We show that each iteration of the **while** statement preserves the least model of $\mathcal{C}$. Consider a single iteration and let $X \supseteq sexp$ be the constraint replaced during this iteration, let $sexp'$ be the result of replacing the occurrence of $s \cup t$ in $sexp$ by Z, and let $\mathcal{C}_u$ be the constraints in $\mathcal{C}$ which are unchanged by this iteration. That is:

[6] We remark that the least model problem for the class of reduced form conjunctions is equivalent to the least model problem for the class of solved form conjunctions. A *solved form* conjunction of constraints has the form $X_1 = sexp_1 \wedge \cdots \wedge X_n = sexp_n$, where the X_i are distinct variables and the $sexp_i$ are set expressions.

input $\mathcal{C}$;
while $\mathcal{C}$ contains a union symbol **do**
  choose an ocurrence of an expression
    of the form $s \cup t$ in $\mathcal{C}$;
  replace this occurrence by a new variable, Z;
  add the constraints $Z \supseteq s$ and $Z \supseteq t$;
endwhile
output $\mathcal{C}$;

Figure 1: **Grammar Form Algorithm**

$\mathcal{C}$ before iteration is: $\mathcal{C}_u \wedge X \supseteq sexp$
$\mathcal{C}$ after iteration is: $\mathcal{C}_u \wedge X \supseteq sexp' \wedge Z \supseteq s \wedge Z \supseteq t$

It is sufficient to prove that

$$\mathcal{C}_u \wedge X \supseteq sexp \underset{\mathcal{V}}{\Longleftrightarrow} \mathcal{C}_u \wedge X \supseteq sexp' \wedge Z \supseteq s \wedge Z \supseteq t \tag{4.1}$$

where $\mathcal{V} = var(\mathcal{C}_u, X, sexp)$. Let $\mathcal{I}$ be a model of the l.h.s. of (4.1). Define $\mathcal{I}'$ to be the same as $\mathcal{I}$ except that on the new variable Z,

$$\mathcal{I}'(Z) \stackrel{\text{def}}{=} \mathcal{I}(s \cup t).$$

Since $\mathcal{I}'(s \cup t) = \mathcal{I}(s \cup t)$, it is immediate that $\mathcal{I}'(s \cup t) = \mathcal{I}'(Z)$, and so by the substitution proposition, $\mathcal{I}'(sexp) = \mathcal{I}'(sexp')$. Now $\mathcal{I}'$ is also a model of $X \supseteq sexp$ since Z does not appear in this constraint. Hence $\mathcal{I}'(X) \supseteq \mathcal{I}'(sexp) = \mathcal{I}'(sexp')$, which proves that $\mathcal{I}'$ is a model of the r.h.s. of (4.1).

To prove the converse, let $\mathcal{I}$ be a model of r.h.s. of (4.1). This implies that $\mathcal{I}(Z) \supseteq \mathcal{I}(s \cup t)$, and this implies, by the substitution proposition, that $\mathcal{I}(sexp') \supseteq \mathcal{I}(sexp)$. Combining this with the fact that $\mathcal{I}$ is a model of $X \supseteq sexp'$ proves that $\mathcal{I}(X) \supseteq \mathcal{I}(sexp') \supseteq \mathcal{I}(sexp)$, and so $\mathcal{I}$ is also a model of the l.h.s. of (4.1). $\square$

The essential benefit of a conjunction $\mathcal{C}$ being in grammar form is that we can construct a grammar which represents the least model of $\mathcal{C}$ by replacing $\supseteq$ by $\Longrightarrow$. Specifically, define that a grammar $\mathcal{G}$ and a conjunction $\mathcal{C}$ *correspond* if

$$X \Longrightarrow sexp \text{ appears in } \mathcal{C} \quad \text{iff} \quad X \supseteq sexp \text{ appears in } \mathcal{G}$$

Proposition 7 *Let $\mathcal{G}$ be a set expression grammar and let $\mathcal{G}$ be a grammar form conjunction of set constraints. If $\mathcal{C}$ and $\mathcal{G}$ correspond then $\mathcal{I}_{\mathcal{G}} = lm_{\mathcal{C}}$.* $\square$

Proof: We first prove that $\mathcal{I}_{\mathcal{G}}$ models $\mathcal{C}$. Let $X \supseteq sexp$ be an arbitrary constraint in $\mathcal{C}$. Then $X \Longrightarrow sexp$ is a rule in $\mathcal{G}$ and so $X \overset{\star}{\Longrightarrow} sexp$. This implies, by proposition 5, that $\mathcal{I}_{\mathcal{G}}(X) \supseteq \mathcal{I}_{\mathcal{G}}(sexp)$. Hence $\mathcal{I}_{\mathcal{G}}$ is a model of each constraint in $\mathcal{C}$.

We now prove that $\mathcal{I}_{\mathcal{G}}$ is the least model of $\mathcal{C}$. Let $\mathcal{I}$ be any model of $\mathcal{C}$. We show that for any ground set expression t,

$$X \overset{\star}{\Longrightarrow} t \quad \text{implies} \quad \mathcal{I}(X) \supseteq \mathcal{I}(t) \tag{4.2}$$

by using induction on the length of the derivation $X \overset{\star}{\Longrightarrow} t$. If the derivation is of length 1, then $X \Longrightarrow t$ is a rule in $\mathcal{G}$. Hence $X \supseteq t$ is a constraint in $\mathcal{C}$, and so $\mathcal{I}(X) \supseteq \mathcal{I}(t)$ because $\mathcal{I}$ is a model of $\mathcal{C}$. This proves the base case.

Suppose now that the derivation $X \overset{\star}{\Longrightarrow} t$ is of the form $X \Longrightarrow sexp_1 \Longrightarrow \ldots \Longrightarrow sexp_{k-1} \Longrightarrow t$ and that the property (4.2) holds for all derivations of length less than k. Let $X_1, \ldots, X_n$ denote the occurrences of variables in $sexp_1$, and let $X_1 \overset{\star}{\Longrightarrow} t_1, \ldots, X_n \overset{\star}{\Longrightarrow} t_n$ be the respective subderivations for these variables in the derivation $sexp_1 \overset{\star}{\Longrightarrow} sexp_k$. This means that if each variable occurrence X_i in $sexp_1$ is replaced by the expression t_i, $1 \leq i \leq n$, then the result is t. Now each expression t_i is ground since $sexp$ is a ground expression. Also the derivations $X_i \overset{\star}{\Longrightarrow} t_i$ are all of length less than k. Hence, by the induction hypothesis, $\mathcal{I}(X_i) \supseteq \mathcal{I}(t_i)$. This implies, by repeated application of the substitution proposition, that $\mathcal{I}(sexp_1) \supseteq \mathcal{I}(t)$. Combining this with the fact that $X \Longrightarrow sexp_1$ is a rule in $\mathcal{G}$, and hence $\mathcal{I}$ is a model of $X \supseteq sexp_1$, proves that $\mathcal{I}(X) \supseteq \mathcal{I}(sexp_1) \supseteq \mathcal{I}(t)$. This completes the induction argument, and proves property (4.2).

To complete the proof that $\mathcal{I}_{\mathcal{G}}$ is the least model of $\mathcal{C}$, we now use property (4.2) to show that $\mathcal{I} \supseteq \mathcal{I}_{\mathcal{G}}$. Since $\mathcal{I}_{\mathcal{G}}(X)$ is defined as the union of the sets $[\![t]\!]$ where t is a ground expression derivable from X, it is sufficient to show the following: for all ground t

$$X \overset{\star}{\Longrightarrow} t \quad \text{implies} \quad \mathcal{I}(X) \supseteq [\![t]\!]$$

This follows immediately from property (4.2), since for any ground expression t, $\mathcal{I}(t) = [\![t]\!]$. $\square$

From the correctness of the algorithm above (proposition 2) and proposition 7, we have

Lemma 1 *Let $\mathcal{C}$ be in reduced form. Then a set expression grammar $\mathcal{G}$ can be constructed such that $\mathcal{I}_{\mathcal{G}}(X) = lm_{\mathcal{C}}(X)$ for all $X \in var(\mathcal{C})$.* □

We now deal with the problem of converting a set expression grammar into an equivalent term grammar. We first apply the preprocessing algorithm described in figure 2 to remove nested occurrences of intersection and projection symbols. The required property of this algorithm is:

Proposition 8 *Let $\mathcal{G}$ be a set expression grammar input to the grammar preprocessing algorithm. Then the algorithm terminates and outputs a set expression grammar $\mathcal{G}'$, whose nested expressions are all atomic, which is equivalent to $\mathcal{G}$ in the sense that*

for all $X \in var(\mathcal{G})$, $\mathcal{I}_{\mathcal{G}}(X) = \mathcal{I}_{\mathcal{G}'}(X)$. □

Proof: The grammar preprocessing algorithm clearly terminates because each iteration of the **while** loop reduces the total number of nested intersection and projection symbols. Furthermore, on termination, all nested occurrences of intersection or projection symbols have been removed.

The correctness of the algorithm is proved by showing that each replacement step of the algorithm (which corresponds to one step of the **while** loop) preserves the meaning of the grammar in the following sense: if $\mathcal{G}_1$ and $\mathcal{G}_2$ respectively denote the grammar just before and just after an iteration, and if X is a variable in $var(\mathcal{G}_1)$, then, for all ground set expressions t,

$$X =_{\mathcal{G}_1}\Rightarrow\star\, t \text{ iff } X =_{\mathcal{G}_2}\Rightarrow\star\, t. \tag{4.3}$$

To prove this statement, let $Y \Longrightarrow sexp$ be the rule which contains the occurrence of s replaced during this iteration of the **while** loop, and let $sexp'$ be the result of replacing this occurrence of s in $sexp$ by Z. Then, the only difference between $\mathcal{G}_1$ and $\mathcal{G}_2$ is that the rule $Y \Longrightarrow sexp$ is replaced by the two rules $Y \Longrightarrow sexp'$ and $Z \Longrightarrow s$. The proof that the l.h.s. of (4.3) implies the r.h.s. is easy, since any application of the rule $Y \Longrightarrow sexp$ in a derivation $X =_{\mathcal{G}_1}\Rightarrow\star\, t$ can be replaced by successive applications of $Y \Longrightarrow sexp'$ and $Z \Longrightarrow s$ to obtain a derivation $X =_{\mathcal{G}_2}\Rightarrow\star\, t$.

To prove the converse, note that if $X =_{\mathcal{G}_2}\Rightarrow\star\, t$, then (i) any application of the rule $Z \Longrightarrow s$ replaces an occurrence of the variable Z and such an occurrence can

input $\mathcal{G}$;

while $\mathcal{G}$ contains a nested intersection or projection symbol **do**

  choose a nested ocurrence of an expression s

    of the form $t_1 \cap t_2$ or $f_{(i)}^{-1}(t)$ in $\mathcal{G}$;

  replace this occurrence by a new variable, Z;

  add the rule $Z \Longrightarrow s$;

endwhile

output $\mathcal{G}$;

Figure 2: **Grammar Preprocessing Algorithm**

only be introduced by an earlier application of the rule $Y \Longrightarrow sexp'$, and (ii) any application of the rule $Y \Longrightarrow sexp'$ introduces an occurrence of the variable Z, and since t is ground, this occurrence must be removed by a subsequent applications of the rule $Z \Longrightarrow s$. Hence applications of the rules $Y \Longrightarrow sexp'$ and $Z \Longrightarrow s$ must occur in pairs in such a way that an occurrence of the first introduces an occurrence of the variable Z which is removed by a subsequent application of the second. It follows that any derivation $X =_{\mathcal{G}_2}\Rightarrow\star\, t$ can be reorganized so that each application of $Y \Longrightarrow sexp'$ is followed immediately by the matching application of $Z \Longrightarrow s$, and so these pairs can be replaced by a single application of $Y \Longrightarrow sexp$. Repeated application of such a replacement step yields a derivation $X =_{\mathcal{G}_1}\Rightarrow\star\, t$. $\square$

We now describe the main algorithm of this section. This algorithm converts a set expression grammar, whose nested expressions are atomic, into a term grammar. The algorithm is essentially based on an exhaustive application of a series of transformations. Each transformation is simple; the main difficulty is termination. Let $\mathcal{G}$ be a set expression grammar in which all nested expressions are atomic. The first transformation serves to implement an obvious notion of substitution.

Transformation 1 (Substitute)
If $\mathcal{G}$ contains a rule $X \Longrightarrow sexp$ such that

- *$sexp$ contains an occurrence of a variable Y and this occurrence is not within the scope of a function symbol (but it may occur inside the scope of a projection or union symbol), and*
- *$Y \Longrightarrow f(s_1, \cdots, s_n)$ is a rule in $\mathcal{G}$ and $f \in \Sigma$ is n-ary, $n \geq 0$,*

then let $sexp'$ be the result of replacing the occurrence of Y in $sexp$ by $f(s_1, \cdots, s_n)$ and add the rule $X \Longrightarrow sexp'$ to $\mathcal{G}$. $\square$

The next transformation deals with the elimination of projection symbols. In condition (b) below, we assume that there exists a recursive function ORACLE($\mathcal{G}, s$) which returns *true* if and only if $s \stackrel{\star}{\Longrightarrow} t$ for some ground atomic set expression such that $[\![t]\!] \neq \emptyset$. We provide an implementation of ORACLE later.

