LILOG Report 53, October 1988, IBM Deutschland, Stuttgart, Germany

Definite Relations over Constraint Languages

Markus Höhfeld[†] *and Gert Smolka*[‡]

† *FB Informatik, Universität Kaiserslautern, West Germany*

‡ *WT LILOG, IBM Deutschland, West Germany*

Abstract. This paper shows that the nice properties of logic programs extend to definite clause specifications over arbitrary constraint languages. The notion of a constraint language sees a constraint as a piece of syntax with unknown internal structure that constrains the values variables can take in interpretations. Examples of constraint languages are Predicate Logic and its sublanguages as well as attributive concept description languages developed for knowledge representation.

Our framework generalizes the constraint logic programming scheme of Jaffar and Lassez to make it applicable to knowledge representation: the constraint language is not required to be a sublanguage of predicate logic and may come with more than one interpretation, and the interpretations of the constraint language are not required to be solution compact.

We present a semantic type discipline for our generalized definite clause specifications and establish a notion of well-typedness that is decidable provided the underlying constraint language is decidable. Finally, we give a type inference rule for computing most general well-typed weakenings of specifications.

The research reported here has been funded by the EUREKA Project Protos (EU 56).

Address for correspondence: Gert Smolka, WT LILOG, IBM Deutschland, Postfach 800880, 7000 Stuttgart 80, West Germany, smolka@ds0lilog.bitnet.

To appear in Journal of Logic Programming

1 Introduction

In the last few years a new model of logic programming has emerged that views a logic programming language as consisting of a constraint language on top of which relations can be defined by means of definite clauses. Different logic programming languages can be obtained by employing different constraint languages. Conventional logic programming is obtained by employing equations that are interpreted in the algebra of first-order terms. Prolog II [Colmerauer et al. 83, Colmerauer 84] employs as constraint language equations and disequations that are interpreted in the algebra of rational trees. The constraint language of Prolog III [Colmerauer 88] is interpreted in an algebra providing rational trees and rational numbers and allows for linear equations and inequations for numbers, boolean expressions for truth values, and equations and disequations for general terms. Other recent examples of constraint logic programming languages are CLP($\mathcal{R}$) [Jaffar/Michaylov 87], CIL [Mukai 87] and CHIP [Dincbas et al. 88].

Jaffar and Lassez [86, 87] were the first to identify the new model, coined the name Constraint Logic Programming, and developed a general framework that is parameterized with respect to the constraint language being employed and yields soundness and completeness results for a generic operational semantics relying on a constraint solver for the employed constraint language. A constraint solver is an algorithm deciding the satisfiability of constraint systems. In conventional logic programming, the constraint solver solves equations in the Herbrand universe, which is accomplished by term unification.

The original motivation for the research reported in this paper was the development of a semantic foundation for the knowledge representation language LOGIN [Aït-Kaci/Nasr 86], where relations are defined with definite clauses over a constraint language consisting of so-called ψ-terms [Aït-Kaci 86].

The first step of this enterprise was to come up with a logical reformulation of Aït-Kaci's ψ-term calculus and led to the development of Feature Logic [Smolka 88], a decidable logic that generalizes Aït-Kaci's formalism by adding negation and quantification. Feature Logic makes explicit that Aït-

Kaci's ψ-terms, the feature descriptions developed by computational linguists [Kaplan/Bresnan 82, Rounds/Kasper 86, Johnson 87], and the knowledge representation language KL-ONE [Brachman/Schmolze 85, Levesque/Brachman 87] are all closely related members of the same family of logics. These logics offer attributive concept descriptions that are interpreted as sets and are built from sorts and binary relations (called attributes, roles or features). Given an attributive concept description C, a constraint $x\colon C$ constrains the values of the variable x to elements of C.

Ideally, the second step of giving a semantic foundation to LOGIN should have consisted in simply applying Jaffar and Lassez's [86, 87] constraint logic programming scheme (CLP, for short) to Feature Logic. However, this failed for three reasons:

1. CLP requires that the constraint language is interpreted in a single fixed domain. This is in accordance with the data structure paradigm underlying current programming languages, which views programs as computing with data structures that are, in most applications, merely representations of the objects one is actually interested in. For knowledge representation, however, data structures as representations of real objects are not adequate. Instead, one talks directly about the objects of interest, as this is accomplished, for instance, by the Tarski semantics of Predicate Logic. Since, in general, we have only partial information about the world we want to reason about, we need to take into account all worlds that are consistent with our partial knowledge. Thus we have to generalize CLP such that the constraint language can come with more than one interpretation and a constraint is considered satisfiable if there is at least one interpretation in which it has a solution.

2. CLP requires that the interpretations of constraint languages be "solution compact", which implies that every element of an interpretation must be obtainable as the unique solution of a possibly infinite set of constraints. While solution compactness is sensible for "data structure" interpretations, it is not acceptable for "real world" interpretations. CLP needs solution compactness since it provides soundness and completeness results for negation as failure. However, since the constraint language

can provide for logical negation (for instance, disequations in Prolog II or set complements in Feature Logic) we feel that there is no further need for the problematic negation as failure.

3. CLP assumes that the constraint language is expressed in Predicate Logic: constraints must be formulas of Predicate Logic and interpretations must be interpretations of Predicate Logic. However, neither Feature Logic, KL-ONE, nor the order-sorted predicate logic underlying Eqlog [Goguen/Meseguer 86] satisfy these assumptions. Although these formalisms can be reduced to Predicate Logic in principle, providing customized model theories and notations for them is crucial in keeping them technically simple and in supporting the adequate intuitions. So what CLP is lacking is a sufficiently abstract formalization of the notion of a constraint language.

In this paper we present a framework that generalizes CLP so that the shortcomings discussed above are completely avoided.

We start with a definition of constraint languages that is general enough to cover all mentioned formalisms. In our analysis, a constraint is a piece of syntax constraining the values the variables occurring in it can take. There is no need to know anything about the internal structure of a constraint. Since we are not concerned with negation as failure, we don't need to impose any requirements on the interpretations of constraint languages. A prominent example of a constraint language is Predicate Logic, where the formulas serve as constraints.

Next we show that every constraint language can be extended conservatively to a constraint language providing for relational atoms, the propositional connectives, and quantification. By taking equations with their Tarski interpretations as constraint language, this construction yields Predicate Logic. We show that, for every set $\mathcal{S}$ of definite clauses in the extension of an arbitrary constraint language $\mathcal{L}$, every interpretation of $\mathcal{L}$ can extended to a minimal model of $\mathcal{S}$. This generalizes the key result of conventional logic programming to our framework, which is not restricted to Horn theories.

We then present an operational semantics for our general definite clause specifications that generalizes the SLD-resolution method [Lloyd 84] employed in conventional logic programming and prove its soundness and completeness.

In the second part of the paper, we present a semantic type discipline for our generalized definite clause specifications. The discipline exploits the idea that declarations of relation symbols in a sorted language can be expressed as implications; for instance, to declare that the relation *plus* takes integers as arguments, we can write the implication

$$plus(x, y, z) \ \rightarrow\ x{:}\,int \ \&\ y{:}\,int \ \&\ z{:}\,int.$$

If Feature Logic is used as underlying constraint language, we can constrain the arguments of a relation with complex feature terms employing intersections, complements and feature constraints. The idea even applies to conventional logic programming, where we can write declarations like

$$p(x, y) \ \rightarrow\ \exists\, z.\ y = f(x, z).$$

We establish a weak notion of well-typedness by saying that a definite clause specification $\mathcal{S}$ is implicitly well-typed with respect to a set $\mathcal{D}$ of declarations if every minimal model of $\mathcal{S}$ is a model of $\mathcal{D}$. Next we establish a strong notion of well-typedness by defining explicitly well-typed clauses and show that explicitly well-typed specifications are implicitly well-typed. Explicit well-typedness is decidable provided the underlying constraint language is decidable. Furthermore, we show that our operational semantics is type safe, that is, the reduction of an explicitly well-typed goal with an explicitly well-typed clause yields again an explicitly well-typed goal. Finally, we give a type inference rule that can be used to compute a most general explicitly well-typed weakening of a specification. We show that, if the explicitly well-typed specification $\mathcal{S}'$ is obtained from $\mathcal{S}$ by type inference, $\mathcal{S}'$ and $\mathcal{S}$ have the same minimal models provided $\mathcal{S}$ is implicitly well-typed.

Acknowledgement. Our definition of constraint languages was influenced by a draft of Manfred Schmidt-Schauß outlining an axiomatic foundation of unification theory.

2 Constraint Languages

We start with a very general definition of constraint languages. The basic idea is that a constraint is some piece of syntax constraining the values of the variables occurring in it. Our definition does not make any assumptions about the syntax of constraints.

