Lecture Notes in Artificial Intelligence 1515
Subseries of Lecture Notes in Computer Science

Flávio Moreira de Oliveira (Ed.)

Advances in Artificial Intelligence

14th Brazilian Symposium on Artificial Intelligence, SBIA'98
Porto Alegre, Brazil, November 1998
Proceedings

Springer

Lecture Notes in Artificial Intelligence 1515

Subseries of Lecture Notes in Computer Science
Edited by J. G. Carbonell and J. Siekmann

Lecture Notes in Computer Science
Edited by G. Goos, J. Hartmanis and J. van Leeuwen

Springer
Berlin
Heidelberg
New York
Barcelona
Hong Kong
London
Milan
Paris
Singapore
Tokyo

Flávio Moreira de Oliveira (Ed.)

Advances in Artificial Intelligence

14th Brazilian Symposium
on Artificial Intelligence, SBIA'98
Porto Alegre, Brazil, November 4-6, 1998
Proceedings

Springer

Series Editors

Jaime G. Carbonell, Carnegie Mellon University, Pittsburgh, PA, USA
Jörg Siekmann, University of Saarland, Saarbrücken, Germany

Volume Editor

Flávio Moreira de Oliveira
Instituto de Informática - PUCRS
Av. Ipiranga 6681, prédio 30, bloc 4
90619-900 Porto Alegre - RS, Brazil
E-mail: flavio@kriti.inf.pucrs.br

Cataloging-in-Publication Data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Advances in artificial intelligence : proceedings / 14th Brazilian Symposium on Artificial Intelligence, SBIA'98, Porto Alegre, Brazil, November 4 - 6, 1998. Flávio Moreira de Oliveira (ed.). - Berlin ; Heidelberg ; New York ; Barcelona ; Budapest ; Hong Kong ; London ; Milan ; Paris ; Singapore ; Tokyo : Springer, 1998
(Lecture notes in computer science ; Vol. 1515 : Lecture notes in artificial intelligence)
ISBN 3-540-65190-X

CR Subject Classification (1998): I.2

ISSN 0302-9743
ISBN 3-540-65190-X Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author
SPIN: 10692710 06/3142 – 5 4 3 2 1 0 Printed on acid-free paper

Preface

The Brazilian Symposium on Artificial Intelligence (SBIA) has been organized by the Interest Group on Artificial Intelligence of the Brazilian Computer Society (SBC) since 1984. In order to promote research in Artificial Intelligence and scientific interaction among Brazilian AI researchers and practitioners, and with their counterparts worldwide, it is being organized as an international forum since 1993. The SBIA proceedings have been published by Springer-Verlag as a part of the Lecture Notes in Artificial Intelligence (LNAI) series since 1995.

The XIVth SBIA, held in 1998 at the PUCRS Campus in Porto Alegre, has maintained the international tradition and standards previously established: 61 papers were submitted and reviewed by an international program committee, from this number, 26 papers were accepted and are included in this volume.

Of course, organizing an event such as SBIA demands a lot of group effort. We would like to thank and congratulate all the program committee members, and the many reviewers, for their work in reviewing and commenting on the submitted papers. We would also like to thank the Pontifical Catholic University of Rio Grande do Sul, host of the XIV SBIA, and the institutions which sponsored it - CNPq, CAPES, BANRISUL, among others. Last but not least, we want to thank all the kind people of the Local Organizing Committee, whose work made the event possible.

Porto Alegre, November 1998 *Flávio Moreira de Oliveira*

Program Committee Members

*Flávio M. de Oliveira (Instituto de Informática - PUCRS, Brazil) ***
Antonio Carlos da Rocha Costa (Univ. Católica de Pelotas, Brazil)
Luis Otávio Alvares (Instituto de Informática - UFRGS, Brazil)
Dibio Borges (Univ. Federal de Goiania, Brazil)
Celso Kaestner (CEFET - PR, Brazil)
Tarcisio Pequeno (LIA - Univ. Federal do Ceará, Brazil)
Edson de Carvalho Filho (DI - UFPE, Brazil)
Ariadne Carvalho (Unicamp - Brazil)
Maria Carolina Monard (USP - São Carlos, Brazil)
Guilherme Bittencourt (LCMI - Univ. Federal de S. Catarina, Brazil)
Sandra Sandri (Inst. Nacional de Pesquisas Espaciais, Brazil)
Wagner Silva (UnB, Brazil)
Edilson Ferneda (Univ. Federal da Paraíba, Brazil)
Helder Coelho (Univ. de Lisboa, Portugal)
Yves Demazeau (Lab. Leibniz/IMAG - Grenoble, France)
Barbara Hayes-Roth (Stanford Univ. - USA)
Stefan Wrobel (GMD, Germany)
Manuela Veloso (Carnegie Mellon - EUA)
John Self (Leeds Univ. - UK)
Peter Ross (Edinburgh Univ. - UK)

Local Organizing Committee

Ana L.V. Leal, Gabriela Conceição, Iára Claudio, João L.T. da Silva, Lúcia M.M. Giraffa, Michael C. Mora, all the people at IIPUCRS, Ricardo Annes, Ricardo M. Bastos, Thais C. Webber, The PET Group.

List of Reviewers

Alexandre Ribeiro
Ana Teresa de Castro Martins
Antonio Carlos da Rocha Costa
Ariadne Carvalho
Arnaldo Moura
Barbara Hayes-Roth
Bertilo F. Becker
Celso Kaestner
Dibio Borges
Doris Ferraz de Aragon
Edilson Ferneda
Edson de Carvalho Filho
Eloi F. Fritsch
Flávio M. de Oliveira
Flavio Soares Correa da Silva
Germano C. Vasconcelos
Guilherme Bittencourt
Hans-Jorg Schneebeli
Helder Coelho
Jacques Wainer
João Balsa
John Self
Lúcia M.M. Giraffa
Luis Antunes
Luis Moniz
Luis Moniz
Luis Otávio Alvares
Manuela Veloso
Mara Abel
Marcelino Pequeno
Maria Carolina Monard
Maria das Gracas Volpe Nunes
Mathias Kirsten
Michael C. Mora
Patrick Doyle
Paulo J. L. Adeodato
Peter Ross
Raul S. Wazlawick
Riverson Rios
Rosa M. viccari
Sandra Sandri
Solange Oliveira Rezende
Stefan Wrobel
Tarcisio Pequeno
Teresa B. Ludermir
Thomas F. Gordon
Vera L.S. de Lima
Wagner Silva
Wilson R. de Oliveira Jr.
Yves Demazeau

Contents

Machine Learning and Neural Networks

Logic Programming

Knowledge Representation

AI Applications

On Personal and Role Mental Attitudes: A Preliminary Dependence-Based Analysis

Jaime Simão Sichman[1,⋆] and Rosaria Conte[2]

[1] University of São Paulo
Computer Engineering Department
Intelligent Techniques Laboratory
Av. Prof. Luciano Gualberto, 158 trav. 3
05508-900 São Paulo SP Brazil
jaime@pcs.usp.br
[2] Italian National Research Council
Institute of Psychology
Division of AI, Cognitive Modelling and Interaction
Viale Marx, 15
00137 Rome Italy
rosaria@pscs2.irmkant.rm.cnr.it

Abstract. In this paper, we present some preliminary results concerning the extension of dependence theory [2] and social reasoning [9] to cope with the notion of *organizational roles*. In order to accomplish this task, we first present a rather informal definition of organization and roles, showing how this additional dimension can be represented within a 3-layered architecture of mental states. We then restrict our analysis to the domain level, by extending our original notions of unilateral and bilateral dependence, as well as that of goal situation, to show that just by representing explicitly the input source of some mental attitudes one can easily explain some interesting social phenomena, like agents' adequacy to roles and vice-versa. For methodological reasons, we divide this analysis along two main axes, i.e., the inter-agent and the intra-agent dimensions.

1 Organizations, Roles and Autonomy

The task of proposing a complete, formal and rather universal representation of *organizations* and their associated roles is a very difficult one. Indeed, several different dimensions are used by researchers in social sciences [3, 5], distributed AI and multi-agent systems [4, 7], and distributed computing [1, 6] to characterize what an organization is and what its main components are. In a certain sense, quite all of these proposed descriptions comprise both a *factual* and a *procedural* dimension.

By factual we mean a mere observable behavior, despite its internal functioning: an organization has its high level goals, its observable inputs and outputs.

⋆ Partially financed by CNPq, Brazil, grant number 301041/95-4.

F. Moreira de Oliveira (Ed.): SBIA'98, LNAI 1515, pp. 1-10, 1998.

On the other hand, the procedural dimension has more to do with how this behavior is obtained: division of labor into roles, establishment of authority and communication links between roles etc.

We do not intend in this paper to propose an alternative formal definition for either of these dimensions. Rather than that, we intend to propose a rather *informal* description for what an organization means for us, sufficient enough to enable us to depict further some interesting phenomena associated with it.

In order to do so, let us consider Wooldrige's and Jennings model of cooperative problem solving [11]. This rather generic theoretical model expresses the main phases involved in a multi- agent approach for problem solving. According to this model, a cooperative problem solving is composed by four different phases: *(i)* recognition of a cooperation potential; *(ii)* coalition formation; *(iii)* plan formation; *(iv)* coalition action.

The first phase is carried on when an autonomous agent infers that he cannot achieve a goal alone (complementarity-based interaction, like in [2, 9]) or when he prefers to cooperate with the others (utility-based interaction, like in [8]). The next step is to propose to one or more possible partners a coalition to achieve the goal. If these partners accept to take part into the coalition, then a phase of plan negotiation starts. Finally, the coalition partners may start to act, which may lead to a successful goal accomplishment.

In this work, we consider an organization as being intuitively the *result* of the 3 first phases mentioned above. In the case of pre-established organizations, these phases are carried on off-line, i.e., agents just engage them, not being responsible for their definition. On the other hand, this is not the case of dynamical organizations (like the so-called communities of practice [5]). Anyway, regarding our problem, the fact of being the organizations either dynamically or statically defined does not have any major influence.

Like stated above, we consider an organization as an entity with both a factual and a procedural dimension. The factual dimension can be represented by a set of goals to be fulfilled. The procedural dimensional can be captured by a set of plans to achieve these goals, a set of actions that should be performed in order to carry on the plans, and a set of resources which are used when these plans are carried on.

For performance reasons, certainly due to Simon's principle of bounded rationality [10], these goals, plans, actions and resources are splitted into several *roles*. The notion of role, therefore, comprises several notions like cooperation (roles cooperate with one another in order to achieve organizational goals) and coordination (there is pre-established control and communication flow between the roles).

In practice, however, roles are played by autonomous agents. Interestingly, we have proposed in our social reasoning model [9] a similar description of autonomous agents, where they have a subjective representation of each other's goals, plans, actions and resources. A question then arises: using this description framework, how can we merge the notion of organizational role with the model of an *autonomous* agent?

2 Personal and Role Mental Attitudes

We propose to merge these notions by representing explicitly the *input source* of these attitudes. In order to do so, let us first consider the following three-layered architecture, represented in Fig. 1.

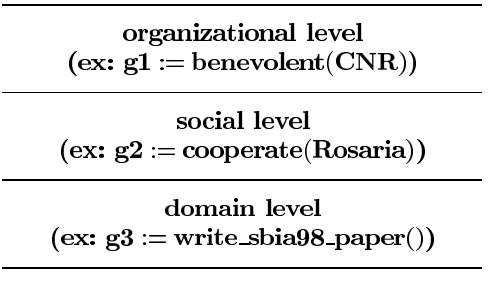


Fig. 1. Architecture layers.

In our previous work [2, 9], autonomous agents decided to cooperate with one another based on their complementarity, i.e., their dependence and power relations. Regarding their mental attitudes, we could consider that a goal to cooperate would be generated at a social level, and this upper level goal would drive the agents to effectively cooperate at a domain level, for instance by performing needed actions or by releasing needed resources. We could then consider that a *personal* domain level goal is a goal whose origin is related to a social level goal derived by the agents' own internal mechanisms, like social reasoning [9], or BDI architectures [7].

