The Completeness of Logic Programming with Sort Predicates

Ken Kaneiwa
Foundations of Informatics Research Division
National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo, 101-8430 Japan

Abstract

An order-sorted logic can be regarded as a generalized first-order predicate logic that includes many and ordered sorts (i.e. a sort-hierarchy). In the fields of knowledge representation and AI, this logic with sort-hierarchy has been used to design a logic-based language appropriate for representing taxonomic knowledge. By incorporating the sort-hierarchy, order-sorted resolution and sorted logic programming have been formalized that provide efficient reasoning mechanisms with structural representation. In this work, Beierle et al. developed an order-sorted logic to couple separated taxonomic knowledge and assertional knowledge. Namely, its language allows us to make use of sorts to denote not only the types of terms but also unary predicates (called sort predicates). In this paper, we propose a sorted logic programming language with sort predicates in order to improve the practicability of the logic proposed by Beierle et al. The linear resolution is obtained by adding inference relative to sort predicates and subsort relations. In the semantics, the terms and formulas that follow the sorted signature extended with sort predicates are interpreted over its corresponding Σ^+-structures. Finally, we build the Herbrand models of programs containing sort predicates, and thus prove the soundness and completeness of this logic programming.

order-sorted logic, sort predicate, logic programming, knowledge base system

1 Introduction

Knowledge representation languages based on logic leads to sound inference systems because of their rigorous syntax, semantics and inference machinery. However, predicate logic used commonly as a tool of knowledge representation is not sufficient for representing structural knowledge, not only 'flat' knowledge [4]. In order to solve the lack of expressivity without losing the theoretical foundation, various logic languages [1, 2, 13, 11, 12] with class-hierarchies (or sort-hierarchies) have been proposed. These are called hybrid languages that include two kinds of knowledge representation: taxonomical knowledge and assertional knowledge. Using these languages, we can describe a sort-hierarchy as taxonomical knowledge, the elements of which are used to declare the sorts of variables, functions and predicates in formulas as assertional knowledge. Among such hybrid languages, Frisch's sorted logic [6] contains a sort theory, which is a set of unary predicate formulas, to handle sort information (e.g. a sort-hierarchy).

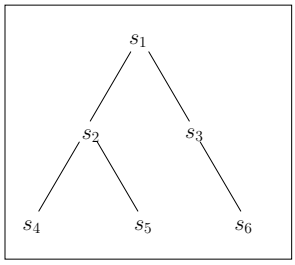
In the area of mathematical logic, many-sorted logic [8] has been studied as first-order predicate logic generalized by introducing many sorts. In particular, it is said to be order-sorted logic [15] if sorts are ordered. These sorted logics provide theoretical foundations for knowledge base systems with class-hierarchies. Table 1 arranges sorted logics and their inference systems for general/Horn clauses, in which sorted logics are classified as one-sorted, many-sorted and order-sorted. One-sorted logic corresponds to first-order predicate logic for which efficient inference systems have been proposed, e.g., resolution principle and logic programming. Also many-/order-sorted logics have their resolution systems [18, 19, 16, 20] and logic programming [9, 7].

On the other hand, Beierle et al. [3] presented a practical order-sorted logic that was inspired by hybrid knowledge base systems. In knowledge representation, sort symbols are used to build a sort-hierarchy and

Table 1: Studies of sorted logics

Sorted logics	Sort expression	Inference for clauses	Inference for Horn clauses
first-order predicate logic	one sort	resolution	logic programming
many-sorted logic	many sorts	sorted unification + resolution	sorted logic programming
order-sorted logic	ordered sorts		
sorted logic with sort predicates [3]	a sort-hierarchy and sort predicates	inference rules for sort predicates + sorted unification + resolution	

Figure 1: The use of sort predicates in a knowledge base



A sort-hierarchy

$p_1(x\colon s_1) \vee p_2(y\colon s_2)$
$p_1(c\colon s_1)$
$s_1(c_2\colon s_2) \vee p_1(c_3\colon s_1)$
$s_4(c_4\colon s_1)$

A knowledge base

to denote the sorts of variables, functions and predicates. Moreover, since sorts are semantically equivalent to unary predicates, it is required that each sort is used as a predicate. However, sorts and predicates are syntactically unrelated to each other, and then ordinary sorted inference systems can not deal with sorts as predicates. Therefore, by introducing unary predicates (called sort predicates) corresponding to sorts and new inference rules, the authors enabled an order-sorted logic to combine sorts in a sort-hierarchy with assertional knowledge in a knowledge base (as shown in Figure 1). $x\colon s$ expresses a variable of sort s, and $c\colon s$ denotes a constant of sort s. For any predicate p, $p(x\colon s)$ expresses that x of sort s is in p. Furthermore, $s(t)$ where s is a sort predicate denotes that a term t is in s. In addition to the extension of the language, they introduced new inference rules for knowledge base reasoning that were based on the subsort relation $s_{\ 1} \leq s_2$ regarded as the implication $s_1(x) \rightarrow s_2(x)$. Let C, C_1, C_2 be clauses, s, s_1, s_2 sorts or sort predicates, t, t_1, t_2 sorted terms and $sort(t)$ the sort of term t. With sorted substitution and unification the two inference rules [1] shown in [3] are given by the following.

$$\frac{\neg s_1(t_1) \vee C_1 \quad s_2(t_2) \vee C_2}{(C_1 \vee C_2)\theta}\ (subsort)$$

where $s_2 \leq s_1$ and $t_1\theta = t_2\theta$.

$$\frac{\neg s(t) \vee C}{C\theta}\ (sort\ predicate)$$

where $sort(t\theta) \leq s$. The first is an inference rule with respect to subsorts, and the second is an elimination rule of sort predicates the assertions of which are false. These rules generate reasoning for sorts in assertional knowledge. However, in order to supply a foundation to implement knowledge representation languages, it is needed that more effective reasoning mechanisms are formalized such as linear resolution for Horn clauses in the sorted logic.

[1] These rules are obtained by simplifying the inference rules proposed in [3]. But the original rules include them and guarantee the soundness.

In this paper, we present logic programming equipped with ordered sorts and sort predicates, and prove the soundness and completeness of the linear resolution. For this purpose, we need to incorporate the following into the logic programming language:

- Horn clausal forms with sort predicates
- An inference rule into which the resolution rules in [3] are integrated
- Linear resolution for goal clauses
- Herbrand Σ^+-models of programs with sort predicates

The first is to employ restricted clausal forms, which are Horn clausal forms $L \leftarrow L_1 \wedge \ldots \wedge L_n$, a finite set of which is called a program. The second and third are to integrate the resolution rules for dealing with sort predicates into a resolution rule, and to define its linear resolution. One of the resolution rules is deleted by identifying a goal (which we call a successful goal) that indicates a contradiction (not only the empty clause), instead of applying the rule. In the forth, we define Σ^+-interpretations of a sort-hierarchy and sort predicates in the semantics. By the interpretations, we construct Herbrand Σ^+-models of programs with sort predicates and then prove the soundness and completeness of the linear resolution.

This paper is organized as follows. Section 2 introduces an order-sorted language with sort predicates, sorted signatures and structures. In Section 3, we define logic programming in the language introduced in Section 2 that is based on resolution rules for dealing with sort predicates. We prove the soundness and completeness of the linear resolution in Section 4. Finally, Section 5 gives our conclusion.

2 An order-sorted language with sort predicates

2.1 Syntax

First, we define the syntax of an order-sorted language with sort predicates. In the syntax, Horn clausal forms are defined as the formulas of the language.

