An Order-Sorted Logic with Predicate-Hierarchy, Eventuality and Implicit Negation

by

KEN KANEIWA

submitted to
Japan Advanced Institute of Science and Technology
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

Supervisor: Professor Satoshi Tojo

School of Information Science
Japan Advanced Institute of Science and Technology

March 2001

Abstract

Order-sorted logics (or many-sorted logics) do not only have rigorous theoretic foundations but also can represent structured information by the sort symbols in a hierarchy, called a sort-hierarchy. This thesis presents the following contributions to the use of order-sorted logic as a knowledge representation language.

First, we propose an extended order-sorted logic as a knowledge representation language that possesses both hierarchies of sorts and predicates. This logic must delete and supplement the arguments of each predicate in order to allow it to derive superordinate and subordinate predicates (each with different argument structure) from the other predicates in a predicate-hierarchy (as distinguished from a sort-hierarchy). Thus, we introduce a Horn clause resolution extended to include inference rules of predicate-hierarchy. These rules include the manipulation of arguments in which the supplementary arguments are quantified differently, depending on whether a predicate is interpreted as an occurrence of an event or a universal property. For the extended logic with a distinction between events and properties in predicates, we give a restricted semantic model by interpreting the constraints on predicate-hierarchy and the manipulation of arguments. Furthermore, we prove the soundness and completeness of this Horn-clause resolution with both hierarchies of sorts and predicates.

Secondly, we present a hybrid inference system which can deal with the properties of implicitly negative sorts in an assertional knowledge base separated from the sort-hierarchy. This solves the problem of the knowledge base being unable to appropriately use a sort as negative information, despite the fact that sorts with negative meaning may be implicitly included in a sort-hierarchy. These implicit negations, called lexical negations in linguistics, are classified as (i) negative affix or (ii) lexicon with negative meaning. Lexical negations are distinct from the negative particle 'not'. We propose the notions of structured sorts, sort relations, and contradiction in a sort-hierarchy. These can define the relationship between the implicit negations and the classical negation, and can declare the exclusivity and the totality of each negation and its affirmation. In this thesis, we present a method of regarding the negative affix as an operator based on strong negation and the lexicon with negative meaning as a sort exclusive to its antonymous sort in the hierarchy. In order to infer from these negations, we combine a structured sort constraint system into a clausal inference system.

Acknowledgments

The author wishes to express his sincere gratitude to his principal advisor Professor Satoshi Tojo of Japan Advenced Institute of Science and Techology for his constant support and kind guidance during this work. The author also wishes to express his thanks to Associate Professor Hajime Ishihara and Professor Tetsuo Asano of Japan Advanced Institute of Science and Technology for their suggestions and continuous encouragements. The lectures of Professor Hiroakira Ono of Japan Advanced Institute of Science and Technology have provided technical guidance and inspiration for the research reported in this thesis.

The author is grateful to Professor Katsumi Nitta of Tokyo Institute of Technology for his helpful suggestions and discussions. The author is grateful to all who have affected or suggested his areas of research. The author devotes his sincere thanks and appreciation to all of them, and his colleagues.

Contents

Abstract i

Acknowledgments ii

1 Introduction 1
- 1.1 Sort and predicate hierarchies . . . 1
- 1.2 Implicit negations . . . 3
- 1.3 Organization of thesis . . . 3

2 Motivations and background 5
- 2.1 Predicate-hierarchy and eventuality . . . 5
  - 2.1.1 Logic with sort-hierarchy . . . 5
  - 2.1.2 Property inheritance in a sort-hierarchy . . . 7
  - 2.1.3 Motivations . . . 8
- 2.2 Implicit negations . . . 11
  - 2.2.1 Knowledge base with sort-hierarchy . . . 11
  - 2.2.2 Lexical negations in natural language . . . 12
  - 2.2.3 Motivations . . . 12
- 2.3 Related work . . . 14
  - 2.3.1 LOGIN . . . 14
  - 2.3.2 New HELIC-II . . . 15
  - 2.3.3 F-logic . . . 16
  - 2.3.4 Quixote . . . 17
  - 2.3.5 Description logics . . . 18

3 Predicate-hierarchy and eventuality in order-sorted logic 21
- 3.1 Event, Property, and Hierarchy . . . 21
  - 3.1.1 A hierarchy of predicates . . . 21
  - 3.1.2 Predicates as Event and Property . . . 23
- 3.2 An order-sorted logic with hierarchies and eventuality . . . 25
  - 3.2.1 Language and signature . . . 26
  - 3.2.2 Order-sorted terms and formulas . . . 27
  - 3.2.3 Σ-structure . . . 30
  - 3.2.4 Restricted structure for hierarchical predicates . . . 31
  - 3.2.5 Interpretation and satisfiability . . . 34
- 3.3 Deduction system . . . 35
  - 3.3.1 Horn clauses . . . 36
  - 3.3.2 Sorted substitution . . . 36

3.3.3 Argument supplementation . . . 38
3.3.4 Horn clause calculus . . . 42
3.3.5 Sorted unification . . . 48
3.3.6 Resolution with predicate-hierarchy . . . 54
3.4 Evaluation . . . 66
3.4.1 Query system . . . 66
3.4.2 The application of the query system to a legal reasoning . . . 70

4 Implicit negations in a sort-hierarchy 75
4.1 Implicitly negative sorts . . . 75
4.1.1 Structured sorts . . . 75
4.1.2 Negations by sort relations . . . 76
4.1.3 A contradiction in a sort-hierarchy . . . 78
4.2 An order-sorted logic with structured sorts . . . 80
4.2.1 Structured sort signature . . . 81
4.2.2 Structured sort terms and formulas . . . 83
4.2.3 Σ^+-structure . . . 85
4.2.4 Sort-hierarchy declaration . . . 87
4.3 Deduction and resolution with structured sorts . . . 88
4.3.1 Structured sort constraint system . . . 89
4.3.2 Clausal inference system with sort predicates . . . 92
4.3.3 Hybrid inference system with clauses and structured sort constraints 95
4.4 Evaluation . . . 98
4.4.1 Examples of refutations . . . 99

5 Conclusions and future work 107
5.1 Conclusions . . . 107
5.2 Future work . . . 108

A Syntax 111
A.1 Order-sorted terms . . . 111
A.2 Order-sorted formulas . . . 111
A.3 Structured sorts . . . 112
A.4 Structured sort terms . . . 112
A.5 Structured sort formulas . . . 112

B Examples 113
B.1 Predicate hierarchy 1 . . . 113
B.2 Predicate hierarchy 2 . . . 113
B.3 Event and property interpretations . . . 114
B.4 A criminal case . . . 114
B.5 Underage drinking . . . 115
B.6 Negative affix: unhappy . . . 116
B.7 Lexicon with negative meaning: loser . . . 116

Publications 121

Chapter 1

Introduction

The goal of knowledge representation is to describe declarative knowledge in the real world, not to put statements to be executed in order into a computer system. In the field of artificial intelligence, many logical deduction systems with rigorous theoretic foundations (syntax and model-theoretic semantics) have been used as knowledge representation languages. In particular, order-sorted logics (or many-sorted logics) do not only have rigorous foundations but also can represent structured information (i.e. the classification of objects [15]) by the sort symbols in a hierarchy, called a sort-hierarchy, in which each sort indexes a subset of the universe (i.e. the set of individuals).

The purpose of this thesis is:

(1) to propose an extended order-sorted logic as a knowledge representation language that possesses both hierarchies of sorts and predicates;

(2) to present a hybrid inference system which can deal with the properties of implicitly negative sorts in an assertional knowledge base, separated from the sort-hierarchy.

1.1 Sort and predicate hierarchies

We are concerned with describing structured information, not only as the classification of objects but also as the classification of predicates, in various kinds of knowledge representation and with richer logical reasoning about the structured information than existent knowledge reasoning systems (or deduction systems). However, knowledge representation in standard logics, even in order-sorted logics with a sort-hierarchy, tends to lose the structure information (e.g. the relationship between predicates fly and $move$) of predicates describing assertions and their reasoning. To represent the relationship between predicates within the framework of logic programming, programmers can build a pseudo-predicate-hierarchy (by a set of formulas of the logical implication form "$p(x) \rightarrow q(x)$" in which a superordinate predicate q has more abstract meaning than the subordinate predicate p); nevertheless, this does not allow the flexible reasoning that can be achieved using a predicate-hierarchy in which each predicate has its own fixed argument structure. Thus, the representation and inference machinery peculiar to predicate-hierarchies (as well as sort-hierarchies) is essential for predicate-hierarchy reasoning that is independent of argument structures.

Logical deduction systems with sorted (or typed) expressions have been investigated from the viewpoints of knowledge representation and rational reasoning. Amongst these systems, there seem to be two approaches: the formalization of a logic language and the implementation of a reasoning system (or logic programming language [6, 27]).

Following the many-sorted logic in Herbrand's thesis [22], several many-sorted logics [53, 18, 29] with different sorts as classes (e.g. points, lines, and planes in geometry) of individuals (but without subsorts, namely all sorts are disjoint) have been formalized as a generalized first-order logic with the same properties as first-order logic. A many-sorted logic with a sort-hierarchy or subsorts is called an order-sorted logic [36]. In the field of computer science, Walther and Cohn have separately developed an order-sorted calculus [50, 51, 14] based on a resolution by a sorted unification algorithm with sort-hierarchy. Since these papers, order-sorted logics have been extended to include more expressive and efficient methods of knowledge representation and automated deduction systems. In the logics [52, 10, 20, 41, 44] with a sort-hierarchy, the researchers are concerned with order-sorted unifications that solve the problem of finding the most general unifier of sorted terms depending on the structure (e.g. lattice) of the sort-hierarchy. In other work related to order-sorted logic, typed logic programming languages [23, 21] have been investigated as Horn clause logics with typed terms extended to include polymorphic types and SLD-resolution (Selection-rule driven Linear Resolution for Definite clauses).

With regard to work that actually implements a reasoning system, the logic programming languages LOGIN [1] and LIFE [2] have been proposed by Aït-Kaci in which ψ-terms together with feature structures [13] (which can describe complicated classes of objects) are introduced. Smolka has proposed Feature Logic [44] to generalize ψ-terms by adding negation and quantification. In New HELIC-II [35, 33] developed as a legal reasoning system, Nitta has proposed a typed term (called an H-term) with a verb-type or a noun-type by extending ψ-terms in order to represent legal knowledge. Alternatively, F-logic [26] and $\mathcal{QUIXOTE}$ [54, 55] have been developed as object-oriented deductive languages with the notions of objects, classes, subclasses, and property inheritance derived from the object-oriented programming paradigm.

We propose an extended order-sorted logic as a knowledge representation language that possesses both hierarchies of sorts and predicates. This logic must delete and supplement the arguments of each predicate in order to allow it to derive superordinate and subordinate predicates (each with different argument structures) from the other predicates in a predicate-hierarchy (as distinguished from a sort-hierarchy). For the manipulation of arguments, we suggest that a distinction between event and property in predicate interpretation leads to the machinery appropriate for predicate-hierarchy reasoning. The notion of event and property is based on work [5, 31, 43] dealing with temporal reasoning (i.e. taking account of various temporal aspects of propositions). In [5], Allen distinguished between event, property, and process in English sentences, and so did McDermott [31] between fact and event. In contrast to the work regarding temporal reasoning, we introduce the new and entirely original idea that event and property assertions respectively afford different quantification to all implicit objects in the real world, not only to spatio-temporal objects. On the basis of this idea, we define an inference mechanism for the predicate-hierarchy. This includes the manipulation of predicate arguments in which the supplementary arguments are quantified differently, depending on whether a predicate is interpreted as an occurrence of an event or a universal property. We develop a Horn clause resolution extended to include inference rules (generalization and specialization rules) of

the predicate-hierarchy that include the supplementation of arguments in addition to sorted substitutions (to enable sort-hierarchy reasoning).

1.2 Implicit negations

When one attempts to denote a sort name by the vocabulary in a natural language (e.g. the sorts *loser* and *unhappy_person*), a sort-hierarchy may implicitly contain sorts with negative meaning (which differs from the classical negation). These implicitly negative sorts (called lexical negations in the literature [38] of linguistics) are classified as (i) negative affix or (ii) lexicon with negative meaning. In addition to a subsort relation corresponding to the classification of objects, knowledge representation using a sort-hierarchy requires describing the exclusivity and the totality of these negative sorts using complex sorts (composed of the operators: conjunction, disjunction, and negation). In fact, order-sorted logics cannot describe and emphasize the negative information in sorts in the hierarchy, and also order-sorted substitutions (or unifications) cannot provide the reasoning mechanism required for the properties of implicitly negative sorts. Thus, in these logics, the negative sorts would give rise to a semantic conflict between sort symbols in the hierarchy.

We present a hybrid inference system which can deal with the properties of implicitly negative sorts in an assertional knowledge base, separated from the sort-hierarchy. Our approach is based on the order-sorted logic Beierle has proposed in [10]. In his logic, a unary predicate (called a sort predicate) expressed by a sort can be expressible in clausal forms whereby the resolution for clauses with sort predicates leads to a close coupling between a sort-hierarchy and an assertional knowledge base. In this thesis, we propose the notions of structured sorts, sort relations, and contradiction in a sort-hierarchy. These can define the relationship between the implicit negations and the classical negation, and can declare the exclusivity and the totality of each negation and its affirmation. In order to infer assertional conclusions from these negations, we define deduction and resolution systems obtained by a combination of a clausal inference system with sort predicates and a structured sort constraint system. The negative information in sorts can be derived from the hierarchy of structured sorts by regarding the negative affix as an operator based on strong negation and the lexicon with negative meaning as an exclusive sort of the antonym.[1] Furthermore, we define a contradiction in a sort-hierarchy that is decided by the exclusivity and the totality (or partiality) of affirmative and negative sorts. Finally, we give the syntax and the semantics of an order-sorted logic with structured sorts, and the consistency of a hybrid inference system from clausal forms and sort constrains is then proved.

1.3 Organization of thesis

We start in Chapter 2 with an introduction to the basic notions of sort-hierarchy (logic with sort-hierarchy, property inheritance in a sort-hierarchy, and knowledge base with sort-hierarchy) and lexical negations in natural language, and then discuss our motivations for extending order-sorted logics (or typed logic programming languages) for knowledge reasoning. Section 2.1 demonstrates the expected reasoning mechanism for both sort and

[1] A word that is opposite in meaning to anther word.

predicate hierarchies using two examples given by a typed logic programming language with a query system. In Section 2.2, we illustrate the conclusions deduced from a sort-hierarchy in a case where the hierarchy includes sort names that are lexical negations. At the end of Chapter 2, we show related approaches to knowledge representation systems.

Chapter 3 sets the focus of the research. It develops an extended order-sorted logic with predicate-hierarchy and eventuality. Section 3.1 considers how we introduce the notions of predicate-hierarchy and eventuality and their corresponding reasoning mechanism into an order-sorted logic. Here we discuss the problems of the framework of former order-sorted logics (or reasoning systems with type expressions) that arise from dealing with a predicate-hierarchy. In Section 3.2, we define the syntax and the semantics of the proposed logic. In Section 3.3, we formalize a Horn clause calculus with inference rules of predicate-hierarchy that include argument supplementation, and then develop a resolution based on this calculus and an order-sorted unification algorithm. In Section 3.4, we evaluate the capability of our logic as a knowledge representation system. Hence, we show that a query system defined by the resolution we develop provides the reasoning mechanism required in our motivational examples given in Chapter 2, and give an example of its application to a legal reasoning.

Chapter 4 presents an order-sorted logic that includes the complex sort expressions of implicit negations. In Section 4.1, we give an account of structured sorts, sort relations, and contradiction in a sort-hierarchy. These notions can be used to declare the properties of implicitly negative sorts in a sort-hierarchy. Section 4.2 and Section 4.3 present the formalization of order-sorted logic with structured sorts, and systems of clausal deduction, clausal resolution, and structured sort constraints. Section 4.4 evaluates the usefulness of the knowledge representation system to deal with implicit negations. This will be shown by derivations (using a hybrid inference system obtained by combining the systems we propose) for the examples in Chapter 2.

In Chapter 5, we give our conclusions and discuss future work.

Chapter 2

Motivations and background

We start in Chapter 2 with an introduction to the basic notions of sort-hierarchy (logic with sort-hierarchy, property inheritance in a sort-hierarchy, and knowledge base with sort-hierarchy) and lexical negations in natural language, and then discuss our motivations for extending order-sorted logics (or typed logic programming languages) for knowledge reasoning. Section 2.1 demonstrates the expected reasoning mechanism for both sort and predicate hierarchies using two examples given by a typed logic programming language with a query system. In Section 2.2, we illustrate the conclusions deduced from a sort-hierarchy in a case where the hierarchy includes sort names that are lexical negations. At the end of Chapter 2, we show related approaches to knowledge representation systems.

2.1 Predicate-hierarchy and eventuality

2.1.1 Logic with sort-hierarchy

We will explain the notation (based on [45]) of order-sorted logic: sort-hierarchy, sorted variables, sorted terms, and sorted substitutions. $\mathcal{S}$ denotes a set $\{s_1, \ldots, s_n\}$ 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 $s_i \sqsubset_S s_j$. Let $\mathcal{D}_\mathcal{S}$ be a set of sort declarations. The subsort relation by $\mathcal{D}_\mathcal{S}$ can be obtained as follows

$$\sqsubset_S = \{(s, s') \in \mathcal{S} \times \mathcal{S} \mid s \sqsubset_S s' \in \mathcal{D}_\mathcal{S}\}.$$

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,

$$s \leq_S s' \ \Leftrightarrow \ s = s', \textit{ or}$$
$$\textit{there exists a } \sqsubset_S \textit{ path from } s \textit{ to } s'.$$

A hierarchy of sorts (or 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 $\bot$) 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 and $s' \leq_S s$ 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 $s_1, s_2 \in \mathcal{S}$.

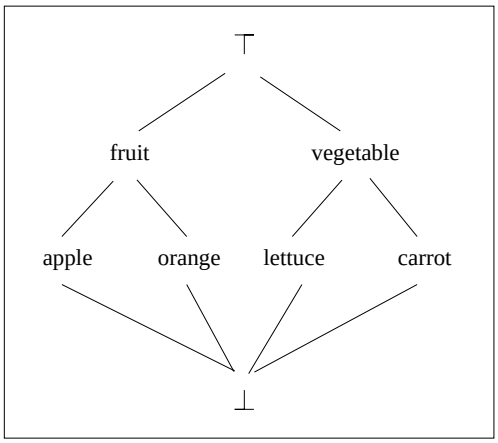

Figure 2.1 A sort-hierarchy

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

$$\begin{aligned} apple &\sqsubset_S fruit, \\ orange &\sqsubset_S fruit, \\ carrot &\sqsubset_S vegetable, \\ lettuce &\sqsubset_S vegetable. \end{aligned}$$

Moreover, by adding the subsort declarations

$$\begin{aligned} fruit &\sqsubset_S \top, \\ vegetable &\sqsubset_S \top, \\ \bot &\sqsubset_S apple, \\ \bot &\sqsubset_S orange, \\ \bot &\sqsubset_S carrot, \\ \bot &\sqsubset_S lettuce. \end{aligned}$$

to the above declarations, the sort-hierarchy in Figure 2.1 can be built.

A variable x of sort s whose domain is restricted (i.e. a subset of the universe) is written as

$$x{:}\, s$$

which is called a *sorted variable*. A sort declaration (called a function declaration) of n-ary function f is denoted by

$$f{:}\, s_1 \times \ldots \times s_n \to s$$

with $s_1, \ldots, s_n, s \in \mathcal{S}$. In particular, a sort declaration of constant c (i.e. 0-ary function) is denoted by $c{:} \to s$. A sort declaration (called a predicate declaration) of n-ary predicate f is denoted by

$$p{:}\, s_1 \times \ldots \times s_n$$

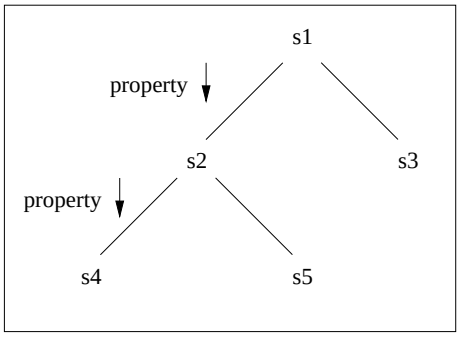

Figure 2.2 Property inheritance

with $s_1, \ldots, s_n \in \mathcal{S}$.

The term $f(t_1, \ldots, t_n)$ is a term of sort s, if the function declaration is $f\colon s_1 \times \ldots \times s_n \rightarrow s$. 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 a n-ary function with the function declaration $f\colon s_1 \times \ldots \times s_n \rightarrow s$. In particular, we write for the term c of sort s

$$c\colon s$$

where c is a constant with the function declaration $c\colon \rightarrow s$. A term restricted by a sort s is called *a sorted term*. For any sorted term t, the function $Sort(t)$ assigns to the sort of term t.

For example, we have

$$\begin{aligned} Sort(x\colon s_1) &= s_1, \\ Sort(f(c\colon s_1, y\colon s_3)\colon s_2) &= s_2. \end{aligned}$$

Let L_i be a literal where $L_i(t)$ denotes that the term t occurs in the literal L_i. A sorted substitution with a subsort relation $\leq_S$ (or a sort-hierarchy) is defined by the following rule [37]:

$$\frac{L_1(x\colon s) \vee \ldots \vee L_n(x\colon s)}{L_1(t) \vee \ldots \vee L_n(t)}\ (\textit{sorted substitution}),$$

where $Sort(t) \leq_S s$.

2.1.2 Property inheritance in a sort-hierarchy

The notion of inheritance [4] in a hierarchy is well-known in artificial intelligence and the object-oriented paradigm in work that is related to nonmonotonic reasoning. Inheritance is a mechanism in which subclasses (or specific classes) inherit the properties of their superclasses (or general classes) in a hierarchy expressed as a semantic network [39]. We now outline an inheritance mechanism in a sort-hierarchy that is the same as that adopted in the logic programming languages LOGIN [1] and LIFE [2], and in the object-oriented deductive languages F-logic [26] and $\mathcal{Quixote}$ [54, 55].

Given the sort-hierarchy shown in Figure 2.2, we assume an assertion

$prop(x{:}\, s_1)$

(described in order-sorted predicate logic) means that s_1 has the property *prop*. In the sort-hierarchy, the property inheritance is a downward reasoning mechanism (shown in Figure 2.2) from a supersort to the subsorts along the subsort relation. That is, if a supersort has a property, then the subsorts also have the same property.

By the subsort declaration $s_2 \sqsubset_S s_1$, the sort s_2 inherits the property *prop* of sort s_1. Hence, we can obtain the following conclusion

$prop(x{:}\, s_2)$

All the other subsorts s_3, s_4, s_5 of s_1 also inherit the property *prop*. We can say that this conclusion is consistent with the classification of objects, since an object in subsort s must an element of supersort s' (with $s \sqsubset_S s'$) and then have the properties of s'.

2.1.3 Motivations

We shall discuss the expected reasoning mechanism for both sort and predicate hierarchies using two examples given by a typed logic programming language with a query system.

Example 1. A hierarchy of predicates

We consider deriving superordinate predicates from subordinate predicates in the hierarchy of predicates.[1] That is, general expressions can be inferred from specific expressions in the predicate-hierarchy.

Given the hierarchy of predicates in Figure 2.3, we expect the following results from a query system in which the answer `yes` or `no` must be returned. First, we assume the argument numbers of the predicates `hit` and `illegal_act` as follow.

```
hit/1
illegal_act/2
```

If a fact `hit(john:man)` holds, then the superordinate predicate `illegal_act` can be derived from the predicate `hit` in direction (1) shown in the hierarchy of Figure 2.3. However, the first query "Did John commit an illegal act against Mary?" expressed by `?-illegal_act(john:man, mary:woman)` will give the answer `no`. It is certain that John hit somebody but not that John hit Mary. Thus, the second query "Did John commit an illegal act against somebody?" expressed by `?-illegal_act(john:man, Y:person)` will yield `yes`.

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

[1]The derivation from a subordinate predicate p_1 to the superordinate predicate p_2 is different from the property inheritance we have explained in the previous section, and it is usually deduced from the rule $p_1(x) \rightarrow p_2(x)$ and logical inference rules without any machinery peculiar to predicate-hierarchies.

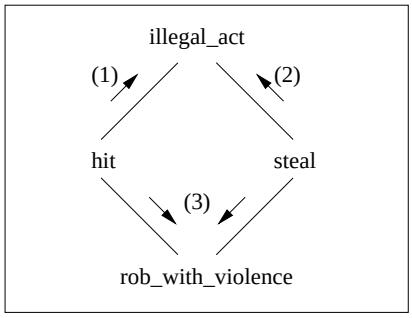

Figure 2.3 A hierarchy of predicates

This exemplifies the case of a derivation of predicate `illegal_act` (higher in the hierarchy of predicates) with more arguments than the predicate `hit` representing the fact `hit(john:man)`. To make the inference above, we have to supplement the second argument (whose role may be *person* and which exists in `illegal_act` but not in `hit`) to the fact `hit(john:man)`. In addition, we have to take account of the quantifications of the supplemented arguments. When the supplemented argument `Y:person` is interpreted as all persons, the answer to the second query may be `no`. Therefore, because this interpretation does not fit with what we expect (i.e. human reasoning), the argument `Y:person` should be interpreted as a person (somebody).

Next we assume the argument numbers of the predicates `illegal_act`, `hit`, `steal` and `rob_with_violence` as follow.

```
illegal_act/1
hit/2
steal/2
rob_with_violence/3
```

Since the predicate `illegal_act` has fewer arguments than the predicate `steal` in the fact `steal(john:man, mary:woman)`, the derivation of the following query results in `yes` from direction (2) in Figure 2.3.

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

This answer is plausible enough, because the fact `steal(john:man, mary:woman)` implies the fact `illegal_act(john:man)` in a broad sense. Namely, the predicate `illegal_act` is more general than the predicate `steal`, and the single argumant `john:man` is more general than the two arguments `john:man` and `mary:woman`.

As for a more complicated inference, the predicate `rob_with_violence` as the conjunction of `hit` and `steal` will be derived from the two predicates (on (3) in Figure 2.3). Since the facts `hit(john:man, mary:woman)` and `steal(john:man, c:wallet)` in an incident imply John's robbing with violence, the query "Did John steal Mary's wallet using robbery with violence?" will yield `yes`.

```
hit(john:man, mary:woman).
```

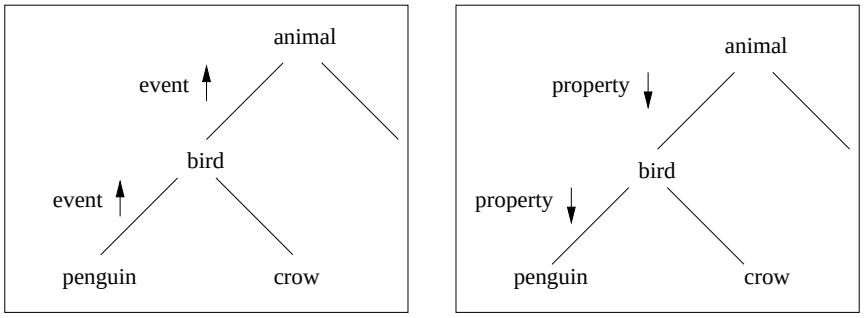

Figure 2.4 The inference by event and property

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

To obtain this answer, we require inference machinery for the predicate-hierarchy whereby the arguments of two predicates `hit` and `steal` are mixed to produce the arguments of predicate `rob_with_violence`.

Example 2. Event and property interpretations

The interpretation of natural language sentences can vary the meaning of a predicate. In knowledge representation using predicates, a predicate has the two roles as *event* and as *property* (discussed in the research [5, 31] into temporal reasoning). For example, we assume that the assertion `fly(X:bird)` holds with the sort-hierarchy in Figure 2.4. This assertion would be ambiguous, because of the two ways of interpreting it (as event and as property) illustrated as follows.

The fact `fly(X:bird)`, when interpreted as an event, states that a bird is flying. Thus, the fact entails that an animal is flying, so that the query `?-fly(X:animal)` results in `yes`. However, the fact does not state that a penguin is flying, and so the answer to query `?-fly(X:penguin)` is `no` as follows.

Interpretation 1: A bird is flying.

```
fly(X:bird).
?-fly(X:animal).
yes.
?-fly(X:penguin).
no.
?-move(X:animal).
yes.
```

Briefly, the assertion as an event can be used to deduce the event assertion of a supersort. As in the left hand side of Figure 2.4, we can deduce the event `fly(X:animal)` from the event `fly(X:bird)`, since the sort `animal` is one of the supersorts of `bird`.

In contrast, the fact `fly(X:bird)` interpreted as a property states that birds have the property of flight. In this case, we can use the mechanism of property inheritance in the

sort-hierarchy as we have mentioned in the previous section. If birds have the property `fly` and `penguin` is a subsort of `bird`, then penguins should have the same property `fly`. Thus, the query `?-fly(X:penguin)` will have the answer `yes` as follows.

Interpretation 2: Birds have the property of flight.

```
fly(X:bird).
?-fly(X:animal).
no.
?-fly(X:penguin).
yes.
?-move(X:animal).
no.
```

However, the assertion `fly(X:bird)` does not imply that all animals have the property of flight, since `animal` is not a subsort of `bird`. As a result, the answer for query `?-fly(X:animal)` will yield `no`. The right hand side of Figure 2.4 demonstrates the mechanism of property inheritance in which subsorts inherit the properties from the supersorts. Using the inheritance mechanism, the conclusion `fly(X:penguin)` can be inferred from the fact `fly(X:bird)` in the sort-hierarchy.

In their interpretation as an event or as a property, these inferences in the two directions shown in the hierarchy in Figure 2.4 are consistent with natural human reasoning from the eventuality.[2]

The queries and the answers we have seen suggest the necessity for order-sorted logics to include an inference machinery that deals with both sort and predicate hierarchies and which can distinguish between event and property in predicate interpretation.

2.2 Implicit negations

2.2.1 Knowledge base with sort-hierarchy

Beierle [10] has proposed a hybrid knowledge representation system that distinguishes between taxonomical information (in the sort-hierarchy) and assertional information (in the assertional knowledge base), by extending an order-sorted logic. This system can deal with the taxonomical information in an assertional knowledge base in which a sort symbol can be expressed as a unary predicate (called a sort predicate) in clausal forms. Since a sort and a unary predicate have the same expressive power, we can regard a subsort declaration $s_1 \sqsubseteq s_2$ as the following logical implication form:

$$s_1(x) \rightarrow s_2(x)$$

where the unary predicates $s_1(x), s_2(x)$ corresponding to the sorts s_1, s_2 are sort predicates.

Let C, C_1, C_2 be clauses, s, s_1, s_2 sorts (or sort predicates), θ a sorted substitution, and t, t_1, t_2 sorted terms. In order to use the information in a sort-hierarchy in a clausal knowledge base (or an assertional knowledge base), the following inference rules:

[2]In this paper, we use *eventuality* as a technical term to indicate the concept that propositions are classified as various temporal aspects (e.g. event or property).

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

where $s_2 \sqsubseteq_S s_1$ and $\theta(t_1) = \theta(t_2)$, and

$$\frac{\neg s(t) \vee C}{C} \ (elimination)$$

where $Sort(t) \sqsubseteq_S s$, are added to his resolution system.

This hybrid knowledge representation system provides a useful way to deal with a sort-hierarchy in a clausal knowledge base.

2.2.2 Lexical negations in natural language

In knowledge representation using a logic with sort-hierarchy, we can describe the classification of objects using a subsort relation. However, a sort-hierarchy may contain sorts which we recognize as implicitly negative from their natural language meaning. These negations are called *lexical negations* in linguistics and are distinct from the negative particle *not*. Order-sorted logics would regard an implicitly negative sort as a positive expression but not as an expression which is opposite to its antonym, since every sort name is a mere string or a symbol. Nevertheless, knowledge representation systems should take account of the fact that the lexical negation *unhappy* (or *loser*) is opposite in meaning to the positive expression *happy* (or *winner*), when a sort-hierarchy has these expressions.

To do this, we have to observe the properties of lexical negations in natural language and then consider dealing with these negations in a sort-hierarchy. In [38], lexical negations (words that implicitly have negative meaning) are classified as follows:

(i) Negative affix (in-,un-,non-):
incoherent, inactive, unfix, nonselfish, illogical, impolite, etc.

(ii) Lexicon with negative meaning:
doubt(believe not), deny(approve not), prohibit(permit not), forget(remember not), etc.

2.2.3 Motivations

In the following, we will illustrate the conclusions deduced from a sort-hierarchy in a case where the hierarchy includes sort names that are lexical negations.

Example 3. Negative affix: *unhappy*

A sort-hierarchy h_1 containing the sort *unhappy* with negative affix *un* is shown in Figure 2.5. The sort *unhappy* is not only a negative expression (unlike the positive expression *happy*) but also a subexpression of $feeling$ (like *happy*). Hence, the sort $feeling$ can be derived from *unhappy* (like *happy*) upward in the sort-hierarchy, whereas it cannot be derived from the classical negation $\neg happy$ of *happy* as follows:

$$\begin{array}{rcl} unhappy(bob) & \vdash_{h_1} & feeling(bob) \\ \neg happy(bob) & \nvdash_{h_1} & feeling(bob) \end{array}$$

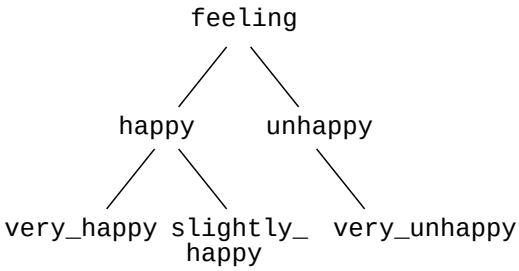

Figure 2.5 A sort-hierarchy including `unhappy`

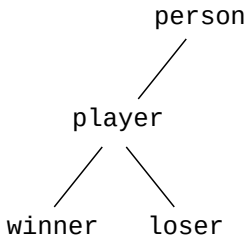

Figure 2.6 A sort-hierarchy including `loser`

where $A \vdash_h B$ indicates that B is derivable from A in a sort-hierarchy h.

The sort *unhappy* is a stronger negative statement than the classical negation $\neg happy$, so that $\neg happy$ can be derived from *unhappy*, but *unhappy* cannot be derived from $\neg happy$ in the following.

$$\begin{array}{rcl} unhappy(bob) & \vdash_{h_1} & \neg happy(bob) \\ \neg happy(bob) & \nvdash_{h_1} & unhappy(bob) \end{array}$$

The fact $\neg feeling(bob)$ that the person *bob* is not feeling yields the following result.

$$\neg feeling(bob) \quad \vdash_{h_1} \quad \neg happy(bob) \wedge \neg unhappy(bob).$$

In contrast, no premise can derive

$$\neg happy(bob) \wedge \underline{\neg\neg happy}(bob).$$

This shows the sort *unhappy* has the meaning of $feeling$, but the classical negation $\neg happy$ does not have the meaning of $feeling$.

Example 4. Lexicon with negative meaning: *loser*

Let a sort-hierarchy h_2 including *loser* be given in Figure 2.6. In this sort-hierarchy, both antonyms *winner* and *loser* (i.e. *winner* is opposite in meaning to *loser*) hold the meaning of objects playing a game (i.e. *player*). Moreover, *loser* is different from the classical negation $\neg winner$ of *winner*, because *loser* means the negative event opposite to an event denoted by *win* but the classical negation $\neg winner$ denys the event denoted

by *win*. Therefore, the supersort *player* can be derived from *loser* (or *winner*), but not from $\neg winner$ as follows.

$$\begin{aligned} loser(tom) &\vdash_{h_2} player(tom) \\ \neg winner(tom) &\nvdash_{h_2} player(tom) \end{aligned}$$

This is related to the difference (discussed in [19, 43]) between the negations of an event and a property.

Furthermore, if the person *tom* is not a player, then the negations $\neg winner$ and $\neg loser$ can be derived. In contrary, if the person *tom* is a player, then *winner* or *loser* holds in the totality (i.e. *tom* must be a winner or a loser) of *winner* and *loser* as follows.

$$\begin{aligned} \neg player(tom) &\vdash_{h_2} \neg winner(tom) \wedge \neg loser(tom) \\ player(tom) &\vdash_{h_2} winner(tom) \vee loser(tom) \end{aligned}$$

By the totality, if the person *tom* is a player but not a loser ($\neg loser$), then *tom* is a winner. If *tom* is neither a winner nor a loser ($\neg winner \wedge \neg loser$), then *tom* is not a player ($\neg player$) as follows.

$$\begin{aligned} player(tom) \wedge \neg loser(tom) &\vdash_{h_2} winner(tom) \\ \neg winner(tom) \wedge \neg loser(tom) &\vdash_{h_2} \neg player(tom) \end{aligned}$$

We require these derivations from implicitly negative sorts. However, it is hard to distinctively describe implicitly negative sorts in the sort-hierarchy, so that many knowledge bases would lose the property that implicit negations are exclusive to their antonyms and partial to their classical negation. Therefore, standard inference systems of order-sorted logics cannot immediately derive the above results from subsorts, supersorts, and classical negation. Thus, relating these negative sorts to their antonyms and classical negation should be considered in an order-sorted logic. In Chapter 4, we will propose a method to describe the properties of lexical negations implicitly included in a sort-hierarchy and develop an inference machinery for outputting the above results..

2.3 Related work

In this section, we show related approaches to knowledge representation systems including complex type expressions and their inference rules.

2.3.1 LOGIN

LOGIN has been developed as a logic programming language by replacing predicate arguments in PROLOG with extended terms, called ψ-terms. This language includes an inheritance mechanism (in an ISA-hierarchy) based on a ψ-term unification algorithm, instead of a resolution principle for clausal forms. A ψ-term accompanies a sort (or a type) symbol s with a finite sequence (called *a feature structure*) of ordered pairs of the form $l \Rightarrow t$ where l is an attribute label and t is its attribute value. As a result, it contributes a more expressive and efficient method of knowledge representation.

For example, the following ψ-term represents a concept "person" by the sort *person* with feature structure $id \Rightarrow name$, $born \Rightarrow date(\ldots)$, $father \Rightarrow person$.

person(id ⇒ name;
born ⇒ date(day ⇒ integer;
month ⇒ monthname;
year ⇒ integer);
father ⇒ person).

ψ-terms allow us to represent more detailed information by feature structures than simple sorts. The following ψ-term

apple(taste ⇒ sour; color ⇒ red)

represents "sour red apples" where *taste* and *color* are attribute labels and *sour* and *red* are their attribute values respectively. The meaning of a ψ-term is narrowed by the succeeding feature structure, e.g., the simple sort *apple* is a more abstract expression than the ψ-term $apple(taste \Rightarrow sour;\ color \Rightarrow red)$. Note that the simple sort *apple* shows "apples" and the ψ-term $apple(taste \Rightarrow sour;\ color \Rightarrow red))$ shows "sour red apples."

In particular, ψ-terms enhance the expressivity of the terminology used to represent complex concepts (both concrete and general concepts with attributes). However, ψ-terms cannot express the classification of verb-expressions (or predicates), because a ψ-term cannot express a relation between objects, it can only express a class of objects, i.e. a complex sort (or type). To deal with the classification of verb-expressions, the knowledge representation language in New HELIC-II (which we will explain in the next section) introduces an extended terminology (called H-terms that are based on ψ-terms) with a type-hierarchy that distinguishes between verb-types and noun-types.

2.3.2 New HELIC-II

In [35], Nitta has presented the legal reasoning system New HELIC-II which is aimed at the modeling of general legal reasoning and is based on his experience with HELIC-II [34]. This system, composed of two inference engines (rule-based reasoning and case based reasoning) provides two important functions in legal reasoning. The first is an argumentation function realized by a typed logic language. The second is a debating function realized by controlling the argumentation. The knowledge representation language in New HELIC-II can describe legal rules, legal cases, and meta-rules by the typed terms (called H-terms) extended to ψ-terms in LOGIN. A type expression in an H-term is classified as a verb-type or a noun-type in the type-hierarchy[3] (as shown in Figure 2.7), where the hierarchy includes the greatest type $\top$ (as the set of all objects) and the least type $\bot$ (as a null set).

We here introduce a typed term to explain the notation of H-terms. Similarly to a ψ-term, a typed term

$$E(l_1 = A_1, \ldots, l_n = A_n)$$

consists of a type expression E and a list $l_1 = A_1, \ldots, l_n = A_n$ of attributes where E is a verb-type or a noun-type, $l_1, \ldots, l_n$ attribute labels, and $A_1, \ldots, A_n$ typed terms. Therefore, a H-term is a typed term such that the root symbol E is a verb-type and the attribute values $A_1, \ldots, A_m$ are typed terms of which each root symbol is a verb-type or a noun-type. We give an example of a legal rule written by H-terms as follows:

[3]In [33], a verb-type is called an event-type or a property-type, and a noun-type is called an object-type.

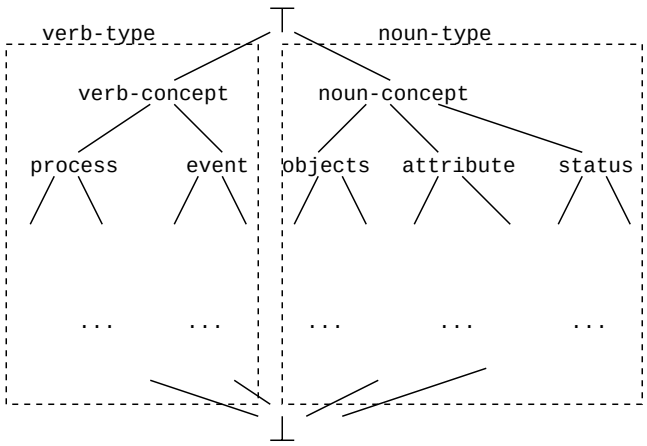

Figure 2.7 Verb type and noun type

$$crime\_of\_violence(a\_object = hit(agent = X/person,\ object = Y/person,\ place = W/\top,\ time = Z/\top)) \leftarrow hit(agent = X/person,\ object = Y/person,\ place = W/\top,\ time = Z/\top).$$

The expressions $crime\_of\_violence(\ldots)$ and $hit(\ldots)$ are H-terms where the root symbols $crime\_of\_violence$ and hit are verb-types and $person$ is a noun-type.

Since H-terms allow us to use either a verb-type or a noun-type in their arguments, it is useful for representing legal knowledge more expressively than ψ-terms. However, its complicated syntax gives rise to the difficulties in defining the semantic models and proving the completeness of the inference system.

2.3.3 F-logic

Frame-logic (called F-logic [26]) is a deductive object-oriented language that is as an extension to O-logic [28]. It includes object identity, complex objects, inheritance, polymorphic types, query method, encapsulation, and other features. In general, object-oriented approaches are insufficient for the formalization of model-theoretic semantics. Alternatively, formal deductive approaches only represent flat data and so do not support notions of the classification of objects. In contrast to these two separate approaches, F-logic as an object-oriented language with logical deduction solves both problems.

The formulas (called F-formulas) in the language for F-logic are classified as is-a assertions, data expressions, or signature expressions. An example of F-formulas is shown as below:

ISA-relations:

$student : person$

$john : student$

Database facts:

$bob[name \rightarrow \text{“}Bob''; age \rightarrow 40;$
$works \rightarrow cs_1[dname \rightarrow \text{“}CS''; mngr \rightarrow bob;$
$assistants \twoheadrightarrow \{john, sally\}]]$

General Class Information:

$$faculty[boss \Rightarrow \{faculty, hi\_paid\_empl\}; \\ age \Rightarrow midaged; degrees \Rightarrow\!\!\!\!\Rightarrow degrees; \\ degrees \rightarrow\!\!\!\!\rightarrow \{phd\}]$$

Deductive rule:

$$E[boss \rightarrow M] \leftarrow E : empl[works \rightarrow D : dept[mngr \rightarrow M : empl]].$$

An is-a assertion denotes an ISA-relation, e.g. $student : person$ declares the class *student* is a subclass of *person*. As can be seen from the above F-formulas, the double-headed arrows $\rightarrow\!\!\!\!\rightarrow$ and $\Rightarrow\!\!\!\!\Rightarrow$ are used as set-valued attributes, and the single-headed arrows $\rightarrow$ and $\Rightarrow$ are used as scalar attributes. Data expressions take two forms:

Scalar data expressions: $O[Q \rightarrow T_1; \ldots]$,
Set-valued data expressions: $O[R \rightarrow\!\!\!\!\rightarrow \{S_1, \ldots, S_k\}; \ldots]$,

which denote database facts. In the above example, the database fact indicates that the object *bob* has the attributes: *name* "Bob", *age* 40, and *works* cs_1. Signature expressions also take two forms:

Scalar signature expressions: $O[V \Rightarrow (A_1, \ldots, A_r); \ldots]$,
Set-valued signature expressions: $O[W \Rightarrow\!\!\!\!\Rightarrow (B_1, \ldots, S_t); \ldots]$,

which denote general class information. In the above example, the general class information shows the concept of "faculty" with attributes *boss*, *age*, and *degrees*.

Moreover, F-logic provides a complete inference system for F-formulas (expressed as ISA-relations, database facts, general class information, and deductive rules) that consists of clausal inference rules, ISA inference rules, and type inference rules.

2.3.4 Quixote

$\mathcal{Q}\mathit{UIXOTE}$ [47, 54, 55] has been developed as a object-oriented deductive language in which the basic notions of knowledge representation are objects and their properties. An object term o denotes an object, and a subsumption constraint $o_1 \sqsubseteq o_2$ over the set of object terms denotes a property of objects. An ordered pair $(o, \{cs_1, \ldots, cs_m\})$ of an object term o and a set $\{cs_1, \ldots, cs_n\}$ of subsumption constraints (such that cs_i is of the form $o.l \sqsubseteq v$, $o.l \sqsupseteq v$, or $o.l = v$) is called an attribute term that represents properties of object o. This ordered pair is written by

$$o/[l_1 op_1 v_1, \ldots, l_n op_2 v_n]$$

with $op_i \in \{=, \rightarrow, \leftarrow\}$ where the list $[l_1 op_1 v_1, \ldots, l_n op_n v_n]$ is called the extrinsic attribute of object term o. Here, if cs_i is of the form $o.l \sqsubseteq v$, then the list includes $l \rightarrow v$. If cs_i is of the form $v \sqsubseteq o.l$, then the list includes $l \leftarrow v$. We write $l = v$ for $l \rightarrow v$ and $l \leftarrow v$. For example, the following expression

$$john/[age = 20, sex = male].$$

is an extrinsic attribute term. A object inherits extrinsic attributes from the super objects. For example, given the following statements

$$swallow \sqsubseteq bird.$$
$$bird/[canfly \rightarrow yes].$$

The object *swallow* inherits the extrinsic attribute $[canfly \rightarrow yes]$ from the object *bird*. Moreover, $\mathcal{Quixote}$ has introduced another attribute (called an intrinsic attribute) representing a property that identifies the object. An intrinsic attribute is included in the object term but not in the attribution of an attribute term. The following example represents the concept "red apples" by a complex object term with an intrinsic attribute.

$$apple[color = red].$$

While the statement $apple[color = red]$ means "red apples," the statement $apple/[color = red]$ means that "apples are red." These extrinsic and intrinsic attributes correspond to a ψ-term $apple(color \Rightarrow red)$ (i.e. a typed term with feature structure) and an atomic formula $red(x{:}\, apple)$ in predicate logic respectively.

