UPMAIL Technical Report No. 115
September 20, 1995
revised March 22, 1996
ISSN 1100-0686

Semantics of Constraint Logic Programs with Bounded Quantifiers

Martin Argenius
Andrei Voronkov*

Computing Science Department
Uppsala University
Box 311, S-751 05 Uppsala,
Sweden

email {d90mar,voronkov}@csd.uu.se

*Supported by a TFR grant

Abstract

This paper describes an extension of constraint logic programming by *bounded quantifiers.* Bounded quantifiers provide the support for finite domain constraint programming in a natural way. We define several semantics for constraint logic programs with bounded quantifiers and prove their equivalence. Our results can be used to define semantics of some existing constraint logic programming languages, like `cc(FD)`.[2]

Section 1

Introduction

Change the stile of citations!

Constraint logic programming is a declarative extension of Horn clause logic programming. In [23, 24] the second author defined another declarative extension of Horn clause logic programming by *bounded quantifiers.* Bounded quantifiers allow one to naturally express the search over finite domains. In this paper we show how to include bounded quantifiers into the constraint logic programming paradigm. It allows one to handle both finite and infinite domain constraints. In addition, bounded quantifiers make it possible to construct finite domains with the desired properties.

Intuitively, bounded quantifiers are quantifiers ranging over finite domains, in particular over finite lists or sets. The domains are structured so that bounded quantifiers reflect the structure of a domain, or ways of iterating elements of a domain. Expressions containing bounded quantifiers give natural and elegant examples of executable specifications. Consider, for example, the specification of disjoint sets

$\texttt{disjoint}(S_1, S_2)$ iff $(\forall x_1 \in S_1)(\forall x_2 \in S_2) x_1 \neq x_2$

This specification implies an obvious way of checking whether sets S_1, S_2 are disjoint. Semantics (even procedural) of the `disjoint` relation described in this way is intuitively clear. The usual representation of this relation in Horn clause logic programming lacks in elegance:

```
disjoint([],S).
disjoint([A|As],S)   :-
   nonmember(A,S),
   disjoint(As,S).
```

```
nonmember(A,[]).
nonmember(A,[B|Bs])   :-
   A ≠ B,
   nonmember(A,Bs).
```

The bounded quantifiers version is also more straightforward and succinct.

Consider another specification related to constraint logic programming. Assume that the predicate `enterprise` defines a finite set (represented by a list) of employees of an enterprise, and we would like to check that every employee has a boss using the predicate `boss_of`. Then we can use the following specification:

```
good_enterprise(E)   :-
   enterprise(E),
   (∀X ∈ E)(∃B ∈ E)boss_of(X,B).
```

This specification using bounded quantifiers can be executed using BQ-resolution defined in [24]. The usual way of treating such a problem in constraint logic programming with finite domains is to introduce the (finite) domain E and restrict the variables X and B to this domain. Using the unbounded existencial quantifier, we can also include infinite domains. For example, if we do not require the boss of an employee to be in the same enterprise, we can change the specification to

$$\begin{array}{l} \texttt{good\_enterprise}(E) \quad \texttt{:-} \\ \quad \texttt{enterprise}(E)\texttt{,} \\ \quad (\forall X \in E)(\exists B)\texttt{boss\_of}(X,B)\texttt{.} \end{array}$$

where the domain for B is not restricted. Moreover, we can use such specifications for constructing finite domains if we provide a procedural interpretation for the specifications with bounded quantifiers. An example is the following program defining the predicate $\texttt{good\_after\_extension}(P,E)$ with the meaning that the enterprise E could be extended with a person P to satisfy the requirements:

$$\begin{array}{l} \texttt{good\_after\_extension}(P,E) \quad \texttt{:-} \\ \quad \texttt{enterprise}(E)\texttt{,} \\ \quad (\forall X \in \texttt{[}P\texttt{|}E\texttt{]})(\exists B \in \texttt{[}P\texttt{|}E\texttt{]})\texttt{boss\_of}(X,B)\texttt{.} \end{array}$$

We can even use the following clause

$$\begin{array}{l} \texttt{good\_enterprise}(E) \quad \texttt{:-} \\ \quad (\forall X \in E)(\exists B \in E)\texttt{boss\_of}(X,B)\texttt{.} \end{array}$$

with the purpose of constructing arbitrary lists of persons that form a good enterprise.

The paper [24] discusses the use of bounded quantifiers in logic programming. This paper extends the results of [24] to constraint logic programming. We only treat semantics and a procedural interpretation in this paper. The results on the expressive power and on negation from [24] could also be straightforwardly generalized to constraint logic programs.

In Section 2 we give the basic definitions of sorts, lists and CBQ-programs, i.e. constraint logic programs with bounded quantifiers. In Section 3 we introduce several semantics of logic programs with bounded quantifiers and prove their equivalence. Section 4 describes CBQ-resolution that serves as the procedural semantics of the language. In Section 5 we discuss related works.

Section 2

Constraint logic programs with bounded quantifiers

In this section we introduce the main notion of the paper — CBQ-programs that generalize constraint logic programs. CBQ-programs use a two-sorted language. In Section 2.1 we introduce sorts and terms. Bounded quantifiers and CBQ-programs are defined in Section 2.2.

2.1 Sorts and lists

The first-order language of this paper uses two sorts: the universal sort univ and the sort of lists list, list being a subsort of univ. We assume that $\mathcal{L}$ is a fixed language consisting of two *finite* sets F of function symbols (constants are considered as function symbols of arity 0) and P of predicate symbols with the associated arities such that F contains the constant `[]` and the binary function symbol `[|]`. Further, we assume that P contains a distinguished subset $\{\mathcal{B}_1, \ldots, \mathcal{B}_k\}$ of *built-in predicate symbols* and $\mathcal{B}_0$ is the binary predicate symbol $=$. The built-in predicate symbols will be used to form *constraints.* We also assume that the symbols $\in$ and $\sqsubseteq$ do not belong to $\mathcal{L}$. Notation $\mathcal{B}_i$ will be used for the built-in predicate symbols throughout this paper. We assume to have countable sets of variables V_{univ} and V_{list} of the sorts univ and list, respectively.

Definition 2.1 (Terms and sorts)
Sets $\mathcal{T}_{\mathsf{univ}}$ and $\mathcal{T}_{\mathsf{list}}$ of *terms* of the sorts univ and list respectively, are defined as follows:

1. Any element of $V_{\mathsf{univ}} \bigcup V_{\mathsf{list}}$ belongs to $\mathcal{T}_{\mathsf{univ}}$;
2. If $f \in F$ is an n-ary function symbol and $t_1, \ldots, t_n \in \mathcal{T}_{\mathsf{univ}}$, then $f(t_1, \ldots, t_n) \in \mathcal{T}_{\mathsf{univ}}$;
3. Any element of V_{list} belongs to $\mathcal{T}_{\mathsf{list}}$;
4. $[\,] \in \mathcal{T}_{\mathsf{list}}$;
5. If $s \in \mathcal{T}_{\mathsf{univ}}$ and $t \in \mathcal{T}_{\mathsf{list}}$ then $[s\,|\,t] \in \mathcal{T}_{\mathsf{list}}$.

Evidently, $\mathcal{T}_{\mathsf{list}} \subseteq \mathcal{T}_{\mathsf{univ}}$.

A term is *ground* iff it uses no variables. *Formulas* are defined as usual, with terms being elements of $\mathcal{T}_{\mathsf{univ}}$. A formula is *closed* iff all occurrences of variables in it are bound. The notions of a *ground instance* of a term and a *closed instance* of a formula are standard.

We shall adopt the standard Prolog notation for elements of $\mathcal{T}_{\mathsf{list}}$:

$[s_1,\ldots,s_n\,|\,t]$ stands for $[s_1\,|\ldots[s_n\,|\,t]\ldots]$
$[s_1,\ldots,s_n]$ stands for $[s_1\,|\ldots[s_n\,|\,[]]\ldots]$

We shall denote variables by x, y, z, u, v, w, terms by r, s, t, predicate symbols by P, Q, R, formulas by φ, χ, ψ and atomic formulas by A, B, C.

Let φ be a formula. We write $\varphi(x)$ to denote all free occurrences of the variable x in φ. Then $\varphi(t)$ denotes the result of replacing the occurrences of x by t. The same notation will be used for the occurrences of several variables, i.e. $\varphi(\bar{x})$ and $\varphi(\bar{t})$ and for occurrences of variable(s) into a term, like $t(\bar{s})$. The set of free variables of a formula φ is denoted by $var(\varphi)$.

2.2 Bounded quantifiers and CBQ-programs

Here we introduce constraint logic programs with bounded quantifiers. They use Σ-formulas defined below:

Definition 2.2 (Σ-formulas and bounded quantifiers)
Σ-formulas in the language $\mathcal{L}$ are defined by the following grammar, where **A** and **S** range over atomic formulas in the language $\mathcal{L}$ and Σ-formulas in the language $\mathcal{L}$, respectively:

$$\mathbf{S} ::= \mathbf{A} \mid \mathbf{S} \wedge \mathbf{S} \mid \mathbf{S} \vee \mathbf{S} \mid (\forall x \in t)\mathbf{S} \mid (\exists x \in t)\mathbf{S} \mid (\forall x \sqsubseteq t)\mathbf{S} \mid (\exists x \sqsubseteq t)\mathbf{S} \mid (\exists x)\mathbf{S}$$

where t is a term of the sort list and x is a variable of the sort univ not occurring in t. Expressions of the form $(\forall x \in t)$, $(\exists x \in t)$, $(\forall x \sqsubseteq t)$ or $(\exists x \sqsubseteq t)$ are called *bounded quantifiers.*

Note that $x \in y$ is not a Σ-formula of the language $\mathcal{L}$, since atomic Σ-formulas can only use predicate symbols of the language $\mathcal{L}$.

Definition 2.3 (CBQ-program)
A *CBQ-program* $\mathcal{P}$ is a finite set of formulas of the form $\varphi_i \supset P_i(x_1,\ldots,x_{n_i})$, denoted

$$P_i(x_1,\ldots,x_{n_i}) \text{ :- } \varphi_i, \tag{2.1}$$

where $i \in \{1,\ldots,m\}$, P_i are pairwise different predicate symbols of $\mathcal{L}$ different from the built-in predicates and φ_i are Σ-formulas in the language $\mathcal{L}$ all whose free variables are in the set $\{x_1,\ldots,x_{n_i}\}$. Formula (2.1) is called the *predicate definition* of P_i in $\mathcal{P}$. $P_i(x_1,\ldots,x_{n_i})$ is the *head* and φ_i is the *body* of this predicate definition.

