Artificial Intelligence and Law **5:** 97–118, 1997.

A Goal-Dependent Abstraction for Legal Reasoning by Analogy

TOKUYASU KAKUTA
Department of Systems Science, Tokyo Institute of Technology, 4259 Nagatsuta, Midori-ku, Yokohama 226, Japan
E-mail: kaku@int.titech.ac.jp

MAKOTO HARAGUCHI and YOSHIAKI OKUBO
Division of Electronics and Information Engineering, Hokkaido University, N-13, W-8, Sapporo 060, Japan
E-mail: {makoto, yoshiaki}@db.huee.hokudai.ac.jp

Abstract. This paper presents a new algorithm to find an appropriate similarity under which we apply legal rules analogically. Since there may exist a lot of similarities between the premises of rule and a case in inquiry, we have to select an appropriate similarity that is *relevant* to both the legal rule and a top goal of our legal reasoning. For this purpose, a new criterion to distinguish the appropriate similarities from the others is proposed and tested. The criterion is based on *Goal-Dependent Abstraction* (GDA) to select a similarity such that an abstraction based on the similarity never loses the necessary information to prove the *ground* (purpose of legislation) of the legal rule. In order to cope with our huge space of similarities, our GDA algorithm uses some constraints to prune useless similarities.

Key words: legal reasoning, analogy, similarity, order-sorted logic, taxonomic hierarchy, goal-dependent abstraction

1. Introduction

Juristic judgments seem to depend on a large amount of common sense knowledge and to use various kinds of criteria to decide which conclusion better suits for a case in inquiry. Many legal reasoning systems in Japan have been designed and implemented (Yoshino et al., 1993; Ohtake et al., 1994; Nitta et al., 1993; Haraguchi, 1995). Those systems have a special kind of knowledge, *taxonomic hierarchy* about legal concepts, that has been used to draw legal conclusions deductively or analogically. According to the type of information the taxonomy has, the reasoning systems perform the deduction efficiently and handle hypotheses in analogical reasoning. In the studies (Ohtake et al., 1994; Haraguchi, 1995), for instance, analogy is regarded as a combined process of *generalization* and *deduction* along the hierarchy (for more details, see Appendix A). Thus, the taxonomic hierarchy plays a very important role in the field of analogical legal reasoning.

A taxonomic hierarchy can be viewed as a representation of similarities between conceptual classes depending on our way to define the concepts. For example, if we define the concept "*boy*" as a "*man*" whose age is young, then "*boy*" should be placed under "*man*". Moreover, every subconcept of "*man*" (including "*boy*") will share the properties of "*man*", for they inherit all the properties of "*man*" from their definition. The similarity thus originates in the definitional structure of concepts. In this sense, such a similarity is said to be *definitional*.

Using the definitional similarities represented by our taxonomic hierarchy, some reasoning systems have been designed to perform analogical interpretation of legal rules (Haraguchi, 1995) and case-based reasoning (Ohtake et al., 1994). In general, since the definitional similarities are standard and objective for most people, the reasoning result produced by the systems would gain more credit.

However, the definitional similarities apart, we have various kinds of similarities between concepts. This means that there might exist many interesting similarities that are not represented by our standard taxonomic hierarchy. Our legal conclusion will be affected by similarities we consider the most important and relevant, just as the studies on legal case-based reasoning systems have already pointed out (Ashley, 1990). Since using only the standard taxonomic hierarchy would be too weak to handle the variety of similarities, a reasoning system should have an ability to detect *non-definitional* similarities. The purpose of this paper is to present a computer program that can find such non-definitional similarities and apply either legal rules or case rules based on the detected similarities.

Before showing the outline of our new approach, some related methodologies (Rouveirol and Puget, 1990; Muggleton, 1990) should be discussed here. Since a similarity can be considered as a set of shared properties between two or more concepts, it is possible to find it by forward reasoning (Inoue et al., 1993; Fujita and Hasegawa, 1991) from the concepts. For instance, a property "*can_contain*" will be derived from each of the premises "*glass_cup*" and "*aluminium_cup*". Thus they share the property "*can_contain*". However, it should be emphasized that:

> Although the forward reasoning to infer the shared properties is theoretically enough to obtain every possible similarity (including definitional ones), it presents no direct way to reject *inappropriate* similarities. It is so important for our reasoning system to have a mechanism to distinguish *relevant* similarities from the inappropriate ones and to examine only relevant ones. This is indeed necessary, since our space of similarities is very huge in general.

For instance, the property "*can_contain*" is not relevant to another property "*breakable*". If we are talking about breakable things, "*can_contain*" plays no role in obtaining a useful consequence and, therefore, should be rejected as an inappropriate similarity. The previous studies of machine learning and non-monotonic reasoning have introduced a set of *integrity constraints* to exclude such inappropriate ones. We can exclude inappropriate ones if our reasoning results based on them violate the constraints. However, it is not certain whether we can succeed in feeding a sufficient set of integrity constraints to our systems in order to exclude

the inappropriate ones. So we propose in this paper another approach based on *Goal-Dependent Abstraction* (Okubo and Haraguchi, 1994b).

Abstraction (Plaisted, 1981; Okubo and Haraguchi, 1994a) is a process for obtaining a more general description about concepts by focusing on their important aspects and disregarding any other aspects. In other words, only the important aspects are extracted and preserved in the abstracted descriptions of the concepts. This paper inherits the framework of theory abstraction proposed by Tenenberg (1989). According to the framework, only *a part of* clauses in the original domain theory describing concepts is abstracted along a given conceptual hierarchy. An abstract theory is then defined as a collection of such abstracted clauses. The remaining clauses are forgotten in the abstracted theory. Intuitively speaking, we are allowed to abstract a clause C in our domain theory, if all clauses that are similar to C under the hierarchy can logically be derived from the theory. For example, let us assume that we have a conceptual hierarchy in which a concept *cup* dominates both concepts *glass_cup* and *aluminum_cup*. Furthermore, assume our domain theory consists of the following clauses:

"*glass_cup can be used to drink*" (1)

"*aluminum_cup can be used to drink*" (2)

"*glass_cup is breakable*". (3)

The clause 1 is abstracted into a clause

"*cup can be used to drink*", (4)

since it can be derived from the domain theory that *aluminum_cup* (that is similar to *glass_cup* under the hierarchy) can also be used to drink. Similarly, the clause 2 can be abstracted into 4 as well. On the other hand, since it cannot be derived from the theory that *aluminum_cup* is breakable, we are not allowed to abstract the clause 3.

Goal-Dependent Abstraction (GDA for short), originally proposed by Okubo and Haraguchi (1994b) and revised in this part, is based on Tenenberg's abstraction. Therefore, it has the same characterization as Tenenberg's one we have just stated.

Assuming such a characterization of abstraction, we can now show our basic idea to extract only appropriate (useful) similarities by GDA.