$\mathcal{Quixote}$ provides a programming language paradigm as well as a knowledge representation system. However, we believe that it is incapable of representing those quantifications of propositions that depend on eventuality in natural language.

2.3.5 Description logics

Description logics (as outlined in [40, 17]) have been proposed as a theoretical approach to terminological knowledge representation, related to semantic networks [39], frame systems [32], and KL-ONE knowledge representation systems [11]. These logics represent structured concepts formed by primitive concepts and attributes, whereas order-sorted logics express assertions concerning objects classified in a sort-hierarchy. The language of a description logic is a language (called a concept language) that consists of a set $\mathbf{C}$ of concept symbols and a set $\mathbf{R}$ of role symbols. In [42], the attribute concept description language $\mathcal{ALC}$ and its sublanguages $\mathcal{AL}, \mathcal{ALE}, \mathcal{ALU}$ have been presented. Given a set $\mathbf{C}$ of concepts $(C, D, ..)$ and a set $\mathbf{R}$ of roles $(R, S, ..)$, the syntax of language $\mathcal{ALC}$ is defined by the following.

$$Concept\ terms : C, D \longrightarrow C \mid \neg C \mid C \sqcap D \mid C \sqcup D \mid \forall R.C \mid \exists R.C.$$

For example, the following concept term

$$\exists Child.Male \sqcap Person$$

indicates "persons that have a male child." Furthermore, the sublanguage $\mathcal{AL}$ is obtained by deleting the disjunction form $C \sqcup D$, restricting $\exists R.C$ to the form $\exists R.\top$ (called unqualified existential quantification), and restricting $\neg C$ to the form $\neg A$ (called simple complement) where A is a concept symbol. By adding constructors (connectives, quantifiers, modal operators, etc.) to $\mathcal{AL}$, various superlanguages [7, 12, 8, 25] have been developed.

An interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a set $\Delta^{\mathcal{I}}$ (called the domain of $\mathcal{I}$) and a function $\cdot^{\mathcal{I}}$ that maps every concept to a subset of $\Delta^{\mathcal{I}}$ and every role to a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ such that

$$\begin{aligned}
&\top^{\mathcal{I}} = \Delta^{\mathcal{I}},\\
&(\forall R.C)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} \mid \forall (a,b) \in R^{\mathcal{I}}{:}\, b \in C^{\mathcal{I}}\},\\
&(\exists R.C)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} \mid \exists (a,b) \in R^{\mathcal{I}}{:}\, b \in C^{\mathcal{I}}\},\\
&(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}},\\
&(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}},\\
&(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} - C^{\mathcal{I}}.
\end{aligned}$$

A knowledge base in description logics is a set of axioms of the form

$$Subsumptions : C \sqsubseteq D$$

where C and D are concept terms. We write $C = D$ for $C \sqsubseteq D$ and $D \sqsubseteq C$. For example, we can define "a man is a human and a male" as follows.

$$Man = Human \sqcap Male$$

Note that the concept Man is defined by the concepts $Human$ and $Male$.

Chapter 3

Predicate-hierarchy and eventuality in order-sorted logic

In this chapter, we develop an extended order-sorted logic with predicate-hierarchy and eventuality.

Section 3.1 considers how we introduce the notions of predicate-hierarchy and eventuality and their corresponding reasoning mechanism into an order-sorted logic. Here we discuss the problems of the framework of former order-sorted logics (or reasoning systems with type expressions) that arise from dealing with a predicate-hierarchy. In Section 3.2, we define the syntax and the semantics of the proposed logic. In Section 3.3, we formalize a Horn clause calculus with inference rules of predicate-hierarchy that include argument supplementation, and then develop a resolution based on this calculus and an order-sorted unification algorithm. In Section 3.4, we evaluate the capability of our logic as a knowledge representation system. Hence, we show that a query system defined by the resolution we develop provides the reasoning mechanism required in our motivational examples given in Chapter 2, and give an example of its application to a legal reasoning.

3.1 Event, Property, and Hierarchy

In this section, we consider the notions of predicate-hierarchy and eventuality, and discuss the problems of the framework of standard logics that arise from dealing with them.

3.1.1 A hierarchy of predicates

In a deduction system with sort-hierarchy, we attempt to introduce a predicate-hierarchy built by a binary relation $\sqsubset_P$ (which we call *a subpredicate relation*) on a set of predicate symbols. For any predicates p_1, p_2, the subpredicate declaration $p_1 \sqsubset_P p_2$ means that p_1 is a subordinate predicate of p_2 and that p_2 is a superordinate predicate of p_1. We assume that every subpredicate relation $\sqsubset_P$ is anti-symmetric. Within the framework of logic programming, the subpredicate relation to build a predicate-hierarchy is expressed by a set of formulas of the logical implication form $p(x) \rightarrow q(x)$ (expressed as a logic representation [46]). By this logical implication form, the following formula

$$fly(x) \rightarrow move(x)$$

denotes that *move* is a superordinate predicate of *fly*, whereby we can infer that if $fly(a)$ holds, then $move(a)$ does.

However, we contend that this form cannot completely represent the relationship between predicates in a hierarchy. The problem comes from the fact that each predicate with its own fixed argument structure leads to a disconnection (e.g. by a difference in the number of arguments) between predicates related to each other in a hierarchy. For example, suppose that the following formulas expressed by the predicates *explain* and *talk* with two arguments are given as:

$$explain(john\colon man, c\colon book)$$
$$talk(john\colon man, mary\colon woman)$$

In the example, the first argument of both predicates is the same, but the second arguments are different. The role of both first arguments is "agent," and the role of the second argument in *explain* is "object" but in *talk* is "coagent." Therefore, the second arguments cannot be unified between *explain* and *talk* when the predicate *talk* is derived from the predicate *explain* along the subpredicate declaration $explain \sqsubset_P talk$. Moreover, if we attempt to write a subpredicate declaration by the logical implication form, we may have to write several formulas to associate arguments in the premise $p(..)$ with those in the conclusion $q(..)$ (e.g. $p(x,y) \rightarrow q(x,y)$, $p(x,y) \rightarrow q(y,x)$, $p(x,x) \rightarrow q(x,x), \ldots$). Hence, they would become more complicated as the number of predicates in the hierarchy increased.

Next we will discuss an inference machinery from a subpredicate relation $\sqsubset_P$ in which each predicate in the predicate-hierarchy may have a different argument structure. Suppose that the subpredicate declaration

$$explain \sqsubset_P talk$$

is given. To fill the gap between the fixed argument structures of predicates *explain* and *talk* we need to supplement missing arguments by appropriate arguments, depending on the argument structures and their argument roles. How do we obtain the necessary information about the argument structures of predicates? In our approach, we use *argument labels*[1] to render arguments roles, e.g., the argument labels *agt* and *obj* written as:

$$explain(\underline{agt} \Rightarrow john\colon man, \underline{obj} \Rightarrow c_1\colon book)$$

uniquely represent the argument roles to determine the argument structure of the predicate *explain*. In this example, *agt* is an argument label representing "agent," and *obj* is an argument label representing "object." This notation would be able to provide the following reasoning mechanism in a query system. Let the fact

$$explain(agt \Rightarrow john\colon man, obj \Rightarrow c_1\colon book)$$

be given. The query

$$?\text{-}talk(agt \Rightarrow x\colon person, coagt \Rightarrow y\colon person)$$

will yield *yes* in the following way.

[1]The argument labels differ from the attribute labels in the Ψ-terms proposed in LOGIN [1].

Fact: $explain(agt \Rightarrow john{:}man, obj \Rightarrow c_1{:}book)$

$\downarrow$ (1) derivation of the superordinate predicate

$talk(agt \Rightarrow john{:}man, obj \Rightarrow c_1{:}book)$

$\downarrow$ (2) deletion of an argument

$talk(agt \Rightarrow john{:}man)$

$\downarrow$ (3) addition of an argument

$talk(agt \Rightarrow john{:}man, coagt \Rightarrow c_2{:}person)$

$\uparrow$ (4) sorted substitution

Query: $talk(agt \Rightarrow x{:}person, coagt \Rightarrow y{:}person)$

In the above reasoning process,

(1) derives the predicate *talk* from the subordinate predicate *explain*,

(2) deletes the second argument $obj \Rightarrow c_1{:}book$ that is a surplus argument in the predicate *talk*,

(3) adds the argument $coagt \Rightarrow c_2{:}person$ that is deficient in the arguments of the predicate *talk*,

(4) substitutes $agt \Rightarrow x{:}person$ for the first argument $agt \Rightarrow john{:}man$ along the subsort declaration $man \sqsubset_S person$, and $coagt \Rightarrow y{:}person$ for the second argument $coagt \Rightarrow c_2{:}person$.

In the manipulations of (2) and (3), the addition and deletion of arguments depend on the argument structures specified by the argument labels in *talk* and *explain*. Moreover, to make the added argument in (3), we define the scope of every argument as a sort in which each argument label determines the sort of the term used to express that argument. Let *SCP* be a function from the set of argument labels to the set of sorts. For the argument labels *agt, obj, coagt*, if we define a function *SCP* such that

$$\begin{aligned} SCP(agt) &= person, \\ SCP(obj) &= thing, \\ SCP(coagt) &= person, \end{aligned}$$

then the sort *person* indicates the scope of *agt* and *coagt*, and the sort *thing* indicates the scope of *obj*. Therefore, the added argument $c_2{:}person$ in (3) can be supplied from the scope $SCP(coagt) = person$ indicated by the argument label *coagt*.

3.1.2 Predicates as Event and Property

In knowledge representation, predicate symbols can be used to represent attributes, properties, or states of objects. For example, $walk(x{:}man)$ means that a man is walking, and $red(y{:}apple)$ means that an apple is red. However, usage of the predicates *walk* and *red* in these assertions may not be consistent. That is, the predicate in $walk(x{:}man)$ implies an event, whereas the predicate in $red(y{:}apple)$ implies a property. Alternatively, the predicate *walk* may also be interpreted as the event "is walking" or the property "can walk." As a result, the two kinds of usage would give rise to erroneous reasoning from a predicate-hierarchy, unless an inference mechanism takes account of the distinction between event and property (as we have seen in section 2.1.3).

In the following, we give a specification that can deal with predicates to be used as events or properties. First, we distinguish between event and property assertions and solve the ambiguity of these assertions by introducing a different notation that identifies each predicate as either an event or a property into an order-sorted logic. Second, we discuss a mechanism for supplementing the deficient arguments in a predicate on the basis of the distinction between the predicate interpreted as an occurrence of an event, and interpreted as a property of objects.

In a typed logic programming language, we can regard $c\colon s$ as an existential expression and $x\colon s$ as a universal expression where $c\colon s$ is a constant of sort s and $x\colon s$ is a variable of sort s. For example, $x\colon bird$ denotes all birds, and $c\colon bird$ denotes a bird where c is a new constant. Given a language with predicates $p_1, \ldots, p_n$, we specify two aspects of a predicate as follows.

Specification 3.1 (Two aspects of a predicate) *For any predicate p_i, $p_i^\bullet$ is the predicate as event and $p_i^\sharp$ is the predicate as property. The concept of assertions defined as an event and as a property is given below:*

Event assertion: *The predicate as event expresses that there is a fact or an occurrence of such an event. An argument of the predicate is limited to an object, so that the following assertion is interpreted as "A bird is flying."*

$$fly^\bullet(c\colon bird) \simeq \exists x\colon bird[fly\_as\_event(x\colon bird)]^2$$

Property assertion: *The predicate as property expresses an attribute of objects. An argument of the predicate is universal in the domain indexed by a sort, so that the following form means "All birds have the property of flight."*

$$fly^\sharp(x\colon bird) \simeq \forall x\colon bird[fly\_as\_property(x\colon bird)]$$

When an argument is supplemented to a predicate in the process of predicate-hierarchy reasoning, the two aspects of the predicate naturally cause differently quantified arguments.

Specification 3.2 *For the predicate $p^\bullet$ as event, the added argument should be one occurrence of an object corresponding to one event. For the predicate $p^\sharp$ as property, the added argument should be globally all objects in a sort but not a unique object. Therefore, we add an existential term $c\colon s$ (representing an object within the sort s) to a predicate interpreted as event, and we add a universal term $x\colon s$ (representing all objects in the sort s) to a predicate interpreted as property.*

Given the predicates $hit^\bullet$ and $hit^\sharp$ with argument labels agt and coagt, the deficient arguments in their predicates are supplemented differently as follow.

[2] We use these expressions to explain the assertions as the closest representation of event and property in first-order predicate logic.

Predicate as event:

$hit^{\bullet}(agt \Rightarrow john\colon man)$

$\downarrow$ *addition of an argument*

$hit^{\bullet}(agt \Rightarrow john\colon man, coagt \Rightarrow \underline{c\colon person})$

(John hit a person.)

where c is a new constant.

Predicate as property:

$hit^{\sharp}(agt \Rightarrow john\colon man)$

$\downarrow$ *addition of an argument*

$hit^{\sharp}(agt \Rightarrow john\colon man, coagt \Rightarrow \underline{x\colon person})$

(John has the property of hitting every person.)

where x is a new variable.

The argument label *coagt* in the predicates $hit^{\bullet}$ and $hit^{\sharp}$ determines the sort *person* as the scope of the second argument due to $SCP(coagt) = person$. Hence, the term $c\colon person$ is added to the predicate $hit^{\bullet}$ as event where c is a new constant, whereas the term $x\colon person$ is added to the predicate $hit^{\sharp}$ as property where x is a new variable.

3.2 An order-sorted logic with hierarchies and eventuality

In this section, we define the syntax and semantics of an extended order-sorted logic with hierarchies and eventuality. This formalization includes the following notions to define both sort and predicate hierarchies and predicates that distinguish between events and properties.

(1) Syntax

- A sorted signature Σ with hierarchical predicates that includes subsort declarations, function declarations, subpredicate declarations and argument structure declarations.
- Sorted terms and formulas that are composed of sorted variables and the sort, function and hierarchical predicate symbols in Σ.

(2) Semantics

- An $H\Sigma$-structure that consists of the universe and an interpretation of the sort, function and predicate symbols in Σ and satisfies the subsort declarations, the function declarations and the subpredicate and argument structure declarations.
- Two translations (argument supplementation and composition of predicates) in a structure that are used to interpret subpredicate declarations.

(3) Inference system

- A Horn clause calculus that includes
  - a generalization rule and specialization rule of predicate-hierarchy
  - argument supplementation for ill-argumented atoms.
- A Horn clause resolution that is based on the Horn clause calculus and an order-sorted unification algorithm.
- A query system with existential and universal variables.

3.2.1 Language and signature

We first define the syntax of an extended order-sorted logic with hierarchies and eventuality. In order-sorted first-order languages, variables, functions and predicates may have sorts that are ordered and different.

Definition 3.1 *An alphabet for an order-sorted first-order language $\mathcal{L}$ consists of the following symbols.*

(1) $\mathcal{S}$ *is a set of sort symbols* $s_1, s_2, \ldots$ *with the greatest sort* $\top$ *and the least* $\bot$.

(2) $\mathcal{F}_n$ *is a set of* n*-ary function symbols* $f_1, f_2, \ldots$.

(3) $\mathcal{P}_n$ *is a set of* n*-ary predicate symbols* $p_1, p_2, \ldots$.

(4) $\mathcal{V}_s$ *is a countably infinite set of variables* $x_1{:}\, s, x_2{:}\, s, \ldots$ *of sort* s.

(5) AL *is a set of predicate argument labels* $a_1, a_2, \ldots$.

(6) $\neg, \wedge, \vee, \rightarrow, \forall, \exists$ *are the connectives and the quantifiers.*

(7) $(,), \Rightarrow, \bullet, \sharp$ *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. $\mathcal{V}$ denotes the set $\bigcup_{s\in\mathcal{S}} \mathcal{V}_s$ of the variables of all sorts. In order to indicate each argument role, the language $\mathcal{L}$ contains predicate argument labels.

Definition 3.2 (Declaration) *A declaration over $\mathcal{S}\cup\mathcal{F}\cup\mathcal{P}$ (for an order-sorted first-order language $\mathcal{L}$) 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* $s_1 \sqsubset_S s_2$,

(2) $\mathcal{D}_\mathcal{F}$ *is a set of function declarations of the form* $f{:}\, s_1 \times \ldots \times s_n \rightarrow s$ *where* $f \in \mathcal{F}_n$ *and* $n \geq 0$,

(3) $\mathcal{D}_\mathcal{P}$ *is a set of subordinate predicate declarations (or simply subpredicate declarations) of the form* $p_1 \sqsubset_P p_2$ *where* $p_1, p_2 \in \mathcal{P}$ *and argument structure declarations of the form* $p{:}\, \{(a_1, s_1), \ldots, (a_n, s_n)\}$ *where* $p \in \mathcal{P}_n$, $n \geq 0$, *and* $a_i \neq a_j$ *if* $i \neq j$,

(4) If $p{:}\, \{(a_1, s_1), \ldots, (a_n, s_n)\} \in \mathcal{D}_\mathcal{P}$, *then* $ARG(p) = \{a_1, \ldots, a_n\}$ *and, for* $1 \leq i \leq n$, $SCP(a_i) = s_i$ *where*

– *SCP is a function from AL to* $\mathcal{S}$,

– *ARG is a function from* $\mathcal{P}$ *to* 2^{AL}.

In the above declaration, the subsort declarations (in $\mathcal{D}_\mathcal{S}$) and the subordinate predicate declarations (in $\mathcal{D}_\mathcal{P}$) express a sort-hierarchy and a predicate-hierarchy respectively. $SCP(a_i)$ denotes a sort as the scope of the argument labeled by a_i. $ARG(p)$ indicates a finite set of argument labels as the unique argument structure of predicate p. We use the abbreviation $ARG(p-q)$ to denote $ARG(p) - ARG(q)$.

Definition 3.3 (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* $\mathcal{S} \cup \mathcal{F} \cup \mathcal{P}$.

Example 3.1 *We show an example of a sorted signature* Σ *for an order-sorted first-order language* $\mathcal{L}$ *with hierarchial predicates. The sorted signature* Σ *comprises the following symbols*

$$\begin{aligned}
\mathcal{S} &= \{person, man, woman, animal, bird, wallet, thing, \top, \bot\},\\
\mathcal{F} &= \{john, mary\},\\
\mathcal{P} &= \{p_1, p_2, q\},
\end{aligned}$$

and the declaration $\mathcal{D} = (\mathcal{D}_\mathcal{S}, \mathcal{D}_\mathcal{F}, \mathcal{D}_\mathcal{P})$ *with*

$$\begin{aligned}
\mathcal{D}_\mathcal{S} &= \{man \sqsubset_S person, woman \sqsubset_S person, wallet \sqsubset_S thing,\\
&\quad \bot \sqsubset_S man, \bot \sqsubset_S woman, \bot \sqsubset_S bird, \bot \sqsubset_S wallet,\\
&\quad person \sqsubset_S \top, animal \sqsubset_S \top, thing \sqsubset_S \top\},\\
\mathcal{D}_\mathcal{F} &= \{john\colon \to man, mary\colon \to woman, c\colon \to wallet\},\\
\mathcal{D}_\mathcal{P} &= \{hit\colon \{(agt, person), (coagt, person)\},\\
&\quad steal\colon \{(agt, person), (obj, \top)\},\\
&\quad fly\colon \{(sbj, \top)\}\},
\end{aligned}$$

Unlike ordinary order-sorted logics, sorted signatures with hierarchical predicates contain subpredicate declarations of the form $p_1 \sqsubset_P p_2$ and argument structure declarations of the form $p\colon \{(a_1, s_1), \ldots, (a_n, s_n)\}$ that indicate a predicate-hierarchy and the various argument structures of the predicates respectively. In the above example, $q \sqsubset_P p_1$ (in $\mathcal{D}_\mathcal{P}$) declares that the predicate q is a subpredicate of predicate p_1, and $p_1\colon \{(agt, person), (coagt, person)\}$ declares that the predicate p_1 consists of two arguments labeled with agt and $coagt$ (which mean an agent and a co-agent respectively) and that the sorts of both these arguments are $person$.

3.2.2 Order-sorted terms and formulas

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

Definition 3.4 (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:*

(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 $c\colon \to s \in \mathcal{D}_{\mathcal{F}}$.

(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 $f\colon s_1 \times \ldots \times s_n \to s \in \mathcal{D}_{\mathcal{F}}$.

(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' where $s' \sqsubset_S s$. We denote by *TERM* the set of all (order-sorted) terms

$$\bigcup_{s \in \mathcal{S}} TERM_s.$$

We define the function *Var* as assigning any order-sorted term to the set of free variables occurring in it.

Definition 3.5 *The function Var from TERM into $2^{\mathcal{V}}$ is defined by:*

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

(ii) $Var(f(t_1, \ldots, t_n)\colon s) = \bigcup_{1 \leq i \leq n} Var(t_i)$. In particular, $Var(c\colon s) = \emptyset$ where $c \in \mathcal{F}_0$.

$TERM_0$ denotes the set of all the 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.

For every predicate symbol $p \in \mathcal{P}$, we use the notation (which we introduced in Section 3.1.2) that $p^\bullet$ is *the event predicate* and $p^\sharp$ *the property predicate*. Given the set $\mathcal{P}$ of all predicate symbols, we write $\mathcal{P}^\bullet$ for the set $\{p^\bullet \mid p \in \mathcal{P}\}$ of all event predicates and $\mathcal{P}^\sharp$ for the set $\{p^\sharp \mid p \in \mathcal{P}\}$ of all property predicates. We define the function $[\ ]\colon \mathcal{P}^\bullet \cup \mathcal{P}^\sharp \to \mathcal{P}$ such that $[\varphi] = p$ if φ is $p^\bullet$ or $p^\sharp$ and the function $\langle\ \rangle\colon \mathcal{P}^\bullet \cup \mathcal{P}^\sharp \to \{\bullet, \sharp\}$ such that

$$\langle\varphi\rangle = \begin{cases} \bullet & \text{if } \varphi = p^\bullet, \\ \sharp & \text{if } \varphi = p^\sharp. \end{cases}$$

For instance, the predicate symbols of $fly^\bullet$ and $fly^\sharp$ are given by

$$[fly^\bullet] = [fly^\sharp] = fly$$

and the eventualities of $fly^\bullet$ and $fly^\sharp$ are given by

$$\begin{aligned} \langle fly^\bullet \rangle &= \bullet, \\ \langle fly^\sharp \rangle &= \sharp. \end{aligned}$$

Definition 3.6 (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:*

(1) If $t_1, \ldots, t_n$ are terms of $s_1, \ldots, s_n$, then $p^\bullet(a_1 \Rightarrow t_1, \ldots, a_n \Rightarrow t_n)$ is the atomic formula for an event predicate where $p^\bullet \in \mathcal{P}^\bullet$ and $p\colon \{(a_1, s_1), \ldots, (a_n, s_n)\} \in \mathcal{D}_{\mathcal{P}}$,

(2) If $t_1, \ldots, t_n$ are terms of $s_1, \ldots, s_n$, then $p^\sharp(a_1 \Rightarrow t_1, \ldots, a_n \Rightarrow t_n)$ is the atomic formula for a property predicate where $p^\sharp \in \mathcal{P}^\sharp$ and $p\colon\{(a_1, s_1), \ldots, (a_n, s_n)\} \in \mathcal{D}_\mathcal{P}$,

(3) If A and B are formulas, then $(\neg A)$, $(A \wedge B)$, $(A \vee B)$, $(A \to B)$, $(\forall x\colon sA)$, and $(\exists x\colon sA)$ are formulas.

The atomic formula for an event predicate is called *an event atomic formula* (or simply *an event atom*) and the atomic formula for a property predicate is called *a property atomic formula* (or simply *a property atom*).

Example 3.2 *For the sorted signature Σ of Example 3.1, we give examples of order-sorted formulas shown as:*

$$hit^\bullet(agt \Rightarrow john\colon man, coagt \Rightarrow mary\colon woman),$$
$$steal^\bullet(agt \Rightarrow john\colon man, obj \Rightarrow c\colon wallet),$$
$$fly^\sharp(sbj \Rightarrow x\colon bird).$$

The first and second event atoms express that "the agent *john* hit the co-agent *mary*" and "the agent *john* stole the object *wallet*" respectively. The third property atom describes that "birds have the property *fly*," i.e., "birds can fly."

Two atoms $\varphi(a_1 \Rightarrow t_1, \ldots, a_n \Rightarrow t_n)$ and $\varphi'(b_1 \Rightarrow r_1, \ldots, b_m \Rightarrow r_m)$ are *equivalent* if $\varphi \equiv \varphi'$ 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.

Example 3.3 *Let A and B be the atoms given by*

$$A \equiv rob\_with\_violence^\bullet(agt \Rightarrow john\colon man, coagt \Rightarrow mary\colon woman),$$
$$B \equiv rob\_with\_violence^\bullet(coagt \Rightarrow mary\colon woman, agt \Rightarrow john\colon man).$$

Then $A \approx B$.

We define the set $FVar(A)$ of free variables occurring in the formula A as follows.

Definition 3.7 *The function $FVar$ from the set $FORM$ of formulas into $2^\mathcal{V}$ is defined by:*

(i) $FVar(\varphi(a_1 \Rightarrow t_1, \ldots, a_n \Rightarrow t_n)) = \bigcup_{1 \le i \le n} Var(t_i)$ where $\varphi \in \mathcal{P}^\bullet \cup \mathcal{P}^\sharp$,

(ii) $FVar(\neg A) = FVar(A)$,

(iii) $FVar(A * B) = FVar(A) \cup FVar(B)$ for $* \in \{\vee, \wedge, \to\}$, and

(iv) $FVar(*A) = FVar(A) - \{x\colon s\}$ for $* \in \{\forall x\colon s, \exists x\colon s\}$.

A formula F is said to be a sentence if the formula is without free variables, i.e. $FVar(F) = \emptyset$. A formula F is said to be a ground formula if the formula is without variables or quantifiers.

An argument is an ordered pair (a, t) where a is an argument label and t is an order-sorted term, denoted by $a \Rightarrow t$. Let μ be a set of arguments. $\bar{\mu}$ is a finite sequence of arguments in μ. We write $\varphi(\bar{\mu})$ where $\varphi \in \mathcal{P}^\bullet \cup \mathcal{P}^\sharp$, when we express any sequence of arguments in μ, i.e., any argument structure constructed by the elements of μ. In order to obtain the set of argument labels occurring in $\bar{\mu}$, we define the function $LS(\bar{\mu}) = \{a_i \in AL \mid a_i \Rightarrow t_i \in \mu\}$.

3.2.3 Σ-structure

We have mentioned in the previous section that atoms composed of the same predicate symbol and the same arguments can be regarded as equivalent. Hence, in the following definition of the semantics of our logic, we will define that the ordering of arguments in a predicate does not alter the interpretation of its atomic formula. For example, the following formulas

$$p^{\bullet}(a_1 \Rightarrow t_1, a_2 \Rightarrow t_2) \textit{ and } p^{\bullet}(a_2 \Rightarrow t_2, a_1 \Rightarrow t_1)$$

are regarded as semantically identical because they have the same argument structure and terms. Instead of the ordering of arguments, we can decide the equivalence by the argument labels that determine each argument role .

Moreover, a requirement of our order-sorted logic is to derive $q^{\bullet}(\bar{\mu})$ from $p^{\bullet}(\bar{\mu}')$ (or $q^{\sharp}(\bar{\mu})$ from $p^{\sharp}(\bar{\mu}')$) along the subpredicate relation $p \sqsubset_P q$ where the argument structures $\bar{\mu}, \bar{\mu}'$ of predicates p, q may be different (e.g. the numbers of arguments). Therefore, the interpretation of the subpredicate relation must include a translation of the argument structure $\bar{\mu}$ of p to the argument structure $\bar{\mu}'$ of q. To begin, we shall give a sorted structure (called a Σ-structure) for the semantics of order-sorted logic with hierarchies and eventuality.

Definition 3.8 *Let $\Sigma = (\mathcal{S}, \mathcal{F}, \mathcal{P}, \mathcal{D})$ be a sorted signature with hierarchical predicates. A Σ-structure is an ordered pair $M = (I, U)$ such that*

- *U is a non-empty set (the universe of M), and*
- *I is a function such that*
  - *if $s \in \mathcal{S}$ then $I(s) \subseteq U$,*
  - *if $s_i \sqsubset_S s_j \in \mathcal{D}_{\mathcal{S}}$ then $I(s_i) \subseteq I(s_j)$,*
  - *if $f \in \mathcal{F}_n$ and $f: s_1 \times \ldots \times s_n \rightarrow s \in \mathcal{D}_{\mathcal{F}}$ then $I(f): I(s_1) \times \ldots \times I(s_n) \rightarrow I(s)$, and*
  - *if $p \in \mathcal{P}_n$ and $p: \{(a_1, s_1), \ldots, (a_n, s_n)\} \in \mathcal{D}_{\mathcal{P}}$ then $I(\varphi) \subseteq \{\rho \in (ARG(p) \rightarrow U) \mid \forall a_i \in ARG(p), \rho(a_i) \in I(SCP(a_i))\}$[3] for $\varphi \in \{p^{\bullet}, p^{\sharp}\}$.*

We call a function $\rho \in (ARG(p) \rightarrow U)$ *an argument interpretation*, which is used for interpreting the event and property predicates $p^{\bullet}, p^{\sharp}$ in the definition of the Σ-structure. In the semantics of first-order logic, the interpretation (denoted $I(p)$) of an n-ary predicate p is a set of n-tuples on the universe U (i.e. a subset of U^n). Instead of an n-tuple on U, we use an argument interpretation:

$$\rho \;=\; \{(a_1, d_1), (a_2, d_2), \cdots, (a_n, d_n)\}$$

defined as a set of ordered pairs (a, d) where $a \in AL$ and $d \in U$. That is, an argument interpretation is a subset (i.e. $\rho \subseteq AL \times U$) of the cartesian product $AL \times U$ of two sets AL and U where AL is the set of argument labels and U is the universe.

[3] For sets X and Y, the set of all functions from X to Y is denoted by $(X \rightarrow Y)$.

Remark 3.1 *In first-order logic, an n-ary predicate (or an n-ary relation) is defined by a set of n-tuples* $(d_1, d_2, \ldots, d_n)$ *on* U. *In contrast to an n-tuple* $(d_1, d_2, \ldots, d_n)$, *we use an argument interpretation* $\{(a_1, d_1), (a_2, d_2), \ldots, (a_n, d_n)\}$ *where* $d_1, \ldots, d_n$ *are labeled but not ordered. Note that the argument interpretation is the function* $\rho: ARG(p) \to U$ *with* $ARG(p) = \{a_1, \ldots, a_n\}$ *(corresponding to a n-tuple* $(d_1, d_2, \ldots, d_n)$ *on* U*) such that* $\rho(a_1) = d_1, \ldots, \rho(a_n) = d_n$. *Similarly, a predicate is interpreted as a set of n-tuples on* U *(i.e. a subset of* U^n*).* $I(p^\bullet)$ *or* $I(p^\sharp)$ *is defined as a set of argument interpretations* ρ *(i.e. a subset of the set of all argument interpretations).*

Example 3.4 *Given a sorted signature* $\Sigma = (\mathcal{S}, \mathcal{F}, \mathcal{P}, \mathcal{D})$ *with*

$$\begin{aligned} \mathcal{S} &= \{person, \top, \bot\}, \\ \mathcal{F} &= \emptyset, \\ \mathcal{P} &= \{p\}, \\ \mathcal{D} &= (\emptyset, \emptyset, \{p\colon \{(agt, person), (obj, \top)\}\}), \end{aligned}$$

we have $ARG(p) = \{agt, obj\}$, $SCP(agt) = person$, *and* $SCP(obj) = \top$. *Let* $U = \{d_1, d_2, d_3\}$ *and* I *be a function such that* $I(person) = \{d_1\}$ *and* $I(\top) = U$. *If* $I(p^\bullet) = \emptyset$ *and* $I(p^\sharp) = \{\rho_1, \rho_2\}$ *such that*

$$\begin{aligned} \rho_1 &= \{(agt, d_1), (obj, d_1)\}, \\ \rho_2 &= \{(agt, d_1), (obj, d_2)\}, \end{aligned}$$

then (U, I) *is a* Σ*-structure, i.e. it satisfies the conditions of Definition 3.8.*

3.2.4 Restricted structure for hierarchical predicates

We will define a restricted Σ-structure that satisfies the constraints given by introducing hierarchical predicates. In order that the structure interprets subpredicate declarations in Σ, we introduce two translations (argument supplementation and composition of predicates) in the structure. As a prerequisite for an interpretation of hierarchical predicates, we consider adjusting the argument structure of predicate p to the argument structure of predicate q when $p \sqsubset_P q$ is declared in $\mathcal{D}_\mathcal{P}$. 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 ones, and

(2) Additional arguments: the set of q's arguments that are not p's arguments.

Let φ be $p^\bullet$ or $p^\sharp$ and let $I(\varphi)$ be an interpretation of predicate φ (defined by a set of argument interpretations $\rho \in (ARG(p) \to U)$). For $p, q \in \mathcal{P}$, the common arguments of p and q are

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

where $\rho \in I(\varphi)$. The additional arguments are

(i) $\{(a_1, d_1), \ldots, (a_n, d_n)\}$ *for some* $d_1 \in I(SCP(a_1)), \ldots, d_n \in I(SCP(a_n))$ *or*

(ii) $\{(a_1, x_1), \ldots, (a_n, x_n)\}$ *for all* $x_1 \in I(SCP(a_1)), \ldots, x_n \in I(SCP(a_n))$

where $ARG(q-p) = \{a_1, \ldots, a_n\}$. (i) is the additional arguments for the event predicate $p^\bullet$ and (ii) for the property predicate $p^\sharp$.

We define the function $LS^*(\rho) = \{a_i \in AL \mid (a_i, d_i) \in \rho\}$ to obtain the set of the argument labels from an argument interpretation ρ.

Definition 3.9 (Argument supplementation in structures) *Let $\Sigma = (\mathcal{S}, \mathcal{F}, \mathcal{P}, \mathcal{D})$ be a sorted signature with hierarchical predicates and $M = (I, U)$ be a Σ-structure. The event interpretation $\iota^\bullet_q$ (resp. the property interpretation $\iota^\sharp_q$) for the argument supplementation to predicate q is a translation of all the argument interpretations in $I(p^\bullet)$ (resp. $I(p^\sharp)$) to a set of argument interpretations such that:*

$$\iota^\bullet_q(I(p^\bullet)) = \{(\rho \cap ARG(q) \times U) \cup \{(a_1, d_1), \ldots, (a_n, d_n)\} \mid \rho \in I(p^\bullet)\},$$

$$\iota^\sharp_q(I(p^\sharp)) = \{(\rho \cap ARG(q) \times U) \cup \{(a_1, x_1), \ldots, (a_n, x_n)\} \mid \rho \in I(p^\sharp), x_i \in I(SCP(a_i))\},$$

where $\{a_1, \ldots, a_n\} = ARG(q) - LS^*(\rho)$[4] and, for $1 \leq i \leq n$, $d_i \in I(SCP(a_i))$.

Example 3.5 *Given a sorted signature $\Sigma = (\mathcal{S}, \mathcal{F}, \mathcal{P}, \mathcal{D})$ with*

$$\begin{aligned}
\mathcal{S} &= \{person, \top, \bot\},\\
\mathcal{F} &= \emptyset,\\
\mathcal{P} &= \{p, q\},\\
\mathcal{D} &= (\emptyset, \emptyset, \{p\text{:}\{(agt, person), (obj, \top)\}, q\text{:}\{(agt, person), (coagt, person)\}\}),
\end{aligned}$$

we have $ARG(p) = \{agt, obj\}$ and $ARG(q) = \{agt, coagt\}$. Let $U = \{d_1, d_2, d_3, d_4, d_5\}$ and I be a function such that $I(person) = \{d_1, d_2, d_3\}$ and $I(\top) = U$. If $I(p^\bullet) = \{\varphi, \psi\}$ with

$$\begin{aligned}
\varphi &= \{(agt, d_1), (obj, d_4)\},\\
\psi &= \{(agt, d_2), (obj, d_5)\},
\end{aligned}$$

then

$$\iota^\bullet_q(I(p^\bullet)) = \{\{(agt, d_1), (coagt, d_3)\}, \{(agt, d_2), (coagt, d_3)\}\}.$$

In the following we define a composition

$$I(\varphi_1) \sqcap \ldots \sqcap I(\varphi_n)$$

of the interpretations of predicates $\varphi_1, \ldots, \varphi_n$ whereby all the argument interpretations in $I(\varphi_1) \cup \cdots \cup I(\varphi_n)$ are integrated into a set (denoted $I(\varphi_1) \sqcap \ldots \sqcap I(\varphi_n)$) of argument interpretations. Let $\varphi_1, \varphi_2, \ldots, \varphi_n$ be event predicates $p^\bullet_1, p^\bullet_2, \ldots, p^\bullet_n$ or property predicates $p^\sharp_1, p^\sharp_2, \ldots, p^\sharp_n$. Given the interpretations $I(\varphi_1), I(\varphi_2), \ldots, I(\varphi_n)$, we use the union $ARG(p_1) \cup ARG(p_2) \cup \cdots \cup ARG(p_n)$ to make a set obtained by integrating the argument interpretations in $I(\varphi_1)$, $I(\varphi_2)$, $\cdots$, $I(\varphi_n)$ and excluding the argument interpretations in disagreement (i.e. 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 $\rho_1 \in I(\varphi_1)$ and $\rho_2 \in I(\varphi_2)$ $(n = 2)$).

[4] We have $LS^*(\rho) = ARG(p)$, since an argument interpretation ρ is a member of $I(\varphi)$ by Definition 3.8.

Definition 3.10 (Composition of predicates in structures) *Let $M = (I, U)$ be a Σ-structure. The composition of predicates is a translation of two sets of argument interpretations into a set of argument interpretations as follows:*

$$I(\varphi_1) \sqcap I(\varphi_2) \;=\; \{\rho_1 \oplus \rho_2 \mid \rho_1 \in I(\varphi_1),\ \rho_2 \in I(\varphi_2),\ \rho_1 \oplus \rho_2 \neq \emptyset\}.$$

We define the operation $\oplus$ to construct an argument interpretation from two argument interpretations as follows.

$$\rho_1 \oplus \rho_2 = \begin{cases} \rho_1 \cup \rho_2 & \text{if } \rho_1(a) = \rho_2(a) \text{ for all } a \in LS^*(\rho_1) \cap LS^*(\rho_2), \\ \emptyset & \text{otherwise.} \end{cases}$$

Here the composition of two predicates can be expanded to the composition of n predicates. Let Π be $\{p_1^\bullet, \ldots, p_n^\bullet\}$ or $\{p_1^\sharp, \ldots, p_n^\sharp\}$.

$$\begin{aligned} \sqcap_{\varphi_i \in \Pi} I(\varphi_i) \;=\; & \{\rho_1 \oplus \rho_2 \oplus \cdots \oplus \rho_n \mid \rho_1 \in I(\varphi_1),\ \rho_2 \in I(\varphi_2), \ldots,\ \rho_2 \in I(\varphi_n), \\ & \rho_1 \oplus \rho_2 \oplus \cdots \oplus \rho_n \neq \emptyset\}. \end{aligned}$$

where $\rho_1 \oplus \rho_2 \oplus \cdots \oplus \rho_n$ denotes $(((\rho_1 \oplus \rho_2) \oplus \cdots) \oplus \rho_n)$.

Example 3.6 *Let $\Sigma = (\mathcal{S}, \mathcal{F}, \mathcal{P}, \mathcal{D})$ with $\mathcal{S} = \{s_1, s_2, \top, \bot\}$, $\mathcal{F} = \emptyset$, $\mathcal{P} = \{p_1, p_2\}$, and*

$$\mathcal{D} = (\emptyset, \emptyset, \{p_1\colon\{(agt, s_1)\}, p_2\colon\{(agt, s_1), (obj, s_2)\}\}).$$

If

$$\begin{aligned} I(p_1^\bullet) \;&=\; \{\{(agt, d_1)\}\}, \\ I(p_2^\bullet) \;&=\; \{\{(agt, d_2), (obj, d_1)\}, \{(agt, d_1), (obj, d_2)\}\}, \end{aligned}$$

then

$$I(p_1^\bullet) \sqcap I(p_2^\bullet) = \{\{(agt, d_1), (obj, d_2)\}\}.$$

$I(p_2^\bullet)$ has the two argument interpretations $\{(agt, d_2), (obj, d_1)\}, \{(agt, d_1), (obj, d_2)\}$, but the composition of $I(p_1^\bullet)$ and $I(p_2^\bullet)$ excludes the argument interpretation $\{(agt, d_2), (obj, d_1)\}$ that disagrees with the argument interpretation $\{(agt, d_1)\}$ in $I(p_1^\bullet)$.

Using the above translations (in Definition 3.9 and Definition 3.10), we define a restricted Σ-structure (which we will call an $H\Sigma$-structure) that satisfies the sorted signature Σ with hierarchical predicates. We define the one-step subpredicate relation $\sqsubset_P^1 = \{(p, q) \in \sqsubset_P \mid (p, r) \text{ or } (r, q) \notin \sqsubset_P\}$.

Definition 3.11 ($H\Sigma$-structure) *Let $\iota_q^\bullet$, $\iota_q^\sharp$ be the event and property interpretations for the argument supplementation to predicate q and $\Sigma = (\mathcal{S}, \mathcal{F}, \mathcal{P}, \mathcal{D})$ a sorted signature with hierarchical predicates. A Σ-structure $M = (I, U)$ is an $H\Sigma$-structure if the following conditions hold:*

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

$$\begin{aligned} \iota_q^\bullet(I(p^\bullet)) \;&\subseteq\; I(q^\bullet) \text{ and} \\ \iota_q^\sharp(I(p^\sharp)) \;&\subseteq\; I(q^\sharp). \end{aligned}$$

(2) If $p_1, \ldots, p_n, q \in \mathcal{P}$ *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* $q \sqsubset_P^1 p_i$*, then*

$$\begin{aligned} \iota_q^\bullet(\sqcap_{p_i^\bullet \in \Pi_1} I(p_i^\bullet)) &\subseteq I(q^\bullet) \text{ and} \\ \iota_q^\sharp(\sqcap_{p_i^\sharp \in \Pi_2} I(p_i^\sharp)) &\subseteq I(q^\sharp) \end{aligned}$$

where $\Pi_1 = \{p_1^\bullet, \ldots, p_n^\bullet\}$ *and* $\Pi_2 = \{p_1^\sharp, \ldots, p_n^\sharp\}$.

Example 3.7 *Consider the sorted signature* $\Sigma = (\mathcal{S}, \mathcal{F}, \mathcal{P}, \mathcal{D})$ *where* $\mathcal{S} = \{s, \top, \bot\}$, $\mathcal{F} = \emptyset$, $\mathcal{P} = \{p_1, p_2, q\}$, *and* $\mathcal{D} = (\emptyset, \emptyset, \mathcal{D}_\mathcal{P})$ *with*

$$\begin{aligned} \mathcal{D}_\mathcal{P} = \; & \{q \sqsubset_P p_1, q \sqsubset_P p_2\} \cup \\ & \{p_1 \colon \{(a_1, s), (a_2, s)\}, \\ & \; p_2 \colon \{(a_1, s), (a_3, \top)\}, \\ & \; q \colon \{(a_1, s), (a_3, \top)\} \; \}. \end{aligned}$$

Let $U = \{d_1, d_2, d_3, d_4\}$ *and* I *be a function such that*

$$\begin{aligned} I(p_1^\bullet) &= \{\rho_1\}, \\ I(p_2^\bullet) &= \{\rho_2\}, \\ I(q^\bullet) &= \{\rho_3\}, \end{aligned}$$

and

$$\begin{aligned} \rho_1 &= \{(a_1, d_1), (a_2, d_2)\}, \\ \rho_2 &= \{(a_1, d_1), (a_3, d_3)\}, \\ \rho_3 &= \{(a_1, d_1), (a_3, d_3)\}. \end{aligned}$$

The composition of predicates $p_1^\bullet, p_2^\bullet$ *is*

$$I(p_1^\bullet) \sqcap I(p_2^\bullet) = \{(a_1, d_1), (a_2, d_2), (a_3, d_3)\}$$

and therefore we can obtain the following

$$\iota_q^\bullet(I(p_1^\bullet) \sqcap I(p_2^\bullet)) = \{(a_1, d_1), (a_3, d_3)\}.$$

The function I *satisfies the second condition of the* $H\Sigma$*-interpretation in Definition 3.11.*

3.2.5 Interpretation and satisfiability

A variable assignment (or an assignment) in a Σ-structure $M = (I, U)$ 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 a Σ-structure $M = (I, U)$, $x \colon s$ a variable, and $d \in I(s)$. The assignment $\alpha[d/x \colon s]$ is defined by $\alpha[d/x \colon s] = \alpha - \{(x \colon s, \alpha(x \colon s))\} \cup \{(x \colon s, d)\}$. We write $\alpha[d_1/x_1 \colon s_1, \ldots, d_n/x_n \colon s_n]$ for $(((\alpha[d_1/x_1 \colon s_1])[d_2/x_2 \colon s_2]) \ldots)[d_n/x_n \colon s_n]$. That is, if $y \colon s = x_i \colon s_i$ for some $i \in \{1, \ldots, n\}$, then $\alpha[d_1/x_1 \colon s_1, \ldots, d_n/x_n \colon s_n](y \colon s) = d_i$. Otherwise, $\alpha[d_1/x_1 \colon s_1, \ldots, d_n/x_n \colon s_n](y \colon s) = \alpha(y \colon s)$.

Definition 3.12 *Let* $\mathcal{I} = \langle M, \alpha \rangle$ *be a* Σ*-interpretation. The denotation* $[\![\;]\!]_\alpha$ *is defined by*

(1) $[\![x\colon s]\!]_\alpha = \alpha(x\colon s)$,

(2) $[\![c\colon s]\!]_\alpha = I(c)$ *with* $I(c) \in I(s)$,

(3) $[\![f(t_1,\ldots,t_n)\colon s]\!]_\alpha = I(f)([\![t_1]\!]_\alpha,\ldots,[\![t_n]\!]_\alpha)$ *with* $I(f)([\![t_1]\!]_\alpha,\ldots,[\![t_n]\!]_\alpha) \in I(s)$.