By considering the organizational dimension, we can simply add another upper layer, where there is a pre-established goal, i.e., the one of being benevolent within the organization. It must be stated that in this work, we do not analyze the problem of engaging in/trying to modify the organizational structure, i.e., their roles and plans. We just consider that this choice has already been made, and cannot be changed. In fact, we could also propose a dependence based explanation for this process, i.e., a kind of reciprocal dependence between the player of the role (who could offer his *savoir-faire* to the organization) and the organization itself, considered as a single agent (who could offer to the player something in exchange, like prestige or money).

So, we propose to use the same mental attitudes (goals, plans, actions and resources) to characterize the agent endogenous mental attitudes (which we call personal mental attitudes) and the ones which he is endowed with by playing the role (which we call role mental attitudes).

A fundamental difference is expressed by a *long-term, non revisable* commitment to the role mental attitudes. Obviously, an agent may decide to stop to play the role or even to quit the organization, but these situations are not treated here. This long term commitment in a certain sense *reduces* the agent's auton-

omy: he must play the role assigned to him in the organization. Interestingly, this reduction of autonomy is the result of the agent's deliberation, although at a higher level: it is a result of his decision to take part into the organization.

In practice, considering the domain level, this reduction of autonomy means that:

1. Whenever another agent proposes a cooperation or exchange regarding a role goal, the agent who receives this proposal *will always accept to participate*. In other words, the problem of reciprocation regarding reciprocal dependence [9] disappears. This is represented by the benevolence assumption in the upper level of Fig. 1;
2. Whenever there is a role plan to achieve a goal, it *must be used*. It is not a question of the agent's option anymore, even if he may have an alternative better plan in his personal repertoire. When there is no role plan, the agent who plays the role may use whatever plan he likes. This latter difference was already noticed and represented in an alternative way by Yu's model of hard of soft dependence within a context of business process re-engineering [12].

Regarding actions and resources, there is an interesting phenomena: by using some intrinsic resources, the *semantic meaning* of an action may change. Basically, role resources may be differentiated in 2 levels: *extendible* and *intrinsic* ones. Extendible resources may be used by an agent when trying to achieve personal or role goals. For instance, it may be the case that for the role of a secretary, an organization may supply her with a computer, that can be used to type either a letter to his boss or a new recipe of Italian food for her husband. Intrinsic resources, however, are used only to change the semantic meaning of an action which belongs to the agent who plays the role. As an example, we can consider what we call the *Achtung Problem*: when uttered by a policeman, this phrase means something different than when said by anyone else. Moreover, the meaning of the utterance also changes whether the policeman is playing his role (policeman) in the organization (society) or not (for instance, when dressed in civilian).

These latter phenomena are very interesting per se and merit a more detailed analysis, which we do not intend to do in this work. What we want to show is that the very fact of just adding two modalities, P and R, denoting respectively personal and role mental attitudes, allows us to explain several interesting phenomena, both in an inter-agent and intra-agent context, as described next.

3 The Inter-Agent Dimension

In this section, we try to explain some phenomena related to the fact that the dependence and power relations between agents may have different input sources for their goals, namely personal and role inputed ones.

Let us remind the core notions of dependence theory [2] and social reasoning [9]. Supposing that i, j and k are variables that denote agents and g and g' are variables that denote goals, we have defined the following three basic dependence relations:

- *Bilateral relations*: when two agents depend on one another to accomplish their goals. This case can be splitted in two-sub-cases:
  - $MD(i, j, g, k)$: agents i and j have a *mutual dependence* if they are dependent on one another for the same goal g;
  - $RD(i, j, g, g', k)$: agents i and j have a *reciprocal dependence* if they are dependent on one another for different goals g and g';
- *Unilateral relations*: when one agents depends on another but the conversive situation is not true:
  - $UD(i, j, g)$: agent i depends on agent j for his goal g, but the latter does not depend on him for any of his goals.

In the case of bilateral relations, the letter k designates the agent k whose plans are used in order to infer the dependence relations. Even if an agent is always supposed to use his own plans[1] when reasoning about the others, we have shown in [9] that in some cases, specifically those involving bilateral relations, it may be interesting for an agent to use his beliefs about his possible partners' plans in order to evaluate if these latter are also aware of the dependence relations[2]. Regarding this work, this subtlety is not important, and therefore in the rest of the paper we will use a shorter notation, i.e., $MD(i, j, g)$ and $RD(i, j, g, g')$.

In the next subsections, we will analyze how these relations can be extended by adding the input source to the agents' mental attitudes. In a first attempt, we will limit ourselves to analyze the agents' goals, except in the case of unilateral relations, as shown just below. In order to do this, we introduce the predicates $P\text{–}is_g(i, g)$ and $R\text{–}is_g(i, g)$ to denote that agent i has the goal g whose input source is respectively personal and role. However, for the unilateral dependence analysis, we will use also the predicates $P\text{–}is_a(i, a)$ and $R\text{–}is_a(i, a)$ to denote actions whose input source is respectively personal and role.

3.1 Mutual Dependence

Considering that a mutual dependence requires a common goal, if we have two different input sources for our agents' common goal, at least three different situations may arise:

$$NEG\_COOP(i, j, g) \equiv MD(i, j, g) \wedge P\text{–}is_g(i, g) \wedge P\text{–}is_g(j, g) \quad (1)$$

$$ASS\_COOP(i, j, g) \equiv MD(i, j, g) \wedge P\text{–}is_g(i, g) \wedge R\text{–}is_g(j, g) \quad (2)$$

$$ORC\_COOP(i, j, g) \equiv MD(i, j, g) \wedge R\text{–}is_g(i, g) \wedge R\text{–}is_g(j, g) \quad (3)$$

Negotiated cooperation (1) corresponds to the situation analyzed in [2, 9], i.e., two autonomous agents decide to cooperate because they are interdependent. However, this decision is inferred by the agents themselves, being a result of a deliberative procedure. Some negotiation about the common plan to be used (phase *iii* presented in section 1) may arise, but at the end the agents are effectively going to cooperate.

[1] This means that if the reasoning agent is i, k is instantiated to i.

[2] In this case, if the reasoning agent is i who depends on j, k is instantiated both to i and j.

Assisted cooperation (2) is quite an interesting situation, as it reflects a special kind of assisted relation. As the input source of agent's j goal g is role, he does not deliberate about the cooperation. Agent i may take advantage from j's role. Consider for example, the *mother/teacher relationship*, where the former finds in the teacher a sort of accomplice with regard to her child.

Orchestrated cooperation (3) is similar to negotiated cooperation (1), but there is no real deliberative procedure of the agents. The cooperation is *pre-established*, in the moment of the design of the roles and common plans in the organization. Regarding the common plan to be used, there is no negotiation, as it is usually predefined too. A nice example of this situation is the *doctor/nurse relationship*.

3.2 Reciprocal Dependence

As in the previous case, considering our two different input sources, we may have also at least three different situations:

$$NEG\_EXCH(i,j,g,g') \equiv RD(i,j,g,g') \wedge P\text{–}is_g(i,g) \wedge P\text{–}is_g(j,g') \quad (4)$$

$$PRO\_EXCH(i,j,g,g') \equiv RD(i,j,g,g') \wedge P\text{–}is_g(i,g) \wedge R\text{–}is_g(j,g') \quad (5)$$

$$ORC\_EXCH(i,j,g,g') \equiv RD(i,j,g,g') \wedge R\text{–}is_g(i,g) \wedge R\text{–}is_g(j,g') \quad (6)$$

Negotiated exchange (4) corresponds to the situation analyzed in [2, 9], i.e., two autonomous agents decide to do a social exchange because they are interdependent for their different goals g and g'. This social goal is autonomously generated. Like the mutual dependence case (1), some negotiation about the common plans to be used (one for each goal) may arise. Moreover, the problem of reciprocation is taken into account: the agents must try to ensure that after doing their part of the deal, the other agent will do his part.

Protected exchange (5) is similar to assisted cooperation (2). As agent j has g' as a goal whose input source was role, he just accepts the exchange with agent i. Regarding agent i, the problem of reciprocation does not arise. Interestingly, from the point of view of agent i, this situation is as good as a mutual dependence, because agent j is due by role reasons to effectively help him to achieve g. A good example of this case is the *policeman/collaborator relationship*, where the former helps the latter in order to obtain information relevant to capture other burglars, while the latter provides information in order to be released.

Orchestrated exchange (6) is similar to orchestrated cooperation (3). Here we have a pre-established social exchange, with no real deliberative procedure by the agents. Regarding the common plans to be used, there is no negotiation either, as they are usually predefined too. An important example of this case is the *incentive provision*, where one partner provides an external reinforcement for the other to perform a given activity.

3.3 Unilateral Dependence

Considering the case of unilateral dependence, we will take into account for our analysis both the goal the dependent agent needs to be accomplished and

the action that he wants the other agent to perform in order to achieve this goal. Therefore, once more we have two mental attitudes, each of them with two different input sources, which may lead to at least three different situations, described below.

$$END\_BENV(i,j,g) \equiv UD(i,j,g) \wedge P\text{–}is_g(i,g) \wedge P\text{–}is_a(j,a) \quad (7)$$

$$IMP\_BNVL(i,j,g) \equiv UD(i,j,g) \wedge P\text{–}is_g(i,g) \wedge R\text{–}is_a(j,a) \quad (8)$$

$$ORC\_BNVL(i,j,g) \equiv UD(i,j,g) \wedge R\text{–}is_g(i,g) \wedge R\text{–}is_a(j,a) \quad (9)$$

Endogenous benevolence (7) corresponds to the situation analyzed in [2, 9], i.e., an autonomous agent depends on the benevolence of the other to obtain what he needs to achieve his own goal. Benevolence can arise by several means, as stated in [9], for instance out of personal reasons (as a mother regarding her child) or for organizational reasons, as shown in (9) below.

Exogenous or imposed benevolence (8) is the case where, once more, as in (2) and (5), we have a kind of "master-slave" relation. As agent j has a as an action whose input source is role, he just accepts to perform it, in whatever plan, despite of the goal being achieved. Regarding agent i, here we have the most classical "master-slave" architecture, when the execution of a service is demanded without explaining reasons associated with it. As examples, we can cite the so well known *client-server architectures* in distributed systems, or the *defendant/defender* relationship from the natural domain of professionals. From the point of view of agent i, this situation is the best possible one: he does not need to offer anything in return and he is assured that agent j will reciprocate by role imposition.

Orchestrated benevolence (9) is similar to the previous cases of bilateral relations shown in (3) and (6), here we have a pre-established benevolence, with no real deliberative procedure by the agents. Agent j is simply *constrained to act*, because his role imposes him this situation. In a certain sense, this situation corresponds to the imposition of a certain internal structure (procedural dimension) of the organization. This situation can be illustrated by the typical *secretary/boss relationship*.

4 The Intra-Agent Dimension

In this section, we want to analyze some interesting phenomena regarding the *adequacy* of agents when playing some organizational roles. This analysis can now be carried on, since we are explicitly representing the input sources of the agents' mental attitudes.

In order to that, let us remind the notion of *goal situation* [9], which relates an agent to a certain goal. Supposing that i denotes an agent and g a certain goal, four different goal situations may hold:

1. $NG(i,g)$: the agent i doesn't have the goal g;
2. $NP(i,g)$: the agent i has the goal g but has not any plan to achieve it;
3. $AUT(i,g)$: the agent i has the goal g and at least one autonomous plan which achieves this goal; an autonomous plan is a plan where an agent may perform all its needed actions;

4. $DEP(i,g)$: the agent i has the goal g and all plans he has to achieve this goal are dependent plans; a dependent plan is a plan where an agent needs the help of the others to perform at least one of its actions.