1.1. FINDING APPROPRIATE SIMILARITIES BY GDA: BASIC IDEA

We first assume that each similarity is encoded as a hierarchy of concepts. For instance, if we assert the concepts "*ceramic_cup*" and "*glass_bottle*" are similar, then we form a new super concept "α" of them. The intended meaning of "α" would be *containers made of delicate material*. Such a hierarchy is considered to be *hypothetical* in the sense that it may not be regarded as an appropriate one. Technically speaking, we encode such a hypothetical hierarchy as a *sort mapping*,

as defined in the next section. For example, the above hierarchy is represented as a sort mapping φ such that $\varphi(\textit{ceramic\_cup}) = \varphi(\textit{glass\_bottle}) = \alpha$.

Given a hypothetical hierarchy and a domain theory, we examine whether the similarity represented by the hierarchy is *appropriate for a goal* or not, according to the following process:

1. We compute an abstract theory based on the framework of Tenenberg. Then we classify the original theory into two groups. One group contains clauses that are abstracted and, therefore, preserved in the abstraction. The other group just contains clauses that are not abstracted and, therefore, forgotten in the abstraction.
2. We postulate a criterion to distinguish appropriate and useful similarities from the others. We consider that a similarity is *appropriate for a goal* G if an explanation (proof) structure of G is preserved in the abstraction based on the corresponding hypothetical hierarchy. Based on an appropriate similarity for G, we can obtain an abstract theory from which the original proof of G can be reconstructed. That is, the abstraction based on the similarity never loses the information necessary to prove G. On the other hand, an unintended abstraction based on another inappropriate hierarchy might lose some information. From the observation, we postulate the condition for our appropriateness of hypothetical hierarchy to preserve the explanation structure of a given goal. This is the criterion we propose in this paper. Since an appropriate abstraction depends on each given goal, we call our approach Goal-Dependent Abstraction: GDA.

Let us consider a similarity between s and s'. As explained previously, we are allowed to abstract a clause C if all clauses are similar to C can be derived from our domain theory. Therefore, if an explanation structure of a goal G can be preserved in the abstraction based on the similarity, the goal G can be derived for the cases of both s and s' in the same way (proof structure). From this viewpoint, we call the above criterion for appropriateness *Substitutability Condition.*

We illustrate here the process for finding appropriate similarities with a simple example. Assume that we have a domain theory consisting of the following clauses:

"*glass_cup has a handle*" (5)

"*aluminum_cup has a handle*" (6)

"*ceramic_cup has a handle*" (7)

"*glass_cup is light*" (8)

"*aluminum_cup is light*" (9)

"*ceramic_cup is light*" (10)

"*glass_cup is breakable*" (11)

"*ceramic_cup is breakable*" (12)

"*if an object has a handle and is light, then it is liftable*" (13)

"*if an object is breakable, then it should be treated carefully*". (14)

Suppose that we are given a goal "*we can lift glass_cup*". For the goal, let us examine a hypothetical hierarchy φ_1 in which a concept α dominates *aluminum_cup*, *ceramic_cup* and *glass_cup*. An explanation (proof) of the goal consists of the clauses (5), (8) and (13). We can easily observe that the explanation can be preserved in the abstraction based on the hierarchy φ_1. Therefore, φ_1 is appropriate for the goal.

On the other hand, let us examine φ_1 for another goal *we should treat ceramic_cup carefully*. An explanation of the goal consists of the clauses (12) and (14). The explanation is not preserved since we cannot be allowed to abstract the clause (12). Therefore, φ_1 is not appropriate for the latter goal. The reader can verify that a hypothetical hierarchy φ_2 in which a concept β dominates only both *ceramic_cup* and *glass_cup* is appropriate for the latter goal.

Thus an appropriate hierarchy is altered depending on a goal we select. Therefore, it is very important to prescribe what is regarded as a goal for our GDA. This problem is concerned with the *purposes* of legal rules and our reasoning; the analogical interpretation of rules. So we turn our discussion to the problem of rule application.

1.2. GROUND OF LEGAL RULE: A GOAL FOR GDA

The legal rules we consider in this paper are supposed to have the following form, for instance:

> If a requirement $A(x : s)$ is satisfied, then it must be $X(x : s)$ or must perform an action $X(x : s)$,

where A and X are predicates, x is a variable ranging over individual objects that are instances of a concept s. A symbol like s is also called a *sort symbol*. Thus the expression $x : s$, that can read "x of s", simply says that x is an instance of s. Conversely, each concept is defined as a set of instances, according to the extensional meaning of concepts. Thus, the following rule

$$\text{if } A(x : \texttt{vehicle}) \text{ then } X(x), \tag{15}$$

means that X holds for an individual vehicle x whenever A does for x, where the sort `vehicle` is a set of individual vehicles.

Moreover, such an "*is_an_instance_of*" relationship is extended, using a so called *inheritance procedure*:

> Suppose x is an instance of s and that s is a subclass of s_1, then we can conclude that x is also an instance of the superclass s_1 of s.

For we consider the extensional meaning of concept, "*is_a_subclass_of*" relationship is understood as a subset relationship between the concepts as sets, and is formally represented as a sort hierarchy, a partially ordered set of sort symbols. Using the procedure, we can get $X(x : \texttt{tank})$ from $A(x)$ provided it is derived from our sort hierarchy that `tank` is a subclass of `vehicle`.

On the other hand, we cannot apply the rule (15) to an individual horse x of `horse`, even if $A(x)$ holds. This is because `horse` is never a `vehicle`.* In such a case, our system tries to apply the rule *analogically* by seeking a similarity between `vehicle` and `horse`. For this purpose, our GDA algorithm is invoked so that we can find an appropriate hypothetical hierarchy containing some α having `vehicle` and `horse` as its subconcepts. The question is what kinds of goals should be given to our GDA algorithm to compute such a similarity (hierarchy) relevant to the legal rule.

Now let us turn our discussion to the problem. In order to concretely discuss the point, let us consider a legal rule "*Vehicles are prohibited from being in a public park*" (Hart, 1958). Moreover, suppose a sort (conceptual) hierarchy such that `car` and `bus` have the super sort `vehicle`, and `horse` and `dog` have the super sort `animal`. This example hierarchy would be considered natural and objective for most people. We can apply the rule to any individual belonging to `car` or `bus`, since we can know that it is an individual of `vehicle` according to the standard hierarchy. On the other hand, the rule cannot be applied to the individuals of `horse`, since `horse` is not a kind of `vehicle` in the hierarchy. In usual legal reasoning, however, it would be natural to apply the rule to the case of `horse` as well. In order to analyze the reason why such an analogical application is natural, we have to take *the reason why the rule has been established* into consideration.

Let us assume a situation where the legal rule is violated, that is, a vehicle enters the park. Such a situation would be considered unpleasant for most people, as it is *dangerous*, *noisy* and the air is *polluted*, etc. We consider that the legal rule has been established to prevent these undesirable circumstances. We call the set of such circumstances the *ground* of the legal rule and consider it as a goal given to our GDA. That is, GDA tries to find a similarity for the undesirable circumstances.