If a Σ-interpretation $\mathcal{I}$ consists of an $H\Sigma$-structure M and an assignment α, then $\mathcal{I}$ is said to be an $H\Sigma$-interpretation. Let $\mathcal{I} = (M, \alpha)$ be a Σ-interpretation. The interpretation $\mathcal{I}[d_1/x_1\colon s_1,\ldots,d_n/x_n\colon s_n]$ with $x_1\colon s_1,\ldots,x_n\colon s_n \in \mathcal{V}$ and $d_1 \in I(s_1),\ldots,d_n \in I(s_n)$ is defined by $(M, \alpha[d_1/x_1\colon s_1,\ldots,d_n/x_n\colon s_n])$.

Remark 3.2 *Like structures, Σ-structures, and $H\Sigma$-structures, there are three classes of interpretations: (unsorted) interpretations, Σ-interpretations, and $H\Sigma$-interpretations. In particular, the interpretations considered in this paper are the Σ-interpretations and the $H\Sigma$-interpretations. By the above definitions, all $H\Sigma$-interpretations must be Σ-interpretations.*

Definition 3.13 *Let $\mathcal{I} = \langle M, \alpha\rangle$ be a Σ-interpretation and F an order-sorted formula. We define the satisfiability relation $\mathcal{I} \models F$ by the following rules:*

- $\mathcal{I} \models p^\bullet(a_1 \Rightarrow t_1,\ldots,a_n \Rightarrow t_n)$ *iff* $\{(a_1,[\![t_1]\!]_\alpha),\ldots,(a_n,[\![t_n]\!]_\alpha)\} \in I(p^\bullet)$,
- $\mathcal{I} \models p^\sharp(a_1 \Rightarrow t_1,\ldots,a_n \Rightarrow t_n)$ *iff* $\{(a_1,[\![t_1]\!]_\alpha),\ldots,(a_n,[\![t_n]\!]_\alpha)\} \in I(p^\sharp)$,
- $\mathcal{I} \models (\neg A)$ *iff* $\mathcal{I} \not\models A$,
- $\mathcal{I} \models (A \wedge B)$ *iff* $\mathcal{I} \models A$ *and* $\mathcal{I} \models B$,
- $\mathcal{I} \models (A \vee B)$ *iff* $\mathcal{I} \models A$ *or* $\mathcal{I} \models B$,
- $\mathcal{I} \models (A \to B)$ *iff* $\mathcal{I} \not\models A$ *or* $\mathcal{I} \models B$,
- $\mathcal{I} \models (\forall x\colon sA)$ *iff for all* $d \in I(s)$, $\mathcal{I}[d/x\colon s] \models A$ *holds*,
- $\mathcal{I} \models (\exists x\colon sA)$ *iff for some* $d \in I(s)$, $\mathcal{I}[d/x\colon s] \models A$ *holds*.

By the above definition, if an atomic formula is satisfied by an 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 $AS \in ATOM/\approx$, $\mathcal{I} \models A$ iff $\mathcal{I} \models B$.

We write $\mathcal{I} \models \Gamma$ ($\mathcal{I}$ is a model of Γ) if $\mathcal{I} \models F$ for all formula $F \in \Gamma$. We say that Γ is Σ-satisfiable if it has a Σ-model. Otherwise, we say that Γ is Σ-unsatisfiable if it has no Σ-models. We write $\Gamma \models F$ (F is a consequence of Γ) if every Σ-model of Γ is a Σ-model of a formula F. We write $B \models A$ if $\{B\} \models A$, i.e., $\{B\}$ is a singleton. In particular, we say that $\mathcal{I}$ is an $H\Sigma$-model if it is an $H\Sigma$-interpretation. We say that Γ is $H\Sigma$-satisfiable if it has an $H\Sigma$-model. Otherwise, we say that 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.

3.3 Deduction system

In Section 3.3, we formalize a Horn clause calculus with inference rules of predicate-hierarchy that include argument supplementation, and then develop a resolution based on this calculus and an order-sorted unification algorithm.

3.3.1 Horn clauses

We prepare to formalize a Horn clause calculus (based on [24]) for the order-sorted logic we propose in the previous section. Given an order-sorted first-order language $\mathcal{L}$, the extended order-sorted first-order language $\mathcal{L}^+$ is obtained by adjoining to the set of *supplement constants* c_a for all $a \in AL$. We assume that every signature for the extended order-sorted first-order language $\mathcal{L}^+$ with hierarchical predicates contains the function declarations $c_a : \rightarrow SCP(a)$ for all $a \in AL$. We write $TERM^+$ for the set of all terms and $FORM^+$ for the set of all formulas in language $\mathcal{L}^+$.

We first define Horn clause forms (used as the syntax of the logic programming language PROLOG) in language $\mathcal{L}^+$.

Definition 3.14 (Horn clauses) *Let $L, L_1, \ldots, L_n$ be atoms. A goal G is denoted by the form*

$$G := L_1, \ldots, L_n.$$

where $n \geq 0$. In particular, we use the notation $\Box$ if $n = 0$, i.e. the goal with no elements. A clause C is denoted by the form

$$C := L \leftarrow G.$$

The set of all clauses is denoted by HCL.

The function $CVar$ from the set of all clauses into $2^{\mathcal{V}}$ is defined by:

$$CVar(L \leftarrow G) = (\bigcup_{L_i \in G} FVar(L_i)) \cup FVar(L).$$

We write $\forall F$ for the universal closure $\forall x_1 : s_1 \ldots \forall x_m : s_m F$ where F is a quantifier-free formula with $FVar(F) = \{x_1 : s_1, \ldots, x_m : s_m\}$ (similarly, we write $\exists F$ for the existential closure). Then clauses $L \leftarrow G$ with $G = L_1, \ldots, L_n$ represent the universal closures $\forall(L_1 \wedge \ldots \wedge L_n \rightarrow L)$.

Definition 3.15 (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 supplementation (which we will explain in the following sections) is applied.

3.3.2 Sorted substitution

Definition 3.16 (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 $\{t_1/x_1 : s_1, \ldots, t_1/x_n : s_n\}$ 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 sorted ground substitution (or simply a ground

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 θ 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 $\theta{\uparrow}V = \{\theta(x{:}\,s)/x{:}\,s \mid x{:}\,s \in V \cap Dom(\theta)\}$.

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

Definition 3.17 *Let A, B be order-sorted formulas, $L, L_1, \ldots, L_n$ atoms, G a goal, and θ a sorted substitution. $E\theta$ is defined by:*

- *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 $E = \varphi(a_1 \Rightarrow t_1, \ldots, a_m \Rightarrow t_m)$ where $\varphi \in \mathcal{P}^{\bullet} \cup \mathcal{P}^{\natural}$, then $E\theta = \varphi(a_1 \Rightarrow t_1\theta, \ldots, a_m \Rightarrow t_m\theta)$,*
- *If $E = \neg A$, then $E\theta = \neg A\theta$,*
- *If $E = A * B$ for $* \in \{\wedge, \vee\}$, then $E\theta = A\theta * B\theta$,*
- *If $E = *A$ for $* \in \{\forall x{:}\,s, \exists x{:}\,s\}$, then $E\theta = *A(\theta{\uparrow}FVar(*A))$,*
- *If $E = L_1, \ldots, L_n$, then $E\theta = L_1\theta, \ldots, L_n\theta$.*
- *If $E = L \leftarrow G$, then $E\theta = L\theta \leftarrow G\theta$,*
- *If $E = \{E_1, \ldots, E_n\}$, then $E\theta = \{E_1\theta, \ldots, E_n\theta\}$.*

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 or quantifiers. 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 other unifier γ of E_1 and E_2 we have $\theta \leq \gamma$.

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

Proof. Suppose that $\mathcal{I} \models \forall x_1{:}\, s_1, \ldots, x_n{:}\, s_n F$ where $\mathcal{I} = (M, \alpha)$ and $FVar(F) = \{x_1{:}\, s_1, \ldots, x_n{:}\, s_n\}$. By Definition 3.13, for all $a_1 \in I(s_1), \ldots, a_n \in I(s_n)$, $\mathcal{I}[a_1/x_1{:}\, s_1, \ldots, a_n/x_n{:}\, s_n] \models F$. Let θ be a sorted substitution. Then we obtain $F\theta$ with $FVar(F\theta) = \{y_1{:}\, s'_1, \ldots, y_k{:}\, s'_k\}$. Let $d_i = [\![(x_i{:}\, s_i)\theta]\!]_{\alpha[b_1/y_1:s'_1,\ldots,b_k/y_k:s'_k]}$ for $b_1 \in I(s'_1), \ldots, b_k \in I(s'_k)$. $\mathcal{I}[d_1/x_1{:}\, s_1, \ldots, d_n/x_n{:}\, s_n] \models F$ iff $\mathcal{I}[b_1/y_1{:}\, s'_1, \ldots, b_k/y_k{:}\, s_k] \models F\theta$. Therefore $\mathcal{I} \models \forall(F\theta)$.

If $\mathcal{I} \models L \leftarrow G$ with $G = L_1, \ldots, L_n$, then $\mathcal{I} \models \forall(L_1 \wedge \ldots \wedge L_n \rightarrow L)$. Therefore, by the above proof for $\forall F$, we have $\mathcal{I} \models \forall(L_1 \wedge \ldots \wedge L_n \rightarrow L)\theta$. ∎

3.3.3 Argument supplementation

We will introduce new inference rules of predicate-hierarchy into a Horn clause calculus for our extended order-sorted logic. First we define an argument supplementation that translates a sequence of arguments to the sequence of arguments for a predicate p. This translation consists of the addition and deletion of arguments based on the argument structure of p.

Let $\Sigma = (\mathcal{S}, \mathcal{F}, \mathcal{P}, \mathcal{D})$ be a sorted signature with hierarchical predicates, $\varphi \in \mathcal{P}^{\bullet} \cup \mathcal{P}^{\sharp}$, $a_1, \ldots, a_n \in AL$, and $t_1 \in TERM_{SCP(a_1)}, \ldots, t_n \in TERM_{SCP(a_n)}$.

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

is said to be an ill-argumented formula if $ARG([\varphi]) \neq \{a_1, \ldots, a_n\}$. All formulas in $FORM$ are said to be well-argumented formulas, and $ATOM (\subset FORM)$ denotes the set of all (well-argumented) atomic formulas. $\Delta(AL)$ is the family of all finite subsets of AL.

$$ATOM^*_{\varphi} = \{\varphi(a_1 \Rightarrow t_1, \ldots, a_n \Rightarrow t_n) \mid \{a_1, \ldots, a_n\} \in \Delta(AL), t_i \in TERM_{s_{a_i}}\}$$

where $s_{a_i} = SCP(a_i)$ denotes the set of well- and ill-argumented atoms with φ. We denote by $ATOM^* = \bigcup_{\varphi \in \mathcal{P}^{\bullet} \cup \mathcal{P}^{\sharp}} ATOM^*_{\varphi}$ the set of all well- and ill-argumented atoms. Note that the ill-argumented atoms are obtained by the argument structures in $\Delta(AL)$.

Example 3.8 *Let $t_1, t_2, \ldots, t_n$ be order-sorted terms. If $ARG(p) = \{a_1, a_2\}$, then the following expressions*

$$p^{\sharp}(a_1 \Rightarrow t_1),$$
$$p^{\sharp}(a_1 \Rightarrow t_1, a_3 \Rightarrow t_3),$$
$$p^{\sharp}(a_1 \Rightarrow t_1, a_3 \Rightarrow t_3, a_4 \Rightarrow t_4),$$
$$\ldots$$
$$p^{\sharp}(a_1 \Rightarrow t_1, a_3 \Rightarrow t_3, \ldots, a_n \Rightarrow t_n)$$

are ill-argumented atoms, but

$$p^{\sharp}(a_1 \Rightarrow t_1, a_2 \Rightarrow t_2)$$

is a well-argumented atom.

We introduce argument supplementation for ill-argumented atoms in the following definition.

Definition 3.18 (Argument supplementation) *Let A be a well- or ill-argumented atom. The argument supplementation σ is a function from $ATOM^*$ 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 $DEL^n(A) = DEL^{n+1}(A)(n > 0)$.[5]

(i) The addition ADD of an argument is defined by

$$ADD(A) = \begin{cases} p^\bullet(\mu \cup \{a \Rightarrow c_a\colon s\}) & \text{if } A = p^\bullet(\bar{\mu}) \text{ and } ARG(p) - LS(\bar{\mu}) \neq \emptyset, \\ p^\sharp(\mu \cup \{a \Rightarrow v\colon s\}) & \text{if } A = p^\sharp(\bar{\mu}) \text{ and } ARG(p) - LS(\bar{\mu}) \neq \emptyset, \\ A & \text{otherwise}, \end{cases}$$

where $a \in ARG(p) - LS(\bar{\mu})$ with $SCP(a) = s$, c_a is the supplement constant of argument label a, and $v\colon s$ is a new variable of sort s.

(ii) The deletion DEL of an argument is defined by

$$DEL(A) = \begin{cases} p^\bullet(\bar{\mu}) & \text{if } A = p^\bullet(\mu \cup \{a \Rightarrow t\}) \text{ and } a \notin ARG(p), \\ p^\sharp(\bar{\mu}) & \text{if } A = p^\sharp(\mu \cup \{a \Rightarrow x\colon s\}) \text{ and } a \notin ARG(p), \\ A & \text{otherwise}, \end{cases}$$

where $SCP(a) = s$ and $t \in TERM_s$.

The following example shows an application of the argument supplementation.

Example 3.9 *Let $\Sigma = (\mathcal{S}, \mathcal{F}, \mathcal{P}, \mathcal{D})$ where $\mathcal{S} = \{s_1, s_2, s_3, s_4\}$, $\mathcal{F} = \{c\}$, $\mathcal{P} = \{p\}$, and*

$$\mathcal{D} = (\emptyset, \{c\colon \rightarrow s_2\}, \{p\colon \{(l_1, s_1), (l_4, s_4)\}\}).$$

The argument supplementation σ is applied to the atom $p^\bullet(l_1 \Rightarrow x_1\colon s_1, l_2 \Rightarrow x_2\colon s_2, l_3 \Rightarrow x_3\colon s_3)$ with event predicate $p^\bullet$ as follows:

$$\begin{aligned} &\sigma(p^\bullet(l_1 \Rightarrow x_1\colon s_1, l_2 \Rightarrow c\colon s_2, l_3 \Rightarrow x_3\colon s_3)) \\ &\quad = ADD(DEL(DEL(p^\bullet(l_1 \Rightarrow x_1\colon s_1, l_2 \Rightarrow c\colon s_2, l_3 \Rightarrow x_3\colon s_3)))) \\ &\quad = ADD(DEL(p^\bullet(l_1 \Rightarrow x_1\colon s_1, l_2 \Rightarrow c\colon s_2))) \\ &\quad = ADD(p^\bullet(l_1 \Rightarrow x_1\colon s_1)) \\ &\quad = p^\bullet(l_1 \Rightarrow x_1\colon s_1, l_4 \Rightarrow c_{l_4}\colon s_4). \end{aligned}$$

Furthermore, the argument supplementation σ is applied to the atom $p^\sharp(l_2 \Rightarrow x_2\colon s_2)$ with property predicate $p^\sharp$ as follows:

$$\begin{aligned} &\sigma(p^\sharp(l_2 \Rightarrow x_2\colon s_2)) \\ &\quad = ADD(ADD(DEL(p^\sharp(l_2 \Rightarrow x_2\colon s_2)))) \\ &\quad = ADD(ADD(p^\sharp(\ ))) \\ &\quad = ADD(p^\sharp(l_1 \Rightarrow v\colon s_1)) \\ &\quad = p^\sharp(l_1 \Rightarrow v\colon s_1, l_4 \Rightarrow v'\colon s_4). \end{aligned}$$

[5] Let f be a function. We write f^n for the composite n functions $f \circ f \circ \cdots \circ f$.

The following lemma indicates that the argument supplementation σ (as a translation of an ill-argumented atom to the well-argumented atom) corresponds to the event and property interpretations $\iota^\bullet, \iota^\sharp$ for the argument supplementation in semantics. $p \sqsubset_P q$ implies $p^\bullet(\bar{\mu}) \models_{H\Sigma} q^\bullet(\bar{\mu}')$ and $p^\sharp(\bar{\mu}) \models_{H\Sigma} q^\sharp(\bar{\mu}')$ where $\bar{\mu}'$ is obtained by adjusting $\bar{\mu}$ to the argument structure of q, i.e., $q^\bullet(\bar{\mu}') = \sigma(q^\bullet(\bar{\mu}))$ and $q^\sharp(\bar{\mu}') = \sigma(q^\sharp(\bar{\mu}))$. Let μ be a set $\{a_1 \Rightarrow t_1, \ldots, a_n \Rightarrow t_n\}$ of arguments and $\varphi \in \mathcal{P}^\bullet \cup \mathcal{P}^\sharp$. We define the restriction of μ to the argument labels of φ by

$$\mu_\varphi = \{a \Rightarrow t \in \mu \mid a \in ARG([\varphi])\}.$$

Lemma 3.2 *Let $\Sigma = (\mathcal{S}, \mathcal{F}, \mathcal{P}, \mathcal{D})$ be a sorted signature with hierarchical predicates, σ the argument supplementation, and $p, p_1, \ldots, p_n, q \in \mathcal{P}$. Then, the following holds:*

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

(a) $p^\bullet(\bar{\mu}) \models_{H\Sigma} \sigma(q^\bullet(\bar{\mu}))$ and
(b) $p^\sharp(\bar{\mu}) \models_{H\Sigma} \forall(\sigma(q^\sharp(\bar{\mu})))$

where $\mu = \{a_1 \Rightarrow t_1, \ldots, a_n \Rightarrow t_n\}$ such that $ARG(p) = \{a_1, \ldots, a_n\}$ and $t_i \in TERM_{0,SCP(a_i)}$.

(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 $q \sqsubset_P^1 p_i$, then

(a) $\{p_1^\bullet(\bar{\mu}_{p_1^\bullet}), \ldots, p_n^\bullet(\bar{\mu}_{p_n^\bullet})\} \models_{H\Sigma} \sigma(q^\bullet(\bar{\mu}))$ and
(b) $\{p_1^\sharp(\bar{\mu}_{p_1^\sharp}), \ldots, p_n^\sharp(\bar{\mu}_{p_n^\sharp})\} \models_{H\Sigma} \forall(\sigma(q^\sharp(\bar{\mu})))$

where $\mu = \{a_1 \Rightarrow t_1, \ldots, a_m \Rightarrow t_m\}$ such that $\bigcup_{1 \le i \le n} ARG(p_i) = \{a_1, \ldots, a_m\}$ and $t_i \in TERM_{0,SCP(a_i)}$.

Proof.
(1) (a) Suppose that $\mathcal{I} \models p^\bullet(\bar{\mu})$ where $\mathcal{I} = (M, \alpha)$ is a $H\Sigma$-interpretation and $\mu = \{a_1 \Rightarrow t_1, \ldots, a_n \Rightarrow t_n\}$, i.e., $\{(a_1, [\![t_1]\!]_\alpha), \ldots, (a_n, [\![t_n]\!]_\alpha)\} \in I(p^\bullet)$. By Definition 3.18,

$$\sigma(q^\bullet(\bar{\mu})) = q^\bullet(b_1 \Rightarrow t_{b1}, \ldots, b_m \Rightarrow t_{bm}, e_1 \Rightarrow c_{e_1}\!: SCP(e_1), \ldots, e_k \Rightarrow c_{e_k}\!: SCP(e_k)),$$

where $ARG(p) \cap ARG(q) = \{b_1, \ldots, b_m\}$, for $1 \le i \le m$, $t_{bi} = t_j$ with $b_i = a_j$, and $ARG(q - p) = \{e_1, \ldots, e_k\}$. Let

$$\begin{aligned} \iota_q^\bullet(I(p^\bullet)) \;=\; & \{(\rho \cap ARG(q) \times U) \cup \\ & \{(e_1, [\![c_{e_1}\!: SCP(e_1)]\!]_\alpha), \ldots, (e_k, [\![c_{e_k}\!: SCP(e_k)]\!]_\alpha)\} \mid \rho \in I(p^\bullet)\}. \end{aligned}$$

Then, by Definition 3.9 and $\mathcal{I} \models p^\bullet(\bar{\mu})$,

$$\begin{aligned} & \{(b_1, [\![t_{b1}]\!]_\alpha, \ldots, (b_m, [\![t_{bm}]\!]_\alpha)\} \cup \\ & \{(e_1, [\![c_{e_1}\!: SCP(e_1)]\!]_\alpha), \ldots, (e_k, [\![c_{e_k}\!: SCP(e_k)]\!]_\alpha)\} \in \iota_q^\bullet(I(p^\bullet))(\subseteq I(q^\bullet)). \end{aligned}$$

Therefore $\mathcal{I} \models \sigma(q^\bullet(\bar{\mu}))$.

(b) Suppose that $\mathcal{I} \models p^\sharp(\bar{\mu})$ where $\mathcal{I} = (M, \alpha)$ is an $H\Sigma$-interpretation and $\mu = \{a_1 \Rightarrow t_1, \ldots, a_n \Rightarrow t_n\}$, i.e., $\{(a_1, [\![t_1]\!]_\alpha), \ldots, (a_n, [\![t_n]\!]_\alpha)\} \in I(p^\sharp)$. By Definition 3.18,

$$\sigma(q^\sharp(\bar{\mu})) = q^\sharp(b_1 \Rightarrow t_{b1}, \ldots, b_m \Rightarrow t_{bm}, e_1 \Rightarrow v_1\colon SCP(e_1), \ldots, e_k \Rightarrow v_k\colon SCP(e_k)),$$

where $ARG(p) \cap ARG(q) = \{b_1, \ldots, b_m\}$, *for* $1 \leq i \leq m$, $t_{bi} = t_j$ *with* $b_i = a_j$, *and* $ARG(q - p) = \{e_1, \ldots, e_k\}$. *Let*

$$\iota^\sharp_q(I(p^\sharp)) = \{(\rho \cap ARG(q) \times U) \cup \{(e_1, d_1), \ldots, (e_k, d_k)\} \mid \rho \in I(p^\sharp), d_i \in I(SCP(e_i))\}.$$

Then, by Definition 3.9 and $\mathcal{I} \models p^\sharp(\bar{\mu})$,

$$\{(b_1, [\![t_{b1}]\!]_{\alpha'}), \ldots, (b_m, [\![t_{bm}]\!]_{\alpha'}), (e_1, [\![v_1\colon s_1]\!]_{\alpha'}), \ldots, (e_k, [\![v_k\colon s_k]\!]_{\alpha'})\} \in \iota^\sharp_q(I(p^\sharp)) \subseteq I(q^\sharp)$$

for all $d_1 \in I(SCP(e_1)), \ldots, d_k \in I(SCP(e_k))$ *where* $s_1 = SCP(e_1), \ldots, s_k = SCP(e_k)$ *and* $\alpha' = \alpha[d_1/v_1\colon s_1, \ldots, d_k/v_k\colon s_k]$. *Therefore, we have* $\mathcal{I} \models \forall v_1\colon s_1 \cdots v_k\colon s_k(\sigma(q^\sharp(\bar{\mu})))$ *with* $FVar(\sigma(q^\sharp(\bar{\mu}))) = \{v_1\colon s_1, \ldots, v_k\colon s_k\}$.

(2) (a) Suppose that, for $1 \leq i \leq n$, $\mathcal{I} \models p^\bullet_i(\bar{\mu}_{p^\bullet_i})$ *where* $\mathcal{I} = (M, \alpha)$ *is an* $H\Sigma$*-interpretation and* $\mu = \{a_1 \Rightarrow t_1, \ldots, a_m \Rightarrow t_m\}$ *such that* $\bigcup_{1 \leq i \leq n} ARG(p_i) = \{a_1, \ldots, a_m\}$ *and* $t_i \in TERM_{0,SCP(a_i)}$, *i.e.,* $\{(a, [\![t]\!]_\alpha) \mid a \Rightarrow t \in \bar{\mu}_{p^\bullet_i}\} \in I(p^\bullet_i)$. *By Definition 3.18,*

$$\sigma(q^\bullet(\bar{\mu})) = q^\bullet(b_1 \Rightarrow t_{b1}, \ldots, b_u \Rightarrow t_{bu}, e_1 \Rightarrow c_{e_1}\colon SCP(e_1), \ldots, e_k \Rightarrow c_{e_k}\colon SCP(e_k)),$$

where $\{a_1, \ldots, a_m\} \cap ARG(q) = \{b_1, \ldots, b_u\}$, *for* $1 \leq l \leq u$, $t_{bl} = t_j$ *with* $b_l = a_j$, *and* $\{e_1, \ldots, e_k\} = ARG(q) - \{a_1, \ldots, a_m\}$. *By Definition 3.10,* $\sqcap_{p^\bullet_i \in \Pi} I(p^\bullet_i)$ *with* $\Pi = \{p^\bullet_1, \ldots, p^\bullet_n\}$ *includes* $\{(a_1, [\![t_1]\!]_\alpha), \ldots, (a_m, [\![t_m]\!]_\alpha)\}$. *Let*

$$\iota^\bullet_q(\sqcap_{p^\bullet_i \in \Pi} I(p^\bullet_i)) = \{(\rho \cap ARG(q) \times U) \cup \{(e_1, d_1), \ldots, (e_k, d_k)\} \mid \rho \in \sqcap_{p^\bullet_i \in \Pi} I(p^\bullet_i)\}$$

where $d_1 = [\![c_{e_1}\colon SCP(e_1)]\!]_\alpha, \ldots, d_n = [\![c_{e_k}\colon SCP(e_k)]\!]_\alpha$.
Then, by Definition 3.9 and $\mathcal{I} \models p^\bullet_1(\bar{\mu}_{p^\bullet_1}), \ldots, \mathcal{I} \models p^\bullet_n(\bar{\mu}_{p^\bullet_n})$,

$$\{(b_1, [\![t_{b1}]\!]_\alpha, \ldots, (b_u, [\![t_{bu}]\!]_\alpha), (e_1, d_1), \ldots, (e_k, d_k)\} \in \iota^\bullet_q(\sqcap_{p^\bullet_i \in \Pi} I(p^\bullet_i)) \subseteq I(q^\bullet)$$

where $d_1 = [\![c_{e_1}\colon SCP(e_1)]\!]_\alpha, \ldots, d_n = [\![c_{e_k}\colon SCP(e_k)]\!]_\alpha$. *Therefore* $\mathcal{I} \models \sigma(q^\bullet(\bar{\mu}))$.

(b) Suppose that $\mathcal{I} \models p^\sharp_1(\bar{\mu}_{p^\sharp_1}), \ldots, \mathcal{I} \models p^\sharp_n(\bar{\mu}_{p^\sharp_n})$ *where* $\mathcal{I} = (M, \alpha)$ *is an* $H\Sigma$*-interpretation and* $\mu = \{a_1 \Rightarrow t_1, \ldots, a_m \Rightarrow t_m\}$ *such that* $\bigcup_{1 \leq i \leq n} ARG(p_i) = \{a_1, \ldots, a_m\}$ *and* $t_i \in TERM_{0,SCP(a_i)}$, *i.e.,* $\{(a, [\![t]\!]_\alpha) \mid a \Rightarrow t \in \bar{\mu}_{p^\sharp_i}\} \in I(p^\sharp_i)$. *By Definition 3.18,*

$$\sigma(q^\sharp(\bar{\mu})) = q^\sharp(b_1 \Rightarrow t_{b1}, \ldots, b_u \Rightarrow t_{bu}, e_1 \Rightarrow v_1\colon SCP(e_1), \ldots, e_k \Rightarrow v_k\colon SCP(e_k)),$$

where $\{a_1, \ldots, a_m\} \cap ARG(q) = \{b_1, \ldots, b_u\}$, *for* $1 \leq l \leq u$, $t_{bl} = t_j$ *with* $b_l = a_j$, *and* $ARG(q) - \{a_1, \ldots, a_m\} = \{e_1, \ldots, e_k\}$. *By Definition 3.10,* $\sqcap_{p^\sharp_i \in \Pi} I(p^\sharp_i)$ *with* $\Pi = \{p^\sharp_1, \ldots, p^\sharp_n\}$ *includes* $\{(a_1, [\![t_1]\!]_\alpha), \ldots, (a_m, [\![t_m]\!]_\alpha)\}$. *Let*

$$\begin{aligned}\iota^\bullet_q(\sqcap_{p^\sharp_i \in \Pi} I(p^\sharp_i)) \;=\; & \{(\rho \cap ARG(q) \times U) \cup \{(e_1, d_1), \ldots, (e_k, d_k)\} \mid \\ & \rho \in \sqcap_{p^\sharp_i \in \Pi} I(p^\sharp_i), d_i \in I(SCP(e_i))\}.\end{aligned}$$

Then, by Definition 3.9 and $\mathcal{I} \models p^\sharp_1(\bar{\mu}_{p^\sharp_1}), \ldots, \mathcal{I} \models p^\sharp_n(\bar{\mu}_{p^\sharp_n})$,

$$\begin{aligned}&\{(b_1, [\![t_{b1}]\!]_{\alpha'}, \ldots, (b_u, [\![t_{bu}]\!]_{\alpha'})\} \cup \\ &\{(e_1, [\![v_1\colon s_1]\!]_{\alpha'}), \ldots, (e_k, [\![v_k\colon s_k]\!]_{\alpha'})\} \in \iota^\bullet_q(\sqcap_{p^\sharp_i \in \Pi} I(p^\sharp_i)) \subseteq I(q^\sharp)\end{aligned}$$

for all $d_1 \in I(SCP(e_1)), \ldots, d_k \in I(SCP(e_k))$ *where* $s_1 = SCP(e_1), \ldots, s_k = SCP(e_k)$ *and* $\alpha' = \alpha[d_1/v_1\colon s_1, \ldots, d_k/v_k\colon s_k]$. *Therefore, we have*

$$\mathcal{I} \models \forall v_1 : s_1 \cdots v_k : s_k (\sigma(q^\sharp(\bar{\mu})))$$

with $FVar(\sigma(q^\sharp(\bar{\mu}))) = \{v_1 : s_1, \ldots, v_k : s_k\}$. ∎

3.3.4 Horn clause calculus

We present a Horn clause calculus for our order-sorted logic with hierarchies and eventuality. The Horn clause calculus contains the following inference rules.

Definition 3.19 (Substitution rule) *Let $L \leftarrow G$ be a clause.*

$$\frac{}{(L \leftarrow G)\theta}$$

where θ is a ground substitution for $L \leftarrow G$.

Definition 3.20 (Cut rule) *Let $L_1 \leftarrow G_1 \cup \{L\}$ and $L \leftarrow G_2$ be ground clauses.*

$$\frac{L_1 \leftarrow G_1 \cup \{L\} \quad L \leftarrow G_2}{L_1 \leftarrow G_1 \cup G_2}$$

Definition 3.21 (Generalization rule for event predicates) *Let $p^\bullet(\bar{\mu}) \leftarrow G$ be a ground clause and σ the argument supplementation.*

$$\frac{p^\bullet(\bar{\mu}) \leftarrow G}{\sigma(q^\bullet(\bar{\mu})) \leftarrow G}$$

where $p \sqsubset_P q$.

Definition 3.22 (Generalization rule for property predicates) *Let $p^\sharp(\bar{\mu}) \leftarrow G$ be a ground clause and σ the argument supplementation.*

$$\frac{p^\sharp(\bar{\mu}) \leftarrow G}{(\sigma(q^\sharp(\bar{\mu})) \leftarrow G)\theta}$$

where $p \sqsubset_P q$ and θ is a ground substitution for $\sigma(q^\sharp(\bar{\mu})) \leftarrow G$.

Definition 3.23 (Specialization rule for event predicates) *Let $p_1^\bullet(\bar{\mu}_{p_1^\bullet}) \leftarrow G$, ..., $p_n^\bullet(\bar{\mu}_{p_n^\bullet}) \leftarrow G$ be ground clauses and σ the argument supplementation. If $p_1, \ldots, p_n (n > 1)$ are all predicates such that $q \sqsubset_P^1 p_i$, then*

$$\frac{p_1^\bullet(\bar{\mu}_{p_1^\bullet}) \leftarrow G_1 \quad \ldots \quad p_n^\bullet(\bar{\mu}_{p_n^\bullet}) \leftarrow G_n}{\sigma(q^\bullet(\bar{\mu})) \leftarrow G_1 \cup \ldots \cup G_n}$$

where $\mu = \{a_1 \Rightarrow t_1, \ldots, a_m \Rightarrow t_m\}$ such that $\bigcup_{1 \le i \le n} ARG(p_i) = \{a_1, \ldots, a_m\}$ and $t_i \in TERM_{0,SCP(a_i)}$.

Definition 3.24 (Specialization rule for property predicates) *Let $p_1^\sharp(\bar{\mu}_{p_1^\sharp}) \leftarrow G$, ..., $p_n^\sharp(\bar{\mu}_{p_n^\sharp}) \leftarrow G$ be ground clauses and σ the argument supplementation. If $p_1, \ldots, p_n (n > 1)$ are all predicates such that $q \sqsubset_P^1 p_i$, then*

$$\frac{p_1^\sharp(\bar{\mu}_{p_1^\sharp}) \leftarrow G_1 \quad \ldots \quad p_n^\sharp(\bar{\mu}_{p_n^\sharp}) \leftarrow G_n}{(\sigma(q^\sharp(\bar{\mu})) \leftarrow G_1 \cup \ldots \cup G_n)\theta}$$

where θ is a ground substitution for $\sigma(q^\sharp(\bar{\mu})) \leftarrow G_1 \cup \ldots \cup G_n$ and $\mu = \{a_1 \Rightarrow t_1, \ldots, a_m \Rightarrow t_m\}$ such that $\bigcup_{1 \le i \le n} ARG(p_i) = \{a_1, \ldots, a_m\}$ and $t_i \in TERM_{0,SCP(a_i)}$.

In the next two examples, we consider an application of the generalization rule for event predicates and the specialization rule for property predicates.

Example 3.10 *Let $\Sigma = (\mathcal{S}, \mathcal{F}, \mathcal{P}, \mathcal{D})$ where*

$$\begin{aligned} \mathcal{S} &= \{s, s_1, s_2, s_3\}, \\ \mathcal{F} &= \emptyset, \\ \mathcal{P} &= \{p_1, p_2, q\}, \end{aligned}$$

and $\mathcal{D} = (\mathcal{D}_\mathcal{S}, \mathcal{D}_\mathcal{F}, \mathcal{D}_\mathcal{P})$ with $\mathcal{D}_\mathcal{S} = \emptyset$, $\mathcal{D}_\mathcal{F} = \emptyset$, and

$$\begin{aligned} \mathcal{D}_\mathcal{P} &= \{p_1 \sqsubset_P p_2\} \cup \\ &\quad \{p_1\colon\{(l_1, s_1), (l_2, s_2)\}, p_2\colon\{(l_1, s_1), (l_3, s_3)\}, q\colon\{(l, s)\}\}. \end{aligned}$$

The generalization rule for event predicates is applied as follows:

$$\frac{p_1^\bullet(l_1 \Rightarrow t_1, l_2 \Rightarrow t_2) \leftarrow q^\bullet(l \Rightarrow t)}{p_2^\bullet(l_1 \Rightarrow t_1, l_3 \Rightarrow c_{l_3}\colon s_3) \leftarrow q^\bullet(l \Rightarrow t)}$$

by $p_1 \sqsubset_P p_2$ where

$$\sigma(p_2^\bullet(l_1 \Rightarrow t_1, l_2 \Rightarrow t_2)) = p_2^\bullet(l_1 \Rightarrow t_1, l_3 \Rightarrow c_{l_3}\colon s_3).$$

A superpredicate p_2 is derived from the predicate p_1 with the argument supplementation σ.

Example 3.11 *Let $\Sigma = (\mathcal{S}, \mathcal{F}, \mathcal{P}, \mathcal{D})$ where*

$$\begin{aligned} \mathcal{S} &= \{s, s_1, s_2, s_3\}, \\ \mathcal{F} &= \emptyset, \\ \mathcal{P} &= \{p_1, p_2, q, r\}, \end{aligned}$$

and $\mathcal{D} = (\mathcal{D}_\mathcal{S}, \mathcal{D}_\mathcal{F}, \mathcal{D}_\mathcal{P})$ with $\mathcal{D}_\mathcal{S} = \emptyset$, $\mathcal{D}_\mathcal{F} = \emptyset$, and

$$\begin{aligned} \mathcal{D}_\mathcal{P} &= \{q \sqsubset_P p_1, q \sqsubset_P p_2\} \cup \\ &\quad \{p_1\colon\{(l_1, s_1)\}, p_2\colon\{(l_1, s_1), (l_3, s_3)\}, q\colon\{(l_1, s_1), (l_2, s_2), (l_3, s_3)\}, r\colon\{(l, s)\}\}. \end{aligned}$$

The specialization rule for property predicates is applied as follows:

$$\frac{p_1^\sharp(l_1 \Rightarrow t_1) \quad p_2^\sharp(l_1 \Rightarrow t_1, l_3 \Rightarrow t_3) \leftarrow r^\sharp(l \Rightarrow t)}{(q^\sharp(l_1 \Rightarrow t_1, l_2 \Rightarrow v\colon s_2, l_3 \Rightarrow t_3) \leftarrow r^\sharp(l \Rightarrow t))\theta}$$

by $q \sqsubset_P p_1$ and $q \sqsubset_P p_2$ where $\theta(v\colon s_2) \in TERM_{s_2}$ and

$$\sigma(q^\sharp(l_1 \Rightarrow t_1, l_3 \Rightarrow t_3)) = q^\sharp(l_1 \Rightarrow t_1, l_2 \Rightarrow v\colon s_2, l_3 \Rightarrow t_3).$$

A subpredicate q is derived from the predicates p_1, p_2 with the argument supplementation σ.

The next theorem guarantees that each rule in the Horn clause calculus preserves validity.

Theorem 3.1 *Let $C, C_1, \ldots, C_n$ be clauses. The conclusion C of each inference rule in the Horn clause calculus is a consequence of its premise $\{C_1, \ldots, C_n\}$ in the class of $H\Sigma$-structures. That is, $\{C_1, \ldots, C_n\} \models_{H\Sigma} C$.*

Proof. For each inference rule we show $\{C_1, \ldots, C_n\} \models_{H\Sigma} C$.

1. Substitution rule. By Lemma 3.1, this is easy to show.

2. Cut rule. Suppose that $\mathcal{I} \models L_1 \leftarrow G_1 \cup \{L\}$ and $\mathcal{I} \models L \leftarrow G_2$ where $\mathcal{I}$ is an $H\Sigma$-interpretation and $L_1 \leftarrow G_1 \cup \{L\}$ and $L \leftarrow G_2$ are ground clauses. Then $\mathcal{I} \models L_{G_1} \wedge L \rightarrow L_1$ and $\mathcal{I} \models L_{G_2} \rightarrow L$ where $L_{G_i} = \wedge_{L_i \in G_i} L_i$. If $\mathcal{I} \models L_{G_1} \wedge L$, then $\mathcal{I} \models L_1$. If $\mathcal{I} \not\models L_{G_1} \wedge L$, then $\mathcal{I} \not\models L_{G_1}$ or $\mathcal{I} \not\models L_{G_2}$. Hence $\mathcal{I} \not\models L_{G_1} \wedge L_{G_2}$. Therefore we have $\mathcal{I} \models L_{G_1} \wedge L_{G_2} \rightarrow L_1$.

3. Generalization rule for event predicates. Suppose that $\mathcal{I} \models G$ where $\mathcal{I}$ is an $H\Sigma$-interpretation. By the hypothesis, $\mathcal{I} \models p^\bullet(\bar{\mu})$. Then, by Lemma 3.2, $\mathcal{I} \models \sigma(q^\bullet(\bar{\mu}))$. Therefore, $\mathcal{I} \models \sigma(q^\bullet(\bar{\mu})) \leftarrow G$.

4. Generalization rule for property predicates. Suppose that $\mathcal{I} \models G$ where $\mathcal{I}$ is an $H\Sigma$-interpretation. By the hypothesis, $\mathcal{I} \models p^\sharp(\bar{\mu})$. Then, by Lemma 3.2, $\mathcal{I} \models \forall(\sigma(q^\sharp(\bar{\mu})))$, and so, by Lemma 3.1, we obtain $\mathcal{I} \models (\sigma(q^\sharp(\bar{\mu})))\theta$ where θ is a ground substitution for $\sigma(q^\sharp(\bar{\mu}))$. Therefore $\mathcal{I} \models (\sigma(q^\sharp(\bar{\mu})) \leftarrow G)\theta$ since G is ground.

5. Specialization rule for event predicates. Suppose that $\mathcal{I} \models L_{G_1} \wedge \ldots \wedge L_{G_n}$ where $\mathcal{I}$ is an $H\Sigma$-interpretation. By the hypothesis, $\mathcal{I} \models p_1^\bullet(\bar{\mu}_{p_1^\bullet}), \ldots, \mathcal{I} \models p_n^\bullet(\bar{\mu}_{p_n^\bullet})$. Then, by Lemma 3.2, $\mathcal{I} \models \sigma(q^\bullet(\bar{\mu}))$. Hence, $\mathcal{I} \models L_{G_1} \wedge \ldots \wedge L_{G_n} \rightarrow \sigma(q^\bullet(\bar{\mu}))$. Therefore, $\mathcal{I} \models \sigma(q^\bullet(\bar{\mu})) \leftarrow G_1 \cup \ldots \cup G_n$.

6. Specialization rule for property predicates. Suppose that $\mathcal{I} \models L_{G_1} \wedge \ldots \wedge L_{G_n}$ where $\mathcal{I}$ is an $H\Sigma$-interpretation. By the hypothesis, $\mathcal{I} \models p_1^\sharp(\bar{\mu}_{p_1^\sharp}), \ldots, \mathcal{I} \models p_n^\sharp(\bar{\mu}_{p_n^\sharp})$. Then, by Lemma 3.2, $\mathcal{I} \models \forall(\sigma(q^\sharp(\bar{\mu}))$, and so, by Lemma 3.1, we obtain $\mathcal{I} \models (\sigma(q^\sharp(\bar{\mu})))\theta$ where θ is a ground substitution for $\sigma(q^\sharp(\bar{\mu}))$. Hence, $\mathcal{I} \models L_{G_1} \wedge \ldots \wedge L_{G_n} \rightarrow (\sigma(q^\sharp(\bar{\mu})))\theta$. Therefore, $\mathcal{I} \models (\sigma(q^\sharp(\bar{\mu})) \leftarrow G_1 \cup \ldots \cup G_n)\theta$ since $G_1, \ldots, G_n$ are ground. ∎

We define the notion of derivation from an application of the rules in the Horn clause calculus as follows.

Definition 3.25 (Derivation) *Let $P = (\Sigma, CS)$ be a program and C, C', C_i clauses. The derivability relation $P \vdash C$ is defined by:*

(i) If $C \in CS$, then $P \vdash C\theta$ such that $C\theta$ is a conclusion of the substitution rule.

(ii) *If $P \vdash C_1$ and $P \vdash C_2$ such that C_1 and C_2 are premises of the cut rule, then $P \vdash C$ such that C is its conclusion.*

(iii) *If $P \vdash C$ such that C is a premise of the generalization rule for event or property predicates, then $P \vdash C'$ such that C' is its conclusion.*

(iv) *If $P \vdash C_1, \ldots, P \vdash C_n$ such that $C_1, \ldots, C_n$ are premises of the specialization rule for event or property predicates, then $P \vdash C$ such that C is its conclusion.*

Let $P = (\Sigma, CS)$ be a program. We say that $L \leftarrow G$ is derivable from P if $P \vdash L \leftarrow G$. We use the abbreviation $P \vdash L$ to denote $P \vdash L \leftarrow$. We write $P \models_{H\Sigma} C$ if $CS \models_{H\Sigma} C$ in the sorted signature Σ. The soundness of the Horn clause calculus is proved as follows.

Theorem 3.2 (Soundness of derivation) *Let P be a program and L an atom. If $P \vdash L$, then $P \models_{H\Sigma} L$.*

Proof. This is proved from Definition 4.7 and Theorem 3.1 by induction on the length of a derivation $P \vdash L$. ∎

We introduce the notion of a Herbrand model that is a Σ-model.

Definition 3.26 *A Herbrand Σ-structure $M_H = (I_H, U_H)$ is a Σ-structure such that*

(i) $U_H = TERM_0$,

(ii) $I(s) = TERM_{0,s} (\subseteq U_H)$ *where* $s \in \mathcal{S}$,

(iii) $I_H(c) = c\colon s$ *where* $c \in F$ *and* $c\colon \rightarrow s \in \mathcal{D}_{\mathcal{F}}$,

(iv) $I_H(f)(t_1, \ldots, t_n) = f(t_1 \ldots, t_n)\colon s$ *where* $f \in F$ *and* $f\colon s_1 \times \ldots \times s_n \rightarrow s \in \mathcal{D}_{\mathcal{F}}$.

An interpretation $\mathcal{I}$ is said to be a Herbrand Σ-interpretation if its structure is a Herbrand Σ-structure. A Herbrand Σ-structure $\mathcal{I}$ is a Herbrand Σ-model of Γ if it is a model of Γ.

Lemma 3.3 *Let $\mathcal{I}_H$ be a Herbrand Σ-interpretation and $L \leftarrow G$ a clause. $\mathcal{I}_H \models L \leftarrow G$ iff $\mathcal{I}_H \models ground(L \leftarrow G)$*

Proof.
($\Rightarrow$) Assume that $\mathcal{I}_H \models L \leftarrow G$. By Lemma 3.1, for any ground substitution θ for $L \leftarrow G$, $\mathcal{I}_H \models (L \leftarrow G)\theta$ $(\in ground(L \leftarrow G))$.
($\Leftarrow$) Assume that $\mathcal{I}_H \models ground(L \leftarrow G)$ where $CVar(L \leftarrow G) = \{x_1\colon s_1, \ldots, x_n\colon s_n\}$. We show that, for all $d_1 \in I(s_1), \ldots, d_n \in I(s_n)$, $\mathcal{I}_H\{d_1/x_1\colon s_1, \ldots, d_n/x_n\colon s_n\} \models L \leftarrow G$. By the hypothesis, for any ground substitution θ for $L \leftarrow G$, $\mathcal{I}_H \models (L \leftarrow G)\theta$ $(\in ground(L \leftarrow G))$. Let $(x_i\colon s_i)\theta = d_i$ for $d_1 \in I(s_i), \ldots, d_n \in I(s_n)$ (where $d_i \in TERM_{0,s_i}$ by Definition 3.26). Then $\mathcal{I}_H \models (L \leftarrow G)\theta$ iff $\mathcal{I}_H\{d_1/x_1\colon s_1, \ldots, d_n/x_n\colon s_n\} \models L \leftarrow G$. Therefore, $\mathcal{I}_H \models L \leftarrow G$. ∎

To show the completeness of the Horn clause calculus, we build an interpretation $\mathcal{I}_P$ that satisfies all the atoms derivable from the program P.