This goal situation is represented subjectively within the agents' minds, and helps them to deliberate whether they should propose or accept to take part in coalitions, as shown in [9]. Let us now consider which interesting phenomena can be described just by adding to this notion our two modalities, P and R. Some very interesting situations regarding the adequacy of agents to roles may be described, which we have initially grouped into six different cases, as described next.

$$ROLE\_ADPT(i,g) \equiv P\text{–}NG(i,g) \wedge (R\text{–}NP(i,g) \vee R\text{–}AUT(i,g) \vee R\text{–}DEP(i,g)) \quad (10)$$

$$PERS\_ADPT(i,g) \equiv R\text{–}NG(i,g) \wedge (P\text{–}NP(i,g) \vee P\text{–}AUT(i,g) \vee P\text{–}DEP(i,g)) \quad (11)$$

$$ROLE\_ENRC(i,g) \equiv R\text{–}NP(i,g) \wedge (P\text{–}AUT(i,g) \vee P\text{–}DEP(i,g)) \quad (12)$$

$$PERS\_ENRC(i,g) \equiv P\text{–}NP(i,g) \wedge (R\text{–}AUT(i,g) \vee R\text{–}DEP(i,g)) \quad (13)$$

$$INCR\_POWR(i,g) \equiv R\text{–}AUT(i,g) \wedge P\text{–}DEP(i,g) \quad (14)$$

$$DIMN\_POWR(i,g) \equiv P\text{–}AUT(i,g) \wedge R\text{–}DEP(i,g) \quad (15)$$

Role goal adoption (10) corresponds to the case where an agent adopts a new goal by role playing, as this goal was not previously a personal one. This situation is quite common when considering the domain level, shown in Fig. 1. In fact, the pre-established cooperation is represented in the two upper levels of this figure, as explained earlier. Nevertheless, one situation is worth meaning, i.e., the one when *negative* goals, whose input source is role, are considered. In such a scenario, the fact of playing the role *restricts* the possible goals an agent is supposed to have. A classical example is what we call the *Priest Dilemma*: a priest can not have the goal of marrying a woman he is in love with. Stated differently, while marrying may be a personal goal, not marrying is a goal imposed by his role. Interestingly, it is exactly this situation which lead to eventual conflicts which can not be solved unless the agent decides not to play the role anymore, or even eventually to leave the organization[3].

In **personal goal adoption** (11), the agent has personal goals that are not supposed to be achieved by his role. Once again, this situation is also quite common when considering the domain level. However, *negative* personal goals may also be in conflict with the role the agent is supposed to play. One example could be that of the *Pacifist Dilemma*: a pacifist who refuses, for conscientious objection, to work in a research project financed by the Army. Indeed, it is quite unreasonable to think that a situation where all agents' role goals could also be personal goals and vice versa could occur. This optimal case, both from an organizational and personal point of view, wold mean a perfect adequacy between the agent who effectively plays the role and the motivational requirements that the organization expects from the role player.

[3] We do not consider in this work, as stated before, the problem of trying to modify the organizational structure.

Role enrichment by personal plans (12) arises when a role goal does not prescribe any plan, and the agent who is playing the role has a plan, either autonomous or dependent, in order to achieve this goal. This situation corresponds to a good adequacy, from the organizational viewpoint, of the role player: he is someone who can effectively decide what to do in the absence of predefined plans. The organization therefore profits from the fact that the role player has alternative ways to fulfill his goals. This situation illustrates exactly what some expressions, like the following job announcement, mean: "Candidates who are interested, motivated and *able to solve problems with their own initiative* will be preferred".

Personal enrichment by role playing (13) is dual to the previous case. It arises when a role goal prescribes a plan to achieve a goal, and the agent who is playing the role had not any previous idea of how this goal could be fulfilled. This situation corresponds to the *Apprentice Advantage*: an agent profits from playing a role to learn how to achieve a certain goal, and this plan may be used in the future even in a scenario of non-role playing. Now it is the agent who profits from the fact of playing a role within the organization.

In **increasing power by role playing** (14), the role prescribes to an agent an autonomous plan, whereas this latter only has dependent plans to achieve the goal. This situation may be viewed as a special case of the previous one (13), where the *quality* of a possible plan to achieve the goal is enhanced . This statement means that we are assuming that autonomous plans are always better than dependent ones, due to coordination and communication overhead costs [9].

In **diminishing power by role playing** (15), the organization limits the role player, in the sense that there is a predefined *orchestrated* cooperation, whereas the agent could possibly achieve the goal by himself. In a certain sense, the agent is sub-utilized when playing the role. We can call this the *Team Paradox*. On the other hand, a more strategic view of the whole organization may explain why not to let a certain important goal in the hands of a sole agent. For instance, an organization may wish not to be vulnerable to a possible decision of the agent of leaving the organization.

5 Conclusions

In this paper, we have presented some preliminary results concerning the extension of dependence theory [2] and social reasoning [9] to cope with the notion of *organizational roles*. We have extended our original notions of unilateral and bilateral dependence, as well as that of goal situation, with two modalities, P and R, to denote respectively personal and role mental attitudes. We have analyzed some phenomena in both inter-agent and intra-agent dimensions.

In a certain sense, we believe that the role playing dimension *quite extinguishes* the autonomy of the agents. However, this autonomy is played earlier, in the moment when an agent has chosen whether it was interesting or not to join an organization or to play certain roles within an organization he was already engaged to. As a consequence of this fact, the fundamental differences between

mutual, reciprocal and unilateral dependences, as discussed in [2,9] *loose* their importance: agents simply cooperate, exchange or perform actions (UD case) because they are supposed to do so by role imposition.

Finally, we can consider that the differences between bilateral and unilateral relations, when expressed in a scenario of a role input source, can consist of a first step towards an idea of *role structure*. For instance, the case of mutual dependence could be desirable for the same/similar hierarchical role degrees (for instance, two researchers in a research laboratory) while a unilateral dependence could be used to express a "master-slave" scenario (for instance, one researcher and one photocopist).

References

1. Eleri Cardozo. *DPSK: A Kernel for Distributed Problem Solving*. Phd Thesis, CAED, Carnegie Mellon University, Pittsburgh, PE, January 1987.
2. Cristiano Castelfranchi, Maria Micelli, and Amedeo Cesta. Dependence relations among autonomous agents. In Eric Werner and Yves Demazeau, editors, *Decentralized A. I. 3*, pages 215–227. Elsevier Science Publishers B. V., Amsterdam, NL, 1992.
3. Rosaria Conte and Cristiano Castelfranchi. Norms as mental objects: From normative beliefs to normative goals. In Khaled Ghedira and François Sprumont, editors, *Pre-proceedings of the 5th European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, Neuchâtel, Switzerland, August 1993.
4. Les Gasser. Boundaries, identity and aggregation: Plurality issues in multiagent systems. In Eric Werner and Yves Demazeau, editors, *Decentralized A. I. 3*, pages 199–212. Elsevier Science Publishers B. V., Amsterdam, NL, 1992.
5. Bernardo A. Huberman and T. Hogg. Communities of practice: Performance and evolution. Technical report, Xerox Palo Alto Research Center, 1994.
6. Naftaly H. Minsky and David Rozenshtein. Controllable delegation: An exercise in law-governed systems. In Norman Meyrowitz, editor, *Proceedings of the ACM Conference on Object-Oriented Programming Systems, Languages and Applications*, pages 371–380, New Orleans, LA, October 1989. SIGPLAN Notices 24(10), oct 89.
7. Anand S. Rao and Michael P. Georgeff. BDI agents: From theory to practice. In Victor Lesser, editor, *Proceedings of the 1st International Conference on Multi-Agent Systems*, pages 312–319, San Francisco, USA, June 1995. IEEE Computer Society Press.
8. Jeffrey S. Rosenschein and Gilad Zlotkin. *Rules of Encounter: Designing Conventions for Automated Negotiation among Computers*. MIT Press, Cambridge, MA, 1994.
9. Jaime Simão Sichman. *Du Raisonnement Social Chez les Agents: Une Approche Fondée sur la Théorie de la Dépendance*. Thèse de Doctorat, Institut National Polytechnique de Grenoble, Grenoble, France, 1995.
10. Herbert A. Simon. *Models of Man*. Wiley, New York, 1957.
11. Michael Wooldridge and Nicholas R. Jennings. Towards a theory of cooperative problem solving. In Yves Demazeau, Jean-Pierre Müller, and John Perram, editors, *Pre-proceedings of the 6th European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, pages 15–26, Odense, Denmark, August 1994.
12. Eric S. K. Yu. Modelling organizations for information systems requirements engineering. In IEEE Computer Society Press, editor, *Proceedings of the 1st IEEE International Symposium on Requirements Engineering*, pages 34–41, San Diego, CA, January 1993.

An Autonomous Agent Architecture and the Locomotion Problem

Claudio Cesar de Sá[1], Guilherme Bittencourt[2], and Nizam Omar[3]

[1] DCC/CCT/UDESC - Campus Universitário, 89.223-100 - Joinville - SC - Brazil
dcc2ccs@dcc.fej.udesc.br
[2] DAS/CTC/UFSC, 88040-900 - Florianópolis - SC - Brazil
gb@lcmi.ufsc.br
[3] IEC/ITA/CTA, 12229-800 - São José dos Campos - SP - Brazil
omar@comp.ita.cta.br

Abstract. This paper presents a three level architecture for an autonomous agent and its application to the navigation problem in an unknown environment. The architecture is structured in three levels, called, reactive, instinctive and cognitive. The reactive level is based on a feed-forward symbolic neural network. The instinctive level is based on a set of predefined behaviors selected by a fuzzy classifier according to the perceived situation. Finally, the cognitive level is supported by symbolic production rules that determine the global behavior of the agent. In this sense, the three levels are responsible by behaviors of increasing complexity. The main characteristics of the architecture are: heterogeneity, hierarchic assembly, behavior-oriented design and biological plausibility. Some examples are also presented, that show the behavior robustness of the proposed architecture in a simulated environment.
Keywords: mobile robots, fuzzy control, neural networks.

1 Introduction

In unknown or dynamically changing environments, the symbolic approach from *Artificial Intelligence* (AI) is not very effective to solve the navigation problem, because of the knowledge maintenance problem [12]. Another approach is to consider a *subsymbolic* or *implicit* representation [2]. In this approach neither a centralized global control nor a symbolic representation of the environment are necessary to determine some adequate sequence of actions [5]. The *Distributed Artificial Intelligence* [6], on the other hand, adopts implementation techniques that are a blend of classical and non-classical approaches [13].

In this paper, we present a general architecture for AA [4] and its instantiation into a system that simulate a mobile AA moving in an unknown environment. The architecture is structured in three levels: *reactive*, *instinctive* and *cognitive*. Functionally, these three levels are similar to the *reactive*, *deliberative* and *meta-management* components of Sloman's architectures for human-like agents [20]. The model also presents some similarities with the Rasmussens's models

F. Moreira de Oliveira (Ed.): SBIA'98, LNAI 1515, pp. 11-20, 1998.

[17]. The proposed architecture can be characterized as a multi-level heterogeneous multi-agent society[1]. The main contributions of the paper are the design of an architecture, integrating neural networks, fuzzy inference and productions systems in a biologically plausible mechanism, and its application to the simulated locomotion problem, where quite robust results were obtained at a very low computational cost. This architecture has been already used in the implementation of a simulated mobile AA whose only expected behavior was to "wander" in an unknown environment [19].

The paper is organized in the following way: in Section 2, the physical structures of the agent and environment are defined. In Section 3, the main features of the general architecture for an AA are sketched according to its levels – reactive, instinctive and cognitive. In Section 4, some examples are presented, demonstrating the robustness of the implemented prototype. Finally, in Section 5, some conclusions and future works are commented upon.

2 Agent and Environment Structure

The adopted simulated autonomous agent has the following characteristics: three proximity sensors, one angular sensor, four wheels and three motors (see figure 1). The proximity sensors simulate physical sonar or infrared sensors and have a short distance range. In animals they would correspond to antennae. They are distributed one in each side of the agent and one in the front side. The angular sensor is independent of the distance and indicates the direction of a predefined target, with respect to the present orientation of the central axis of the AA. It could be physically implemented through some electromagnetic wave – light or microwaves, for instance – detector and it is intended to simulate the olfactory sense of animals. Finally, the AA has three motors connected to its front wheels, one in the left, one in the right and one in the front. If the lateral motors are activated with different speeds, the AA turn to one side or the other.