As previously mentioned, if GDA finds an appropriate similarity between sorts s_1 and s_2 for a goal G, it is implied that the goal G can be proved for the cases of both s_1 and s_2. Therefore, a similarity between `vehicle` and `horse` for the ground is found by GDA: `horse` would cause the same undesirable circumstances as `vehicle` would do. In such a case, it would be natural to analogically apply the legal rule to the case of `horse`, since the legal rule has been established to prevent the circumstances.

* One might claim that a horse can be a means of transportation and is, therefore, a vehicle. However, in our sort hierarchy based on which the inheritance procedure is working, we distinguish "*is_an_instance_of*" and "*is_a_subclass_of*" from "*can_be_viewed_as*" or "*plays_a_role_of*" relationships. It is indeed a subject of this paper to connect `horse` and `vehicle` based on a functional similarity between them.

The process for identifying the ground of legal rule is precisely discussed later.

1.3. LEGAL REASONING BY ANALOGY BASED ON GDA

Now we can summarize the whole process of our legal reasoning by analogy. Our legal reasoning is carried out according to the following four steps. Each step is briefly illustrated with a simple example of the vehicle in the park problem.

(1) Detecting legal rule that causes unification failure
Proving the legal goal, our system detects a legal rule we fail to apply. Technically speaking, such a failure of rule application is formalized as a *unification failure*, as precisely described later. The detected legal rule is a subject of analogical application.

Let us assume that we have a legal rule such that "*a vehicle is prohibited from being a park*". Moreover, assume that our legal goal is "*a horse cannot be in the park*". As previously discussed, since there is no "*a_subclass_of*" relation between `horse` and `vehicle` in our conceptual hierarchy, we cannot apply the legal rule to the case of `horse`. Namely, our legal goal cannot be proved without analogical application of the rule. Therefore, the legal rule is considered as a subject of analogical application and is handed to the next step.

(2) Identifying the ground of legal rule
We infer possible undesirable circumstances that are caused when the detected legal rule is violated. The set of these circumstances is the ground of the legal rule.

Let us consider the ground of the legal rule in our example. In order to obtain the ground, we try to observe what happens if a vehicle enters the park. Technically speaking, the legal rule in our domain theory is replaced by its negation, and then the logical consequences of the new theory are tried to obtain by forward reasoning. As a result, we would observe some circumstances, for instance,

"*the vehicle attracts notice*",

"*many people consider the vehicle to be dangerous*",

"*children are interested in the vehicle*", etc.

From such a set of various circumstances, we extract a collection of *undesirable* ones. To distinguish undesirable ones from the others, we suppose that a set of undesirable properties is given beforehand. According to such properties, we select undesirable circumstances. In the example, if "*dangerous*" is registered, as an undesirable property, the second one,

"*many people consider the vehicle to be dangerous*",

would be extracted and is considered as the ground of the legal rule.

(3) Finding appropriate similarity to resolve the unification failure
Given the ground of the legal rules as a goal, GDA tries to find a hypothetical hierarchy (an appropriate similarity) that makes the legal rule analogically applicable. In a word, GDA tries to find all concepts such that for each of the concepts, an explanation (proof) structure of the ground can be preserved. In this sense, these concepts are considered to be similar to each other with respect to the ground. Technically speaking, GDA finds a similarity that can resolve the unification failure detected at the first step. It should be noted here that there might exist a case where GDA fails to find such a similarity. In such a case, we consider that an analogical application of the legal rule would not be adequate for proving our legal goal.

Let us consider an explanation of the ground in our example. Suppose our domain theory contains the following knowledge:

"*vehicles are bigger than human*",

"*vehicles are movable*",

"*a vehicle is in the park*" and

"*many people consider an object to be dangerous if the object is bigger than human, movable and in the park*".

The ground, "*many people consider the vehicle to be dangerous*", can be explained from the knowledge. Furthermore, suppose the following knowledge is also contained in our theory:

"*horses are bigger than human*",

"*horses are movable*" and

"*a horse is in the park*".

We can easily see that the explanation structure of the ground is preserved for the concept `horse`. That is, "*many people consider the horse to be dangerous*" can be explained in the same way. Therefore, we consider that `horse` and `vehicle` are similar to each other.

(4) Applying the legal rule based on the appropriate similarity
Based on the appropriate similarity found by GDA, our system analogically applies the rule in order to prove our legal goal according to the framework of *Sorted-Generalization* (Haraguchi, 1995).

In our example, since the similarity between `horse` and `vehicle` is found out by GDA, we analogically apply the original legal rule to the case of `horse`. As a result, we obtain the legal conclusion "*the horse is prohibited from being in the park*".

In this paper, we especially focus on the first three steps (the last step has been precisely investigated in the literature (Haraguchi, 1995)). The technical aspects will be described in the following sections with a simple illustration.

This paper is organized as follows. Since our underlying language to describe abstraction and to present our GDA algorithm is an *order-sorted first order language*, we present fundamental terminology and definitions in Section 2. In Section 3, our approach is precisely discussed. We illustrate with a simple example how our GDA algorithm behaves and what constraint GDA uses to prune useless similarities in our huge search space. In Section 4, we report our experimental results. In the final section, we discuss the future researches based on the experimentation.

2. Terminology and definitions

In this section, we present some fundamental definitions to describe our GDA.

2.1. ORDER-SORTED LANGUAGE

First we introduce a function-free *order-sorted language*. As we have mentioned in the previous section, our methodology concerns a conceptual hierarchy. According to an order-sorted representation, a concept and a conceptual hierarchy are coded as a *sort symbol* $s \in S$ and a partially ordered set $(S, \preceq)$ of sort symbols, respectively, where S is the set of all sort symbols. Moreover, the partial order $s_1 \preceq s_2$ means that the concept denoted by s_1 is a subconcept of s_2. Each sort s can be understood as a set of possible instances $a_1, a_2, \ldots$ of s. Assuming such an order-sorted signature, our domain theory is defined as a collection of order-sorted clauses of the form

$$A \leftarrow B_1, \ldots, B_k,$$

where A and B_j are positive literals of the form $p(t_1, \ldots, t_m)$. Each term t_j is a variable or an instance both of which are typed by some sort.

2.2. SORT MAPPING

In addition to our sort hierarchy, we have to handle a hypothetical hierarchy representing our similarities. It is defined as a *sort mapping* φ: $S \rightarrow S'$, where S' is a set of new sort symbols not appearing in S. Two sorts s_1 and s_2 is said to be *similar* under φ if $\varphi(s_1) = \varphi(s_2)$. Thus we consider that the hierarchy represents a similarity. In what follows, the terms "*similarity*", "*hypothetical hierarchy*" and "*sort mapping*" are used alternatively. A sort mapping φ is easily extended to a mapping over a set of clauses as follows:

$$\varphi(p(x : s)) = p(x : \varphi(s))$$
$$\varphi(A \leftarrow B_1, \ldots, B_n) = \varphi(A) \leftarrow \varphi(B_1), \ldots, \varphi(B_n).$$

2.3. ABSTRACTION BASED ON SORT MAPPING

As we have discussed before, our GDA inherits the characterization from the abstraction framework of Tenenberg (1989). So we introduce here the abstraction framework. Exactly speaking, an order-sorted version of the framework is presented.