Transformation 2 (Projection-Elimination)
If $\mathcal{G}$ contains a rule $X \Longrightarrow sexp$ such that

(a) sexp contains an expression of the form $f_{(i)}^{-1}(f(s_1, \ldots, s_n))$, where $f \in \Sigma$ is n-ary, $1 \leq i \leq n$, and

(b) ORACLE$(\mathcal{G}, f(s_1, \ldots, s_n))$,

then let $sexp'$ be the result of replacing the occurrence of $f_{(i)}^{-1}(f(s_1, \ldots, s_n))$ in sexp by s_i and add the rule $X \Longrightarrow sexp'$ to $\mathcal{G}$. □

The next transformation deals with eliminating intersection symbols. We note that given two term grammars, we can construct a third grammar whose language is the intersection of the first two [10] [2]. We cannot, however, use this result to eliminate occurrences of the intersection symbol and reduce a set expression grammar into a term grammar. This is because a grammar which has intersection symbols *within* its rules cannot, in general, be broken into intersection-free parts whose languages combine to equal the original language. Hence:

Transformation 3 ($\cap$-distribution)
If $\mathcal{G}$ contains a rule $X \Longrightarrow sexp$ such that sexp contains an occurrence of an expression of the form $f(s_1, \cdots, s_n) \cap f(t_1, \cdots, t_n)$ then let $sexp'$ denote the result of replacing this occurrence in sexp by $f(Z_1, \ldots, Z_n)$, where $Z_1, \ldots, Z_n$ are new variables, and add the following rules to $\mathcal{G}$:

- *$X \Longrightarrow sexp'$, and*
- *$Z_i \Longrightarrow s_i \cap t_i$, $1 \leq i \leq n$.* □

The final transformation deals with $\top$ symbol. It is essentially describes special cases of the above transformations for expressions such as $sexp \cap \top$ and $f_{(i)}^{-1}(\top)$.

Transformation 4 ($\top$-distribution)
If $\mathcal{G}$ contains a rule $X \Longrightarrow sexp$ such that sexp contains an expression of the form $s \cap \top$ or $f_{(i)}^{-1}(\top)$, then let $sexp'$ denote the result of replacing this occurrence in sexp by s or $\top$ respectively, and add the rule $X \Longrightarrow sexp'$ to $\mathcal{G}$. □

Let $\mathcal{G} \to \mathcal{G}'$ denote that the grammar $\mathcal{G}'$ is obtained from the grammar $\mathcal{G}$ by a single application of one of transformations 1-4. We now prove that transformations 1-4 preserve the meaning of a grammar in the following sense:

Proposition 9 (Correctness of a Single Transformation)
If $\mathcal{G} \to \mathcal{G}'$ then

$$\mathcal{I}_{\mathcal{G}'}(X) = \mathcal{I}_{\mathcal{G}}(X) \textit{ for all } X \in var(\mathcal{G}). \quad \square$$

Proof: We first prove the following substitution property for set expression grammars.

> **[Grammar Substitution Property]:** Let $X \Longrightarrow sexp$ be a rule in a set expression grammar $\mathcal{G}$, let t be a subexpression of $sexp$ and let $sexp'$ be the result of replacing an occurrence of t in $sexp$ by t'. Then $\mathcal{I}_{\mathcal{G}}(t') \subseteq \mathcal{I}_{\mathcal{G}}(t)$ implies $\mathcal{I}_{\mathcal{G}\cup\{X \Longrightarrow sexp'\}} = \mathcal{I}_{\mathcal{G}}$.

To prove this property, let $\mathcal{C}$ be the grammar form constraints corresponding to $\mathcal{G}$. By proposition 7, $lm_{\mathcal{C}} = \mathcal{I}_{\mathcal{G}}$. Since $X \supseteq sexp$ is a constraint in $\mathcal{C}$ and $\mathcal{I}_{\mathcal{G}}$ is a model of $\mathcal{C}$, $\mathcal{I}_{\mathcal{G}}(X) \supseteq \mathcal{I}_{\mathcal{G}}(sexp)$. Also, by the substitution property, $\mathcal{I}_{\mathcal{G}}(sexp) \supseteq \mathcal{I}_{\mathcal{G}}(sexp')$. Hence $\mathcal{I}_{\mathcal{G}}$ satisfies $X \supseteq sexp'$. Combining this with proposition 7 gives the following chain of equalities, and this proves the grammar substitution property.

$$\mathcal{I}_{\mathcal{G}\cup\{X \supseteq sexp'\}} = lm\,(\mathcal{C} \cup \{X \supseteq sexp'\}) = lm\,(\mathcal{C}) = \mathcal{I}_{\mathcal{G}}.$$

The proposition can now be proved by four applications of the grammar substitution property, one for each of the possible transformations. Suppose that transformation 1 (substitution) is applied. Since $Y \Longrightarrow f(s_1, \cdots, s_n)$ is a rule in $\mathcal{G}$, $\mathcal{I}_{\mathcal{G}}(Y) \supseteq \mathcal{I}_{\mathcal{G}}(f(s_1, \cdots, s_n))$ by proposition 7. Recalling that $sexp'$ is obtained from $sexp$ by replacing an occurrence of Y by $f(s_1, \cdots, s_n)$, we have, on applying the above grammar substitution property, that

$$\mathcal{I}_{\mathcal{G}\cup\{X \Longrightarrow sexp'\}} = \mathcal{I}_{\mathcal{G}},$$

and this proves the proposition for the case of the substitution transformation.

Suppose that the transformation applied is transformation 2 (projection elimination). Since ORACLE($\mathcal{G}, f(s_1, \ldots, s_n)$), it follows that $\mathcal{I}_{\mathcal{G}}(f(s_1, \cdots, s_n)) \neq \emptyset$, and so

$$\mathcal{I}_{\mathcal{G}}(f_{(i)}^{-1}(f(s_1, \cdots, s_n))) = \mathcal{I}_{\mathcal{G}}(s_i)$$

The grammar substitution property can now be applied to complete the proof for this case.

Suppose that the transformation applied is transformation 3 ($\cap$-distribution). Let $\mathcal{G}'$ denote the grammar $\mathcal{G} \cup \{Z_i \Longrightarrow s_i \cap t_i : 1 \leq i \leq n\}$. It is clear that

$$\mathcal{I}_{\mathcal{G}'}(X) = \mathcal{I}_{\mathcal{G}}(X),\ X \in var(\mathcal{G})$$

since $\mathcal{G}'$ is obtained from $\mathcal{G}$ by adding rules for the variables $Z_1, \ldots, Z_n$, and these variables do not appear in $\mathcal{G}$. To complete the proof for this transformation, we now show that $\mathcal{I}_{\mathcal{G}'} = \mathcal{I}_{\mathcal{G}' \cup \{X \Longrightarrow sexp'\}}$. First, note that the only rule for each variable Z_i is $Z_i \Longrightarrow s_i \cap t_i$. Hence if $f(Z_1, \ldots, Z_n) =_{\mathcal{G}'}\Rightarrow\star\ sexp$ where $sexp$ is some ground expression, then by rearranging this derivation so that the rules replacing the variables Z_i are used first, it is clear that there is a derivation $f(s_1 \cap t_1, \ldots, s_n \cap t_n)$ $=_{\mathcal{G}'}\Rightarrow\star\ sexp$. It follows, using proposition 4, that

$$\mathcal{I}_{\mathcal{G}'}(f(Z_1, \ldots, Z_n)) \subseteq \mathcal{I}_{\mathcal{G}'}(f(s_1 \cap t_1, \ldots, s_n \cap t_n)).$$

Combining this with the easily verifiable distributivity property of $\cap$ over f, we have

$$\mathcal{I}_{\mathcal{G}'}(f(Z_1, \ldots, Z_n)) \subseteq \mathcal{I}_{\mathcal{G}'}(f(s_1 \cap t_1, \ldots, s_n \cap t_n)) = \mathcal{I}_{\mathcal{G}'}(f(s_1, \ldots, s_n) \cap f(t_1, \ldots, t_n)).$$

Again the grammar substitution property can be applied to complete the proof for this case.

Finally, suppose that the transformation applied is transformation 4. Now, for any interpretation $\mathcal{I}$, $\mathcal{I}(s \cap \top) = \mathcal{I}(s)$ and $\mathcal{I}(f_{(i)}^{-1}(\top)) = \mathcal{I}(\top)$. Hence the correctness for this transformation follows easily from the grammar substitution property. $\Box$

The main algorithm for this section can now be presented in figure 3. This algorithm is essentially an exhaustive application of transformations 1 - 4, with three modifications. The first involves the removal of duplicate expressions before adding a rule to $\mathcal{G}$. Specifically, any rule can be written in the form $X \Longrightarrow t_1 \cap \ldots \cap t_n$, where $t_1, \ldots, t_n$ do not contain the $\cap$ symbol. The removal of duplicate expressions can then be defined by the rule: delete all t_i such that there is a t_j, $j < i$, and $t_i = t_j$.

The second modification involves restricting the use of transformations which do not add new rules. Specifically, define that a transformation is *applicable* to $\mathcal{G}$ if the pre-conditions of the transformation are satisfied and the transformation adds some new rules[7] to $\mathcal{G}$. Also, define that $\mathcal{G}$ is *closed* under a transformation τ if the transformation τ is not applicable to $\mathcal{G}$. The restricted use of transformations can now be characterized as: a transformation is only used when it is applicable.

The third modification involves carefully controlling the new variables added by the $\cap$-distribution transformation so that only a finite number of these variables need to be introduced. The essential idea behind this mechanism is a special naming scheme: each introduced variable is of the form $V_{\{t_1,\cdots,t_n\}}$ where $t_1, \cdots, t_n$ are atomic expressions appearing in $\mathcal{G}$ at the start of the algorithm. The intention here is that such a variable is equivalent to the expression $t_1 \cap \cdots \cap t_n$. Thus a variable such as $V_{\{f(X),Y,f(Z)\}}$ not only represents $f(X) \cap Y \cap f(Z)$, but also expressions such as $V_{\{f(X),Y\}} \cap f(Z)$ and $V_{\{f(X),f(Z)\}} \cap V_{\{f(X),Y\}}$. To facilitate the use of this naming scheme, let $\mathcal{N}$ denote the mapping from an expression (including a variable of the form V_T) to the set of atomic expressions it represents[8]:

$$\mathcal{N}(t) = \begin{cases} T & \text{if } t \text{ is } V_T; \\ \mathcal{N}(t_1) \cup \mathcal{N}(t_2) & \text{if } t \text{ is } t_1 \cap t_2; \\ \{t\} & \text{otherwise.} \end{cases}$$

For example, $\mathcal{N}(V_{\{f(X),f(Z)\}} \cap V_{\{f(X),Y\}}) = \{f(X), Y, f(Z)\}$. Similarly $\mathcal{N}(V_{\{f(X),Y\}} \cap f(Z))$ and $\mathcal{N}(f(X) \cap Y \cap f(Z))$ are also equal to $\{f(X), Y, f(Z)\}$.

We now prove that the grammar simplification algorithm terminates and is correct in the sense that the grammar output by the algorithm is a term grammar which describes the least model of the input set expression grammar. In a number of the following arguments dealing with properties of rules added by transformations, we will, for notational convenience, ignore the fact that duplicate terms are deleted from the r.h.s. of a rule before it is added to $\mathcal{G}$. It is straightforward, but tedious, to expand the proofs to deal with this issue, since the absence or presence of duplicate expressions in rules is of no consequence. One exception to this is the argument for termination, where the proof explicitly depends on the deletion of duplicate terms.

We start by establishing an important invariant of the algorithm. One main purpose of this invariant is to bound the number of new variables that are introduced during the algorithm.

Proposition 10 (Subexpression-Invariant) *Let $\mathcal{G}_0$ be a set expression grammar in which all nested expressions are atomic and let $\mathcal{G}_0$ be input to the grammar*

[7] This test is made after duplicate expressions have been removed from the potentially new rules.

