SET CONSTRUCTORS, FINITE SETS, AND LOGICAL SEMANTICS

BHARAT JAYARAMAN AND DEVASHIS JANA

▷ The use of sets in declarative programming has been advocated by several authors in the literature. A representation often chosen for finite sets is that of *scons*, parallel to the list constructor *cons*. The logical theory for such constructors is usually tacitly assumed to be some formal system of classical set theory. However, classical set theory is formulated for a general setting, dealing with both finite and infinite sets, and not making any assumptions about particular set constructors. In giving logical-consequence semantics for programs with finite sets, it is important to know exactly what connection exists between sets and set constructors. The main contribution of this paper lies in establishing these connections rigorously. We give a formal system, called *SetAx*, designed around the *scons* constructor. We distinguish between two kinds of set constructors, $scons(x, y)$ and $dscons(x, y)$, where both represent $\{x\} \cup y$, but $x \in y$ is possible in the former, while $x \notin y$ holds in the latter. Both constructors find natural uses in specifying sets in logic programs. The design of *SetAx* is guided by our choice of *scons* as a primitive symbol of our theory rather than as a defined one, and by the need to deduce nonmembership relations between terms, to enable the use of *dscons*. After giving the axioms *SetAx*, we justify it as a suitable theory for finite sets in logic programming by (i) showing that the set constructors indeed behave like finite sets; (ii) providing a framework for establishing the correctness of set unification; and (iii) defining a Herbrand structure and providing a basis for discussing logical consequence semantics for logic programs with finite sets. Together, these results provide a rigorous foundation for the set constructors in the context of logical semantics.

KEYWORDS: set constructors, finite sets, Zermelo-Fraenkel set theory, logical-consequence semantics, freeness axioms, set unification, Herbrand structure ◁

Address correspondence to Bharat Jayaraman, Department of Computer Science, State University of New York at Buffalo, 226 Bell Hall, Buffalo, NY 14260, E-Mail: bharat@cs.buffalo.edu

1. INTRODUCTION

The use of sets has been advocated by several authors in the literature on logic programming [5, 6, 8, 14, 17, 20] and deductive databases [1, 2, 26]. In studying the inclusion of sets in logic programs, it is natural to study finite sets at first. A representation often chosen for finite sets is that of *scons*, parallel to the list constructor *cons*. The use of this constructor for declarative programming was first introduced in [16] and has been advocated by other authors in the literature as well. The logical theory for such constructors is usually tacitly assumed to be some formal system of classical set theory, such as *Zermelo-Fraenkel* (ZF) set theory [7, 29]. However, classical set theory is formulated for a general setting, dealing with both finite and infinite sets, and not making any assumptions about particular set constructors. In giving logical-consequence semantics for logic programs with finite sets, it is important to know exactly what connection exists between sets and set constructors. The main contribution of this paper lies in establishing these connections rigorously, thereby filling an important gap in the literature.

We give a formal system, called $SetAx$, whose purpose is to facilitate giving logical semantics for logic programs involving finite sets and set constructors. We can distinguish two kinds of set constructors: $scons(x, y)$ and $dscons(x, y)$. In both cases, the set represented is $\{x\} \cup y$. However, in the former, $x \in y$ is possible, whereas in the latter, $x \notin y$ is required to hold. The *dscons* is called the disjoint set constructor (from $\{x\} \cap y = \emptyset$). Both constructors find natural uses in specifying sets in logic programs. The *scons* is used for specifying sets in terms of parts that may well overlap with each other. The *dscons* is used for specifying sets in terms of an element and remainder. The *scons* constructor in [15, 16, 17] has been used in both senses—on the left-hand sides of rules it was used to mean *dscons*, while on the right-hand sides of rules it was used to mean *scons*.

Two factors complicate the design of *SetAx*: First, *scons* is a primitive symbol in our theory, rather than a defined one. This is natural in logic programming where data constructors are primitive symbols and form the basis of a Herbrand domain. As a result, possible simplifications in the classical axioms have to be investigated, because appropriate sets are guaranteed to exist by virtue of being nameable by scons terms, i.e., terms formed from *scons*. The other factor is that, because of the *dscons* constructor, we need to be able to deduce nonmembership relations, and as a consequence, certain inequalities between objects. As a result, an appropriate set of freeness axioms have to be investigated. The freeness axioms should be such that terms that are not provably equal from the other axioms can be proved to be unequal using the freeness axioms.

A natural question that arises at the outset is whether we could have adapted some well-known axiom system instead of *SetAx*. An obvious candidate is ZF without the axiom of infinity (ZF^-), together with a definition for *scons*, such as $set(y) \to scons(x, y) = \{x\} \cup y$, and an axiom like $set(x) \to finite(x)$ that makes all sets finite. Another candidate might be the von-Neumann-Bernays-Goedel ($vNBG$) axiomatization of set theory, especially the clauses given by Boyer *et al* [3]. The choice of *scons* as a primitive symbol, however, allows simplifications in the axioms; the axiom system *SetAx* was developed by exploring the precise nature of these simplications. For example, in the above definition of *scons*, the axiom in ZF expressing the closure of union is required to justify the definition. With *scons* as a primitive, such an axiom becomes unnecessary due to the presence of certain scons terms. We find that the axioms $SetAx$ do have a simpler form than ZF^- or $vNBG$—almost all of them are in definite clausal form—with intuitive 'permutation' and 'absorption' properties of *scons* forming the basis of its equational theory, instead of extensionality of ZF or $vNBG$. We first described this axiom system in [12], and a different axiomatization of this set constructor was formulated in [4, 5] for the {log} programming language. We provide a detailed comparison between these two systems in section 8.

After giving the axioms $SetAx$, we justify it as a suitable theory for finite sets in logic programming. A primary concern with $SetAx$ is its consistency, especially given the presence of inequality assertions in its freeness axioms. We therefore first build a model for $SetAx$ through an inductive construction. We then show that the set constructors indeed behave like finite sets in classical set theory. That is, we show that $SetAx^-$, i.e., $SetAx$ without the freeness axioms, is equivalent to ZF restricted to finite sets. We use ZF for this purpose, rather than $vNBG$, since ZF is generally better known. To justify the freeness axioms, basically we need to show that if $SetAx^- \not\models (t_1 = t_2)$ then $SetAx \models \neg(t_1 = t_2)$. That is, terms that are not provably equal from the rest of the axioms can be proved unequal with the aid of the freeness axioms. We also show that either $SetAx \models (t_1 = t_2)$ or $SetAx \models (t_1 = t_2)$, thereby showing that $SetAx$ is complete for all equality atoms of the above form.

In order to justify the freeness axioms, we appeal to unification, as the two are closely related. The unification procedure for set constructors is far more involved than first-order unification since the set constructors have a nontrivial equality theory. As a consequence, the unification of two set terms could result in multiple maximally general unifiers. For example, unifying $\{x, y\} = \{1, 2\}$ results in two solutions: (i) $\{x \mapsto 1,\ y \mapsto 2\}$, and (ii) $\{x \mapsto 2,\ y \mapsto 1\}$. A very brief sketch of such a unification procedure was first given in [17] and subsequently improved upon in [4]. Our set unification procedure is based upon that in [5], but our presentation follows the conventions and definitions laid out in [18], *i.e.,* the unification procedure is presented using rewrite rules, and explicit existential quantification is used for the new variables introduced during unification. Such an approach is necessitated since we have an untyped system, unlike the axiom system in [5]; it also facilitates a simpler statement of the theorems pertaining to the justification of freeness.

The unification of terms built up from *scons* is a very special case of ACI-unification [19]. Examples of ACI-unification include the unification of set terms built up the $\cup$ (union) constructor and unification in boolean rings [24]. The interested reader may consult the reference [27] for a catalog of several different types of unification problems, including their complexity. Note that *scons* is not associative, commutative, or idempotent, but enjoys closely related properties, namely, $scons(x, scons(y, s)) = scons(y, scons(x, s))$ and $scons(x, scons(x, s)) = scons(x, s)$. These properties are more correctly referred to as *permutation* and *absorption*, respectively. The *dscons* operator bears the same relation with $+$ (disjoint-union) as *scons* bears with $\cup$. Unlike $\cup$ and $+$, the first argument of *scons* and *dscons* does not have to be a set. We consider *scons* and *dscons* (rather than $\cup$ or $+$) because they have proven to be of more practical interest in set-oriented declarative programming languages [16, 2, 4, 5, 17, 28].

Finally, we establish the standard or Herbrand structure, and justify that the permutation and absorption properties of *scons* are indeed adequate for deriving the structure, as sometimes assumed in the literature. We also show that the Herbrand interpretations model *SetAx*, which is not immediate since *SetAx* is not all in definite clausal form. Together, these results provide a rigorous foundation for the set constructors in the context of logical semantics.

The rest of this paper is organized as follows. In section 2 we give the set constructors in a first-order logical framework. We give a transform for *dscons* so that it can be understood in terms of *scons*. In section 3, we present the set axioms *SetAx*, in section 4 we prove their consistency, and in section 5 we show their equivalence with finite set theory. In section 6, we use set unification to justify the adequacy of the freeness axioms in *SetAx*. In section 7, we establish the Herbrand structure and show that Herbrand interpretations model *SetAx*. Finally, in section 8, we present our conclusions and further comparisons with related work.

Our usage of standard definitions of logic programming follows that in [22]. We also make use of a few results from [7] and [29] in relating our set constructors to finite sets.

2. SET CONSTRUCTORS

We consider the set constructors in a logical framework, i.e., a first-order language **L** with equality, having an alphabet Σ possessing a set of variables Σ_V, a finite set of constructor symbols Σ_C, a finite set of predicate symbols Σ_P, and auxiliary symbols. Constructor symbols are used to build data objects, i.e., terms. Typical symbols are: $u, v, w, x, y, z, u_0, v', \ldots$ for variables, $a, b, c, d, a_0, b', \ldots$ for constructor symbols, $s, t, s_0, t', \ldots$ for terms, and $p, q, r, p_0, q', \ldots$ for predicate symbols.

We require that Σ_C contain the symbols $\{scons, \emptyset\}$ to represent sets, and Σ_P contain the symbols $\{=, set, \in\}$ to represent set predicates (we include equality among the set predicates). As a result, we will need the usual axioms of equality, viz., reflexivity and substitutivity, which we call *EqAx* (and which we lay out in section 3). Let $\Sigma_{\overline{C}}$ denote $\Sigma_C \backslash \{scons, \emptyset\}$ to refer to the non-set constructor symbols. We intend the constant $\emptyset$ to denote the empty set and the term $scons(s, t)$ to denote $\{s\} \cup t$ if t denotes a set. We do not take *dscons* as a primitive constructor, but will introduce it as a conditionally defined symbol since it is very close in meaning to *scons*. Terms of the form $scons(s, t)$ and $dscons(s, t)$ are called scons terms and dscons terms respectively. All terms that are not sets will be considered as individuals. More suggestive notation for $scons(x, y)$ and $dscons(x, y)$ are $\{x/y\}$ and $\{x \backslash y\}$ respectively. We take *dscons* to be conditionally defined as follows. (All our logical definitions are labelled by Dn.)

(D1): $x \notin y \to \{x \backslash y\} = \{x/y\}$

When $x \in y$, we regard it as a *don't care* situation. So we intend $\{x \backslash t\}$ to denote $\{x\} \cup t$ if t denotes a set and $x \notin t$. Thus $\{1, 2\}$ can be represented by $\{1/\{2/\emptyset\}\}$, $\{2/\{1/\{2/\emptyset\}\}\}$, $\{1\backslash\{2\backslash\emptyset\}\}$, etc., but not by $\{2\backslash\{1\backslash\{2\backslash\emptyset\}\}\}$.