Given a sort mapping φ: $L \mapsto L'$ and an order-sorted theory T in L, we can transform T into an abstract theory in L' according to the following definition:

DEFINITION 1 (SortAbs). *SortAbs*$_\varphi(T)$ defined as follows is called the *abstract clause set of* T *based on* φ:

$$\textit{SortAbs}_\varphi(T) = \{C' \mid \forall C \quad C \in \varphi^{-1}(C') \rightarrow T \vdash C\},$$

where a clause C is said to be an *instance* of C' if $\varphi(C) = C'$, and $\varphi^{-1}(C')$ is defined as the set of instances of C', that is, $\varphi^{-1}(C') = \{C \mid \varphi(C) = C'\}$. □

Intuitively speaking, *SortAbs*$_\varphi(T)$ consists of only clauses such that every possible instance can be proved from T. For instance, suppose we have a sort mapping

$$\varphi : \{\texttt{car, bus, bicycle}\} \mapsto \{\alpha\}.$$

A clause

$$\texttt{movable}(\_ : \texttt{car}) : -\texttt{true} \tag{16}$$

is an instance of

$$\texttt{movable}(\_ : \alpha) : -\texttt{true}. \tag{17}$$

Moreover, the clause (17) is contained in *SortAbs*$_\varphi(T)$ if T can prove the following two clauses in addition to the clause (16):

$$\texttt{movable}(\_ : \texttt{bus}) : -\texttt{true}. \tag{18}$$

$$\texttt{movable}(\_ : \texttt{bicycle}) : -\texttt{true}. \tag{19}$$

It should be noted here that, if we add a new sort mountain, for which

$$\texttt{movable}(\_ : \texttt{mountain}) : -\texttt{true}. \tag{20}$$

is not provable, to the similarity class

$$\varphi : \{\texttt{car, bus, bicycle, mountain}\} \mapsto \{\alpha\},$$

then the clause (17) is no longer contained in *SortAbs*$_\varphi(T)$, since an instance (20) of (17) is provable from T. As a result, the clauses (16), (18), (19) are forgotten by the abstraction based on the mapping φ. In other words, they cannot be preserved in the abstract theory.

<Sort Hierarchy>

```
animal ≺ object.     vehicle ≺ object.          ship ≺ object.
horse ≺ animal.       car ≺ vehicle.        mountain ≺ object.
elephant ≺ animal.  man ≺ animal.  park ≺ place.  zoo ≺ place.
horse1 ∈ horse.     stable1 ∈ place.     central_park ∈ park.
```

<Domain Knowledge>

```
rule : not_in(X:vehicle,Y:park) :- move(X,Z:place,Y).

danger(X:object) :- big(X), movable(X), in(X,Z:place), gather(Y:man,Z).

gather(X:man,Y:place) :- rest(X,Y).
gather(_:man,Y:place) :- pleasant(Y).
pleasant(_:zoo) :- true.
rest(_:man,_:park) :- true.
big(_:horse) :- true.
big(_:vehicle) :- true.
big(_:elephant) :- true.
big(_:ship) :- true.
big(_:mountain) :- true.
big(_:car) :- true.

movable(_:vehicle) :- true.
movable(_:ship) :- true.
movable(_:animal) :- true.

move(horse1,stable1,central_park) :- true.
```

<CF-Predicates>

```
cf([danger/1]).
```

Figure 1. Example of sort hierarchy and legal domain theory.

3. Finding appropriate similarity for legal reasoning by analogy

In this section, we precisely discuss our process for finding appropriate similarities based on which our legal reasoning by analogy is carried out. Throughout this section, a simple example shown in Figure 1 is used for the illustration.

The example is a description of the legal knowledge and a case for the "*vehicle in the park*" problem with which we deal in this paper. The representation of clauses is Prolog-like. Each clause of the form $\mathtt{A} : -\mathtt{B}_n$ can be interpreted as $\mathtt{A} \leftarrow \mathtt{B}_1, \ldots, \mathtt{B}_n$. Our legal rule is represented as a clause with `rule:` in the figure. The rule says that if a vehicle `X` moves from a place `Z` to a park `Y`, the vehicle `X` should not enter the park `Y`. Our case is represented as the last clause `move(horse1, stable1, central_park) : –true`. It denotes a situation in which a horse `horse1` moves from a place `stable1` to a park `central_park`. Furthermore, `cf([...])` represents the

set of *CF-predicates* denoting undesirable circumstances. With this example, we examine whether horse can enter `park` or not.

3.1. DETECTING LEGAL RULE CAUSING UNIFICATION FAILURE

In this subsection, we describe a process for detecting a legal rule that is a subject of analogical application.

3.1.1. *Unification failure as a trigger of* GDA

As we have mentioned previously, we suppose a situation in which some legal rule is not applicable to a case represented by a set of facts. Since we describe every rule and fact in terms of order-sorted signature, such an application failure would be a type error in many cases. To illustrate the point, suppose we have the following legal rule and a legal goal, respectively:

`not_in(X : vehicle, Y : park)` $\leftarrow$

$\leftarrow$ `not_in(Z : horse, Y : park).`

If we have a sort hierarchy in which `horse` $\preceq$ `vehicle`, then we can apply the rule to the goal. This is because we can deduce `not_in(X : horse, Y : park)` $\leftarrow$ from `not_in(X : vehicle, Y : park)` $\leftarrow$ according to the type information stored in the sort hierarchy. On the other hand, if we classify `horse` as a special kind of `animal`, the rule is not applicable to the goal, for our horse is not regarded as a vehicle according to the classification. In other words, the rule is bound to the type `vehicle`, while the goal is concerned with another type `horse`. So the application of rule to the case of `horse` results in a *type violation*. We call such a phenomenon a *unification failure* and record it in a set of sorts $\{$`horse, vehicle`$\}$. It should be noted that in order to analogically apply the legal rule later, we have to find a similarity by GDA that resolves the unification failure. In this example, suppose we succeed in finding an appropriate similarity φ such that $\varphi($`horse`$) = \varphi($`vehicle`$)$. Our legal rule bound to `vehicle` is now generalized to a *hypothetical rule* bound to a new sort α dominating both `horse` and `vehicle`. The hypothetical rule is clearly applicable to the case of `horse`. This kind of generalization is called *Sorted Generalization* (Haraguchi, 1995). For more details, see the literature.

Generally speaking, we may have several unification failures $\{S_j\}$ for some set S_j of sorts. Therefore, a resolution of unification failures is defined as follows:

DEFINITION 2 (Resolution of Unification Failure). For a family $S = \{S_j\}$ of sort set S_j, a sort mapping φ is said to resolve S_j if $\varphi(S_j)$ is a singleton set for each j. □

Given some unification failures represented by $\{S_j\}$, GDA has to find a sort mapping that can resolve every S_j. It will work as a constraint for our possible similarities.