Definition 2.1 *The alphabet of an order-sorted language $\mathcal{L}$ with sort predicates contains the following symbols.*

(1) S: an infinite set of sort symbols $s_1, s_2, \ldots$ including the greatest sort $\top$

(2) F_n: an infinite set of n-ary ($n \geq 0$) function symbols $f_1, f_2, \ldots$

(3) P_n: an infinite set of n-ary ($n \geq 0$) predicate symbols $p_1, p_2, \ldots$

(4) V_s: an infinite set of variables $x{:}\, s, y{:}\, s, z{:}\, s, \ldots$ of sort s

(5) $\leftarrow, (,)$: the connective and auxiliary symbols

For all sorts $s \in S - \{\top\}$, the predicates p_s, called *sort predicates*, indexed by the sorts s are introduced, in which sort predicates are unary predicates, i.e., $p_s \in P_1$. The set of sort predicates is denoted by $P_S = \{p_s \mid s \in S - \{\top\}\}$. We assume that the language $\mathcal{L}$ contains sort predicates.

Definition 2.2 (Signatures) *A signature of $\mathcal{L}$ with sort predicates is a tuple $\Sigma = (S, \Omega, \leq)$ such that:*

(1) $(S, \leq)$ is a partially ordered set of sorts.

(2) If $f \in F_n$, then $f{:}\, s_1 \times \cdots \times s_n \rightarrow s \in \Omega$.

(3) If $p \in P_n$, then $p{:}\, s_1 \times \cdots \times s_n \in \Omega$. In particular, for all sort predicates $p_s \in P_S$, $p_s{:}\, \top \in \Omega$.

Next, given a signature Σ of language $\mathcal{L}$ we define expressions: terms, atomic formulas (atoms), goals and clauses.

Definition 2.3 (Terms) *The set $\mathcal{T}_s$ of terms of sort s is defined by the following.*

(1) If $x\colon s \in V_s$, then $x\colon s \in \mathcal{T}_s$.

(2) If $t_1 \in \mathcal{T}_{s_1}, \ldots, t_n \in \mathcal{T}_{s_n}$, $f \in F_n$ and $f\colon s_1 \times \cdots \times s_n \to s \in \Omega$, then $f(t_1, \ldots, t_n)\colon s \in \mathcal{T}_s$.

(3) If $t \in \mathcal{T}_{s'}$ and $s' \leq s$, then $t \in \mathcal{T}_s$.

The set of terms of all sorts is denoted by $\mathcal{T} = \bigcup_{s \in S} \mathcal{T}_s$. The set of variables of all sorts is denoted by $V = \bigcup_{s \in S} V_s$. The function $sort\colon \mathcal{T} \to S$ that assigns to each term its sort is defined by: (i) $sort(x\colon s) = s$ and (ii) if $f \in F_n$ and $f\colon s_1 \times \cdots \times s_n \to s \in \Omega$, then $sort(f(t_1, \ldots, t_n)\colon s) = s$.

Definition 2.4 *The function $Var\colon \mathcal{T} \to 2^V$ is defined by the following.*

(1) $Var(x\colon s) = \{x\colon s\}$,

(2) $Var(c\colon s) = \emptyset$ for $c \in F_0$, and

(3) $Var(f(t_1, \ldots, t_n)\colon s) = \bigcup_{1 \leq i \leq n} Var(t_i)$ for $f \in F_n (n \geq 1)$.

$\mathcal{T}_0 = \{t \in \mathcal{T} | Var(t) = \emptyset\}$ is the set of terms without variables. The terms $t \in \mathcal{T}_0$ without variables are called ground terms. Moreover, the set of ground terms of sort s is denoted by $\mathcal{T}_{0,s} = \mathcal{T}_0 \cap \mathcal{T}_s$.

Definition 2.5 *Given a signature Σ, the set $\mathcal{A}$ of atoms, the set $\mathcal{G}$ of goals and the set $\mathcal{C}$ of clauses are defined by the following.*

(1) If $t_1 \in \mathcal{T}_{s_1}, \ldots, t_n \in \mathcal{T}_{s_n}$, $p \in P_n$ and $p\colon s_1 \times \cdots \times s_n \in \Omega$, then $p(t_1, \ldots, t_n) \in \mathcal{A}$.

(2) If $L_1, \ldots, L_n \in \mathcal{A}$ $(n \geq 0)$, then $\{L_1, \ldots, L_n\} \in \mathcal{G}$.

(3) If $G \in \mathcal{G}$ and $L \in \mathcal{A}$, then $L \leftarrow G \in \mathcal{C}$.

An atom $p_s(t)$ with $p_s \in P_S$ is denoted by $s(t)$ when this will not cause confusion. In (2) of Definition 2.5, $G = \{L_1, \ldots L_n\}$ is said to be the empty goal (denoted $G = \Box$) if $n = 0$. The clauses $L \leftarrow G$ are denoted by $L \leftarrow$ if G is the empty goal.

Example 2.1 *Consider the signature $\Sigma = (S, \Omega, \leq^*)$ of language $\mathcal{L}$, where $\leq^*$ is the reflexive and transitive closure of $\leq$, consisting of*

$$
\begin{array}{rcl}
S & = & \{\ man, student, male\_student, person, \top\ \}, \\
\leq & = & \{\ (man, person), (student, person), \\
 & & \quad (male\_student, man), \\
 & & \quad (male\_student, student), (person, \top)\ \}, \\
\Omega & = & \{p_s\colon \top \mid s \in S\} \cup \\
 & & \{\ studying\colon person,\ \ john\colon \to man\ \}.
\end{array}
$$

We give an example of clauses as follows.

$$
\begin{array}{l}
male\_student(john\colon man) \leftarrow, \\
studying(x\colon person) \leftarrow \{student(x\colon person)\}.
\end{array}
$$

Definition 2.6 *The function $EVar\colon \mathcal{A} \cup \mathcal{G} \cup \mathcal{C} \to 2^V$ is defined by the following.*

(1) $EVar(p(t_1, \ldots, t_n)) = Var(t_1) \cup \cdots \cup Var\ (t_n)$

(2) $EVar(\{L_1, \ldots, L_n\}) = EVar(L_1) \cup \cdots \cup EVar(L_n)$

(3) $EVar(L \leftarrow G) = EVar(L) \cup EVar(G)$

2.2 Semantics

Definition 2.7 (Σ-structures) *Given a signature Σ, a Σ-structure M is a pair (U, I) such that:*

- *U is a non-empty set.*
- *I is a function where*
  - *if $s \in S$, then $I(s) \subseteq U$ (In particular, $I(\top) = U$),*
  - *if $s_i \leq s_j$, then $I(s_i) \subseteq I(s_j)$,*
  - *if $f \in F_n$ and $f\colon s_1 \times \cdots \times s_n \to s \in \Omega$, then $I(f)\colon I(s_1) \times \cdots \times I(s_n) \to I(s)$, and*
  - *if $p \in P_n$ and $p\colon s_1 \times \cdots \times s_n \in \Omega$, then $I(p) \subseteq I(s_1) \times \cdots \times I(s_n)$.*

A Σ-structure $M = (U, I)$ is said to be a Σ^+-structure if the following conditions hold.

- If $s \in S$, then $I(s) \subseteq I(p_s)$.
- If $s_i \leq s_j$, then $I(p_{s_i}) \subseteq I(p_{s_j})$.

The first condition means that the interpretation of sort s affects the interpretation of sort predicate $p_{\ s}$. This leads to that if $d \in I(s)$, then $d \in I(p_s)$. It is required that all sort predicates semantically satisfy at least the condition. Additionally, we can impose the strong condition $I(s) = I(p_{\ s})$. But to avoid reducing the class of models, the weak condition $I(s) \subseteq I(p_{\ s})$ is chosen.