A *query* to a CBQ-program $\mathcal{P}$ is any Σ-formula.

To make CBQ-programs shorter and to make the syntax closer to that of ordinary logic programs we introduce some notation. Instead of the predicate definition

$$P(\bar{x}) \text{ :- } \exists\bar{y}_1(\bar{x} = \bar{t}_1 \wedge \varphi_1) \vee \ldots \vee \exists\bar{y}_n(\bar{x} = \bar{t}_n \wedge \varphi_n)$$

where $\bar{y}_i$ are all the variables of t_i, and all variables in $\bar{x}$ are different from variables in $\bar{y}_i$, we shall write the set of formulas $\{\varphi_i \supset P(\bar{t}_i) \mid i \in \{1,\ldots,n\}\}$, denoted

$$P(\bar{t}_1) \text{ :- } \varphi_1$$
$$\vdots$$
$$P(\bar{t}_n) \text{ :- } \varphi_n$$

This notation is similar to the translation used in the completion of logic programs [6]. We call such formulas *clauses* and often alternatively consider a CBQ-program as a finite set of clauses. Following the Prolog syntax, we shall sometimes terminate clauses or predicate definitions with a period.

A formal semantics of bounded quantifiers will be introduced in Section 3. Informally, they can be explained as follows. $x \in t$ means that x is an element of the list t. $x \sqsubseteq t$ means that x is a list obtained by removing zero or more elements from the beginning of the list t. For example, $(\forall x \in \texttt{[}1,2\texttt{]})\varphi(x)$ is equivalent to $\varphi(\texttt{[}1,2\texttt{]}) \wedge \varphi(\texttt{[}2\texttt{]}) \wedge \varphi(\texttt{[]})$.

In order to distinguish variables from constants, we make the following convention about the syntax. Variable names either are denoted by letters $x, y, z, u, v, w, \_$, maybe with indices, or start with upper-case letters written in the typewriter font, like `X` or `John`.

We also need notation to distinguish sorts of variables in programs. By default we assume that a variable v is of the sort list iff it occurs in one of the following expressions:

1. $(Qx \in v)$;
2. $\texttt{[}t_1, \ldots, t_n \,|\, v\texttt{]}$;
3. $(Qx \sqsubseteq v)$.

where Q is $\forall$ or $\exists$.

We have already given several examples of CBQ-programs in Section 1. In order to show the importance of the bounded quantifier $(\forall x \sqsubseteq t)$ we give another example here. The following nonrecursive CBQ-program describes ordered lists:

```
ordered(L) :-
   (∀S ⊑ L)(S = [] ∨ singleton(S) ∨ ordered_first_two(S)).

singleton([_]).

ordered_first_two([X,Y|_]) :-
   X ≤ Y.
```

In a similar way, one can describe the transitive closure of any binary relation.

2.3 The order-sorted predicate calculus

Check here and below what happens to $\preceq$

Since we use a two-sorted language, we should have a corresponding proof theory. There is an additional complication caused by the fact that a term may have two sorts. We introduce the relation $\preceq$ among terms in the following way: $s \preceq t$ iff every sort of s is also a sort of t. For our two-sorted case $s \preceq t$ is equivalent to the statement that if t has the sort list, then s has the sort list.

The order-sorted predicate calculus we use differs from the ordinary non-sorted predicate calculus in the following restrictions on the axioms for the quantifiers. The axioms $\forall x \varphi(x) \supset \varphi(t)$ and $\varphi(t) \supset \exists x \varphi(x)$ can only be applied when $t \preceq x$. All predicates and function symbols except for `[]` and `[|]` have the universal sort.

Section 3

Semantics of CBQ-programs

The theory of logic programming is based on the fundamental fact that the model-theoretic semantics coincides with the provability by SLD-resolution and some other kinds of semantics, for example the fixpoint semantics or provability in intuitionistic and classical logics. In this section we extend some of these semantics to CBQ-programs.

In Section 3.1 we introduce a model-theoretic semantics for CBQ-programs. The semantics is based on a special kind of Herbrand models — those in which built-in predicates have a predefined interpretation and lists are generated using `[]` and `[|]`. In Section 3.2 we introduce a fixpoint semantics that serves as a bridge between the declarative model-theoretic semantics and the procedural interpretation introduced later. The fixpoint semantics is based on an adaption of Herbrand models to bounded quantifiers — so-called Σ-saturated sets.

In Section 3.3 we define the proof theory of CBQ-programs. Section 3.4 introduces the so-called natural semantics for CBQ-programs. The natural semantics is based on the natural calculus for proving Σ-formulas. In fact, the procedural semantics (CBQ-resolution of Section 4) can be obtained by adding constraints to a top-down proof procedure for the natural calculus.

3.1 Model-theoretic semantics

The model-theoretic semantics of CBQ-programs is a modification of the model-theoretic semantics for logic programs with a special treatment of lists and built-in predicates. In order to use lists correctly we have to restrict the class of intended models to *CBQ-models* defined below.

For technical reasons we introduce two new languages. The language $\mathcal{L}^+$ is obtained from the language $\mathcal{L}$ by adding two binary predicate symbols $\in$ and $\sqsubseteq$. The language $\mathcal{L}^-$ is obtained from $\mathcal{L}$ by removing all the built-in predicate symbols. Note that the three languages have the same function symbols and differ only in their predicate symbols. *From now on we assume that all bodies of predicate definitions of CBQ-programs and queries to CBQ-programs are Σ-formulas in the language $\mathcal{L}$.*

Since bounded quantifiers do not belong to the standard syntax of first-order logic, we also consider them as standard abbreviations. For example $(\forall x \in t)\varphi$ is an abbreviation for $\forall x(x \in t \supset \varphi)$ and $(\exists x \sqsubseteq t)\varphi$ is an abbreviation for $\exists x(x \sqsubseteq t \wedge \varphi)$. Then any formula in the language $\mathcal{L}$ is also a Σ-formula in the language $\mathcal{L}$, and any Σ-formula in the language $\mathcal{L}$ is a formula in the language $\mathcal{L}^+$.

Since built-in predicates should have a pre-defined interpretation, we assume that there are some fixed relations $\mathcal{R}_i$ on the set of all ground terms of the same arity as the predicates $\mathcal{B}_i$. We

1. Axioms for the built-in predicates: all axioms of the form
$$\mathcal{B}_i(t_1, \ldots, t_{n_i})$$
such that $\langle t_1, \ldots, t_{n_i} \rangle \in \mathcal{R}_i$.

2. Axioms for lists:
$$\begin{array}{l} [s_1 \,|\, t_1] = [s_2 \,|\, t_2] \supset s_1 = s_2 \wedge t_1 = t_2 \\ \neg r \in [] \\ r \in [s \,|\, t] \equiv (r = s \vee r \in t) \\ r \sqsubseteq [] \equiv r = [] \\ r \sqsubseteq [s \,|\, t] \equiv (r = [s \,|\, t] \vee r \sqsubseteq t) \end{array}$$
where r, s, s_i are arbitrary terms of the sort univ, t, t_i are arbitrary terms of the sort list.

3. Induction axioms:
$$\varphi([\,]) \wedge \forall x \forall y (\varphi(y) \supset \varphi([x \,|\, y])) \supset \forall y \varphi(y)$$
where φ is any formula, x is a variable of the sort univ, y is a variable of the sort list.

Figure 3.1: The list theory **LT**

also assume that the interpretation $\mathcal{R}_0$ of the equality symbol $=$ is the identity relation on the set of ground terms.

Definition 3.1 (Theory LT and CBQ-models)
The *theory of lists* **LT** is a first-order theory in the language $\mathcal{L}^+$ with axioms shown in Figure 3.1.

Let $\mathfrak{M}$ be a two-sorted model with sorts univ and list in the language $\mathcal{L}^+$. The model $\mathfrak{M}$ is a *CBQ-model* iff the following hold:

1. The interpretation of the sort list is a subset of the interpretation of the sort univ.

2. All formulas of **LT** are true in $\mathfrak{M}$.

Since the notion of a CBQ-model depends on the interpretations $\mathcal{R}_1, \ldots, \mathcal{R}_k$ of the built-in predicates, it would be more correct to call them CBQ($\mathcal{R}_1, \ldots, \mathcal{R}_k$)-models. Instead, we fix $\mathcal{R}_1, \ldots, \mathcal{R}_k$ for the whole paper. The same is related to the theory **LT** since it depends on $\mathcal{R}_1, \ldots, \mathcal{R}_k$. Instead of the (maybe infinite) number of axioms for built-in predicates we could have used any theory that is sound and complete for closed atomic formulas $\mathcal{B}_i(t_1, \ldots, t_{n_i})$ w.r.t. $\mathcal{R}_i$.

Definition 3.2 (Model of CBQ-program)
A CBQ-model $\mathfrak{M}$ is a *model of the program* $\mathcal{P}$ iff all predicate definitions in $\mathcal{P}$ are true in $\mathfrak{M}$.

We recall that all CBQ-programs are considered to be in a fixed language $\mathcal{L}$. The main notion of this section is that of a *Herbrand model.*

Definition 3.3 (Herbrand universe and Herbrand model)
The *Herbrand universe HU* is the set of all closed atomic formulas in the language $\mathcal{L}^-$. A CBQ-model $\mathfrak{M}$ is a *Herbrand model* iff

1. The carrier set of $\mathfrak{M}$ is the set of all ground terms of the language $\mathcal{L}$;
2. The interpretation of the sort list is the set of all ground terms of the sort list;
3. The interpretation f_μ of any function symbol f is defined by $f_\mu(t_1,\ldots,t_n) = f(t_1,\ldots,t_n)$;
4. The interpretation of any built-in predicate $\mathcal{B}_i$ is the relation $\mathcal{R}_i$.

Note that expressions like $[a\,|\,b]$ where b is a constant different from $[]$ are not considered as ground terms because $[]$ is the only constant of the sort list. Also note that Herbrand models *by definition* are CBQ-models, i.e. they satisfy all axioms of **LT**.

In fact, induction axioms are trivially satisfied on any Herbrand model. The following theorem shows that the truth of axioms for lists on Herbrand models means that relations $\in$ and $\sqsubseteq$ have a certain unique interpretation.

