Order-Sorted Logic Programming with Predicate Hierarchy

Ken Kaneiwa

National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan

Abstract

Order-sorted logic has been formalized as first-order logic with sorted terms where sorts are ordered to build a hierarchy (called a sort-hierarchy). These sorted logics lead to useful expressions and inference methods for structural knowledge that ordinary first-order logic lacks. Nitta et al. pointed out that for legal reasoning a sort-hierarchy (or a sorted term) is not sufficient to describe structural knowledge for event assertions, which express facts caused at some particular time and place. The event assertions are represented by predicates with n arguments (i.e. n-ary predicates), and then a particular kind of hierarchy (called a predicate hierarchy) is built by a relationship among the predicates. To deal with such a predicate hierarchy, which is more intricate than a sort-hierarchy, Nitta et al. implemented a typed (sorted) logic programming language extended to include a hierarchy of verbal concepts (corresponding to predicates). However, the inference system lacks a theoretical foundation because its hierarchical expressions exceed the formalization of order-sorted logic. In this paper, we formalize a logic programming language with not only a sort-hierarchy but also a predicate hierarchy. This language can derive general and concrete expressions in the two kinds of hierarchies. For the hierarchical reasoning of predicates, we propose a manipulation of arguments in which surplus and missing arguments in derived predicates are eliminated and supplemented. As discussed by Allen, McDermott and Shoham in research on temporal logic and as applied by Nitta et al. to legal reasoning, if each predicate is interpreted as an event or action (not as a static property), then missing arguments should be supplemented by existential terms in the argument manipulation. Based on this, we develop a Horn clause resolution system extended to add inference rules of predicate hierarchies. With a semantic model restricted by interpreting a predicate hierarchy, the soundness and completeness of the Horn-clause resolution is proven.

Key words: Order-sorted logic, Predicate hierarchy, Typed logic programming, Linear resolution, Knowledge representation

1 Introduction

In the field of artificial intelligence, order-sorted logics have been studied as formal knowledge representation languages for handling structural knowledge, such as the classification of objects [11]. These logics incorporate sort symbols, which index subsets of the universe and are ordered to build a hierarchy (called a sort-hierarchy). In addition, logical deduction systems with sorted expressions can be regarded as useful tools from the viewpoint of efficient and rational reasoning about structural knowledge.

Following many-sorted logic in Herbrand's thesis [17], several many-sorted systems [42,13,15,26] have been formalized as a generalized first-order logic with different sorts as classes (e.g. points, lines, and planes in geometry) of individuals (but without subsorts, namely all sorts are disjoint). Moreover, many-sorted logic with a sort-hierarchy or subsorts is called order-sorted logic [31,32]. Walther and Cohn separately developed an order-sorted calculus [39,40,10] based on a resolution by a sorted unification algorithm with a sort-hierarchy. Since then, order-sorted logics have been extended to design more expressive languages or efficient deduction systems [41,8,14,33,43,23,21]. The researchers are mainly concerned with order-sorted unification that solves the problem of finding the most general unifier of sorted terms depending on the structure (e.g. lattice) of the sort-hierarchy. In related work, typed logic programming [18,16] with polymorphic types has been developed.

On the other hand, with regard to work that actually implements a deductive language with types (or sorts), the logic programming languages LOGIN [2] and LIFE [3] were proposed, in which ψ-terms together with feature structures [9], which can describe complicated classes of objects, were introduced. Smolka proposed Feature Logic [35] to generalize ψ-terms by adding negation and quantification. Alternatively, F-logic [24] and $\mathcal{QUIXOTE}$ [44,45] were developed as object-oriented deductive languages with the notions of objects, classes, subclasses and property inheritance [37,5] derived from the object-oriented programming paradigm.

For practical knowledge representation and reasoning such as legal reasoning, Nitta et al. [29,30] pointed out that a sort-hierarchy (or a sorted term) is not sufficient to describe structural knowledge for event assertions, which express facts caused at some particular time and place. The event assertions are represented by predicates with n arguments (i.e. n-ary predicates), and then a particular kind of hierarchy (called a predicate hierarchy) is built by a relationship among the predicates. To deal with such a predicate hierarchy, which is more intricate than a sort-hierarchy, Nitta et al. implemented a

Email address: kaneiwa@nii.ac.jp (Ken Kaneiwa).
URL: http://research.nii.ac.jp/~kaneiwa/ (Ken Kaneiwa).

typed (sorted) logic programming language extended for developing the legal reasoning system New HELIC-II [29,30]. The language contains a typed term (called an H-term) classified into a verb-type or a noun-type that is obtained by extending a ψ-term in LOGIN. The verb-types and noun-types can build two separated hierarchies corresponding to the hierarchies of predicates and sorts. However, the inference system lacks a theoretical foundation because its hierarchical expressions exceed the formalization of order-sorted logic.

To provide a theoretical formalization of their work, this paper extends an order-sorted logic by incorporating a predicate hierarchy and its reasoning. This extended logic is theoretically formalized as an order-sorted logic programming language that is based on order-sorted resolution and typed logic programming. In standard logic programming, a way to deal with the hierarchical relationship between unary predicates is to build a class-hierarchy (by a set of formulas of the logical implication form "$p(x) \rightarrow q(x)$" in which a predicate q has more abstract meaning than a predicate p). This form is simple and can be used to express IS-A relations. As presented in [37], IS-A relations (e.g. $Elephant(x) \rightarrow GrayThing(x)$) and IS-NOT-A relations (e.g. $RoyalElephant(x) \rightarrow \neg GrayThing(x)$) construct inheritance networks on which properties are inherited only along the IS-A relations.

However, in order to represent event assertions and their hierarchical relationship (as described in New-HELIC-II), we are required to deal with a hierarchy consisting of n-ary predicates used for describing event assertions (called event predicates) which might have different arities and are distinguished from predicates used for property assertions. For instance, an event assertion $act\_of\_violence(John)$ (or $act\_of\_violence(John,\ e_1)$ with an event identifier e_1) implies a more abstract assertion $illegal\_act(John, @)$ (or $illegal\_act(John, @, e_1)$), where @ is a missing argument, by using a hierarchical relationship of the unary predicate $act\_of\_violence$ and the binary predicate $illegal\_act$. For such event predicates $p, q, r, s, t, \ldots$, is it possible to precisely describe all the hierarchical relationships between them using the implication forms $p(x) \rightarrow q(x)$, $q(x) \rightarrow r(x, y)$, $r(x, y) \rightarrow s(x, z)$, $s(x, y) \rightarrow t(x, z, v)$, $\ldots$? Do these forms infer our desired results by an application of ordinary inference rules (or resolution rules)? Such an ad-hoc method seems not to give us general and flexible reasoning from hierarchical predicates with various arguments. This is because (i) the above does not manipulate the difference between argument structures in predicates as event assertions, and (ii) arguments without denoted roles might result in an incorrect connection between predicates. For example, the logical implication form $act\_of\_violence(x) \rightarrow illegal\_act(x, y)$ does not lead to the operation that a missing argument in $act\_of\_violence$ is substituted with y that is existential and means a person. Moreover, $steal(x, y) \rightarrow illegal\_act(y)$ results in the incorrect connection that if x stole y, then y committed an illegal act.

In this paper we formalize a logic programming language with not only a sort-hierarchy but also a predicate hierarchy. This language can derive general and concrete expressions in the two kinds of hierarchies. A manipulation of arguments is proposed for generating hierarchical reasoning of predicates. In order to derive predicates in the hierarchy, surplus and missing arguments in the derived predicates are eliminated and supplemented. Moreover, the manipulation of arguments can be specified by distinguishing event predicates from property predicates, as was introduced in [22]. The notion of events and properties is based on works [6,28,34] dealing with temporal reasoning (i.e. taking into account the various temporal aspects of propositions). In [6], Allen distinguished between event, property and process in English sentences, and so did McDermott [28] between fact and event. In contrast to work regarding temporal reasoning, Kaneiwa and Tojo [22] introduced the new and entirely original idea that event and property assertions respectively afford different quantification to implicit objects in the real world, not only to spatio-temporal objects. Based on this idea, if each predicate is interpreted as an event or action (not as a static property), then missing arguments should be supplemented by existential terms in the argument manipulation. With this manipulation, we present a Horn clause resolution system extended to add inference rules of predicate hierarchies.

This paper is organized as follows. We start in Section 2 with an introduction to the basic notions of order-sorted logic (logic with sort-hierarchy). In Section 3, we present a motivation to extend order-sorted logic programming, and discuss reasoning upon structural knowledge that is derived from a predicate hierarchy. Section 4 proposes an order-sorted logic incorporating both sort and predicate hierarchies. We define the syntax and the semantics of the proposed logic. In Section 5, we develop a Horn clause resolution system for the hierarchical reasoning of predicates, and prove the soundness and completeness of the system. In Section 7, we give our conclusion and discuss future work.

2 Preliminaries

In this section we will introduce the notions of order-sorted logic [36]: *sort-hierarchy, sorted variables and sorted terms.* $\mathcal{S}$ denotes a set $\{s_1, s_2, \ldots\}$ of *sorts* where each sort indexes a class of individuals. A subsort declaration for $\mathcal{S}$ is an ordered pair (s_i, s_j) of sorts (denoted by $s_i \sqsubset_S s_j$). A $\sqsubset_S$ path from s to s' is a finite sequence $x_0, x_1, \ldots, x_n$ in $\mathcal{S}$ such that $x_0 = s,\ x_n = s'$ and, for $1 \leq i \leq n$, $x_{i-1} \sqsubset_S x_i$. We denote by $\leq_S$ the reflexive and transitive closure of $\sqsubset_S$. That is,

$$s \leq_S s' \Leftrightarrow s = s',\ \textit{or}$$

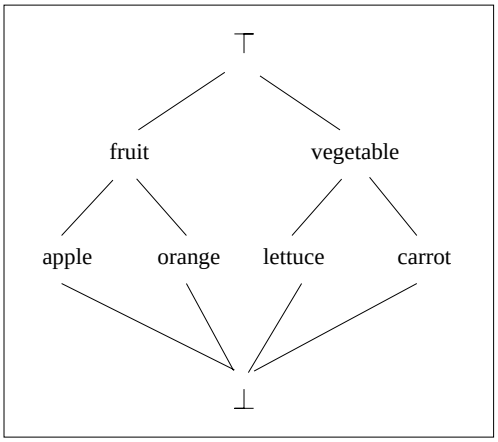

Fig. 1. A sort-hierarchy

$$there\ exists\ a\ \sqsubset_S path\ from\ s\ to\ s'.$$

A sort-hierarchy is an ordered pair $(\mathcal{S}, \leq_S)$ where $\mathcal{S}$ is a set of sorts (containing the greatest sort $\top$ and the least sort $\bot$) and $\leq_S$ is a reflexive and transitive subsort relation. A sort s is a *lower bound* of s_1 and s_2 if $s \leq_S s_1$ and $s \leq_S s_2$. A sort s is a *greatest lower bound* of s_1 and s_2 if s is a lower bound of s_1 and s_2 and $s' \leq_S s$ holds for all lower bounds s' of s_1 and s_2, written as $s = glb(s_1, s_2)$. $(\mathcal{S}, \leq_S)$ is called a *lower semi-lattice* if $glb(s_1, s_2)$ exists for all $s_1, s_2 \in \mathcal{S}$.

For example, the following subsort declarations express that the sorts *apple* and *orange* are subsorts of $fruit$.

$$apple \sqsubset_S fruit,$$
$$orange \sqsubset_S fruit.$$

By adding subsort declarations to the above declarations, the sort-hierarchy in Fig. 1 can be built.

A variable x of sort s (called a *sorted variable*) whose domain is restricted (i.e. a subset of the universe) is written as $x\colon s$. A sort declaration (called a *function declaration*) of an n-ary function f is denoted by $f\colon \langle s_1, \ldots, s_n, s\rangle$ with $s_1, \ldots, s_n, s \in \mathcal{S}$. In particular, a sort declaration of a constant c (i.e. a 0-ary function) is denoted by $c\colon \langle s\rangle$. A sort declaration (called a *predicate declaration*) of an n-ary predicate p is denoted by $p\colon \langle s_1, \ldots, s_n\rangle$ with $s_1, \ldots, s_n \in \mathcal{S}$.

A term of the form $f(t_1, \ldots, t_n)$ is of sort s if the function declaration is $f\colon \langle s_1, \ldots, s_n, s\rangle$. We denote the term $f(t_1, \ldots, t_n)$ of sort s by $f(t_1, \ldots, t_n)\colon s$ where f is an n-ary function with the function declaration $f\colon \langle s_1, \ldots, s_n, s\rangle$ and $t_1, \ldots, t_n$ are terms of $s_1, \ldots, s_n$. In particular, we write $c\colon s$ for the term c of sort s where c is a constant with the function declaration $c\colon \langle s\rangle$. A term restricted by a sort s is called a *sorted term*. For any sorted term t, the function $Sort(t)$ denotes its sort to term t.

3 Motivation

We will discuss desired reasoning in a predicate hierarchy which informally specifies an expressive logic programming language extended to include both sort and predicate hierarchies. Similar to a sort hierarchy, a predicate hierarchy is built by a partial order over n-ary predicates that represents a relationship between general and specific predicates. Each predicate has a fixed argument structure defined by the predicate declaration (as explained in Section 2). In the hierarchy, general predicates can be derived from more specific predicates. This derivation is based upon the fact that specific assertions imply less informative assertions, for example, "John committed an act of violence against Mary" implies "John committed an illegal act against someone."

Given the hierarchy of predicates in Fig. 2, the following results (answered by a query system in which `yes` or `no` must be returned [1]) are conceivable in a sorted logic programming language.

Example 3.1 *Consider the case where the predicate declarations of the predicates* `act_of_violence` *and* `illegal_act` *are given as follows, i.e., the argument structures* [2] *.*

```
act_of_violence: <person>
illegal_act: <person,person>
```

If the fact `act_of_violence(john:man)` *holds, then the superordinate predicate* `illegal_act` *can be derived from the predicate* `act_of_violence` *in direction (1) of Fig. 2. However, the first query "did John commit an illegal act against Mary?" expressed by* `?-illegal_act(john:man, mary:woman)` *should give the answer* `no`*. It is certain that John committed an act of violence against someone but not certain that John did it against Mary. Thus, the second query "did John commit an illegal act against someone?" expressed by* `?-illegal_act(john:man, Y:person)` *should yield* `yes`*.*

```
act_of_violence(john:man).
?-illegal_act(john:man, mary:woman).
```

[1] In standard logic programming, the closed world assumption holds. Thus, if a query formula A cannot be proved from a program, then it is regarded as false, i.e., the answer is no.

[2] As defined in Section 2, order-sorted logic (even first-order logic) contains predicate symbols whose arities are fixed in the signature of an order-sorted language. If the arities of predicates are not fixed, then the meaning of the predicates cannot be uniquely defined in the semantics. Furthermore, the rigorous definition of the sorts and arities of predicates lead to the advantage of excluding type errors and ill-argumented formulas.

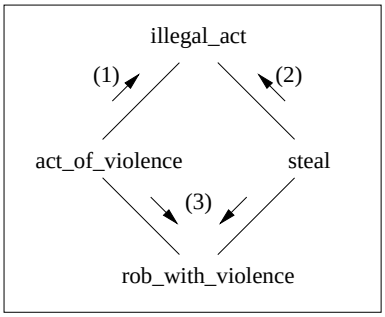

Fig. 2. A hierarchy of predicates

```
no.
?-illegal_act(john:man, Y:person).
yes.
```

This exemplifies the case that the derived predicate `illegal_act` *is higher in the hierarchy and has more arguments than the predicate* `act_of_violence`*. To generate the inference above, we must supplement the second argument (whose role is person and which exists in* `illegal_act` *but does not exist in* `act_of_violence`*) to the fact* `act_of_violence(john:man)`*. In addition, we must take into account of the quantification of its supplemented argument. If this missing argument is universally quantified by all persons, then the meaning of the fact is changed. Therefore because this quantification does not fit with what we expect (i.e. human reasoning), the supplemented argument should be existentially quantified by a person (someone).*

Example 3.2 *Consider the case where the predicate declarations of the predicates* `illegal_act` *and* `steal` *are given as follows.*

```
illegal_act: <person>
steal: <person,thing>
```

The arguments of the predicate `illegal_act` *are fewer than the arguments in the fact* `steal(john:man, c1:wallet)`*, and thus the following query should result in* `yes` *from direction (2) in Fig. 2.*

```
steal(john:man, c1:wallet).
?-illegal_act(john:man).
yes.
?-illegal_act(c1:wallet).
no.
```

Since `steal(john:man, c1:wallet)` *implies* `illegal_act(john:man)` *in a broad sense, the answer to the first query is plausible enough. Namely, the predicate* `illegal_act` *is more general than the predicate* `steal`*, and the single argument* `john:man` *is less informative than the two arguments* `john:man` *and*

`c1:wallet`. *However, the answer to the second query with one argument should be* `no` *since the assertion* `illegal_act(c1:wallet)` *does not make sense where* `c1:wallet` *does not follow the predicate declaration of* `illegal_act`.