Given a function f, $Dom(f)$ denotes the domain of f. Let V' be a subset of V. A variable assignment on a Σ-structure $M = (U, I)$ is a function $\alpha\colon V' \to U$ where for all $x\colon s \in V'$, $\alpha(x\colon s) \in I(s)$. α is simply called a variable assignment if $Dom(\alpha) = V$. Otherwise (i.e. $Dom(\alpha) \neq V$), it is called a partial variable assignment. The composition of a variable assignment α and a partial variable assignment β is defined by $\alpha\beta = \alpha - \{(x\colon s, \alpha(x\colon s)) \mid x\colon s \in Dom(\beta)\} \cup \beta$. A Σ-interpretation $\mathcal{I}$ is a pair (M, α) of a Σ-structure M and a variable assignment α. The interpretation $(M, \alpha\beta)$ is denoted by $\mathcal{I}\beta$.

Definition 2.8 *Given a Σ-interpretation $\mathcal{I} = (M, \alpha)$, the denotation $[\![\ ]\!]_\alpha\colon \mathcal{T} \to U$ is defined by the following.*

- $[\![x\colon s]\!]_\alpha = \alpha(x\colon s)$
- $[\![c\colon s]\!]_\alpha = I(c)$
- $[\![f(t_1, \ldots, t_n)\colon s]\!]_\alpha = I(f)([\![t_1]\!]_\alpha, \ldots, [\![t_n]\!]_\alpha)$

By the definition above, we define a satisfiability relation $\mathcal{I} \models E$ where $\mathcal{I}$ is a Σ-interpretation and E is a formula.

Definition 2.9 *Let $\mathcal{I} = (M, \alpha)$ be a Σ-interpretation. The Σ-satisfiability relation $\models_\Sigma \subseteq \mathcal{I} \times (\mathcal{A} \cup \mathcal{G} \cup \mathcal{C})$ is defined inductively as follows.*

- *$\mathcal{I} \models_\Sigma p(t_1, \ldots, t_n)$ iff $([\![t_1]\!]_\alpha, \ldots, [\![t_n]\!]_\alpha) \in I(p)$.*
- *$\mathcal{I} \models_\Sigma \{L_1, \ldots, L_n\}$ iff $\mathcal{I} \models_\Sigma L_1$, $\ldots$, $\mathcal{I} \models_\Sigma L_n$.*
- *$\mathcal{I} \models_\Sigma L \leftarrow G$ iff for any partial variable assignment γ such that $Dom(\gamma) = EVar(L \leftarrow G)$, if $\mathcal{I}\gamma \models_\Sigma G$, then $\mathcal{I}\gamma \models_\Sigma L$.*

Let $\mathcal{I}$ be a Σ-interpretation. $\mathcal{I}$ is said to be a Σ-model of Γ (denoted $\mathcal{I} \models_\Sigma \Gamma$) if for every formula $A \in \Gamma$, $\mathcal{I} \models_\Sigma A$. Γ is said to be Σ-satisfiable if Γ has a Σ-model. Otherwise, it said to be Σ-unsatisfiable. A is a consequence of Γ in the class of Σ-structures (denoted $\Gamma \models_\Sigma A$) if every Σ-model of Γ is a Σ-model of A. In particular, $\{B\} \models_\Sigma A$ is denoted by $B \models_\Sigma A$. A Σ-interpretation $\mathcal{I} = (M, \alpha)$ is a Σ^+-interpretation if M is a Σ^+-structure. Its satisfiability relation is called a Σ^+-satisfiability relation $\models_{\Sigma^+}$. Likewise, Σ^+-models, Σ^+-satisfiability, Σ^+-unsatisfiability and consequences in the class of Σ^+-structures can be defined.

3 Logic Programming

Definition 3.1 (Program) *Let Σ be a signature. A program is a finite set $\mathcal{P}_\Sigma \subseteq \mathcal{C}$ of clauses.*

Definition 3.2 (Sorted substitution) *A sorted substitution is a partial function $\theta: V \to \mathcal{T}$ such that:*

- $\theta(x{:}\, s) \in \mathcal{T}_s - \{x{:}\, s\}$,
- $Dom(\theta) \subseteq V$ *is finite.*

By the first condition, a sorted substitution is restricted to a mapping from sorted variables $x{:}\, s$ to sorted terms t ($\in \mathcal{T}_s$), where $\mathcal{T}_s$ includes terms of the sort s of the variable $x{:}\, s$ and its subsorts[2]*. Each sorted substitution can be represented by a finite set $\{x_1{:}\, s_1/t_1, \ldots, x_n{:}\, s_n/t_n\}$.*

Let θ be a sorted substitution. θ is called a sorted ground substitution if for every variable $x{:}\, s \in Dom(\theta)$, $\theta(x{:}\, s)$ is a ground term, i.e., $Var(\theta(x{:}\, s)) = \emptyset$. Let V' be a subset of V. The restriction of a sorted substitution θ to V' is defined by $\theta \upharpoonright V' = \{x{:}\, s/\theta(x{:}\, s) \mid x{:}\, s \in V' \cap Dom(\theta)\}$. θ is a sorted ground substitution for V' if $V' \subseteq Dom(\theta)$ and $\theta\upharpoonright V'$ is a sorted ground substitution. The identity substitution the domain of which is empty is denoted by ϵ.

The sorted substitution is extended to the expressions: terms, atoms, goals and clauses.

Definition 3.3 *Let $E \in \mathcal{T} \cup \mathcal{A} \cup \mathcal{G} \cup \mathcal{C}$. $E\theta$ is defined by the following.*

- $x{:}\, s\theta = \theta(x{:}\, s)$ *if* $x{:}\, s \in Dom(\theta)$,
- $x{:}\, s\theta = x{:}\, s$ *if* $x{:}\, s \notin Dom(\theta)$,
- $f(t_1, \ldots, t_n)\theta = f(t_1\theta, \ldots, t_n\theta)$,
- $p(t_1, \ldots, t_n)\theta = p(t_1\theta, \ldots, t_n\theta)$,
- $\{L_1, \ldots, L_n\}\theta = \{L_1\theta, \ldots, L_n\theta\}$,
- $(L \leftarrow G)\theta = L\theta \leftarrow G\theta$.

Let θ_1, θ_2 be sorted substitutions. The composition $\theta_1\theta_2$ of substitutions θ_1 and θ_2 is defined by $(x{:}\, s)\theta_1\theta_2 = ((x{:}\, s)\theta_1)\theta_2$. θ_2 is more general than θ_1 if there exists a substitution γ such that $\theta_1 = \theta_2\gamma$. Let $E_1, E_2 \in \mathcal{T} \cup \mathcal{A} \cup \mathcal{G} \cup \mathcal{C}$. A sorted substitution θ is a unifier of E_1 and E_2 if $E_1\theta = E_2\theta$. A unifier θ of E_1 and E_2 is *a most general unifier* (mgu) if it is more general than every unifier of E_1 and E_2. In this paper, in order to decide a unique most general unifier, we assume that given sort-hierarchies are meet-semilattices. Namely, for any two sorts in a sort-hierarchy, there exists the greatest lower bound. If a sort-hierarchy is not a meet-semilattice, then a meet-semilattice constructed from it (as shown in [4]) can be obtained.