Theorem 3.1 (Herbrand models and relations $\in, \sqsubseteq$)
Let $\mathfrak{M}$ be a Herbrand model. Then

1. Any ground term of the sort list has the form $[]$ or $[s_1,\ldots,s_n]$ for some ground terms $s_1,\ldots,s_n$;
2. $\mathfrak{M} \not\models t \in []$;
3. $\mathfrak{M} \models t \in [s_1,\ldots,s_n]$ iff t is one of $s_1,\ldots,s_n$;
4. $\mathfrak{M} \models t \sqsubseteq []$ iff t is $[]$;
5. $\mathfrak{M} \models t \sqsubseteq [s_1,\ldots,s_n]$ iff t has the form $[s_i,\ldots,s_n]$ for some $1 \le i \le n+1$.

Proof. Consider, for instance, property (3) (other properties are similar). By using axioms for lists, we can prove $\mathfrak{M} \models t \in [s_1,\ldots,s_n] \equiv t = s_1 \vee \ldots \vee t = s_n$. Using properties of equality on $\mathfrak{M}$ we conclude that $\mathfrak{M} \models t \in [s_1,\ldots,s_n]$ iff t is one of $s_1,\ldots,s_n$. □

We introduce a relation $\subseteq$ between Herbrand models of a program:

Definition 3.4 (Relation $\subseteq$)
Relation $\subseteq$ between Herbrand models is defined in the following way: $\mathfrak{M}_1 \subseteq \mathfrak{M}_2$ iff for any closed Σ-formula φ, if $\mathfrak{M}_1 \models \varphi$ then $\mathfrak{M}_2 \models \varphi$.

Lemma 3.1 Let $\mathfrak{M}_1, \mathfrak{M}_2$ be Herbrand models of a program $\mathcal{P}$. Then $\mathfrak{M}_1 \subseteq \mathfrak{M}_2$ iff for any closed atomic formula φ of the language $\mathcal{L}^-$, $\mathfrak{M}_1 \models \varphi$ implies $\mathfrak{M}_2 \models \varphi$.

Proof. The "only if" direction is obvious, because each atomic formula is a Σ-formula.

The "if" direction is by induction on the structure of the formula φ. We only consider one case, when φ has the form $(\forall x \in t)\psi(x)$. Since t is a ground term of the sort list, t has the form $[s_1,\ldots,s_n]$ for some ground terms $s_1,\ldots,s_n$. By Theorem 3.1, for any Herbrand model $\mathfrak{M}$, $\mathfrak{M} \models \varphi$ iff $\mathfrak{M} \models \psi(s_1) \wedge \ldots \wedge \psi(s_n)$. By the induction hypothesis, $\mathfrak{M}_1 \models \psi(s_i)$ implies $\mathfrak{M}_2 \models \psi(s_i)$. Hence, $\mathfrak{M}_1 \models \varphi$ implies $\mathfrak{M}_2 \models \varphi$. □

Lemma 3.2 Any Herbrand model is uniquely defined by the set of closed atomic formulas of the language $\mathcal{L}^-$ true in it. More precisely, if $\mathfrak{M}_1, \mathfrak{M}_2$ are two Herbrand models such that for every closed atomic formula A of the language $\mathcal{L}^-$ we have $\mathfrak{M}_1 \models A$ iff $\mathfrak{M}_2 \models A$, then $\mathfrak{M}_1$ is isomorphic to $\mathfrak{M}_2$.

Proof. Obviously, the identity mapping on the set of ground terms is an isomorphism between $\mathfrak{M}_1$ and $\mathfrak{M}_2$. □

CBQ-programs do not have the model intersection property for Σ-formulas. Indeed, it is straightforward to construct two Herbrand models, such that the formula $A \vee B$ is true in both, but neither A nor B are. However, we can prove that the model intersection property holds for atomic formulas.

Definition 3.5 (Intersection of Herbrand models)
Let $\mathfrak{M}_i,\ i \in I$ be Herbrand models, where I is any set. Their *intersection* $\bigcap_{i \in I} \mathfrak{M}_i$ is the model $\mathfrak{M}$ such that for any closed atomic formula φ of the language $\mathcal{L}^-$, $\mathfrak{M} \models \varphi$ iff for all $i \in I$, $\mathfrak{M}_i \models \varphi$.

Note that by Lemma 3.2, the intersection of Herbrand models is uniquely defined.

Lemma 3.3 Let $\mathfrak{M}_i,\ i \in I$ be Herbrand models and φ be a closed Σ-formula such that $\bigcap_{i \in I} \mathfrak{M}_i \models \varphi$. Then $\mathfrak{M}_i \models \varphi$, for all $i \in I$.

Proof. We use induction on the structure of φ. Denote $\bigcap_{i \in I} \mathfrak{M}_i$ by $\mathfrak{M}$. For atomic φ it is straightforward from definitions. Among other cases we consider only one. Let φ have the form $(\exists x \in [s_1, \ldots, s_n])\psi(x)$ and $\mathfrak{M} \models \varphi$. By Theorem 3.1 $\mathfrak{M} \models \psi(s_j)$, for some $j \in \{1, \ldots, n\}$. The induction hypothesis yields $\mathfrak{M}_i \models \psi(s_j)$, for all $i \in I$. By Theorem 3.1 $\mathfrak{M}_i \models \varphi$, for all $i \in I$. □

Lemma 3.4 Let $\mathfrak{M}_i,\ i \in I$ be Herbrand models of a CBQ-program $\mathcal{P}$. Then $\bigcap_{i \in I} \mathfrak{M}_i$ is also a Herbrand model of $\mathcal{P}$.

Proof. Denote $\bigcap_{i \in I} \mathfrak{M}_i$ by $\mathfrak{M}$. Let $P(\bar{x})$ `:-` $\varphi(\bar{x})$ be a predicate definition in $\mathcal{P}$. We have to show $\mathfrak{M} \models \varphi(\bar{t}) \supset P(\bar{t})$, for every tuple $\bar{t}$ of ground terms. Suppose $\mathfrak{M} \models \varphi(\bar{t})$. By Lemma 3.3, $\mathfrak{M}_i \models \varphi(\bar{t})$, for all $i \in I$. Since all $\mathfrak{M}_i$ are models of $\mathcal{P}$, we have $\mathfrak{M}_i \models P(\bar{t})$, for all $i \in I$. Since $\mathfrak{M}$ is the intersection of all $\mathfrak{M}_i$, $\mathfrak{M} \models P(\bar{t})$. Then $\mathfrak{M} \models \varphi(\bar{x}) \supset P(\bar{x})$. □

Theorem 3.2 (Existence of the least Herbrand model)
For every CBQ-program $\mathcal{P}$ there exists a Herbrand model $\mathfrak{M}$ of $\mathcal{P}$ which is the least w.r.t. $\subseteq$ among all Herbrand models.

Proof. By Lemma 3.4 the intersection of all Herbrand models of $\mathcal{P}$ is a model of $\mathcal{P}$. Obviously this model is minimal.

□

We shall call this model *the least Herbrand model* of $\mathcal{P}$.

Definition 3.6 ($\mathbf{Th}_\Sigma(\mathcal{P})$)
Given a CBQ-program $\mathcal{P}$, $\mathbf{Th}_\Sigma(\mathcal{P})$ denotes the set of all closed Σ-formulas which are true in the least Herbrand model of $\mathcal{P}$.

3.2 Fixpoint semantics

The definition of the fixpoint semantics is quite standard with some special care of Σ-formulas. A program is considered a monotonic mapping on the Σ-saturated sets that generalize Herbrand interpretations to include Σ-formulas:

Definition 3.7 (Σ-saturated sets)
A Σ-*saturated set* is any set $\mathfrak{I}$ of ground Σ-formulas in the language $\mathcal{L}$ satisfying the following properties:

1. $\mathcal{B}_i(s_1, \ldots, s_{n_i}) \in \mathfrak{I}$ iff $\langle s_1, \ldots, s_{n_i} \rangle \in \mathcal{R}_i$;
2. $\varphi \wedge \psi \in \mathfrak{I}$ iff $\varphi \in \mathfrak{I}$ and $\psi \in \mathfrak{I}$;
3. $\varphi \vee \psi \in \mathfrak{I}$ iff $\varphi \in \mathfrak{I}$ or $\psi \in \mathfrak{I}$;
4. $\exists x \varphi(x) \in \mathfrak{I}$ iff for some term t such that $t \preceq x$, $\varphi(t) \in \mathfrak{I}$;
5. $(\exists x \in [s_1, \ldots, s_n])\varphi(x) \in \mathfrak{I}$ iff for some $i \in \{1, \ldots, n\}$ we have $\varphi(s_i) \in \mathfrak{I}$;
6. $(\forall x \in [s_1, \ldots, s_n])\varphi(x) \in \mathfrak{I}$ iff for all $i \in \{1, \ldots, n\}$, we have $\varphi(s_i) \in \mathfrak{I}$;
7. $(\exists x \sqsubseteq [s_1, \ldots, s_n])\varphi(x) \in \mathfrak{I}$ iff for some $i \in \{1, \ldots, n+1\}$ we have $\varphi([s_i, \ldots, s_n]) \in \mathfrak{I}$;
8. $(\forall x \sqsubseteq [s_1, \ldots, s_n])\varphi(x) \in \mathfrak{I}$ iff for all $i \in \{1, \ldots, n+1\}$ we have $\varphi([s_i, \ldots, s_n]) \in \mathfrak{I}$.

Definition 3.8 ($\mathrm{Atom}(\mathfrak{I})$**)**
For a Σ-saturated set $\mathfrak{I}$, we denote by $\mathrm{Atom}(\mathfrak{I})$ the subset of $\mathfrak{I}$ consisting of all atomic formulas of the language $\mathcal{L}^-$.

It is easy to see that for any two Σ-saturated sets $\mathfrak{I}, \mathfrak{J}$, $\mathfrak{I} \subseteq \mathfrak{J}$ iff $\mathrm{Atom}(\mathfrak{I}) \subseteq \mathrm{Atom}(\mathfrak{J})$. Hence, any Σ-saturated set $\mathfrak{I}$ is uniquely characterized by the set $\mathrm{Atom}(\mathfrak{I})$. Thus, we can define

Definition 3.9 (Σ-closure $\mathrm{Cl}_\Sigma(A)$**)**
Let A be a set of closed atomic formulas of the language $\mathcal{L}^-$. Then the Σ-*closure* $\mathrm{Cl}_\Sigma(A)$ of A is the Σ-saturated set S such that $\mathrm{Atom}(S) = A$.

Definition 3.10 ($\mathcal{P}^*$**)**
Let $\mathcal{P}$ be a program. We use $\mathcal{P}^*$ to denote the set of all closed instances of predicate definitions in $\mathcal{P}$.