We will follow the convention that

$\{t_1, t_2, \ldots, t_n/t_{n+1}\}$

denotes $\{t_1/\{t_2/\cdots/\{t_n/t_{n+1}\}\cdots\}\}$ for $n \geq 0$. When we know that a scons or dscons term is an individual, e.g., $scons(1, 2)$, we will usually use *scons* or *dscons* instead of the brace notation. The set of variables occurring in a syntactic object X will be denoted by $Var(X)$. An arrow over a symbol will denote a tuple of objects, i.e., $\vec{t} = (t_1, \ldots, t_m)$, for some $m \geq 0$. Similarly, $\vec{x} = \vec{y}$ will be used as an abbreviation for $x_1 = y_1 \wedge \cdots \wedge x_n = y_n$ for some $n \geq 0$. Finally, $\equiv$ will be used for syntactic identity.

The possible syntactic forms a term may have are, as usual, x or $c(t_1, \ldots, t_n)$, for some $n \geq 0$. An alternative useful set of syntactic forms for terms is as follows. Every term has one of the forms $\{t_1, t_2, \ldots, t_m/x\}$, $\{t_1, t_2, \ldots, t_m/\emptyset\}$, or $\{t_1, t_2, \ldots, t_m/c(\vec{t'})\}$ with $c \in \Sigma_{\overline{C}}$ and $m \geq 0$. When $m = 0$ we take the terms to be, respectively, x, $\emptyset$, and $c(\vec{t'})$. Thus, any term t has the form $\{t_1, \ldots, t_m/s\}$, for some $m \geq 0$ and some s whose outermost symbol is not *scons*, and this form is unique. It is useful to define a function $last(t)$ such that $last(t) = s$.

The following programs `P1` and `P2` provide illustrations of the use of the set constructors. In program `P1`, the intended mode of use of `list_set` is to convert a list into a set, while in program `P2`, the intended mode of use of `permute_set` is to convert a set of elements into a list of some permutation of the elements, with no duplications.

```
P1:
list_set([], ∅)
list_set([x|y], {x/z}) ← list_set(y,z)

P2:
permute_set(∅, [])
permute_set({x\y}, [x|z]) ← permute_set(y,z)
```

Thus, the query `list_set([1,1], x)` leads to the answer `x = {1/{1/∅}} = {1}` and the query `permute_set({1/{1/∅}}, x)` leads to the answer `x = [1]`.

In the SuRE programming language (Subsets, Relations and Equations) [13], we can express such mode information more declaratively by defining these same operations using equational and subset assertions, as shown below (note that SuRE functions can be invoked only with ground arguments):

```
P3:
list_set([]) = ∅
list_set([x|y]) = {x/list_set(y)}

P4:
permute_set(∅) ⊇ {[]}
permute_set({x\y}) ⊇ {[x|z]}  ← permute_set(y) ⊇ {z}
```

We assume that programs in the first-order language **L** are in clausal form. Clauses that contain dscons terms have to undergo a transformation, called the *disjointness transformation*, because of the way we view the *dscons* constructor. The transformation adds appropriate nonmembership atoms to the bodies of these clauses. For example, if $\{x\backslash\{y\backslash z\}\}$ is used in a clause C, then it is changed to $\{x/\{y/z\}\}$, and $y \notin z \wedge x \notin \{y/z\}$ is added to the body of C. For example, the transform, applied to `P2` above, would yield

```
permute_set({x/y}, [x|z]) ← permute_set(y,z), x∉y
```

Without such a transform, since clauses are universally quantified, an instance of the clause C could well have $y \in z$ or $x \in \{y/z\}$. The meaning of the clause would then depend upon some arbitrary don't care case of the definition of *dscons*, something that is not desirable.

Formally, the transformation is given by means of the operations $\tilde{t}$ and $disjoint(t)$ where t may contain dscons subterms. The operation $disjoint(t)$ explicates all the relevant nonmembership conditions arising from t, and the operation $\tilde{t}$ converts all the *dscons* in t to *scons*. Recall that $\vec{t} = (t_1, \ldots, t_m)$. Let $\tilde{\vec{t}} = (\tilde{t_1}, \ldots, \tilde{t_m})$, and let

$$nonmem(\vec{t}) = nonmem(t_1) \wedge \cdots \wedge nonmem(t_m).$$

Then $disjoint(t)$ is given by:

$$disjoint(x) = true, x \in \Sigma_V$$
$$disjoint(c(\vec{t})) = disjoint(\vec{t}), \text{ for } c \in \Sigma_C$$
$$disjoint(\{t_1 \backslash t_2\}) = disjoint(t_1) \wedge disjoint(t_2) \wedge \tilde{t_1} \notin \tilde{t_2}$$

Thus, for a term t possibly containing dscons symbols, if $disjoint(t)$ holds then the dscons symbols in t can be understood in the intended sense. More precisely, we have

$$EqAx \models disjoint(t) \rightarrow t = \tilde{t},$$

where $EqAx$ is given in section 3.

Let $\varphi(\vec{s})$ be a quantifier-free formula with $\vec{s}$ being the tuple of terms occuring in φ, and possibly containing dscons subterms. Then, $(\forall nonmem(\vec{s}))\varphi(\tilde{\vec{s}})$ expresses that φ is universally closed relative to the intended usage of *dscons*. Similarly, $(\exists nonmem(\vec{s}))\varphi(\tilde{\vec{s}})$ expresses that φ is existentially closed relative to the intended usage of *dscons*. In particular, when φ is a clause $A(\vec{s}) \leftarrow B(\vec{t})$, where $\vec{s}$ and $\vec{t}$ are the tuples of terms appearing in the head and body, then the disjoint transform of the clause is given by:

$$A(\tilde{\vec{s}}) \leftarrow B(\tilde{\vec{t}}), disjoint(\vec{s}), disjoint(\vec{t}).$$

3. THE SET AXIOMS

We now axiomatise the intended interpretations of the set-theoretic predicates and constructors in a theory *SetAx*. To do so, we will also need definitions to express the logical statements more meaningfully. Since some of the definitions will need some of the axioms or their consequences for their justification, and since some of the axioms are better stated using some of the definitions, we will give the definitions, axioms and theorems in an interleaved manner. We repeat here the definition D1, for the sake of completeness.

(D1): $x \notin y \rightarrow \{x \backslash y\} = \{x/y\}$
(D2): $(y \subseteq z \leftrightarrow \forall x(x \in y \rightarrow x \in z)) \leftarrow set(y) \wedge set(z)$
(D3): $indiv(x) \leftrightarrow \neg set(x)$

Defined symbols augment the set of terms and formulae. We adopt the following convention. Unless otherwise stated, a term will ordinarily contain only primitive symbols, while a formula may contain defined predicate symbols as well.

Let *SetAx* be the following axioms and axiom schemas:

Finite Sets ($FinSetAx$):
(FS1): $set(\emptyset)$
(FS2): $set(\{x/y\}) \leftarrow set(y)$
(FS3): $indiv(c(\vec{z}))$ for all $c \in \Sigma_{\overline{C}}$
(FS4): $indiv(\{x/y\}) \leftarrow indiv(y)$
(FS5): $induction(z, \vec{u}, \psi) \leftarrow set(z)$ where $induction(z, \vec{u}, \psi) \equiv$
$[\psi(\emptyset, \vec{u}) \wedge [\forall x \forall y (set(y) \wedge x \in z \wedge y \subseteq z \wedge \psi(y, \vec{u})) \rightarrow \psi(\{x/y\}, \vec{u})]] \rightarrow \psi(z, \vec{u})$
and ψ involves only set predicates and definitions,
and $\vec{u}$ are its extra free variables.

Membership ($MemAx$):
(M1): $x \notin \emptyset$
(M2): $x \notin y \leftarrow indiv(y)$
(M3): $(x \in \{y/z\} \leftrightarrow x = y \vee x \in z) \leftarrow set(z)$
(M4): $set(z) \wedge z \neq \emptyset \rightarrow \exists x(x \in z \wedge \forall y(y \in x \rightarrow y \notin z))$

Permutation and Absorption ($PAAx$):
(PA1): $\{x/\{y/z\}\} = \{y/\{x/z\}\} \leftarrow set(z)$
(PA2): $\{x/\{x/z\}\} = \{x/z\} \leftarrow set(z)$

Equality ($EqAx$):
(I1): $x = x$
(I2): $\vec{x} = \vec{y} \leftarrow x_1 = y_1 \wedge \ldots \wedge x_n = y_n$
(I3): $c(\vec{x}) = c(\vec{y}) \leftarrow \vec{x} = \vec{y}$ for all $c \in \Sigma_C$
(I4): $(p(\vec{x}) \leftrightarrow p(\vec{y})) \leftarrow \vec{x} = \vec{y}$ for all $p \in \Sigma_P$

Freeness ($FreeAx$):
(F1): $c(\vec{x}) \neq d(\vec{y})$ for all $c, d \in \Sigma_C$ such that $c \not\equiv d$
(F2): $c(\vec{x}) \neq c(\vec{y}) \leftarrow \neg(\vec{x} = \vec{y})$ for all $c \in \Sigma_{\overline{C}}$
(F3): $\big(scons(x_1, x_2) \neq scons(y_1, y_2) \leftarrow \neg(x_1 = y_1 \wedge x_2 = y_2)\big)$
$\leftarrow indiv(x_2) \wedge indiv(y_2)$
(F4): $t[x] \neq x \leftarrow indiv(t[x])$
(F5): $\{s[x]/y\} \neq x$

We briefly motivate the above axioms. Axioms FS1–4 state which terms denote sets and which denote individuals. The induction axiom FS5 is needed to capture the notion of finiteness of sets, as is well known in set theory. While at first it may not appear natural, the need for it becomes apparent in proving even simple properties such as Proposition 2 below. The membership axioms M1–3 state when two terms are in the member relation and when they are not. The axiom M4 is just the axiom of regularity of ZF and is included here since it is not deducible from the other axioms of $SetAx$. Axioms PA1–2 express that finite sets do not differ on account of order or repetition of their elements in their enumerations. The equality axioms $EqAx$ are the usual rules of reflexivity and substitutivity of the equality predicate.

The freeness axioms $FreeAx$ are needed to assert certain nonmembership relations, for use with the *dscons* constructor. Statements about (non)membership can be reduced to statements about (in)equalities, and vice-versa. For example, the dscons term $\{1\backslash\{2/\emptyset\}\}$ is well-defined as a set if $1 \notin \{2/\emptyset\}$, i.e., if $1 \neq 2$. More generally, we want $FreeAx$ to be such that terms that are not provably equal from the rest of the axioms can be proved unequal with the aid of $FreeAx$. As such, the axioms F1–5 are motivated by unification, i.e., the non-unifiable cases of terms, corresponding to failure of logical consequence from $SetAx$, motivate the inequality axioms. Axiom F1 expresses that different constructor symbols lead to distinct terms. Axioms F2–3 express injectivity, while axioms F4–5 are like the occurs check axiom (and were described in [17]). The notation $t[x]$ refers to a term t with x occurring somewhere in it. The axiom F5 imitates axiom M4 to a certain extent. For example $x \notin x$ can be deduced from either.

Let $SetAx^{-} = SetAx \backslash FreeAx$. Then $SetAx^{-}$ is the appropriate theory when the *scons* constructor alone is considered, while $SetAx$ is the appropriate theory when both *scons* and *dscons* are considered. Also $SetAx$ is the appropriate theory when *scons* alone together with nonmembership relations are considered. We now list some properties of $SetAx$. They find use in subsequent proofs.

Proposition 3.1. $SetAx \models \varphi$, where φ is any of:

(i) $x \in y \to set(y)$
(ii) $set(z) \to x \in \{x/z\}$
(iii) $x \in z \to x \in \{y/z\}$
(iv) $set(z) \leftrightarrow set(\{x_1/\{x_2/\cdots\{x_n/z\}\cdots\}\})$ for any $n \geq 1$
(v) $set(z) \to (x \in \{x_1/\{x_2/\cdots\{x_n/z\}\cdots\}\} \leftrightarrow x = x_1 \vee \cdots \vee x = x_n \vee x \in z)$ for any $n \geq 1$

Proposition 3.2. $SetAx \models \varphi$, where φ is any of:

(i) $x \in y \to \exists z(y = \{x/z\} \wedge set(z) \wedge x \notin z)$
(ii) $set(z) \to z = \emptyset \vee \exists x(x \in z)$
(iii) $x \in y \to \{x/y\} = y$
(iv) $set(x) \to s \notin x$ for any term s containing x.