Example 3.3 *Furthermore, reasoning in the predicate hierarchy can be extended by identifying each event in assertions. In this example, predicates express event assertions but do not contribute anything to the determination of whether events in these assertions are independent or they happened simultaneously. Denoting each event identity as an argument in predicate formulas makes it explicit. If two event assertions represent one event denoted by the same event identifier*[3] *, then predicates representing them can derive a specific predicate in a hierarchy. In the following example, we consider the case where the predicate declarations of* `illegal_act`, `act_of_violence`, `steal` *and* `rob_with_violence` *are given with an event identity as an argument.*

```
illegal_act: <person,event>
act_of_violence: <person,person,event>
steal: <person,thing,event>
rob_with_violence: <person,person,thing,event>
```

As shown by (3) of Fig. 2, the conjunction of `act_of_violence` *and* `steal` *yields the predicate* `rob_with_violence`. *If* e_1 *identifies an event, then the facts* `act_of_violence(john:man, mary:woman, e1)` *and* `steal(john:man, c1:wallet, e1)` *in an incident imply John's robbing with violence at* e_1. *Hence, the query "did John steal Mary's wallet using robbery with violence at* e_1*?" should yield* `yes`.

```
act_of_violence(john:man, mary:woman, e1).
steal(john:man, c1:wallet, e1).
?-rob_with_violence(john:man, mary:woman, c1:wallet, e1).
yes.
```

To derive this answer, adequate machinery for hierarchical reasoning of predicates such that the arguments of the two predicates `act_of_violence` *and* `steal` *are mixed to produce the arguments of the predicate* `rob_with_violence` *is indispensable.*

Meanwhile, the queries "did John steal a watch from Mary using robbery with violence at e_1*?" and "did John steal anything from Tom using robbery with violence at* e_1*?" yield* `no`, *even if the fact* `steal(john:man, c2:watch, e2)` *is added.*

```
steal(john:man, c2:watch, e2).
```

[3] Each event identifier can be used to denote one event which consists of several event assertions described by predicates.

```
?-rob_with_violence(john:man, mary:woman, c2:watch, e1).
no.
?-rob_with_violence(john:man, tom:man, X, e1).
no.
```

These examples show inferences using a predicate hierarchy. The inferences are consistent with natural human reasoning, when predicates are used to represent events and have their respectively unique argument structures. These suggest the necessity for order-sorted logic programming to include a reasoning mechanism for a predicate hierarchy that can adjust the difference between argument structures (i.e. manipulating surplus and missing arguments of derived predicates). In the next section we will propose an extended order-sorted logic for handling the hierarchical reasoning of predicates.

4 An order-sorted logic with sort and predicate hierarchies

In this section we define the syntax and semantics (based on [36]) of an order-sorted logic with sort and predicate hierarchies.

4.1 Language and signature

The syntax of an order-sorted language extended to contain hierarchical predicates (to build a predicate hierarchy) is introduced.

Definition 4.1 *An alphabet for an order-sorted first-order language $\mathcal{L}$ consists of the following symbols:*

(1) $\mathcal{S}$ *is a countable set of sort symbols with the greatest sort* $\top$ *and the least sort* $\bot$.
(2) $\mathcal{F}_n$ *is a countable set of n-ary function symbols.*
(3) $\mathcal{P}_n$ *is a countable set of n-ary predicate symbols.*
(4) $\mathcal{E}$ *is a countable set of event identifiers (denoted* e_1, e_2*).*
(5) $\mathcal{V}_s$ *is a countably infinite set of variables of sort s.*
(6) $\mathcal{AL}$ *is a countable set of predicate argument labels (denoted* a_1, a_2*).*
(7) $\wedge, \rightarrow, \forall$ *are the connectives and the universal quantifier.*
(8) $(,), \Rightarrow$ *are the auxiliary symbols.*

We denote by $\mathcal{P}$ the set $\bigcup_{n\geq 0} \mathcal{P}_n$ of all predicate symbols and by $\mathcal{F}$ the set $\bigcup_{n\geq 0} \mathcal{F}_n$ of all function symbols. In the language $\mathcal{L}$, predicates of event assertions (called *event predicates*) are distinguished in the predicates in $\mathcal{P}$. We denote by $\mathcal{P}^{\bullet}$ ($\subseteq \mathcal{P}$) the set of event predicates. $\mathcal{V}$ denotes the set $\bigcup_{s\in\mathcal{S}} \mathcal{V}_s$ of

variables of all sorts. Variables, functions and predicates have ordered and different sorts, and predicate argument labels a_i are used to indicate the argument roles of each predicate. The declarations of sorts, functions and predicates are given by the following.

Definition 4.2 (Declaration) *A declaration over $\mathcal{S} \cup \mathcal{F} \cup \mathcal{P}$ (for $\mathcal{L}$) is an ordered triple $\mathcal{D} = (\mathcal{D}_\mathcal{S}, \mathcal{D}_\mathcal{F}, \mathcal{D}_\mathcal{P})$ such that*

(1) $\mathcal{D}_\mathcal{S}$ is a set of subsort declarations of the form $s_i \sqsubset_S s_j$ where $s_i, s_j \in \mathcal{S}$.
(2) $\mathcal{D}_\mathcal{F}$ is a set of function declarations of the form $f\colon \langle s_1, \ldots, s_n, s\rangle$ where $f \in \mathcal{F}_n (n \geq 0)$ and $s, s_1, \ldots, s_n \in \mathcal{S}$.
(3) $\mathcal{D}_\mathcal{P}$ is a set of sub-predicate declarations of the form $p_i \sqsubset_P p_j$ where $p_i, p_j \in \mathcal{P}^\bullet$, and argument structure declarations of the form $p\colon \{a_1\colon s_1, \ldots, a_n\colon s_n\}$ where $p \in \mathcal{P}_n (n \geq 0)$, $s, s_1, \ldots, s_n \in \mathcal{S}$ and $a_1, \ldots, a_n \in \mathcal{AL} (a_i \neq a_j$ if $i \neq j)$.
(4) If $p\colon \{a_1\colon s_1, \ldots, a_n\colon s_n\} \in \mathcal{D}_\mathcal{P}$, then $Arg(p) = \{a_1, \ldots, a_n\}$ and, for $1 \leq i \leq n$, $Scp(a_i) = s_i$ where Arg is a function from $\mathcal{P}$ to $2^{\mathcal{AL}}$ and Scp is a function from $\mathcal{AL}$ to $\mathcal{S}$.

The subsort declarations (in $\mathcal{D}_\mathcal{S}$) express a sort-hierarchy, and the sub-predicate declarations (in $\mathcal{D}_\mathcal{P}$) express a predicate hierarchy. $Arg(p)$ indicates a finite set of argument labels as the unique argument structure of predicate p. $Scp(a_i)$ denotes a sort s_i as the scope of the argument labeled by a_i. We use the abbreviation $Arg(p - q)$ to denote $Arg(p) - Arg(q)$.

Definition 4.3 (Sorted signature with hierarchical predicates) *A signature for an order-sorted first-order language $\mathcal{L}$ with hierarchical predicates (or simply a sorted signature with hierarchical predicates) is an ordered quadruple $\Sigma = (\mathcal{S}, \mathcal{F}, \mathcal{P}, \mathcal{D})$ where $\mathcal{S}$ is the set of all sort symbols, $\mathcal{F}$ the set of all function symbols, $\mathcal{P}$ the set of all predicate symbols and $\mathcal{D} = (\mathcal{D}_\mathcal{S}, \mathcal{D}_\mathcal{F}, \mathcal{D}_\mathcal{P})$ a declaration over $\mathcal{S} \cup \mathcal{F} \cup \mathcal{P}$.*

Unlike ordinary order-sorted logics, the sorted signature with hierarchical predicates contains sub-predicate declarations $p_i \sqsubset_P p_j$ and argument structure declarations $p\colon \{a_1\colon s_1, \ldots, a_n\colon s_n\}$.

Example 4.1 *This example shows a sorted signature Σ_1 with hierarchical predicates that comprises the following symbols:*

$$\mathcal{S} = \{person,\ man,\ woman,\ \top,\ \bot\},$$
$$\mathcal{F} = \{john,\ mary\},$$
$$\mathcal{P}^\bullet = \{act\_of\_violence,\ illegal\_act\},$$

and the declaration $\mathcal{D} = (\mathcal{D}^*_\mathcal{S}, \mathcal{D}_\mathcal{F}, \mathcal{D}_\mathcal{P})$, where $\mathcal{D}^*_\mathcal{S}$ is the reflexive and transitive closure of $\mathcal{D}_\mathcal{S}$, constructed by

$$\mathcal{D}_{\mathcal{S}} = \{\bot \sqsubset_S man,\ \bot \sqsubset_S woman,$$
$$man \sqsubset_S person,\ woman \sqsubset_S person,\ person \sqsubset_S \top\},$$
$$\mathcal{D}_{\mathcal{F}} = \{john\colon \langle man\rangle,\ mary\colon \langle woman\rangle\},$$
$$\mathcal{D}_{\mathcal{P}} = \{act\_of\_violence \sqsubset_P illegal\_act\} \cup$$
$$\{act\_of\_violence\colon \{actor\colon person\},$$
$$illegal\_act\colon \{actor\colon person, vic\colon person\}\}.$$

In the above example, $act\_of\_violence \sqsubset_P illegal\_act$ (in $\mathcal{D}_{\mathcal{P}}$) declares that the predicate $act\_of\_violence$ is a sub-predicate of $illegal\_act$, and the predicate declaration $illegal\_act\colon \{actor\colon person, vic\colon person\}$ defines that the predicate $illegal\_act$ consists of two arguments labeled with $actor$ and vic, which mean an actor and a victim respectively, and that the sort of both of these arguments is $person$.

4.2 Order-sorted terms and formulas

We define the expressions *order-sorted term* and *formula* of the order-sorted first-order language $\mathcal{L}$.

Definition 4.4 (Sorted terms) *Let $\Sigma = (\mathcal{S}, \mathcal{F}, \mathcal{P}, \mathcal{D})$ be a sorted signature with hierarchical predicates. The set $TERM_s$ of terms of sort s is defined by the following rules:*

(1) A variable $x\colon s$ is a term of sort s.
(2) A constant $c\colon s$ is a term of sort s where $c \in \mathcal{F}_0$ and $c\colon \langle s\rangle \in \mathcal{D}_{\mathcal{F}}$.
(3) If $t_1, \ldots, t_n$ are terms of sorts $s_1, \ldots, s_n$, then $f(t_1, \ldots, t_n)\colon s$ is a term of sort s where $f \in \mathcal{F}_n$ and $f\colon \langle s_1, \ldots, s_n, s\rangle \in \mathcal{D}_{\mathcal{F}}$.
(4) If t is a term of sort s', then t is a term of sort s where $s' \sqsubset_S s \in \mathcal{D}_{\mathcal{S}}$.

The terms of sort s include the terms of all the subsorts s' with $s' \sqsubset_S s$. We denote by $TERM = \bigcup_{s\in\mathcal{S}} TERM_s$ the set of all (order-sorted) terms. The function Var from $TERM$ into $2^{\mathcal{V}}$ is defined by (i) $Var(x\colon s) = \{x\colon s\}$ and (ii) $Var(f(t_1, \ldots, t_n)\colon s) = \bigcup_{1\le i\le n} Var(t_i)$. In particular, $Var(c\colon s) = \emptyset$ where $c \in \mathcal{F}_0$. $TERM_0$ denotes the set of all terms without variables, i.e., $TERM_0 = \{t \in TERM \mid Var(t) = \emptyset\}$. A term t is said to be a ground term if $t \in TERM_0$. $TERM_{0,s}$ denotes the set of all ground terms of sort s.

Definition 4.5 (Sorted formulas) *Let $\Sigma = (\mathcal{S}, \mathcal{F}, \mathcal{P}, \mathcal{D})$ be a sorted signature with hierarchical predicates. The set $FORM$ of sorted formulas is defined by the following rules:*

(1) If $t_1, \ldots, t_n$ are terms of sorts $s_1, \ldots, s_n$, then $p(a_1 \Rightarrow t_1, \ldots, a_n \Rightarrow t_n)$ with $p \in \mathcal{P}_n$ and $p^{\langle e_i\rangle}(a_1 \Rightarrow t_1, \ldots, a_n \Rightarrow t_n)$ with $e_i \in \mathcal{E}$ and $p \in \mathcal{P}_n^{\bullet}$ are

atomic formulas where $p\colon\{a_1\colon s_1, \ldots, a_n\colon s_n\} \in \mathcal{D}_{\mathcal{P}}$.

(2) If A *and* B *are formulas, then* $(A \wedge B)$, $(A \to B)$ *and* $(\forall x\colon sA)$ *are formulas.*

An atomic formula is called simply an *atom*. *ATOM* denotes the set of atomic formulas. We use the symbols $\varphi_p, \varphi_q, \ldots$ to denote predicates $p, q, \ldots$ or predicates with an event identifier $p^{\langle e_i\rangle}, q^{\langle e_i\rangle}, \ldots$.

Example 4.2 *For the sorted signature* Σ_1 *of Example 4.1, we give an example of order-sorted formulas shown as.*

$$act\_of\_violence^{\langle e_1\rangle}(actor \Rightarrow john\colon man),$$
$$illegal\_act(actor \Rightarrow john\colon man, vic \Rightarrow mary\colon woman),$$

where $act\_of\_violence, illegal\_act \in \mathcal{P}^{\bullet}$. The first and second atoms express "the actor John committed an act of violence at e_1" and "the actor John committed an illegal act against the victim Mary."

In the language $\mathcal{L}$, two atoms $\varphi_p(a_1 \Rightarrow t_1, \ldots, a_n \Rightarrow t_n)$ and $\varphi_q(b_1 \Rightarrow r_1, \ldots, b_m \Rightarrow r_m)$ are *equivalent* if $\varphi_p = \varphi_q$ and $\{a_1 \Rightarrow t_1, \ldots, a_n \Rightarrow t_n\} = \{b_1 \Rightarrow r_1, \ldots, b_m \Rightarrow r_m\}$. We write $A \approx B$ to indicate that the atoms A and B are equivalent. For instance, let A and B be the atoms given by

$$A = illegal\_act(actor \Rightarrow john\colon man, vic \Rightarrow mary\colon woman),$$
$$B = illegal\_act(vic \Rightarrow mary\colon woman, actor \Rightarrow john\colon man).$$

Then $A \approx B$.

We define the set $FVar(A)$ of free variables occurring in a formula A. The function $FVar$ from *FORM* into $2^{\mathcal{V}}$ is defined by the following rules:

(1) $FVar(\varphi_p(a_1 \Rightarrow t_1, \ldots, a_n \Rightarrow t_n)) = \bigcup_{1\le i\le n} Var(t_i)$,
(2) $FVar(A * B) = FVar(A) \cup FVar(B)$ for $* \in \{\wedge, \to\}$,
(3) $FVar(\forall x\colon sA) = FVar(A) - \{x\colon s\}$.

A formula F is said to be a sentence if it is without free variables (i.e. $FVar(F) = \emptyset$). We write $\forall F$ for the universal closure $\forall x_1\colon s_1 \cdots \forall x_m\colon s_m F$ where F is a formula with $FVar(F) = \{x_1\colon s_1, \ldots, x_m\colon s_m\}$. A formula F is said to be a ground formula if it is without variables.

Given a sorted signature Σ with hierarchical predicates, an *argument* is an ordered pair (a, t) where a is an argument label and t is an order-sorted term of sort $Scp(a)$ (denoted by $a \Rightarrow t$). An argument is ground if it is without variables. A set of arguments is said to be an argument set (denoted μ) if it is

finite and contains none of the same argument labels. Let μ be an argument set. $\bar{\mu}$ denotes a sequence of all arguments in μ. We write $\varphi_p(\bar{\mu})$ (or $\varphi_p(\mu)$) when we express any sequence of arguments in μ, i.e., any sequence constructed by all elements of μ. The set of argument labels occurring in $\bar{\mu}$ is defined by the function $ls(\bar{\mu}) = \{a_i \in \mathcal{AL} \mid a_i \Rightarrow t_i \in \mu\}$. μ is an argument set of predicate p if $Arg(p) = ls(\bar{\mu})$.

