ICOT Technical Memorandom: TM-1186

TM-1186

Proceedings of the FGCS Workshop on Knowledge Representation and its Applications

by

H. Yasukawa (Matsushita)

March, 1993

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03)3456-3191～5

Institute for New Generation Computer Technology

Proceedings of the FGCS Workshop on Knowledge Representation and its Applications

6th June 1992
Shiba Park Hotel

Contents

Organizing Committee

Hideyuki Nakashima, Co-Chair	ETL, Japan
Syun Titiya, Co-Chair	Chiba University, Japan
Hideki Yasukawa, Co-Chair	Matsushita Electric Industrial Co. Ltd., Japan
Robert Kowalski	Imperial College, UK

Preface

This volume contains the papers presented at the FGCS'92 Post-Conference Workshop on Knowledge Representation and its Applications held in Tokyo, Japan, on 6th June, 1992.

The workshop was intended to be a place for discussing about wide range of topics around knowledge representation and its applications.

Around 30 researchers worldwide participates the workshop, and among them, nine speakers gave talks. They made the workshop a place for significant discussions.

We would like to thank all the participants, especially nine speakers giving impressive talks. We also thank members of ICOT for their help in organizing the workshop.

Abductive Logic Programming
DRAFT

October 1991, revised January 1992

A.C. Kakas
Department of Computer Science,
University of Cyprus,
75 Kallipoleos Street,
Nicosia T.T.134, Cyprus.

R.A. Kowalski, F. Toni
Department of Computing
Imperial College of Science, Technology and Medicine
180 Queen's Gate
London SW7 2BZ, UK.

Abstract

This paper is a survey and critical overview of recent work on the extension of Logic Programming to perform Abductive Reasoning (Abductive Logic Programming). We outline the general framework of Abduction and its applications to Knowledge Assimilation and Default Reasoning, concentrating on the use of Abduction to clarify the understanding of Negation as Failure as a form of Default Reasoning. We also analyse the links between Abduction and the extension of Logic Programming obtained by adding a form of explicit negation. Finally we discuss the relation between Abduction and Truth Maintenance.

1 Introduction

This paper is a survey and analysis of work on the extension of logic programming to perform abductive reasoning. The purpose of the paper is to provide a critical overview of some of the main research results, in order to develop a common framework for evaluating these results, to identify the main unresolved problems, and to indicate directions for future work. The emphasis is not on technical details but on relationships and common features of different approaches. Some of the main issues we will consider are the contributions that abduction can make to the problems of reasoning with negative or incomplete information, the evolution of knowledge, and the semantics of logic programming and its extensions.

The philosopher Pierce first introduced the notion of abduction. In [Peirce31] he identified three distinguished forms of reasoning.

Deduction, an analytic process based on the application of general rules to particular cases, with the inference of a result.

Induction, synthetic reasoning which infers the rule from the case and the result.

Abduction, another form of synthetic inference, but of the case from a rule and a result.

Pierce further characterised abduction as the "probational adoption of a hypothesis" as explanation for observed facts (results), according to known laws. "It is however a weak kind of inference, because we cannot say that we believe in the truth of the explanation, but only that it may be true"[Peirce31].

Abduction is widely used in common-sense reasoning, for instance in diagnosis, to reason from effect to cause [Pople73, Charniak&McDermott85]. We consider here an example drawn from [Pearl87].

Example 1.1
Consider the following theory T

$$\begin{aligned} \textit{grass-is-wet} &\leftarrow \textit{rained-last-night} \\ \textit{grass-is-wet} &\leftarrow \textit{sprinkler-was-on} \\ \textit{shoes-are-wet} &\leftarrow \textit{grass-is-wet}. \end{aligned}$$

If we observe that our shoes are wet, and we want to know why this is so, *rained-last-night* is a possible explanation, i.e. a set of hypotheses that together with the explicit knowledge in T implies the given observation. *Sprinkler-was-on* is another alterative explanation.

Abduction consists of computing such explanations for observations. It is a form of non-monotonic reasoning, because explanations which are consistent with one state of a knowledge base may become inconsistent with new information. In the example above the explanation *rained-last-night* may turn out to be false, and the alternative explanation *sprinkler-was-on* may be the true cause for the given observation. The existence of **multiple explanations** is a general characteristic of abductive reasoning, and the selection of "preferred" explanations is an important problem.

1.1 Abduction in logic

Given a set of sentences T (a theory presentation), and a sentence G (observation), the abductive task consists in finding a set of sentences Δ (abductive explanation for G) such that:

(1) $T \cup \Delta \models G$,

(2) $T \cup \Delta$ is consistent.

The first property corresponds directly to Peirce's definition of abduction. The second condition is a natural additional requirement. This characterisation of abduction is independent of the language in which T, G and Δ are formulated. The logical implication sign $\models$ in (1) can alternatively be replaced by a deduction operator $\vdash$.

Cox and Pietrzykowski [Cox&Pietrzykowski86] specify other desiderable properties of abductive explanations. For instance, an explanation should be **basic**, i.e. cannot be explained in terms of other explanations. For example, in example 1.1 the explanation

$$\textit{grass-is-wet}$$

for the observation

shoes-are-wet

is not very satisfactory, because it is not basic. An explanation should also be **minimal**, i.e. not subsumed by a different one. For example, in the propositional theory

$$\begin{array}{lcl} p & \leftarrow & q \\ p & \leftarrow & q, r \end{array}$$

$\{q, r\}$ is a non-minimal explanation for p while $\{q\}$ is minimal.

In practice we want to restrict the set of sentences that can be part of an explanation. This restriction is generally domain specific. Typically, we are interested in explanations which convey some reason why the observations hold, e.g. we do not want to explain one effect in terms of another effect, but in terms of some cause. Hence the explanations are often required to be only in terms of sentences that belong to a special pre-specified class called **abducibles**. Notice that there are no general criteria for deciding what is an abducible or not. In this paper we will assume that the abducibles are always given.

Additional criteria have also been proposed to restrict the number of candidate explanations:

- Once we restrict the hypotheses to belong to a specified set of sentences, we can further restrict, without loss of generality, the hypotheses to atoms (that "name" these sentences) which are predicates explicitly indicated as abducible, as shown in [Poole88a].
- In section 1.2 we will discuss the use of integrity constraints to reduce the number of possible explanations.
- Additional information can help to discriminate between different explanations, by rendering some of them more appropriate or plausible than others. For example [Sattar&Goebel89] use "crucial literals" to discriminate between two mutually incompatible explanations. When the crucial literals are tested, one of the explanations is rejected. More generally in [Evans&Kakas91a] the notion of corroboration is used to select explanations. An explanation fails to be corroborated if some of its logical consequences are not observed. A related technique is presented in [Sergot83], where information is obtained from the user during the process of query evaluation.
- Moreover various (domain specific) criteria of preference can be specified. They impose a (partial) order on the sets of hypotheses which leads to the discrimination of explanations [Charniak&McDermott85, Brewka89, Hobbs et al.91]

So far we have presented a semantic characterisation of abduction and discussed some heuristics to deal with the multiple explanation problem, but we have not described any proof procedures for computing abduction. Various authors have suggested the use of top-down, goal-oriented computation, based on the use of deduction to drive the generation of abductive hypotheses. Cox and Pietrzykowski [Cox&Pietrzykowski86] construct hypotheses from the "dead ends" of linear resolution proofs. Finger and Genesereth [Finger&Genesereth85] generate "deductive solutions to design problems" using the "residue" left behind in resolution proofs. Poole, Goebel and Aleliunas [Poole et al.87] also use linear resolution to generate hypotheses. In contrast, the ATMS [deKleer86, Reiter&deKleer87] computes abductive explanations bottom-up.

Abduction can also be applied to logic programming. A **general logic program** is a set of Horn clauses extended by negation as failure [Clark78], i.e. **clauses** of the form:

$$A \leftarrow L_1, \ldots, L_n$$

where each L_i is either an atom A_i or its negation $\sim A_i$ [1], A is an atom and each variable occurring in the clause is implicitly universally quantified. Abduction can be computed in logic programming by extending SLD and SLDNF [Eshghi&Kowalski88, Eshghi&Kowalski89, Chen&Warren89, Kakas&Mancarella90b, Kakas&Mancarella91a]. Instead of failing in a proof when a selected subgoal fails to unify with the head of any rule, the subgoal can be viewed as a hypothesis. This is similar to viewing abducibles as "askable" conditions which are treated as qualifications to answers to queries [Sergot83]. In the same way that it is useful to distinguish a subset of all predicates as "askable", it is useful to distinguish certain predicates as abducible. In fact, it is generally convenient to choose, as abducible predicates, ones which are not conclusions of any clause. As we shall remark at the beginning of section 5, this restriction can be imposed without loss of generality, and has the added advantage of ensuring that all explanations will be basic.

There are other formalisations of abduction. We mention them for completeness, but in the sequel we will concentrate on the logic-based view previously described.

- [Reggia83, Allemand et al.87] present a mathematical characterisation, where abduction is defined over sets of observations and hypotheses, in terms of coverings and parsimony.
- [Levesque89] gives an account of abduction at the "knowledge level". He characterises abduction in terms of a (modal) logic of beliefs, and shows how the logic-based approach to abduction can be understood in terms of a particular kind of belief.

In the previous description of abduction we have briefly discussed both the **semantics** and a **proof procedure** for abduction. The relationship between semantics and proof procedures can be understood as a special case of the relationship between program specifications and programs. A program specification characterises what is the intended result expected from the execution of the program. In the same way semantics can be viewed as an abstract, possibly non-constructive definition of what is to be computed by the proof procedure. From this point of view, semantics is not so much concerned with explicating meaning in terms of truth and falsity, as it is with providing an abstract specification which "declaratively" expresses what we want to compute. This specification view of semantics is effectively the one adopted in most recent work on the semantics of logic programming, which restricts interpretations to Herbrand interpretations. This restriction to Herbrand interpretations means that interpretations are purely syntactic objects, which have no bearing on the correspondence between language and "reality".

One important alternative way to specify the semantics of a language, which will be used in the sequel, is through the **translation** of sentences expressed in one language into sentences of another language, whose semantics is already well understood. For example if we have a sentence in a typed logic language of the form "there exists an x of type t such that the property p holds" we can translate this into a sentence of the form $\exists x\, (p(x) \land t(x))$, where t is a new predicate to represent the type t, whose semantics is then given by the

[1] In the sequel we will represent NAF as $\sim$.

familiar semantics of first-order logic. Similarly the typed logic sentence "for all x of type t such that the property p holds" becomes the sentence $\forall x(p(x) \leftarrow t(x))$. Hence instead of developing a new semantics for the typed logic language, we apply the translation and use the existing semantics of first-order logic.

1.2 Integrity Constraints

Abduction as presented so far can be restricted by the use of integrity constraints, as previously mentioned. Integrity constraints are useful to avoid unintended updates to a database or knowledge base. They can also be used to represent desired properties of a program [Lever91].
The concept of integrity constraints first arose in the field of databases and to a lesser extent in the field of AI knowledge representation. The basic idea is that only certain knowledge base states are considered acceptable, and an integrity constraint is meant to enforce these legal states. When abduction is used to perform updates (see section 2), we can use integrity constraints to reject abductive explanations.

Given a set of integrity constraints, I, of first-order closed formulae, the second condition (2) of the semantic definition of abduction (see section 1.1) can be replaced by:

(2′) $T \cup \Delta$ satisfies I.

As previously mentioned, we will generally consider as hypotheses in Δ atoms drawn from predicates explicitly indicated as abducible. In the sequel an **abductive framework** will be given as a triple $\langle T, A, I\rangle$, where A is the set of abducible predicates, i.e. $\Delta \subseteq A$ [2]. Note that the set of integrity constraints can be empty. In this case the semantics of the abductive framework $\langle T, A, \emptyset\rangle$ is that indicated in the original definition.

There are several ways to define what it means for a knowledge base KB ($T \cup \Delta$ in our case) to satisfy an integrity constraint ϕ (in our framework $\phi \in I$). The **consistency view** requires that:

$$KB \text{ satisfies } \phi \text{ iff } KB \cup \phi \text{ is consistent.}$$

Alternatively the **theoremhood view** requires that:

$$KB \text{ satisfies } \phi \text{ iff } KB \models \phi.$$

These definitions have been proposed in the case where the theory is a logic program P by [Kowalski&Sadri87] and [Lloyd&Topor85] respectively, where KB is the Clark completion [Clark78] of P.

Another view of integrity constraints [Reiter88, Reiter90, Kakas&Mancarella90b], [Kowalski90, Kakas91] regards these as **epistemic** or **meta-level** statements about the content of the database. In this case the integrity constraints are understood as statements at a different level from those in the knowledge base. They specify what is true about the knowledge base rather than what is true about the world modelled by the knowledge base. When later we consider abduction in logic programming (see sections 4,5), the semantics of integrity constraints will be defined in the spirit of this third view, as statements that must hold about a separately defined (abductive) logic program.

[2] Here and in the rest of this paper we will use the same symbol A to indicate both the set of abducible predicates and the set of all their ground instances.

For each such semantics, we have a specification of the integrity checking problem. Although the different views of integrity satisfaction are conceptually very different, the integrity checking procedures based upon these views are not so different in practice (e.g. [Lloyd&Topor85, Decker86, Kowalski&Sadri87]). They are mainly concerned with avoiding the inefficiency which arises if all the integrity constraints are tested after each update. A common idea of all these procedures is to render integrity checking more efficient by exploiting the assumption that the database before the update satisfies the integrity constraints, and therefore if integrity constraints are violated after the update, this violation should depend upon the update itself. In [Kowalski&Sadri87] this assumption is exploited by reasoning forward from the updates. This idea is exploited for the purpose of checking the satisfaction of abductive hypotheses in [Eshghi&Kowalski89], [Kakas&Mancarella90d], [Kakas&Mancarella91a]. Although this procedure was originally formulated for the consistency view of constraint satisfaction, it has proved equally appropriate for the semantics of integrity constraints in abductive logic programming.

1.3 Applications

In this section we briefly describe some of the applications of abduction in AI.

Abduction can be used to generate causal explanations for **fault diagnosis** (see for example [Console et al.89], [Preist&Eshghi92]). In medical diagnosis, for example, the candidate hypotheses are the possible causes (diseases), and the observations are the symptoms to be explained [Reggia83, Poole88b]. Abduction can also be used for model-based diagnosis [Reiter87, Eshghi90]. In this case the theory describes the "normal" behaviour of the system, and the task is to find a set of hypotheses of the form "some component A is not normal" that explains why the behaviour of the system is not normal.

Abduction can be used to perform **high level vision** (e.g. recognition of graphical objects [Cox&Pietrzykowski86]). The hypotheses are the objects to be recognised, and the observations are the descriptions of objects.

Abduction can be used in **natural language understanding** to interpret ambiguous sentences [Charniak&McDermott85, Stickel89, Hobbs90, Gabbay&Kempson91]. The abductive explanations correspond to the various possible interpretations of such sentences.

In **planning** problems plans can be viewed as explanations of the given goal state to be reached [Eshghi88, Shanahan89].

These applications of abduction can all be understood as generating hypotheses which are causes for observations which are effects. An application that does not necessarily have a direct causal interpretation is **knowledge assimilation** [Kowalski79, Miyaki et al.84, Kunifiji et al.86, Kakas&Mancarella91a]. The assimilation of a new datum can be performed by adding to the theory new hypotheses that are explanations for the datum. **Database view updates** [Kakas&Mancarella90b, Bry90] are an important special case of knowledge assimilation. Update requests are interpreted as observations to be explained. The explanations of the observations are transactions that satisfy the update request. We will discuss knowledge assimilation in greater detail in section 2.

Another important application which can be understood in terms of a "non-causal" use of abduction is **default reasoning.** Default reasoning concerns the use of general rules to derive information in the absence of contradictions. In the application of abduction to default reasoning conclusions are viewed as observations to be explained by means of assumptions which hold by default unless a contradiction can be shown [Poole88a, Eshghi&Kowalski88]. As Poole [Poole88a] argues, the use of abduction avoids the need to develop a non-classical, non-monotonic logic for default reasoning. In section 3 we will further discuss the use of abduction for default reasoning in greater detail. Because negation as failure in logic programming is a form of default reasoning, its interpretation by means of abduction will be discussed in section 4.

2 Knowledge Assimilation

Abduction takes place in the context of assimilating new knowledge (information, belief or data) into a theory (or knowledge base). There are four possible deductive relationships between the current knowledge base (KB), the knowledge, and the new KB which arises as a result [Kowalski79].

1. The new information is already deducible from the current KB. The new KB, as a result, is identical with the current one.
2. The current KB = $KB_1 \cup KB_2$ can be decomposed into two parts. One part KB_1 together with the new information can be used to deduce the other part KB_2. The new KB is KB_1 together with the new information.
3. The new information violates the integrity of the current KB. Integrity can be restored by modifying or rejecting one or more of the assumptions which lead to the contradiction.
4. The new information is independent from the current KB. The new KB is obtained by adding the new information to the current KB.

In case (4) the KB can, alternatively, be augmented by an explanation for the new datum [Kowalski79, Kunifiji et al.86, Kakas&Mancarella91a]. In [Kunifiji et al.86] the authors have developed a system for knowledge assimilation (KA) based on this use of abduction. They have identified the basic issues associated with such a system and proposed solutions for some of these.

Various motivations can be given for the addition of an abductive explanation instead of the new datum in case (4) of the process of KA. For example, in natural language understanding or in diagnosis, the assimilation of information naturally demands an explanation. In other cases the addition of an explanation as a way of assimilating new data is forced by the particular way in which the knowledge is represented in the theory. Consider for example a problem of temporal reasoning formulated in the Event Calculus [Kowalski&Sergot86]. This contains an axiom that expresses the persistence of a property P from the time that it is initiated by an event E to a later time T:

$$\begin{aligned} \mathit{holds\text{-}at}(P, T) \quad \leftarrow \quad & \mathit{happens}(E), \\ & \mathit{time}(E) < T, \\ & \mathit{initiates}(E, P), \\ & \mathit{persists}(\mathit{time}(E), P, T). \end{aligned}$$

New information about the predicate ***holds-at*** can be assimilated by adding an explanation in terms of some event that generates this property together with an appropriate assumption that the property persists [Eshghi88, Shanahan89, Kakas&Mancarella89]. This has the additional effect that the new KB will imply that the property holds until it is terminated in the future [Shanahan89]. This way of assimilating new information can also be used to resolve conflicts between the current KB and the new information [Kakas&Mancarella89, Shanahan89]. Suppose for example that the current KB contains the fact (expressed informally) "Mary has book1 at time t_0" and that the persistence axiom predicts that "Mary has book1 at time t_1" where $t_0 < t_1$. The new information "John has book1 at time t_1" contradicts the prediction, and cannot be added explicitly to the KB. It is however possible to remove the contradiction by adding the explanation that an event has happened where "Mary gives John book1 between t_0 and t_1".

Once a hypothesis has been generated as an explanation for an external datum, it itself needs to be assimilated into the KB. In the simplest situation, the explanation is just added to the KB, i.e. only case (4) applies without further abduction. Case (1) doesn't apply, if abductive explanations are required to be basic. However case (2) may apply, and can be particularly useful for discriminating between alternative explanations for the new information. For instance we may prefer a set of hypotheses which entails information already in the KB, i.e. hypotheses that render the KB as "compact" as possible.

Example 2.1
Suppose the current KB contains

$$\begin{array}{ll} p & \leftarrow q \\ p & \\ r & \leftarrow q \\ r & \leftarrow s \end{array}$$

and r is the new datum to be assimilated. The explanation q is preferable to the explanation s, because q implies both r and p, but s only implies r.

Notice however that the use of case (2) to remove redundant information can cause problems later. If we need to retract previously inserted informations, entailed information which is no longer explicitly in the KB might be lost.

It is interesting to note that case (3) can be used to check the integrity of any abductive hypotheses generated in case (4).

Any violation of integrity detected in case (3) can be remedied in several ways [Kowalski79]. The new input can be retracted as in conventional databases. Alternatively the new input can be upheld and some other assumptions can be withdrawn. This is the case with view updates. The task of translating the update request on the view predicates to an equivalent update on the extensional part (as in case (4) of KA) is achieved by finding an abductive explanation for the update in terms of ground instances of extensional predicates [Kakas&Mancarella90b]. Any violation of integrity is dealt with by changing the extensional part of the database.

The general problem of belief revision has been studied formally in [Gärdenfors88, Nebel89, Nebel91, Doyle91]. Gärdenfors proposes a set of axioms for rational belief revision containing such constraints on the new theory as "no change should occur to the theory when trying to delete a fact that is not already present" and "the result of revision should not depend on the syntactic form of the new data". These axioms ensure that there is always a unique way of performing belief revision. However Doyle argues that, for applications in AI, this uniqueness property is too strong. He proposes instead the notion of "economic rationality", in which the revised sets of beliefs are optimal with respect to a set of preference criteria on the possible beliefs states. This notion has been used to study the evolution of databases by means of updates [Kakas91b].

KA and belief revision are also related to truth maintenance systems. We will discuss truth maintenance and its relationship with abduction in section 6.

3 Default Reasoning viewed as Abduction

Default reasoning concerns the application of general rules to draw conclusions provided the application of the rules does not result in contradictions. Given, for example, the general rules "birds fly" and "penguins are birds that do not fly" and the only fact about Tweety that Tweety is a bird, we can derive the default conclusion that Tweety flies. However, if we are now given the extra information that Tweety is a penguin, we can also conclude that Tweety does not fly. In ordinary, common sense reasoning, the rule that penguins do not fly has priority over the rule that birds fly, and consequent this new conclusion that Tweety does not fly causes the original conclusion to be withdrawn.

One of the most important formalisations of default reasoning is the Default Logic of Reiter [Reiter80]. Reiter separates beliefs into two kinds, ordinary sentences used to express "facts" and default rules of inference used to express general rules. A **default rule** is an inference rule of the form

$$\frac{\alpha(x) : M\beta_1(x), \ldots, \beta_n(x)}{\gamma(x)}$$

which expresses, for all ground instances t of x [3], that $\gamma(t)$ can be derived if $\alpha(t)$ holds and each of $\beta_i(t)$ is consistent, where $\alpha(x)$, $\beta_i(x)$, $\gamma(x)$ are first-order formulae. Default rules provide a way of extending an underlying incomplete theory. Different applications of the defaults can yield different extensions.

As already mentioned in section 1, [Poole et al.87, Poole88a] proposes an alternative formalisation of default reasoning in terms of abduction. Like Reiter, Poole also distinguishes two kinds of beliefs:

- beliefs that belong to a consistent set of first order sentences $\mathcal{F}$ representing "facts", and
- beliefs that belong to a set of first order formulae D representing defaults.

Perhaps the most important difference between Poole's and Reiter's formalisations is that Poole uses sentences (and formulae) of classical first order logic to express defaults, while

[3] We use the notation x to indicate a tuple of variables $x_1, \ldots, x_n$.

Reiter uses rules of inference. Given a Theorist framework $\langle \mathcal{F}, D \rangle$, default reasoning can be thought of as theory formation. A new theory is formed by extending the existing theory $\mathcal{F}$ with a set Δ of sentences which are ground instances of formulae in D. The new theory $\mathcal{F} \cup \Delta$ should be consistent. This process of theory formation is a form of abduction, where ground instances of defaults in D are the candidate abducibles. Poole (theorem 5.1 in [Poole88a]) shows that the semantics of the theory formation framework $\langle \mathcal{F}, D \rangle$ is equivalent to that of an abductive framework $\langle \mathcal{F}', A, \emptyset \rangle$ (see section 1.2) where the default formulae are all atomic. The set of abducibles A consists of a new predicate

$$p_w(x)$$

for each default formula

$$w(x)$$

in D with free variables x. The new predicate is said to "name" the default. The set $\mathcal{F}'$ is the set $\mathcal{F}$ augmented with a sentence

$$\forall x\, [p_w(x) \rightarrow w(x)]$$

for each default in D.

The theory formation framework and its correspondence with the abductive framework can be illustrated by the flying-birds example.

Example 3.1
In this case, the framework $\langle \mathcal{F}, D \rangle$ is [4]

$$\begin{aligned} \mathcal{F} &= \{ \; penguin(x) \rightarrow bird(x), \\ &\qquad penguin(x) \rightarrow \neg fly(x), \\ &\qquad penguin(Tweety), \\ &\qquad bird(John) \} \\ D &= \{ \; bird(x) \rightarrow fly(x) \}. \end{aligned} \tag{1}$$

The priority of the rule that penguins do not fly over the rule that birds fly is obtained by regarding the first rule as a fact and the second rule as a default. The atom $fly(John)$ is a default conclusion which holds in $\mathcal{F} \cup \Delta$ with

$$\Delta = \{ \, bird(John) \rightarrow fly(John) \, \}.$$

We obtain the same conclusion by naming the default (1) by means of a predicate $birds\text{-}fly(x)$, adding to $\mathcal{F}$ the new "fact"

$$birds\text{-}fly(x) \rightarrow [bird(x) \rightarrow fly(x)] \tag{2}$$

and extending the resulting augmented set of facts $\mathcal{F}'$ with the set of hypotheses $\Delta' = \{ \, birds\text{-}fly(John) \, \}$. On the other hand, the conclusion $fly(Tweety)$ cannot be derived, because the extension

$$\Delta = \{ \, bird(Tweety) \rightarrow fly(Tweety) \, \}$$

is inconsistent with $\mathcal{F}$, and similarly the extension

$$\Delta' = \{ \, birds\text{-}fly(Tweety) \, \}$$

is inconsistent with $\mathcal{F}'$.

[4] Here variables occurring in formulae of $\mathcal{F}$ are assumed to be universally quantified. However, formulae of D are to be understood as schemata standing for the set of all their ground instances.

Poole shows that normal defaults without prerequisites in Reiter's default logic

$$\frac{:\mathrm{M}\beta(x)}{\beta(x)}$$

can be simulated by Theorist (abduction) simply by making the predicates $\beta(x)$ abducible. He shows that the default logic extensions in this case are equivalent to maximal sets of ground instances of the default formulae $\beta(x)$ that can consistently be added to the set of facts.

Maximality of abductive hypotheses is a natural requirement for default reasoning, because we want to apply defaults whenever possible. However, maximality is not appropriate for other uses of abductive reasoning. In particular, in diagnosis we are generally interested in explanations which are minimal.

In the attempt to use abduction to simulate default rules in general, however, Poole needs to use integrity constraints. The new theory $\mathcal{F} \cup \Delta$ should be consistent with these constraints. Default rules of the form:

$$\frac{\alpha(x) : \mathrm{M}\beta(x)}{\gamma(x)}$$

are translated into "facts", which are implications

$$\gamma(x) \leftarrow \alpha(x), M_\beta(x)$$

where M_β is a new predicate, and $M_\beta(x)$ is a default formula (abducible). An integrity constraint

$$\neg M_\beta(x) \leftarrow \neg \beta(x)$$

is needed to link the new predicate appropriately with the predicate β. A second integrity constraint

$$\neg M_\beta(x) \leftarrow \neg \gamma(x)$$

is needed to prevent the application of the contrapositive

$$\neg \alpha(x) \leftarrow \neg \gamma(x), M_\beta(x)$$

of the implication, in the attempt to make the implication behave like an inference rule. This use of integrity constraints is different from their intended use in abductive frameworks as presented in section 1.2.

Poole's attempted simulation of Reiter's general default rules is not exact. He presents a number of examples and argues that, where the two formulations differ, Reiter's default logic gives counterintuitive results. In fact, it is possible to dispute some of these examples. But, more importantly, there are other examples where the Theorist approach arguably gives the wrong result. The most important of these is the now notorious Yale shooting problem of [Hanks&McDermott86, Hanks&McDermott87]. As [Morris88] and [Eshghi&Kowalski88] argue, this can be reduced to the simple propositional form

$$p \leftarrow \sim q$$

$$q \leftarrow \sim r.$$

Hanks and McDermott showed, in effect, that the default theory, whose facts consist of the two propositional sentences above and whose defaults are the two normal defaults

$$\frac{: \mathrm{M} \sim q}{\sim q} \quad \frac{: \mathrm{M} \sim r}{\sim r}$$

has two extensions: one in which $\sim r$, and therefore q holds; and one in which $\sim q$, and therefore p holds. The second extension is intuitively incorrect. Hanks and Mc Dermott showed that many other approaches to default reasoning give similarly incorrect results. However, [Morris88] showed that the default theory, which has no facts but contains the two non-normal defaults

$$\frac{: \mathrm{M} \sim q}{p} \quad \frac{: \mathrm{M} \sim r}{q}$$

yields only one extension, containing q, which is the correct result. In contrast, all natural representations of the problem in Theorist give incorrect results.

As [Eshghi&Kowalski88], [Evans89] and [Apt&Bezen90] observe, the Yale shooting problem has the form of a logic program, and interpreting negation in the problem as negation as failure yields only the correct result. Moreover, [Eshghi&Kowalski88] and [Kakas&Mancarella89] show how to retain the correct result when negation as failure is interpreted as a form of abduction.

On the other hand, the Theorist framework does overcome the problem that some default theories do not have extensions and hence cannot be given any meaning within default logic. In the next section we will see that this problem also occurs in logic programming, but that it can also be overcome by an abductive treatment of negation as failure. We will also see that the resulting abductive interpretation of negation as failure can be regarded as a hybrid which treats defaults as abducibles in Theorist but treats clauses as inference rules in default logic.

The inference rule interpretation of logic programs, makes logic programming extended with abduction especially suitable for default reasoning. Integrity constraints can be used, not for preventing application of contrapositives, but for representing negative information and exceptions to defaults.

Example 3.2
The default (1) in the flying-birds example 3.1 can be represented by the logic program

$$fly(x) \leftarrow bird(x), birds\text{-}fly(x),$$

with the abducible predicate $birds\text{-}fly(x)$. Note that this clause is equivalent to the "fact" (2) obtained by renaming the default (1) in Theorist. The exception can be represented by an integrity constraint:

$$\neg fly(x) \leftarrow penguin(x).$$

The resulting logic program, extended by means of abduction and integrity constraints, gives similar results to the Theorist formulation of example 3.1.

In sections 4 and 5 we will see other ways of performing default reasoning in logic programming. In section 4 we will introduce negation as failure as a form of default reasoning, and we will study its relationship with abduction. In section 5 we will consider an extended logic programming framework that contains clauses with negative conclusions and avoids the use of explicit integrity constraints, in some cases.

4 Negation as Failure as Abduction

We noted in the previous section that default reasoning can be performed by means of abduction in logic programming by explicitly introducing abducibles into rules. Default reasoning can also be performed with the use of negation as failure (NAF) [Clark78] in general logic programs. NAF provides a natural and powerful mechanism for performing non-monotonic and default reasoning. As we have seen, it provides a simple solution to the Yale shooting problem. The abductive interpretation of NAF that we will present below provides further evidence for the suitability of abduction for default reasoning.

To see how NAF can be used for default reasoning, we return to the flying-birds example.

Example 4.1
The NAF formulation differs from the logic program with abduction presented in the last section (example 3.2) by employing a negative condition

$$\sim \mathit{abnormal\text{-}bird}(x)$$

instead of a positive abducible condition

$$\mathit{birds\text{-}fly}(x)$$

and by employing a positive conclusion

$$\mathit{abnormal\text{-}bird}(x)$$

in an ordinary program clause, instead of a negative conclusion

$$\neg\, \mathit{fly}(x)$$

in an integrity constraint. The two predicates *abnormal-bird* and *birds-fly* are opposite to one another. Thus in the NAF formulation the default is expressed by the clause

$$\mathit{fly}(x) \leftarrow \mathit{bird}(x), \sim \mathit{abnormal\text{-}bird}(x)$$

and the exception by the clause

$$\mathit{abnormal\text{-}bird}(x) \leftarrow \mathit{penguin}(x).$$

In this example, both the abductive formulation with an integrity constraint and the NAF formulation give the same result.

4.1 Logic programs as abductive frameworks

The similarity between abduction and NAF can be used to give an abductive interpretation of NAF. This interpretation was presented in [Eshghi&Kowalski88] and [Eshghi&Kowalski89], where negative literals are interpreted as abductive hypotheses that can be assumed to hold provided they are consistent with the program and a canonical set of integrity constraints. A general logic program P is thereby transformed into an abductive framework $\langle P^*, A^*, I^* \rangle$ (see section 1) in the following way.

- A new predicate symbol p^* (the opposite of p) is introduced for each p in P, and A^* is the set of all these predicates.
- P^* is P where each negative literal $\sim p(t)$ has been substituted for by $p^*(t)$.
- I^* is a set of all integrity constraints of the form [5]:

$$\forall x \neg [p(x) \wedge p^*(x)] \text{ and}$$
$$\forall x \,[p(x) \vee p^*(x).]$$

The semantics of the abductive framework $\langle P^*, A^*, I^*\rangle$, in terms of **extensions** $P^* \cup \Delta$ of P^*, where $\Delta \subseteq A^*$, gives a semantics for the original program P. A conclusion Q holds with respect to P if and only if Q has an abductive explanation in the framework $\langle P^*, A^*, I^*\rangle$. This transformation of P into $\langle P^*, A^*, I^*\rangle$ is an example of the method, described at the end of section 1.1, of giving a semantics to a language by translating it into another language whose semantics is already known.

The integrity constraints in I^* play a crucial role in capturing the meaning of NAF. The denials express that the newly introduced symbols p^* are the negations of the corresponding p. They prevent an assumption $p^*(t)$ if $p(t)$ holds. On the other hand the disjunctive integrity constraints force a hypothesis $p^*(t)$ whenever $p(t)$ does not hold.

Hence we define the meaning of the integrity constraints I^* as follows: An extension $P^* \cup \Delta$ (which is a propositional Horn theory) of P^* **satisfies** I^* if and only if for every ground atom t,

$$\begin{array}{lcl} P^* \cup \Delta & \not\models & t \wedge t^*, \text{ and} \\ P^* \cup \Delta & \models & t \quad \text{or} \quad P^* \cup \Delta \models t^*. \end{array}$$

[Eshghi&Kowalski89] show that there is a one to one correspondence between stable models [Gelfond&Lifschitz88] of P and abductive extensions of P^*. We recall the definition of stable model:

Let P be a general logic program, and assume that all the clauses in P are ground. For any set M of ground atoms, let P_M be the definite program obtained by deleting from P:

i) each rule that contains a negative literal $\sim A$, with $A \in M$,

ii) all negative literals in the remaining rules.

If the minimal (Herbrand) model of P_M coincides with M, then M is a **stable model** for P.

The correspondence between the stable model semantics of a program P and abductive extensions of P^* is given by:

- For any stable model M of P, the extension $P^* \cup \Delta$ satisfies I^*, where $\Delta = \{d^* \mid d \text{ is a ground atom}, d \notin M\}$.

[5] In the original paper the disjunctive integrity constraints were written in the form

$$\text{Demo}(P^* \cup \Delta, p(t)) \vee \text{Demo}(P^* \cup \Delta, p^*(t)),$$

where t is any ground term. This formulation makes explicit a particular (meta-level) interpretation of the disjunctive integrity constraint. The simpler form

$$\forall x\,[p(x) \vee p^*(x)]$$

is neutral with respect to the interpretation of integrity constraints.

- For any Δ such that $P^* \cup \Delta$ satisfies I^*, there is a stable model M of P, where $M = \{d \,|\, d \text{ is a ground atom}, d^* \notin \Delta\}$.

Notice that the disjunctive integrity constraints in the abductive framework correspond to a totality requirement that every atom must be either true or false in the stable model semantics. Several authors have argued that this totality requirement is too strong, because it prevents us from giving a semantics to some programs, for example $p \leftarrow \sim p$. We would like to be able to assign a semantics to every program in order to have modularity, as otherwise one part of the program can affect the meaning of another unrelated part. We will see below that the disjunctive integrity constraint also causes problems for the implementation of the abductive framework for NAF.

Notice that the semantics of NAF in terms of abductive extensions is more syntactic than model-theoretic. It is a semantics in the sense that it is a non-constructive specification. Similarly, the stable model semantics, as is clear from its correspondence with abductive extensions, is not so much a semantics as a non-constructive specification of what should be computed. The computation itself is performed by means of a proof procedure.

4.2 An abductive proof procedure for logic programming

In addition to having a clear and simple semantics for abduction, it is also important to have an effective method for computing abductive explanations. Any such method will be very useful in practice in view of the many diverse applications of abductive reasoning, including default reasoning. The Theorist framework of [Poole et al.87, Poole88a] provides such an implementation of abduction by means of a resolution based proof procedure.

In their study of NAF through abduction Eshghi and Kowalski [Eshghi&Kowalski89] have defined an abductive proof procedure for NAF in logic programming. We will describe this procedure in some detail as it also serves as the basis for computing abductive explanations more generally within logic programming with other abducibles and integrity constraints (see section 5).

The abductive proof procedure interleaves two types of computation. The first type, referred to as the **abductive phase**, is standard SLD- resolution, that generates (negative) hypotheses and adds them to the set of abducibles being generated, while the second type, referred to as the **consistency phase** [6], incrementally checks that the hypotheses satisfy the integrity constraints for NAF, I^*. Integrity checking of a hypothesis $p^*(t)$ reasons forward one step using a denial integrity constraint to derive the goal $\leftarrow p(t)$. Thereafter it reasons backward in SLD-fashion in all possible ways. Integrity checking succeeds if all the branches of the resulting search space fail finitely, in other words, if the contrary of $p^*(t)$, namely $p(t)$, finitely fails to hold. Whenever the potential failure of a branch of the consistency phase search space is due to the failure of a selected abducible, say $q^*(s)$, a new abductive phase of SLD-resolution is triggered for the goal $\leftarrow q(s)$, to ensure that the disjunctive integrity constraint $q^*(s) \vee q(s)$ is not violated by the failure of both $q^*(s)$ and $q(s)$. This attempt to show $q(s)$ can require in turn the addition of further abductive assumptions to the set of hypotheses which is being generated.

[6] We use the term "consistency phase" for historical reasons. However, in view of the precise definition of integrity constraint satisfaction, some other term might be more appropriate.

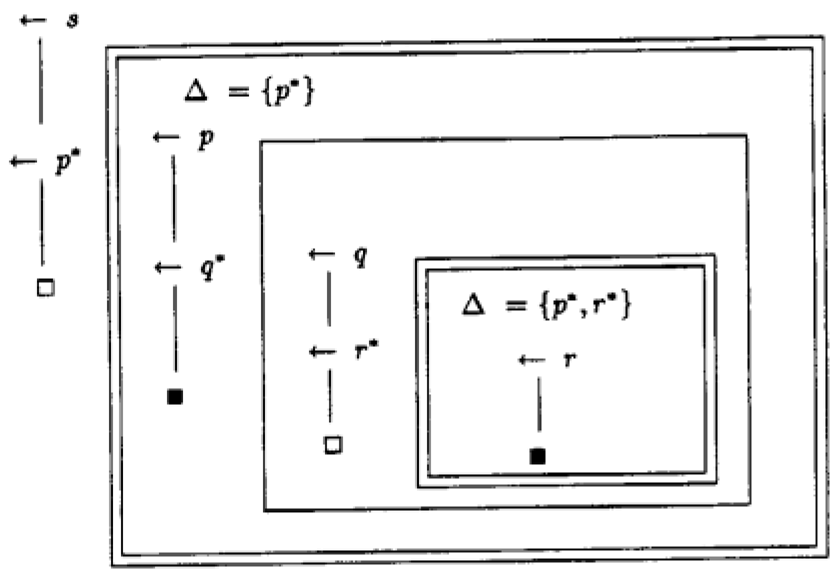


Figure 1: computation for example 4.2

To illustrate the procedure consider the following stratified logic program, which is a minor elaboration of the propositional form of the Yale shooting problem discussed in section 3.

Example 4.2

$$\begin{array}{lcl} s & \leftarrow & \sim p \\ p & \leftarrow & \sim q \\ q & \leftarrow & \sim r \end{array}$$

The query $\leftarrow s$ succeeds with answer $\Delta = \{p^*, r^*\}$. The computation is shown in figure 1. Parts of the search space enclosed by a double box show the incremental integrity checking of the latest abducible added to the explanation Δ. For example, the outer double box shows the integrity check for the abducible p^*. For this we start from $\leftarrow p$ (resulting from the resolution of p^* with the integrity constraint $\neg(p \wedge p^*)$) and resolve backwards in SLD-fashion to show that all branches end in failure, depicted here by a black box. During this consistency phase for p^* a new abductive phase (shown in the single box) is generated when q^* is selected since the disjunctive integrity constraint $q^* \vee q$ implies that failure of q^* is only allowed provided that q is provable. The SLD proof of q requires the addition of r^* to Δ, which in turn generates a new consistency phase for r^* shown in the inner double box. The goal $\leftarrow r$ fails trivially because there are no rules for r and so r^* and the enlarged explanation $\Delta = \{p^*, r^*\}$ is consistent. Tracing the computation backwards, we see that q holds therefore q^* fails and, therefore p^* is consistent and the original query $\leftarrow s$ succeeds.

In general, an abductive phase succeeds if and only if one of its branches ends in a white box (indicating that no subgoals remain to be solved). It fails finitely if and only if all branches end in a black box (indicating that some subgoal cannot be solved). A consistency phase fails if and only if one of its branches ends in a white box (indicating that integrity has been violated). It succeeds finitely if and only if all branches end in a black box (indicating that integrity has not been violated).

It is instructive to compare the computation space of the abductive proof procedure with that of SLDNF. It is easy to see that these are closely related but that they have some important differences. A successful derivation of the abductive proof procedure will produce together with the usual answer obtained from SLDNF additional information, namely the abductive explanation Δ. This additional information can be useful in different ways, in particular to avoid recomputation of negative subgoals. More importantly, as the next example will show, this information will allow the procedure to handle non-stratified programs and queries for which SLDNF is incomplete. In this way the abductive proof procedure generalises SLDNF significantly. Furthermore, the abductive explanation Δ produced by the procedure can be recorded and used in any subsequent revision of the beliefs held by the program, in a similar fashion to truth maintenance systems [Kakas&Mancarella91a]. In fact, this abductive treatment of NAF allows us to identify a close connection between logic programming and truth maintenance systems in general (see section 6). Another important difference is the distinction that the abductive proof procedure for NAF makes between the abductive and consistency phases. This allows a natural extension of the procedure to a more general framework where we have other hypotheses and integrity constraints in addition to those for NAF [Kakas&Mancarella90b, Kakas&Mancarella90c, Kakas&Mancarella90d].

To see how the abductive proof procedure extends SLDNF, consider the following non-stratified program.

Example 4.3

$$\begin{array}{lcl} s & \leftarrow & p \\ s & \leftarrow & q \\ p & \leftarrow & \sim q \\ q & \leftarrow & \sim p \end{array}$$

The query $\leftarrow\ s$ has no SLDNF refutation. Moreover, the SLDNF proof procedure, executing the query, goes into an infinite loop. However, in the corresponding abductive framework the query has two answers, $\Delta = \{p^*\}$ and $\Delta = \{q^*\}$, corresponding to the two stable models of the program. The computation for the first answer is shown in figure 2. The outer abductive phase generates the hypothesis p^* and triggers the consistency phase for p^* shown in the double box. In general, whenever a hypothesis is tested for integrity, we can add the hypothesis to Δ either at the beginning or at the end of the consistency phase. When this addition is done at the beginning (as originally defined in [Eshghi&Kowalski89]) this extra information can be used in any subordinate abductive phase. In this example, the hypothesis p^* is used in the subordinate abductive proof of q to justify the failure of q^* and consequently to render p^* acceptable. In other words, the acceptability of p^* as a hypothesis is proved under the assumption of p^*. The same abductive proof procedure, but where each new hypothesis is added to Δ only at the successful completion of its consistency phase, provides a sound proof procedure for the well-founded semantics [VanGelder et al.88].

Example 4.4
Consider the query $\leftarrow\ p$ with respect to the abductive framework corresponding to the

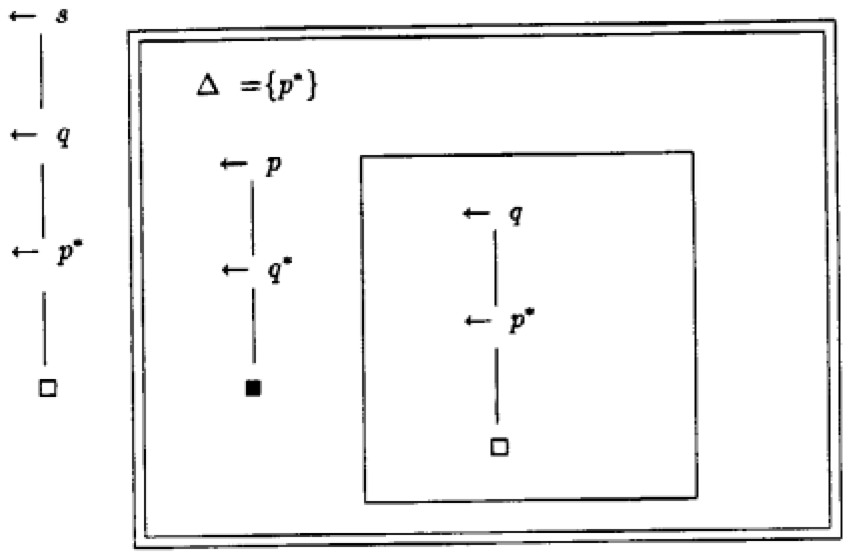


Figure 2: computation for example 4.3

following program

$$\begin{array}{lll} r & \leftarrow & \sim r \\ r & \leftarrow & q \\ p & \leftarrow & \sim q \\ q & \leftarrow & \sim p. \end{array}$$

The abductive proof procedure succeeds with the explanation $\{q^*\}$, but the only set of hypotheses which satisfies the integrity constraints is $\{p^*\}$.

So, as [Eshghi&Kowalski89] show by means of this example, the abductive proof procedure is not always sound with respect to the above abductive semantics of NAF. It is possible, however, to argue that it is the semantics and not the proof procedure that is at fault. Indeed, [Saccà&Zaniolo90, Przymusinski90] and others have argued that the totality requirement of stable models is too strong. They relax this requirement and consider partial or three-valued stable models instead. In the context of the abductive semantics of NAF this is an argument against the disjunctive integrity constraints.