Definition 3.27 *Let P be a program and L an atom. A deduction interpretation $\mathcal{I}_P$ of P is a Herbrand Σ-interpretation such that*

$$\mathcal{I}_P \models L \textit{ iff } P \vdash L.$$

The next lemma shows that the deduction interpretation $\mathcal{I}_P$ is an $H\Sigma$-model of P.

Lemma 3.4 *Let P be a program. A deduction interpretation $\mathcal{I}_P$ of P is an $H\Sigma$-model of P.*

Proof. In order to prove that $\mathcal{I}_P$ is a model of P, we have to show $\mathcal{I}_P \models L \leftarrow G$ for all clauses $L \leftarrow G$ in $P = (\Sigma, CS)$. Let $(L \leftarrow G) \in CS$ with $G = L_1, \ldots, L_n$ and θ be a ground substitution for $L \leftarrow G$. Suppose that $\mathcal{I}_P \models L_1\theta \wedge \ldots \wedge L_n\theta$. By the definition of $\mathcal{I}_P$, for $1 \leq i \leq n$, $P \vdash L_i\theta$. Then, we can get $P \vdash L_1\theta \wedge \ldots \wedge L_n\theta \rightarrow L\theta$ by the substitution rule and $P \vdash L\theta$ by the cut rule. By the definition of $\mathcal{I}_P$, $\mathcal{I}_P \models L\theta$. Hence, we have $\mathcal{I}_P \models (l \leftarrow G)\theta$. Since $\mathcal{I}_P$ is a Herbrand Σ-interpretation, by Lemma 3.3, $\mathcal{I}_P \models L \leftarrow G$ iff $\mathcal{I}_P \models \mathit{ground}(L \leftarrow G)$ iff $\mathcal{I}_P \models (L \leftarrow G)\theta$ for all ground substitutions θ for $L \leftarrow G$. Therefore, $\mathcal{I}_P \models L \leftarrow G$.

Next, we will show that $\mathcal{I}_P$ is an $H\Sigma$-interpretation. Let $\mathcal{I}_P = (M_H, \alpha)$ be a deduction interpretation of P. For $p \sqsubset q \in \mathcal{D}_\mathcal{P}$, we assume that

$$\{(a_1, t_1), \ldots, (a_n, t_n)\} \in I_H(p^\bullet)$$

where $t_i \in \mathit{TERM}_{0,SCP(a_i)}$. Then, by Definition 3.9,

$$(\rho \cap \mathit{ARG}(q) \times U_H) \cup \{(e_1, c_{e_1} : \mathit{SCP}(e_1)), \ldots, (e_k, c_{e_k} : \mathit{SCP}(e_k))\} \in \iota_q^\bullet(I_H(p^\bullet))$$

where $\rho = \{(a_1, t_1), \ldots, (a_n, t_n)\}$. Furthermore, by Definition 3.13, $\mathcal{I}_P \models p^\bullet(\bar{\mu})$ with $\mu = \{a_1 \Rightarrow t_1, \ldots, a_n \Rightarrow t_n\}$. We can infer $P \vdash p^\bullet(\bar{\mu})$ by the definition of $\mathcal{I}_P$. Then $P \vdash \sigma(q^\bullet(\bar{\mu}))$ by the generalization rule for event predicates and therefore $P \models \sigma(q^\bullet(\bar{\mu}))$ by the definition of $\mathcal{I}_P$. By Definition 3.18,

$$\sigma(q^\bullet(\bar{\mu})) = q^\bullet(b_1 \Rightarrow t_{b1}, \ldots, b_m \Rightarrow t_{bm}, e_1 \Rightarrow c_{e_1} : \mathit{SCP}(e_1), \ldots, e_k \Rightarrow c_{e_k} : \mathit{SCP}(e_k)),$$

where $\mathit{ARG}(p) \cap \mathit{ARG}(q) = \{b_1, \ldots, b_m\}$, for $1 \leq i \leq m$, $t_{bi} = t_j$ with $b_i = a_j$, and $\mathit{ARG}(q - p) = \{e_1, \ldots, e_k\}$. Hence,

$$\{(b_1, t_{b1}), \ldots, (b_m, t_{bm}), (e_1, c_{e_1} : \mathit{SCP}(e_1)), \ldots, (e_k, c_{e_k} : \mathit{SCP}(e_k))\} \in I_H(q^\bullet)$$

where $\{(b_1, t_{b1}), \ldots, (b_m, t_{bm})\} = \{(a_1, t_1), \ldots, (a_n, t_n)\} \cap (\mathit{ARG}(q) \times U_H)$.

In the case of property predicates, we assume that

$$\{(a_1, t_1), \ldots, (a_n, t_n)\} \in I_H(p^\sharp)$$

where $t_i \in \mathit{TERM}_{0,SCP(a_i)}$. Then, by Definition 3.9,

$$(\rho \cap \mathit{ARG}(q) \times U_H) \cup \{(e_1, d_1), \ldots, (e_k, d_k)\} \in \iota_q^\sharp(I_H(p^\sharp))$$

for all $d_1 \in \mathit{TERM}_{0,SCP(e_1)}$, $\ldots, d_n \in \mathit{TERM}_{0,SCP(e_n)}$ where $\rho = \{(a_1, t_1), \ldots, (a_n, t_n)\}$. Furthermore, by Definition 3.13, $\mathcal{I}_P \models p^\sharp(\bar{\mu})$ with $\mu = \{a_1 \Rightarrow t_1, \ldots, a_n \Rightarrow t_n\}$. We can infer $P \vdash p^\sharp(\bar{\mu})$ by the definition of $\mathcal{I}_P$. Then $P \vdash (\sigma(q^\sharp(\bar{\mu})))\theta$ where θ is any ground substitution for $\sigma(q^\sharp(\bar{\mu}))$ by the generalization rule for property predicates and therefore $P \models (\sigma(q^\sharp(\bar{\mu})))\theta$ by the definition of $\mathcal{I}_P$. By Definition 3.18,

$$\sigma(q^\sharp(\bar{\mu})) = q^\sharp(b_1 \Rightarrow t_{b1}, \ldots, b_m \Rightarrow t_{bm}, e_1 \Rightarrow v_1{:}\, SCP(e_1), \ldots, e_k \Rightarrow v_k{:}\, SCP(e_k)),$$

where $ARG(p) \cap ARG(q) = \{b_1, \ldots, b_m\}$, for $1 \leq i \leq m$, $t_{bi} = t_j$ with $b_i = a_j$, and $ARG(q - p) = \{e_1, \ldots, e_k\}$. Hence,

$$\{(b_1, t_{b1}), \ldots, (b_m, t_{bm}), (e_1, d_1), \ldots, (e_k, d_k)\} \in I_H(q^\sharp)$$

for all $d_1 \in TERM_{0,SCP(e_1)}$, $\ldots, d_n \in TERM_{0,SCP(e_n)}$ where $\{(b_1, t_{b1}), \ldots, (b_m, t_{bm})\} = \rho \cap (ARG(q) \times U_H)$.

Moreover, for $q \sqsubset p_1, \ldots, q \sqsubset p_n \in \mathcal{D}_\mathcal{P}(n > 1)$ where $p_1, \ldots, p_n$ are all predicates such that $q \sqsubset^1_P p_i$, we assume that

$$\{(a_1, t_1), \ldots, (a_m, t_m)\} \in \sqcap_{p_i^\bullet \in \Pi} I(p_i^\bullet)$$

where $\bigcup_{1 \leq i \leq n} ARG(p_i) = \{a_1, \ldots, a_m\}$ and $t_i \in TERM_{0,SCP(a_i)}$. Then, by Definition 3.9,

$$(\rho \cap ARG(q) \times U_H) \cup \{(e_1, c_{e_1}{:}\, SCP(e_1)), \ldots, (e_k, c_{e_k}{:}\, SCP(e_k))\} \in \iota_q^\bullet(\sqcap_{p_i^\bullet \in \Pi} I(p_i^\bullet))$$

where $\rho = \{(a_1, t_1), \ldots, (a_m, t_m)\}$ and $\Pi = \{p_1^\bullet, \ldots, p_n^\bullet\}$. By Definition 3.13 and 3.26, for $1 \leq i \leq n$, $\mathcal{I}_P \models p_i^\bullet(\bar{\mu}_{p_i^\bullet})$ where $\mu = \{a_1 \Rightarrow t_1, \ldots, a_m \Rightarrow t_m\}$. We can infer $P \vdash p_i^\bullet(\bar{\mu}_{p_i^\bullet})$ for $1 \leq i \leq n$ by the definition of $\mathcal{I}_P$. Then $P \vdash \sigma(q^\bullet(\bar{\mu}))$ by the specialization rule for event predicates and therefore we have $\mathcal{I}_P \models \sigma(q^\bullet(\bar{\mu}))$ by the definition of $\mathcal{I}_P$. By Definition 3.18,

$$\sigma(q^\bullet(\bar{\mu})) = q^\bullet(b_1 \Rightarrow t_{b1}, \ldots, b_u \Rightarrow t_{bu}, e_1 \Rightarrow c_{e_1}{:}\, SCP(e_1), \ldots, e_k \Rightarrow c_{e_k}{:}\, SCP(e_k)),$$

where $\{a_1, \ldots, a_m\} \cap ARG(q) = \{b_1, \ldots, b_u\}$, for $1 \leq l \leq u$, $t_{bl} = t_j$ with $b_l = a_j$, and $ARG(q) - \{a_1, \ldots, a_m\} = \{e_1, \ldots, e_k\}$. Hence,

$$\{(b_1, t_{b1}), \ldots, (b_u, t_{bu}), (e_1, c_{e_1}{:}\, SCP(e_1)), \ldots, (e_k, c_{e_k}{:}\, SCP(e_k)) \in I(q^\bullet)$$

where $\{(b_1, t_{b1}), \ldots, (b_u, t_{bu})\} = \{(a_1, t_1), \ldots, (a_m, t_m)\} \cap (ARG(q) \times U_H)$.

In the case of property predicates, we assume that

$$\{(a_1, t_1), \ldots, (a_m, t_m)\} \in \sqcap_{p_i^\sharp \in \Pi} I(p_i^\sharp)$$

where $\bigcup_{1 \leq i \leq n} ARG(p_i) = \{a_1, \ldots, a_m\}$ and $t_i \in TERM_{0,SCP(a_i)}$. Then, by Definition 3.9,

$$(\rho \cap ARG(q) \times U_H) \cup \{(e_1, d_1), \ldots, (e_k, d_k)\} \in \iota_q^\bullet(\sqcap_{p_i^\sharp \in \Pi} I(p_i^\sharp))$$

for all $d_1 \in TERM_{0,SCP(e_1)}, \ldots, d_n \in TERM_{0,SCP(e_n)}$ where $\rho = \{(a_1, t_1), \ldots, (a_m, t_m)\}$ and $\Pi = \{p_1^\sharp, \ldots, p_n^\sharp\}$. By Definition 3.13 and 3.26, for $1 \leq i \leq n$, $\mathcal{I}_P \models p_i^\sharp(\bar{\mu}_{p_i^\sharp})$ where $\mu = \{a_1 \Rightarrow t_1, \ldots, a_m \Rightarrow t_m\}$. We can infer $P \vdash p_i^\sharp(\bar{\mu}_{p_i^\sharp})$ for $1 \leq i \leq n$ by the definition of $\mathcal{I}_P$. Then $P \vdash (\sigma(q^\sharp(\bar{\mu})))\theta$ where θ is any ground substitution for $\sigma(q^\sharp(\bar{\mu}))$ by the specialization rule for property predicates and therefore we have $\mathcal{I}_P \models (\sigma(q^\sharp(\bar{\mu})))\theta$ by the definition of $\mathcal{I}_P$. By Definition 3.18,

$$\sigma(q^\sharp(\bar{\mu})) = q^\sharp(b_1 \Rightarrow t_{b1}, \ldots, b_u \Rightarrow t_{bu}, e_1 \Rightarrow v_1{:}\, SCP(e_1), \ldots, e_k \Rightarrow v_k{:}\, SCP(e_k)),$$

where $\{a_1, \ldots, a_m\} \cap ARG(q) = \{b_1, \ldots, b_u\}$, for $1 \leq l \leq u$, $t_{bl} = t_j$ with $b_l = a_j$, and $ARG(q) - \{a_1, \ldots, a_m\} = \{e_1, \ldots, e_k\}$. Hence,

$$\{(b_1, t_{b1}), \ldots, (b_u, t_{bu}), (e_1, d_1), \ldots, (e_k, d_k)\} \in I(q^\sharp)$$

for all $d_1 \in TERM_{0,SCP(e_1)}, \ldots, d_n \in TERM_{0,SCP(e_n)}$ where $\{(b_1, t_{b1}), \ldots, (b_u, t_{bu})\} = \{(a_1, t_1), \ldots, (a_m, t_m)\} \cap ARG(q) \times U_H$. ∎

For any program P we can build a deduction interpretation $\mathcal{I}_P$ that is an $H\Sigma$-model of P. In the following theorem, the completeness of the Horn clause calculus is shown by the fact that the deduction interpretation $\mathcal{I}_P$ is an $H\Sigma$-model of P (by Lemma 3.4).

Theorem 3.3 (Completeness of derivation) *Let P be a program, L an atom, and θ a ground substitution for L. If $P \models_{H\Sigma} L\theta$, then $P \vdash L\theta$.*

Proof. Let $\mathcal{I}$ be an $H\Sigma$-interpretation and $P = (\Sigma, CS)$. Assume that $P \models_{H\Sigma} L\theta$ (i.e. $CS \models_{H\Sigma} L\theta$ in Σ.) Then, by Lemma 3.4, $\mathcal{I}_P$ is an $H\Sigma$-model of P, that is $\mathcal{I}_P \models L\theta$. By the definition of $\mathcal{I}_P$, $P \vdash L\theta$ is proved. ∎

3.3.5 Sorted unification

In order to develop a resolution for our extended order-sorted logic, we need to consider the unification of order-sorted atoms to be embedded in the resolution. Let $\varphi(a_1 \Rightarrow t_1, \ldots, a_n \Rightarrow t_n)$ and $\varphi(b_1 \Rightarrow r_1, \ldots, b_n \Rightarrow r_n)$ be order-sorted atoms. To unify these atoms, we apply a unification algorithm to the pair of sequences $(t_1, \ldots, t_n)$ and $(r'_1, \ldots, r'_n)$ where $r'_i \equiv r_j$ if $a_i \equiv b_j$ for $1 \leq i, j \leq n$.

We shall give an order-sorted unification algorithm (based on [10, 30]) by translations on systems of equations. First, we introduce the basic notions of the equational system in [45] as follows.

Definition 3.28 (Equational system)

(i) An equation is a pair of terms, denoted as $t \doteq t'$. An equational system ES is a set of equations. Let $(t_1, \ldots, t_n)$ and $(r_1, \ldots, r_n)$ be two sequences of terms. The set

$$\{t_1 \doteq r_1, \ldots, t_n \doteq r_n\}$$

of equations is an equational system.

(ii) The substitution θ is a unifier of equation $t \doteq r$ if $t\theta = r\theta$. The substitution θ is a unifier of equational system ES if $t\theta = r\theta$ for every equation $t \doteq r \in ES$. We write $u(ES)$ for the set of unifiers of ES. The restriction of $u(ES)$ to a set V of variables is defined by $u(ES)\!\uparrow\! V = \{\theta\!\uparrow\! V \mid \theta \in u(ES)\}$.

(iii) Let ES, ES' be equational systems. ES is more general than ES', written $ES \leq ES'$, if $u(ES') \subseteq u(ES)$. ES and ES' are equivalent, written $ES \sim ES'$, if $u(ES) = u(ES')$. We write $ES \sim_V ES'$ if $u(ES)\!\uparrow\! V = u(ES')\!\uparrow\! V$.

The solved sets of equations (i.e. the solved equational systems) are constructed by the following definition.

Definition 3.29 *The equational system* $ES = \{x_1 : s_1 \doteq t_1, \ldots, x_n : s_n \doteq t_n\}$ *is solved, in the case*

(i) $x_1 : s_1, \ldots, x_n : s_n$ *are pairwise distinct,*

(ii) $t_i \in TERM_{s_i}$ *for* $1 \leq i \leq n$, *and*

(iii) $x_i : s_i \notin Var(t_j)$ *for* $1 \leq i, j \leq n$.

Condition (ii) indicates that t_i is of sort s_i so that $x_i : s_i$ can be substituted with t_i. A substitution θ is idempotent if $\theta\theta = \theta$, which is equivalent to $Dom(\theta) \cap Var(Cod(\theta)) = \emptyset$.

Definition 3.30 *Let* $ES = \{x_1 : s_1 \doteq t_1, \ldots, x_n : s_n \doteq t_n\}$ *be an equational system. If* ES *is solved, then it determines a unique substitution* $\theta_{ES} = \{t_1/x_1 : s_1, \ldots, t_n/x_n : s_n\}$.

The next lemma indicates the essential properties of solved equational systems.

Lemma 3.5 *Let* ES *be an equational system. If* ES *is solved, then* θ_{ES} *is a most general unifier of* ES *and idempotent.*

Proof. By conditions (i) and (ii) in Definition 3.29, θ_{ES} is well defined and a sorted substitution. We show that θ_{ES} is a most general unifier of ES. By condition (iii) in Definition 3.29 and the form of solved equational systems, we can say that $(x_i : s_i)\theta_{ES} = t_i\theta_{ES}$ for all $x_i : s_i \doteq t_i \in ES$, i.e., θ_{ES} is a unifier of ES.

Let $\gamma \in u(ES)$ and $y : s \in \mathcal{V}$. If $y : s = x_i : s_i (\in Dom(\theta_{ES}))$, then $(x_i : s_i)\gamma = t_i\gamma = (x_i : s_i)\theta_{ES}\gamma$. Otherwise, $(y : s)\gamma = (y : s)\theta_{ES}\gamma$ since $(y : s)\theta_{ES} = y : s$. Hence, for every other unifier γ of ES, we have $\theta_{ES} \leq \gamma$. Therefore, θ_{ES} is most general.

The idempotence is shown by the definition of the composition of substitutions. Since $(x_i : s_i)\theta_{ES} = t_i = t_i\theta_{ES}$, we have $(y : s)\theta_{ES}\theta_{ES} = ((y : s)\theta_{ES})\theta_{ES} = (y : s)\theta_{ES}$ for every $y : s \in \mathcal{V}$. Therefore, $\theta_{ES}\theta_{ES} = \theta_{ES}$. ∎

Let $\Sigma = (\mathcal{S}, \mathcal{F}, \mathcal{P}, \mathcal{D})$ be a sorted signature with hierarchical predicates. If the sort-hierarchy built by $\mathcal{D}_{\mathcal{S}}$ in Σ is not a lower semi-lattice, then the most general unifier of two terms $t_1, t_2 \in TERM$ may not be unique [52]. In the following unification algorithm, we assume that every sort-hierarchy is a lower semi-lattice.

Definition 3.31 (Sorted unification algorithm) *Let* $(ES; S)$ *be a pair of equational systems. A translation of* $(ES; S)$ *in an 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 $t_i \not\equiv r_i$.

Transposition $(ES \cup \{t \doteq x : s\}; S) \Longrightarrow (ES \cup \{x : s \doteq t\}; S)$
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 $\tau = \{t/x\colon s\}$ *if* $t \in TERM_s$[6] *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* $\tau = \{z\colon s_3/x\colon s_1, z\colon s_3/y\colon s_2\}$
if s_3 *is neither* s_1, s_2, *nor* $\perp$.

We write $(ES; S) \Rightarrow_u (ES'; S')$ if $(ES'; S')$ is derivable from $(ES; S)$ by applying one rule from the order-sorted unification algorithm. The following lemma demonstrates one property of the order-sorted unification algorithm.

Lemma 3.6 *Let ES be an equational system. For each equation $t \doteq t' \in ES$, at most only one rule from the order-sorted unification algorithm can be applied to it.*

Proof.
Let $t, t' \in \mathcal{V}$ with $t = x\colon s_1$ and $t' = y\colon s_2$.

- If $t \equiv t'$, then the identity rule is applied.
- If $s_1 <_S s_2$, then the transposition rule is applied.
- If $s_2 <_S s_1$, then the substitution 1 rule is applied.
- If $glb(s_1, s_2) = \perp$, then no rule can be applied.
- Otherwise, i.e., when $glb(s_1, s_2) \neq s_1 \neq s_2 \neq \perp$, the substitution 2 rule is applied.

Let $t \in \mathcal{V}$ with $t = x\colon s$ and $t' \notin \mathcal{V}$.

- If $t \in Var(t')$ or $t' \notin TERM_s$, then no rule can be applied.
- Otherwise, the substitution 1 rule is applied.

Let $t \notin \mathcal{V}$ and $t' \in \mathcal{V}$.

- The transposition rule can be applied.

Let $t, t' \notin \mathcal{V}$ (i.e. t, t' are of the forms $f(t_1, \ldots, t_n)$ and $g(r_1, \ldots, r_m)$).

- If $t \equiv t'$, then the identity rule is applied.
- If $f \equiv g$ (where must $m = n$) but $t_i \not\equiv r_i$ for some $i \in \{1, \ldots, n\}$, then the decomposition rule is applied.
- Otherwise, i.e., when $f \not\equiv g$, no rule can be applied. ∎

We define the complexity of an equational system, represented as the number of symbols occurring in it.

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

Definition 3.32 *The number of occurrences of function symbols and variables in a term t is defined by*

(i) $\| x\colon s \| = 1$ *and*

(ii) $\| f(t_1, \ldots, t_n) \| = \| t_1 \| + \cdots + \| t_n \| + 1$.

Furthermore, the number of occurrences of function symbols and variables in an equational system $ES = \{t_1 \doteq r_1, \ldots, t_n \doteq r_n\}$ *is defined by*

$$\| ES \| = \sum_{i=1}^{n} (\| t_i \| + \| r_i \|).$$

The set of variables occurring in an equational system ES is defined by $ESVar(ES) = \bigcup_{t_i \doteq r_i \in ES} (Var(t_i) \cup Var(r_i))$. The set of equations of the form $t \doteq x\colon s$ in ES where $t \notin \mathcal{V}$ or $t \in \mathcal{V}_{s'}$ with $s' <_S s$ is defined by $tp(ES) = \{t \doteq x\colon s \in ES \mid t \notin \mathcal{V},\ or\ t \in \mathcal{V}_{s'}\ s.t.\ s' <_S s\}$, and the transposition rule is applicable to the equation form.

Definition 3.33 *The complexity of an equational system ES is given by the triple*

$$\mu(ES) = \langle |ESVar(ES)|, \| ES \|, |tp(ES)| \rangle.$$

Let $(ES_0; S_0) \Rightarrow_u (ES_1; S_1) \Rightarrow_u \ldots \Rightarrow_u (ES_n; S_n)$ be a sequence of applications of rules from the order-sorted unification algorithm. We define the set of used variables for $ES_i \cup S_i$ in the sequence as follows:

$$UV(ES_i \cup S_i) = ESVar(ES_0 \cup S_0) \cup ESVar(ES_1 \cup S_1) \cup \cdots \cup ESVar(ES_i \cup S_i).$$

The next two lemmas will be used to prove the correctness of the order-sorted unification algorithm.

Lemma 3.7 *Let $(ES_0; S_0) \Rightarrow_u (ES_1; S_1) \Rightarrow_u \ldots \Rightarrow_u (ES_n; S_n)$ be a sequence of applications of rules from the order-sorted unification algorithm. If $(ES_i; S_i) \Rightarrow_u (ES_j; S_j)$ for $0 \leq i < j \leq n$, then $ES_i \cup S_i \sim_{UVar(ES_i \cup S_i)} ES_j \cup S_j$.*

Proof. We will prove that if $(ES_i; S_i) \Rightarrow_u (ES_j; S_j)$, then $u(ES_i \cup S_i) \upharpoonright UV(ES_i \cup S_i) = u(ES_j \cup S_j) \upharpoonright UV(ES_i \cup S_i)$. Consider each case of the rules from the order-sorted unification algorithm.

(a) Identity. We obtain $ES_j \cup S_j = (ES_i - \{t \doteq t\}) \cup S_i$ by this rule. Then $u(ES_i \cup S_i) = u(ES_j \cup S_j)$ since $t\theta = t\theta$ for all substitutions θ.

(b) Decomposition. We obtain $ES_j = (ES_i - \{f(t_1, \ldots, t_n) \doteq f(r_1, \ldots, r_n)\}) \cup \{t_1 \doteq r_1, \ldots, t_n \doteq r_n\}$ and $S_j = S_i$ by this rule. By Definition 3.17, $f(t_1, \ldots, t_n)\theta = f(t_1\theta, \ldots, t_n\theta)$. Hence $f(t_1, \ldots, t_n)\theta = f(r_1, \ldots, r_n)\theta$ if and only if $t_i\theta = r_i\theta$ for $1 \leq i \leq n$. It follows $u(ES_i \cup S_i) = u(ES_j \cup S_j)$.

(c) Transposition. $(x\colon s)\theta = t\theta$ if and only if $t\theta = (x\colon s)\theta$. Hence we have $u(ES_i \cup S_i) = u(ES_j \cup S_j)$.

(d) Substitution 1. Let $ES \cup \{x\colon s \doteq t\}$ be an equational system. For all $\theta \in u(ES \cup \{x\colon s \doteq t\} \cup S_i)$, we have $(x\colon s)\theta = t\theta$. If $t \in TERM_s$, then $\{t/x\colon s\}$ is a sorted substitution. Then, for all $t_i \in TERM$, $t_i\{t/x\colon s\}\theta = t_i\theta$.

By the substitution 1 rule, if $ES_i = ES \cup \{x\colon s \doteq t\}$ then $ES_j = ES\{t/x\colon s\}$ and $S_j = S_i\{t/x\colon s\} \cup \{x\colon s \doteq t\}$ where $t \in TERM_s$ and $x\colon s \notin Var(t)$. Hence

$\theta \in u(ES_i \cup S_i)$
iff $t_k\theta = r_k\theta$ for all $t_k \doteq r_k \in ES_i \cup S_i$
iff $t_k\{t/x\!:\!s\}\theta = r_k\{t/x\!:\!s\}\theta$ (by the above conclusion and $(x\!:\!s)\theta = t\theta$)
for all $t_k\{t/x\!:\!s\} \doteq r_k\{t/x\!:\!s\} \in (ES \cup S_i)\{t/x\!:\!s\}$
iff $t'_k\theta = r'_k\theta$ for all $t'_k \doteq r'_k \in (ES_j \cup S_j)$
iff $\theta \in u(ES_j \cup S_j)$.

(e) Substitution 2. Let $ES \cup \{x\!:\!s_1 \doteq y\!:\!s_2\}$ be an equational system. For all $\theta \in u(ES \cup \{x\!:\!s_1 \doteq y\!:\!s_2\} \cup S_i)$, we have $(x\!:\!s_1)\theta = (y\!:\!s_2)\theta$. If $glb(s_1, s_2) = s_3$ with $glb(s_1, s_2) \neq s_1 \neq s_2 \neq \bot$, then $\{z\!:\!s_3/x\!:\!s_1, z\!:\!s_3/y\!:\!s_2\}$ is a sorted substitution. Then, for all $t_i \in TERM$, $t_i\{z\!:\!s_3/x\!:\!s_1, z\!:\!s_3/y\!:\!s_2\}\gamma\theta = t_i\theta$ where $\gamma = \{(x\!:\!s_1)\theta/z\!:\!s_3\}$ ($(x\!:\!s_1)\theta \in TERM_{s_3}$ since $glb(s_1, s_2) = s_3$).

By the substitution 2 rule, if $ES_i = ES \cup \{x\!:\!s_1 \doteq y\!:\!s_2\}$ then $ES_j = ES\tau$ and $S_j = S_i\tau \cup \{x\!:\!s_1 \doteq z\!:\!s_3, y\!:\!s_2 \doteq z\!:\!s_3\}$ where $\tau = \{z\!:\!s_3/x\!:\!s_1, z\!:\!s_3/y\!:\!s_2\}$. Hence

$\theta \in u(ES_i \cup S_i)\!\uparrow\! UV(ES_i \cup S_i)$
iff $t_k\theta = r_k\theta$ for all $t_k \doteq r_k \in ES_i \cup S_i$
iff $t_k\tau\gamma\theta = r_k\tau\gamma\theta$ for all $t_k\tau \doteq r_k\tau \in (ES \cup S_i)\tau$
(by the above conclusion and $(x\!:\!s_1)\gamma\theta = (z\!:\!s_3)\gamma\theta = (y\!:\!s_2)\gamma\theta$
where $\tau = \{z\!:\!s_3/x\!:\!s_1, z\!:\!s_3/y\!:\!s_2\}$ and $\gamma = \{(x\!:\!s_1)\theta/z\!:\!s_3\}$)
iff $\theta = r'_k\gamma\theta$ and $(x\!:\!s_1)\gamma\theta = (z\!:\!s_3)\gamma\theta = (y\!:\!s_2)\gamma\theta$ for all $t'_k \doteq r'_k \in (ES_j \cup S_j)$
where $\tau = \{z\!:\!s_3/x\!:\!s_1, z\!:\!s_3/y\!:\!s_2\}$
iff $\gamma\theta \in u(ES_j \cup S_j)$ with $\gamma = \{(x\!:\!s)\theta/z\!:\!s_3\}$
iff $\theta \in u(ES_j \cup S_j)\!\uparrow\! UV(ES_i \cup S_i)$.

Therefore, we have $ES_i \cup S_i \sim_{UVar(ES_i \cup S_i)} ES_j \cup S_j$. ∎

Lemma 3.8 *Let ES, ES' be equational systems, and let V, V' be sets of variables such that $V \subseteq V'$. If $ES \sim_{V'} ES'$, then $ES \sim_V ES'$.*

Proof. We assume $ES \sim_{V'} ES'$. That is, $\{\theta\!\uparrow\! V' \mid \theta \in u(ES)\} = \{\theta\!\uparrow\! V' \mid \theta \in u(ES')\}$. Then

$$\begin{aligned}\{\theta\!\uparrow\! V \mid \theta \in u(ES)\} &= \{\theta'\!\uparrow\! V \mid \theta' \in \{\theta\!\uparrow\! V' \mid \theta \in u(ES)\}\} \\ &= \{\theta\!\uparrow\! V \mid \theta' \in \{\theta'\!\uparrow\! V' \mid \theta \in u(ES')\}\} \\ &= \{\theta\!\uparrow\! V \mid \theta \in u(ES')\}.\end{aligned}$$

Therefore $ES \sim_V ES'$. ∎

The lexicographic ordering on tuples of natural numbers is defined by: $\langle t_1, \ldots, t_n\rangle >_l \langle r_1, \ldots, r_n\rangle$ iff there exists $i \in \{1, \ldots, n\}$ such that $t_j = r_j$ for $1 \leq j < i$ and $t_i > r_i$. We show that the order-sorted unification algorithm always terminates for any ordered pair of equational systems.

Lemma 3.9 *The order-sorted unification algorithm terminates, when a pair $(ES; \emptyset)$ of equational systems is given.*

Proof. We show that if $(ES; S) \Rightarrow_u (ES'; S')$, then $\mu(ES) >_l \mu(ES')$. The identity and the decomposition rules decrease $\| ES \|$ but dose not increase $|ESVar(ES)|$. By an application of the transposition rule, both $\| ES \|$ and $|ESVar(ES)|$ are not decreased but $|tp(ES)|$ is lowered. The substitution 1 and the substitution 2 rules reduce the number of variables in ES. ∎

The correctness of the order-sorted unification algorithm is shown in the following theorem.

Theorem 3.4 (Correctness of sorted unification) *Let $(ES_0; S_0)$ with $S_0 = \emptyset$ be a pair of equational systems. Then there exists a finite sequence*

$$(ES_0; S_0) \Rightarrow_u (ES_1; S_1) \Rightarrow_u \ldots \Rightarrow_u (ES_n; S_n)$$

of applications of rules from the order-sorted unification algorithm. If ES_n is empty, then $\theta_{S_n}{\uparrow}UV(ES_0)$ is a most general unifier of ES_0. Otherwise, there exists no most general unifier of ES_0.

Proof. By Lemma 3.9, the sequence $(ES_0; S_0) \Rightarrow_u (ES_1; S_1) \Rightarrow_u \ldots \Rightarrow_u (ES_n; S_n)$ with $S_0 = \emptyset$ is finite.

If ES_n is empty, then $ES_n \cup S_n$ is solved. Then, By Lemma 3.5, $\theta_{ES_n \cup S_n}$ is a most general unifier of $ES_n \cup S_n$, and $\theta_{ES_n \cup S_n}{\uparrow}UV(ES_0 \cup S_0) \in u(ES_n \cup S_n){\uparrow}UV(ES_0 \cup S_0)$. Thus $\theta_{ES_n \cup S_n}{\uparrow}UV(ES_0 \cup S_0)$ is a most general unifier of

$$\{x\colon s \doteq t \in (ES_n \cup S_n) \mid x\colon s \in UV(ES_0 \cup S_0)\},$$

i.e., $\theta_{ES_n \cup S_n}{\uparrow}UV(ES_0 \cup S_0)$ is most general in $u(ES_n \cup S_n){\uparrow}UV(ES_0 \cup S_0)$. Then, by Lemma 3.7, we have

$$\begin{aligned}
u(ES_0 \cup S_0){\uparrow}UV(ES_0 \cup S_0) &= u(ES_1 \cup S_1){\uparrow}UV(ES_0 \cup S_0),\\
u(ES_1 \cup S_1){\uparrow}UV(ES_1 \cup S_1) &= u(ES_2 \cup S_2){\uparrow}UV(ES_1 \cup S_1),\\
&\ldots\\
u(ES_{n-1} \cup S_{n-1}){\uparrow}UV(ES_{n-1} \cup S_{n-1}) &= u(ES_n \cup S_n){\uparrow}UV(ES_{n-1} \cup S_{n-1}).
\end{aligned}$$

Lemma 3.8 implies $u(ES_0 \cup S_0){\uparrow}UV(ES_0 \cup S_0) = u(ES_1 \cup S_1){\uparrow}UV(ES_0 \cup S_0) = \cdots = u(ES_n \cup S_n){\uparrow}UV(ES_0 \cup S_0)$ since $UV(ES_0 \cup S_0) \subseteq UV(ES_1 \cup S_1) \subseteq \cdots \subseteq UV(ES_n \cup S_n)$. Therefore $\theta_{S_n}{\uparrow}UV(ES_0)$ is a most general unifier of ES_0 since $u(ES_0 \cup S_0){\uparrow}UV(ES_0 \cup S_0) = u(ES_0 \cup S_0)$ and $S_0 = ES_n = \emptyset$.

Otherwise, ES_n contains at least one of the following equations:

(i) $f(t_1, \ldots, t_n) \doteq g(r_1, \ldots, r_m)$ where $f \not\equiv g$,

(ii) $x\colon s \doteq t$ where $x\colon s \in FVar(t)$ or $t \notin TERM_s$, or

(iii) $x_1\colon s_1 \doteq y_2\colon s_2$ where $glb(s_1, s_2) = \bot$.

In the each case, ES_n is not unifiable. Therefore there exists no most general unifier of $ES_0 \cup S_0$, since $u(ES_0 \cup S_0) = u(ES_n \cup S_n){\uparrow}UV(ES_0 \cup S_0) = \emptyset$. ∎

In the following example, we demonstrate how rules in the order-sorted unification algorithm are applied to unify sorted terms.

Example 3.12 *Let* $\Sigma = (\mathcal{S}, \mathcal{F}, \mathcal{P}, \mathcal{D})$ *such that* $\mathcal{S} = \{s_1, s_2, s_3, \bot, \top\}$, $\mathcal{F} = \{f, c\}$, $\mathcal{P} = \emptyset$, *and* $\mathcal{D} = (\mathcal{D}_\mathcal{S}, \mathcal{D}_\mathcal{F}, \mathcal{D}_\mathcal{P})$ *with*

$$\begin{aligned}\mathcal{D}_S &= \{s_3 \sqsubset_S s_1, s_3 \sqsubset_S s_2, \bot \sqsubset_S s_3, s_2 \sqsubset_S \top, s_2 \sqsubset_S \top\},\\ \mathcal{D}_\mathcal{F} &= \{f\colon s_1 \times s_2 \to s_3, c\colon \to s_2\},\\ \mathcal{D}_\mathcal{P} &= \emptyset.\end{aligned}$$

Given the equational system

$$\{z\colon s_3 \doteq f(x\colon s_1, c\colon s_2), x\colon s_1 \doteq y\colon s_2\},$$

we can obtain the following finite sequence by applying rules from the order-sorted unification algorithm.

$$\begin{aligned}&(\{x\colon s_1 \doteq y\colon s_2, f(x\colon s_1, c\colon s_2) \doteq z\colon s_3\}; \emptyset)\\ &\quad \Rightarrow_u (\{f(v\colon s_3, c\colon s_2) \doteq z\colon s_3\}; \{x\colon s_1 \doteq v\colon s_3, y\colon s_2 \doteq v\colon s_3\}) \quad \textit{(by substitution 2)}\\ &\quad \Rightarrow_u (\{z\colon s_3 \doteq f(v\colon s_3, c\colon s_2)\}; \{x\colon s_1 \doteq v\colon s_3, y\colon s_2 \doteq v\colon s_3\}) \quad \textit{(by transposition)}\\ &\quad \Rightarrow_u (\emptyset; \{x\colon s_1 \doteq v\colon s_3, y\colon s_2 \doteq v\colon s_3, z\colon s_3 \doteq f(v\colon s_3, c\colon s_2)\}) \quad \textit{(by substitution 1).}\end{aligned}$$

As a result, the sorted substitution

$$\theta = \{v\colon s_3/x\colon s_1, v\colon s_3/y\colon s_2, f(v\colon s_3, c\colon s_2)/z\colon s_3\}$$

is a most general unifier of $\{z\colon s_3 \doteq f(x\colon s_1, c\colon s_2), x\colon s_1 \doteq y\colon s_2\}$.

3.3.6 Resolution with predicate-hierarchy

In this section, we formalize a (Horn clause) resolution for our order-sorted logic with hierarchies and eventuality that is based on the linear resolution in [16]. The Horn clause resolution contains the following inference rules.

Definition 3.34 (Resolvent) *Let* $P = (\Sigma, CS)$ *be a program.*

(i) **R1-resolution rule.** *Let* G *be a goal and* $L' \leftarrow G' \in CS$. *If* θ *is a unifier of* $L \in G$ *and* L', *then* $(G - \{L\})\theta \cup G'\theta$ *is an unstricted resolvent of* G *with respect to* L *and* $L' \leftarrow G'$. *We write*

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

(ii) **R2-resolution rule.** *Let* G *be a goal and* $\varphi(\bar{\mu}') \leftarrow G' \in CS$. *If* $\langle\varphi\rangle = \langle\psi\rangle$, $[\varphi] \sqsubset_P [\psi] \in \mathcal{D}_\mathcal{P}$, *and* θ *is a unifier of* $\psi(\bar{\mu}) \in G$ *and* $\sigma(\psi(\bar{\mu}'))$, *then* $(G - \{\psi(\bar{\mu})\})\theta \cup G'\theta$ *is an unstricted resolvent of* G *with respect to* $\psi(\bar{\mu})$ *and* $\varphi(\bar{\mu}') \leftarrow G'$. *We write*

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

(iii) **R3-resolution rule.** *Let* G *be a goal and* $\varphi_1(\bar{\mu}_1) \leftarrow G_1$, ..., $\varphi_n(\bar{\mu}_n) \leftarrow G_n \in CS$. *If* $\langle\varphi_1\rangle = \cdots = \langle\varphi_n\rangle = \langle\psi\rangle$ *where* $[\varphi_1], \ldots, [\varphi_n] (n > 1)$ *are all predicates such that* $[\psi] \sqsubset^1_P [\varphi_i] \in \mathcal{D}_\mathcal{P}$, *and* θ *is a unifier of* $\psi(\bar{\mu}) \in G$ *and* $\sigma(\psi(\bar{\mu}'))$ *with* $\mu' = \mu_1 \cup \cdots \cup \mu_n$, *then* $(G - \{\psi(\bar{\mu})\})\theta \cup (G_1 \cup \ldots \cup G_n)\theta$ *is an unstricted resolvent of* G *with respect to* $\psi(\bar{\mu})$ *and* $\varphi_1(\bar{\mu}_1) \leftarrow G_1$, ..., $\varphi_n(\bar{\mu}_n) \leftarrow G_n$. *We write*

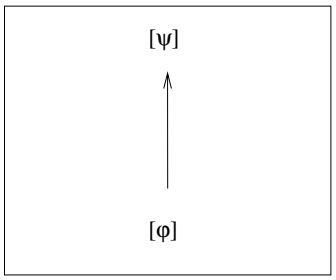

Figure 3.1 A subpredicate relation for R2-resolution rule

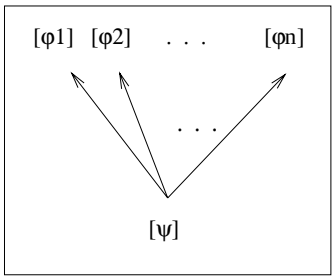

Figure 3.2 A subpredicate relation for R3-resolution rule

$$G \xrightarrow{\theta}_{R3} (G - \{\psi(\bar{\mu})\})\theta \cup (G_1 \cup \ldots \cup G_n)\theta.$$

Figure 3.1 (resp. Figure 3.2) shows a subpredicate relation as a condition for applying the R2-resolution rule (resp. the R3-resolution rule). We write $G \xrightarrow{\theta} G'$ if $G \xrightarrow{\theta}_{R1} G'$, $G \xrightarrow{\theta}_{R2} G'$, or $G \xrightarrow{\theta}_{R3} G'$. An unstricted resolvent is a resolvent if the unifier θ is most general.

Example 3.13 *The sorted signature Σ comprises the following symbols*

$$\begin{aligned}
\mathcal{S} &= \{s_1, s_2, s_3, \top, \bot\},\\
\mathcal{F} &= \{f, c\},\\
\mathcal{P} &= \{p_1, p_2, q\},
\end{aligned}$$

and the declaration $\mathcal{D} = (\mathcal{D}_\mathcal{S}, \mathcal{D}_\mathcal{F}, \mathcal{D}_\mathcal{P})$ constructed by

$$\begin{aligned}
\mathcal{D}_\mathcal{S} &= \{s_3 \sqsubset_S s_1, s_3 \sqsubset_S s_2,\\
&\qquad \bot \sqsubset_S s_3, s_1 \sqsubset_S \top, s_2 \sqsubset_S \top\},\\
\mathcal{D}_\mathcal{F} &= \{f\colon s_1 \times s_2 \to s_3, c\colon \to s_2\},\\
\mathcal{D}_\mathcal{P} &= \{p_1 \sqsubset_P p_2\} \cup\\
&\qquad \{p_1\colon\{(l_1, \top)\},\\
&\qquad p_2\colon\{(l_1, \top), (l_2, s_3)\},\\
&\qquad q\colon\{(l_1, \top)\}\ \}.
\end{aligned}$$

The program P is the ordered pair (Σ, CS) with

$$CS = \{p_1^\sharp(l_1 \Rightarrow y\colon s_2) \leftarrow q^\bullet(l_1 \Rightarrow y\colon s_2)\}.$$

With respect to the program P, we have an unstricted resolvent of $p_2^\sharp(l_1 \Rightarrow x{:}\, s_1, l_2 \Rightarrow f(x{:}\, s_1, c{:}\, s_2))$ as follows

$$p_2^\sharp(l_1 \Rightarrow x{:}\, s_1, l_2 \Rightarrow f(x{:}\, s_1, c{:}\, s_2)) \xrightarrow{\theta}_{R2} q^\bullet(l_1 \Rightarrow v{:}\, s_3)$$

where $\theta = \{v{:}\, s_3/x{:}\, s_1, v{:}\, s_3/y{:}\, s_2, f(v{:}\, s_3, c{:}\, s_2)/z{:}\, s_3\}$ is a most general unifier of $p_2^\sharp(l_1 \Rightarrow x{:}\, s_1, l_2 \Rightarrow f(x{:}\, s_1, c{:}\, s_2))$ and $p_2^\sharp(l_1 \Rightarrow y{:}\, s_2, l_2 \Rightarrow z{:}\, s_3)$ $(= \sigma(p_2^\sharp(l_1 \Rightarrow y{:}\, s_2))$, which is the result of Example 3.12.

Definition 3.35 (Resolution) *Let P be a program. A finite sequence*

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

is an unstricted resolution of G_0 with respect to P where $n \geq 0$. We denote it by $P{:}\, G_0 \xrightarrow{\theta}\!\!\!\!\rightarrow G_n$ with $\theta = \theta_1 \cdots \theta_n$.

A resolution is called successful if $G_n = \square$, that is $P{:}\, G_0 \xrightarrow{\theta}\!\!\!\!\rightarrow \square$. In that case, the composition $\theta{:}= \theta_1 \cdots \theta_n$ of the substitutions to the variables in the initial goal G_0 is called a computed answer substitution. An unstricted resolution is called a resolution if the unstricted resolvents are resolvents.

Lemma 3.10 *Let $\mathcal{I}$ be a Σ-interpretation, and let $F_1, \ldots, F_n$ be formulas. $\mathcal{I} \models \forall F_1, \ldots,$ $\mathcal{I} \models \forall F_n$ if and only if $\mathcal{I} \models \forall(F_1' \wedge \ldots \wedge F_n')$ where $F_1', \ldots, F_n'$ are variants of $F_1, \ldots, F_n$ such that $FVar(F_1') \cap \cdots \cap FVar(F_n') = \emptyset$.*

Proof. Let $F_1', \ldots, F_n'$ be variants of $F_1, \ldots, F_n$ with $FVar(F_1') \cap \cdots \cap FVar(F_n') = \emptyset$.