4.3 Argument manipulation for event predicates

We present an argument manipulation that translates any argument set to the argument set of an event predicate $p \in \mathcal{P}^{\bullet}$. As a syntactic operation, it is embedded in inference rules of the Horn clause resolution system we will propose in Section 5, that is a linear resolution system devised to deal with hierarchical reasoning of predicates. This manipulation consists of addition and deletion of arguments based on the argument structure of p (i.e. $Arg(p)$). For this, a language $\mathcal{L}$ must be extended to the language $\mathcal{L}^+$ obtained by adjoining to a set of *supplement constants* $c_1, \ldots, c_n$. We write $TERM^+$, $ATOM^+$ and $FORM^+$ for the set of terms, the set of atoms and the set of formulas in $\mathcal{L}^+$.

Moreover, we need to permit the language $\mathcal{L}^+$ an atomic formula (called an *ill-argumented atom*) consisting of ill arguments in order to directly derive a predicate $\varphi_q(\bar{\mu})$ from a predicate $\varphi_p(\bar{\mu})$ if $p \sqsubset_P q$. If the argument structures of p and q are different (i.e. $\bar{\mu}$ coincides with the argument structure of p but not the argument structure of q), then the ill-argumented atom $\varphi_q(\bar{\mu})$ must be reformed by manipulating the arguments. To distinguish such atoms, every atom in $ATOM^+$ is said to be a well-argumented atom. Let $\Sigma = (\mathcal{S}, \mathcal{F} \cup \{c_1, \ldots, c_n\}, \mathcal{P}, \mathcal{D})$ be a sorted signature and let $p \in \mathcal{P}^{\bullet}$, $a_1, \ldots, a_n \in \mathcal{AL}$ ($a_i \neq a_j$ if $i \neq j$) and $t_1 \in TERM^+_{Scp(a_1)}, \ldots, t_n \in TERM^+_{Scp(a_n)}$. The following form

$$\varphi_p(a_1 \Rightarrow t_1, \ldots, a_n \Rightarrow t_n)$$

is said to be an ill-argumented atom if $Arg(p) \neq \{a_1, \ldots, a_n\}$. $ATOM^+_{\Delta}$ denotes the set of well- and ill-argumented atoms in $\mathcal{L}^+$.

Each ill-argumented atom is not actually a well-formed formula since it is composed by arguments that do not follow its argument structure in a sorted signature. In standard predicate logic, each argument structure is given only by the number of arguments. Namely, $p(t_1, \ldots, t_n)$ is ill-argumented if p is not an n-ary predicate. In our logic, each well-argumented atom follows not only the number of arguments but also the argument structure defined as a finite set of argument labels. The following is an example of well- and ill-argumented

atoms.

Example 4.3 *Let $t_1, t_2, \ldots, t_n$ be order-sorted terms. If $Arg(p) = \{a_1, a_2\}$, then the following expressions*

$$\varphi_p(a_1 \Rightarrow t_1),$$
$$\varphi_p(a_1 \Rightarrow t_1, a_3 \Rightarrow t_3),$$
$$\varphi_p(a_1 \Rightarrow t_1, a_2 \Rightarrow t_2, a_3 \Rightarrow t_3),$$
$$\ldots$$
$$\varphi_p(a_1 \Rightarrow t_1, a_2 \Rightarrow t_2, \ldots, a_n \Rightarrow t_n)$$

are ill-argumented atoms, but

$$\varphi_p(a_1 \Rightarrow t_1, a_2 \Rightarrow t_2)$$

is a well-argumented atom.

In the following definition, an argument manipulation for ill-argumented atoms is formally introduced.

Definition 4.6 (Argument manipulation) *Let A be a well- or ill-argumented atom. The addition ADD of an argument is defined by*

$$ADD(A) = \begin{cases} \varphi_p(\mu \cup \{a \Rightarrow c\colon Scp(a)\}) \text{ if } A = \varphi_p(\bar{\mu}) \text{ and } Arg(p) - ls(\bar{\mu}) \neq \emptyset, \\ \\ A \qquad\qquad\qquad\qquad \text{otherwise}, \end{cases}$$

where $a \in Arg(p) - ls(\bar{\mu})$ and c is a new supplement constant of sort $Scp(a)$. The deletion DEL of an argument is defined by

$$DEL(A) = \begin{cases} \varphi_p(\bar{\mu}) \text{ if } \quad A = \varphi_p(\mu \cup \{a \Rightarrow t\}) \text{ and } a \notin Arg(p), \\ \\ A \qquad \text{otherwise}. \end{cases}$$

The argument manipulation σ is a function from $ATOM^+_\Delta$ to $ATOM^+$ defined by

$$\sigma(A) = ADD^m(DEL^n(A))$$

where m is the least number such that $ADD^m(A) = ADD^{m+1}(A)(m > 0)$ and n is the least number such that $DEL^n(A) = DEL^{n+1}(A)(n > 0)$. [4]

4.4 Σ-structure

We now introduce sorted structures (called Σ-structures) in standard order-sorted logic, which are used to define restricted Σ-structures (called $H\Sigma$-structures) in the semantics of our proposed logic. As mentioned in Section 4.2, atoms composed of the same predicate and the same arguments can be regarded as equivalent even if the arguments in each atom are differently ordered. For example, the following atoms

$$\varphi_p(a_1 \Rightarrow t_1, a_2 \Rightarrow t_2) \text{ and } \varphi_p(a_2 \Rightarrow t_2, a_1 \Rightarrow t_1)$$

are regarded as semantically identical because these arguments are constructed by the same argument set. Instead of the ordering of arguments, the equivalence can be decided by the argument labels denoting their argument roles. Since arguments in a predicate are eliminated and supplemented by the argument manipulation, the position of each argument might be changed. To recognize the argument role of such an argument, argument labels are necessary. In the semantics of the logic that follows this notion, the order of arguments in each predicate does not alter the interpretation of its atom. On the basis of this, Σ-structures are defined with a small modification of sorted structures (in standard order-sorted logic) as follows.

Definition 4.7 *Let $\Sigma = (\mathcal{S}, \mathcal{F}, \mathcal{P}, \mathcal{D})$ be a sorted signature with hierarchical predicates. A Σ-structure is an ordered pair $M = (U, I)$ such that*

(1) U is a non-empty set (the universe of M), and
(2) I is a function where
- *(a) $I(s) \subseteq U$, for $s \in \mathcal{S}$,*
- *(b) $I(s_i) \subseteq I(s_j)$, for $s_i \sqsubset_S s_j \in \mathcal{D}_\mathcal{S}$,*
- *(c) $I(f)\colon I(s_1) \times \cdots \times I(s_n) \to I(s)$, for $f \in \mathcal{F}_n$ with $f\colon \langle s_1, \ldots, s_n, s\rangle \in \mathcal{D}_\mathcal{F}$,*
- *(d) $I(p) \subseteq X_p$, for $p \in \mathcal{P}_n$ with $p\colon \{a_1\colon s_1, \ldots, a_n\colon s_n\} \in \mathcal{D}_\mathcal{P}$, where $X_p = \{\rho \in (Arg(p) \to U) \mid \forall a_i \in Arg(p)[\rho(a_i) \in I(Scp(a_i))]\}$* [5] *,*
- *(e) $I(p^{\langle e_i\rangle}) \subseteq I(p)$, for $p \in \mathcal{P}^\bullet$.*

We call a function $\rho \in X_p$ an *argument interpretation* of predicate p, which is used for interpreting p in the Σ-structure. A set of argument interpretations

[4] Let f be a function. We write f^n for the composite n functions $f \circ f \circ \cdots \circ f$.
[5] For sets X and Y, the set of all functions from X to Y is denoted by $(X \to Y)$.

$\rho = \{(a_1, d_1), (a_2, d_2), \cdots, (a_n, d_n)\}$ of p where (a_i, d_i) is an ordered pair of $a_i \in \mathcal{AL}$ and $d_i \in U$ defines the interpretation $I(p)$. This is based on the fact that in the semantics of first-order logic an n-ary predicate p is a set (i.e. a subset of U^n) of ordered n-tuples on the universe U.

4.5 Restricted Σ-structure for hierarchical predicates

A requirement of logic with predicate hierarchy is that $\varphi_p(\bar{\mu})$ implies $\sigma(\varphi_q(\bar{\mu}))$ with the argument manipulation σ if $p \sqsubset_P q$ holds and in particular the argument structures of q, $p \in \mathcal{P}^\bullet$ are different. To obtain the semantics, the predicate q derived in a hierarchy must be interpreted to include the manipulation of the argument structure $\bar{\mu}$ of the predicate p. The semantic constraint on the hierarchical relationship between predicates is defined by a restricted Σ-structure ($H\Sigma$-structure). Then we will introduce two translations in structures: *argument manipulation* and *composition of predicates* that are used to restrict Σ-structures. The argument manipulation in semantics corresponds to what syntactically manipulates arguments (in Definition 4.6), and the composition of predicates interprets an integration of argument structures in predicates representing an incident (as in Example 3.3).

First, the argument manipulation in structures is given as adjusting the interpretation of a predicate p to the argument structure of a predicate q. The adjusted arguments consist of the following two parts:

(1) Common arguments: the intersection of the set of p's arguments and the set of q's, and
(2) Additional arguments: the set of q's arguments that are not p's arguments.

Let $p, q \in \mathcal{P}^\bullet$ and let $M = (U, I)$ be a Σ-structure. The common arguments of p and q are given by

$$\rho \cap (Arg(q) \times U)$$

where $\rho \in I(p)$. The additional arguments are given by

$$\{(a_1, d_1), \ldots, (a_n, d_n)\} \; for \; d_1 \in I(Scp(a_1)), \ldots, d_n \in I(Scp(a_n))$$

where $Arg(q - p) = \{a_1, \ldots, a_n\}$. As discussed in Example 3.1, missing arguments should be existentially quantified in event assertions. Corresponding to this, we have that there exist $d_1, \ldots, d_n$, and the union of the common arguments $\rho \cap (Arg(q) \times U)$ and the additional arguments $\{(a_1, d_1), \ldots, (a_n, d_n)\}$ belongs to a derived predicate q in the interpretation.

The function $ls^*(\rho) = \{a_i \in \mathcal{AL} \mid (a_i, d_i) \in \rho\}$ is defined as the set of argument labels from an argument interpretation ρ.

Definition 4.8 (Argument manipulation in structures)
Let $\Sigma = (\mathcal{S}, \mathcal{F}, \mathcal{P}, \mathcal{D})$ be a sorted signature with hierarchical predicates and let $M = (U, I)$ be a Σ-structure. An interpretation ι_q for the argument manipulation to a predicate q is a translation of an argument interpretation ρ to an argument interpretation of q such that

$$\iota_q(\rho) = (\rho \cap Arg(q) \times U) \cup \{(a_1, d_1), \ldots, (a_n, d_n)\}$$

where $\{a_1, \ldots, a_n\} = Arg(q) - ls^*(\rho)$ [6] *and, for $1 \leq i \leq n$, $d_i \in I(Scp(a_i))$.*

To interpret the additional arguments as existentially quantified, ι_q is defined by one of the interpretations for the argument manipulation. Namely, the additional arguments $\{(a_1, d_1), \ldots, (a_n, d_n)\}$ are given by choosing $d_1 \in I(Scp(a_1)), \ldots, d_n \in I(Scp(a_n))$ and then it determines an interpretation ι_q for the argument manipulation.

Secondly, we define a composition $I(p_1^{\langle e_i \rangle}); \cdots; I(p_n^{\langle e_i \rangle})$ of interpretations of predicates $p_1^{\langle e_i \rangle}, \ldots, p_n^{\langle e_i \rangle}$ with the same event identifier e_i where the argument interpretations in $I(p_1^{\langle e_i \rangle}), \ldots, I(p_n^{\langle e_i \rangle})$ are integrated into a set of argument interpretations. The union $Arg(p_1) \cup \cdots \cup Arg(p_n)$ is used to make such a set obtained by integrating the argument interpretations in $I(p_1^{\langle e_i \rangle})$, $\cdots$, $I(p_n^{\langle e_i \rangle})$ and excluding the argument interpretations in disagreement (e.g. ρ_1 and ρ_2 are in disagreement if $\rho_1(a) \neq \rho_2(a)$ for some $a \in Arg(p_1) \cup Arg(p_2)$ where $\rho_1 \in I(p_1^{\langle e_i \rangle})$ and $\rho_2 \in I(p_2^{\langle e_i \rangle})$). This composition is employed to embody the interpretation that predicates $p_1^{\langle e_i \rangle}, \ldots, p_n^{\langle e_i \rangle}$ imply a specific predicate $q^{\langle e_i \rangle}$ such that $q \sqsubset_P p_1, \ldots, q \sqsubset_P p_n$. For example, the predicates $act\_of\_violence^{\langle e_1 \rangle}$ and $steal^{\langle e_1 \rangle}$ imply the predicate $rob\_with\_violence^{\langle e_1 \rangle}$ for $rob\_with\_violence \sqsubset_P act\_of\_violence$ and $rob\_with\_violence \sqsubset_P steal$.

Definition 4.9 (Composition of predicates in structures)
Let $\Sigma = (\mathcal{S}, \mathcal{F}, \mathcal{P}, \mathcal{D})$ be a sorted signature with hierarchical predicates. The operation $\sqcap$ for two argument interpretations is defined by

$$\rho_1 \sqcap \rho_2 = \begin{cases} 1 & \text{if } \rho_1(a) = \rho_2(a) \text{ for all } a \in ls^*(\rho_1) \cap ls^*(\rho_2), \\ 0 & \text{otherwise.} \end{cases}$$

[6] We have $ls^*(\rho) = Arg(p)$ by Definition 4.7 if the argument interpretation ρ is a member of $I(\varphi_p)$.

Let $M = (U, I)$ be a Σ-structure. The composition of two sets $I(p_1^{\langle e_i \rangle}), I(p_2^{\langle e_i \rangle})$ of argument interpretations is defined as follows:

$$I(p_1^{\langle e_i \rangle}); I(p_2^{\langle e_i \rangle}) = \{\rho_1 \cup \rho_2 \mid \rho_1 \in I(p_1^{\langle e_i \rangle}), \rho_2 \in I(p_2^{\langle e_i \rangle}), \rho_1 \sqcap \rho_2 = 1\}.$$

Here the composition of two predicates can be expanded to the composition of n predicates as follows:

$$I(p_1^{\langle e_i \rangle}); \cdots; I(p_n^{\langle e_i \rangle}) = (((I(p_1^{\langle e_i \rangle}); I(p_2^{\langle e_i \rangle})); \cdots); I(p_n^{\langle e_i \rangle})).$$

Let $\sqsubset_P$ be a sub-predicate relation. The one-step sub-predicate relation is defined by: $p \sqsubset_P^1 q$ if $p \sqsubset_P q$, $p \neq q$, and there exists no $\sqsubset_P$ path from p to q except for $p \sqsubset_P q$.

In the following, a restricted Σ-structure (called an $H\Sigma$-structure) on a sorted signature Σ with hierarchical predicates can be defined using the two translations (in Definition 4.8 and Definition 4.9).

Definition 4.10 ($H\Sigma$-structure) *Let $M = (U, I)$ be a Σ-structure on a sorted signature $\Sigma = (\mathcal{S}, \mathcal{F}, \mathcal{P}, \mathcal{D})$ with hierarchical predicates. M is an $H\Sigma$-structure with $\iota_{\mathcal{P}}$ if there exists a set $\iota_{\mathcal{P}}$ of interpretations ι_q (for the argument manipulation) to all event predicates $q \in \mathcal{P}^\bullet$ and the following conditions hold:*

(1) If $p, q \in \mathcal{P}^\bullet$ and $p \sqsubset_P q \in \mathcal{D}_{\mathcal{P}}$, then

$$\{\iota_q(\rho) \mid \rho \in I(\varphi_p)\} \subseteq I(\varphi_q)$$

where $\langle \varphi_p, \varphi_q \rangle$ is $\langle p, q \rangle$, $\langle p^{\langle e_i \rangle}, q^{\langle e_i \rangle} \rangle$ or $\langle p^{\langle e_i \rangle}, q \rangle$.

(2) If $p_1, \ldots, p_n, q \in \mathcal{P}^\bullet$ and $q \sqsubset_P p_1, \ldots, q \sqsubset_P p_n \in \mathcal{D}_{\mathcal{P}} (n > 1)$ where $p_1, \ldots, p_n$ are all predicates such that $q \sqsubset_P^1 p_j$, then

$$\{\iota_q(\rho) \mid \rho \in I(p_1^{\langle e_i \rangle}); \cdots; I(p_n^{\langle e_i \rangle})\} \subseteq I(q^{\langle e_i \rangle}).$$

In what follows, we will deal only with $H\Sigma$-structures.