A **constraint language** is a tuple $(\mathrm{VAR}, \mathrm{CON}, \mathcal{V}, \mathrm{INT})$ such that

1. VAR is a decidable, infinite set whose elements are called **variables**
2. CON is a decidable set whose elements are called **constraints**
3. $\mathcal{V}$ is a computable function that assigns to every constraint ϕ a finite set $\mathcal{V}\phi$ of variables, called the **variables constrained by** ϕ
4. INT is a nonempty set of **interpretations**, where every interpretation $\mathcal{I} \in \mathrm{INT}$ consists of a nonempty set $D^{\mathcal{I}}$, called the **domain of** $\mathcal{I}$, and a **solution mapping** $[\![\cdot]\!]^{\mathcal{I}}$ such that:

   4.1 an **$\mathcal{I}$-assignment** is a mapping $\mathrm{VAR} \to D^{\mathcal{I}}$, and $\mathrm{ASS}^{\mathcal{I}}$ is the set of all $\mathcal{I}$-assignments

   4.2 $[\![\cdot]\!]^{\mathcal{I}}$ is a function mapping every constraint ϕ to a set $[\![\phi]\!]^{\mathcal{I}}$ of $\mathcal{I}$-assignments, where the $\mathcal{I}$-assignments in $[\![\phi]\!]^{\mathcal{I}}$ are called the **solutions of ϕ in $\mathcal{I}$**

   4.3 a constraint ϕ constrains only the variables in $\mathcal{V}\phi$, that is, if $\alpha \in [\![\phi]\!]^{\mathcal{I}}$ and β is an $\mathcal{I}$-assignment that agrees with α on $\mathcal{V}\phi$, then $\beta \in [\![\phi]\!]^{\mathcal{I}}$.

Predicate logic a prominent example of a constraint language: the well-formed formulas are the constraints, $\mathcal{V}\phi$ are the free variables of a formula ϕ, and for every Tarski interpretation $\mathcal{I}$ the solutions $[\![\phi]\!]^{\mathcal{I}}$ are the $\mathcal{I}$-assignments satisfying ϕ. Viewing predicate logic as a constraint language abstracts away from the syntactic details of formulas.

The following definitions are all made with respect to some given constraint language. Most of the definitions generalize terminology that is well-known for predicate logic.

A constraint ϕ is **satisfiable** if there exists at least one interpretation in which ϕ has a solution. A constraint ϕ is **valid** in an interpretation $\mathcal{I}$ if $[\![\phi]\!]^{\mathcal{I}} = \mathrm{ASS}^{\mathcal{I}}$, that is, every $\mathcal{I}$-assignment is a solution of ϕ in $\mathcal{I}$. Conversely, we say that an interpretation $\mathcal{I}$ **satisfies** a constraint ϕ if ϕ is valid in $\mathcal{I}$. An interpretation is a **model** of a set Φ of constraints if it satisfies every constraint in Φ.

A **renaming** is a bijection $\mathrm{VAR} \to \mathrm{VAR}$ that is the identity except for finitely many exceptions. If ρ is a renaming, we call a constraint ϕ' a **ρ-variant** of a constraint ϕ if

$$\mathcal{V}\phi' = \rho(\mathcal{V}\phi) \quad \text{and} \quad [\![\phi]\!]^{\mathcal{I}} = [\![\phi']\!]^{\mathcal{I}}\rho := \{\alpha\rho \mid \alpha \in [\![\phi']\!]^{\mathcal{I}}\}$$

for every interpretation $\mathcal{I}$. A constraint ϕ' is called a **variant** of a constraint ϕ if there exists a renaming ρ such that ϕ' is a ρ-variant of ϕ.

Proposition 2.1. *A constraint is satisfiable if and only if each of its variants is satisfiable. Furthermore, a constraint is valid in an interpretation $\mathcal{I}$ if and only if each of its variants is valid in $\mathcal{I}$.*

A constraint language is **closed under renaming** if every constraint ϕ has a ρ-variant for every renaming ρ. A constraint language is **closed under intersection** if for every two constraints ϕ and ϕ' there exists a constraint ψ such that $[\![\phi]\!]^{\mathcal{I}} \cap [\![\phi']\!]^{\mathcal{I}} = [\![\psi]\!]^{\mathcal{I}}$ for every interpretation $\mathcal{I}$. A constraint language is **decidable** if the satisfiability of its constraints is decidable.

Let Φ be a set of constraints and $\mathcal{I}$ be an interpretation. The **solutions of Φ in $\mathcal{I}$** are defined as

$$[\![\Phi]\!]^{\mathcal{I}} := \bigcup_{\phi \in \Phi} [\![\phi]\!]^{\mathcal{I}},$$

where $[\![\Phi]\!]^{\mathcal{I}} := \emptyset$ if Φ is empty. Note that this definition interprets a set of constraints disjunctively, while the above definition of a model interprets a set of constraints conjunctively. To ease our notation, we often abbreviate a singleton $\{\phi\}$ to ϕ.

Given a set V of variables, the V**-solutions** of a set Φ of constraints in an interpretation $\mathcal{I}$ are defined as

$$[\![\Phi]\!]^{\mathcal{I}}_V := \{\alpha|_V \mid \alpha \in [\![\Phi]\!]^{\mathcal{I}}\} = \bigcup_{\phi \in \Phi} \{\alpha|_V \mid \alpha \in [\![\phi]\!]^{\mathcal{I}}\}$$

where $\alpha|_V$ is the restriction of α to V. We say that a set of constraints Φ is V**-subsumed** by a set of constraints Φ' and write $\Phi \preceq_V \Phi'$ if $[\![\Phi]\!]^{\mathcal{I}}_V \subseteq [\![\Phi']\!]^{\mathcal{I}}_V$ for every interpretation $\mathcal{I}$. Obviously, V-subsumption defines a preorder on sets of constraints. The corresponding equivalence relation

$$\Phi \sim_V \Phi' \quad :\Longleftrightarrow \quad \Phi \preceq_V \Phi' \ \wedge \ \Phi' \preceq_V \Phi$$

is called V**-equivalence**.

Proposition 2.2. *Renaming is homomorphic with respect to to V-subsumption, that is, if ρ and ρ' are renamings that agree on V, ϕ' is a ρ-variant of a constraint ϕ, ψ' is a ρ'-variant of a constraint ψ, and $\phi \preceq_V \psi$, then $\phi' \preceq_{\rho(V)} \psi'$.*

A constraint language is called **compact** if for every set V of variables, every constraint ϕ, and every set of constraints Φ, ϕ is V-subsumed by Φ if and only if ϕ is V-subsumed by some finite subset of Φ.

Predicate Logic is a compact and undecidable constraint language that is closed under renaming and intersection.

3 Relational Extensions

We now present a construction that, given a constraint language $\mathcal{L}$ and a set $\mathcal{R}$ of relation symbols, extends $\mathcal{L}$ conservatively to a constraint language $\mathcal{R}(\mathcal{L})$ providing for relational atoms, the propositional connectives, and quantification. If the constraints of $\mathcal{L}$ are the equations between first-order terms and the interpretations of $\mathcal{L}$ are the usual Tarski interpretations, then the extension $\mathcal{R}(\mathcal{L})$ is Predicate Logic.

From now on we assume that a set of **relation symbols** is given, where every relation symbol comes with a natural number specifying the number of arguments it takes.

A constraint language $\mathcal{L}$ and a decidable set $\mathcal{R}$ of relation symbols define a constraint language $\mathcal{R}(\mathcal{L})$ extending $\mathcal{L}$ as follows:

1. the variables of $\mathcal{R}(\mathcal{L})$ are the variables of $\mathcal{L}$
2. the constraints of $\mathcal{R}(\mathcal{L})$ are defined inductively as follows:
   - 2.1 every constraint of $\mathcal{L}$ is a constraint of $\mathcal{R}(\mathcal{L})$
   - 2.2 if r is a relation symbol of $\mathcal{R}$ and $\vec{x}$ is a tuple of pairwise distinct variables, then the **atom** $r(\vec{x})$ is a constraint of $\mathcal{R}(\mathcal{L})$, provided the tuple $\vec{x}$ has as many elements as r has arguments
   - 2.3 the **empty conjunction** $\emptyset$ is a constraint of $\mathcal{R}(\mathcal{L})$; furthermore, if F and G are constraints of $\mathcal{R}(\mathcal{L})$, then the **conjunction** $F \ \& \ G$ and the **implication** $F \rightarrow G$ are constraints of $\mathcal{R}(\mathcal{L})$
   - 2.4 if x is a variable and F is a constraint of $\mathcal{R}(\mathcal{L})$, then the **existential quantification** $\exists x.F$ is a constraint of $\mathcal{R}(\mathcal{L})$
3. the variables constrained by a constraint of $\mathcal{R}(\mathcal{L})$ are defined inductively as follows: if ϕ is an $\mathcal{L}$-constraint, then $\mathcal{V}\phi$ is defined as in $\mathcal{L}$; $\mathcal{V}(r(x_1,\ldots,x_n)) := \{x_1,\ldots,x_n\}$; $\mathcal{V}\emptyset := \emptyset$; $\mathcal{V}(F \ \& \ G) := \mathcal{V}F \cup \mathcal{V}G$; $\mathcal{V}(F \rightarrow G) := \mathcal{V}F \cup \mathcal{V}G$; $\mathcal{V}(\exists x.F) := \mathcal{V}F - \{x\}$
4. an interpretation $\mathcal{A}$ of $\mathcal{R}(\mathcal{L})$ is obtained from an $\mathcal{L}$-interpretation $\mathcal{I}$ by choosing for every relation symbol $r \in \mathcal{R}$ a relation $r^{\mathcal{A}}$ on $D^{\mathcal{I}}$ taking the right number of arguments, and by defining:
   - 4.1 $D^{\mathcal{A}} := D^{\mathcal{I}}$
   - 4.2 $[\![\phi]\!]^{\mathcal{A}} := [\![\phi]\!]^{\mathcal{I}}$ if ϕ is an $\mathcal{L}$-constraint
   - 4.3 $[\![r(\vec{x})]\!]^{\mathcal{A}} := \{\alpha \in \mathrm{ASS}^{\mathcal{A}} \mid \alpha(\vec{x}) \in r^{\mathcal{A}}\}$
   - 4.4 $[\![\emptyset]\!]^{\mathcal{A}} := \mathrm{ASS}^{\mathcal{A}}, \qquad [\![F \ \& \ G]\!]^{\mathcal{A}} := [\![F]\!]^{\mathcal{A}} \cap [\![G]\!]^{\mathcal{A}}$

4.5 $[\![F \rightarrow G]\!]^{\mathcal{A}} := (\mathrm{ASS}^{\mathcal{A}} - [\![F]\!]^{\mathcal{A}}) \cup [\![G]\!]^{\mathcal{A}}$

4.6 $[\![\exists x.F]\!]^{\mathcal{A}} := \{\alpha \in \mathrm{ASS}^{\mathcal{A}} \mid \exists\, \beta \in [\![F]\!]^{\mathcal{A}}\ \forall\, y \in \mathcal{V}F.\ \ y = x\ \vee\ \alpha(x) = \alpha(y)\}$.

Since $\mathcal{R}(\mathcal{L})$ is a constraint language, all definitions we have made for constraint languages apply to $\mathcal{R}(\mathcal{L})$. This shows in particular that the notion of a constraint language can be applied iteratively.

As mentioned before, this construction yields Predicate Logic if the constraints of $\mathcal{L}$ are the equations between first-order terms and the interpretations of $\mathcal{L}$ are the usual Tarski interpretations. In $\mathcal{R}(\mathcal{L})$ an atom $r(s_1, \ldots, s_n)$ takes the form

$$\exists x_1 \ldots \exists x_n.\ (p(x_1, \ldots, x_n)\ \&\ x_1 = s_1\ \&\ \ldots\ \&\ x_n = s_n),$$

where $x_1, \ldots, x_n$ are pairwise distinct variables not occurring in the argument terms $s_1, \ldots, s_n$.

In the rest of the paper $\mathcal{R}$ will always be a decidable set of relation symbols and $\mathcal{L}$ will always be a constraint language. Furthermore, ϕ and ψ will always denote $\mathcal{L}$-constraints and F and G will always denote $\mathcal{R}(\mathcal{L})$-constraints.

Proposition 3.1. *Let π be a renaming and F be an $\mathcal{R}(\mathcal{L})$-constraint. Then F' is a π-variant of F if F' can be obtained from F by replacing every variable x with $\pi(x)$ and every $\mathcal{L}$-constraint ϕ with a π-variant of ϕ. Thus $\mathcal{R}(\mathcal{L})$ is closed under renaming if $\mathcal{L}$ is closed under renaming.*

4 Definite Clauses

A **definite clause** is an $\mathcal{R}(\mathcal{L})$-constraint of the form

$$A_1\ \&\ \ldots\ \&\ A_n\ \&\ \phi \rightarrow B,$$

where $n \geq 0$, $A_1, \ldots, A_n$ and B are atoms, and ϕ is an $\mathcal{L}$-constraint. We may write a clause as $B \leftarrow \phi\ \&\ G$ or $B \leftarrow G$. A **definite clause specification** is a set of definite clauses.

Conventional logic programs are definite clause specifications over $\mathcal{E}$, where the constraints of $\mathcal{E}$ are conjunctions of equations between first-order terms and the corresponding ground term algebra is the only interpretation of $\mathcal{E}$. To meet our definition of definite clauses, the clause

```
app(H.R, L, H.RL) ← app(R, L, RL),
```

for instance, is rewritten to the equivalent clause

```
app(X, L, Y) ← (X=H.R & Y=H.RL) & app(R, L, RL)
```

having `X=H.R & Y=H.RL` as $\mathcal{E}$-constraint.

We will show that the nice properties of conventional logic programs extend to definite clause specifications over arbitrary constraint languages.

The **base** of an $\mathcal{R}(\mathcal{L})$-interpretation $\mathcal{A}$ is the $\mathcal{L}$-interpretation that $\mathcal{A}$ is extending. Two $\mathcal{R}(\mathcal{L})$-interpretations are called **base equivalent** if they have the same base.

We define a partial ordering on the set of all $\mathcal{R}(\mathcal{L})$-interpretations by:

$$\mathcal{A} \subseteq \mathcal{B} \quad :\Longleftrightarrow \quad \mathcal{A} \text{ and } \mathcal{B} \text{ are base equivalent and } \quad \forall\, r \in \mathcal{R}.\ \ r^{\mathcal{A}} \subseteq r^{\mathcal{B}}.$$

Proposition 4.1. *Let $\mathcal{A}$ and $\mathcal{B}$ be two $\mathcal{R}(\mathcal{L})$-interpretations and A be an $\mathcal{R}(\mathcal{L})$-atom. Then $[\![A]\!]^{\mathcal{A}} \subseteq [\![A]\!]^{\mathcal{B}}$ if $\mathcal{A} \subseteq \mathcal{B}$.*

The **intersection** $\bigcap_{i\in I} \mathcal{A}_i$ of a family $(\mathcal{A}_i)_{i\in I}$ of base equivalent $\mathcal{R}(\mathcal{L})$-interpretations is obtained by intersecting the denotations of the relation symbols and is again an $\mathcal{R}(\mathcal{L})$-interpretation. Analogously, the **union** $\bigcup_{i\in I} \mathcal{A}_i$ of a family $(\mathcal{A}_i)_{i\in I}$ of base equivalent $\mathcal{R}(\mathcal{L})$-interpretations is obtained by joining the denotations of the relation symbols and is again an $\mathcal{R}(\mathcal{L})$-interpretation.

Proposition 4.2. *Let $\mathcal{I}$ be an $\mathcal{L}$-interpretation. Then the set of all $\mathcal{R}(\mathcal{L})$-interpretations extending $\mathcal{I}$ is a complete lattice.*

Proposition 4.3. *The intersection of a family of base equivalent models of a definite clause specification $\mathcal{S}$ is a model of $\mathcal{S}$.*

The following theorem generalizes the key result for conventional logic programs to general definite clause specifications.

Theorem 4.4. **[Definiteness]** *Let $\mathcal{S}$ be a definite clause specification in $\mathcal{R}(\mathcal{L})$ and $\mathcal{I}$ be an $\mathcal{L}$-interpretation. Then the equations*

$$r^{\mathcal{A}_0} := \emptyset, \quad r^{\mathcal{A}_{i+1}} := \{\alpha(\vec{x}) \mid (r(\vec{x}) \leftarrow G) \in \mathcal{S} \ \wedge \ \alpha \in [\![G]\!]^{\mathcal{A}_i}\}$$

define a chain $\mathcal{A}_0 \subseteq \mathcal{A}_1 \subseteq \cdots$ of $\mathcal{R}(\mathcal{L})$-interpretations whose base is $\mathcal{I}$. Moreover, the union $\bigcup_{i \geq 0} \mathcal{A}_i$ is the least model of $\mathcal{S}$ extending $\mathcal{I}$.

Proof. By induction on i one easily verifies that $\mathcal{A}_i \subseteq \mathcal{A}_{i+1}$. Since every $\mathcal{A}_i$ is an $\mathcal{R}(\mathcal{L})$-interpretation extending $\mathcal{I}$, the union $\mathcal{A} := \bigcup_{i \geq 0} \mathcal{A}_i$ is an $\mathcal{R}(\mathcal{L})$-interpretation extending $\mathcal{I}$.