Lemma 3.1 *Let $\mathcal{I}$ be a Σ-interpretation, $L \leftarrow G$ a clause and θ a sorted substitution. If $\mathcal{I} \models_\Sigma L \leftarrow G$, then $\mathcal{I} \models_\Sigma (L \leftarrow G)\theta$.*

Proof. Suppose $\mathcal{I} \models_\Sigma L \leftarrow G$ where $\mathcal{I} = (M, \alpha)$ is a Σ-interpretation. By Definition 2.9, for any partial variable assignment β such that $Dom(\beta) = EVar(L \leftarrow G)$, if $\mathcal{I}\beta \models_\Sigma G$, then $\mathcal{I}\beta \models_\Sigma L$. On the other hand, let γ be a partial variable assignment defined by $Dom(\gamma) = EVar((L \leftarrow G)\theta)$ and let $\beta(x{:}\, s) = [\![\theta(x{:}\, s)]\!]_{\alpha\gamma}$. Then, $\mathcal{I}\beta \models_\Sigma L \leftarrow G$ iff $\mathcal{I}\gamma \models_\Sigma (L \leftarrow G)\theta$. Hence we have $\mathcal{I} \models_\Sigma (L \leftarrow G)\theta$. ∎

Let θ be a sorted substitution and C a clause. $C\theta$ is called an instance of C. The set of all ground instances of C is denoted by $ground(C)$. We write $ground(\Delta)$ for $\bigcup_{c\in\Delta} ground(C)$.

Let $\mathcal{P}_\Sigma$ be a program and G a goal. Linear resolution is a proof method obtained through applications of the following inference rule to a goal G and clauses in $\mathcal{P}_\Sigma$.

[2] By Definition 2.3, $\mathcal{T}_s$ denotes the set of terms of sort s and its subsorts.

Definition 3.4 (Cut rule) *Let $L' \leftarrow G' \in \mathcal{P}_\Sigma$. If θ is a unifier of $L \in G$ and L', then $(G - \{L\})\theta \cup G'\theta$ is a subgoal of G with respect to L and $L' \leftarrow G'$, written by*

$$G \xrightarrow{\theta}_{R1} (G - \{L\})\theta \cup G'\theta$$

Furthermore, sort predicates allow us to introduce two inference rules that are based on the inference rules of [3] explained in Section 1. The linear resolution version of the rules is given by the following.

Definition 3.5 (Subsort rule) *Let $s(t) \in G$ and $s'(t') \leftarrow G' \in \mathcal{P}_\Sigma$. If $s' \leq s$ and θ is a unifier of t and t', then $(G - \{s(t)\})\theta \cup G'\theta$ is a subgoal of G with respect to $s(t)$ and $s'(t') \leftarrow G'$, written by*

$$G \xrightarrow{\theta}_{R2} (G - \{s(t)\})\theta \cup G'\theta$$

Definition 3.6 (Sort predicate rule) *If $s(t) \in G$ and $sort(t\theta) \leq s$, then $(G - \{s(t)\})\theta$ is a subgoal of G with respect to $s(t)$, written by*

$$G \xrightarrow{\theta}_{R3} (G - \{s(t)\})\theta$$

However, ordinary logic programming has only one inference rule, while the three inference rules above prevent the simplified proof procedure. Thus, we present an inference rule to be integrated. To eliminate the need to use the sort predicate rule, we newly define a successful goal as follows.

Definition 3.7 (Successful goals) *Let $G \in \mathcal{G}$ be a goal. G is called a successful goal if every $L \in G$ is an atom $L = s(t)$ with a sort predicates and there exists an instance t' of t such that $sort(t') \leq s$. The empty goal or a successful goal is denoted by $\perp$.*

Instead of applying the sort predicate rule that eliminates an atom with a sort predicate, the atom is left in a goal. For example, if the subgoal

$$\{s_1(t_1), s_2(t_2)\}$$

where $sort(t_1) \leq s_1$ and $sort(t_2) \leq s_2$ is derived, then this goal is recognized as the success of linear resolution, and then continuing the derivation is forbidden.

In addition, in order to integrate the cut rule and the subsort rule, the function

$$\langle L \rangle = \{L\} \cup \{s'(t) \mid L \equiv s(t) \text{ and } s' \leq s\}$$

is defined. Using this, the cut rule is redesigned to incorporate the subsort rule as follows.

Definition 3.8 (Extended resolution rule) *Let $L \in G$ and $L' \leftarrow G' \in \mathcal{P}_\Sigma$. If θ is a unifier of $L_0 \in \langle L \rangle$ and L', then $(G - \{L\})\theta \cup G'\theta$ is a subgoal of G with respect to L and $L' \leftarrow G'$, written by*

$$G \xrightarrow{\theta}_{R4} (G - \{L\})\theta \cup G'\theta$$

In the following, we define an (unrestricted) derivation of linear resolution obtained by the extended resolution rule.

Definition 3.9 (Linear Resolution) *Let $\mathcal{P}_\Sigma$ be a program. A finite sequence*

$$\mathcal{P}_\Sigma : G_0 \xrightarrow{\theta_1}_{R4} G_1 \xrightarrow{\theta_2}_{R4} \cdots \xrightarrow{\theta_n}_{R4} G_n$$

is called an unrestricted derivation from G_0 in a program $\mathcal{P}_\Sigma$ ($n \geq 0$). The unrestricted derivation is denoted by $\mathcal{P}_\Sigma : G_0 \xrightarrow{\theta} G_n$ where $\theta = \theta_1 \cdots \theta_n$. For $1 \leq i \leq n$, the restriction $\theta_i \uparrow EVar(G_{i-1})$ of substitution θ_i to each goal G_{i-1} is denoted by $\theta_i^\uparrow$. We use the notation $\theta^\uparrow = \theta_1^\uparrow \cdots \theta_n^\uparrow$.

An unrestricted derivation is called a derivation if its unifiers are most general. A derivation $\mathcal{P}_\Sigma: G_0 \xrightarrow{\theta} \!\!\!\!\rightarrow G_n$ is called successful if G_n is a successful goal, i.e., $\mathcal{P}_\Sigma: G_0 \xrightarrow{\theta} \!\!\!\!\rightarrow \bot$. The composition $(\theta_1 \cdots \theta_n)\!\uparrow\! EVar(G_0)$ of the substitutions for the variables in the initial goal G_0 is called a computed answer substitution.

Example 3.1 *We give an example of a derivation over the signature $\Sigma = (S, \Omega, \leq^*)$ of Example 2.1. Let $\mathcal{P}_\Sigma$ be a program given by the following clause set.*

$$\begin{aligned}\mathcal{P}_\Sigma \;=\; & \{\ male\_student(john\colon man) \leftarrow, \\ & studying(x\colon person) \leftarrow \{student(x\colon person)\}\ \}.\end{aligned}$$

A derivation of the goal $\{studying(john\colon man)\}$ is successful as follows.

$$\begin{aligned}\mathcal{P}_\Sigma :\ \{studying(john\colon man)\} \;&\xrightarrow{\theta}_{R4} \\ \{student(john\colon man)\} \;&\xrightarrow{\epsilon}_{R4}\ \bot\end{aligned}$$

where $\theta = \{x\colon person/john\colon man\}$. The first application of the extended resolution rule is a derivation step for the clause $studying(x\colon person) \leftarrow \{student(x\colon person)\}$ from the goal $\{studying(john\colon man)\}$, and the second application is a derivation for the subsort relation $male\_student \leq student$ and the clause $male\_student(john\colon man) \leftarrow$ from the goal $\{student(john\colon man)\}$.

4 The Completeness of Linear Resolution

In order to show the completeness of linear resolution [3], we attach two conditions to derivations defined in Section 3. (1) we assign for each different clause in a program a different variable; And, that these assigned variables are also different to variables assigned to a goal. (2) released variables are never reintroduced.