Proof. (i): Use induction axiom FS5 with $\psi(y,x) \equiv x \in y \to \exists z(y = \{x/z\} \wedge set(z) \wedge x \notin z)$.
(ii): Use induction axiom FS5 with $\psi(z) \equiv z = \emptyset \vee \exists x(x \in z)$.
(iii)–(iv): These are immediate from (i) and other axioms. □

In models of ZF, non-well-founded sets, such as an infinite sequence $\cdots s_{k+1} \in s_k \in \cdots \in s_2 \in s_1$, do not exist. Such sequences appear possible in models of $SetAx$ and $SetAx^{-}$—for e.g., consider the sequence $s_i = \{s_{i+1}/\{i/\emptyset\}\}$ for all $i \geq 1$. (In ZF, infinite sets are used to show that such sequences cannot exist.) However, infinite sequences in which the sets repeat, i.e., $s_i = s_j$ for some $i \neq j$, do not exist, as shown by the next proposition.

Proposition 3.3. For any $n \geq 1$, there is no sequence of sets $x_1, \ldots, x_n$ such that $x_n \in x_{n-1} \in \ldots x_1 \in x_n$. That is, $SetAx \models \neg\exists x_1, \ldots, x_n(set(x_1) \wedge \cdots \wedge set(x_n) \wedge x_n \in x_{n-1} \in \ldots x_1 \in x_n)$.

Proof. Suppose there were such a sequence. Then we can use Prop. 2(i) and F5 to show a contradiction. □

4. CONSISTENCY OF SETAX

We build a structure $\mathcal{F} = \langle U, \Sigma_P, \Sigma_C \rangle$ from classical set theory as follows. We inductively construct the universe U as the union of a collection I of individuals and a collection S of finite sets. Below, $\mathcal{P}_{fin}$ stands for the finite-powerset operator, i.e., it gives the collection of all *finite* subsets of a set. We represent all finite sets in the universe by enumerating their elements within braces. Thus $\{\}$ stands for the empty set.

$$I_0 = \emptyset, \quad S_0 = \emptyset, \quad U_0 = I_0 \cup S_0$$

and for $i \geq 1$

$$\begin{aligned}I_i &= \{c(\vec{x}) \mid c \in \Sigma_{\overline{C}}, c\ n\text{-ary}, n \geq 0, \vec{x} \in U_{i-1}^n\} \\ &\quad \cup \{scons(x_1, x_2) \mid x_1 \in U_{i-1}, x_2 \in I_{i-1}\} \\ S_i &= \mathcal{P}_{fin}(U_{i-1}) \\ U_i &= I_i \cup S_i\end{aligned}$$

Finally, we define

$$I = \bigcup_{i<\omega} I_i, \quad S = \bigcup_{i<\omega} S_i, \quad U = \bigcup_{i<\omega} U_i$$

In the above construction, note that when $i = 1$, I_i and S_i simplify to $I_1 = \{c \mid c \text{ a constant in } \Sigma_C, c \not\equiv \emptyset\}$, and $S_1 = \big\{\{\}\big\}$. Also, the objects in I_i, $i \geq 1$, are not sets. If $x, y \in S$, then $x = y$ iff they have the same members. If $x, y \in I$ and $x \equiv a(\vec{u}), y \equiv b(\vec{v})$ for some $a, b \in \Sigma_{\overline{C}} \cup \{scons\}$ and some $\vec{u}, \vec{v}$ of the appropriate kind, then $x = y$ iff $a \equiv b$ and $\vec{u} = \vec{v}$.

Some properties based on the above construction are: (i) I_i, S_i, and U_i are monotonic in i, i.e., for $0 \leq i \leq j$, $I_i \subseteq I_j$, $S_i \subseteq S_j$, and $U_i \subseteq U_j$, (ii) $I \cap S = \emptyset$, and (iii) $I \cup S = U$. From these, it follows that, for every object e, there is a least level in the universe at which it occurs; and any other object mentioning e occurs at a level that is at least one higher than e. Note that saying $x \in I$ and x occurs at a least level $i > 0$ is the same as saying it is the unique i such that $x \in I_i \backslash I_{i-1}$. Similarly for $x \in S$ and $x \in U$.

The constructor interpretations in $\mathcal{F}$ are given by:

(i) $c^{\mathcal{F}}(\vec{x}) = c(\vec{x}) \qquad$ for $c \in \Sigma_{\overline{C}}$ and $\vec{x} \in U^n$,
(ii) $\emptyset^{\mathcal{F}} = \{\}$, and
(iii) $scons^{\mathcal{F}}(x, y) = scons(x, y)$, if $x \in U, y \in I$
$\quad\quad scons^{\mathcal{F}}(x, y) = \{x\} \cup y$, if $x \in U, y \in S$.

Here, $c^{\mathcal{F}}(\vec{x}) \in I$ for $c \in \Sigma_{\overline{C}}$, $\emptyset^{\mathcal{F}} \in S$, and $scons^{\mathcal{F}}(x, y) \in S$ iff $y \in S$. From the constructor interpretations, it follows from a simple inductive proof that for a term t with free variables $x, \vec{y}$, if x is assigned an object in U whose least level is $i \geq 1$, then for any assignment $\vec{y} \in U^n$, $n \geq 0$, we have $t^{\mathcal{F}} \in U_j \backslash U_{j-1}$ for some $j \geq i$. Here, if $t \equiv \{s_1, \ldots, s_m / x\}$, $m \geq 0$ and x does not occur in $s_1, \ldots, s_m$, then $j = i$ is possible. In all other cases of t, we have $j > i$.

For predicate interpretations in $\mathcal{F}$, we need only specify interpretations for the set predicate symbols, and can leave the others arbitrary. Let $x, y \in U$. Then the set predicate interpretations are:

(i) $set^{\mathcal{F}}(x) \Leftrightarrow x \in S$
(ii) $x \in^{\mathcal{F}} y \Leftrightarrow y \in S$ and $x \in y$
(iii) $x =^{\mathcal{F}} y \Leftrightarrow x = y$

Here, $x \in y$ and $x = y$ have their usual meanings, viz., membership and identity, respectively.

Theorem 4.1. $\mathcal{F} \models SetAx$

PROOF. By considering each set axiom in turn and using the above construction and its properties. We illustrate for the cases of axioms FS5, M4, PA1 and F5 below; the remaining cases are treated in [11].

(FS5): Let $z \in U$, and $\vec{u} \in U^n$, for some $n \geq 0$. Assume $set^{\mathcal{F}}(z)$, i.e., $z \in S$. Assume $\psi(\emptyset, \vec{u})$ and $\forall x \forall y(set(y) \wedge x \in z \wedge y \subseteq z \wedge \psi(y, \vec{u}) \rightarrow \psi(\{x/y\}, \vec{u}))$ are satisfied in $\mathcal{F}$. To show $(\psi(z, \vec{u}))^{\mathcal{F}}$.
Case (1): $z = \{\}$. So $(\psi(z, \vec{u}))^{\mathcal{F}}$, from the assumption.
Case (2): $z \neq \{\}$. Let $z = \{x_1, \ldots, x_k\}$, $k \geq 1$. Consider the sets $v_i \in U$, defined by: $v_i = \{x_1, \ldots, x_i\}$, for $0 \leq i \leq k$. Claim: For all $i \geq 0$, if $i \leq k$, then $(\psi(v_i, \vec{u}))^{\mathcal{F}}$ holds. Proof of the claim is by induction on i, and is trivial. Applying the claim to $i = k$, we get $v_k = z$ and $(\psi(z, \vec{u}))^{\mathcal{F}}$.

(M4): Assume $z \in S$ and $z \neq \{\}$. So $z = \{x_1, \ldots, x_m\}$ for some $m \geq 1$. If any $x_j \in I$ or $x_j = \{\}$, for $1 \leq j \leq m$, then x_j is a witness to the consequent. When every $x_j \in S$ and $x_j \neq \{\}$, for $1 \leq j \leq m$, then let i_j be the least level of occurrence of x_j. Choose $i = \min(i_1, \ldots, i_m)$ and let $x_l \in S_i$. We have $x_l = \{y_1, \ldots, y_n\}$ for some $n \geq 1$, and $y_k \in U_{i-1}$, for $1 \leq k \leq n$. So, the least level of occurrence of any y_k is below i, whereas every element of z occurs at a least level equal to or greater than i. Therefore, no element of x_l is in z and x_l is a witness to the consequent.

(PA1): Let $x, y, z \in U$. Assume $set^{\mathcal{F}}(z)$, i.e., $z \in S$. We have $(\{x/\{y/z\}\})^{\mathcal{F}} = \{x\} \cup \{y\} \cup z = \{y\} \cup \{x\} \cup z = (\{y/\{x/z\}\})^{\mathcal{F}}$.

(F5): Let $x, \vec{z}$ be the free variables of s. So $x, y, \vec{z}$ are the free variables of $scons(s, y)$ with possibly $x \equiv y$. Let $x, y \in U, \vec{z} \in U^n$ be an assignment of these variables. Let $x \in U_i \backslash U_{i-1}$.
Case (1): $y \in I$. Then $(indiv(scons(s, y)))^{\mathcal{F}}$ holds with $scons(s, y)$ containing x and different from x. So $(scons(s, y) \neq x)^{\mathcal{F}}$ by F4.
Case (2): $y \in S$. Let $s^{\mathcal{F}} \in U_j \backslash U_{j-1}$. Then $j \geq i$. Suppose $y = \{\}$. Then $(scons(s, y))^{\mathcal{F}} = \{s^{\mathcal{F}}\} \in S_{j+1} \backslash S_j$. So $j + 1 > i$ and $(scons(s, y) \neq x)^{\mathcal{F}}$. Suppose $y \neq \{\}$, i.e., $y = \{x_1, \ldots, x_m\}, m \geq 1$, and $x_1, \ldots, x_m$ are all present in U_k and k is the least such level at which they are all present. Let $l = \max(j, k)$. So $(scons(s, y))^{\mathcal{F}} = \{s^{\mathcal{F}}, x_1, \ldots, x_m\} \in S_{l+1} \backslash S_l$. Hence $l + 1 > i$, and thus it follows that $(scons(s, y) \neq x)^{\mathcal{F}}$. □

5. RELATING SETAX TO ZF

In this section we justify $SetAx^-$ to be an adequate theory of finite sets for logic programming. We do so by relating $SetAx^-$ and ZF restricted to finite sets, as follows: The language of ZF has the primitive symbols $\emptyset$, $\in$, and $=$, while other constructor and predicate symbols are neither assumed nor excluded. The symbol *set* is a defined one in ZF but equivalently may be taken as a primitive and its definition taken as an axiom. We begin with ZF without the axiom of infinity (ZF^-), together with $set(x) \rightarrow finite(x)$, which we take as an adequate mathematical theory of finite sets (see [7] Chapter 2, Section 3.6). To this we add a definition of *scons*, viz., $set(y) \to \{x/y\} = \{x\} \cup y$. From the motivations underlying $SetAx$ we see the need to make explicit assumptions about constructor symbols. We take FS3 and FS4 to be those assumptions (FS4 is an assumption about *scons* not covered by its definition). Now let $FinZF$ refer to the entire collection of these axioms. We then show that $SetAx^-$ is equivalent to $FinZF$. The forward direction essentially uses induction while the reverse direction uses well-known properties of ZF.

We list ZF^- below (except for the standard properties of reflexivity and substitutivity of the equality relation).