We illustrate the process for detecting a legal rule causing a unification failure with the example shown in Figure 1. Let us assume that we have a legal goal

```
not_in(X : horse, Y).
```

We can easily see that the goal cannot be proved in the sense of order-sorted logic. The failure is due to the legal rule

```
not_in(X : vehicle, Y : park) : −move(X, Z : place, Y).
```

Since the order-sorted unification of `horse` and `vehicle` is failed, we cannot unify the goal with the head of the rule. Therefore, we obtain a unification failure {`horse`, `vehicle`} and should find a similarity (hypothetical hierarchy) that resolves it.

Thus, by tracing the inference process, we can detect a legal rule that causes a unification failure.

3.2. IDENTIFYING GROUND OF LEGAL RULE

Here we describe the process for identifying the ground of legal rule detected at the previous step. The ground is considered as a goal for our GDA. Intuitively speaking, the ground is a set of undesirable circumstances for most people. We consider that the legal rule has been established to prevent these circumstances.

Let us assume that a legal rule $r : A \leftarrow B_1, \ldots, B_n$ was detected as a subject to analogical application. In order to identify the ground of r, we add the negation of r to our domain theory T_0. It is equivalent to adding the premise $B_1, \ldots, B_n$ and the negation of the conclusion $\neg A$. Intuitively speaking, this addition corresponds to imaginarily produce a situation in which the legal rule r is violated. After that, we try to infer undesirable circumstances from the extended domain theory by forward reasoning. That is, we examine what undesirable circumstances happen if the legal rule is violated. It would be considered that the purpose of legislation of the legal rule is to prevent these undesirable circumstances. For such a reasoning, our system is given a set of *CF-predicates* denoting undesirable circumstances. If our system succeeds in finding some CF-predicates in the space of possible consequences of the extended theory, the set of such CF-predicates is considered as the *ground* of the legal rule. That is, the ground of legal rule r, denoted by *ground*(r), is formally defined as follows:

$$\textit{ground}(r) = \{g \mid g \text{ is a CF-predicate and } T_0 - \{r\} \cup \{\neg r\} \vdash g\}.$$

We illustrate the process of identifying the ground of the legal rule

```
not_in(X : vehicle, Y : park) : −move(X, Z : place, Y)
```

that is detected at the previous step with our example. The negation of the rule is

```
¬not_in(c1, c2) ∧ move(c1, c3, c2),
```

where c1, c2, c3 are Skolem constants belonging to vehicle, park and place, respectively. It should be noted that we transform the negation of not_in into in based on a user defined data since our current system supports only definite clauses.

Adding the negated rule

$$\texttt{in(c1, c2)} \wedge \texttt{move(c1, c3, c2)}$$

to the domain theory, we can derive only one CF-predicate danger(c1). Therefore, the ground of the legal rule is {danger(c1)}.

3.3. FINDING APPROPRIATE SIMILARITY BY GDA

At this step, our GDA computes an *appropriate* similarity (hypothetical hierarchy) φ with respect to the ground of a given legal rule. This subsection presents our GDA algorithm precisely.

First of all, we have to define our appropriateness of similarity. Assume that we are given the ground of a legal rule r, *ground*(r). Firstly, from our extended domain theory $T = T_0 - \{r\} \cup \{\neg r\}$, we collect clauses that are used for a proof of the CF-predicates in *ground*(r). Let us denote the set of collected clauses by *Proof(ground*(r)*)*. An appropriateness of similarity is defined as follows:

DEFINITION 3 (Appropriate Similarity). We say that a sort mapping φ is an *appropriate similarity w.r.t. ground*(r), if the following two conditions are satisfied:

- *Substitutability Condition*
  $\varphi($*Proof*$($*ground*$(r)))) \subseteq$ *SortAbs*$_\varphi(T)$
- *Similarity Inheritance Condition*
  $\varphi(s_1) = \varphi(s)$ whenever $s_1 \preceq s_2$ and $\varphi(s_2) = \varphi(s)$, where $\preceq$ is the ordering on sorts assumed in our standard sort hierarchy $(S, \preceq)$. □

Let us assume that a similarity φ such that $\varphi(s) = \varphi(s')$ is appropriate w.r.t. a ground G. From the definition of *SortAbs*, it is ensured by Substitutability Condition that the ground G can be proved for the cases of both s and s'. In other words, we can arbitrarily replace any sort appearing in *Proof*(G) with a similar sort under φ. Such a replacement never affects proving G.

Similarity Inheritance Condition (SI-Condition for short) asserts that, if we observe some similarity between s and s_1, we should also have the same similarity between s and any subconcept of s_1. For instance, if car $\preceq$ vehicle and vehicle is similar to ship, then car should be similar to ship as well. Generally speaking, similarities satisfying SI-Condition preserve the following property: if a is similar to b and a' is a subconcept of a, then a' inherits this similarity relation from its super concept a, namely, a' is also similar to b. Therefore, such a property is called *Similarity Inheritance*. We propose adopting SI-Condition based on this property as a requirement for our appropriate similarities. We should notice that

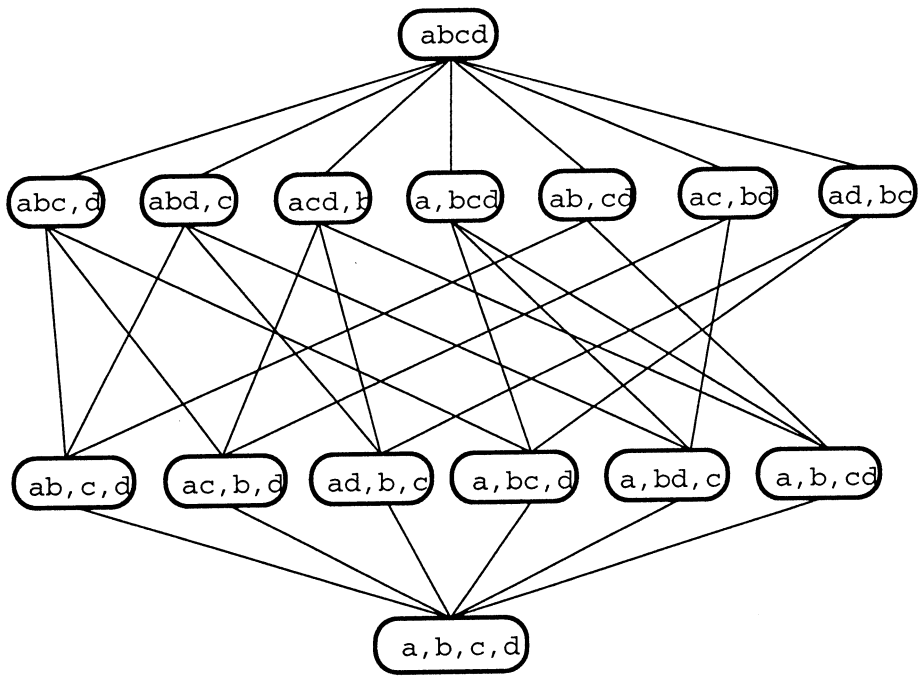

Figure 2. Possible partitions.

since we represent "x is also similar to y" by $\varphi(x) = \varphi(y)$, we can directly prove SI-Condition from the property above.