Lemma 4.1 *Let $\mathcal{I}$ be a Σ-interpretation and L an atom. If $\mathcal{I} \models_\Sigma L \leftarrow$, then $\mathcal{I} \models_\Sigma L$.*

Proof. Suppose $\mathcal{I} \models_\Sigma L \leftarrow$ where $\mathcal{I} = (M, \alpha)$ is a Σ-interpretation. By Definition 2.9, for any partial variable assignment β such that $Dom(\beta) = EVar(L \leftarrow)$, we have $\mathcal{I}\beta \models_\Sigma L$. Let $\beta(x\colon s) = \alpha(x\colon s)$. Then, $\mathcal{I}\beta \models_\Sigma L$ iff $\mathcal{I} \models_\Sigma L$. Therefore, $\mathcal{I} \models_\Sigma L$ can be derived. ∎

Theorem 4.1 (Soundness) *Let $\mathcal{P}_\Sigma$ be a program and G a goal. If $\mathcal{P}_\Sigma: G \xrightarrow{\theta} \!\!\!\!\rightarrow \bot$, then $\mathcal{P}_\Sigma \models_{\Sigma^+} G\theta$.*

Proof. By induction on the length n of linear resolution: If $n = 1$, then there exists a derivation $\mathcal{P}_\Sigma: G(= L) \xrightarrow{\theta}_{R4} \bot$. By Definition 3.8, for a clause $L' \leftarrow \in \mathcal{P}_\Sigma$, we have a unifier θ of $L_0 \in \langle L\rangle$ and L'. Let $\mathcal{I}$ be a Σ^+model of $\mathcal{P}_\Sigma$. By Lemma 3.1, $\mathcal{I} \models_{\Sigma^+} (L' \leftarrow)\theta$. We consider two cases. (i) $L_0 = L$. $\mathcal{I} \models_{\Sigma^+} (L \leftarrow)\theta$ since $L_0\theta = L'\theta$. (ii) $L_0 = s'(t)$. $L = s(t)$, $L' = s'(t')$ and $s' \leq s$. Then $\mathcal{I} \models_{\Sigma^+} (s'(t') \leftarrow)\theta$ (by $L_0\theta = L'\theta$) and $I(s') \subseteq I(s)$ imply $\mathcal{I} \models_{\Sigma^+} (s(t) \leftarrow)\theta$. By Lemma 4.1, $\mathcal{I} \models_{\Sigma^+} L\theta$ holds.

If $n > 1$, then there exists a derivation

$$\begin{aligned}\mathcal{P}_\Sigma: G \xrightarrow{\theta_1}_{R4} (G - \{L\})\theta_1 \cup G'\theta_1 \xrightarrow{\theta_2}_{R4} G_2 \\ \xrightarrow{\theta_3}_{R4} \ \cdots \ \xrightarrow{\theta_n}_{R4} \bot\end{aligned}$$

where $L' \leftarrow G' \in \mathcal{P}_\Sigma$ and $L_0\theta_1 = L'\theta_1 (L_0 \in \langle L\rangle)$. By induction hypothesis, $\mathcal{I} \models_{\Sigma^+} ((G - \{L\})\theta_1 \cup G'\theta_1)\theta'$ where $\theta' = \theta_2 \cdots \theta_n$. This and $\mathcal{I} \models_{\Sigma^+} (L' \leftarrow G')\theta_1\theta'$ show that: (i) if $L_0 = L$, then $\mathcal{I} \models_{\Sigma^+} (L \leftarrow)\theta_1\theta'$ (by $L\theta_1 = L'\theta_1$). (ii) if $L_0 = s'(t)$, then $L = s(t)$, $L' = s'(t')$ and $s' \leq s$. So $\mathcal{I} \models_{\Sigma^+} (s(t) \leftarrow)\theta_1\theta'$ is inferred from $\mathcal{I} \models_{\Sigma^+} (s'(t') \leftarrow)\theta_1$ (by $L_0\theta_1 = L'\theta_1$) and $I(s') \subseteq I(s)$. Hence, we obtain $\mathcal{I} \models_{\Sigma^+} L\theta_1\theta'$ by Lemma 4.1. ∎

We define a Herbrand structure for the order-sorted language to prove the completeness of linear resolution.

[3] The paper [17] shows that the completeness theorem for logic programming in [14] contains an incorrect state for substitutions. Thus, we prove the completeness by the method as in [5].

Definition 4.1 *A Herbrand structure $M_H = (I_H, U_H)$ is a structure such that:*

(1) $U_H = \mathcal{T}_0$,

(2) $I_H(s) = \mathcal{T}_{0,s}$,

(3) if $c \in F_0$ *and* $c\colon \rightarrow s \in \Omega$, *then* $I_H(c) = c\colon s$,

(4) if $f \in F_n$ *and* $f\colon s_1 \times \cdots \times s_n \rightarrow s \in \Omega$, *then* $I_H(f)(t_1, \ldots, t_n) = f(I_H(t_1), \ldots, I_H(t_n))\colon s$, *and*

(5) if $p \in P_n$ *and* $p\colon s_1 \times \cdots \times s_n \in \Omega$, *then* $I_H(p) \subseteq I_H(s_1) \times \cdots \times I_H(s_n)$.

A Herbrand interpretation $\mathcal{I}_H$ is an interpretation such that its structure is a Herbrand structure. By $\mathcal{T}_{0,s} \subseteq \mathcal{T}_{0,s'}$ for $s \leq s'$, it is clear that all Herbrand structures are Σ-structures.

Lemma 4.2 *Let $\mathcal{I}_H$ be a Herbrand interpretation and $L \leftarrow G$ a clause. $\mathcal{I}_H \models_\Sigma L \leftarrow G$ if and only if $\mathcal{I}_H \models_\Sigma ground(L \leftarrow G)$.*

Proof.
($\Rightarrow$) Let $\mathcal{I}_H \models_\Sigma L \leftarrow G$. By Lemma 3.1, for every instance of $L \leftarrow G$, $\mathcal{I}_H \models_\Sigma (L \leftarrow G)\theta (\in ground(L \leftarrow G))$.
($\Leftarrow$) Suppose $\mathcal{I}_H \models_\Sigma ground(L \leftarrow G)$. We have to show that for any partial variable assignment β such that $Dom(\beta) = EVar(L \leftarrow G)$, if $\mathcal{I}_H\beta \models_\Sigma G$, then $\mathcal{I}_H\beta \models_\Sigma L$. We have $I_H(s_i) = \mathcal{T}_{0,s_i}$ (by Definition 4.1), and $(L \leftarrow G)\theta \in ground(L \leftarrow G)$ for any sorted ground substitution θ for $EVar(L \leftarrow G)$. Then we can let $\theta(x_i\colon s_i) = \beta(x_i\colon s_i)$ for $x_i\colon s_i \in EVar(L \leftarrow G)$. By assumption, if $\mathcal{I}_H\beta \models_\Sigma G$, then $\mathcal{I}_H\beta \models_\Sigma L$. ∎

We define a derivation tree of a ground clause C in $\mathcal{P}_\Sigma$ as follows.

Definition 4.2 *Let $\mathcal{P}_\Sigma$ be a program. A derivation tree of a ground clause C in $\mathcal{P}_\Sigma$ is a tree that satisfies the following.*

(1) The root is labeled with C.

(2) Every node is labeled with a ground clause.

(3) Every leaf is labeled with one of the following clauses:

- $L \leftarrow G \in ground(\mathcal{P}_\Sigma)$,
- $s(t) \leftarrow$ *with* $sort(t) = s'$ *and* $s' \leq s$, *and*
- $s(t) \leftarrow s'(t)$ *with* $s' \leq s$.

(4) Every non-leaf node is labeled with a clause $L \leftarrow G_1 \cup G_2$ such that $L \leftarrow G_1 \cup \{L'\}$ and $L' \leftarrow G_2$ are its children.

Definition 4.3 *Let $\mathcal{P}_\Sigma$ be a program. A canonical interpretation $\mathcal{I}_\mathcal{P}$ is a Herbrand interpretation such that:*

$\mathcal{I}_\mathcal{P} \models_{\Sigma^+} L$ iff there exists a derivation tree of a ground clause $L \leftarrow$ in $\mathcal{P}_\Sigma$.