To show that $\mathcal{A}$ is a model of $\mathcal{S}$, let $A \leftarrow G$ be a clause of $\mathcal{S}$ and $\alpha \in [\![G]\!]^{\mathcal{A}}$. We have to show that $\alpha \in [\![A]\!]^{\mathcal{A}}$. By the iterative definition of $\mathcal{A}$ we know that there is some i such that $\alpha \in [\![G]\!]^{\mathcal{A}_i}$. Hence $\alpha \in [\![A]\!]^{\mathcal{A}_{i+1}} \subseteq [\![A]\!]^{\mathcal{A}}$.

To show that $\mathcal{A}$ is a minimal model of $\mathcal{S}$, let $\mathcal{B}$ be a base equivalent model of $\mathcal{S}$. By induction on i one verifies easily that $\mathcal{A}_i \subseteq \mathcal{B}$ for every i. Hence $\mathcal{A} = \bigcup_{i \geq 0} \mathcal{A}_i \subseteq \mathcal{B}$. □

A set $\mathcal{M}$ of $\mathcal{R}(\mathcal{L})$-constraints is called a **definite specification** if every $\mathcal{L}$-interpretation can be extended to a minimal model of $\mathcal{M}$. The Definiteness Theorem says that every definite clause specification is a definite specification. Many of the interesting properties of definite clause specifications depend solely on their definiteness. If $\mathcal{M}$ is a definite specification in $\mathcal{R}(\mathcal{L})$, then $\mathcal{M}$ uniquely defines the relations of $\mathcal{R}$, that is, for every $\mathcal{L}$-interpretation $\mathcal{M}$ defines unique minimal denotations for the relation symbols of $\mathcal{R}$.

A **goal** is a possibly empty conjunction of $\mathcal{L}$-constraints and $\mathcal{R}(\mathcal{L})$-atoms. To ease our notation, we identify a goal with the multiset consisting of its constraints.

An **observation** is an implication $\phi \rightarrow G$ consisting of an $\mathcal{L}$-constraint ϕ and a goal G.

Proposition 4.5. *Let $\mathcal{M}$ be a definite specification. Then an observation is valid in every model of $\mathcal{M}$ if and only if it is valid in every minimal model of $\mathcal{M}$.*

Let $\mathcal{M}$ be a definite specification. An **$\mathcal{M}$-answer** of a goal G is a satisfiable $\mathcal{L}$-constraint ϕ such that the observation $\phi \rightarrow G$ is valid in every model of $\mathcal{M}$. The preceding proposition says that the $\mathcal{M}$-answers of a goal are completely characterized by the minimal models of $\mathcal{M}$. Thus we say that a set Φ of $\mathcal{M}$-answers of a goal G is **complete** if $[\![\Phi]\!]^{\mathcal{A}}_{\mathcal{V}G} = [\![G]\!]^{\mathcal{A}}_{\mathcal{V}G}$ for every minimal model $\mathcal{A}$ of $\mathcal{M}$.

Proposition 4.6. *Let $\mathcal{M}$ be a definite specification, G be a goal, ϕ be an $\mathcal{M}$-answer of G, and Φ be a complete set of $\mathcal{M}$-answers of G. Then:*

1. $[\![\phi]\!]^{\mathcal{I}}_{\mathcal{V}G} \subseteq [\![\Phi]\!]^{\mathcal{I}}_{\mathcal{V}G}$ *for every $\mathcal{L}$-interpretation $\mathcal{I}$*
2. *if $\mathcal{L}$ is compact, then there exists a finite subset $\Phi' \subseteq \Phi$ such that $[\![\phi]\!]^{\mathcal{I}}_{\mathcal{V}G} \subseteq [\![\Phi']\!]^{\mathcal{I}}_{\mathcal{V}G}$ for every $\mathcal{L}$-interpretation $\mathcal{I}$.*

Proof. The second claim follows immediately from the first claim. To show the first claim, suppose that $\mathcal{I}$ is an $\mathcal{L}$-interpretation. Since $\mathcal{M}$ is definite, there exists a minimal model $\mathcal{A}$ of $\mathcal{M}$ whose base is $\mathcal{I}$. Hence

$$[\![\phi]\!]^{\mathcal{I}}_{\mathcal{V}G} = [\![\phi]\!]^{\mathcal{A}}_{\mathcal{V}G} \subseteq [\![G]\!]^{\mathcal{A}}_{\mathcal{V}G} = [\![\Phi]\!]^{\mathcal{A}}_{\mathcal{V}G} = [\![\Phi]\!]^{\mathcal{I}}_{\mathcal{V}G}$$

since ϕ is an $\mathcal{M}$-answer of G and Φ is a complete set of $\mathcal{M}$-answers of G. $\square$

5 Operational Semantics

In this section we show that one can obtain a complete interpreter for general definite clause specifications by generalizing the SLD-resolution method [Lloyd 84] employed in conventional logic programming. Although our proofs are much more general than the proofs for conventional logic programming given in [Lloyd 84], they are clearer and simpler. In particular, we give a new complexity measure based on a multiset ordering that provides for a strong completeness result making a careful distinction between don't care and don't know choices.

In the following we assume that $\mathcal{L}$ and $\mathcal{R}$ are given, $\mathcal{S}$ is a definite clause specification in $\mathcal{R}(\mathcal{L})$, and V is a finite set of variables.

We define $(\mathcal{S}, V)$**-goal reduction** as the binary relation $\xrightarrow{\mathrm{r}}_{\mathcal{S},V}$ on the set of goals given by the rule:

$$
\begin{aligned}
A \mathbin{\&} G \xrightarrow{\mathrm{r}}_{\mathcal{S},V} F \mathbin{\&} G & \\
&\text{if } A \leftarrow F \text{ is a variant of a clause of } \mathcal{S} \\
&\quad \text{such that } (V \cup \mathcal{V}G) \cap \mathcal{V}F \subseteq \mathcal{V}A.
\end{aligned}
$$

Proposition 5.1. [Soundness of Goal Reduction] *If* $G \xrightarrow{\mathrm{r}}_{\mathcal{S},V} F$*, then* $[\![F]\!]^{\mathcal{A}} \subseteq [\![G]\!]^{\mathcal{A}}$ *for every model* $\mathcal{A}$ *of* $\mathcal{S}$.

We will now show that goal reduction is a complete rule for inferring $\mathcal{S}$-answers, provided all necessary variants of the clauses of $\mathcal{S}$ exist, which is certainly the case if $\mathcal{L}$ and hence $\mathcal{R}(\mathcal{L})$ are closed under renaming. The most important ingredient of the completeness proof is a well-founded complexity measure on goals that can be decreased by goal reduction. From the Definiteness Theorem we know that every minimal model $\mathcal{A}$ of $\mathcal{S}$ can be obtained as the union $\mathcal{A} = \bigcup_{i\geq 0} \mathcal{A}_i$ of a chain $\mathcal{A}_0 \subseteq \mathcal{A}_1 \subseteq \cdots$ being uniquely defined for $\mathcal{A}$. This provides for the following definitions:

1. if $\mathcal{A}$ is a minimal model of $\mathcal{S}$, A is an atom and $\alpha \in [\![A]\!]^{\mathcal{A}}$, then the **complexity** of α for A in $\mathcal{A}$ is

$$\mathrm{comp}(\alpha, A, \mathcal{A}) := \min\{i \mid \alpha \in [\![A]\!]^{\mathcal{A}_i}\}$$

2. if $\mathcal{A}$ is a minimal model of $\mathcal{S}$, G is a goal, and $\alpha \in [\![G]\!]^{\mathcal{A}}$, then the **complexity** $\mathrm{comp}(\alpha, G, \mathcal{A})$ of α for G in $\mathcal{A}$ is the multiset consisting of the complexities $\mathrm{comp}(\alpha, A, \mathcal{A})$ of the atoms A in G.

On the multiset complexities we define a well-founded total ordering by

$$
\begin{aligned}
M \leq M' \;:\Longleftrightarrow\;& \exists \text{ multisets } X \subseteq M \text{ and } X' \subseteq M' \text{ such that} \\
& M = (M' - X') \cup X \quad \text{and} \\
& \forall\, x \in X\; \exists\, x' \in X'.\;\; x < x',
\end{aligned}
$$

where $\subseteq$, $-$, $\cup$, and $\in$ stand for the appropriate multiset operations (see [Dershowitz/Manna 79] for details on multiset orderings).