An abductive semantics of NAF without disjunctive integrity constraints has been proposed by [Dung91a]. The abductive proof procedure is sound with respect to this improved semantics. [Satoh&Iwayama92], on the other hand, show how to extend the abductive proof procedure of [Eshghi&Kowalski89] to deal correctly with the stable model semantics. Their extension modifies the integrity checking method of [Kowalski&Sadri87] and deals with arbitrary integrity constraints expressed in the form of denials.

Finally, we note that, in order to capture the semantics more closely for programs such as $p \leftarrow p$ where $\sim p$ holds, we can define a non-effective extension of the proof procedure, that allows infinite failure in the consistency phases.

4.3 Negation as hypothesis revisited

[Dung91a] replaces the disjunctive integrity constraints by a weaker requirement that the set of negative hypotheses Δ be maximal. Unfortunately, simply replacing the disjunctive integrity constraints by maximality does not work, as shown in the following example.

Example 4.5
With this change the program

$$p \leftarrow \sim q$$

has two maximally consistent extensions $\Delta_1 = \{p^*\}$ and $\Delta_2 = \{q^*\}$. However, only the second extension is computed both by SLDNF and by the abductive proof procedure. Moreover, for the same reason as in the case of the propositional Yale shooting problem discussed above, only the second extension is intuitively correct.

To avoid such problems Dung defines a more subtle notion of maximality. He associates with every logic program P an abductive framework $\langle P^*, A^*, I^* \rangle$ where I^* contains only denials

$$\forall x \neg [p(x) \wedge p^*(x)]$$

as integrity constraints [7]. Then, given sets Δ, E of (negative) hypotheses, i.e. $\Delta \subseteq A^*$ and $E \subseteq A^*$, E can be said to **attack** Δ (relative to P^*) if $P^* \cup E \vdash p$ for some $p^* \in \Delta$. Dung calls an extension $P^* \cup \Delta$ of P^* **preferred** if

- $P^* \cup \Delta$ is consistent with I^* and
- Δ is maximal with respect to the property that for every attack E against Δ (relative to P^*) Δ attacks E.

Thus a preferred extension can be thought of as a maximal consistent set of hypotheses that contains its own defence against all attacks. In [Dung91a] a consistent set of hypotheses Δ (not necessarily maximal) satisfying the property of containing its own defence against all attacks is said to be **acceptable** (to P^*). In fact, Dung's definition is not formulated explicitly in terms of the notion of attack and defence, but is equivalent to the one just presented.

Preferred extensions solve the problem with disjunctive integrity constraints in example 4.4 and with maximal consistency semantics in example 4.5. In example 4.4 the preferred extension semantics sanctions the derivation of p by means of an abductive derivation with generated hypotheses $\{q^*\}$. In fact, Dung proves that the abductive proof procedure is always sound with respect to the preferential semantics. In example 4.5 the definition of preferred extension excludes the maximally consistent extension $\{p^*\}$, because there is no defence against the attack q^*.

The preferred extension semantics provides a unifying framework for various approaches to the semantics of negation in logic programming. [Kakas&Mancarella91b] show that it is equivalent to Saccà and Zaniolo's partial stable model semantics [Saccà&Zaniolo90]. Like the partial stable model semantics, it includes the stable model semantics as a special case. Dung also shows that the well-founded model [VanGelder et al.88] is the least consistent extension that can be constructed bottom-up from the empty set of negative hypotheses, by adding incrementally all acceptable hypotheses. Thus the well-founded

[7] For simplicity we continue to indicate the abductive framework for NAF in the same way.

semantics is minimalist and sceptical, whereas the preferential semantics is maximalist and credulous.

[Kakas&Mancarella91c, Kakas&Mancarella91d] propose a modification of the preferred extension semantics. Their proposal can be illustrated by the following example.

Example 4.6
In the abductive framework corresponding to the program

$$\begin{array}{lcl} p & \leftarrow & \sim q \\ q & \leftarrow & \sim q \end{array}$$

consider the set of hypotheses $\Delta = \{p^*\}$. The only attack against Δ is $E = \{q^*\}$, and the only attack against E is E itself. Thus Δ is not an acceptable extension of the program according to the preferred extension semantics, because Δ can not defend itself against E. The empty set is the only preferred extension. However, intuitively Δ should be acceptable because the only attack E against Δ attacks itself, and therefore should not be regarded as an acceptable attack against Δ.

To deal with this kind of example, Kakas and Mancarella modify Dung's semantics, increasing the number of ways in which an attack E can be defeated. Whereas Dung only allows Δ to defeat an attack E, they also allow E to defeat itself. They call a set of hypotheses Δ **stable** if

- Δ is maximal with respect to the property that for every attack E against Δ (relative to $P^* \cup \Delta$) $E \cup \Delta$ attacks E.

Note that here the condition "$P^* \cup \Delta$ is consistent with I^*" is subsumed by the new maximality condition. Like the original definition of preferred extension, the definition of stable set of hypotheses was not originally formulated in terms of attack, but is equivalent to the one presented above.

[Kakas&Mancarella91d] argue that the notion of defeating an attack needs to be liberalised further. They illustrate their argument with the following example.

Example 4.7
Consider the program P

$$\begin{array}{lcl} s & \leftarrow & \sim p \\ p & \leftarrow & \sim q \\ q & \leftarrow & \sim r \\ r & \leftarrow & \sim p. \end{array}$$

Here the only attack against the hypothesis s^* is $E = \{p^*\}$. But although $P^* \cup \{s^*\} \cup E$ does not attack E, E is not a valid attack because it is not stable (or acceptable) according to the definition above.

To generalise the reasoning in example 4.7, we need to liberalise further the conditions for defeating E. Kakas and Mancarella suggest a recursive definition in which a set of hypotheses is deemed acceptable if no attack against any hypothesis in the set is acceptable. More formally, given an initial set of hypotheses Δ_0, a set of hypotheses Δ, is **acceptable** to Δ_0 iff

for every attack E against $\Delta \setminus \Delta_0$, E is not acceptable to $\Delta \cup \Delta_0$.

The semantics of a program P can be identified with any Δ which is maximally acceptable to the empty set of hypotheses $\emptyset$.
Notice that, as a special case, we obtain a basis for the definition:

$$\Delta \text{ is acceptable to } \Delta_0 \text{ if } \Delta \subseteq \Delta_0.$$

Therefore, if Δ is acceptable to $\emptyset$ then Δ is consistent.
Notice, too, that applying the recursive definition twice, and starting with the base case, we obtain an approximation to the recursive definition

$$\Delta \text{ is acceptable to } \Delta_0 \text{ if for every attack } E \text{ against } \Delta \setminus \Delta_0,\ E \cup \Delta \cup \Delta_0 \text{ attacks } E.$$

Thus, the stable theories are those which are maximally acceptable to $\emptyset$, where acceptability is defined by this approximation to the recursive definition.

4.4 The abductive proof procedure revisited

As mentioned above, the incorrectness (with respect to the stable model semantics) of the abductive proof procedure can be remedied by adopting the preferred extension or stable theory semantics. This is because the different phases of the proof procedure can be interpreted in terms of the notions of attack and defence. To illustrate this interpretation, consider again figure 1 of example 4.2. The consistency phase for p^*, shown in the outer double box, can be understood as searching for any attack against p^*. The only attack, namely q^*, is counterattacked (thereby defending p^*) by assuming the additional hypothesis r^*, as this implies q. Hence the set $\Delta = \{p^*, r^*\}$ is acceptable, i.e. it can defend itself against any attack, since all attacks against p^* are counterattacked by r^* and there are no attacks against r^*. Similarly, figure 2 of example 4.3 shows how the attack q^* against p^* is counterattacked by p^* itself.

In general, the proof procedure constructs an acceptable set of negative hypotheses, a subset of which is sufficient to solve the original goal. The remaining hypotheses are necessary to defend this sufficient subset against any attack. With the help of this new interpretation it is possible to extend the proof procedure to capture more fully the stable theory semantics and more generally the semantics given by the recursive definition for acceptability at the end of section 4.3. The extension of the proof procedure involves temporarily remembering a (selected) attack E and using E itself together with the subset of Δ generated so far, to counterattack E, in the subordinate abductive phase.
For example 4.6 of section 4.3, as shown in figure 3, to defend against the attack q^* on p^*, we need to temporarily remember q^* and use it in the subordinate abductive phase to prove q and therefore to attack q^* itself.

This reinterpretation of the original abductive proof procedure in terms of an improved semantics, and the extension of the proof procedure to capture further improvements in the semantics, is an interesting example of the interaction that can arise between a program (proof procedure in this case) and its specification (semantics).

5 Abductive Logic Programming

Abductive Logic Programming (ALP) is an extension of LP that supports abduction in general, and not only for NAF. This extension was introduced already in section 1, as

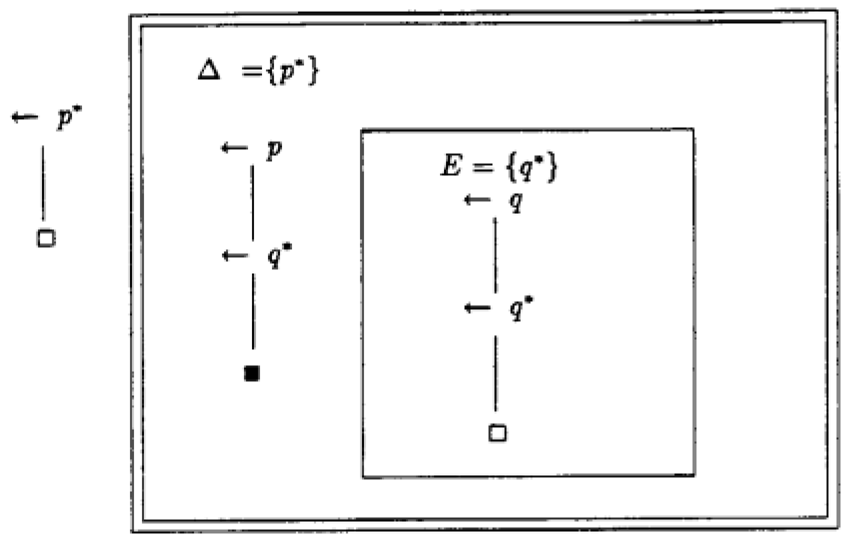


Figure 3: computation for example 4.6 with respect to the revisited proof procedure

the special case of an abductive framework $\langle T, A, I\rangle$, where T is a logic program. In this paper we will assume, without lost of generality, that abducible predicates do not have definitions in T, i.e. do not appear in the heads of clauses in the program T [8]. This assumption has the important advantage that all explanations are thereby guaranteed to be basic.

Semantics and proof procedure for ALP have been proposed by [Eshghi&Kowalski88], [Kakas&Mancarella90a] and [Chen&Warren89]. Chen and Warren extend the perfect model semantics of [Przymusinski89] to include abducibles and integrity constraints over abducibles. Here we shall concentrate on the proposal of Kakas and Mancarella, which extends the stable model semantics.

5.1 Generalised stable model semantics

[Kakas&Mancarella90a] develop a semantics for ALP by generalising the stable model semantics for logic programming. Let $\langle P, A, I\rangle$ be an abductive framework, where P is a general logic program, and let Δ be a subset of A. $M(\Delta)$ is a **generalised stable model** of $\langle P, A, I\rangle$ iff

- $M(\Delta)$ is a stable model of $P \cup \Delta$, and
- $M(\Delta) \models I$.

[8] In the case in which abducibile predicates have definitions in T, auxiliary predicates can be introduced in such a way that the resulting program has no definitions for the abducible predicates, This can be done by means of a transformation similar to the one used to separate extensional and intensional predicates in deductive databases [Minker82]. For example for each abducible predicate $a(x)$ in T we can introduce a new predicate $\delta_a(x)$ and add the clause

$$a(x) \leftarrow \delta_a(x).$$

The predicate $a(x)$ is not abducible anymore, while $\delta_a(x)$ becomes abducible.

Here the semantics of the integrity constraints I is defined by the second condition in the definition above. Consequently, an abductive extension $P \cup \Delta$ of the program P **satisfies** I if and only if there exists a stable model $M(\Delta)$ of $P \cup \Delta$ such that I is true in $M(\Delta)$. This is in the spirit of the epistemic or meta-level view of integrity constraints discussed in section 1.2, in the sense that the integrity constraints I are statements that must hold true about the program $P \cup \Delta$ without the integrity constraints.

The generalised stable models are defined independently from any query. However, given a query Q, we can define an abductive explanation for Q in $\langle P, A, I\rangle$ to be any subset Δ of A such that

- $M(\Delta)$ is a generalised stable model of $\langle P, A, I\rangle$, and
- $M(\Delta) \models Q$.

Example 5.1
Consider the program P:

$$p \leftarrow b$$
$$q \leftarrow a$$

with $A = \{a, b\}$ and integrity constraints I

$$\neg \ [q \wedge b] \text{ and}$$
$$q \vee b.$$

The interpretations $M(\Delta_1) = \{b, p\}$, where $\Delta_1 = \{b\}$, and $M(\Delta_2) = \{a, q\}$, where $\Delta_2 = \{a\}$, are the only generalised stable models of $\langle P, A, I\rangle$. Moreover Δ_1 is an abductive explanation for p.

An alternative, and perhaps more fundamental way of understanding the generalised stable model semantics is by using abduction both for hypothetical reasoning and for NAF. The negative literals in $\langle P, A, I\rangle$ can be viewed as further abducibles according to the transformation described in section 4. The set of abducible predicates then becomes $A \cup A^*$, where A^* is the set of negative abducibles introduced by the transformation. This results in a new abductive framework $\langle P^*, A \cup A^*, I \cup I^*\rangle$, where I^* is the set of special integrity constraints introduced by the transformation of section 4. The semantics of the abductive framework $\langle P^*, A \cup A^*, I \cup I^*\rangle$ can then be given by the sets Δ^* of hypotheses drawn from $A \cup A^*$ which satisfy the integrity constraints $I \cup I^*$.

Example 5.2
Consider P:

$$\begin{array}{lcl} p & \leftarrow & b, \sim q \\ q & \leftarrow & a \end{array}$$

with $A = \{a, b\}$ and $I = \{\}$. If Q is $\leftarrow p$ then $\Delta^* = \{b, q^*, a^*\}$ is an explanation for Q in $\langle P^*, A \cup A^*, I^*\rangle$. Note that a^* is in Δ^* because I^* contains the disjunctive integrity constraint $a \vee a^*$.

Kakas and Mancarella show a one to one correspondence between the generalised stable models of $\langle P, A, I\rangle$ and the sets of hypotheses Δ that satisfy the transformed framework $\langle P^*, A \cup A^*, I \cup I^*\rangle$. Moreover they show that for any abductive explanation Δ^* for a query Q in $\langle P^*, A \cup A^*, I \cup I^*\rangle$, the subset Δ consisting of abducibles only in A is an abductive explanation for Q in $\langle P, A, I\rangle$.

Example 5.3
Consider the framework $\langle P, A, I\rangle$ and the query Q of the example 5.2. We have already seen that $\Delta^* = \{b, q^*, a^*\}$ is an explanation for Q in $\langle P^*, A \cup A^*, I^*\rangle$. Accordingly the subset $\Delta = \{b\}$ is an explanation for Q in $\langle P, A, I\rangle$.

Note that the generalised stable model semantics as defined above requires that for each abducible a, either a or a^* holds. This can be relaxed by dropping the disjunctive integrity constraints $a \vee a^*$ and defining the set of abducible hypotheses A to include both a and a^*.

Generalised stable models combine the use of abduction for default reasoning (in the form of NAF) with the use of abduction for other forms of hypothetical reasoning. The first kind of abduction requires hypotheses to be maximised, while the second kind usually requires them to be minimised. The definition of generalised stable models appropriately maximises NAF hypotheses, but neither maximises nor minimises other hypotheses. In practice, however, the abductive proof procedure generates only hypotheses that are relevant for a proof. Because of this, it tends to minimise the generation of both kinds of hypotheses. On the other hand, the proof procedure also generates as many hypotheses as it needs for a proof. In this sense, it tends to maximise the generation of hypotheses. This property of the proof procedure and its relationship with the semantics needs to be investigated further.

5.2 Abductive proof procedure for ALP

In [Kakas&Mancarella90b, Kakas&Mancarella90c, Kakas&Mancarella90d] proof procedures are given to compute abductive explanations in ALP. These extend the abductive proof procedure for NAF [Eshghi&Kowalski89] described in section 4.2, retaining the basic structure which interleaves an abductive phase that generates and collects abductive hypotheses with a consistency phase that incrementally checks these hypotheses for integrity. We will illustrate these extended proof procedure by means of examples.

Example 5.4
Consider again example 4.2. The abductive proof procedure for NAF fails on the query $\leftarrow p$. Ignoring, for the moment, the construction of the set Δ, the computation is that shown inside the outer double box of figure 1 with the abductive and consistency phases interchanged, i.e. the type of each box changed from a double box to a single box and vice-versa. Suppose now that we have the same program and query but in an ALP setting where the predicate r is abducible. The query will then succeed with the explanation $\Delta = \{q^*, r\}$ as shown in figure 4. As before the computation arrives at a point where r needs to be proved. Whereas this failed before, this succeeds now by abducing r. Hence by adding the hypothesis r to the explanation we can ensure that q^* is acceptable.

An important feature of the abductive proof procedures is that they avoid performing a full general-purpose integrity check (such as the forward reasoning procedure of [Kowalski&Sadri88]). In the case of a negative hypothesis, q^* for example, a general-purpose forward reasoning integrity check would have to use rules in the program such as $p \leftarrow q^*$ to derive p. The optimised integrity check in the abductive proof procedures, however, avoids this inference and only reasons forward one step with the integrity constraint $\neg(q \wedge q^*)$, deriving the resolvent $\leftarrow q$, and then reasoning backward from the

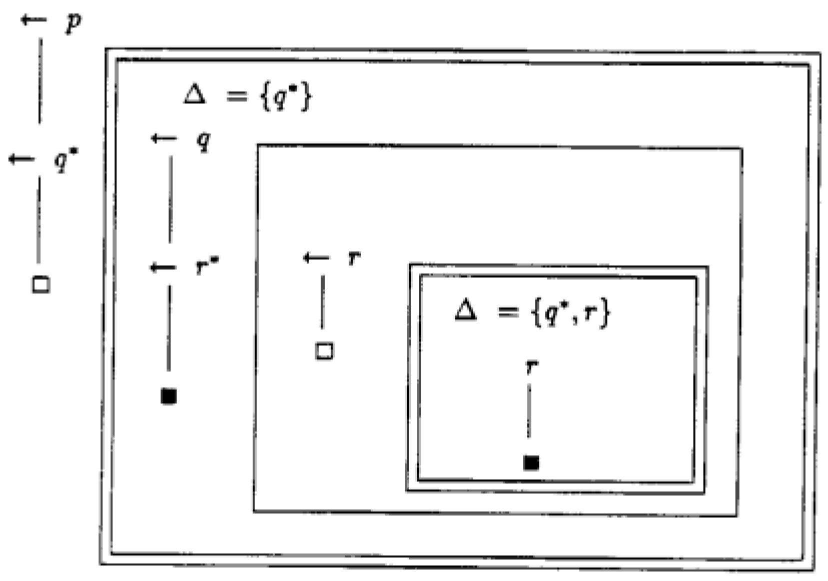


Figure 4: extended proof procedure for example 4.2

resolvent.

Similarly, the integrity check for a positive hypothesis, r for example, avoids reasoning forward with any rules which might have r in the body. Indeed, in a case, such as the example 5.4 above, where there are no domain specific integrity constraints, the integrity check for a positive abducible, such as r, simply consists in checking that its complement, in our example r^*, does not belong to Δ.

To ensure that this optimised form of integrity check is correct, the proof procedure is extended to record those positive abducibles it needs to assume absent to show the integrity of other abducibles in Δ. So whenever a positive abducible, which is not in Δ, is selected in a branch of a consistency phase the procedure fails on that branch and at the same time records that this abducible needs to be absent. This extension is illustrated by the following example.

Example 5.5
Consider the program

$$\begin{aligned} p &\leftarrow \sim q, r \\ q &\leftarrow r \end{aligned}$$

where r is abducible and the query is $\leftarrow p$ (see figure 5). The acceptability of q^* requires the absence of the abducible r. The simplest way to ensure this is by adding r^* to Δ. This, then, prevents the abduction of r and the computation fails. Notice that the proof procedure does not reason forward from r to test its integrity. This test has been performed backwards in the earlier consistency phase for q^*, and the addition of r^* to Δ ensures that it is not necessary to repeat it.

The way in which the absence of abducibles is recorded depends on how the negation of abducibles is interpreted. Under the stable and generalised stable model semantics,

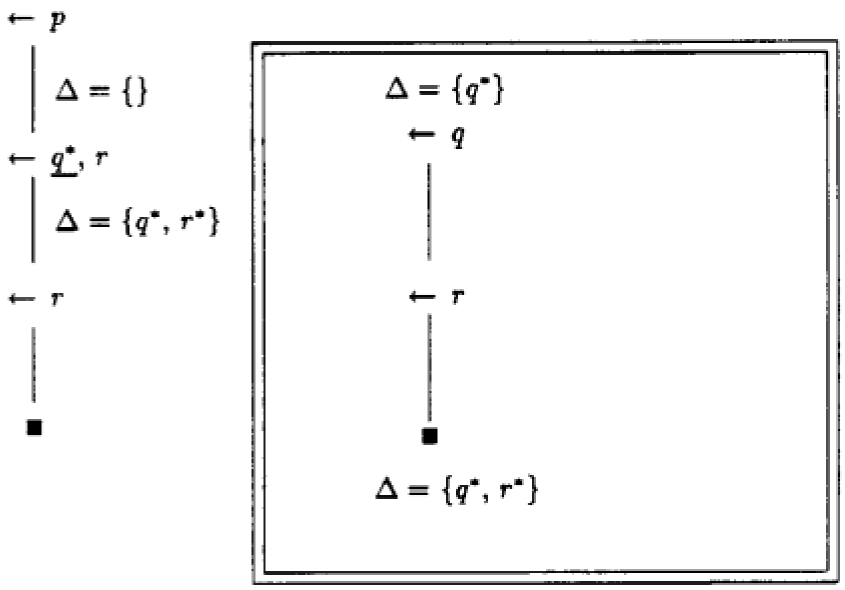


Figure 5: extended proof procedure for example 5.5

as we have assumed in example 5.5 above, the required failure of a positive abducible is recorded by adding its complement to Δ. However, in general it is not always appropriate to assume that the absence of an abducible implies its negation. On the contrary, it may be appropriate (see section 5.3) to treat abducibles as open rather than closed, and correspondingly to treat the negation of abducible predicates as open. As we shall argue later, this might be done by treating such a negation as a form of explicit negation, which is also abducible. In this case recording the absence of a positive abducible by adding its complement to Δ is too strong, and we will use a separate (purely computational) data structure to hold this information.

Integrity checking can also be optimised when there are domain specific integrity constraints, provided the constraints can be formulated as denials [9] containing at least one literal whose predicate is abducible. In this case the abductive proof procedure needs only a minor extension [Kakas&Mancarella90c, Kakas&Mancarella90d]: when a new hypothesis is added to Δ, the proof procedure resolves the hypothesis against any integrity constraint containing that hypothesis, and then reasons backward from the resolvent. To illustrate this extension consider the following example.

Example 5.6
Let the abductive framework be:

[9]Notice that any integrity constraint can be transformed into a denial (possibly with the introduction of new auxiliary predicates). For example:

$$p \leftarrow q \equiv \neg[q \wedge \neg p],$$
$$p \vee q \equiv \neg[\neg p \wedge \neg q].$$

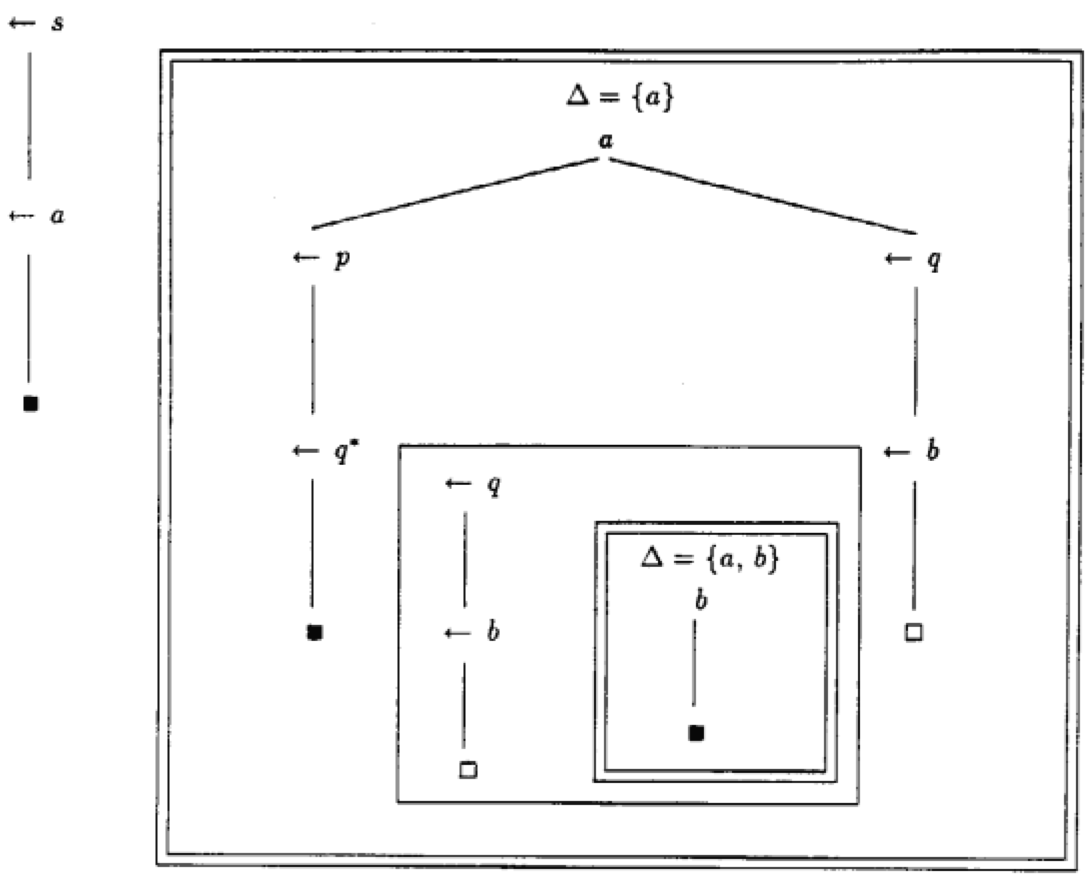


Figure 6: extended computation for example 5.6

$$\begin{array}{llll} P: & s \leftarrow a & I: & \neg[a \wedge p] \\ & p \leftarrow \sim q & & \neg[a \wedge q] \\ & q \leftarrow b & & \end{array}$$

where a, b are abducible and the query is $\leftarrow s$ (see figure 6).

Assume that the integrity check for a is performed Prolog-style, by resolving first with the first integrity constraint and then with the second. The first integrity constraint requires the additional hypothesis b as shown in the inner single box. The integrity check for b is trivial, as b does not appear in any integrity constraint. But $\Delta = \{a, b\}$ violates the integrity constraints, as can be seen by reasoning forward from b to q and then resolving with the second integrity constraint $\neg[a \wedge q]$. However, the proof procedure does not perform this forward reasoning and does not detect this violation of integrity at this stage. Nevertheless the proof procedure is sound because the violation is found later by backward reasoning when a is resolved with the second integrity constraint. This shows that $\Delta = \{a\}$ is unacceptable because it is incompatible with b which is needed to defend Δ against the attack q^*.

In summary, the overall effect of additional integrity constraints is to increase the size of the search space during the consistency phase, with no significant change to the basic structure of the backward reasoning procedure.

The abductive proof procedures described above suffer from the same soundness problem shown in section 4 for the abductive proof procedure for NAF. This problem can be solved similarly, by replacing stable models with any of the non-total semantics for NAF mentioned in section 4 (partial stable models, preferred extensions, stable theories).

Finally, we note that the abductive proof procedures described here perform many of the functions of a truth maintenance system. The relationships between ALP and truth maintenance will be discussed in section 6.

5.3 Stable model semantics extended with explicit negation

In general logic programs, negative information is inferred by means of NAF. This is appropriate when the Closed World Assumption (CWA) [Reiter78], that the program gives a complete definition of the positive instances of a predicate, can safely be applied. It is not appropriate when the definition of a predicate is incomplete and therefore "open" (OWA), as in the case of abducible predicates.
For open predicates it is possible to extend logic programs to allow **explicit negation** in the conclusions of clauses. (As we shall see later, in sections 5.7 and 5.8, this is related to the use of integrity constraints expressed in the form of denials.) In this section we will discuss the extension proposed by [Gelfond&Lifschitz90]. This extension is based on the stable model semantics, and can be understood, therefore, in terms of abduction, as we have already seen.

Gelfond and Lifschitz define the notion of **extended logic programs**, consisting of clauses of the form:

$$L_0 \leftarrow L_1, \ldots, L_m, \sim L_{m+1}, \ldots, \sim L_n,$$

where $n \geq m \geq 0$ and each L_i is either an atom (A) or the explicit negation of an atom $(\neg A)$. This negation denoted by "$\neg$" is called "classical negation" in [Gelfond&Lifschitz90]. However, as we will see below, because the contrapositive of extended clauses do not hold, the term "classical negation" is inaccurate. For this reason we use the term "explicit negation" instead.

The semantics of an extended program is given by its answer sets, which are like stable models but consist of both positive and (explicit) negative literals. Perhaps the easiest way to understand the semantics is to transform the extended program P into a general logic program P' without explicit negation, and to apply the stable model semantics to the resulting general logic program. The transformation consists in replacing every occurrence of explicit negation $\neg p(t)$ by a new (positive) atom $p'(t)$. The stable models of P', which do not contain an implicit contradiction of the form $p(t)$ and $p'(t)$, correspond to the **consistent answer sets** of P. The corresponding answer sets of P contain explicit negative literals $\neg p(t)$ wherever the stable models contain $p'(t)$. In [Gelfond&Lifschitz90] the answer sets are defined directly on the extended program by modifying the definition of the stable model semantics. The consistent answer sets of P also correspond to the generalised stable models (see section 5.1) of P' with a set of integrity constraints

$\neg\, [p(x) \wedge p'(x)]$, for every predicate p.

In the general case a stable model of P' might contain an implicit contradiction of the form $p(t)$ and $p'(t)$. In this case the corresponding **inconsistent answer set** is defined to be the set of all the ground (positive and explicit negative) literals. It is in this sense that explicit negation can be said to be "classical". The same effect can be obtained by explicitly augmenting P' by the clauses

$$q(x) \leftarrow p(x), p'(x)$$

for every predicate symbol q in P'. Then the **answer sets** of P simply correspond to the stable models of the augmented set of clauses. If these clauses are not added, then the resulting treatment of negation gives rise to a **paraconsistent** logic, i.e. one in which contradictions can be tolerated.

Notice that, although Gelfond and Lifschitz define the answer set semantics directly without transforming the program and then applying the stable model semantics, the transformation can also be used with any other semantics for the resulting transformed program. Thus [Przymusinski90] for example applies the well-founded semantics to extended logic programs. Similarly any other semantics can also be applied. This is one of the main advantages of transformational semantics in general.

An important problem for the practical use of extended programs is how to distinguish whether a negative condition is to be interpreted as explicit negation or as NAF. We will discuss this problem in section 7.

5.4 Simulation of abduction through NAF

[Satoh&Iwayama91] show that an abductive logic program can be transformed into a logic program without abducibles but where the integrity constraints remain. Although they do not employ explicit negation, the transformation implicitly simulates explicit negation by the introduction of new predicates. For each abducible predicate p in A, a new predicate p' is introduced representing the complement [10] of p and a new pair of clauses:

$$p(x) \leftarrow \sim p'(x)$$
$$p'(x) \leftarrow \sim p(x)$$

is added to the program. In effect abductive assumptions of the form $p(t)$ are thereby transformed into NAF assumptions of the form $\sim p'(t)$. Satoh and Iwayama apply the generalised stable model semantics to the transformed program. However, as we have already remarked in the case of the semantics of explicit negation, the transformational semantics, which is effectively employed by Satoh and Iwayama, has the advantage that any semantics can be used for the resulting transformed program (e.g. as in [Pereira et al.91a], see below).

Example 5.7
Consider the abductive framework $\langle P, A, I\rangle$ of example 5.1. The transformation generates a new theory P' with the additional clauses

$$a \leftarrow \sim a'$$

[10] Satoh and Iwayama use the notation $p^*(x)$ instead of $p'(x)$ and consider only propositional programs.

$$a' \leftarrow \sim a$$
$$b \leftarrow \sim b'$$
$$b' \leftarrow \sim b.$$

P' has two generalised stable models that satisfy the integrity constraints, namely $M'_1 = M(\Delta_1) \cup \{a'\} = \{b, p, a'\}$, and $M'_2 = M(\Delta_2) \cup \{b'\} = \{a, q, b'\}$ where $M(\Delta_1)$ and $M(\Delta_2)$ are the generalised stable models seen in example 5.1.

Similar methods for transforming abductive assumptions into NAF assumptions are employed by [Inoue91b] and [Pereira et al.91a]. They transform extended logic programs augmented with abduction into extended logic programs without abduction by adding to the program a new pair of clauses

$$p(x) \leftarrow \sim \neg p(x)$$
$$\neg p(x) \leftarrow \sim p(x)$$

for each abducible predicate p. Notice that the transformation is identical to that of Satoh and Iwayama, except for the use of explicit negation instead of new predicates. [Inoue91b] and [Pereira et al.91a] assign different semantics to the resulting program. Whereas [Inoue91b] applies the answer set semantics, [Pereira et al.91a] apply the well-founded semantics and the extended stable model semantics of [Przymusinski90]. The well-founded semantics can be thought of as representing a minimal incomplete view of the world and the extended stable model semantics as representing different ways of extending this view by abducing negative hypotheses. [Pereira et al.91c] have also developed proof procedures for this semantics. These procedures can be used as abductive proof procedure for ALP.

As mentioned above, [Pereira et al.91a] understand the transformed programs in terms of (three-valued) extended stable models. The extended stable model semantics has the advantage that it gives a semantics to every logic program and it does not force abducibles to be either believed or disbelieved. But the advantage of the transformational approach, as we have already remarked, is that the semantics of the transformed program is independent of the transformation. Any semantics can be used for the transformed program (including even a transformational one, e.g. replacing explicitly negated atoms $\neg p(t)$ by a new atom $p'(t)$).

5.5 Computation of abduction through TMS

[Satoh&Iwayama91] present a method for computing generalised stable models for logic programs with integrity constraints represented as denials. The method is a bottom-up computation based upon the TMS procedure of [Doyle79]. Although the computation is not goal-directed, goals (or queries) can be represented as denials and be treated as integrity constraints.

Compared with other bottom-up procedures for computing generalised stable model semantics, which first generate stable models and then test the integrity constraints, the method of Satoh and Iwayama dynamically uses the integrity constraints during the process of generating the stable models, in order to prune the search space more efficiently.

Example 5.8
Consider the program P:

$$\begin{aligned} p &\leftarrow q \\ r &\leftarrow \sim q \\ q &\leftarrow \sim r \end{aligned}$$

and the set of integrity constraints $I = \{\neg p\}$. P has two stable models $M_1 = \{p, q\}$ and $M_2 = \{r\}$, but only M_2 satisfies I. The proof procedure of [Satoh&Iwayama91] deterministically computes only the intended model M_2, without also computing and rejecting M_1.

5.6 Restoring consistency of answer sets

The answer sets of an extended program are not always consistent.

Example 5.9
The extended logic program:

$$\begin{aligned} &\neg fly(x) \leftarrow \sim bird(x) \\ &fly(x) \leftarrow bat(x) \\ &bat(Tom) \end{aligned}$$

has no consistent answer set.

As mentioned in section 5.3, this problem can be dealt with by employing a paraconsistent semantics. Alternatively, in some cases it is possible to restore consistency by removing some of the NAF assumptions implicit in the answer set. In the example above we can restore consistency by rejecting the NAF assumption $\sim bird(Tom)$ even though $bird(Tom)$ does not hold. We then get the consistent set $\{bat(Tom), fly(Tom)\}$. This problem has been studied in [Dung91b] and [Pereira et al.91b]. Both of these studies are primarily concerned with the related problem of inconsistency of the well-founded semantics when applied to extended logic programs [Przymusinski90].

To deal with the problem of inconsistency in extended logic programs, [Dung91b] applies the definition of the preferred extension semantics to a new abductive framework derived from an extended logic program. An extended logic program P is first transformed into an ordinary general logic program P' by renaming explicitly negated literals $\neg p(t)$ by positive literals $p'(t)$. The resulting program is then further transformed into an abductive framework by renaming NAF literals $\sim q(t)$ by positive literals $q^*(t)$ and adding the integrity constraints

$$\forall x \neg [q(x) \wedge q^*(x)]$$

as described in section 4.3. Thus if p' expresses the explicit negation of p the set A^* will contain p'^* as well as p^*. Moreover Dung includes in I^* additional integrity constraints of the form

$$\forall x \neg [p(x) \wedge p'(x)]$$

to prevent contradictions.

Extended preferred extensions are then defined in the same way as preferred extensions in section 4 but with this new set I^* of integrity constraints. The new integrity constraints in I^* have the effect of removing a NAF hypothesis when it leads to a contradiction.

[Pereira et al.91b] employ a similar approach in the context of Przymuszynski's extended stable models [Przymusinski90]. It consists in identifying explicitly all the possible sets of NAF hypotheses which lead to an inconsistency and then restoring consistency by removing at least one hypothesis from each such set. This method can be viewed as an application of belief revision, where if inconsistency can be attributed to an abducible hypothesis or a retractable atom (see below section 5.7), then we can reject the hypothesis to restore consistency. In fact Pereira, Aparicio and Alferes have also used this method to study counterfactual reasoning [Pereira et al.91d].

Both methods can deal only with inconsistencies that can be attributed to NAF hypotheses, as shown by the following example.

Example 5.10
It is not possible to restore consistency by removing NAF hypotheses given the program:

$$p$$

$$\neg p.$$

However, [Inoue91b, Inoue91a] suggests a general method of restoring consistency, which is applicable to this case. This method (see section 5.8) is based on [Poole88a] and [Geffner90] and consists in isolating inconsistencies by finding maximally consistent subprograms. In this approach a knowledge system is represented by a pair (P, H), where:

1. P and H are both extended logic programs,
2. P represents a set of facts,
3. H represents a set of assumptions.

The semantics is given using abduction as in [Poole88a] (see section 3) in terms of theory extensions $P \cup E$ of P, with $E \subseteq H$ maximal with respect to set inclusion, such that $P \cup E$ has a consistent answer set.

In this approach, whenever the answer set of an extended logic program P is inconsistent, it is possible to reason with it by regarding it as a knowledge system of the form

$$(\emptyset, P).$$

For example 5.10 this will give two alternative semantics, $\{p\}$ or $\{\neg p\}$.

5.7 Abduction as retractability

An alternative way of viewing abduction, which emphasises the defeasibility of abducibles, is **retractability** [Goebel et al.86]. Instead of regarding abducibles as atoms to be consistently added to a theory, they can be considered as assertions in the theory to be retracted in the presence of contradictions until consistency (or integrity) is restored (c.f. section 5.6).

One approach to this understanding of abduction is presented in [Kowalski&Sadri88]. Kowalski and Sadri present a transformation from a general logic program P with integrity constraints I, together with some indication of how to restore consistency, to a new general logic program P' without integrity constraints. Restoration of consistency is indicated by nominating one atom as retractable in each integrity constraint [11]. Integrity constraints are represented as denials, and the atom to be retracted must occur positively in the integrity constraint. The (informally specified) semantics is that whenever an integrity constraint of the form

$$\neg [p \wedge q]$$

has been violated, where the atom p has been nominated as retractable, then consistency should be restored by retracting the instance of the clause of the form

$$p \leftarrow r$$

which has been used to derive the inconsistency. Notice that retracting abducible hypotheses is a special case where the abducibility of a predicate a is represented by an assertion

$$a(x).$$

To avoid inconsistency, the program P with integrity constraints I can be transformed to a program P' without integrity constraints which is always consistent with I; and if P is inconsistent with I, then P' represents one possible way to restore consistency (relative to the choice of the retractable atom).

Given an integrity constraint of the form

$$\neg [p \wedge q]$$

where p is retractable, the transformation replaces every clause of the form

$$p \leftarrow r$$

by the clause

$$p \leftarrow r, \sim q$$

where the condition $\sim q$ needs to be further transformed, if necessary, into general logic program form, and where the transformation needs to be repeated for every integrity constraint. Kowalski and Sadri show that if P is a stratified program with appropriately stratified integrity constraints I, so that the transformed program P' is stratified, then P' computes the same consistent answers as P with I.

The Kowalski-Sadri transformation is (almost) the inverse of the Eshghi-Kowalski transformation, which interprets NAF as abduction. To see this, consider again the propositional form of the Yale shooting problem.

Example 5.11
Given the program

$$p \leftarrow \sim q$$
$$q \leftarrow \sim r$$

applying the Eshghi-Kowalski transformation gives

$$p \leftarrow q^*$$

[11] Many different atoms can be retractable in the same integrity constraint. Alternative ways of nominating retractable atoms correspond to alternative ways of restoring consistency in P.

$$q \leftarrow r^*$$

$$\neg [p \wedge p^*]$$

$$\neg [q \wedge q^*]$$

$$\neg [r \wedge r^*]$$

together with the disjunctive integrity constraints. To apply the Kowalski-Sadri transformation these disjunctive integrity constraints are replaced by the stronger (but retactable) assertions

$$p^*$$

$$q^*$$

$$r^*.$$

Applying the Kowalski-Sadri transformation now yields

$$\begin{array}{lcl} p & \leftarrow & q^* \\ q & \leftarrow & r^* \\ p^* & \leftarrow & \sim p \\ q^* & \leftarrow & \sim q \\ r^* & \leftarrow & \sim r. \end{array}$$

If we are only interested in the clauses defining the predicates, p, q and r, in the original program, this can be simplified to

$$p \leftarrow \sim q$$

$$q \leftarrow \sim r$$

which is the original program.

It is interesting to note that the (informal) **retraction semantics** of the intermediate program with integrity constraints and retractable assumptions yields the single (correct) semantics for this example, namely the one in which the assumption q^* is retracted. It would be useful to study the retraction semantics in more general and more formal terms and to compare it with the preferred extension and stable theory semantics.

The retraction semantics and the associated transformation can be applied more generally to cases of default reasoning where the retractable atoms do not correspond to abducible predicates.

Example 5.12
Consider the program

$$\begin{array}{l} fly(x) \leftarrow bird(x) \\ walk(x) \leftarrow ostrich(x) \\ bird(x) \leftarrow ostrich(x) \\ ostrich(John) \end{array}$$

and the integrity constraint

$$\neg [fly(x) \wedge ostrich(x)],$$

with $fly(x)$ retractable. The integrity constraint is violated, because both $ostrich(John)$ and $fly(John)$ hold. Integrity can be restored by retracting the instance

$$fly(John) \leftarrow bird(John)$$

of the first clause in the program.

Similarly the transformed program avoids inconsistency in general by replacing the first clause and the integrity constraint by the more restrictive clause

$$fly(x) \leftarrow bird(x), \sim ostrich(x).$$

5.8 Rules and exceptions in logic programming

One problem with the retraction semantics is that the equivalence of the original program with the transformed program was proved only in the case where the transformed program is locally stratified. Moreover the proof of equivalence is based on a tedious comparison of search spaces for the two programs. This problem was solved in a subsequent paper [Kowalski&Sadri90] by re-expressing integrity constraints as extended clauses where the retractable atoms are expressed as explicitly negated conclusions. By appropriately modifying the answer set semantics to retract clauses whose positive conclusions contradict clauses with negative conclusions, the equivalence of the original program and the transformed program can be proved more simply and without any restrictions. Moreover, the new formulation with explicitly negated conclusions is more informative than the earlier formulation with integrity constraints, which only constrained positive information and did not add negative information explicitly.

In the new formulation it is natural to interpret clauses with negative conclusions as exceptions, and clauses with positive conclusions as default rules. In the flying-bird example of the previous section, in particular, the integrity constraint

$$\neg [fly(x) \wedge ostrich(x)]$$

with $fly(x)$ retractable would now be formulated as an exception

$$\neg fly(x) \leftarrow ostrich(x)$$

to the "general" rule

$$fly(x) \leftarrow bird(x).$$

To capture the intention that exceptions should override general rules, the answer set semantics is modified, so that instances of clauses with positive conclusions are retracted if they are contradicted by explicit negative information.

Kowalski and Sadri also present a new transformation, which preserves the new semantics, and is a more elegant form of the original transformation. In the case of the flying-birds example the new transformation gives the clause

$$fly(x) \leftarrow bird(x), \sim \neg fly(x).$$

This can be further transformed by "macroprocessing" the call to $\neg fly(x)$, giving the result of the original transformation

$$fly(x) \leftarrow bird(x), \sim ostrich(x).$$

In general, the new transformation introduces a new condition

$$\sim \neg p(t)$$

into every clause with a positive conclusion $p(t)$. The condition is vacuous if there are no exceptions with $\neg p$ in the conclusion. The answer set semantics of the new program is equivalent to the modified answer set semantics of the original program, and both are consistent. Moreover, the transformed program can be further transformed into a general logic program by renaming explicit negations $\neg p$ by new positive predicates p'. Because of this renaming, positive and negative predicates can be handled symmetrically, and therefore in effect clauses with positive conclusions can represent exceptions to rules with (renamed) negative conclusions. Thus, for example, a negative rule such as

$$\neg fly(x) \leftarrow ostrich(x)$$

with a positive exception

$$fly(x) \leftarrow super\text{-}ostrich(x)$$

can be transformed into a clause

$$\neg fly(x) \leftarrow ostrich(x), \sim fly(x)$$

and all occurrences of the negative literal $\neg fly(x)$ can be renamed by a new positive literal $fly'(x)$.

A more direct approach to the problem of treating positive and negative predicates symmetrically in default reasoning is presented in [Inoue91b, Inoue91a] following the methods of [Poole88a] and [Geffner90] (see section 5.6 for a discussion). This work is another interesting application of the notion of maximal consistency to extend logic programming for default reasoning.