Definition ($SetDef$):
$set(y) \leftrightarrow \exists x(x \in y) \vee y = \emptyset$
Axiom of Extensionality ($ExtAx$):
$set(y) \wedge set(z) \to (\forall x(x \in y \leftrightarrow x \in z) \to y = z)$
Sum Axiom ($SumAx$):
$set(y) \to \exists z(set(z) \wedge \forall x(x \in z \leftrightarrow \exists u(set(u) \wedge x \in u \wedge u \in y)))$
Powerset Axiom ($PowerAx$):
$set(y) \to \exists z(set(z) \wedge \forall x(set(x) \to (x \in z \leftrightarrow x \subseteq y)))$
Axiom Schema of Replacement ($ReplAx$):
$$set(u) \to \big(\forall x \forall y \forall z(x \in u \wedge \varphi(x,y,\vec{w}) \wedge \varphi(x,z,\vec{w}) \to y = z) \\ \to \exists v(set(v) \wedge \forall y(y \in v \leftrightarrow \exists x(x \in u \wedge \varphi(x,y,\vec{w}))))\big)$$
Axiom of Regularity ($RegAx$):
$set(z) \wedge z \neq \emptyset \to \exists x(x \in z \wedge \forall y(y \in x \to y \notin z))$
Axiom of Choice ($ChoiceAx$):
For any set y there is a function f such that
for any non-empty subset z of y, $f(z) \in z$

Theorem 5.1. $SetAx \models \varphi$, where φ is any of: (i) $SetDef$, (ii) $ExtAx$, (iii) $SumAx$, (iv) $PowerAx$, (v) $RegAx$, and (vi) $ReplAx$.

Proof. (i) $SetDef$: Straightforward. The remaining are proved using the induction axiom. We just give the ψ in each case.
(ii) $ExtAx$: $\psi(y) \equiv \forall z(set(z) \wedge \forall x(x \in y \leftrightarrow x \in z) \to y = z)$.
(iii) $SumAx$: $\psi(y) \equiv \exists z(set(z) \wedge \forall x(x \in z \leftrightarrow \exists u(set(u) \wedge x \in u \wedge u \in y)))$.
(iv) $PowerAx$: $\psi(y) \equiv \exists z(set(z) \wedge \forall x(set(x) \to (x \in z \leftrightarrow x \subseteq y)))$.
(v) $ReplAx$: $\psi(u,\vec{w}) \equiv \forall x \forall y \forall z(x \in u \wedge \varphi(x,y,\vec{w}) \wedge \varphi(x,z,\vec{w}) \to y = z)$
$\to \exists v(set(v) \wedge \forall y(y \in v \leftrightarrow \exists x(x \in u \wedge \varphi(x,y,\vec{w}))))$.
At the induction step in (iii) and (iv), an auxiliary induction is required to show that a certain set, which cannot be expressed purely as a scons term, exists. For example, the overall proof for $SumAx$ is as follows.

Assume $set(y)$.

Basis: It is easy to show $\psi(\emptyset)$. Take $\emptyset$ as witness for z.

Induction Step: To show $\forall x', y'(set(y') \wedge x' \in y \wedge y' \subseteq y \wedge \psi(y') \to \psi(\{x'/y'\}))$. Assume $set(y')$, $x' \in y$, $y' \subseteq y$ and $\psi(y')$.
Case(1): $indiv(x')$. Since x' has no elements, a witness for z in $\psi(y')$ is a witness for z in $\psi(\{x'/y'\})$. It is easy to show that $\psi(\{x'/y'\}) \leftrightarrow \psi(y')$.
Case(2): $set(x')$. The intuition is as follows. Let z_1 be a witness for z in $\psi(y')$, i.e., $z_1 = \bigcup y'$. Then, we want $\bigcup\{x'/y'\} = x' \cup (\bigcup y') = x' \cup z_1$ to be a witness for z in

$\psi(\{x'/y'\})$. But, we cannot express $x' \cup z_1$ as a scons term since actual elements of x' are not known. So, we do an auxiliary induction on x' as follows.

We show that $\mathit{induction}(x', y', \rho)$ holds, where $\rho(x', y') \equiv \psi(\{x'/y'\})$, and

$$\begin{aligned}&\mathit{induction}(x', y', \rho) \equiv\\ &\quad \rho(\emptyset, y') \ \wedge\\ &\quad \forall x'', y''[\mathit{set}(y'') \wedge x'' \in x' \wedge y'' \subseteq x' \wedge \rho(y'', y') \to \rho(\{x''/y''\}, y')] \ \to \ \rho(x', y')\end{aligned}$$

The auxiliary basis step $\rho(\emptyset, y') \equiv \psi(\{\emptyset/y'\})$ is established in the same way as for case (1) since $\emptyset$ has no elements. The auxiliary induction step is established by using $\{x''/z_2\}$ as a witness for z in $\rho(\{x''/y''\}, y')$, where z_2 is a witness for z in $\rho(y'', y')$. Thus we have $\rho(x'/y')$, i.e., $\psi(\{x'/y'\})$. Hence $\psi(y)$. □

We need to develop the theory further before deducing the remaining axiom *ChoiceAx* of ZF^- from $SetAx^-$. But we can now directly borrow in $SetAx^-$ all the properties of the axioms of ZF^- deduced in above theorem.

In *SetAx* and $SetAx^-$, we adopt all the defined symbols of ZF associated with ZF^-, except *set*, which is taken as primitive in *SetAx*. These adopted symbols are given the same definitions in *SetAx* as in ZF. That the definitions are well-defined in *SetAx* follows from their well-definedness in ZF^- and from Theorem 5. We list some of these definitions below.

$$\begin{aligned}\text{(D4)}: \ &\big(\{x \mid \varphi(x, \vec{u})\} = y \leftrightarrow \mathit{set}(y) \wedge \forall x(x \in y \leftrightarrow \varphi(x, \vec{u}))\big)\\ &\qquad\qquad \leftarrow \ \exists y\big(\mathit{set}(y) \wedge \forall x(x \in y \leftrightarrow \varphi(x, \vec{u}))\big)\end{aligned}$$

Here, $\vec{u}$ are the other free variables in φ. The antecedent expresses the condition of interest, viz., that the intuitively appropriate collection specified by the set abstraction does form a (finite) set. Some of the definitions below are specified using set abstraction. In each case, the intuitively appropriate collection does exist as a set, because it is known to do so in ZF^-.

(D5): $(y \cap z = \{x \mid x \in y \wedge x \in z\}) \leftarrow \mathit{set}(y) \wedge \mathit{set}(z)$
(D6): $(y \cup z = \{x \mid x \in y \vee x \in z\}) \leftarrow \mathit{set}(y) \wedge \mathit{set}(z)$
(D7): $(\mathcal{P}(y) = \{x \mid \mathit{set}(x) \wedge x \subseteq y\}) \leftarrow \mathit{set}(y)$
(D8): $\bigcup y = \{x \mid \exists z(x \in z \wedge z \in y)\} \leftarrow \mathit{set}(y)$
(D9): $\{y, z\} = w \leftrightarrow \mathit{set}(w) \wedge \forall x(x \in w \leftrightarrow x = y \vee x = z)$
(D10): $\{x\} = \{x, x\}$,
$\{x, y, z\} = \{x, y\} \cup \{z\}$,
$\{x, y, z, w\} = \{x, y\} \cup \{z, w\}$, and so on.

Similarly, other definitions needed are of set difference $y \backslash z$, union of sets $\cup_{i=1}^{n} x_i$, ordered pair $\langle x, y\rangle$, a set being a relation $\mathit{relation}(u)$, a set being a function $\mathit{function}(f)$, and the value of a function at an argument $f(x) = y$. Their definitions are as in [29] Ch. 2,3. Using these definitions a formal statement of the Axiom of Choice can be given, and it can be deduced using induction, as is well-known in set theory ([7] p. 62).

Axiom of Choice (*ChoiceAx*):
$\mathit{set}(z) \to \exists f(\mathit{set}(f) \wedge \mathit{function}(f) \wedge \forall x(\mathit{set}(x) \wedge x \subseteq z \wedge x \neq \emptyset \to f(x) \in x))$.

Now we can show that $SetAx^- \models \mathit{ChoiceAx}$ as follows. Assume $\mathit{set}(z)$ and use induction axiom with

$$\psi(z) \equiv \exists f(\mathit{set}(f) \wedge \mathit{function}(f) \wedge \forall x(\mathit{set}(x) \wedge x \subseteq z \wedge x \neq \emptyset \to f(x) \in x)).$$

The following definition of minimality is used to give the definition of finiteness in sets. It states that a set x is minimal amongst a collection of sets y if it is minimal with respect to the order relation $\subseteq$. For example, if $y = \{\{1,2\},\{1\},\{3\}\}$, then $x = \{1\}$ is a minimal element in y, as is $\{3\}$, i.e., $minimal(\{1\}, y)$ and $minimal(\{3\}, y)$ hold. We follow Tarski's definition of finiteness which states that a set z is finite exactly when every non-empty family of subsets of z has a minimal element. The more usual definition of finiteness of a set, viz., being equinumerous to a natural number is known to be equivalent to Tarski's definition ([29] section 4.2, section 5.2).

(D11): $[minimal(x,y) \leftrightarrow [x \in y \ \wedge \ set(x) \ \wedge \ \forall z[set(z) \wedge z \in y \wedge z \subseteq x \rightarrow z = x]]] \leftarrow \ set(y)$

(D12): $\big[finite(z) \leftrightarrow \forall y[set(y) \wedge y \neq \emptyset \wedge y \subseteq \mathcal{P}(z) \rightarrow \exists x\, minimal(x,y)]\big] \ \leftarrow \ set(z)$

Some properties based on the above definitions are given next. They establish that *scons* does indeed have its intended meaning, and that *set* in $SetAx^-$ does refer to finite sets.

Theorem 5.2. $SetAx^- \models \varphi$, *where* φ *is any of:*
(i) $set(y) \rightarrow \{x/y\} = \{x\} \cup y$
(ii) $\{x_1/\{x_2/\cdots\{x_n/\emptyset\}\cdots\}\} = \{x_1, x_2, \ldots, x_n\}$ *for any* $n \geq 1$
(iii) $set(z) \rightarrow finite(z)$

Proof. (i): Assume $set(y)$. So $set(\{x/y\})$ by FS2. Also $set(\{x\} \cup y)$ by D9, D10, D7 & D4. Now we show $\forall z(z \in \{x/y\} \leftrightarrow z \in \{x\} \cup y)$, and apply *ExtAx*. We have $z \in \{x/y\} \leftrightarrow z = x \vee z \in y \leftrightarrow z \in \{x\} \vee z \in y \leftrightarrow z \in \{x\} \cup y$, by M3 and from properties of ZF^-, viz., Theorems 43 & 20, Chapter 2 [29].

(ii): By induction on n and from D9 & D10.

(iii): Assume $set(z)$ and use induction axiom with
$\psi(z) \equiv finite(z) \equiv \forall y[set(y) \ \wedge \ y \neq \emptyset \ \wedge \ y \subseteq \mathcal{P}(z) \rightarrow \exists x\, minimal(x,y)]$ □

We have not gone into the issue of the independence of each of the axioms of *SetAx* as it is inessential to this work. We note that axiom FS3 can be deduced from the remaining axioms (from F1 and Prop. 1(i)).

Finally, we have the reverse implication.

Theorem 5.3. $FinZF \models SetAx^-$

Proof. Straightforward, using the properties of ZF^-, such as in Chapters 2 and 4 of [29]. We give a few cases below; the full proof considering all cases is given in Jana's dissertation [11].

$FinSetAx$: (FS5): We have $set(z) \wedge finite(z) \rightarrow induction(z, \vec{w}, \psi)$, by Theorem 32, Chapter 4 [29]. Hence $set(z) \rightarrow induction(z, \vec{w}, \psi)$ since $set(z) \rightarrow finite(z)$ is in $FinZF$.