Now we are ready for the definition of the complexity measure we are actually going to use. Let $\mathcal{A}$ be a minimal model of $\mathcal{S}$, G be a goal and $\alpha \in [\![G]\!]^{\mathcal{A}}_V$. Then the V-**complexity** of α for G in $\mathcal{A}$ is

$$\mathrm{comp}_V(\alpha, G, \mathcal{A}) := \min\{\mathrm{comp}(\beta, G, \mathcal{A}) \mid \beta \in [\![G]\!]^{\mathcal{A}} \ \wedge \ \alpha = \beta|_V\},$$

where the minimum is taken with respect to the multiset ordering.

Theorem 5.2. [Completeness of Goal Reduction] *Let $\mathcal{L}$ be closed under renaming, $\mathcal{A}$ be a minimal model of $\mathcal{S}$, G be a goal, A be an atom in G, and $\alpha \in [\![G]\!]^{\mathcal{A}}_V$. Then there exists a clause C in $\mathcal{S}$ such that*

1. *$(\mathcal{S}, V)$-goal reduction of G on A using a variant of C is possible*
2. *if G_1 is obtained from G by $(\mathcal{S}, V)$-goal reduction on A using a variant of C, then $\alpha \in [\![G_1]\!]^{\mathcal{A}}_V$ and $\mathrm{comp}_V(\alpha, G_1, \mathcal{A}) < \mathrm{comp}_V(\alpha, G, \mathcal{A})$.*

Proof. Let $G = A \;\&\; G'$ and $\beta \in [\![A \;\&\; G']\!]^{\mathcal{A}}$ such that $\alpha = \beta|_V$ and $\mathrm{comp}_V(\alpha, G, \mathcal{A}) = \mathrm{comp}(\beta, G, \mathcal{A})$. Furthermore, let $A = r(\vec{x})$ and $i := \mathrm{comp}(\beta, A, \mathcal{A})$. Then $\beta\vec{x} \in r^{\mathcal{A}_i}$. Hence there exists a clause $r(\vec{y}) \leftarrow F$ in $\mathcal{S}$ and an assignment $\gamma \in [\![F]\!]^{\mathcal{A}_{i-1}}$ such that $\gamma\vec{y} = \beta\vec{x}$.

Now let π be a renaming and $r(\vec{x}) \leftarrow H$ be a π-variant of $r(\vec{y}) \leftarrow F$ such that $\vec{x} = \pi(\vec{y})$ and $(V \cup \mathcal{V}G') \cap \mathcal{V}H \subseteq \mathcal{V}r(\vec{x})$. Such a variant always exists since $\mathcal{L}$ and hence $\mathcal{R}(\mathcal{L})$ are closed under renaming, V is finite, and there are infinitely many variables. Since $H \;\&\; G'$ can be obtained from G by an $(\mathcal{S}, V)$-goal reduction on A, we have the first claim.

To show the second claim, we have to show that $\alpha \in [\![H \;\&\; G']\!]^{\mathcal{A}}_V$ and that $\mathrm{comp}_V(\alpha, H \;\&\; G', \mathcal{A}) < \mathrm{comp}_V(\alpha, G, \mathcal{A})$.

We know that $\gamma\pi^{-1} \in [\![H]\!]^{\mathcal{A}_{i-1}}$ and that $\gamma\pi^{-1}$ and β agree on $\vec{x}$. Hence there exists an assignment $\delta \in \mathrm{ASS}^{\mathcal{A}}$ that agrees with β on $V \cup \mathcal{V}G'$ and with $\gamma\pi^{-1}$ on $\mathcal{V}H$. One verifies easily that δ agrees with α on V, that $\delta \in [\![G']\!]^{\mathcal{A}}$, and that $\delta \in [\![H]\!]^{\mathcal{A}_{i-1}} \subseteq [\![H]\!]^{\mathcal{A}}$. Hence $\alpha \in [\![H \ \& \ G']\!]_V^{\mathcal{A}}$ and

$$\begin{aligned}
\mathrm{comp}_V(\alpha, H \ \& \ G', \mathcal{A}) &\leq \mathrm{comp}(\delta, H \ \& \ G', \mathcal{A}) \\
&= \{\mathrm{comp}(\delta, H, \mathcal{A})\} \cup \mathrm{comp}(\delta, G', \mathcal{A}) \\
&< \{i\} \cup \mathrm{comp}(\beta, G', \mathcal{A}) \\
&= \mathrm{comp}(\beta, G, \mathcal{A}) = \mathrm{comp}_V(\alpha, G, \mathcal{A}).
\end{aligned}$$

□

Corollary 5.3. [Weak Completeness of Goal Reduction] *Let $\mathcal{L}$ be closed under renaming, $\mathcal{A}$ be a minimal model of $\mathcal{S}$, G be a goal and $\alpha \in [\![G]\!]_V^{\mathcal{A}}$. Then there exists an $\mathcal{S}$-answer ϕ of G such that $G \xrightarrow{\mathrm{r}}{}^{*}_{\mathcal{S},V} \phi$ and $\alpha \in [\![\phi]\!]_V^{\mathcal{A}}$.*

Proof. By induction on $\mathrm{comp}_V(\alpha, G, \mathcal{A})$, using the Completeness and Soundness Theorems. □

The Completeness Theorem is stronger than the corollary since it makes a careful distinction between don't care and don't know choices: a complete interpreter can choose any atom in the goal to be reduced, has to try all clauses defining the relation symbol of the atom, and can reduce the goal with any suitable variant of the clause being tried.

In conventional logic programming the search space is significantly reduced by exploiting the fact that only clauses whose head unifies with the atom to be reduced need to be tried. This crucial optimization generalizes nicely to our framework. To show this, we define an additional inference rule, called V**-constraint solving**:

$$\begin{aligned}
\phi \ \& \ \phi' \ \& \ G \ \xrightarrow{\mathrm{c}}_V \ & \phi'' \ \& \ G \\
& \text{if } \ \phi \ \& \ \phi' \sim_{V \cup \mathcal{V}G} \phi'' \text{ and} \\
& \quad \phi,\ \phi',\ \text{and } \phi'' \text{ are } \mathcal{L}\text{-constraints.}
\end{aligned}$$

Proposition 5.4. [Constraint Solving] *Let G be a goal and $G \xrightarrow{\mathrm{c}}_V G'$. Then:*

1. *$[\![G]\!]_V^{\mathcal{A}} = [\![G']\!]_V^{\mathcal{A}}$ for every interpretation $\mathcal{A}$ of $\mathcal{R}(\mathcal{L})$*

2. *$comp_V(\alpha, G, \mathcal{A}) = comp_V(\alpha, G', \mathcal{A})$ for every minimal model $\mathcal{A}$ of $\mathcal{S}$ and every $\alpha \in [\![G]\!]_V^{\mathcal{A}}$.*

Next we require that the underlying constraint language $\mathcal{L}$ comes with a set of **normal $\mathcal{L}$-constraints** such that every normal $\mathcal{L}$-constraint is satisfiable, and that for every satisfiable conjunction of $\mathcal{L}$-constraints and every finite set V of variables there exists a V-equivalent normal $\mathcal{L}$-constraint. For conventional logic programming, the normal constraints are the equational representations of idempotent substitutions.

Finally, we require that the goal to be reduced contains only one $\mathcal{L}$-constraint that has to be normal and that the constraints in the clauses of $\mathcal{S}$ be normal. Obviously, a definite clause specification can be transformed to this format without changing its models.

The optimized interpreter works as follows: immediately after a goal reduction step, the constraint solving rule is applied to the conjunction ϕ & ϕ' consisting of the normal constraint from the reduced goal and the normal constraint from the applied clause, where a so-called **constraint solver** attempts to compute a normal constraint that is equivalent to ϕ & ϕ'. If the constraint solver detects that ϕ & ϕ' is unsatisfiable, then the interpreter tries immediately another clause since this part of the search space cannot contain any answers. In conventional logic programming, the constraint solver is given by a term unification procedure, where unification succeeds if and only if the corresponding equations are satisfiable in the ground term algebra.

6 A Type Discipline

A **declaration** is an $\mathcal{R}(\mathcal{L})$-implication of the form

$$A \;\rightarrow\; \exists y_1. \ldots . \exists y_n. \; \phi,$$

where A is an atom, ϕ is a satisfiable $\mathcal{L}$-constraint, and $y_1, \ldots, y_n$ are the variables in $\mathcal{V}\phi - \mathcal{V}A$. For convenience we use the abbreviation $A \rightarrow_{\exists} \phi$.