Lemma 4.3 *Let $\mathcal{P}_\Sigma$ be a program. A canonical interpretation $\mathcal{I}_\mathcal{P}$ is a Σ^+-model of $\mathcal{P}_\Sigma$.*

Proof. First, we show that $\mathcal{I}_\mathcal{P}$ is a Σ^+-interpretation. Since $\mathcal{I}_\mathcal{P}$ is a Herbrand interpretation, it is a Σ-interpretation. We will prove $I(s) \subseteq I(p_s)$. Let $t \in I(s) (= \mathcal{T}_{0,s})$. Then there is $s' \leq s$ with $sort(t) = s'$. Hence we have a derivation tree of $s(t) \leftarrow$. By Definition 4.3, $\mathcal{I}_\mathcal{P} \models_{\Sigma^+} s(t)$ and therefore $t \in I(p_s)$. On the other hand, we want to show $I(p_{s'}) \subseteq I(p_s)$ for $s' \leq s$. Let $t \in I(s')$. $\mathcal{I}_\mathcal{P} \models_{\Sigma^+} s'(t)$ yields a derivation tree of $s'(t) \leftarrow$. Moreover, we have a derivation tree of $s(t) \leftarrow s'(t)$. Hence, because a derivation tree of $s(t) \leftarrow$ can be obtained, $\mathcal{I}_\mathcal{P} \models_{\Sigma^+} s(t)$. Therefore, $t \in I(p_s)$.

Next, we prove that $\mathcal{I}_\mathcal{P}$ is a model of $\mathcal{P}_\Sigma$. Let $L \leftarrow G (= \{L_1, \ldots, L_n\}) \in \mathcal{P}_\Sigma$ and let θ be a ground substitution for $EVar(L \leftarrow G)$. If $\mathcal{I}_\mathcal{P} \models_{\Sigma^+} G\theta$, then $\mathcal{I}_\mathcal{P} \models_{\Sigma^+} L_i\theta$. By Definition 4.2, there is a derivation tree of $L_i\theta \leftarrow$. Since $(L \leftarrow G)\theta \in ground(\mathcal{P}_\Sigma)$, we obtain a derivation tree of $(L \leftarrow G)\theta$. Then, a derivation

tree of $L\theta$ can be given by the definition of derivation trees. Then $\mathcal{I}_{\mathcal{P}} \models_{\Sigma^+} L\theta$. Hence $\mathcal{I}_{\mathcal{P}} \models_{\Sigma^+} (L \leftarrow G)\theta$. By Lemma 4.2, $\mathcal{I}_{\mathcal{P}} \models_{\Sigma^+} L \leftarrow G$ is shown.

Since $\mathcal{I}_{\mathcal{P}}$ is a Σ^+-interpretation, the following statements must hold for $s' \leq s$ (by the definition of satisfiability relations).

- $\mathcal{I}_{\mathcal{P}} \models_{\Sigma^+} s(t)$ for $sort(t) = s'$.
- If $\mathcal{I}_{\mathcal{P}} \models_{\Sigma^+} s'(t)$, then $\mathcal{I}_{\mathcal{P}} \models_{\Sigma^+} s(t)$.

It is clear by the proof of that $\mathcal{I}_{\mathcal{P}}$ is a Σ^+-interpretation. ∎

Lemma 4.4 *Let $\mathcal{P}_\Sigma$ be a program. If there exists a derivation tree of a ground clause $L \leftarrow G$ in $\mathcal{P}_\Sigma$, then $\mathcal{P}_\Sigma : G \longrightarrow\!\!\!\!\!\rightarrow \bot$ implies $\mathcal{P}_\Sigma : L \longrightarrow\!\!\!\!\!\rightarrow \bot$.*

Proof. We show the lemma by induction on the height n of a derivation tree of a ground clause $L \leftarrow G$.

$n = 1$: we have to consider the following three cases for $L \leftarrow G$. (i) $L \leftarrow G \in ground(\mathcal{P}_\Sigma)$. If $\mathcal{P}_\Sigma : G \longrightarrow\!\!\!\!\!\rightarrow \bot$, then $\mathcal{P}_\Sigma : L \xrightarrow{\epsilon}_{R4} G \longrightarrow\!\!\!\!\!\rightarrow \bot$. (ii) $s(t) \leftarrow$. $s' \leq s$ and $sort(t) = s$ must be ture. Because $\{s(t)\}$ is a successful goal, $\mathcal{P}_\Sigma : s(t) \longrightarrow\!\!\!\!\!\rightarrow \bot$. (iii) $s(t) \leftarrow s'(t)$. $s' \leq s$ holds. If $\mathcal{P}_\Sigma : s'(t) \longrightarrow\!\!\!\!\!\rightarrow \bot$, then $\mathcal{P}_\Sigma : s(t) \xrightarrow{\epsilon}_{R4} s'(t) \longrightarrow\!\!\!\!\!\rightarrow \bot$.

$n > 1$: by induction hypothesis, if $\mathcal{P}_\Sigma : G_1 \cup \{L'\} \longrightarrow\!\!\!\!\!\rightarrow \bot$, then $\mathcal{P}_\Sigma : L \longrightarrow\!\!\!\!\!\rightarrow \bot$, and if $\mathcal{P}_\Sigma : G_2 \longrightarrow\!\!\!\!\!\rightarrow \bot$, then $\mathcal{P}_\Sigma : L' \longrightarrow\!\!\!\!\!\rightarrow \bot$ where $G = G_1 \cup G_2$. Hence if $\mathcal{P}_\Sigma : G_1 \cup G_2 \longrightarrow\!\!\!\!\!\rightarrow \bot$, then $\mathcal{P}_\Sigma : L \xrightarrow{\epsilon}_{R4} G_1 \cup G_2 \longrightarrow\!\!\!\!\!\rightarrow \bot$. ∎

Theorem 4.2 (Ground completeness) *Let $\mathcal{P}_\Sigma$ be a program and G a ground goal. If $\mathcal{P}_\Sigma \models_{\Sigma^+} G$, then there exists a derivation $\mathcal{P}_\Sigma : G \xrightarrow{\theta} \!\!\!\!\!\rightarrow \bot$.*

Proof. Let $\mathcal{I} \models_{\Sigma^+} \mathcal{P}_\Sigma$. By assumption, $\mathcal{I} \models_{\Sigma^+} G$. Then $\mathcal{I}_{\mathcal{P}} \models_{\Sigma^+} G$ by Lemma 4.3. By Definition 4.3, there exists a ground clause $L \leftarrow$ in $\mathcal{P}_\Sigma$ for all $L \in G$. Hence, $\mathcal{P}_\Sigma : L \longrightarrow\!\!\!\!\!\rightarrow \bot$ by Lemma 4.4. Therefore, we can obtain $\mathcal{P}_\Sigma : G \xrightarrow{\theta} \!\!\!\!\!\rightarrow \bot$. ∎

The next lemma and its proof are based on Lemma 5.33 in [5].

Lemma 4.5 *Let $\mathcal{P}_\Sigma$ be a program. If $\mathcal{P}_\Sigma$ has an unrestricted derivation*

$$\mathcal{P}_\Sigma : G_0 \xrightarrow{\theta_1}_{R4} G_1 \xrightarrow{\theta_2}_{R4} \cdots \xrightarrow{\theta_n}_{R4} G_n$$

where $x : s \in EVar(G_0\theta_1^{\uparrow}\cdots\theta_n^{\uparrow}) - EVar(G_n)$, then for some $G_i, G_j (i < j)$, $x : s \in EVar(G_i\theta_j^{\uparrow}) - EVar(G_j)$.

Proof. We show the lemma on induction of the length n of an unrestricted derivation.

$n = 1$: trivial by assumption.