Based on the definition of appropriateness, we can examine whether a given sort mapping is appropriate for the ground or not. Therefore, our GDA tries to find appropriate similarities adopting a *generate-and-testing* strategy. Candidate sort mappings are generated and then tested for their appropriateness. Each candidate can be viewed as a partition of the set of sorts S. For example, let us consider a set of sorts $\{\mathtt{a},\ \mathtt{b},\ \mathtt{c},\ \mathtt{d}\}$. A partition of the set

$$\{\{\mathtt{a},\ \mathtt{b}\}, \{\mathtt{c},\ \mathtt{d}\}\}$$

corresponds to a sort mapping φ such that $\varphi(\mathtt{a}) = \varphi(\mathtt{b})$ and $\varphi(\mathtt{c}) = \varphi(\mathtt{d})$. That is, each element of the partition (called a *cell*) denotes a class of similar sorts under the mapping. As the image for each cell, we can arbitrarily assign a new sort not appearing in S.

In general, however, there are a huge number of partitions (candidate mappings) generated from S. Since it is not practical to test all candidates, it is necessary to prune useless candidate generations in order to find appropriate similarities efficiently. We present a property based on which we can obtain such a pruning. To state the property, we need to introduce an ordering on partitions (i.e. similarities).

DEFINITION 4 (Ordering on Partitions). For any partitions P and P' of a non-empty set, $P \prec P'$ and P is called a refinement of P' iff for any cell c of P, there exists a cell c' of P' such that $c \subseteq c'$. □

It should be noted that the possible partitions of a non-empty set form a lattice under the ordering. Figure 2 shows the lattice formed by the partitions of $\{$`a, b, c, d`$\}$.

Based on the ordering, we have the following property that is useful in pruning useless candidate generations.

PROPOSITION 1. *Let φ be a similarity. If φ is not appropriate w.r.t. the ground of a legal rule, then each similarity φ' such that $\varphi \preceq \varphi'$ is not appropriate w.r.t. the ground.* □

Proof. The proposition can be proved in a similar way found in (Okubo and Haraguchi, 1994b). □

The property implies that if we found that a similarity φ is not appropriate, then we do not have to generate any similarity preceded by φ since the similarity is not appropriate as well. In order to obtain as much pruning of candidate generations as possible, GDA firstly generates the minimum candidate and tests it. Then, its immediate successors are generated and then tested. Only for appropriate candidates, GDA generates their immediate successors. Such a process is iterated until no candidate is generated.

In addition, we can prune candidate generations based on the following two facts:

- Since our purpose is to find similarities that can resolve a unification failure to prove our legal goal analogically, we can focus on only cells that is relevant to the unification failure. Therefore, any cell irrelevant to the failure can be ignored when partitions are generated.
- Although SI-Condition is a part of the definition of appropriateness, we do not examine in our test-step whether the condition is satisfied or not. Instead we use SI-Condition as a constraint on our candidate generations in fact. That is, we generate only candidates satisfying the condition.

Our GDA algorithm is shown in Figure 3 (for further details, see Appendix B).

We can illustrate the process of finding appropriate similarities with the previous example. Recall that the ground of the detected legal rule is $\{$`danger(c1)`$\}$. The set of clauses *Proof*($\{$`danger(c1)`$\}$) in T_0 used for a proof of `danger(c1)` is

```
{rest(_ : man, _ : park) : −true.
gather(X : man, Y : place) : −rest(X, Y).
in(c1 : vehicle, c2 : park) : −true.
movable(_ : vehicle) : −true.
big(_ : vehicle) : −true.
```

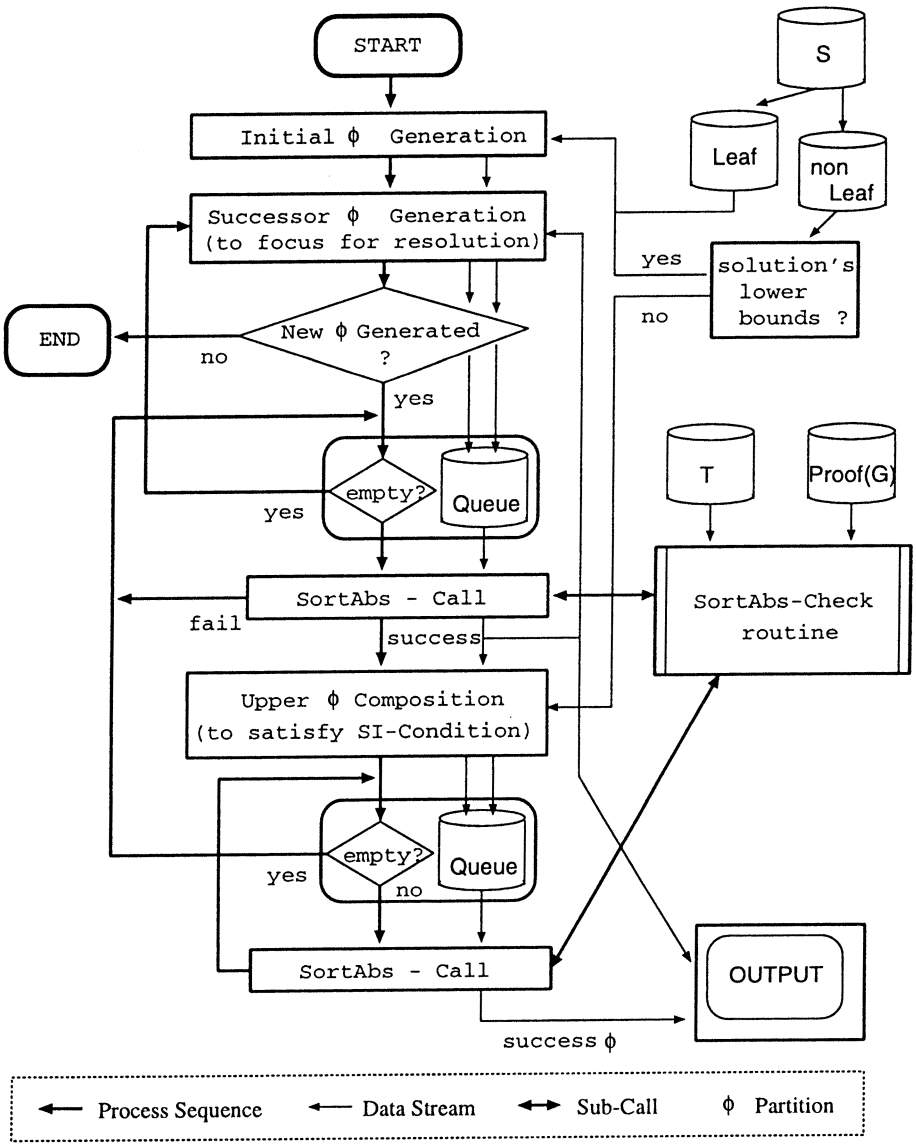

Figure 3. Overview of GDA algorithm.

```
danger(X : object) : —
    big(X), movable(X), in(X, Z : place), gather(Y : man, Z).}.
```

Based on the set, we examine whether a similarity satisfies Substitutability Condition in Definition 3.