4.6 Interpretation and satisfiability

We define an interpretation of expressions in our proposed logic and a satisfiability relation of the interpretation and expressions. The interpretation includes argument manipulation (in structures) by attaching $\iota_{\mathcal{P}}$ (a set of interpretations for the argument manipulation to all event predicates) to an ordered pair of an $H\Sigma$-structure and a variable assignment. When a formula

with supplemented arguments is true in the interpretation with $\iota_{\mathcal{P}}$, this means that there exists an argument manipulation and its interpretation satisfies the formula. As a result, supplemented arguments (i.e. supplement constants) in the formula are quantified existentially in the semantics.

A variable assignment (or an assignment) in an $H\Sigma$-structure $M = (U, I)$ is a function $\alpha\colon \mathcal{V} \to U$ such that $\alpha(x\colon s) \in I(s)$ for all variables $x\colon s \in \mathcal{V}$. Let α be an assignment in an $H\Sigma$-structure $M = (U, I)$, let $x\colon s$ be a variable, and let $d \in I(s)$. The variable assignment $\alpha[x\colon s/d]$ is defined by $\alpha - \{(x\colon s, \alpha(x\colon s))\} \cup \{(x\colon s, d)\}$. We write $\alpha[x_1\colon s_1/d_1, \ldots, x_n\colon s_n/d_n]$ for $(((\alpha[x_1\colon s_1/d_1])[x_2\colon s_2/d_2])$ $\ldots)[x_n\colon s_n/d_n]$. That is, if $y\colon s = x_i\colon s_i$ for some $1 \leq i \leq n$, then $\alpha[x_1\colon s_1/d_1, \ldots,$ $x_n\colon s_n/d_n](y\colon s) = d_i$. Otherwise, $\alpha[x_1\colon s_1/d_1, \ldots, x_n\colon s_n/d_n](y\colon s) = \alpha(y\colon s)$. An $H\Sigma$-interpretation is an ordered triple $\mathcal{I} = (M, \iota_{\mathcal{P}}, \alpha)$ where M is an $H\Sigma$-structure with $\iota_{\mathcal{P}}$ and α is an assignment. The interpretation $\mathcal{I}[x_1\colon s_1/d_1, \ldots,$ $x_n\colon s_n/d_n]$ is defined by $(M, \iota_{\mathcal{P}}, \alpha[x_1\colon s_1/d_1, \ldots, x_n\colon s_n/d_n])$.

Definition 4.11 *Let $\mathcal{I} = (M, \iota_{\mathcal{P}}, \alpha)$ with $M = (U, I)$ be an $H\Sigma$-interpretation. The denotation $[\![\ ]\!]_\alpha\colon TERM^+ \to U$ is defined by the following rules:*

(1) $[\![x\colon s]\!]_\alpha = \alpha(x\colon s)$.
(2) If c is a normal constant, then $[\![c\colon s]\!]_\alpha = I(c)$.
(3) If c is a supplement constant, then $[\![c\colon s]\!]_\alpha = d$ *where $a \Rightarrow c\colon s$ is added in $\sigma(\varphi_q(a_1 \Rightarrow t_1, \ldots, a_n \Rightarrow t_n))$ and $(a, d) \in \iota_q(\{(a_1, [\![t_1]\!]_\alpha), \ldots, (a_n, [\![t_n]\!]_\alpha)\})$ with $\iota_q \in \iota_{\mathcal{P}}$.*
(4) $[\![f(t_1, \ldots, t_n)\colon s]\!]_\alpha = I(f)([\![t_1]\!]_\alpha, \ldots, [\![t_n]\!]_\alpha)$ *with* $I(f)([\![t_1]\!]_\alpha, \ldots, [\![t_n]\!]_\alpha) \in I(s)$.

In the definition supplement constants are interpreted by the corresponding elements assigned in $\iota_{\mathcal{P}}$.

Definition 4.12 *Let $\mathcal{I} = (M, \iota_{\mathcal{P}}, \alpha)$ with $M = (U, I)$ be an $H\Sigma$-interpretation and F an order-sorted formula. The satisfiability relation $\mathcal{I} \models_{H\Sigma} F$ is defined by the following rules:*

(1) $\mathcal{I} \models_{H\Sigma} \varphi_p(a_1 \Rightarrow t_1, \ldots, a_n \Rightarrow t_n)$ *iff* $\{(a_1, [\![t_1]\!]_\alpha), \ldots, (a_n, [\![t_n]\!]_\alpha)\} \in I(\varphi_p)$.
(2) $\mathcal{I} \models_{H\Sigma} (A \wedge B)$ *iff* $\mathcal{I} \models_{H\Sigma} A$ *and* $\mathcal{I} \models_{H\Sigma} B$.
(3) $\mathcal{I} \models_{H\Sigma} (A \to B)$ *iff* $\mathcal{I} \not\models_{H\Sigma} A$ *or* $\mathcal{I} \models_{H\Sigma} B$.
(4) $\mathcal{I} \models_{H\Sigma} (\forall x\colon s A)$ *iff for all* $d \in I(s)$, $\mathcal{I}[x\colon s/d] \models_{H\Sigma} A$.

If an atomic formula is satisfied by an $H\Sigma$-interpretation $\mathcal{I}$, then also all the equivalent atoms must be satisfied by it. $ATOM/\approx$ is the quotient set of $ATOM$ modulo $\approx$. Then for any $A, B \in AS$ with $AS \in ATOM/\approx$, $\mathcal{I} \models_{H\Sigma} A$ iff $\mathcal{I} \models_{H\Sigma} B$. We write $\mathcal{I} \models_{H\Sigma} \Gamma$ ($\mathcal{I}$ is an $H\Sigma$-model of Γ) if $\mathcal{I}$ is an $H\Sigma$-interpretation and $\mathcal{I} \models_{H\Sigma} F$ for every formula $F \in \Gamma$. Γ is $H\Sigma$-satisfiable if it has an $H\Sigma$-model, and Γ is $H\Sigma$-unsatisfiable if it has no $H\Sigma$-models. We write $\Gamma \models_{H\Sigma} F$ (F is a consequence of Γ in the class of $H\Sigma$-structures) if

every $H\Sigma$-model of Γ is an $H\Sigma$-model of a formula F.

The following two lemmas will be proved by the fact that the argument manipulation σ (as a translation of an ill-argumented atom $\varphi_q(\mu)$ to the well-argumented atom) corresponds to the interpretation ι_q for the argument manipulation to the predicate q.

Lemma 4.1 *If $p \sqsubset_P q \in \mathcal{D}_\mathcal{P}$, then $\forall\varphi_p(\bar{\mu}) \models_{H\Sigma} \forall\sigma(\varphi_q(\bar{\mu}))$ where $\langle\varphi_p, \varphi_q\rangle$ is $\langle p, q\rangle$, $\langle p^{\langle e_i\rangle}, q^{\langle e_i\rangle}\rangle$ or $\langle p^{\langle e_i\rangle}, q\rangle$ and μ is an argument set of p.*

Proof. Let $\mathcal{I} \models_{H\Sigma} \forall y_1{:}\, s_1 \cdots y_r{:}\, s_r\varphi_p(\bar{\mu})$ where $\mathcal{I} = (M, \iota_\mathcal{P}, \alpha)$. Then for all $d_1 \in I(s_1), \ldots, d_r \in I(s_r)$, $\mathcal{I}[y_1{:}\, s_1/d_1, \ldots, y_r{:}\, s_r/d_r] \models_{H\Sigma} \varphi_p(\bar{\mu})$. By Definition 4.6, $\sigma(\varphi_q(\bar{\mu})) = \varphi_q(a_1 \Rightarrow t_1, \ldots, a_m \Rightarrow t_m, b_1 \Rightarrow c_1{:}\, Scp(b_1), \ldots, b_k \Rightarrow c_k{:}\, Scp(b_k))$, where $Arg(q) \cap ls(\bar{\mu}) = \{a_1, \ldots, a_m\}$, for $1 \leq l \leq m$, $a_l \Rightarrow t_l \in \mu$, and $Arg(q) - ls(\bar{\mu}) = \{b_1, \ldots, b_k\}$. By Definitions 4.8, 4.10 and 4.11, $I(\varphi_q)$ includes

$$\{(a_1, [\![t_1]\!]_\alpha), \ldots, (a_m, [\![t_m]\!]_\alpha)\} \cup \{(b_1, [\![c_1{:}\, Scp(b_1)]\!]_\alpha), \ldots, (b_k, [\![c_k{:}\, Scp(b_k)]\!]_\alpha)\}.$$

So $\mathcal{I}[y_1{:}\, s_1/d_1, \ldots, y_r{:}\, s_r/d_r] \models_{H\Sigma} \sigma(\varphi_q(\bar{\mu}))$. This derives $\mathcal{I} \models_{H\Sigma} \forall y_1{:}\, s_1 \cdots y_r{:}\, s_r$ $\sigma(\varphi_q(\bar{\mu}))$ with $FVar(\sigma(\varphi_q(\bar{\mu}))) \subseteq \{y_1{:}\, s_1, \ldots, y_r{:}\, s_r\}$. ∎

Lemma 4.2 *If $q \sqsubset_P p_1, \ldots, q \sqsubset_P p_n \in \mathcal{D}_\mathcal{P}(n > 1)$ where $p_1, \ldots, p_n$ are all predicates such that $q \sqsubset^1_P p_j$, then $\{\forall p_1^{\langle e_i\rangle}(\bar{\mu}_1), \ldots, \forall p_n^{\langle e_i\rangle}(\bar{\mu}_n)\} \models_{H\Sigma} \forall\sigma(q^{\langle e_i\rangle}(\bar{\mu}))$ where, for $1 \leq j \leq n$, μ_j is an argument set of p_j and $\mu = \mu_1 \cup \cdots \cup \mu_n$ is an argument set.*

Proof. Let $\mathcal{I} \models_{H\Sigma} \forall p_j^{\langle e_i\rangle}(\bar{\mu}_j)$ for $1 \leq j \leq n$ where $\mathcal{I} = (M, \iota_\mathcal{P}, \alpha)$. Then $\mathcal{I} \models_{H\Sigma}$ $\forall y_1{:}\, s_1 \cdots y_r{:}\, s_r(p_1^{\langle e_i\rangle}(\bar{\mu}_1) \wedge \cdots \wedge p_n^{\langle e_i\rangle}(\bar{\mu}_n))$ with $FVar(p_1^{\langle e_i\rangle}(\bar{\mu}_1) \wedge \cdots \wedge p_n^{\langle e_i\rangle}(\bar{\mu}_n)) = \{y_1{:}\, s_1, \ldots, y_r{:}\, s_r\}$. Thus, for $1 \leq j \leq n$, $\mathcal{I}[y_1{:}\, s_1/d_1, \ldots, y_r{:}\, s_r/d_r] \models_{H\Sigma} p_j^{\langle e_i\rangle}(\bar{\mu}_j)$ for all $d_1 \in I(s_1), \ldots, d_r \in I(s_r)$. By Definition 4.6, $\sigma(q^{\langle e_i\rangle}(\bar{\mu})) = q^{\langle e_i\rangle}(a_1 \Rightarrow$ $t_1, \ldots, a_m \Rightarrow t_m, b_1 \Rightarrow c_1{:}\, Scp(b_1), \ldots, b_k \Rightarrow c_k{:}\, Scp(b_k))$, where $\mu = \mu_1 \cup$ $\cdots \cup \mu_n$, $Arg(q) \cap ls(\bar{\mu}) = \{a_1, \ldots, a_m\}$, for $1 \leq l \leq m$, $a_l \Rightarrow t_l \in \mu$, and $Arg(q) - ls(\bar{\mu}) = \{b_1, \ldots, b_k\}$. Since $I(p_1^{\langle e_i\rangle}); \cdots; I(p_n^{\langle e_i\rangle})$ includes $\{(a, [\![t]\!]_\alpha) \mid$ $a \Rightarrow t \in \mu\}$ (by Definition 4.9), we have

$$\{(a_1, [\![t_1]\!]_\alpha, \ldots, (a_m, [\![t_m]\!]_\alpha), (b_1, d'_1), \ldots, (b_k, d'_k)\} \in I(q^{\langle e_i\rangle})$$

where $d'_1 = [\![c_1{:}\, Scp(b_1)]\!]_\alpha, \ldots, d'_k = [\![c_k{:}\, Scp(b_k)]\!]_\alpha$ (by Definitions 4.8, 4.10 and 4.11). Hence $\mathcal{I}[y_1{:}\, s_1/d_1, \ldots, y_r{:}\, s_r/d_r] \models_{H\Sigma} \sigma(q^{\langle e_i\rangle}(\bar{\mu}))$ can be proved. Therefore, we have $\mathcal{I} \models_{H\Sigma} \forall y_1{:}\, s_1 \cdots y_r{:}\, s_r\sigma(q^{\langle e_i\rangle}(\bar{\mu}))$ with $FVar(\sigma(q^{\langle e_i\rangle}(\bar{\mu})))$ $\subseteq \{y_1{:}\, s_1, \ldots, y_r{:}\, s_r\}$. ∎

5 Horn clause resolution with predicate hierarchy

The purpose of this section is to present a Horn clause resolution system that is extended to include inference rules of predicate hierarchies with the argument manipulation σ and an order-sorted unification algorithm.

5.1 Horn clauses

Before developing the Horn clause resolution system for the proposed logic, we define Horn clausal forms in $\mathcal{L}^+$ (used as the syntax of logic programming).

Definition 5.1 (Horn clauses) *Let $L, L_1, \ldots, L_n$ be atoms. A goal G is denoted by the form $G := \{L_1, \ldots, L_n\}$ ($n \geq 0$). In particular, we use the notation $\Box$ if $n = 0$ (i.e. the goal is the empty set). A clause C is denoted by the form $C := L \leftarrow G$. In particular, we write $L \leftarrow$ for $L \leftarrow \Box$. The set of all clauses is denoted by CFORM.*

We use the abbreviation L to denote a goal $\{L\}$ that is a singleton. We define the function $CVar: CFORM \rightarrow 2^{\mathcal{V}}$ by: $CVar(L \leftarrow G) = (\bigcup_{L_i \in G} FVar(L_i)) \cup FVar(L)$. Clauses $L \leftarrow \{L_1, \ldots, L_n\}$ represent the universal closures $\forall(L_1 \wedge \ldots \wedge L_n \rightarrow L)$.

Definition 5.2 (Program) *A (logic) program $P = (\Sigma, CS)$ consists of a sorted signature Σ with hierarchical predicates and a finite set CS of clauses without supplement constants.*

Note that any supplement constant does not belong to the program P (exactly the set CS of clauses), because it is used only in formulas to which a sorted substitution or an argument manipulation is applied.

5.2 Sorted substitution

Definition 5.3 (Sorted substitution) *A sorted substitution is a function θ mapping from a finite set of variables to the set $TERM^+$ of all terms in $\mathcal{L}^+$ where $\theta(x{:}\, s) \neq x{:}\, s$ and $\theta(x{:}\, s) \in TERM^+_s$.*

Let θ be a sorted substitution. $Dom(\theta)$ denotes the domain of θ and $Cod(\theta)$ denotes the codomain of θ. The sorted substitution θ can be represented as a finite set $\{x_1{:}\, s_1/t_1, \ldots, x_n{:}\, s_n/t_n\}$ where $Dom(\theta) = \{x_1{:}\, s_1, \ldots, x_n{:}\, s_n\}$ and $\theta(x_1{:}\, s_1) = t_1, \ldots, \theta(x_n{:}\, s_n) = t_n$. Let V be a set of sorted variables. θ is called a ground (sorted) substitution for V if $Var(\theta(x{:}\, s)) = \emptyset$ for all $x{:}\, s \in V$, i.e.,

$\theta(x{:}\,s)$ is a ground term. θ is called a ground substitution if $Var(\theta(x{:}\,s)) = \emptyset$ for all $x{:}\,s \in Dom(\theta)$. We write ϵ for the identity substitution given by the empty set. A sorted substitution θ is a renaming if it is injective on $Dom(\theta)$ and $Cod(\theta)$ is a set of variables. The restriction of a substitution θ to a set V of variables is defined by $\theta{\restriction}V = \{x{:}\,s/\theta(x{:}\,s) \mid x{:}\,s \in V \cap Dom(\theta)\}$.

We define an extension of the sorted substitution θ to expressions (terms, formulas, goals and clauses).