$\mathcal{I} \models \forall F_1, \ldots, \mathcal{I} \models \forall F_n$
- iff $\mathcal{I} \models \forall F_1', \ldots, \mathcal{I} \models \forall F_n'$
- iff for $1 \leq i \leq n$, for all $d_1 \in I(s_1), \ldots, d_k \in I(s_k)$,
  $\mathcal{I}[d_1/x_1{:}\, s_1, \ldots, d_k/x_k{:}\, s_k] \models F_i'$ where $FVar(F_i) = \{x_1{:}\, s_1, \ldots, x_k{:}\, s_k\}$
- iff for all $d_1' \in I(s_1'), \ldots, d_m' \in I(s_m')$,
  $\mathcal{I}[d_1'/y_1{:}\, s_1', \ldots, d_m'/y_m{:}\, s_m'] \models F_1', \ldots, \mathcal{I}[d_1'/y_1{:}\, s_1', \ldots, d_m'/y_m{:}\, s_m'] \models F_n'$
  where $FVar(F_1') \cup \cdots \cup FVar(F_n') = \{y_1{:}\, s_1', \ldots, y_m{:}\, s_m'\}$
- iff $\mathcal{I} \models \forall(F_1' \wedge \ldots \wedge F_n')$ ∎

In order to prove the soundness of the resolution rules, we will generalize Lemma 3.2 in the following lemma.

Lemma 3.11 *Let $\Sigma = (\mathcal{S}, \mathcal{F}, \mathcal{P}, \mathcal{D})$ be a sorted signature with hierarchical predicates, σ the argument supplementation, and $p, p_1, \ldots, p_n, q \in \mathcal{P}$ Then, the following holds:*

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

(a) $\forall(p^\bullet(\bar{\mu})) \models_{H\Sigma} \forall(\sigma(q^\bullet(\bar{\mu})))$ and

(b) $\forall(p^\sharp(\bar{\mu})) \models_{H\Sigma} \forall(\sigma(q^\sharp(\bar{\mu})))$

where $\mu = \{a_1 \Rightarrow t_1, \ldots, a_n \Rightarrow t_n\}$ such that $ARG(p) = \{a_1, \ldots, a_n\}$ and $t_i \in TERM_{SCP(a_i)}$.

(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* $q \sqsubset_P^1 p_i$*, then*

(a) $\{\forall(p_1^\bullet(\bar{\mu}_{p_1})), \ldots, \forall(p_n^\bullet(\bar{\mu}_{p_n}))\} \models_{H\Sigma} \forall(\sigma(q^\bullet(\bar{\mu})))$ *and*

(b) $\{\forall(p_1^\sharp(\bar{\mu}_{p_1})), \ldots, \forall(p_n^\sharp(\bar{\mu}_{p_n}))\} \models_{H\Sigma} \forall(\sigma(q^\sharp(\bar{\mu})))$

where $\mu = \{a_1 \Rightarrow t_1, \ldots, a_m \Rightarrow t_m\}$ *such that* $\bigcup_{1 \leq i \leq n} ARG(p_i) = \{a_1, \ldots, a_m\}$ *and* $t_i \in TERM_{SCP(a_i)}$.

Proof.
(1) (a) If $\mathcal{I} \models \forall y_1 \colon s_1 \cdots y_r \colon s_r (p^\bullet(\bar{\mu}))$ where $\mathcal{I} = (M, \beta)$ and $\mu = \{a_1 \Rightarrow t_1, \ldots, a_n \Rightarrow t_n\}$, then, for all $u_1 \in I(s_1), \ldots, u_r \in I(s_r)$, $\mathcal{I}[u_1/y_1 \colon s_1, \ldots, u_r/y_r \colon s_r] \models p^\bullet(\bar{\mu})$. By the proof of Lemma 3.2, $\mathcal{I}[u_1/y_1 \colon s_1, \ldots, u_r/y_r \colon s_r] \models \sigma(q^\bullet(\bar{\mu}))$. Therefore $\mathcal{I} \models \forall y_1 \colon s_1 \cdots y_r \colon s_r (\sigma(q^\bullet(\bar{\mu})))$ with $FVar(\sigma(q^\bullet(\bar{\mu}))) \subseteq \{y_1 \colon s_1, \ldots, y_r \colon s_r\}$.

(b) If $\mathcal{I} \models \forall y_1 \colon s_1 \cdots y_r \colon s_r (p^\sharp(\bar{\mu}))$ where $\mathcal{I} = (M, \beta)$ and $\mu = \{a_1 \Rightarrow t_1, \ldots, a_n \Rightarrow t_n\}$, then, for all $u_1 \in I(s_1), \ldots, u_r \in I(s_r)$, $\mathcal{I}[u_1/y_1 \colon s_1, \ldots, u_r/y_r \colon s_r] \models p^\sharp(\bar{\mu})$. By the proof of Lemma 3.2, $\mathcal{I}[u_1/y_1 \colon s_1, \ldots, u_r/y_r \colon s_r] \models \forall v_1 \colon s_1' \ldots v_k \colon s_k' (\sigma(q^\sharp(\bar{\mu})))$. Therefore $\mathcal{I} \models \forall y_1 \colon s_1 \cdots y_r \colon s_r v_1 \colon s_1' \ldots v_k \colon s_k' (\sigma(q^\sharp(\bar{\mu})))$ with $FVar(\sigma(q^\sharp(\bar{\mu}))) \subseteq \{y_1 \colon s_1, \ldots, y_r \colon s_r, v_1 \colon s_1', \ldots, v_k \colon s_k'\}$.

(2) (a) If $\mathcal{I} \models \forall(p_i^\bullet(\bar{\mu}_{p_i^\bullet}))$ for $1 \leq i \leq n$ where $\mathcal{I} = (M, \beta)$, then, by Lemma 3.10, $\mathcal{I} \models \forall y_1 \colon s_1 \cdots y_r \colon s_r (p_1^\bullet(\bar{\mu}_{p_1^\bullet}) \wedge \ldots \wedge p_n^\bullet(\bar{\mu}_{p_n^\bullet}))$ with $FVar(p_1^\bullet(\bar{\mu}_{p_1^\bullet}) \wedge \ldots \wedge p_n^\bullet(\bar{\mu}_{p_n^\bullet})) = \{y_1 \colon s_1, \ldots, y_r \colon s_r\}$. Hence, for $1 \leq i \leq n$, $\mathcal{I}[d_1/y_1 \colon s_1, \ldots, d_r/y_r \colon s_r] \models (p_i^\bullet(\bar{\mu}_{p_i^\bullet}))$ for all $d_1 \in I(s_1), \ldots, d_r \in I(s_r)$. By the proof of Lemma 3.2, $\mathcal{I}[d_1/y_1 \colon s_1, \ldots, d_r/y_r \colon s_r] \models \sigma(q^\bullet(\bar{\mu}))$. Therefore $\mathcal{I} \models \forall y_1 \colon s_1 \cdots y_r \colon s_r (\sigma(q^\bullet(\bar{\mu})))$ with $FVar(\sigma(q^\bullet(\bar{\mu}))) \subseteq \{y_1 \colon s_1, \ldots, y_r \colon s_r\}$.

(b) If $\mathcal{I} \models \forall(p_i^\sharp(\bar{\mu}_{p_i^\sharp}))$ for $1 \leq i \leq n$ where $\mathcal{I} = (M, \beta)$. Then, by Lemma 3.10, $\mathcal{I} \models \forall y_1 \colon s_1 \cdots y_r \colon s_r (p_1^\sharp(\bar{\mu}_{p_1^\sharp}) \wedge \ldots \wedge p_n^\sharp(\bar{\mu}_{p_n^\sharp}))$ with $FVar(p_1^\sharp(\bar{\mu}_{p_1^\sharp}) \wedge \ldots \wedge p_n^\sharp(\bar{\mu}_{p_n^\sharp})) = \{y_1 \colon s_1, \ldots, y_r \colon s_r\}$. Hence, for $1 \leq i \leq n$, $\mathcal{I}[d_1/y_1 \colon s_1, \ldots, d_r/y_r \colon s_r] \models (p_i^\sharp(\bar{\mu}_{p_i^\sharp}))$ for all $d_1 \in I(s_1), \ldots, d_r \in I(s_r)$. By the proof of Lemma 3.2, $\mathcal{I}[d_1/y_1 \colon s_1, \ldots, d_r/y_r \colon s_r] \models \forall v_1 \colon s_1' \ldots v_k \colon s_k' \sigma(q^\sharp(\bar{\mu}))$. Therefore $\mathcal{I} \models \forall y_1 \colon s_1 \cdots y_r \colon s_r v_1 \colon s_1' \ldots v_k \colon s_k' (\sigma(q^\sharp(\bar{\mu})))$ with $FVar(\sigma(q^\sharp(\bar{\mu}))) \subseteq \{y_1 \colon s_1, \ldots, y_r \colon s_r, v_1 \colon s_1', \ldots, v_k \colon s_k'\}$. ∎

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

Theorem 3.5 (Soundness of resolution) *Let* P *be a program and* G *a goal. If there exists a successful resolution of* G *with 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 $\theta = \theta_1 \cdots \theta_n$ be a successful resolution.
Base case: $n = 1$.

(a) If $G \xrightarrow{\theta_1}_{R1} \square$, then, by Definition 3.34, θ_1 is a unifier of $L' \in CS$ and $L(= G)$, i.e., $L'\theta_1 = L\theta_1$. Suppose that $\mathcal{I} \models P$ where $\mathcal{I}$ is an $H\Sigma$-interpretation. Then $\mathcal{I} \models \forall(L'\theta_1)$. Therefore $P \models_{H\Sigma} G\theta_1$ since $L'\theta_1 = L\theta_1 (= G\theta_1)$.

(b) If $G \xrightarrow{\theta_1}_{R2} \square$, then, by Definition 3.34, θ_1 is a unifier of $\psi(\bar{\mu})(= G)$ and $\sigma(\psi(\bar{\mu}'))$, i.e., $(\psi(\bar{\mu}))\theta_1 = (\sigma(\psi(\bar{\mu}')))\theta_1$ where $\varphi(\bar{\mu}') \in CS$. Suppose that $\mathcal{I} \models P$ where $\mathcal{I}$ is an $H\Sigma$-interpretation. We can get $\mathcal{I} \models \forall(\sigma(\psi(\bar{\mu}')))$ by Lemma 3.11 (1) since $[\varphi] \sqsubset_P [\psi] \in \mathcal{D}_{\mathcal{P}}$ and $\mathcal{I} \models \forall(\varphi(\bar{\mu}'))$. Then, by Lemma 3.1, $\mathcal{I} \models \forall((\sigma(\psi(\bar{\mu}')))\theta_1)$. Therefore $P \models_{H\Sigma} G\theta_1$ since $(\psi(\bar{\mu}))\theta_1 = (\sigma(\psi(\bar{\mu}')))\theta_1$.

(c) If $G \xrightarrow{\theta_1}_{R3} \square$, then, by Definition 3.34, θ_1 is a unifier of $\psi(\bar{\mu})(= G)$ and $\sigma(\psi(\bar{\mu}'))$ with $\mu' = \mu_1 \cup \cdots \cup \mu_n$, i.e., $\psi(\bar{\mu})\theta_1 = (\sigma(\psi(\bar{\mu}')))\theta_1$ where $\varphi_1(\bar{\mu}_1), \ldots, \varphi_n(\bar{\mu}_n) \in CS$. Suppose that $\mathcal{I} \models P$ where $\mathcal{I}$ is an $H\Sigma$-interpretation. Since $[\psi] \sqsubset_P [\varphi_1], \ldots, [\psi] \sqsubset_P [\varphi_n] \in \mathcal{D}_{\mathcal{P}}(n > 1)$ and $\mathcal{I} \models \forall(\varphi_1(\bar{\mu}_1)), \ldots, \mathcal{I} \models \forall(\varphi_n(\bar{\mu}_n))$, we can obtain $\mathcal{I} \models \forall(\sigma(\psi(\bar{\mu}')))$ by Lemma 3.11 (2). Then, by Lemma 3.1, $\mathcal{I} \models \forall((\sigma(\psi(\bar{\mu}')))\theta_1)$. Therefore $P \models_{H\Sigma} G\theta_1$ since $\psi(\bar{\mu})\theta_1 = (\sigma(\psi(\bar{\mu}')))\theta_1$.

Induction step: $n > 1$.

(a) If $G \xrightarrow{\theta_1}_{R1} G_1$, 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 $L \in G$ and $L' \leftarrow G' \in CS$ 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 $P \models_{H\Sigma} (L' \leftarrow G')\theta_1\theta'$. Therefore $P \models_{H\Sigma} G\theta_1\theta'$.

(b) If $G \xrightarrow{\theta_1}_{R2} G_1$, then

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

is a resolution of $G_1(= G'\theta_1 \cup (G - \{\psi(\bar{\mu})\})\theta_1)$ where $\psi(\bar{\mu}) \in G$ and $\varphi(\bar{\mu}') \leftarrow G' \in CS$, $(\sigma(\psi(\bar{\mu}')))\theta_1 = (\psi(\bar{\mu}))\theta_1$ and $[\varphi] \sqsubset_P [\psi] \in \mathcal{D}_{\mathcal{P}}$. By the induction hypothesis, $P \models_{H\Sigma} (G'\theta_1 \cup (G - \{\psi(\bar{\mu})\})\theta_1)\theta'$ with $\theta' = \theta_2 \cdots \theta_n$. Then $P \models_{H\Sigma} \varphi(\bar{\mu}')\theta_1\theta'$ since $P \models_{H\Sigma} (\varphi(\bar{\mu}') \leftarrow G')\theta_1\theta'$. By Definition 3.2, $P \models_{H\Sigma} (\sigma(\psi(\bar{\mu}'))\theta_1\theta' (= \psi(\bar{\mu})\theta_1\theta')$. Therefore $P \models_{H\Sigma} G\theta_1\theta'$.

(c) If $G \xrightarrow{\theta_1}_{R3} G_1$, then

$$(G_1 \cup \cdots \cup G_m)\theta_1 \cup (G - \{\psi(\bar{\mu})\})\theta_1 \xrightarrow{\theta_2} G_2 \xrightarrow{\theta_3} \cdots \xrightarrow{\theta_n} \square$$

is a resolution where $\psi(\bar{\mu}) \in G$, $\varphi_1(\bar{\mu}_1) \leftarrow G_1, \ldots, \varphi_m(\bar{\mu}_m) \leftarrow G_m \in CS$, $\psi(\bar{\mu})\theta_1 = (\sigma(\psi(\bar{\mu}')))\theta_1$ with $\mu' = \mu_1 \cup \cdots \cup \mu_m$ and $[\psi] \sqsubset_P [\varphi_1], \cdots, [\psi] \sqsubset_P [\varphi_m] \in \mathcal{D}_{\mathcal{P}}$. By the induction hypothesis, $P \models_{H\Sigma} ((G_1 \cup \cdots \cup G_m)\theta_1 \cup (G - \{\psi(\bar{\mu})\})\theta_1)\theta'$ with $\theta' = \theta_2 \cdots \theta_n$. Then, for $1 \leq i \leq m$, $P \models_{H\Sigma} \varphi_i(\bar{\mu}_i)\theta_1\theta'$ since $P \models_{H\Sigma} (\varphi_i(\bar{\mu}_i) \leftarrow G_i)\theta_1\theta'$. By Definition 3.2, $P \models_{H\Sigma} (\sigma(\psi(\bar{\mu}'))\theta_1\theta' (= \psi(\bar{\mu})\theta_1\theta)$. Therefore $P \models_{H\Sigma} G\theta_1\theta'$. ∎

Let $P = (\Sigma, CS)$ be a program and G a goal. We denote $(\Sigma, CS \cup \{G\})$ by $(\Sigma, CS) \cup G$. We use the abbreviation $L \longrightarrow\!\!\!\!\!\rightarrow G$ when we do not need to emphasize the substitution θ in $L \xrightarrow{\theta}\!\!\!\!\!\rightarrow G$. In order to make the Horn clause resolution complete, we supplement the following resolution rules.

Definition 3.36 *Let $P = (\Sigma, CS)$ be a program.*

(i) **R2$^+$-resolution rule.** *Let G be a goal. If $\langle\varphi\rangle = \langle\psi\rangle$, $[\varphi] \sqsubset_P [\psi] \in \mathcal{D}_\mathcal{P}$, and θ is a unifier of $\psi(\bar{\mu}) \in G$ and $\sigma(\psi(\bar{\mu}'))$, then $(G - \{\psi(\bar{\mu})\})\theta \cup \{\varphi(\bar{\mu}')\}\theta$ is an unstricted resolvent of G with respect to $\psi(\bar{\mu})$ and $\varphi(\bar{\mu}')$. We write*

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

(ii) **R3$^+$-resolution rule.** *Let G be a goal. If $\langle\varphi_1\rangle = \cdots = \langle\varphi_n\rangle = \langle\psi\rangle$ where $[\varphi_1], \ldots, [\varphi_n] (n > 1)$ are all predicates such that $[\psi] \sqsubset^1_P [\varphi_i] \in \mathcal{D}_\mathcal{P}$, and θ is a unifier of $\psi(\bar{\mu}) \in G$ and $\sigma(\psi(\bar{\mu}'))$ with $LS(\bar{\mu}') = \bigcup_{1 \le i \le n} ARG([\varphi_i])$, then $(G - \{\psi(\bar{\mu})\})\theta \cup \{\varphi_1(\bar{\mu}'_{\varphi_1}), \ldots, \varphi_n(\bar{\mu}'_{\varphi_n})\}\theta$ is an unstricted resolvent of G with respect to $\psi(\bar{\mu})$ and $\varphi_1(\bar{\mu}'_{\varphi_1}), \ldots, \varphi_n(\bar{\mu}'_{\varphi_n})$. We write*

$$G \xrightarrow{\theta}_{R3^+} (G - \{\psi(\bar{\mu})\})\theta \cup \{\varphi_1(\bar{\mu}'_{\varphi_1}), \ldots, \varphi_n(\bar{\mu}'_{\varphi_n})\}\theta.$$

We write $G \xrightarrow{\theta}_+ G'$ if $G \xrightarrow{\theta} G'$, $G \xrightarrow{\theta}_{R2^+} G'$, or $G \xrightarrow{\theta}_{R3^+} G'$. We use the notation $P{:}L \xrightarrow{\theta}\!\!\!\!\!\rightarrow_+ G$ to denote an unstricted resolution with the R2$^+$-resolution rule and R3$^+$-resolution rule. The soundness of the Horn clause resolution with the rules R2$^+$,R3$^+$ is proved as follows.

Theorem 3.6 (Soundness of resolution with R2$^+$, R3$^+$) *Let P be a program and G a goal. If there exists a successful resolution of G with computed answer substitution θ such that $P{:}L \xrightarrow{\theta}\!\!\!\!\!\rightarrow_+ G$, 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

$$P{:}G \xrightarrow{\theta_1}_+ G_1 \xrightarrow{\theta_2}_+ G_2 \xrightarrow{\theta_3}_+ \cdots \xrightarrow{\theta_n}_+ \Box$$

with $\theta = \theta_1 \cdots \theta_n$ be a successful resolution.
Base case: $n = 1$.

(a) If $G \xrightarrow{\theta_1}_{R1} G_1$, $G \xrightarrow{\theta_1}_{R2} G_1$, $G \xrightarrow{\theta_1}_{R3} G_1$, then $P \models_{H\Sigma} G\theta$ by the proof of Theorem 3.5.

(b) If $G \xrightarrow{\theta_1}_{R2^+} \Box$, then, by Definition 3.34, θ_1 is a unifier of $\psi(\bar{\mu})(= G)$ and $\sigma(\psi(\bar{\mu}'))$, i.e., $(\psi(\bar{\mu}))\theta_1 = (\sigma(\psi(\bar{\mu}')))\theta_1$. Suppose that $\mathcal{I} \models P$ where $\mathcal{I}$ is an $H\Sigma$-interpretation. We can get $\mathcal{I} \models \forall(\sigma(\psi(\bar{\mu}')))$ by Lemma 3.11 (1) since $[\varphi] \sqsubset_P [\psi] \in \mathcal{D}_\mathcal{P}$ and $\mathcal{I} \models \forall(\varphi(\bar{\mu}'))$. Then, by Lemma 3.1, $\mathcal{I}\forall((\sigma(\psi(\bar{\mu}')))\theta_1)$. Therefore, $P \models_{H\Sigma} G\theta_1$ since $(\psi(\bar{\mu}))\theta_1 = (\sigma(\psi(\bar{\mu}')))\theta_1$.

(c) If $G \xrightarrow{\theta_1}_{R3^+} \square$, then, by Definition 3.34, θ_1 is a unifier of $\psi(\bar{\mu})(= G)$ and $\sigma(\psi(\bar{\mu}'))$ with $\mu' = \mu_1 \cup \cdots \cup \mu_n$, i.e., $\psi(\bar{\mu})\theta_1 = (\sigma(\psi(\bar{\mu}')))\theta_1$. Suppose that $\mathcal{I} \models P$ where $\mathcal{I}$ is an $H\Sigma$-interpretation. Since $[\psi] \sqsubset_P [\varphi_1], \ldots, [\psi] \sqsubset_P [\varphi_n] \in \mathcal{D}_{\mathcal{P}}(n > 1)$ and $\mathcal{I} \models \forall(\varphi_1(\bar{\mu}_1)), \ldots, \mathcal{I} \models \forall(\varphi_n(\bar{\mu}_n))$, we can obtain $\mathcal{I} \models \forall(\sigma(\psi(\bar{\mu}')))$ by Lemma 3.11 (2). Then, by Lemma 3.1, $\mathcal{I} \models \forall((\sigma(\psi(\bar{\mu}')))\theta_1)$. Therefore $P \models_{H\Sigma} G\theta_1$ since $\psi(\bar{\mu})\theta_1 = (\sigma(\psi(\bar{\mu}')))\theta_1$.

Induction step: $n > 1$.

(a) If $G \xrightarrow{\theta_1}_{R1} G_1$, $G \xrightarrow{\theta_1}_{R2} G_1$, $G \xrightarrow{\theta_1}_{R3} G_1$, then $P \models_{H\Sigma} G\theta$ by the proof of Theorem 3.5.

(b) If $G \xrightarrow{\theta_1}_{R2^+} G_1$, then

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

is a resolution of $G_1(= (G - \{\psi(\bar{\mu})\})\theta_1 \cup \{\varphi(\bar{\mu}')\}\theta_1)$ where $\psi(\bar{\mu}) \in G$, $\sigma(\psi(\bar{\mu}'))\theta_1 = \psi(\bar{\mu})\theta_1$, and $[\varphi] \sqsubset_P [\psi] \in \mathcal{D}_{\mathcal{P}}$. By the induction hypothesis, $P \models_{H\Sigma} ((G - \{\psi(\bar{\mu})\})\theta_1 \cup \{\varphi(\bar{\mu}')\}\theta_1)\theta'$ with $\theta' = \theta_2 \cdots \theta_n$. By Definition 3.2, $P \models_{H\Sigma} (\sigma(\psi(\bar{\mu}')))\theta_1\theta'(= \psi(\bar{\mu})\theta_1\theta')$. Therefore $P \models_{H\Sigma} G\theta_1\theta'$.

(c) If $G \xrightarrow{\theta_1}_{R3^+} G_1$, then

$$(G - \{\psi(\bar{\mu})\})\theta_1 \cup \{\varphi_1(\bar{\mu}'_{\varphi_1}), \ldots, \varphi_m(\bar{\mu}'_{\varphi_m})\}\theta_1 \xrightarrow{\theta_2}_{+} G_2 \xrightarrow{\theta_3}_{+} \cdots \xrightarrow{\theta_n}_{+} \square$$

is a resolution where $\psi(\bar{\mu}) \in G$, $\psi(\bar{\mu})\theta_1 = \sigma(\psi(\bar{\mu}'))\theta_1$ with $LS(\bar{\mu}') = \bigcup_{1 \le i \le n} ARG([\varphi_i])$, and $[\psi] \sqsubset_P [\varphi_1], \cdots, [\psi] \sqsubset_P [\varphi_m] \in \mathcal{D}_{\mathcal{P}}$. By the induction hypothesis, $P \models_{H\Sigma} ((G - \{\psi(\bar{\mu})\})\theta_1 \cup \{\varphi_1(\bar{\mu}'_{\varphi_1}), \ldots, \varphi_m(\bar{\mu}'_{\varphi_m})\}\theta_1)\theta'$ with $\theta' = \theta_2 \cdots \theta_n$. By Definition 3.2, $P \models_{H\Sigma} (\sigma(\psi(\bar{\mu}'))\theta_1\theta'(= \psi(\bar{\mu})\theta_1\theta)$. Therefore $P \models_{H\Sigma} G\theta_1\theta$. ∎

By the completeness of the Horn clause calculus and the following two lemmas, we will prove the completeness of the Horn clause resolution.

Lemma 3.12 *Let P be a program and G a goal. $P{:}\, L \longrightarrow\!\!\!\!\!\rightarrow_+ \square$ for all $L \in G$ if and only if $P{:}\, G \longrightarrow\!\!\!\!\!\rightarrow_+ \square$.*

Proof. ($\Leftarrow$) Trivial.
($\Rightarrow$) By the hypothesis, the resolution $P{:}\, G \longrightarrow\!\!\!\!\!\rightarrow_+ \square$ can be built. ∎

Lemma 3.13 *Let $P = (\Sigma, CS)$ be a program and $L \leftarrow G$ a ground clause. If $P \vdash L \leftarrow G$, then $P{:}\, G \longrightarrow\!\!\!\!\!\rightarrow_+ \square$ implies $P{:}\, L \longrightarrow\!\!\!\!\!\rightarrow_+ \square$.*

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

(a) If $L \leftarrow G$ is derived by the cut rule, then

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

where $G = G' \cup G''$. By the induction hypothesis, $P{:}\, G' \cup \{L'\} \longrightarrow\!\!\!\!\!\rightarrow_+ \Box$ implies $P{:}\, L \longrightarrow\!\!\!\!\!\rightarrow_+ \Box$ and $P{:}\, G'' \longrightarrow\!\!\!\!\!\rightarrow_+ \Box$ implies $P{:}\, L' \longrightarrow\!\!\!\!\!\rightarrow_+ \Box$. If $P{:}\, G' \cup G'' \longrightarrow\!\!\!\!\!\rightarrow_+ \Box$, then $P{:}\, G' \cup \{L'\} \longrightarrow\!\!\!\!\!\rightarrow_+ \Box$. Therefore $P{:}\, L \longrightarrow\!\!\!\!\!\rightarrow_+ \Box$.

(b) If $L \leftarrow G$ with $L = \psi(\bar{\mu})$ is derived by the generalization rule for event predicates (or property predicates), then

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

where $[\varphi] \sqsubset_P [\psi]$ and $(\sigma(\psi(\bar{\mu}')))\theta = \psi(\bar{\mu})$. By the induction hypothesis, $P{:}\, G \longrightarrow\!\!\!\!\!\rightarrow_+ \Box$ implies $P{:}\, \varphi(\bar{\mu}') \longrightarrow\!\!\!\!\!\rightarrow_+ \Box$. Then, we can obtain the resolution $P{:}\, \psi(\bar{\mu}) \xrightarrow{\theta}_{R2^+} \varphi(\bar{\mu}')$ since $\varphi(\bar{\mu}')$ is ground. Hence if $P{:}\, G \longrightarrow\!\!\!\!\!\rightarrow_+ \Box$, then $P{:}\, \psi(\bar{\mu}) \xrightarrow{\theta}_{R2^+} \varphi(\bar{\mu}') \longrightarrow\!\!\!\!\!\rightarrow_+ \Box$.

(c) If $L \leftarrow G$ with $L = \psi(\bar{\mu}_0)$ is derived by the specialization rule for event predicates (or property predicates), then

$$P \vdash \varphi_1(\bar{\mu}_{\varphi_1}) \leftarrow G_1, \ldots, P \vdash \varphi_m(\bar{\mu}_{\varphi_m}) \leftarrow G_m$$

where $[\varphi_1], \ldots, [\varphi_n]$ are all predicates such that $[\psi] \sqsubset^1_P [\varphi_i]$, $\mu = \{a_1 \Rightarrow t_1, \ldots a_m \Rightarrow t_m\}$ such that $\bigcup_{1 \le i \le m} ARG([\bar{\mu}_{\varphi_i}]) = \{a_1, \ldots, a_k\}$ and $t_i \in TERM_{0,SCP(a_i)}$, and $\psi(\bar{\mu}_0) = (\sigma(\psi(\bar{\mu})))\theta$. By the induction hypothesis, for $1 \le i \le m$, $P{:}\, G_i \longrightarrow\!\!\!\!\!\rightarrow_+ \Box$ implies $P{:}\, \varphi_i(\bar{\mu}_{\varphi_i}) \longrightarrow\!\!\!\!\!\rightarrow_+ \Box$. Then we can obtain the resolution $P{:}\, \psi(\bar{\mu}_0) \xrightarrow{\theta}_{R3^+} \{\varphi_1(\bar{\mu}_{\varphi_1}), \ldots, \varphi_m(\bar{\mu}_{\varphi_m})\}$ since $\varphi_1(\bar{\mu}_{\varphi_1}), \ldots, \varphi_m(\bar{\mu}_{\varphi_m})$ are ground. Therefore, if $P{:}\, G_1 \cup \ldots \cup G_m \longrightarrow\!\!\!\!\!\rightarrow_+ \Box$, then $P{:}\, \psi(\bar{\mu}_0) \xrightarrow{\theta}_{R3^+} \{\varphi_1(\bar{\mu}_{\varphi_1}), \ldots, \varphi_m(\bar{\mu}_{\varphi_m})\} \longrightarrow\!\!\!\!\!\rightarrow_+ \Box$. ∎

We prove the completeness of the Horn clause resolution for ground goals.

Theorem 3.7 (Ground completeness of resolution with R2$^+$, R3$^+$)
Let P be a program and G a goal. If G is a ground goal such that $P \models_{H\Sigma} G$, then there exists a successful resolution of G with respect to P such that $P{:}\, L \longrightarrow\!\!\!\!\!\rightarrow_+ G$.

Proof. Suppose that $P \models_{H\Sigma} G(= L_1, \ldots, L_n)$ where G is a ground goal. Then $P \models_{H\Sigma} L_1, \ldots, P \models_{H\Sigma} L_n$. By Theorem 3.3, we have $P \vdash L_1, \ldots, P \vdash L_n$. By Lemma 3.13, $P{:}\, L_1 \longrightarrow\!\!\!\!\!\rightarrow_+ \Box, \ldots, P{:}\, L_n \longrightarrow\!\!\!\!\!\rightarrow_+ \Box$. Therefore, by Lemma 3.12, $P{:}\, G \longrightarrow\!\!\!\!\!\rightarrow_+ \Box$. ∎

The following lemma guarantees that there are resolvents for unstricted resolvents.

Lemma 3.14 *Let G be a goal. If G has an unrestricted resolvent with respect to $L \in G$ and a clause C (or clauses $C_1, \ldots, C_n$) such that $L \xrightarrow{\theta}_+ G$, then G has a resolvent with respect to them.*

Proof. By the hypothesis, there is an unstricted resolvent G' of goal G such that $L \xrightarrow{\theta}_+ G$.

(a) If $G \xrightarrow{\theta}_{R1} G'$ with $G' = G_0\theta \cup (G - \{L\})\theta$, then $L \in G$, $L_0 \leftarrow G_0 \in CS$, and $L\theta = L_0\theta$. Let θ' be a most general unifier of L and L_0. We have a resolvent $G \xrightarrow{\theta'}_{R1} G_0\theta' \cup (G - \{L\})\theta'$ with respect to L and $L_0 \leftarrow G_0$.

(b) If $G \xrightarrow{\theta_1}_{R2} G'$ with $G' = G_0\theta \cup (G - \{\psi(\bar{\mu})\})\theta$, then $\psi(\bar{\mu}) \in G$, $\varphi(\bar{\mu}') \leftarrow G_0 \in CS$, $(\sigma(\psi(\bar{\mu}')))\theta = \psi(\bar{\mu})\theta$, and $[\varphi] \sqsubset_P [\psi] \in \mathcal{D}_\mathcal{P}$. Let θ' be a most general unifier of $\psi(\bar{\mu})$ and $\sigma(\psi(\bar{\mu}'))$. There exists a resolvent $G \xrightarrow{\theta'}_{R2} G_0\theta' \cup (G - \{\psi(\bar{\mu})\})\theta'$ with respect to $\psi(\bar{\mu})$ and $\varphi(\bar{\mu}') \leftarrow G_0$.

(c) If $G \xrightarrow{\theta}_{R3} G'$ with $G' = (G_1 \cup \cdots \cup G_m)\theta \cup (G - \{\psi(\bar{\mu})\})\theta$, then $\psi(\bar{\mu}) \in G$, $\varphi_1(\bar{\mu}_1) \leftarrow G_1, \ldots, \varphi_n(\bar{\mu}_n) \leftarrow G_n \in CS$, and $\psi(\bar{\mu})\theta = (\sigma(\psi(\bar{\mu}')))\theta$ with $\mu' = \mu_1 \cup \cdots \cup \mu_m$, $[\psi] \sqsubset_P [\varphi_1], \ldots, [\psi] \sqsubset_P [\varphi_m]$. Let θ' be a most general unifier of $\psi(\bar{\mu})$ and $\sigma(\psi(\bar{\mu}'))$. We have a resolvent $G \xrightarrow{\theta'}_{R3} (G_1 \cup \cdots \cup G_m)\theta' \cup (G - \{\psi(\bar{\mu})\})\theta'$ with respect to $\psi(\bar{\mu})$ and $\varphi_1(\bar{\mu}_1) \leftarrow G_1, \ldots, \varphi_n(\bar{\mu}_n) \leftarrow G_n$.

(d) If $G \xrightarrow{\theta}_{R2^+} G'$ or $G \xrightarrow{\theta}_{R3^+} G'$, then similar to (b) and (c). ∎

The following two lemmas are needed to prove the completeness of the Horn clause resolution for general goals.

Lemma 3.15 (One step lifting) *Let G_0, G, G' be goals. If G is an unstricted resolvent of $G_0\theta_0$ with $G_0\theta_0 \xrightarrow{\theta}_+ G$, then G' is a resolvent of G_0 with $G_0 \xrightarrow{\theta'}_+ G$ where there exists a substitution γ such that $(\theta_0{\uparrow}CVar(G_0))\theta = \gamma\theta'$, and $G = G'\gamma$.*

Proof. Assume that $G_0\theta_0 \xrightarrow{\theta}_+ G$ with respect to $L \in G_0\theta_0$ and clauses $C_1, \ldots, C_m$. Since $CVar(G_0) \cap CVar(C_1) \cap \ldots \cap CVar(C_m) = \emptyset$, we have $C_i(\theta_0{\uparrow}CVar(G_0)) = C_i$ for $1 \leq i \leq m$. Hence, $G_0 \xrightarrow{(\theta_0{\uparrow}CVar(G_0))\theta}_+ G$. By Lemma 3.14, G_0 has a resolvent G' (with respect to L and $C_1, \ldots, C_m$) with $G_0 \xrightarrow{\theta'}_+ G'$ where there exists a substitution γ such that $(\theta_0{\uparrow}CVar(G_0))\theta = \theta'\gamma$, and $G = G'\gamma$. ∎

Lemma 3.16 (Lifting) *Let P be a program. If P has an unstricted resolution*

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

then P has a resolution

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

where $\gamma_0 = \theta_0$ and, for $1 \leq i \leq n$, there exists a substitution γ_i such that $(\gamma_{i-1} \uparrow CVar(G_{i-1}))\theta_i = \theta'_i\gamma_i$, and $G_i = G'_i\gamma_i$.

Proof. This lemma is proved by induction on the length n of an unstricted resolution of $G_0\theta_0$.
$n = 1$: By Lemma 3.15, this is clear.
$n > 1$: By the induction hypothesis, there exists a resolution

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

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 Lemma 3.15 and $P\colon G_{n-1} \xrightarrow{\theta_n}_+ G_n$ with $G_{n-1} = G'_{n-1}\gamma_{i-1}$, we have a resolution $P\colon G'_{n-1} \xrightarrow{\theta'_n}_+ G'_n$ where there exists a substitution γ_n such that $(\gamma_{n-1}\uparrow CVar(G_{n-1}))\theta_n = \theta'_n\gamma_n$, and $G_n = G'_n\gamma_n$. ∎

The next lemma shows the completeness of the Horn clause resolution with a ground substitution θ_0 for goal G_0.

Lemma 3.17 *Let P be a program, G_0 a goal, and θ_0 a ground substitution for G_0. If G_0 is a goal such that $P \models_{H\Sigma} G_0\theta_0$, then there exists a successful resolution of G_0 with a computed answer substitution θ' such that $P\colon L \xrightarrow{\theta'}\!\!\!\!\!\rightarrow_+ G$ and $G_0\theta_0 = G_0\theta'\gamma$.*

Proof. Suppose $P \models_{H\Sigma} G_0\theta_0$. Then, by Theorem 3.7, $P\colon G_0\theta_0 \xrightarrow{\theta}\!\!\!\!\!\rightarrow_+ \square$ with $\theta = \theta_1 \cdots \theta_n$. Hence, by Lemma 3.16, $P\colon G_0 \xrightarrow{\theta'}\!\!\!\!\!\rightarrow_+ \square$ with $\theta' = \theta'_1 \cdots \theta'_n$ where $\gamma_0 = \theta_0$ and, for $1 \leq i \leq n$, there exists a substitution γ_i such that $(\gamma_{i-1} \uparrow CVar(G_{i-1}))\theta_i = \theta'_i\gamma_i$. Then we have $G\theta_0\theta = G\theta'\gamma$ by the proof of Theorem 5.37 in [16]. Since $\theta\uparrow CVar(G\theta) = \emptyset$ (by the fact that θ is a ground substitution for G), we can obtain $G_0\theta_0\theta = G_0\theta_0$. Therefore $\theta'\uparrow CVar(G)$ is a computed answer substitution such that $G\theta_0 = G\theta'\gamma$. ∎

In the following theorem, we shows that the Horn clause resolution with the $R2^+$-resolution rule and $R3^+$-resolution rule is complete.

Theorem 3.8 (Completeness of resolution with R2$^+$, R3$^+$) *If $P \models_{H\Sigma} G\theta$ where θ is a sorted substitution, then there exists a successful resolution of G with a computed answer substitution θ' such that $P\colon L \xrightarrow{\theta'}\!\!\!\!\!\rightarrow_+ G$ and $G\theta = G\theta'\gamma$.*

Proof. Let $\beta = \{c_{x_1:s_1}\colon s_1/x_1\colon s_1, \ldots, c_{x_n:s_n}\colon s_n/x_n\colon s_n\}$ where $CVar(G\theta) = \{x_1\colon s_1, \ldots, x_n\colon s_n\}$ and $c_{x_1:s_1}\colon s_1, \ldots, c_{x_n:s_n}\colon s_n$ are new constants. If $P \models_{H\Sigma} G\theta$, then $P \models_{H\Sigma} G\theta\beta$. By Lemma 3.17, there exists a successful resolution $G \longrightarrow\!\!\!\!\!\rightarrow_+ \square$ with a computed answer substitution θ' such that $G\theta\beta = G\theta'\gamma$. Because the new constants $c_{x_1:s_1}\colon s_1, \ldots, c_{x_n:s_n}\colon s_n$ do not occur in $G\theta'$, we have $c_{x_1:s_1}/y_1\colon s_1, \ldots, c_{x_n:s_n}/y_n\colon s_n \in \gamma$. 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 $\gamma' = (\gamma - \{c_{x_1:s_1}/y_1\colon s_1, \ldots, c_{x_n:s_n}/y_n\colon s_n\}) \cup \gamma_0$. Let $G\theta = G'[x_1\colon s_1, \ldots, x_n\colon s_n]$. We have $G\theta'\gamma' = G'[x_1\colon s_1, \ldots, x_n\colon s_n] = G\theta$. Therefore, $G\theta = G\theta'\gamma'$. ∎

Let us look at resolution processes with respect to the two examples (Example 1 and Example 2) we saw in Section 2.1.3. The sort-hierarchies 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 3.14 *The sorted signature Σ_1 comprises the following symbols*

$$\begin{aligned} \mathcal{S} &= \{person, man, woman, \top, \bot\}, \\ \mathcal{F} &= \{john, mary, c\}, \\ \mathcal{P} &= \{rob\_with\_violence, hit, steal, illegal\_act\}, \end{aligned}$$

and the declaration $\mathcal{D} = (\mathcal{D}_{\mathcal{S}}, \mathcal{D}_{\mathcal{F}}, \mathcal{D}_{\mathcal{P}})$ *constructed by*

$$\begin{aligned}
\mathcal{D}_{\mathcal{S}} &= \{man \sqsubset_S person, woman \sqsubset_S person, wallet \sqsubset_S thing,\\
&\quad \bot \sqsubset_S man, \bot \sqsubset_S woman, \bot \sqsubset_S wallet,\\
&\quad person \sqsubset_S \top, thing \sqsubset_S \top\ \},\\
\mathcal{D}_{\mathcal{F}} &= \{john{:}\rightarrow man, mary{:}\rightarrow woman, c{:}\rightarrow wallet\},\\
\mathcal{D}_{\mathcal{P}} &= \{hit \sqsubset_P illegal_act\} \cup\\
&\quad \{hit{:}\{(agt, person)\},\\
&\quad illegal_act{:}\{(agt, person), (coagt, person)\}\ \}.
\end{aligned}$$

The argument structure of hit consists of one argument labeled with agt*, and the argument structure of* $illegal\_act$ *consists of two arguments labeled with* $agt, coagt$*. The program* P_1 *is the ordered pair* (Σ_1, CS_1) *with*

$$CS_1 = \{hit^{\bullet}(agt \Rightarrow john{:}man)\}.$$

$hit^{\bullet}(agt \Rightarrow john{:}man)$ *means that "the agent john hit." With respect to the program, the resolution of the goal* $illegal\_act^{\bullet}(agt \Rightarrow john{:}man,\ coagt \Rightarrow mary{:}woman)$ *fails as follows*

$$illegal\_act^{\bullet}(agt \Rightarrow john{:}man, coagt \Rightarrow mary{:}woman) \longrightarrow fail,$$

but we have a resolution of the goal $illegal\_act^{\bullet}(agt \Rightarrow john{:}man, coagt \Rightarrow y{:}person)$ *as follows*

$$illegal\_act^{\bullet}(agt \Rightarrow john{:}man, coagt \Rightarrow y{:}person) \xrightarrow{\theta}_{R2} \Box$$

where $\theta = \{c_{coagt}{:}person/y{:}person\}$.

Furthermore, the sorted signature $\Sigma_1' = (\mathcal{S}, \mathcal{F}, \mathcal{P}, \mathcal{D}')$ *is obtained by replacing* $\mathcal{D}$ *in* Σ_1 *with* $\mathcal{D}' = (\mathcal{D}_{\mathcal{S}}, \mathcal{D}_{\mathcal{F}}, \mathcal{D}_{\mathcal{P}}')$ *where*

$$\begin{aligned}
\mathcal{D}_{\mathcal{P}}' &= \{rob_with_violence \sqsubset_P hit,\\
&\quad rob_with_violence \sqsubset_P steal,\\
&\quad hit \sqsubset_P illegal_act,\\
&\quad steal \sqsubset_P illegal_act\} \cup\\
&\quad \{rob_with_violence{:}\{(agt, person), (coagt, person), (obj, thing)\},\\
&\quad hit{:}\{(agt, person), (coagt, person)\},\\
&\quad steal{:}\{(agt, person), (obj, \top)\},\\
&\quad illegal_act{:}\{(agt, person)\}\ \}.
\end{aligned}$$

The argument structure of $rob\_with\_violence$ *consists of three arguments labeled with* agt*,* $coagt$*,* obj*, and the argument structure of* $illegal\_act$ *consists of one argument labeled with* agt*. The argument structures of hit and steal consist of two arguments labeled with* $agt, coagt$ *and* agt, obj *respectively. The program* P_1' *is the ordered pair* (Σ_1', CS_1') *with*

$$\begin{aligned}
CS_1' &= \{hit^{\bullet}(agt \Rightarrow john{:}man, coagt \Rightarrow mary{:}woman),\\
&\quad steal^{\bullet}(agt \Rightarrow john{:}man, obj \Rightarrow c{:}wallet)\}.
\end{aligned}$$

$hit^{\bullet}(agt \Rightarrow john{:}man, coagt \Rightarrow mary{:}woman)$ *means that "the agent john hit the coaent mary," and* $steal^{\bullet}(agt \Rightarrow john{:}man, obj \Rightarrow c{:}wallet)$ *means that "the agent john stolen the wallet c." With respect to the program, we have the following resolutions*

$$illegal\_act^{\bullet}(agt \Rightarrow john{:}man) \xrightarrow{\epsilon}_{R2} \Box,$$

$$rob\_with\_violence^{\bullet}(agt \Rightarrow john{:}man, coagt \Rightarrow mary{:}woman, obj \Rightarrow c{:}wallet) \xrightarrow{\epsilon}_{R3} \Box,$$

$$rob\_with\_violence^{\bullet}(agt \Rightarrow x{:}person, coagt \Rightarrow y{:}person, obj \Rightarrow z{:}thing) \xrightarrow{\theta}_{R3} \Box$$

where $\theta = \{john{:}man/x{:}person, mary{:}woman/y{:}person, c{:}wallet/z{:}thing\}$.

Example 3.15 *The sorted signature* Σ_2 *comprises the following symbols*

$$\begin{aligned}\mathcal{S} &= \{penguin, crow, bird, animal, \top, \bot\},\\ \mathcal{F} &= \{c\},\\ \mathcal{P} &= \{fly, move, walk\},\end{aligned}$$

and the declaration $\mathcal{D} = (\mathcal{D}_{\mathcal{S}}, \mathcal{D}_{\mathcal{F}}, \mathcal{D}_{\mathcal{P}})$ *constructed by*

$$\begin{aligned}\mathcal{D}_{\mathcal{S}} &= \{penguin \sqsubset_S bird, crow \sqsubset_S bird, bird \sqsubset_S animal,\\ &\quad \bot \sqsubset_S penguin, \bot \sqsubset_S crow, \bot, animal \sqsubset_S \top\},\\ \mathcal{D}_{\mathcal{F}} &= \{c{:}\rightarrow bird\},\\ \mathcal{D}_{\mathcal{P}} &= \{fly \sqsubset_P move,\\ &\quad walk \sqsubset_P move\} \cup\\ &\quad \{fly{:}\{(sbj, \top)\},\\ &\quad walk{:}\{(sbj, \top)\},\\ &\quad move{:}\{(sbj, \top)\}\ \}.\end{aligned}$$

The program P_2 *is the ordered pair* (Σ_2, CS_2) *with*

$$\begin{aligned}CS_2 &= \{fly^{\bullet}(sbj \Rightarrow c{:}bird),\\ &\quad\ fly^{\sharp}(sbj \Rightarrow x{:}bird)\}.\end{aligned}$$

$fly^{\bullet}(sbj \Rightarrow c{:}bird)$ *means that "a bird* c *is filing," and* $fly^{\sharp}(sbj \Rightarrow x{:}bird)$ *means that "all birds have the property of fight" (or "all birds can fly"). With respect to the program, a successful resolution of the goal* $move^{\bullet}(sbj \Rightarrow y{:}animal)$ *(with the event predicate* $move^{\bullet}$*) is derived as follow*

$$fly^{\bullet}(sbj \Rightarrow y{:}animal) \xrightarrow{\theta_1}_{R1} \Box$$

where $\theta_1 = \{c{:}bird/y{:}animal\}$*, but the resolution of the goal* $move^{\bullet}(sbj \Rightarrow y{:}penguin)$ *fails as follows*

$$move^{\bullet}(sbj \Rightarrow y{:}penguin) \longrightarrow_{R2} fail.$$

And a successful resolution of the goal $move^{\bullet}(sbj \Rightarrow y{:}animal)$ *is derived as follows*

$$move^{\bullet}(sbj \Rightarrow y{:}animal) \xrightarrow{\theta_3}_{R2} \Box$$

where $\theta_3 = \{c{:}bird/y{:}animal\}$.