Strictly speaking, $\mathcal{P}^*$ is not a CBQ-program but a set of formulas.

Definition 3.11 (Immediate consequence operator $\mathbf{I}_\mathcal{P}$)
The *immediate consequence operator* defined by a program $\mathcal{P}$ is the function $\mathbf{I}_\mathcal{P}$ on the set of all Σ-saturated sets defined as follows:

$$\mathbf{I}_\mathcal{P}(\mathfrak{I}) = \mathrm{Cl}_\Sigma(\{\psi \mid \text{ there is a formula } \varphi \in \mathfrak{I} \text{ such that } (\varphi \supset \psi) \in \mathcal{P}^*\})$$

Lemma 3.5 (Monotonicity of $\mathbf{I}_\mathcal{P}$)
The operator $\mathbf{I}_\mathcal{P}$ is monotonic, i.e. for any two Σ-saturated sets $\mathfrak{I}$, $\mathfrak{J}$, if $\mathfrak{I} \subseteq \mathfrak{J}$ then $\mathbf{I}_\mathcal{P}(\mathfrak{I}) \subseteq \mathbf{I}_\mathcal{P}(\mathfrak{J})$.

Proof. Straightforward. □

Before proving the continuity of the immediate consequence operator, we shall introduce a definition, which is interesting by itself.

Definition 3.12 (Witness)
Let φ be a closed Σ-formula. A set W of closed atomic formulas in the language $\mathcal{L}^-$ is called a *witness* for φ iff for every Σ-saturated set S, if $W \subseteq S$, then $\varphi \in S$.

Lemma 3.6 (Finite witness lemma)
Let S be a Σ-saturated set and $\varphi \in S$. Then there is a finite witness $W \subseteq S$ for φ.

Proof. By induction on the structure of φ. We show a finite witness $W(\varphi)$ for each φ, leaving the (trivial) proof for the reader. Consider all possible cases:

1. If φ is atomic, then $W(\varphi) = \begin{cases} \emptyset, & \text{if } \varphi \text{ has the form } \mathcal{B}_i(t_1, \ldots, t_{n_i}) \\ \{\varphi\}, & \text{otherwise} \end{cases}$
2. $W(\varphi_1 \wedge \varphi_2) = W(\varphi_1) \cup W(\varphi_2)$.
3. $W(\varphi_1 \vee \varphi_2) = \begin{cases} W(\varphi_1), & \text{if } \varphi_1 \in S \\ W(\varphi_2), & \text{if } \varphi_2 \in S \end{cases}$
4. Let φ have the form $\exists x \psi(x)$. Then there is a term $t \preceq x$ such that $\psi(t) \in S$. Define $W(\varphi) = W(\psi(t))$.
5. Let φ have the form $(\exists x \in [s_1, \ldots, s_n])\psi(x)$. Then there is $i \in \{1, \ldots, n\}$ such that $\psi(s_i) \in S$. Define $W(\varphi) = W(\psi(s_i))$ for any such i.
6. Let φ have the form $(\forall x \in [s_1, \ldots, s_n])\psi(x)$. Then $W(\varphi) = W(\psi(s_1)) \cup \ldots \cup W(\psi(s_n))$.

The remaining two cases are similar. □

Lemma 3.7 (ω-continuity of $\mathbf{I}_{\mathcal{P}}$)
The operator $\mathbf{I}_{\mathcal{P}}$ is ω-continuous, i.e. for every increasing sequence $\mathfrak{I}_0 \subseteq \mathfrak{I}_1 \subseteq \ldots$ of Σ-saturated sets we have

$$\mathbf{I}_{\mathcal{P}}\Big(\bigcup_{n=0}^{\omega} \mathfrak{I}_n\Big) = \bigcup_{n=0}^{\omega} \mathbf{I}_{\mathcal{P}}(\mathfrak{I}_n)$$

Proof. It is enough to prove $\mathbf{I}_{\mathcal{P}}(\bigcup_{n=0}^{\omega} \mathfrak{I}_n) \subseteq \bigcup_{n=0}^{\omega} \mathbf{I}_{\mathcal{P}}(\mathfrak{I}_n)$ and $\mathbf{I}_{\mathcal{P}}(\bigcup_{n=0}^{\omega} \mathfrak{I}_n) \supseteq \bigcup_{n=0}^{\omega} \mathbf{I}_{\mathcal{P}}(\mathfrak{I}_n)$

($\subseteq$) Let $\varphi \in \mathbf{I}_{\mathcal{P}}(\bigcup_{n=0}^{\omega} \mathfrak{I}_n)$. By Lemma 3.6 there is a finite witness $\{\psi_1, \ldots, \psi_m\} \subseteq \mathbf{I}_{\mathcal{P}}(\bigcup_{n=0}^{\omega} \mathfrak{I}_n)$ for φ. By the definition of $\mathbf{I}_{\mathcal{P}}$, there are $(\psi_i \texttt{ :- } \varphi_i) \in \mathcal{P}^*$, such that $\varphi_i \in \bigcup_{n=0}^{\omega} \mathfrak{I}_n$, for all $i \in \{1, \ldots, m\}$. Since $\mathfrak{I}_n$ are increasing, there is k such that $\varphi_i \in \mathfrak{I}_k$, for all $i \in \{1, \ldots, m\}$. Thus, $\psi_i \in \mathbf{I}_{\mathcal{P}}(\mathfrak{I}_{k+1})$, for all $i \in \{1, \ldots, m\}$. Since $\{\psi_1, \ldots, \psi_m\}$ is a witness for φ, we have $\varphi \in \mathbf{I}_{\mathcal{P}}(\mathfrak{I}_{k+1})$ and hence $\varphi \in \bigcup_{n=0}^{\omega} \mathbf{I}_{\mathcal{P}}(\mathfrak{I}_n)$.

($\supseteq$) Immediate by the monotonicity of $\mathbf{I}_{\mathcal{P}}$.

□

Theorem 3.3 (Existence of the least fixpoint)
There is the least fixed point $\mathbf{Lfp}(\mathcal{P})$ of the operator $\mathbf{I}_{\mathcal{P}}$ among all Σ-saturated sets. It can be computed as

$$\mathbf{Lfp}(\mathcal{P}) = \bigcup_{n=0}^{\omega} \mathfrak{I}_i,$$

where $\mathfrak{I}_0 = \mathrm{Cl}_{\Sigma}(\emptyset)$, and $\mathfrak{I}_{i+1} = \mathbf{I}_{\mathcal{P}}(\mathfrak{I}_i)$. Moreover $\mathbf{Lfp}(\mathcal{P})$ coincides with $\mathbf{Th}_{\Sigma}(\mathcal{P})$.

Proof. Standard, using the continuity of $\mathbf{I}_{\mathcal{P}}$. □

3.3 The provability semantics

Classical proof systems for lists are obtained from the first order classical predicate calculus by adding axioms expressing properties of lists and built-in functions.

Definition 3.13 (Calculus $\mathbf{Clt}(\mathcal{P})$)
The calculus $\mathbf{Clt}(\mathcal{P})$ (the classical list theory) is obtained from the classical predicate calculus with equality by adding the axioms of **LT** plus all predicate definitions of $\mathcal{P}$.

The calculus $\mathbf{Clt}(\mathcal{P})$ can be used as the proof theory for logic programming with bounded quantifiers. Using this calculus one can prove properties of lists and programs. The following theorem shows the soundness and completeness of $\mathbf{Clt}(\mathcal{P})$.

Theorem 3.4 (Soundness and completeness of $\mathbf{Clt}(\mathcal{P})$)
A closed Σ-formula φ is provable in $\mathbf{Clt}(\mathcal{P})$ iff $\varphi \in \mathbf{Th}_\Sigma(\mathcal{P})$.

Proof. The "only if" direction is straightforward: it is easy to see that every formula provable in $\mathbf{Clt}(\mathcal{P})$ is true in any CBQ-model of $\mathcal{P}$.

Let us prove the "if" direction. Using axioms for lists and the induction axiom of $\mathbf{Clt}(\mathcal{P})$, one can prove the following properties of $\mathbf{Clt}(\mathcal{P})$ (compare with Definition 3.7):

1. For all ground terms $t_1, \ldots, t_{n_i}$ if $\langle t_1, \ldots, t_{n_i} \rangle \in \mathcal{R}_i$ then $\mathbf{Clt}(\mathcal{P}) \vdash \mathcal{B}_i(t_1, \ldots, t_{n_i})$;
2. $\mathbf{Clt}(\mathcal{P}) \vdash \varphi \wedge \psi$, if $\mathbf{Clt}(\mathcal{P}) \vdash \varphi$ and $\mathbf{Clt}(\mathcal{P}) \vdash \psi$;
3. $\mathbf{Clt}(\mathcal{P}) \vdash \varphi \vee \psi$, if $\mathbf{Clt}(\mathcal{P}) \vdash \varphi$ or $\mathbf{Clt}(\mathcal{P}) \vdash \psi$;
4. $\mathbf{Clt}(\mathcal{P}) \vdash \exists x \varphi(x)$, if for some term $t \preceq x$, $\mathbf{Clt}(\mathcal{P}) \vdash \varphi(t)$;
5. $\mathbf{Clt}(\mathcal{P}) \vdash (\exists x \in [s_1, \ldots, s_n])\varphi(x)$, if $\mathbf{Clt}(\mathcal{P}) \vdash \varphi(s_i)$ for some $i \in \{1, \ldots, n\}$;
6. $\mathbf{Clt}(\mathcal{P}) \vdash (\forall x \in [s_1, \ldots, s_n])\varphi(x)$, if $\mathbf{Clt}(\mathcal{P}) \vdash \varphi(s_i)$ for every $i \in \{1, \ldots, n\}$;
7. $\mathbf{Clt}(\mathcal{P}) \vdash (\exists x \sqsubseteq [s_1, \ldots, s_n])\varphi(x)$, if $\mathbf{Clt}(\mathcal{P}) \vdash \varphi([s_i, \ldots, s_n])$ for some $i \in \{1, \ldots, n+1\}$;
8. $\mathbf{Clt}(\mathcal{P}) \vdash (\forall x \sqsubseteq [s_1, \ldots, s_n])\varphi(x)$, if $\mathbf{Clt}(\mathcal{P}) \vdash \varphi([s_i, \ldots, s_n])$ for every $i \in \{1, \ldots, n+1\}$.