Definition 5.4 *Let A, B be order-sorted formulas, $L, L_1, \ldots, L_n$ atoms, G a goal and θ a sorted substitution. $E\theta$ (based on [7,25]) is defined by the following rules:*

- *If $E = x{:}\,s$ and $x{:}\,s \in Dom(\theta)$, then $E\theta = \theta(x{:}\,s)$.*
- *If $E = x{:}\,s$ and $x{:}\,s \notin Dom(\theta)$, then $E\theta = x{:}\,s$.*
- *If $E = f(t_1, \ldots, t_m){:}\,s$, then $E\theta = f(t_1\theta, \ldots, t_m\theta){:}\,s$.*
- *If $E = \varphi_p(a_1 \Rightarrow t_1, \ldots, a_m \Rightarrow t_m)$, then $E\theta = \varphi_p(a_1 \Rightarrow t_1\theta, \ldots, a_m \Rightarrow t_m\theta)$.*
- *If $E = (A * B)$ for $* \in \{\wedge, \rightarrow\}$, then $E\theta = (A\theta * B\theta)$.*
- *If $E = (\forall x{:}\,sA)$, then $E\theta = (\forall x{:}\,sA(\theta{\restriction}FVar(\forall x{:}\,sA)))$.*
- *If $E = \{L_1, \ldots, L_n\}$, then $E\theta = \{L_1\theta, \ldots, L_n\theta\}$.*
- *If $E = L \leftarrow G$, then $E\theta = L\theta \leftarrow G\theta$.*

Let θ be a sorted substitution and E an expression. We call $E\theta$ an instance of E by θ. An expression E is ground if E is without variables. θ is called a ground substitution for E if $E\theta$ is ground. We denote the set of all ground instances of E by $ground(E)$. Let ES be a set $\{E_1, \ldots, E_n\}$ of expressions. We define $ground(ES) = \bigcup_{E_i \in ES} ground(E_i)$. In particular, $ground(CS) = \bigcup_{C \in CS} ground(C)$ where CS is a set of clauses. Let θ and γ be sorted substitutions. The composition of θ and γ (denoted $\theta\gamma$) is defined by

$$(x{:}\,s)\theta\gamma = ((x{:}\,s)\theta)\gamma.$$

An expression E is a variant of an expression E' if there exists a renaming θ such that $E = E'\theta$. Let E_1 and E_2 be expressions. A substitution θ is a unifier of E_1 and E_2 if $E_1\theta = E_2\theta$. A substitution θ is more general than γ (denoted $\theta \leq \gamma$) if there exists λ such that $\gamma = \theta\lambda$. A unifier θ of E_1 and E_2 is called a most general unifier if for every unifier γ of E_1 and E_2 we have $\theta \leq \gamma$.

5.3 Sorted unification and resolution

We will introduce a unification algorithm for order-sorted atoms that is used in order-sorted resolution. Let $\varphi_p(a_1 \Rightarrow t_1, \ldots, a_n \Rightarrow t_n)$ and $\varphi_p(b_1 \Rightarrow r_1, \ldots, b_n \Rightarrow r_n)$ be atoms containing the same predicate p. To unify these, a unification al-

gorithm is applied to the pair of sequences $(t_1, \ldots, t_n)$ and $(r'_1, \ldots, r'_n)$ where $r'_i = r_j$ if $a_i = b_j$ for $1 \leq i, j \leq n$. The order-sorted unification algorithm (based on [8,27,36]) is defined by translations on systems of equations.

Definition 5.5 (Sorted unification algorithm)
Let $(t_1, \ldots, t_n)$ and $(r_1, \ldots, r_n)$ be sequences of sorted terms. Let (ES, S) with $ES = \{t_1 \doteq r_1, \ldots, t_n \doteq r_n\}$ and $S = \emptyset$ be an initial pair of equational systems. A translation of (ES, S) in the order-sorted unification algorithm is defined by the following rules:

Identity $(ES \cup \{t \doteq t\}, S) \Longrightarrow (ES, S)$ *if* $t \notin \mathcal{V}$.

Decomposition $(ES \cup \{f(t_1, \ldots, t_n) \doteq f(r_1, \ldots, r_n)\}, S) \Longrightarrow$
$(ES \cup \{t_1 \doteq r_1, \ldots, t_n \doteq r_n\}, S)$
if there exists at least one $t_i \neq r_i$.

Transposition $(ES \cup \{t \doteq x\colon s\}, S) \Longrightarrow (ES \cup \{x\colon s \doteq t\}, S)$
if $t \notin \mathcal{V}$ *or* $t \in \mathcal{V}_{s'}$ *with* $s' <_S s$.

Substitution 1 $(ES \cup \{x\colon s \doteq t\}, S) \Longrightarrow (ES\tau, S\tau \cup \{x\colon s \doteq t\})$
where $\tau = \{x\colon s/t\}$ *if* $t \in TERM_s$ [7] *and* $x\colon s \notin Var(t)$.

Substitution 2 $(ES \cup \{x\colon s_1 \doteq y\colon s_2\}, S) \Longrightarrow$
$(ES\tau, S\tau \cup \{x\colon s_1 \doteq z\colon s_3, y\colon s_2 \doteq z\colon s_3\})$
where $s_3 = glb(s_1, s_2)$ *and* $\tau = \{x\colon s_1/z\colon s_3, y\colon s_2/z\colon s_3\}$
if s_3 *is neither* s_1, s_2, *nor* $\perp$.

If ES is translated into the empty set, then the algorithm terminates. We obtain a most general unifier $\{x_1\colon s_1/t'_1, \ldots, x_m\colon s_m/t'_m\}$ for $(t_1, \ldots, t_n)$ and $(r_1, \ldots, r_n)$ if $ES = \emptyset$ and $S = \{x_1\colon s_1 \doteq t'_1, \ldots, x_m\colon s_m \doteq t'_m\}$. The correctness of this order-sorted unification algorithm is shown in [19].

The Horn clause resolution with predicate hierarchy is obtained by the following inference rules that is an extension of the linear resolution in [12].

Definition 5.6 (Resolvent) *Let $P = (\Sigma, CS)$ be a program and let $\langle \varphi_p, \varphi_q \rangle$ be $\langle p, q \rangle$, $\langle p^{\langle e_i \rangle}, q^{\langle e_i \rangle} \rangle$ or $\langle p^{\langle e_i \rangle}, q \rangle$.*

- **R1-resolution rule.** *Let G be a goal and let $L' \leftarrow G' \in CS$. If θ is a unifier of $L \in G$ and L', then $(G - \{L\})\theta \cup G'\theta$ is an unrestricted resolvent*

[7] Recall that $TERM_s$ contains not only the terms of sort s but also the terms of subsorts of s.

of G with respect to L and $L' \leftarrow G'$. We write

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

- **R2-resolution rule.** *Let G be a goal and let $\varphi_p(\bar{\mu}') \leftarrow G' \in CS$. If $p \sqsubset_P q \in \mathcal{D}_{\mathcal{P}}$ and θ is a unifier of $\varphi_q(\bar{\mu}) \in G$ and $\sigma(\varphi_q(\bar{\mu}'))$, then $(G - \{\varphi_q(\bar{\mu})\})\theta \cup G'\theta$ is an unrestricted resolvent of G with respect to $\varphi_q(\bar{\mu})$ and $\varphi_p(\bar{\mu}') \leftarrow G'$. We write*

$$G \xrightarrow{\theta}_{R2} (G - \{\varphi_q(\bar{\mu})\})\theta \cup G'\theta.$$

- **R3-resolution rule.** *Let G be a goal and let $p_1^{\langle e_i \rangle}(\bar{\mu}_1) \leftarrow G_1, \ldots, p_n^{\langle e_i \rangle}(\bar{\mu}_n) \leftarrow G_n \in CS$. If $p_1, \ldots, p_n (n > 1)$ are all predicates such that $q \sqsubset_P^1 p_j \in \mathcal{D}_{\mathcal{P}}$, and θ is a unifier of $q^{\langle e_i \rangle}(\bar{\mu}) \in G$ and $\sigma(q^{\langle e_i \rangle}(\bar{\mu}'))$ where $\mu' = \mu_1 \cup \cdots \cup \mu_n$ is an argument set, then $(G - \{q^{\langle e_i \rangle}(\bar{\mu})\})\theta \cup (G_1 \cup \cdots \cup G_n)\theta$ is an unrestricted resolvent of G with respect to $q^{\langle e_i \rangle}(\bar{\mu})$ and $p_1^{\langle e_i \rangle}(\bar{\mu}_1) \leftarrow G_1, \ldots, p_n^{\langle e_i \rangle}(\bar{\mu}_n) \leftarrow G_n$. We write*

$$G \xrightarrow{\theta}_{R3} (G - \{q^{\langle e_i \rangle}(\bar{\mu})\})\theta \cup (G_1 \cup \cdots \cup G_n)\theta.$$

We write $G \xrightarrow{\theta} G'$ if $G \xrightarrow{\theta}_{R1} G'$, $G \xrightarrow{\theta}_{R2} G'$ or $G \xrightarrow{\theta}_{R3} G'$. An unrestricted resolvent is a resolvent if the unifier θ is most general.

Definition 5.7 (Resolution) *Let P be a program. A finite sequence*

$$P\colon G_0 \xrightarrow{\theta_1} G_1 \xrightarrow{\theta_2} G_2 \xrightarrow{\theta_3} \cdots \xrightarrow{\theta_n} G_n$$

is an unrestricted resolution of G_0 with respect to P ($n \geq 0$). We denote it by $P\colon G_0 \xrightarrow{\theta}\!\!\!\!\rightarrow G_n$ with $\theta = \theta_1 \cdots \theta_n$.

$P\colon G_0 \xrightarrow{\theta}\!\!\!\!\rightarrow G_n$ is called successful if $G_n = \Box$. We write $G_0 \longrightarrow\!\!\!\!\rightarrow fail$ if there exists no successful resolution of G_0. We use the abbreviation $G_0 \longrightarrow\!\!\!\!\rightarrow G_n$ when we do not need to emphasize the substitution θ in $G_0 \xrightarrow{\theta}\!\!\!\!\rightarrow G_n$. An unrestricted resolution is called a resolution if the unrestricted resolvents are resolvents. The composition $(\theta_1 \cdots \theta_n)\!\uparrow\! CVar(G_0)$ of the substitutions to the variables in the initial goal G_0 is called a computed answer substitution. For $1 \leq i \leq n$, the restriction of θ_i to the goal G_{i-1} (i.e. $\theta_i \!\uparrow\! CVar(G_{i-1})$) is denoted by $\theta_i^{\uparrow}$. Moreover, we write $\theta^{\uparrow}$ for $\theta_1^{\uparrow} \cdots \theta_n^{\uparrow}$.

In the rest of Section 5.3, we demonstrate resolution processes concerning the examples we have seen in Section 3. The sort and predicate hierarchies are expressed in sorted signatures. The facts in logic programs are described as clauses, and the queries are given by goals.

Example 5.1 *The program P_1 is the ordered pair (Σ_1, CS_1) of Σ_1 (in Example 4.1) and*

$$CS_1 = \{act\_of\_violence(actor \Rightarrow john{:}\, man) \leftarrow\}.$$

This fact $act\_of\_violence(actor \Rightarrow john{:}\, man)$ *means that "the actor John committed an act of violence." With respect to the program, the resolution of the goal* $illegal\_act(actor \Rightarrow john{:}\, man, vic \Rightarrow mary{:}\, woman)$ *fails as follows.*

$$illegal\_act(actor \Rightarrow john{:}\, man, vic \Rightarrow mary{:}\, woman) \longrightarrow fail$$

However, a successful resolution of the goal $illegal\_act(actor \Rightarrow john{:}\, man, vic \Rightarrow y{:}\, person)$ *can be obtained as follows.*

$$illegal\_act(actor \Rightarrow john{:}\, man, vic \Rightarrow y{:}\, person) \xrightarrow{\theta}_{R2} \square$$

In the resolution, $\theta = \{y{:}\, person/c{:}\, person\}$ *where* c *is a new supplement constant.*

Example 5.2 *The sorted signature* $\Sigma_2 = (\mathcal{S}, \mathcal{F}, \mathcal{P}, \mathcal{D})$ *comprises the following symbols:*

$$\begin{aligned}
\mathcal{S} &= \{person,\ man,\ woman,\ wallet,\ thing,\ \top,\ \bot\},\\
\mathcal{F} &= \{john,\ c_1\},\\
\mathcal{P}^{\bullet} &= \{act_of_violence,\ steal\},
\end{aligned}$$

and the declaration $\mathcal{D} = (\mathcal{D}^*_{\mathcal{S}}, \mathcal{D}_{\mathcal{F}}, \mathcal{D}_{\mathcal{P}})$*, where* $\mathcal{D}^*_{\mathcal{S}}$ *is the reflexive and transitive closure of* $\mathcal{D}_{\mathcal{S}}$*, constructed by*

$$\begin{aligned}
\mathcal{D}_{\mathcal{S}} = \{&\bot \sqsubset_S man,\ \bot \sqsubset_S woman,\ \bot \sqsubset_S wallet,\\
&man \sqsubset_S person,\ woman \sqsubset_S person,\ wallet \sqsubset_S thing,\\
&person \sqsubset_S \top,\ thing \sqsubset_S \top\},\\
\mathcal{D}_{\mathcal{F}} = \{&john{:}\, \langle man\rangle,\ c_1{:}\, \langle wallet\rangle\},\\
\mathcal{D}_{\mathcal{P}} = \{&steal \sqsubset_P illegal_act\} \cup\\
&\{steal{:}\, \{actor{:}\, person, obj{:}\, thing\},\\
&illegal_act{:}\, \{actor{:}\, person\}\}.
\end{aligned}$$

The argument structure of illegal_act consists of one argument (labeled with actor), and the predicate steal have two arguments (labeled with actor and obj). The program P_2 *is the ordered pair* (Σ_2, CS_2) *where*

$$CS_2 = \{steal(actor \Rightarrow john{:}\, man, obj \Rightarrow c_1{:}\, wallet) \leftarrow\}.$$

A successful resolution of $illegal\_act(actor \Rightarrow john\colon man)$ *is given in the program* P_2 *as follows.*

$$illegal\_act(actor \Rightarrow john\colon man) \xrightarrow{\epsilon}_{R2} \square$$

However, for the following goal there exist no successful resolutions.

$$illegal\_act(obj \Rightarrow c_1\colon wallet) \longrightarrow fail$$

More precisely, the goal does not belong the sorted formulas because obj is not defined as an argument of the predicate illegal_act.

Example 5.3 *The sorted signature* Σ_3 *comprises the following symbols:*

$$\begin{aligned}
\mathcal{S} &= \{person,\ man,\ woman,\ wallet,\ watch,\ thing,\ \top,\ \bot\},\\
\mathcal{F} &= \{john,\ tom,\ mary,\ c_1,\ c_2\},\\
\mathcal{P}^{\bullet} &= \{rob_with_violence,\ act_of_violence,\ steal,\ illegal_act\},
\end{aligned}$$

and the declaration $\mathcal{D} = (\mathcal{D}^*_{\mathcal{S}}, \mathcal{D}_{\mathcal{F}}, \mathcal{D}_{\mathcal{P}})$*, where* $\mathcal{D}^*_{\mathcal{S}}$ *is the reflexive and transitive closure of* $\mathcal{D}_{\mathcal{S}}$*, constructed by*

$$\begin{aligned}
\mathcal{D}_{\mathcal{S}} = \{&\bot \sqsubset_S man,\ \bot \sqsubset_S woman,\ \bot \sqsubset_S wallet,\ \bot \sqsubset_S watch,\\
&man \sqsubset_S person,\ woman \sqsubset_S person,\ wallet \sqsubset_S thing,\\
&watch \sqsubset_S thing,\ person \sqsubset_S \top,\ thing \sqsubset_S \top\},\\
\mathcal{D}_{\mathcal{F}} = \{&john\colon \langle man\rangle,\ tom\colon \langle man\rangle,\ mary\colon \langle woman\rangle,\\
&c_1\colon \langle wallet\rangle,\ c_2\colon \langle watch\rangle\},\\
\mathcal{D}_{\mathcal{P}} = \{&rob_with_violence \sqsubset_P act_of_violence,\ rob_with_violence \sqsubset_P steal,\\
&act_of_violence \sqsubset_P illegal_act,\ steal \sqsubset_P illegal_act\} \cup\\
&\{rob_with_violence\colon \{actor\colon person, vic\colon person, obj\colon thing\},\\
&act_of_violence\colon \{actor\colon person, vic\colon person\},\\
&steal\colon \{actor\colon person, obj\colon thing\},\\
&illegal_act\colon \{actor\colon person\}\}.
\end{aligned}$$

The argument structures of rob_with_violence and illegal_act consist of three arguments (labeled with actor, vic and obj) and one argument (labeled with actor). The predicates act_of_violence and steal have two arguments (labeled with actor, vic and actor, obj respectively). The program P_3 *is the ordered pair* (Σ_3, CS_3) *where*

$$\begin{aligned}
CS_3 = \{&act_of_violence^{\langle e_1\rangle}(actor \Rightarrow john\colon man, vic \Rightarrow mary\colon woman) \leftarrow,\\
&steal^{\langle e_1\rangle}(actor \Rightarrow john\colon man, obj \Rightarrow c_1\colon wallet) \leftarrow,\\
&steal^{\langle e_2\rangle}(actor \Rightarrow john\colon man, obj \Rightarrow c_2\colon watch) \leftarrow\}.
\end{aligned}$$

The first and second facts indicate that "the actor John committed an act of violence against the victim Mary at e_1" and that "the actor John stole the wallet c_1 at e_1." Additionally, the third fact expresses "the actor John stole the watch c_2 at e_2." With respect to the program, we have the following successful resolutions.

$$rob\_with\_violence^{\langle e_1 \rangle}(actor \Rightarrow john\colon man, vic \Rightarrow mary\colon woman, \\ obj \Rightarrow c_1\colon wallet) \xrightarrow{\epsilon}_{R3} \square$$

$$rob\_with\_violence^{\langle e_1 \rangle}(actor \Rightarrow x\colon person, vic \Rightarrow y\colon person, \\ obj \Rightarrow z\colon thing) \xrightarrow{\theta}_{R3} \square$$

In the second resolution, $\theta = \{x\colon person/john\colon man, y\colon person/mary\colon woman, z\colon thing/c_1\colon wallet\}$. However, for the following goals there exist no successful resolutions.

$$rob\_with\_violence^{\langle e_1 \rangle}(actor \Rightarrow john\colon man, vic \Rightarrow mary\colon woman, \\ obj \Rightarrow c_2\colon watch) \longrightarrow fail$$

$$rob\_with\_violence^{\langle e_1 \rangle}(actor \Rightarrow john\colon man, vic \Rightarrow tom\colon man, \\ obj \Rightarrow z\colon thing) \longrightarrow fail$$

5.4 Soundness and completeness of resolution

The soundness of the Horn clause resolution is proved as follows.