In contrast to the above goals with event predicates, we consider resolutions of goals with property predicates. A successful resolution of the goal $fly^{\sharp}(sbj \Rightarrow y{:}animal)$ *(with the property predicate* $fly^{\sharp}$*) is obtained as follow*

$$fly^\sharp(sbj \Rightarrow y{:}animal) \xrightarrow{\theta'_1}_{R1} \square$$

where $\theta'_1 = \{x{:}bird/y{:}animal\}$*, a successful resolution of the goal* $move^\sharp(sbj \Rightarrow y{:}penguin)$ *is obtained as follow*

$$move^\sharp(sbj \Rightarrow y{:}penguin) \xrightarrow{\theta'_2}_{R2} \square$$

where $\theta'_2 = \{y{:}penguin/x{:}bird\}$*, and a successful resolution of the goal* $move^\sharp(sbj \Rightarrow y{:}animal)$ *is obtained as follow*

$$move^\sharp(sbj \Rightarrow y{:}animal) \xrightarrow{\theta'_3}_{R2} \square$$

where $\theta'_3 = \{x{:}bird/y{:}animal\}$.

However, we do not require that there are successful resolutions of $fly^\sharp(sbj \Rightarrow y{:}animal)$ and $move^\sharp(sbj \Rightarrow y{:}animal)$ because we have contended that their answers should be *no*. In the next section, we will present a query system that is based on the Horn clause resolution with hierarchies and eventuality, which solves this problem.

3.4 Evaluation

In Section 3.4, we evaluate the capability of our logic as a knowledge representation system. Hence, we show that a query system defined by the resolution we develop provides the reasoning mechanism required in our motivational examples given in Chapter 2, and give an example of its application to a legal reasoning.

3.4.1 Query system

As we have mentioned in Section 2.1.3, we are finally concerned with a knowledge representation system (called a query system) that can be asked questions about a knowledge base (given as a program). A PROLOG-like logic programming language is regarded as a kind of query system. If a user inputs a query

```
?-p1(args_1),...,pn(args_n).
```

where `args_i` is a sequence $(t_1, \ldots, t_n)$ of arguments, then the answer `yes` or `no` must be returned. In the Horn clause resolution proposed in the previous section, the goal G corresponds to a query `?-Q` where both G and `Q` are formed by a sequence $(L_1, \ldots, L_n)$ of atoms. Furthermore, if the goal G and the query `?-Q` have the same logical meaning, then we can directly define our expected query system by the Horn clause resolution. However, the query system in our motivational examples quantifies each variable (occurring in the queries) alternatively as existential or universal.

In Example 2. in Section 2.1.3, the query

```
?-fly#(sbj=>X:bird).
```

with a property predicate (where we denote event and property predicates in queries by `p*` and `p#` respectively) indicates "Do all birds have the property of flight?" or "Can all birds fly?" In contrast, the query

```
?-fly*(sbj=>X:bird).
```

with an event predicate indicates that "Is a bird flying?" Clearly, the first and second queries include differently quantified variables where an existential variable occurs in event atoms and a universal variable in property atoms. However, the goal G used in resolutions seems to be a query with existential variables, since we can obtain $P \models_{H\Sigma} G\theta$ but not $P \models_{H\Sigma} G$ if there exists a successful resolution of goal G with a computed answer substitution θ.

Thus, we will present a query system based on the Horn clause resolution for extended order-sorted logic with hierarchies and eventuality. It not only gives the answer `yes` or `no` to a user's question with respect to a program P but also allows us to express query terms (or formulas) that distinguish between existential variables $x{:}\, s$ and universal variables $X{:}\, s$. In the following, we introduce the language $\mathcal{L}_Q$ for a query system that extends an order-sorted first-order language $\mathcal{L}$. $\mathcal{V}_{Q,s}$ is a set of universal variables $X_1{:}\, s, X_2{:}\, s, \ldots, X_n{:}\, s$ of sort s where $\mathcal{V}_{Q,s}$ and $\mathcal{V}_s$ (regarded as a set of existential variables) are disjoint. We denote by V_Q the union of sets $\mathcal{V}_{Q,s_1}, \mathcal{V}_{Q,s_2}, \ldots, \mathcal{V}_{Q,s_n}$ for all sorts $s_1, s_2, \ldots, s_n$.

Definition 3.37 (Query language) *Given an order-sorted first-order language $\mathcal{L}$, the query language $\mathcal{L}_Q$ is constructed by adding $\mathcal{V}_Q$.*

In the same manner as sorted terms, we build query terms by adding universal variables in which the existing variables in $\mathcal{L}$ are regarded as existential variables.

Definition 3.38 (Query terms) *Let $\Sigma = (\mathcal{S}, \mathcal{F}, \mathcal{P}, \mathcal{D})$ be a sorted signature with hierarchical predicates. The set $QTERM_s$ of query terms of sort s is defined by:*

(1) An existential variable $x{:}\, s \in V_s$ is a term of sort s,

(2) A universal variable $X{:}\, s \in V_{Q,s}$ is a term of sort s,

(3) A constant $c{:}\, s$ is a term of sort s where $c \in \mathcal{F}_0$ and $c{:} \to s \in \mathcal{D}_\mathcal{F}$,

(4) If $t_1, \ldots, t_n$ are terms of sorts $s_1, \ldots, s_n$, then $f(t_1, \ldots, t_n){:}\, s$ is a term of sort s where $f \in \mathcal{F}_n$ and $f{:}\, s_1 \times \ldots \times s_n \to s \in \mathcal{D}_\mathcal{F}$,

(5) If t is a term of sort s', then t is a term of sort s where $s' \sqsubset_S s \in \mathcal{D}_\mathcal{S}$.

Remark 3.3 *The existential and universal variables are distinctively defined in the query terms, but treated equally in resolution processes for our order-sorted logic. Thus, if these variables are free variables, then they can be equally substituted for sorted terms (or query terms).*

We denote by *QTERM* the set $\bigcup_{s\in\mathcal{S}} QTERM_s$ of all query terms. The set *QFORM* of all query formulas is inductively defined by the terms in *QTERM* in the same way that *FORM* was constructed from *TERM*. We use query formulas to express goals G or substituted clauses C but not clauses in a program P. We call a clause form expressed by query atoms *a query clause* and write *QCL* for the set of all query clauses.

In the following, we define the function *UVar* as assigning any query term to the set of universal variables occurring in it.

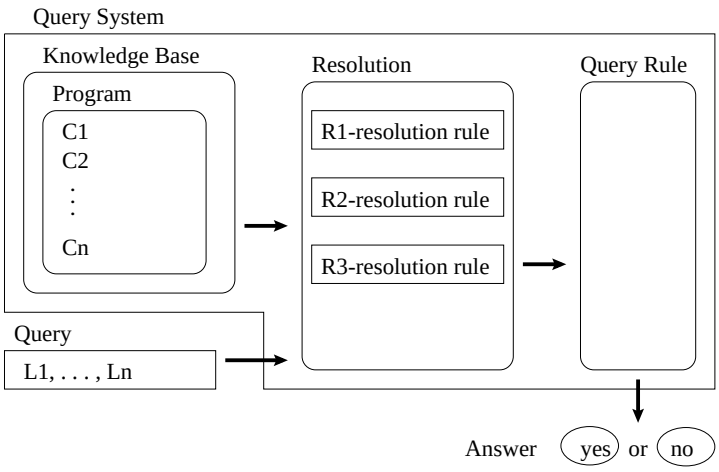

Figure 3.3 System components

Definition 3.39 *The function UVar from the set QTERM of query terms into* $2^{\mathcal{V}_Q}$ *is defined by:*

(i) $UVar(x{:}\, s) = \emptyset$,

(ii) $UVar(X{:}\, s) = \{X{:}\, s\}$,

(iii) $UVar(f(t_1, \ldots, t_n){:}\, s) = UVar(t_1) \cup \cdots \cup UVar(t_n)$.

The function $FVar{:}\, FORM_{\Sigma^+} \to 2^{\mathcal{V}}$ is defined by Var in Definition 3.7. Similarly, the function $FUVar{:}\, QFORM \to 2^{\mathcal{V}_Q}$ is defined by $UVar$ and the function $CUVar{:}\, QCL \to 2^{\mathcal{V}_Q}$ by $FUVar$. Moreover, the function $EVar$ (assigning any query term to the set of existential variables occurring in it) is defined by $EVar(t) = Var(t) - UVar(t)$. We define $FEVar(E) = FVar(E) - FUVar(F)$ and $CEVar(E) = CVar(E) - CUVar(F)$. We develop a query system with a query language $\mathcal{L}_Q$ (where the language contains existential and universal variables) as follows.

Definition 3.40 (Query system) *Let P be a program and let G be a goal. The query system Query is defined by the following query rule.*

(i) If there exists a successful resolution $P{:}\, G \overset{\theta}{\longrightarrow\!\!\!\!\!\rightarrow} \square$ *and* $Dom(\theta) \cap CUVar(G) = \emptyset$, *then* $Query(G) = yes$.

(ii) Otherwise, $Query(G) = no$.

The components of the above query system are described in Figure 3.3.

Remark 3.4 *In our query system, a query with existential variables* $x{:}\, s$ *implies inquiring whether the statement holds for an instance (corresponding to a ground term), and a query with universal variables* $X{:}\, s$ *implies inquiring whether the statement holds for all instances (corresponding to a free variable). The following theorem includes this consideration.*

Theorem 3.9 (Correctness of query system) *Let P be a program and let G be a goal as a query formula. Then, the following holds:*

- *If* $Query(G) = yes$*, then there exists a ground substitution* θ *for* $CEVar(G)$ *with* $Dom(\theta) \cap CUVar(G) = \emptyset$ *and* $P \models_{H\Sigma} G\theta$*.*
- *If* $Query(G) = no$*, then* $P \not\models_{H\Sigma} G\theta$ *for all ground substitutions* θ *for* $CEVar(G)$ *with* $Dom(\theta) \cap CUVar(G) = \emptyset$*.*

Proof. Suppose that $Query(G) = yes$. By Definition 3.40, we can obtain a successful resolution $P{:}G \xrightarrow{\theta}\!\!\!\!\rightarrow \Box$ with $Dom(\theta) \cap CUVar(G) = \emptyset$. Hence, by Theorem 3.5, $P \models_{H\Sigma} G\theta$. We define a ground substitution θ' for $CEVar(G)$ where $\theta \leq \theta'$ (θ is more general than θ') and $Dom(\theta') \cap CUVar(G) = \emptyset$. By Lemma 3.1 and $\theta \leq \theta'$, $P \models_{H\Sigma} G\theta'$.

Suppose that $P \models_{H\Sigma} G\theta$ where θ is a ground substitution for $CEVar(G)$ with $Dom(\theta) \cap CUVar(G) = \emptyset$. By Theorem 3.8, we have a successful resolution of goal G with a computed answer substitution θ' such that $G\theta = G\theta'\gamma$. By the hypothesis ($Dom(\theta) \cap CUVar(G) = \emptyset$) and $\theta' \uparrow CVar(G) \leq \theta \uparrow CVar(G)$, $Dom(\theta') \cap CUVar(G) = \emptyset$. Therefore, by Definition 3.40, $Query(G) = yes$. ∎

The following two examples give the answers to queries for the programs in Example 3.14 and Example 3.15.

Example 3.16 *Given the program* $P_1 = (\Sigma_1, CS_1)$ *in Example 3.14, we can obtain the following answers*

$$Query(illegal\_act^{\bullet}(agt \Rightarrow john{:}man, coagt \Rightarrow mary{:}woman)) = no,$$

$$Query(illegal\_act^{\bullet}(agt \Rightarrow john{:}man, coagt \Rightarrow y{:}person)) = yes$$

by applying the query system to the results of the resolutions in Example 3.14. Given the program $P_1' = (\Sigma_1', CS_1')$ *in Example 3.14. The following answers*

$$Query(illegal\_act^{\bullet}(agt \Rightarrow john{:}man)) = yes,$$

$$Query(rob\_with\_violence^{\bullet}(agt \Rightarrow john{:}man, coagt \Rightarrow mary{:}woman, obj \Rightarrow c{:}wallet)) = yes$$

are derived from the results of the resolutions in Example 3.14.

Example 3.17 *Given the program* P_2 *in Example 3.15, the answer to the query* $fly^{\bullet}(sbj \Rightarrow y{:}animal)$ *is*

$$Query(fly^{\bullet}(sbj \Rightarrow y{:}animal)) = yes,$$

since we have the successful resolution $G_1 \xrightarrow{\theta_1}_{R1} \Box$ *with* $G_1 = fly^{\bullet}(sbj \Rightarrow y{:}animal)$, $\theta_1 = \{c{:}bird/y{:}animal\}$ *and therefore* $Dom(\theta_1) \cap CUVar(G_1) = \emptyset$ *in Example 3.15. The answer to the query* $fly^{\bullet}(sbj \Rightarrow y{:}penguin)$ *is*

$$Query(fly^{\bullet}(sbj \Rightarrow y{:}penguin)) = no,$$

as the resolution of the goal $move^{\bullet}(sbj \Rightarrow y{:}penguin)$ *fails in Example 3.15. Moreover, the answer to the query* $move^{\bullet}(sbj \Rightarrow y{:}animal)$ *is*

$$Query(move^{\bullet}(sbj \Rightarrow y{:}animal)) = yes,$$

by the successful resolution $G_3 \xrightarrow{\theta_3}_{R2} \square$ *with* $G_3 = move^{\bullet}(sbj \Rightarrow y{:}animal)$, $\theta_3 = \{c{:}bird/y{:}animal\}$ *and therefore* $Dom(\theta_3) \cap CUVar(G_3) = \emptyset$ *in Example 3.15.*

Next we give the answers to queries that include property formulas. The answer to the query $fly^{\sharp}(sbj \Rightarrow Y{:}animal)$ *is*

$$Query(fly^{\sharp}(sbj \Rightarrow Y{:}animal)) = no,$$

because although we can obtain the successful resolution $G'_1 \xrightarrow{\theta'_1}_{R1} \square$ *with* $G'_1 = fly^{\sharp}(sbj \Rightarrow Y{:}animal)$ *and* $\theta'_1 = \{x{:}bird/Y{:}animal\}$, $Dom(\theta'_1) \cap CUVar(G'_1) = \{Y{:}animal\}$ *in Example 3.15. The answer to the query* $fly^{\sharp}(sbj \Rightarrow Y{:}penguin)$ *is*

$$Query(move^{\sharp}(sbj \Rightarrow Y{:}penguin)) = yes,$$

since we have the successful resolution $G'_2 \xrightarrow{\theta'_2}_{R2} \square$ *with* $G'_2 = move^{\sharp}(sbj \Rightarrow Y{:}penguin)$, $\theta'_2 = \{Y{:}penguin/x{:}bird\}$ *and therefore* $Dom(\theta'_2) \cap CUVar(G'_2) = \emptyset$ *in Example 3.15. Moreover, the answer to the query* $move^{\sharp}(sbj \Rightarrow Y{:}animal)$ *is*

$$Query(move^{\sharp}(sbj \Rightarrow Y{:}animal)) = no,$$

because although we have the successful resolution $G'_3 \xrightarrow{\theta'_3}_{R2} \square$ *with* $G'_3 = move^{\sharp}(sbj \Rightarrow Y{:}animal)$ *and* $\theta'_3 = \{x{:}bird/Y{:}animal\}$, $Dom(\theta'_3) \cap CUVar(G'_3) = \{Y{:}animal\}$ *in Example 3.15.*

These results show that the query system for $\mathcal{L}_Q$ gives the answers required in Example 1 and Example 2 in Section 2.1.3.

3.4.2 The application of the query system to a legal reasoning

This section shows the the query system we have proposed is useful for other knowledge representation cases. In the following, we consider the application of our query system to two legal reasoning examples.

A criminal case:

We describe the first legal case in our order-sorted first-order language. The sorted signature Σ with hierarchical predicates is an ordered quadruple $(\mathcal{S}, \mathcal{F}, \mathcal{P}, \mathcal{D})$ with

$$\begin{aligned}
\mathcal{D}_{\mathcal{S}} &= \{man \sqsubset_S person, woman \sqsubset_S person, bat \sqsubset_S thing,\\
&\quad \bot \sqsubset_S man, \bot \sqsubset_S woman, \bot \sqsubset_S bat, person \sqsubset_S \top, thing \sqsubset_S \top\},\\
\mathcal{D}_{\mathcal{F}} &= \{john{:}\rightarrow man, mary{:}\rightarrow woman, c{:}\rightarrow bat\},\\
\mathcal{D}_{\mathcal{P}} &= \{die \sqsubset_P legal_act, hit \sqsubset_P illegal_act,\\
&\quad murder \sqsubset_P illegal_act, illegal_act \sqsubset_P act\} \cup\\
&\quad \{die{:}\{(agt, person)\},\\
&\quad legal_act{:}\{(agt, person)\},\\
&\quad illegal_act{:}\{(agt, person)\},\\
&\quad hit{:}\{(agt, person), (coagt, person), (tool, thing), (place, \top)\},\\
&\quad murder{:}\{(agt, person), (coagt, person)\},\\
&\quad intent_to_murder{:}\{(agt, person), (coagt, person)\},\\
&\quad act{:}\{(agt, person), (coagt, person)\}\ \}.
\end{aligned}$$

The program P is an ordered pair $(\Sigma, F \cup R)$ with

$$\begin{aligned} F &= \{hit^{\bullet}(agt = john{:}\,man, coagt = mary{:}\,woman, tool = c_1{:}\,bat, \\ &\qquad place = c_2{:}\,home), \\ &\quad die^{\bullet}(agt = mary{:}\,woman), \\ &\quad intent\_to\_murder^{\bullet}(agt = john{:}\,man, coagt = mary{:}\,woman)\ \}, \\ R &= \{murder^{\bullet}(agt = x{:}\,person, coagt = y{:}\,person) \leftarrow \\ &\quad act^{\bullet}(agt = x{:}\,person, coagt = y{:}\,person), die^{\bullet}(agt = y{:}\,person), \\ &\quad intent\_to\_murder^{\bullet}(agt = x{:}\,person, coagt = y{:}\,person)\ \}. \end{aligned}$$

Given the program P as a legal knowledge base, our query system will give the following answers. First, the query "Did John murder a person" yields the answer `yes`. Second, the query "Did John commit an illegal act?" also yields the answer `yes` as follows.

```
?-murder*(agt=>john:man,coagt=>Y:person).
yes.
?-illegal_act*(agt=>john:man).
yes.
```

where we denote existential and universal variables in queries by `Y` and `(Y)` respectively.

In the query system we propose, we will explain the inference process of the query

```
?-illegal_act*(agt=>john:man).
```

In order to obtain the answer `yes` from this query, there must exist a successful resolution of the goal $illegal\_act^{\bullet}(agt \Rightarrow john{:}\,man)$. With respect to the program P, the empty clause is derived as follow

$$\begin{aligned} & illegal\_act^{\bullet}(agt \Rightarrow john{:}\,man) \\ \xrightarrow{\theta_1}_{R2}\ & act^{\bullet}(agt \Rightarrow john{:}\,man, coagt \Rightarrow mary{:}\,woman), \\ & die^{\bullet}(agt \Rightarrow mary{:}\,woman), \\ & intent\_to\_murder^{\bullet}(agt \Rightarrow john{:}\,man, coagt \Rightarrow mary{:}\,woman) \\ \xrightarrow{\epsilon}_{R1}\ & die^{\bullet}(agt \Rightarrow mary{:}\,woman), \\ & intent\_to\_murder^{\bullet}(agt \Rightarrow john{:}\,man, coagt \Rightarrow mary{:}\,woman) \\ \xrightarrow{\epsilon}_{R1}\ & intent\_to\_murder^{\bullet}(agt \Rightarrow john{:}\,man, coagt \Rightarrow mary{:}\,woman) \\ \xrightarrow{\epsilon}_{R1}\ & \square \end{aligned}$$

where $\theta_1 = \{mary{:}\,woman/x{:}\,person, y{:}\,beer/y{:}\,alcoholic, z{:}\,bar/z{:}\,space\}$. Furthermore, since there exists a successful resolution of goal G where $G = illegal\_act^{\bullet}(agt \Rightarrow john{:}\,man)$ and therefore $Dom(\theta_1) \cap CUVar(G) = \emptyset$, the answer to the query $illegal\_act^{\bullet}(agt \Rightarrow john{:}\,man)$ is

$$Query(illegal\_act^{\bullet}(agt \Rightarrow john{:}\,man)) = yes$$

where $Query$ is the query system defined in the previous section.

Underage drinking:

We give the second legal case as follows. The sorted signature Σ with hierarchical predicates is an ordered quadruple $(\mathcal{S}, \mathcal{F}, \mathcal{P}, \mathcal{D})$ with

$$
\begin{array}{rcl}
\mathcal{D}_{\mathcal{S}} & = & \{adult \sqsubset_S person, minor \sqsubset_S person, beer \sqsubset_S alcoholic, \\
 & & alcoholic \sqsubset_S thing, bar \sqsubset_S space, \\
 & & \bot \sqsubset_S adult, \bot \sqsubset_S minor, \bot \sqsubset_S beer, \\
 & & person \sqsubset_S \top, thing \sqsubset_S \top space \sqsubset_S \top\}, \\
\mathcal{D}_{\mathcal{F}} & = & \{peter\colon \to minor, mary\colon \to adult\}, \\
\mathcal{D}_{\mathcal{P}} & = & \{underage\_drinking \sqsubset_P illegal\_act, legal\_act \sqsubset_P act\} \cup \\
 & & \{illegal\_act\colon \{(agt, person)\}, \\
 & & drink\colon \{(agt, person), (obj, thing), (place, \top)\}, \\
 & & underage\_drinking\colon \{(agt, person)\}, \\
 & & act\colon \{(agt, person), (coagt, person)\}\ \}.
\end{array}
$$

The program P is an ordered pair $(\Sigma, F \cup R_1 \cup R_2)$ with

$$
\begin{array}{rcl}
F & = & \{drink^{\bullet}(agt = peter\colon minor, obj = y\colon beer)\}, \\
R_1 & = & \{underage\_drinking^{\bullet}(agt = x\colon minor) \leftarrow \\
 & & drink^{\bullet}(agt = x\colon minor, obj = y\colon alcoholic)\ \}, \\
R_2 & = & \{drink^{\sharp}(agt = x\colon adult, obj = y\colon alcoholic)\ \}.
\end{array}
$$

In the set R_2 in P, a legal rule interpreted as "Adults have the right to drink alcoholic" is declared. Thus, the first query "Does Mary have the right to drink alcoholic?" will give the answer `yes` in our query system. The answer to the more concrete query "Does Mary have the right to drink beer in a bar" is also `yes`. However, the third query "Did Mary drink alcoholic?" will yield `no`.

```
?-drink#(agt=>mary:adult,obj=>(Y):alcoholic,place=>(Z):T).
yes
?-drink#(agt=>mary:adult,obj=>Y:beer,place=>Z:bar).
yes
?-drink*(agt=>mary:adult,obj=>Y:alcoholic).
no
```

In the following, we explain the inference process of the query

```
?-drink#(agt=>mary:adult,obj=>Y:beer,place=>Z:bar).
```

To derive the answer `yes` from this query, there must exist a successful resolution of the goal $drink^{\sharp}(agt \Rightarrow mary\colon adult, obj \Rightarrow y\colon beer, place \Rightarrow z\colon bar)$. With respect to the program P, the empty clause is derived as follow

$$drink^{\sharp}(agt \Rightarrow mary\colon adult, obj \Rightarrow y\colon beer, place \Rightarrow z\colon bar) \xrightarrow{\theta_1}_{R1} \square.$$

where $\theta_1 = \{mary{:}\,adult/x{:}\,adult, y{:}\,beer/y{:}\,alcoholic, z{:}\,bar/z{:}\,space\}$. Moreover, since there exists a successful resolution of goal G where $G = drink^\sharp(agt \Rightarrow mary{:}\,adult, obj \Rightarrow y{:}\,beer, place \Rightarrow z{:}\,bar)$ and therefore $Dom(\theta_1) \cap CUVar(G) = \emptyset$, the answer to the query $drink^\sharp(agt \Rightarrow mary{:}\,adult, obj \Rightarrow y{:}\,beer, place \Rightarrow z{:}\,bar)$ is

$$Query(drink^\sharp(agt \Rightarrow x{:}\,mary, obj \Rightarrow y{:}\,beer, place \Rightarrow z{:}\,bar)) = yes$$

where $Query$ is the query system defined in the previous section.

Considering the query "Did Peter commit an illegal act?" with regard to Peter's drinking, the answer `yes`, since the answer to the query "Does Peter have the right to drink alcoholic?" is `no` as follows.

```
?-illegal_act*(agt=>peter:minor).

yes

?-drink#(agt=>peter:minor,obj=>(Y):alcoholic).

no
```

The query system proposed in the previous section can be used as a legal reasoning system that deduces logical conclusions from legal cases, legal rules, and background knowledge. In the query language in this system, a legal case is written by ground clauses indicating event assertions, a legal rule is represented by sorted clauses and classified as laws concerning an event or a property, and background knowledge corresponds to a sort-hierarchy and a predicate-hierarchy.

Chapter 4

Implicit negations in a sort-hierarchy

Chapter 4 presents an order-sorted logic that includes the complex sort expressions of implicit negations. In Section 4.1, we give an account of structured sorts, sort relations, and contradiction in a sort-hierarchy. These notions can be used to declare the properties of implicitly negative sorts in a sort-hierarchy. Section 4.2 and Section 4.3 present the formalization of order-sorted logic with structured sorts, and systems of clausal deduction, clausal resolution, and structured sort constraints. Section 4.4 evaluates the usefulness of the knowledge representation system to deal with implicit negations. This will be shown by derivations (using a hybrid inference system obtained by combining the systems we propose) for the examples in Chapter 2.

4.1 Implicitly negative sorts

In order to deal with implicitly negative sorts in a sort-hierarchy, we introduce structured sorts, sort relations, and contradiction in a sort-hierarchy into an order-sorted logic. These notions can be used to declare the properties of implicitly negative sorts in a sort-hierarchy.

4.1.1 Structured sorts

We consider the representation of sorts in a hierarchy whose names are declared as lexical negations (classified as negative affixes or lexicons with negative meaning). In this thesis, a sort denoted by a word with negative affix is said to be *a negative sort* and a sort denoted by a lexicon with negative meaning is said to be *an opposite sort*. In general, we call these sorts *implicitly negative sorts*. To represent these negative sorts, we introduce the notation of structured sorts and relations between sorts whereby a negative sort is defined by the structured sort with strong negation operator [48] [49] and an opposite sort is defined by exclusivity. In particular, we denote an opposite sort as exclusive to its antonymous sort in a hierarchy, so that these two sorts exclude each other but neither sort is negative. In fact, we should not say that an opposite sort is negative, rather we should say that these two sorts are opposite in meaning.

Structured sorts are constructed from atomic sorts, the connectives $\sqcap, \sqcup$, and the negative operators $-, \sim$ as follows.

Structured sorts:

$angry \sqcap hungry$	Conjunction

$winner \sqcup loser$	Disjunction
$\overline{happy}$	Complement (classical negation)
$\sim happy$	Negative sort (strong negation)
$\top$	Greatest sort
$\bot$	Least sort

In the above structured sorts, the sorts *angry*, *hungry*, *winner*, *loser*, *happy*, $\top$, and $\top$ are atomic sorts. $angry \sqcap hungry$ denotes the conjunction of *angry* and *hungry* that means the elements are in *angry* and *hungry*. $winner \sqcup loser$ denotes the disjunction of *winner* and *loser* that means the elements are in *winner* or *loser*. $\overline{happy}$ denotes the complement (corresponding to the classical negation) of *happy* that means the elements are not in *happy*, and $\sim happy$ denotes the negative sort (corresponding to the strong negation) of *happy* that means the elements are in *feeling opposite to happy* (or *unhappy*). $\top$ and $\bot$ are the greatest sort and the least sort in a sort-hierarchy.

When we denote by $\overline{winner}$ the complement of the sort *winner*, it corresponds to the negative formula $\neg winner(x)$ in first-order predicate logic. Thus, if we use the structured sort $\overline{winner}$ as a sort predicate, then the formula expressed by $\overline{winner}(x)$ is semantically equivalent to the negative formula $\neg winner(x)$[1]. On the basis of this expression, an assertion with the unary predicates expressed by structured sorts (i.e. the sort predicates) is written as the following.

Assertion:

$angry \sqcap hungry(bob)$ $(=\ (angry \sqcap hungry)(bob))$,
$\sim winner(tom)$ $(=\ (\sim winner)(tom))$.

Note that $\sim winner(tom)$ is the abbreviation of $(\sim winner)(tom)$ but not $\sim(winner(tom))$ because $\sim winner$ is used as a sort predicate.

4.1.2 Negations by sort relations

We now give several relations between structured sorts in order to represent implicitly negative sorts embedded in a sort-hierarchy. '$\sqsubseteq_S$' denotes a subsort relation between structured sorts where $\sqsubseteq_S$ is a partial order (i.e. reflexive, anti-symmetric, and transitive). Furthermore, '$=_S$' denotes an equivalence relation between structured sorts, '$\|$' denotes an exclusivity relation between structured sorts, and '$|_{s_i}$' denotes a totality relation between structured sorts as follows.

Sort relations:

$s \sqsubseteq_S s'$	Subsort relation
$s =_S s'$	Equivalence relation
$s \,\|\, s'$	Exclusivity relation
$s \,\|_{s_i}\, s'$	Totality relation

The subsort relation $s \sqsubseteq_S s'$ declares that the structured sort s is a subsort of structured sort s'. The equivalence relation $s =_S s'$ declares the equivalence of the two sorts, and the exclusivity relation $s \,\|\, s'$ declares the mutual exclusivity of elements in the two sorts. The totality relation $s \,|_{s_i}\, s'$ declares that the union of s and s' is equivalent to the sort s_i. Hence, the equivalence relation, exclusivity relation, and totality relation are defined by

[1]This semantic equivalence will be discussed in Section 4.2.

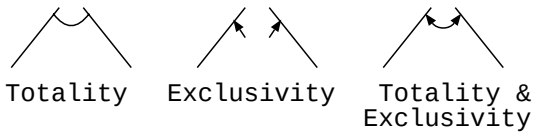

Figure 4.1 Totality and exclusivity relations between structured sorts

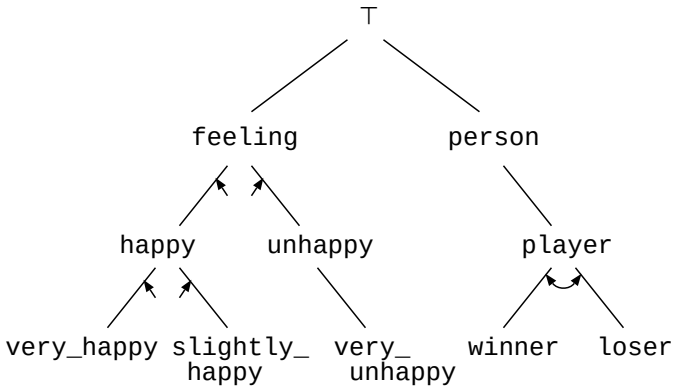

Figure 4.2 A sort-hierarchy built by sort relations

$(s \sqsubseteq_S s') \wedge (s' \sqsubseteq_S s)$, $(s \sqcap s') =_S \bot$, and $(s \sqcup s') =_S s_i$ respectively. We use the notation in Figure 4.1 to illustrate the exclusivity and totality relations in a sort-hierarchy.

Using these sort relations, we can define the following properties (totality, partiality, and exclusivity) to be used for declaring various negations (in particular, lexical negations).

Totality:

$$winner \mid_{player} loser$$

Partiality:

$$(happy \sqcup unhappy) \sqsubseteq_S feeling$$

Exclusivity:

$$happy \parallel unhappy, winner \parallel loser$$

The totality expression indicates the property that every element in *player* is at the least one of the two sorts *winner* and *loser*, and the partiality expression indicates the property that some elements in *feeling* are not in the two sorts *happy* and *unhappy*. The exclusivity expression indicates the property that all elements in *happy* are not in *unhappy* and all elements in *unhappy* are not in *happy*. For example, Figure 4.2 shows a sort-hierarchy built by these properties and the subsort relation.

The properties characterize three types of negations (complement, negative sort, and opposite sort) in a sort-hierarchy built up using structured sorts. Table 4.1 gives the properties and expressions of these negations in a sort-hierarchy. On the basis of the law of excluded middle $(A \vee \neg A)$ in classical logic, the sort *happy* and its complement $\overline{happy}$ have the properties totality in the greatest sort $\top$ and exclusivity axiomatized by the sort relations in (1) of Table 4.1. The negative sort $\sim happy$ has the properties partiality and exclusivity axiomatized by the sort relations in (2) of Table 4.1. Since a

Table 4.1 Three negations

Negation type	Expression	Relationship		Property
(1) Complement (classical negation)	$\overline{happy}$	$\overline{happy} \mid_{\top} happy$ $\overline{happy} \parallel happy$	(in Axioms)	totality exclusivity
(2) Negative sort (strong negation)	$\sim happy$	$\sim happy \parallel happy$ $\sim happy \sqsubseteq_S \overline{happy}$	(in Axioms)	exclusivity partiality
(3) Opposite sort (antonym)	sad	$sad \parallel happy$	(in Declarations)	exclusivity

negative sort has the same properties as the strong negation, the word *unhappy* with negative affix can be defined by the equivalence relation between $\sim happy$ and *unhappy* (i.e. $\sim happy =_S unhappy$). Moreover, the opposite sort *sad* of *happy* must be declared by the exclusivity relation $sad \parallel happy$ in (3) of Table 4.1. Although complement (1) and negative sort (2) are semantically equivalent to the classical negation and the strong negation respectively, opposite sort (3) does not correspond uniquely to either one of them. Furthermore, it is not necessary to establish this correspondence. In addition, opposite sort (3) without a negative operator is not syntactically regarded as a negation and so its exclusivity must be defined as a declaration. In Figure 4.3, we show a sort-hierarchy with negative sorts and complement sorts. In Figure 4.4, we show a sort-hierarchy with opposite sorts and complement sorts.

In a sort-hierarchy, the explicit location of the three negations is obtained from the relations to their positive sorts, so that it can be used to provide a reasoning mechanism from implicitly negative sorts (specifying lexical negations) and the classical negation. Therefore, the representation of lexical negations and their reasoning can be generated from the axioms and declarations of the three types of negations, where an axiom is a premise in a knowledge base and a declaration is a background knowledge each user defines.

4.1.3 A contradiction in a sort-hierarchy

We present a contradiction in a sort-hierarchy containing the three negations (complement, negative sort, and opposite sort) that we have explained. Unlike simply introducing a new negative operator, an opposite sort is not regarded as a negation in a knowledge base. Thus, we cannot syntactically decide whether there is a contradiction between an opposite sort and its antonymous sort and we cannot establish the relations to their complements and negative sorts. For example, a contradiction between the sorts *winner* and *loser* could not be determined and the relationship between *loser* and $\overline{winner}$ could not be established.

A deductive system with implicitly negative sorts has to determine a contradiction in a sort-hierarchy in order that it can provide a sound inference mechanism derived from

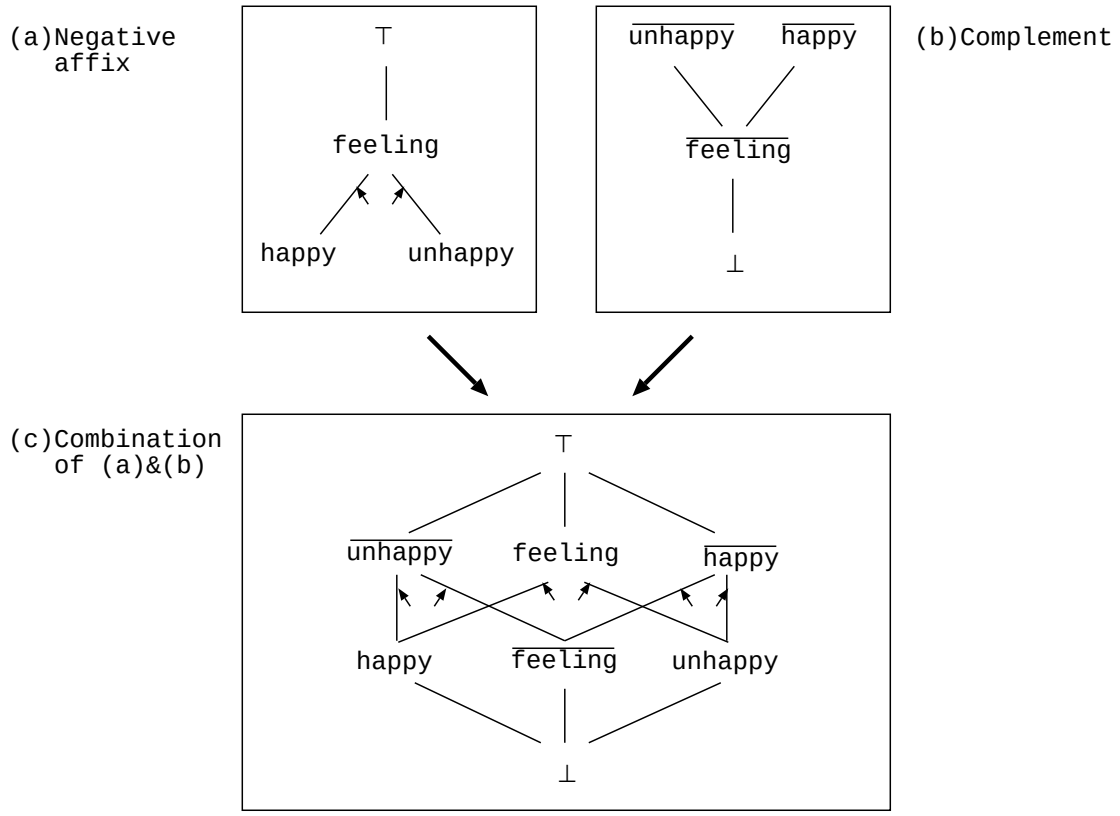

Figure 4.3 A sort-hierarchy with (a) negative affix and (b) complement

the three negations and their relations to each other. In classical logic, we can say that a set Δ of formulas is contradictory if a formula A and the classical negation $\neg A$ as below

$$A, \neg A$$

are simultaneously derivable from Δ. In this case, we can syntactically establish the contradiction, because $\neg A$ indicates the negation of A by the negative operator $\neg$. Given the opposite sorts s and s' (e.g. *winner* and *loser*), we should also say that Δ is contradictory if the following formulas

$$s(x), s'(x)$$

denoted by the sort predicates s and s' are simultaneously derivable from Δ. This indicates that the sort symbols s and s' have a negative relation to each other in our language definition in the same way that a negative operator is recognized as a negative sign in all logics. Note that contradictions should be determined not from the fact that the sorts s and s' are semantically opposite in meaning but from in the logic the fact that they are defined to be syntactically exclusive.

Using an exclusivity relation between sorts, we give a definition of contradictions in a sort-hierarchy that supports deduction from the three negations. A set Δ of formulas is said to be contradictory if there exist sorts s, s' such that $s \parallel s'$ and $s(t)$ and $s'(t)$ are derivable from Δ. In section 4.2, we will redefine the notion of contradiction in a sort-hierarchy that enables our deduction system to ensure the consistency of a knowledge base with sort-hierarchy through the sort relations.

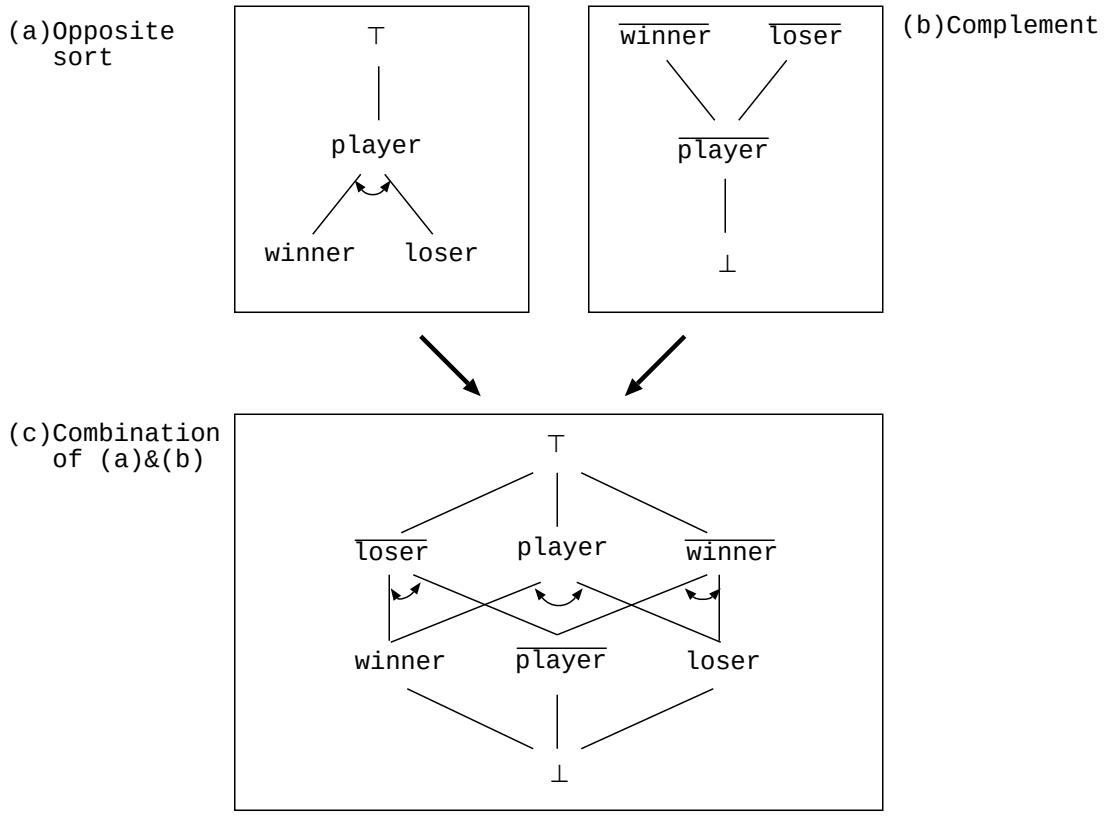

Figure 4.4 A sort-hierarchy with (a) opposite sort and (b) complement

4.2 An order-sorted logic with structured sorts

On the basis of the specification we propose in Section 4.1, we define the syntax and semantics (using the notation in [9]) of an order-sorted logic with structured sorts. The formalization contains the following notions:

(1) Syntax

- A structured sort signature Σ^+ that includes the set $\mathcal{S}^+$ of structured sorts and function and predicate declarations on $\mathcal{S}^+$.
- Structured sort terms and formulas that are obtained by expanding atomic sorts into structured sorts.

(2) Semantics

- A Σ^+-structure that consists of the universe and an interpretation of the structured sorts and the function and predicate symbols in Σ^+, and satisfies the function and predicate declarations on $\mathcal{S}^+$.

(3) Inference system

- A sort-hierarchy declaration that consists of the set $\mathcal{S}^+$ of structured sorts and a finite set D of subsort declarations.
- A structured sort constraint system with a sort-hierarchy declaration that contains axioms and inference rules for subsort declarations.
- A clausal inference system with sort predicates that contains inference rules of sort predicates related to subsort declarations.

4.2.1 Structured sort signature

Given a set $\mathcal{S}$ of sort symbols, every sort $s_i \in \mathcal{S}$ is called an atomic sort. We define the set of structured sorts composed by the atomic sorts, the connectives $\sqcap, \sqcup$, and the negative operators $-, \sim$ as follows.

Definition 4.1 (Structured sorts) *Given a set $\mathcal{S}$ of atomic sorts, the set $\mathcal{S}^+$ of structured sorts is defined by:*

(1) If $s \in \mathcal{S}$, then $s \in \mathcal{S}^+$,

(2) If $s, s' \in \mathcal{S}^+$, then $(s \sqcap s')$, $(s \sqcup s')$, $(\overline{s})$, $(\sim s) \in \mathcal{S}^+$.

The structured sort $\overline{s}$ is called the classical negation of sort s and the structured sort $\sim s$ is called the strong negation of sort s. For convenience, we can denote $s \sqcap s'$, $s \sqcup s'$, $\overline{s}$ and $\sim s$ without parentheses when there is no possibility of confusion. The next example presents structured sorts composed of atomic sorts.

Example 4.1 *Given the atomic sorts male, student, person and happy, we can give structured sorts as follows.*

$$student \sqcap \overline{male},$$
$$person \sqcup \sim happy.$$

The structured sort $student \sqcap \overline{male}$ means "students that are not male," and the structured sort $person \sqcup \sim happy$ means "individuals that are persons or unhappy."

We define a sorted signature on the set $\mathcal{S}^+$ of structured sorts. $\mathcal{F}_n$ is a set of n-ary function symbols $(f, f_0, f_1, \ldots)$, and $\mathcal{P}_n$ is a set of n-ary predicate symbols $(p, p_0, p_1, \ldots)$. Let $\mathcal{S}^+ = \{s_1, \ldots, s_n\}$ be the set of structured sorts built by $\mathcal{S}$. We introduce *the sort predicates* $p_{s_1}, \ldots, p_{s_n}$ (discussed in [10]) indexed by sorts $s_1, \ldots, s_n$ where p_{s_i} is a unary predicate (i.e. $p_{s_i} \in \mathcal{P}_1$) and equivalent to the sort s_i. We simply write s for p_s when this will not cause confusion. For example, instead of the formula $p_s(t)$ where t is a term, we use the notation $s(t)$. We denote by $\mathcal{P}_{\mathcal{S}^+}$ the set $\{p_s \in \mathcal{P}_1 \mid s \in \mathcal{S}^+ - \{\top, \bot\}\}$ of the sort predicates indexed by all sorts in $\mathcal{S}^+$ in $\mathcal{P}_1$.