$MemAx$: (M3): Assume $set(z)$. Now $x = y \vee x \in z \leftrightarrow x \in \{y\} \vee x \in z \leftrightarrow x \in \{y\} \cup z \leftrightarrow x \in \{y/z\}$, by Theorems 43 & 20, Chapter 2 [29], and by definition of *scons*.

$PAAx$: (PA1): Assume $set(z)$. So $set(\{y\} \cup z)$ and $set(\{x\} \cup z)$, i.e., $set(\{y/z\})$ and $set(\{x/z\})$. From definition of *scons*, and from Theorems 22 & 21, Chapter

2 [29], we get, $\{x/\{y/z\}\} = \{x\} \cup \{y/z\} = \{x\} \cup \{y\} \cup z = \{y\} \cup \{x\} \cup z = \{y\} \cup \{x/z\} = \{y/\{x/z\}\}$. (PA2): Assume $set(z)$. So $set(\{x\} \cup z)$, i.e., $set(\{x/z\})$. From definition of *scons*, and from Theorems 22 & 23, Chapter 2 [29], we get, $\{x/\{x/z\}\} = \{x\} \cup \{x/z\} = \{x\} \cup \{x\} \cup z = \{x\} \cup z = \{x/z\}$. $\square$

6. UNIFICATION AND FREENESS IN SETAX

As noted in the introduction, our main motivation for discussing unification is to justify that the freeness axioms of *SetAx* are an adequate system to enforce inequalities among objects that are not provably equal in $SetAx^-$.

We note at the outset that certain inequalities can be deduced in $SetAx^-$ itself, such as $x \neq y \leftarrow set(x) \wedge indiv(y)$, and $(y \neq z \leftarrow \neg\forall x(x \in y \leftrightarrow x \in z)) \leftarrow set(y) \wedge set(z)$. (The latter is just the contrapositive of the converse of *ExtAx*.) Also, M4 helps to deduce inequalities like $x \neq \{x\}$ and $y \neq \{\{y\}\}$.

Our presentation of the unification procedure is based on the following proposition.

Proposition 6.1. $SetAx^- \models \varphi$, where φ is any of:

(i) $set(w) \to \big(\{x_1,\ldots,x_m/w\} = \{y_1,\ldots,y_n/w\} \leftrightarrow$
$\big(\bigvee_{j=1}^{n}((x_1 = y_j \wedge \{x_2,\ldots,x_m/w\} = \{y_1,\ldots,y_{j-1},y_{j+1},\ldots,y_n/w\})$
$\vee(x_1 = y_j \wedge \{x_1,\ldots,x_m/w\} = \{y_1,\ldots,y_{j-1},y_{j+1},\ldots,y_n/w\})$
$\vee(x_1 = y_j \wedge \{x_2,\ldots,x_m/w\} = \{y_1,\ldots,y_n/w\}))$
$\vee\exists z(set(z) \wedge w = \{x_1/z\} \wedge \{x_2,\ldots,x_m/z\} = \{y_1,\ldots,y_n/z\})\big)\big)$

(ii) $set(x) \wedge set(y) \to \big(\{x_1/x\} = \{y_1/y\} \leftrightarrow$
$\big((x_1 = y_1 \wedge x = y) \vee \exists z(set(z) \wedge x = \{y_1/z\} \wedge y = \{x_1/z\})\vee$
$(x_1 = y_1 \wedge y = \{x_1/x\}) \vee (x_1 = y_1 \wedge x = \{y_1/y\})\big)\big)$

PROOF. (i): Assume $set(w)$. ($\leftarrow$): Straightforward. ($\rightarrow$): Take cases of $x_1 \in \{y_1,\ldots,y_n\}$ and $x_1 \notin \{y_1,\ldots,y_n\}$.

Case(1) : $x_1 \in \{y_1,\ldots,y_n\}$. Let $1 \leq j \leq n$. We have a number of subcases, one for each j. Hence, we index the subcases by j. Subcase (1.j): $x_1 = y_j$. Take combinations of cases of x_1 being in and not being in $\{x_2,\ldots,x_m/w\}$, with cases of y_j being in and not being in $\{y_1,\ldots,y_{j-1},y_{j+1},\ldots,y_n/w\}$. We show two of these combinations.
Subsubcase (1.j.1): $x_1 \notin \{x_2,\ldots,x_m/w\}$, $y_j \notin \{y_1,\ldots,y_{j-1},y_{j+1},\ldots,y_n/w\}$. So, $\{x_2,\ldots,x_m/w\} = \{y_1,\ldots,y_{j-1},y_{j+1},\ldots,y_n/w\}$.
Subsubcase (1.j.2): $x_1 \notin \{x_2,\ldots,x_m/w\}$, $y_j \in \{y_1,\ldots,y_{j-1},y_{j+1},\ldots,y_n/w\}$. So, $\{x_1,\ldots,x_m/w\} = \{y_1,\ldots,y_{j-1},y_{j+1},\ldots,y_n/w\}$.
The subsubcase (1.j.4) of $x_1 \in \{x_2,\ldots,x_m/w\}$, $y_j \in \{y_1,\ldots,y_{j-1},y_{j+1},\ldots,y_n/w\}$ gives the same conclusion as subsubcase (1.j.1).

Case(2) : $x_1 \notin \{y_1,\ldots,y_n\}$. So, $x_1 \in w$, i.e., $\exists z(set(z) \wedge w = \{x_1/z\} \wedge x_1 \notin z)$. Then, $set(z) \wedge w = \{x_1/z\} \wedge x_1 \notin z$. Subcase (2.1): $x_1 \notin \{x_2,\ldots,x_m\}$. So, $\{x_2,\ldots,x_m/z\} = \{y_1,\ldots,y_n/z\}$ by removing x_1 from $\{x_1,\ldots,x_m/w\}$ and from $\{y_1,\ldots,y_n/w\}$. Hence $\exists z(set(z) \wedge w = \{x_1/z\} \wedge \{x_2,\ldots,x_m/z\} = \{y_1,\ldots,y_n/z\})$.
Subcase (2.2): $x_1 \in \{x_2,\ldots,x_m\}$. It is easy to see that $\exists z_1(set(z_1) \wedge w = \{x_1/z_1\} \wedge \{x_2,\ldots,x_m/z_1\} = \{y_1,\ldots,y_n/z_1\}$ by using $\{x_1/z\}$ as witness for z_1.

(ii): Assume $set(x)$, $set(y)$. ($\leftarrow$): Straightforward. ($\rightarrow$): Take combinations of cases of $x_1 \notin x$, $x_1 \in x$, with cases of $y_1 \notin y$, and $y_1 \in y$. We show three of these cases.

Case(1) : $x_1 \notin x$, $y_1 \notin y$. Subcase (1.1): $x_1 = y_1$. Then $x = y$. So, $x_1 = y_1 \wedge x = y$. Subcase (1.2): $x_1 \neq y_1$. Then, $x_1 \in y$, i.e., $\exists z(set(z) \wedge y = \{x_1/z\} \wedge x_1 \notin z)$. So, $\{x_1/x\} = \{y_1/\{x_1/z\}\} = \{x_1/\{y_1/z\}\}$. Then, $x_1 \notin \{y_1/z\}$. So, $x = \{y_1/z\}$. Thus, $\exists z(set(z) \wedge x = \{y_1/z\} \wedge y = \{x_1/z\})$.

Case(2) : $x_1 \notin x$, $y_1 \in y$. So, $y = \{x_1/x\}$, and $y_1 \in \{x_1/x\}$, i.e., $y_1 = x_1$ or $y_1 \in x$. Subcase (2.1): $y_1 = x_1$. So, $x_1 = y_1 \wedge y = \{x_1/x\}$. Subcase (2.2): $y_1 \in x$. So, $\exists z(set(z) \wedge x = \{y_1/z\})$, i.e., $set(z) \wedge x = \{y_1/z\}$. It is easy to see that $y = \{x_1/\{y_1/z\}\} = \{x_1/\{x_1/z\}\} = \{x_1/z\}$. Thus, $\exists z(set(z) \wedge x = \{y_1/z\} \wedge y = \{x_1/z\})$.

Case(4) : $x_1 \in x$, $y_1 \in y$. So, $x = y$, and $\exists z(set(z) \wedge x = \{x_1/z\})$, i.e., $x = \{x_1/z\}$. Therefore, $y_1 \in \{x_1/z\}$. Subcase (4.1): $y_1 = x_1$. So, $x = \{y_1/z\} \wedge y = \{x_1/z\}$, i.e., $\exists z(set(z) \wedge x = \{y_1/z\} \wedge y = \{x_1/z\})$. Subcase (4.2): $y_1 \in z$. So, $\exists z_1(set(z_1) \wedge z = \{y_1/z_1\})$, i.e., $set(z_1) \wedge z = \{y_1/z_1\}$. It is easy to see that $x = \{y_1/\{x_1, y_1/z_1\}\}$ and $y = \{x_1/\{x_1, y_1/z_1\}\}$. Thus, $\exists z(set(z) \wedge x = \{y_1/z\} \wedge y = \{x_1/z\})$ (take $\{x_1, y_1/z_1\}$ as witness). $\square$

In our context, unification involves not just solving *equality* constraints, but is generalized to solving a conjunction of atoms involving the set predicates *set*, $=$, and $\in$. We use the term *constraint* not in the specific sense in which it is meant in CLP languages [10], but rather in the general sense of an atom whose head is either a set-predicate, an equality-predicate, or a membership-predicate. The unification process repeatedly transforms a given unification problem using *rewrite rules* until it ultimately ends in simplified forms called *solved forms*. Solved forms have the desirable property that all solutions can be easily read off from them.

The rewrite rules are formed from equivalences such as those in the above proposition. Thus, an atom of the form appearing on the lhs of the equivalence is rewritten by any of the disjuncts appearing on the rhs of the equivalence. This rewriting is a nondeterministic process because, in general, there could be multiple maximally general unifiers. We would like to note that our use of a nondeterministic rewriting process should not be surprising. In this respect, set unification is similar to other unification problems where an equality theory is involved, e.g., the unification of typed lambda terms [9, 25]. The unification procedure in these papers is also given in terms of a nondeterminstic rewriting process.

From the above proposition, we see that in solving a set equation, other objects such as existential variables or the *set* predicate can appear in a disjunct on the rhs of the equivalence. Hence, our definitions below reflect these additional forms. A *constraint* χ is a conjunction of the form $\exists \vec{z}(A_1 \wedge \cdots \wedge A_n)$, where $n \geq 0$ and each A_i is of the form $set(s)$, $s = t$, or $s \in t$. The $\vec{z}$ are the bound variables of χ. The free variables of χ are denoted by $Var(\chi)$ and we seek values of these variables for which there are values of the existential variables that make the constraint hold in *SetAx*. It is convenient to abbreviate a constraint as a multiset $K = \exists \vec{z}\{A_i\}_{i=1}^{n}$ and let $Var(K)$ be its free variables.

A *solved form* ζ is a constraint of the form $\exists \vec{z}\{set(y_1), \cdots, set(y_m), x_1 = t_1, \cdots, x_n = t_n\}$ where $m, n \geq 0$, and the x_i's are distinct free variables that occur exactly once in ζ. Also, each existential variable occurs at least once in the multiset, and if any y_j is an existential variable, then it occurs among the t_i. For example, $\zeta_1 \equiv \exists z_1\{x = \{1/z_1\}, set(z_1), z = f(u,1), y = \emptyset\}$ is a solved form, and all its solutions can be read off by giving all possible set values for z_1 and all possible values for u, and thereby obtaining the values of x, z, and y. After this, the existential variable z_1 can be omitted or forgotten, leaving only the values of the free variables.