[8] $\mathcal{N}$ is well defined since $\mathcal{N}((t_1 \cap t_2) \cap t_3) = \mathcal{N}(t_1) \cup \mathcal{N}(t_2) \cup \mathcal{N}(t_3) = \mathcal{N}(t_1 \cap (t_2 \cap t_3))$.

input $\mathcal{G}$;

1. **repeat**

2. **exhaustively-apply** transformations 1, 2 and 4 to $\mathcal{G}$;

3. **if** transformation 3 is applicable to $\mathcal{G}$
(that is, $\mathcal{G}$ contains a rule $X \Longrightarrow sexp$ such that $sexp$
contains an occurrence of the expression $f(s_1, \cdots, s_n) \cap f(t_1, \cdots, t_n)$)
then

4. **add** $X \Longrightarrow sexp'$ to $\mathcal{G}$ where $sexp'$ is obtained by replacing
the occurrence of $f(s_1, \cdots, s_n) \cap f(t_1, \cdots, t_n)$ in $sexp$
by $f(V_{\mathcal{N}(t_1) \cup \mathcal{N}(s_1)}, \cdots, V_{\mathcal{N}(t_n) \cup \mathcal{N}(s_n)})$;

5. **for** each $i : 1 \leq i \leq n$ **do**

6. **if** $V_{\mathcal{N}(t_i) \cup \mathcal{N}(s_i)}$ is a new variable in $\mathcal{G}$ **then**

7. add $V_{\mathcal{N}(t_i) \cup \mathcal{N}(s_i)} \Longrightarrow t_i \cap s_i$ to $\mathcal{G}$

endif

endfor

endif

until neither transformation 1, 2, 3 nor 4 is applicable to $\mathcal{G}$;

output the rules in $\mathcal{G}$ which do not contain intersection or projection symbols;

Figure 3: **Grammar Simplification Algorithm**

simplification algorithm. Then, at all times during execution of the algorithm, all nested expressions sexp of $\mathcal{G}$ are atomic and are such that $\mathcal{N}(sexp)$ contains only atomic subexpressions of $\mathcal{G}_0$.

Proof: We say that $\mathcal{G}$ *satisfies the invariant* if all nested subexpressions $sexp$ of $\mathcal{G}$ are atomic and such that $\mathcal{N}(sexp)$ contains only subexpressions from $\mathcal{G}_0$. The proposition can then be restated as: $\mathcal{G}$ satisfies the invariant at all times during execution of the algorithm.

Clearly $\mathcal{G}$ satisfies the invariant at the start of line 1 of the algorithm. Suppose now that the invariant holds at some stage during execution of the algorithm. Since each step of the algorithm is essentially an application of one of the transformations substitute, projection elimination, $\cap$-distribution or $\top$-distribution, it suffices to show that the rules added by a single application of one of these transformations satisfies the invariant, given that $\mathcal{G}$ satisfies the invariant. Let $atomic(\mathcal{G}_0)$ denote the atomic subexpressions of $\mathcal{G}_0$, and consider the four cases.

Consider a single application of the substitution transformations. In this case the only rule added is $X \Longrightarrow sexp'$ where $sexp'$ is the result of replacing an un-nested occurrence of a variable by $f(t_1, \cdots, t_n)$. Hence the only nested expressions in this rule which did not already appear in $X \Longrightarrow sexp$ are those that appear in $f(t_1, \cdots, t_n)$. However $f(t_1, \cdots, t_n)$ is obtained from the r.h.s. of some other rule in $\mathcal{G}$. Hence any rule added by the substitution transformation also satisfies the invariant.

Consider a single application of the projection-elimination transformation. In this case the only rule added is $X \Longrightarrow sexp'$ where $sexp'$ is the result of replacing an un-nested occurrence of the form $f_{(i)}^{-1}(f(s_1, \cdots, s_n))$ in $sexp$ by s_i. Hence no new nested expressions are introduced.

Consider a single application of the $\cap$-distribution transformation. We refer to the modified version of this transformation given in lines 4-7 of the algorithm. This transformation adds a new rule at line 4 and may also add new rules at line 7. Now the only nested expressions in a rule added at line 4 are of the form $V_{\mathcal{N}(t_i)\cup\mathcal{N}(s_i)}$ where s_i and t_i are nested expressions in $\mathcal{G}$. Since $\mathcal{G}$ satisfies the invariant, s_i and t_i are both atomic expressions such that $\mathcal{N}(t_i)$ and $\mathcal{N}(s_i)$ are subsets of $atomic(\mathcal{G}_0)$. Hence $\mathcal{N}(V_{\mathcal{N}(t_i)\cup\mathcal{N}(s_i)}) = \mathcal{N}(t_i) \cup \mathcal{N}(s_i) \subseteq atomic(\mathcal{G}_0)$, and so any rule added at line 4 must satisfy the invariant. Consider now a rule added at line 7. The only nested subexpressions of such a rule appear as nested subexpressions of s_i and t_i, and since s_i and t_i appear in $\mathcal{G}$, this case is proved.

Consider a single application of the $\top$-distribution transformation. In this case the only rule added is $X \Longrightarrow sexp'$ where $sexp'$ is the result of replacing an unnested expression of the form $s \cap \top$ or $f_{(i)}^{-1}(\top)$ by s or $\top$ respectively. Hence no new nested expressions are added, and so any rule added by the $\top$-distribution transformation also satisfies the invariant. $\square$

We now justify the modified application of transformation 3 in the grammar simplification algorithm by proving the following invariant. In essence it establishes the correctness of the introduced variables V_S by showing that $V_{\mathcal{N}(s)\cup\mathcal{N}(t)}$ and $s \cap t$ are equivalent under $\mathcal{I}_{\mathcal{G}}$.

Proposition 11 *Let the grammar simplification algorithm be input with a set expression grammar in which all nested expressions are atomic. Let $\mathcal{G}$ be the grammar obtained after any completed iteration*[9] *of the* **repeat** *loop. If V_T is a variable in $\mathcal{G}$ and $sexp$ is a set expression containing only variables from $\mathcal{G}$ such that $T = \mathcal{N}(sexp)$ then $\mathcal{I}_{\mathcal{G}}(V_T) = \mathcal{I}_{\mathcal{G}}(sexp)$.*

Proof: Let s be a set expression such that $\mathcal{N}(s)$ is $\{t_1, \cdots, t_n\}$ where the t_j are distinct atomic expressions which appear in $\mathcal{G}$. We prove that

$$\mathcal{I}_{\mathcal{G}}(s) = \mathcal{I}_{\mathcal{G}}(t_1 \cap \cdots \cap t_n). \tag{4.4}$$

The proposition then follows by applying (4.4) twice, once with $s = V_T$ and the other with $s = sexp$, noting that since V_T is a variable in $\mathcal{G}$, T contains only atomic expressions which appear in $\mathcal{G}$.

We now prove (4.4). Let $\mathcal{G}_0, \mathcal{G}_1, \cdots, \mathcal{G}_l, \cdots$ denote the sequence of grammars produced by the algorithm such that $\mathcal{G}_l$ is the grammar produced after l iterations. Since $\mathcal{G}$ is $\mathcal{G}_l$ for some l, it suffices to show, for all iterations l, that if s is restricted to contain only variables from $\mathcal{G}_l$ then

$$\mathcal{I}_{\mathcal{G}_l}(s) = \mathcal{I}_{\mathcal{G}_l}(t_1 \cap \cdots \cap t_n). \tag{4.5}$$

We proceed by induction. If $l = 0$ then s does not contain any variables of the form V_T. Clearly it can be written as $s_1 \cap \cdots \cap s_m$ such that $\{s_1, \cdots, s_m\} = \{t_1, \cdots, t_n\}$. Suppose that d is an element of the l.h.s. of (4.5). Then $d \in \mathcal{I}_{\mathcal{G}_l}(s_i)$ for all $1 \leq i \leq m$. Since each s_i appears as some t_j and vice-versa, this implies that $d \in \mathcal{I}_{\mathcal{G}_l}(t_j)$ for all $1 \leq j \leq n$. Hence $d \in \mathcal{I}_{\mathcal{G}_l}(t_1 \cap \cdots \cap t_n)$, the r.h.s. of (4.5). The converse proof, that any d in the r.h.s. of (4.5) is also in the l.h.s., is identical. This proves the base case.

[9] An iteration here refers to a complete execution of the body of the **repeat** loop (lines 2 – 7) of the grammar simplification algorithm.

Now consider the l^{th} iteration, and suppose that (4.5) holds for all iterations before the l^{th}. Again s can be written as $s_1 \cap \cdots \cap s_m$ where the s_i are atomic. Now suppose that s contains a variable which does not appear in $\mathcal{G}_{l-1}$. Such a variable must be a variable of the form V_T, and this variable must appear as one of the s_i, $1 \le i \le m$. Moreover, this variable must have been introduced during the l^{th} iteration, and hence during this iteration a rule of the form $V_T \Longrightarrow u \cap v$ is added such that $T = \mathcal{N}(u) \cup \mathcal{N}(v)$. This is the only rule for V_T in $\mathcal{G}_l$ and so $\mathcal{I}_{\mathcal{G}_l}(V_T) = \mathcal{I}_{\mathcal{G}_l}(u \cap v)$. Now, on setting

$$s' = s_1 \cap s_{i-1} \cap (u \cap v) \cap s_{i+1} \cap \cdots \cap s_m$$

it is clear that $\mathcal{N}(s') = \mathcal{N}(s)$ and $\mathcal{I}_{\mathcal{G}_l}(s') = \mathcal{I}_{\mathcal{G}_l}(s)$. This step can be repeated to remove all of the variables which are introduced in the l^{th} iteration. The resulting expression, call it s'' has the property that $\mathcal{N}(s'') = \mathcal{N}(s)$ and $\mathcal{I}_{\mathcal{G}_l}(s'') = \mathcal{I}_{\mathcal{G}_l}(s)$. Finally, since all variables in s'' appear in $\mathcal{G}_{l-1}$, the induction hypothesis can be applied to obtained

$$\mathcal{I}_{\mathcal{G}_l}(s) = \mathcal{I}_{\mathcal{G}_l}(s'') = \mathcal{I}_{\mathcal{G}_l}(t_1 \cap \cdots \cap t_n), \quad \square$$

The next proposition establishes a property used in the following lemma.

Proposition 12 (Exhaustive Application of Substitution) *Let $\mathcal{G}$ be a set expression grammar which is closed under the substitution transformation, let X be a variable, and let sexp be a ground atomic set expression. If $X \stackrel{\star}{\Longrightarrow} sexp$ is the shortest derivation of a ground expression sexp from X, then the first rule used in this derivation is of the form $X \Longrightarrow sexp'$ such that $sexp'$ is a non-variable atomic expression.* $\square$

Proof: The proof is an easy induction on the length of the derivation $X \stackrel{\star}{\Longrightarrow} sexp$. If the length is 0 or 1, then the result is immediate. Now suppose that the result holds for derivations of length $n \ge 1$, and consider a derivation of length $n + 1$. This derivation is of the form:

$$X \Longrightarrow sexp' \Longrightarrow sexp'' \stackrel{\star}{\Longrightarrow} sexp$$

where the derivation $sexp' \stackrel{\star}{\Longrightarrow} sexp$ of length n and is the shortest derivation of $sexp$ from $sexp'$. Suppose that $sexp'$ is a variable. Then by the induction hypothesis $sexp''$ is a non-variable atomic set expression. Hence the pre-conditions of the substitution transformation are satisfied, and since $\mathcal{G}$ is closed under the substitution transformation, the rule $X \Longrightarrow sexp''$ must already appear in $\mathcal{G}$. However this gives a shorter derivation of $X \stackrel{\star}{\Longrightarrow} sexp$. Hence it must be the case that $sexp'$ is not a variable. Also, since $sexp$ is atomic, $sexp'$ must be atomic. $\square$

The following lemma is a core part of the correctness of the grammar simplification algorithm. Each part of the lemma makes an incremental step towards part (d), which essentially says that at the end of the **repeat** loop of the algorithm, all information about the least model is contained in "explicit" derivations of the form $X \stackrel{\star}{\Longrightarrow} a$ such that a does not contain any projection or intersection symbols. This means that after the **repeat** loop, the "non-explicit" part of $\mathcal{G}$ can be simply deleted, and this is the effect of the final **output** step of the grammar simplification algorithm. The main difficulty in the proof involves the variables of the form V_T. The introduction of this variable scheme[10] needed for termination reasons and its purpose was to provide a representation for expressions of the form $s_i \cap t_i$ introduced by the $\cap$-distribution transformation. However, given a derivation of the form

$$X \Longrightarrow f(s) \cap f(t) \stackrel{\star}{\Longrightarrow} f(s') \cap f(t')$$

we can argue that the $\cap$-distribution transformation adds a rule of the form $X \Longrightarrow f(V_{\mathcal{N}(s)\cup\mathcal{N}(t)})$, but we cannot argue that therefore there exists a derivation of the form $X \stackrel{\star}{\Longrightarrow} f(s' \cap t')$, because in general there will not be a derivation $V_{\mathcal{N}(s)\cup\mathcal{N}(t)} \stackrel{\star}{\Longrightarrow} s' \cap t'$. Fortunately we can construct an "equivalent" derivation. Establishing an appropriate notion of equivalence is part of the following lemma.