Proposition 6.1. *A declaration $r(\vec{x}) \rightarrow_\exists \phi$ is* **valid** *in an $\mathcal{R}(\mathcal{L})$-interpretation $\mathcal{A}$ if and only if $r^\mathcal{A} \subseteq \{\alpha(\vec{x}) \mid \alpha \in [\![\phi]\!]^\mathcal{A}\}$.*

Declarations prescribe upper bounds for relations. If $\mathcal{L}$ is a constraint language with sorts, typical declarations might be:

```
plus(X,Y,Z) →∃ X:int & Y:int & Z:int
likes(X,Y) →∃ X:person & Y:person.
```

If Feature Logic [Smolka 88] is employed as the underlying constraint language, the arguments of a relation can be constrained with feature terms employing intersections, unions, complements and feature constraints. Similar declarations are possible using the concept and role descriptions of KL-ONE [Levesque/Brachman 87]. The idea even applies to conventional logic programming, where we can write declarations like

$$p(x,y) \;\rightarrow\; \exists\, z.\; y = f(x,z).$$

Giving declarations for the relation symbols of a definite clause specification makes it easier to understand the specification since looking at the declarations alone already gives one a rough understanding of the specified relations. Declarations are much easier to understand than clauses since a declaration specifies an upper bound for a relation without recourse to other relations.

We establish an undecidable notion of well-typedness by saying that a definite specification $\mathcal{M}$ **satisfies** a set $\mathcal{D}$ of declarations if every minimal model of $\mathcal{M}$ is a model of $\mathcal{D}$.

Proposition 6.2. *Let $\mathcal{M}$ be a definite specification and $\mathcal{D}$ be a set of declarations. Then the following conditions are equivalent:*

1. *$\mathcal{M}$ satisfies $\mathcal{D}$*
2. *$\mathcal{M} \cup \mathcal{D}$ is a definite specification*
3. *$\mathcal{M}$ and $\mathcal{M} \cup \mathcal{D}$ have the same minimal models.*

Furthermore, if the above conditions are satisfied, then an observation is valid in every model of $\mathcal{M}$ if and only if it is valid in every model of $\mathcal{M} \cup \mathcal{D}$.

Proof. "(1) $\Rightarrow$ (2)". Let $\mathcal{I}$ be an $\mathcal{L}$-interpretation. We have to show that $\mathcal{I}$ can be extended to a minimal model of $\mathcal{M} \cup \mathcal{D}$. Since $\mathcal{M}$ is a definite specification, $\mathcal{I}$ can be extended to a minimal model $\mathcal{A}$ of $\mathcal{M}$. Hence we know by our assumption that $\mathcal{A}$ is a model of $\mathcal{M} \cup \mathcal{D}$. To show that $\mathcal{A}$ is a minimal model of $\mathcal{M} \cup \mathcal{D}$, let $\mathcal{B} \subseteq \mathcal{A}$ be a model of $\mathcal{M} \cup \mathcal{D}$. Then $\mathcal{B}$ is in particular a model of $\mathcal{M}$ and hence $\mathcal{B} = \mathcal{A}$ since $\mathcal{A}$ is a minimal model of $\mathcal{M}$.

"(2) $\Rightarrow$ (3)". Let $\mathcal{A}$ be a minimal model of $\mathcal{M}$. Since $\mathcal{M} \cup \mathcal{D}$ is a definite specification by assumption, we know that $\mathcal{M} \cup \mathcal{D}$ has a minimal model $\mathcal{B}$ such that $\mathcal{A}$ and $\mathcal{B}$ have the same base. In particular, we know that $\mathcal{B}$ is a model of $\mathcal{M}$. Since $\mathcal{A}$ is a minimal model of $\mathcal{M}$, we know that $\mathcal{A} \subseteq \mathcal{B}$. Since $\mathcal{B}$ is a model of $\mathcal{D}$, we hence know that $\mathcal{A}$ is a model of $\mathcal{M} \cup \mathcal{D}$. Since $\mathcal{B}$ is a minimal model of $\mathcal{M} \cup \mathcal{D}$, we thus know that $\mathcal{A} = \mathcal{B}$. Hence $\mathcal{A}$ is a minimal model of $\mathcal{M} \cup \mathcal{D}$.

Let $\mathcal{A}$ be a minimal model of $\mathcal{M} \cup \mathcal{D}$. Then $\mathcal{A}$ is a model of $\mathcal{M}$ and, since $\mathcal{M}$ is definite, $\mathcal{M}$ has a minimal model $\mathcal{B} \subseteq \mathcal{A}$. Since $\mathcal{A}$ is a model of $\mathcal{D}$, we know that $\mathcal{B}$ is a model of $\mathcal{D}$. Hence $\mathcal{B}$ is a model of $\mathcal{M} \cup \mathcal{D}$ and, since $\mathcal{A}$ is a minimal model of $\mathcal{M} \cup \mathcal{D}$, we know that $\mathcal{A} = \mathcal{B}$. Hence $\mathcal{A}$ is a minimal model of $\mathcal{M}$.

"(3) $\Rightarrow$ (1)". Trivial.

The observational equivalence of $\mathcal{M}$ and $\mathcal{M} \cup \mathcal{D}$ follows from (3) and Proposition 4.5. □

In practice, a major advantage of type disciplines is that one can detect specification errors automatically by checking whether a specification is well-typed. This, of course, requires that the well-typedness of a specification is decidable. Our current notion of well-typedness, however, is undecidable even if the underlying constraint language is decidable. We will now devise a stronger more syntactically oriented notion of well-typedness that is decidable if the underlying constraint language is decidable.

An atom A is **well-typed** under an $\mathcal{L}$-constraint ϕ with respect to a declaration D if $\phi \preceq_{\mathcal{V}A} \psi$ for every variant $A \rightarrow_\exists \psi$ of D. Note that, if A and D have different relation symbols, then A is well-typed under every $\mathcal{L}$-constraint with respect to D.

Proposition 6.3. *Let ϕ be an $\mathcal{L}$-constraint and $A \rightarrow_\exists \psi$ be a variant of a declaration D. Then A is well-typed under ϕ with respect to D if and only if $\phi \preceq_{\mathcal{V}A} \psi$.*

Proof. Follows from Proposition 2.2. □

Let $\mathcal{D}$ be a set of declarations. A definite clause C is **well-typed** with respect to $\mathcal{D}$ if every atom of C is well-typed under the $\mathcal{L}$-constraint of C with respect to every declaration of $\mathcal{D}$. (For technical convenience, we don't require that the $\mathcal{L}$-constraint of a well-typed clause be satisfiable.) A definite clause specification $\mathcal{S}$ is **well-typed** with respect to $\mathcal{D}$ if every clause of $\mathcal{S}$ is well-typed with respect to $\mathcal{D}$.

Proposition 6.4. *Let $\mathcal{L}$ be a constraint language such that, for every renaming ρ and every finite set V of variables, ρ-variants are computable and V-subsumption is decidable. Then the well-typedness of finite definite clause specifications with respect to finite sets of declarations is decidable.*

Theorem 6.5. *Let $\mathcal{L}$ be closed under renaming, $\mathcal{S}$ be a definite clause specification and $\mathcal{D}$ be a set of declarations. Then $\mathcal{S}$ satisfies $\mathcal{D}$ if $\mathcal{S}$ is well-typed with respect to $\mathcal{D}$.*

Proof. Let $\mathcal{A}$ be a minimal model of $\mathcal{S}$, $r(\vec{x}) \rightarrow_\exists \phi$ be a declaration of $\mathcal{D}$, and α be an $\mathcal{A}$-assignment such that $\alpha\vec{x} \in r^{\mathcal{A}}$. We have to show that there exists an assignment $\gamma \in [\![\phi]\!]^{\mathcal{A}}$ that agrees with α on $\mathcal{V}\vec{x}$.

Since $\mathcal{L}$ is closed under renaming, we can assume without loss of generality that $\vec{x} = \vec{y}$ for every clause $r(\vec{y}) \leftarrow G$ in $\mathcal{S}$.

Using the construction of the Definiteness Theorem, we know that $\alpha\vec{x} \in r^{\mathcal{A}_{i+1}}$ for some i. Hence there exists a clause $r(\vec{x}) \leftarrow \psi \;\&\; F$ and an assignment

$\beta \in [\![\psi]\!]^{\mathcal{A}}$ such that $\beta\vec{x} = \alpha\vec{x}$. Since $\mathcal{S}$ is well-typed, we know $\psi \preceq_{\mathcal{V}\vec{x}} \phi$. Hence there exists an assignment $\gamma \in [\![\phi]\!]^{\mathcal{A}}$ such that γ agrees with β and hence with α on $\mathcal{V}\vec{x}$. □

A goal G is **well-typed** with respect to $\mathcal{D}$ if every atom in G is well-typed under some $\mathcal{L}$-constraint in G with respect to $\mathcal{D}$.

Proposition 6.6. [Well-Typed Programs Don't Go Wrong] *Let $\mathcal{S}$ be a definite clause specification that is well-typed with respect to a set $\mathcal{D}$ of declarations, and let G be a goal that is well-typed with respect to $\mathcal{D}$. Then G' is well-typed with respect to $\mathcal{D}$ if G' is obtained from G by $(\mathcal{S}, V)$-goal reduction or V-constraint solving.*

7 Type Inference

In the following we assume that $\mathcal{S}$ is a definite clause specification and $\mathcal{D}$ is a set of declarations.