$n > 1$: for $x : s \in EVar(G_0\theta_1^{\uparrow}\cdots\theta_n^{\uparrow})$, there exists $x' : s' \in EVar(G_0\theta_1^{\uparrow}\cdots\theta_{n-1}^{\uparrow})$ such that $x : s \in EVar(x' : s'\theta_n^{\uparrow})$. We consider two cases. (i) $x' : s' \in EVar(G_{n-1})$. $x : s \in EVar(x' : s'\theta_n^{\uparrow})$ implies $x : s \in EVar(G_{n-1}\theta_n^{\uparrow})$. Hence, $x : s \in EVar(G_{n-1}\theta_n^{\uparrow}) - EVar(G_n)$. (ii) $x' : s' \notin EVar(G_{n-1})$. By induction hypothesis, if $x : s \in EVar(G_0\theta_1^{\uparrow}\cdots\theta_{n-1}^{\uparrow})$, then for some $G_i, G_j (i < j)$, we have $x' : s' \in EVar(G_i\theta_j^{\uparrow}) - EVar(G_j)$. Therefore, $x' : s' = x' : s'\theta_n^{\uparrow}$ (by $\theta_n^{\uparrow} = \theta_n \uparrow EVar(G_{n-1})$) implies $x : s = x' : s'$ $(\in EVar(G_0\theta_1^{\uparrow}\cdots\theta_{n-1}^{\uparrow}\theta_n^{\uparrow}))$. ∎

Lemma 4.6 (Lifting lemma) *Let $\mathcal{P}_\Sigma$ be a program. If $\mathcal{P}_\Sigma$ has an unrestricted derivation*

$$\mathcal{P}_\Sigma : G_0\theta_0 \xrightarrow{\theta_1}_{R4} G_1 \xrightarrow{\theta_2}_{R4} \cdots \xrightarrow{\theta_n}_{R4} G_n,$$

then $\mathcal{P}_\Sigma$ has a derivation

$$\mathcal{P}_\Sigma : G_0 \xrightarrow{\theta'_1}_{R4} G'_1 \xrightarrow{\theta'_2}_{R4} \cdots \xrightarrow{\theta'_n}_{R4} G'_n$$

and the following holds.

(i) $\gamma_0 = \theta_0$, and for $1 \leq i \leq n$, $(\gamma_{i-1} \uparrow EVar(G_{i-1}))\theta_i = \theta'_i\gamma_i$ and $G_i = G'_i\gamma_i$, and

(ii) there exists a substitution γ'_n with $G_0\theta_0\theta_1^{\uparrow}\cdots\theta_n^{\uparrow} = G_0\theta'^{\uparrow}_1\cdots\theta'^{\uparrow}_n\gamma'_n$.

Proof. We show the lemma on induction of the length n of a derivation.

$n = 1$: there exists a derivation $\mathcal{P}_\Sigma : G_0\theta_0 \xrightarrow{\theta_1}_{R4} G_1$ where $L\theta_0 \in G_0\theta_0$, $L' \leftarrow G' \in \mathcal{P}_\Sigma$ and θ_1 is a unifier of $L_0\theta_0 \in \langle L\theta_0\rangle$ and L'. $EVar(G_0) \cap EVar(L' \leftarrow G') = \emptyset$ implies $(L' \leftarrow G')\theta_0{\uparrow}EVar(G_0) = L' \leftarrow G'$. Hence, $\mathcal{P}_\Sigma : G_0 \overset{(\theta_0 \uparrow EVar(G_0))\theta_1}{\longrightarrow}_{R4} G_1$. On the other hand, let θ'_1 be a mgu of L_0 and L'. It follows $\mathcal{P}_\Sigma : G_0 \xrightarrow{\theta'_1}_{R4} G'_1$ with $(\gamma_0{\uparrow}EVar(G_0))\theta_1 = \theta'_1\gamma_1$ and $G_1 = G'_1\gamma_1$.

Moreover, we will prove $G_0\theta_0\theta_1{\uparrow} = G_0\theta'_1{\uparrow}\gamma'_1$. By $(\theta_0 \uparrow EVar(G_0))\theta_1 = \theta'_1\gamma_1$, we have $G_0\theta_0\theta_1 = G_0\theta'_1\gamma_1$. Hence, $G_0\theta_0\theta_1{\uparrow} = G_0\theta'_1{\uparrow}\gamma_1$.

$n > 1$: by induction hypothesis, we have a derivation

$$\mathcal{P}_\Sigma : G_0 \xrightarrow{\theta'_1}_{R4} G'_1 \xrightarrow{\theta'_2}_{R4} \cdots \xrightarrow{\theta'_{n-1}}_{R4} G'_{n-1}$$

where $\gamma_0 = \theta_0$, and for $1 \le i \le n-1$, there exists a substitution γ_i such that $(\gamma_{i-1}{\uparrow}EVar(G_{i-1}))\theta_i = \theta'_i\gamma_i$ and $G_i = G'_i\gamma_i$. By assumption, there exists an unrestricted derivation $\mathcal{P}_\Sigma : G_{n-1} \xrightarrow{\theta_n}_{R4} G_n$ $(G_{n-1} = G'_{n-1}\gamma_{n-1})$ where $L \in G_{n-1}$, $L' \leftarrow G' \in \mathcal{P}_\Sigma$ and θ_n is a unifier of $L_0 \in \langle L\rangle$ and L'. $EVar(G'_{n-1}) \cap EVar(L' \leftarrow G') = \emptyset$ implies $(L' \leftarrow G')\gamma_{n-1}{\uparrow}EVar(G'_{n-1}) = L' \leftarrow G'$. Then, $\mathcal{P}_\Sigma : G'_{n-1} \overset{(\gamma_{n-1} \uparrow EVar(G'_{n-1}))\theta_n}{\longrightarrow}_{R4} G_n$. On the other hand, let θ'_n be a mgu of L_0 and L'. This infers $\mathcal{P}_\Sigma : G'_{n-1} \xrightarrow{\theta'_n}_{R4} G'_n$ where $(\gamma_{n-1}{\uparrow}EVar(G_{n-1}))\theta_n = \theta'_n\gamma_n$ and $G_n = G'_n\gamma_n$.

In addition, $G_0\theta_0\theta_1{\uparrow}\cdots\theta_n{\uparrow} = G_0\theta'_1{\uparrow}\cdots\theta'_n{\uparrow}\gamma'_n$ will be shown. Let γ'_i be defined by extending γ_i. If $x{:}\,s \in EVar(G_j\theta_k{\uparrow}) - EVar(G_k)$ for some $G_j, G_k (1 \le j < k \le n)$, then let $x{:}\,s\gamma'_i = x{:}\,s\gamma'_{i-1}\theta_i{\uparrow}$. If $x{:}\,s \in EVar(G'_{i-1}\theta'_i) \cup EVar(G'_i)$ with $G'_0 = G_0$, then let $x{:}\,s\gamma'_i = x{:}\,s\gamma_i$. We will show $x{:}\,s\theta'_n{\uparrow}\gamma'_n = x{:}\,s\gamma'_{n-1}\theta_n{\uparrow}$ in two cases. (a) $x{:}\,s \in EVar(G'_{n-1})$. By the definition of γ'_i and $(\gamma_{n-1} \uparrow EVar(G_{n-1}))\theta_n = \theta'_n\gamma_n$, we have $x{:}\,s\theta'_n{\uparrow}\gamma'_n = x{:}\,s\theta'_n{\uparrow}\gamma_n = x{:}\,s\gamma_{n-1}\theta_n{\uparrow} = x{:}\,s\gamma'_{n-1}\theta_n{\uparrow}$. (b) $x{:}\,s \in EVar(G_j\theta_k{\uparrow}) - EVar(G_k)$ for some $G_j, G_k (1 \le j < k \le n)$. $x{:}\,s\theta'_n{\uparrow} = x{:}\,s$ derives $x{:}\,s\theta'_n{\uparrow}\gamma'_n = x{:}\,s\gamma'_n = x{:}\,s\gamma'_{n-1}\theta_n{\uparrow}$. Now we consider $y{:}\,s' \in EVar(G_0)$ and $y{:}\,s'\theta'_1{\uparrow}\cdots\theta'_{n-1}{\uparrow} = t$. By Lemma 4.5, we can say that $x{:}\,s$ in $Var(t)$ is (a) or (b). Therefore, $y{:}\,s'\theta_0\theta_1{\uparrow}\cdots\theta_n{\uparrow} =_{I.H.} y{:}\,s'\theta'_1{\uparrow}\cdots\theta'_{n-1}{\uparrow}\gamma'_{n-1}\theta_n{\uparrow} = t\gamma'_{n-1}\theta_n{\uparrow} = t\theta'_n{\uparrow}\gamma'_n = y{:}\,s'\theta'_1{\uparrow}\cdots\theta'_n{\uparrow}\gamma'_n$ is proved. ∎