Lemma 2 (Exhaustive Transformation Application) *Let the algorithm be input with a set expression grammar in which all nested expressions are atomic. Then, if the* **repeat** *loop of the algorithm terminates, the following properties hold of the resulting grammar* $\mathcal{G}$*:*

(a) If $t_1, \cdots, t_n$ *is a sequence of atomic expressions appearing in* $\mathcal{G}$*,* $sexp$ *is an expression such that* $\mathcal{N}(sexp) = \{t_1, \cdots, t_n\}$*, and* $t_i \stackrel{\star}{\Longrightarrow} t'_i$*,* $1 \leq i \leq n$ *then there is a derivation* $sexp \stackrel{\star}{\Longrightarrow} v_1 \cap \cdots \cap v_m$ *such that* $\{v_1, \cdots, v_m\} = \{t'_1, \cdots, t'_n\}$[11]

(b) If $sexp \stackrel{\star}{\Longrightarrow} sexp'$ *where* $sexp'$ *is a ground expression not containing union or projection symbols and* $d \in [\![sexp']\!]$*, then there is a derivation* $sexp \stackrel{\star}{\Longrightarrow} t_1 \cap \cdots \cap t_n \stackrel{\star}{\Longrightarrow} sexp''$ *such that* $sexp''$ *is a ground expression not containing union or projection symbols,* $d \in [\![sexp'']\!]$ *and* $\{t_1, \cdots, t_n\} = \mathcal{N}(sexp)$*.*

(c) For all variables X*, if* $X \stackrel{\star}{\Longrightarrow} sexp$*,* $sexp$ *is a ground expression not containing projection symbols and* $d \in [\![sexp]\!]$*, then there exists a ground atomic set expression* a *such that* $X \stackrel{\star}{\Longrightarrow} a$ *and* $d \in [\![a]\!]$*.*

[10] We remark that the variable scheme V_T is somewhat analogous to the powerset construction used to transform an NFA to a DFA.

[11] The sequences $t_1, \cdots, t_n$ and $t'_1, \cdots, t'_n$ and $v_1, \cdots, v_m$ may have duplicate elements.

(d) *For all variables X, if $X \stackrel{\star}{\Longrightarrow}$ sexp for some ground sexp such that $d \in [\![sexp]\!]$, then there exists a ground atomic set expression a such that $d \in [\![a]\!]$ and $X \stackrel{\star}{\Longrightarrow} a$.*

Proof for part (a): Let V_T be a variable in $\mathcal{G}$ which is introduced by the algorithm during an application of the $\cap$-distribution transformation. When the variable is first introduced, a rule of the form $V_T \Longrightarrow s \cap t$ is added to $\mathcal{G}$ such that $T = \mathcal{N}(s) \cup \mathcal{N}(t)$ (see line 7 of the grammar simplification algorithm). Let $initial(V_T)$ denote this rule. Now, starting with the expression $sexp$, pick an occurrence of a variable of the form V_T, and replace this occurrence using the rule $initial(V_T)$. Let $sexp_1$ denote the result. Now repeat this process to obtain $sexp_2$, $sexp_3$, $\cdots$ Eventually this process must terminate since if a variable of the form $V_{T'}$ appears in the r.h.s. of a rule $initial(V_T)$, then the variable $V_{T'}$ must have been introduced before V_T. Let the number of replacement steps used be k. It is easy to verify that these replacement steps replace only un-nested variables, since no nested occurrences of variables of the form V_T can be introduced. Also, each rule used is of the form $V \Longrightarrow s \cap t$ such that $\mathcal{N}(V) = \mathcal{N}(s \cap t)$. It follows that $\mathcal{N}(sexp)$ $= \mathcal{N}(sexp_1) = \mathcal{N}(sexp_2) = \cdots = \mathcal{N}(sexp_k)$. Finally, write $sexp_k$ as $u_1 \cap \cdots \cap u_m$ where the u_i are atomic. Then since $sexp_k$ does not contain any variables of the form V_S, $\mathcal{N}(sexp_k) = \{u_1, \cdots, u_m\}$. Hence, this yields a derivation

$$sexp \stackrel{\star}{\Longrightarrow} u_1 \cap \cdots \cap u_m$$

such that $\{u_1, \cdots, u_m\} = \{t_1, \cdots, t_n\}$. Hence, $u_1 \cap \cdots \cap u_m$ can be written as $t_{i_1} \cap \cdots \cap t_{i_m}$, where each i_j is such that $1 \leq i_j \leq n$, and each t_i appears in $t_{i_1} \cap \cdots \cap t_{i_m}$. Hence $t_{i_1} \cap \cdots \cap t_{i_m} \stackrel{\star}{\Longrightarrow} t'_{i_1} \cap \cdots \cap t'_{i_m}$. Since $\{t'_{i_1}, \cdots, t'_{i_m}\} = \{t'_1, \cdots, t'_n\}$, part (a) is proved.

Proof for part (b): Since $sexp$ is assumed not to contain union or projection symbols, it can be written as $s_1 \cap \cdots \cap s_m$ where each s_i is an atomic expression. Then there exist s'_i such that $s_i \stackrel{\star}{\Longrightarrow} s'_i$ and $sexp'$ is $s'_1 \cap \cdots \cap s'_m$. The remainder of the proof involves showing constructing, for each $1 \leq i \leq m$, a derivation $s_i \stackrel{\star}{\Longrightarrow} t_{i,1} \cap \cdots \cap t_{i,n_i} \stackrel{\star}{\Longrightarrow} s''_i$ such that s''_i does not contain any projection or union symbols, $d \in [\![s''_i]\!]$ and $\{t_{i,1}, \cdots, t_{i,n_i}\} = \mathcal{N}(s_i)$. Part (b) then follows immediately by composing these derivations.

To construct these derivations, first note that if s_i is not a variable of the form V_T, then the derivation $s_i \stackrel{\star}{\Longrightarrow} s'_i$ will do, since $\mathcal{N}(s_i) = \{s_i\}$. It is therefore sufficient to prove the following for each variable of the form V_T appearing in $\mathcal{G}$:

$$\text{If } V_T \stackrel{\star}{\Longrightarrow} s \text{ such that } s \text{ does not contain the projection symbol and } d \in [\![s]\!] \text{ then there exists a derivation } V_T \stackrel{\star}{\Longrightarrow} t_1 \cap \cdots \cap t_n \stackrel{\star}{\Longrightarrow} s' \text{ such that } s' \text{ does not contain any projection symbols, } d \in [\![s']\!] \text{ and } \{t_1, \cdots, t_n\} = T. \tag{4.6}$$

To prove this property let $\mathcal{G}_0, \mathcal{G}_1, \cdots, \mathcal{G}_i, \cdots, \mathcal{G}_N$ denote the sequence of grammars obtained after each transformation application; that is, $\mathcal{G}_0$ is the input grammar, $\mathcal{G}_i$ is the grammar obtained after i transformation applications, and $\mathcal{G}_N$ is the resultant grammar $\mathcal{G}$ at the end of the **repeat** loop, where N is the total number of transformation applications. Now, each variable of the form V_T can be identified with the transformation application which introduced this variable. Write $\eta(V_T) = n$ to denote that the variable V_T is introduced by the n^{th} transformation application. Clearly the only rule for V_T in $\mathcal{G}_{\eta(V_T)}$ is of the form $V_T \Longrightarrow u_1 \cap u_2$ such that u_1 and u_2 are atomic expression and $T = \mathcal{N}(u_1) \cup \mathcal{N}(u_2)$. Hence there are no subexpressions of the form $f_{(j)}^{-1}(w)$ in this rule, and so transformations dealing with projection (specifically, the projection elimination transformation and the part of the $\top$-distribution transformation dealing with expressions of the form $f_{(j)}^{-1}(\top)$) cannot be applied to such a rule. Also, any transformation application to a rule of this form yields a rule of the form $V_T \Longrightarrow sexp$ where $sexp$ is an atomic expression, or $sexp$ is of the form $a \cap a'$, for some atomic expressions a and a'. By repeatedly applying these observations, it is easy to verify that, for $i \geq \eta(V_T)$, all transformations in $\mathcal{G}_i$ for V_T are of the form $V_T \Longrightarrow sexp$ where $sexp$ is either an atomic expression, or of the form $a \cap a'$ for some atomic expression a and a' which appear in $\mathcal{G}_{i-1}$. Therefore, if $V_T \Longrightarrow sexp$ is a rule in $\mathcal{G}_i$ for some i, then either

(i) $sexp$ is of the form $u_1 \cap u_2$ where $T = \mathcal{N}(u_1) \cup \mathcal{N}(u_2)$ and u_1 and u_2 are expressions which appear in $\mathcal{G}_{i-1}$, or

(ii) there exists a rule $V_T \Longrightarrow sexp'$ in $\mathcal{G}_j$ for some $j < i$ such that $V_T \Longrightarrow sexp$ is obtained from $V_T \Longrightarrow sexp'$ by an application of the $\cap$-distribution transformation, or

(iii) there exists a rule $V_T \Longrightarrow sexp'$ in $\mathcal{G}_j$ for some $j < i$ such that $V_T \Longrightarrow sexp$ is obtained from $V_T \Longrightarrow sexp'$ by an application of the $\top$-distribution transformation, or

(iv) there exist rules $V_T \Longrightarrow sexp'$ and $X \Longrightarrow t$ in $\mathcal{G}_j$ for some $j < i$ such that $V_T \Longrightarrow sexp$ is obtained from $V_T \Longrightarrow sexp'$ and $X \Longrightarrow t$ by an application of the substitution transformation.

We now use this case analysis to prove the property (4.6). The argument is essentially an induction argument on $V_T \stackrel{\star}{\Longrightarrow} s$.

Where $d \in HU$, V_T is a variable appearing in $\mathcal{G}$ and s is a ground expression, let $\nabla(d, V_T \stackrel{\star}{\Longrightarrow} s)$ be (α, β, γ) such that α is the number of symbols in d, β is $\eta(V_T)$ and γ is the smallest integer such that the first rule in the derivation $V_T \stackrel{\star}{\Longrightarrow} s$ appears in $\mathcal{G}_\gamma$. We proceed by induction on ∇. The base case, where $\alpha = 0$ or $\beta = 0$ or γ is 0, is vacuous since there are no terms with 0 symbols and no variables in $\mathcal{G}_0$ of the form V_T. Now consider $\nabla(d, V_T \stackrel{\star}{\Longrightarrow} s) = (\alpha, \beta, \gamma)$ such that (4.6) holds for $(\alpha', \beta', \gamma')$ if either $\alpha' < \alpha$ or $\alpha' = \alpha$ and $\beta' < \beta$, or $\alpha' = \alpha$ and $\beta' = \beta$ and $\gamma' < \gamma$. Write $V_T \stackrel{\star}{\Longrightarrow} s$ as

$$V_T \Longrightarrow sexp \stackrel{\star}{\Longrightarrow} s$$

and consider the cases as outlined above. In case (i), $sexp$ is of the form $u_1 \cap u_2$ where $T = \mathcal{N}(u_1) \cup \mathcal{N}(u_2)$, u_i appears in $\mathcal{G}_{\alpha-1}$, $i = 1, 2$, and there exists u'_i, $i = 1, 2$ such that $u_i \stackrel{\star}{\Longrightarrow} u'_i$, $d \in [\![u'_i]\!]$ and $u'_1 \cap u'_2$ is s. If u_i is of the form V_S then since it appears in $\mathcal{G}_{\alpha-1}$, the induction hypothesis can be applied to obtain $u_i \stackrel{\star}{\Longrightarrow} t_1 \cap \cdots \cap t_n \stackrel{\star}{\Longrightarrow} u''_i$ where u''_i does not contain any projection symbols, $d \in [\![u''_i]\!]$, and $\{t_1, \cdots, t_n\}$ is $\mathcal{N}(u_i)$. On the other hand, if u_i is not of the form V_S, then $\mathcal{N}(u_i) = \{u_i\}$. Hence in either case there exists a derivation $u_i \stackrel{\star}{\Longrightarrow} t_{i,1} \cap \cdots \cap t_{i,n_i} \stackrel{\star}{\Longrightarrow} u''_i$ such that u''_i does not contain any projection symbols, $d \in [\![u''_i]\!]$ and $\{t_{i,1}, \cdots, t_{i,n_i}\}$ is $\mathcal{N}(u_i)$. Combining these facts proves first that

$$V_T \Longrightarrow u_1 \cap u_2 \stackrel{\star}{\Longrightarrow} t_{1,1}, \cdots, t_{1,n_1} \cap t_{2,1}, \cdots, t_{2,n_2} \stackrel{\star}{\Longrightarrow} u''_1 \cap u''_2.$$

Second $u''_1 \cap u''_2$ does not contain any projections symbols. Third $d \in [\![u''_1 \cap u''_2]\!]$. Finally, $T = \mathcal{N}(u_1) \cup \mathcal{N}(s_2) = \{t_{1,1}, \cdots, t_{1,n_1}, t_{2,1}, \cdots, t_{2,n_2}\}$. This proves (4.6) in case (i).