The environment consists of a rectangular workspace where obstacles, that consists of arbitrary polygons, can be introduced through a graphical interface.Typical environments in our experiments are shown in figures 3, 4 and 5. Some environments also contain *targets*, i.e., fixed points that are used to determine the value of the angular sensor of the AA. Finally, each particular environment contains a unique *exit* point. When the AA reaches this point, the simulation terminates.

3 The Underlying Architecture

The proposed general architecture for a mobile AA consists of three biologically inspired hierarchical levels. The lower levels are associated with sensory

[1] The proposed architecture for a mobile agent is based on three multi-agent societies. This can lead to confusion about the meaning of the term "agent". To avoid this confusion, when referring to the agents in these three societies, we always mention the name of the society – reactive, instinctive or cognitive.

information and reactive actions, the higher levels are associated with more sophisticated survival mechanisms such as instincts, memory and reasoning. The following sections describe each of the three proposed levels and how they were implemented.

3.1 Reactive Level

This level consists of a three layer feed-forward symbolic neural network [18]. These symbolic neural networks differ from the usual neural networks in the sense that its neurons interchange symbols among them, instead of numbers as in the usual neural networks. In our case, each symbolic neuron is in fact a *fuzzy inference machine* and the symbols interchanged consists of *linguistic variables* associated with *fuzzy sets* [11]. The symbolic neural network in the reactive level of our AA has three inputs – x_l, x_f and x_r –, associated with the values of the proximity sensors outputs. The distance from an obstacle is normalized into values between 0, for a full blockage, up to 1, for a free path. The symbolic neural network for this level is sketched in figure 2. This neural network has a biological inspiration, where each neuron is *exciting* or *inhibiting* the others. The functionality and the semantic of the connections in the network have similarities with R. Beer's work [3].

The reactive agents in each layer of the network present similar functions. The neurons in the first layer just distribute the fuzzified input values to the neurons in the hidden layer. This hidden layer can be interpreted as encoding the "resources" to be used by the three motors. The output of the third layer is used to calculate the pair (d, θ) that determines one step of movement of the AA. This can be formalized in the following way. At each time instant, the behavior of the AA is determined by one of a set of functions: $b_k : P \times S \to A$, where P is the perception field given by the tuple (x_l, x_f, x_r, ϕ) (although – ϕ –, the target angle, is not used in the reactive level, we include it in the perception field for simplicity, and also because we intend to used it, at this level, in future tasks), S is the internal state of the AA (see section 3.3) and A is the possible actions set, given by pairs (d, θ), meaning that the AA will move d units of distance in the direction given by θ radians with respect to its central axis.

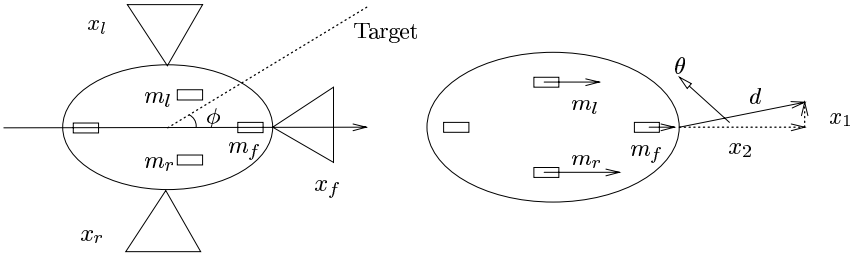


Fig. 1. The Autonomous Agent

Given m_l, m_f and m_r, the defuzzified values obtained from the output associated with the nodes #8, #7 and #9, where the adopted defuzzification method is the *Center of Area* [22], the action (d, θ) is determined through the following formulas (see figure 1):

$$x_1 = \mid m_l - m_r \mid \qquad x_2 = m_f + 2\, min(m_l, m_r)$$

$$d = \sqrt{x_1^2 + x_2^2} \qquad \theta = \frac{m_l - m_r}{\mid m_l - m_r \mid}\, arctan\left(\frac{x_1}{x_2}\right)$$

where the factor $\frac{m_l - m_r}{\mid m_l - m_r \mid}$ determines the sign of the θ angle and, therefore, the direction of vector x_1. These formulas are not intended to be an exact physical model of a real robot but, for our purposes, it is a good enough approximation. To avoid that the AA collide with an obstacle previously out of the range of the proximity sensors, the distance to be cover by each movement – d – is normalized in such a way that its maximum corresponds to 50 % of the proximity sensor maximum range.

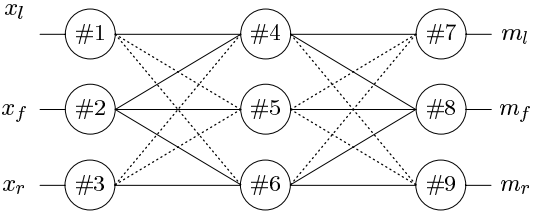


Fig. 2. The Reactive Level Network

The rule bases in each reactive agent were experimentally determined with the goal of producing a robust wandering behavior, where the AA avoids obstacles but does not have any specific target location to reach. During this experiments, we concluded that the proposed neural network would work better and more economically if some of its arcs have weight zero. The neural network that presented the best results in the experiments and that was adopted in the AA implementation is shown by the solid lines in figure 2. This reduced network has presented a robust behavior for the wandering navigation problem. However, because of its inherent symmetry, it presented conflict problems. Consequently, a superior level is necessary to suppress or to control conflicts.

3.2 Instinctive Level

In order to implement the intended skills of reaching a target and keeping the AA near the left or right wall, it is necessary that the agents of the reactive level be submitted to some restrictions on their behavior. These restrictions are implemented through the instinctive level agents. The only action of these

instinctive agents is to enable or disable the action of specific reactive level agents. They monitor the proximity sensors and the target direction and, if the AA either moves in the opposite direction with respect to the target, or approach to much a wall, or get to far from a reference wall, they opportunistically disable a group of reactive agents in such a way that the AA takes the appropriate action. Each group of reactive agents that remains activated after the action of an instinctive agent is called a *functional scheme* [2]. Each functional scheme can be formally defined as one of the behavior function defined in section 3.1: $b_k : P \times S \rightarrow A$.

In terms of our implementation, these behavior functions differ from each other only by the weights associated with the arcs of the neural networks and by the specific fuzzy rules associated with each of its nodes.

The instinctive agent that should take control at each moment is determined by a fuzzy classifier [16]. This classifier handles the usual cases with suitable performance. It uses the fuzzified values of the proximity sensors and, associated with each sensor, a filtering pattern that consists also of a fuzzy set. Each instinctive agent has threshold values to be activated, which were chosen empirically, considering singular contours and shapes in the environment. This can be formalize by a function: $h : P \times S \rightarrow B$, where B is the set of all possible behavior functions – b_k. The function h determines which behavior should be adopted in a given situation (and in a given state, see section 3.3).

Obviously, infinite instinctive agents would be necessary to handle all possible circumstances [21]. In our distributed implementation, unpredictable circumstances may also activate several instinctive agents, resulting in imprecise actions. To avoid this we have developed an auxiliary classifier that is triggered in case of conflict. This auxiliary classifier selects an instinctive agent by the smallest difference among the three inputs and its threshold values and it is inspired by the Minkowski distance, where the best matching occurs with smallest differences [11]. If no instinctive agent is activated, the reactive level is kept free to implement its default wandering strategy until a situation occur that triggers one of the instinctive agents.

Table 1. Behavior definitions

Behavior	Functional Scheme
Dead-end	$\{\#1, \#4, very(\#7)\}$ or $\{\#3, \#6, very(\#9)\}$ randomly
Keep-left	$\{\#1, \#2, \#4, \#5, fairly(\#7), \#8\}$
Keep-right	$\{\#2, \#3, \#5, \#6, \#8, fairly(\#9)\}$
Left-corner	$\{\#1, \#2, \#4, \#5, very(\#7), fairly(\#8)\}$
Right-corner	$\{\#2, \#3, \#5, \#6, fairly(\#8), very(\#9)\}$
Straight-away	$\{\#2, \#5, very(\#8)\}$ $\quad \theta = 0$
Go-round	$\{\#1, \#4, \#7\}$ or $\{\#3, \#6, \#9\}$ depending on ϕ
Left-turn	$\{\#2, \#3, \#5, \#6, \#8, \#9\}$
Right-turn	$\{\#1, \#2, \#4, \#5, \#7, \#8\}$

Each instinctive agent selects a functional scheme, i.e., a subset of reactive agents to remain active, and sometimes applies a *fuzzy quantifier* [11] to some of the outputs of the last layer reactive agents. The behavior definitions, in terms of active reactive agents and fuzzy quantifiers, are shown in table 1. The first behavior – *Dead-end* – is activated independently of the state. The next four behaviors – *Keep-left, Keep-right, Left-corner* and *Right-corner* – are used in the states where it is intended that the AA keeps moving along a wall. Finally, the last four behaviors – *Straight-away, Go-round, Left-turn* and *Right-turn* – are used in states where the target search mode is activated.

3.3 Cognitive Level

The main goal of the cognitive level is to establish the global plan of the AA behavior. Its functions include the definition of the targets to be reached by the AA, the determination of which wall – right or left – the AA should keep near of, the definition of the appropriate threshold values that activate the instinctive agents and, finally, the specification of which instinctive agents are allowed to compete for control at each moment.

The cognitive agents monitor the instinctive level agents and in this sense they present some kind of primitive memory. The action of the cognitive level, in its simplest form, can be formalized by the function: $p : B^n \times E \times S \rightarrow S$, where B^n is the set of all tuples of behavior functions of size n and E is a finite set of *special signals*. The function p determines the next state of the AA, given the last n behavior functions adopted by the AA at the instinctive level, some special signal and the present state.

The special signals in E are intended to simulate internal goals of the AA, such as thirst or hunger, in the case of animals, or low battery level, in case of mobile robots. In our implementation, these are limited to a few situations: $E = \{normal, target, exit\}$, where the symbol *normal* is associated with the absence of special signal, the symbol *target* indicates that a predefined target point has been reached and, analogously, the symbol *exit* indicates that the exit point has been reached.

Formally, each state in S can be thought of as a binary tuple where each element corresponds to a particular behavior function – b_k. According to the value of the element of the tuple - 1 or 0 – the associated behavior is allowed or not to compete for the reactive level control. In this formalization, the *null* state, where all elements of the tuple are zero, corresponds again to the default wandering behavior.

The hierarchical nature of the cognitive level preserves the robustness of the lower levels functionalities. Being independent of the sensory input, the cognitive agents are purely symbolic and, therefore, they are implemented through symbolic production rules. The production rules that implement the function p have the following form: **if** $RE(B)$ **and** $signal = e_i$ **and** $state = s_i$ **then** *new-state* $\leftarrow s_j$, where $e_i \in E$, $s_i \in S$ and $RE(B)$ represents a syntactical condition expressed by a *regular expression* [8] over the alphabet B, a set of symbols associated with the behavior functions b_k. It is necessary that this condition is satisfied by the

last n adopted behaviors for the rule to be fired. Regular expressions are expressive enough to allow the definition of any finite language. As the elements of B^n are tuple of a finite size n and the number of possible behaviors is also finite, i.e., $\mid B \mid < \infty$, any particular pattern, or class of patterns, of behavior function symbols can be described by a regular expression. The regular expressions have also the interesting property that, given a language defined by some regular expression, it is possible to automatically generate an *automaton* [8] that recognizes this language. The possibility of implementing the cognitive level based on automata increases the biological plausibility of the proposed architecture, because even the adopted *symbolic* processing can be implemented using subsymbolic mechanisms, like neural networks [16].