As a possible direction for future work, it would be desiderable to reconcile the different approaches of Inoue and of Kowalski and Sadri. Such a reconcilation might attempt to treat NAF hypotheses and other kinds of defaults uniformly as cases of abductive reasoning, generalising appropriately the preferred extension and stable theory semantics of NAF.

5.9 A methodology for default reasoning with explicit negation

Compared with other authors, who primarily focus on extending or modifying the semantics of logic programming to deal with default reasoning, [Pereira et al.91a] develop a methodology for performing default reasoning with extended logic programs. Defaults of the form "normally if q then p" are represented by an extended clause

$$p \leftarrow q, \sim \neg nameqp, \sim \neg p \qquad (3)$$

where the condition $nameqp$ is a name given to the default. The condition $\sim \neg p$ deals with exceptions to the conclusion of the rule, whilst the condition $\sim \neg nameqp$ deals with exceptions to the rule itself. An exception to the rule would be represented by an extended clause of the form

$$\neg nameqp \leftarrow r$$

where the condition r represents the conditions under which the exception holds. In the flying-birds example, the second clause of

$$fly(x) \leftarrow bird(x), \sim \neg birds\text{-}fly, \sim \neg fly(x) \tag{4}$$
$$\neg birds\text{-}fly(x) \leftarrow penguin(x) \tag{5}$$

expresses that the default named *birds-fly* does not apply for penguins.

The possibility of expressing both exceptions to rules as well as exceptions to predicates is useful for representing hierarchies of exceptions. Suppose we want to change (5) to the default rule "penguins usually don't fly". This can be done by replacing (5) by

$$\neg fly(x) \leftarrow penguin(x), \sim \neg penguins\text{-}don't\text{-}fly(x), \sim fly(x) \tag{6}$$

where *penguins-don't-fly* is the name assigned to the new rule. To give preference to the more specific default represented by (6) over the more general default (4), we add the additional clause

$$\neg birds\text{-}fly(x) \leftarrow penguin(x), \sim \neg penguins\text{-}don't\text{-}fly(x).$$

Then to express that superpenguins fly, we can add the rule:

$$\neg penguins\text{-}don't\text{-}fly(x) \leftarrow superpenguin(x).$$

[Pereira et al.91a] use the well-founded semantics extended with explicit negation to give a semantics for this methodology for default reasoning. However it is worth noting that any other semantics of extended logic programs could also be used. For example [Inoue91b, Inoue91a] uses an extension of the answer set semantics (see section 5.6). Moreover Inoue bases his method on a slightly different transformation, where exceptions to conclusions of rules do not need to be given explicitly by the extra-condition $\sim \neg p$ in equation (3).

Note that these methodologies can be seen as a refinement of the direct use of the transformation presented in section 5.7.

5.10 Abduction through deduction from the completion

In the proposals presented so far, hypotheses are generated by backward reasoning with the clauses of logic programs used as inference rules. An alternative approach is presented in [Console et al.91]. Here clauses of programs are interpreted as if-halves of if-and-only-if definitions that are obtained from the completion of the program [Clark78] restricted to non-abducible predicates. Forward reasoning with the only-if-halves of these definitions, starting from the observation to be explained, generates abductive hypotheses deductively.

Given a logic program P with abducible predicates A without definitions in P, let P_C denote the completion of the non-abducible predicates in P. An **explanation formula** for an observation O is the most specific formula F such that

$$P_C \cup \{O\} \models F,$$

where F is more specific than F' iff $\models F \rightarrow F'$.

The authors define also a proof procedure that generates explanation formulas for observations. This proof procedure unfolds a given observation O by means of the completion P_C. Termination and soundness of the proof procedure are ensured for a restricted class of programs (i.e. hierarchical). The explanation formula resulting from the computation characterises all the different abductive explanations for O, as exemplified in the following example.

Example 5.13
Consider the following program P

$$\begin{array}{lll} \textit{wobbly-wheel} & \leftarrow & \textit{broken-spokes} \\ \textit{wobbly-wheel} & \leftarrow & \textit{flat-tyre} \\ \textit{flat-tyre} & \leftarrow & \textit{punctured-tube} \\ \textit{flat-tyre} & \leftarrow & \textit{leaky-valve}, \end{array}$$

where the predicates without definitions are considered to be abducible. The completion P_C is:

$$\textit{wobbly-wheel} \leftrightarrow \textit{broken-spokes} \lor \textit{flat-tyre}$$

$$\textit{flat-tyre} \leftrightarrow \textit{punctured-tube} \lor \textit{leaky-valve}.$$

If O is *wobbly-wheel* then the most specific explanation F is

$$\textit{broken-spokes} \lor \textit{punctured-tube} \lor \textit{leaky-valve},$$

corresponding to the abductive explanations $\Delta_1 = \{\textit{broken-spokes}\}$, $\Delta_2 = \{\textit{punctured-tube}\}$ and $\Delta_3 = \{\textit{leaky-valve}\}$.

A discussion of the general phenomenon that reasoning with the if-halves of definitions can often simulate reasoning with the only-if-halves, and vice versa can be found in [Kowalski91].

6 Abduction and Truth Maintenance

In this section we will consider the relationship between truth maintenance (TM) and abduction. TM systems have historically been presented from a procedural point of view. However, we will be concerned primarily with the semantics of TM systems and the relationship to the semantics of abductive logic programming.

A TM system is part of an overall reasoning system which consists of two components: a domain dependent problem solver which performs inferences and a domain independent TM system which records these inferences. Inferences are comunicated to the TM system by means of **justifications**, which in the simplest case can be written in the form

$$p \leftarrow p_1, \ldots, p_n$$

expressing that the proposition p can be derived from the propositions $p_1, \ldots, p_n$. Justifications include **premises**, in the case $n = 0$, representing propositions which hold in all contexts. Propositions can depend upon **assumptions** which vary from context to context.

TM systems can also record **nogoods**, which can be written in the form

$$\neg\,(p_1, \ldots, p_n),$$

meaning that the propositions $p_1, \ldots, p_n$ are incompatible and therefore cannot hold together.

Given a set of justifications and nogoods, the task of a TM system is to determine which propositions can be derived on the basis of the justifications, without violating the nogoods.

For any such TM system there is a straight-forward correspondence with abductive logic programs:

- justifications correspond to propositional Horn clause programs,
- nogoods correspond to propositional integrity constraints,
- assumptions correspond to abducible hypotheses, and
- contexts correspond to acceptable sets of hypotheses.

The semantics of a TM system can accordingly be understood in terms of the semantics of the corresponding propositional logic program with abducibles and integrity constraints.

The two most popular systems are the justification-based TM system (JTMS) of [Doyle79] and the assumption-based TM system (ATMS) of [deKleer86].

6.1 Justification-based truth maintenance

A justification in a JTMS can be written in the form

$$p \leftarrow p_1, \ldots, p_n, \sim p_{n+1}, \ldots, \sim p_m,$$

expressing that p can be derived (i.e. is IN in the current set of beliefs) if $p_1, \ldots, p_n$ can be derived (are IN) and $p_{n+1}, \ldots, p_m$ cannot be derived (are OUT).

For each proposition occurring in a set of justifications, the JTMS determines an IN or OUT label, taking care to avoid circular arguments and thus ensuring that each proposition which is labelled IN has well-founded support. The JTMS incrementally revises beliefs when a justification is added or deleted.

The JTMS uses nogoods to record contradictions discovered by the problem solver and to perform **dependency-directed backtracking** to change assumptions in order to restore consistency. In the JTMS changing an assumption is done by changing an OUT label to IN.

Suppose, for example, that we are given the justifications

$$p \leftarrow \sim q$$

$$q \leftarrow \sim r$$

corresponding to the propositional form of the Yale shooting problem. As [Morris88] observes, these correctly determine that q is labelled IN and that r and p are labelled OUT. If the JTMS is subsequently informed that p is true, then dependency-directed backtracking will install a justification for r, changing its label from OUT to IN. Notice that this is similar to the behaviour of the extended abductive proof procedure described in example 5.4, section 5.2.

Several authors have observed that the JTMS can be given a semantics corresponding to the semantics of logic programs, by interpreting justifications as propositional logic program clauses, and interpreting $\sim p_i$ as NAF of p_i. [Pimentel&Cuadrado89, Elkan90, Kakas&Mancarella90c, Giordano&Martelli90], in particular, show that a well-founded labelling for a JTMS corresponds to a stable model of the corresponding logic program. Several authors [Reinfrank&Dessler89, Fujiwara&Honiden89, Elkan90, Kakas&Mancarella90c] exploiting the interpretation of stable models as autoepistemic expansions [Gelfond&Lifschitz88], have shown a correspondence between well-founded labellings and stable expansions of the set of justifications viewed as autoepistemic theories.

The JTMS can also be understood in terms of abduction using the abductive approach to the semantics of NAF, as shown in [Kakas&Mancarella90c], [Giordano&Martelli90], [Dung91c]. This has the advantage that the nogoods of the JTMS can be interpreted as integrity constraints of the abductive framework. The correspondence between abduction and the JTMS is reinforced by [Satoh&Iwayama91], who give a proof procedure to compute generalised stable models using the JTMS (see section 5.5).

6.2 Assumption-based truth maintenance

Justifications in ATMS have the more restricted Horn clause form

$$p \leftarrow p_1, \ldots, p_n.$$

However, whereas the JTMS maintains only one implicit context of assumptions at a time, the ATMS explicitly records with every proposition the different sets of assumptions which provide the foundations for its belief. In ATMS assumptions are propositions that have been pre-specified as assumable. Each record of assumptions that supports a proposition p can also be expressed in Horn clause form

$$p \leftarrow a_1, \ldots, a_n$$

and can be computed from the justifications, as we illustrate in the following example.

Example 6.1
Suppose that the ATMS contains justifications

$$\begin{aligned} p &\leftarrow a, b \\ p &\leftarrow b, c, d \\ q &\leftarrow a, c \\ q &\leftarrow d, e \end{aligned}$$

and the single nogood

$$\neg(a, b, e)$$

where a, b, c, d, e are assumptions. Given the new justification

$$r \leftarrow p, q$$

the ATMS computes explicit records of r's dependence on the assumptions:

$$\begin{aligned} r &\leftarrow a, b, c \\ r &\leftarrow b, c, d, e. \end{aligned}$$

The dependence

$$r \leftarrow a, b, d, e.$$

is not recorded because its assumptions violate the nogood. The dependence

$$r \leftarrow a, b, c, d$$

is not recorded because it is subsumed by the dependence

$$r \leftarrow a, b, c.$$

[Reiter&deKleer87] show that, given a set of justifications, nogoods, and candidate assumptions, the ATMS can be understood as computing minimal and consistent abductive explanations in the propositional case (where assumptions are interpreted as abductive hypotheses). This abductive interpretation of ATMS has been developed further by [Inoue90], who gives an abductive proof procedure for the ATMS.

Given an abductive logic program P and goal G, the explicit construction in ALP of a set of hypotheses Δ, which together with P implies G and together with P satisfies any integrity constraints I, is similar to the record

$$G \leftarrow \Delta$$

computed by the ATMS. There are, however, some obvious differences. Whereas ATMS deals only with propositional justifications, relying on a separate problem solver to instantiate variables, ALP deals with general clauses, combining the functionalities of both a problem solver and a TM system. Ignoring the propositional nature of a TM system, ALP can be regarded as a hybrid of JTMS and ATMS, combining the non-monotonic negative assumptions of JTMS and the positive assumptions of ATMS, and allowing both positive and negative conditions in both justifications and nogoods [Kakas&Mancarella90c]. Other non-monotonic extensions of ATMS have been developed by [Junker89] and [Rodi&Pimentel91].

It should be noted that one difference between ATMS and ALP is the requirement in ATMS that only minimal sets of assumptions be recorded. This minimality of assumptions is essential for the computational efficiency of the ATMS. However, it is not essential for ALP, but can be imposed as an additional requirement when it is needed.

7 Conclusions and Future Work

In this paper we have surveyed a number of proposals for extending logic programming to perform abductive reasoning. We have seen that such extensions are closely linked with other extensions including NAF, integrity constraints, explicit negation, default reasoning, and belief revision.

Perhaps the most important link, from the perspective of logic programming, is that between abduction and NAF. On the one hand, we have seen that abduction generalises NAF, to include not only negative but also positive hypotheses, and to include general integrity constraints. On the other hand, we have seen that logic programs with abduction can be transformed into logic programs with NAF together with integrity constraints or explicit negation. The link between abduction and NAF includes both their semantics and their implementations.

We have argued that semantics can best be understood as providing a specification for an implementation. From this point of view, a semantics is a "declarative" specification, which might be non-constructive, but need not be concerned with meaning-theoretic notions such as "truth" and "falsity". Thus an overtly syntactic, but non-constructive, specification given in terms of maximally consistent extensions is just as much a "semantics" as one involving (covertly syntactic) stable models.

We have seen the importance of clarifying the semantics of abduction and of defining a semantics that helps to unify abduction, NAF, and default reasoning within a common framework. We have seen, in particular, that an implementation which is incorrect under one semantics (e.g. [Eshghi&Kowalski89]) can be correct under another (e.g. [Dung91a]).

Despite the recent advances in the semantics of NAF there is still room for improvement. One possibility is to explore further the direction set by [Kakas&Mancarella91d] and [Dung91a] which characterises the acceptability of a set of hypotheses Δ recursively in terms of the non-acceptability of all atacks against Δ. Another is to identify an appropriate concept of maximal consistency, perhaps along the lines of the retractability semantics suggested by [Kowalski&Sadri88]. The two possibilities need not be mutually exclusive. The former, recursive specification would be closer to an implementation than the latter. But the two specifications might otherwise be equivalent.

The use of abduction for NAF is a special case. It is necessary therefore to define a semantics that deals appropriately both with this case and with the other cases. In particular, we need to deal both with abductive hypotheses which need to be maximised for default reasoning and with other abductive hypotheses which need to be minimised. It is interesting that the abductive proof procedure can be regarded as both maximising and minimising the two kinds of abducibles. It maximises them in the sense that it (locally) makes as many abductive assumptions as are necessary to construct a proof. It minimises them in the sense that it makes no more assumptions than necessary. Perhaps this is another case where the implementation of abduction is more correct than the (semantic) specification.

It is an important feature of the abductive interpretation of NAF that it possesses an elegant and powerful proof procedure, which significantly extends SLDNF and which can be extended in turn to accommodate other abducibles and other integrity constraints. Future work on the semantics of ALP needs to preserve and develop further this existing close relationship between semantics and proof procedure.

The abductive proof procedure needs to be extended and improved in various ways. One such extension is the generation of non-ground hypotheses, containing variables. This problem, which has been studied in part by [Eshghi88], [Poole87] and [Chen&Warren89],

involves the treatment of the equality predicate as a further abducible. Because NAF is a special case of abduction, the problem of constructive negation in logic programming [Chan88, Barbuti et al.90] is a special case of constructive abduction.

We have argued that the implementation of abduction needs to be considered within a broader framework of implementing knowledge assimilation (KA). We have seen that abduction can be used to assist the process of KA and that abductive hypotheses themselves need to be assimilated. Moreover, the general process of checking for integrity in KA might be used to check the acceptability of abductive hypotheses.

It seems that an efficient implementation of KA can be based upon combining two processes: backward reasoning both to generate abductive hypotheses and to test whether the input is redundant and forward reasoning both to test input for consistency and to test whether existing information is redundant. Notice that the abductive proof procedure for ALP already has this feature of interleaving backward and forward reasoning. Such implementations of KA need to be integrated with improvements of the abductive proof procedure considered in isolation.

We have seen that the process of belief revision also needs to be considered within a KA context. In particular, it could be useful to investigate relationships between the belief revision frameworks of [Gärdenfors88, Doyle91, Nebel89, Nebel91] and various integrity constraint checking and restoration procedures.

The extension of logic programming to include integrity constraints is useful both for abductive logic programming and for deductive databases applications. We have seen, however, that for many applications the use of integrity constraints can be replaced by clauses with explicitly negated conclusions. Moreover, the use of explicit negation seems to have several advantages, including the ability to represent and derive negative information.

The relationship between integrity constraints and explicit negation needs to be investigated further: To what extent does this relationship, which holds for abduction and default reasoning, hold for other uses of integrity constraints, such as those concerning deductive databases; and what are the implications of this relationship on the semantics and implementation of integrity constraints?

Whatever the answers to these questions, it is clear that the combination of explicit negation and implicit NAF is very useful for knowledge representation in general. It is important, however, to obtain a deeper understanding of the relationships between these two forms of negation. It is clear, for example, that if $\sim p$ holds then $\neg p$ must be consistent. However, it is not the case that if $\neg p$ is consistent, then $\sim p$ holds, as in the following example

$$p \leftarrow \sim p$$
$$\neg p.$$

Thus, there is no simple relationship whereby one form of negation clearly subsumes the other.

Another problem, which we have already mentioned, is how to decide whether a negative condition should be understood as explicit negation or as NAF. One possibility might be simply to interpret the negation as NAF if the closed world assumption applies, and as explicit negation if the open world assumption applies. Moreover the presence of any rules in which the predicate of the condition occurs explicitly negated in a conclusion would suggest that the open world assumption applies and the negated condition therefore is explicit. Another, complementary possibility is to recognise that the open world assumption must apply to any predicate explicitly declared as abducible. Consequently, any negated condition whose predicate is abducible must be interpreted as explicit negation.

We have seen that explicit negation does not obey the laws of contraposition. This is further strong evidence that the semantics of clauses should be interpreted in terms of inference rules and not in terms of implications. Because of the similarity between default rules in Default Logic and clauses interpreted as inference rules in logic programming, this provides further evidence also for the possibility of developing a uniform semantics and implementation in which NAF, abduction, and default reasoning can be combined.

We have remarked upon the close links between the semantics of logic programming with abduction and the semantics of truth maintenance systems. The practical consequences of these links, both for building applications and for efficient implementations, need further investigation. What is the significance, for example, of the fact that TMSs and ATMSs correspond only to the propositional case of logic programs?

We have observed a duality between forward reasoning with only-if-halves of definitions and logic programming-style backward reasoning with if-halves. Could this duality apply also to a possible correspondence between inconsistency in truth maintenance systems and failures in logic programming?

More generally, are there other links to be discovered between extensions of logic programming and other uses of logic in AI? To what extent, for example, does logic programming need to be extended to include reasoning with disjunctive information? In particular, can disjunctions

$$p \vee q$$

be adequately represented by clauses of the form

$$p \leftarrow \sim q$$

$$q \leftarrow \sim p?$$

We believe that our survey supports the belief that abduction is an important and powerful extension of logic programming. It also points forward to the possibility that at some time in the future further extensions of logic programming might be fully adequate and appropriate for many, if not all, knowledge representation and reasoning tasks in AI.

Acknowledgements

This research was supported by Fujitsu Research Laboratories and by the Esprit Basic Research Action Compulog. The authors are grateful to Katsumi Inoue and Ken Satoh for helpful comments on an earlier draft.

References

[Allemand et al.87] Allemand, D., Tanner, M., Bylander, T., Josephson, J., *On the computational complexity of hypothesis assembly.* IJCAI87, Milan, Italy (1987)

[Apt&Bezen90] Apt, K.R., Bezem, M., *Acyclic programs.* Proc. 7th ICLP, MIT Press (1990) 579–597

[Baral&Subrahmanian90] Baral, C.R., Subrahmanian, V.S., *Stable and extension class theory for logic programs and default logics.* Proc. 3rd International Workshop on Non-Monotonic Reasoning, Lake Tahoe, California (1990)

[Barbuti et al.90] Barbuti, R., Mancarella, P., Pedreschi, D., Turini, F., *A transformational approach to negation in logic programming.* Journal of Logic Programming, Vol.8 (1990) 201-228

[Bidoit&Froixdevaux88] Bidoit, N., Froixdevaux, Ch., *Negation by default and non stratifiable logic programs.* Internal report 437 (1988)

[Brewka89] *Preferred subtheories: an extended logical framework for default reasoning.* Proc. IJCAI89 (1989) 1043–1048

[Brogi et al.90] Brogi, A., Mancarella, P., Pedreschi, D., Turini, F., *Composition operators for logic theories.* Proc. Symp. on Computational Logic, LCNS, Springer Verlag (1990)

[Bry90] Bry, F., *Intensional updates: abduction via deduction.* Proc. 7th ICLP90, MIT Press (1990) 561–575

[Chan88] Chan, D., *Constructive negation based on the completed database.* Proc. 5th ICLP (R.A. Kowalski and K. Bowen eds.) MIT Press, Cambridge, Mass. (1988) 111–125

[Charniak&McDermott85] Charniak, E., McDermott, D., *Introduction to artificial intelligence.* Addison-Wesley (1985)

[Chen&Warren89] Chen, W., Warren, D.S., *Abductive Logic Programming.* Research Report, Dept. of Comp. Science, State Univ. of New York at Stony Brook (1989)

[Clark78] Clark,K.L., *Negation as failure.* Logic and Data Bases, Gallaire,H. & Minker,J. eds. (1978) 293–322

[Console et al.89] Console, L., Dupré, D., Torasso, P. *A Theory for diagnosis for incomplete causal models.* Proc. 11th IJCAI89 (1989) 1311

[Console et al.91] Console, L., Dupré, D., Torasso, P. *On the relationship between abduction and deduction.* Journal of Logic and Computation, 2(5) (1991)

[Cox&Pietrzykowski86] Cox, P. T., Pietrzykowski, T., *Causes for events: their computation and applications.* Proc. CADE86 (1986) 608–621

[Decker86] Decker, H., *Integrity enforcement on deductive databases.* Proc. EDS86, Charleston, SC (1986) 271–285

[Doyle79] Doyle, J., *A truth maintenance system.* Artif. Intell. 12 (1979) 231–272

[Doyle91] Doyle, J., *Rational belief revision.* Proc. 2nd International Conference on Principles of Knowledge Representation and Reasoning, Cambridge, Massachusetts (1991) 163

[Dung91a] Dung, P.M., *Negation as hypothesis: an abductive foundation for logic programming* Proc. ICLP91, MIT Press (1991)

[Dung91b] Dung, P.M., Ruamviboonsuk, P., *Well-founded reasoning with classical negation.* Proc. 1st International Workshop on Logic Programming and Non-monotonic Reasoning, Nerode, Marek and Subrahmanian eds., Washington DC (1991) 120

[Dung91c] Dung, P.M., *An abductive foundation for non-monotonic truth maintenance.* Proc. 1st World Conference on Fundamentals of AI, Paris, M.de Glas ed. (1991)

[Elkan90] Elkan *A rational reconstruction of non-monotonic truth maintenance systems.* Artif. Intell. 43, (1990) 219–234

[Eshghi88] Eshghi, K., *Abductive planning with event calculus.* Proc. 5th ICLP88, MIT Press (1988) 562

[Eshghi90] Eshghi, K., *Diagnoses as stable models.* Proc. 1st International Workshop on Principles of Diagnosis, Menlo Park (1990)

[Eshghi&Kowalski88] Eshghi, K., Kowalski, R.A., *Abduction through deduction.* Technical Report, Department of Computing, Imperial College, London (1988)

[Eshghi&Kowalski89] Eshghi, K., Kowalski, R.A., *Abduction compared with negation by failure.* Proc. 6th ICLP89, MIT Press (1989) 234–255

[Evans89] Evans, C.A., *Negation as failure as an approach to the Hanks and McDermott problem.* Proc. second Int. Symp. on AI, Monterrey, Mexico (1989)

[Evans&Kakas91a] Evans, C.A., Kakas, A.C., *Hypothetico-deductive reasoning.* to appear in FGCS-92 (1992)

[Kakas91b] Kakas, A.C., *On the evolution of databases.* Technical Report, Logic Programming Group, Imperial College, London (1991)

[Finger&Genesereth85] Finger, J.J., Genesereth, M.R., *RESIDUE: a deductive approach to design synthesis.* Report no. CS-85-1035, Stanford University (1985)

[Fujiwara&Honiden89] Fujiwara, Y., Honiden, S., *Relating the TMS to Autoepistemic Logic.* Proc. IJCAI89 (1989) 1199–1205

[Gabbay&Kempson91] Gabbay, D.M., Kempson, R.M., *Labelled abduction and relevance reasoning.* Workshop on Non-Standard Queries and Non-Standard Answers, Toulose, France (1991)

[Gallaire&Nicolas78] Gallaire, H., Nicolas, J.M., *Data base: theory vs. interpretation.* Gallaire and Minker (eds.) Logic and Data Bases, Plenum Press, New York, (1978) 33–54

[Gärdenfors88] Gärdenfors, P., *Knowledge in flux: modeling the dynamics of epistemic states.* MIT Press, Cambridge, MA, (1988) [Geffner90] Geffner, H., *Casual theories for non-monotonic reasoning*. Proc. 8th AAAI90 (1990) 524

[Gelfond&Lifschitz88] Gelfond, M., Lifschitz, V., *The Stable model semantics for logic programs*. Proc. fifth Int. Conf. and Symp. on LP, MIT Press (1988) 1070–1080

[Gelfond&Lifschitz90] Gelfond, M., Lifschitz, V., *Logic programs with classical negation*. Proc. seventh Int. Conf. and Symp. on LP, MIT Press (1990) 579–597

[Goebel et al.86] Goebel, R., Furukawa, K., Poole, D., *Using definite clauses and integrity constraints as the basis for a theory formation approach to diagnostic reasoning*. Proc. ICLP86 211–222

[Giordano&Martelli90] Giordano, L., Martelli, A., *Generalized stable model semantics, truth maintenance and conflict resolution*. Proc. ICLP90 (1990) 427–411

[Hanks&McDermott86] Hanks, S., McDermott, D., *Default reasoning, non-monotonic logics, and the frame problem*. Proc. Am. Assoc. Artif. Intell. Natl. Conf. Philadelphia (1986) 328–333

[Hanks&McDermott87] Hanks, S., McDermott, D., *Default reasoning, non-monotonic logics, and the frame problem*. AI Journal 35 (1987)

[Hobbs et al.91] Hobbs, J.R., Stickel, M., Appelt, D., Martin, P., *Interpretation as abduction*. Technical Report 499 Artificial Intelligence Center, Computing and Engineering Sciences Division (1990)

[Hobbs90] Hobbs, J.R., *An integrated abductive framework for discourse interpretation*. Proc. AAAI Symposium on Automated Abduction, Stanford (1990) 10

[Junker89] Junker, U., *A correct non-monotonic ATMS*. Proc. 11th IJCAI, Detroit (1989) 1049–1054

[Inoue90] Inoue, K., *An abductive procedure for the CMS/ATMS*. Proc. ECAI-90 International Workshop on Truth Maintenance, Stockholm, Sweden, Martins(ed.), Springer Verlag (1991)

[Inoue91a] Inoue, K., *Hypothetical reasoning in logic programs*. ICOT Research Center report (1991)

[Inoue91b] Inoue, K., *Extended logic programs with default assumptions*. Proc. ICLP91, Paris (1991) 490

[Jaffar&Lassez87] Jaffar, J., Lassez, J-L, *Constraint Logic Programming*. POPL87, Munich Germany (1987)

[Kakas91] Kakas, A. C., *Deductive databases as theories of belief*. Technical report, Logic Programming Group, Imperial College, London (1991)

[Kakas&Mancarella89] Kakas, A. C., Mancarella, P., *Anomalous models and abduction*. Proc. 2nd Int. Symp. on Artif. Int., Monterrey, Mexico (1989)

[Kakas&Mancarella90a] Kakas, A. C., Mancarella, P., *Generalized Stable Models: a Semantics for Abduction*. Proc. 9th European Conference on Artificial Intelligence, ECAI90, Stockolm (1990) 385–391

[Kakas&Mancarella90b] Kakas, A. C., Mancarella, P., *Database updates through abduction.* Proc. 16th International Conference on Very Large Databases, VLBB90, Brisbane, Australia (1990)

[Kakas&Mancarella90c] Kakas, A. C., Mancarella, P., *On the relation of truth maintenance and abduction.* Proc. of the 1st Pacific Rim International Conference on AI, PRICAI90, Nagoya, Japan (1990)

[Kakas&Mancarella90d] Kakas, A. C., Mancarella, P., *Abductive logic programming.* Proc. of NACLP90 Workshop on Non-Monotonic Reasoning and Logic Programming, Austin, Texas (1990)

[Kakas&Mancarella91a] Kakas, A. C., Mancarella, P., *Knowledge assimilation and abduction.* Proc. ECAI90 International Workshop on Truth Maintenance, Stockholm, Sweden, Martins(ed.), Springer Verlag (1991)

[Kakas&Mancarella91b] Kakas, A. C., Mancarella, P., *Preferred extensions are partial stable models.* to appear in Journal of Logic Programming

[Kakas&Mancarella91c] Kakas, A. C., Mancarella, P., *Negation as stable hypotheses.* Proc. 1st International Workshop on Logic Programming and Non-Monotonic Reasoning, Nerode, Marek and Subrahmanian eds., Washington DC (1991) 275

[Kakas&Mancarella91d] Kakas, A. C., Mancarella, P., *Stable theories for logic programs.* Proc. ISLP91, San Diego (1991)

[deKleer86] deKleer, J., *An assumption-based TMS.* Artif. Intell. Journal 32 (1986)

[Kowalski79] Kowalski, R.A., *Logic for problem solving.* New York: Elsevier (1979)

[Kowalski87] Kowalski, R.A., *Belief revision without constraints.* Computational Intelligence, Volume 3, Number 3, N. Cercone, G. McCalla eds (1987) 194

[Kowalski90] Kowalski, R.A., *Problem and promises of computational logic.* Proc. Symposium on Computational Logic, Lloyd ed., Springer Verlag (1990)

[Kowalski91] Kowalski, R.A., *Logic programming in artificial intelligence.* Proc. IJCAI91, Sidney (1991)

[Kowalski&Sadri87] Kowalski, R.A., Sadri, F., *An application of general purpose theorem-proving to database integrity.* J. Minker ed., Foundations of Deductive Databases and Logic Programming, Morgan Kaufmann Publishers, Palo Alto (1987)

[Kowalski&Sadri88] Kowalski, R.A., Sadri, F., *Knowledge representation without integrity constraints.* Technical Report, Department of Computing, Imperial College, London (1988)

[Kowalski&Sadri90] Kowalski, R.A., Sadri, F., *Logic programs with exception.* Proc. 7th ICLP90, MIT Press (1990) 598–613

[Kowalski&Sergot86] Kowalski, R.A., Sergot, M., *A logic-based calculus of events.* New Generation Computing, vol.4 (1986) 267

[Kunifiji et al.86] Kunifuji, S., Tsurumaki, K., Furukawa, K., *Consideration of a hypothesis-based reasoning system.* Journal of Japanese Society for Artificial Intelligence, Vol. 1, No. 2 (1986) 228–237

[Lever91] Lever, J. M., *Combining induction with resolution in logic programming.* PhD Thesis, Department of Computing, Imperial College, London (1991)

[Levesque89] Levesque, H.J., *A knowledge-level account of abduction.* Proc. 11th International Joint Conference on AI (1989) 1061

[Lloyd87] Lloyd, J. W., *Foundations of Logic Programming.* second edition, Springer Verlag (1987)

[Lloyd&Topor85] Lloyd, J. W., Topor, R.W., *A basis for deductive database system.* J. Logic Programming 2 (1985) 93–109

[Marek&Truszczynski89] Marek, W., Truszczynski, M., *Stable semantics for logic programs and default theories.* Proc. NACLP89, MIT Press (1989)

[Minker82] Minker, J., *On indefinite databases and the closed world assumption.* Proc. of the 6th Conference on automated Deduction (New York) Springer-Verlag Lecture Notes in Computer Science, No 138 (1982) 292-308

[Miyaki et al.84] Miyaki, T., Kunifuji, S., Kitakami, H., Furukawa, K., Takeuchi, A., Yokota, H., *A knowledge assimilation method for logic databases.* International Symposium on Logic Programming, Atlantic City, NJ. (1984) 118–125

[Morris88] Morris, P. H., *The anomalous extension problem in default reasoning.* AI Journal 35 (1988) 383–399

[Nebel89] Nebel, B., *A knowledge level analysis of belief revision.* Proc. 1st International Conference on Principles of Knowledge Representation and Reasoning, Brachman, Levesque and Reiter eds., San Meteo, CA. Morgan Kaufmann (1989) 301–311

[Nebel91] Nebel, B., *Belief revision and default reasoning: syntax-based approaches.* Proc. 2nd International Conference on Principles of Knowledge Representation and Reasoning, Allen, Fikes and Sandwell eds., Cambridge, Massachusetts (1991) 417

[Pearl87] Pearl, J., *Embracing causality in formal reasoning.* Proc. AAAI 87, Seattle, Washington (1987) 360–373

[Peirce31] Peirce, C.S., *Collected papers of Charles Sanders Peirce.* Vol.2, 1931–1958, Hartshorn et al. eds., Harvard University Press

[Pereira et al.91a] Pereira, L.M., Aparicio, J.N., Alferes, J.J., *Non-monotonic reasoning with well-founded semantics.* Proc. ICLP91, MIT Press (1991) 475

[Pereira et al.91b] Pereira, L.M., Aparicio, J.N., Alferes, J.J., *Contradiction removal within well-founded semantics.* Proc. 1st International Workshop on Logic Programming and Non-monotonic Reasoning, Nerode, Marek and Subrahmanian eds., Washington DC (1991) 105

[Pereira et al.91c] Pereira, L.M., Aparicio, J.N., Alferes, J.J., *Derivation procedures for extended stable models.* Proc. IJCAI91 (1991) 863–868

[Pereira et al.91d] Pereira, L.M., Aparicio, J.N., Alferes, J.J., *Counterfactual reasoning based on revising assumptions.* Proc. ISLP91, San Diego (1991)

[Pimentel&Cuadrado89] Pimentel, S. G., Cuadrado, J. L., *A truth maintenance system based on stable models.* Proc. NACLP89, MIT Press (1989)

[Pople73] Pople, H. E. Jr., *On the mechanization of abductive logic.* Proc. 3rd IJCAI (1973) 147–152

[Poole87] Poole, D., *Variables in hypotheses.* Proc. IJCAI87 (1987) 905–908

[Poole88a] Poole, D., *A logical framework for default reasoning.* Artif. Intell. vol.36 (1988) 27–47

[Poole88b] Poole, D., *Representing knowledge for logic-based diagnosis.* Proc. of the Int. Conf. on Fifth Generation Computer System (1988) 1282–1290

[Poole et al.87] Poole, D., Goebel, R.G., Aleliunas, *Theorist: a logical reasoning system for default and diagnosis.* N. Cercone and G. McCalla eds. The Knowledge Fronteer: Essays in the Representation of Knowledge, Springer Verlag, (1987) 331–352

[Preist&Eshghi92] Preist, C., Eshghi, K., *Consistency-based and abductive diagnoses as generalised stable models.* Proc. FGCS (1992)

[Przymusinski89] Przymusinski, T.C., *On the declarative and procedural semantics of logic programs.* Journal of Automated Reasoning, 5 (1989) 167–205

[Przymusinski90] Przymusinski, T.C., *Extended stable semantics for normal and disjunctive programs.* Proc. ICLP90 (1990) 459–477

[Reggia83] Reggia, J., *Diagnostic experts systems based on a set-covering model.* International Journal of Man Machine Studies, 19(5) (1983) 437–460

[Reinfrank&Dessler89] Reinfrank, M., Dessler, O., *On the relation between truth maintenance and non-monotonic logics.* Proc. IJCAI89, Detroit, MI (1989) 1206–1212

[Reiter78] Reiter, R., *On closed world data bases.* Gallaire and Minker eds., (1978) 55–76

[Reiter80] Reiter, R., *A Logic for default reasoning.* Artif. Intell. vol.13 (1,2) (1980) 81–132

[Reiter87] Reiter, R., *A theory of diagnosis from first principle.* Artif. Intell. Journal vol.32 (1987)

[Reiter88] Reiter, R., *On integrity constraints.* Proc. 2nd Conference on Theoretical Aspects of Reasoning about Knowledge, Moshe Y. Vardi ed., Pacific Grove, California (1988) 97

[Reiter90] Reiter, R., *On asking what a database knows.* Proc. Symposium on Computational Logic, Lloyd ed., Springer Verlag (1990)

[Reiter&deKleer87] Reiter, R., deKleer, J., *Foundations of assumption-based truth maintenance systems: preliminary report.* Proc. AAAI87, Seattle (1987) 183–188

[Rodi&Pimentel91] Rodi, W.L., Pimentel, S.G., *A non-monotonic ATMS using stable bases.* Proc. 2nd International Conference on Principles of Knowledge Representation and Reasoning, Allen, Fikes and Sandwell eds., Cambridge, Massachusetts (1991) 485

[Saccà&Zaniolo90] Saccà, D., Zaniolo, C., *Stable models and non determinism for logic programs with negation* Proc. ACM SIGMOD-SIGACT Symp. on Principles of Database Systems (1990) 205–217

[Satoh&Iwayama91] Iwayama, N., Satoh, K., *Computing abduction using the TMS.* Proc. ICLP91, Paris (1991)

[Satoh&Iwayama92] Iwayama, N., Satoh, K., *A correct top-down proof procedure for a general logic program with integrity constraints.* ICOT Technical Report (1992)

[Sattar&Goebel89] Sattar, A., Goebel, R., *Using crucial literals to select better theories.* Technical Report, Dept. of Computer Science, University of Alberta, Canada (1989)

[Selman&Levesque90] Selman, B., Levesque, H.J., *Abductive and default reasoning: a computational core.* Proc. AAAI90 (1990) 343–348

[Sergot83] Sergot,M., *A query-the-user facility for logic programming.* Integrated Interactive Computer Sytem (eds. Degano and Sandwell) North Holland Press (1983) 27–41

[Shanahan89] Shanahan, M., *Prediction is deduction but explanation is abduction.* Proc. IJCAI89 (1989) 1055

[Stickel89] *A prolog-like inference system for computing minimum-cost abductive explanations in natural-language interpretation.* Proc. Honk Kong Int. Computer Conf. (1989)

[VanGelder et al.88] Van Gelder, A., Ross, K.A., Schlipf, J.S., *Unfounded sets and the well-founded semantics for general logic programs.* Proc. ACM SIGMOD-SIGACT, Symp. on Principles of Database Systems (1988)

Explanation in the Situation Calculus

Murray Shanahan

Imperial College
Department of Computing,
180 Queen's Gate,
London SW7 2BZ.
England.

Tel: +44 71 589 5111 x 5076
Email: mps@doc.ic.ac.uk

April 1992
DRAFT

Abstract

Explanation, that is reasoning from effects to causes, is a form of reasoning fundamental to Artificial Intelligence. The situation calculus is the oldest and best known formalism for representing change in Artificial Intelligence. Using the so-called stolen car problem as an example, this paper explores different techniques for explanation within the framework of the situation calculus. It discusses two styles of representation for explanation problems: the standard style found in the existing literature, and an alternative style introduced here. Two approaches to explanation are compared: the deductive approach usually found in the literature, and a less common abductive approach.

Introduction

Much attention has been given to the problem of formalising prediction, that is reasoning forwards in time from causes to effects, and in particular to the problem of formalising default persistence to overcome the frame problem [McCarthy, 1986], [Lifschitz, 1986], [Kautz, 1986], [Haugh, 1987], [Lifschitz, 1987], [Shoham, 1988], [Baker, 1989]. Some attention has also been given to the converse problem of formalising temporal explanation (or postdiction), that is reasoning backwards in time from effects to causes [Morgenstern and Stein, 1988], [Lifschitz and Rabinov, 1989], [Shanahan, 1989], [Baker, 1989]. Temporal explanation is certainly as important as prediction, as it underlies planning and diagnosis, as well as being a fundamental mode of reasoning in its own right, so a thorough understanding of its nature is basic to Artificial Intelligence.

The situation calculus [McCarthy and Hayes, 1969] is the oldest and best-understood logic-based formalism for representing change in AI. The situation calculus results from choosing one of the simplest ontologies possible for such a formalism: one which includes situations, actions and fluents. A situation is an instantaneous snapshot of the world, and a fluent is anything whose value is subject to change. Despite the simplicity of this ontology, the situation calculus has considerable expressive power [Gelfond *et al.*, 1991]. This paper is a study of explanation within the framework of the situation calculus.[1] Two styles of representation for explanation problems are compared: the standard style used in the existing literature, and an alternative style. And two fundamentally different approaches to explanation are explored: the deductive approach and the abductive approach. The paper presents the standard and alternative styles of representation first, then looks at the deductive approach, using both styles, and finally investigates the abductive approach.

Most attempts to formalise temporal explanation have adopted the deductive approach [Morgenstern and Stein, 1988], [Lifschitz and Rabinov, 1989], [Baker, 1989], [Baker, 1991]. Suppose we have a formula T which captures the timeless laws of change in a given domain, and a formula H representing when certain time-varying facts are true. According to the deductive approach, the explanation of an additional such fact F will be among the logical consequences of $T \wedge H \wedge F$. According to the abductive approach [Shanahan, 1989], an explanation is a formula Δ such that $T \wedge H \wedge \Delta$ has F among its logical consequences.

Throughout this paper, I will use the so-called stolen car problem (SCP) as a benchmark [Kautz, 1986]. Suppose I park my car in the morning and go to work. At lunch time, I might reasonably apply default persistence and infer that the car is still where I left it. However, when I return to the car park in the evening I find that it has gone. Its disappearance requires an explanation. That is to say, we want to reason backwards in time to the (possible) causes of the car's disappearance. In this case, the only reasonable explanation for the car's disappearance is that it was stolen some time between morning and evening. So my previous conclusion that the car was still there at lunch time is open to question. The car may have been stolen any time after I parked it and before I observed that it was gone, so I cannot say anything about its whereabouts at lunch time.

1. Representing Explanation Problems in Situation Calculus

Several authors have attempted to deal with temporal explanation within the framework of the situation calculus [Lifschitz and Rabinov, 1989], [Baker, 1989], [Baker, 1991]. The ontology of the situation calculus includes situations, actions and fluents. I will employ

[1] Debates about the limitations of the situation calculus are outside the scope of this paper. However, the issues discussed are also relevant to other temporal reasoning formalisms, such as the event calculus [Shanahan, 1989].

variables of three sorts corresponding to this ontology.[2] We write Result(a,s) to denote the situation which results when action a is performed in situation s, and we write Holds(f,s) to represent that fluent f holds in situation s. If a fluent holds in a situation then we say that it has the value true, and if it does not hold, we say it has the value false.

To represent a particular domain using the situation calculus, we write two sets of sentences, one set describing which fluents change value as a result of performing each action (so-called axioms of motion), and one set describing which retain their value (so-called frame axioms). The main concern of a great deal of research on the formal representation of change has been the frame problem, or how to eliminate the need to write explicit frame axioms. One of the most successful attempts to overcome the frame problem is Baker's [1989], [1991].[3] His solution does not suffer from the difficulties pointed out by Hanks and McDermott [1987] and correctly handles ramifications (derived properties). It can also cope with certain explanation problems. In particular, Baker [1989] represents the stolen car scenario as follows,

$$\neg\text{Holds}(\text{Stolen},S0) \tag{SR1}$$

$$S2 = \text{Result}(\text{Wait},\text{Result}(\text{Wait},S0)) \tag{SR2}$$

$$\text{Holds}(\text{Stolen},S2) \tag{SR3}$$

Before discussing the merits of any particular approach to explanation, I would like to question whether this is a good representation of the SCP in the first place. The meaning of Result(Wait,Result(Wait,S0)) is the situation which results when two successive Wait actions are performed in situation S0. The assertion that S2 equals this situation means that the only two actions which occur between S0 and S2 are the two Wait actions. It is implicit in this assertion that nothing else happens between S0 and S2. However, the whole point of the SCP is that we do not know what actions take place between S0 and S2. We don't know what S2 equals in terms of the Result function. Since the intended meaning of Wait is an action which has no effect, then it doesn't seem likely that S2 equals Result(Wait,Result(Wait,S0)).[4] However, since it is only by default that waiting has no effect, it is still possible to conclude that one of the wait actions is responsible for the car's disappearance.

Rather than half-heartedly asserting that nothing happens between S0 and S2 and allowing default reasoning to overide this assertion to conclude that wait actions sometimes have strange effects, a more correct representation of the SCP asserts nothing about S2 beyond the fact that it is the result of a sequence of actions which starts in situation S0. Then the aim of explanation is to characterise S2 in terms of the result function, that is to characterise the sequence of actions which starts in S0 and leads to a situation S2 in which the car is gone.

Accordingly, I suggest the following representation of the SCP,

$$\text{Holds}(\text{Car-parked},S0) \tag{AR1}$$

$$\neg\text{Holds}(\text{Car-parked},S2) \tag{AR2}$$

$$\text{Follows}(S2,S0) \tag{AR3}$$

where Follows is defined thus,

---

[2] Variables begin with lower-case letters. Predicate and function symbols begin with upper-case letters. All variables are universally quantified unless otherwise indicated.

[3] To follow closely the argument of this paper, the reader will require some knowledge of Baker's approach to the frame problem.

[4] In fact, the very idea of a "wait" action seems rather strange, and the idea of a sequence of two wait actions seems stranger still. Surely waiting is a pause between actions rather than an action in its own right. A "sneeze" action might be an appropriate substitute, where sneezing is assumed to have no appreciable effect.

$$\text{Follows}(sc,sa) \leftrightarrow sc=sa \vee \exists a,sb\,[sc=\text{Result}(a,sb) \wedge \text{Follows}(sb,sa)] \qquad \text{(AR4)}$$

and where we have the following axiom of motion,

$$\neg\text{Holds}(\text{Car-parked},\text{Result}(\text{Steal},s)) \qquad \text{(AR5)}$$

The point being made here applies to explanation using the situation calculus in general, and is not restricted to the SCP. Lifschitz and Rabinov [1989] use the same style as Baker to represent a bloodless variation of the Yale shooting problem [Hanks and McDermott, 1987]. In the usual Yale shooting problem, we are asked to consider whether the fluent Alive holds as a result of loading, waiting and shooting, and are expected to conclude that it does not. Lifschitz and Rabinov use the standard representation of the shooting problem, but then add an assertion which is equivalent to,

$$\text{Holds}(\text{Alive},\text{Result}(\text{Shoot},\text{Result}(\text{Wait},\text{Result}(\text{Load},S0))))$$

Their approach to explanation introduces the idea of a *miracle*, which is an unexpected effect of an action. Once again, in their approach default reasoning is expected to overide the above "half-hearted" assertion that nothing happens between loading and shooting to conclude that in fact the Wait action unloads the gun. As before, I suggest that the task of explanation is to determine exactly what sequence of actions takes place between loading and shooting. So we might represent the bloodless Yale shooting problem like this.

$$S1=\text{Result}(\text{Load},S0)$$

$$S3=\text{Result}(\text{Shoot},S2)$$

$$\text{Follows}(S2,S1)$$

$$\text{Holds}(\text{Alive},S3)$$

where Follows is defined as before, and we have the following axiom of motion,

$$\neg\text{Holds}(\text{Loaded},\text{Result}(\text{Unload},s))$$

The question then is what S2 equals in terms of the Result function. In what follows, the style of representation exemplified by [Baker, 1989], [Baker, 1991] and [Lifschitz and Rabinov, 1989] will be called the *standard* style, and that suggested here will be called the *alternative* style. I will now examine both styles of representation in the context of the deductive approach to explanation, and later will examine both styles in the context of the abductive approach.