In case (ii), $sexp$ is of the form $f(V_{\mathcal{N}(v_1)\cap\mathcal{N}(w_1)}, \cdots, V_{\mathcal{N}(v_k)\cap\mathcal{N}(w_k)})$ and $\mathcal{G}$ contains a rule of the form $V_T \stackrel{\star}{\Longrightarrow} f(v_1, \cdots, v_k) \cap f(w_1, \cdots, w_k)$. Also, there exist ground u_i which do not contain any projection symbols, such that $V_{\mathcal{N}(v_i)\cap\mathcal{N}(w_i)} \stackrel{\star}{\Longrightarrow} u_i$, $1 \le i \le k$, and s is $f(u_1, \cdots, u_k)$. Now, since $d \in [\![f(u_1, \cdots, u_k)]\!]$, d is of the form $f(d_1, \cdots, d_k)$ and $d_i \in [\![u_i]\!]$, $1 \le i \le k$. Hence the induction hypothesis can be applied, and so, for $1 \le i \le k$, there exists a derivation $V_{\mathcal{N}(v_i)\cap\mathcal{N}(w_i)} \stackrel{\star}{\Longrightarrow} t_{i,1} \cap \cdots \cap t_{i,n_i} \stackrel{\star}{\Longrightarrow} s'_i$ such that the s'_i are ground expressions not containing projection symbols, $d \in [\![s'_i]\!]$ and $\{t_{i,1}, \cdots, t_{i,n_i}\} = \mathcal{N}(v_i) \cap \mathcal{N}(w_i)$. Further, each s'_i can be written as $s_{i,1}, \cdots, s_{i,n_i}$ such that $t_{i,j} \stackrel{\star}{\Longrightarrow} s_{i,j}$, $1 \le j \le n_i$. Now, for $1 \le i \le k$, $\mathcal{N}(v_i)$ and $\mathcal{N}(w_i)$ are both subsets of $\{t_{i,1}, \cdots, t_{i,n_i}\}$. Hence by part (a), there exist derivations

$$v_i \stackrel{\star}{\Longrightarrow} v'_i \quad \text{such that } d_i \in [\![v'_i]\!],\ 1 \le i \le k,$$
$$w_i \stackrel{\star}{\Longrightarrow} w'_i \quad \text{such that } d_i \in [\![w'_i]\!],\ 1 \le i \le k,$$

such that the v_i' and w_i' do not contain any projection symbols. Hence there is a derivation

$$V_T \Longrightarrow f(v_1, \cdots, v_k) \cap f(w_1, \cdots, w_k) \stackrel{\star}{\Longrightarrow} f(v_1', \cdots, v_k') \cap f(w_1', \cdots, w_k')$$

where $f(v_1', \cdots, v_k') \cap f(w_1', \cdots, w_k')$ is a ground expression which does not contain any projection symbols such that $f(d_1, \cdots, d_k) \in [\![f(v_1', \cdots, v_k') \cap f(w_1', \cdots, w_k')]\!]$. Since $T = \{f(v_1, \cdots, v_k), f(w_1, \cdots, w_k)\}$, the proof for case (ii) is complete.

In case (iii), there is a rule of the form $V_T \Longrightarrow \top \cap sexp$ or $V_T \Longrightarrow sexp \cap \top$ in $\mathcal{G}_{\alpha-1}$. Now, since $V_T \Longrightarrow sexp \stackrel{\star}{\Longrightarrow} s$, either the derivation $V_T \Longrightarrow sexp \cap \top \stackrel{\star}{\Longrightarrow} s \cap \top$ or $V_T \Longrightarrow \top \cap sexp \stackrel{\star}{\Longrightarrow} \top \cap s$ can be constructed. In either case, d is an element of $[\![\top \cap sexp]\!]$ and $[\![sexp \cap \top]\!]$. Now the first rule used in these derivations appears in $\mathcal{G}_{\alpha-1}$, and hence the induction hypothesis can be applied to prove this case.

In case (iv), there are rules $V_T \Longrightarrow u$ and $X \Longrightarrow v$ in $\mathcal{G}_{\alpha-1}$ such that the result of replacing an occurrence of X in u by v is $sexp$. Now since $V_T \Longrightarrow sexp \stackrel{\star}{\Longrightarrow} s$, the derivation $V_T \Longrightarrow u \Longrightarrow sexp \stackrel{\star}{\Longrightarrow} s$ can be constructed, where the first rule used is $V_T \Longrightarrow u$ and the second is $X \Longrightarrow v$. Now these rule both appear in $\mathcal{G}_{\alpha-1}$, and hence the induction hypothesis can be again be applied to prove this case.

Proof for part (c): Where $d \in HU$, X is a variable and $sexp$ is a ground expression define $\nabla(d, X, sexp)$ to be (α, β) such that α is the number of symbols in d and β is the length of the derivation $X \stackrel{\star}{\Longrightarrow} sexp$. We proceed by induction on ∇. The base case, where $\alpha = 0$ or $\beta = 0$, is vacuous since there are no terms with 0 symbols and no derivations of $sexp$ from X of 0 length because $sexp$ is ground expression and X is a variable. Now consider $\nabla(d, X, sexp) = (\alpha, \beta)$ such that part (c) holds for (α', β') if either $\alpha' < \alpha$ or $\alpha' = \alpha$ and $\beta' < \beta$. Let $X \stackrel{\star}{\Longrightarrow} sexp$ be a minimal length derivation of $sexp$ from X, and write this derivation as

$$X \Longrightarrow s_1 \cap \cdots \cap s_n \stackrel{\star}{\Longrightarrow} sexp$$

where the s_i, $1 \leq i \leq n$ are atomic expressions. Then there exist t_i, $1 \leq i \leq n$, such that $s_i \stackrel{\star}{\Longrightarrow} t_i$ and $sexp$ is $t_1 \cap \cdots \cap t_n$. Since $d \in [\![sexp]\!]$, $d \in [\![t_i]\!]$, $1 \leq i \leq n$. Suppose that one of the s_i is a variable. Since $s_i \stackrel{\star}{\Longrightarrow} t_i$ is a shorter derivation than the derivation $X \stackrel{\star}{\Longrightarrow} sexp$ then the induction hypothesis may be applied (using the fixed d) to obtain an atomic a_i such that $s_i \stackrel{\star}{\Longrightarrow} a_i$ and $d \in [\![a_i]\!]$. Pick the shortest such derivation of a_i from s_i. By proposition 12, this derivation must be of the form $s_i \Longrightarrow s_i' \stackrel{\star}{\Longrightarrow} a_i$ where s_i' is a non-variable atomic expression (i.e. of the form $f(\cdots)$, $f \in \Sigma$). Hence a derivation of the following form can be constructed:

$$X \Longrightarrow s'_1 \cap \cdots \cap s'_m \overset{\star}{\Longrightarrow} sexp' \tag{4.7}$$

where $d \in [\![sexp']\!]$ and each s'_i, $1 \leq i \leq m$ is a non-variable atomic expression. Let d be $f(d_1, \cdots, d_k)$. This means that the top-most symbol of each s'_i must either be f or $\top$. Now, suppose that for some of the s'_i are $\top$. Then the $\top$-distribution transformation can be used repeatedly to yield a rule

$$X \Longrightarrow s'_{i_1} \cap \cdots \cap s'_{i_m}, \quad 1 \leq i_1 < i_2 < \cdots < i_m \leq n$$

Since $\mathcal{G}$ is closed under the $\top$-distribution transformation, this rule must already be present. Using this rule, a new derivation of the form (4.7) can be constructed where each s'_i, is a non-variable atomic expression, $1 \leq i \leq m$. Therefore, each of the s'_i must be of the form $f(\cdots)$. Hence $s'_1 \cap \cdots \cap s'_n$ can be written in the form $f(u_{1,1}, \cdots, u_{k,1}) \cap \cdots \cap f(u_{m,1}, \cdots, u_{m,k})$ for some $u_{i,j}$, $1 \leq i \leq k$, $1 \leq j \leq m$. In particular, the repeated application of the the $\cap$-distribution transformation yields the following rule:

$$X \Longrightarrow f(V_{T_1}, \cdots, V_{T_k})$$

where $T_i = \mathcal{N}(u_{i,1}) \cup \cdots \cup \mathcal{N}(u_{i,k})$, $1 \leq i \leq k$. Now, since $sexp'$ is derivable from $f(u_{1,1}, \cdots, u_{k,1}) \cap \cdots \cap f(u_{m,1}, \cdots, u_{m,k})$ it must be the case that there exist $v_{i,j}$ such that $1 \leq i \leq m$ and $1 \leq j \leq k$ such that

- $sexp'$ is of the form $f(v_{1,1}, \cdots, v_{k,1}) \cap \cdots \cap f(v_{m,1}, \cdots, v_{m,k})$
- $u_{i,j} \overset{\star}{\Longrightarrow} v_{i,j}$, $1 \leq i \leq m$, $1 \leq j \leq k$
- $d_i \in [\![v_{i,j}]\!]$, $1 \leq i \leq m$, $1 \leq j \leq k$

Now, fix i, $1 \leq i \leq k$. Clearly $u_{i,1} \cap \cdots \cap u_{i,m} \overset{\star}{\Longrightarrow} v_{i,1} \cap \cdots \cap v_{i,m}$. Hence by (b) there exists a derivation of the form

$$u_{i,1} \cap \cdots \cap u_{i,m} \overset{\star}{\Longrightarrow} t_1 \cap \cdots \cap t_l \overset{\star}{\Longrightarrow} sexp''$$

for some t_j, $1 \leq j \leq l$, such that $d_i \in [\![sexp'']\!]$, $\mathcal{N}(u_{i,1} \cap \cdots \cap u_{i,m}) = \{t_1, \cdots, t_l\}$, and $sexp''$ does not contain any projection symbols. Hence there exist t'_j, $1 \leq j \leq l$ such that $t_j \overset{\star}{\Longrightarrow} t'_j$ and $sexp''$ is $t_1 \cap \cdots \cap t_l$ and $d_i \in [\![t'_j]\!]$. Now $T_i = \mathcal{N}(u_{i,1} \cap \cdots \cap u_{i,m})$, and hence by (a), $V_{T_i} \overset{\star}{\Longrightarrow} v_1 \cap \cdots \cap v_p$ such that $\{v_1, \cdots, v_p\} = \{t'_1, \cdots, t'_l\}$. In particular $v_1 \cap \cdots \cap v_p$ does not contain any projection symbols since $t'_1, \cdots, t'_l$ appear in $sexp''$ and $sexp''$ does not contain any projection symbols. Also $d_i \in [\![v_1 \cap \cdots \cap v_p]\!]$. Since d_i has fewer symbols than d, the induction hypothesis may be applied to yield a derivation $V_{T_i} \overset{\star}{\Longrightarrow} a_i$ such that a_i is atomic and $d_i \in [\![a_i]\!]$. Summarizing:

$$\begin{array}{lll} V_T & \Longrightarrow & f(V_{T_1}, \cdots, V_{T_k}) \\ V_{T_1} & \stackrel{\star}{\Longrightarrow} & a_1 \\ & \cdots & \\ V_{T_k} & \stackrel{\star}{\Longrightarrow} & a_k \end{array}$$

Hence there is a derivation $V_T \stackrel{\star}{\Longrightarrow} f(a_1, \cdots, a_k)$ where $f(a_1, \cdots, a_k)$ is atomic and $f(d_1, \cdots, d_k) \in [\![f(a_1, \cdots, a_k)]\!]$.

Proof for part (d): The proof proceeds by induction on the length of the derivation $X \stackrel{\star}{\Longrightarrow} sexp$. The base case, where the derivation is of length 0 is vacuous since a derivation $X \stackrel{\star}{\Longrightarrow} sexp$ of length 0 implies that X is $sexp$, however $sexp$ is ground and X is a variable. Now suppose that part (d) holds for derivations of length less than κ, let $X \stackrel{\star}{\Longrightarrow} sexp$ be a derivation of length κ, and let $d \in [\![sexp]\!]$. Write the derivation $X \stackrel{\star}{\Longrightarrow} sexp$ as

$$X \Longrightarrow s_1 \cap \cdots \cap s_n \stackrel{\star}{\Longrightarrow} sexp$$

where the s_i do not contain the intersection symbol. That the derivation can be written this way follows from the fact that any expressions which appear nested in $\mathcal{G}$ are atomic (see proposition 10). Let $X_1, \cdots, X_m$ denote the occurrences of variables in $s_1 \cap \cdots \cap s_n$. For each occurrence X_i, there is an expression x_i such that $X_i \stackrel{\star}{\Longrightarrow} x_i$ and such that $sexp$ is the result of replacing each occurrence X_i in $s_1 \cap \cdots \cap s_n$ by x_i. Since the length of the derivations $X_i \stackrel{\star}{\Longrightarrow} x_i$, $1 \leq i \leq m$, are each less than κ, the induction hypothesis can be applied to obtain

If $d \in [\![x_i]\!]$, then there exists an atomic a such that $d \in [\![a]\!]$ and $X_i \stackrel{\star}{\Longrightarrow} a$. (4.8)

We now prove by induction that for each j, $1 \leq j \leq n$, there exists a derivation of the form

$$X \Longrightarrow t_1 \cap \cdots \cap t_j \cap s_{j+1} \cap \cdots \cap s_n \stackrel{\star}{\Longrightarrow} u_1 \cap \cdots \cap u_n \tag{4.9}$$

such that $t_1, \cdots, t_j$ are are non-variable atomic expressions, $u_1, \cdots, u_n$ are ground expressions, and

- $d \in [\![u_i]\!]$, $1 \leq i \leq n$;
- $t_i \stackrel{\star}{\Longrightarrow} u_i$, $1 \leq i \leq j$;
- $s_i \stackrel{\star}{\Longrightarrow} u_i$, $j+1 \leq i \leq n$, and
- $u_1, \cdots, u_j$ are atomic expressions.