4 Experimental Results

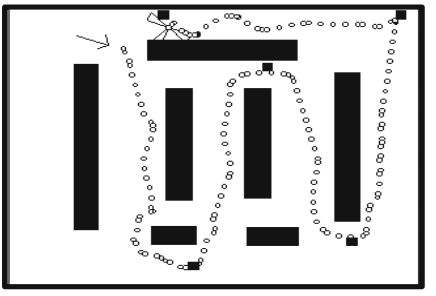
In this section, we show through some examples the robustness of the proposed AA architecture in several environments. In the first example, the intended behavior is that the AA keeps its left proximity sensor near the left wall, in order to find the exit out of a (topologically restricted) maze. The chosen path demonstrates the ability of the AA to navigate in unknown environments, similar results have been obtained by [1]. To obtain this behavior only four of the instinctive agents were necessary. The path through the maze and the active instinctive agents, along with their threshold values associated with each of the proximity sensors, are shown in figure 3. These threshold values were determined experimentally.

Behavior	x_l	x_f	x_r
Dead-end	0.13	0.13	0.13
Keep-left	0.45	0.99	0.85
Left-corner	0.35	0.99	0.45
Straight-away	0.99	0.99	0.5

Fig. 3. A Typical Maze Environment

In the second example, the goal of the AA is to visit a sequence of locations in the workspace. In this case, the concept of *target* becomes relevant and more instinctive agents are necessary. In particular both instinctive agents that navigate near the walls are necessary: *Keep-left* and *Keep-right*. The path through the maze, the instinctive agents used in the example and their threshold values are shown in figure 4.

In the first two examples, the cognitive level action was trivial, in the sense that it was independent of the adopted instinctive level behaviors. Its effect was just to change the state of the AA, when each event of reaching a target or the exit occurred, i.e., in these cases, the function p depended only on E and not on B^n. The instinctive agents competing for the reactive level control did not change



Dead-end	0.13	0.13	0.13
Keep-left	0.5	0.99	0.85
Keep-right	0.99	0.5	0.85
Left-corner	0.35	0.99	0.35
Right-corner	0.99	0.35	0.35
Straight-away	0.99	0.99	0.5
Go-round	0.9	0.9	0.55
Left-turn	0.99	0.99	0.5
Right-turn	0.99	0.99	0.5

Fig. 4. A Multi-Target Environment

during the experiment. In the third example, the action of the cognitive level is more effective. Consider the environment presented in figure 5. The fact that the target is located beyond a closed corner would lead the instinctive agent society of figure 4 into an infinite loop. To prevent this situation, the cognitive level should present some rules whose conditions detect an infinite loop. A pattern condition, represented with the aid of the regular expression syntax, can recognize a loop to the left, given the number – N – of left turns without right turns that will define "infinite loop". When such a pattern is detected, the AA changes to a state where the targets are no longer taken into account, and it is kept wandering for some time. After some cycles, the state of the AA is returned to the target search mode, hopefully, from a point that will not lead the AA back into the loop. If it is the case, the wandering mode is triggered again, this time for a longer time.

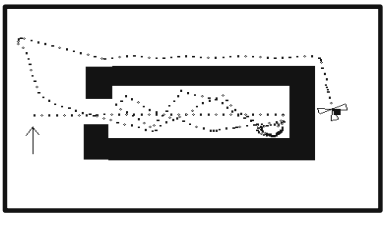


Fig. 5. A Hidden Target

This can be represented by the following production rule:

if $[(B - \text{Right-turn})^*\text{Left-turn}(B - \text{Right-turn})^*]^N$ **and** $signal = normal$ **and** $state = searching$ **then** $new\text{-}state \leftarrow wandering$

5 Conclusions

This paper presents a general multi-level, hierarchical, heterogeneous, multi-agent architecture for an AA. The proposed architecture is tested through the

implementation of a simulated AA in an unknown environment where it should navigate. The proposed architecture has three hierarchical levels, which had been defined and implemented under biological inspiration. The lower level has a wandering reactive ability implemented by a feed-forward fuzzy neural network. The intermediate level consists of instinctive agents whose actions enable or disable reactive level agents, according to a fuzzy classifier based on the proximity sensor output values. Finally, the upper level presents a society of cognitive agents, based on symbolic production rules, that acts over the instinctive level agents to generate interesting behaviors.

Although different AI techniques have been successfully combined in some systems (e.g., neural networks and expert systems [7]), no general theory of such *Hybrid Systems* is presently available. We identify two main contributions in our work: (i) the integration in a coherent architecture, that combines biological plausibility with interesting theoretical properties, of several distinct AI techniques, such as neural networks, fuzzy control and production rules, and (ii) the efficacy and robustness of the obtained behavior and its very low computational cost.

With respect to the AA's computational cost, the global behavior of the AA is determined by the frequent execution of some function b_k in order to calculate the next action. These functions are just a feed-forward fuzzy inference process, that can be compiled into a very fast algorithm. The AA also needs to calculate, less frequently, the function h that determines the next behavior that should take control. This function is a fuzzy classifier, also a low cost function. Finally, even less frequently, it is necessary to calculate the function: p that monitors the adopted behaviors and the special signals and enable appropriate instinctive agents to compete for the reactive level control. This function is implemented through a production rule system. If we compare the cost of these functions to the cost of the alternative "representational" approach, where the environment would be geometrically represented and the paths would be determined by complex planning algorithms, it is surprising that the locomotion performance obtained is so similar, with so much less resources.

Future projects include the integration of a *vision* sensor in the AA architecture, the definition of more instinctive agents and the implementation of more complex cognitive level examples. A medium term goal is to use a physical robot with the characteristics of the simulated AA, such as the Khepera robot [14], and to control its locomotion by remote control [9] using the proposed architecture. We also intend to use evolutionary computation to automatically tun the parameters of the system, and some *inductive learning* [15] or *reinforcement learning* [10] techniques to improve the efficacy of instinctive/cognitive levels interface.

References

1. T. L. Anderson and M. Donath. Animal behavior as a paradigm for developing robot autonomy. In Pattie Maes, editor, *Designing Autonomous Agents*, pages 145–168. Bradford Book - MIT Press, 1991.

2. M. A. Arbib and J.-S. Liaw. Sensorimotor transformations in the world of frogs and robots. *Artificial Intelligence*, 72:53–79, jan 1995.
3. R. D. Beer, H. J. Chiel, and L. S. Sterling. A biological perpective on agent autonomous design. In Pattie Maes, editor, *Designing Autonomous Agents*, pages 169–186. Bradford Book - MIT Press, 1991.
4. G. Bittencourt. In the quest of the missing link. In *Proceedings of IJCAI 15, Nagoya, Japan, August 23-29*, pages 310–315. Morgan Kaufmann (ISBN 1-55860-480-4), 1997.
5. P.A. Brooks. Intelligence without representation. *Artificial Intelligence (Special Volume Foundations of Artificial Intelligence)*, 47(1-3):139–159, January 1991.
6. E.H. Durfee. The distributed artificial intelligence melting pot. *IEEE Transactions on Systems, Man and Cybernetics*, 21(6):1301–1306, November 1991. Special Issue on Distributed Artificial Intelligence.
7. L. Fu. Rule generation from neural networks. *IEEE Transactions on Systems, Man and Cybernetics*, 24(8):1114–1124, August 1994.
8. J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison Wesley Publishing Company, Reading, MA, 1979.
9. M. Inaba. Remote-brained robots. In *Proceedings of IJCAI 15, Nagoya, Japan, August 23-29*, pages 1593–1606. Morgan Kaufmann (ISBN 1-55860-480-4), 1997.
10. L. P. Kaelbling, M. L. Littman, and A.W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.
11. G.J. Klir and T.A. Folger. *Fuzzy Sets, Uncertainty, and Information*. Prentice Hall, Englewood Cliffs, N.J, 1988.
12. P. Maes, editor. *Designing Autonomous Agents*. Bradford Book - MIT Press, 1991.
13. A. Meystel. Knowledge-based controller for intelligent mobile robots, 1986. Drexel University, Philadelphia, PA 19104, USA.
14. O. Michel. *Khepera Simulator Package version 2.0: Freeware mobile robot simulator*. University of Nice Sophia–Antipolis, France, 1997.
15. H. Motoda and K. Yoshida. Machine learning techniques to make computers easier to use. In *Proceedings of IJCAI 15, Nagoya, Japan, August 23-29*, pages 1622–1631. Morgan Kaufmann (ISBN 1-55860-480-4), 1997.
16. C. W. Omlin, K. K. Thornber, and C. Lee Giles. Fuzzy finite-state automata can be deterministically encoded into recurrent neural networks. Technical Report Technical Report CS-TR-3599 and UMIACS-96-12, University of Maryland, April 1996.
17. Steen Rasmussen. Aspects of information, life, reality, and physics. In C. Langton, C. Taylor, J.D. Farmer, and S. Rasmussen, editors, *Artificial Life II*. Addison-Wesley, SFI Studies in the Sciences of Complexity, Vol. X, Redwood City, CA, 1991.
18. A.F. Rocha. *Neural Nets: A Theory for Brains and Machines*, volume 638 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 1992.
19. C.C. de Sá, G. Bittencourt, and N. Omar. An architecture for a mobile autonomous agent. In E. Gelenbe and N. Schmajuk, editors, *Proceedings of the Workshop on Biologically Inspired Autonomous Systems*, March 1996.
20. A. Sloman. What sort of architecture is required for a human-like agent ? In *Proceedings of AAAI-96*, August 1996.
21. R. M. Turner. Context-sensitive reasoning for autonomous agents and cooperative distributed problem solving. In *Proceedings of the 1993 IJCAI Workshop on Using Knowledge in Its Context*, 1993.
22. R.R. Yager and D.P. Filev, editors. *Essentials of Fuzzy Modeling and Control*. John Wiley & Sons, Inc., 1994.

Building Object-Agents from a Software Meta-Architecture

Analía Amandi[1] and Ana Price[2]

[1] Universidad Nac. del Centro de la Pcia. de Bs. As. - Fac. de Cs. Exactas - ISISTAN
San Martin 57 - (7000) Tandil - Bs. As. Argentina
amandi@exa.unicen.edu.ar
[2] Universidade Federal do Rio Grande do Sul - Instituto de Informática
Caixa Postal 15064 - Porto Alegre - RS Brasil
anaprice@inf.ufrgs.br

Abstract. Multi-agent systems can be viewed as object-oriented systems in which their entities show an autonomous behavior. If objects could acquire such skill in a flexible way, agents could be built exploiting object-oriented techniques and tools. There are several ways for building agents from objects: defining common interfaces and behavior in abstract superclasses, wrapping objects with agent behavior using composition techniques, etc. However these ways present problems for becoming objects in agents and for adapting the behavior assigned to agents, especially whether it is required in a dynamic way. This article presents one alternative in which agent characteristics (such as perception, communication, reaction, deliberation and learning) can be dynamically added, deleted, and adapted to objects using a particular computational reflection form achieved by meta-objects.

1 Introduction

Agent-oriented and object-oriented programming work on cooperative entities. These entities have limited competence, which is defined by a set of actions that are able to execute. However, the following definitions show an important difference between the entities that each one of the paradigms work on.

An agent is an autonomous entity that has action capabilities, which are used in a non-deterministic behavioral way and internal knowledge for supporting their decisions and the execution of its behavioral capabilities. An object is an entity that has action capabilities defined in a deterministic way and internal knowledge for supporting the execution of its behavioral capabilities.

Objects and agents define a well-bounded behavior but there is one difference: objects use it in given ways and agents can decide the way they take. Thus, additional characteristics must be designed for achieving object-agents: communication among agents, perception of changes in the environment, and decision processes related to what to do next.

Communication capabilities determine how agents of different types can communicate among them. Examples of communication performatives: achieve (aGoal), askIf (aQuery), askAll (aQuery), etc.

F. Moreira de Oliveira (Ed.): SBIA'98, LNAI 1515, pp. 21-30, 1998.

Perception capabilities allow an agent to know facts without receiving any information from communications. Perception capabilities allow an agent to observe activities performed by other agents (a conversation among agents, transference of information between agents, destruction of agents, agents changing of communities, etc.) and environment changes (i.e. external sensors becoming enabled, locking of internal structures, etc.).