2. The Deductive Approach in the Standard Style

Underlying the deductive approach to explanation championed by Morgenstern and Stein [1988], Baker [1989], [1991], and Lifschitz and Rabinov [1989] is a deductive approach to the assimilation of knowledge. Let us suppose that we have a formula T which represents an agent's knowledge about the world. Then, if the agent learns that F is the case, where F is not a consequence of T, the deductive approach to assimilating F is simply to add it to T. The formula T ∧ F then represents the agent's knowledge about the world.

Using this approach, how is the SCP tackled within the framework of the situation calculus? Let's consider the standard style of representation first. In addition to (SR1) to (SR3), we need a frame axiom. A common frame axiom is,

$$[\text{Holds}(f,s) \leftrightarrow \text{Holds}(f,\text{Result}(a,s))] \leftarrow \neg\text{Ab}(a,f,s) \qquad (1)$$

The frame problem is normally overcome by minimising the extension of Ab in some way, using circumscription for example. In Baker's work [1989], [1991], this is achieved by introducing an "existence-of-situations" axiom, then circumscribing, minimising Ab and allowing the Result function to vary. This avoids the problem Hanks and McDermott encountered with McCarthy's formulation [McCarthy, 1986], [Hanks and McDermott, 1987]. However, since the SCP doesn't involve actions with preconditions, it doesn't run into the Hanks-McDermott problem, and McCarthy's formulation, which minimises Ab and allows Holds to vary, is adequate.

Initially, we know just (SR1) and (SR2). With Wait the only action in the domain of discourse, nothing is abnormal, so minimising Ab using either McCarthy's or Baker's technique yields simply,

$$\neg\mathrm{Ab}(a,f,s)$$

from which we can conclude,

$$\neg\mathrm{Holds}(\mathrm{Stolen},S2)$$

Using the deductive approach to explanation, when we learn (SR3) we simply add it to (SR1), (SR2) and (1), and derive a new set of conclusions. From (SR1) to (SR3) and (1), Baker [1989] gets,

$$\mathrm{Ab}(\mathrm{Wait},\mathrm{Stolen},S0) \vee \mathrm{Ab}(\mathrm{Wait},\mathrm{Stolen},S1)$$

This seems to be the consequence we intuitively expect, using the standard style of representation: the car is either stolen during the first Wait action or during the second, and we cannot say for sure which of these disjuncts is true. Minimising Ab simply reduces the set of models to those in which one of the disjuncts is true, the other one false, and Ab is false for everything else. However, this consequence doesn't really constitute an explanation at all. It simply says that one of the Wait actions must have been abnormal. From (1), it can be seen that the abnormality of a Wait action is not sufficient to bring about a change in the value of Stolen. It is a necessary condition of such a change, not a sufficient one.

Furthermore, if the domain is widened a little, other difficulties arise. Suppose the domain includes actions with preconditions, thus necessitating a form of minimisation different to McCarthy's. The best-known candidates at present are chronological minimisation [Shoham, 1988], [Kautz, 1986], [Lifschitz, 1986], causal minimisation [Haugh, 1986], [Lifschitz, 1987] and Baker's state-based minimisation [1989], [1991]. As Baker points out [1989], chronological minimisation, which postpones change until as late as possible, will insist that the car is stolen during the second Wait action; causal minimisation can be modified to cope with explanation [Lifschitz and Rabinov, 1989], but has problems with ramifications (derived properties); and his own approach, whilst adequate for the simple version of the problem presented above, falls apart as soon as another fluent is introduced which holds in S0.

Why should the need to tackle explanation problems interfere with our efforts to overcome the frame problem? In a later section, I will discuss the abductive approach to explanation, which doesn't interfere with minimisation in any way, but first I will examine the deductive approach applied to the alternative style of representation suggested in Section 1.

3. The Deductive Approach in the Alternative Style

What happens when the deductive approach to explanation is used with the alternative style of representation? From (AR1) to (AR4) and (1), we have,

$$\exists a,sa,sb\ [\mathrm{Ab}(a,\mathrm{Car\text{-}parked},sa) \wedge sb=\mathrm{Result}(a,sa) \wedge \\ \mathrm{Follows}(sa,S0) \wedge \mathrm{Follows}(S2,sb)]$$

From (AR5) and (1), minimising Ab using either McCarthy's or Baker's approach, we have,

$$Ab(a,f,s) \leftrightarrow a=Steal \wedge f=Car\text{-}parked \wedge Holds(Car\text{-}parked,s)$$

and therefore,

$$\exists sa,sb\ [sb=Result(Steal,sa) \wedge Follows(sa,S0) \wedge Follows(S2,sb)]$$

In other words, there is a Steal action between situations S0 and S2, which is the intuitively correct explanation. To simplify sentences of the above form, I introduce a new predicate. The formula Between(a,s1,s2) represents that an action a occurs between situations s1 and s2, and is defined as follows.

$$\begin{aligned}Between(a,sa,sd) \leftrightarrow \exists sb,sc\ [sc=Result(a,sb) \wedge \\ Follows(sb,sa) \wedge Follows(sd,sc)]\end{aligned} \qquad (AR6)$$

Then, the above explanation of the car's disappearance can be abbreviated to,

$$Between(Steal,S0,S2)$$

So the deductive approach to the SCP seems to work using the alternative representation. Unlike the standard representation, the alternative representation doesn't encounter difficulties with explanation problems in richer domains. Suppose that we employ Baker's approach to minimisation — the Result function is allowed to vary, and there is an axiom asserting, for all possible combinations of fluents, the existence of a situation in which that combination holds. The problem that Baker reports [1989] using the standard representation is that the assertion that the car is not in the car park in S2 forces a new abnormality. There is a variety of choices for this abnormality, each of which satisfies Axiom (1) whilst allowing the car to disappear. Unfortunately, in a domain of any complexity, some of them are both counter-intuitive and minimal.

With the alternative representation, using Baker's approach to minimisation, this problem simply doesn't arise. The assertion that the car is not in the car park in S2 does not force a new abnormality. Rather, it forces a Steal action to occur between S0 and S2, and Steal actions are abnormal with respect to Car-parked anyway. So the minimisation of Ab is unaffected.

However, the approach described here is not complete without further minimisation to eliminate the possibility of other disruptive actions taking place between S0 and S2. Suppose the domain is expanded to include the fluent Guarded, which means that a security guard is on duty in the car park, and the two actions Start-tea, which represents that the guard goes for a tea break, and End-tea, which represents that he returns from a tea break. Initially the car park is guarded, but while the guard is at tea, it is unguarded.

$$Holds(Guarded,S0) \qquad (AR7)$$

$$\neg Holds(Guarded,Result(Start\text{-}tea,s)) \qquad (AR8)$$

$$Holds(Guarded,Result(End\text{-}tea,s)) \qquad (AR9)$$

From (AR1) to (AR9), and knowing nothing about the guard's tea breaks, we would like to conclude by default that the car park is still guarded in S2. Unfortunately, (AR3) is too weak to allow this conclusion. It simply says that there is some sequence of actions between S0 and S2, and does not disallow the possibility that a Start-tea action occurs without a corresponding End-tea, and that Guarded doesn't hold in S2.

The alternative style of representation for explanation problems presupposes a framework which can cope with sequences of actions about which not everything is known. In the SCP, for example, we don't know what actions have taken place between S0 and S2. However, we would like to assume by default that nothing happens we don't know about, thus permitting the conclusion that the guard is still on duty in S2. Although the issue is tangential to the main topic of this paper, I will outline a way to achieve this.

Naively, it seems that what we would like to do is minimise Between in parallel with Ab. However, in the presence of an existence-of-situations axiom, this minimisation would be meaningless, because all possible sequences of actions mapping one situation to another are present in all models. What we really want to do is to select only models such that the sequence of actions mapping S0 to S2 doesn't include unnecessary events. Whether or not a model meets this criteria depends on which situations S0 and S2 denote in that model. In other words, we would like to minimise Between(a,S0,S2), letting S0 and S2 vary. More generally, the policy we require is to minimise Between(a,sa,sb) for every sa and sb for which there are situation constants in the language, allowing all those situation constant to vary.

One way to get this effect might be to use pointwise circumscription [Lifschitz, 1986]. Alternatively, two new predicates can be introduced as follows. The formula Named(s) represents that there is a situation constant in the language for s (situation s is named). An axiom schema for Named is assumed which will cover every situation constant. So we have, in the SCP example,

$$\text{Named(S0)}$$

$$\text{Named(S2)}$$

The formula AbBetween(a,s1,sb) represents that sa and sb are named, and there is a sequence of actions mapping sa to sb which includes a.

$$\text{AbBetween(a,sa,sb)} \leftarrow \text{Named(sa)} \wedge \text{Named(sb)} \wedge \text{Between(a,sa,sb)}$$

Then, AbBetween is minimised in parallel with Ab, and every situation constant in the language is allowed to vary, along with Between and Follows. Since AbBetween is mentioned only in this axiom, the mathematics of this modification should be relatively simple.[5] From now on, I will assume this new circumscription policy whenever I use the alternative style of representation.

4. Preconditions and Ramifications

To complete the picture for the deductive approach with the alternative style, I will briefly investigate its application to an explanation problem involving a precondition. Consider (AR1) to (AR4) and (AR6) to (AR9),[6] but suppose that it is a precondition of a successful theft that the car park is unguarded. So instead of (AR5) we have,

$$\neg\text{Holds(Car-parked,Result(Steal,s))} \leftarrow \neg\text{Holds(Guarded,s)} \qquad \text{(AR10)}$$

Now what can we conclude from the fact that the car is not parked in S2? The only plausible explanation, given the knowledge we have, is that the guard went for tea, leaving the car park unguarded, and then the car was stolen. As before, from (AR1) to (AR4) and (1), we have,

[5] For the argument of the paper to go through, it is only necessary for there to exist some technique which will achieve the required minimisation, so no proof is supplied that this particular technique works. The sketch given here is offered as evidence that a working technique can be found.

[6] In fact, (AR9) is superfluous in the following examples, but is included for realism.

$$\exists a,sa,sb\ [Ab(a,Car\text{-}parked,sa) \wedge sb=Result(a,sa) \wedge Follows(sa,S0) \wedge Follows(S2,sb)]$$

From (AR8) to (AR10), minimising Ab according to Baker's approach, we have,

$$Ab(a,f,s) \leftrightarrow [a=Steal \wedge f=Car\text{-}parked \wedge Holds(Car\text{-}parked,s) \wedge \neg Holds(Guarded,s)] \vee [a=Start\text{-}tea \wedge f=Guarded \wedge Holds(Guarded,s)] \vee [a=End\text{-}tea \wedge f=Guarded \wedge \neg Holds(Guarded,s)]$$

and therefore,

$$\exists sa,sb\ [sb=Result(Steal,sa) \wedge Follows(sa,S0) \wedge Follows(S2,sb) \wedge \neg Holds(Guarded,s)]$$

Then, a similar argument applied to the Holds conjunct of the above formula yields,

$$\exists sa,sb,sc,sd\ [sb=Result(Steal,sa) \wedge Follows(sa,S0) \wedge Follows(S2,sb) \wedge sd=Result(Start\text{-}tea,sc) \wedge Follows(sc,S0) \wedge Follows(sa,sd)]$$

which simplifies to,

$$\exists s\ [Between(Start\text{-}tea,S0,s) \wedge Between(Steal,s,S2)]$$

In other words, the guard goes for tea and then the car is stolen — exactly the desired result. However, if we represent the same knowledge in a slightly different way, using ramifications, difficulties arise. Let's introduce a new fluent At-tea, and suppose that there are other new fluents and actions in the domain of discourse, but no new axioms mentioning the actions and fluents already defined. The car park is unguarded while this fluent holds. Instead of (AR8) and (AR9) above, we have the following.

$$\neg Holds(Guarded,s) \leftarrow Holds(At\text{-}tea,s) \qquad (AR11)$$

$$Holds(At\text{-}tea,Result(Start\text{-}tea,s)) \qquad (AR12)$$

$$\neg Holds(At\text{-}tea,Result(End\text{-}tea,s)) \qquad (AR13)$$

Using this style of representation, the minimisation of Ab is different. We have,

$$Ab(a,Car\text{-}parked,s) \leftrightarrow a=Steal \wedge Holds(Car\text{-}parked,s) \wedge \neg Holds(Guarded,s)$$

$$Ab(a,At\text{-}tea,s) \leftrightarrow [a=Start\text{-}tea \wedge Holds(At\text{-}tea,s)] \vee [a=End\text{-}tea \wedge \neg Holds(At\text{-}tea,s)]$$

and therefore, as before, we have,

$$\exists sa,sb\ [sb=Result(Steal,sa) \wedge Follows(sa,S0) \wedge Follows(S2,sb) \wedge \neg Holds(Guarded,s)]$$

But, since (AR11) is an implication rather than a biconditional, the strongest thing we can now conclude about Ab with respect to Guarded is the following.

$$Ab(a,Guarded,s) \leftarrow a=Start\text{-}tea \wedge Holds(Guarded,s)$$

Therefore, the conjunct ¬Holds(Guarded,s) in the explanation cannot be fully expanded. We are left with the following weak explanation.

$$\exists a,s,sc,sd\ [\text{Between}(\text{Steal},s,S2) \wedge \text{Ab}(a,\text{Guarded},sc) \wedge$$
$$sd=\text{Result}(a,sc) \wedge \text{Follows}(sc,S0) \wedge \text{Follows}(s,sd)]$$

The explanation tells us that some action occurred between S0 and the Steal action, and that this action was abnormal with respect to Guarded, but it doesn't give us any idea what the action was. Of course, it's true that "explanations come to an end somewhere," but this seems a little premature. This problem would be particularly acute if the Start-tea action had further preconditions. These wouldn't be dealt with at all by this approach. Since (AR11) to (AR13) constitute a perfectly reasonable representation of the domain, it would be better to adopt an approach to explanation which can cope adequately with ramifications.

5. The Abductive Approach

Abduction is widely considered to be a mode of reasoning fundamental to AI [Charniak and McDermott, 1985, Chapter 8], with applications as diverse as diagnosis [Reggia *et al.*, 1983], [Reiter, 1987], planning [Goebel and Goodwin, 1987], [Eshghi, 1988], plan recognition [Kautz, 1987], natural language interpretation [Hobbs *et al.*, 1990], default reasoning [Poole, 1988], [Eshghi and Kowalski, 1989], [Kakas and Mancarella, 1990] and explanation, the subject of this paper [Shanahan, 1989]. According to the abductive approach to explanation in the situation calculus, given a theory T comprising axioms of motion and the frame axiom (and any other necessary general axioms, such as Baker's "existence of situations"), and a history H representing that certain fluents hold in certain situations, to explain a new fact F representing that a fluent holds in a given situation we need to find a formula Δ such that $T \wedge H \wedge \Delta$ has F among its logical consequences.

In order to avoid trivial or weak explanations, a certain set of predicates are distinguished as *abducible*. Explanations have to be in terms of abducible predicates. Furthermore, to overcome the frame problem, some form of minimisation will be required. So more precisely, we say that, given T and H as above, a formula Δ is an explanation of a fact F if $\text{CIRC}[T \wedge H \wedge \Delta;\ P^*;\ Q^*] \vDash F$ and Δ contains only abducible predicates, where P^* and Q^* are sets of predicates corresponding to a suitable circumscription policy to overcome the frame problem.[7] Of course, there may be many such Δ's to explain any given fact. It is also convenient to avoid explanations which are subsumed by other explanations. So we say that, given T and H, an explanation Δ of F is *minimal* if there is no explanation of F which is a subset of Δ.

In these abductive terms, what is the general form of an explanation problem expressed in the situation calculus? We are usually required to explain a conjunction of positive or negative Holds literals. Let's consider the SCP, using the standard style of representation first. We want to explain (SR3), and we require explanations in terms of previously unsuspected abnormalities. So the obvious policy is to make Ab abducible.

Let T be (1) and H be (SR1) $\wedge$ (SR2). Let Δ be Ab(Wait,Stolen,S0), and assume either McCarthy's or Baker's circumscription policy. As pointed out in Section 2, the abnormality of one of the Wait actions is a necessary but not a sufficient condition for the car to be stolen. Appropriately then, Δ is not an explanation of (SR3) at all according to the abductive approach. Similarly, if we let Δ be Ab(Wait,Stolen,Result(Wait,S0)), then it is still no explanation. In fact, given the standard representation and the abductive approach with Ab made abducible, the disappearance of the car literally defies explanation. Furthermore, since it incorporates no knowledge of Steal actions, the standard representation doesn't permit any explanation of the car's disappearance without the inclusion in Δ of new axioms of motion.

Now let's consider the alternative style. The explanations we require are in terms of the sequence of actions which takes place between two situations. So the obvious abduction policy

[7] $\text{CIRC}[\psi;\ P^*;\ Q^*]$ denotes the circumscription of the formula ψ minimising P^* and allowing Q^* to vary, where P^* and Q^* are sets of predicates.

is to make Between abducible. In the SCP, we want to explain (AR2). Let T be the conjunction of (1) and (AR4) to (AR6), and let H be (AR1) $\wedge$ (AR3). Suppose we minimise abnormality according to either McCarthy's or Baker's approach, and we also minimise AbBetween. Consider Δ=Between(Steal,S0,S2). Does this constitute an explanation?

Minimising AbBetween yields S2=Result(Steal,S0). Then, applying (AR5), we have $\neg$Holds(Car-parked,S2). So Δ is indeed an explanation. There are other explanations too, but each of these involves a sequence of Steal actions. It is easy to see that Δ subsumes all of these explanations, and therefore all minimal explanations will be equivalent to Δ. This approach bears a strong similarity to that of Green [1969] and Kowalski [1979, Chapter 6] to plan formation in the situation calculus, in which resolution generates a binding of the form s=Result(a1,Result(a2,...)) to solve a goal of the form Holds(f,s). This binding conforms exactly to the abductive idea of an explanation with the alternative style of representation, where equality is made abducible.

Note that if we asserted that another action, say going to lunch, occured between S0 and S2, then this Δ would still constitute an explanation, and would furthermore be neutral about the relative order of lunch time and the car's theft. So it would not be possible to conclude, in the presence of Δ, that the car was still in the car park at lunch time.

Next, we'll examine how the abductive approach fares with the alternative style of representation with preconditions. Recall the variant of the SCP with the security guard, which includes axioms (AR7) to (AR10). Once again, we want to explain (AR2). This time, assume Baker's minimisation technique, to ensure that the precondition is properly treated. Let T be the conjunction of an existence of situations axiom with (1), (AR4), (AR6) and (AR8) to (AR10), and let H be the conjunction of (AR1), (AR3) and (AR7). Let Δ be,

$$\exists s\ [\text{Between(Start-tea,S0,s)} \wedge \text{Between(Steal,s,S2)}]$$

This time, the minimisation of AbBetween gives S2=Result(Steal,Result(Start-tea,S0)). Applying (AR8) we get $\neg$Holds(Guarded,Result(Start-tea,S0)). Finally, applying (AR10) we get $\neg$Holds(Car-parked,S2). So Δ is an explanation. Again there are other explanations, involving sequences of Steal, Start-tea and End-tea actions, and again these are all subsumed by Δ, so any minimal explanation will be equivalent to Δ.

The deductive approach didn't cope well when the same problem was represented in a slightly different way using ramifications. Let's see how the abductive approach manages. Let T be the conjunction of an existence of situations axiom with (1), (AR4), (AR6) and (AR10) to (AR13). Let H be the conjunction of (AR1), (AR3) and (AR7), as before. Does Δ still constitute an explanation of (AR2)? The minimisation of AbBetween is the same as before, and we can still show that S2=Result(Steal,Result(Start-tea,S0)). By applying (AR12) and (AR11), we see that $\neg$Holds(Guarded,Result(Start-tea,S0)). Then, applying (AR10) we get $\neg$Holds(Car-parked,S2). So, as before, Δ is an explanation, and all minimal explanations will be equivalent to it.

Discussion

This paper is intended to be a critical exploration of various approaches to explanation within the framework of the situation calculus. I have distinguished a standard and an alternative style of representation, and two approaches to reasoning, deductive and abductive. I have argued that the standard representation is counter-intuitive, and that it runs into difficulties with certain examples, using both the deductive and abductive approaches. I have also argued that the deductive approach, using both styles of representation, is unsatisfactory for certain examples. Accordingly, I cautiously recommend the abductive approach with the alternative style of representation. However, a number of issues remain to be discussed.

For example, the paper has adopted the situation calculus, with circumscription as a means of default reasoning, and has employed Baker's approach to overcoming the frame problem. There are, of course, many alternatives. In particular, abduction itself can be used for default reasoning although I have not addressed this possibility [Poole, 1988], [Eshghi and Kowalski, 1989], [Kakas and Mancarella, 1990]. Neither have I attempted to generalise the paper's conclusions to other formalisms for representing change. However, I conjecture that the lessons learned here will apply to other formalisms, other forms of default reasoning and other approaches to the frame problem. The justification for this conjecture is the representation and reasoning techniques discussed are not specific to the formalism or the form of default reasoning used. Similarly, when a technique goes wrong, it goes wrong for a deep reason, not for a reason which depends on the formalism or form of default reasoning.

The paper has concentrated on variants of a single example, the stolen car problem. So how general are the paper's conclusions? Well, it only requires one counter-example to show the inadequacy of a technique, and the SCP has served admirably as the basis of such counter-examples in this paper. Furthermore, like the Yale shooting scenario in the context of temporal projection, the SCP is simple but representative of a class of explanation problems we would like to solve. Considerable insight into why a technique fails can be gained from studying how it fails on a single representative example.

Of course, to show that a technique <u>is</u> adequate, it is not enough to show that it works on a single example plus a few variants. So what conclusions can be drawn about the abductive approach, other than that it sometimes works when other approaches fail? The impression given in this paper is that abduction and deduction are competing approaches to explanation. But one thing that distinguished the presentations of the deductive and abductive approaches was that the former incorporated a definition of explanation. So, it could be argued that abduction isn't a particular <u>approach</u> to explanation, it is the <u>nature</u> of explanation. A particular approach to explanation might perform abduction directly, or it might simulate it through deduction, so long as the explanations it produced conformed to the abductive definition. Under this interpretation, there is no need to show the adequacy of the abductive approach, because it supplies the very criterion of adequacy.

The next issue I wish to address is knowledge assimilation. A problem like the stolen car problem can be thought of simply as a reasoning problem — what are the possible explanations of the car's disappearance. Alternatively, it can be thought of as a knowledge assimilation problem — how is the fact of the car's disappearance to be assimilated. The abductive and deductive approaches to explanation imply different views of knowledge assimilation. Suppose that we have a knowledge base in the form of a formula T. Under a classical, deductive view of knowledge assimilation, new facts are always added directly to T. With an abductive view of knowledge assimilation, not every fact is eligible for direct addition to T. Sometimes the assimilation of a new fact G demands the addition of a formula Δ of a certain form to T such that $T \wedge \Delta \models G$ [Kowalski, 1979, Chapter 13]. That is, new facts sometimes have to be explained through abduction.

Using abduction with the situation calculus, I suggest that assimilating a new Holds fact, such as the fact that my car is not in the car park in the evening, demands the addition of a formula representing that certain actions take place, so that the new fact becomes a logical consequence of the knowledge base. With the stolen car problem, there is a unique minimal explanation, but this not necessarily the case. One approach to dealing with multiple explanations is to add the disjunction of all minimal explanations to the knowledge base, but this issue is beyond the scope of this paper.

This seems to beg two important questions. Why do some facts demand explanation when others do not? And why are some predicates abducible when others are not? In so far as a problem like the SCP is viewed simply as a reasoning problem, these questions are not very important, since the answers have to be written into the specification of the problem. But taking the wider, knowledge assimilation view, the questions become more pressing. A simple and obvious answer is that anything which can be considered a first cause doesn't require explanation, whereas anything which cannot be considered a first cause does require explanation. For example, we might decide to consider the occurrence of an action as a first cause, but not the effects of an action. This is a partial justification for making Between abducible, and insisting that Holds facts, except those about the initial situation, must be explained. Clearly though, these issues merit further study.

Finally, an important question is the relationship between abduction and deduction [Console *et al.*, 1991], [Konolige, 1992]. When do they coincide? Or, if abduction is adopted as the specification of explanation as suggested above, when does deduction conform to that specification? And why does abduction work in some cases when deduction doesn't? In essence, abduction finds sufficient conditions for a fact to hold, whilst deduction only finds necessary conditions. Under certain conditions, necessary conditions are also sufficient conditions. This is the case when the knowledge involved is expressed in terms of biconditionals. The frame axiom (1), for example, makes it a necessary and sufficient condition for a fluent to hold in Result(a,s) that the fluent holds in s, given that a isn't abnormal in this context. Furthermore, one-way implications can sometimes behave like biconditionals in this way when minimisation is involved, because minimisation often has the effect of "completing" the implication, that is turning it into a biconditional. This was the case with Ab in the SCP. However, there is no reason to suppose that necessary and sufficient conditions will always coincide, even in the presence of minimisation, as we saw with ramifications. Deduction failed with the SCP when ramifications were involved because the implication (AR11) does not behave as a biconditional.

Acknowledgements

Thanks to Tony Kakas, Bob Kowalski and Vladimir Lifschitz. The author is supported by an SERC Advanced Research Fellowship.

References

[Baker, 1989], A.B.Baker, A Simple Solution to the Yale Shooting Problem, *Proceedings 1989 Knowledge Representation Conference*, p 11.

[Baker, 1991], A.B.Baker, Nonmonotonic Reasoning in the Framework of the Situation Calculus, *Artificial Intelligence*, vol 49 (1991), p 5.

[Charniak and McDermott, 1985] E.Charniak and D.McDermott, Introduction to Artificial Intelligence, Addison-Wesley, 1985.

[Console *et al.*, 1991] L.Console, D.Dupré and P.Torasso, On the Relationship between Abduction and Deduction, *Journal of Logic and Computation*, vol 1 (1991), p 661.

[Eshghi, 1988] K.Eshghi, Abductive Planning with Event Calculus, *Proceedings 5th International Conference on Logic Programming* (1988), p 562.

[Eshghi and Kowalski, 1989] K.Eshghi and R.A.Kowalski, Abduction Compared with Negation by Failure, *Proceedings 6th International Conference on Logic Programming* (1989), p 234.

[Gelfond *et al.*, 1991] M.Gelfond, V.Lifschitz and A.Rabinov, What Are the Limitations of the Situation Calculus? in *Essays for Bledsoe*, ed R.Boyer, Kluwer Academic (1991), p 167.

[Goebel and Goodwin, 1987] R.G.Goebel and S.D.Goodwin, Applying Theory Formation to the Planning Problem, *Proceedings of the 1987 Workshop on the Frame Problem*, p 207.

[Green, 1969] C.Green, Applications of Theorem Proving to Problem Solving, *Proceedings IJCAI 69*, p 219.

[Hanks and McDermott, 1987] S.Hanks and D.McDermott, Nonmonotonic Logic and Temporal Projection, *Artificial Intelligence*, vol 33 (1987), p 379.

[Haugh, 1987] B.A.Haugh, Simple Causal Minimizations for Temporal Persistence and Projection, *Proceedings AAAI 87*, p 218.

[Hobbs *et al.*, 1990] J.R.Hobbs, M.Stickel, D.Appelt and P.Martin, Interpretation as Abduction, Technical Report, SRI International, Menlo Park CA, (1990).

[Kakas and Mancarella, 1990] A.C.Kakas and P.Mancarella, Generalized Stable Models: A Semantics for Abduction, *Proceedings ECAI 90*, p 385.

[Kautz, 1986] H.Kautz, The Logic of Persistence, *Proceedings AAAI 86*, p 401.

[Kautz, 1987] H.Kautz, A Formal Theory of Plan Recognition, PhD Thesis, University of Rochester, Rochester N.Y., 1987.

[Konolige, 1992] K.Konolige, Abduction Versus Closure in Causal Theories, *Artificial Intelligence*, vol 53 (1992), p 255.

[Kowalski, 1979] R.A.Kowalski, Logic for Problem Solving, North Holland, 1979.

[Lifschitz, 1986] V.Lifschitz, Pointwise Circumscription: Preliminary Report, *Proceedings AAAI 86*, p 406.

[Lifschitz, 1987] V.Lifschitz, Formal Theories of Action, *Proceedings of the 1987 Workshop on the Frame Problem*, p 35.

[Lifschitz and Rabinov, 1989] V.Lifschitz and A.Rabinov, Miracles in Formal Theories of Action, *Artificial Intelligence*, vol 38 (1989), p 225.

[McCarthy, 1986] J.McCarthy, Applications of Circumscription to Formalizing Common Sense Knowledge, *Artificial Intelligence*, vol 26 (1986), p 89.

[McCarthy and Hayes, 1969] J.McCarthy and P.J.Hayes, Some Philosophical Problems from the Standpoint of Artificial Intelligence, in *Machine Intelligence 4*, ed D.Michie and B.Meltzer, Edinburgh University Press (1969).

[Morgenstern and Stein, 1988] L.Morgenstern and L.A.Stein, Why Things Go Wrong: A Formal Theory of Causal Reasoning, *Proceedings AAAI 88*, p 518.

[Poole, 1988] D.Poole, A Logical Framework for Default Reasoning, *Artificial Intelligence*, vol 36 (1988), p 27.

[Reggia *et al.*, 1983] J.A.Reggia, D.S.Nau and P.Wang, Diagnostic Expert Systems Based on a Set Covering Model, *International Journal of Man-Machine Studies*, vol 19 (1983), p 437.

[Reiter, 1987] R.Reiter, A Theory of Diagnosis from First Principles, *Artificial Intelligence*, vol 32 (1987), p 57.

[Shanahan, 1989] M.P.Shanahan, Prediction Is Deduction but Explanation Is Abduction, *Proceedings IJCAI 89*, p 1055.

[Shoham, 1988] Y.Shoham, Reasoning About Change: Time and Change from the Standpoint of Artificial Intelligence, MIT Press (1988).

Explanation Reconfiguration in Abductive Reasoning

(Extended Abstract)

Makoto Motoki

C&C Systems Research Laboratories, NEC Corporation
4-1-1 Miyazaki, Miyamae-ku, Kawasaki 216, JAPAN

e-mail: motoki@btl.cl.nec.co.jp

1 Introduction

Abduction is a form of reasoning that derives a plausible explanation from incomplete knowledge to explain a set of given data. A number of researchers have proposed abductive reasoning frameworks [8, 1, 5, 6, 3, 10]. Basically, they all define abductions in the following way: when a set of data p is given, abduction finds an explanation $\Delta(\subseteq D)$ such that $T \cup \Delta \models p$, $T \cup \Delta$ *is consistent*, where T is a set of consistent facts and D is a set of hypotheses (not necessarily consistent).

For use in the real world, abductive frameworks should, additionally, be made defeasible, i.e., in order to make an existing explanation consistent with a newly generated explanation for a new set of data, it should be possible to reconfigure the previously existing explanation.

This paper proposes a framework for abduction which includes a mechanism for explanation reconfiguration. That is, in our framework of abduction, the following definition is added to the above-mentioned definition of abduction:

> Assume that an abductive framework has a set of facts T and a set of previously constructed explanations Δ_1 such that $T \cup \Delta_1$ *is consistent*, and that a given data q is to be explained. When a newly derived explanation Δ_2 of q leads to the inconsistency
>
> $$T \cup \Delta_1 \cup \Delta_2 \models q,\ T \cup \Delta_1 \cup \Delta_2 \textit{ is inconsistent},$$
>
> our abductive framework provides a procedure to reconfigure Δ_1 in order to derive $\Delta_1' \cup \Delta_2$ such that
>
> $$T \cup \Delta_1' \cup \Delta_2 \models q,\ T \cup \Delta_1' \cup \Delta_2 \textit{ is consistent}.$$

We use extended logic programs [2] as a background theory to formalize our framework. Extended logic programs can represent a definition of a negative literal ($\neg a$), such as $\neg a \leftarrow b$, and this allows us to introduce into our framework the concept, used in Theorist [8], of *constraints*. In Theorist, a *constraint* prevents a hypothesis a from being applicable under circumstances b.

Our approach to abduction is to introduce contrapositives of constraints (e.g. $\neg a \leftarrow b$ etc.) and/or other facts (e.g. $b \leftarrow c$ etc.) only when they would block the application of a hypothesis (e.g. a etc.) in a newly generated explanation Δ_2 under the circumstance $T \cup \Delta_1 \cup \Delta_2$. Roughly speaking, we use these contrapositives (e.g. $a \rightarrow \neg b$ and $\neg b \rightarrow \neg c$) to derive a new consistent set of hypotheses including all elements of Δ_2 by determining what would be true or false in Δ_1 if a blocked hypothesis a were true. This determination produces the desired reconfiguration of the previous explanation Δ_1.

This paper proposes a meta-rule, *Empirical Contrapositive Control (ECC)*, for introducing contrapositive variants into our abduction framework.

2 Framework of the Proposed Abduction

This section defines a defeasible abductive reasoning and its framework.

Definition 1 Let Lit be the set of ground literals in the language Π. A set of clauses Π are *consistent* if and only if there are answer sets of Π and no answer set is Lit. Π is *inconsistent* if and only if there is an answer set of Π which is equal to Lit.

Definition 2 An *abduction framework* is a quadruple (T, Δ, D, C) where

- T is a theory comprised by a consistent set of clauses of the form $H \leftarrow L_1, \cdots, L_n$ (H is an atom and $H \notin D$),
- $\Delta (\subseteq D)$ is a previous explanation for a previous goal g_o such that $T \cup \Delta \vdash g_o$, $T \cup \Delta$ *is consistent*,
- D is a set of hypotheses consisting of literals,
- C is a set of constraints consisting of a set of extended logic programs of the form $L_0 \leftarrow L_1, \cdots, L_n$ where $L_0 \in D$, and $T \cup C$ *is consistent*.

Definition 3 Let T be a set of theories, C be a set of constraints, and Δ_1 be an explanation for a goal g_1 such that $T \cup C \cup \Delta_1 \models g_1$, $T \cup C \cup \Delta_1$ *is consistent*. Let Δ_2 be an explanation for a new goal g_2 such that $T \cup C \cup \Delta_2 \models g_2$, $T \cup C \cup \Delta_2$ *is consistent*, but $T \cup C \cup \Delta_1 \cup \Delta_2$ *is inconsistent*. *Explanation reconfiguration* is to find a set of explanations $\Delta_1' \cup \Delta_2$ such that $T \cup C \cup \Delta_1' \cup \Delta_2$ *is consistent*. (where $\Delta_1' = \Delta_1^{sub} \cup \Delta^{sub}$, $\Delta_1^{sub} \subseteq \Delta_1$, $\Delta^{sub} \subseteq D - (\Delta_1 \cup \Delta_2)$). Here, $\models$ represents the entailment of extended logic programs.

Definition 4 Let T be a set of theories, Δ_1 be a previous explanation. Let $\overline{p}$ be the complementary literal of p. *Empirical Contrapositive Control (ECC)* is a meta rule such that

for $p \leftarrow q_1 \cdots q_i \cdots q_n$ $(n \geq 1)$,

ECC introduces $\overline{p}, q_1, \cdots, q_{i-1}, q_{i+1}, \cdots, q_n \rightarrow \overline{q_i}$ if $T \cup C \cup \Delta_1 \vdash p$ and $T \cup C \cup \Delta_1 \vdash q_1, \wedge \cdots \wedge q_n$.

Basically, ECC introduces contrapositive variants of rules whose both left and right hand sides can be derived from $T \cup C \cup \Delta_1$. From the definition of explanation reconfiguration, in any case where explanation reconfiguration is needed, there are at least one pair of complementary literals derived from $T \cup C \cup \Delta_1 \cup \Delta_2$, but no pair of complementary literal is derived from $T \cup C \cup \Delta_1$.

According to Definition 2, only constraints derive negative literals, so that, in the answer set, negative forms of literals are only those in D. Thus, in this situation, there is at least one hypothesis, say A, in Δ_2 such that $T \cup C \cup \Delta_1 \vdash \neg A$. Obviously, both left and right hand sides of all rules involving a derivation of $\neg A$ can be derived from $T \cup C \cup \Delta_1$. This indicates that, by using ECC, we can obtain contrapositives of all rules involving the derivation of the negative form of A in Δ_2. Assume that we have R as an atom in Δ_1, and have $\neg P \leftarrow Q$ and $Q \leftarrow R$ in $T \cup C$. Since $\neg P, Q, R$ are derived from $T \cup C \cup \Delta_1$, we can obtain the following contrapositives by ECC; $P \rightarrow \neg Q$; $\neg Q \rightarrow \neg R$. Using these contrapositives, we can conclude that R in a previous explanation should be removed ($\neg R$ should hold instead) if P in a new explanation holds. In this way, we use ECC in the process of explanation reconfiguration. Next section presents precise procedure of explanation reconfiguration.

3 The Procedure of Explanaiton Reconfiguration

This procedure consists of two parts; consistent explanation generation procedure and explanation reconfiguration procedure. These procedures return reconfigured explanations for a new goal. In the procedure, $\overline{p}$ represents the complementary literal of p.

Explanation Generation(EG)

EG consists of pre explanation generation(PEG) procedure which finds hypotheses necessary for explaining a new goal G, and consistency check($CC1$) procedure which checks consistency of these hypotheses. EG generates an explanation Δ_2 such that $T \cup C \cup \Delta_2$ *is consistent*.

$\underline{PEG(G, \Delta_1)}$

Let T be theories, D be hypotheses, and Δ_1 be previous explanations. When a goal G of the form $\leftarrow A_1, \cdots, A_n$ is given, EG starts from a triple $(\leftarrow A_1, \cdots, A_n, \Delta_1, \{\})$ and continues the following procedures until it gets the form of $(false, \Delta_1, \Delta_2)$. EG returns Δ_2, and do $CC1(T, C, \Delta_2)$. If $(false, \Delta_1, \Delta_2)$ cannot be derived, EG returns "there is no explanation for G"

Assume that the current triple is $(\leftarrow A_1, \cdots, A_n, \Delta_1, \Delta_2)$

- if $(A_i \in D) \& (A_i \notin \Delta_1)$ then let the current triple be $(\leftarrow A_1, \cdots, A_{i-1}, A_{i+1}, \cdots, A_n,\ \Delta_1, \Delta_2 \cup \{A_i\})$
- if $A_i \in T \cup \Delta_1$ then let the current triple be $(\leftarrow A_1, \cdots, A_{i-1}, A_{i+1}, \cdots, A_n,\ \Delta_1, \Delta_2)$
- if $A_i \leftarrow B_1, \cdots, B_m \in T$ then let the current triple be $(\leftarrow A_1, \cdots, A_{i-1}, B_1, \cdots, B_m, A_{i+1}, \cdots, A_n, \Delta_1, \Delta_2)$

$\underline{CC1(T, C, \Delta_2)}$

- **if** there is $d_2 \in \Delta_2$ such that $T \cup C \cup \Delta_2 \vdash \overline{d_2}$ **then return** "there is no explanation for G" **else** do ER.

Explanation Reconfiguration(ER)

ER finds a subset of D, $\Delta_1 - Discarded \cup \overline{Discarded} \cup \Delta_2 \cup Added$, which comprises a consistent answer set $\alpha(\Pi)$, and returns the positive literals in this set as a new explanation. Here, $\overline{Discarded}$ represents the set of complementary literals of all elements in $Discarded$. $CC2$ finds the subset of Δ_1 negated by Δ_2, REC finds the subset of Δ_1 deriving a negative form of literal in Δ_2, and these two sets comprise $Discarded$. REC also finds the set $Added$ ($\in D - (\Delta_1 \cup \Delta_2)$) which are in $\alpha(\Pi)$ iff $\overline{Discarded}$ are in $\alpha(\Pi)$.

$\underline{CC2(T, C, \Delta_1, \Delta_2)}$

```
1. Discarded := {}; for every $d_1 \in \Delta_1$ if $T \cup C \cup \Delta_2 \vdash \overline{d_1}$ then Discarded := Discarded $\cup \{d_1\}$.
2. if $T \cup C \cup \Delta_1 \not\vdash \overline{d_2}$ for all $d_2 \in \Delta_2$ then return $\Delta_1 -$ Discarded $\cup \Delta_2$ as an explanation for $G$.
   if not, do $REC(T, C, \Delta_1, \Delta_2', ECC)$ where $\Delta_2'$ is a subset of $\Delta_2$ such that $T \cup C \cup \Delta_1 \vdash \neg d_2'$ for all $d_2' \in \Delta_2$
```

<u>$REC(T, C, \Delta_1, \Delta_2', ECC)$</u>

```
1. For every $p \in \Delta_2'$, find constraints $\overline{p} \leftarrow q_1, \cdots, q_n$ such that $T \cup C \cup \Delta_1 \vdash \overline{p}$ and $T \cup C \cup \Delta_1 \vdash q_1 \wedge \cdots \wedge q_n$
   and get their contrapositive variants of the form $p, q_1, \cdots, q_{i-1}, q_{i+1}, \cdots, q_n \rightarrow \overline{q_i}$ by using $ECC$
2. Added := {}. For every $q_i$
   repeat Find theories $q_i \leftarrow r_1, \cdots, r_m$ such that $T \cup C \cup \Delta_1 \vdash q_i$ and $T \cup C \cup \Delta_1 \vdash r_1 \wedge \cdots \wedge r_m$,
   get their contrapositive variants of the form $\overline{q_i}, r_1, \cdots, r_{j-1}, r_{j+1}, \cdots, r_m \rightarrow \overline{r_j}$ by using $ECC$, and regard $r_j$ as
   a new $q_i$ until $ECC$ introduces no more contrapositive variants.
   For every consequence $\overline{r_j}$
       if $r_j \in \Delta_1$ then Discarded := Discarded $\cup \{r_j\}$ else if $\overline{r_j} \in D - \Delta_1 - \Delta_2$ then Added := Added $\cup \{\overline{r_j}\}$.
3. Find $d' \in \Delta_1 -$ Discarded $\cup \Delta_2 \cup$ Added such that $T \cup C \cup \Delta_1 -$ Discarded $\cup \Delta_2 \cup$ Added $\vdash \neg d'$.
   Find $d'' \in$ Discarded such that $T \cup C \cup \Delta_1 -$ Discarded $\cup \Delta_2 \cup$ Added $\vdash d''$.
   if there is no $d'$ and $d''$ then return ($\Delta_1 -$ Discarded $\cup \Delta_2 \cup$ Added)
       else return "there is no explanation for $G$"
```

Example

Assume that a set of theories are $\{P \leftarrow Q;\ S \leftarrow T;\ S \leftarrow \neg S;\ R \leftarrow\}$, a set of constraints are $\{\neg Q \leftarrow R, S;\ \neg U \leftarrow Q\}$, a set of hypotheses $\{Q, T, U, V\}$, and a previous explanation Δ_1 is $\{T, U, V\}$. For a new goal P, EG generates $\{Q\}$, $CC2$ finds $T \cup C \cup \Delta_2 \vdash \neg U$ ($U \in \Delta_1$), i.e., $\{U\}$ as *Discarded*. $CC2$ also finds $T \cup C \cup \Delta_1 \cup \Delta_2 \vdash \neg Q$ ($Q \in \Delta_2$). REC introduces $\{\neg S \leftarrow Q, R;\ \neg R \leftarrow Q, S\}$ as contrapositives of $\neg Q \leftarrow R, S$, $\{\neg T \leftarrow \neg S\}$ as a contrapositive of $S \leftarrow T$, and finds there is no more contrapositive introduced by ECC. As consequences of contrapositives, REC finds $\neg R$ and $\neg T$. Since T is a hypothesis in Δ_1, REC derives $\{T\}$ as *Discarded*. Finally, ER generates $\{Q, V\}$ as a new explanation. Note that V in Δ_1, which is not inconsistent with Δ_2, is intact in this procedure.

Some Properties of Defeasible Abduction

ER procedure is a sound procedure of *explanation reconfiguration* defined in Definition 3.

Theorem [Soundness of ER procedure] Let T be a set of theories, C be a set of constraints, D be hypotheses, Δ_1 be a previous explanation, Δ_2 be a new explanation, $\Delta_1' \subseteq \Delta_1$, and $\Delta' \subseteq D - (\Delta_1 \cup \Delta_2)$. Assume that ER returns $\Delta_1' \cup \Delta' \cup \Delta_2$. If both $T \cup C \cup \Delta_1$ and $T \cup C \cup \Delta_2$ *are consistent* but $T \cup C \cup \Delta_1 \cup \Delta_2$ *is inconsistent*, then $T \cup C \cup \Delta_1' \cup \Delta' \cup \Delta_2$ *is consistent*.

As shown above, we proposed a sound explanation reconfiguration procedure by introducing contrapositive variants of both constraints and theories. However, the unrestricted introduction of contrapositive variants deteriorates the performance of ER procedure and also can cause unwanted side effects.

For example, assume that ECC introduces contrapositives of arbitrary constraints and theories in Example. We will obtain $\{\neg Q \leftarrow \neg P;\ U \leftarrow \neg Q;\ S \leftarrow \neg S\}$ besides those introduced by original ECC.