Let us actually examine several similarities. Candidate similarities are generated from the set of sort symbols

$$\{\texttt{vehicle, object, park, place, man, horse, elephant, mountain, zoo, car, ship, animal}\}.$$

Recall that the similarities we are interested in are the ones each of which has a cell containing `vehicle` and `horse` to resolve the unification failure. In each candidate similarity below, it is assumed that any cell not represented explicitly is a singleton set.

- $\varphi_1 : \{\ldots, \{\texttt{vehicle, horse, park}\}, \ldots\}$
  Let us assign a new sort α as the image of `vehicle`, `horse` and `park` under φ_1. A clause $\varphi_1(\texttt{movable(\_ : vehicle) :-true})$ in $\varphi(\textit{Proof}(\{\texttt{danger(c1)}\}))$ (that is, $\texttt{movable(\_ : } \alpha\texttt{) : } -\texttt{true}$) cannot be contained in $\textit{ShortAbs}_{\varphi_1}(T)$. Because an instance of the clause, $\texttt{movable(\_ : park)} : -\texttt{true}$, is not provable from our domain theory. That is, φ_1 cannot satisfy Substitutability Condition and is, therefore, not appropriate. According to Proposition 1, GDA does not generate any candidate mapping φ' such that $\varphi_1 \preceq \varphi'$. For example, $\{\ldots, \{\texttt{vehicle, horse, park, elephant}\}, \ldots\}$ is out of generation.

- $\varphi_2 : \{\ldots, \{\texttt{vehicle, horse, elephant}\}, \ldots\}$
  Although the similarity satisfies Substitutability Condition, it violates SI-Condition. Therefore, φ_2 is considered to be not appropriate. To satisfy SI-Condition, the cell has to contain `car`. It should be noted that as previously mentioned, our GDA in fact does not generate such a candidate violating SI-Condition.

We can obtain the following similarity as an appropriate one:

$$\{\{\texttt{vehicle, horse, elephant, car}\}, \{\texttt{object}\}, \{\texttt{park}\}, \{\texttt{place}\}, \{\texttt{man}\}, \{\texttt{mountain}\}, \{\texttt{zoo}\}, \{\texttt{ship}\}, \{\texttt{animal}\}\}.$$

It should be emphasized here that our GDA can find a similarity *not only* between `vehicle` and `horse` *but also* between `vehicle` and `elephant`. Although our original legal goal is `not_in(X : horse, Y)`, we can also analogically derive another legal goal `not_in(X : elephant, Y)` based on the similarity.

4. Experimental results

Based on the above discussions, we have implemented a system by SICStus Prolog 3 on SparcStation 20 and PC/AT (486DX2-80MHz). Since our system has currently been under improvement, its domain theory is restricted to be a set of function-free

definite Horn-clauses. The system uses a meta-interpreter on Prolog as refutation-mechanism in each step, and is based on GDA algorithm presented in Section 3.3. In this section, we show our experimental results by the system.

The domain theory shown in Figure 1 and a legal goal `not_in(X : horse, Y)` has been given to the system. In the theory, we have 12 sort symbols. For a N-elements set, the number of possible partitions of the set is given as

$$B_N = \sum_{i=1}^{N} \binom{N-1}{i-1} B_{N-i}, \quad \text{where} \quad B_0 = 1.$$

Therefore, the number of possible partitions (candidate mappings) in the example is 4,213,597 (about 4×10^6). However, with the help of the properties to prune useless candidate generations stated in the previous section, the number of partitions we actually tested is only 18. Thus, we can reduce the search space drastically.

We have got the following four appropriate similarities:

$\{\ldots, \{$`horse, vehicle, car`$\}, \ldots\}$

$\{\ldots, \{$`horse, vehicle, car, elephant`$\}, \ldots\}$

$\{\ldots, \{$`horse, vehicle, car, ship`$\}, \ldots\}$

$\{\ldots, \{$`horse, vehicle, car, elephant, ship`$\}, \ldots\}$.

Our intended similarities would be

$\{\ldots, \{$`horse, vehicle, car`$\}, \ldots\}$

$\{\ldots, \{$`horse, vehicle, car, elephant`$\}, \ldots\}$.

The other two similarities that would be out of our intention are obtained since our domain theory is incomplete. The total computation time is 360 msec on SparcStation 20 and 1978 msec on PC/AT.

5. Concluding Remarks

There still remain many problems to be solved. Among them, two major problems are discussed briefly in this section.

5.1. KNOWLEDGE REPRESENTATION

The language we use in this paper is too much restricted, for it is function-free. It is necessary to extend our GDA algorithm so that it can cope with a class of similarities concerning not only *is_a_subclass* relationships but also role-filler constraints (Nebel, 1990). The latter kind of constraint will be necessary to describe complex legal sentences and is useful to obtain a new syntactic rule to specify what similarities we must find. In fact, such a new GDA taking role-filler relations into account is now under developing.

5.2. PREFERENCE RELATION ON SIMILARITIES

A function to measure the distance of concepts would be also needed even for our approach. GDA algorithm might fail to find intended similarities when our domain theory does not have a good knowledge to distinguish important similarities from the other. To cope with the incompleteness problem of our knowledge base, it seems desirable for our reasoning system to measure the distance of concepts approximately. Such a measure will introduce a preference relation over possible similarities, and is also useful for our GDA algorithm to approach the intended similarities more quickly.

Appendix A: Analogical reasoning by generalization and deduction

We present here a simple example of analogy as a combined process of generalization and deduction (Haraguchi, 1995).

Let us assume that we have a taxonomic hierarchy in which concepts s_1 and s_2 have the same super concept s'. In such a case, we consider that s_1 and s_2 are similar to each other. The hierarchy means that any individual belonging to s_1 or s_2 is also an individual of s'. Moreover, assume we have a domain theory consisting of a fact $q(a : s_1)$ and a rule $p(x : s_2) \leftarrow q(x : s_2)$. The fact says that an individual a of s_1 has a property q. The rule means that if an individual of s_2 has the property q, then it also has another property p.

Let us try to prove a goal $p(a : s_1)$ from the domain theory. However, the goal cannot be proved. Since a is not an individual of s_2 we cannot apply the rule to the case of a. In such a case, we expand the applicability of the rule by *generalizing* s_2 into s' along the hierarchy. That is, we obtain a new rule $p(x : s') \leftarrow q(x : s')$ saying that if any individual of s' (i.e. ones of s_1 and s_2) has the property q, then it has the property p as well. Although the original rule is applicable only to the case of s_2, the new rule is applicable also to the case of s_1 that is similar to s_2. Namely, an analogical application of the original rule can be realized by the generalization. By using the generalized rule, we can *deductively* derive the original goal $p(a : s_1)$. Thus analogical reasoning can be viewed as a combined process of generalization and deduction.