We define a sorted signature on $\mathcal{S}^+$ extended to include structured sorts and sort predicates.

Definition 4.2 (Sorted signature on $\mathcal{S}^+$) *A sorted signature on $\mathcal{S}^+$ (which we call a structured sort signature) is an ordered quadruple $\Sigma^+ = (\mathcal{S}^+, \mathcal{F}, \mathcal{P}, \mathcal{D})$ satisfying the following conditions:*

(1) $\mathcal{S}^+$ *is the set of all structured sorts constructed by* $\mathcal{S}$.

(2) $\mathcal{F}$ *is the set* $\bigcup_{n\geq 0}\mathcal{F}_n$ *of all function symbols* $f_1, f_2, \ldots$

(3) $\mathcal{P}$ *is the set* $\bigcup_{n\geq 0}\mathcal{P}_n$ *of all predicate symbols* $p_1, p_2, \ldots$

(4) $\mathcal{D}$ *is a set of sort declarations such that:*

(i) If $f \in \mathcal{F}_n$, *then* $f\colon s_1 \times \ldots \times s_n \to s \in \mathcal{D}$. *In particular, if* $c \in \mathcal{F}_0$, *then* $c\colon \to s \in \mathcal{D}$.

(ii) If $p \in \mathcal{P}_n$, *then* $p\colon s_1 \times \ldots \times s_n \in \mathcal{D}$. *In particular, if* $p_s \in \mathcal{P}_{\mathcal{S}^+}$ *and* $s \in \mathcal{S}^+$, *then* $p_s\colon \top \in \mathcal{D}$.

The sort declarations of functions (called function declarations) and the sort declarations of predicates (called predicate declarations) are given by structured sorts in $\mathcal{S}^+$. The structured sort signatures do not include subsort declarations.

In the following two examples, we show structured sort signatures for Example 3 and Example 4 in Section 2.2. Given a set $\mathcal{S}$ of atomic sorts, the set $\mathcal{S}^+$ of structured sorts is constructed as follows.

Example 4.2 *Negative affix: unhappy*
Let

$$\begin{aligned}\mathcal{S} \;=\; \{&feeling, happy, unhappy, very\_happy, slightly\_happy, very\_unhappy,\\ &person, woman, man, \top, \bot\}.\end{aligned}$$

Consider the structured sort signature $\Sigma^+ = (\mathcal{S}^+, \mathcal{F}, \mathcal{P}, \mathcal{D})$ *that comprises the following symbols*

$$\begin{aligned}\mathcal{S}^+ &= \mathcal{S} \cup \{feeling \sqcap happy, feeling \sqcup happy, \overline{feeling}, {\sim}feeling, \ldots\},\\ \mathcal{F} &= \{bob\},\\ \mathcal{P} &= \{p_{feeling}, p_{happy}, p_{unhappy}, \ldots\},\\ \mathcal{D} &= \{bob\colon \to person\} \cup\\ &\quad \{p_{feeling}\colon \top,\ p_{happy}\colon \top,\ p_{unhappy}\colon \top, \ldots\}.\end{aligned}$$

where $p_{feeling}, p_{happy}, p_{unhappy}$ *are sort predicates and* $p_{feeling}\colon \top$, $p_{happy}\colon \top$, $p_{unhappy}\colon \top$ *are the predicate declarations.*

For all the structured sorts in $\mathcal{S}^+$, their sort predicates and the predicate declarations are declared in $\mathcal{P}$ and $\mathcal{D}$. The declarations of sort predicates must be of form $p\colon \top$.

Example 4.3 *Lexicon with negative meaning: loser*
Let

$$\mathcal{S} = \{person, player, winner, loser, \top, \bot\}$$

Consider the structured sort signature $\Sigma^+ = (\mathcal{S}^+, \mathcal{F}, \mathcal{P}, \mathcal{D})$ *that comprises the following symbols*

$$\begin{aligned}
\mathcal{S}^+ &= \mathcal{S} \cup \{person \sqcap player, person \sqcup player, \overline{person}, {\sim}person, \ldots\},\\
\mathcal{F} &= \{tom, father\},\\
\mathcal{P} &= \{p_{person}, p_{player}, p_{winner}, p_{loser}, \ldots\},\\
\mathcal{D} &= \{tom{:} \rightarrow person,\ \ father{:} person \rightarrow person\} \cup\\
&\quad\ \{p_{person}{:}\top,\ p_{player}{:}\top,\ p_{winner}{:}\top,\ p_{loser}{:}\top, \ldots\}.
\end{aligned}$$

where $p_{person}, p_{player}, p_{winner}, p_{loser}$ *are sort predicates and* $p_{person}{:}\top$, $p_{player}{:}\top$, $p_{winner}{:}\top$, $p_{loser}{:}\top$ *are the predicate declarations.*

An alphabet for an order-sorted first-order language $\mathcal{L}_{\mathcal{S}^+}$ of structured sort signature Σ^+ contains the following: the set $\mathcal{V}^+ = \bigcup_{s \in \mathcal{S}^+} \mathcal{V}_s^+$ of variables for all structured sorts (where $\mathcal{V}_s^+$ is a set of variables $x_1{:}s, x_2{:}s, \ldots$ for structured sort s), the connectives $\neg, \wedge, \vee, \rightarrow$, the quantifiers $\forall, \exists$, and the auxiliary symbols '(', ')', ','. The sort of each variable is denoted by a structured sort. For instance,

$$x{:}(s \sqcap s') \sqcup {\sim}s''$$

is a variable of structured sort $(s \sqcap s') \sqcup {\sim}s''$.

4.2.2 Structured sort terms and formulas

In this section, we give the expressions *structured sort term* and *formula* for our order-sorted first-order language with structured sorts. Structured sort terms (resp. formulas) are sorted terms (resp. formulas) whereby the simple sort expressions are expanded into structured sorts.

First, we define terms over structured sort signatures in the usual manner of order-sorted predicate logics.

Definition 4.3 (Structured sort terms) *Let* $\Sigma^+ = (\mathcal{S}^+, \mathcal{F}, \mathcal{P}, \mathcal{D})$ *be an structured sort signature. The set* $TERM_{\Sigma^+,s}$ *of terms of structured sort* s *is defined by:*

(1) A variable $x{:}s$ *is a term of structured sort* s *where* $s \in \mathcal{S}^+$.

(2) A constant $c{:}s$ *is a term of structured sort* s *where* $s \in \mathcal{S}^+$, $c \in \mathcal{F}_0$, *and* $c{:} \rightarrow s \in \mathcal{D}$.

(3) If $t_1, \ldots, t_n$ *are terms of structured sorts* $s_1, \ldots, s_n$, *then* $f(t_1, \ldots, t_n){:}s$ *is a term of structured sort* s *where* $s \in \mathcal{S}^+$, $f \in \mathcal{F}_n$, *and* $f{:}s_1 \times \ldots \times s_n \rightarrow s \in \mathcal{D}$.

We denote by $TERM_{\Sigma^+}$ the set of all structured sort terms $\bigcup_{s \in \mathcal{S}^+} TERM_{\Sigma^+,s}$. In the definition of structured sort terms, the set $TERM_{\Sigma^+,s}$ of terms of structured sort s contain no terms of their subsorts because there are no subsort declarations in the structured sort signature Σ^+. For the sorted substitutions, subsort declarations will be derived using a sort constraint system in order to substitute variables with terms of the subsorts.

Example 4.4 *For the structured sort signature* Σ^+ *of Example 4.3, we give an example of structured sort terms shown as*

$$x{:}loser \sqcap player,$$
$$father(tom{:}person \sqcap \overline{player}).$$

$x{:}\,loser \sqcap player$ is a variable of structured sort $loser \sqcap player$ whose domain is the set of individuals that are players and losers. The function $father(tom{:}\,person \sqcap \overline{player})$ expresses "*tom*'s father where *tom* is a person but not a player."

We define the function $Sort(t)$ giving the sort of structured sort term t. Since a structured sort term carries its sort information in the form '$:s$', the sort can be easily obtained from it.

Definition 4.4 *Let $\Sigma^+ = (\mathcal{S}^+, \mathcal{F}, \mathcal{P}, \mathcal{D})$ be a structured sort signature. The function $Sort{:}\,TERM_{\Sigma^+} \to \mathcal{S}^+$ is defined by*

(1) $Sort(x{:}\,s) = s$ *where* $s \in \mathcal{S}^+$,

(2) $Sort(c{:}\,s) = s$ *where* $s \in \mathcal{S}^+$,

(3) $Sort(f(t_1, \ldots, t_n){:}\,s) = s$ *where* $s \in \mathcal{S}^+$.

This function *Sort* is used to indicate the conditions of inference rules in the deduction and resolution systems which we will present in Section 4.3.

We define formulas over structured sort signatures in the usual manner of order-sorted predicate logics.

Definition 4.5 (Structured sort formulas) *Let $\Sigma^+ = (\mathcal{S}^+, \mathcal{F}, \mathcal{P}, \mathcal{D})$ be a structured sort signature. The set $FORM_{\Sigma^+}$ of structured sort formulas is defined by:*

(1) If $t_1, \ldots, t_n$ are terms of $s_1, \ldots, s_n$, then $p(t_1, \ldots, t_n)$ is an atomic formula (or simply an atom) where $p \in \mathcal{P}_n$ and $p{:}\,s_1 \times \ldots \times s_n \in \mathcal{D}$,

(2) If A and B are formulas, then $(\neg A)$, $(A \wedge B)$, $(A \vee B)$, $(A \to B)$, $(\forall x{:}\,sA)$, and $(\exists x{:}\,sA)$ are formulas.

We write $ATOM_{\Sigma^+}$ ($\subset FORM_{\Sigma^+}$) for the set of all structured sort atoms, and write $ATOM_{\Sigma^+, \mathcal{P}_{\mathcal{S}^+}}$ for the set of all structured sort atoms defined by $\mathcal{P}_{\mathcal{S}^+}$, i.e. $ATOM_{\Sigma^+, \mathcal{P}_{\mathcal{S}^+}} = \{p(t) \in ATOM_{\Sigma^+} \mid p \in \mathcal{P}_{\mathcal{S}^+}\}$. We can omit parentheses from formulas when it is clear from the context.

In the following example, a sort synbol is used as not only a component of structured sorts but also a predicate, i.e., the sort predicate.

Example 4.5 *For the structured sort signature Σ^+ of Example 4.3, we give an example of structured sort formulas shown as*

$$winner(x{:}\,person \sqcap player),$$
$$\neg player(tom{:}\,person) \vee player(tom{:}\,person),$$
$$\overline{loser \sqcup winner}(x{:}\,person) \to \neg player(x{:}\,person).$$

The first formula means that "x is a person and player that is a winner." The second formula means that "the person *tom* is either a player or not a player." The expression *player* is used as a sort in the first formula, but it is used as a predicate (i.e. the sort predicate) in the second formula. The third formula means that "if person x is neither a loser nor a winner, then x is not a player."

We introduce literals in order to represent formulas in clause form. A positive literal is an atomic formula $p(t_1, \ldots, t_n)$, and a negative literal is the negation $\neg p(t_1, \ldots, t_n)$ of an atomic formula. A literal is a positive or a negative literal.

Definition 4.6 *Let $L_1, \ldots, L_n (n \geq 1)$ be literals. The following formula is said to be a clause form (or simply a clause).*

$$L_1 \vee \ldots \vee L_n$$

For instance, the following expression given in Example 4.5

$$\neg player(tom\colon person) \vee player(tom\colon person)$$

is of clause form where $\neg player(tom\colon person)$ is a negative literal and $player(tom\colon person)$ is a positive literal. We denote by CS_{Σ^+} ($\subset FORM_{\Sigma^+}$) the set of all clauses.

In our logic, we cannot completely declare subsort relations on $\mathcal{S}^+$ because the set of subsort declarations representing a subsort relation must be infinite. Hence, we first give a finite set of subsort declarations, so that the subsort relation should be derived by a sort constraint system. For this purpose, we want to deal with subsort declarations as subsort formulas but not as static expressions in signatures. Let $\Sigma^+ = (\mathcal{S}^+, \mathcal{F}, \mathcal{P}, \mathcal{D})$ be a structured sort signature. For $s, s' \in \mathcal{S}^+$, $s \sqsubseteq_S s'$ is a subsort declaration over Σ^+ that indicates s is a subsort of s'. For instance,

$$player \sqcap winner \sqsubseteq_S person$$

is a subsort declaration over the structured sort signature in Example 4.3. We denote by $D_{\mathcal{S}^+} = \{s \sqsubseteq_S s' | s, s' \in \mathcal{S}^+\}$ the set of all subsort declarations on $\mathcal{S}^+$.

4.2.3 Σ^+-structure

As in the semantics of standard order-sorted logics, we consider a structure that consists of the universe and an interpretation over $\mathcal{S}^+ \cup \mathcal{F} \cup \mathcal{P}$ and satisfies the function and predicate declarations on $\mathcal{S}^+$. The interpretation of atomic sorts are defined by subsets of the universe (in particular, the greatest sort $\top$ is interpreted by the universe and the least sort $\bot$ is interpreted by the empty set). Hence, the interpretation of structured sorts is constructed by the interpretation of atomic sorts and the operations of set theory. Function symbols and predicate symbols are interpreted by functions and predicates over the universe.

Definition 4.7 *Given a structured sort signature $\Sigma^+ = (\mathcal{S}^+, \mathcal{F}, \mathcal{P}, \mathcal{D})$. A Σ^+-structure is an ordered pair $M^+ = (U, I^+)$ such that*

(1) U is a non-empty set,

(2) I^+ is a function on $\mathcal{S}^+ \cup \mathcal{F} \cup \mathcal{P}$ where

- $I^+(s) \subseteq U$ *(in particular, $I^+(\top) = U$ and $I^+(\bot) = \emptyset$),*
  $I^+(s \sqcap s') = I^+(s) \cap I^+(s')$,
  $I^+(s \sqcup s') = I^+(s) \cup I^+(s')$,
  $I^+(\overline{s}) = I^+(\top) - I^+(s)$,
  $I^+(\sim s) \subseteq I^+(\top) - I^+(s)$.
- $I^+(f)\colon I^+(s_1) \times \ldots \times I^+(s_n) \to I^+(s)$ *where* $f \in \mathcal{F}_n$ *and* $f\colon s_1 \times \ldots \times s_n \to s \in \mathcal{D}$.

– $I^+(p) \subseteq I^+(s_1) \times \ldots \times I^+(s_n)$ *where* $p \in \mathcal{P}_n$ *and* $p\colon s_1 \times \ldots \times s_n \in \mathcal{D}$ *(in particular,* $I^+(p_s) = I^+(s)$ *where* $p_s \in \mathcal{P}_{\mathcal{S}^+}$ *and* $p_s\colon \top \in \mathcal{D}$*).*

The strong negation $\sim s$ of sort s is interpreted as a subset of the classical negation $\overline{s}$. For instance, we can obtain

$$I^+(\sim happy) \subseteq I^+(\overline{happy})$$

for the structured sorts $\sim happy$ and $\overline{happy}$. The sort predicates p_s are semantically equivalent to the sorts s, i.e., $I^+(p_s) = I^+(s)$.

A variable assignment (or simply an assignment) in a Σ^+-structure $M^+ = (I^+, U)$ is a function $\alpha^+\colon \mathcal{V}^+ \to U$ where $\alpha^+(x\colon s) \in I^+(s)$ for all variables $x\colon s \in \mathcal{V}^+$. Let α^+ be an assignment in a Σ^+-structure $M^+ = (I^+, U)$, $x\colon s$ a variable in $\mathcal{V}^+$, and $d \in I^+(s)$. The assignment $\alpha^+[d/x\colon s]$ is defined by $\alpha^+[d/x\colon s] = (\alpha^+ - \{(x\colon s, \alpha^+(x\colon s))\}) \cup \{(x\colon s, d)\}$. We write $\alpha^+[d_1/x_1\colon s_1, \ldots, d_n/x_n\colon s_n]$ for $(((\alpha^+[d_1/x_1\colon s_1])[d_2/x_2\colon s_2]) \ldots)[d_n/x_n\colon s_n]$. That is, if $y\colon s = x_i\colon s_i$ for some $i \in \{1, \ldots, n\}$, then $\alpha^+[d_1/x_1\colon s_1, \ldots, d_n/x_n\colon s_n](y\colon s) = d_i$. Otherwise, $\alpha^+[d_1/x_1\colon s_1, \ldots, d_n/x_n\colon s_n](y\colon s) = \alpha^+(y\colon s)$. For instance, if there exists an assignment α such that

$$\alpha(y\colon s) = d,$$

then we have

$$\alpha[d'/y\colon s](y\colon s) = d'.$$

We now define an interpretation over structured sort signatures Σ^+. If an interpretation $\mathcal{I}^+$ consists of a Σ^+-structure M^+ and an assignment α^+ in M^+, then $\mathcal{I}^+$ is said to be a Σ^+-interpretation.

Definition 4.8 *Let* $\mathcal{I}^+ = (M^+, \alpha^+)$ *be a* Σ^+*-interpretation. The denotation* $[\![\ ]\!]_{\alpha^+}$ *is defined by*

(1) $[\![x\colon s]\!]_{\alpha^+} = \alpha^+(x\colon s)$,

(2) $[\![c\colon s]\!]_{\alpha^+} = I^+(c)$ *with* $I^+(c) \in I^+(s)$,

(3) $[\![f(t_1, \ldots, t_n)\colon s]\!]_{\alpha^+} = I^+(f)([\![t_1]\!]_{\alpha^+}, \ldots, [\![t_n]\!]_{\alpha^+})$.

We formalize a satisfiability relation indicating that a Σ^+-interpretation satisfies structured sort formulas and subsort declarations.

Definition 4.9 *Let* $\mathcal{I}^+ = (M^+, \alpha^+)$ *be a* Σ^+*-interpretation and* F *a structured sort formula or a subsort declaration in* $FORM_{\Sigma^+} \cup D_{\mathcal{S}^+}$*. We define the satisfiability relation* $\mathcal{I} \models F$ *by the following rules:*

(1) $\mathcal{I}^+ \models p(t_1, \ldots, t_n)$ *iff* $([\![t_1]\!]_{\alpha^+}, \ldots, [\![t_n]\!]_{\alpha^+}) \in I^+(p)$,

(2) $\mathcal{I}^+ \models (\neg A)$ *iff* $\mathcal{I}^+ \not\models A$,

(3) $\mathcal{I}^+ \models (A \wedge B)$ *iff* $\mathcal{I}^+ \models A$ *and* $\mathcal{I}^+ \models B$,

(4) $\mathcal{I}^+ \models (A \vee B)$ *iff* $\mathcal{I}^+ \models A$ *or* $\mathcal{I}^+ \models B$,

(5) $\mathcal{I}^+ \models (A \to B)$ *iff* $\mathcal{I}^+ \not\models A$ *or* $\mathcal{I}^+ \models B$,

(6) $\mathcal{I}^+ \models (\forall x{:}\, s)A$ *iff for all* $d \in I^+(s)$, $\mathcal{I}^+[d/x{:}\, s] \models A$ *holds,*

(7) $\mathcal{I}^+ \models (\exists x{:}\, s)A$ *iff for some* $d \in I^+(s)$, $\mathcal{I}^+[d/x{:}\, s] \models A$ *holds,*

(8) $\mathcal{I}^+ \models s \sqsubseteq_S s'$ *iff* $I^+(s) \subseteq I^+(s')$.

Let F be a structured sort formula or a subsort declaration and let $\Gamma \subseteq FORM_{\Sigma^+} \cup D_{\mathcal{S}^+}$. If $\mathcal{I}^+ \models F$, then $\mathcal{I}^+$ is said to be a Σ^+-model of F. We denote $\mathcal{I}^+ \models \Gamma$ if $\mathcal{I}^+ \models F$ for every $F \in \Gamma$. If $\mathcal{I}^+ \models \Gamma$, then $\mathcal{I}^+$ is said to be a Σ^+-model of Γ. If every Σ^+-interpretation $\mathcal{I}^+$ is a Σ^+-model of F, then F is said to be Σ^+-valid. We write $(D, \Delta) \models_{\Sigma^+} F$ (F is a consequence of (D, Δ) in the class of Σ^+-structures) if every Σ^+-model of (D, Δ) is a Σ^+-model of F ($\in CS_{\Sigma^+} \cup D_{\mathcal{S}^+}$).

The next lemma shows that we can use the connectives $\sqcap, \sqcup$ as the logical connectives $\wedge, \vee$ and the negative operator $-$ as the logical negation $\neg$ when we build structured sort formulas.

Lemma 4.1 *Let* $\mathcal{I}^+$ *be a* Σ^+*-interpretation ,* s, s' *structured sorts, and* t *a structured sort term. The following statements hold.*

(1) $\mathcal{I}^+ \models s(t) \wedge s'(t)$ *iff* $\mathcal{I}^+ \models s \sqcap s'(t)$.

(2) $\mathcal{I}^+ \models s(t) \vee s'(t)$ *iff* $\mathcal{I}^+ \models s \sqcup s'(t)$.

(3) $\mathcal{I}^+ \models \neg s(t)$ *iff* $\mathcal{I}^+ \models \overline{s}(t)$.

(4) If $\mathcal{I}^+ \models \sim s(t)$, *then* $\mathcal{I}^+ \models \neg s(t)$.

Proof. By Definition 4.7, 4.8, and 4.9, this is clear. ∎

Note that the structured sorts $s, s', \overline{s}, \sim s, s \sqcap s'$ and $s \sqcup s'$ are used as the sort predicates. We can say that $\sim s(t)$ implies $\neg s(t)$ by the above lemma.

4.2.4 Sort-hierarchy declaration

We require to build a sort-hierarchy over $\mathcal{S}^+$, instead of subsort declarations in sorted signatures of typical order-sorted logics. In the next definition, the sort-hierarchy is obtained by a finite set of subsort declarations.

Definition 4.10 (Sort-hierarchy declaration) *A sort-hierarchy declaration is an ordered pair* $H = (\mathcal{S}^+, D)$ *where*

(1) $\mathcal{S}^+$ *is the set of structured sorts constructed by* $\mathcal{S}$,

(2) D *is a finite set* $\{s_1 \sqsubseteq_S s'_1, s_2 \sqsubseteq_S s'_2, \ldots\}$ *of subsort declarations on* $\mathcal{S}^+$.

Furthermore, we denote a sort equivalence relation by $=_S$, an exclusivity relation by $\|$, and a totality relation by $|_{s_i}$. The declarations of these relations are defined by subsort declarations as follows.

Definition 4.11 *A sort equivalence declaration, an exclusivity declaration and a totality declaration are defined respectively by*

- $s =_S s'$ *iff* $s \sqsubseteq_S s'$ *and* $s' \sqsubseteq_S s$.
- $s \parallel s'$ *iff* $(s \sqcap s') =_S \bot$.
- $s \mid_{s_i} s'$ *iff* $(s \sqcup s') =_S s_i$.

We use the abbreviation $s \mid s'$ to denote $s \mid_\top s'$. For instance, we denote $happy \mid_\top \overline{happy}$ by $happy \mid \overline{happy}$. By the above definition, we can represent a sort equivalence declaration, an exclusivity declaration and a totality declaration as the subsort declarations in D with a sort-hierarchy declaration $H = (\mathcal{S}^+, D)$. These notations are useful for declaring complicated sort relations.

Example 4.6 *The sort-hierarchy declaration* $H = (\mathcal{S}^+, D)$ *consists of the set* $\mathcal{S}^+$ *of structured sorts constructed by*

$$\mathcal{S} = \{person, winner, loser, player, \bot, \top\}$$

and the finite set D *of subsort declarations with*

$$\begin{aligned} D = \{&winner \sqsubseteq_S player, player \sqsubseteq_S person,\\ &loser \sqsubseteq_S player,\\ &winner \mid_{player} loser,\\ &winner \parallel loser\}. \end{aligned}$$

The sorts $winner$ and $loser$ are subsorts of $player$, and the sort $player$ is a subsort of $person$. The totality declaration $winner \mid_{player} loser$ indicates that $winner$ and $loser$ have the property totality in the sort $player$. The exclusivity declaration $winner \parallel loser$ indicates that $winner$ and $loser$ are mutually exclusive.

Let $H = (\mathcal{S}^+, D)$ be a sort-hierarchy declaration and Δ a set of clauses. If $\mathcal{I}^+$ is a Σ^+-model of both D and Δ, then $\mathcal{I}^+$ is said to be a Σ^+-model of (D, Δ), denoted by $\mathcal{I}^+ \models (D, \Delta)$. If (D, Δ) has a Σ^+-model , then (D, Δ) is Σ^+-satisfiable. If (D, Δ) has no Σ^+-model , then (D, Δ) is Σ^+-unsatisfiable. In the deduction and resolution systems we will present in the next section, their rules are applied to clauses in Δ (which expresses an assertional knowledge base), related to a subsort relation derivable from H.

4.3 Deduction and resolution with structured sorts

In this section, we present deduction and resolution systems (which we call hybrid inference systems) combined with a structured sort constraint system. These systems adopt the method of coupling a clausal knowledge base and a sort-hierarchy proposed in the hybrid knowledge representation system [10] in which every sort can be used to express the sort predicate which is therefore included in clauses. First, we introduce two structured sort constraint systems. Next, we build a deduction system with structured sorts, and then we design a resolution system with structured sorts that is based on the inference rules of the deduction system.

4.3.1 Structured sort constraint system

We develop two constraint systems with respect to a subsort relation on the set $\mathcal{S}^+$ of structured sorts. These constraint systems includes axioms and inference rules for subsort declarations on $\mathcal{S}^+$. The first constraint system is an axiomatic system that is based on the basic laws of Boolean algebra for sets and the properties of classical negation, strong negation and subsort relations.

Definition 4.12 *Let s, s', s'' be structured sorts. The axioms and rules of structured sort constraint system CS_1 are given by:*

Reflexivity

$s \sqsubseteq_S s$

Idempotency

$s \sqsubseteq_S s \sqcap s \qquad s \sqsubseteq_S s \sqcup s$

Commutativity

$(s \sqcap s') \sqsubseteq_S (s' \sqcap s) \quad (s \sqcup s') \sqsubseteq_S (s' \sqcup s)$

Associativity

$(s \sqcap s') \sqcap s'' \sqsubseteq_S s \sqcap (s' \sqcap s'') \qquad (s \sqcup s') \sqcup s'' \sqsubseteq_S s \sqcup (s' \sqcup s'')$

Distributivity

$(s \sqcup s') \sqcap s'' \sqsubseteq_S (s \sqcap s'') \sqcup (s' \sqcap s'') \qquad (s \sqcap s') \sqcup s'' \sqsubseteq_S (s \sqcup s'') \sqcap (s' \sqcup s'')$

Conjunction

$(s \sqcap s') \sqsubseteq_S s \qquad (s \sqcap \top) \sqsubseteq_S s \qquad (s \sqcap \bot) \sqsubseteq_S \bot$

Disjunction

$s \sqsubseteq_S (s \sqcup s') \qquad (s \sqcup \top) \sqsubseteq_S \top \qquad (s \sqcup \bot) \sqsubseteq_S s$

Least and greatest sorts

$\bot \sqsubseteq_S s \qquad s \sqsubseteq_S \top$

Classical negation

$s \parallel \overline{s} \qquad s \mid \overline{s}$

$(\overline{s \sqcap s'}) \sqsubseteq_S \overline{s} \sqcap \overline{s'} \qquad (\overline{s \sqcup s'}) \sqsubseteq_S \overline{s} \sqcup \overline{s'}$

Strong negation

$s \,\|\!\sim s \qquad \sim s \sqsubseteq_S \overline{s}$

$\sim(s \sqcap s') \sqsubseteq_S \sim s \sqcap \sim s' \qquad \sim(s \sqcup s') \sqsubseteq_S \sim s \sqcup \sim s'$

Absorption

$(s \sqcap s') \sqcup s \sqsubseteq_S s \quad (s \sqcup s') \sqcap s \sqsubseteq_S s$

Transitivity rule

$$\frac{s \sqsubseteq_S s' \quad s' \sqsubseteq_S s''}{s \sqsubseteq_S s''}$$

In these axioms the difference between classical negation and strong negation is specified by the subsort declaration $\sim s \sqsubseteq_S \overline{s}$, the exclusivity declarations $s \parallel \overline{s}$ and $s \,\|\!\sim s$ and the fact that the totality declaration $s \mid \overline{s}$ are axiomatized for classical negation but the totality declaration $s \mid \sim s$ is not axiomatized for strong negation. For example,

$$happy \mid \overline{happy}$$

is an instance of axiom scheme $s \mid \overline{s}$, but no axiom scheme exists for

$$happy \mid \sim happy.$$

The next constraint system improves the applicability of axioms and inference rules in comparison with the constraint system CS_1.

Definition 4.13 *Let s, s', s'' be structured sorts. The axioms and rules of structured sort constraint system CS_2 are given by:*

Reflexivity, Idempotency, Commutativity, Associativity, Distributivity, Least and greatest sorts, Transitivity rule

Introduction rule

$$\frac{s \sqsubseteq_S s'}{s'' \sqcap s \sqsubseteq_S s'' \sqcap s'}$$

Elimination rule

$$\frac{s \sqcup s' \sqsubseteq_S s \sqcup s'' \quad s \parallel s' \quad s \parallel s''}{s' \sqsubseteq_S s''}$$

The constraint system CS_2 is obtained by adding introduction and elimination rules to the axioms and inference rules of the constraint system CS_1. These additional rules are useful for deriving subsort declarations that are either reduced or built up. The introduction rule adds a sort s'' to the premise $s \sqsubseteq_S s'$ so that

$$s'' \sqcap s \sqsubseteq_S s'' \sqcap s'$$

can be derived. The elimination rule deletes the sort s from the premise $s \sqcup s' \sqsubseteq_S s \sqcup s''$ when $s \parallel s'$ and $s \parallel s''$ are derived so that

$s' \sqsubseteq_S s''$

can be derived. Therefore the second constraint system CS_2 is more suitable for the derivation of subsort declarations than the axiomatic constraint system CS_1.

We have to define the notion of a derivation for any inference system X. A derivation of an expression (a clause or a subsort declaration) from a sort-hierarchy declaration is defined as follows.

Definition 4.14 (Derivation with structured sorts) *Let $H = (\mathcal{S}^+, D)$ be a sort-hierarchy declaration and Δ a set of clauses. A derivation of F_n in a system X from (D, Δ) is a finite sequence $F_1, F_2, \ldots, F_n$ such that*

(i) $F_i \in D$,

(ii) $F_i \in \Delta$,

(iii) F_i *is an axiom of system* X, *or*

(iv) F_i *follows from* $F_j (j < i)$ *by one of the rules of system* X.

We write $(D, \Delta) \vdash_X F$ if F has a derivation from (D, Δ) in the system X. A refutation is a derivation of the empty clause $\Box$ from (D, Δ), written as $(D, \Delta) \vdash_X \Box$. This notion of derivations can be used for the structured sort constraint systems CS_1, CS_2, and the clausal inference systems which we will present. The following example shows a derivation in the structured sort constraint system CS_1.

Example 4.7 *Given the ordered pair $(D, \emptyset)$ with the finite set D of subsort declarations in Example 4.6, we can say*

$$(D, \emptyset) \vdash_{CS_1} loser \sqsubseteq_S player$$

by $loser \sqsubseteq_S player \in D$ *and*

$$(D, \emptyset) \vdash_{CS_1} player \sqsubseteq_S \top$$

by the axiom least and greatest sorts. Therefore, we can obtain

$$(D, \emptyset) \vdash_{CS_1} loser \sqsubseteq_S \top$$

by the transitivity rule.

We define a structured sort substitution with respect to a subsort relation derivable in a structured sort constraint system X. That is, the subsort declarations are obtained by an application of the rules from X so that the substitution is defined via the subsort declarations.

Definition 4.15 (Structured sort substitution) *Let X be a structured sort constraint system. A structured sort substitution is a function θ mapping from a finite set of variables to the set $TERM_{\Sigma^+}$ of all structured sort terms where $\theta(x\colon s) \neq x\colon s$ and $\theta(x\colon s) \in TERM_{\Sigma^+, s'}$ with $s' \neq \bot$ and $(D, \Delta) \vdash_X s' \sqsubseteq_S s$.*

The condition $s' \neq \bot$ indicates that none of the terms of sort $\bot$ can be substituted for variables. If there does not exist $s' \sqsubseteq_S s$ such that $(D, \Delta) \vdash_X s' \sqsubseteq_S s$ with $s' \neq s$ for all sorts s, then the structured sort substitutions in the system X correspond to many-sorted substitutions (i.e. not order-sorted substitutions).

Example 4.8 *Given the ordered pair $(D, \emptyset)$ in Example 4.7, we have $(D, \emptyset) \vdash_{CS_1} loser \sqsubseteq_S \top$. Then,*

$$\theta = \{c{:}\, loser / x{:}\, \top\}$$

is a structured sort substitution in the constraint system CS_1. That is, $\theta(x{:}\, \top) = c{:}\, loser$.

We define an extension of structured sort substitution θ to structured sort terms and formulas (containing clauses).

Definition 4.16 *Let A, B be structured sort formulas, s a structured sort, and θ a structured sort substitution. $E\theta$ is defined by:*

- *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 $E = p(t_1, \ldots, t_m)$, then $E\theta = p(t_1\theta, \ldots, t_m\theta)$,*
- *If $E = \neg A$, then $E\theta = \neg A\theta$,*
- *If $E = A * B$ for $* \in \{\wedge, \vee\}$, then $E\theta = A\theta * B\theta$,*
- *If $E = *A$ for $* \in \{\forall x{:}\, s, \exists x{:}\, s\}$, then $E\theta = *A(\theta{\uparrow}FVar(*A))$,*

A structured sort substitution θ is a unifier of A and B if $A\theta = B\theta$ where A and B are structured sort formulas in $FORM_{\Sigma^+}$.

4.3.2 Clausal inference system with sort predicates

We present two clausal inference systems (deduction and resolution systems) for clauses in structured sort formulas. The clauses may include sort predicates, e.g., $p(t_1, t_2) \vee s(t)$ where s is a sort predicate. We denote atomic formulas by $s(t)$ to emphasize the sort predicates in them.

Definition 4.17 (Cut rule) *Let L, L' be positive literals and C, C' clauses.*

$$\frac{\neg L \vee C \quad L' \vee C'}{(C \vee C')\theta}$$

where there exists a unifier θ for L and L' (i.e. $L\theta = L'\theta$).

The cut rule is one of the usual rules included in clausal inference systems. We use the cut rule as an inference rule in both the deduction and resolution systems. In addition to the cut rule, the deduction and resolution systems have to include inference rules of sort predicates related to subsort declarations. First, we define inference rules in the deduction system as follows.

Definition 4.18 (Deduction rules with sort predicates) *Let s, s' be structured sorts or sort predicates, t a structured sort term, and C a clause. Deduction rules with sort predicates are given as follows.*

Subsort rule

$$\frac{C \vee s'(t) \quad s' \sqsubseteq_S s}{C \vee s(t)} \qquad \frac{C \vee \neg s(t) \quad s' \sqsubseteq_S s}{C \vee \neg s'(t)}$$

Sort predicate rule

$$\frac{s' \sqsubseteq_S s}{s(t)}$$

where $s' = Sort(t)$.

We write *deduction system DS* for the system defined by the cut rule in Definition 4.17 and the deduction rules in Definition 4.18. The following example shows an application of the subsort rule.

Example 4.9 *Given the ordered pair* (D, Δ) *with the set* D *in Example 4.7 and*

$$\Delta = \{loser(tom{:}person)\},$$

the subsort rule is applied as follows.

$$\frac{loser(tom{:}person) \quad loser \sqsubseteq_S player}{player(tom{:}person)}$$

That is, we have $(D, \Delta) \vdash_{DS} player(tom{:}person)$ by the above application of the subsort rule to $(D, \Delta) \vdash_{DS} loser(tom{:}person)$ and $(D, \Delta) \vdash_{DS} loser \sqsubseteq_S player$.

On the basis of the above deduction system, we design inference rules in the resolution system. The inference rules in the resolution system are proof procedures to delete two contradictory literals (e.g. $\neg s(t)$ and $s(t)$) from their premises, similar to the cut rule. For $s \in \mathcal{S}^+$, we use the notation $Neg.s$ to denote $\overline{s}$ or $\sim s$. $Neg.s$ informally means the negation of sort s.

Definition 4.19 (Resolution rules with sort predicates) *Let* s, s', s_i *be structured sorts or sort predicates,* L, L' *positive literals,* t, t' *structured sort terms, and* C, C' *clauses. Resolution rules with sort predicates are given as follows.*

Subsort rule

$$\frac{\neg s(t) \vee C \quad s'(t') \vee C' \quad s' \sqsubseteq_S s}{(C \vee C')\theta}$$

$$\frac{Neg.s(t) \vee C \quad s'(t') \vee C' \quad s' \sqsubseteq_S s}{(C \vee C')\theta} \qquad \frac{\neg Neg.s'(t') \vee C' \quad \neg s(t) \vee C \quad s' \sqsubseteq_S s}{(C \vee C')\theta}$$

where there exists a unifier θ *for* t *and* t' *(i.e.* $t\theta = t'\theta$*).*

Sort predicate rule

$$\frac{\neg s(t) \vee C \quad s' \sqsubseteq_S s}{C} \qquad \frac{Neg.s(t) \vee C \quad s' \sqsubseteq_S s}{C}$$

where $s' = Sort(t)$.

Negation rule

$$\frac{Neg.s(t) \vee C \quad s(t') \vee C'}{(C \vee C')\theta} \qquad \frac{\neg Neg.s(t) \vee C \quad \neg s(t') \vee C'}{(C \vee C')\theta}$$

where there exists a unifier θ for t and t' (i.e. $t\theta = t'\theta$).

Disjunction rule

$$\frac{s \sqcup s'(t) \vee C \quad \neg s(t') \vee C'}{(s'(t) \vee C \vee C')\theta}$$

$$\frac{s \sqcup s'(t) \vee C \quad Neg.s(t') \vee C'}{(s'(t) \vee C \vee C')\theta} \qquad \frac{\neg Neg.(s \sqcup s')(t) \vee C \quad \neg s(t') \vee C'}{(s'(t) \vee C \vee C')\theta}$$

where there exists a unifier θ for t and t' (i.e. $t\theta = t'\theta$).

Exclusivity rule

$$\frac{s(t) \vee C \quad s'(t') \vee C' \quad s \parallel s'}{(C \vee C')\theta}$$

$$\frac{\neg Neg.s(t) \vee C \quad s'(t') \vee C' \quad s \parallel s'}{(C \vee C')\theta} \qquad \frac{\neg Neg.s(t) \vee C \quad \neg Neg.s'(t') \vee C' \quad s \parallel s'}{(C \vee C')\theta}$$

where there exists a unifier θ for t and t' (i.e. $t\theta = t'\theta$) [2].

Totality rule

$$\frac{s_i(t) \vee C \quad \neg s(t') \vee C' \quad s \mid_{s_i} s'}{(s'(t) \vee C \vee C')\theta}$$

$$\frac{s_i(t) \vee C \quad Neg.s(t') \vee C' \quad s \mid_{s_i} s'}{(s'(t) \vee C \vee C')\theta} \qquad \frac{\neg Neg.s_i(t) \vee C \quad \neg s(t') \vee C' \quad s \mid_{s_i} s'}{(s'(t) \vee C \vee C')\theta}$$

where there exists a unifier θ for t and t' (i.e. $t\theta = t'\theta$).

In particular, the exclusivity rule and the totality rule are useful for resolutions with respect to negations embedded in a sort-hierarchy. The exclusivity rule will be applied when an opposite sort is declared as $s \parallel s'$. We write *resolution system RS* for the system defined by the cut rule in Definition 4.17 and the resolution rules in Definition 4.19. The following example shows an application of the exclusivity rule.

Example 4.10 *Given the ordered pair (D, Δ) with the set D in Example 4.7 and*

$$\Delta = \{winner(x{:}person), loser(tom{:}person)\},$$

the exclusivity rule is applied as follows.

[2] In one rule, $Neg.s$ and $Neg.s'$ must indicate the same negation type, i.e. $\overline{s}, \overline{s'}$ or $\sim s, \sim s'$

$$\frac{player(x{:}person) \vee winner(x{:}person) \quad loser(tom{:}person) \quad winner \parallel loser}{player(tom{:}person)}$$

with the unifier $\theta = \{tom{:}person/x{:}person\}$ *for the terms* $x{:}person$ *and* $tom{:}person$.

That is, we have

$$(D, \Delta) \vdash_{RS} player(tom{:}person)$$

by the above application of the exclusivity rule to

$$\begin{aligned}(D, \Delta) &\vdash_{RS} player(x{:}person) \vee winner(x{:}person),\\ (D, \Delta) &\vdash_{RS} loser(tom{:}person),\\ (D, \Delta) &\vdash_{RS} winner \parallel loser.\end{aligned}$$

4.3.3 Hybrid inference system with clauses and structured sort constraints

We define a hybrid inference system obtained by combining a clausal inference system with a structured sort constraint system. The deduction and resolution rules in the hybrid system are applied to clauses including sort predicates, so that it can deal with sort-hierarchy information in an assertional knowledge base where the subsort declarations are derived from an application of the rules in the structured sort constraint system.

Definition 4.20 (Hybrid inference system) *A hybrid inference system is a system obtained by adding the axioms and rules in a constraint system into a clausal inference system. We write* $X + Y$ *for the hybrid inference system obtained from a clausal inference system* X *and a constraint system* Y.

Therefore, the hybrid inference system $X + Y$ can be regarded as an extension of the clausal inference system X. We write $(D, \Delta) \vdash_{X+Y} F$ if F has a derivation from (D, Δ) in the hybrid inference system $X + Y$.

The next lemma shows the validity of the axioms in the structured sort constraint systems CS_1, CS_2.

Lemma 4.2 *The axioms of the structured sort constraint systems* CS_1, CS_2 *are valid.*

Proof. We will prove that the axiom $(s \sqcap \bot) =_S \bot$ is valid. Suppose that $\mathcal{I}^+ = (M^+, \alpha^+)$ is a Σ^+-interpretation. By Definition 4.7, $I^+(\bot) = \emptyset$ implies $I^+(s \sqcap \bot) = I^+(s) \cap I^+(\bot) = \emptyset$. Then we have $\mathcal{I}^+ \models (s \sqcap \bot) \sqsubseteq_S \bot$ and $\mathcal{I}^+ \models \bot \sqsubseteq_S (s \sqcap \bot)$. Similarly, the validity of the other axioms can be proved. ∎

Similar to Lemma 3.1, we prove that an instance $C\theta$ of clause C is a consequence of the clause C in the class of Σ-structures as follows.

Lemma 4.3 *Let* $\mathcal{I}^+$ *be a* Σ^+*-interpretation,* C *a clause, and* θ *a structured sort substitution. If* $\mathcal{I}^+ \models C$, *then* $\mathcal{I}^+ \models C\theta$. *That is,* $C \models_{\Sigma^+} C\theta$.

Proof. As for Lemma 3.1. ∎

The next lemma shows the soundness of the inference rules in the structured sort constraint systems CS_1, CS_2.

Lemma 4.4 *Let $F, F_1, \ldots, F_n$ be subsort declarations. The conclusion F of each rule in the structured sort constraint systems CS_1, CS_2 is a consequence of its premise $\{F_1, \ldots, F_n\}$ in the class of Σ^+-structures. That is, $\{F_1, \ldots, F_n\} \models_{\Sigma^+} F$.*

Proof. For each rule we show $\{F_1, \ldots, F_n\} \models_{\Sigma^+} F$.

1. Transitivity rule. Suppose that $I^+(s) \subseteq I^+(s')$ and $I^+(s') \subseteq (s'')$. Therefore $I^+(s) \subseteq I^+(s'')$.

2. Introduction rule. Assume that a Σ^+-interpretation $\mathcal{I}^+ = (M^+, \alpha^+)$ with $M^+ = (U, I^+)$ satisfies $s \sqsubseteq_S s'$. Since $I^+(s) \subseteq I^+(s')$ holds, we can obtain $I^+(s'' \sqcap s) \subseteq I^+(s'' \sqcap s')$ by Definition 4.7. Hence $\mathcal{I}^+ \models s'' \sqcap s \sqsubseteq_S s'' \sqcap s'$. Therefore $s \sqsubseteq_S s' \models_{\Sigma^+} s'' \sqcap s \sqsubseteq_S s'' \sqcap s'$.

3. Elimination rule. Suppose that $I^+(s) \cup I^+(s') = I^+(s) \cup I^+(s'')$, $I^+(s) \cap I^+(s') = \emptyset$, and $I^+(s) \cap I^+(s'') = \emptyset$. Let $d \in I^+(s')$. Since $I^+(s') \subseteq I^+(s) \cup I^+(s') \subseteq I^+(s) \cup I^+(s'')$, we have $d \in I^+(s) \cup I^+(s'')$. $d \in I(s')$ and $I^+(s) \cap I^+(s') = \emptyset$ imply $d \notin I^+(s)$. Therefore $d \in I^+(s'')$. ∎

The next lemma shows the soundness of the inference rules in the structured sort constraint systems DS, RS.

Lemma 4.5 *Let $F, F_1, \ldots, F_n$ be clauses or subsort declarations. The conclusion F of each rule in the clausal inference systems DS, RS is a consequence of its premise $\{F_1, \ldots, F_n\}$ in the class of Σ^+-structures. That is, $\{F_1, \ldots, F_n\} \models_{\Sigma^+} F$.*

Proof. For each rule, we show $\{F_1, \ldots, F_n\} \models_{\Sigma^+} F$.

1. Cut rule. Suppose that $\mathcal{I}^+ \models C \vee \neg L$ and $\mathcal{I}^+ \models L' \vee C'$. There exists a structured sort substitution θ such that $L\theta = L'\theta$, and $\mathcal{I}^+ \models (C \vee \neg L)\theta$ and $\mathcal{I}^+ \models (L' \vee C')\theta$. If $\mathcal{I}^+ \not\models L\theta$, then $\mathcal{I}^+ \not\models L'\theta$. So $\mathcal{I}^+ \models C'\theta$ holds. If $\mathcal{I}^+ \models L\theta$, then $\mathcal{I}^+ \not\models \neg L\theta$. Then we have $\mathcal{I}^+ \models \theta C$. Therefore $\mathcal{I}^+ \models C\theta \vee C'\theta$. Is follows that the conclusion of the cut rule is a consequence of its premise.