Theorem 5.1 (Soundness of resolution) *Let P be a program and G a goal. If there exists a successful resolution of G with a computed answer substitution θ, then $P \models_{H\Sigma} G\theta$.*

Proof. This theorem is proved by induction on the length n of a successful resolution. Let $P = (\Sigma, CS)$ be a program and let

$$P\colon G \xrightarrow{\theta_1} G_1 \xrightarrow{\theta_2} G_2 \xrightarrow{\theta_3} \cdots \xrightarrow{\theta_n} \square$$

with $(\theta_1 \cdots \theta_n)\!\uparrow\! CVar(G)$ be a successful resolution.
Base case: $n = 1$.

- If $G \xrightarrow{\theta_1}_{R1} \square$, then θ_1 is a unifier of $L' \leftarrow\in CS$ and $L(= G)$. Let $\mathcal{I}$ be an $H\Sigma$-model of P. Then $\mathcal{I} \models_{H\Sigma} \forall(L'\theta_1)$. Hence $P \models_{H\Sigma} G\theta_1$ since $L'\theta_1 = L\theta_1 (= G\theta_1)$.
- If $G \xrightarrow{\theta_1}_{R2} \square$, then $\varphi_q(\bar{\mu})\theta_1 = \sigma(\varphi_q(\bar{\mu}'))\theta_1$ where $G = \varphi_q(\bar{\mu})$ and $\varphi_p(\bar{\mu}') \leftarrow\in CS$. Let $\mathcal{I}$ be an $H\Sigma$-model of P. By Lemma 4.1, $\mathcal{I} \models_{H\Sigma} \forall\sigma(\varphi_q(\bar{\mu}'))$ since

$p \sqsubset_P q \in \mathcal{D}_{\mathcal{P}}$ and $\mathcal{I} \models_{H\Sigma} \forall\varphi_p(\bar{\mu}')$. Then $\mathcal{I} \models_{H\Sigma} \forall(\sigma(\varphi_q(\bar{\mu}'))\theta_1)$. Therefore, $P \models_{H\Sigma} G\theta_1$ since $\varphi_q(\bar{\mu})\theta_1 = \sigma(\varphi_q(\bar{\mu}'))\theta_1$.

- If $G \xrightarrow[R3]{\theta_1} \square$, then $q^{\langle e_i\rangle}(\bar{\mu})\theta_1 = \sigma(q^{\langle e_i\rangle}(\bar{\mu}'))\theta_1$ where $G = q^{\langle e_i\rangle}(\bar{\mu}), p_1^{\langle e_i\rangle}(\bar{\mu}_1) \leftarrow , \ldots, p_n^{\langle e_i\rangle}(\bar{\mu}_n) \leftarrow \in CS$ and $\mu' = \mu_1 \cup \cdots \cup \mu_n$. Let $\mathcal{I}$ be an $H\Sigma$-model of P. Since $q \sqsubset_P p_1, \ldots, q \sqsubset_P p_n \in \mathcal{D}_{\mathcal{P}} (n > 1)$ and $\mathcal{I} \models_{H\Sigma} \forall p_1^{\langle e_i\rangle}(\bar{\mu}_1), \ldots, \mathcal{I} \models_{H\Sigma} \forall p_n^{\langle e_i\rangle}(\bar{\mu}_n)$, we have $\mathcal{I} \models_{H\Sigma} \forall\sigma(q^{\langle e_i\rangle}(\bar{\mu}'))$ by Lemma 4.2. Then, $\mathcal{I} \models_{H\Sigma} \forall(\sigma(q^{\langle e_i\rangle}(\bar{\mu}'))\theta_1)$. Therefore, $P \models_{H\Sigma} G\theta_1$ since $q^{\langle e_i\rangle}(\bar{\mu})\theta_1 = \sigma(q^{\langle e_i\rangle}(\bar{\mu}'))\theta_1$.

Induction step: $n > 1$.

- If $G \xrightarrow[R1]{\theta_1} G_1$, then

  $$G'\theta_1 \cup (G - \{L\})\theta_1 \xrightarrow{\theta_2} G_2 \xrightarrow{\theta_3} \cdots \xrightarrow{\theta_n} \square$$

  is a resolution of $G_1 (= G'\theta_1 \cup (G - \{L\})\theta_1)$ where $L \in G$, $L' \leftarrow G' \in CS$ and $L'\theta_1 = L\theta_1$. By the induction hypothesis, $P \models_{H\Sigma} (G'\theta_1 \cup (G - \{L\})\theta_1)\theta'$ with $\theta' = \theta_2 \cdots \theta_n$. Then $P \models_{H\Sigma} L'\theta_1\theta' (= L\theta_1\theta')$ since $P \models_{H\Sigma} (L' \leftarrow G')\theta_1\theta'$. Hence $P \models_{H\Sigma} G\theta_1\theta'$.
- If $G \xrightarrow[R2]{\theta_1} G_1$, then

  $$G'\theta_1 \cup (G - \{\varphi_q(\bar{\mu})\})\theta_1 \xrightarrow{\theta_2} G_2 \xrightarrow{\theta_3} \cdots \xrightarrow{\theta_n} \square$$

  is a resolution of $G_1 (= G'\theta_1 \cup (G - \{\varphi_q(\bar{\mu})\})\theta_1)$ where $\varphi_q(\bar{\mu}) \in G$, $\varphi_p(\bar{\mu}') \leftarrow G' \in CS$, $\sigma(\varphi_q(\bar{\mu}'))\theta_1 = \varphi_q(\bar{\mu})\theta_1$ and $p \sqsubset_P q \in \mathcal{D}_{\mathcal{P}}$. By the induction hypothesis, $P \models_{H\Sigma} (G'\theta_1 \cup (G - \{\varphi_q(\bar{\mu})\})\theta_1)\theta'$ with $\theta' = \theta_2 \cdots \theta_n$. Then $P \models_{H\Sigma} \varphi_p(\bar{\mu}')\theta_1\theta'$ since $P \models_{H\Sigma} (\varphi_p(\bar{\mu}') \leftarrow G')\theta_1\theta'$. By Lemma 4.1, $P \models_{H\Sigma} \sigma(\varphi_q(\bar{\mu}'))\theta_1\theta' (= \varphi_q(\bar{\mu})\theta_1\theta')$. Hence $P \models_{H\Sigma} G\theta_1\theta'$.
- If $G \xrightarrow[R3]{\theta_1} G_1$, then

  $$(G_1 \cup \cdots \cup G_m)\theta_1 \cup (G - \{q^{\langle e_i\rangle}(\bar{\mu})\})\theta_1 \xrightarrow{\theta_2} G_2 \xrightarrow{\theta_3} \cdots \xrightarrow{\theta_n} \square$$

  is a resolution of $G_1 (= (G_1 \cup \cdots \cup G_m)\theta_1 \cup (G - \{q^{\langle e_i\rangle}(\bar{\mu})\})\theta_1)$ where $q^{\langle e_i\rangle}(\bar{\mu}) \in G$ and $p_1^{\langle e_i\rangle}(\bar{\mu}_1) \leftarrow G_1, \ldots, p_m^{\langle e_i\rangle}(\bar{\mu}_m) \leftarrow G_m \in CS$, $q^{\langle e_i\rangle}(\bar{\mu})\theta_1 = \sigma(q^{\langle e_i\rangle}(\bar{\mu}'))\theta_1$ with $\mu' = \mu_1 \cup \cdots \cup \mu_m$, and $q \sqsubset_P p_1, \cdots, q \sqsubset_P p_m \in \mathcal{D}_{\mathcal{P}}$. By the induction hypothesis, $P \models_{H\Sigma} ((G_1 \cup \cdots \cup G_m)\theta_1 \cup (G - \{q^{\langle e_i\rangle}(\bar{\mu})\})\theta_1)\theta'$ with $\theta' = \theta_2 \cdots \theta_n$. Then for $1 \leq j \leq m$, $P \models_{H\Sigma} p_j^{\langle e_i\rangle}(\bar{\mu}_j)\theta_1\theta'$ since $P \models_{H\Sigma} (p_j^{\langle e_i\rangle}(\bar{\mu}_j) \leftarrow G_j)\theta_1\theta'$. By Lemma 4.2, $P \models_{H\Sigma} \sigma(q^{\langle e_i\rangle}(\bar{\mu}'))\theta_1\theta' (= q^{\langle e_i\rangle}(\bar{\mu})\theta_1\theta')$. So $P \models_{H\Sigma} G\theta_1\theta'$.

According to $G\theta_1 \cdots \theta_n = G(\theta_1 \cdots \theta_n)\uparrow CVar(G)$, we see that $P \models_{H\Sigma} G\theta$ with $\theta = (\theta_1 \cdots \theta_n)\uparrow CVar(G)$. ∎

Examples 5.1, 5.2 and 5.3 show that the Horn clause resolution proposed yields the hierarchical reasoning of predicates (illustrated by the examples in

Section 3). Next, in order to make the Horn clause resolution complete, two resolution rules R2$^+$ and R3$^+$ obtained by modifying the resolution rules R2 and R3 must be complemented. This is because the rules R2 and R3 skip subdivided steps for derivations upon a predicate hierarchy. That is, the rule R2 is applied to $\varphi_q(\bar{\mu}) \in G$, $\varphi_p(\bar{\mu}') \leftarrow G' \in CS$ and $p \sqsubset_P q$, whereas R2$^+$ is applied to $\varphi_q(\bar{\mu}) \in G$ and $p \sqsubset_P q$. Also, the rule R3 is applied to $q^{\langle e_i \rangle}(\bar{\mu}) \in G$, $p_1^{\langle e_i \rangle}(\bar{\mu}_1) \leftarrow G_1, \ldots, p_n^{\langle e_i \rangle}(\bar{\mu}_n) \leftarrow G_n \in CS$ and $q \sqsubset_P^1 p_1, \ldots, q \sqsubset_P^1 p_n$, but R3$^+$ is applied to $q^{\langle e_i \rangle}(\bar{\mu}) \in G$ and $q \sqsubset_P^1 p_1, \ldots, q \sqsubset_P^1 p_n$. The concatenation of the rules R1 and R2$^+$ (resp. R1 and R3$^+$) yields the rule R2 (resp. R3).

We proceed to the definition of the resolution rules R2$^+$ and R3$^+$.

Definition 5.8 *Let $P = (\Sigma, CS)$ be a program and let $\langle \varphi_p, \varphi_q \rangle$ be $\langle p, q \rangle$, $\langle p^{\langle e_i \rangle}, q^{\langle e_i \rangle} \rangle$ or $\langle p^{\langle e_i \rangle}, q \rangle$.*

- **R2$^+$-resolution rule.** *Let G be a goal and let μ' be an argument set of p. If $p \sqsubset_P q \in \mathcal{D}_\mathcal{P}$ and θ is a unifier of $\varphi_q(\bar{\mu}) \in G$ and $\sigma(\varphi_q(\bar{\mu}'))$, then $(G - \{\varphi_q(\bar{\mu})\})\theta \cup \{\varphi_p(\bar{\mu}')\}\theta$ is an unrestricted resolvent of G with respect to $\varphi_q(\bar{\mu})$ and $\varphi_p(\bar{\mu}')$. We write*

$$G \xrightarrow{\theta}_{R2^+} (G - \{\varphi_q(\bar{\mu})\})\theta \cup \{\varphi_p(\bar{\mu}')\}\theta.$$

- **R3$^+$-resolution rule.** *Let G be a goal and let $\mu_1, \ldots, \mu_n$ be argument sets of $p_1, \ldots, p_n$. If $p_1, \ldots, p_n (n > 1)$ are all predicates such that $q \sqsubset_P^1 p_j \in \mathcal{D}_\mathcal{P}$, and θ is a unifier of $q^{\langle e_i \rangle}(\bar{\mu}) \in G$ and $\sigma(q^{\langle e_i \rangle}(\bar{\mu}'))$ where $\mu' = \mu_1 \cup \cdots \cup \mu_n$ is an argument set, then $(G - \{q^{\langle e_i \rangle}(\bar{\mu})\})\theta \cup \{p_1^{\langle e_i \rangle}(\bar{\mu}_1), \ldots, p_n^{\langle e_i \rangle}(\bar{\mu}_n)\}\theta$ is an unrestricted resolvent of G with respect to $q^{\langle e_i \rangle}(\bar{\mu})$ and $p_1^{\langle e_i \rangle}(\bar{\mu}_1), \ldots, p_n^{\langle e_i \rangle}(\bar{\mu}_n)$. We write*

$$G \xrightarrow{\theta}_{R3^+} (G - \{q^{\langle e_i \rangle}(\bar{\mu})\})\theta \cup \{p_1^{\langle e_i \rangle}(\bar{\mu}_1), \ldots, p_n^{\langle e_i \rangle}(\bar{\mu}_n)\}\theta.$$

We write $G \xrightarrow{\theta}_+ G'$ if $G \xrightarrow{\theta} G'$, $G \xrightarrow{\theta}_{R2^+} G'$, or $G \xrightarrow{\theta}_{R3^+} G'$. $P\colon L \xrightarrow{\theta}\!\!\!\!\rightarrow_+ G$ (or $L \longrightarrow\!\!\!\!\rightarrow_+ G$) denotes an unrestricted resolution with these rules. The soundness of the Horn clause resolution with the rules R2$^+$,R3$^+$ is proved as follows.

Theorem 5.2 (Soundness of resolution with R2$^+$, R3$^+$) *Let P be a program and G a goal. If there exists a successful resolution $P\colon G \longrightarrow\!\!\!\!\rightarrow_+ \square$ with a computed answer substitution θ, then $P \models_{H\Sigma} G\theta$.*

Proof. Similar to Theorem 5.1. ∎

As a prerequisite notion for the proof of the completeness, we define a derivation tree in a program P for a clause C as follows.