Below are the rewrite rules that can be applied to a constraint. In each rule, a selected atom (set, equality, or membership) is rewritten to the form on the rhs of the rule. One can think of this as a form of outermost rewriting (at the multiset level). We use the convention that w, x, y, and z are variables at the term-level (not variables at the meta-level). Thus, for example, rules S1 and S2 below do not overlap because S1 is applicable when the argument of *set* is $\emptyset$ but S2 is applicable only when this argument is a variable. We also remind the reader that constructors c and d in the rewrite rules E8, E9, and M3 stand for uninterpreted function symbols.

Here, F denotes failure in solving the constraint, and $\{x \mapsto t\}$ represents a substitution which we take as applied on the left. By applying a substitution to a constraint K we mean that it is applied to the multiset of K. Also, we use the phrase 'x occurs in K' iff x occurs in the multiset of K.

(S1) $K \cup \{set(\emptyset)\} \Rightarrow K$
(S2) $K \cup \{set(x)\} \Rightarrow K$
if $set(x)$ occurs in K.
(S3) $\exists z K \cup \{set(z)\} \Rightarrow K$
if $set(z)$ does not occur in K
(S4) $K \cup \{set(c(\vec{t}))\} \Rightarrow F$
if $c \in \Sigma_{\overline{C}}$
(S5) $K \cup \{set(\{s_1/s_2\})\} \Rightarrow K \cup \{set(s_2)\}$

(E1) $\exists z K \Rightarrow K$
if z does not occur in K
(E2) $\exists z K \cup \{z = t\} \Rightarrow K$
if $z \notin Var(t)$ and z does not occur in K
(E3) $K \cup \{x = x\} \Rightarrow K$
(E4) $K \cup \{x = t\} \Rightarrow \{x \mapsto t\}K \cup \{x = t\}$
if $x \notin Var(t)$ and x occurs in K
(E5) $K \cup \{x = t\} \Rightarrow \exists z K \cup \{set(z), x = t'\}$
if $x \in Var(t)$, $x \not\equiv t$,
$t \equiv \{t_1, \ldots, t_n/x\}$ with $n \geq 1$, and $x \notin Var(t_1, \ldots, t_n)$
Here $t' \equiv \{t_1, \ldots, t_n/z\}$, z a new variable
(E6) $K \cup \{x = t\} \Rightarrow F$
if $x \in Var(t)$, $x \not\equiv t$,
$\neg\ (t \equiv \{t_1, \ldots, t_n/x\}$ with $n \geq 1$ and $x \notin Var(t_1, \ldots, t_n))$
(E7) $K \cup \{t = x\} \Rightarrow K \cup \{x = t\}$
if t is not a variable
(E8) $K \cup \{c(\vec{s}) = d(\vec{t})\} \Rightarrow F$
if $c \not\equiv d$
(E9) $K \cup \{c(s_1, \ldots, s_m) = c(t_1, \ldots, t_m)\} \Rightarrow K \cup \{s_1 = t_1, \ldots, s_m = t_m\}$
if $m \geq 0$
(E10) $K \cup \{\{s_1, \ldots, s_m/w\} = \{t_1, \ldots, t_n/w\}\} \Rightarrow K \cup \{set(w), s_1 = t_j,$
$\{s_2, \ldots, s_m/w\} = \{t_1, \ldots, t_{j-1}, t_{j+1}, \ldots, t_n/w\}\}$, for $1 \leq j \leq n$

(E11) $K \cup \{\{s_1, \dots, s_m/w\} = \{t_1, \dots, t_n/w\}\} \Rightarrow K \cup \{set(w), s_1 = t_j,$
$\{s_1, \dots, s_m/w\} = \{t_1, \dots, t_{j-1}, t_{j+1}, \dots, t_n/w\}\}$, for $1 \le j \le n$
(E12) $K \cup \{\{s_1, \dots, s_m/w\} = \{t_1, \dots, t_n/w\}\} \Rightarrow K \cup \{set(w), s_1 = t_j,$
$\{s_2, \dots, s_m/w\} = \{t_1, \dots, t_n/w\}\}$, for $1 \le j \le n$
(E13) $K \cup \{\{s_1, \dots, s_m/w\} = \{t_1, \dots, t_n/w\}\} \Rightarrow \exists z K \cup \{set(z), w = \{s_1/z\},$
$\{s_2, \dots, s_m/z\} = \{t_1, \dots, t_n/z\}\}$
Here z is a new variable
(E14) $K \cup \{\{s_1/s_2\} = \{t_1/t_2\}\} \Rightarrow \exists z K \cup \{\{s_1/z\} = t_2, s_2 = \{t_1/z\}, set(z)\}$
if $\neg(last(s_2) \equiv last(t_2) \equiv$ a variable)
Here z is a new variable.
(E15) $K \cup \{\{s_1/s_2\} = \{t_1/t_2\}\} \Rightarrow K \cup \{s_1 = t_1, s_2 = \{t_1/t_2\}, set(t_2)\}$
if $\neg(last(s_2) \equiv last(t_2) \equiv$ a variable)

(E16) $K \cup \{\{s_1/s_2\} = \{t_1/t_2\}\} \Rightarrow K \cup \{s_1 = t_1, \{s_1/s_2\} = t_2, set(s_2)\}$
if $\neg(last(s_2) \equiv last(t_2) \equiv$ a variable)

(M1) $K \cup \{s \in \emptyset\} \Rightarrow F$
(M2) $K \cup \{s \in x\} \Rightarrow \exists z K \cup \{set(z), x = \{s/z\}\}$
Here, z is a new variable
(M3) $K \cup \{s \in c(\vec{t})\} \Rightarrow F$
if $c \in \Sigma_{\overline{C}}$
(M4) $K \cup \{s \in \{t_1/t_2\}\} \Rightarrow K \cup \{set(last(t_2)), s = t_1\}$
(M5) $K \cup \{s \in \{t_1/t_2\}\} \Rightarrow K \cup \{s \in t_2\}$

By successively applying rules to a constraint we obtain a derivation. For example, the following sequence constitutes a derivation where the atom being rewritten is underlined.

$\{\underline{\{1/x\} = \{u/y\}}, \{v/x\} = \{2/y\}\}$
$\Downarrow$ E14
$\exists z_1 \{set(z_1), \underline{x = \{u/z_1\}}, y = \{1/z_1\}, \{v/x\} = \{2/y\}\}$
$\Downarrow$ E4
$\exists z_1 \{set(z_1), x = \{u/z_1\}, \underline{y = \{1/z_1\}}, \{v/\{u/z_1\}\} = \{2/y\}\}$
$\Downarrow$ E4
$\exists z_1 \{set(z_1), x = \{u/z_1\}, y = \{1/z_1\}, \underline{\{v/\{u/z_1\}\} = \{2/\{1/z_1\}\}}\}$
$\Downarrow$ E9
$\exists z_1 \{set(z_1), x = \{u/z_1\}, y = \{1/z_1\}, v = 2, \underline{\{u/z_1\} = \{1/z_1\}}\}$
$\Downarrow$ E9
$\exists z_1 \{set(z_1), x = \{u/z_1\}, y = \{1/z_1\}, v = 2, \underline{u = 1}, z_1 = z_1\}$
$\Downarrow$ E4
$\exists z_1 \{set(z_1), x = \{1/z_1\}, y = \{1/z_1\}, v = 2, u = 1, \underline{z_1 = z_1}\}$
$\Downarrow$ E3
$\exists z_1 \{set(z_1), x = \{1/z_1\}, y = \{1/z_1\}, v = 2, u = 1\}$

At times, more than one rule can apply to a selected atom, such as rules E8 and E13 to E15 at the first rewriting step above. Then all possible rewritings can be recorded as multiple branches arising from that constraint. This leads to a tree of derivations.

Starting from a constraint K and using the rewrite rules above repeatedly until none is applicable results in a finite tree of derivations, each leaf of which is either F or a solved form. We use the correctness and termination proof in [5] to assert this. The termination proof is obtained by first showing that rules S1-5 easily terminate from the reduction in size of terms. We invoke the proof in [5] for the termination of rules E1-16 which are solely in terms of *set* and $=$. Finally, rules M1-5 easily terminate by strict reduction in size of terms and because we know *set* and $=$ already terminate. The termination proof requires that at any step of the derivation the leftmost atom that can be rewritten be selected for rewriting.

Proposition 6.2. Let $\Omega = \{\zeta_1, \ldots, \zeta_k\}$ for some $k \geq 0$ be the set of all solved forms obtained from the rewriting procedure for a constraint χ. Then,
(i) $SetAx^- \models \forall(\zeta_i \rightarrow \chi)$
(ii) $SetAx \models \forall(\chi \rightarrow \zeta_1 \vee \cdots \vee \zeta_k)$

Proof. (i): Consider each rewrite rule 'lhs $\Rightarrow$ rhs' in turn and show that $SetAx^- \models \forall(\text{rhs} \rightarrow \text{lhs})$. By transitivity and by finiteness of the rewriting tree for χ it follows that $SetAx^- \models \forall(\zeta_i \rightarrow \chi)$.

(ii): Consider all possible forms of an atom for rewriting. For each atom, suppose there are $n \geq 1$ applicable rewrite rules 'lhs $\Rightarrow$ rhs$_i$', for $1 \leq i \leq n$. Show that $SetAx \models \forall(\text{lhs} \rightarrow \text{rhs}_1 \vee \cdots \vee \text{rhs}_n)$. Then, by induction on the depth of the rewrite tree, it is easy to show that $SetAx \models \forall(\chi \rightarrow \zeta_1 \vee \cdots \vee \zeta_k)$. □

The following properties about $SetAx$ and $SetAx^-$ can now be deduced.

Theorem 6.1. Let $\Omega = \{\zeta_1, \ldots, \zeta_k\}$ for some $k \geq 0$ be the set of all solved forms obtained from the rewriting procedure for a constraint χ. Then,
(i) $SetAx \models \forall(\chi \leftrightarrow \zeta_1 \vee \cdots \vee \zeta_k)$
(ii) $SetAx^- \models \exists\chi \;\Leftrightarrow\; \Omega \neq \emptyset$
(iii) $SetAx \models \exists\chi \;\Leftrightarrow\; SetAx^- \models \exists\chi$
(iv) $SetAx \models \exists\chi$ or $SetAx \models \neg\exists\chi$
(v) $SetAx^- \not\models \exists\chi \;\Rightarrow\; SetAx \models \neg\exists\chi$

Proof. (i): Follows from the previous proposition and by quantifier and clausal logic.

(ii): ($\Rightarrow$) Follows from (i) above and consistency of *SetAx*. ($\Leftarrow$) Let $\zeta \in \Omega$ witness $\Omega \neq \emptyset$. Starting from $SetAx^- \models \forall(\zeta \rightarrow \chi)$, we show $SetAx^- \models \exists\chi$. Let $\zeta \equiv \exists\vec{z}(set(y_1) \wedge \cdots \wedge set(y_m) \wedge x_1 = t_1 \wedge \cdots \wedge x_n = t_n)$ where $m, n \geq 0$ and let $\vec{u} = Var(\chi)$ be the free variables of χ. In none of the rewrite rules are any new free variables introduced, so that we have $Var(\zeta) \subseteq Var(\chi)$. Hence, $SetAx^- \models \forall\vec{u}, \vec{z}(set(y_1) \wedge \cdots \wedge set(y_m) \wedge x_1 = t_1 \wedge \cdots \wedge x_n = t_n \rightarrow \chi)$.
We next substitute instances for $\vec{u}$ and $\vec{z}$ such that the antecedent holds. Let substitutions θ, σ be $\theta = \{y_1 \mapsto \emptyset, \ldots, y_m \mapsto \emptyset\}$ and $\sigma = \{x_1 \mapsto \theta t_1, \ldots, x_n \mapsto \theta t_n\}$. Clearly $\sigma\theta(set(y_1) \wedge \cdots \wedge set(y_m) \wedge x_1 = t_1 \wedge \cdots \wedge x_n = t_n)$ holds in $SetAx^-$. Hence $SetAx^- \models \forall\sigma\theta\chi$. This gives $SetAx^- \models \exists\chi$.