Using this property we can prove that for any Σ-saturated set $\mathfrak{I}$, if for all $\varphi \in \mathrm{Atom}(\mathfrak{I})$ $\mathbf{Clt}(\mathcal{P}) \vdash \varphi$, then for all $\varphi \in \mathfrak{I}$ $\mathbf{Clt}(\mathcal{P}) \vdash \varphi$.

Consider now the characterization of the least Herbrand model of $\mathcal{P}$ proved in Theorem 3.3: a closed Σ-formula φ is true in the least Herbrand model iff $\varphi \in \mathfrak{I}_i$, for some i. Thus it suffices to show that all formulas from $\mathfrak{I}_i$ are provable in $\mathbf{Clt}(\mathcal{P})$, for all i. In view of the preceding paragraph, it is enough to prove that all formulas of $\mathrm{Atom}(\mathfrak{I}_i)$ are provable in $\mathbf{Clt}(\mathcal{P})$. The proof is by induction on i:

$i = 0$ Obvious, since $\mathrm{Atom}(\mathfrak{I}_i) = \emptyset$.

$i > 0$ Let $\varphi \in \mathrm{Atom}(\mathfrak{I}_i)$. Then there is a formula $(\psi \supset \varphi) \in \mathcal{P}^*$ such that $\psi \in \mathfrak{I}_{i-1}$. The induction hypothesis yields $\mathbf{Clt}(\mathcal{P}) \vdash \psi$. Since all formulas in $\mathcal{P}^*$ are provable in $\mathbf{Clt}(\mathcal{P})$, we have $\mathbf{Clt}(\mathcal{P}) \vdash \varphi$.

□

The provability semantics here is based on classical logic. We can also define the sound and complete constructive provability semantics for CBQ-programs based on intuitionistic logic in the way similar to [24].

3.4 The natural semantics

The natural semantics of this section is based on the natural calculus. The rules of the natural calculus treat Σ-formulas in a simple and elegant way. The calculus $\mathbf{Nat}(\mathcal{P})$ introduced below is similar to the ground positive hyperresolution on Horn clauses. At the same time the natural calculus serves as a basis for the procedural semantics of CBQ-programs.

A reason for introducing the natural calculus as an intermediate semantics between the model theoretic semantics and the procedural semantics (CBQ-resolution of Section 4), is the fact that the least fixpoint construction does not directly correspond to computations, as in the case of SLD-resolution. For the pure logic programming case (the only built-in predicate is equality $=$) we proved in [23, 24] that there are CBQ-programs, consisting of only one nonrecursive definition, which can express arbitrary computable sets of ground terms. For such programs the least fixpoint construction stabilizes on the first step ($\mathfrak{I}_1$), which does not correspond to real computations. The natural semantics does reflect computations by CBQ-resolution, which is the main procedure to execute CBQ-programs.

The natural calculus has something in common with the foundations of logic programming proposed in [13, 1, 14].

Definition 3.14 (Calculus $\mathbf{Nat}(\mathcal{P})$)
The provable objects of the calculus $\mathbf{Nat}(\mathcal{P})$ are closed Σ-formulas of the language $\mathcal{L}$. Its inference rules are shown in Figure 3.2.

Theorem 3.5 (Soundness and completeness of $\mathbf{Nat}(\mathcal{P})$)
For any closed Σ-formula φ, $\varphi \in \mathbf{Th}_\Sigma(\mathcal{P})$ iff $\mathbf{Nat}(\mathcal{P}) \vdash \varphi$.

Proof.

1. It is easy to see that the rules of $\mathbf{Nat}(\mathcal{P})$ are derivable in $\mathbf{Clt}(\mathcal{P})$. By Theorem 3.4, if $\mathbf{Nat}(\mathcal{P}) \vdash \varphi$ then $\varphi \in \mathbf{Th}_\Sigma(\mathcal{P})$.
2. Consider Σ-saturated sets $\mathfrak{I}_i$ from the proof of Theorem 3.3. Obviously, every atomic formula in $\mathfrak{I}_0$ is provable in $\mathbf{Nat}(\mathcal{P})$. Rules for the connectives, the existential quantifier and bounded quantifiers allow all Σ-formulas from a Σ-saturated set to be proved from atomic formulas of this Σ-saturated set and formulas $\mathcal{B}_i(t_1, \ldots, t_{n_i})$ such that $\langle t_1, \ldots, t_{n_i} \rangle \in \mathcal{R}_i$ (see Definition 3.7 of Σ-saturated sets). Finally, rules for $\mathcal{P}$ allow all atomic formulas of $\mathfrak{I}_{i+1}$ to be obtained from formulas of $\mathfrak{I}_i$. Application of Theorem 3.3 on the coincidence of $\mathbf{Lfp}(\mathcal{P})$ with $\mathbf{Th}_\Sigma(\mathcal{P})$ concludes the proof.

$\square$

1. Axioms are formulas of the form $\mathcal{B}_i(t_1, \ldots, t_{n_i})$ such that $t_1, \ldots, t_{n_i}$ are ground terms and $\langle t_1, \ldots, t_{n_i} \rangle \in \mathcal{R}_i$.
2. Rules for $\mathcal{P}$:
$$\frac{\varphi}{A}$$
if A `:-` φ belongs to $\mathcal{P}^*$.
3. Rules for the logical connectives:
$$\frac{\varphi}{\varphi \vee \psi} \qquad \frac{\psi}{\varphi \vee \psi} \qquad \frac{\varphi \quad \psi}{\varphi \wedge \psi}$$
4. A rule for the existential quantifier:
$$\frac{\varphi(t)}{\exists x \varphi(x)}$$
where t is a term such that $t \preceq x$.
5. Rules for the bounded quantifiers (here t is a term of the sort list, s is an arbitrary term):
$$\frac{\varphi(s)}{(\exists x \in \texttt{[}s\,|\,t\texttt{]})\varphi(x)} \qquad \frac{(\exists x \in t)\varphi}{(\exists x \in \texttt{[}s\,|\,t\texttt{]})\varphi}$$
$$\frac{}{(\forall x \in \texttt{[]})\varphi} \qquad \frac{\varphi(s) \quad (\forall x \in t)\varphi(x)}{(\forall x \in \texttt{[}s\,|\,t\texttt{]})\varphi(x)}$$
$$\frac{\varphi(t)}{(\exists x \sqsubseteq t)\varphi(x)} \qquad \frac{(\exists x \sqsubseteq t)\varphi}{(\exists x \sqsubseteq \texttt{[}s\,|\,t\texttt{]})\varphi}$$
$$\frac{\varphi(\texttt{[]})}{(\forall x \sqsubseteq \texttt{[]})\varphi(x)} \qquad \frac{\varphi(\texttt{[}s\,|\,t\texttt{]}) \quad (\forall x \sqsubseteq t)\varphi(x)}{(\forall x \sqsubseteq \texttt{[}s\,|\,t\texttt{]})\varphi(x)}$$

Figure 3.2: The natural calculus $\mathbf{Nat}(\mathcal{P})$

Section 4

Operational semantics

The operational semantics of CBQ-programs is based on the notion of a constraint. Constraints are defined as formulas of a special form using only built-in predicate symbols. The query answering mechanism is CBQ-resolution defined below. Instead of returning a substitution as in the case of logic programming, CBQ-resolution returns a satisfiable constraint that represents a condition on the query variables. Any substitution that makes the answer constraint true is a correct answer substitution. This semantics is *not* oriented to a particular collection of built-in relations $\mathcal{R}_i$, but rather to a general constraint logic programming scheme. The definitions and notations used here are similar to those in [8]. The difference comes out in bounded quantification.

Definition 4.1 (Constraint)
A *constraint* is a formula of the form $\exists x_1 \dots x_n(A_1 \wedge \dots \wedge A_m)$ where $n \geq 0, m \geq 0$ and A_i, for $i \in \{1, \dots, m\}$, are atoms $\mathcal{B}_i(t_1, \dots, t_{k_i})$. When $n, m = 0$ the constraint is denoted by *true*. A constraint $\mathcal{C}$, where $var(\mathcal{C}) = \bar{x}$, is satisfiable iff $\exists \bar{x} \mathcal{C}$ is true in all Herbrand models.

Since the truth of a formula only using built-in predicates does not depend on a Herbrand model, we shall call formulas true on all Herbrand models simply *true* formulas. **maybe valid?**

In the following the empty list of Σ-formulas is denoted by ε. When $\bar{s}$ and $\bar{t}$ are tuples of terms $s_1, \dots, s_n$ and $t_1, \dots, t_n$, respectively, then $\bar{s} = \bar{t}$ denotes $s_1 = t_1 \wedge \dots \wedge s_n = t_n$.

Definition 4.2 (Goal)
A *goal* $\mathcal{G}$ has the form $\mathcal{C} \,\Box\, \varphi_1, \dots, \varphi_n$, where $\mathcal{C}$ is a constraint and φ_i for $i \in \{1, \dots, n\}$ are Σ-formulas.

Intuitively, a goal represents the list of Σ-formulas to be checked on the least Herbrand model together with an additional constraint on its variables.

Definition 4.3 (Derivation step)
A *derivation* step for the goal $\mathcal{G}$ of the form $\exists \bar{x} \mathcal{C} \,\Box\, \varphi_1, \dots, \varphi_n$ in the program $\mathcal{P}$ results in a goal $\mathcal{G}'$ of the form $\mathcal{C}' \,\Box\, \varphi_1, \dots, \varphi_{i-1}, \Gamma, \varphi_{i+1}, \dots, \varphi_n$, and is denoted by $\mathcal{G} \overset{\mathcal{P}}{\rightsquigarrow} \mathcal{G}'$, where Γ is a list of Σ-formulas, the constraint $\mathcal{C}'$ is satisfiable and φ_i, Γ and $\mathcal{C}'$ are as in one of the rows in Table 4.1.

Definition 4.4 (CBQ-derivation)
A *CBQ-derivation* of a goal $\mathcal{G}$ in a program $\mathcal{P}$ is a finite or infinite sequence of goals, starting with $\mathcal{G}$, such that every goal is obtained from the previous by means of a derivation step in $\mathcal{P}$.