Clearly, this naive version of ECC produces unnecessary contrapositives for reconfiguration like $\{\neg Q \leftarrow \neg P;\ U \leftarrow \neg Q\}$. Moreover, this introduces harmful contrapositives like $\{S \leftarrow \neg S\}$. Since this contrapositive variant is exactly the same form as the original rule, REC procedure results in infinite loop.

On the other hand, ECC introduces contrapositives only when both left and right hand sides of a rule can be derived from $T \cup C \cup \Delta_1$. Furthermore, ER procedure only applies ECC to clauses deriving the negative form of a hypothesis in Δ_2 ($\neg Q$ in Example). Thus, in the ER procedure, ECC introduces contrapositive variants of rules if and only if these rules are in the derivations of $\neg Q$. These controls make it possible for ER to derive hypotheses in Δ_1 to be discarded if the negative form of a hypothesis in Δ_2 can be derived.

4 Discussion & Related Work

This paper presented a framework of defeasible abduction capable of reconfiguring previous explanations in accordance with a new requirement. As a mechanism to handle defeasible abduction, we proposed a meta-rule ECC which introduces contrapositive variants of theories and constraints.

While our framework described limited *theories* to general logic programs, this restriction can be expanded to extended logic programs by elaborating the consistency checking procedure.

Our framework uses constraints to prevent certain hypotheses from being applicable under a specific circumstance. This idea is similar to the idea of *constraint* in Theorist [8]. However, when a hypothesis in a newly added explanation is prevented, Theorist removes this new hypothesis, whereas our defeasible abduction removes a part of the previous explanation so that the rest of the explanation satisfies this constraint.

Kakas & Mancarella [5, 6] developed abductive framework on general logic programs with integrity constraints, which is the special case of extended logic programs. In this framework, it's impossible to introduce contrapositive variants. They did not describe any detailed exposition of their theory for extended logic programs.

Kowalski & Sadri [7] used extended logic programs to express exceptions. Basically, their expression of exceptions corresponds to our constraint expression. Their approach always prefers negative literals to positive ones. This method is too limited to deal with explanation reconfiguration.

Inoue [3] proposed an abductive framework based on extended logic programs. This framework focuses on deriving a plausible explanation for *one* goal, whereas our framework focuses on the defeasible aspects of abductive reasoning.

References

[1] Eshghi, K. and Kowalski, R.: Abduction Compared with Negation by Failure, in *Proc. 6th International Conference on Logic Programming,* (1989) pp 234–254

[2] Gelfond,M and Lifschitz, V.: Logic programs with classical negation, in *Proc. 7th International Conference on Logic Programming,* (1990) pp 579–597

[3] Inoue, K. : Extended Logic Programs with Default Assumption, In *8th International Conference on Logic Programming,* (1991) pp 490–504

[4] Kakas, A.C. and Mancarella, P.: Anomalous Models and Abduction, in *Proc. 2nd International Symposium on Artificial Intelligence,* (1989)

[5] Kakas, A.C. and Mancarella, P.: Generalized Stable Models: A Semantic for Abduction, in Proc. 9th European Conference on Artificial Intelligence, (1990) pp 385–391

[6] Kakas, A.C. and Mancarella, P.: On the relation between Truth Maintenance and Abduction, in Proc. Pacific Rim International Conference on Artificial Intelligence '90, (1990) pp 438–443

[7] Kowalski, R. and Sadri, F.: Logic Programming with Exceptions, *New Generation Computing, 9* (1991) pp 387–400

[8] Poole, D.: A Logical Framework for Default Reasoning, *Artificial Intelligence 36* (1988) pp 27–47

[9] Reiter, R.: A Logic for Default Reasoning, in *Artificial Intelligence 13*, (1980) pp 81–132

[10] Selman, B. and Levesque, H.J.: Abductive and Default Reasoning: A Computational Core, *AAAI-90,*(1990) pp 343–348

A Formal Scheme of Semantic Networks (Extended Abstract)

Stephen T. C. Wong

Institute for New Generation Computer Technology (ICOT)
21F. Mita-Kokusai Bldg., 1-4-28, Mita, Minato-ku, Tokyo 108, Japan
e-mail: wong@icot.or.jp Tel: +81-3-3456-3069 Fax. +81-3-3456-1618

1 Introduction

A semantic network is a structure for representing knowledge as a pattern of interconnected nodes and links [4]. Semantic networks have been used extensively in AI applications. Most network models of these applications, however, are rather loosely defined. They lack a rational and systematic means to specify network entities such as nodes and links. Consequently, the information processing behavior of these *ad-hoc* models are difficult to predict. That is, such network models, sometimes, may derive inconsistent conclusions and are not robust for future expansion or modification.

Object-orientation is a potentially important paradigm for software programming. Compared to procedural software programs, object-oriented programs exhibit advantageous features of information processing, such as data encapsulation and properties inheritance, and offer better maintainability. Nevertheless, the terminology and notations vary widely, and currently, there has no one agreeable way to construct these programs.

The observation is that, as discussed in Section 2, the paradigm of object-orientation can help to provide precise and rigorous specifications of semantic networks. The purpose of this abstract to outline such a knowledge representation scheme that is grounded on the formal characterization of objects and their properties[1].

The type of AI systems considered in this scheme are *logical* knowledge-based systems. A semantic network organizes the set of flat logical clauses or sentences in the knowledge base into a network of localized theories. In this way, it offers the advantages of encapsulation and modularization, as in object-oriented software programs, and provides the system designer with a knowledge representation tool more expressive than first order logic.

This abstract first describes the intuitive concepts behind the network scheme: properties, objects, relations, and multiple orders of relations between these entities. It then tightens these concepts with formal specifications, mainly at the set-theoretical level. The representation scheme has been implemented in DOOS[2], a knowledge representation system for developing application prototypes, in particular, in the domain of structural engineering [1].

2 Guiding Principles

The representation scheme comprises of three knowledge levels. (1) objects, (2) determinate relationships between objects, and (3) relations of proportion among these determinate relationships. The philosophical motivation of this structuring of knowledge is due to the earlier work of Frege[3] and recent extensions of it by Bigelow and Pargetter[2]. Before presenting a formal account of this structure, in this section, I briefly describe the intuitive concepts of constructing these entities in each of the knowledge levels.

[1]This work is based on the author's research that was done at the Center of Advanced Technology for Large Structural Systems (ATLSS), an NSF sponsored Engineering Research Center at Lehigh University, USA.

[2]DOOS stands for Deductive Object-Oriented Semantic network and is written on top of Quintus Prolog.

Let us first consider level 1 objects. One characteristic about modeling a large structural system, such as a bridge or a building, is that there are real objects in the physical world which a knowledge base describes. Observation reveals to us the properties and relations of particular physical objects. For example, a bridge girder may be made out of steel and is composed of two other types of objects, namely, flange and web.

The basic entities of a network model of a physical structure are objects and relations. An object is a node, and a relation between two objects is a link. The interactions and interrelations of these entities construe the behaviors and properties exhibited by the overall physical structure. An object is a collection of properties. A property does not exist on its own. Whenever we talk about a property, we must refer it as a property of some object or a group of objects. There are also multiple orders of properties. For example, two objects of different colors, say one gray and the other red, share a common property of having a property which is a color. That common property is a *second order* property.

Further, if we are willing to take the realist point of view, we can then extend the notion of objects toward abstract concepts. For example, the objects of one of DOOS applications for bridge fatigue analysis include not only physical components of bridges but also abstract concepts such as fracture mechanics, diagnostic procedures, and remedy solutions. The properties of these objects of abstraction, however, are often difficult to observe and less determinate. Hence, the modeling of abstract concepts as objects will be subjective and be dependent of the applications.

For level 2 relations among objects, sometimes, we can ground them with the properties of objects. For example, consider two objects of different masses. If object o_1 is more massive than the other object o_2, then this relationship is denoted as *more_massive*(o_1, o_2), which is a relation between the mass property of o_1 and o_2. Another example would be the *isa* relation, this scheme defines *isa* as a parent and child relationship between two objects, or two sets of properties.

However, not all level 2 relations rest on properties. Consider the analysis of spatial relations, like *west of*. Suppose one tries to ground this relation in intrinsic properties of location. Object o_1 is west of object o_2, because o_1 is here and o_2 is there. But why should o_1's being here, and o_2 there, entail the former being west of the latter. Only because 'here' is a position which is west of the position which constitute 'there'. One presupposes the existence of spatial relations between their positions. Another case of this sort is temporal relations. The relation of *earlier* to *later* is arguably not solely a product of the properties of related events. Such kinds of relations are called intrinsic relations and cannot be grounded in properties alone.

The knowledge level 3 relations of proportion classify level 2 relations into equivalent classes. Within each equivalent class of level 2, level 3 proportions will also impose an ordering. It is this ordering which explains how an object can be closer to a second than it is a third regarding certain quantifiable property, e.g. mass, length, velocity, and volume. For example, for three pairs of objects, $< o_1, o_2 >, < o_3, o_4 >, < o_5, o_6 >$, o_1 is five times as massive as o_2, o_3 is six times as massive as o_4, and o_5 is one hundred times as massive as o_6. Then there is an important respect in which the first two pairs are more closely similarly to the another than either is to the third. Such a comparison can be extended to cross-category similarities. Two physical objects may differ in mass to just the same degree as they differ in, say, length, e.g., one may be twice the mass and twice the length of the other. Yet there is something these two different relations share, the same degree of proportion: *twice*.

The level 3 relations not only define a rich network of second-order properties and relations among objects, but also explain the pattern of entailment relations which hold among them. The experience with DOOS applications in the ATLSS center indicates that such higher order relations work reasonably well for the characterization of large structural systems, as the properties of physical components of such systems generally are comparable. Nevertheless, the three-level structure may not adequate to describe relations concerning non-physical aspects of objects such as preferences and opinions. One of the representation methods that supplements the scheme in dealing with less discriminating relations is presented in [6]. In the following sections, I work out the idea of three-level structuring of knowledge into a formalized scheme of semantic networks.

3 Objects and Properties

The formal language used is a mix of set theory and first order logic with customary set-theoretical symbols and logical connectives such as $\neg$(negation), $\vee$(or), $\wedge$(and), $\rightarrow$(implication), and so forth, but without explicit quantifiers. To reflect the underlying programming language Prolog of DOOS, a formula with free variables is treated as an existential sentence. In addition, the definition of a constant symbol also includes a list. This definition enables us to treat the instantiation of variables with constants only.

A network model M about the physical world is a tuple $< O, C, S, R >$, where O is a set of objects or nodes, C is a set of relations or links, S is a set of integrity constraints, and R is a set of revision rules. An object is an abstraction of an entity of the physical world. It is characterized by its state, which in turns is defined by a set of properties. That is, for any model M and any $o \in O$,

Definition 1 *$o = name(o) \wedge (p_1(ts_1, o) \wedge \cdots \wedge p_n(ts_n, o))$, where $name(o)$ is the unique name of the object and $p_i(ts_i, o)$ is the i^{th} property of o with the term ts_i as its determinate value.*

Definition 2 *For any property of object o, $p(ts, o)$, if ts is a constant term, then the property is called a determinate property; otherwise, if ts is a variable, then the property is called an indeterminate property.*

Strictly speaking, the $name(o)$ is also a property of o but it is singled out to indicate the uniqueness of object's name. Denote the set of properties of o as $Prop(o)$, that is, $Prop(o) = \{p_1(ts_1, o), \cdots, p_2(ts_n, o)\}$. Then, object o can also be expressed as $\{name(o), Prop(o)\}$. $Prop(o)$ can be partitioned into two sets: essential properties $Prop_e(o)$ and hereditary properties $Prop_h(o)$. The former set consists of those non-inherited properties of object o, and the latter set is inherited from parent objects of o.

Definition 3 *For any model M and any $o \in O$, $Prop(o) = Prop_h(o) \cup Prop_e(o)$.*

For clarity I consider only single determinate value for every property but the actual implementation involves properties of multiple terms such as $p(ts_1, \cdots, ts_m, o)$. In addition, the predicate p of a property $p(ts_i, o)$ is called a determinable, and the set of determinables of o is denoted as $Db(o)$. Any two objects having the same set of determinables are said to belong to the same type. The set of objects that are of the same type as o is defined below.

Definition 4 *$Type(o) = \{o\} \cup \{o_i \mid Db(o) = Db(o_i)\}$*

The notion of class for objects is defined in terms of their properties:

Definition 5 *$Class(o) = \{o_i \mid Prop(o) \subsetneq Prop(o_i)\} \subseteq O$.*

Furthermore, for any objects o, o', $Class(o)$ is the superclass of $Class(o')$ or $Class(o')$ is a subclass of $Class(o)$ if $Prop(o')$ is a subset of $Prop(o)$, i.e., $Class(o')$ is a subset of $Class(o)$.

A set of uniqueness conditions for the objects and their properties are stated in below. For any $o, o_i, o_j \in O$ such that $i \neq j$,

Condition 1 *Unique name condition*
$name(o_i) \neq name(o_j)$

Condition 2 *Unique property condition*
If $name(o_i) \neq name(o_j)$, then $Prop(o_i) \neq Prop(o_j)$

Condition 3 *Unique determinable condition*
If $p_i(ts_i, o), p_j(ts_j, o) \in Prop_e(o)$, then $p_i \neq p_j$.

Next, internal to an object, a property is equivalent to one of the following logical forms:

Definition 6 *Property form*
For any $o \in O$, $p(ts, o) \equiv p(ts)$ (unit clause) or
$p(ts, o) \equiv a_1 \wedge \cdots \wedge a_m \rightarrow p(ts)$ where $a_1, \cdots a_m$ are first order sentences which may include properties of its own or other objects of O (if-then clause).

For example, consider the loading cycle of an object o_{bridge} that models a bridge.

$$\begin{aligned} loading\_cycle(ts_3, o_{bridge}) \quad \equiv \quad & age(ts_1, o_{bridge}) \wedge average\_daily\_traffic(ts_2, o_{bridge}) \wedge \\ & ts_3 = ts_1 \times ts_2 \times 365) \rightarrow loading\_cycle(ts_3) \end{aligned}$$

This may be read as: "The loading cycle of a bridge, o_{bridge}, is equal to the multiplication of its age, the average daily traffic through the bridge, and a factor of 365 (a year)."

Note that both $age(ts_1, o_{bridge})$ and $average\_daily\_traffic(ts_2, o_{bridge})$ are properties of the same object o_{bridge}. Suppose that $age(50, o_{bridge}), average\_daily\_traffic(1000, o_{bridge}) \in Prop(o_{bridge})$, that is, the bridge is fifty years old and in average has a traffic load of a thousand per day. Then we would get $loading\_cycle(1725000, o_{bridge})$, as $50 \times 1000 \times 365 = 1725000$.

Note that the syntactical form of a property is similar to a Prolog clause. This is expected as the DOOS system is written on top of Prolog. Since an object consists of a set of properties, it can be considered as a localized theory. A network of localized theories thus forms the theory of the knowledge-based system about the physical world. The localization of logical clauses or sentences offers the advantages of encapsulation and modularization to logic-based systems in analogy to object-orientation for software programs.

Denote the set of properties or the localized theory of an object o as T_o. I write, as in classical logic, $T_o \vdash p$ if and only if (iff) there exists a derivation (or a proof) of p from T_o or any subset of T_o[3]. The rules for valuating properties then are,

Rule 1 *Determinate property*
For any ground property $p(ts, o)$ of an object o, where ts is not a variable, $p(ts, o)$ is true if $p \in Db(o)$ such that $T_o \vdash p(ts)$; $p(ts, o)$ is false if $p \in Db(o)$ such that either $T_o \vdash p(ts')$ with $ts' \neq ts$ or $T_o \vdash \neg p(ts)$. Otherwise, $p(ts, o)$ is unknown.

Rule 2 *Indeterminate property*
For any non-ground property $p(v, o)$, where v is a variable, $p(v, o)$ is true if $p \in Db(o)$ such that $T_o \vdash p(c/v)$, where c is a constant that instantiates v; otherwise, $p(v, o)$ is unknown.

The derivation of unknown values in this scheme differs from the mode of negation-as-failure in traditional Prolog-based systems. The reason is that a knowledge-based system contains only partial knowledge of the physical world. And when the system cannot derive a determinate property or instantiate an indeterminate one, it does not mean that the property is false — the information for determining the truth-value of that property may be available later during the interaction with human users or another knowledge-based systems.

4 Relations

This subsection presents some examples about the specifications of relations in C of M. As mentioned in Section 2, the level 2 relations are either grounded with properties of related objects or intrinsic in nature. As an example, the relation *longer* between two objects can be defined in terms of their *length* properties:

Definition 7 *For any objects $o_1, o_2 \in O$, $longer(o_1, o_2)$ iff $length(ts_1, o_1) \in Prop(o_1)$ and $length(ts_2, o_2) \in Prop(o_2)$ such that $ts_1 > ts_2$*[4].

[3] I do not specify the inference rules here as it would depend on particular logical systems which the network model built on, however, inference rules of such systems should include at least the equivalence of modus ponens.

[4] Note that the mathematical relation $>$ (greater than) is also a relation. Relations of this sort, however, are over terms not objects, thus, differ from the semantic relations or links discussed in this abstract.

One can also make use of the same set of properties to define other relations of the two objects, e.g., to quantify how many times an object is longer than another.

Definition 8 *For any objects $o_1, o_2 \in O$, $times\_longer(o_1, o_2, n)$ iff $length(ts_1, o_1) \in Prop(o_1)$ and $length(ts_2, o_2) \in Prop(o_2)$ such that $n = ts_1/ts_2$.*

In Definition 8, the term n indicates how many times o_1 is longer than o_2.

One particular type of semantic networks under much study is the inheritance hierarchical network which is based on parent-child relations such as *isa* and *isn't* [5]. Two of such inheritance relations *isa* and *disj*, for example, can be specified as level 2 relations in the following manner:

Definition 9 *Inheritance relations*
$isa(o_1, o_2)$ *iff* $Prop(o_1) \subseteq Prop(o_2)$
$disj(o_1, o_2)$ *iff* $Prop(o_1) \cap Prop(o_2) = \emptyset$

Basically, an *isa* relation defines the set-inclusion relation between properties of any two objects. A *disj* relation defines the set-exclusion relation between two objects that cannot be defined using an *isa* relation. It is worth noting that an object's name is not inheritable. From Definition 9, one can easily derive that *isa* is reflexive, transitive, and asymmetric while *disj* is symmetric; henceforth, I will take $disj(o_1, o_2)$ to mean either $disj(o_1, o_2)$ or $disj(o_2, o_1)$ in the discussion.

Note that another potential important relation *part_of* does not lend itself to be treated in the set theoretic analysis. Indeed, there still lacks a satisfactory theory of mereology (the relations of parts to wholes). Most schemes of semantic network simply consider a complex node as a composite of its constituent nodes. The problems with such kinds of definitions, however, are many.

First, an obvious problem is that the same constituent objects can form different composite objects of distinct properties. Consider an example from molecular chemistry. If one says that a methane object is nothing more than a mereological sum of hydrogen, carbon, and bonding, then one cannot explain, for instance, the difference between methane and butane. Butane molecules are also made up of a hydrogen (ten) and carbon (four) atoms bonded (thirteen times) in a particular configuration. Secondly, certain properties exhibited by a composite object may never be explained by accumulated properties of its subparts. Properties of methane, such as its freezing point and ionization energy, cannot be explained in terms of accumulated properties of its subparts but must be explained by their interactions according to some chemical laws. Thirdly, there often exists conflicting properties between subparts. In this scheme, *part_of* is an intrinsic relation, while the behavior of a composite object is the product of the interaction of its subparts instead of the accumulated properties of them.

Level 3 relations are defined in terms of level 2 relations. For example, the relation *closer* states that an object is closer in length to a second object than it does to a third one and can be defined in terms of *times_longer* relation as follows.

Definition 10 *For any $o_1, o_2, o_3 \subset O$, $closer([o_1, o_2], [o_1, o_3])$ **holds** iff $times\_longer(o_1, o_2, n_1)$ and $times\_longer(o_1, o_3, n_2)$ hold in M such that $n_1 < n_2$.*

It is expected the set of levels 2 and 3 relations in C would vary from applications to applications. Nevertheless, the principles of constructing semantic relations discussed here is generic enough for a variety of applications.

5 Integrity Constraints

Integrity constraints on the semantic relationships (or links) among objects are important in both ensuring the correctness of the network model and deducing implicit relations between objects. For example, from $part\_of(o_1, o_2)$ and $part\_of(o_2, o_3)$, it follows that $part\_of(o_1, o_3)$, as *part_of* satisfies the constraint of transitivity. This section presents a set of constraints of the two ***inheritance*** relations defined in Section 4, *isa* and *disj*, to exemplify the function of integrity constraints in S. For any $o, o_1, o_2 \in O$,

Constraint 1 $isa(o, o)$

Constraint 2 $isa(o_1, o_2) \wedge isa(o_2, o_3) \rightarrow isa(o_1, o_3)$

Constraint 3 $disj(o_1, o_2) \wedge isa(o_1, o_3) \rightarrow disj(o_2, o_3)$

Constraint 4 $disj(o_1, o_1) \rightarrow disj(o_1, o_2)$

Constraint 5 $disj(o_1, o_1) \rightarrow isa(o_2, o_1)$

One can easily derive this set of constraints from Definition 9. Let us denote D as a set of isa and $disj$ relations for pairs of objects in M and $r(o_1, o_2)$ as either $isa(o_1, o_2)$ or $disj(o_1, o_2)$. The set of all deductive consequences of D using the inheritance constraints (Constraints 1 to 5), i.e., $\{r(o_1, o_2) : D \vdash r(o_1, o_2)\}$, is written as $Cn(D)$ and is called the inheritance set of M.

In addition, the set of inheritance constraints is *adequate* for a class of inheritance relations when all relations belong to that class in M are members of $Cn(D)$. This notion of adequacy is important as one would like to have an inheritance set which includes all possible inheritance relations derivable in a network model. In what follows, I introduce the notions of inconsistency and redundancy of M.

Definition 11 *Consistency*
For any model M, any $o \in O$, if $Prop(o) = \emptyset$ or $disj(o, o)$, then o is inconsistent; and if M contains an inconsistent object, then it is inconsistent.

Definition 12 *Redundancy*
For any model M, any $o \in O$, if there is some other object $o_1 \in O$ such that $Prop(o) = Prop(o_1)$, then o is redundant; and if M contains a redundant object, then it is redundant.

The arrival of new information during the interaction with the human users and other knowledge base systems may force the local system to change its abstraction about the physical world. This implies that the set of inheritance relations and objects of the underlying network model has to be updated or modified. Thus, checking the integrity of a model during the operation is necessary. In below, I state, without proof, a list of statements that are used in DOOS to detect the inconsistency and redundancy of M during an update. One can refer to [6] for the proofs, which are derived from the definitions of inheritance relations and constraints.

For any model M, any $o, o_1, o_2, o_3, g_1, g_2 \in O$,

1) o is inconsistent if $disj(o, o) \in Cn(D)$.

2) o_1 is redundant iff $o_2 \neq o_1$, such that $isa(o_1, o_2), isa(o_2, o_1) \in Cn(D)$.

3) (a) Constraints 1 and 2 are adequate for the derivations of isa in M.
   (b) Constraint 4 is adequate for the derivations of disjointness relations in M.
   (c) The inheritance constraints are adequate for the join class of isa and $disj$ relations in M.

4) (a) If M is consistent, then $disj(o_1, o_2)$ can be derived by applying Constraint 3 only.
   (b) If M is inconsistent, then $disj(o_1, o_2)$ can be derived by using Constraint 3 and Constraint 4 in the last step.

5) (a) If M is consistent, then $isa(o_1, o_3)$, $o_1 \neq o_3$, can be derived by applying Constraint 2 alone.
   (b) If M is inconsistent, then $isa(o_1, o_3)$, $o_1 \neq o_3$, can be derived by (i) steps in Statement 5.a, or (ii) intermediate steps in Statement 4.b and Constraint 5 with or without Constraint 2 in the last steps.

6) (a) If M is consistent and $disj(o_1, o_2) \in Cn(D) - D$, then $disj(o_1, o_2)$ can be derived by the following relations: (1) $disj(g_1, g_2)$; (2) $isa(g_1, o_1) \in Cn(I)$ (deductive consequence using only isa relations); (3) $isa(g_2, o_2) \in Cn(I)$

(b) If M is inconsistent and $disj(o_1, o_2) \in Cn(D) - D$, then $disj(o_1, o_2)$ can be derived by the following relations: (1) $disj(g_1, g_2)$; (2) $isa(g_1, o_1) \in Cn(I)$; (3) $isa(g_2, o_1) \in Cn(I)$.

(c) If $o_1, o_2 \in Class(o)$ and $disj(o_1, o_2) \in O$, then $disj(o, o)$.

The implication of Statement 3 is that the set of inheritance constraints is compact such that if any relation $r(o_1, o_2)$ belongs to $Cn(D)$, then $r(o_1, o_2)$ is a consequence of finite subset of D. Statements (4-6) are concerned with using a subset of the inheritance constraints to derive relations of particular class of relations, i.e., isa or $disj$, in $Cn(D)$ of any M.

In addition, Statement 6.c says that for any consistent network model, every class in the model must not contain any disjointness relation for any two of its objects. Such a restriction, however, does not apply to objects of different classes. This property can be used together with the following statement to ensure that the original design intent of the classification of objects will not be violated in the later modification of the network model. Let me define that any object in a class with no descendant as a leaf object, then the statement is:

7) If all leaf objects of $Class(o_1)$ are disjointed from all leaf objects of $Class(o_2)$, then all objects in $Class(o_1)$ are disjointed from those in $Class(o_2)$.

6 Revision Rules

Revision rules in R are concerned with various forms of revision of knowledge base such as updates of the set of properties in objects, the generation and removal of objects, the modification of semantic links, and so forth. An update transaction is executed only when a request is issued and its prerequisite is satisfied; otherwise, it is aborted.

A representative example about the revision of object properties is provided here. There are three kinds of operations for updating properties of an object: the addition of a new property, the removal of a property, and the revision of an existing method. Addition and removal of properties are primitive operations. The revision of a property in an object is a composition of removal and addition operations, i.e., to revise an existing property $p_i(ts, o)$ of an object o in its model, the old property must first be removed and a new property of different value is then added. The rule for property addition is stated as follows.

Rule 3 *Property addition*
$request\_add\_property(name(o), p(ts, o)) \wedge add\_prop\_condition(name(o), p(ts, o))$
$add\_property(name(o), p(ts, o))$

This rule may be read as: "If a request for adding a property $p(ts, o)$ of an object o has been asserted and the condition or prerequisite of adding that property is satisfied, then one can infer $add\_property(name(o), p(ts, o))$."

The prerequisite $add\_prop\_condition(name(o), p(ts, o))$ is true only when (a) object o exists; (b) $p(ts, o)$ does not already exist (unique property condition); (c) no other property of o has the same label (unique determinable condition); and (d) no other object o_1 with $isa(o_1, o) \in Cn(D)$ such that $Prop(o_1) = Prop(o) \cup p(ts, o)$ (redundancy checking). In addition, some of the revision rules defined in R also require the use of Statements (1-7).

7 Concluding Remarks

In this abstract, I have discussed key features of a new, formal scheme for semantic network representation. The underlying concept of this scheme is the three-level structuring of knowledge. The concepts of object-orientation and logical inference are also blended, coherently, into this network scheme. The three-level structure of knowledge offers a rational and systematic means to define nodes (objects) and links (relations) of network models. This, subsequently, enables the system designer of a network model to derive formal properties of the model as well as to better predict its behavior of information processing. The presented scheme has been implemented in DOOS knowledge representation system.

References

[1] ATLSS Center, *Sixth-year renewal proposal to the NSF*, Vol. 2: Projects, Publications, and Biosketches, Lehigh University, Bethlehem, PA, 1992, Section 7.

[2] J. Bigelow and R. Pargetter, *Science and necessity*, Cambridge University Press, Cambridge, 1990.

[3] G. Frege, (translated by J. L. Austin) *The foundations of arithmetic: A logico-mathematical enquiry into the concept of number*, second edition, Oxford, Blackwell, 1959.

[4] J. F. Sowa, ed., *Principles of semantic networks - explorations in the representation of knowledge*, Morgan Kaufmann, San Mateo, CA, 1991.

[5] D. S. Touretzsky, *The mathematics of inheritance systems*, Morgan Kaufmann, Palo Alto, CA, 1986.

[6] S. T. C. Wong, *A framework for the description and design of cooperative knowledge-based systems*, Doctoral dissertation, CSEE Department, Lehigh University, PA, 1991.

EXPANSION AND SUCCESSION IN THE MERM MENTAL REPRESENTATIONS MODEL

Alfredo M. Maeda **Jun-ichi Aoe**

Dept. of Information Science and Intelligent Systems
University of Tokushima
2-1 Minami Josanjima Cho. Tokushima-Shi 770, JAPAN
E-mail. maeda@j-aoe.is.tokushima-u.ac.jp

Abstract

Among the diverse knowledge representation formalisms, those with cognitive scientific bases have proven to have high expressive power. Such formalisms try to represent aspects of the world as cognized by humans. Within the human cognitive capabilities, the psychological phenomena of succession and expansion have been barely addressed in knowledge representation systems. Expansion helps explain why humans access small, limited pieces of knowledge at a time. Succession refers to the order of occurrences of events. An analysis of these two phenomena and their specification within a representational formalism, called MERM, is presented in this paper. MERM allows the representation of mental models as cognized by humans. The representation of the expansion and succession phenomena are useful in diverse applications, such as natural language understanding, tutoring systems, and perceptual robotics.

1. Introduction.

The degree of perception/action human beings have with the external world depends mainly on the capability of the *mind-brain* to process information. Perhaps the most important of the information processing capabilities of the mind-brain is the capability of knowledge [Lara 1987]. In this concern three important philosophical bases should be taken into account [Lara 1990].

1) The belief in the existence of an *external reality*, which exists independently of the fact of being or not perceived.
2) Direct access to the *reality perceived* through the senses.
3) A *theoretical reality* to describe the *physical reality* where aspects of the world such as matter, energy, etc. appear.

From the philosophical standpoint, to know how external reality is represented within the mind-brain is not the core issue [Poncairé 1984]. The important issue is the agreement of individuals on one same physical object when making reference to it, independently of the way the object is represented within their brains. In other words, the same *mental models* of aspects of the world exist within the mind-brains of human beings.

The inert structures operated by means of cognitive processes are called *representations*, or declarative knowledge [Stillings 1987]. The actions to be taken in order to achieve a goal are referred to as *processes*, or procedural knowledge. Although the theory that declarative knowledge [Anderson 1976, 1983; Anderson and Bower 1973] is a network of propositions that allows the representation of specific facts, account is not taken of the general knowledge that allows a system to understand such facts in the first place. For example, to represent an apple pie, the storage of an atom or relation *apple_pie* is insufficient. The creation of a mental model that actually corresponds to an apple pie is mandatory. Mental models are constructions of aspects of the world which have a *relation-structure* similar to the aspects they represent [Rogers et al. 1992]. Since mental models can be manipulated, inferences over them are possible.

2. Related Research.

As our insights on human cognition increase, the development of better representational formalisms also increases. Previous well-known knowledge representation theories—such as Conceptual Dependencies [Schank 1972, 1975; Schank and Abelson 1977], Frames [Minsky 1975], Semantic Memory [Quillian 1966, 1968], and Cognitive Representation Theory [Wilensky 1987]—have helped account for some psychological phenomena, such as recalling and explanation.

These theories, with exception of Conceptual Dependencies, share the capability to generate hierarchic taxonomies with inheritance. Conceptual Dependency structures are language-independent. Apparently the majority of commonalties of these theories lie in their inconveniences. Ambiguities in the definitions of the theories and difficulties in the representation of quantification and definiteness [Brachman 1979; Maeda 1992a,b; McCarthy and Hayes 1969; Wilensky 1987]. However important, little attention has been paid to the psychological phenomena of *expansion* and *succession*.

Expansion refers to the way humans take into account aspects associated to a certain circumstance only when those aspects need to be considered. MERM's expansion label is a procedural attachment method that resembles *demons* [Charniak 1974; Minsky 1975]. Succession refers to the order in which things are to be considered or done. A description of these phenomena and their representation in MERM (MEntal Representations Model) is given in this paper.

Approaches to solve different problems concerning temporal relations have been developed. Some notable work on reasoning about temporal logic is, for example, time maps [Dean & MacDermott 1987], interval algebra [Allen 1983], and point algebra [Vilain & Kautz 1986] among others. Succession in MERM is independent of time. It is the order of events, and not the moment in time when events take place, what is of importance in MERM's succession label.

2. Brief Description of MERM

MERM (MEntal Representations Model) is a theoretical model for the representation of aspects of the world in machines. It is the result of research in Cognitive Psychology, AI, Computational Linguistics, and Philosophy. MERM helps account for some psychological phenomena. Mental models represented in MERM do not consist of encyclopedic knowledge or formal definitions, and are independent of linguistic syntactic constructions. Linguistic and non-linguistic aspects of the world are representable. The structure of MERM is similar to that of semantic networks and cognitive representation theory. The substantial difference lies in the different repertory, and semantics, of primitives that conform the model.

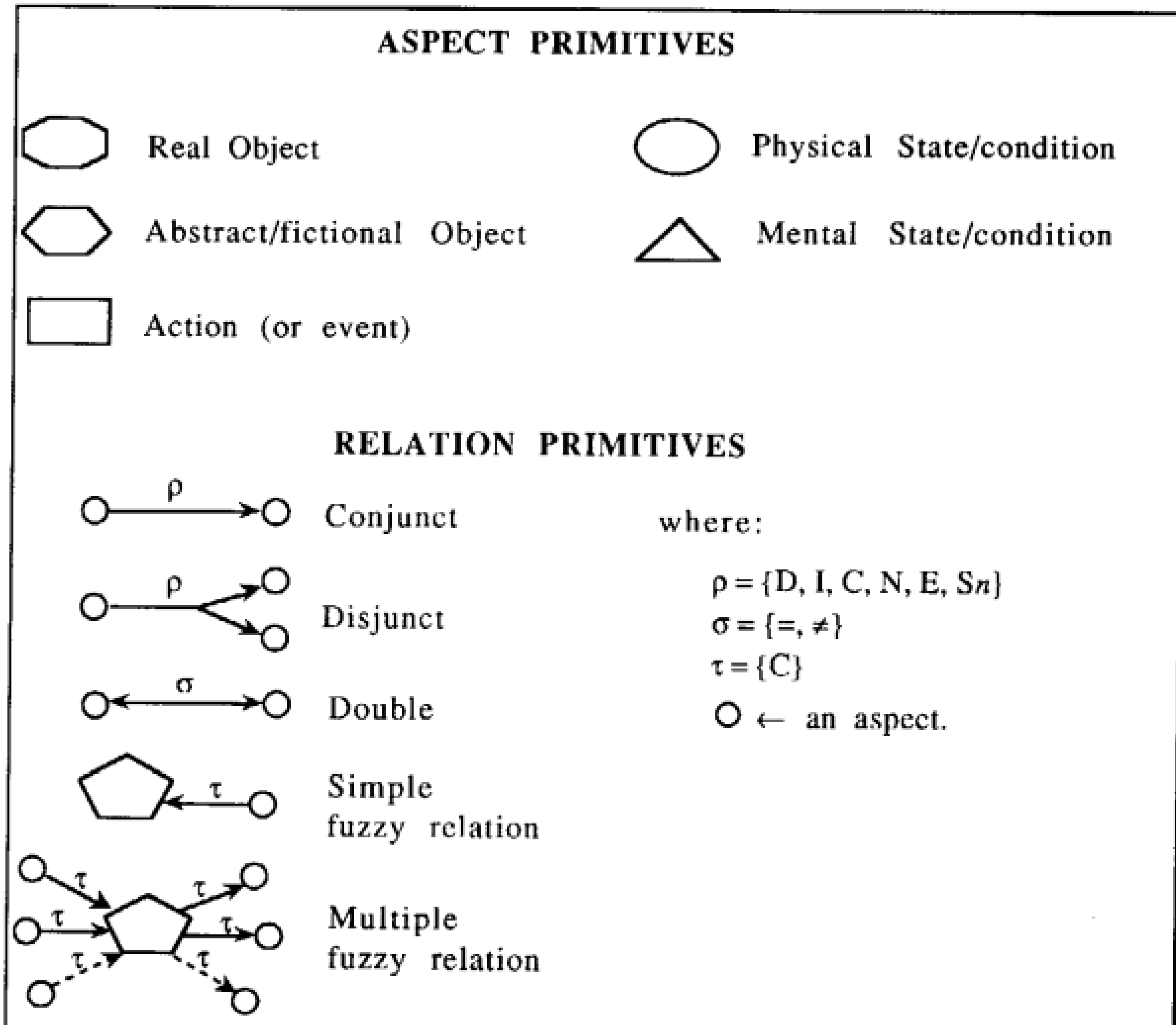
MERM consists of one set of *primitive aspects* and one set of *primitive relations* (see Fig. 1). These sets of primitives are combined to conform representations of aspects of the world.

2.1 The MERM Aspect Primitives.

An aspect represents an individual, identifiable entity of the world. Dog, toy, run, and anger are examples of aspects. It can be very specific or rather general, depending on the level of fine-grain desired by knowledge base designers. It is possible to specify, for example, a car as an aspect, or decompose it into more detailed aspects such as engine, tire, door, etc.. The meaning of an aspect of the world and the inferences about it are implicit in the representations.

Five different primitive aspects have been defined in MERM, as shown in Figure 1. Objects such as mouse and book are *real objects*. Numbers in general are *abstracts*. Prometheus and tweety are examples of *fictional objects*. Swim and eat are *actions*. Hungry and sick are considered as *physical state/conditions*. Sad and happy are represented as *mental state/conditions*. No restriction exist in the associativity of the different aspects in order to form representations.

The five *aspect* primitives provide a fine-grained classification of aspects. This is useful in the study of psychological phenomena and other cognitive processes because it allows the representation of human cognition on computers more accurately. However, the diversity of aspects results rather cumbersome for designers when building practical applications. In fact, the current trend in the development of knowledge



where:

$\rho = \{D, I, C, N, E, S_n\}$
$\sigma = \{=, \neq\}$
$\tau = \{C\}$
O ← an aspect.

LABELS

D	← dominate
I	← instantiate
C	← constraint
E	← expansion
S_n	← succession
=	← equality
≠	← inequality

$N[D|Bl|Bu]$ ← number/amount

where:

$D \in \Im$	← default value
$Bl \in \Im|\_$	← lower boundary
$Bu \in \Im|\_$	← upper boundary
$\Im$	← set of Integer numbers
_	← anonymous variable

Figure 1. Definition of MERM, the MEntal Representations Model

representation systems is not generalization but specialization. This trend has been used in a number of systems, such as the CLASSIC system [Brachman et al. 1992].

In MERM the usage of the five *primitive aspects* is not mandatory. For practical purposes it is possible to reduce the set of *primitive aspects* to only two, namely *objects* and *events*. The process is rather simplistic. Both *real* and *fictional/abstract objects* can be reduced to only one simple representation of objects and represented as, to say, hexagons. Similarly, *physical* and *mental state/conditions* can be considered as *events* (or actions) and represented within rectangles. This simplification avoids confusion in the decision of which *primitive aspect* to use in the representation of a given aspect of reality. Thus, the painstaking process of creation of knowledge bases is reduced significantly.

2.2. The MERM Relation Primitives.

Aspects are correlated by means of *relations*. A relation consists of a link connecting aspects. The resulting network provides a mental model of a concept. Following is a description of these relation primitives.

CONJUNCT RELATION. The most commonly used relation is the *conjunct* relation (C), which links by means of a simple arrow two *aspects* whose relation is specified with a ρ *label*[1]. A mental model for the concepts running, jogging, walking, and hiking is represented in Figure 2. All the conjunct relations emanating from an aspect must be considered, accordingly with their respective labels, to obtain the meaning the concept being represented. For example, the four 'C' relations attached to the action running have to be analyzed to understand what is being represented about running..

DISJUNCT RELATION. A *disjunct* relation implies that a given aspect is *ρ-related* to only one of the group of aspects addressed. The *disjunct relation* between running and both competition and necessity is represented with a "branched" relation in Figure 2. This relation states that the assertion of one of the related states is enough to validate running.

DOUBLE RELATION. The representation of unidirectional relations, where one *aspect* affects the other participant aspect in the relation is possible by means of the conjunct and disjunct relations. However, there are cases where relations that affect the two participant aspects are necessary. Such relationship is represented in MERM by means of the *double relation*. The type of double relation is established by a σ label. Figure <<>> in section <<>> contains a double relation with an equality label on it.

[1] cf. seccion four.

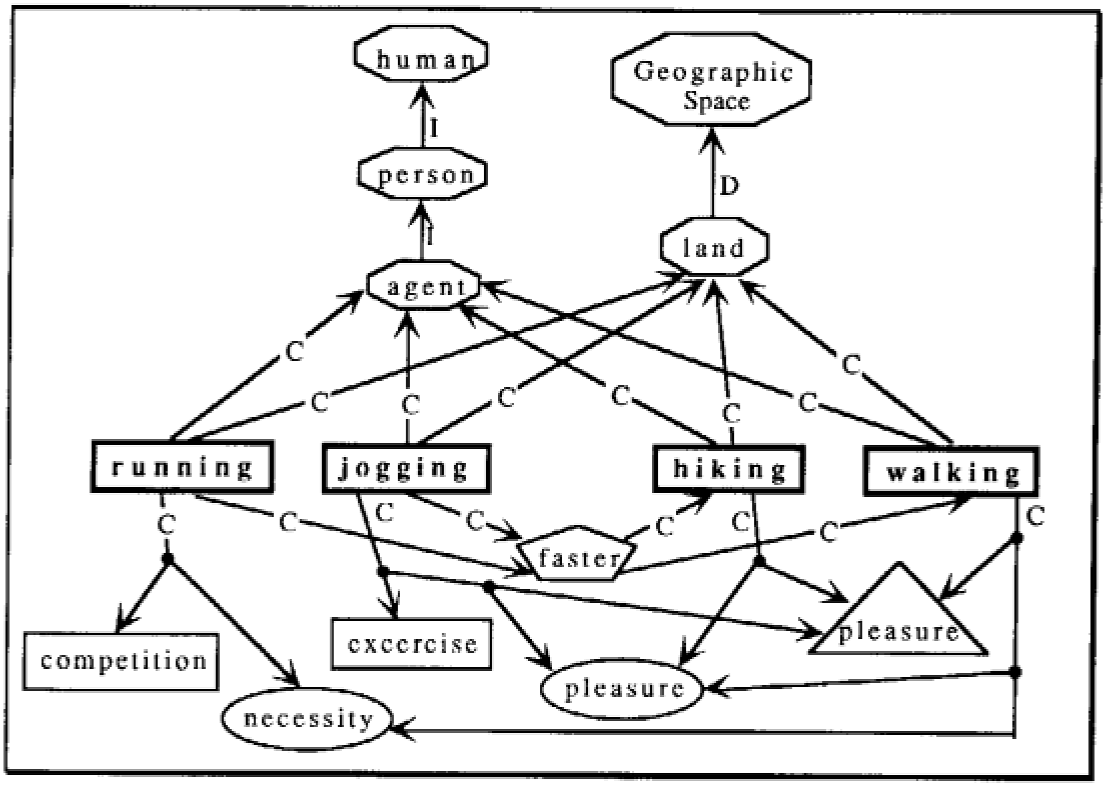


Figure 2. MERM representation of running, jogging hiking and walking.

SIMPLE FUZZY RELATION. Cases exist when the specification of a fuzzy concept is necessary to describe concepts or events. The *simple fuzzy relation* is used when, for example, imprecise quantifiers need to be attached to an aspect. Simple fuzzy relations are represented within pentagons, and are called *simple* because only input links are attached to it. This means that simple fuzzy relations are not affected by the environment. For practical purposes, tables or membership functions such as those defined by Zadeh [1879, 1983, 1989], or those shown in [Kandel 1991] can be used to define *fuzzy labels*.

MULTIPLE FUZZY RELATION. Definiteness in the relation between several aspects within a representation may exist. When a statement such as "Running is faster than walking" is being interpreted, the precise difference in speed between running and walking cannot be stated since fast is a word with a vague meaning. Multiple fuzzy relations are represented within pentagons—as in the case of simple fuzzy relations. The difference lies in that multiple fuzzy relations also have inputs.

A multiple fuzzy relation can consist of a non-linguistic function such as, for example, a function of three variables—*f(x,y,z)*,—where each variable corresponds to a link associated to the fuzzy relation. A table or a membership function might be used to define the kind of fuzzy relation [Bouchon-Meunier 1992; Kosko 1992; Hall & Kandel 1992].

4. THE RELATION LABELS

Eight *relation labels* have been defined in MERM to provide relations with a specific meaning. These labels are defined as ρ, σ and τ labels in Figure 1. Their description is given below.

DOMINATE LABEL. The correspondent to *is-a links* in semantic networks. It is represented with the character *D*. Dominate establishes hierarchies with inheritance of properties between aspects.. In Figure 2, land is dominated by geographic-space, indicating the hierarchical dependence of land. All the properties of geographic-space are inherited by land.

THE INSTANTIATE LABEL. Denoted with the character *I*, indicates that the relation between aspects of a representation and aspects of a particular situation are instantiated. For example, in Figure 2, person is instantiated to human, indicating that the former is a particular instance of the later.

CONSTRAINT LABEL. It is used to representations restriction or condition between aspects. Represented with the character *C*, The *constraint label* indicates that certain aspect α needs, in order to be valid, another aspect β ρ-related to it. That is, the validity of the aspect α depends on the validity of the aspect β (see Figure 2).

EQUALITY AND INEQUALITY. The *equality* between two aspects is established by means of the *equal label*, represented with the symbol "=". This label is applicable only to double relations. The *inequality* between two aspects is obtained by means of the *inequality relation*, represented with the symbol "≠". These two labels are semantic opposites. For example, some children consider the relationship between spoon, fork and knife to be as depicted in Figure 3.

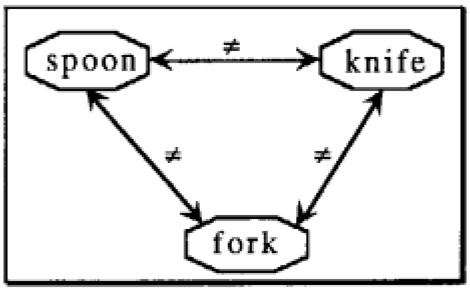


Figure 3. Relation between spoon, fork and knife.