The case where $j = 0$ is immediate. Now suppose that (4.9) holds for α, and consider $\alpha + 1 \leq n$. The expression $s_{\alpha+1}$ is either (i) a variable, (ii) a non-variable atomic expression, or (iii) of the form $f^{-1}(s)$. In case (i), $s_{\alpha+1}$ is one of the X_i. Hence by (4.8) there is an atomic a such that a is derivable from X_i and $d \in [\![a]\!]$. Pick a minimum length derivation of a from X_i. Then, by proposition 12, this derivation has the form $s_{\alpha+1} \Longrightarrow u \stackrel{\star}{\Longrightarrow} a$ such that u is a non-variable atomic expression. Since $\mathcal{G}$ is closed under the substitution transformation, $\mathcal{G}$ must contain the transformation

$$X \Longrightarrow t_1 \cap \cdots \cap t_\alpha \cap u \cap s_{\alpha+2} \cap \cdots \cap s_n,$$

and hence the following derivation can be constructed in $\mathcal{G}$

$$X \Longrightarrow t_1 \cap \cdots \cap t_\alpha \cap u \cap s_{\alpha+2} \cap \cdots \cap s_n \stackrel{\star}{\Longrightarrow} u_1 \cap \cdots \cap u_\alpha \cap a \cap u_{\alpha+1} \cap \cdots \cap u_n$$

and this proves (4.9) for $j = \alpha + 1$.

In case (ii), the proof is immediate, since $s_{\alpha+1}$ is already a non-variable atomic expression, and hence $t_{\alpha+1}$ can be set to $s_{\alpha+1}$.

Finally, in case (iii), $s_{\alpha+1}$ is of the form $f^{-1}_{(l)}(v)$ for some k-ary function symbol f such that $1 \leq l \leq k$ and $f^{-1}_{(l)}(v) \stackrel{\star}{\Longrightarrow} u_{\alpha+1}$ and $d \in [\![u_{\alpha+1}]\!]$. This means that $u_{\alpha+1}$ must be of the form $f^{-1}_{(l)}(u)$ such that $v \stackrel{\star}{\Longrightarrow} u$ and there exists a term of the form $d' = f(d_1, \cdots, d_k) \in [\![u]\!]$ such that d_l is d. Now, since v is a nested expression in $\mathcal{G}$, it must be an atomic expression. Also, each variable in v is one of the occurrences X_i. Recall that the occurrences of variables in v are denoted by $X_1, \cdots, X_m$. Now, where w is a subexpression of v, define that $w[y_1, \cdots y_m]$ is the expression obtained by replacing any occurrence X_i which also appears within w by y_i. In particular $X_i[y_1, \cdots y_m] = y_i$. Also, if $f(w_1, \cdots, w_k)$ is a subexpression of v, then $f(w_1, \cdots, w_k)[y_1, \cdots y_m] = f(w_1[y_1, \cdots y_m], \cdots, w_k[y_1, \cdots y_m])$. We now prove, for all subexpressions w of v, that

$$\begin{array}{l}\text{If } e \in [\![w[x_1, \cdots, x_m]]\!] \text{ then there exists an atomic } a \\ \text{such that } w \stackrel{\star}{\Longrightarrow} a \text{ and } e \in [\![a]\!]\end{array} \tag{4.10}$$

The proof is by structural induction on subexpressions w of v. If w is a variable, then w is X_i for some i. Hence $w[x_1, \cdots, x_m]$ is just x_i. But x_i is atomic, and so a can be chosen as x_i. If w is $\perp$, then $e \in [\![w]\!]$ is not possible and so no a is needed. If w is $\top$, then a can be chosen to be $\top$. Finally, suppose that w is of the form $f(w_1, \cdots, w_k)$ such that (4.10) holds for $w_1, \cdots, w_k$. If $e \in f(w_1, \cdots, w_k)[x_1, \cdots, x_m]$, then e is of the form $f(e_1, \cdots, e_k)$ and $e_i \in w_i[x_1, \cdots, x_m]$, $1 \leq i \leq k$. Hence applying the assumption that (4.10) holds for

$w_1, \cdots, w_k$ implies that there exist $a_1, \cdots, a_k$ such that $e_i \in [\![a_i]\!]$ and $w_i \stackrel{\star}{\Longrightarrow} a_i$, $1 \leq i \leq k$. Clearly $f(a_1, \cdots, a_k)$ is atomic, $w \stackrel{\star}{\Longrightarrow} f(a_1, \cdots, a_k)$ and $f(e_1, \cdots, e_k) \in [\![f(a_1, \cdots, a_k)]\!]$. This proves that (4.10) holds for all subexpressions w of v. In particular it holds for v. Hence there exists an atomic a_v such that $v \stackrel{\star}{\Longrightarrow} a_v$ and $d \in [\![a_v]\!]$.

We now show that $\mathcal{G}$ must contain a rule of the form

$$X \Longrightarrow t_1 \cap \cdots \cap t_\alpha \cap f_{(l)}^{-1}(v') \cap u_{\alpha+2} \cap \cdots \cap s_n \tag{4.11}$$

such that $v' \stackrel{\star}{\Longrightarrow} a_v$ and v' is not a variable. If v is already a variable, then there is nothing to show. If v is a variable, then choose the derivation $v \stackrel{\star}{\Longrightarrow} a_v$ to be a minimal length derivation of a_v from v. Now, by proposition 12 this derivation is of the form $v \Longrightarrow v' \stackrel{\star}{\Longrightarrow} a_v$ such that v' is not a variable. Hence $\mathcal{G}$ contains the rules

$$\begin{array}{lcl} X & \Longrightarrow & t_1 \cap \cdots \cap t_\alpha \cap \cap f_{(l)}^{-1}(v) \cap u_{\alpha+2} \cap \cdots \cap s_n \\ v & \Longrightarrow & v' \end{array}$$

and since $\mathcal{G}$ is closed under the substitution transformation, it must also contain the rule (4.11).

Since $d' \in f_{(l)}^{-1}(a_v)$, v' must either be $\top$ or $f(v'_1, \cdots, v'_k)$. Also $v' \stackrel{\star}{\Longrightarrow} a_v$, and so ORACLE$(\mathcal{G}, v')$ is *true*. Hence if v' is $\top$, then the preconditions of the $\top$-distribution transformation are satisfied, and if v' is $f(v'_1, \cdots, v'_k)$ then the preconditions of the $\cap$-distribution transformation are satisfied. Since $\mathcal{G}$ is closed under both transformations, it must contain a rule of the form

$$X \Longrightarrow t_1 \cap \cdots \cap t_\alpha \cap t_{\alpha+1} \cap u_{\alpha+2} \cap \cdots \cap s_n \tag{4.12}$$

where $t_{\alpha+1}$ is either $\top$ or v'_l. Recalling the d' is of the form $f(d_1, \cdots, d_k)$ such that d_l is d, $f(v_1, \cdots, v_k) \stackrel{\star}{\Longrightarrow} a_v$, a_v is atomic and $d' \in [\![a_v]\!]$, it follows that a_v is of the form $f(a_1, \cdots, a_k)$ such that $a_1, \cdots, a_k$ are atomic, $v_i \stackrel{\star}{\Longrightarrow} a_i$, and $d_i \in [\![a_i]\!]$. Hence $v_l \stackrel{\star}{\Longrightarrow} a_l$ for some atomic a_l such that $d = d_l \in [\![a_l]\!]$. This completes case (iii), and also completes the proof that (4.9) holds for all $1 \leq j \leq n$.

We now use (4.9) to complete the proof for part (d). In the case where $j = n$, (4.9) implies that there is a derivation $X \Longrightarrow t_1 \cap \cdots \cap t_n$ such that $t_i \stackrel{\star}{\Longrightarrow} u_i$, $1 \leq i \leq n$, the u_i are atomic and $d \in [\![u_i]\!]$. Hence there exists an expression $sexp'$ which does not contain any projection symbols such that $X \stackrel{\star}{\Longrightarrow} sexp'$ and $d \in [\![sexp']\!]$. Hence part (c) can be applied to complete the proof for part (d). $\square$

Combining these three propositions, we have

Lemma 3 (Grammar Simplification Algorithm) *Let $\mathcal{G}$ be a set expression grammar in which all nested expressions are atomic and let $\mathcal{G}$ be input to the grammar simplification algorithm. Then*

- *the algorithm terminates and outputs some set expression grammar $\mathcal{G}'$;*
- *$\mathcal{G}'$ is a term grammar;*
- *$\mathcal{G}'$ is equivalent to $\mathcal{G}$ in the sense that:*
  for all $X \in var(\mathcal{G})$, $\mathcal{I}_{\mathcal{G}'}(X) = \mathcal{I}_{\mathcal{G}}(X)$. $\square$

Proof: We first prove that the algorithm terminates. Let $\mathcal{G}_0$ denote the grammar input to the algorithm. Execution of the **repeat** loop of the algorithm then defines a sequence of grammars $\mathcal{G}_0$, $\mathcal{G}_1$, $\mathcal{G}_2$, ... such that

$$\mathcal{G}_0 \rightarrow \mathcal{G}_1 \rightarrow \mathcal{G}_2 \rightarrow \ldots$$

Furthermore, each $\mathcal{G}_i$, $i \geq 1$ contains all the rules in $\mathcal{G}_{i-1}$ plus at least one new rule. Now, the subexpression invariant proposition proves that, for all $i \geq 0$, all nested expressions $sexp$ of $\mathcal{G}_i$ are atomic and are such that $\mathcal{N}(sexp)$ contains only atomic subexpressions of $\mathcal{G}_0$. This means that if $X \Longrightarrow sexp$ is a rule added during the algorithm, then

- any occurrence in $sexp$ of an expression of the form $s \cap t$ or $f_{(i)}^{-1}(s)$ cannot appear inside the scope of a function or projection symbol, but it may appear inside the scope of an $\cap$ symbol, and
- any occurrence in $sexp$ of an expression of the form $f_{(i)}^{-1}(s)$ is such that s is atomic.

Hence any rule added during execution of the algorithm must be of the form

$$X \Longrightarrow a_1 \cap \ldots \cap a_k,\ k \geq 1$$

where each a_j, $1 \leq j \leq k$ is either

- an atomic expression, in which case it must also appear in $\mathcal{G}_0$;

- of the form $f_{(i)}^{-1}(a)$ where a is an atomic expression which appears in $\mathcal{G}_0$, f is an n-ary function symbol which appears in $\mathcal{G}_0$ and $1 \leq i \leq n$;
- a variable which appears in $\mathcal{G}_0$, or
- a introduced variable of the form V_S such that $S \subseteq atomic(\mathcal{G}_0)$.

Furthermore, because duplicate expressions are removed from a rule before the rule is added to the grammar by the algorithm, the a_j must be distinct. It follows that there are only a finite number of distinct rules that can be added during execution of the algorithm. Since only new rules are added, it must eventually be the case that a grammar $\mathcal{G}_m$ is obtained such that no transformation is applicable. The main loop of the algorithm will then terminate, and a grammar $\mathcal{G}' \subseteq \mathcal{G}_m$ is output.

That the output grammar $\mathcal{G}'$ is in fact a term grammar is clear since each rule $X \Longrightarrow sexp$ in $\mathcal{G}'$ is a rule from $\mathcal{G}_m$ which satisfies the extra property that $sexp$ does not contain intersection or projection symbols (that is, $sexp$ is atomic).