Theorem 4.3 (Completeness) *Let $\mathcal{P}_\Sigma$ be a program, G a goal and θ a sorted substitution. If $\mathcal{P}_\Sigma \models_{\Sigma^+} G\theta$, then there exists a successful derivation $\mathcal{P}_\Sigma : G \xrightarrow{\theta'}\!\!\!\!\!\rightarrow \bot$ such that $G\theta = G\theta'{\uparrow}\gamma$.*

Proof. Let $c_i{:}\,s_i$ be a new constant for $x_i{:}\,s_i \in EVar(G\theta)$. Let β be a sorted substitution defined by $Dom(\beta) = EVar(G\theta)$ and $\beta(x_i{:}\,s_i) = c_i{:}\,s_i$. $\mathcal{P}_\Sigma \models_{\Sigma^+} G\theta$ implies $\mathcal{P}_\Sigma \models_{\Sigma^+} G\theta\beta$. By Theorem 4.2, we have a derivation $\mathcal{P}_\Sigma : G\theta\beta \xrightarrow{\delta}\!\!\!\!\!\rightarrow \bot$. Lemma 4.6 yields a derivation $\mathcal{P}_\Sigma : G \xrightarrow{\theta'}\!\!\!\!\!\rightarrow \bot$ with $G\theta\beta\delta{\uparrow} = G\theta'{\uparrow}\gamma$. Since $G\theta\beta$ has no variables, $G\theta\beta = G\theta'{\uparrow}\gamma$. $G\theta'{\uparrow}$ must contain no new constants $c_i{:}\,s_i$, and hence $\gamma(y_i{:}\,s_i) = c_i{:}\,s_i$ holds for a variable $y_i{:}\,s_i$ (the sort of which is the same as of $x_i{:}\,s_i$). Let γ_0 be defined by $(y_i{:}\,s_i)\gamma_0 = x_i{:}\,s_i$ and let $\gamma' = \{(a, b) \in \gamma \mid a \ne y_i{:}\,s_i\} \cup \gamma_0$. Therefore, we obtain $G\theta = G\theta'{\uparrow}\gamma'$. ∎

5 Conclusion

This paper has formalized an order-sorted logic programming language with sort predicates, in which each sort can be used as a unary predicate and that contains a complete inference system. Based on the resolution system with sort predicates developed by Beierle et al., we have presented a practical inference method that is obtained by linear resolution (from the techniques in logic programming) with sort predicates. Specifically, inserting the inference rules relative to subsorts and sort predicates to linear resolution provides the complete inference system as an extension to refutation, in which the empty goal or a successful goal is derived from a goal. The completeness is proved by using the fact that a derivation tree from a program is used to build a canonical model as a Σ^+-model. The semantics corresponding to the extended inference system is defined by the fact that the interpretation of a subsort relation influences the satisfiability of formulas with sort predicates.

Related to this research, logic programming languages with sort expressions have been studied from the viewpoint of knowledge representation. One of the languages is the logic programming language LOGIN [1], in which a complicated term (called ψ-term) allows for a kind of sort expression (called partially ordered type structures) that is ordered and with feature structures. In the approach, sort expressions in ψ-terms can describe what are represented ordinarily by predicates (e.g. $person(x)$), and then these reduce reasoning steps. But, it deals with neither sort predicates for representing knowledge nor the combination of predicates and sorts unlike our work. As another approach, Kakuta et al. [10] used order-sorted logic programming as an application to legal reasoning. This does not either discuss knowledge representation for sort predicates.

Although this paper has attached importance to practical aspects by adopting the proof procedure of logic programming, the results are rigorously theoretical. Recently, to theoretically show the soundness of programming languages, it has been considered that the formalization of programming languages is essential. For example, the compiler of the functional programming language ML was implemented on the basis of the theoretical results. Similar to this, the results of this paper can formally guarantee the soundness and completeness when we implement a logic programming language with sort predicates.

References

[1] H. Aït-Kaci and R. Nasr. LOGIN: A logic programming language with built-in inheritance. *Journal of Logic Programming*, pages 185–215, 1986.

[2] H. Aït-Kaci and A. Podelski. Towards a meaning of LIFE. *Journal of Logic Programming*, pages 195–234, 1993.

[3] C. Beierle, U. Hedtsuck, U. Pletat, P.H. Schmitt, and J. Siekmann. An order-sorted logic for knowledge representation systems. *Artificial Intelligence*, 55:149–191, 1992.

[4] A. G. Cohn. Taxonomic reasoning with many sorted logics. *Artificial Intelligence Review*, 3:89–128, 1989.

[5] K. Doets. *From Logic to Logic Programming*. The MIT Press, 1994.

[6] A. M. Frisch. A general framework for sorted deduction: Fundamental results on hybrid reasoning. In *In Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, 1989.

[7] M. Hanus. Logic programming with type specifications. In F. Pfenning, editor, *Types in Logic Programming*. The MIT Press, 1992.

[8] J. Herbrand. In W. D. Goldfarb, editor, *Logical Writings*. Harvard University Press, 1971.

[9] P. M. Hill and R. W. Topor. A semantics for typed logic programs. In F. Pfenning, editor, *Types in Logic Programming*. The MIT Press, 1992.

[10] T. Kakuta, M. Haraguchi, and Y. Okubo. A goal-dependent abstraction for legal reasoning by analogy. *International Journal of Artificial Intelligence and Law*, 5(1-2):97–118, 1997.

[11] K. Kaneiwa and S. Tojo. Event, property, and hierarchy in order-sorted logic. In *Proceedings of the 1999 Int. Conf. on Logic Programming*, pages 94–108. The MIT Press, 1999.

[12] K. Kaneiwa and S. Tojo. An order-sorted resolution with implicitly negative sorts. In *Proceedings of the 2001 Int. Conf. on Logic Programming*, pages 300–314. Springer-Verlag, 2001. LNCS 2237.

[13] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *J. ACM*, 42(4):741–843, 1995.

[14] J. W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 1987.

[15] A. Oberschelp. Untersuchungen zur mehrsortigen quantorelogik. *Mathematische Annalen 145*, pages 297–333, 1962.

[16] M. Schmidt-Schauss. *Computational Aspects of an Order-Sorted Logic with Term Declarations*. Springer-Verlag, 1989.

[17] J. C. Shepherdson. The role of standardising apart in logic programming. *Theoretical Computer Science*, 129(1):143–166, 1994.

[18] C. Walther. *A Many-Sorted Calculus Based on Resolution and Paramodulation*. Pitman and Kaufman Publishers, 1987.

[19] C. Walther. Many-sorted unification. *Journal of the Association for Computing Machinery*, 35:1, 1988.

[20] T. Weibel. An order-sorted resolution in theory and practice. *Theoretical Computer Science*, 185(2):393–410, 1997.