NUMBER/AMOUNT. Objects of the world are often described in term or their components. Such components are essential in the representation of the objects themselves. For example a common answer to the question "what is an elephant?" is: an elephant is a big, brown animal with four legs, big ears, and a long nose. Note that since not all animals have four legs, the number of legs is provided explicitly. This is not the case for ears or nose because it is well-known that animals have, in general, two ears and one nose. The same criteria is used to compare objects in the world, such as how to differentiate between a tricycle, a bicycle, a carriage, and a car.. These components of objects are essential in the representation of the objects themselves.

The specification of number and amount of aspects in MERM is obtained by means of the *number/amount label* (see Figure 1). Variations in the default quantity are controlled by the boundary values. An anonymous variable means that any value is valid. It can be placed in any and as many arguments as necessary. A simplified representation of carriage and automobile is shown in Figure 4. The number of wheels in a carriage is four. Note that since the three parameters in the label have the same value, no variations in the amount of wheels is allowed. For the case of automobiles, four wheels are the typical case. However, automobiles with three or more wheels are considered as valid in the representation. A more important difference is the presence of an engine in automobiles and the absence of it in carriage. In general, number/amount labels allow the representation of quantities with closed or open boundaries.

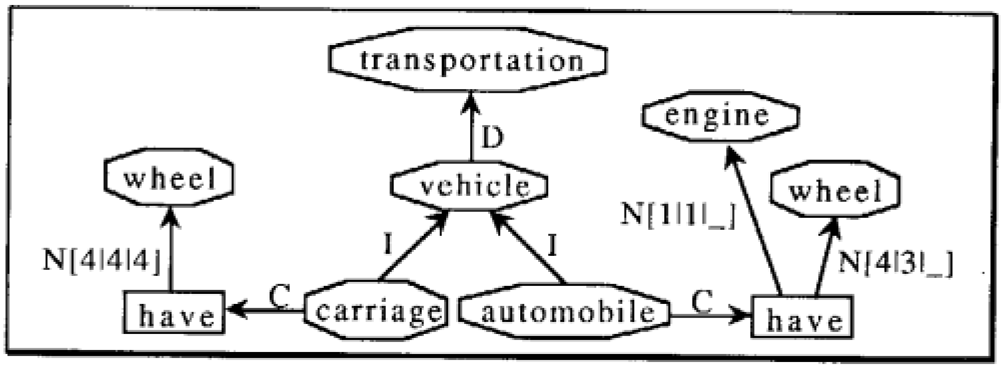


Figure 4. An example of the usage of the number/amount label

5. EXPANSION

Recalling is a psychological phenomenon that has been taken into account by a number of researchers in the development of theories of cognition based on language. Schank [1972, 1975; and Abelson 1977] provided, by means of his Conceptual Dependency theory, an explanation of the reason why people remember discourse contents instead of literal sentences. Quillian [1966, 1968] explained *recalling* by showing how complex concepts, compared with simple concepts, take more time to be recalled. A similar problem, known as partitioning, was analyzed by Hendrix [1975], who proposes a partitioning of semantic networks into units called *spaces*.

Hendrix's partitioning concept is used for different purposes such as quantification and the consideration of hypothetical situations. Another psychological phenomenon closely related with recalling is a phenomenon that I call *expansion*.

To discuss *expansion*, consider the following situation. Yoshi is getting ready to attend a wedding in the early afternoon. Once on his way to the wedding, Yoshi starts getting so hungry that he decides to stop at a restaurant and have some early lunch. That way he'll avoid either been fainting or having his stomach making strange sounds, which would disturb the wedding ceremony. When thinking about eating, Yoshi has in mind only the idea of ingesting food in order to satiate his hunger—of course eating includes both food and beverage. Some Kentucky Fried Chicken and Pepsi seem to be a good choice. But then, from Yoshi's mind pops out that fried chicken is eaten with the hands. That implies a high risk to make a mess and put some oil spots on his tuxedo, not to mention other inconveniences that result from eating such food. Yoshi finally decides to eat some fast food at a Chinese restaurant. He is good at using chopsticks so he won't make a mess and will make it on time for the wedding.

In what way did the decision process take place? Yoshi has in his mind a conception of what eating is and implies. However, not all the aspects linked to eating are accessed and used every time people think about eating. A base cluster of aspects concerning eat are always present when people think about eating. But there are also other aspects which are accessed only if a particular situation requires it. That is the reason why Yoshi did not think about hands and chopsticks until it was necessary to do so. Representations of aspects of reality exist within the human mind. When a mental representation is being accessed by a person, only the portion of the representation that is significant for the current situation is taken into account. The remaining portions of the representation are accessed only when needed

Expansion can be informally defined as *"the eventual consideration of non-primordial portions of an aspect of reality."* The *eventual* in the definition means that expansion takes place in an as-necessary basis. *Expansion* is represented in MERM by means of the *expansion label* (E) and can be used in ρ-relations.

At a restaurant, when we ask for steak we do not think beforehand that knife and fork are needed or if we have them on the table. This is simply because we assume they will be there by the time the food is served. However, if once being ready to start eating we realize that, to say, the fork is missing, then we make conscious of it and request one. Under a similar basis, a representation of the concept eat can be like the one shown in Figure 5. With that representation, a computer can explain that eating consists of the ingestion of food carried out by a person by means of the mouth. However, a representation for eating should also consider tools for eating as well as drinks as part of the eating event. Since these are not primordial parts of the concept they can be considered as extensions. In that case the representation would be like the one depicted in Figure 6. That way, to perform inferences about how different kinds of food are to be eaten—e.g., Mexican or Chinese food,— it becomes important to consider the tools to be used. It is then when expansion takes place. Once the aspect for tools has been activated, its corresponding representation in the context is attached to the concept of eating.

Similarly, beverages are brought into account during an eating action only if they are not at hand. This is again because the idea that eating implies drinking too is taken for granted. But if this scheme does not succeed then the representation is expanded to the inclusion of drinking as an action that takes place together with eating (see Figure 6).

Machines with representations implemented in MERM can access the different portions that conform those representations in a way similar to that of human beings. This is a characteristic of MERM that allows machines to emulate human-like behavior in the issues of *recalling* and *expansion*.

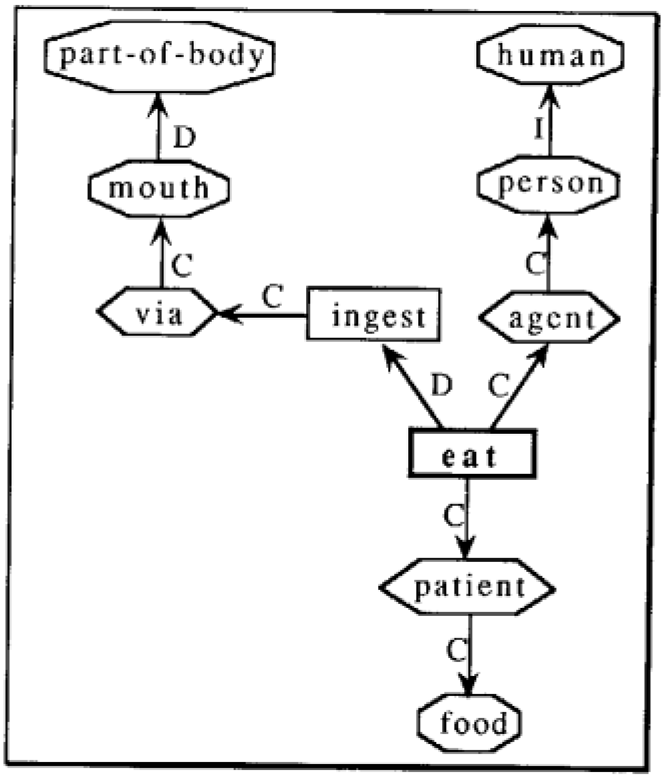


Figure 5. Simple representation of eat.

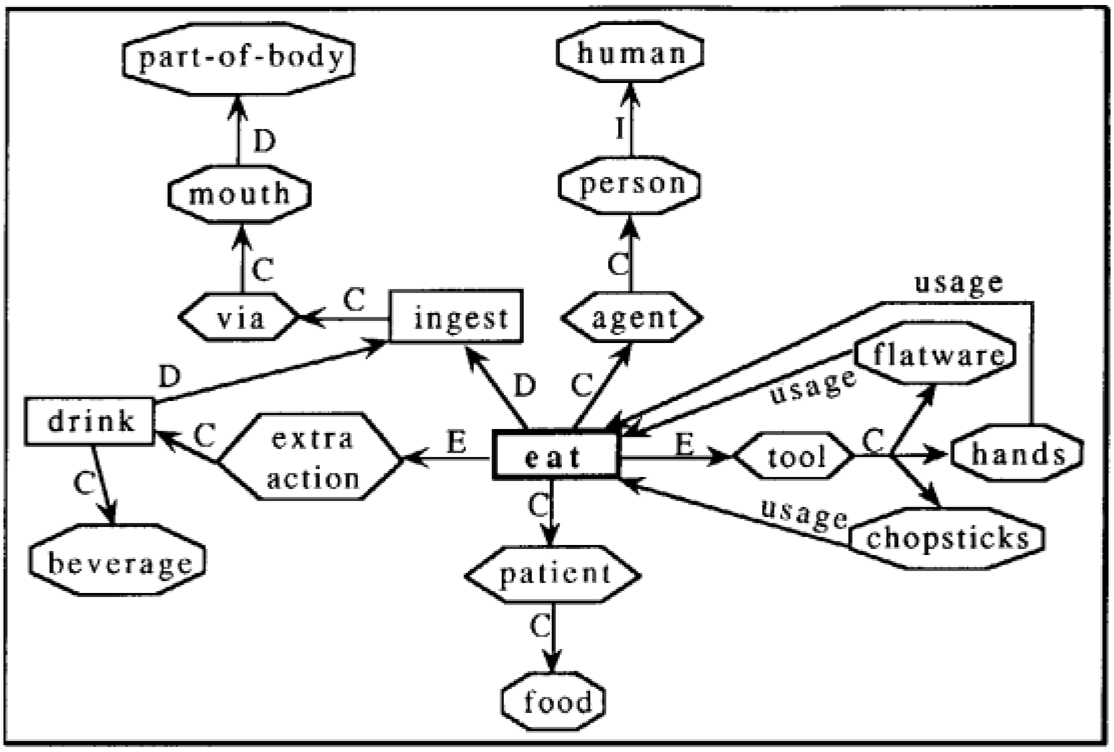


Figure 6. Expansion labels attached to the concept of eating.

6. SUCCESSION

Actions in a real situation have an execution order. Representational models that do not include succession of events in actions cannot represent the manner in which an action takes place in a space-time continuum [Maeda 1992]. The consequence is poor inferencing capability when dealing with actions.

MERM has three different was to deal with the problem of time.

1) The representation of fixed dates and time by usage of ρ *relations*. A representation for the sentence "Yoshi will go to Hawaii tomorrow" can be as depicted in Figure 7.

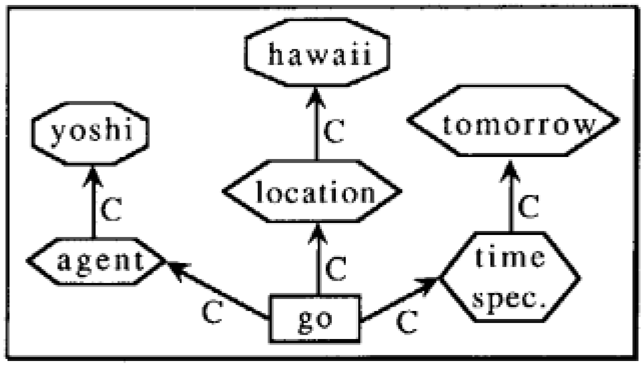


Figure 7. Representation of *"Yoshi will go to Hawaii tomorrow"*

2) The representation of non-specific, or indefinite, time in a given situation by means of *fuzzy relations*. In Figure 8, a representation for the sentence "Patty visited Stacy a few days ago" is depicted. Note that few is a fuzzy quantifier. Its definition can be specified by using a membership function.

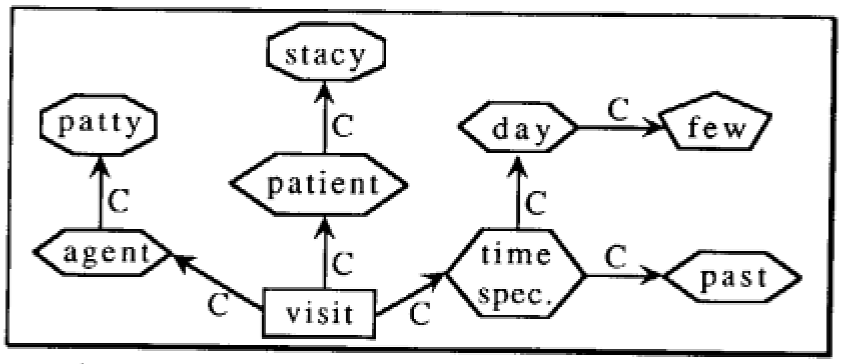


Figure 8. *"Patty visited Stacy a few days ago."*

3) The representation of the order in which an event or a series of events take place are represented by means of a special label called *succession label*. The *succession label* is the main discussion in this section.

A *succession label* is represented with an *Sn.* label—see Figure 1,—where *S* is the *succession* identifier and *n* is the sequential order, or succession, of the event. Different portions of an event may take place simultaneously, hence *Sn.*'s with the same value are permitted. The usefulness of succession lies in that it allows the representation of complex procedures step by step. The hole process of an assembly plant can be represented at different levels of detail.

Consider the concept of eat shown in Figure 6 above. Succession labels are necessary to describe the different stages of the action of eating. A simple example of the action eating, with succession labels included, is depicted in Figure 9. According to that representation, the initial state of the action eat consists of having food available and a hungry person—the agent of the action—who starts the eating process. Note that even if there is food available, the action does not take place if the agent is not hungry. The process of eating finishes when either the agent is satisfied or when there is no more food available. Obviously, the food the agent had available at the beginning of the process must be the same as that which he/she finished afterwards. This condition is specified in the representation by means of equality labels. If the succession of events is not specified, a machine may then think it is perfectly normal to finish eating before actually starting to eat. A representation with *succession relations* specified whenever necessary helps a machine behave more naturally and perform accurate inferences about situations.

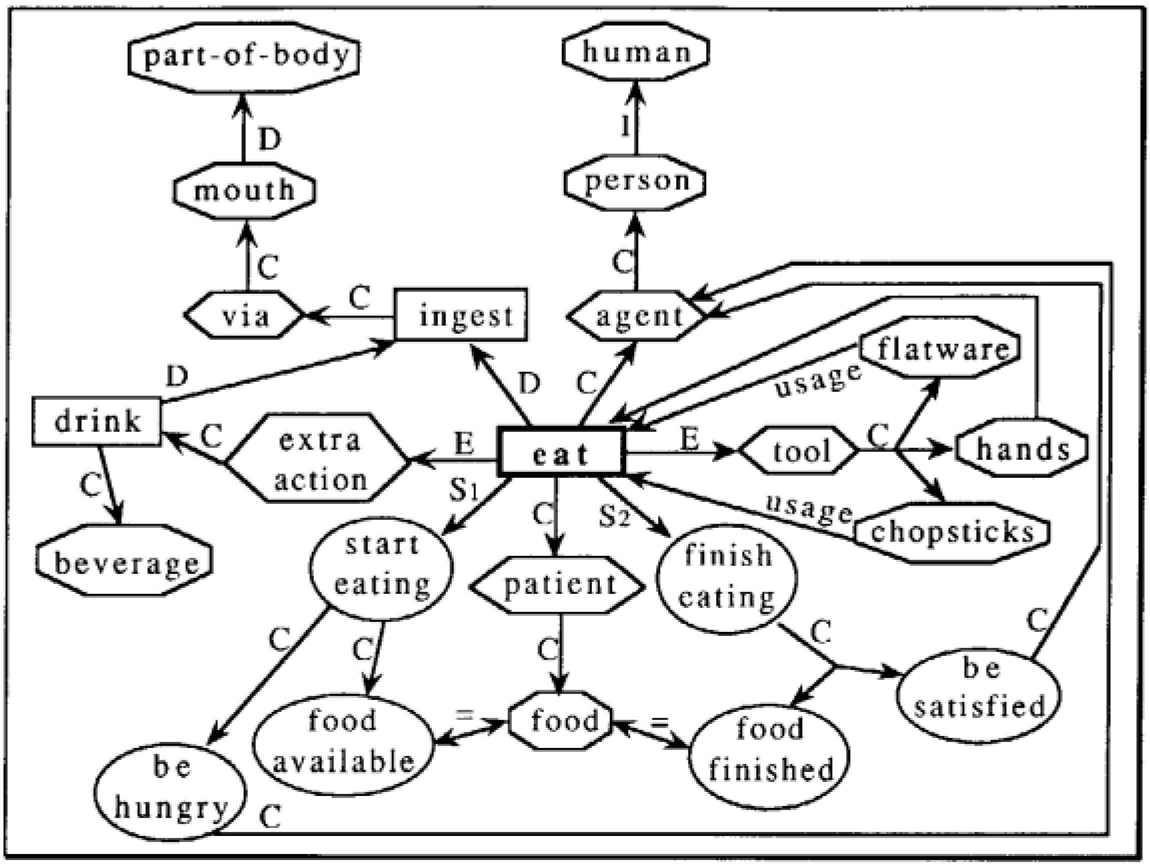


Figure 9. Succession labels represent the order in which an event takes place.

7. APPLICATIONS

Expansion and *succession*—MERM, in general—help give account for psychological phenomena and serve as a tool for the test of cognitive theories. They are applicable in virtually any area of AI. In tutoring systems, expansion and succession provide more human-like behavior in machines, independently of their role (tutor, student, etc.). The representation of these phenomena allows a precise description of sequences of actions necessary in learning or tutoring.

In natural language understanding systems and human-machine intelligent interfaces, a more precise representation of linguistic and non-linguistic aspects of reality can be obtained. This allows machines to perform accurate inferences over aspects of the world—real or artificial.

In perceptual robotics, the consideration of new or diverse situations can be attained to provide robots with more accurate inferencing capabilities and adaptability to new situations. The sequence of actions to be performed by a robot can be implicit within the representation of the object to be assembled or in the representation of the assemblage process.

6. CONCLUSIONS

Cognitive scientific bases ought to be considered in the development of representational theories. This is necessary in order obtain sound representations that agree with human cognition. The representation of human cognitive capabilities helps us in the understanding of the human mind-brain as well as in the development of machines with higher degree of intelligence.

Representations in MERM correspond to a description of the way individuals conceive aspects of the world. Thus representations are tailored according to different mental models. This allows MERM based systems to behave in a way relatively close to human behavior.

Expansion is very important in the emulation of human cognition and the explanation of some psychological phenomena, such as recalling and remembering. Humans access knowledge in a way analog to expansion. Succession has been considered in previous representational theories. However, we think our representation is more natural due to it has no direct relation with time. Instead, succession considers only the order in which things take place in the world. Both phenomena can be applied in virtually any area or Artificial Intelligence because of their generality.

REFERENCES

Allen, J. F. 1983. Maintaining Knowledge About Temporal Intervals. *Comm. ACM* 26(11):832-843.

Anderson, John R., and Gordon H. Bower. 1973. *Human Associative Memory*. Winston & Sons, Washington.

Anderson, John R. 1976. *Language, Memory, and Thought*. Erlbaum, NJ.

Anderson, John R. 1983. *The Architecture of Cognition*. Harvard U. Press. Cambridge MA.

Brachman, Ronald J. 1979. On the Epistemological Status of Semantic Networks. In *Associative Networks: Representation and Use of Knowledge by Computers* (Nicholas V. Findler, Ed.). Academic Press, Orlando Florida

Charniak, E. 1974. Toward a Model of Children's Story Comprehension. *Ph. D. Thesis, AI-TR-266*. Artificial Intelligence Laboratory, MIT. Cambridge, Mass.

Dean, T. L., and D. V. McDermott. 1987. Temporal Database Management. *Artificial Intelligence* 32:1-55.

Hendrix, Gary G. 1975. Expanding the Utility of Semantic Networks Through Partitioning. Proc. Fourth IJCAI. pp.115-121. Tblisi, Georgia, USSR.

Lara, Rolando. 1987. *Cibernética del Cerebro.* CECSA. Mexico.

Lara, Nydia. 1990. *La Dinámica de la Estructura de la Percepción Como Herramienta Cognoscitiva Común de la Filosofía de la Ciencia.* Master's dissertation thesis. Instituto de Investigaciones Biomédicas, Universidad Nacional Autónoma de México, México.

Maeda, Alfredo M. 1992a. A Model of Mental Representations for Implementation on Automata. M. E. Dissertation Thesis. Dept. of Information Science and Intelligent Systems, University of Tokushima, Japan.

Maeda, Alfredo M. 1992b. Mental Models Representation. Proc. Int. Computer Science Conf., ICSC'92. Hong Kong.

McCarthy, J., and P. J. Hayes. 1969. Some Philosophical Problems From the Standpoint of Artificial Intelligence. *Machine Intelligence*, 4:463-502 (Meltzer, and Mitchie, eds.). Edinburgh Univ. Press.

Minsky, Marvin. 1975. A Framework for Representing Knowledge. *The Psychology of Computer Vision.* (P. H. Winston, ed.), McGraw-Hill, 211-277.

Poncairé, Henry. 1984. *Filosofía de la Ciencia*, (2nd. ed.). CONACYT. México. (original reference not available).

Quillian, M. Ross. 1966. *Semantic Memory.* Report AFCRL-66-189. Bolt Beranek & Newman, Cambridge MA.

Quillian, M. Ross. 1968. Semantic Memory. In *Semantic Information Processing.* (Minsky, Marvin. Ed.). MIT Press, Cambridge, Massachusetts.

Rogers, Yvonne, Andrew Rutherford, and Peter A. Bibby. 1992. *Models in The Mind: Theory, Perspective & Application.* Academic Press.

Schank, Roger. 1972. Conceptual Dependency: A Theory of Natural Language Understanding. *Cognitive Psychology*, 3:552-631.

Schank, Roger. 1975. *Conceptual Information Processing.* North Holland.

Schank, R., and R. Abelson. 1977. *Scripts, Plans, Goals, and Undrstanding: An Inquiry into Human Knowledge Structures.* Lawrence Erlbraum Associates.

Stillings, Neil A. 1987. Cognitive Psychology: The Architecture of Mind. In *Cognitive Science: An Introduction.* (Stillings, Neil A., Mark H. Feinstein, Jay L. Garfield, Edwina L. Rissland, David A. Rosenbaum, Steven A. Weisler, and Lynne Baker-Ward). MIT Press, Cambridge MA.

Vilain, M., and H. Kautz. 1986. Constraint Propagation Algorithms for Temporal Reasoning. *Proc. AAAI-86*:377-382. philadelphia, PA.

Wilensky, Robert. 1987. *Some Problems and Proposals for Knowledge Representation*. Report No.UCB/CSD 87/351. Computer Science Division, Univ. of California, Berkeley.

Kandel, Abraham (Ed.). 1991. *Fuzzy Expert Systems*. CRC Press. USA.

Zadeh, Lotfi A. 1979. A Theory of Approximate Reasoning. *Machine Intelligence 9*. pp.149-194. (J.E. Hayes, D. Michie, and L.I. Kulich, Eds.). Wiley, New York.

Zadeh, Lotfi A. 1983. A Computational Approach to Fuzzy Quantifiers in Natural Languages. *Comp & Maths with Appls,* 9(1):149-184. Pergamon Press, Great Britain,

Zadeh, Lotfi A. 1989. Knowledge Representation in Fuzzy Knowledge. *IEEE Transactions on Knowledge and Data Engineering,* (1)1:89-100.

Conceptual Mapping and Bidirectional Machine Translation

Koichi Takeda

Tokyo Research Laboratory, IBM Research
5-19 Sanban-cho, Chiyoda-ku, Tokyo 102, Japan
Phone: 81-3-288-8263, 81-3-265-4370(FAX)
takeda@trl.vnet.ibm.com

Knowledge-based machine translation(KBMT) has emerged as a promising approach for multi-domain, multi-lingual translation since, in particular, it facilitates inheritance-based specification of concepts and object-oriented design of a parser, a paraphraser, and a generator. In this paper, we describe theoretical and practical aspects of KBMT, which appear to define the essentials for future NLP systems.

Topic Area: Machine Translation, Conceptual Representation

Keywords: Bidirectional machine translation, conceptual mapping, conceptual transfer

1. Introduction

Multi-domain/multi-lingual machine translation (M^3T, for short) has been one of the most significant goals of NLP research and development, and there will be even more demands for such technologies especially in European and Pacific Rim countries. Knowledge-based machine translation (KBMT)[3, 12, 20] appears to be a promising approach for M^3T since it provides us AI-based methodologies for organizing a set of concepts (i.e., primitives for conceptual representation of meaning) along with an inheritance hierarchy and for designing object-oriented procedures for parsing, paraphrasing, and generating sentences. Idiosyncratic nature of translation, both lexical and structural, can be incorporated into the definition of each concept and its associated methods.

There are, however, a couple of steps to go further to make KBMT systems more than just a toy program. That is, KBMT should also provide us with a set of well-defined mappings between syntactic and conceptual representations. The former includes full of linguistic phenomena such as coordinated structures and long-distance dependencies, and the latter serves as interlingua which integrates meaning representations of all language-dependent expressions. Another problem is a formulation of "transfer" between a pair of languages such as English and Japanese. It is often the case that a conceptual representations of a sentence and its translation are not identical since a rich set of concepts reflects subtle differences that characterize how we express information in source and target languages. If we try to design a set of concepts so that a sentence and its translation should always map to the same conceptual representation, which is so-called *pivot* translation, we will have too many paraphrases (sentences with the same conceptual representation) which should be differentiated, or some information in a sentence, which contributes to choosing correct translation, will be missing from the conceptual representation of the sentence. Therefore, the notion of "transfer" has to be introduced to define (semi-)equivalence relationships among conceptual representations that are pharaphrasable.

In this paper, we describe a framework of conceptual representation and its mapping to/from syntactic structures, which turns out to be quite effective for translating sentences in computer manuals. The framework has also been employed in a prototype English-Japanese MT system called **Shalt2**[20]. We show that the framework is *bidirectional* in the sense that there are two algorithms that build syntactic structures from a conceptual representation, and conceptual representations from a syntactic structure, respectively. We also introduce an algorithm that performs *conceptual transfer*, a method for paraphrasing conceptual structures.

2. Concepts and Inheritance

A *concept* is a primitive to represent specific meaning in the real world. It has a unique name and zero or more *slots* for holding its modifiers to represent complex meaning. Some slots are used to define is-a and part-of relationships among concepts. Concepts correspond to the notion of *classes* of the object-oriented knowledge bases (e.g., Cattell[1]) or *categories* of the CYC project[7]. For example, one of the concepts expressed by the verb "insert" and the noun "insertion" would be defined as follows.†

```
(defconcept *insert
   (is-a (value *action))
   (:agent (sem *human *system))
   (:theme (sem *physical-object))
   (:goal (sem *location *physical-object)))
```

This is the definition of the concept *insert with an is-a link to a concept *action, and three slots – :agent, :theme, and :goal. The (value ...) facet shows an actual filler of the slot, while the (sem ...) facet shows the *selectional restrictions* on the slot.†† An *instance* of a concept is a specific representation of meaning of a word or a phrase in a sentence.††† A conventional definition of a plain ontology, however, turns out to be insufficient for establishing a framework for conceptual mapping. *Exclusive inheritance*, *meta concepts*, and *secondary concepts*, described below, are thus introduced to the set of concept definitions, and the extended set of concept definitions is called NLCS (Natural Language Class System)[18].

2.1 Exclusive Inheritance

A concept inherits each slot definition from its *ancestors* (superordinate concepts), i.e., the transitive closure of the fillers of the is-a slot, unless the slot is redefined. Although a concept can have more than one immediate ancestor, we restrict its inheritance to be *exclusive* rather than general *multiple* inheritance which has been employed in many object-oriented systems. The exclusive inheritance allows a concept to inherit slot definitions from only one of its immediate ancestor. The idea behind exclusive inheritance is to realize certain identity of verbal and nominal word senses without mixing the slot definitions of both. For example, most verbs have nominal counterparts in a natural language, such as "insert" and "insertion," and such a pair usually shares slot definitions (:agent, :theme, and :goal) and selectional restrictions, except that "insert" inherits tense, aspect, and modality from its "verbal" ancestor but not cardinality, ordinal, and definiteness (that is, the quality of being indefinite or definite) from its "nominal" ancestor, although these features are inherited by "insertion."†††† Besides, exclusive inheritance prevents an instance from having exponentially many inheritance paths at a time, which is a serious problem with multiple inheritance. The following concept definitions

```
(defconcept *physical-action
```

†We use frames[9, 13] for concept definitions and their implementation as KBMT-89 did[3].

††In KBMT systems, a concept might not be just a part of static "ontological" knowledge. We can also define *methods*, or associated procedures, for each concept. Showing definitions or corresponding words of a concept, for example, could be certainly defined as useful methods, but the issue of designing methods in general is beyond the scope of this paper.

†††We use a hyphen and a number following a concept name (*insert-1, *diskette-1, ...) when it is necessary to show instances explicitly. We often identify concept names and instance names, and simply say "*X" instead of "an instance of a concept *X" when it is not confusing.

††††One may claim that "verbal" and "nominal" distinction of concepts are syntactically biased definition and that there really should be one *action sub-hierarchy for both verbs and their nominal counterparts. We observed, however, that the expression "an insertion of a diskette" can be identified with "inserting a diskette," but the expression "the insertion of a diskette" should not be. The exclusive inheritance easily settles this kind of problem. Interestingly, "adjective (adverbial)" and "nominal" inheritances also (e.g., "thick" and "thickness") need to be exclusive. The above observation also holds for Japanese.

```
   (is-a (value *predicate)))
(defconcept *mental-object
   (is-a (value *object)))
(defconcept *action
   (is-a (value *physical-action *mental-object)))
```

allow every instance of *descendants* (subordinate concepts) of *action to inherit slot definitions from either *physical-action or *mental-object.

2.2 Meta Concepts

There are three meta concepts in NLCS – *var, *set, and *fun – to represent concepts that are not included in the normal ontological hierarchy.

The first, *var, is a variable that ranges over a set of NLCS concepts, which are *constants.* Pronouns and question words in natural languages usually carry this kind of incomplete concept. A schematic definition of *var is

```
(defconcept *var
   (domain)
   (range)
   (referent))
```

where *domain* specifies an initial set of concepts that an individual instance of *var ranges over, *range* specifies a set of (possibly complex) instances that satisfy inter/intra-sentential constraints on *var, and *referent* specifies an instance that *var actually refers to in the context. Neither selectional restrictions on the above slot fillers nor the is-a slot can be predefined for *var.

The second, *set, is a set constructor that can represent a coordinated structure in natural languages. A schematic definition of *var is

```
(defconcept *set
   (type)
   (member))
```

whose slot definitions are obvious.

The third, *fun, is a function from NLCS instances to NLCS instances. It captures the meaning of a so-called *semifunctional* word. For example, in some usages, the verb "take" does not really indicate any specific action until it gets an argument such as "a walk," "a rest," "a look." It is therefore well characterized as a function. A schematic definition of *fun is

```
(defconcept *fun
   (domain)
   (range)
   (argument))
```

where *domain* and *range* are the same as those of *var, and *argument* specifies an argument to *fun. We will discuss instances of these meta concepts in detail after we introduce conceptual mapping rules.

2.3 Secondary Concepts

Since we allow only exclusive inheritance, NLCS certainly lacks the ability to organize ontological hierarchy from various viewpoints, unlike ordinary multiple inheritance. *Secondary concepts* are therefore introduced to compensate for this inability. A secondary concept only defines a collection of other concepts. For example,

```
(defvconcept *option
   (def (*math-coprocessor
         *hard-disk *software)))
```

```
(defvconcept *movable-thing
   (def (include :proprety *movable)))
```

show two types of secondary concept, *option and *movable-thing. The *option is a collection of the concepts *math-coprocessor, *hard-disk, and *software. The *movable-thing is a concept that includes any instance with one of the :property fillers being *movable. The secondary concepts differ from *primary concepts* (concepts defined in the ontological hierarchy) in the sense that they have neither is-a ancestors nor inheritance paths. Each member of the secondary concept determines them dynamically.

Note that the maintainability of an ontological hierarchy drastically declines if we allow concepts such as *option to be primary rather than secondary, as *option would have many is-a links from anything that could be an *option*.† The second type of secondary concept helps incorporate so-called *semantic features* into the NLCS. Existing machine-readable dictionaries (for example, LDOCE[14]) often have entries with semantic features such as HUMAN, LIQUID, and VEHICLE that may not fit into a given ontological concept hierarchy. Secondary concept definitions make it possible to integrate such entries into the NLCS.

3. Conceptual Mapping Rules

Conceptual mapping rules define lexical and structural correspondences between syntactic and conceptual representations.†† A *lexical* mapping rule has the form

```
(emap *insert <=l=> insert ((CAT v) (SUBCAT trans))
   (role =sem (*physical-action))
   (:agent =syn (SUBJ))
   (:theme =syn (OBJ))
   (:goal =syn (PPADJUNCT ((PREP into) (CAT n)))))
```

where a transitive verb "insert"††† maps to or from an instance of *insert. The *role* slot shows that *physical-action is specified as an exclusive ancestor of this instance. The *structural mapping (=syn)* between three slots (:agent, :theme, and :goal) and grammatical roles (SUBJ, OBJ, and PPADJUNCT) are also defined in this rule. The :agent filler, for example, should be an instance that is mapped from a syntactic SUBJ of the verb "insert." The :goal filler must be a prepositional phrase consisting of a noun with the preposition "into." The fragments of syntactic feature structures following a lexical word or a grammatical function in a mapping rule specify the minimum structures that must subsume feature structures of candidate syntactic constituents. These structural mappings are specific to this instance. A sample grammatical rule that triggers this mapping rule would be:

$$
\begin{array}{ll}
V \rightarrow & \text{insert} \\
& \langle V\ \text{CAT} \rangle = \text{v} \\
& \langle V\ \text{SUBCAT} \rangle = \text{trans} \\
& \langle V\ \text{FORM} \rangle = \text{finite} \\
& \langle V\ \text{SUBJ AGR NUM} \rangle = \text{pl},
\end{array}
$$

where the first two equations are minimally required. Certain syntactic feature values such as *singular* and *imperative* are not lexical entries, but they are relevant to conceptual representation. A variant of lexical mapping rule, for example,

```
(emap *singular <=f=> singular)
```

is used to define mapping between feature values and instances of concepts.††††

The *structural* mapping rule

†There should be a lexical mapping from the word "option" itself to the secondary concept *option. In this case, the lowest common parent of all "def" fillers of *option will be the is-a filler of the *option instance.

††By syntactic structures we mean feature structures of unification grammars such as LFG[5], PATR-II[17], and HPSG[15] or dependency structures[8, 11]. Throughout the paper, we use PATR-II notation to describe grammar rules.

†††"Emap" stands for mapping rules for English.

††††Agreement features *sg* and *pl* should not be confused with the quality of a noun being singular or plural. The agreement features do not contribute to the conceptual representations.

```
(emap *physical-action <=s=>
   (:mood =syn (MOOD))
   (:time =syn (TENSE)))
```

specifies that the slots :mood and :time map to or from the grammatical roles MOOD and TENSE, respectively. Unlike the structural mapping in a lexical mapping rule, these slot mappings can be inherited by any instance of a subclass of *physical-action. The *insert instance defined above, for example, can inherit these :mood and :time mappings.

Thus, assuming that lexical mapping rules are similarly provided for nouns (diskette and drive) and feature values (imperative, present, and so on), we will obtain a complex instance of *insert

```
(*insert-1
 (:mood  (*imperative-1))
 (:time  (*present-1))
 (:theme (*diskette-1 (:definiteness (*indefinite-1))
                      (:number (*singular-1))))
 (:goal  (*drive-1 (:definiteness (*definite-1))
                   (:number (*singular-2)))))
```

for the sentence "Insert the diskette into the drive."

3.1 Examples of Conceptual Mapping

We are now ready to introduce some of interesting examples which illustrate the importance of gadgets defined in the previous section. Although we procedurally describe the mapping below, it is indeed possible to give a declarative definition of the conceptual mapping.

3.1.1 Coordinated Structures

Coordinated structures are so commonly used in documents that we found as much as 27% of all the sentences in a PC manual included some forms of a coordinated structure. Syntactic accounts for coordinated structures have been proposed by Kaplan and Maxwell[6] using a notion of a *set* of f-structures[†] that was briefly mentioned by Kaplan and Bresnan[5]. Since they defined an f-structure for a coordinated structure to be a set of constituent f-structures except for that of a conjunction, it was not possible to represent, for exmaple, that a coordinated noun phrase has (person 3) (number pl) features as a whole. Therefore, we define a feature structure of a coordinated structure to be a pair $\langle f, \{f_1, \ldots, f_k\}\rangle$, where f is a feature structure that is peculiar to the coordinated structure, including a feature structure of a conjunction, and $f_1, \ldots, f_k$ are feature structures of coordinated constituents. We denote *first(F)* $= f$ and *rest(F)* $= \{f_1, \ldots, f_k\}$ for a complex feature structure $F = \langle f, \{f_1, \ldots, f_k\}\rangle$. For any simple feature structure F, first(F) = rest(F) = F. For example,

$$\begin{aligned} \mathrm{NP} \rightarrow\ & \mathrm{NP_1\ CONJ\ NP_2} \\ & \langle \mathrm{NP\ CONJ}\rangle =_s \langle \mathrm{CONJ}\rangle \\ & \langle \mathrm{NP\ NUM}\rangle =_s \mathrm{pl} \\ & \langle \mathrm{NP}\rangle \ni \langle \mathrm{NP_1}\rangle \\ & \langle \mathrm{NP}\rangle \ni \langle \mathrm{NP_2}\rangle \end{aligned}$$

generates the feature structure

$$\langle ((\mathrm{CONJ}\ ((\mathrm{SIGN\ and})\ (\mathrm{CAT\ conj})))\ (\mathrm{NUM\ pl})), \{f_1, f_2\}\rangle$$

for NP when $\mathrm{NP_1} = f_1$, $\mathrm{NP_2} = f_2$, and CONJ = ((SIGN and) (CAT conj)). The equation, ⟨NP NUM⟩ =s pl, defines a complex feature structure F for NP such that NUM of first(F) unifies with *pl*.[††]

[†] Rounds[16] formulated the logical properties of *set-valued* feature structures.

[††] To be precise, equations have to be extended in order to specify unification of a complex feature structure and a simple/complex feature structure. Takeda defined a new equation =s for LFG in addition to = and ∋ equations[19], where $F_1 =_s F_2$ unifies first(F_1) with F_2, and $F_1 = F_2$ unifies rest(F_1) with F_2, in order to use the same grammar rules for both simple and complex syntactic structures as much as possible.

An instance of a *set is created for each complex feature structure. The structural mapping rule for *set is

```
(emap *set <=s=>
   (type =syn first(# CONJ))
   (member =syn rest(#)))
```

where # denotes a complex feature structure to be mapped. Likewise, we can build instances of *set for phrases "either A or B," "A, B, then C," "A as well as B," etc.

Note the difference between coordinated structures and prepositional phrases (adjuncts) although the latter is formulated in terms of complex feature structures in Kaplan and Bresnan[5] as follows.

$$\begin{array}{lllll} \mathrm{VP} \rightarrow & \mathrm{V} & \mathrm{NP} & \mathrm{NP} & \mathrm{PP}^* \\ & & (\uparrow OBJ) \ni\downarrow & (\uparrow OBJ2) \ni\downarrow & \downarrow\in(\uparrow ADJUNCTS) \end{array}$$

However, the complex feature structures in ADJUNCTS[†] should not be mapped to a *set instance because a presumable instance

```
(*set-1 (member (value *tuesday-1 *tokyo-1)))
```

for two adjacent PP's "on Tuesday in Tokyo" clearly screws up the :location and :date roles. For this reason, we have another kind of equation > for non-set-forming constituents.[††] That is,

$$\begin{array}{lll} \mathrm{VP}_1 \rightarrow & \mathrm{VP}_2 & \mathrm{PP} \\ & \langle \mathrm{VP}_1 \rangle = \langle \mathrm{VP}_2 \rangle & \\ & \langle \mathrm{VP}_1\ PPADJUNCT \rangle > \langle \mathrm{PP} \rangle & \end{array}$$

where each PP fills a specific role of the head VP.

3.1.2 Pronouns, WH-Words, and Gaps

The pronoun "one" in the sentence "Flip down the red one." will be mapped to the instance

```
(*var-1
   (domain (value *physical-object-1))
   (range (value *lever-1))
   (referent (value *power-switch-1)))
```

where selectional restriction of the verb "flip down" on its :theme slot is *lever, and the pronoun actually refers to a power switch that appeared in the context. The mapping rule for this pronouns is

```
(emap *var <=l=> one ((CAT pro))
   (domain =sem (*physical-object))
   (:definiteness =syn (DET))
   (:property =syn (APMOD)))
```

which maps the pronoun to the instance *var-1 during the parsing process. The feature value "definite" and the adjective "red" are similarly mapped to the instances *definite-1 and *red-1, respectively, to be modifiers of *var-1. Since *var-1 is a variable, the slot mappings (:definiteness =syn (DET)) for the determiner "the" and (:property =syn (APMOD)) for the adjective phrase "red" are applied to each *domain filler*, and we get

```
(*physical-object-2
   (:definiteness (value *definite-1))
   (:property (value *red-1)))
```

which becomes the *range* filler of *var-1. When this *var-1 becomes the :theme filler of an instance of "flip down", where (:theme (sem *lever)) and *lever is-a *physical-object, the selectional constraint narrows down the possible referent of *var-1 from *physical-object-2 to

[†]PP* is a regular expression to denote any number of adjuncts PP in the right hand side of the rule.

[††]This equation is called an "append" operation in KBMT-89, and is inappropriately used for both adjuncts and coordinated structures.

```
(*lever-1
   (:definiteness (value *definite-1))
   (:property (value *red-1)))
```

Finally, contextual constraints (or *anaphora resolver*) determines the *referent* filler of *var-1.

Note that *var-1 is so informative that it allows a generator to produce at least three different syntactic representations. The first, "the red one," is just the inverse of the above conceptual mapping. That is, an instance of *var with (domain (value *physical-object)) maps to "one," and the :property and :definiteness modifiers of the range filler map to "the red." The second, "the red lever," is recovered from the range filler alone. The third, "the red power switch" is recovered from the referent filler and the :property and :definiteness modifiers of the range filler. The second and third ones are paraphrases of the first one.

Mapping rules for *var can be defined for WH-words and gaps. The WH-word "why" can be mapped to *var by

```
(emap *var <=l=> why ((CAT wh))
   (domain =sem (*reason)))
```

A gap in a relative clause can be mapped to *var by

```
(emap *var <=l=> @gap)
```

where a referent filler is immediately obtained from the antecedent noun of the relative clause.† It is not possible, however, to get paraphrases from these instances of *var.

Other interesting usages of *var include the handling of unknown words. An unknown word triggers the lexical mapping rule

```
(emap *var <=l=> @unknown
   (string =syn (string)))
   (domain =sem (*object))
```

which means the unknown word could be any object with the string slot filled with the character string of the word. If we have to assume that the unknown word is a verb, an adjective, etc., we can define similar lexical mapping rules. Inheritance of slot mappings from potential domain fillers such as *object, *physical-action, or *attribute would help us build conceptual representations including the instances of such unknown words.

3.1.3 Semifunctional Words

Instances of *fun are used to represent incomplete word senses of semifunctional words "take," "do," "make," etc. An example of conceptual mapping rule for "take" is

```
(emap *fun <=l=> take ((CAT v) (SUBCAT trans))
   (domain =sem ((*bring obj)/*person (*photograph obj)/*picture ...)))
```

where an actual specification is a huge list of

> (instance returned from the function, grammatical function of the argument) / semantic restrictions on the argument

pairs. We will have the *fun instance

```
(*fun-1
   (domain (value ((*bring obj)/*person (*photograph obj)/*picture ...)))
   (range (value (*seize-1 *digest-1)))
   (argument (value (*aspirin-1 (:number (value *singular-1))))))
```

†An empty sign of a gap is encoded as @gap.

for the phrase "Take aspirin," where *seize-1 and *digest-1 include *aspirin-1 possibly as :theme filler provided that (*seize-1 obj)/*small-object and (*digest obj)/*food are listed in the domain slot.[††]

Some of auxiliary verbs in Japanese and the English verbs "seem" and "be" can be mapped to *fun to reduce bi-clausal analysis of these verbs with a VP complement into an instance of VP with a "presumption" feature. That is, the phrase "X seems to know Y." will be mapped to

```
(*know-1
   (:presumption (value *true-1))
   (:mood (value *declarative-1))
   (:time (value *present-1))
   (:agent (value *X-1))
   (:theme (value *Y-1)))
```

using the conceptual mapping rule

```
(emap *fun <=l=> seem ((CAT v) (SUBCAT intrans) (comp-type VPBAR))
   (domain =sem (/*action))
   (:presumption =sem (*true)))
```

where */*action* maps any instance of *action to itself.

3.1.4 Integers, Character Strings, and Derivative words

Integers and character strings can be arbitrarily large and long. Therefore, two generic lexical mapping rules are defined for these lexical entities. Any integer X is mapped to an instance of *integer with the :value slot filled with the integer X. Similarly, any character string "Y" embedded in a sentence is mapped to an instance of *string with the string slot filled with "Y."

Generic mapping rules are also necessary to define consistent mapping rules among derivative words. An intuitive example is a verb and its infinitive and present participle forms as illustrated below.

```
(emap *insert <=l=> insert ((CAT v) (SUBCAT trans) (FORM infinite))
   (role =sem (*physical-action))
   (:agent =syn (PPADJUNCT ((PREP for) (CAT n) (PREMOD +)))
   (:theme =syn (OBJ))
   (:goal =syn (PPADJUNCT ((PREP into) (CAT n)))))

(emap *insert <=l=> inserting ((CAT v) (SUBCAT trans) (FORM prtpart))
   (role =sem (*physical-action))
   (:agent =syn (PREMOD ((CAT (*or n pro)) (FORM genitive )))
   (:theme =syn (OBJ))
   (:goal =syn (PPADJUNCT ((PREP into) (CAT n)))))
```

By generalizing this regularity, we can define generic mapping rules for derivative words.