To complete the proof, it remains to show that $\mathcal{G}'$ is equivalent to $\mathcal{G}$. Now, repeated application of the correctness of the individual transformations (proposition 9) proves that

for all $X \in var(\mathcal{G})$, $\mathcal{I}_{\mathcal{G}_m}(X) = \mathcal{I}_{\mathcal{G}}(X)$

Therefore, it is sufficient to prove that $\mathcal{I}_{\mathcal{G}'} = \mathcal{I}_{\mathcal{G}_m}$. First, $\mathcal{I}_{\mathcal{G}'} \subseteq \mathcal{I}_{\mathcal{G}_m}$ because $\mathcal{G}'$ is a subset of $\mathcal{G}_m$ and so any derivation $X \stackrel{\star}{\Longrightarrow} sexp$ in $\mathcal{G}'$ is also a derivation in $\mathcal{G}_m$. Now, to prove $\mathcal{I}_{\mathcal{G}'} \supseteq \mathcal{I}_{\mathcal{G}_m}$, let X be a variable and let $d \in \mathcal{I}_{\mathcal{G}_m}(X)$. This implies that there exists an ground expression such that $X =_{\mathcal{G}_m}\Rightarrow\star\, sexp$ and $d \in [\![sexp]\!]$. Since none of the transformations are applicable to $\mathcal{G}_m$, lemma 2 is applicable. Hence there exists a ground atomic expression t such that $X =_{\mathcal{G}_m}\Rightarrow\star\, t$ and $d \in [\![t]\!]$. Now if the derivation $X =_{\mathcal{G}_m}\Rightarrow\star\, t$ used rules which contained intersection or projection symbols, then the result of the derivation would also contain such symbols. However, t is atomic, and so each rule used must be of the form $Y \Longrightarrow a$ where a is atomic. Hence all such rules are present in $\mathcal{G}'$. Therefore $X =_{\mathcal{G}'}\Rightarrow\star\, t$, which implies that $d \in \mathcal{I}_{\mathcal{G}'}(X)$. This completes the proof that $\mathcal{I}_{\mathcal{G}'} = \mathcal{I}_{\mathcal{G}_m}$. $\square$

We now return to the problem of showing that the function ORACLE used in transformation 2 is recursive. We recall that ORACLE($\mathcal{G}, X$) is true iff there exists a ground atomic set expression such that $X \stackrel{\star}{\Longrightarrow} t$ and $[\![t]\!] \neq \emptyset$. Since t does not contain projection or intersection symbols, its derivation from X in $\mathcal{G}$ cannot involve rules which contain a projection symbol or intersection symbol. Hence ORACLE($\mathcal{G}, X$) iff ORACLE($\mathcal{G}', X$) where $\mathcal{G}'$ is the subset of rules from $\mathcal{G}$ which do not contain the projection symbol or intersection symbol. Such a grammar $\mathcal{G}'$ is in fact a term grammar. Hence standard algorithms can be used to determine whether or not $\mathcal{I}_{\mathcal{G}'}(X) = \emptyset$, and this completes an implementation of ORACLE. Summarizing the sub-algorithms described in this subsection, we have:

Theorem 1 *Let $\mathcal{C}$ be a reduced form conjunction of set constraints. The algorithm described in this subsection, when input with $\mathcal{C}$:*

- *terminates, yielding a term grammar $\mathcal{G}$, and*
- *$\mathcal{I}_{\mathcal{G}}(X) = lm_{\mathcal{C}}(X)$ for all $X \in var(\mathcal{C})$.* □

5 Definite Set Constraints

We now apply the algorithms developed in section 4 to the the satisfiability problem for set constraints. We first prove a least model property.

Proposition 13 *A conjunction $\mathcal{C}$ of definite set constraints has a least model whenever it has a model.* □

Proof: Let $\mathcal{M}$ be the collection of models of $\mathcal{C}$. We prove that the intersection of these models defined by

$$(\bigcap \mathcal{M})(X) \stackrel{\text{def}}{=} \bigcap_{\mathcal{I} \in \mathcal{M}} \mathcal{I}(X),\ X \in VAR$$

is also a model of $\mathcal{C}$. Since this model is contained in all models in $\mathcal{M}$, the proposition then follows. This model intersection property is proved using the following two facts[12]

$$\begin{aligned} (\bigcap \mathcal{M})(a) &= \bigcap_{\mathcal{I} \in \mathcal{M}} \mathcal{I}(a) \\ (\bigcap \mathcal{M})(sexp) &\subseteq \bigcap_{\mathcal{I} \in \mathcal{M}} \mathcal{I}(sexp) \end{aligned}$$

where a and $sexp$ are an atomic set expression and a set expression respectively. These two facts can be easily verified using a structural induction argument on a and $sexp$ respectively.

[12]Note that $\supseteq$ in the second fact cannot be strengthened to equality.

Now, let $a \supseteq sexp$ be a constraint in $\mathcal{C}$. Then $\mathcal{I}(a) \supseteq \mathcal{I}(sexp)$ for all $\mathcal{I} \in \mathcal{M}$. Combining this with the above two facts gives:

$$\begin{aligned} (\bigcap \mathcal{M})(a) &= \bigcap_{\mathcal{I} \in \mathcal{M}} \mathcal{I}(a) \\ &\supseteq \bigcap_{\mathcal{I} \in \mathcal{M}} \mathcal{I}(sexp) \\ &\supseteq (\bigcap \mathcal{M})(sexp). \end{aligned}$$

It follows that $\bigcap \mathcal{M}$ is a model of each constraint in $\mathcal{C}$. Hence $\bigcap \mathcal{M}$ is the least model of $\mathcal{C}$. $\square$

The following transformation serves to simplify the l.h.s. of each expressions in a conjunction of definite set constraints.

Transformation 5 (Flattening) *Let $\mathcal{C}$ be a conjunction of definite constraints, and let $f(t_1, \cdots, t_n) \supseteq sexp$, $n \geq 1$, be a constraint in $\mathcal{C}$. Then replace this constraint with the constraints $t_1 \supseteq f^{-1}_{(1)}(sexp)$, $\cdots$, $t_n \supseteq f^{-1}_{(n)}(sexp)$.* $\square$

This transformation does not preserve equivalence because the transformed conjunction admits, in general, more models than the original conjunction possesses. However, we do have

Lemma 4 *Let $\mathcal{C}$ be a conjunction of definite set constraints, and let $\mathcal{C}'$ be the result of one or more applications of the flattening transformation on $\mathcal{C}$. Then*

- *if $\mathcal{C}'$ is unsatisfiable, then $\mathcal{C}$ is unsatisfiable;*
- *if $lm_{\mathcal{C}'} \not\models \mathcal{C}$, then $\mathcal{C}$ is unsatisfiable;*
- *if $lm_{\mathcal{C}'} \models \mathcal{C}$, then $lm_{\mathcal{C}'} = lm_{\mathcal{C}}$.* $\square$

Proof: Let application of the flattening transformation replace the constraint

$$f(t_1, \cdots, t_n) \supseteq sexp \tag{5.13}$$

by the constraints

$$t_1 \supseteq f^{-1}_{(1)}(sexp), \cdots, t_n \supseteq f^{-1}_{(n)}(sexp) \tag{5.14}$$

We first show that

(a) If $\mathcal{I}$ is a model of (5.13) then it is a model of (5.14).

(b) If $\mathcal{I}$ is a model of (5.14) such that $\mathcal{I}(sexp) \subseteq f(HU, \cdots, HU)$, then $\mathcal{I}$ is a model of (5.13).

To show (a), suppose that $\mathcal{I}$ is a model of (5.13). We now show that this implies that $\mathcal{I}$ is a model of $t_i \supseteq f^{-1}_{(i)}(sexp)$, $1 \leq i \leq n$. Suppose that $d \in \mathcal{I}(f^{-1}_{(i)}(sexp))$. Then there must exist a term $f(d_1, \cdots, d_n)$ in $\mathcal{I}(sexp)$ such that $d = d_i$. Now since $\mathcal{I}(f(t_1, \cdots, t_n)) \supseteq \mathcal{I}(sexp)$, $f(d_1, \cdots, d_n) \in \mathcal{I}(f(t_1, \cdots, t_n))$, and so $d = d_i \in \mathcal{I}(t_i)$. Hence $\mathcal{I}$ is a model of $t_i \supseteq f^{-1}_{(i)}(sexp)$ over all i. It follows that $\mathcal{I}$ is a model of $\mathcal{C}'$.

To show (b), suppose that $\mathcal{I}$ is a model of (5.14). Since $\mathcal{I}(sexp) \subseteq f(HU, \cdots, HU)$ we have that

$$f(f^{-1}_{(1)}(\mathcal{I}(sexp)), \cdots, f^{-1}_{(n)}(\mathcal{I}(sexp))) = \mathcal{I}(sexp).$$

Now $\mathcal{I}(t_i) \supseteq \mathcal{I}(f^{-1}_{(i)}(sexp))$, $1 \leq i \leq n$, and hence:

$$\begin{array}{rcl} \mathcal{I}(f(t_1, \cdots, t_n)) & = & f(\mathcal{I}(t_1), \cdots, \mathcal{I}(t_n)) \\ & \supseteq & f(\mathcal{I}(f^{-1}_{(1)}(sexp)), \cdots, \mathcal{I}(f^{-1}_{(n)}(sexp))) \\ & = & f(f^{-1}_{(1)}(\mathcal{I}(sexp)), \cdots, f^{-1}_{(n)}(\mathcal{I}(sexp))) \\ & = & \mathcal{I}(sexp) \end{array}$$

We now complete the proof by showing that (a) and (b) imply the lemma. First, if (5.14) has no models, then (a) implies that (5.13) cannot have any models. This proves the first part of the lemma. Second, suppose that $lm_{\mathcal{C}'} \not\models \mathcal{C}$. Then, applying (b) with $\mathcal{I}$ replaced by $lm_{\mathcal{C}'}$, proves that $lm_{\mathcal{C}'}$ must map $sexp$ into a set which is not contained in $f(HU, \ldots, HU)$. This means that there exists a term $d \in lm_{\mathcal{C}'}(sexp)$ whose top-most symbol is not f. Let $\mathcal{I}$ be a model of $\mathcal{C}$. By (a) $\mathcal{I}$ is a model of $\mathcal{C}'$ and so $\mathcal{I} \supseteq lm_{\mathcal{C}'}$. Since $d \in lm_{\mathcal{C}'}(sexp)$, $d \in \mathcal{I}(sexp)$. But d cannot be an element of $\mathcal{I}(f(t_1, \ldots, f_n))$, and so $\mathcal{I}$ cannot be a model of $\mathcal{C}$, which contradicts the choice of $\mathcal{I}$. Hence $\mathcal{C}$ cannot have any models. This proves the second part of the lemma.

To prove the third part of the lemma, suppose that $lm_{\mathcal{C}'} \models \mathcal{C}$. This implies that $lm_{\mathcal{C}} \subseteq lm_{\mathcal{C}'}$. Also, from (a), $lm_{\mathcal{C}'} \subseteq lm_{\mathcal{C}}$. Thus $lm_{\mathcal{C}} = lm_{\mathcal{C}'}$, and the third and final part of the lemma is proved. $\square$

An exhaustive application of the flattening transformation on a conjunction $\mathcal{C}$ can be easily shown to terminate. Constraints in the resulting conjunction $\mathcal{C}'$ are of the form (a) $X \supseteq sexp$ or (b) $c \supseteq sexp$ where X is variable, c is a constant and $sexp$ is a set expression. Let $active(\mathcal{C}')$ denote the subset of $\mathcal{C}'$ corresponding to (a). The following is the key property we require of constraints like $\mathcal{C}'$:

Lemma 5 *Let $\mathcal{C}$ be a conjunction of definite set constraints where each constraint is of the form $X \supseteq sexp$ or $c \supseteq sexp$ such that sexp is a set expression, X is a variable and c is a constant. Then*

- $lm(active(\mathcal{C}))$ *exists;*
- *if* $lm(active(\mathcal{C})) \not\models \mathcal{C}$, *then* $\mathcal{C}$ *is unsatisfiable;*
- *if* $lm(active(\mathcal{C})) \models \mathcal{C}$, *then* $lm(active(\mathcal{C})) = lm_{\mathcal{C}}$. □

Proof: It is clear that $lm(active(\mathcal{C}))$ exists since $active(\mathcal{C})$ is a reduced form conjunction of constraints. To prove the remaining two parts of the lemma, let $c \supseteq sexp$ be a constraint in $\mathcal{C} - active(\mathcal{C})$. Since $active(\mathcal{C}) \subseteq \mathcal{C}$, $lm(active(\mathcal{C})) \subseteq \mathcal{I}$ where $\mathcal{I}$ is any model of $\mathcal{C}$. Hence if $\mathcal{C}$ has a model $\mathcal{I}$, then $lm_{\mathcal{C}}$ satisfies $c \supseteq sexp$ because

$$lm_{\mathcal{C}}(c) = \mathcal{I}(c) \supseteq \mathcal{I}(sexp) \supseteq lm_{\mathcal{C}}(sexp).$$

In summary then, if $\mathcal{C}$ has a model, then $lm(active(\mathcal{C}))$ is also a model of $\mathcal{C}$. Hence $\mathcal{C}$ is satisfiable if and only if $lm(active(\mathcal{C}))$ is also a model of $\mathcal{C}$. Moreover, if $\mathcal{C}$ has a model, then $lm(active(\mathcal{C})) \supseteq lm_{\mathcal{C}}$ and $lm(active(\mathcal{C})) \subseteq lm_{\mathcal{C}}$. □

Recall that given a term grammar representation of an interpretation $\mathcal{I}$, we can determine whether or not an arbitrary set constraint is *true* in $\mathcal{I}$. Combining this with Theorem 1 and the above lemmas, we have that the following algorithm decides the satisfiability of an input conjunction $\mathcal{C}$ of definite set constraints.