In this table x and $\bar{x}$ are used in different ways. Must be changed

	φ_i	Γ	$\mathcal{C}'$
1	$\mathcal{B}_j(t_1,\ldots,t_{n_j})$		$\exists\bar{x}(\mathcal{C}\wedge\mathcal{B}_j(t_1,\ldots,t_{n_j}))$
2a	$\psi_1\vee\psi_2$	ψ_1	$\exists\bar{x}\mathcal{C}$
2b	$\psi_1\vee\psi_2$	ψ_2	$\exists\bar{x}\mathcal{C}$
3	$\psi_1\wedge\psi_2$	ψ_1,ψ_2	$\exists\bar{x}\mathcal{C}$
4	$\exists x\psi(x)$	$\psi(u)$	$\exists\bar{x}\exists u\mathcal{C}$
5a	$(\exists x\in t)\psi(x)$	$\psi(z)$	$\exists\bar{x}\exists y\exists z(\mathcal{C}\wedge t=\texttt{[}z\,\texttt{\|}\,y\texttt{]})$
5b	$(\exists x\in t)\psi,$	$(\exists x\in y)\psi$	$\exists\bar{x}\exists y\exists z(\mathcal{C}\wedge t=\texttt{[}z\,\texttt{\|}\,y\texttt{]})$
6a	$(\forall x\in t)\psi$		$\exists\bar{x}(\mathcal{C}\wedge t=\texttt{[]})$
6b	$(\forall x\in t)\psi(x)$	$\psi(z),(\forall x\in y)\psi(x)$	$\exists\bar{x}\exists y\exists z(\mathcal{C}\wedge t=\texttt{[}z\,\texttt{\|}\,y\texttt{]})$
7a	$(\exists x\sqsubseteq t)\psi(x)$	$\psi(t)$	$\exists\bar{x}\mathcal{C}$
7b	$(\exists x\sqsubseteq t)\psi,$	$(\exists x\sqsubseteq y)\psi$	$\exists\bar{x}\exists y\exists z(\mathcal{C}\wedge t=\texttt{[}z\,\texttt{\|}\,y\texttt{]})$
8a	$(\forall x\sqsubseteq t)\psi(x)$	$\psi(\texttt{[]})$	$\exists\bar{x}(\mathcal{C}\wedge t=\texttt{[]})$
8b	$(\forall x\sqsubseteq t)\psi(x)$	$\psi(t),(\forall x\sqsubseteq y)\psi(x)$	$\exists\bar{x}\exists y\exists z(\mathcal{C}\wedge t=\texttt{[}z\,\texttt{\|}\,y\texttt{]})$
9	$P(\bar{t})$	$\psi(\bar{t})$	$\exists\bar{x}\mathcal{C}$

In this table in line 4 u is a variable not occurring in $\mathcal{G}$ or $\mathcal{C}$ of the same sort as x, in lines 5a–8b y is a variable not occurring in $\mathcal{G}$ or $\mathcal{C}$ of the sort list and z is a variable not occurring in $\mathcal{G}$ or $\mathcal{C}$ of the sort univ. In line 9 the predicate definition of P in $\mathcal{P}$ is $P(\bar{v})$ `:-` $\psi(\bar{v})$.

Table 4.1: Goal reduction for derivation steps

The length of a CBQ-derivation is the number of steps in the derivation.

Successful CBQ-derivations terminate at the empty set of formulas, returning *answer constraints:*

Definition 4.5 (Successful CBQ-derivation)
A *successful* CBQ-derivation of a goal $\mathcal{G}$ in $\mathcal{P}$ is a derivation of $\mathcal{G}$ in $\mathcal{P}$, whose last element is a goal of the form $(\mathcal{C}'\,\Box\,\varepsilon)$. The constraint $\mathcal{C}'$ is called the *answer constraint* of this derivation.

In the rest of this section $\overset{\mathcal{P}}{\leadsto}{}^*$ will denote the reflexive and transitive closure of $\overset{\mathcal{P}}{\leadsto}$.

Lemma 4.1 Let $\mathcal{C},\mathcal{C}'$ be quantifier-free constraints and there is a derivation $\exists\bar{x}\mathcal{C}\,\Box\,\varphi_1,\ldots,\varphi_n \overset{\mathcal{P}}{\leadsto}{}^* \exists\bar{y}\mathcal{C}'\,\Box\,\psi_1,\ldots,\psi_m$. Then $\mathfrak{M}\models\mathcal{C}'\wedge\psi_1\wedge\ldots\wedge\psi_m\supset\mathcal{C}\wedge\varphi_1\wedge\ldots\wedge\varphi_n$ for every Herbrand model $\mathfrak{M}$ of $\mathcal{P}$.

Proof. By induction on the length l of the derivation. Suppose $l=1$. According to the definition of a derivation step the goal $\exists\bar{x}\mathcal{C}\,\Box\,\varphi_1,\ldots,\varphi_n$ reduces to a goal $\exists\bar{y}\mathcal{C}_1\,\Box\,\varphi_1,\ldots,\varphi_{i-1},\Gamma,\varphi_{i+1},\ldots,\varphi_n$ where φ_i, Γ and $\mathcal{C}'=\exists\bar{y}\mathcal{C}_1$ are as in one of the rows of Table 4.1. Let $\mathfrak{M}$ be any Herbrand model of $\mathcal{P}$. Obviously $\mathfrak{M}\models\mathcal{C}'\supset\mathcal{C}$. Then it is enough to prove $\mathfrak{M}\models\mathcal{C}_1\wedge\bigwedge_{\psi\in\Gamma}\psi\supset\varphi_i$. Consider some possible cases where φ_i, Γ and $\mathcal{C}'$ correspond to one of the rows in Table 4.1. The other cases are similar.

1. Then $\mathcal{C}_1$ is $\mathcal{C}\wedge\mathcal{B}_j(t_1,\ldots,t_{n_j})$ and we have to prove $\mathfrak{M}\models\mathcal{C}\wedge\mathcal{B}_j(t_1,\ldots,t_{n_j})\supset\mathcal{B}_j(t_1,\ldots,t_{n_j})$, which is obvious.

4. Then $\mathcal{C}_1$ is $\mathcal{C}$ and we have to prove $\mathfrak{M}\models\mathcal{C}\wedge\psi(u)\supset\exists x\psi(x)$. This is straightforward.

6*b*. Then $\mathcal{C}_1$ is $\mathcal{C} \wedge t = \texttt{[}z\,|\,y\texttt{]}$ and we have to prove $\mathfrak{M} \models \mathcal{C} \wedge t = \texttt{[}z\,|\,y\texttt{]} \wedge \psi(z) \wedge (\forall x \in y)\psi(x) \supset (\forall x \in t)\psi(x)$. Straightforward.

8*a*. Then $\mathcal{C}_1$ is $\mathcal{C} \wedge t = \texttt{[]}$ and we have to prove $\mathfrak{M} \models \mathcal{C} \wedge t = \texttt{[]} \wedge \psi(\texttt{[]}) \supset (\forall x \sqsubseteq t)\psi(x)$. Straightforward.

9. Then $\mathcal{C}_1$ is $\mathcal{C}$ and we have to prove $\mathfrak{M} \models \mathcal{C} \wedge \varphi(\bar{t}) \supset P(\bar{t})$. This follows from $\mathfrak{M} \models \forall \bar{v}(\varphi(\bar{v}) \supset P(\bar{v}))$.

The induction step $l > 1$ is straightforward. □

Theorem 4.1 (Soundness)
Let $\mathfrak{M}$ be any Herbrand model of a program $\mathcal{P}$. If there exists a derivation $\mathit{true} \,\Box\, \varphi \overset{\mathcal{P}}{\leadsto}{}^{*} \mathcal{C} \,\Box\, \varepsilon$ then $\mathfrak{M} \models \mathcal{C} \supset \varphi$.

Proof. The constraint $\mathcal{C}$ has the form $\exists \bar{x}\mathcal{C}'$, for a quantifier-free $\mathcal{C}'$. Without loss of generality we can assume that the variables $\bar{x}$ do not occur in φ. By Lemma 4.1, $\mathfrak{M} \models \mathcal{C}' \supset \varphi$. Since the variables $\bar{x}$ do not occur in φ we have $\mathfrak{M} \models \exists \bar{x}\mathcal{C}' \supset \varphi$ which is the same as $\mathfrak{M} \models \mathcal{C} \supset \varphi$. □

We prove two technical lemmas about the composition of derivations:

Lemma 4.2 Suppose there are derivations $\mathit{true} \,\Box\, \varphi_1 \overset{\mathcal{P}}{\leadsto}{}^{*} \exists \bar{x}\mathcal{C}_1 \,\Box\, \varepsilon$ and $\mathit{true} \,\Box\, \varphi_2 \overset{\mathcal{P}}{\leadsto}{}^{*} \exists \bar{y}\mathcal{C}_2 \,\Box\, \varepsilon$, for quantifier-free constraints $\mathcal{C}_1$ and $\mathcal{C}_2$, the constraint $\mathcal{C}_1 \wedge \mathcal{C}_2$ is satisfiable and $\bar{x}$, $\bar{y}$ are tuples of pairwise different variables not occurring in φ_1 or φ_2. Then there is a derivation $\mathit{true} \,\Box\, \varphi_1, \varphi_2 \overset{\mathcal{P}}{\leadsto}{}^{*} \exists \bar{x}\bar{y}(\mathcal{C}_1 \wedge \mathcal{C}_2) \,\Box\, \varepsilon$.

Proof. Obvious. □

Lemma 4.3 Suppose there are derivations

$$\begin{array}{l} \exists \bar{x}\mathcal{C} \,\Box\, \varphi_1 \overset{\mathcal{P}}{\leadsto}{}^{*} \exists \bar{x}\bar{y}_1\mathcal{C}_1 \,\Box\, \varepsilon \\ \mathit{true} \,\Box\, \varphi_2 \overset{\mathcal{P}}{\leadsto}{}^{*} \exists \bar{y}_2\mathcal{C}_2 \,\Box\, \varepsilon \\ \vdots \\ \mathit{true} \,\Box\, \varphi_n \overset{\mathcal{P}}{\leadsto}{}^{*} \exists \bar{y}_n \,\Box\, \varepsilon \end{array}$$

for quantifier-free constraints $\mathcal{C}_1, \ldots, \mathcal{C}_n$, such that $\mathcal{C}_1 \wedge \ldots \wedge \mathcal{C}_n$ is satisfiable and $\bar{x}$, $\bar{y}_1, \ldots, \bar{y}_n$ are tuples of different variables not occurring in $\varphi_1, \ldots, \varphi_n$. Then there exists a derivation $\exists \bar{x}\mathcal{C} \,\Box\, \varphi_1, \ldots, \varphi_n \overset{\mathcal{P}}{\leadsto}{}^{*} \exists \bar{x}\bar{y}_1 \ldots \bar{y}_n(\mathcal{C}_1 \wedge \ldots \wedge \mathcal{C}_n) \,\Box\, \varepsilon$.