2. Deduction rules with predicate sorts:

   (1) Subsort rule. (i) Suppose $\mathcal{I}^+ \models s'(t)$ and $\mathcal{I}^+ \models s' \sqsubseteq_S s$. If $\mathcal{I}^+ \models s'(t)$, then $[\![t]\!]_{\alpha^+} \in I(s')$. By $\mathcal{I}^+ \models s' \sqsubseteq_S s$, we have $[\![t]\!]_{\alpha^+} \in I(s)$. Therefore, $\mathcal{I}^+ \models s(t)$.

   (ii) Suppose $\mathcal{I}^+ \models \neg s(t)$ and $\mathcal{I}^+ \models s' \sqsubseteq_S s$. If $\mathcal{I}^+ \models \neg s(t)$, then $[\![t]\!]_{\alpha^+} \notin I(s)$. By $\mathcal{I}^+ \models s' \sqsubseteq_S s$, we have $[\![t]\!]_{\alpha^+} \notin I(s')$. Therefore, $\mathcal{I}^+ \models \neg s(t)$.

   (3) Sort predicate. Let $\mathcal{I}^+$ be a Σ^+-model of $s' \sqsubseteq_S s$. Since $Sort(t) = s'$ we have $[\![t]\!]_{\alpha^+} \in I(s')$. Then $I(s') \subseteq I(s)$ implies $[\![t]\!]_{\alpha} \in I(s)$. Hence $\mathcal{I}^+ \models s(t)$.

3 Resolution rules with predicate sorts:

We now prove the statement only for the first type of each resolution rule (the statement for the other types can be proved by Lemma 4.1 and the proof of the first type).

Case $Neg.s = \overline{s}$:

(1) Subsort rule. Assume that $\mathcal{I}^+ \models \overline{s}(t) \vee C$, $\mathcal{I}^+ \models s'(t') \vee C'$, and $\mathcal{I}^+ \models s \sqsubseteq_S s'$. Let θ be an structured sort substitution such that $\theta(t) = \theta(t')$. By $I^+(s') \subseteq I^+(s)$, $[\![t\theta]\!]_{\alpha^+} \notin I^+(s)$ implies $[\![t'\theta]\!]_{\alpha^+} \notin I^+(s')$. If $\mathcal{I}^+ \models (\overline{s}(t))\theta$, then $\mathcal{I}^+ \models C'\theta$. If $\mathcal{I}^+ \not\models (\overline{s}(t))\theta$, then $\mathcal{I}^+ \models C\theta$. Therefore $\mathcal{I}^+ \models (C \vee C')\theta$.

(2) Sort predicate rule. Assume $\mathcal{I}^+ \models \overline{s}(t) \vee C$ and $\mathcal{I}^+ \models s' \sqsubseteq_S s$. $\mathcal{I}^+ \models s' \sqsubseteq_S s$ implies $\mathcal{I}^+ \not\models \overline{s}(t)$.

(3) Negation rule1: Assume that $\mathcal{I}^+ \models \overline{s}(t) \vee C$ and $\mathcal{I}^+ \models s'(t') \vee C'$. Let θ be an structured sort substitution such that $\theta(t) = \theta(t')$. By Definition 4.7, $[\![t\theta]\!]_{\alpha^+} \in I^+(\overline{s})$ implies $[\![t'\theta]\!]_{\alpha^+} \notin I^+(s)$. If $\mathcal{I}^+ \models (\overline{s}(t))\theta$, then $\mathcal{I}^+ \models C'\theta$. If $\mathcal{I}^+ \not\models (\overline{s}(t))\theta$, then $\mathcal{I}^+ \models C\theta$. Therefore $\mathcal{I}^+ \models (C \vee C')\theta$.

(4) Disjunction rule. Assume that $\mathcal{I}^+ \models s \sqcup s'(t) \vee C$ and $\mathcal{I}^+ \models \overline{s}(t') \vee C'$. Then, by Lemma 4.1, $\mathcal{I}^+ \models s(t) \vee s'(t) \vee C$ and $\mathcal{I}^+ \models \neg s(t') \vee C'$. Let θ be an structured sort substitution such that $\theta(t) = \theta(t')$. Similar to the proof of the cut rule, we can obtain $\mathcal{I}^+ \models s'(t)\theta \vee C\theta \vee C'\theta$.

(5) Exclusivity rule. Suppose that $\mathcal{I}^+ \models s(t) \vee C$, $\mathcal{I}^+ \models s'(t') \vee C'$, and $I^+(s) \cap I^+(s') = \emptyset$. Let θ be a structured sort substitution such that $\theta(t) = \theta(t')$. So $\mathcal{I}^+ \models (s(t) \vee C)\theta$ and $\mathcal{I}^+ \models (s'(t') \vee C')\theta$. By $I^+(s) \cap I^+(s') = \emptyset$, either $\mathcal{I}^+ \models s(t)\theta$ or $\mathcal{I}^+ \models s'(t')\theta$ does not hold. By the hypothesis, $\mathcal{I}^+ \models C\theta$ or $\mathcal{I}^+ \models C'\theta$. Therefore $\mathcal{I}^+ \models C\theta \vee C'\theta$.

(6) Totality rule. Assume that $\mathcal{I}^+ \models s_i(t) \vee C$, $\mathcal{I}^+ \models \overline{s'}(t') \vee C'$, and $\mathcal{I}^+ \models s \mid_{s_i} s'$, i.e. $I^+(s) \cup I^+(s') = I^+(s_i)$. Let θ be a structured sort substitution such that $\theta(t) = \theta(t')$. If $I^+(s) \cup I^+(s') = I^+(s_i)$, then $\mathcal{I}^+ \models s(t) \vee s'(t') \vee C$. Similar to the proof of the cut rule, we can obtain $\mathcal{I}^+ \models s'(t)\theta \vee C\theta \vee C'\theta$. Therefore the conclusion is a consequence of its premise.

Case $Neg.s(t) = \sim s$: This is proved by Lemma 4.1 and the result of above (1)-(6). ∎

The soundness of the structured sort constraint systems CS_1, CS_2, the deduction system DS and the resolution system RS can be shown by proving the soundness of each axiom and inference rule.

Theorem 4.1 *Let $H = (\mathcal{S}^+, D)$ be a sort-hierarchy declaration, Δ a set of clauses, and X a system. If $(D, \Delta) \vdash_X F$, then $(D, \Delta) \models_{\Sigma^+} F$.*

Proof. By Lemma 4.2, 4.4, and 4.5, this is proved. ∎

We give the notion of contradiction in an exclusivity relation from the sort-hierarchy. This notion is defined by deciding whether there is a contradiction between an opposite sort and its antonymous sort.

Definition 4.21 *Let $H = (\mathcal{S}^+, D)$ be a sort-hierarchy declaration, Δ a set of clauses, and X a system. (D, Δ) is said to be contradictory on an exclusivity relation if there exists structured sorts s, s' such that $(D, \Delta) \vdash_X s \parallel s'$ and $(D, \Delta) \vdash_X s(t)$ and $(D, \Delta) \vdash_X s'(t)$. (D, Δ) is said to be logically contradictory if $(D, \Delta) \vdash_X A$ and $(D, \Delta) \vdash_X \neg A$.*

The contradiction between A and $\neg A$ (corresponding to "logically contradictory" in the above definition) is defined in the usual manner of logics. We say that (D, Δ) is consistent if (D, Δ) is neither contradictory on an exclusivity relation nor logically contradictory.

Theorem 4.2 *Let $H = (\mathcal{S}^+, D)$ be a sort-hierarchy declaration and Δ a set of clauses. If (D, Δ) has a Σ^+-model, then (D, Δ) is consistent.*

Proof. Suppose that $\mathcal{I}^+$ is a Σ^+-model of (D, Δ). If (D, Δ) is contradictory on an exclusivity relation, then there exists s, s' such that $(D, \Delta) \vdash_X s \parallel s'$ and $(D, \Delta) \vdash_X s(t)$ and $(D, \Delta) \vdash_X s'(t)$. By Theorem 4.1, $\mathcal{I}^+ \models s \parallel s'$ and then $\mathcal{I}^+ \models s(t)$ and $\mathcal{I}^+ \models s'(t)$. Then $I^+(s) \cap I^+(s') = \emptyset$ but $[\![t]\!] \in I^+(s)$ and $[\![t]\!] \in I^+(s')$. If (D, Δ) is logically contradictory, then $\mathcal{I}^+ \models \neg A$ and $\mathcal{I}^+ \models A$. Hence, the both cases are contradiction to the hypothesis. Therefore (D, Δ) is consistent. ∎

In this thesis we especially require resolution systems that are more efficient than deduction systems. The next corollary guarantees that the hybrid inference systems $CS_1 + RS$ and $CS_2 + RS$ are sound.

Corollary 4.1 *Let CS be the structured sort constraint system CS_1 or CS_2, $H = (\mathcal{S}^+, D)$ a sort-hierarchy declaration, and Δ a set of clauses. If $(D, \Delta) \vdash_{CS+RS} \square$, then $(D, \Delta) \models_{\Sigma^+} \square$.*

Proof. When the empty clause $\square$ is derived, the final rule applied in the refutation must be one of the rules in the resolution system RS. We consider each case as follows:

1. Cut rule. There exists a structured sort substitution θ such that $L\theta = L'\theta$, and $(D, \Delta) \vdash_{CS+RS} \neg L$ and $(D, \Delta) \vdash_{CS+RS} L'$. So, by Theorem 4.1, we have $(D, \Delta) \models_{\Sigma^+} \neg L$ and $(D, \Delta) \models_{\Sigma^+} L'$. Now assume that $\mathcal{I}^+$ is a Σ^+-model of (D, Δ). Then $\mathcal{I}^+ \models L\theta$ and $\mathcal{I}^+ \not\models L'\theta (= L\theta)$ contradicts our assumption. Since (D, Δ) has no Σ^+-model, $(D, \Delta) \models_{\Sigma^+} \square$ is proved.

2. Resolution rules. Similar to 1. ∎

4.4 Evaluation

Section 4.4 evaluates the usefulness of our hybrid inference system for the knowledge representation to deal with implicit negations. This will be shown by derivations (using a hybrid inference system obtained by combining the systems we propose) for the examples in Chapter 2.

4.4.1 Examples of refutations

We will adopt the notion of a proof tree in order to represent a derivation process from an ordered pair (D, Δ) where D is a set of subsort declarations and Δ is a set of clauses. By the following definition, a derivation process of F from (D, Δ) can be described by a tree with the root labeled with F.

Definition 4.22 *Let X be a system and F a clause or a subsort declaration. Given an ordered pair (D, Δ) where D is a set of subsort declarations and Δ is a set of clauses. A proof tree T for F is a finite labeled tree satisfying the following:*

(1) The root is labeled with F,

(2) The nodes are labeled with clauses or subsort declarations,

(3) The leaves are labeled with axioms in X or elements from (D, Δ),

(4) If a node is labeled with F', then the children must be labeled with $F_1, \ldots, F_n$ such that
$\dfrac{F_1 \ldots F_n}{F'}$
is a rule in X.

In particular, a refutation process of the empty clause $\Box$ from (D, Δ) is shown by a proof tree with the root $\Box$. The proof trees are applicable to derivations in any inference system we have proposed.

In Example 3 and Example 4 in Section 2.2, $A \vdash_h B$ means that a conclusion B is derivable from a premise A in a sort-hierarchy h. We consider the refutation for the ordered pair $(D_h, \Delta_{A,B})$ corresponding to $A \vdash_h B$ where the set D_h of subsort declarations expresses h and the set $\Delta_{A,B}$ of clauses represents A and the negation of B. We can establish the validity of the clause B for A by the refutation of $\Delta_{A,B} = S_A \cup \{\neg B\}$ where S_A is a consistent set of clauses corresponding to A. This clause B is called a goal for the refutation. Recall that every goal is expressed by a finite sequence $L_1, \ldots, L_n$ of atoms that corresponds to the negation $\neg(L_1 \wedge \ldots \wedge L_n)$ of the conjunction of the atoms. We define the translation of goal G into the negative disjunction of the atoms $L_1, \ldots, L_n$ in G as follows.

Definition 4.23 *The function $T\colon (ATOM_{\Sigma^+})^n \to CS_{\Sigma^+}$ is defined by:*

$$T(G) = \neg L_1 \vee \neg L_2 \vee \ldots \vee \neg L_n$$

where G is a goal with $G = L_1, \ldots, L_n$.

The translation $T(G)$ results in the clause form of G to which inference rules in clausal inference systems can be applied. In the following example, we show refutations corresponding to the derivability relations (in Example 3)

$$\begin{aligned}
unhappy(bob) &\vdash_{h_1} feeling(bob),\\
\neg happy(bob) &\nvdash_{h_1} feeling(bob),\\
unhappy(bob) &\vdash_{h_1} \neg happy(bob),\\
\neg happy(bob) &\nvdash_{h_1} unhappy(bob),\\
\neg feeling(bob) &\vdash_{h_1} \neg happy(bob) \wedge \neg unhappy(bob)
\end{aligned}$$

from the sort-hierarchy h_1 shown in Figure 2.5.

Example 4.11 *Let Σ^+ be the structured signature of Example 4.2. We define the consistent sets S_1, S_2, S_3 of clauses and the finite set D of subsort declarations such that*

$$\begin{aligned}
S_1 &= \{unhappy(bob\colon person)\},\\
S_2 &= \{\neg happy(bob\colon person)\},\\
S_3 &= \{\neg feeling(bob\colon person)\},\\
D &= \{happy \sqsubseteq_S feeling, very_happy \sqsubseteq_S happy, slightly_happy \sqsubseteq_S happy,\\
&\quad unhappy \sqsubseteq_S feeling, very_unhappy \sqsubseteq_S unhappy,\\
&\quad unhappy =_S \sim happy\}.
\end{aligned}$$

The sets S_1, S_2, S_3 of clauses represents the premises in the derivability relations in Example 3, and the set D of subsort declarations expresses the sort-hierarchy h_1. We attempt to decide the validity of the following goals:

$$\begin{aligned}
G_1 &= feeling(x\colon person),\\
G_2 &= \overline{happy}(x\colon person),\\
G_3 &= unhappy(x\colon person),\\
G_4 &= \overline{happy}(x\colon person),\ \overline{unhappy}(x\colon person).
\end{aligned}$$

These goals are the conclusions of the derivability relations in Example 3. Then we can obtain the following clause forms of G_1, G_2, G_3, G_4.

$$\begin{aligned}
T(G_1) &= \neg feeling(x\colon person),\\
T(G_2) &= \neg\overline{happy}(x\colon person),\\
T(G_3) &= unhappy(x\colon person),\\
T(G_4) &= \neg\overline{happy}(x\colon person) \vee \neg\overline{unhappy}(x\colon person).
\end{aligned}$$

Using the consistent sets S_1, S_2, S_3 of clauses, the set D of subsort declarations and the clause forms $T(G_1), T(G_2), T(G_3), T(G_4)$, we consider the refutations of (D, Δ_1), (D, Δ_2), (D, Δ_3) and (D, Δ_4) where

$$\begin{aligned}
\Delta_1 &= S_1 \cup \{T(G_1)\},\\
\Delta_2 &= S_2 \cup \{T(G_1)\},\\
\Delta_3 &= S_1 \cup \{T(G_2)\},\\
\Delta_4 &= S_2 \cup \{T(G_3)\},\\
\Delta_5 &= S_3 \cup \{T(G_4)\}.
\end{aligned}$$

The following proof trees describe refutation processes of (D, Δ_1), (D, Δ_2), ...,(D, Δ_5) in the hybrid inference system $CS_2 + RS$ (obtained by combining the systems CS_2 and RS). We use the abbreviations

$$\begin{aligned}
H &= happy,\\
UH &= unhappy,\\
F &= feeling,\\
x_p &= x\colon person,\\
b_p &= bob\colon person.
\end{aligned}$$

(1) $unhappy(bob) \vdash_{h_1} feeling(bob)$

A refutation process from (D, Δ_1) *with* $\Delta_1 = S_1 \cup \{T(G_1)\}$ *is shown by the following proof tree.*

$$\frac{\neg F(x_p) \quad UH(b_p) \quad UH \sqsubseteq_S F}{\square}\ (S1)$$

(2) $\neg happy(bob) \nvdash_{h_1} feeling(bob)$

The refutation process from (D, Δ_2) *with* $\Delta_2 = S_2 \cup \{T(G_1)\}$ *fails as below.*

$$\frac{\neg F(x_p) \quad \neg H(b_p) \quad H \sqsubseteq_S F}{fail}$$

(3) $unhappy(bob) \vdash_{h_1} \neg happy(bob)$

A refutation process from (D, Δ_3) *with* $\Delta_3 = S_1 \cup \{T(G_2)\}$ *is shown by the following proof tree.*

$$\frac{\neg \overline{H}(x_p) \quad UH(b_p) \quad \dfrac{\dfrac{UH =_S \sim H}{H \sqcap UH =_S H \sqcap \sim H}\ (I) \quad H \parallel UH \quad H \parallel \sim H}{H \parallel UH}\ (E)}{\square}\ (E2)$$

(4) $\neg happy(bob) \nvdash_{h_1} unhappy(bob)$

A refutation process from (D, Δ_4) *with* $\Delta_4 = S_2 \cup \{T(G_3)\}$ *is failed below.*

$$\frac{\neg UH(x_p) \quad \neg H(b_p) \quad \dfrac{\dfrac{UH =_S \sim H}{H \sqcap UH =_S H \sqcap \sim H}\ (I) \quad H \parallel UH \quad H \parallel \sim H}{H \parallel UH}\ (E)}{fail}$$

(5) $\neg feeling(bob) \vdash_{h_1} \neg happy(bob) \wedge \neg unhappy(bob)$

A refutation process (D, Δ_5) *with* $\Delta_5 = S_3 \cup \{T(G_4)\}$ *is shown by the following proof tree.*

$$\frac{\dfrac{\neg \overline{H}(x_p) \vee \neg \overline{UH}(x_p) \quad \neg F(b_p) \quad UH \sqsubseteq_S F}{\neg \overline{H}(b_p)}\ (S3) \quad \neg F(b_p) \quad H \sqsubseteq_S F}{\square}\ (S3)$$

In the above proof trees, the applied inference rules in the hybrid inference system $CS_2 + RS$ are denoted as follows:

$$\begin{aligned}
(T) &\quad \textit{Transitivity rule},\\
(I) &\quad \textit{Introduction rule},\\
(E) &\quad \textit{Elimination rule},\\
(C) &\quad \textit{Cut rule},\\
(S1) \sim (S3) &\quad \textit{Subsort rule},\\
(P1) \sim (P2) &\quad \textit{Sort predicate rule},\\
(N1) \sim (N2) &\quad \textit{Negation rule},\\
(D1) \sim (D3) &\quad \textit{Disjunction rule},\\
(E1) \sim (E3) &\quad \textit{Exclusivity rule},\\
(T1) \sim (T3) &\quad \textit{Totality rule}.
\end{aligned}$$

In the following example, we show refutations corresponding to the derivability relations (in Example 4)

$$\begin{aligned}
loser(tom) &\vdash_{h_2} player(tom),\\
\neg winner(tom) &\nvdash_{h_2} player(tom),\\
\neg player(tom) &\vdash_{h_2} \neg winner(tom) \wedge \neg loser(tom),\\
player(tom) &\vdash_{h_2} winner(tom) \vee loser(tom),\\
player(tom) \wedge \neg loser(tom) &\vdash_{h_2} winner(tom),\\
\neg winner(tom) \wedge \neg loser(tom) &\vdash_{h_2} \neg player(tom)
\end{aligned}$$

from the sort-hierarchy h_2 shown in Figure 2.6.

Example 4.12 *Let Σ^+ be the structured sort signature of Example 4.3. We consider the consistent sets $S_1, \ldots, S_6$ of clauses and the finite set D of subsort declarations such that*

$$\begin{aligned}
S_1 &= \{loser(tom{:}person)\},\\
S_2 &= \{\neg winner(tom{:}person)\},\\
S_3 &= \{\neg player(tom{:}person)\},\\
S_4 &= \{player(tom{:}person)\},\\
S_5 &= \{player(tom{:}person),\ \neg loser(tom{:}person)\},\\
S_6 &= \{\neg winner(tom{:}person),\ \neg loser(tom{:}person)\},\\
D &= \{winner \sqsubseteq_S player, player \sqsubseteq_S person,\\
&\quad loser \sqsubseteq_S player,\\
&\quad winner \mid_{player} loser,\\
&\quad winner \parallel loser\}.
\end{aligned}$$

The sets $S_1, \ldots, S_6$ of clauses represents the premises in the derivability relations in Example 4, and the set D of subsort declarations expresses the sort-hierarchy h_2. We attempt to decide the validity of the following goals:

$$G_1 = player(x{:}person),$$

$$
\begin{aligned}
G_2 &= \overline{winner}(x{:}person),\ \overline{loser}(x{:}person),\\
G_3 &= winner(x{:}person),\\
G_4 &= loser(x{:}person),\\
G_5 &= \overline{player}(x{:}person).
\end{aligned}
$$

These goals are the conclusions of the derivability relations in Example 4. Then we can obtain the following clause forms of G_1, G_2, G_3, G_4, G_5.

$$
\begin{aligned}
T(G_1) &= \neg player(x{:}person),\\
T(G_2) &= \neg\overline{winner}(x{:}person) \vee \neg\overline{loser}(x{:}person),\\
T(G_3) &= \neg winner(x{:}person),\\
T(G_4) &= \neg loser(x{:}person),\\
T(G_5) &= \neg\overline{player}(x{:}person)
\end{aligned}
$$

Using the consistent sets $S_1, \ldots, S_6$ *of clauses, the set* D *of subsort declarations and the clause forms* $T(G_1), \ldots, T(G_5)$, *we can consider the refutations of* (D, Δ_1), (D, Δ_2), $\ldots$, (D, Δ_7) *where*

$$
\begin{aligned}
\Delta_1 &= S_1 \cup \{T(G_1)\},\\
\Delta_2 &= S_2 \cup \{T(G_1)\},\\
\Delta_3 &= S_3 \cup \{T(G_2)\},\\
\Delta_4 &= S_4 \cup \{T(G_3)\},\\
\Delta_5 &= S_4 \cup \{T(G_4)\},\\
\Delta_6 &= S_5 \cup \{T(G_3)\},\\
\Delta_7 &= S_6 \cup \{T(G_5)\}.
\end{aligned}
$$

The following proof trees describe refutation processes of (D, Δ_1), (D, Δ_2), $\ldots$, (D, Δ_7) *in the hybrid inference system* $CS_2 + RS$ *(obtained by combining the systems* CS_2 *and* RS*). We use tha abbreviation*

$$
\begin{aligned}
W &= winner,\\
L &= loser,\\
PL &= player,\\
x_p &= x{:}person,\\
t_p &= tom{:}person.
\end{aligned}
$$

(1) $loser(tom) \vdash_{h_2} player(tom)$

A refutation process from (D, Δ_1) *with* $\Delta_1 = S_1 \cup \{T(G_1)\}$ *is shown by the following proof tree.*

$$
\frac{\neg L(x_p) \quad PL(t_p) \quad L \sqsubseteq_S PL}{\square}\ (S1)
$$

(2) $\neg winner(tom) \nvdash_{h_2} player(tom)$

A refutation process from (D, Δ_2) *with* $\Delta_2 = S_2 \cup \{T(G_1)\}$ *is failed below.*

$$\frac{\neg L(x_p) \quad \neg W(t_p) \quad W \parallel L}{fail}$$

(3) $\neg player(tom) \vdash_{h_2} \neg winner(tom) \wedge \neg loser(tom)$
A refutation process from (D, Δ_3) *with* $\Delta_3 = S_3 \cup \{T(G_2)\}$ *is shown by the following proof tree.*

$$\dfrac{\dfrac{\neg\overline{W}(x_p) \vee \neg\overline{L}(x_p) \quad \neg P(t_p) \quad W \sqsubseteq_S PL}{\neg\overline{L}(x{:}P)}\,(S3) \qquad \neg P(t_p) \quad L \sqsubseteq_S PL}{\square}\,(S3)$$

(4) $\neg player(tom) \vdash_{h_2} winner(tom)$
A refutation process from (D, Δ_4) *with* $\Delta_4 = S_4 \cup \{T(G_3)\}$ *is shown by the following proof tree.*

$$\frac{\neg W(x_p) \quad PL(t_p) \quad W \mid_{PL} L}{\square}\,(T2)$$

(5) $player(tom) \vdash_{h_2} loser(tom)$
A refutation process from (D, Δ_5) *with* $\Delta_5 = S_4 \cup \{T(G_4)\}$ *is shown by the following proof tree.*

$$\dfrac{\dfrac{\neg W(x_p) \vee PL(t_p) \quad W \mid_{PL} L}{L(t_p)}\,(T1) \qquad \neg L(t_p)}{\square}\,(C)$$

(6) $player(tom) \wedge \neg loser(tom) \vdash_{h_2} winner(tom)$
A refutation process from (D, Δ_6) *with* $\Delta_6 = S_5 \cup \{T(G_3)\}$ *is shown by the following proof tree.*

$$\dfrac{\neg W(t_p) \qquad \dfrac{PL(tom{:}P) \quad \neg L(tom{:}P) \quad W \mid_{PL} L}{W(t_p)}\,(T1)}{\square}\,(C)$$

(7) $\neg winner(tom) \wedge \neg loser(tom) \vdash_{h_2} \neg player(tom)$
A refutation process from (D, Δ_7) *with* $\Delta_7 = S_6 \cup \{T(G_5)\}$ *is shown by the following proof tree.*

$$\dfrac{\dfrac{\neg\overline{PL}(t_p) \quad \neg W(t_p) \quad W \mid_{PL} L}{L(t_p)}\,(T3) \qquad \neg L(t_p)}{\square}\,(C)$$

The derivability relation $player(tom) \vdash_{h_2} winner(tom) \vee loser(tom)$ *is established by the above proof tree (4) or (5).*

Chapter 5

Conclusions and future work

This chapter discuss conclusions and future work. We summarize the major results of this thesis and suggest extensions to the logics we have proposed.

5.1 Conclusions

This thesis has presented two novel order-sorted logics that can respectively deal with (i) the classification of predicates and (ii) implicit negations in a hierarchy, to represent various structured information.

First, we have proposed an extended order-sorted logic that includes not only a sort-hierarchy but a predicate-hierarchy that can be used to provide reasoning mechanisms (property inheritance, event reasoning, and derivation of general and concrete expressions). In particular, predicate-hierarchy reasoning with flexible argument structures enhances the usefulness and the feasibility of the knowledge representation system. In order to derive hierarchical predicates with different argument structures, the inference machinery includes the manipulation of argument supplementations that can precisely distinguish *event* from *property* as two different aspects of a predicate. By defining a new supplementation operation in the two hierarchy inference rules (specialization and generalization rules of the predicate-hierarchy), we are able to deal successfully with argument manipulation. Thus we have defined an inference system based on a Horn clause resolution with hierarchical predicates. In the semantics of our logic, we interpret the predicate-hierarchy by introducing a restricted sorted structure (which we call the $H\Sigma$-structure). This structure ensures the soundness and the completeness of the resolution for our extended order-sorted logic with hierarchical predicates and eventuality. In addition, we have developed a query system constructed by the Horn clause resolution that provides the reasoning mechanism for the two examples (based on human reasoning) shown in Section 2.1.3.

Second, we have presented a hybrid inference system comprising of a clausal inference system and a structured sort constraint system. This system includes structured sort expressions composed atomic sorts, connectives, and negative operators, in order to deal with negative sorts implicitly embedded in a sort-hierarchy. To represent the implicitly negative sorts, we have proposed the notation of sort relations (subsort relation, equivalence relation, exclusivity relation, and totality relation) on the set of structured sorts, where the properties (based on lexicon negations in natural language) of implicitly negative sorts are axiomatized and declared by the exclusivity, partiality, and totality

relations to their positive sorts. Therefore, the structured sort constraint system can derive conclusions from the relationship between classical negation, strong negation, and antonyms in a sort-hierarchy. Furthermore, contradiction in the sort-hierarchy as defined by the exclusivity relation allows us to establish the relationship between opposite sorts and prove the consistency of our logic with structured sorts. This consistency allows a sound inference from a sort with respect to not only positive meaning as a subsort of the general expression but also with respect to negative meaning as an exclusive sort to its antonymous sort.

5.2 Future work

The extension to knowledge representation, the implementation of typed logic programming languages and the remaining work of formalization are discussed as follows.

(1) Knowledge representation

In addition to the event/property distinction for a predicate, we need to investigate the proper treatment of negation in predicate-hierarchy reasoning. Because of the ambiguity of predicate expressions, the meaning of negation and its reasoning is not always the same as follows:

- The negative meaning of an event is stronger than the negation of a property.
- Supplementary arguments to positive and negative assertions are differently quantified.

Moreover, the semantic models of any logic extended to treat negation in predicate-hierarchy reasoning must be newly proposed, in the same way that the Kripke semantics of constructive logic [3] with the strong negation (as a non-classical negation) was defined.

(2) Typed logic programming language

We plan to develop a logic programming language for our logic with structured sorts by restricting the structured sort formulas to the definite clause forms, and this will contribute to the implementation of a knowledge system with implicit negations. However, a program with implicit negations may cause the problem that a set of definite clauses with structured sorts is inconsistent. In some case, even though it has no negative literal a definite clause may represent a negative assertion in which an implicitly negative sort is used as the sort predicate. Programs including this would lose the property in logic programming that all the sets of definite clauses are consistent.

The integration of the two logics separately presented in this thesis is needed to implement a logic programming language that includes the notions of predicate-hierarchy, eventuality and implicit negation. In the formalization of this logic, we must consider a sorted signature with hierarchical predicates and structured sorts for the integrated language. Moreover, we must give the syntax and semantics of the language and present an extended resolution system. Although the work seems to be difficult, we contend that such notions are significant in representing knowledge naturally and precisely in logic programming languages.

(3) Formalization

The completeness of the Horn clause resolution with hierarchies and eventuality can be proved by adding the $R2^+$-resolution rule and $R3^+$-resolution rule. However, we might say that the proof procedures of these rules are semi-resolutions because not all of the clauses in the premises are reduced by an application of the rules. Therefore, we need to present a complete resolution system without the rules $R2^+$ and $R3^+$. To do this, we must extend the $R2$-resolution rule and $R3$-resolution rule to include the derivations of the rules $R2^+$ and $R3^+$.

Furthermore, the remaining work for our order-sorted logic with structured sorts is to prove the completeness theorem for the hybrid inference systems we have proposed. Since the derivations of clauses are extended to include rules of sort predicates related to subsort declarations, we have to consider the correspondence of the derivability to Σ^+-models in semantics.

Appendix A

Syntax

Let $\Sigma = (\mathcal{S}, \mathcal{F}, \mathcal{P}, \mathcal{D})$ be a sorted signature with hierarchical predicates. The order-sorted terms and formulas in this thesis are defined by the following syntax rules:

A.1 Order-sorted terms

$s, s', s_1, \ldots, s_n \in \mathcal{S}$:	sort symbols
$f \in \mathcal{F}_n$:	n-ary function symbols
$x\colon s \in \mathcal{V}_s$:	sorted variables
t, t' :	sorted terms of sort s, s'
$t_1, \ldots, t_n$:	sorted terms of sort $s_1, \ldots, s_n$

$$t ::= x\colon s \mid f(t_1, \ldots, t_n)\colon s \mid t'$$

where $f\colon s_1 \times \ldots \times s_n \to s \in \mathcal{D}_{\mathcal{F}}$ and $s' \sqsubset_S s \in \mathcal{D}_{\mathcal{S}}$.

A.2 Order-sorted formulas

$p \in \mathcal{P}_n$:	n-ary predicate symbols
$x\colon s \in \mathcal{V}_s$:	sorted variables
$a_1, \ldots, a_n \in AL$:	predicate argument labels
$t_1, \ldots, t_n$:	sorted terms of sort $s_1, \ldots, s_n$
A, A_1, A_2 :	sorted formulas
$\neg, \wedge, \vee, \to$:	the connectives
$\forall, \exists$:	the quantifiers

$$\begin{aligned} A ::= {} & p^{\bullet}(a_1 \Rightarrow t_1, \ldots, a_n \Rightarrow t_n) \mid p^{\sharp}(a_1 \Rightarrow t_1, \ldots, a_n \Rightarrow t_n) \mid \\ & \neg A \mid A_1 \wedge A_2 \mid A_1 \vee A_2 \mid A_1 \to A_2 \mid \forall x\colon s A \mid \exists x\colon s A \end{aligned}$$

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

Let $\Sigma = (\mathcal{S}^+, \mathcal{F}, \mathcal{P}, \mathcal{D})$ be a structured sort signature. The structured sorts, terms, and formulas in this thesis are defined by the following syntax rules:

A.3 Structured sorts

$s_0 \in \mathcal{S}$:	atomic sort symbols
$s, s_1, s_2 \in \mathcal{S}^+$:	structured sorts
$\sqcap, \sqcup, -, \sim$:	the connectives

$$s ::= s_0 \mid \overline{s} \mid \sim s \mid s_1 \sqcap s_2 \mid s_1 \sqcup s_2$$

A.4 Structured sort terms

$s, s_1, \ldots, s_n \in \mathcal{S}^+$:	structured sorts
$f \in \mathcal{F}_n$:	n-ary function symbols
$x{:}\, s \in \mathcal{V}_s^+$:	sorted variables
t :	structured sort terms of s
$t_1, \ldots, t_n$:	structured sort terms

$$t ::= x{:}\, s \mid f(t_1, \ldots, t_n){:}\, s$$

where $f{:}\, s_1 \times \ldots \times s_n \to s \in \mathcal{D}$.

A.5 Structured sort formulas

$p \in \mathcal{P}_n$:	n-ary predicate symbols
$(p_s \in \mathcal{P}_{\mathcal{S}^+}$:	sort predicate symbols)
$x{:}\, s \in \mathcal{V}_s^+$:	sorted variables
$t, t_1, \ldots, t_n$:	structured sort terms
A, A_1, A_2 :	sorted formulas
$\neg, \wedge, \vee, \to$:	the connectives
$\forall, \exists$:	the quantifiers

$$\begin{aligned} A ::= {} & p(t_1, \ldots, t_n) \mid p_s(t) \mid \\ & \neg A \mid A_1 \wedge A_2 \mid A_1 \vee A_2 \mid A_1 \to A_2 \mid \forall x{:}\, s A \mid \exists x{:}\, s A \end{aligned}$$

where $p{:}\, s_1 \times \ldots \times s_n \in \mathcal{D}$.

Appendix B

Examples

B.1 Predicate hierarchy 1

Background knowledge

Sort-hierarchy: $man \sqsubset_S person, woman \sqsubset_S person, wallet \sqsubset_S thing,$
$\bot \sqsubset_S man, \bot \sqsubset_S woman, \bot \sqsubset_S wallet,$
$person \sqsubset_S \top, thing \sqsubset_S \top.$

Predicate-hierarchy: $hit \sqsubset_P illegal\_act.$

Function declaration: $john: \rightarrow man, mary: \rightarrow woman, c: \rightarrow wallet.$

Predicate declaration: $hit: \{(agt, person)\},$
$illegal\_act: \{(agt, person), (coagt, person)\}.$

Knowledge base

Horn clauses: $hit^{\bullet}(agt \Rightarrow john: man).$

B.2 Predicate hierarchy 2

Background knowledge

Sort-hierarchy: $man \sqsubset_S person, woman \sqsubset_S person, wallet \sqsubset_S thing,$
$\bot \sqsubset_S man, \bot \sqsubset_S woman, \bot \sqsubset_S wallet,$
$person \sqsubset_S \top, thing \sqsubset_S \top.$

Predicate-hierarchy: $rob\_with\_violence \sqsubset_P hit,$
$rob\_with\_violence \sqsubset_P steal,$
$hit \sqsubset_P illegal\_act,$
$steal \sqsubset_P illegal\_act.$

Function declaration: $john: \rightarrow man, mary: \rightarrow woman, c: \rightarrow wallet.$

Predicate declaration: $rob\_with\_violence\colon \{(agt, person), (coagt, person), (obj, thing)\}$,
$hit\colon \{(agt, person), (coagt, person)\}$,
$steal\colon \{(agt, person), (obj, \top)\}$,
$illegal\_act\colon \{(agt, person)\}$.

Knowledge base

Horn clauses: $hit^{\bullet}(agt \Rightarrow john\colon man, coagt \Rightarrow mary\colon woman)$,
$steal^{\bullet}(agt \Rightarrow john\colon man, obj \Rightarrow c\colon wallet)$.

B.3 Event and property interpretations

Background knowledge

Sort-hierarchy: $penguin \sqsubset_S bird, crow \sqsubset_S bird, bird \sqsubset_S animal$,
$bot \sqsubset_S penguin, \bot \sqsubset_S crow, \bot, animal \sqsubset_S \top$.

Predicate-hierarchy: $fly \sqsubset_P move$,
$walk \sqsubset_P move$.

Function declaration: $c\colon \rightarrow bird$.

Predicate declaration: $fly\colon \{(sbj, \top)\}$,
$walk\colon \{(sbj, \top)\}$,
$move\colon \{(sbj, \top)\}$.

Knowledge base

Horn clauses: $fly^{\bullet}(sbj \Rightarrow c\colon bird)$,
$fly^{\sharp}(sbj \Rightarrow x\colon bird)$.

B.4 A criminal case

Background knowledge

Sort-hierarchy: $man \sqsubset_S person, woman \sqsubset_S person, bat \sqsubset_S thing$,
$\bot \sqsubset_S man, \bot \sqsubset_S woman, \bot \sqsubset_S bat, person \sqsubset_S \top, thing \sqsubset_S \top$.

Predicate-hierarchy: $die \sqsubset_P legal\_act$,
$hit \sqsubset_P illegal\_act$,
$murder \sqsubset_P illegal\_act$,
$illegal\_act \sqsubset_P act$.

Function declaration: $john\colon \rightarrow man$,
$mary\colon \rightarrow woman$,
$c\colon \rightarrow bat$.

Predicate declaration: $die\colon\{(agt, person)\}$,
$legal\_act\colon\{(agt, person)\}$,
$illegal\_act\colon\{(agt, person)\}$,
$hit\colon\{(agt, person), (coagt, person), (tool, thing), (place, \top)\}$,
$murder\colon\{(agt, person), (coagt, person)\}$,
$intent\_to\_murder\colon\{(agt, person), (coagt, person)\}$,
$act\colon\{(agt, person), (coagt, person)\}$.

Knowledge base

Horn clauses:

$hit^{\bullet}(agt = john\colon man, coagt = mary\colon woman, tool = c_1 : bat, place = c_2 : home)$,
$die^{\bullet}(agt = mary\colon woman)$,
$intent\_to\_murder^{\bullet}(agt = john\colon man, coagt = mary\colon woman)$,
$murder^{\bullet}(agt = x : person, coagt = y : person) \leftarrow$
$act^{\bullet}(agt = x : person, coagt = y : person)$,
$die^{\bullet}(agt = y : person)$,
$intent\_to\_murder^{\bullet}(agt = x : person, coagt = y : person)$.

B.5 Underage drinking

Background knowledge

Sort-hierarchy: $adult \sqsubset_S person, minor \sqsubset_S person, beer \sqsubset_S alcoholic$,
$alcoholic \sqsubset_S thing, bar \sqsubset_S space$,
$\bot \sqsubset_S adult, \bot \sqsubset_S minor, \bot \sqsubset_S beer$,
$person \sqsubset_S \top, thing \sqsubset_S \top space \sqsubset_S \top$,

Predicate-hierarchy: $underage\_drinking \sqsubset_P illegal\_act$,
$legal\_act \sqsubset_P act$.

Function declaration: $peter\colon \rightarrow minor, mary\colon \rightarrow adult,,$

Predicate declaration: $\{illegal\_act\colon\{(agt, person)\}$,
$drink\colon\{(agt, person), (obj, thing), (place, \top)\}$,
$underage\_drinking\colon\{(agt, person)\}$,
$act\colon\{(agt, person), (coagt, person)\}$.

Knowledge base

Horn clauses:

$drink^{\bullet}(agt = peter\colon minor, obj = y\colon beer)\}$,
$underage\_drinking^{\bullet}(agt = x\colon minor) \leftarrow drink^{\bullet}(agt = x\colon minor, obj = y\colon alcoholic)$,
$drink^{\sharp}(agt = x\colon adult, obj = y\colon alcoholic)$.

B.6 Negative affix: unhappy

Background knowledge

Sort-hierarchy: $unhappy =_S \sim happy$,
$happy \sqsubseteq_S feeling$,
$unhappy \sqsubseteq_S feeling$.

Function declaration: $bob: \rightarrow person$.

Predicate declaration: $p_{feeling}: \top,\ p_{happy}: \top,\ p_{unhappy}: \top, \ldots$

Knowledge base

Clauses:
(1) $unhappy(bob: person), \neg feeling(x: person)$.
(2) $\neg happy(bob: person), \neg feeling(x: person)$.
(3) $unhappy(bob: person), \neg\overline{happy}(bob: person)$.
(4) $\neg happy(bob: person), unhappy(x: person)$.
(5) $\neg feeling(bob: person), \neg\overline{happy}(bob: person) \vee \neg\overline{unhappy}(bob: person)$.

B.7 Lexicon with negative meaning: loser

Background knowledge

Sort-hierarchy: $winner \sqsubseteq_S player$,
$loser \sqsubseteq_S player$,
$winner \mid_{player} loser$,
$winner \parallel loser$.

Function declaration: $tom: \rightarrow person$,
$father: person \rightarrow person$.

Predicate declaration: $p_{person}: \top,\ p_{player}: \top,\ p_{winner}: \top,\ p_{loser}: \top, \ldots$

Knowledge base

Clauses:
(1) $loser(tom: person), \neg player(x: person)$.
(2) $\neg winner(tom: person), \neg player(x: person)$.
(3) $\neg player(tom: person), \neg\overline{winner}(x: person) \vee \neg\overline{loser}(x: person)$.
(4) $player(tom: person), \neg winner(x: person)$.
(5) $player(tom: person), \neg loser(x: person)$.
(6) $player(tom: person), \neg loser(tom: person), \neg winner(x: person)$.
(7) $\neg winner(tom: person), \neg loser(tom: person), \neg\overline{player}(x: person)$.

Bibliography

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

[2] H. Aït-Kaci and A. Podelski. Towards a meaning of life. *Journal of Logic Programming*, 16(3&4):195–234, 1993.

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

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

[5] J. F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23:123–154, 1984.

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

[7] F. Baader and P. Hanschke. A scheme for intergrating concrete domains into concept languages. In *Twelfth Internatinal Conference on Artificial intelligence*, pages 452–457, 1991.

[8] F. Baader and U. Sattker. Expressive number restrictions in description logics. *Journal of Logic and Computation*, 9(3):319–350, 1999.

[9] D. Basin, S. Matthews, and L. Vigano. Labelled propositional modal logics: Theory and practice. *Journal of Logic Computation*, 7(6):685–717, 1997.

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

[11] R. J. Brachman. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, 1999.

[12] M. Buchheit, F. M. DOnini, and A. Schaerf. Decidable reasoning in terminological knowledge representation system. In *Proceedings of IJCAI'93*, pages 704–709, 1993.

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

[14] A. G. Cohn. A more expressive formulation of many sorted logic. *Journal of Automated Reasoning*, 3:113–200, 1987.

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

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

[17] F. D. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. Reasoning in description logic. In G. Brewka, editor, *Principles of Knowledge Representation*. CSLI Publications, FoLLI, 1996.

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

[19] G. Fauconnier. *Espas Mentaux*. Editions de Minuit, 1984.

[20] Alan M. Frisch. The substitutional framwork for sorted deduction: fundamental results on hibrid reasoning. *Artificial Intelligence*, 49:161–198, 1991.

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

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

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

[24] W. Hodges. Logical feature of Horn clauses. In Dov M. Gabby, C. J. Hogger, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming Vol.1*. Oxford University Press, 1993.

[25] I. Horrocks. A description logic with transitive and inverse roles and role hierarchies. *Journal of Logic and Computation*, 9(3):385–410, 1999.

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

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

[28] D. Maier. A logic for objects. In *In Workshop on Foundations of Deductive Databases and Logic Programming*, Washington D.C., 1986.

[29] M. Manzano. Introduction to many-sorted logic. In *Many-sorted Logic and its Applications*, pages 3–86. John Wiley and Sons, 1993.

[30] A. Martelli and U. Montanari. An efficient unification algorithm. *ACM Trans. Programm. Languages Systems*, 4:258–282, 1982.

[31] D. V. McDermott. A temporal logic for reasoning about processes and plans. *Cognitive Science*, 6:101–155, 1982.

[32] M. Minsky. A framework for representing knowledge. In J. Haugeland, editor, *Mind Design*. The MIT Press, 1981.

[33] K. Nitta and 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.

[34] K. Nitta, Y. Ohtake, S. Maeda, M. Ono, H. Osaki, and K. Sakane. HELIC-II: Legal reasoning system on the parallel inference machine. *New Gener. comput.*, 11(34):423–448, 1993.

[35] K. Nitta, S. Tojo, and et al. Knowledge representation of new HELIC-II. In *Workshop on Legal Application of Logic Programming,ICLP '94*, 1994.

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

[37] A. Oberschelp. Order sorted predicate logic. In *Workshop on Sorts and Types in Artificial Intelligence*, 1989.

[38] A. Ota. *Hitei No Imi (in Japanese)*. Taishukan, 1980.

[39] M. R. Quilian. Semantic memory. In M. Minsky, editor, *Semantic Information Processing*. The MIT Press, 1968.

[40] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, 1995.

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

[42] M. Schmidt-Schauss and G. Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48:1–26, 1991.

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

[44] G. Smolka. Feature-constraint logics for unification grammars. *Journal of Logic Programming*, 12:51–87, 1992.

[45] R. Socher-Ambrosius and P. Johann. *Deduction Systems*. Springer-Verlag, 1996.

[46] I. S. Torsun. *Foundations of Intelligent Knowledge-Based Systems*. Academic Press, 1995.

[47] H. Tsuda. *Studies on Logic Programming Language Constraint-based Natural Language Analysis*. PhD thesis, The University of Tokyo, 1997.

[48] G. Wagner. Logic programming with strong negation and inexact predicates. *Journal of Logic Computation*, 1(6):835–859, 1991.

[49] G. Wagner. *Vivid Logic:Knowledge-Based Reasoning with Two Kinds of Negation*. Springer-Verlag, 1994.

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

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

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

[53] H. Wang. Logic of many-sorted theories. *Journal of Symbolic Logic*, 3:105–116, 1952.

[54] H. Yasukawa, H. Tsuda, and K. Yokota. Objects, properies, and modules in $\mathcal{QUIXOTE}$. In *Proc. FGCS'92*, pages 257–268, 1992.

[55] K. Yokota. *Quixote:A Constraint Based Approach to a Deductive Object-Oriented Database*. PhD thesis, Kyoto University, 1994.