4. Conceptual Paraphrasing Rules

An adjective modifying a noun in English might be a conjugable verb in translation, and their differences result in added/missing information in their conceptual structures. This fact implies that we have to provide a set of *conceptual paraphrasing rules* to describe a set of equivalent and semi-equivalent conceptual representations. Practically, these rules should be sensitive to the target language, but not to the source language, since the definition of equivalence among conceptual representations depends on the cultural and pragmatic background of the language in which a translation has to be expressed. An example of a paraphrasing rule is

[††]The *small-object is a typical secondary class for defining inanimate movable objects except for vehicles, etc. Any grammatical function or adjunct, excluding the one which is mapped to the argument slot of the *fun (e.g., obj), is examined against each domain filler of the *fun instance for determining conceptual slot to be mapped.

```
(equiv (*equal (:agent (*X (:number (*V))))
               (:theme (*Y/*person
                             (:definiteness (*indefinite))
                             (:number (*W)))))
       (*Z/*action (:agent (*X (:number (*V)))))
       (such-that (humanization *Z *Y)
                  (sibling *V *W)))
```

where *X, *Y, ..., are variables that denote fillers of value facets, *Y/*person specifies *Y to be an instance of any descendant of *person, *equal is roughly the verb "be," *humanization* is a relation that holds for pairs such as (*singer, *sing) and (*swimmer, *swim), and *sibling* holds for two instances of the same concept. Intuitively, this rule specifies an equivalence relationship between sentences such as "Tom is a good singer" and "Tom sings well," as the following bindings hold:

```
(*equal-1
   (:mood (value *declarative-1))
   (:time (value *present-1))
   (:agent (value *tom-1 (:number (value *singular-1))))
   (:theme (value *singer-1 (:property (value *good-1))
                            (:definiteness (value *indefinite-1))
  (:number (value *singular-2)))))

(*sing-1
   (:mood (value *declarative-1))
   (:time (value *present-1))
   (:agent (value *tom-1 (:number (value *singular-1))))
   (:property (value *good-1)))

where *X = *tom-1, *Y = *singer-1, *Z = *sing-1,
      *V = *singular-1, and *W = *singular-2.
```

All the instances that have no binding in a rule must remain unchanged as the same slot fillers (e.g., *declarative and *present), while some fillers that have bindings in a rule may be missing from a counterpart instance (e.g., *indefinite and *W above). Note that *good has lexical mappings to the adjective "good" and the adverb "well."

5. Generatror-Driven Conceptual Transfer

Given a conceptual representation C, which is mapped from a sentence in source languages, we have to map C to another conceptual representation C' that has a well-defined mapping to a feature structure of a target language using a set of conceptual paraphrasing rules. This mapping from C to C' is called *conceptual transfer*. If the given conceptual representation already has a well-defined mapping to a feature structure, the conceptual transfer is just an identity mapping. It is important that conceptual transfer should be related with the mapping to a feature structure, because there are generally many members in a set of (semi-)equivalent conceptual structures. The existence of well-defined mapping not only guarantees that the generator can produce a sentence in the target language, but also effectively eliminates unsuccessful paraphrasing.

In addition to the paraphrasing rules mentioned earlier, the following general rules are included in the conceptual transfer.† The paraphrasing rules are composed to make a complex mapping.

- Projection: Map an instance with a filled slot *s* to an instance of the same concept with the unfilled slot *s*. Projection corresponds to deletion of a slot *s*.

†These are semi-equivalent rules. Equivalent rules have higher priority when the rules are to be applied.

- Generalization: Map an instance of a concept *X to an instance of one of the ancestors of *X.
- Specialization: Map an instance of a concept *X to an instance of one of the descendants of *X.

A projection rule is frequently used when we translate English nouns into Japanese ones, as in the following example:

```
diskette       (*diskette (:num (*sg)))
diskettes      (*diskette (:num (*pl)))
a diskette     (*diskette (:num (*sg))
                          (:def (*indef)))
the diskettes  (*diskette (:num (*pl))
                          (:def (*def)))
ディスケット   (*diskette)
```

Here, the four English noun phrases above are usually translated by the same Japanese noun phase[†] (the fifth one), which does not carry any information on :num and :def. We provide a paraphrasing rule for translation in the opposite direction such that for any instance of the *object can obtain appropriate :num and :def fillers. The parser, however, is responsible for determining these fillers in most cases. In general, the designer of semi-equivalent rules for translation in one direction has to provide a way of inferring missing information for translation in the opposite direction. Generalization and specialization rules are complementary and can be paired to become equivalent rules when a specialization rule for any instance of a class x is unambiguous. That is, without losing any fillers, one can always choose an instance of a subclass y to which x can be uniquely mapped. A generalization from each y to x provides the opposite mapping.

An algorithm for conceptual transfer is a top-down, recursive procedure. Let us view a conceptual structure as a tree and its fillers as subtrees.[††] The algorithm then works as follows:

1. If a root has no lexical mapping, find a paraphrasing rule to map the root to some other instance. Use generalization or specialization when there is no appropriate paraphrasing rule.
2. Find a feature structure for each subtree. If there is no feature structure for some subtree, map the subtree to another instance. If this fails, map the root to some other instance. Start from step 1.
3. Compose the feature structures for the subtrees to make an entire feature structure. If this fails, map the root to some other instance. Start from step 1.

Once the feature structures for the subtrees have been computed, it is not necessary to re-compute them, even when the root is mapped to another instance, as long as the filler is unchanged. By scheduling the equivalent and semi-equivalent rules appropriately, the algorithm can be made to terminate in order to produce a feature structure, although we may have to sacrifice some of the original semantic content.

6. Bidirectional Machine Translation

We briefly summarize the theoretical properties of NLCS framework.

- Lexical mapping rules are bidirectinal. Any feature structure F that maps to an instance I using the rule R is subsumed by the feature structure F' specified in R. Wedekind showed that F' suffices to generate F as long as conceptually relevant features are unchanged[22]. This property also holds for mapping rules of complex feature structures and meta classes. However, the mapping is not necessarily one-to-one, and abuse of *fun mapping rules, for example, could easily result in extreme ambiguities in mapping from a conceptual representation to feature structures.[†††]

[†]One exception is that deictic noun phrases are translated when we use the Japanese counterpart "その" for the determiner "the".

[††]A general graph structure can be converted into a tree by duplicating each node which has more than one parent.

[†††](*X (:theme *Y)) can be mapped to "make Y", "do Y", "take Y" if (*X obj)/*Y is in domain fillers of *fun mapping rules for "make", "do", and "take".

- Structural mapping rules are bidirectional (provable by mathematical induction on the number of mapping rules.) Thus, any composition of lexical and structural mapping rules is bidirectional.
- Equivalent paraphrasing rules are trivially bidirectional.
- Semi-equivalent paraphrasing rules are bidirectional if there exists an inverse rule R' for any R such that R maps I to I' iff R' maps I' to I for any instances I and I'.

7. Conclusion

As discussed in van Noord [21] and Dymetman [2], bidirectionality holds for unification grammars and lossless transfer rules based on logical forms as semantic representation. Our results confirmed the similar result for object-oriented concept definitions and mapping schemes that can handle broad range of syntactic structures. Furthermore, recent activities on KBMT approach explore the areas such as automatic knowledge acquisition [4] and efficient interlingua translation [10] which have been considered major bottlenecks of building practical KBMT systems. With our mapping scheme described in this paper, bidirectional KBMT systems can be developed for a broad-coverage, practical application domains.

Although a logic-based approach is important for understanding the meaning of sentences, and logical framework with inference mechanism can do more powerful paraphrasing than our "conceptual transfer" approach, our approach is often better suited for machine translation since translating "what is expressed" is usually preferred to translating "what is implied". The former is what "conceptual" paraphrasing is primarily designed for. Conceptual representations can also be a convenient tool to derive logical formulae since the mapping between conceptual representations and logical formulae can be described without worrying about syntactic details.

8. Acknowledgments

I would like to thank Shiho Ogino for her discussions on Japanese syntactic theories that inspired me to refine the ideas of conceptual mapping and conceptual transfer.

References

1) R. G. G. Cattell. "Introduction" in special section for Next-Generation Database Systems. *Communications of the ACM*, 34(10):31–33, Oct. 1991.

2) M. Dymetman. "Inherently Reversible Grammars, Logic Programming and Computability". In *Proc. of ACL Workshop on Reversible Grammar in Natural Language Processing*, pages 20–30, Berkeley, California, June 1991.

3) K. Goodman and S. Nirenburg, editors. *"The KBMT Project: A Case Study in Knowledge-Based Machine Translation"*. Morgan Kaufmann Publishers, San Mateo, California, 1991.

4) A. G. Hauptmann. "From Syntax to Meaning in Natural Language Processing". In *Proc. of the AAAI'91*, pages 125–130, 1991.

5) R. Kaplan and J. Bresnan. "Lexical-Functional Grammar: A Formal System for Generalized Grammatical Representation". In J. Bresnan, editor, *"Mental Representation of Grammatical Relations"*, pages 173–281. MIT Press, Cambridge, Mass., 1982.

6) R. M. Kaplan and J. T. Maxwell III. "Constituent Coordination in Lexical-Functional Grammar". In *Proc. of the 12th International Conference on Computational Linguistics*, pages 303–305, Aug. 1988.

7) D. B. Lenat, R. V. Guha, K. Pittman, D. Pratt, and M. Shepherd. "CYC: Toward Programs with Common Sense". *Communications of the ACM*, 33(8):30–49, Aug. 1990.

8) H. Maruyama. "Structural Disambiguation with Constraint Propagation". In *Proc. of the 28th Annual Meeting of ACL*, volume 3, pages 31–38, June 1990.

9) M. Minsky. "A Framework for Representing Knowledge". In P. Winston, editor, *The Psychology of Computer Vision*, pages 211–277. McGraw-Hill, 1975.

10) T. Mitamura, E. H. Nyberg, 3rd, and J. G. Carbonell. "An Efficient Interlingua Translation System for Multi-lingual Document Production". In *Proc. of the Machine Translation Summit III*, July 1991.

11) K. Nagao. "Dependency Analyzer: A Knowledge-Based Approach to Structural Disambiguation". In *Proc. of the 13th International Conference on Computational Linguistics*, pages 484–489, Aug. 1990.

12) S. Nirenburg, J. Carbonell, M. Tomita, and K. Goodman, editors. *"Machine Translation: A Knowledge-Based Approach"*. Morgan Kaufmann Publishers, San Mateo, California, 1991.

13) E. H. Nyberg. *"A User's Guide to the FrameKit Knowledge Representation System"*, July 1987. Preliminary Draft.

14) Procter P. *Longman Dictionary of Contemporary English.* Longman Group Limited, Harlow and London, England, 1978.

15) C. Pollard and I. A. Sag. *"An Information-Based Syntax and Semantics, Vol.1 Fundamentals"*. CSLI Lecture Notes, Number 13, 1987.

16) W. C. Rounds. "Set Values for Unification-Based Grammar Formalisms and Logic Programming". Technical Report 129, CSLI, 1988.

17) S. M. Shieber. *"An Introduction to Unification-Based Approaches to Grammar"*. CSLI Lecture Notes, Number 4, Stanford, CA, 1986.

18) K. Takeda. "Designing Natural Language Objects". In *Proc. of 2nd International Symposium on Database Systems for Advanced Applications*, pages 444–448, Tokyo, Japan, April 1991.

19) K. Takeda. "Unification Grammars for Processing Coordinated Structures". Unpublished manuscript, 1992.

20) K. Takeda, N. Uramoto, T. Nasukawa, and T. Tsutsumi. "Shalt2 - A Symmetric Machine Translation System with Conceptual Transfer". Technical Report RT0068, Tokyo Research Laboratory, IBM Research (to appear in COLING'92), Nov. 1991.

21) G. van Noord. "Reversible Unification Based Machine Translation". In *Proc. of the 13th International Conference on Computational Linguistics*, volume 2, pages 299–304, Helsinki, Aug. 1990.

22) J. Wedekind. "Generation as Structure Driven Derivation". In *Proc. of the 12th International Conference on Computational Liguistics*, pages 732–737, August 1988.

Bottom-up Parallel Parser by Model Generation Theorem Prover Extended Abstract

Masayuki Fujita
Frank O'Carroll
Koichi Furukawa
Institute for New Generation Computer Technology

October 29, 1992

Abstract

This extended abstract shows topdown control in a bottom-up system produced by the magic set transformation has a very clear correspondence to optimization of a bottom-up parser. The well known chart parser can be obtained from the DCG grammar by a very formal transformation method. A bottom-up theorem prover MGTP developed at ICOT becomes a parallel chart parser because of this transformation.

1 Parallel Bottom-up Theorem Prover MGTP

A model generation theorem prover (**MGTP**) implemented in KL1 searches for proofs of specification expressed as logical formulae and it runs on a parallel machine:Multi-PSI. **MGTP** is a hyper-resolution based bottom up (infers from premises to goal) prover. Thanks to KL1 programming technology **MGTP** is simple but works very efficiently if problems are *range-restricted*. The inference mechanism of **MGTP** is similar to SATCHMO[MB88] in principle.

The MGTP prover adopts model generation as a basic proof procedure. We assume that a theorem to be proven is negated and transformed to a set of clauses, then we try to refute the clause set as in the resolution method. A clause is represented in an implicational form:

$$A_1, A_2, \ldots, A_n \rightarrow C_1; C_2; \ldots; C_m$$

where $A_i (1 \leq i \leq n)$ and $C_j (1 \leq j \leq m)$ are atoms; the antecedent is a conjunction of $A_1, A_2, \ldots, A_n$; the consequent is a disjunction of $C_1, C_2, \ldots, C_m$.

There are the following two rules in the model generation method.

- Model extension rule: If there is a clause, $A \rightarrow C$, and a substitution σ such that $A\sigma$ is satisfied in a model M and $C\sigma$ is not satisfied in M, then extend the model M by adding $C\sigma$ into the model M.
- Model rejection rule: If there is a negative clause whose antecedent $A\sigma$ is satisfied in a model M, then reject the model M.

The task of model generation is to try to construct a model for a given set of clauses starting with a null set as a model candidate. If the clause set is satisfiable, a model should be found. The method can also be used to prove that the clause set is unsatisfiable, by exploring every possible model candidate to see that no model exists for the clause set.

The model generation method does not need full unification during conjunctive matching of the antecedent literals against model elements if the range-restrictedness[1] condition is imposed on problem clauses. When range-restrictedness is satisfied, it is sufficient to consider one-way unification, or matching, instead of full unification with occurs check because a model candidate constructed by the model generation rules should contain only ground atoms. This property is favors a prover implemented in KL1 since matching is easily realized with head unification and the variables in the given clauses can be represented as KL1 variables. Experimental results show that the MGTP prover is efficient in solving range-restricted non-Horn problems. For details of techniques for implementing MGTP, refer to [HFF90], [FH90].

2 Magic Set Transformation

Bottom-up reasoning systems do not care whether the produced resolvent is relevant to the refutation or not. The magic set transformation of Horn-programs is a deductive database technique to introduce top-down control to a bottom-up generator of consequences. A naive bottom-up theorem prover will have a goal oriented control with this method. If we let P_* and Q be atomic literals and n, a natural number and $G(\_)$ be a meta predicate that means $\_$ is a goal, then a rough sketch of the magic set transformation rule is as follows:

$$\begin{array}{rcl} P_1, \ldots, P_n & \rightarrow & Q \\ & \Rightarrow & \\ G(Q) & \rightarrow & G(P_1) \\ G(Q), P_1 & \rightarrow & G(P_2) \\ & \cdots & \\ G(Q), P_1, \ldots, P_n & \rightarrow & Q \end{array} \qquad (1)$$

[1]To ensure range-restrictedness, a *dom/1* predicate is added to the antecedents of problem clauses and extra clauses for the predicate are added to the original set of clauses, if necessary. This transformation does not change the satisfiability of the original set of clauses.

$$\begin{array}{rcl}
s & \rightarrow & np,\ vp \\
np & \rightarrow & a,\ n \\
np & \rightarrow & prp,\ n \\
vp & \rightarrow & v,\ a \\
vp & \rightarrow & v,\ av \\
a & \rightarrow & [failing] \\
prp & \rightarrow & [failing] \\
a & \rightarrow & [hard] \\
av & \rightarrow & [hard] \\
n & \rightarrow & [students] \\
v & \rightarrow & [looked]
\end{array}$$

Figure 1: Natural Language Parsing Example Problem

The transformed clauses may not be range restricted even if the initial clause set is so, because each goal within the metapredicate G may generate a goal with new variables. This is a serious problem for a bottom-up prover which employs range restrictedness for achieving good performance. This problem is bypassed by introducing new predicate symbols called 'adornment'[Beeri89] if the problem domain has no function symbols, which is usually assumed in the database problem domain.

3 Magic Set Application to Language Parser

Applying the magic-set transformation to context-free grammar rules translated into range restricted Horn-clauses can produce a bottom-up left-to-right parser with top-down control. This is very similar to the well-known chart parser.

By using a tiny example, we will show and discuss:

(1) how to represent the parsing problem as a problem in MGTP,

(2) a chart parser produced by the magic set transformation of (1),

(3) various levels of parallelization on a PIM machine.

3.1 Definite Clause Representation of Context Free Grammar

We use a tiny example in fig. 1 to show how to represent parsing problems.

The example sentence is "failing students looked hard."

The parsing problem is more constrained than a usual theorem proving problem in that only adjacent subtrees can form higher level parse trees. So that MGTP only constructs correct parses, this information must somehow be added into the model. After adding start and end locations to the parse tree, the constrained clauses appear in fig.2. The rule for this is very simple.

3.2 Applying the Magic Set Transformation

Since all clauses in the previous example are all Horn clauses, we can use the magic set method appropriate to Horn clauses, and the adornment technique[Beeri89]to keep the clauses range-restricted, giving the MGTP clauses in fig.3.

This parser (1) predicts non-terminal symbols from the top down, (2) proceeds left to right, and (3) re-uses prevously constructed subtrees in basically the same manner as a chart parser.

In this example, since there are four different possible parse trees, MGTP has now way to find all of them, since $s(\_)$ is a query. However, if we change the control strategy to generate all possible model elements, then we can find all parses. But all the possible parse trees are generated within a single model, so it is difficult to decide when to terminate the search.

3.3 Parallelizing Alternative Productions

In the example, np and vp each have two alternative productions:

$$np \rightarrow a,\ n$$
$$np \rightarrow prp,\ n$$

and

$$vp \rightarrow v,\ a$$
$$vp \rightarrow v,\ av$$

The search for each alternative can be executed in parallel by splitting the model with a disjunctive representation (fig.4).

This method of parallelization has two advantages:

(i) Since at most one tree appears in each model, refutation can be used to control termination. It provides us with favorable conditions for avoiding useless model generation.

(ii) The search is split into independent searches of the alternatives, needing no further intercommunication, a great advantage in parallelization.

On the other hand, redundancy is introduced, for if such alternative productions have constituents in common, the same constituents will be constructed in each model, duplicating work.

```
% goal
s(0,4,_) ---> false.

% s --> np,vp
np(S,I,NP),vp(I,E,VP) ---> s(S,E,s(NP,VP)).

% np --> a,n
a(S,I,A),n(I,E,N) ---> np(S,E,np(A,N)).

% np --> prp,n
prp(S,I,PRP),n(I,E,N) ---> np(S,E,np(PRP,N)).

% vp --> v,a
v(S,I,V),a(I,E,A) ---> vp(S,E,vp(V,A)).

% vp --> v,av
v(S,I,V),av(I,E,AV) ---> vp(S,E,vp(V,AV)).

% a --> [failing]
failing(S,E,F) ---> a(S,E,a(F)).

%prp --> [failing]
failing(S,E,F) ---> prp(S,E,prp(F)).

% a --> [hard]
hard(S,E,H) ---> a(S,E,a(H)).

%av --> [hard]
hard(S,E,H) ---> av(S,E,av(H)).

% n --> [students]
students(S,E,St) ---> n(S,E,n(St)).

% v --> [looked]
looked(S,E,L) ---> v(S,E,v(L)).

% input sentence to be parsed
% [failing students looked hard]
true ---> failing(0,1,failing).
true ---> students(1,2,students).
true ---> looked(2,3,looked).
true ---> hard(3,4,hard).
```

Figure 2: The example as MGTP clauses

```
% goal
true ---> gs(0,4).
s(0,4,_) ---> false

% s --> np,vp
gs(S,E) ---> gnp(S).
gs(S,E),np(S,I,_) ---> gvp(I).
gs(S,E),np(S,I,01),vp(I,E,02) ---> s(S,E,s(01,02)).

% np --> a,n
gnp(S) ---> ga(S).
gnp(S),a(S,I,_) ---> gn(I).
gnp(S),a(S,I,01),n(I,E,02) ---> np(S,E,np(01,02)).

% np --> prp,n
gnp(S) ---> gprp(S).
gnp(S),prp(S,I,_) ---> gn(I).
gnp(S),prp(S,I,01),n(I,E,02) ---> np(S,E,np(01,02)).

% vp --> v,a
gvp(S) ---> gv(S).
gvp(S),v(S,I,_) ---> ga(I).
gvp(S),v(S,I,01),a(I,E,02) ---> vp(S,E,vp(01,02)).

% vp --> v,av
gvp(S) ---> gv(S).
gvp(S),v(S,I,_) ---> gav(I).
gvp(S),v(S,I,01),av(I,E,02) ---> vp(S,E,vp(01,02)).

% a --> [failing]; prp --> [failing]
ga(S),failing(S,E) ---> a(S,E,a(failing)).
gprp(S),failing(S,E) ---> prp(S,E,prp(failing)).

% a --> [hard]; av --> [hard]
ga(S),hard(S,E) ---> a(S,E,a(hard)).
gav(S),hard(S,E) ---> av(S,E,av(hard)).

% n --> [students]
gn(S),students(S,E) ---> n(S,E,n(students)).

% v --> [looked]
gv(S),looked(S,E) ---> v(S,E,v(looked)).
```

Figure 3: Top down parser produced by applying the magic set transformation (modified part of fig. 2)

```
% s --> np,vp
gs(S,E) ---> gnp1(S);gnp2(S).
gs(S,E),np(S,I,_) ---> gvp1(I);gvp2(I).
gs(S,E),np(S,I,01),vp(I,E,02) ---> s(S,E,s(01,02)).

% np --> a,n
gnp1(S) ---> ga(S).
gnp1(S),a(S,I,_) ---> gn(I).
gnp1(S),a(S,I,01),n(I,E,02) ---> np(S,E,np(01,02)).

% np --> prp,n
gnp2(S) ---> gprp(S).
gnp2(S),prp(S,I,_) ---> gn(I).
gnp2(S),prp(S,I,01),n(I,E,02) ---> np(S,E,np(01,02)).

% vp1 --> v,a
gvp1(S) ---> gv(S).
gvp1(S),v(S,I,_) ---> ga(I).
gvp1(S),v(S,I,01),a(I,E,02) ---> vp(S,E,vp(01,02)).

% vp2 --> v,av
gvp2(S) ---> gv(S).
gvp2(S),v(S,I,_) ---> gav(I).
gvp2(S),v(S,I,01),av(I,E,02) ---> vp(S,E,vp(01,02)).
```

Figure 4: Model splitting using alternative rules from the top-down parser (modified part of fig. 3)

In this example, since we divide the model corresponding to the two *np* alternatives, then the search for the two *vp* subtrees is duplicated. It is rather difficult to estimate the degree of redundancy since it depends on the particular grammar and input text.

References

[Beeri89] Beeri, C., Ramakrishnan, R., On the Power of Magic, revised version of PODS'87(pp269-283), 1989.

[FH90] Fujita, H. and Hasegawa, R., Implementing A Parallel Theorem Prover in KL1, in *Proc. of KL1 Programming Workshop '90*, pp.140-149, 1990 (in Japanese).

[HFF90] Hasegawa, R., Fujita, H. and Fujita, M., A Parallel Theorem Prover in KL1 and Its Application to Program Sysnthesis, *Italy-Japan-Sweden Workshop*, ICOT TR-588, 1990.

[MB88] Manthey, R., and Bry, F., SATCHMO: a theorem prover implemented in Prolog, in *Proc. of CADE 88, Argonne, Illinois*, 1988.

[OM87] Okumura, A. and Matsumoto, Y. Parallel Programming with Layered Streams, *Proc. of 4th Symposium on Logic Programming*, pp. 343-350. San Francisco, 1987

Automatic Generation of Semantic Code Trees

R.Sugimura, K.Fujita, M.Ishikawa,
M.Kawagoe, S.Aoyama, T.Cornish

June 2, 1992

Abstract

A basic method for generating a SEmantic Code Tree (abbreviated to SECT) is proposed. The method presented here has two procedures. In the first procedure, predicates like verbs or adverbs in natural language are collected from example sentences, and pairs of the predicates and their arguments in subject case are chosen from the examples. In the second procedure, the chosen pairs are transformed into SECT. Three kinds of algorithm for the transformation from the chosen pairs into SECT play an important role in the method. The algorithms always terminate. They enable us to get the SECT of desirable features. In SECT, a group of words are represented using a generated non-terminal tree node. SECT is suitable for checking agreement between a predicate and its argument. Some other fundamental features are also described

1 Introduction

We propose a basic method to generate a semantic code tree for natural language analysis. In natural language analysis, constraints on semantic agreement between a predicate and its argument (such as a verb and its complements) are usually implemented previously in the analysis part to reduce many of the ambiguities generated from the analysis of a sentence. The total number of semantic constraints amounts to the combinatorial number of a predicate and its arguments. Therefore we usually condense the constraints into the form of a thesaurus.

The thesaurus [3][4][5] condenses words using a tree. Words or concepts are mapped onto nodes of a tree, and relations between words are represented by links. The typical relation is IS-A. The IS-A relation is comparatively easy to be understood by researchers, so this relation between words is most widely used as the basic framework of a thesaurus[7]. In the IS-A relation, a property of a word is inherited by its descendant words in a tree. Therefore, this feature is used to condense word meanings, and the tree can be used for natural language understanding [9] [8].

In spite of the popularity of the IS-A relation, pairs of words in an IS-A relation are constructed by inspiration. It is therefore very hard to make a clear definition of what is an IS-A relation, and what is not [5]. This situation results in that there are a variety of thesauri. Moreover, it is very hard to maintain a thesaurus data structure.

In this paper, we will present a new method which enables us to get an objective thesaurus which eliminates fuzzy inspiration.

In this method, we suppose that relations between words can be obtained through actual usage of words, so that we will not presuppose semantic features.

At the first stage, the predicate and its complements are collected. Then, they are represented in the form of a set of trees which have no nonterminal node except the root node. The tree is rewritten into a new tree with non-terminal nodes. In this paper, 3 kinds of rewriting algorithm will be presented.

The first algorithm aims principally at the reduction of the number of links of the tree. The second algorithm applies its rewriting rule from the leaf nodes to the root note. The third algorithm applies its rewriting rule from the root node to the leaf nodes.

Non-terminal nodes which are generated from rewriting will get a unique name, and be used in natural language analysis. [6]

SECT is built up using concrete examples, therefore there is no exceptional node. The resultant structure is similar to some thesauri. The non-terminal nodes of SECT are characterized by the predicates which define them.

2 Overall Structure of a Semantic Code Tree

Generally speaking, sentence analysis is usually carried out by checking the appropriateness of an argument in a predicate.

For example, let us consider the Japanese sentences "machineが生き生きした 鳥に かなうわけがない" (A machine cannot match a lively bird) and "羽が生き生きとした鳥にかなうわけがない。"($^+$zero$^+$ cannot match a bird with fresh wings).

In sentence analysis of the examples, it will be checked whether the noun "machine" (machine) can be the subject of the adjective "fresh/lively" (生き生きした). In an analysis program, knowledge like 1) that the subject of "fresh" should be a living thing, and 2) that a bird is a living thing, are previously implemented using the "is_a" relation. A machine cannot be a living thing, so that a "machine" (machine) can only modify a verb "can not match (かなうわけがない)".

In the sentence analysis of the second sentence, the noun "羽" (wing) is a living thing, and a living thing can be the subject of the adjective "生き生きした"(fresh), therefore "羽" (wing) becomes the subject of "生き生きした"(fresh).

Let us define the IS-A relation to be reflexive and transitive.

$$\begin{aligned} is\_a(A,B) &\leftarrow is\_a(A,C) \wedge is\_a(C,B) && (1)\\ & \quad is\_a(A,A). && (2) \end{aligned}$$

Semantic restrictions on arguments of a predicate can be defined as follows.

$$\forall A(PRED_k(atom_j) \wedge is\_a(atom_j, A) \quad \rightarrow \quad PRED_k(A)) \qquad (3)$$

For example, if $atom_j$ is appropriate to $PRED_k$, A will be appropriate for $PRED_k$.

3 Generation of Semantic Codes Tree

3.1 Structure of Semantic Codes Tree

Roughly speaking, the structure of the semantic code tree can be recognized as a vector as follows.

$$<< laugh/1, move/1, demand-attention/0 >>$$

Therefore, the semantic code tree constructs a lattice.

$$\begin{tikzcd}[column sep=small]
 & \ll \text{a/0,b/0,...,z/0} \gg \arrow[dl, dash] \arrow[dr, dash] & \\
\ll \text{a/1,b/0,...,z/0} \gg & & \ll \text{a/0,b/0,...,z/1} \gg \\
\\
\ll \text{a/0,b/1,...,z/1} \gg \arrow[dr, dash] & & \ll \text{a/1,b/1,...,z/0} \gg \arrow[dl, dash] \\
 & \ll \text{a/1,b/1,...,z/1} \gg &
\end{tikzcd}$$

In the following sections, we will present a top-down generation algorithm, and bottom-up algorithm.

3.2 Desirable Features of SECT

SECT satisfies the following condition. That means, if A is appropriate for a predicate, the transitive closure of A, derived from the IS_A relation, will be appropriate for the predicate.

$$\forall(P(X) \wedge is\_a(A,X) \leftarrow P(A)) \tag{4}$$
$$is\_a(A,B) \leftarrow is\_a(A,C) \wedge is\_a(C,B) \tag{5}$$

3.3 Collection of a predicate-argument pair

At first, we should collect a predicate-argument pair from the example sentences. v() is a predicate, and t() is its argument.

```
is_a(v(demand-attention),t(child)).   is_a(v(demand-attention),t(puppy)).
is_a(v(demand-attention),t(baby)).   is_a(v(eat),t(child)).
is_a(v(eat),t(puppy)).   is_a(v(eat),t(baby)).
is_a(v(eat),t(adult)).   is_a(v(eat),t(tiger)).
is_a(v(eat),t(microbe)). is_a(v(move),t(child)).
is_a(v(move),t(puppy)).     is_a(v(move),t(baby)).
is_a(v(move),t(adult)).     is_a(v(move),t(tiger)).
is_a(v(move),t(microbe)).   is_a(v(move),t(machine)).
is_a(v(move),t(car)).       is_a(v(move),t(train)).
is_a(v(exist),t(child)). is_a(v(exist),t(puppy)).
is_a(v(exist),t(baby)). is_a(v(exist),t(adult)).
is_a(v(exist),t(tiger)).
is_a(v(exist),t(microbe)).
is_a(v(exist),t(machine)). is_a(v(exist),t(car)).
is_a(v(exist),t(train)). is_a(v(breakdown),t(machine)).
is_a(v(breakdown),t(car)).   is_a(v(breakdown),t(train)).
is_a(v(laugh),t(child)).     is_a(v(laugh),t(puppy)).
is_a(v(laugh),t(baby)).     is_a(v(laugh),t(adult)).
is_a(v(laugh),t(tiger)).        is_a(v(borne),t(child)).
is_a(v(borne),t(puppy)). is_a(v(borne),t(baby)).
is_a(v(borne),t(tiger)).
```

3.4 Example

From the relations above, we will get a new relation as follows. n() is a new node generated.

```
is_a(v(exist),n(16)). is_a(v(move),n(16)).
therefore, n(16) exists and moves
     lower node: n(7),n(15)
is_a(v(eat),n(15)).
therefore n(15) X eat
```

```
     lower node: n(13),t(microbe)
is_a(v(laugh),n(13)).
therefore n(13) laugh
     lower node: n(12),t(adult)
is_a(v(borne),n(12)).
therefore n(12) borne
     lower node n(5),t(tiger)
is_a(v(demand-attention),n(5)).
therefore n(5) demand-attention
     lower node t(child),t(puppy),t(baby)
is_a(v(breakdown),n(5)).
therefore n(7) breakdown
     lower node t(machine),t(car),t(train)
```

From the results mentioned above, relations including only n() and t() as their arguments are selected as illustrated on the following page.

Each node X in SECT is characterized by V in a relation "is_a(v(V), X)". For example, n(12) and its descendant nodes can be the SUBJECT of 「borne」 (be born), and these nodes are in one of the subclasses of n(13), which can 「laugh」 (laugh). Moreover, things which can 「laugh」 (laugh), are in one of the subclasses of n(15), which can 「eat」 (eat) and 「move」 (move).

As mentioned above, this categorization is automatically given by direct relations of predicates and their arguments, and each node has no so-called semantic ambiguity.

We now show the rewriting procedure to get these result.

3.4.1 OBJECT

The object of rewriting is a set of predicates formed with

$$\text{Object} \overset{def}{=} \{X | X = is\_a(A, B)\} \tag{6}$$

The character $is\_a(v(X), t(Y))$. cannot be rewritten by rewriting, where v() is a predicate and t(Y) is its argument. In short, t(Y), which can be reached from v(X) using (7) and (8), cannot be changed by rewriting.

$$is\_a(A, A) \tag{7}$$
$$is\_a(A, B) \leftarrow \quad is\_a(A, C) \wedge is\_a(C, B). \tag{8}$$

3.4.2 Random Method

We show here the random rewriting method. This method has no specific direction (top-down, bottom-up), and is a procedure to reduce the number of the links.

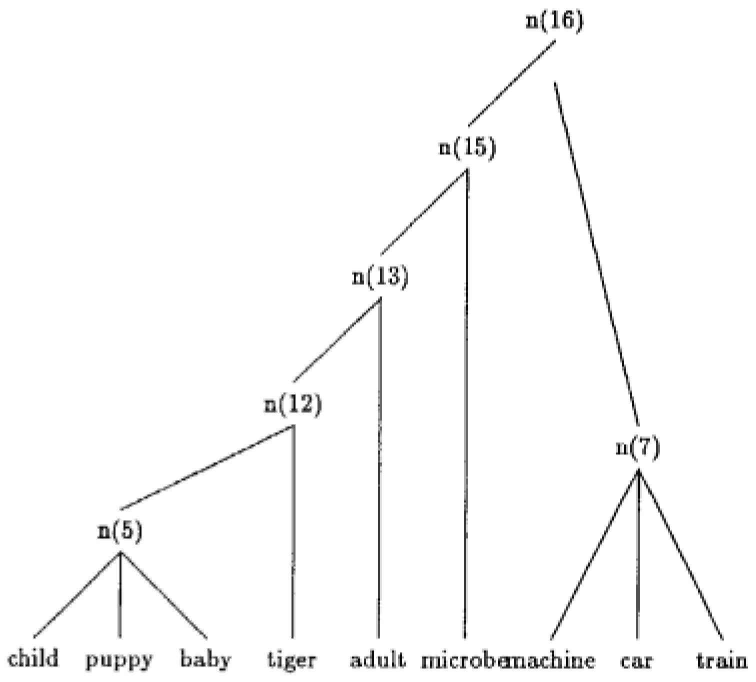
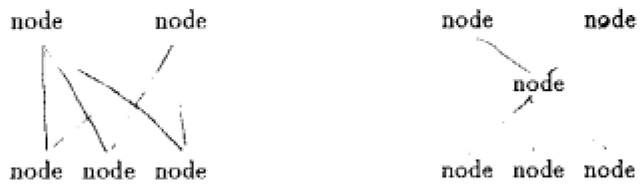


Figure 1: Generated semantic codes

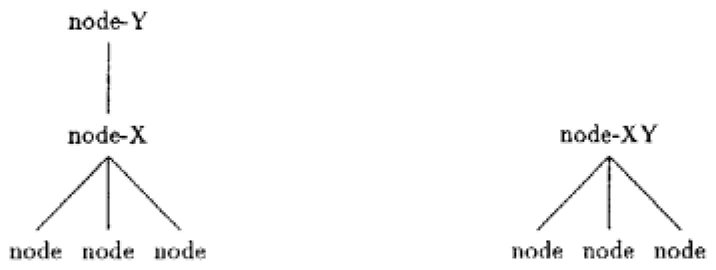
3.4.3 Algorithm

1. Find two nodes (P_nodes1) sharing more than 2 nodes as their children node. If you can't, exit.
2. Make one node as a child of P_nodes1 and as a parent of C_nodes1. And, delete links from P_nodes1 to C_nodes1.
3. If there is a node with one child and more than one parent node, rewrite this node and its child node to one new node.
4. If there is a node with more than one child and one parent node, rewrite this node and its parent node to one new node.
5. goto 1

with 1–2, rewriting is done as follows:



with 3–4, rewriting is done as follows:



This random algorithm can be regarded as a rewriting procedure to rewrite sets with their denotations to common subsets.

In short, proc. 1 finds out two sets sharing a common part, and proc. 2 expresses this common part as one set. With proc. 3 and 4, redundant sets generated from different sets are eliminated.

Once common subsets have been made, bigger common subsets cannot be generated, even if some common subsets can be a bigger one.

This is a reason why the algorithm does NOT have completeness. This is described later.

3.4.4 Completeness of the rewriting rule 1

We shall consider whether the rewriting rule mentioned above can terminate and satisfy the convergence of sets.

We can think of the rewriting rule as the problem of how to rewrite tree structures which have the same leaves while satisfying the restrictions (7) and (8). However, in order to simplify the discussion, now we think of the rewriting object and the rewriting procedure as the operations of sets mentioned above.

One node and the leaves of the tree below it are considered to be a set given its extension. Leaves are an extension of a set and nodes are a set themselves.

The first object W of rewriting is shown below. $V_i (1 \leq i \leq n)$ is a predicate and t_{ij} is an argument of the predicate.

$$\begin{aligned} W &= \{V_1, V_2, V_3, \ldots, V_n\} && (9) \\ V_1 &= \{t_{1a}, \ldots, t_{1m}\} && (10) \\ &\vdots && (11) \\ V_n &= \{t_{na}, \ldots, t_{np}\} && (12) \end{aligned}$$

Also, the restrictions which rewriting rules should satisfy are considered below.

$$\begin{aligned} is\_a(A, B) &\leftarrow A = B. && (13) \\ is\_a(A, B) &\leftarrow A \ni C \wedge is\_a(C, B). && (14) \end{aligned}$$

The restrictions (7) and (8) correspond to (13) and (14) respectively. The "IS_A" is defined below.

$$is\_a(A, B) \overset{def}{=} A \ni B.$$

The symbol $\Rightarrow$ represents rewriting. Rewriting is an operation in which the set representation below is obtained from the set representation above. V_i, V_j are rewritten to V_i', V_j' and then V_k is created.

$$\begin{aligned} V_i \cap V_j &= \{E_1, \ldots E_n\} \wedge n > 1 && (15) \\ &\Rightarrow && (16) \\ V_i' &= \{V_i \cap (V_j)^c, V_k\} && (17) \\ V_j' &= \{V_j \cap (V_i)^c, V_k\} && (18) \\ V_k &= V_i \cap V_j && (19) \end{aligned}$$

If the set A is only an element of the set B, elements of the set A are changed into elements of the set B and then the set A is deleted. This condition is useful not to create an infinite number of representations which satisfy the restriction (14).

The operation which gets the common subset of two sets given their extensions terminates within a finite number of executions. This procedure is the one for getting the common subset of finite sets with finite elements by applying inductive procedures. Therefore, termination is guaranteed.

However, the convergence of the tree structures obtained by this procedure is not guaranteed. One counter example will be sufficient to show this. For example, we shall think about rewriting a structure as below.

$$\begin{aligned} V_2 &= \{2,4,6,12,10,20,30,60\} && (20)\\ V_3 &= \{3,6,9,12,15,30,45,60\} && (21)\\ V_5 &= \{5,10,15,20,25,30,45,60\} && (22) \end{aligned}$$

Although we can get three results from this rewriting, we shall show only two results as below.

The first result If we start the procedure getting the common set (26) of V_2 and V_3, we get the result below.

$$\begin{aligned} V_2' &= \{2,4,V_{23},V_{25}\} && (23)\\ V_3' &= \{3,9,V_{22},V_{35}\} && (24)\\ V_5' &= \{5,25,V_{25},V_{35},V_{235}\} && (25)\\ V_{23} &= \{6,12,V_{235}\} && (26)\\ V_{25} &= \{10,20\} && (27)\\ V_{35} &= \{15,45\} && (28)\\ V_{235} &= \{30,60\} && (29) \end{aligned}$$

The second result If we start the procedure getting the common set (34) of V_2 and V_5, we get the result below.

$$\begin{aligned} V_2' &= \{2,4,V_{23},V_{25}\} && (30)\\ V_3' &= \{3,9,V_{23},V_{35},V_{235}\} && (31)\\ V_5' &= \{5,25,V_{25},V_{35}\} && (32)\\ V_{23} &= \{6,12\} && (33)\\ V_{25} &= \{10,20,V_{235}\} && (34) \end{aligned}$$

$$V_{35} = \{15, 45\} \quad (35)$$
$$V_{235} = \{30, 60\} \quad (36)$$

The confluence of tree structures is not guaranteed as mentioned above. This is caused by the fact that a new node is not created in this algorithm in the case that a node which defines the part already exists, even if a node which has the same characteristics exists.

However, the relation of v() and t() obtained by the "IS_A" operation of the results is not changed.

There are cases where, even if more than one nodes have common elements like (27)(28) above, they are not defined as one node. Using the reasoning rules below, we can check whether any two nodes should be one node.

$$\begin{aligned} parent(A, Parent) &\leftarrow is\_a(Parent, A) \\ parent(A, Parent) &\leftarrow is\_a(X, A) \wedge parent(X, Parent) \end{aligned}$$

If a node "Parent" can be found for node1 and node2 using the above predicate "parent", the pairs of nodes $node_1, node_2$ can be treated as those which indicate one concept.

$$parent(node_1, Parent) \wedge parent(node_2, Parent)$$

3.4.5 Computational complexity of rewriting rule 1

At any time i, it is necessary to check whether a set has a common set with a subset s(i) which was generated before the time i. The number of sub-sets that is generated at a point subsequent to i is S(i). As a result, the maximum number of sub-set is $S(i+1) = S(i) \times 2 + 1$. The solution of the above formula is $S(n) = 2^n - 1$, because S(1) is 1. This result means that the number of sub-sets at a final time point n is 2**n - 1. If a is the computational complexity for every check, the total computational complexity for rewriting rule 1 is $\sum_{i=1}^{n} a(2^n - 1) = a(2^{n+1} - 2 - n)$. It is unusual for all sub-sets to have a common set with other sub-sets. In practice, considering this fact, a heuristic method to decrease the computational complexity is needed. The minimum computational complexity for rewriting rule 1 is n^2, when there is no common set.

3.5 Rewriting by the bottom-up method

We present here the algorithm for generating a semantic code tree from the bottom upwards for above model.

3.5.1 Algorithm

This algorithm includes two procedures. In the first step, the pair is_a(A,B), is transformed to the following vector whose root is B.

$$\begin{array}{rcl}
child, puppy, baby & :: & demand - attention/1, borne/1, eat/1, \\
 & :: & laugh/1, move/1, breakdown/0, exist/1 \\
tiger & :: & demand - attention/0, borne/1, eat/1, \\
 & :: & laugh/1, move/1, breakdown/0, exist/1 \\
adult & :: & demand - attention/0, borne/0, eat/1, \\
 & :: & laugh/1, move/1, breakdown/0, exist/1 \\
microbe & :: & demand - attention/0, borne/0, eat/1, \\
 & :: & laugh/0, move/1, breakdown/0, exist/1 \\
machine, car, train & :: & demand - attention/0, borne/0, eat/0, \\
 & :: & laugh/0, move/1, breakdown/1, exist/1
\end{array}$$

Secondly, the relation between these vectors is transformed in the following procedure.

1. Check the inclusive relation between any two vectors Vi and Vj. If one vector is a real sub-set of another, the vector that is a sub-set is linked to the other vector. If the two vectors Vi and Vj, have no common set, do no operation. If the two vectors have a common set, do the following procedure: Search for the same vector as the above common set. If the same vectors are found, these vectors are linked to the vectors Vi and Vj. If the same vector is not found, enter the common set as a new vector and link it to the vectors Vi and Vj.

2. If there is at least one new vector, set this vector to the object for transformation and return 1. If there is no new vector, stop the procedure.

For example, the result of transformation applied to the above example.

$$\begin{array}{rcll}
child, puppy, baby & :: & demand - attention/1, borne/1, eat/1, & (37) \\
 & :: & laugh/1, move/1, breakdown/0, exist/1 & \\
 & => & & \\
tiger & :: & demand - attention/0, borne/1, eat/1, & (38) \\
 & :: & laugh/1, move/1, breakdown/0, exist/1 & \\
 & => & (37) & \\
adult & :: & demand - attention/0, borne/0, eat/1, & (39) \\
 & :: & laugh/1, move/1, breakdown/0, exist/1 & \\
 & => & (38) &
\end{array}$$

$$\begin{array}{rcll} microbe & :: & demand-attention/0, borne/0, eat/1, & (40) \\ & :: & laugh/0, move/1, breakdown/0, exist/1 & \\ & => & (39) & \\ machine, car, train & :: & demand-attention/0, borne/0, eat/0, & (41) \\ & :: & laugh/0, move/1, breakdown/1, exist/1 & \\ & => & & \\ newvector & :: & demand-attention/0, borne/0, eat/0, & (42) \\ & :: & laugh/0, move/1, breakdown/1, exist/1 & \\ & => & (41), (40) & \end{array}$$

Note that in order to implement the procedure characteristic of this algorithm that reduces the number of links, we only have to generate a new vector when there is more than one element in the intersection of two vectors (that is, verbs).