Proof. Similar to that of Lemma 4.2. □

The completeness theorem is a reformulation of the completeness theorem for SLD-resolution:

Theorem 4.2 (Completeness)
Let $\mathfrak{M}$ be the least Herbrand model of a program $\mathcal{P}$, $\varphi(\bar{x})$ is a Σ-formula all whose free variables are among $\bar{x}$, $\bar{s}$ are terms, $\bar{t}$ are ground terms, $\mathfrak{M} \models \varphi(\bar{t})$ and $\bar{s} = \bar{t}$ is satisfiable. Then there exists a derivation $\mathit{true} \,\Box\, \varphi(\bar{s}) \overset{\mathcal{P}}{\leadsto} \mathcal{C} \,\Box\, \varepsilon$ such that $\bar{s} = \bar{t} \supset \mathcal{C}$ is true.

Proof. **This proof should be compared literally with that of ELP'96.** Let us call the complexity of a formula the number of occurrences of connectives and quantifiers in it. By Theorem 3.3 $\mathfrak{M} \models \varphi(\bar{t})$ iff $\varphi(\bar{t})$ belongs to some $\mathfrak{I}_i$, where the Σ-saturated sets $\mathfrak{I}_i$ are defined by $\mathfrak{I}_0 = \mathrm{Cl}_\Sigma(\emptyset)$, and $\mathfrak{I}_{i+1} = \mathbf{I}_{\mathcal{P}}(\mathfrak{I}_i)$. Thus, it is enough to prove that there exists a derivation $\mathit{true} \,\Box\, \varphi(\bar{s}) \overset{\mathcal{P}}{\leadsto} \mathcal{C} \,\Box\, \varepsilon$ such that $\bar{s} = \bar{t} \supset \mathcal{C}$ is true whenever $\varphi(\bar{t}) \in \mathfrak{I}_i$, for some i. The proof is by induction on the ordinal $i\omega + j$ where i is the minimal number such that $\varphi(\bar{t}) \in \mathfrak{I}_i$ and j is the complexity of $\varphi(\bar{t})$.

1. Suppose $j = 0$. Then we have two different cases depending on the form of the formula $\varphi(\bar{x})$.

   (a) $\varphi(\bar{x})$ has the form $\mathcal{B}_j(r_1, \ldots, r_{n_j})$ and $\varphi(\bar{t}) \in \mathfrak{I}_i$. Then the required derivation is $\mathit{true} \,\square\, \varphi(\bar{s}) \overset{\mathcal{P}}{\leadsto} \varphi(\bar{s}) \,\square\, \varepsilon$. Obviously $\varphi(\bar{s})$ is satisfiable and $\bar{s} = \bar{t} \supset \varphi(\bar{s})$ is true.

   (b) $\varphi(\bar{x})$ is an atomic formula $P(r_1(\bar{x}), \ldots, r_n(\bar{x}))$, where P is not a built-in predicate and $\varphi(\bar{t}) \in \mathfrak{I}_i$. Then there is a clause $\psi(\bar{y}) \supset P(\bar{y}) \in \mathcal{P}$ such that $\psi(r_1(\bar{t}), \ldots, r_n(\bar{t})) \in \mathfrak{I}_{i-1}$. We introduce a formula $\psi'(\bar{x}) = \psi(r_1(\bar{x}), \ldots, r_n(\bar{x}))$, then $\psi'(\bar{t}) \in \mathfrak{I}_{i-1}$. By the induction hypothesis, there is a derivation $\mathit{true} \,\square\, \psi'(\bar{s}) \overset{\mathcal{P}}{\leadsto}{}^* \mathcal{C} \,\square\, \varepsilon$ where $\bar{s} = \bar{t} \supset \mathcal{C}$ is true. The required derivation of $\mathit{true} \,\square\, \varphi(\bar{s})$ is $\mathit{true} \,\square\, P(r_1(\bar{s}), \ldots, r_n(\bar{s})) \overset{\mathcal{P}}{\leadsto} \mathit{true} \,\square\, \psi'(\bar{s}) \overset{\mathcal{P}}{\leadsto}{}^* c \,\square\, \varepsilon$.

2. Suppose $j > 0$. Then $\varphi(\bar{x})$ is not an atomic formula. We consider some cases according to the outermost connective of $\varphi(\bar{x})$.

   (a) $\varphi(\bar{x})$ has the form $\psi_1(\bar{x}) \vee \psi_2(\bar{x})$ and $\varphi(\bar{t}) \in \mathfrak{I}_i$. Then either $\psi_1(\bar{t}) \in \mathfrak{I}_i$ or $\psi_2(\bar{t}) \in \mathfrak{I}_i$. Consider for example the case when $\psi_1(\bar{t}) \in \mathfrak{I}_i$. The complexity of $\psi_1(\bar{t})$ is smaller than the complexity of $\varphi(\bar{t})$ so there exists a derivation $\mathit{true} \,\square\, \psi_1(\bar{s}) \overset{\mathcal{P}}{\leadsto}{}^* \mathcal{C} \,\square\, \varepsilon$ such that $\bar{s} = \bar{t} \supset \mathcal{C}$ is true. The required derivation of $\mathit{true} \,\square\, \varphi(\bar{s})$ is $\mathit{true} \,\square\, \psi_1(\bar{s}) \vee \psi_2(\bar{s}) \overset{\mathcal{P}}{\leadsto} \mathit{true} \,\square\, \psi_1(\bar{s}) \overset{\mathcal{P}}{\leadsto}{}^* c \,\square\, \varepsilon$. In the same way when $\psi_2(\bar{s}) \in \mathfrak{I}_i$.

   (b) $\varphi(\bar{x})$ has the form $\psi_1(\bar{x}) \wedge \psi_2(\bar{x})$ and $\varphi(\bar{t}) \in \mathfrak{I}_i$. Then both $\psi_1(\bar{t}) \in \mathfrak{I}_i$ and $\psi_2(\bar{t}) \in \mathfrak{I}_i$. The complexities of $\psi_1(\bar{t})$ and $\psi_2(\bar{t})$ are smaller than that of $\varphi(\bar{t})$ so there exist derivations $\mathit{true} \,\square\, \psi_1(\bar{s}) \overset{\mathcal{P}}{\leadsto}{}^* \exists \bar{y} \mathcal{C}_1 \,\square\, \varepsilon$ and $\mathit{true} \,\square\, \psi_2(\bar{s}) \overset{\mathcal{P}}{\leadsto}{}^* \exists \bar{z} \mathcal{C}_2 \,\square\, \varepsilon$, where $\mathcal{C}_1$ and $\mathcal{C}_2$ are quantifier-free constraints such that $\bar{s} = \bar{t} \supset \exists \bar{y} \mathcal{C}_1$ and $\bar{s} = \bar{t} \supset \exists \bar{z} \mathcal{C}_2$ are true. Without loss of generality we can assume that the variables in $\bar{y}$ and $\bar{z}$ are pairwise different and do not occur in $\bar{s}$. Since $\bar{s} = \bar{t} \supset \exists \bar{y} \mathcal{C}_1$ and $\bar{s} = \bar{t} \supset \exists \bar{y} \mathcal{C}_2$ are true, $\bar{s} = \bar{t} \supset \exists \bar{y} \exists \bar{z} (\mathcal{C}_1 \wedge \mathcal{C}_2)$ is true. Satisfiability of $\bar{s} = \bar{t}$ implies that $\mathcal{C}_1 \wedge \mathcal{C}_2$ is satisfiable. By Lemma 4.2 there is a derivation $\mathit{true} \,\square\, \psi_1(\bar{s}), \psi_2(\bar{s}) \overset{\mathcal{P}}{\leadsto}{}^* \exists \bar{y} \bar{z} (\mathcal{C}_1 \wedge \mathcal{C}_2) \,\square\, \varepsilon$. The required derivation of $\mathit{true} \,\square\, \varphi(\bar{s})$ is $\mathit{true} \,\square\, \psi_1(\bar{s}) \wedge \psi_2(\bar{s}) \overset{\mathcal{P}}{\leadsto} \mathit{true} \,\square\, \psi_1(\bar{s}), \psi_2(\bar{s}) \overset{\mathcal{P}}{\leadsto}{}^* \exists \bar{y} \bar{z} (\mathcal{C}_1 \wedge \mathcal{C}_2) \,\square\, \varepsilon$.

   (c) $\varphi(\bar{x})$ has the form $(\forall u \in r(\bar{x})) \psi(\bar{x}, u)$. Then $(\forall u \in r(\bar{t})) \psi(\bar{t}, u) \in \mathfrak{I}_i$. The term $r(\bar{t})$ has the form $[r_1, \ldots, r_m]$ for some ground terms $r_1, \ldots, r_m$. Let $z_1, \ldots, z_m, y_1, \ldots, y_m$ be variables not occurring in $\bar{s}$. Since the complexities of the formulas $\psi(\bar{x}, z_j)$ are smaller than that of $\varphi(\bar{x})$ and $\psi(\bar{t}, r_j) \in \mathfrak{I}_i$, there are derivations $\mathit{true} \,\square\, \psi(\bar{s}, z_j) \overset{\mathcal{P}}{\leadsto}{}^* \exists \bar{v}_j \mathcal{C}_j \,\square\, \varepsilon$ where $\bar{s} = \bar{t} \wedge z_j = r_j \supset \exists \bar{v}_j \mathcal{C}_j$ is true, for all $j \in \{1, \ldots, m\}$. Without loss of generality we can assume that all variables in $\bar{v}_1, \ldots, \bar{v}_m, y_1, \ldots, y_m, z_1, \ldots, z_m$ are different and do not occur in $\bar{s}$. The truth of $\bar{s} = \bar{t} \wedge z_j = r_j \supset \exists \bar{v}_j \mathcal{C}_j$ implies the truth of $\bar{s} = \bar{t} \wedge z_1 = r_1 \wedge \ldots \wedge z_m = r_m \supset \exists \bar{v}_1 \ldots \bar{v}_m (\mathcal{C}_1 \wedge \ldots \wedge \mathcal{C}_m)$. Since the variables x_j, z_j do not occur in $\bar{s}$, the formula $\bar{s} = \bar{t} \supset \exists y_1 z_1 \ldots y_m z_m \bar{v}_1 \ldots \bar{v}_m (r(\bar{s}) = [z_1 \,|\, y_1] \wedge y_1 = [z_2 \,|\, y_2] \wedge \ldots \wedge y_{m-1} = [z_m \,|\, y_m] \wedge y_m = [] \wedge \mathcal{C}_1 \wedge \ldots \wedge \mathcal{C}_m)$ is true. Since $\bar{s} = \bar{t}$ is satisfiable, the formula $r(\bar{s}) = [z_1 \,|\, y_1] \wedge y_1 = [z_2 \,|\, y_2] \wedge \ldots \wedge y_{m-1} = [z_m \,|\, y_m] \wedge y_m = [] \wedge \mathcal{C}_1 \wedge \ldots \wedge \mathcal{C}_m$ is satisfiable. Then by Lemma 4.3 there is a derivation

$$
\begin{array}{l}
\exists y_1 z_1 \ldots y_m z_m (r(\bar{s}) = [z_1 \,|\, y_1] \wedge y_1 = [z_2 \,|\, y_2] \wedge \ldots \wedge y_{m-1} = [z_m \,|\, y_m] \wedge y_m = []) \\
\quad \square\, \psi(\bar{s}, z_1), \ldots, \psi(\bar{s}, z_m) \\
\overset{\mathcal{P}}{\leadsto}{}^* \exists y_1 z_1 \ldots y_m z_m \bar{v}_1 \ldots \bar{v}_m (r(\bar{s}) = [z_1 \,|\, y_1] \wedge y_1 = [z_2 \,|\, y_2] \wedge \ldots \wedge y_{m-1} = [z_m \,|\, y_m] \wedge \\
\qquad\qquad y_m = [] \wedge \mathcal{C}_1 \wedge \ldots \wedge \mathcal{C}_m) \,\square\, \varepsilon
\end{array}
$$