Definition 5.9 (Derivation tree) *Let $P = (\Sigma, CS)$ be a program, let C be a ground clause and let $\langle \varphi_p, \varphi_q \rangle$ be $\langle p, q \rangle$, $\langle p^{\langle e_i \rangle}, q^{\langle e_i \rangle} \rangle$ or $\langle p^{\langle e_i \rangle}, q \rangle$. A derivation tree in P for C is a finite labeled tree such that*

(1) the root is labeled with C,
(2) every node is labeled with a ground clause,
(3) every leaf is labeled with a clause in ground(P),
(4) every non-leaf node C_k is one of the following clauses:
  (a) $C_k = L_1 \leftarrow G_1 \cup G_2$ where its children are labeled with $L_1 \leftarrow G_1 \cup \{L\}$ and $L \leftarrow G_2$,
  (b) $C_k = \sigma(\varphi_q(\bar{\mu})) \leftarrow G$ where $p \sqsubset_P q$ and its child is labeled with $\varphi_p(\bar{\mu}) \leftarrow G$, and
  (c) $C_k = \sigma(q^{\langle e_i \rangle}(\bar{\mu})) \leftarrow G_1 \cup \cdots \cup G_n$ where μ is a ground argument set, $p_1, \ldots, p_n$ $(n > 1)$ are all predicates such that $q \sqsubset^1_P p_j$, and its children are labeled with $p_1^{\langle e_i \rangle}(\bar{\mu}_1) \leftarrow G_1$, $\ldots$, $p_n^{\langle e_i \rangle}(\bar{\mu}_n) \leftarrow G_n$ where $\mu = \mu_1 \cup \cdots \cup \mu_n$.

We write $P \vdash C$ if there exists a derivation tree in P for a clause C. To show the completeness of the Horn clause resolution, we construct a canonical interpretation $\mathcal{I}_P$ [16] that satisfies each atom derivable in a program P.

Definition 5.10 *Let P be a program and L an atom. A canonical interpretation $\mathcal{I}_P$ of P is an ordered triple $(M_H, \iota_{\mathcal{P}}, \alpha)$ such that*

(1) $M_H = (I_H, U_H)$ where
  (a) $U_H = TERM_0^+$,
  (b) $I_H(s) = TERM_{0,s}^+ (\subseteq U_H)$, for $s \in \mathcal{S}$,
  (c) $I_H(c) = c{:}\, s$, for $c \in \mathcal{F}_0$ with $c{:}\, \langle s \rangle \in \mathcal{D}_{\mathcal{F}}$,
  (d) $I_H(f)(t_1, \ldots, t_n) = f(I_H(t_1), \ldots, I_H(t_n)){:}\, s$, for $f \in \mathcal{F}_n$ with $f{:}\, \langle s_1, \ldots, s_n, s \rangle \in \mathcal{D}_{\mathcal{F}}$,
  (e) $I_H(p) \subseteq \{\rho \in (Arg(p) \rightarrow U_H) \mid \forall a_i \in Arg(p)[\rho(a_i) \in I_H(Scp(a_i))]\}$,
  (f) $I_H(p^{\langle e_i \rangle}) \subseteq I_H(p)$, for $p \in \mathcal{P}^{\bullet}$.
(2) $\iota_{\mathcal{P}}$ is a set of interpretations ι_q (for the argument manipulation) to all event predicates $q \in \mathcal{P}^{\bullet}$ such that

$$\iota_q(\rho) = (\rho \cap Arg(q) \times U_H) \cup \{(a_1, c_1{:}\, Scp(a_1)), \ldots, (a_n, c_n{:}\, Scp(a_n))\}$$

where ρ is an argument interpretation, $\{a_1, \ldots, a_n\} = Arg(q) - ls^*(\rho)$ and $c_1, \ldots, c_n$ are the supplement constants introduced in $\sigma(\varphi_q(\bar{\mu}))$ with $\mu = \{a \Rightarrow t \mid (a, t) \in \rho\}$.
(3) $\mathcal{I}_P \models_{H\Sigma} L$ iff there exists a derivation tree in P for $L \leftarrow$.

The next lemma shows that the canonical interpretation $\mathcal{I}_P$ satisfies P (i.e. it is an $H\Sigma$-model of P).

Lemma 5.1 *Let P be a program. A canonical interpretation $\mathcal{I}_P$ of P is an*

HΣ-model of P.

Proof. In order to prove that $\mathcal{I}_P$ is a model of P, we show $\mathcal{I}_P \models_{H\Sigma} L \leftarrow G$ for all clauses $L \leftarrow G$ in $P = (\Sigma, CS)$. Let $L \leftarrow G \in CS$ and let θ be a ground substitution for $L \leftarrow G$. Suppose that $\mathcal{I}_P \models_{H\Sigma} L_1\theta \wedge \ldots \wedge L_n\theta$ with $G = \{L_1, \ldots, L_n\}$. By the definition of $\mathcal{I}_P$, for $1 \leq i \leq n$, $P \vdash L_i\theta \leftarrow$. Then $P \vdash L\theta \leftarrow L_1\theta \wedge \ldots \wedge L_n\theta$ because $L\theta \leftarrow L_1\theta \wedge \ldots \wedge L_n\theta \in \mathit{ground}(P)$, and thus $P \vdash L\theta \leftarrow$ by Definition 5.9. So $\mathcal{I}_P \models_{H\Sigma} L\theta$. Hence $\mathcal{I}_P \models_{H\Sigma} (L \leftarrow G)\theta$ is proved. $\mathcal{I}_P \models_{H\Sigma} L \leftarrow G$ iff $\mathcal{I}_P \models_{H\Sigma} \mathit{ground}(L \leftarrow G)$ iff $\mathcal{I}_P \models_{H\Sigma} (L \leftarrow G)\theta$ for all ground substitutions θ for $L \leftarrow G$. It follows that $\mathcal{I}_P \models_{H\Sigma} L \leftarrow G$.

Next, we have to show that $\mathcal{I}_P$ is an $H\Sigma$-interpretation. Let $\mathcal{I}_P = (M_H, \iota_{\mathcal{P}}, \alpha)$. For $p \sqsubset_P q \in \mathcal{D}_{\mathcal{P}}$ and $\rho \in I_H(\varphi_p)$, by Definition 5.10, $\iota_q(\rho) = (\rho \cap Arg(q) \times U_H) \cup \{(a_1, c_1\colon Scp(a_1)), \ldots, (a_k, c_k\colon Scp(a_k))\}$ where $Arg(q) - ls^*(\rho) = \{a_1 \ldots, a_k\}$. Now by Definition 4.12 and $\rho \in I_H(\varphi_p)$, we have $\mathcal{I}_P \models_{H\Sigma} \varphi_p(\bar{\mu}_\rho)$ with $\mu_\rho = \{a \Rightarrow t \mid (a,t) \in \rho\}$. Then $P \vdash \varphi_p(\bar{\mu}_\rho) \leftarrow$ by the definition of $\mathcal{I}_P$. Hence $P \vdash \sigma(\varphi_q(\bar{\mu}_\rho)) \leftarrow$ by Definition 5.9 and $p \sqsubset_P q$, and thus $\mathcal{I}_P \models_{H\Sigma} \sigma(\varphi_q(\bar{\mu}_\rho))$. Hence $(\rho \cap (Arg(q) \times U_H)) \cup \{(a_1, c_1\colon Scp(a_1)), \ldots, (a_k, c_k\colon Scp(a_k))\} \in I_H(\varphi_q)$.

For $q \sqsubset_P p_1, \ldots, q \sqsubset_P p_n \in \mathcal{D}_{\mathcal{P}}(n > 1)$ and for $\rho \in I_H(p_1^{\langle e_i \rangle}); \cdots; I_H(p_n^{\langle e_i \rangle})$, we have $\iota_q(\rho) = (\rho \cap Arg(q) \times U_H) \cup \{(a_1, c_1\colon Scp(a_1)), \ldots, (a_k, c_k\colon Scp(a_k))\}$ where $Arg(q) - ls^*(\rho) = \{a_1 \ldots, a_k\}$. By Definitions 4.9 and 4.12 and by $\rho \in I_H(p_1^{\langle e_i \rangle}); \cdots; I_H(p_n^{\langle e_i \rangle})$, $\mathcal{I}_P \models p_j^{\langle e_i \rangle}(\bar{\mu}_j)$ for $1 \leq j \leq n$ where $\mu_1 \cup \cdots \cup \mu_n = \{a \Rightarrow t \mid (a,t) \in \rho\}$. Then $P \vdash p_j^{\langle e_i \rangle}(\bar{\mu}_j) \leftarrow$ for $1 \leq j \leq n$ by the definition of $\mathcal{I}_P$. By Definition 5.9 and $q \sqsubset_P p_1, \ldots, q \sqsubset_P p_n$, $P \vdash \sigma(q^{\langle e_i \rangle}(\bar{\mu}_\rho)) \leftarrow$ where $\mu_\rho = \mu_1 \cup \cdots \cup \mu_n$. Then $\mathcal{I}_P \models \sigma(q^{\langle e_i \rangle}(\bar{\mu}_\rho))$. So $(\rho \cap (Arg(q) \times U_H)) \cup \{(a_1, c_1\colon Scp(a_1)), \ldots, (a_k, c_k\colon Scp(a_k))\} \in I_H(q^{\langle e_i \rangle})$. ∎

Lemma 5.2 *Let $P = (\Sigma, CS)$ be a program and $L \leftarrow G$ a ground clause. If $P \vdash L \leftarrow G$, then $P\colon G \longrightarrow\!\!\!\!\!\rightarrow_+ \square$ implies $P\colon L \longrightarrow\!\!\!\!\!\rightarrow_+ \square$.*

Proof. We prove this lemma by induction on the height n of a derivation tree of $P \vdash L \leftarrow G$.
Base case: $n = 1$.
Since $L \leftarrow G \in \mathit{ground}(P)$, there must be a clause $L' \leftarrow G' \in CS$ such that $(L' \leftarrow G')\theta = L \leftarrow G$. Hence if $P\colon G \longrightarrow\!\!\!\!\!\rightarrow_+ \square$, then $P\colon L \xrightarrow{\theta}_{R1} G \longrightarrow\!\!\!\!\!\rightarrow_+ \square$.
Induction step: $n > 1$.

- If $L \leftarrow G$ has only the children $L \leftarrow G' \cup \{L'\}$ and $L' \leftarrow G''$ where $G = G' \cup G''$, then

  $$P \vdash L \leftarrow G' \cup \{L'\} \textit{ and } P \vdash L' \leftarrow G''.$$

  By the induction hypothesis, $P\colon G' \cup \{L'\} \longrightarrow\!\!\!\!\!\rightarrow_+ \square$ implies $P\colon L \longrightarrow\!\!\!\!\!\rightarrow_+ \square$, and

$P\colon G'' \longrightarrow\!\!\!\!\!\rightarrow_+ \Box$ implies $P\colon L' \longrightarrow\!\!\!\!\!\rightarrow_+ \Box$. If $P\colon G' \cup G'' \longrightarrow\!\!\!\!\!\rightarrow_+ \Box$, then $P\colon G' \cup \{L'\} \longrightarrow\!\!\!\!\!\rightarrow_+ \Box$. Hence $P\colon L \longrightarrow\!\!\!\!\!\rightarrow_+ \Box$.

- If $L \leftarrow G$ with $L = \varphi_q(\bar{\mu})$ has only the child $\varphi_p(\bar{\mu}') \leftarrow G$ where $p \sqsubset_P q$ and $\varphi_q(\bar{\mu}) = \sigma(\varphi_q(\bar{\mu}'))$, then

  $$P \vdash \varphi_p(\bar{\mu}') \leftarrow G.$$

  By the induction hypothesis, $P\colon G \longrightarrow\!\!\!\!\!\rightarrow_+ \Box$ implies $P\colon \varphi_p(\bar{\mu}') \longrightarrow\!\!\!\!\!\rightarrow_+ \Box$. So, we can obtain the resolution $P\colon \varphi_q(\bar{\mu}) \xrightarrow{\epsilon}_{R2^+} \varphi_p(\bar{\mu}')$ since $\varphi_p(\bar{\mu}')$ is ground. Hence if $P\colon G \longrightarrow\!\!\!\!\!\rightarrow_+ \Box$, then $P\colon \varphi_q(\bar{\mu}) \xrightarrow{\epsilon}_{R2^+} \varphi_p(\bar{\mu}') \longrightarrow\!\!\!\!\!\rightarrow_+ \Box$.
- If $L \leftarrow G$ with $L = q^{\langle e_i \rangle}(\bar{\mu})$ has only the children $p_1^{\langle e_i \rangle}(\bar{\mu}_1) \leftarrow G_1, \ldots, p_m^{\langle e_i \rangle}(\bar{\mu}_m) \leftarrow G_m$ where $G = G_1 \cup \cdots \cup G_m$, $p_1, \ldots, p_m$ are all predicates such that $q \sqsubset_P^1 p_j$, and $q^{\langle e_i \rangle}(\bar{\mu}) = \sigma(q^{\langle e_i \rangle}(\bar{\mu}'))$ with $\mu' = \mu_1 \cup \cdots \cup \mu_m$, then

  $$P \vdash p_1^{\langle e_i \rangle}(\bar{\mu}_1) \leftarrow G_1, \ldots, P \vdash p_m^{\langle e_i \rangle}(\bar{\mu}_m) \leftarrow G_m.$$

  By the induction hypothesis, for $1 \leq j \leq m$, $P\colon G_j \longrightarrow\!\!\!\!\!\rightarrow_+ \Box$ implies $P\colon p_j^{\langle e_i \rangle}(\bar{\mu}_j) \longrightarrow\!\!\!\!\!\rightarrow_+ \Box$. Now there exists the resolution $P\colon q^{\langle e_i \rangle}(\bar{\mu}) \xrightarrow{\epsilon}_{R3^+} \{p_1^{\langle e_i \rangle}(\bar{\mu}_1), \ldots, p_m^{\langle e_i \rangle}(\bar{\mu}_m)\}$ since $p_1^{\langle e_i \rangle}(\bar{\mu}_1), \ldots, p_m^{\langle e_i \rangle}(\bar{\mu}_m)$ are ground. So if $P\colon G_1 \cup \cdots \cup G_m \longrightarrow\!\!\!\!\!\rightarrow_+ \Box$, then $P\colon q^{\langle e_i \rangle}(\bar{\mu}) \xrightarrow{\epsilon}_{R3^+} \{p_1^{\langle e_i \rangle}(\bar{\mu}_1), \ldots, p_m^{\langle e_i \rangle}(\bar{\mu}_m)\} \longrightarrow\!\!\!\!\!\rightarrow_+ \Box$. ∎

The completeness of the Horn clause resolution (with the rules $R2^+$, $R3^+$) for ground goals is proved as follows.

Theorem 5.3 (Ground completeness of resolution with $R2^+$, $R3^+$) *Let P be a program and G a ground goal. If $P \models_{H\Sigma} G$, then there exists a successful resolution $P\colon G \longrightarrow\!\!\!\!\!\rightarrow_+ \Box$.*

Proof. Suppose that $P \models_{H\Sigma} \{L_1, \ldots, L_n\}$ where $L_1, \ldots, L_n$ are ground. Since $\mathcal{I}_P \models_{H\Sigma} P$ we have $\mathcal{I}_P \models_{H\Sigma} L_1, \ldots, \mathcal{I}_P \models_{H\Sigma} L_n$. Then by Definition 5.10, $P \vdash L_1 \leftarrow, \ldots, P \vdash L_n \leftarrow$. So $P\colon L_1 \longrightarrow\!\!\!\!\!\rightarrow_+ \Box, \ldots, P\colon L_n \longrightarrow\!\!\!\!\!\rightarrow_+ \Box$ by Lemma 5.2. This derives that there exists $P\colon \{L_1, \ldots, L_n\} \longrightarrow\!\!\!\!\!\rightarrow_+ \Box$. ∎

The following lemma is needed to prove the completeness of the Horn clause resolution for general goals.

Lemma 5.3 (Lifting) *Let P be a program. If P has an unrestricted resolution*

$$P\colon G_0\theta_0 \xrightarrow{\theta_1}_+ G_1 \xrightarrow{\theta_2}_+ G_2 \xrightarrow{\theta_3}_+ \cdots \xrightarrow{\theta_n}_+ G_n,$$

then P has a resolution

$$P\colon G_0 \xrightarrow{\theta'_1}_+ G'_1 \xrightarrow{\theta'_2}_+ G'_2 \xrightarrow{\theta'_3}_+ \cdots \xrightarrow{\theta'_n}_+ G'_n,$$

where (i) $\gamma_0 = \theta_0$ *and, for* $1 \leq i \leq n$, $(\gamma_{i-1} \uparrow CVar(G'_{i-1}))\theta_i = \theta'_i\gamma_i$ *and* $G_i = G'_i\gamma_i$*, and (ii) there exists a substitution* γ'_n *such that* $G_0\theta_0\theta_1^{\uparrow}\cdots\theta_n^{\uparrow} = G_0\theta'^{\uparrow}_1\cdots\theta'^{\uparrow}_n\gamma'_n$.