Decision capabilities determine the course of action that an agent takes in a given point of time. Such course of action can be characterized as reactive or deliberative [5]. Reactive behavior allows agents to act without thinking before act. Deliberative behavior shows a meditative selection of a particular set of actions. We can recognize three behavioral levels in agents (see Figure 1). The lower level shows us what an agent can do in terms of actions. An agent robot, for example, could execute the following actions: go, turnLeft, turnRight, etc. This level represents the characteristic behavior of objects. The next level represents social capabilities inherent to agents. Objects only can receive known messages, they don't manage a common protocol of communication as agent can do. Agents communicates with other agents requesting help to achieve a given goal, giving notice of a trouble, advising about a problem, giving information, etc. In addition, agents can observe events happened in their environment. The third level represents the set of decision capabilities that are used for agents to determine actions to execute. To do this, the observed events and the received communications are considered basic information for such decision processes.

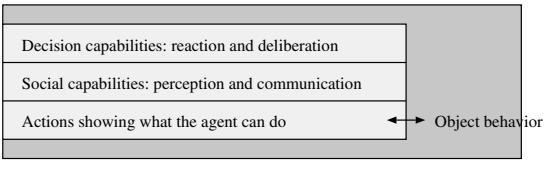


Fig. 1. Agent behavioral levels

A multi-agent system can in summary be considered as an object-oriented system in which object-agents has associated extra capabilities. This idea is not new, [8, 16] have shown the potential characteristics of the object-oriented paradigm for building agents. However, there are several alternative ways in which these extra capabilities are associated to simple objects.

This paper presents a software architecture named Brainstorm. This architecture allows objects to be the main support of agents and meta-objects to be the way for flexible adding the mentioned agent-specific capabilities.

The breakdown of the paper is as follows. Section 2 identifies the advantages of the usage of meta-objects. Section 3 presents an architecture based on the meta-object concept. Section 4 relates results and experiences with the architecture. Sections 5 and 6 expose related works and conclusions.

2 Agents from meta-objects

Some languages for building agents accept classes representing types of agents. This approach has one disadvantage it doesn't take software reuse into account.

Using inheritance and composition techniques, behavior and decision reuse become possible. However, this design presents some troubles: agent developers must manage each relation among agent parts and the perception design forces changes in code of the object. The administration of parts involves control of the activation order (i.e. react before deliberate) and implicit delegation of execution responsibility. The perception solution through change announces imposes alterations in the basic code of agents, avoiding the full possibility of dynamic changes in the observation targets.

The usage of computational reflection through meta-objects is an alternative that has not been studied until now. Computational reflection is an activity carried out by a computational system when it executes computations on its own computation [11]. From this point of view, a reflective system is divided in two levels: an object level and a reflective level. The object level manages the actions and data of the application domain. The reflective part, located in a meta-level, manages actions and data of the object level.

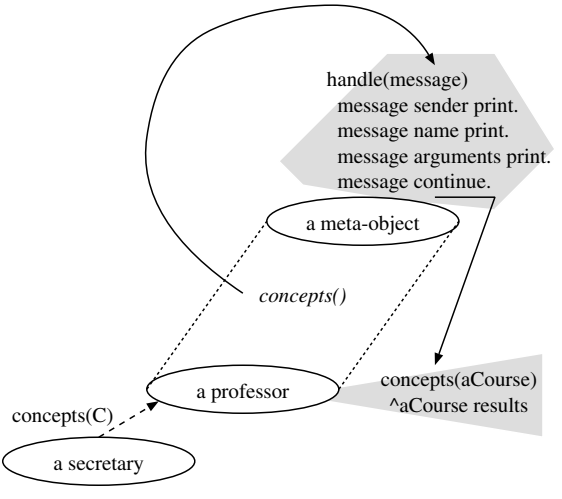


Fig. 2. Meta-objects

Figure 2 shows an example of computational reflection from an object-oriented point of view. In the object-oriented approach, the concept of meta-object materializes the concept of computational reflection. The example shows as a meta-object interferes in the computation of the messages received by a given professor. The meta-object prints the sender, the name, and the arguments of each message

received and so that returns the control to the professor. Coming back with the agent design, the problems found using only inheritance and composition can be solved using computational reflection techniques. Meta-objects can interfere in the passive behavior of objects turning it an autonomous behavior. Therefore, a multi-agent system can be considered as an object-oriented system, which has associated to it a meta-system responsible for a typical agent behavior. The next section presents a solution for designing agents from object in this way.

3 The architecture Brainstorm

A software architecture [15] describes software components, patterns of component composition and restrictions on these patterns, which allow systems to be built. An agent architecture allow multi-agent systems to be built in term of given components and interfaces among them.

We have developed an agent architecture named Brainstorm for extending objects to support agent behavior. In such way, agent characteristic behavior has been encapsulated in components and specific interfaces have been defined for connecting those agent components.

In the Brainstorm context, a multi-agent system is defined as a reflective system, which is causally connected to an object-oriented system defined in the base level. The causal connection is established when an object receives a message: it is intercepted by the meta-level, which decides what to do with it. For example, an agent can decide to deny a specific request of another agent.

The architecture has been built using a meta-object approach. An object and a set of meta-objects define an agent. Each meta-object incorporates agent functionality to simple objects.

The functionality that can be added to objects from meta-objects can be observed in Figure 3. We can see two object-agents. The agent A has associated seven meta-objects and the agent B has four. Requirements of each agent originate the necessity of different quantity and types of meta-objects. For example, the agent A uses a particular communication language (by a communication meta-object), a particular set of mental attitudes (by a knowledge meta-object) and functionality related to their hybrid behavior (reaction (by a reactor meta-object), deliberation (by a deliberator meta-object) and learning (by a learner meta-object)), and perceive changes in its environment originated by the agent B and one given object (by two perception meta-objects).

In the Figure, we can see agent A observing agent B by a perceptor meta-object. In addition to perceptive connections, agents can interact among them sending messages using a communication language. Also, cooperative decisions can be made using a special object (cooperative element in the Figure), which can be acceded by a given set of agents in contrast of the rest of the private components. Several reflective levels composed by meta-objects can be observed in each agent. Each level is connected to the lower level by reflection. In other words, each level intercepts all event of its lower level and changes its limited behavior. A communication meta-object, a knowledge meta-object and a set of

perception meta-objects compose the first meta-level. These meta-objects works on messages received by the base level (the object base). This level is responsible for recognizing messages that belong to the communication language, for managing the mental state, and observing messages that can represent interesting situations for the agent.

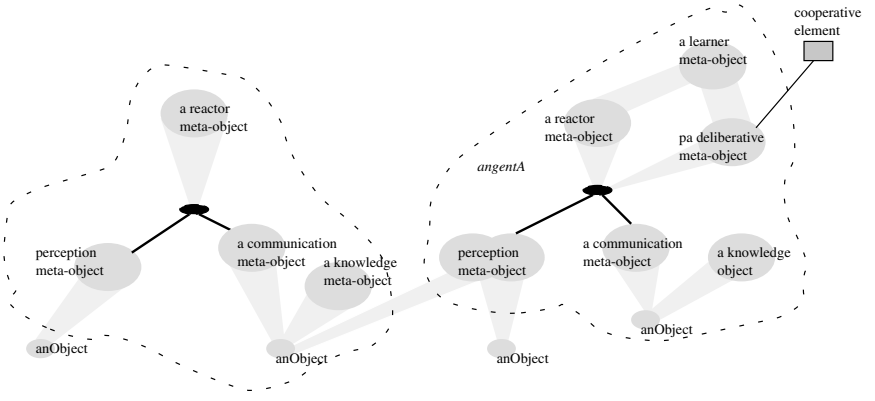


Fig. 3. Brainstorm agents

The second meta-level is responsible for acting. The possible ways for acting are reaction and/or deliberation. Therefore, it is possible have in this level a reactor meta-object and/or a deliberator meta-object. These meta-objects work on communications processed by the communication meta-object and on interesting situations discovered by the situation manager. A situation manager (black circle in Figure 3) analyzes whether a situation of interest is present. For it, it uses as data the received communications and the messages observed by perceptors.

The third meta-level is optional. Learner meta-objects are responsible for learning from experiences and for using these experiences in future decisions. These meta-objects can work on reactions and on decision algorithms used by deliberators.

The mentioned meta-objects are the main architectural components in terms of agent instances. In terms of agent class there is n architectural component named MetaAgent, which is materialized in the meta-level of a class. This component is responsible for the process of meta-object instantiation. When an agent class receives a class instantiation message, its associated meta-agent intercepts it and generates the meta-level of the new object-agent. Meta-agents create the meta-objects needed for satisfying the agent specification. Thus, it is possible to define different types of agents in the same system.

The following sections present in detail each component of the architecture Brainstorm.

3.1 The communication component

Communication meta-object classes define communication protocols, which may be adopted by agents. Agents that want to interact have associated instances of the same communication meta-object class. When an object-agent receives a message m that belongs to its communication language, this message is intercepted by its agent communication meta-object by a classical meta-object mechanism. This mechanism actives the methods handleMessage() of the first meta-level with the intercepted message as argument.

That message is then analyzed by the communication meta-object for deciding what to do with the message. If the message has been defined either in the object class as an acting capability or in its communication meta-object class as part of its communication language, it is registered on a communication message base using an architectural protocol register(). The priority of managing that message is lower for default.

In this occasion, the deliberator meta-object can intercept the message record() to define other message priority. For doing so, the deliberator uses its own mental state. For instance, if a request sender is the boss of the receiver, it has the highest priority of attending. Then, the control returns to the meta-communicator and it records that message in the priority defined by the meta-deliberator.

Each one of the meta-components has defined independent responsibilities. This meta-component work together showing a transparent connection by a reflection mechanism.

3.2 The perception component

Agents can observe events taking place in their environment. Agents and objects compose the environment in Brainstorm, and so the message passing is the only source of events. These events can draw a situation of interest for agents.

To perceive events, agents shall include definitions of perception meta-objects (instances of MetaPerception class). These meta-objects observe other agents or objects, reflecting their received messages. Both agents and objects can be observed without they know they are observed in contract of the classical announce of changes where the observed object must announce its movements.

Using meta-objects for percepting events in the environment, agents have two advantageous. One is that the transparency of the perception, the observed agent or object is not conscience of the observation. The other advantageous is that the agents can in runtime move their observation targets.

This architectural component use the common protocol for meta-objects, the method handleMessage() for intercepting observed messages.

3.3 The situation manager component

The situation manager is responsible for searching situations of interest. For doing so, it evaluates its perceptions, received messages and the mental state. This component has static connections with components of the firts meta-level.

Perceptor meta-objects informs to the situation manager all observed event, the communicator meta-object informs to the situation manager all received communication. The situation manager can see and use the knowledge and believes encapsulated in the component named brain. Here we present one architectural component, brain. This component is responsible for the mental state of the agent. Details about this component are showed in the next section.

The situation manager uses information about perception, communication and mental attitudes for detecting interesting situation. Whenever an interesting situation is detected, a method using a protocol newSituation() is invoked on the own situation manager. Transparently this message is reflected by the meta-objects MetaReactor and MetaDeliberator of the next meta-level. These meta-objects decide how to manage such new situation.

For example, an agent Agenda may want to know when given people take particular types of commitments. The agent wants to be more careful on the evaluation of accepting of commitments that involve people that are related to rivalry groups. The agent could ask to each agent agenda for such information (through a message), but a better solution would be to put sensors on the agendas to observe their commitment management.

The implementation of these sensors is very simple: the agent agenda associates a perception meta-object to the another agendas to observe the reception of the messages canAccept(aCommitment) and accept(aCommitment). With information about intercepted messages and the mental state, the agenda' situation manager can analyze whether such commitment is a interesting fact. If it is the case, it sends the message newSituation() to itself, which is reflected by both the reaction and deliberation meta-objects that evaluates what to do with such new situation. The reactor can produce a reaction. The deliberator can analyze what actions to take.

3.4 The knowledge component

This architectural meta-component permits objects managing logical structures for mental attitudes. Object-agents need an instance of this component.