3.5.2 Completeness of the bottom-up method

Unlike rewriting rule 1, this procedure is complete. This is because all the relationships among the subsets are calculated.

3.5.3 Computational complexity of bottom-up method

The computational complexity for the bottom-up method is the same as for rewriting rule 1. However, because the elements to be calculated can be handled as a vector, the constant terms in an equation can be reduced.

3.6 Rewriting by top-down method

We present here the algorithm for generating semantic code trees from top to bottom for the above model.

3.6.1 Algorithm

The procedure consists of two sub procedures. First, all the sets of is_a(A,B) relations are converted into these vectors for example:

$$\begin{array}{rcl} demand-attention & :: & child/1, puppy/1, baby/1, \\ & :: & tiger/0, adult/0, microbe/0, \\ & :: & machine/0, cat/0, train/0 \\ borne & :: & child/1, puppy/1, baby/1, \\ & :: & tiger/1, adult/0, microbe/0, \end{array}$$

$$
\begin{array}{rcl}
 & :: & machine/0, car/0, train/0 \\
laugh & :: & child/1, puppy/1, baby/1, \\
 & :: & tiger/1, adult/1, microbe/0, \\
 & :: & machine/0, car/0, train/0 \\
eat & :: & child/1, puppy/1, baby/1, \\
 & :: & tiger/1, adult/1, microbe/1, \\
 & :: & machine/0, car/0, train/0 \\
move, exist & :: & child/1, puppy/1, baby/1, \\
 & :: & tiger/1, adult/1, microbe/1, \\
 & :: & machine/1, car/1, train/1 \\
breakdown & :: & child/0, puppy/0, baby/0, \\
 & :: & tiger/0, adult/0, microbe/0, \\
 & :: & machine/1, car/1, train/1
\end{array}
$$

Next, the relationships among these vectors are calculated with the procedure below.

1. With any vectors V_I, V_J selected, the inclusion relation is investigated. If one is a proper subset of another, a link is set up from the vector which is a larger set to the proper subset. If there are no intersections, do nothing. If there is an intersection, see whether there are any vectors the same as that intersection. If there are, the pointers are linked to that vector from each of the vectors containing the intersection. If there are no such vectors, the intersection is registered as a new vector and the pointers are linked to the intersection from each vector.

2. If there is at least one newly registered vector, this is added as the vectors to be rewritten and the procedure returns to 1 . If not, the algorithm terminates.

For example, the following are the results when rewriting procedure 1 is applied.

$$
\begin{array}{rcll}
demand-attention & :: & child/1, puppy/1, baby/1, & (43) \\
 & :: & tiger/0, adult/0, microbe/0, & \\
 & :: & machine/0, car/0, train/0 & \\
 & => & & \\
borne & :: & child/1, puppy/1, baby/1, & (44) \\
 & :: & tiger/1, adult/0, microbe/0, & \\
 & :: & machine/0, car/0, train/0 &
\end{array}
$$

$$\begin{array}{rcll}
 & => & (43) & \\
\mathit{laugh} & :: & \mathit{child}/1, \mathit{puppy}/1, \mathit{baby}/1, & (45)\\
 & :: & \mathit{tiger}/1, \mathit{adult}/1, \mathit{microbe}/0, & \\
 & :: & \mathit{machine}/0, \mathit{car}/0, \mathit{train}/0 & \\
 & => & (44) & \\
\mathit{eat} & :: & \mathit{child}/1, \mathit{puppy}/1, \mathit{baby}/1, & (46)\\
 & :: & \mathit{tiger}/1, \mathit{adult}/1, \mathit{microbe}/1, & \\
 & :: & \mathit{machine}/0, \mathit{car}/0, \mathit{train}/0 & \\
 & => & (45) & \\
\mathit{move}, \mathit{exist} & :: & \mathit{child}/1, \mathit{puppy}/1, \mathit{baby}/1, & (47)\\
 & :: & \mathit{tiger}/1, \mathit{adult}/1, \mathit{microbe}/1, & \\
 & :: & \mathit{machine}/1, \mathit{car}/1, \mathit{train}/1 & \\
 & => & (46)(48) & \\
\mathit{breakdown} & :: & \mathit{child}/0, \mathit{puppy}/0, \mathit{baby}/0, & (48)\\
 & :: & \mathit{tiger}/0, \mathit{adult}/0, \mathit{microbe}/0, & \\
 & :: & \mathit{machine}/1, \mathit{car}/1, \mathit{train}/1 & \\
 & => & &
\end{array}$$

Note that in order to implement the procedure characteristic of this algorithm that reduces the number of links, we only have to generate a new vector when there is more than one element in the intersection of two vectors (that is, nouns).

3.6.2 Completeness of the top-down method

Unlike rewriting rule 1, this procedure is complete. This is because all the relationships among the subsets are calculated.

3.6.3 Computational Complexity of the top-down method

Computational complexity is the same as that of the rewriting rule 1 . But the vectorization of the elements to be calculated realizes the reduction of constant terms.

4 Experiment

We carried out an experiment to construct a medium size thesaurus with this algorithm. The most frequently used 3000 pairs were selected in advance, as the terms which can be predicates and their arguments . These pairs were the sources for rewriting.

Table 1: Rewriting time

Algorithm	Time	Implementation language
Random	80 hours/3000 pairs	Prolog
Top Down	30 hours / 3000 pairs	Prolog,C
Bottom Up	30 hours / 3000 pairs	Prolog,C

Prolog was used for the calculation of vectors in the top-down and bottom-up fashion, and C was used for the calculation of the dependency among vectors. The depth of the semantic code system which was obtained with this rewriting algorithm was 18. The following is a part of the results.

```
n(1)0:v(ある-exist);v(いる-exist);
   t(類 たぐい-kind)1:
   n(11)1:v(取り替える-exchange);
      n(10)2:v(降る-fall);
         t(そっち そっち-there)3:n(320);
   n(21)1:v(くたびれる-tired);
      n(20)2:v(かみ付く-bite);
         n(19)3:v(うなだれる-deject);
            n(18)4:v(かき集める-scrape up);
               t(自身 じしん-self)5:n(334);n(369);n(390);v(取りこむ-capture);
               n(24)5:v(けしかける-urge);
                  t(多数 たすう-majority)6:v(治す-cure);
                  n(26)6:v(こぼす-spill);
                     t(後任 こうにん-successor)7:v(取り替える-chenge);
                     n(35)7:v(辞する-quit);
                        t(小数 しょうすう-minority)8:
                        n(48)8:v(治す-cure);v(借りる-borrow);v(弱める-weaken);
                           t(過半数 かはんすう-majority)9:v(取り替える-change);
                           t(尼 あま-nun)9:
                     n(38)7:n(337);
                        n(37)8:v(腰掛ける-sit);
                           n(59)9:n(358);n(371);
                              t(海軍 かいぐん-navy)10:n(361);
                              n(64)10:v(弱める-weaken);
                                 t(敵 かたき-enemy)11:
                                 t(両親 りょうしん-parents)11:n(370);n(387);
                           n(133)9:n(356);n(392);
                              t(売り手 うりて-seller)10:n(368);
                              n(136)10:v(降る-fall);v(降ろす-lower);v(取り外す-detach);
                                 n(132)11:v(差し出す-present);
                                    n(118)12:n(365);
                                       t(親父 おやじ-father)13:ar child)15:
```

5 Characteristics of SECT

Below, we consider the characteristics of SECT and, especially, in relation to learning.

5.1 The capacity of the tree structure

Using the random method for rewriting, the number of ordered-pairs that express the tree will be less than before rewriting. When two parent-nodes share two child-nodes, the number is equal to that before rewriting.

If we assume that a man makes a new concept from the usage of a word when he is learning a language, we can say that this method is an embodiment of human action. In other words, if we assume that a man learns every word usage and after he has learned an amount of various usages, he arranges these usages to form some groups of words, this random method, that reduces words only by the relation of the kind of predicates to their arguments, may simulate human action.

5.2 Specification of the tree structure

In SECT, the more predicates and their arguments that are provided, the more complex the tree structure that includes the word becomes. This characteristic may involve the use of learning.

6 conclusion

We have described a method for an automatic semantic code generating system. We have constructed a semantic code system based on this method. Though we are going to use the code system obtained in this method for natural language analysis, we must adopt a method for node reduction that resembles this method in order to use this code system as the semantic code system for natural language analysis. Moreover, we need to consider specifically which pairs should be selected amongst a large quantity of pairs of predicates and their arguments.

We would also like to consider this method from the point of view of cognitive science in addition to the approach above.

References

[1] Barwise,J. and Perry,J., Situations and attitudes, MIT Press, Cambridge 1983

[2] J. Barwise. Recent Developments in Situation Semantics. In M. Nagao, editor, *Language and Artificial Intelligence*, pages 387–399, Amsterdam, 1987. North-Holland.

[3] Bunrui-Goi-Hyou (in Japanese) The National Language Research Institute

[4] Susumu OHNO, Masato HAMANISHI, Ruigo-shin-jiten (in Japanese) 1981

[5] Makoto NAGAO Gengo-Kougaku (in Japanese) Sho-ko-do pp.147–173,1983

[6] Seiichi NAKAGAWA, Mikio YAMAMOTO, Kazuaki WAKAHARA, An Inductive Learning System of Syntactic and Semantic Analysis Rules of Natural Language Processing Trans. JEICA, vol.30,No.1, pp.72–80,1989

[7] Japan Electronic Dictionary Research Institute, Ltd. EDR Densi-ka jisho (in Japanese) TR-016,EDR, 1989

[8] Takenobu TOKUNAGA, Makoto IWAYAMA, Tadashi KAMIWAKI, Hozumi TANAKA LangLAB: A Natural Language Analysis System, Trans. JEICA, vol.29,No.7, pp.703-711,1988

[9] Takenobu TOKUNAGA, Manabu OKUMURA, Hozumi TANAKA, Introducing Views to the Conceptual Hierarchy Trans. JEICA, vol.29,No.7, pp.970-975, 1989

Dynamics and Flexible Inference

HASIDA, Kôiti
Institute for New Generation Computer Technology (ICOT)*

Abstract

Dynamical logic (not *dynamic* logic) is a system of logic whose semantics is regarded as a sort of dynamics. The declarative semantics is defined by formulating the degree of violation in terms of *potential energy*. A control scheme for inference also emerges thereof through energy minimization principle, which does not commit us to untractable computation for consistency maintenance, among others. This inborn integration of declarative semantics and inference control guarantees that inferences are sensitive to semantic relevance; that is, to context. The proposed control scheme engenders various aspects of language processing, including syntactic parsing, semantic and pragmatic interpretation, sentence generation, and so on, without any specific procedures for the different tasks.

1 Introduction

Information processing in language use is quite partial at each context. People refer to only a small part of possibly relevant information, and carry out just a tiny subset of the possible inferences pertaining to that partial information. This *partiality of information* gives rise to very diverse contexts and accordingly diverse patterns of information flow. That is, the range of information processing must drastically change from one situation to another, in order for the information exploited in total to encompass as much of the potentially relevant information as possible.

So the traditional sequential architecture for natural language processing (typically, a sequence of syntactic analysis, semantic analysis, pragmatic analysis, extralinguistic inference, generation planning, and surface generation) is unpromising, as widely recognized, because it imposes too strong static restriction on the way information flows. In fact, syntactic analysis does not totally precede semantic or pragmatic comprehension. Generation planning and surface generation are much more intertwined than this architecture presumes, as pointed out by Danlos [2], among others.

A promising approach to capture the diversity of information flow with a tractable complexity of the information processing system is to design the system in terms of *constraint*; that is, without stipulating any domain/task-dependent information flow. The resulting flow of information will be complex and non-modular, but the underlying architecture itself might still be modular, with modules such as syntactic constraint, semantic constraint, pragmatic constraint, and extralinguistic constraint.

In the present paper we consider *dynamical logic* (not *dynamic* logic) as a formalism of constraint. In this logic, the declarative semantics is defined by measuring the degree of violation of the constraint in terms of real-valued *potential energy*. From the linguistic point of view, this enables us to capture the grammaticality or acceptability of sentences and discourses in a graded manner. From a computational point of view, this provides a very robust system of constraint

*The author is currently at Natural Language Section, Electrotechnical Laboratory, 1-1-4 Umezono, Tukuba, Ibaraki 305 JAPAN.

which practically never becomes absolutely inconsistent. So we are not committed to global consistency maintenance, which is intractable in a powerful system, such as first-order logic, that is necessary in order to account for language use.

Furthermore, the graded notion of violation as potential energy provides some control scheme for inferences on the basis of energy minimization principle. So the dynamics provides both declarative semantics and inference control at the same time. Such an inborn integration of semantics and inference could guarantee systematic reflection of semantic relevance in inferences. An inference method based on a dynamics will be shown to derive computational methods tailored so far in natural language processing, such as marker passing [1, 5], weighted abduction [4, 8], and semantic head-driven generation [7].

2 Constraint Network

A constraint is a set of *clauses*. A clause is basically a disjunction of *literals*. A *literal* is an *atomic constraint* preceded by a sign. An atomic constraint is an *atomic formula* such as p(X,Y,Z)[1] or an *equality* such as X=Y. Signs are '+' and '-'. For any atomic formula α, literal $+\alpha$ stands for just α, and $-\alpha$ stands for $\neg\alpha$. Names beginning with capital letters represent variables, and the other names predicates. A clause is written as a sequence of the included literals followed by a period. The order among literals is not significant. So (1) and (2) represent the same clause, which means (3) in a rough, crisp approximation.

(1) -p(U,Y) +q(Z) -U=f(X) -X=Z.

(2) +q(Z) -p(f(Z),Y).

(3) $\forall U, X, Y\ \{\neg p(U, Y) \vee q(X) \vee U \neq f(X)\}$

There is only one clause, called the *top clause*, containing literal **+true**. The top clause corresponds to the query in Prolog.[2] That is, top clause (4) represents top-level hypothesis (5).

(4) **+true** -p(X) +q(X,Y).

(5) $\exists X, Y\{p(X)\ \wedge\ \neg q(X, Y)\}$

The computation is to tailor the best hypothesis to explain the top-level one. The top clause may change as computation proceeds, in particular when interactions with the world take place.

A constraint is regarded as a network. For instance, the following constraint may be graphically shown as in Figure 1.

(i) **+true** -p(A) -q(B).

(ii) +p(X) -r(X,Y) -p(Y).

(iii) +r(X,Y) -q(X).

In such a graphical representation, a clause is a closed domain containing the atomic constraints constituting that clause. Atomic constraints without such indication are referred to as negative literals in clauses. An argument of an atomic formula is shown either as a '•' or as an identifier. Equalities between arguments are links. Equalities in clauses are called *intraclausal equalities*, and those outside of clauses are called *extraclausal equalities*.

[1] A binding is also regarded as an atomic formula. For example, X=f(Y) is an atomic formula with binary predicate =f.

[2] Theoretically, Prolog uses false instead of true here. The reason why we use true will be understood on the basis of exclusion energy to be discussed later.

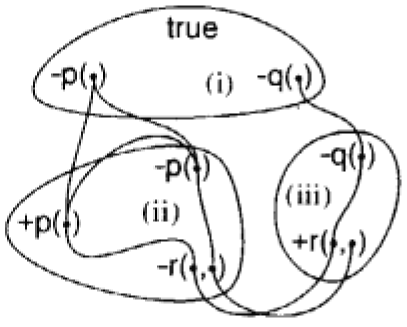


Figure 1: Constraint Network.

It is important to note that the constraint network contains no objects corresponding to literals apart from atomic formulas. Positive and negative literals are just two types of manifestations of atomic formulas. So for any atomic formula α in the constraint, literals $+\alpha$ and $-\alpha$ both exist even if one of them appears in no clause. Also, α may appear in two different clauses as two literals.

We will write $\alpha \circ \beta$ to mean that atomic formulas α and β are unifiable. We consider that two literals are unifiable iff they consist of the same sign and unifiable atomic formulas. We regard each part of constraint network as a set of instances, and $\alpha \circ \beta$ as meaning that $I(\alpha) \cap I(\beta) \neq \emptyset$ is possible. I is an interpretation function which maps sets of instances to sets of objects (state of affairs, in the case of atomic formulas) in the world. So unifiability is not transitive. We assume two atomic formulas are unifiable if and only if their corresponding arguments are directly connected through an extraclausal equality, and that every extraclausal equality connects two corresponding arguments of two unifiable atomic formulas. For each zero-ary predicate, the constraint network contains only one atomic formula with it. Also, for each constant term, there is only one binding to it.

We assume that initially all the atomic formulas with the same predicate are unifiable with each other. So at the beginning the extraclausal equalities constitute a complete graph for every argument place of every predicate.[3] Such a configuration changes as symbolic computation proceeds in the way discussed later.

3 Dynamics

Now we introduce a dynamics to define the declarative semantics of the constraint network described above. Each atomic constraint α has an *activation value* x_α, which is a real number such that $0 < x_\alpha < 1$ and may be regarded as the truth value (or a subjective probability of the truth) of α. The activation values of literals are defined so that $x_{+\alpha} = x_\alpha$ and $x_{-\alpha} = (1 - x_\alpha)$ for any atomic constraint α. The *potential energy* U of the entire constraint network is a function of the activation values, and represents the degree of violation of the constraint. U gives rise to a *field of force* to change the state of the system so as to decrease U.

Suppose there are n distinct atomic constraints in the given constraint, and hence n activation values, x_1 through x_n. Then the current state of the system is regarded as a point (6) in the n-dimensional Euclidean space, and the global potential energy U defines a field of force (7).[4]

[3]There can hence be $O(N^2)$ extraclausal equalities, for N different atomic formulas sharing the same predicate. So an efficient encoding schema would be necessary to avoid that space complexity.

[4]For any matrix A, ${}^tA$ is the transposition of A. Incidentally, another formulation of dynamics might include

(6) $$\vec{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$$ (7) $$\vec{F} = -\mathrm{grad}U = \begin{pmatrix} -\frac{\partial U}{\partial x_1} \\ \vdots \\ -\frac{\partial U}{\partial x_n} \end{pmatrix}$$

$\vec{F}$ causes *spreading activation*: when $\vec{F} \neq \vec{0}$, a change of x_i so as to reduce U influences the neighboring parts of the constraint network, which causes further changes of activation values there, and thus state transition propagates across the network. In the long run, the assignment of the activation values will settle upon a stable equilibrium with $\vec{F} = \vec{0}$. The resulting state gives a minimal value of U.[5] That is, the resultant $\vec{x}$ satisfies the constraint best in some neighborhood.

The declarative semantics of the entire constraint is decomposed into several aspects. U is the sum of the local energies each representing one such aspect, so that U captures the global declarative semantics. The types of energy are *disjunction energy, exclusion energy, completion energy, assimilation energy, transitivity energy, binding energy*, and *normalization energy*.

The *disjunction energy* of a clause captures the disjunction of the literals: at least one literal should be true in a clause. Consider the following clause.

(8) **-p +q**.

The disjunctive meaning of this clause is that either **-p** or **+q** should be true. The below disjunction energy represents the degree of violation of this meaning.

(9) $$Dx_{\mathbf{p}}(1 - x_{\mathbf{q}})$$

D is a positive constant associated with clause (8). Note that (9) is small iff either $x_{\mathbf{p}}$ or $(1 - x_{\mathbf{q}})$ is small; keep in mind that the activation values are between 0 and 1. In dynamic terms, **q** is excited by **p** to the extent that **p** is excited, and **p** is inhibited by **q** to the extent that **q** is inhibited.

Exclusion energy represents the mutual exclusion of the literals in a clause, by which we mean that at most one literal should be true. So in (8), for example, only one of **-p** and **+q** may be true. This supports abductive inferences, to assume **p** when given **q**, and assume **-q** when given **-p**. The exclusion energy of (8) is the following formula.

(10) $$E(1 - x_{\mathbf{p}})x_{\mathbf{q}}$$

E is a constant associated with clause (8). If **q** means that you are in Japan, for instance, E is larger when **p** means that you are in Tokyo than when it means that you are in Imabari, a small city in the island of Sikoku. Incidentally, it is due to exclusion energy that top clause (4) means (5).

Henceforth we do not spell out energy functions any more, because mathematical details are not very important here. The definitions of the energies are found in Appendix.

In the ordinary practice, two atomic formulas with the same predicate are equivalent if they share the same arguments for the corresponding argument places. The *assimilation energy* between two unifiable atomic formulas captures this in a relaxed fashion: two unifiable atomic formulas should have similar truth values to the extent that all the extraclausal equalities between them are excited. So for instance **p(X,Y)** and **p(U,V)** tend to have similar activation values when the extraclausal equality between **X** and **U** and that between **Y** and **V** are both strongly excited.

additional field of force, so that $\vec{F} = \vec{f} - \mathrm{grad}U$.

[5] When $\vec{F}$ is not entirely attributed to U, spreading activation is not directly linked with minimalization of U.

The assimilation energy between α and β has a constant factor $(s_{\alpha\beta} + s_{\beta\alpha})$. $s_{\alpha\beta}$ is a non-negative constant called the *subsumption coefficient* of α as to β. We say α *subsumes* β to mean $I(\alpha) \supseteq I(\beta)$. When $\alpha \circ \beta$, $s_{\alpha\beta} = 1$ if α subsumes β, and otherwise $s_{\alpha\beta} = s_0$ for a small positive constant s_0. So α and β strongly influence each other when they are in subsumption relation. When $\alpha \circ \beta$ is false, $s_{\alpha\beta} = 0$.

The three types of energy introduced so far account for marker passing [1, 5] as an emergent property of the dynamics. Consider the following discourse for example.

(11) Taro got a book. He paid one thousand yen.

Figure 2 shows the network involved in the abductive inference to assume that Taro bought

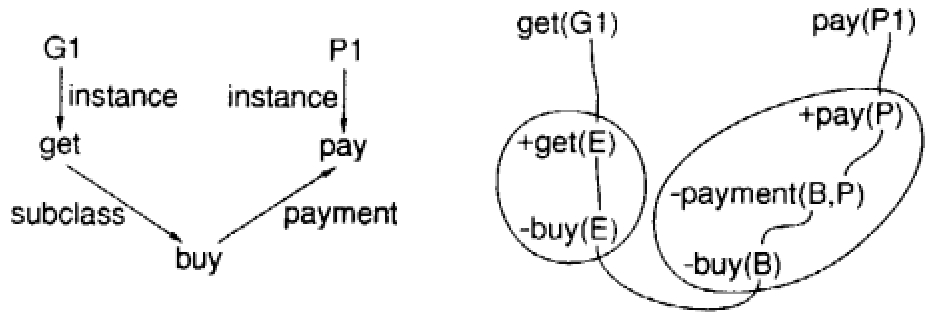


Figure 2: Marker Passing for (11).

the book. In the left is the marker-passing network encoded[6] by the constraint network in the right. A node in marker passing network corresponds to an argument or a predicate in our constraint. A directed edge from an argument node to a predicate node represents that the argument satisfies the predicate, and a link between two predicate nodes represent a clause referring to the two predicates. The directions of the edges are irrelevant to the direction of marker passing. **get(G1)** and **pay(P1)** are created upon reading/hearing (11), where **G1** and **P1** stand for the event of Taro's getting a book and that of his paying money, respectively. In marker passing, the abductive inference of Taro's buying the book will be suggested by a collision of markers passed down from **G1** and **P1** along the path between them.

In our framework, if the activation value of **get(G1)** is large, then it excites **get(E)** due to assimilation energy, **get(E)** excites **buy(E)** due to exclusion energy, **buy(E)** excites **buy(B)** due to assimilation energy, and **buy(B)** excites **pay(P)** due to disjunction energy. **get(E)** is similarly excited indirectly by **pay(P1)**. So **get(E)**, **buy(E)**, **buy(B)** and **pay(P)** are excited stronger than when there were no such path. As will be discussed in the next section, the abductive inference to instantiate the two clauses on the path are expected to cause stronger excitation of **get(G1)** and **pay(P1)**, and hence is strongly preferred.

Of course how much a path contributes to such analog inferences depends on the dynamical properties of the path. For instance, the path in Figure 2 would not give rise to the above marker-passing-like inference if the exclusion energies of the two clauses have much smaller effects.[7] Strength of analog inference also depends on the length of the path. Obviously, inference more readily go through shorter paths.

Now let us get back on track and discuss other types of energy. A *transitive cycle* is a cycle of equalities $\Delta = \delta_0\delta_1\cdots\delta_{k-1}$ where either $\delta_{(i-1)\bmod k}$ or $\delta_{i \bmod k}$ is intraclausal for every i.[8]

[6] Charniak [1] employs a similar encoding scheme.

[7] What Charniak [1] calls isa-plateau can be understood along the same line.

[8] We may take into account only such cycles, since two unifiable atomic formulas always have their arguments directly connected with each other through extraclausal equalities.

Transitivity of equality as to Δ is regarded as excluding the cases where just one equality in Δ is false. We postulate the *transitivity energy* of Δ to capture this. Since detection of cycles is a very costly computation, we will have to consider some approximate method for efficient processing of transitivity energy instead of guaranteeing perfect detection of transitive cycles. We do not go further into such implementation details.

Transitivity energy accounts for some standard inferences. Let us consider the following discourse.

(12) Tom bought a telescope. He saw a girl with it.

We assume that *he* and *it* in the second sentence are anaphoric with *Tom* and *the telescope*, respectively, in the first sentence. There is an attachment ambiguity in the second sentence, about whether the prepositional phrase *with it* modifies *saw* or *a girl*. Let us assume that the structure of the constraint generated by processing this discourse looks like Figure 3. Each

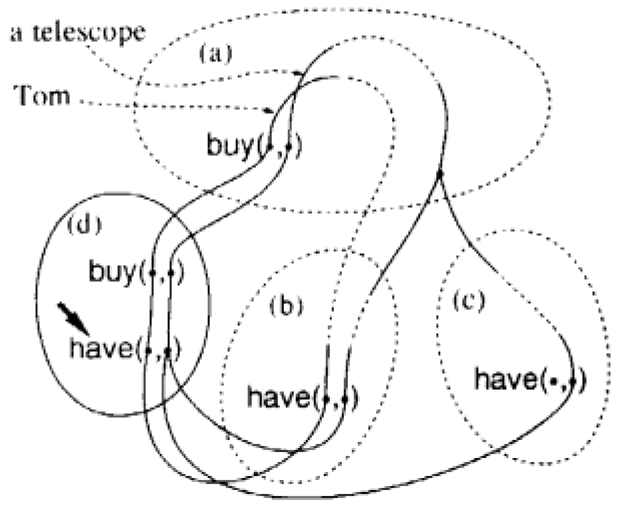


Figure 3: Semantic Association Concerning (12).

region in a dashed closed curve represents a cluster of clauses. These clauses have been created by symbolic inference as described in the next section. (a) is a set of clauses including the top clause. (b) and (c) represent the two alternative readings of the second sentence of (12), each derived by abductive inferences. The **buy(•,•)** in (a) is a part of the hypothesis obtained by interpreting the first sentence. Its first argument stands for Tom and the second the telescope, so that the whole thing means that Tom buys the telescope at some time. Thus, reading (b) means that Tom has the telescope when he sees the girl, and (c) that the girl has it when Tom sees her. Clause (d) is an inference rule to the effect that if A buys B then A comes to have B.[9] Due to this inference rule, the **buy(•,•)** in (a) can imply the **have(•,•)** in (b) but not that in (c), so (b) is more plausible than (c).

This difference between (b) and (c) is captured by transitivity energy. Note that there are two transitive cycles both going through the **buy(•,•)**s in (a) and (d). So these two atomic formulas tend to strongly excite each other due to assimilation energy, provided that every relevant equality is excited. These two cycles also both go through the **have(•,•)**s in (b) and (d), making them tend to strongly excite each other, too. On the other hand, there is only one transitive cycle which goes through both the **have(•,•)**s in (c) and (d). Hence the associative

[9]We ignore the temporal relation between the taking and the having here.

inference based on the **buy(•,•)** in (a) through (d) supports the **have(•,•)** in (b) more strongly than it supports the **have(•,•)** in (c).

Before going to symbolic inference, let us introduce one more type of energy. A *completion energy* of a literal captures a somewhat extended notion of completion: To complete literal (not predicate) ξ means that ξ should be inferred either deductively or abductively[10] on the basis other than the one on which ξ was first postulated. In more concrete terms, the completion energy of ξ is small so that ξ may get easily excited, only when there exists a strongly excited literal η subsumed by ξ (hence $s_{\xi\eta} = 1$).[11] For example, +q(X) is completed and somehow postulated (say, based on clause +p(X) -q(X).. abductively), then it should be inferred from another reason, such as by subsuming another literal +q(Y) inferred on the basis of a clause such as +q(Y) -r(Y). deductively or -q(Y) +s(Y). abductively. Completion energy implements assumability cost [4], as discussed more in the next section.

4 Symbolic Inference

Here we consider just one type of symbolic operation called *subsumption*. It is a sort of program transformation to create a new subsumption *relation*. A subsumption *operation* concerns a pair of unifiable atomic formulas. A subsumption operation from atomic formula α to β is regarded as an application of the rule Φ to α, where Φ is a clause containing β. As expected, Φ and hence β are instantiated to Φ' and β', respectively, and α and β' are connected more tightly than α and β used to be. In the usual terminology of resolution theorem proving, subsumption corresponds to resolution if α and β appear as literals with opposite signs; otherwise it corresponds to factoring. Unlike in the ordinary resolution and factoring, however, in subsumption α just subsumes β' rather than unifying with it. Note that α can subsume another atomic formula γ in addition to β, even if β and γ are not unifiable. This enables OR-parallel inference concerning α.

A subsumption from α to β is shown in Figure 4. β is divided into β' and β''. β' is the

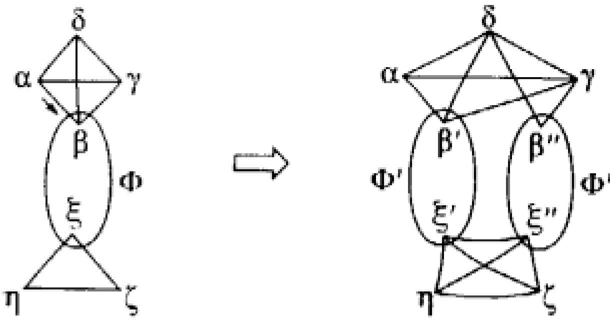


Figure 4: Subsumption Operation From Atomic Formula α to β.

maximum subset of β subsumed by α. We assume $\beta'' = \beta - \beta'$, so that $I(\beta'') \cap I(\alpha) = \emptyset$. Neither α nor β' is hence unifiable with β'', as indicated in the figure. If it is somehow known that α subsumes β from the beginning, then no copy (division) need to happen. When the division of β actually takes place, then it causes a duplication of the clause Φ, atomic formula ξ in it accordingly dividing into ξ' and ξ''.[12] Unlike in the division of β, ξ' and ξ'' are unifiable both

[10] In this respect, only deduction is considered in the standard completion.

[11] In the tentative definition in Appendix, the completion energy contains $(1 - s_{\xi\eta}x_\eta)$ as a factor.

[12] If α and β belonged to the same clause, then α is also divided into α' and α''. Let α' and β' belong to one clause and hence α'' and β'' belong to another, then α' and β'' subsume each other and α'' and β'' are not unifiable.

with each other and with all the atomic formulas unifiable with ξ, because there is no reason to believe $I(\xi') \cap I(\xi'') = \emptyset$ etc.

We omit further details of combinatorial aspects of symbolic inference, due to the space limitation, and go on to the dynamical aspect. Subsumption generates new atomic constraints and thus redefines U. $s_{\alpha\beta'}$ is set to 1, because α subsumes β'. $s_{\xi'\xi''}$ and $s_{\xi''\xi'}$ are both set to s_0, because we are not sure about the subsumption relation between ξ' and ξ''. The other coefficients are simply inherited along with the copy of the part of the constraint network.

Since subsumption is a local operation, it may take place in parallel at many different places. Now we consider how to control such computation based on the dynamics in a distributed fashion: that is, without recourse to any centralized control.

As the preference score for a subsumption, we could use the expected contribution of that subsumption to reduction of a penalty function P. P is defined to be H at the equilibrium of spreading activation due to U that is proximate from the current state of the network. H is an energy (probably a part of U) which measures the degree of unsatisfaction of the top-level hypothesis. So P is regarded as a function of the parameters of U. H could be the disjunction energy of clause +**true**., which indirectly excites the literals in the top-level hypothesis (that is, inhibits the literals other than +**true** in the top clause).

As discussed above, a subsumption from atomic formula α to β divides β into β' and β'', setting $s_{\alpha\beta'}$ to 1. So the expected influence of this subsumption to reduction of P could be estimated by $-\frac{\partial P}{\partial s_{\alpha\beta}}$. By employing generalized backpropagation [6], this value can be efficiently computed for all the subsumption coefficients. The space complexity of that computation is linear with regard to the size of the constraint network, and its parallel time complexity practically constant.

Weighted abduction [1, 8] emerges from our framework. In weighted abduction, just as in the current framework, one attempts to tailor a best hypothesis to explain the observed fact. A hypothesis is a conjunction of literals. Each literal in a hypothesis is assigned an assumability cost, which is a cost of assuming the literal. A hypothesis is better when the total assumability cost is smaller. Assumability cost may be reduced by unifying the literals. For instance, if the current hypothesis contains p(A) and p(B) one of which has a large cost, then this cost will be reduced by unifying them. Assumability cost is inherited through abduction. For example, a cost of p(A) in the current hypothesis is inherited down to q(A) and r(A) when p(A) is resolved by clause +p(X) -q(X) -r(X)..

Assumability cost is basically captured by completion energy, because that energy is to the effect that the literal in question must be inferred otherwise than the way it was first postulated, or it will be inhibited. So an *inherent* cost is encoded by the constant factor of the completion energy. This gives rise to a high preference score of subsumption from the atomic formula α in question, because if α comes to subsume another atomic formula β then probably the completion energy of α is reduced due to $s_{\alpha\beta} = 1$, which will be indicated by a large value of $-\frac{\partial P}{\partial s_{\alpha\beta}}$. An *inherited* cost is captured along the same line. For example, when +p(A) with a large cost subsumes +p(X) in clause +p(X) -q(X) -r(X)., the completion energy of +p(A) may probably be still large, but it will decrease if q(X) and r(X) get more excited. So the preference score of subsumptions from q(X) and r(X) tend to be large, corresponding to the inherited cost in weighted abduction.

Our framework is more flexible and dynamic than weighted abduction. That is, we allow inferences concerning a hypothesis to influence the state of other hypotheses, whereas in weighted abduction assumability costs change only due to unification involving the atomic formulas carrying those costs. So our method is more appropriate to account for phenomena such as belief revision.

Now let us go back to the examples concerning Figure 2 and Figure 3 in the previous section. In the former, the subsumption from get(G1) to get(E) is particularly promising for increasing the activation of **get(G1)** and hence reducing P, because **get(E)** is strongly excited due to the path from **pay(P1)**. Here we assume that literals involved in semantic interpretation of an utterance are at least weakly completed in general. So **get(G1)** is somewhat completed, and hence will get more excited by subsumption. Similar discussion holds for the subsumption from **pay(P1)** to **pay(P)** and also for the one between **buy(E)** and **buy(B)**. Consequently, the abductive inference along the path is highly preferred. In Figure 3, the inference based on clause (d) is more readily applied to the interpretation (b) than to (c), because the former contributes more to the reduction of P.

Now let us consider sentence generation for a little more complex example. We assume the initial state of computation is given by a constraint including the following clauses, among much more others.

(A) +**true** -**s**(RUN,W0,W1) -**run**(RUN)$^{\$}$ -**agt**(RUN,kim)$^{\$}$ ⋯.

(B) +**s**(SEM,X,Z) -**np**(SBJSEM,X,Y) -**vp**(SEM,SBJSEM,Y,Z).

(C) +**np**(kim,X,Y) -X=['kim'|Y].

(D) +**vp**(R,AGT,X,Y) -X=['runs'|Y] -**run**(R) -**agt**(R,AGT).

A '\$' attached to an atomic constraint represents a cost (alias a large completion energy) so that the atomic constraint is a goal. The two goals in (A) together amount to a macroscopic goal to assert that kim runs. So the expected sentence to be generated is *Kim runs*.

Figure 5 graphically represents the above clauses. All the transitive cycles going through

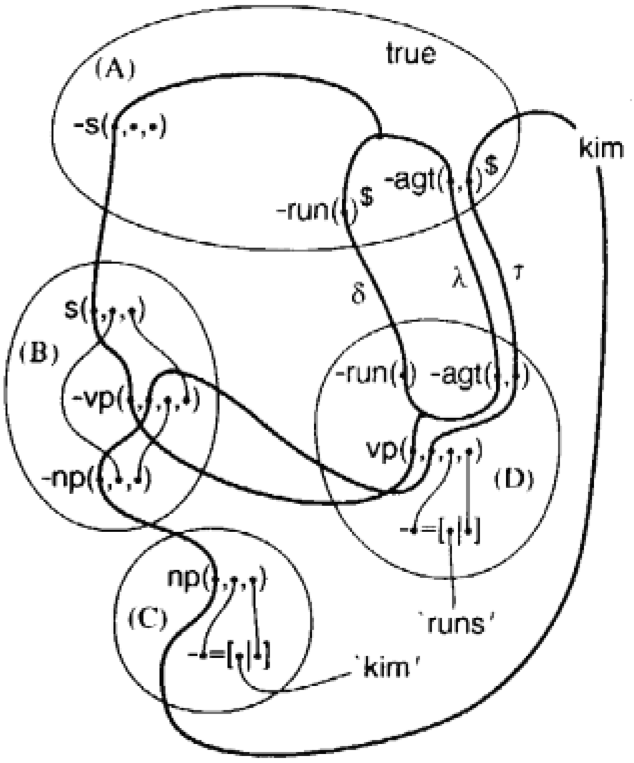


Figure 5: Sentence Generation

the goals are depicted by thick curves. Out of the extraclausal equalities, only those involved in transitive cycles are shown.

Note that the four clauses are nearly all that pertain to transitive cycles going through the goals. So due to the joint effect of marker-passing-like inferences around these cycles, almost only these clauses are very likely to be involved in the inferences about to take place. Due to the shortest transitive cycle and the cost, the most probable inference to fire here is probably the subsumption from run(RUN)§ in (A) to run(R) in (D). This subsumption will strengthen the effect of the transitivity energy of the transitive cycles including the extraclausal equality between the two atomic formulas, which raises the preferences of further subsumptions along these cycles. We skip further details of this generation process, but the expected subsumptions are fired perhaps in parallel approximately in the order dependent on a more global computational context.

Sentence generation in our framework integrates the two stages, planning (what to say) and surface generation (how to say), unlike the traditional models of sentence generation which separate the two stages and either serialize them or interleave them with each other. For instance, if our system has generated *Taro got a book* and *He paid thousand yen*, it is not probably motivated any more to say that he bought the book. Thus, the system is not generally fully decided on what to say when it starts saying anything. Also, the decision on what to say and that on how to say are not distinguished at all in the design of the system.

To account for efficient parsing, we need to formulate how to handle bindings. Hasida [3] shows that chart parsing emerges from general heuristics to control subsumptions to deal with bindings. These heuristics could be formulated in terms of the dynamics discussed above.

5 Concluding Remarks

We have discussed a framework of constraint for designing a cognitive system. To capture the partiality and the corresponding situatedness of cognition, the constraint is situated in a field of force derived from potential energy representing the degree of violation. This field of force gives rise to analog inference as spreading activation, and also controls symbolic computation to transform the constraint. Not only nearly logical inferences and abductive inferences but also associative inferences emerge out of such a dynamics.

A distinguished feature of our framework is that the control scheme for inference is derived from a dynamics which also provides the declarative semantics. In comparison, the other frameworks such as marker passing stipulate the inference control apart from the declarative semantics. The inborn integration of declarative semantics and inference control as in our method will not only provide a clear perspective of the design, but also guarantee emergent reflection of semantic relevance in information processing. In this connection, our method is integrated also in another sense that it controls analog and symbolic inferences based on the same dynamics. This is a strong advantage over the methods such as Waltz and Pollack's [10] which separate the two inference schemes.

The current framework should be extended with respect to several points. First, some partial processing method is necessary for dealing with transitive cycles, although at any rate a massively parallel computational system is essential to implement our theory. Second, *deletion* should be incorporated in addition to subsumption, in order to prevent the constraint network from unlimited growth. Probably deletion is regarded as a reverse of subsumption, and hence the control of deletion may be formulated along the same line as that of subsumption. Third, the control method should take into account consistency checking as well. Consistency checking pertaining to binding is discussed in [3]. In order to handle consistency maintenance in general, we will have to give preferences not only to subsumptions which seem to decrease P but also to those which seem to increase P. So perhaps $|\frac{\partial P}{\partial s_{\alpha\beta}}|$ is a better preference score of subsumption than $-\frac{\partial P}{\partial s_{\alpha\beta}}$. Finally, learning is vitally necessary for both the parameters of the energy functions and the symbolic structure of the constraint. Just as in the control of symbolic inference, the parameter learning could be guided by $-\frac{\partial U}{\partial s}$ for each parameter s of U to learn. This amounts to a generalization of Suttner and Ertel's [9] method.

Let us finish up by a few words about an implication for linguistic studies. Linguistic competence is regarded as corresponding to the declarative semantics, and linguistic performance corresponding to the control of inferences. So our framework unifies linguistic competence and performance, in the sense that both follows from the same dynamics. If this were too much to say, at least the competence/performance distinction is an artifact depending on what kind of logic we employ to describe natural language. The clear distinction of competence and performance has been motivated by the traditional symbolic logic lacking inference control except for closure operation (exhaustive inference).

Appendix

In the general form, the disjunction energy and the exclusion energy of clause Φ are (13) and (14), respectively.

(13) $$D_\Phi \prod_{\xi} (1 - r_\xi x_\xi)$$ (14) $$E_\Phi \sum_{\xi \neq \eta} r_\xi x_\xi r_\eta x_\eta$$

ξ and η range over the literals in Φ. r_ξ is a constant such that $0 < r_\xi \leq 1$. In the digital approximation, (13) means that at least one literal should be true, whereas (14) means that at most one literal may be true. For (8), (9) and (10), $r_{-p} = r_{+q} = 1$.

Suppose $\alpha \circ \beta$ for two atomic formulas α and β. Then the assimilation energy of the pair of α and β is defined as follows.

(15) $$-A_\pi (s_{\alpha\beta} + s_{\beta\alpha})(x_\alpha - \frac{1}{2})(x_\beta - \frac{1}{2}) \prod_{\delta} x_\delta$$

A_π is a positive constant associated with predicate π shared by α and β. δ ranges over the extraclausal equalities connecting the corresponding arguments of α and β.

The transitivity energy U_Δ of transitive cycle $\Delta = \delta_0 \delta_1 \cdots \delta_{k-1}$ is defined as below.

(16) $$U_\Delta = \begin{cases} -t \prod_i (e_i - \theta) & (e_i < \theta \text{ for at most one } i) \\ 0 & (\text{otherwise}) \end{cases}$$

e_i is the activation value of δ_i, and θ is a constant such that $0 < \theta < 1$. t is a positive constant.Note that the transitivity energy is large when just one equality in Δ has a small activation value.

The completion energy of a literal ξ is defined as follows.

(17) $$C_\xi x_\xi \prod_{\xi \circ \eta} (1 - s_{\xi\eta} x_\eta)$$

C_ξ is a positive constant. The subsumption coefficients concerning literals are defined so that $s_{+\alpha+\beta} = s_{-\alpha-\beta} = s_{\alpha\beta}$ for every pair of atomic formulas α and β. In the digital approximation, the positive (negative) completion energy means that some η unifiable with ξ should be true in order for ξ to be true.[13]

The *normalization energy* of atomic constraint α is defined to be the following.

(18) $$T\{x_\alpha \log x_\alpha + (1 - x_\alpha) \log (1 - x_\alpha)\}$$

[13]In the ordinary completion as in Prolog, the atomic formula part of ξ must appear as a negative literal in a clause and that of η as a positive literal in a clause.

T is a positive constant called the *temperature*. $x_\alpha = \frac{1}{1+\exp(-F_\alpha')/T}$ at equilibria of force follows from (18), where F_α' stands for the total force to α due to the causes other than normalization energy. So this energy normalizes the activation value so that $0 < x_\alpha < 1$.

References

[1] E. Charniak. A neat theory of marker passing. In *Proceedings of AAAI '86*, pages 584–588, 1986.

[2] L. Danlos. Conceptual and linguistic decision in generation. In *Proceedings of COLING '84*, pages 501–504, 1984.

[3] K. Hasida. Common heuristics for parsing, generation, and whatever ... In *Proceedings of the Workshop on Reversible Grammar in Natural Language Processing*, pages 81–90, Berkeley, 1991. held in connection with 29th Annual Meeting of the Association for Computational Linguistics.

[4] J. Hobbs, M. Stickel, D. Appelt, and P. Martin. Interpretation as abduction. Technical Note 499, SRI International, 1990.

[5] P. Norvig. Marker passing as a weak method for text inferencing. *Cognitive Science*, 13:569–620, 1989.

[6] F.J. Pineda. Generalization of backpropagation to recurrent and higher order neural net works. In Anderson D.Z. editor, *Neural Information Processing Systems*, pages 602–611, 1988.

[7] S.M. Shieber, G. van Noord, and R.C. Moore. A semantic-head-driven generation algorithm for unification-based formalisms. In *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics*, pages 7–17, 1989.

[8] E. Stickel, M.' Rationale and methods for abductive reasoning in natural-language interpretation. In R. Studer, editor, *Proceedings, Natural Language and Logic, International Scientific Symposium*, number 459 in Lecture Notes in Artificial Intelligence, pages 233–252. Springer Verlag, 1989.

[9] B. Suttner, C. and W. Ertel. Automatic acquisition of search guiding heuristics. In *Proceedings of the 10th International Conference on Automated Deduction (CADE)*, pages 470–484, 1990.

[10] D. Waltz and J. Pollack. Massively parallel parsing: A strongly interactive model of natural language interpretation. *Cognitive Science*, 9:51–74, 1985.