Appendix B: GDA algorithm for legal reasoning by analogy

<Given>

$\Sigma = (S, \preceq)$:
- S : a set of sorts
- $\preceq$: a partial order on S

Id : a bijective mapping defined as follows:

$$Id : \begin{cases} S \mapsto \text{ positive-integers}, & \text{for any sorts} \\ 2^S \mapsto \text{ negative-integers}, & \text{for any cells} \end{cases}$$

(The value of $Id(s)$ is called ID of s.)

T : a set of all order-sorted clauses defined as domain knowledge

R : a set of resolution sort sets (i.e. a set of cells)

Pf : a set of all clauses in a proof

<Find>

Ans : a set of partitions

<Procedures>

GDA-MAIN:

1. generate initial partitions.
   - 1.1 encode S, R by Id-mapping.
     S_0:= (all leaf sorts $\in S$) $-$ (all sorts $\in R$).
   - 1.2 P_0:= make minimum partition from S_0.
   - 1.3 encode each cell $c \in R$ to ID as the value of $Id(c)$.
   - 1.4 R_0:=$\{\ \}$.
   - 1.5 convert each cell $c \in R$ to $\{Id(c)\}$, push $\{Id(c)\}$ to R_0.
   - 1.6 P:=$\{\ \}$, p:=P_0.
   - 1.7 **FOR** any cell $c \in R_0$,
     p:=$\{c\} \cup p$.
     IF $c == \{ID_0\}$ such that ID_0 is our minimum integer **THEN** mark $c \in p$.
     push p to P.
   - 1.8 remove failure partitions by **ABS-CHECK** from P.
   - 1.9 *OutStack*:=$\{\ \}$,
     push all elements in P to *OutStack*,
     FOR any $p \in P$,
     CALL SI-GENERATION(p).
2. generate immediate successors.
   - 2.1 **IF** $P == \{\ \}$ **GOTO** 5.
   - 2.2 *NewStack*:=$\{\ \}$.
   - 2.3 **FOR** any $p \in P$,
     get marked cell $c_0 \in p$,
     FOR any cell $c \in p$ ($c \neq c_0$),
     IF *MaxId*$(c_0) \neq$ *MinId*(c)
     THEN
     NewCell:=$c_0 \cup c$,
     NewP:=$p - c_0 - c +$ *NewCell*
     mark *NewCell* $\in$ *NewP*,
     push *NewP* to *NewStack*.
3. test SortAbs.
   - 3.1 **FOR** any $p \in$ *NewStack*,
     CALL ABS-CHECK(p),
     IF ABS-CHECK is success
     THEN
     push p to *OutStack*,
     CALL SI-GENERATION(p).
     ELSE remove p from *NewStack*.
   - 3.2 P:=*NewStack*.
4. **GOTO** 2.
5. output.
   - 5.1 **FOR** any $P \in$ *OutStack*,
     FOR any $c \in P$,
     FOR any ID $\in c$,
     decode ID to the original sort by Id^{-1}.
     normalize c.
     normalize P,
     Ans:=P.
   - 5.2 output *Ans*.

ABS-CHECK(p): check p based on SortAbs of Definitions 1 and 3.

SI-GENERATION(p_0):

1. compose p combining focused cells (i.e. non-singleton cells) in p_0 and non-leaf sorts such that for each cell $c \in p$ except to singleton cells,

$$\forall \in c \quad \forall y \preceq x \quad y \in c.$$

2. FOR any p,
   CALL ABS-CHECK(p),
   IF ABS-CHECK is success **THEN** push p to *OutStack*.

References

Ashley, K. D. (1990). *Modelling Legal Argument: Reasoning with Cases and Hypotheticals.* MIT Press: Cambridge, MA.

Beierle, C. et al. (1992). An Order-Sorted Logic for Knowledge Representation Systems. *Artificial Intelligence* 55:149–191.

Fujita, H. & Hasegawa, R. (1991). A Model Generation Theorem Prover in KL1 Using a Ramified-Shack Algorithm. In *Logic Programming*. Proceedings of the *Eight International Conference*, 535–548. MIT Press: Cambridge, MA.

Haraguchi, M. (1995). A Reasoning System for Legal Analogy. *Machine Intelligence* **14**: 323–346.

Hart, H. L. A. (1958). Positivism and the Separation of Law and Mortals. *Harvard Law Review* **71**: 593–629.

Inoue, K., Ohta, Y., Hasegawa, R. & Nakashima, M. (1993). Bottom-Up Abduction by Model Generation. In *IJCAI-93*. Proceedings of the *Thirteenth International Joint Conference on Artificial Intelligence*, 1: 102–108. Morgan Kaufmann: San Mateo, CA.

Muggleton, S. (1988). A Strategy for Constructing New Predicates in First Order Logic. In *EWSL 88*. Proceedings of the *Third European Working Session on Learning*, 123–130. Pitman Publishing: London.

Muggleton, S. (1990). Inductive Logic Programming. In Arikawa, S. (ed.) *Algorithmic Learning Theory*, 42–62. Springer-Verlag: Berlin.

Nebel, B. (1990). *Reasoning and Revision in Hybrid Representation Systems*, LNAI 422. Springer-Verlag: Berlin.

Nitta, K., Wong, S. & Ohtake, Y. (1993). A Computational Model for Trial Reasoning. In Proceedings of the *Fourth International Conference on Artificial Intelligence and Law*, 20–29. Association for Computing Machinery: New York.

Ohtake, Y., Nitta, K., Maeda, S., Ono, M., Osaki, H. & Sakane, K. (1994). Legal Reasoning System HELIC-II. *Transactions of Information Processing Society of Japan* **35**(6): 986–996.

Okubo, Y. & Haraguchi, M. (1994a). Planning with Abstraction Based on Partial Predicate Mappings. *New Generation Computing – An International Journal* **12**(4): 409–437.

Okubo, Y. & Haraguchi, M. (1994b). Constructing Predicate Mappings for Goal-Dependent Abstraction. In Arikawa, S. (ed.) *Algorithmic Learning Theory '94*, LNAI 872, 516–531. Springer-Verlag: Berlin.

Plaisted, D. A. (1981). Theorem Proving with Abstraction. *Artificial Intelligence* **16**: 47–108.

Rouveirol, C. & Puget, J. F. (1990). Beyond Inversion of Resolution. In *Machine Learning*: Proceedings of the *Seventh International Conference*, 122–130. Morgan Kaufmann: San Mateo, CA.

Tenenberg, J. D. (1989). Abstracting First-Order Theories. In *Change of Representation and Inductive Bias*, 67–69. Kluwer Academic Publishers: Dordrecht, the Netherlands.

Walter, C. (1988). Many-Sorted Unification. *Journal of the Association for Computing Machinery* **35**(1): 1–17.

Yoshino, H., Haraguchi, M., Sakurai, S. & Kagayama, S. (1993). Towards a Legal Analogical Reasoning System. In Proceedings of the *Fourth International Conference on Artificial Intelligence and Law*, 110–116. Association for Computing Machinery: New York.