   Then the required derivation of $\mathit{true} \,\square\, \varphi(\bar{s})$ is

$$
\begin{array}{ll}
 & \mathit{true} \,\Box\, (\forall u \in r(\bar{s}))\psi(\bar{s}, u) \\
\overset{\mathcal{P}}{\rightsquigarrow} & \exists y_1 z_1 (r(\bar{s}) = [z_1 \,|\, y_1]) \,\Box\, \psi(\bar{s}, z_1), (\forall u \in y_1)\psi(\bar{s}, u) \\
 & \vdots \\
\overset{\mathcal{P}}{\rightsquigarrow} & \exists y_1 z_1 \ldots y_m z_m (r(\bar{s}) = [z_1 \,|\, y_1] \wedge y_1 = [z_2 \,|\, y_2] \wedge \ldots \wedge y_{m-1} = [z_m \,|\, y_m]) \\
 & \qquad \Box\, \psi(\bar{s}, z_1), \ldots, \psi(\bar{s}, z_m), (\forall u \in y_m)\psi(\bar{s}, u) \\
\overset{\mathcal{P}}{\rightsquigarrow} & \exists y_1 z_1 \ldots y_m z_m (r(\bar{s}) = [z_1 \,|\, y_1] \wedge y_1 = [z_2 \,|\, y_2] \wedge \ldots \wedge y_{m-1} = [z_m \,|\, y_m] \wedge y_m = []) \\
 & \qquad \Box\, \psi(\bar{s}, z_1), \ldots, \psi(\bar{s}, z_m) \\
\overset{\mathcal{P}}{\rightsquigarrow}{}^{*} & \exists y_1 z_1 \ldots y_m z_m \bar{v}_1 \ldots \bar{v}_m (r(\bar{s}) = [z_1 \,|\, y_1] \wedge y_1 = [z_2 \,|\, y_2] \wedge \ldots \wedge y_{m-1} = [z_m \,|\, y_m] \wedge \\
 & \qquad\qquad y_m = [] \wedge \mathcal{C}_1 \wedge \ldots \wedge \mathcal{C}_m) \,\Box\, \varepsilon
\end{array}
$$

□

Note that the statement of Theorem 4.2 implies that the formula $\bar{s} = \bar{t} \wedge \mathcal{C}$ is satisfiable.

Section 5

Related work

In this section we shall briefly survey related work.

Bounded quantifiers in logic programming were considered in [10, 9, 11, 17, 18, 7, 3, 22, 23, 2, 24]. In [10, 11, 9] the concept of Σ-programs have been defined using Σ-formulas. The model-theoretic and the fixpoint semantics of our paper are very similar to those of [10, 11, 9]. In [17, 18] it is shown how to integrate (flat) finite sets in logic programming with the idea of database applications. Sets and other *collection types* were also considered in the database-related papers [4, 5, 19, 12, 25, 20]. Those papers study the expressive power and practical applications of languages with collection types. Bounded quantification over hereditarily finite sets is introduced in [7]. In that paper sets are added to logic programming using a special set unification algorithm. In [3, 2] a limited version of bounded quantification (lists of a fixed length only) has been introduced to Prolog and Gödel. The motivation was to provide a support for arrays, and in particular for their parallel processing. General constraints over finite sets are considered in [16]. **Add Apt.**

The semantics of constraint logic programs were described in many papers (see the recent survey [15]). Our presentation is very similar to that of [8], except for the handling of bounded quantifiers and a slightly different treatment of bound variables in constraints. The operational semantics for finite domains that is similar to our semantics is introduced in the form of *cardinality operators* of [22, 21].

Our paper is different from the above mentioned papers in the systematic treatment of logic programming both with constraints and bounded quantifiers. As a consequence, we obtain a uniform semantics for handling constraints both over finite and infinite domains. With some modifications, we can apply our semantics to characterize existing constraint logic programming languages, like `cc(FD)` of [22, 21]. For example, the cardinality operators may be described as bounded quantifiers like $\exists^{\geq n}\varphi$, although for their correct treatment one needs to consider negation which is outside of the scope of this paper.

Our presentation was not bound to any particular domain. In some particular domains there may be specific helpful kinds of bounded quantifiers. For example, for integers we could include a bounded quantification over integer intervals $[k..n]$ consisting of numbers from k to n:

$$A(\bar{x}) \ \texttt{:-}\ (\forall m \in [k..n])\varphi(\bar{x}, m).$$

Results of this paper can be extended to such integer intervals as well, with the difference that integers are considered as a predefined sort.

Other applications of bounded quantifiers in logic programming are considered in [24].

Bibliography

[1] J.H. Andrews. Proof-theoretic characterizations of logic programming. In *Mathematical Foundations of Computer Science*, number 379 in Lecture Notes in Computer Science, pages 145–154, Porąbka-Kozubnik, Poland, 1989.

[2] J. Barklund and P.M. Hill. Extending Gödel for expressing restricted quantification and arrays. UPMAIL Technical Report 102, Uppsala University, Computing Science Department, March 1995.

[3] J. Barklund and H. Millroth. Providing iteration and concurrency in logic programs through bounded quantifications. In *Proc. of the International Conference on Fifth Generation Computer Systems*, pages 817–824, Tokyo, 1992. ICOT.

[4] C. Beeri, Sh. Naqvi, R. Ramakrishnan, O. Shmueli, and Sh. Tsur. Sets and negation in a logic database language (LDL1). In *Proc. 6th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 21–36. ACM Press, 1987.

[5] C. Beeri, S. Naqvi, O. Schmueli, and S. Tsur. Set constructors in a logic database language. *Journal of Logic Programming*, 10:181–232, 1991.

[6] K.L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Data Base*, pages 293–322. Plenum Press, New York, 1978.

[7] A. Dovier, E.G. Omodeo, E. Pontelli, and G. Rossi. {log}: A logic programming language with finite sets. In *Proc. ICLP'91*, pages 109–124. MIT Press, 1991.

[8] M. Gabbrielli, G.M. Dore, and G. Levi. Observable semantics for constraint logic programs. *Journal of Logic and Computation*, 5(2):133–171, 1995.

[9] S.S. Goncharov, Yu.L. Ershov, and D.I. Sviridenko. Semantic programming. In *IFIP'86*, pages 1093–1100. Elsevier Science, 1986.

[10] S.S. Goncharov and D.I. Sviridenko. Σ-programming (in Russian). volume 120 of *Vychislitelnye Systemy*, pages 3–29. Novosibirsk, 1985.

[11] S.S. Goncharov and D.I. Sviridenko. Theoretical aspects of Σ-programming. In *Mathematical Methods of Specification and Synthesis of Software Systems'85*, volume 215 of *Lecture Notes in Computer Science*, pages 169–179, 1986.

[12] S. Grumbach and T. Milo. Towards tractable algebras for bags. In *ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 49–58, 1993.

[13] M. Hagiya and T. Sakurai. Foundation of logic programming based on inductive definition. *New Generation Computing*, 2(1):59–77, 1984.

[14] L. Hallnäs and P. Schroeder-Heister. A proof-theoretic approach to logic programming I. Clauses as rules. *Journal of Logic and Computation*, 1(2), 1990.

[15] J. Jaffar and M. Maher. Constraint logic programming: a survey. *Journal of Logic Programming*, 19,20:503–581, 1994.

[16] D. Kozen. Rational spaces and set constraints. In P.D. Mosses, M. Nielsen, and M.I. Schwartzbach, editors, *TAPSOFT'95: Theory and Practice of Software Development*, volume 915 of *Lecture Notes in Computer Science*, pages 42–61. Springer Verlag, 1995.

[17] G.M. Kuper. Logic programming with sets. In *Proc. 6th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 11–20. ACM Press, 1987.

[18] G.M. Kuper. Logic programming with sets. *Journal of Computer and System Sciences*, 41:44–64, 1990.

[19] O. Schmueli, S. Tsur, and C. Zaniolo. Compilation of set terms in the logic data language (LDL). *Journal of Logic Programming*, 12:89–119, 1992.

[20] K. Vadaparty. On the power of rule-based languages for nested data models. *Journal of Logic Programming*, 21:155–175, 1994.

[21] P. Van Hentenryck, W. Saraswat, and Y. Deville. Design, implementation and evaluation of the constraint language `cc(fd)`. In A. Podelski, editor, *Constraint Programming: Basics and Tools*, volume 910 of *Lecture Notes in Computer Science*, pages 293–316. Springer Verlag, 1995.

[22] P. Van Hentenryck, H. Simonis, and M. Dincbas. Constraint satisfaction using constraint logic programming. *Artificial Intelligence*, 58:113–159, 1992.

[23] A. Voronkov. Logic programming with bounded quantifiers. In A. Voronkov, editor, *Logic Programming*, volume 592 of *Lecture Notes in Artificial Intelligence*, pages 486–514. Springer Verlag, 1992.

[24] A. Voronkov. Logic programming with bounded quantifiers revisited. UPMAIL Technical Report 114, Uppsala University, Computing Science Department, September 1995.

[25] L. Wong. Normal forms and conservative properties for query languages over collection types. In *ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 26–36, 1993.