> **exhaustively-apply** the flattening transformation to $\mathcal{C}$ to obtain $\mathcal{C}'$;
> **run** the grammar simplification algorithm on $active(\mathcal{C}')$ to obtain grammar $\mathcal{G}$;
> **if** $\mathcal{I}_{\mathcal{G}} \models \mathcal{C}$ **then** output $\mathcal{G}$;
> **else** output "unsatisfiable";

Theorem 2 *Let $\mathcal{C}$ be a conjunction of definite set constraints. Then*

- *the above algorithm terminates terminates on input* $\mathcal{C}$*;*
- *if* $\mathcal{C}$ *is satisfiable, the algorithm outputs a term grammar* $\mathcal{G}$ *such that* $\mathcal{I}_{\mathcal{G}}(X) = lm_{\mathcal{C}}(X)$ *for all* $X \in var(\mathcal{C})$*;*
- *if* $\mathcal{C}$ *is unsatisfiable, the algorithm outputs "unsatisfiable".* □

Proof: It is clear that the algorithm terminates since (i) an exhaustive application of the flattening transformation clearly terminates (each application removes one function symbol from the l.h.s. of the constraints), and (ii) lemma 3 proves that the grammar simplification algorithm terminates. That the algorithm is correct can be proved as follows. Let $\mathcal{C}'$ and $\mathcal{I}_\mathcal{G}$ be as constructed by the grammar above algorithm. Then

$$\mathcal{I}_\mathcal{G}(X) = (lm(active(\mathcal{C}')))\,(X) \text{ for all } X \in var(active(\mathcal{C}')).$$

We further assume that the new variables introduced by the grammar simplification algorithm are different from those in $var(\mathcal{C}) - var(active(\mathcal{C}'))$. Clearly it is always possible to choose the new variables so that this is the case. Hence $\mathcal{I}_\mathcal{G}(X) = (lm(active(\mathcal{C}')))\,(X)$ for all $X \in var(\mathcal{C})$, and so $\mathcal{I}_\mathcal{G}$ satisfies $\mathcal{C}$ if and only if $lm(active(\mathcal{C}'))$ satisfies $\mathcal{C}$. From lemma 5 and repeated application of lemma 4, we have

$$\mathcal{C} \text{ is satisfiable} \quad \text{iff} \quad lm_{active(\mathcal{C}')} \models \mathcal{C} \quad \text{iff} \quad \mathcal{I}_\mathcal{G} \text{ satisfies } \mathcal{C}.$$

Also, if $\mathcal{C}$ is satisfiable, then

$$lm_\mathcal{C}(X) = lm_{\mathcal{C}'}(X) = \mathcal{I}_\mathcal{G}(X) \text{ for all } X \in var(\mathcal{C}). \quad \square$$

References

[1] W. Ackermann, "Solvable Cases of the Decision Problem", Studies in Logic and Foundations of Mathematics, North-Holland, 1962.

[2] J. Doner, "Tree Acceptors and Some of their Applications", *Journal of Computer and System Sciences* 4, pp 406 - 451, 1970.

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

[4] N. Heintze and J. Jaffar, "A Finite Presentation Theorem for Approximating Logic Programs", IBM technical report RC 16089 (# 71415), August 1990, 66 pp. (An abstract of this paper appears in *Proc. ACM-POPL-90*, pp. 197-209.)

[5] 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.

[6] R. McNaughton, "Parenthesis Grammars", *Journal of the ACM* 14(3), pp 490 - 500, 1967.

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

[8] M.O. Rabin, "Decidability of Second-order Theories and Automata on Infinite Trees", *Transactions of the American Math. Society* 141, pp 1 - 35, 1969.

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

[10] J.W. Thatcher and J.B. Wright, "Generalized Finite Automata with an Application to a Decision Problem of Second-order Logic", *Mathematical Systems Theory* 2, pp 57 - 82, 1968.

Appendix I: Grammar Simplification Examples

This appendix contains two examples illustrating the construction of the least model of reduced form set constraints. The relevant algorithms here are the grammar form algorithm and the grammar simplification algorithm (see section 4). Relevant transformations are indicated in parenthesis. Rules added by multiple applications of the substitution transformation are denote by (substitution*), and the intermediate rules are omitted. Finally, the rules which are output by the grammar simplification algorithm are indicated by †.

Example 1 Consider the following conjunction of constraints

$$\begin{array}{lcl} X & \supseteq & (f(X,Y) \cap f(Y,Z)) \ \cup \ a \\ Y & \supseteq & a \cup f(Y,Y) \\ Z & \supseteq & a \end{array}$$

After the preprocessing steps, the following set expression grammar is obtained and input to the grammar simplification algorithm.

$$\begin{array}{llcl} & X & \Longrightarrow & f(X,Y) \cap f(Y,Z) \\ \dagger & X & \Longrightarrow & a \\ \dagger & Y & \Longrightarrow & a \\ \dagger & Y & \Longrightarrow & f(Y,Y) \\ \dagger & Z & \Longrightarrow & a \end{array}$$

The grammar simplification algorithm then adds the following rules.

$$\begin{array}{lrcll} \dagger & X & \Longrightarrow & f(Z_{\{X,Y\}}, Z_{\{Y,Z\}}) & (\cap\text{-distribution}) \\ & Z_{\{X,Y\}} & \Longrightarrow & X \cap Y & \\ & Z_{\{Y,Z\}} & \Longrightarrow & Y \cap Z & \\ & Z_{\{X,Y\}} & \Longrightarrow & a \cap a & (\text{substitution*}) \\ & Z_{\{X,Y\}} & \Longrightarrow & a \cap f(Y,Y) & (\text{substitution*}) \\ & Z_{\{X,Y\}} & \Longrightarrow & f(Z_{\{X,Y\}}, Z_{\{Y,Z\}}) \cap a & (\text{substitution*}) \\ & Z_{\{X,Y\}} & \Longrightarrow & f(Z_{\{X,Y\}}, Z_{\{Y,Z\}}) \cap f(Y,Y) & (\text{substitution*}) \\ & Z_{\{Y,Z\}} & \Longrightarrow & f(Y,Y) \cap a & (\text{substitution*}) \\ & Z_{\{Y,Z\}} & \Longrightarrow & a \cap a & (\text{substitution*}) \\ \dagger & Z_{\{X,Y\}} & \Longrightarrow & f(Z_{\{X,Y\}}, Z_{\{Y,Z\}}) & (\cap\text{-distribution}) \\ \dagger & Z_{\{X,Y\}} & \Longrightarrow & a & (\cap\text{-distribution}) \\ \dagger & Z_{\{Y,Z\}} & \Longrightarrow & a & (\cap\text{-distribution}) \end{array}$$

Example 2 This example illustrates the simplification of constraints involving projection symbols. Consider the constraint $X \supseteq a \cup f(f(f(f(X)))) \cup f^{-1}(f^{-1}(X))$. After the preprocessing steps, the following grammar is obtained

$$
\begin{array}{clcl}
\dagger & X & \Longrightarrow & a \\
\dagger & X & \Longrightarrow & f(f(f(f(X)))) \\
 & X & \Longrightarrow & f^{-1}(X_1) \\
 & X_1 & \Longrightarrow & f^{-1}(X)
\end{array}
$$

The grammar simplification algorithm then adds the following rules.

$$
\begin{array}{clcll}
 & X_1 & \Longrightarrow & f^{-1}(f(f(f(f(X))))) & \text{(substitution)} \\
\dagger & X_1 & \Longrightarrow & f(f(f(X))) & \text{(projection elimination)} \\
 & X & \Longrightarrow & f^{-1}(f(f(f(X)))) & \text{(substitution)} \\
\dagger & X & \Longrightarrow & f(f(X)) & \text{(projection elimination)} \\
 & X_1 & \Longrightarrow & f^{-1}(f(f(X))) & \text{(substitution)} \\
\dagger & X_1 & \Longrightarrow & f(X) & \text{(projection elimination)} \\
 & X & \Longrightarrow & f^{-1}(f(X)) & \text{(substitution)} \\
 & X & \Longrightarrow & X & \text{(projection elimination)}
\end{array}
$$

Appendix II: Program Approximation Examples

One application of set constraints is in the approximate modeling of programs. The idea here is to extract from a program a conjunction of constraints which capture an approximation of the desired program property. In this section we will use this approach to obtain a conservative approximation of the values that program variables may assume during execution. We give only an informal outline of how the appropriate set constraints may be obtained from a program; a more detailed formulation of the extraction process is currently being investigated.

Example 3 Consider a simple imperative programming language with basic facilities for data structure creation (e.g. *cons* and *nil* for list creation) and data destructuring/projection (e.g. *car* and *cdr* for list destructuring). We also assume a simple template matching facility. This takes the form of a condition, $x == template$, which is *true* if the value of x is an instance of *template*. Now, let P be the program

0. $x := cons(1, cons(2, cons(3, nil)))$;
1. **while** $(x == cons(z_1, cons(z_2, z_3)))$ $\{$
2. $\quad y := car(x)$;
3. $\quad x := cdr(x)$;
4. $\}$

The focus of this example is the while loop at lines 1-4 which, for concreteness, we have preceded by a very simple initialization to the variable x. The general idea, however, is that the loop will be in the context of some larger program. Then, the possible values for x on entry to the loop will be determined, not by a simple initialization, but by set constraints corresponding to the rest of the program[13].

To model the above program P, we first introducing set variables, corresponding to the program variables, whose intent is to capture the possible values that the program variables may take. Constraints corresponding to P may then be written[14]:

---

[13] In particular, the set of possible values for x may be infinite.

[14] The specific use of the complement symbol here is inconsequential, and the algorithms described in this paper can be easily adapted to solve these constraints.

$$\begin{array}{lll}
(A) & X_{\text{entry}} & \supseteq & cons(1, cons(2, cons(3, nil))) \\
(B) & X_{\text{while}} & \supseteq & X_{\text{entry}} \\
(C) & X_{\text{body}} & \supseteq & X_{\text{while}} \cap cons(\top, cons(\top, \top)) \\
(D) & Y & \supseteq & cons^{-1}_{(1)}(X_{\text{body}}) \\
(E) & X_{\text{while}} & \supseteq & cons^{-1}_{(2)}(X_{\text{body}}) \\
(F) & X_{\text{exit}} & \supseteq & X_{\text{while}} \cap \overline{cons(\top, cons(\top, \top))}
\end{array}$$

where $cons^{-1}_{(1)}$ and $cons^{-1}_{(2)}$ respectively model *car* and *cdr*. Constraint (A) corresponds to line 0; constraints (B) and (C) correspond to line 1; constraints (D) and (E) correspond to lines 2 and 3 respectively; and constraint (F) corresponds to lines 1-4. The four variables X_{entry}, X_{while}, X_{body} and X_{exit} are intended to capture, respectively, the values of the variable X (a) before the execution of the while statement, (b) at the head of the while statement (c) at the start of the body of the while statement and (d) after execution of the while statement. (A single set variable could be used to capture all of these uses of X, resulting in a simpler, but less accurate, analysis.) The set of values of y is modeled by the set variable Y. The least model of the above constraints describes a conservative approximations to program variable values and is given by

$$\begin{array}{lll}
X_{\text{entry}} & = & \{cons(1, cons(2, cons(3, nil)))\} \\
X_{\text{while}} & = & \{cons(1, cons(2, cons(3, nil))), \\
 & & \ cons(2, cons(3, nil)), \\
 & & \ cons(3, nil)\} \\
X_{\text{body}} & = & \{cons(1, cons(2, cons(3, nil))), \\
 & & \ cons(2, cons(3, nil))\} \\
Y & = & \{1, 2\} \\
X_{\text{exit}} & = & \{cons(3, nil)\}.
\end{array}$$

Example 4 An important aspect of the set constraint approach to program approximation which is not illustrated example 1, is the ability to uniformly deal with infinite sets of values. We conclude with a simple example of an analysis to approximate the success set of a logic program. Let m and n be some given positive integers. Let $P_{m,n}$ be the logic program

$p(a).$
$p(s^m(X)) \leftarrow p(X).$
$p(Y) \leftarrow p(s^n(Y)).$

Then, on letting Υ denote an approximation to the set of terms t such that $p(t)$ is true, we obtain an approximation to the meaning of p by noting that either (a) t

is a; (b) t is $s^m(t')$ such that t' is in Υ; or (c) t is such that $s^n(t)$ is in Υ. Writing this as a set constraint, we obtain:

$$\Upsilon \supseteq a \cup s^m(\Upsilon) \cup s^{-n}(\Upsilon).$$

The least model of this constraint gives $\Upsilon = \{s^{\alpha k}(a)\}$ where k is the greatest common factor of m and n, and α ranges over the natural numbers. Hence, for this example, we have in fact obtained an exact description of the success set of $P_{m,n}$.