(iii): Show $SetAx^- \not\models \exists\chi \Rightarrow SetAx \not\models \exists\chi$, which follows from (ii) and (i) above, and from consistency of *SetAx*.

(iv): Follows from (iii), (ii), and (i) above.

(v): Follows from (ii) and (i) above. □

The property (v) above justifies the adequacy of $FreeAx$ (by taking $\chi \equiv s = t$ or $\chi \equiv s \in t$). The property (iv) above shows that $SetAx$ is a complete theory over formulae φ of the form $\exists\chi$ or $\neg\exists\chi$. We note that *SetAx* is not complete over all closed formulae. For example, $SetAx \not\models \exists x\, indiv(x)$ and $SetAx \not\models \neg\exists x\, indiv(x)$, when $\Sigma_{\overline{C}} = \emptyset$. These can be shown, respectively, by using a model of *SetAx* of pure sets and a model of *SetAx* containing at least one individual. These models can be constructed from the structure $\mathcal{F}$ in section 4.

7. HERBRAND STRUCTURE

As is usual in logic programming, we seek to focus on just one kind of interpretation, the Herbrand-like interpretations. Such interpretations have a fixed domain and fixed functional assignment for the constructor symbols. The domain is based on a quotient structure formed from the set of ground terms modulo a relation that is equality between ground terms. To denote that it is a quotient structure, we use the notation $\cong$, as in Herbrand $\cong$-structure. In the context of programming, the domain contains the data objects built from the (primitive and not defined) data constructor symbols. The predicate interpretations are allowed to vary, leading thereby to different interpretations.

We now define the standard or Herbrand $\cong$-structure $\mathcal{H} = \langle U_{\cong}, \Sigma_P, \Sigma_C\rangle$ below. Let s, t be ground terms. The relation $\cong$ is given by: $s \cong t$ iff $SetAx \models s = t$. Here, extensionality can cause syntactically different ground terms to be related — for example, $\{1/\{2/\emptyset\}\} \cong \{2/\{1/\emptyset\}\}$. By virtue of *EqAx*, $\cong$ is an equivalence relation. Let $[t]$ denote the equivalence class containing ground term t. A somewhat simpler characterization of the $\cong$-relation is usually stated in the literature ([17, 26]), viz., using just *PAAx* and *EqAx* to define the relation. Since *set* appears in *PAAx* we also need some axioms from *FinSetAx*. Let *SetPAAx* be FS1 $\cup$ FS2 $\cup$ *PAAx* $\cup$ *EqAx*. The next proposition justifies this more intuitive characterisation of $\cong$.

Proposition 7.1. Let s, t be ground terms. Then $SetAx \models s = t \;\Leftrightarrow\; SetPAAx \models s = t$.

Proof. ($\Leftarrow$) Immediate. ($\Rightarrow$) By induction on s, t. We show just the following case: $s \equiv \{s_1, \ldots, s_m/\emptyset\}, t \equiv \{t_1, \ldots, t_n/\emptyset\}, m, n \geq 1$.

Let $SetAx \models s_1 = s_{i_1}$, …, $SetAx \models s_1 = s_{i_k}$, and $SetAx \not\models s_1 = s_{i_{k+1}}$, …, $SetAx \not\models s_1 = s_{i_m}$, where $i_1, \ldots, i_m$ is a permutation of $1, \ldots, m$. We have, $k \geq 1$ since $SetAx \models s_1 = s_1$.

Let $SetAx \models s_1 = t_{j_1}$, …, $SetAx \models t_1 = t_{j_l}$, and $SetAx \not\models s_1 = t_{j_{l+1}}$, …, $SetAx \not\models s_1 = t_{j_n}$, where $j_1, \ldots, j_n$ is a permutation of $1, \ldots, n$. We have, $l \geq 1$ since $SetAx \models s_1 \in t$, i.e., $SetAx \models s_1 = t_1 \vee \cdots \vee s_1 = t_n$, i.e., $SetAx \models s_1 = t_1$ or …or $SetAx \models s_1 = t_n$, by completeness of *SetAx*. We also have $SetAx \models s_1 \notin \{s_{i_{k+1}}, \ldots, s_{i_m}/\emptyset\}$, and $SetAx \models s_1 \notin \{t_{j_{l+1}}, \ldots, t_{j_n}/\emptyset\}$. So, $SetAx \models \{s_{i_{k+1}}, \ldots, s_{i_m}/\emptyset\} = \{t_{j_{l+1}}, \ldots, t_{j_n}/\emptyset\}$.

By induction hypothesis, we have
$SetPAAx \models s_1 = s_{i_1}$, …, $SetPAAx \models s_1 = s_{i_k}$, and
$SetPAAx \models s_1 = t_{j_1}$, …, $SetPAAx \models t_1 = t_{j_l}$, and
$SetPAAx \models \{s_{i_{k+1}}, \ldots, s_{i_m}/\emptyset\} = \{t_{j_{l+1}}, \ldots, t_{j_n}/\emptyset\}$.

Thus, $SetPAAx \models \{s_1, \ldots, s_m/\emptyset\}$
$\overset{PA1}{=} \{s_{i_1}, \ldots, s_{i_k}, s_{i_{k+1}}, \ldots, s_{i_m}/\emptyset\}$
$\overset{PA2}{=} \{s_1, s_{i_{k+1}}, \ldots, s_{i_m}/\emptyset\}$

$\overset{PA2}{=} \{s_1, t_{j_{l+1}}, \ldots, t_{j_n}/\emptyset\}$
$\overset{PA2}{=} \{t_{j_1}, t_{j_{l+1}}, \ldots, t_{j_n}/\emptyset\}$
$\overset{PA2}{=} \{t_{j_1}, \ldots, t_{j_l}, t_{j_{l+1}}, \ldots, t_{j_n}/\emptyset\} \overset{PA1}{=} \{t_1, \ldots, t_n/\emptyset\}$. □

The domain $U_{\cong}$ of $\mathcal{H}$, called the *Herbrand* $\cong$*-Universe*, is: $U_{\cong} = \{[t] \mid t \text{ a ground term}\}$. The constructor interpretations in $\mathcal{H}$ are, for all $c \in \Sigma_C$: $c^{\mathcal{H}}([t_1], \ldots, [t_n]) = [c(t_1, \ldots, t_n)]$. It is easily verified that $c^{\mathcal{H}}$ is well-defined. Let p be *set*, $\in$, or $=$, and $t_1, \ldots, t_n$ be ground terms. Then, $p^{\mathcal{H}}([t_1], \ldots, [t_n])$ iff $SetAx \models p(t_1, \ldots, t_n)$. It is easily verified that $p^{\mathcal{H}}([t_1], \ldots, [t_n])$ is well-defined. The definition is motivated by our seeking the least relations that might model $SetAx$.

For example, the quotient universe construction and the above definition gives identity as the interpretation of equality, i.e., $[s] = [t]$ iff $[s] =^{\mathcal{H}} [t]$, for ground terms s, t. Additional justification for the above set-predicate interpretations is provided by the following proposition.

Proposition 7.2. Based on the above $U_{\cong}$ and constructor interpretations, the set predicate interpretations that model SetAx are unique.

Proof. This follows from the completeness of $SetAx$ over the ground set predicate atoms. Let $\mathcal{H}_1$, $\mathcal{H}_2$ be two structures with the above domain $U_{\cong}$ and above constructor interpretations, but with different set-predicate interpretations; and let $\mathcal{H}_1$, $\mathcal{H}_2$ both model $SetAx$. Let p be *set*, $\in$, or $=$, and $t_1, \ldots, t_n$ be ground terms. Now if $SetAx \models p(t_1, \ldots, t_n)$ then $p^{\mathcal{H}_1}([t_1], \ldots, [t_n])$ and $p^{\mathcal{H}_2}([t_1], \ldots, [t_n])$ must both hold. So for $\mathcal{H}_1$ and $\mathcal{H}_2$ to differ in their set-predicate interpretations, there must be a p and $t_1, \ldots, t_n$ such that $SetAx \not\models p(t_1, \ldots, t_n)$ but $p^{\mathcal{H}_1}([t_1], \ldots, [t_n])$ and not $p^{\mathcal{H}_2}([t_1], \ldots, [t_n])$ hold. However, then $SetAx \models \neg p(t_1, \ldots, t_n)$, i.e., not $p^{\mathcal{H}_1}([t_1], \ldots, [t_n])$. Contradiction. □

We will subsequently show that $\mathcal{H}$ models $SetAx$. The above proposition shows that the set-predicate interpretations given in $\mathcal{H}$ are the only possible ones that can lead to a model of $SetAx$ (based on the fixed universe and constructor interpretations). It is convenient to abbreviate $p^{\mathcal{H}}([t_1], \ldots, [t_n])$ as $[p(t_1, \ldots, t_n)]$, for a predicate symbol p. The next two propositions use the consistency of $SetAx$ and the completeness of $SetAx$ on the ground set predicate atoms. To compress the statements in the propositions, several cases which are better stated separately have been combined together.

Proposition 7.3. The interpretation of set in $\mathcal{H}$ is:
(i) not $[set(scons(s_1, \ldots, scons(s_m, c(\vec{s'})) \cdots))]$, $\quad m \geq 0, c \in \Sigma_{\overline{C}}$
(ii) $[set(\{s_1, \ldots, s_m/\emptyset\})]$, $\quad m \geq 0$

Proposition 7.4. The interpretations of $\in$ and $=$ in $\mathcal{H}$ are given (mutually recursively) below. (Symmetric cases are omitted.)
($\in$.1) not $[s \in scons(t_1, \ldots scons(t_m, c(\vec{t'})) \cdots)]$, $\quad m \geq 0, c \in \Sigma_{\overline{C}}$
($\in$.2) $[s \in \{t_1, \ldots, t_m/\emptyset\}] \Leftrightarrow [s = t_1]$ *or* $\cdots$ *or* $[s = t_m], m \geq 0$
(=.1) not $[c(\vec{s'}) = d(\vec{t'})]$, $\quad c \not\equiv d, c, d \in \Sigma_{\overline{C}}$
(=.2) $[c(s_1, \ldots, s_n) = c(t_1, \ldots, t_n)] \Leftrightarrow [s_1 = t_1]$ *and* $\cdots$ *and* $[s_n = t_n]$, $c \in \Sigma_{\overline{C}}$
(=.3) not $[scons(s_1, \ldots scons(s_m, c(\vec{s'})) \cdots) = scons(t_1, \ldots scons(t_n, d(\vec{t'})) \cdots)]$
$m, n \geq 0, m \neq n, c, d \in \Sigma_{\overline{C}}$

(=.4) $[scons(s_1, \ldots scons(s_m, c(\vec{s'})) \cdots) = scons(t_1, \ldots scons(t_m, d(\vec{t'})) \cdots)] \Leftrightarrow$
$[s_1 = t_1]$ *and* $\cdots$ *and* $[s_m = t_m]$ *and* $[c(\vec{s'}) = d(\vec{t'})]$, $m \geq 1, c, d \in \Sigma_{\overline{C}}$

(=.5) not $[scons(s_1, \ldots scons(s_m, c(\vec{s'})) \cdots) = \{t_1, \ldots, t_n/\emptyset\}]$, $m, n \geq 0, c \in \Sigma_{\overline{C}}$

(=.6) $[\{s_1, \ldots, s_m/\emptyset\} = \{t_1, \ldots, t_n/\emptyset\}] \Leftrightarrow$
$[s_1 \in \{t_1, \ldots, t_n/\emptyset\}]$ *and* $\cdots$ *and*
$[s_m \in \{t_1, \ldots, t_n/\emptyset\}]$ *and*
$[t_1 \in \{s_1, \ldots, s_m/\emptyset\}]$ *and* $\cdots$ *and* $[t_n \in \{s_1, \ldots, s_m/\emptyset\}]$, $m, n \geq 0$

In $\mathcal{H}$, while the set predicate interpretations are fixed, the non-set predicate interpretations are allowed to vary, since we want them to depend on the logic program at hand. This leads to different interpretations, and the following definition gives a suitable way to specify them. The *Herbrand* $\cong$*-Base*, $B_{\cong}$, is: $B_{\cong} = \{[A] \mid A$ a ground atom with initial symbol not a set predicate$\}$, and a *Herbrand* $\cong$*-interpretation* is a subset of $B_{\cong}$. We do not include set predicate atoms in $B_{\cong}$, because their interpretations in $\mathcal{H}$ are fixed and therefore can be factored out from consideration.