Proof. This lemma is proved by induction on the length n of an unrestricted resolution of $G_0\theta_0$.
$n = 1$: There exists $G_0\theta_0 \xrightarrow{\theta_1}_+ G_1$ with respect to $L \in G_0\theta_0$ and clauses $C_1, \ldots, C_m$. For $1 \leq i \leq m$, $C_i(\theta_0 \uparrow CVar(G_0)) = C_i$ where $CVar(G_0) \cap CVar(C_i) = \emptyset$, and thus there exists $G_0 \xrightarrow{(\theta_0 \uparrow CVar(G_0))\theta_1}_+ G_1$. Hence we have a resolution $G_0 \xrightarrow{\theta'_1}_+ G'_1$ with respect to L and $C_1, \ldots, C_m$ where there exists a substitution γ_1 such that $(\theta_0 \uparrow CVar(G_0))\theta_1 = \theta'_1\gamma_1$ and $G_1 = G'_1\gamma_1$.
$n > 1$: By the induction hypothesis, there exists a resolution

$$P\colon G_0 \xrightarrow{\theta'_1}_+ G'_1 \xrightarrow{\theta'_2}_+ G'_2 \xrightarrow{\theta'_3}_+ \cdots \xrightarrow{\theta'_{n-1}}_+ G'_{n-1}$$

where $\gamma_0 = \theta_0$ and, for $1 \leq i \leq n-1$, there exists a substitution γ_i such that $(\gamma_{i-1} \uparrow CVar(G_{i-1}))\theta_i = \theta'_i\gamma_i$ and $G_i = G'_i\gamma_i$. By assumption, we have $P\colon G_{n-1} \xrightarrow{\theta_n}_+ G_n$ with respect to $L \in G_{n-1}$ and clauses $C_1, \ldots, C_m$ where $G_{n-1} = G'_{n-1}\gamma_{n-1}$. For $1 \leq i \leq m$, $C_i(\gamma_{n-1} \uparrow CVar(G'_{n-1})) = C_i$ where $CVar(G'_{n-1}) \cap CVar(C_i) = \emptyset$, and hence we have $G'_{n-1} \xrightarrow{(\gamma_{n-1} \uparrow CVar(G'_{n-1}))\theta_n}_+ G_n$. This yields the resolution $G'_{n-1} \xrightarrow{\theta'_n}_+ G'_n$ with respect to L and $C_1, \ldots, C_m$ where there exists a substitution γ_n such that $(\gamma_{n-1} \uparrow CVar(G'_{n-1}))\theta_n = \theta'_n\gamma_n$ and $G_n = G'_n\gamma_n$.

(ii) $G_0\theta_0\theta_1^{\uparrow}\cdots\theta_n^{\uparrow} = G_0\theta'^{\uparrow}_1\cdots\theta'^{\uparrow}_n\gamma'_n$ can be proved by the method used in Theorem 5.37 in [12]. ∎

In the following theorem, we show that the Horn clause resolution (with the rules $R2^+$, $R3^+$) is complete.

Theorem 5.4 (Completeness of resolution with R2$^+$, R3$^+$) *Let P be a program, G a goal and θ a sorted substitution. If $P \models_{H\Sigma} G\theta$, then there exists a successful resolution $P\colon G \xrightarrow{\theta'}\!\!\!\!\rightarrow_+ \square$ with $G\theta = G\theta'^{\uparrow}\gamma$.*

Proof. Let the substitution $\beta = \{x_1\colon s_1/c_{x_1:s_1}\colon s_1, \ldots, x_n\colon s_n/c_{x_n:s_n}\colon s_n\}$ where $CVar(G\theta) = \{x_1\colon s_1, \ldots, x_n\colon s_n\}$ and $c_{x_1:s_1}, \ldots, c_{x_n:s_n}$ are new constants. If $P \models_{H\Sigma} G\theta$, then $P \models_{H\Sigma} G\theta\beta$. So by Theorem 5.3, there exists a unrestricted resolution $P\colon G\theta\beta \xrightarrow{\delta}\!\!\!\!\rightarrow_+ \square$. By Lemma 5.3, we have a resolution $P\colon G \xrightarrow{\theta'}\!\!\!\!\rightarrow_+ \square$ with $G\theta\beta\delta^{\uparrow} = G\theta'^{\uparrow}\gamma$. Moreover, since $G\theta\beta$ is ground, $G\theta\beta\delta^{\uparrow} = G\theta\beta$. Because $c_{x_1:s_1}\colon s_1, \ldots, c_{x_n:s_n}\colon s_n$ do not occur in $G\theta'$, we have $y_1\colon s_1/c_{x_1:s_1}, \ldots, y_n\colon s_n/c_{x_n:s_n} \in \gamma$. Let γ_0 be defined by $(y_i\colon s_i)\gamma_0 = x_i\colon s_i$ for $1 \leq i \leq n$, and let $\gamma' = (\gamma - \{y_1\colon s_1/c_{x_1:s_1}, \ldots, y_n\colon s_n/c_{x_n:s_n}\}) \cup \gamma_0$. It follows that $G\theta = G\theta'^{\uparrow}\gamma'$. ∎

6 Related work

The logical system presented in this paper is related to an extension of order-sorted logics and typed (sorted) logic programming languages, for practical knowledge representation.

Beierle et al. [8] developed an order-sorted logic to combine taxonomical knowledge and assertional knowledge in knowledge representation systems. In the logic, sorts s can be used to denote not only the types of terms (e.g. $x\colon s$ and $c\colon s$) but also unary predicates (e.g. $s(t)$), called *sort predicates* [8,20]. Using this notion, we can derive formulas with sort predicates (as assertional knowledge) from sort-hierarchies (as taxonomical knowledge). For example, a subsort relation $s_1 \leq s_2$ implies the formula $s_1(x) \rightarrow s_2(x)$ with sort predicates s_1, s_2. Consequently, a sorted resolution system was extended by adding inference rules concerning subsort relations and sort predicates. On the other hand, Frisch [14] proposed an order-sorted logic that contains a sort theory (instead of a sort signature) to describe sort information in first-order logic. A sort theory is a set of formulas constructed only by sort predicates. In addition to a subsort relation $s_1 \leq s_2$ (represented by the formula $s_1(x) \rightarrow s_2(x)$) it can describe more complicated sort information (e.g. $s_1(x) \wedge s_2(x) \rightarrow \neg s_3(x)$). However, neither approach deals with a hierarchy of n-ary predicates and manipulating arguments in the predicates as this paper proposes. The sort predicates and a sort-hierarchy only correspond to a hierarchy of unary predicates.

The logic programming language LOGIN is equipped with typed terms including feature structures (called ψ-terms), which can represent what are expressible by predicates in ordinary logic programming. For example, the predicate symbol *apple* (used in the formula $apple(x)$) can be represented as a type in the following ψ-term.

$$X\colon apple[taste \Rightarrow sour; color \Rightarrow red]$$

which expresses "sour red apples." Such types are ordered and build a class-hierarchy, together with feature structures (such as $[taste \Rightarrow sour; color \Rightarrow red]$ in the above ψ-term) that give us expressive types being able to describe more specific types with attributes.

In the legal reasoning system New HELIC-II developed by Nitta et al., a typed logic programming language was used as an inference engine for legal reasoning. The language provides term expressions (called H-terms) obtained by extending ψ-terms in LOGIN. In legal reasoning systems, the description of a legal affair consisting of events is needed from which legal results are inferred. In addition to class-hierarchies limited to represent nominal concepts,

New HELIC-II allows us to represent a hierarchy of verbal concepts indicating events. The hierarchical reasoning for the verbal concepts is based on the fact that informative verbal concepts result in general verbal concepts. For example, an event *illegal_acting* is derived from a more informative event *acting_of_violence*. Obviously, it is distinguished from reasoning in a hierarchy of nominal concepts (as a sort-hierarchy). For formalizing the logic programming language in New HELIC-II, our work provides a theoretical foundation of an order-sorted logic that is extended by incorporating a predicate hierarchy corresponding to a hierarchy of verbal concepts.

Furthermore, the argument manipulation proposed in this paper is based on the work in [22]. The authors presented a way to supplement missing arguments for the event and property aspects of assertions as follows.

Event assertion:
$hit(actor \Rightarrow john : man) \longrightarrow hit(actor \Rightarrow john : man, \underline{vic \Rightarrow c{:}\, person})$
"John hit a person."
Property assertion:
$hit(actor \Rightarrow john : man) \longrightarrow hit(actor \Rightarrow john : man, \underline{vic \Rightarrow x{:}\, person})$
"John has the property of being able to hit any person."

The supplemented arguments $c{:}\, person$ and $x{:}\, person$ are interpreted as a person and any person respectively. By adopting the supplementation for event assertions, our work formalizes a sorted logic programming language with predicate hierarchy. In the area of databases, there is a well-known approach to deal with incomplete information, related to missing arguments. It introduces null values for representing missing information in databases [38,1]. Compared with this approach, the argument manipulation contains two new ideas. First, it distinguishes supplemented arguments in the event and property aspects of predicates. Existential and universal terms are supplemented to event and property assertions respectively. Secondly, it uses sorted terms for supplemented arguments that are differently quantified and restricts each domain by sorts. The sorts in supplemented arguments are determined by each argument role and their restricted domains result in adequate supplementation. Hence, we can say that order-sorted logic is a useful tool to express supplemented arguments, not only a sort-hierarchy. The argument manipulation is applied to the hierarchical reasoning of predicates and is operated by sorted terms in a sort-hierarchy. In other words, our inference method for predicate hierarchies is actualized by interacting the two kinds of hierarchies and the argument manipulation with sorted terms.

7 Conclusions and future work

This paper has presented an order-sorted logic programming language that is extended by a reasoning mechanism for a predicate hierarchy, in addition to substitutions in a sort-hierarchy. As a generalized language for structural knowledge, it can enrich hierarchical reasoning, namely, it enables us to derive general and concrete expressions in the two kinds of hierarchies. In particular, hierarchical reasoning of predicates enhances the usefulness and the feasibility of logical knowledge representation systems, such as representing event assertions in legal reasoning. The inference machinery for deriving general and concrete predicates that allows for various argument structures is obtained by including an argument manipulation that follows the event aspect of predicates. By embedding this new manipulation in the inference rules proposed: specialization and generalization rules for hierarchical predicates, we are able to deal successfully with derivations of flexibly argumented and hierarchical predicates (i.e. we can set various argument structures for predicates in the hierarchy) in logic programming. Specifically, we have developed a Horn clause resolution system equipped with the notion of a predicate hierarchy. In the semantics of this language, the predicate hierarchy is interpreted in the class of restricted Σ-structures (called $H\Sigma$-structures). The semantic models ensure the soundness and the completeness of the resolution for the extended order-sorted logic with sort and predicate hierarchies.

We believe that further research is needed on the meaning of negation derived from the event aspect of predicates. Due to the event and property aspects of predicates [22], negative assertions do not always have uniform interpretation and reasoning. If the negation of an event means an opposite and disjointed event (which we call negative event), then its meaning is stronger than the negation of a property. Hence, in order to derive general predicates in a hierarchy, differently quantified arguments must be supplemented to negative event assertions and the negation of assertions in the argument manipulation. For these assertions, strong negation (proposed in constructive logic [4]) is a prime candidate to represent the negative event assertions. By introducing this strong negation with classical negation, we can formalize the diversity of negations in event and property assertions, and develop an inference system for full formulas or general causal forms in our proposed logic.

Acknowledgments

The author is grateful to Satoshi Tojo, Katsumi Nitta, Hajime Ishihara and Robert A. Kowalski for their valuable comments and suggestions.

References

[1] S. Abiteboul, R. Hull, V. Vianu, Foundations of Databases, Addison-Wesley, 1995.

[2] H. Aït-Kaci, R. Nasr, LOGIN: A logic programming language with built-in inheritance, Journal of Logic Programming 3 (3) (1986) 185–215.

[3] H. Aït-Kaci, A. Podelski, Towards a meaning of LIFE, Journal of Logic Programming 16 (3) (1993) 195–234.

[4] S. Akama, Constructive predicate logic with strong negation and model theory, Notre Dame Journal of Formal Logic 29 (1) (1988) 18–27.

[5] R. Al-Asady, Inheritance Theory: An Artificial Intelligence Approach, Ablex Publishing Corporation, 1995.

[6] J. F. Allen, Towards a general theory of action and time, Artificial Intelligence 23 (1984) 123–154.

[7] K. R. Apt, From Logic Programming to Prolog, Prentice-Hall, 1997.

[8] C. Beierle, U. Hedtsück, U. Pletat, P. Schmitt, J. Siekmann, An order-sorted logic for knowledge representation systems, Artificial Intelligence 55 (1992) 149–191.

[9] B. Carpenter, The Logic of Typed Feature Structure, Cambridge University Press, 1992.

[10] A. G. Cohn, A more expressive formulation of many sorted logic, Journal of Automated Reasoning 3 (1987) 113–200.

[11] A. G. Cohn, Taxonomic reasoning with many sorted logics, Artificial Intelligence Review 3 (1989) 89–128.

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

[13] H. B. Enderton, A Mathematical Introduction to Logic, Academic Press, 1972.

[14] A. M. Frisch, The substitutional framework for sorted deduction: fundamental results on hybrid reasoning, Artificial Intelligence 49 (1991) 161–198.

[15] J. H. Gallier, Logic for Computer Science. Foundations of Automatic Theorem Proving, Harper & Row, 1986.

[16] M. Hanus, Logic programming with type specifications, in: F. Pfenning (Ed.), Types in Logic Programming, The MIT Press, 1992.

[17] J. Herbrand, in: W. D. Goldfarb (Ed.), Logical Writings, Harvard University Press, 1971.

[18] P. M. Hill, R. W. Topor, A semantics for typed logic programs, in: F. Pfenning (Ed.), Types in Logic Programming, The MIT Press, 1992.

[19] K. Kaneiwa, An order-sorted logic with predicate hierarchy, eventuality, and implicit negation, Ph.D. thesis, Japan Advanced Institute of Science and Technology (2001).

[20] K. Kaneiwa, The completeness of logic programming with sort predicates, Systems and Computers in Japan 35 (1) (2004) 37–46.

[21] K. Kaneiwa, R. Mizoguchi, Ontological knowledge base reasoning with sort-hierarchy and rigidity, in: Proceedings of the Ninth International Conference on the Principles of Knowledge Representation and Reasoning (KR2004), 2004.

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

[23] K. Kaneiwa, S. Tojo, An order-sorted resolution with implicitly negative sorts, in: Proceedings of the 2001 Int. Conf. on Logic Programming, Springer-Verlag, 2001, pp. 300–314, LNCS 2237.

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

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

[26] M. Manzano, Introduction to many-sorted logic, in: Many-sorted Logic and its Applications, John Wiley and Sons, 1993, pp. 3–86.

[27] A. Martelli, U. Montanari, An efficient unification algorithm, ACM Trans. Programm. Languages Systems 4 (1982) 258–282.

[28] D. V. McDermott, A temporal logic for reasoning about processes and plans, Cognitive Science 6 (1982) 101–155.

[29] K. Nitta, et al., Knowledge representation of new HELIC-II, in: Workshop on Legal Application of Logic Programming,ICLP '94, 1994.

[30] K. Nitta, et al., New HELIC-II: a software tool for legal reasoning, in: Proceedings of 5th Int. Conft. on Artificial Intelligence and Law, ACM Press, 1995.

[31] A. Oberschelp, Untersuchungen zur mehrsortigen quantorelogik, Mathematische Annalen 145 (1962) 297–333.

[32] A. Oberschelp, Order sorted predicate logic, in: Workshop on Sorts and Types in Artificial Intelligence, 1989.

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

[34] Y. Shoham, Reasoning about Change, The MIT Press, 1988.

[35] G. Smolka, Feature-constraint logics for unification grammars, Journal of Logic Programming 12 (1992) 51–87.

[36] R. Socher-Ambrosius, P. Johann, Deduction Systems, Springer-Verlag, 1996.

[37] D. S. Touretzky, The Mathematics of Inheritance Systems, Pitman, 1986.

[38] R. van der Meyden, Logical approaches to incomplete information: A survey, in: J. Chomicki, G. Saake (Eds.), Logics for Databases and Information Systems, Kluwer academic publishers, 1998, pp. 307–336.

[39] C. Walther, A mechanical solution of Schuber's steamroller by many-sorted resolution, Artificial Intelligence 26 (2) (1985) 217–224.

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

[41] C. Walther, Many-sorted unification, Journal of the Association for Computing Machinery 35 (1988) 1.

[42] H. Wang, Logic of many-sorted theories, Journal of Symbolic Logic 3 (1952) 105–116.

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

[44] H. Yasukawa, H. Tsuda, K. Yokota, Objects, properties, and modules in $\mathcal{QUIXOTE}$, in: Proc. FGCS'92, 1992, pp. 257–268.

[45] K. Yokota, Quixote: A constraint based approach to a deductive object-oriented database, Ph.D. thesis, Kyoto University (1994).