The architecture prescribes objects using modules composed by logic clauses for managing mental attitudes. The type of clauses this component can manage is selected among a big set of possibilities since there is not a general agreement about both the set of mental attitudes and the relationships among them that should shape agent mental states. Several formal frameworks have been proposed [3, 14, 17, 10] in order to elucidate relations among mental components. Particular agents can adopt any extension of classical logic programming that applies these theoretical concepts.

The object uses logic modules in the base level. This usage is reflected by the meta-knowledge and managed by the component named brain. The component Brain has defined as its interface the needed protocol for maintaining and consulting a knowledge base. The Brainstorm architecture prescribes that Brain is a specialization of the classical logic programming.

From the programming point of view, this capability is achieved since an integration of an object-oriented language with a logic language is possible. Extensions of classical logic programming (such as [4,2]) for managing mental attitudes are used as a base of the component Brain.

The extensions to a Prolog interpreter can be easily added as specialization of Brain component since the knowledge meta-object administrates the integration with objects. This meta-object permits the combination of different logic modules referred by instance variables and methods and makes it temporally available for use.

3.5 The reaction component

The reaction meta-object has the functionality of reaction to given situations. A situation and a sequence of actions that are executed when this situation is detected compose a reaction itself. So this component would in summary be to use for quick answers to particular situations.

The reaction meta-object of an agent manages immediately each detected interesting situation whether a reaction was defined for such situation. After that, the deliberator also can act.

The Brainstorm architecture prescribes a meta-object with reflective connections with the situation manager in the lower level and with the learner meta-object in the upper level. This prescription has the flexibility of changing the behavior of agents as main advantageous.

3.6 The deliberation component

The deliberation component is responsible for the intelligent and autonomous acting form of the agent. The architecture prescribes a meta-object for this component. This meta-object is connected by reflection with the situation manager, the meta-communicator and the meta-learner (if there is one).

An agent acting form is determined by the manner in which it has to act when: (a) it perceives a situation of interest, (b) it receives a message, and (c) it decides to do something. Different acting forms are defined in different subclasses of the MetaDeliberator class or its parts. Brainstorm architecture defines two important parts for abstracting decision behavior: a simple decision algorithm and a social decision algorithm.

When an agent perceives a situation (by its situation manager), it must decide what to do in such situation. For doing so, the meta-deliberation intercepts the detected situation and delegates it to one of its simple decision algorithm.

When an agent receives a message, it has to attend the message. For doing so, first the meta-deliberation intercepts the received message for defining a priority of attending. Second when the message is attended it delegates the communication message to one of its social decision algorithm. Such algorithm analyzes the communication considering the social state and decides what is the answer. An agent also act without the necessity of any communication or perception of situations. A quantum of time is given to each agent for acting freely

if parallel process is not possible. Agents use this time for treating pending communications, making plans to achieve their goals (using the available planning algorithms), analyzing their old plans, goals and intentions, and executing actions in their environment. Therefore, it is necessary to have defined priority rules for guiding the decisions about what to do next.

3.7 The learner component

The learner component is defined as a meta-object by the Brainstorm architecture. It intercepts reactions and deliberations. The reactions are analyzed permitting future changes on themselves. Deliberations can be altered incorporating new points of view on the object on analyses.

For example, an instance of this component is a case-based reasoning engine. This element can intercept, for example, a deliberation process that uses a planning algorithm, deciding the resulting action plan.

4 Results and experiences

The Brainstorm architecture has been formally specified using the Z language. For these schemes, seven properties were proved. We have been proved that all message received by an object-agent take a right way from an agent behavior point of view. Detail of these proves can be found in [1].

In addition to the formal results, two experiences were made. First, a multi-agent system for simulating robots in a room was developed using the Smalltalk language. Second, a multi-agent system for managing personal agendas was implemented in the same language, reusing several decision algorithms.

These experimental cases showed that the abstractions defined in Brainstorm can be directly used and helped to construct a base of reusable decision algorithms and communication languages.

5 Related work

Several agent architectures have been proposed in the literature (i.e. [6, 12, 9]) describing the main components and interfaces of software agents. These agent architectures prescribe a computational solution for agent construction.

In the other hand, lots computational solutions of software agents have been built on object-oriented languages and lots agent specific languages have been developed on object-oriented concepts [7, 13]. In spite of that, the existent agent architectures have not considered in their design the usage of object-oriented concepts.

6 Conclusions

Brainstorm is a software architecture that prescribes an object-oriented agent solution, allowing thus the usage of object-oriented languages in the agent implementations. The object-oriented base has two advantageous: old object-oriented

systems can be extended for becoming multi-agent systems and all support for object-oriented systems can be used for building agents. The Brainstorm architecture prescribes several reflective components materialized by meta-objects. This prescription makes possible the dynamic modification of the structure and behavior of agents. Such flexibility also allows old objects become agents.

References

1. A. Amandi. *Programação de Agentes Orientada a Objetos.* PhD thesis, Porto Alegre: UFRGS, Brasil, 1997.
2. A. Amandi and A. Price. An extension of logic programming for manipulating agent mental states. In *Proceedings of the Artificial Intelligence and Soft Computing*, pages 311–314, 1997.
3. P. Cohen and H. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42(2), 1990.
4. M. Costa Mora, J. Lopes, and H. Coelho. Modeling intentions with extended logic programming. In *Proceedings of the Brazilian Symposium on Artificial Intelligence*, pages 69–78, 1995.
5. Y. Demazeau and J. P. Muller. From reactive to intentional agents. *In Decentralized Artificial Intelligence 2*, pages 3–14, 1991.
6. I. Ferguson. Touringmachines: Autonomous agents with attitudes. *Computer*, 25(5):51–55, May 1992.
7. M. A. Fisher. Survey of concurrent metatem-the language and its applications. In *Temporal Logic*, pages 480–505. Springer-Verlag, 1994. (LNAI 827).
8. L. Gasser and J. P. Briot. Object-oriented concurrent programming and distributed artificial intelligence. In *Distributed Artificial Intelligence: Theory and Praxis.* Kluwer, 1992.
9. B. Hayes-Roth. An architecture for adaptive intelligent systems. *Artificial Intelligence*, 72(1-2):329–365, January 1995.
10. Z. Huang, M. Masuch, and L. Pólos. Alx, an action logic for agents with bounded rationality. *Artificial Intelligence*, 82(1):75–127, January 1996.
11. P. Maes. Concepts and experiments in computational reflection. In *Proceedings of the Object-Oriented Programming Systems, Languages, and Applications Conference*, pages 147–155, 1987.
12. J. Müller and M. Pischel. Modeling interacting agents in dynamic environments. In *Proceeding of the European Conference on Artificial Intelligence*, pages 709–713, 1994.
13. A. Poggi. Daisy: An object-oriented system for distributed artificial intelligence. In *Intelligent Agents*, pages 341–354. Springer-Verlag, 1995. (LNAI 890).
14. A. Rao and M. Georgeff. Modelling rational agents within a bdi-architecture. In *Proceedings of Knowledge Representation and Reasoning (KR'91)*, pages 473–484, April 1991.
15. M. Shaw and D. Garlan. *Software Architectures.* Prentice Hall, 1996.
16. Y. Shoham. Agent-oriented programming. *Artificial Intelligence*, 60(1):51–92, March 1993.
17. B. van Linder, W. van der Hoek, and J. Meyer. Formalising motivational attitudes of agents: On preferences, goals, and commitments. In *Intelligent Agent II.* Springer-Verlag, 1996.

Agent's Programming from a Mental States Framework

Milton Corrêa[1] and Helder Coelho[2]
[1] Serviço Federal de Processamento de Dados, Rio de Janeiro, Brazil
correa@unisys.com.br

[2] Departamento de Informática, Faculdade de Ciências, Universidade de Lisboa, Portugal
hcoelho@di.fc.ul.pt

Abstract. In the paper we present a methodology to agents' programming based on a Mental States Framework (MSF) in which mental states types are defined in terms of basic components as an External Content; criterions to determine unsatisfaction, uncertainty, urgency, insistence, intensity and importance associated to a mental state; laws of causality through which a mental state can produce another; and control mechanisms for provoking, selecting, suspending and canceling the processing of a mental state.

Besides the mental states types named Belief, Desire and Intention as usually defined, this framework also includes Expectation and any possible other mental states' type that may be important when modeling interactions among agents in a more complex society.

According to this methodology, the agents' architectures are defined by selecting a set of mental states as established in this framework. And, in the paper it is shown that the Object Oriented Programming is well suitable and interesting to the implementation of those architectures. Therefore an Agent Oriented Programming is seen as an interaction among mental states and architectures spaces by MSF and the Object Oriented Programming.

1 Introduction

The agents' programming paradigm being the support for the fast technology development based on agents in the 90's comes to supply the growing processing complexity in what concerns information in organization, in process control in distributed system in which the techniques upon centralized control become impossible to be applied as for instance, international air traffic control, interoperability among knowledge systems or information exchange among heterogeneous information systems as in internet either in a company environment or in groups of companies where a lot of information are found in legacy systems [25].

Great efforts have been devoted in agents' conceptualization and theories concerning agents' interactions in a cooperative society, i.e. in building multi-agents systems (MAS).

Regarding the engineering out look, there are proposals for agents' architectures and programming techniques that are still under development. Nevertheless, there is a great gap not fulfilled between theory and agents' building engineering.

Most of MAS architectures may be described as a common set of components which suit the basic capacities for the agents: reactivity, that is, react to stimulation or environment changes; deliberation, that means, the agent decides according to reasoning; interactions

F. Moreira de Oliveira (Ed.): SBIA'98, LNAI 1515, pp. 31-39, 1998

with other agents; a set of actions; goal oriented; autonomy, that is, being able to take decision based on their internal states without external intervention [16].

A sort of architectures that has been largely explored in recent years is based in the agents' modeling from Beliefs, Desires and Intentions (the so-called BDI architectures) [2, 19].

In Corrêa and Coelho [10] is proposed a Mental States Framework (MSF) to standardize the commonly notion of usual mental states as Beliefs, Desires and Intentions and other not so much used in DAI as Expectations. It is shown that from this framework the research and application to build agents with new other possible types of mental states (as Hopes) can be possible. A space of mental states is built up on types of mental states and a space for possible agents' architectures is built up on that mental state space.

In this paper is shown that this framework can be applied to agent programming based on the Object Oriented Programming paradigm. Our aim is to establish a methodology: 1) The mental states are characterized in terms of a small set of basic attributes; 2) we argue that these attributes are, at least, sufficient to define mental states; 3) other possible mental states could also be characterized by assigning them a set of those attributes; 4) This mental states model can explain agents' interactions; 5) An Agent Programming as an extension of an Object Oriented Programming results from this mental states framework.

Our starting point is the observation that mental states (MS) as Beliefs, Desires and Intentions, are commonly defined as being structured on basic components. Cohen and Levesque [5] defined Intention as being structured on „Choice" plus „Commitment", in other works "Commitment" is also thought as more basic than Intention [4, 19]. Werner [24] treated Intentions as Strategies, and Bratman [2] defined them as Plans. The components of Belief usually are a Proposition and a "true" or "false" value associated to it or a degree of Certainty and can also have a degree of Importance [18].

On the other hand, Sloman[20, 21] pointed out that urgency, importance, intensity and insistence are behind agent's actions. In Corrêa and Coelho[8] and Corrêa[9] is shown that a complete functioning agent needs a specification based on the notion of architecture, but highlighted by the dynamics of mental reactions. The way all mental sates change and interact is carefully controlled [11]. We argue that other mental states than Belief, Desires and Intentions can be necessary to understand and to explain the complexities of agents behavior in a society and a complete and consistent theory about agents' mental states has not only to explain and build the artificial agents but also to understand the natural ones [12].