For logic programs P, one is interested in Herbrand $\cong$-models of $P \cup SetAx$. Hence it is useful to show that Herbrand $\cong$-interpretations model *SetAx*. It is not immediate that Herbrand $\cong$-interpretations model *SetAx*, since *SetAx* is not all in definite clause form. Examples are axioms FS5 and M3.

We now show that $\mathcal{H}$ models *SetAx*. We have the following correspondence between $\mathcal{H}$ and $\mathcal{F}$, viz., that $\mathcal{H}$ is isomorphic to $\mathcal{F}$, where $\mathcal{F}$ is the model of *SetAx* constructed previously. Let the correspondence be given by $(\bullet)^{\circ}\colon U_{\cong} \to U$, where

$[\emptyset]^{\circ} = \{\}$
$[c(t_1, \ldots, t_n)]^{\circ} = c([t_1]^{\circ}, \ldots, [t_n]^{\circ})$, for $c \in \Sigma_{\overline{C}}$
$[scons(t_1, \ldots, scons(t_m, c(\vec{t'})) \cdots)]^{\circ} = scons([t_1]^{\circ}, \ldots, scons([t_m]^{\circ}, [c(\vec{t'})]^{\circ}) \cdots)$,
for $c \in \Sigma_{\overline{C}}$ and $m \geq 1$
$[\{t_1, \ldots, t_m/\emptyset\}]^{\circ} = \{[t_1]^{\circ}, \ldots, [t_m]^{\circ}\}$, for $m \geq 1$

Clearly, by induction on t, we have $[t]^{\circ} \in U$. Also, $[s] = [t] \Leftrightarrow [s]^{\circ} = [t]^{\circ}$ holds, by induction on s, t, and by the previous proposition. From these, it follows that $(\bullet)^{\circ}$ is well-defined.

Theorem 7.1. $\mathcal{H}$ is isomorphic to $\mathcal{F}$, excluding the non-set predicate interpretations.

Proof. It is straightforward to show that $(\bullet)^{\circ}$ is bijective, that the constructor interpretations correspond, i.e., $(c^{\mathcal{H}}([t_1], \ldots, [t_n]))^{\circ} = c^{\mathcal{F}}([t_1]^{\circ}, \ldots, [t_n]^{\circ})$, and that the set-predicate interpretations correspond, i.e., $p^{\mathcal{H}}([t_1], \ldots, [t_n]) \Leftrightarrow p^{\mathcal{F}}([t_1]^{\circ}, \ldots, [t_n]^{\circ})$.
$\square$

Theorem 7.2. Every Herbrand $\cong$-interpretation models SetAx.

Since we have used the consistency of *SetAx* in proving properties that the above theorem depends on, we cannot substitute it for Theorem 4 for establishing that *SetAx* is consistent. However, we have not checked alternate ways of leading to the above theorem without using consistency.

The above theorem justifies the following definitions. A *Herbrand* $\cong$*-model* of a sentence φ is a Herbrand $\cong$-interpretation that models φ. Let φ and ψ be sentences. Then ψ is a *Herbrand* $\cong$*-logical consequence* of φ, denoted by $\varphi \models_{\mathcal{H}} \psi$, if ψ is true in all Herbrand $\cong$-models of φ.

8. CONCLUSIONS AND RELATED WORK

Sets are an important data object in both mathematics and computing. Mathematicians have paid considerable attention to sets through axiomatic approaches. Logical semantics also demands similar precision in discussing sets. Our treatment in this paper takes such a step by giving a rigorous support for the use of set constructors and finite sets in logic programming: We gave a set of axioms, collectively called *SetAx*, designed around the *scons* constructor. We distinguished between two kinds of set constructors, $scons(x, y)$ and $dscons(x, y)$, both of which have founded use in logic programs with sets. Our design of *SetAx* was influenced by the choice of *scons* as a primitive symbol of our theory rather than as a defined one, and by the need to deduce nonmembership relations between terms, to enable the use of *dscons*. The main results of this paper are: (i) we have shown that the set constructors indeed behave like finite sets; (ii) we have provided a framework for establishing the correctness of set unification; and (iii) we defined a Herbrand structure and provided a basis for discussing logical consequence semantics for logic programs with finite sets. This system was used to give a declarative semantics for the language SuRE [11], and can be used for other languages based upon these set constructors.

The axiom system $SetAx^-$ has essentially been described in [23], but only section 3 and the deduction of extensionality from *SetAx* of our paper overlaps with that work. A different axiomatic treatment of the set constructors is given in [5] for the {log} programming language. In that paper the constructor *scons* is referred to as `with`. Among the main differences are (i) we have an untyped system and make use of the predicates $set(x)$ and $indiv(x)$ to test for sets and individuals respectively; (ii) we make crucial use of the induction axiom in our development; (iii) we consider permutation and absorption to be axioms, but not extensionality; and (iv) we do not have the notion of the *kernel* of a term, which is used in [5]. On a technical level, the most important difference is the typed vs. the untyped approach. We do not use types because they are not needed to establish connections either with set theory or logic programming. As a result, $scons(s, t)$ denotes a set $\{s\} \cup t$ only if t denotes a set; otherwise, $scons(s, t)$ denotes an individual, not a set. In this sense, the set constructor *scons* is similar to the list constructor *cons* of untyped functional languages such as Lisp. On a methodological level, the axioms for the `with` constructor are to be taken on an intuitive basis; whether they capture all the properties of finite sets, i.e., whether their axioms are sufficient in addition to being plausible, has not been discussed. The main contribution of this paper is that we provide a rigorous justification that our axioms are *both* plausible and sufficient to capture the notion of finite sets.

Finally, we note that our work differs fundamentally in objectives from those of [1, 2, 20] in that we are interested in establishing logical foundations that will facilitate giving logical-consequence semantics, whereas the above works are primarily interested in model-theoretic semantics. Another approach which also differs fundamentally from our work is the embedding of sets within a constraint logic programming (CLP) framework (e.g., [6, 8, 21]). Here, the emphasis is on operational semantics (solving set constraints or developing consistency techniques), whereas the focus of our paper is more on logical, or declarative, foundations for set constructors.

ACKNOWLEDGMENTS

This research was supported by grant CCR 9613703 from the National Science Foundation.

REFERENCES

1. Abiteboul, S. and Grumbach, S.: A Rule-Based Language with Functions and Sets, *ACM Trans. on Database Systems*, vol. 16, no. 1, pp. 1–30, 1991.
2. Beeri, C., Naqvi, S., Shmueli, O., and Tsur, S.: Set Constructors in a Logic Database Language, *Jnl. Logic Programming*, vol. 10, pp. 181–232, 1991.
3. Boyer, R., Lusk, E., McCune, W., Overbeek, R., Stickel, M., and Wos, L.: Set Theory in First-Order Logic: Clauses for Goedel's Axioms, *Jnl. Automated Reasoning*, vol. 2, no. 3, pp. 287–327, 1986.
4. Dovier, A., Omodeo, E. G., Pontelli, E., and Rossi, G.: {log}: A Logic Programming Language with Finite Sets, *Proc. Eight Int. Conf. on Logic Programming*, Paris, Jun. 1991, pp. 111–124.
5. Dovier, A., Omodeo, E. G., Pontelli, E., and Rossi, G.: {log}: A Language for Programming in Logic with Finite Sets, *Jnl. of Logic Programming*, vol. 28, no. 1, pp. 1–44, 1996.
6. Dovier, A., Rossi, G.: Embedding Extensional Finite Sets in CLP, *Proc. Intl. Symp. on Logic Programming*, Vancouver, BC, Oct. 1993, pp. 540–554.
7. Fraenkel, A. A., Bar-Hillel, Y., Levy, A. and van Dalen, D.: *Foundations of Set Theory*, North-Holland, 1973.
8. Gervet, C.: Conjunto: Constraint Logic Programming with Finite Set Domains. *Proc. Intl. Symp. on Logic Programming*, Ithaca, NY, Oct. 1994, pp. 339–358.
9. Huet, G. P.: A Unification Algorithm for Typed λ-Calculus, *Theoretical Computer Science*, vol. 1. no. 1, pp. 27–57, 1975.
10. Jaffar, J. and Lassez, J-L.: Constraint Logic Programming, in: *Proc. 14th ACM Symp. on Principles of Programming Languages*, pp. 111–119, 1987.
11. Jana, D.: Semantics of Subset-Logic Languages, Ph.D. dissertation, Department of Computer Science, SUNY-Buffalo, August 1994.
12. Jana, D., and Jayaraman, B.: Set Constructors, Finite Sets, and Logical Semantics, Unpublished manuscript 1992, subsequently made available as TR 94-030, Department of Computer Science, SUNY-Buffalo, August 1994.
13. Jayaraman, B.: The SuRE Programming Framework, TR 91–11, SUNY at Buffalo, New York, Aug. 1991.
14. Jayaraman, B.: Implementaton of Subset-Equational Programming, *Jnl. of Logic Programming*, vol. 12, no. 4, pp. 299–324.
15. Jayaraman, B. and Nair A.: "Subset-Logic Programming: Application and Implementation," *Proc. JICSLP*, pp. 841-858, Seattle, 1988.
16. Jayaraman, B. and Plaisted, D. A.: "Functional Programming with Sets," *Proc. Third FPCA*, Portland, 1987, pp. 194-210, Springer-Verlag.
17. Jayaraman, B. and Plaisted, D. A.: Programming with Equations, Subsets, and Relations, *Proc. North American Conf. Logic Programming*, Cleveland, Oct. 1989, pp. 1051–1068.

18. Jouannaud, J.-P. and Kirchner, C.: Solving Equations in Abstract Algebras: A Rule-Based Survey of Unification, in: *Computational Logic: Essays in Honor of Alan Robinson*, MIT Press, 1991, pp. 257–321.

19. Kapur, D., and Narendran, P: A Unification Algorithm for ACI Theories (Preliminary Report), SUNY Albany Technical Report, April 1993.

20. Kuper, G. M.: Logic Programming with Sets, *Jnl. of Computer and System Sciences*, vol. 41, no. 1, pp. 44–64, 1990.

21. Legeard, B. and Legros, E.: Short Overview of the CLPS System, *Proc. Prog. Language Implementation and Logic Programming (PLILP)*, pp. 431–433, 1991.

22. Lloyd, J. W.: *Foundations of Logic Programming*, Springer-Verlag, 1987.

23. Manna, Z. and Waldinger, R.: *The Logical Basis for Computer Programming: vol. 1: Deductive Reasoning*, Addison-Wesley, 1985.

24. Martin, U., and Nipkow, T.: Unification in Boolean Rings, *Proc. 8th Conf. on Automated Deduction*, pp. 506–513, Springer LNCS 230, 1986.

25. Nadathur, G. and Miller, D.: *Higher-Order Horn Clauses*, *JACM*, vol. 31, pp. 777-814, 1989.

26. Naqvi, S. and Tsur, S.: *A Logical Language for Data and Knowledge Bases*, Computer Science Press, New York, 1989.

27. Siekmann, J.: Unification Theory, *Journal of Symbolic Computation*, vol. 7, no. 1, pp. 207–274, 1989.

28. Stolzenburg, F.: An Algorithm for General Set Unification and its Complexity, *Journal of Automated Reasoning*, 1995.

29. Suppes, P.: *Axiomatic Set Theory*, 1960, Dover edn., New York, 1972.