We will show that, if $\mathcal{S}$ satisfies $\mathcal{D}$, one can compute, by superposing the declarations of $\mathcal{D}$ with the clauses of $\mathcal{S}$, a definite clause specification $\mathcal{S}'$ that is well-typed with respect to $\mathcal{D}$ such that $\mathcal{S}$ and $\mathcal{S}'$ have the same minimal models. Thus $\mathcal{S}$ and its well-typed version $\mathcal{S}'$ are observationally equivalent. We will also show that, in general, $\mathcal{S}' \cup \mathcal{D}$ is semantically weaker than $\mathcal{S} \cup \mathcal{D}$, that is, has more nonminimal models than $\mathcal{S} \cup \mathcal{D}$.

This result together with the results of the preceding section clarifies the relationship between our two notions of well-typedness. Type inference is also useful for practical applications since one can write an abbreviated definite clause specification $\mathcal{S}$ together with a set $\mathcal{D}$ of declarations and automatically infer the "intended" well-typed specification $\mathcal{S}'$ satisfying $\mathcal{D}$. If type inference is used for this purpose, it isn't necessary that the abbreviated specification $\mathcal{S}$ satisfies $\mathcal{D}$.

We start by defining a quasi-ordering on definite clauses:

$$(A \leftarrow \phi \;\&\; G) \;\preceq\; (A \leftarrow \phi' \;\&\; G) \quad :\Longleftrightarrow \quad \phi' \preceq_{\mathcal{V}A \cup \mathcal{V}G} \phi.$$

If $C \preceq C'$, we say that C' is a **weakening** of C. Note that, if C' is a weakening of C, the clauses C and C' are equal up to their $\mathcal{L}$-constraints.

To render a clause well-typed with respect to $\mathcal{D}$, we will replace it with a minimal weakening that is well-typed with respect to $\mathcal{D}$. The next propositions says that it doesn't matter which minimal well-typed weakening we choose.

Proposition 7.1. *If C' is a weakening of C, then every model of C is a model of C'.*

To compute minimal well-typed weakenings, we define the following type inference rule for definite clauses:

$$(A \leftarrow \phi \mathbin{\&} G) \quad \xrightarrow{\mathrm{t}}_{\mathcal{D}} \quad (A \leftarrow \phi' \mathbin{\&} G)$$

$$\begin{array}{ll} \text{if} & B \text{ is an atom in } A \mathbin{\&} G \text{ and} \\ & \quad B \rightarrow_{\exists} \psi \text{ is a variant of a declaration of } \mathcal{D} \text{ such that} \\ & \qquad \mathcal{V}\psi \cap (\mathcal{V}A \cup \mathcal{V}\phi \cup \mathcal{V}G) \subseteq \mathcal{V}B, \\ & \qquad \phi \preceq_{\mathcal{V}B} \psi \text{ does not hold, and} \\ & \qquad \phi' \sim_{\mathcal{V}A \cup \mathcal{V}G} \phi \mathbin{\&} \psi. \end{array}$$

Theorem 7.2. **[Type Inference]** *Let $\mathcal{L}$ be closed under renaming and intersection and let C be a definite clause. Then:*

1. *there are no infinite chains $C \xrightarrow{\mathrm{t}}_{\mathcal{D}} C_1 \xrightarrow{\mathrm{t}}_{\mathcal{D}} C_2 \xrightarrow{\mathrm{t}}_{\mathcal{D}} \cdots$*
2. *if the type inference rule $\xrightarrow{\mathrm{t}}_{\mathcal{D}}$ cannot be applied to C, then C is well-typed with respect to $\mathcal{D}$*
3. *if $C \xrightarrow{\mathrm{t}}_{\mathcal{D}} C'$, then C' is a weakening of C such that*

   3.1 if C'' is a weakening of C that is well-typed with respect to $\mathcal{D}$, then C'' is a weakening of C'

   3.2 if $\mathcal{S}$ satisfies $\mathcal{D}$ and $\mathcal{S}'$ is obtained from $\mathcal{S}$ by replacing C with C', then $\mathcal{S}$ and $\mathcal{S}'$ have the same minimal models.

Proof. *1.* C has finitely many pairs (B, D) such that B is an atom of C that is not well-typed under the $\mathcal{L}$-constraint of C with respect to the declaration $D \in \mathcal{D}$. An application of the type inference rule reduces the number of these pairs.

2. The claim is easily verified using that $\mathcal{L}$ is closed under renaming and intersection.

3. Let $C = (A \leftarrow \phi \;\&\, G)$, $C' = (A \leftarrow \phi' \;\&\, G)$, B be an atom in C, $B \rightarrow_{\exists} \psi$ be a variant of a declaration of $\mathcal{D}$ such that $\mathcal{V}\psi \cap (\mathcal{V}A \cup \mathcal{V}\phi \cup \mathcal{V}G) \subseteq \mathcal{V}B$, and $\phi' \sim_{\mathcal{V}A \cup \mathcal{V}G} \phi \;\&\, \psi$. Then C' is obviously a weakening of C.

3.1. Let $C'' = (A \leftarrow \phi'' \;\&\, G)$ be well-typed with respect to $\mathcal{D}$ and let $\phi'' \preceq_{\mathcal{V}A \cup \mathcal{V}G} \phi$. We have to show that $\phi'' \preceq_{\mathcal{V}A \cup \mathcal{V}G} \phi'$. Since we know that $\phi' \sim_{\mathcal{V}A \cup \mathcal{V}G} \phi \;\&\, \psi$, it suffices to show that $\phi'' \preceq_{\mathcal{V}A \cup \mathcal{V}G} \phi \;\&\, \psi$. Let $\mathcal{I}$ be an $\mathcal{L}$-interpretation and $\alpha \in [\![\phi'']\!]^{\mathcal{I}}$. We have to show that there exists an assignment $\beta \in [\![\phi \;\&\, \psi]\!]^{\mathcal{I}}$ that agrees with α on $\mathcal{V}A \cup \mathcal{V}G$.

Since $\phi'' \preceq_{\mathcal{V}A \cup \mathcal{V}G} \phi$, we know that there exists an assignment $\beta \in [\![\phi]\!]^{\mathcal{I}}$ that agrees with α on $\mathcal{V}A \cup \mathcal{V}G$. Since B is well-typed under ϕ'' with respect to $B \rightarrow_{\exists} \psi$, we know that $\phi'' \preceq_{\mathcal{V}B} \psi$. Thus there exists an assignment $\gamma \in [\![\psi]\!]^{\mathcal{I}}$ that agrees with α and hence with β on $\mathcal{V}B$. Since $\mathcal{V}\psi \cap (\mathcal{V}A \cup \mathcal{V}\phi \cup \mathcal{V}G) \subseteq \mathcal{V}B$, we can assume without loss of generality that γ agrees with β on $\mathcal{V}A \cup \mathcal{V}\phi \cup \mathcal{V}G$. Thus $\gamma \in [\![\phi \;\&\, \psi]\!]^{\mathcal{I}}$ and γ agrees with α on $\mathcal{V}A \cup \mathcal{V}G$.

3.2. Let $\mathcal{S}$ satisfy $\mathcal{D}$ and let $\mathcal{S}'$ be obtained from $\mathcal{S}$ by replacing C with C'. Furthermore, let $\mathcal{I}$ be an $\mathcal{L}$-interpretation and let $\mathcal{A}_0 \subseteq \mathcal{A}_1 \subseteq \cdots$ and $\mathcal{A}'_0 \subseteq \mathcal{A}'_1 \subseteq \cdots$ be the chains defining the extensions of $\mathcal{I}$ to minimal models of $\mathcal{S}$ and $\mathcal{S}'$ as in the proof of the Definiteness Theorem. We show by induction on i that $\mathcal{A}_i = \mathcal{A}'_i$ for every $i \geq 0$. For $i = 0$ the claim is trivial. To show $\mathcal{A}_{i+1} = \mathcal{A}'_{i+1}$, it suffices to show that $[\![\phi \;\&\, G]\!]^{\mathcal{A}_i}_{\mathcal{V}A} = [\![\phi' \;\&\, G]\!]^{\mathcal{A}_i}_{\mathcal{V}A}$.

3.2.1. Let $\alpha \in [\![\phi' \;\&\, G]\!]^{\mathcal{A}_i}$. We show that there exists an assignment $\beta \in [\![\phi \;\&\, G]\!]^{\mathcal{A}_i}$ that agrees with α on $\mathcal{V}A$. Since $\phi' \sim_{\mathcal{V}A \cup \mathcal{V}G} \phi \;\&\, \psi$, we know that there exists an assignment $\beta \in [\![\phi]\!]^{\mathcal{A}_i}$ that agrees with α on $\mathcal{V}A \cup \mathcal{V}G$. Hence $\beta \in [\![\phi \;\&\, G]\!]^{\mathcal{A}_i}$.

3.2.2. Let $\alpha \in [\![\phi \;\&\, G]\!]^{\mathcal{A}_i}$ and B be an atom in G. We show that there exists an assignment $\gamma \in [\![\phi' \mathbin{\&} G]\!]^{\mathcal{A}_i}$ that agrees with α on $\mathcal{V}A$. Since $\mathcal{S}$ satisfies $\mathcal{D}$, we know that $\mathcal{A}_i$ satisfies $B \rightarrow_\exists \psi$. Hence there exists an assignment $\beta \in [\![\psi]\!]^{\mathcal{A}_i}$ that agrees with α on $\mathcal{V}B$. Since $\mathcal{V}\psi \cap (\mathcal{V}A \cup \mathcal{V}\phi \cup \mathcal{V}G) \subseteq \mathcal{V}B$, we can assume without loss of generality that β agrees with α on $\mathcal{V}A \cup \mathcal{V}\phi \cup \mathcal{V}G$. Hence $\beta \in [\![\phi \mathbin{\&} \psi \mathbin{\&} G]\!]^{\mathcal{A}_i}$. Since $\phi' \sim_{\mathcal{V}A \cup \mathcal{V}G} \phi \mathbin{\&} \psi$, there exists an assignment $\gamma \in [\![\phi']\!]^{\mathcal{A}_i}$ that agrees with β on $\mathcal{V}A \cup \mathcal{V}G$. Hence $\gamma \in [\![\phi' \mathbin{\&} G]\!]^{\mathcal{A}_i}$ and γ agrees with α on $\mathcal{V}A$.

3.2.3. Let $\alpha \in [\![\phi \mathbin{\&} G]\!]^{\mathcal{A}_i}$ and $B = A$. We show that there exists an assignment $\gamma \in [\![\phi' \mathbin{\&} G]\!]^{\mathcal{A}_i}$ that agrees with α on $\mathcal{V}A$. Since $\mathcal{S}$ satisfies $\mathcal{D}$, we know that $\mathcal{A}_{i+1}$ satisfies $A \rightarrow_\exists \psi$. Since $\alpha \in [\![A]\!]^{\mathcal{A}_{i+1}}$, there exists an assignment $\beta \in [\![\psi]\!]^{\mathcal{A}_i} = [\![\psi]\!]^{\mathcal{A}_{i+1}}$ that agrees with α on $\mathcal{V}A$. Since $\mathcal{V}\psi \cap (\mathcal{V}\phi \cup \mathcal{V}G) \subseteq \mathcal{V}A$, we can assume without loss of generality that β agrees with α on $\mathcal{V}A \cup \mathcal{V}\phi \cup \mathcal{V}G$. Hence $\beta \in [\![\phi \mathbin{\&} \psi \mathbin{\&} G]\!]^{\mathcal{A}_i}$. Since $\phi' \sim_{\mathcal{V}A \cup \mathcal{V}G} \phi \mathbin{\&} \psi$, there exists an assignment $\gamma \in [\![\phi']\!]^{\mathcal{A}_i}$ that agrees with β on $\mathcal{V}A \cup \mathcal{V}G$. Hence $\gamma \in [\![\phi' \mathbin{\&} G]\!]^{\mathcal{A}_i}$ and γ agrees with α on $\mathcal{V}A$. □

Corollary 7.3. *Let $\mathcal{S}'$ be obtained from $\mathcal{S}$ by replacing every clause of $\mathcal{S}$ by a minimal weakening that is well-typed with respect to $\mathcal{D}$. Then $\mathcal{S}'$ is well-typed with respect to $\mathcal{D}$ and, if $\mathcal{S}$ satisfies $\mathcal{D}$, then $\mathcal{S}$ and $\mathcal{S}'$ have the same minimal models.*

One could expect that $\mathcal{S}$ and $\mathcal{S}'$ not only have the same minimal models but have the same models in general. By Proposition 7.1 we know that every model of $\mathcal{S}$ is a model of $\mathcal{S}'$. However, the following example shows that the other direction doesn't hold. This means that $\mathcal{S}'$ is semantically weaker than $\mathcal{S}$ in that it allows for more nonminimal models than $\mathcal{S}$.

Example 7.4. Let $\mathcal{L}$ be the constraint language whose constraints are conjunctions of equations between first-order terms and let the ground term algebra be the only interpretation of $\mathcal{L}$. Furthermore, let a declaration D and definite clauses C and C' be given as follows:

$$D: \quad p(x) \rightarrow_\exists x = a$$
$$C: \quad p(x) \leftarrow q(x)$$
$$C': \quad p(x) \leftarrow x = a \mathbin{\&} q(x).$$

The minimal model of C has empty denotations for p and q and thus trivially satisfies D. Note that C' can be obtained from C with type inference modulo D. Now let $\mathcal{B}$ be an interpretation such that $p^{\mathcal{B}} = \{a\}$ and $q^{\mathcal{B}}$ is the set of all ground terms (assume that there is more than one). Obviously, $\mathcal{B}$ is a model of C' and D but is not a model of C.

References

H. Aït-Kaci, An Algebraic Semantics Approach to the Effective Resolution of Type Equations. Theoretical Computer Science 45, 1986, 293–351.

H. Aït-Kaci and R. Nasr, LOGIN: A Logic Programming Language with Built-In Inheritance. The Journal of Logic Programming, 1986, 3, 185–215.

R.J. Brachman and J.G. Schmolze, An Overview of the KL-ONE Knowledge Representation System. Cognitive Science 9(2), 1985, 171–216.

A. Colmerauer, H. Kanoui, and M. Van Caneghem, Prolog, Theoretical Principles and Current Trends. Technology and Science of Informatics 2,4, 1983, 255–292.

A. Colmerauer, Equations and Inequations on Finite and Infinite Trees. Proc. of the 2nd International Conference on Fifth Generation Computer Systems, 1984, 85–99.

A. Colmerauer, Final Specifications for Prolog-III. Manuscript, Esprit Reference Number P1210(1106), February 1988. (See also: Opening the Prolog-III Universe, Byte Magazine, August 1987.)

N. Dershowitz and Z. Manna, Proving Termination with Multiset Orderings. Communications of the ACM, 22, 1979, 465–476.

M. Dincbas, P. Van Hentenryck, H. Simonis, A Aggoun and T. Graf, Applications of CHIP to Industrial and Engineering Problems. Proceedings of the First International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, Tullahoma, Tennessee, June 1988.

J.A. Goguen and J. Meseguer, Eqlog: Equality, Types, and Generic Modules for Logic Programming. In D. DeGroot and G. Lindstrom (eds.), Logic Programming, Functions, Relations, and Equations; Prentice Hall 1986.

J. Jaffar and J.-L. Lassez, Constraint Logic Programming. Technical Report, Department of Computer Science, Monash University, June 1986.

J. Jaffar and J.-L. Lassez, Constraint Logic Programming. Proc. of the 14th ACM Symposium on Principles of Programming Languages, Munich, 1987, 111–119.

J. Jaffar and S. Michaylov, Methodology and Implementation of a CLP System. Proceedings of the 4th International Conference on Logic Programming, J.-L. Lassez (Ed.), MIT Press, 1987.

M.E. Johnson, Attribute-Value Logic and the Theory of Grammar. PhD Dissertation, Stanford University, 1987. To appear as CSLI Lecture Notes.

R. Kaplan and J. Bresnan, Lexical-Functional Grammar, a Formal System for Grammatical Representation. In J. Bresnan (Ed.), The Mental Representation of Grammatical Relations, The MIT Press, 1982, 173–381.

H.J. Levesque and R.J. Brachman, Expressiveness and Tractability in Knowledge Representation and Reasoning. Computational Intelligence 3, 1987, 78–93.

J.W. Lloyd, Foundations of Logic Programming. Springer Verlag, 1984.

K. Mukai, Anadic Tuples in Prolog. Technical Report TR-239, ICOT, Tokyo, 1987.

W.C. Rounds and R.T. Kasper, A Complete Logical Calculus for Record Structures Representing Linguistic Information. Proc. of the First IEEE Symposium on Logic in Computer Science, Boston, 1986, 38–43.

G. Smolka, A Feature Logic with Subsorts. LILOG Report 33, IBM Deutschland, West Germany, May 1988. To appear in the proceedings of the Workshop on Unification Formalisms—Syntax, Semantics and Implementation, Titisee, The MIT Press.