For instance, Expectation is a mental state that enables more flexibility and more complex behaviors [8, 17, and 23]. Expectations and Intentions can complement one another. Intentions can embody an agent's expectations that the agent act in ways to satisfy those intentions, while expectations can lead an agent to form intentions to check that those expectations are satisfied. Our approach considers mental states as organizations of agent's internal processing of information related to their actions, and a fundamental feature of these organizations is that they are related to situations in the world (intentionality). Mental states are guides for agent's actions such that the agent behavior can be explained on them and, on the other hand, mental states can interact to produce the agent behavior. We assume that there is a limited set of basic attributes such that a mental state is defined in terms of some combination of them.

2 A Framework for Mental States

In order to have a theoretical structure to define mental states we need to find an agreement about their basic components or attributes. This can be obtained by observing the conceptions and applications of the usual mental states and filtering a common set of them.

On the other hand, these attributes must be put together and analyzed if they are capable to offer a base to define the usual mental states of DAI (Belief, Desires and Intentions) and other not well applied yet, although known from Psychology, Philosophy and Economics as relevant to explain human behavior as Expectations, Hopes and Necessities. When we diversify applications and work within Social and Human Sciences or more complexity interactions of economic agents we are forced to adopt such other mental states [9, 20, 21].

To make references to these attributes we organize them in three groups: the first called "nucleus" contains the mental state's external content and a set of criterions for unsatisfaction, uncertainty, urgency, intensity, insistence and intensity; the second called "laws" contains a set of possible causal relationships among mental states and the last called "controls" contains a set of controls to trigger, choose, suspend and cancel mental states.

2.1 Nucleus

These attributes define proper characteristics of MS that is, the MS with the same set of attributes are classified as the same type as shown in the paragraphs below, but every particular MS is distinguished, at least, by its external content. Normally an MS is distinguished from other MS of same type by the attributes of the nucleus.

- External Content (Ex. Content). The mental states have external significance; that is to say, they are linked to the world in terms of what they are about. This external content is represented by a logical proposition (e.g. in a multimodal logic).

- Unsatisfaction is related to the stimulation of actions or changes of mental states. We consider that this component is the central motivator of actions and the producer of other mental states. A function s_X (s_X: Á Á ... Á§Á) is defined as a „degree of unsatisfaction". For instance, a MS X is satisfied if $s_X(r) \bullet 0$ (r°Á Á ...Á) So, we need some criteria to decide if a MS is satisfied or not.

- Uncertainty is a measure of agent confidence regarding the situation that corresponds to the mental state. A function c_X (c_X: Á Á ... Á§Á) is defined as a „degree of uncertainty".

- Urgency is a measure of how much time remains to the point the corresponding MS X must be satisfied. A function u_X (u_X: Á Á ... Á§Á) is defined as the „degree of urgency".

- Intensity is related to the agent's pledge and energy dedicated to an MS. For example, if an agent is trying to satisfy an MS, the intensity of this MS is connected to how actively or vigorously this satisfaction is pressed and adopted. A function v_X (v_X: Á Á ... Á§Á) is defined as a „degree of intensity".

- Importance is related to a valuation in terms of benefits and costs the agent has of a corresponding mental state situation. A function m_X (m_X: Á Á ... Á§Á) is defined as a „degree of importance".

- Insistence is related to how much dificult it is for an agent to abandon a MS. For instance, if the agent strongly insists on some goal, this goal will not be abandoned easily . A function n_X (n_X: Á Á ... Á§Á) is defined as a „degree of insistence".

2.2 Laws

The laws define how the mental states are combined to cause other mental states or agent's actions. For example, under certain conditions, a Belief and a Desire cause another Desire: an agent A's Desire to learn mathematics and the Belief that agent B knows mathematics and that B can teache A, cause A's Desire to learn mathematics from B. Another law is: a Desire and a Belief can cause an Intention: A's Desire to learn mathematics from B and A's

Belief that in order to learn from agent B there is a strategy. That is, A must know how to learn from another agent. Thus, there is an A's Intention to learn mathematics from B.

A collection of laws relating Belief, Desire, Intention and Expectation is presented in figure 1 according to Corrêa [9]. Another demonstration of such mental states dynamics applied in a Tutor/Learner session is shown in Moussale et al. [15].

2.3 Controls

These attributes define how and when an MS causes another, can be suspended, canceled or stayed active. An MS is active when it produces or influences another MS, causes an agent action or contributes indirectly to an agent action. An MS is suspended when it is temporarily inactive. A MS is cancelled when it is permanently inactive.

Let X be a mental state. The possible laws will be triggered if $s_X(r) > 0$ (r°Á_Á_...Á) and at least one of the conditions (C1 to C7) below occurs:

C1) $u_X(r) > 0$ and $v_X(r) > 0$ and $m_X(r) > 0$;

C2) $u_X(r) > 0$ and $m_X(r) > 0$; C3) $u_X(r) > 0$ and $v_X(r) > 0$;

C4) $v_X(r) > 0$ and $m_X(r) > 0$; C5) $m_X(r) > 0$; C6) $v_X(r) > 0$; C7) $u_X(r) > 0$

C8) Canceling or suspending a MS:

If X is an active MS and $g_X(r) < 0$ then cancel X

If X and Y are active and conflicting MS then

suspend X if $g_X(r) < g_Y(r)$ or suspend Y if $g_Y(r) < g_X(r)$;

else if $g_X(r) = g_Y(r)$ choose X or Y randomly.

Where $g_X(r)$ and $g_Y(r)$ are the interrupting functions, defined in terms of urgency, intensity and importance of the corresponding MS.

We will not present here a specific definition of conflicting mental states. We consider that two MS of the same type are conflicting when they cannot coexist as active mental states at the same time in the agent mind.

C9) Activation of a suspended mental state.

If X is a suspended MS and there is no other active MS conflicting with X then X is activated, unless if there is an active MS Y conflicting with X then X is activated if $n_X(r) > 0$, X is maintained suspended if $g_X(r) > 0$ or X is canceled.

C10) Finding a strategy to satisfy an MS.

If there is no strategy or means to satisfy a MS X then if „it is possible to find a strategy or means to satisfy X" then „find and adopt this strategy or means“ else if $g_X(r) > 0$ X is suspended otherwise X is canceled.

C11) Find alternatives when an adopted strategy doesn't work anymore.

If K is an adopted strategy to satisfy an MS X and, and at some moment, it is not possible to satisfy X through K then find another strategy if possible and $n_X(r) > 0$; suspend X if $g_X(r) > 0$ or cancels X.

The relationships among these attributes and the mental states, are shown in the table of figure 1.

	B	D	I	E	H	N	P
Ex. Content	x	x	x	x	x	x	x
Unsatisfaction		x	x	x	x	x	
Uncertainty	x		x	x	x		x
Urgency		x	x	x		x	
Importance	x	x	x	x	x	x	
Intensity		x	x	x		x	x
Insistence	x	x	x	x	x	x	
L1) B<=	x			x			x
L2) B <= B+	x			x			x
L3) D <=		x			x	x	
L4) D <= B+		x		x			
L5) I <=			x				
L6) I <=B+		x	x				
L7) E <=	x		x				
L8) H <=	x	x					
L9) P <=	x	x		x			
L10) A <=			x				x
Control C1		x	x	x		x	
Control C2		x	x	x		x	
Control C3		x	x	x		x	
Control C4		x	x	x		x	
Control C5		x	x	x	x	x	
Control C6		x	x	x		x	
Control C7		x	x	x		x	
Control C8	x		x	x			
Control C9	x		x	x			
Control C10			x				
Control C11			x				

Figure 1: Table of relationships among mental states Beliefs (B), Desires (D), Intentions (I), Expectations (E), Hopes (H), Necessities (N), Perceptions (P) and their possible attributes.

In this MSF (as shown in figure 1), the controls C1 to C7 act as trigger mechanisms, so that an MS X alone (or combined with another) will produce other mental states through its possible laws assigned in the table. The controls C8 and C9 act as filters to activate or suspend a MS. C10 and C11 govern MS directly depending on agents' actions. These controls (C8 to C11) are a meta-strategy that corresponds to the so-called "commitment" of a mental state [4, 5, and 19].

In figure 1 the types of mental states are indicated in the columns of the table and their corresponding attributes (marked with a x) are indicated in the rows. The group of rows labeled from L1 to L9 corresponds to the laws relating the mental states among them, that is, let W some MS indicated in a column, so if in this column a row labeled by "L1) B <=" is marked then a Belief can be caused by MS W (B <= W), the row labeled "L2) B <= B +" means that a Belief can be caused by another belief and W (B <= B + W) and similarly, "L3) D <=" means D <= W, "L4) D <= B+" means D <= D + W etc... The row labeled "L10) A <=" means that the mental state W can produce directly some action A of the agent (A <= W). The correspondence among the mental states and the controls C1, C2...is also shown in the table. For example, the Beliefs (B) are defined by assigning them the attributes Ex. Content, Uncertainty, Insistence, Importance, L1, L2, L7, L8, L9, C8, and C9.

In the paper we are considering only individual mental states, however the notions of social mental states are also necessary for more precise and complete modeling of agents' interactions in a society [6, 14, 22]. We are also developing these social notions according to our theory, but we don't have space to present these advances here.

3 Agent's Programming

This methodology to build agents, to be useful, should not only be a guide to classify and make comparison among mental states agents' architectures since it provides a space to define them, but it should also be a guide to program agents. The approach of mental states as defined from the basic attributes shown in table of figure 1 is very adequate to map mental states to Classes and Objects of Object Oriented Programming (OOP) as shown below. Our experiments with agent programming according to the theory of the paper are currently in progress. In these testbeds the types of mental states are programmed as Classes in an OOP language as, for example, the Java programming language [1] and, a particular mental state is a particular object of the corresponding class. So, to program an agent we need to define their mental states as objects as sketched below for the mental state Desire:

```
Desire desire1 = new Desire (Extenal_Content_D1)
Desire desire2 = new Desire (Extenal_Content_D2);
                ...
Belief  belief1 = new Belief (External_Content_B1 );
Belief  belief2 = new Belief (External_Content_B2 );
```

That is, desire1, desire2 are objects of class Desire and belief1, belief2 are objects of class Belief. The External_Content (D1, D2, B1, and B2) are Propositions. So, there are classes defined also to support the representation and logic operations of these Propositions.

A programming sketch of the mental states desires from the framework of figure 1 can be:

```
import Proposition;
public class Desire extends Thread {
    public Situation External_Content;
    public Desire(Proposition External_Content) {
        this.External_Content = External_Content;
        }
    public void run() { // this object desire runs concurrently with other mental states
                while (unsatisfaction() > 0 & insistence() > 0)
        {// controls to trigger corresponding laws
         if ( urgency() > 0 & importance() > 0 0  & intensity() > 0)
         // trigger a law to produce another mental state
          triggerDesiresLaw(External_Content);
        }
    }
    public float unsatisfaction() { //this is a method for a desires' unsatisfaction}
    public float urgency(){// this is a method for a desires' urgency}
    public float insistence() { // this is a method for a desires' insistence }
    public float importance() { // this is a method for a desires' importance}
    public float intensity() { // this is a method for a desires' intensity}
    public void triggerDesiresLaw(Situation External_Content)
     { // this is a  method to trigger the desire corresponding laws to produce other
                        mental states: another desire or an intention}
}// End of Desire
```

Thus, the beliefs (or the knowledge) are modeled as objects according to the OOP. So, it is also included in some of these objects, methods for planning, update beliefs and reason. We note, for example, that differently from some approaches on intentions [3, 16] these ones according to the MSF are complex structures that besides plans also contain controls to decide about their maintenance, suspension or canceling, and decide in case of conflicts, or search for alternative plans in case of failure.

We have no space and it is neither our intention to present here a complete nor a consistent programming of the mental state framework. We only wish to show in a very sketched way how the structure presented in figure 1 can be directly described in an object programming language as, for example, Java. It is important to note that a Desire object is implemented as a „Java thread', that is, it runs concurrently with other mental states which, on the other hand, is a „Java thread". The other mental states are programmed similarly. For example, it is possible simultaneous non-conflicting different intentions be active. So, the concurrent computing is a basis for this programming framework.

4 Relationships with other Architectures

The SEM architecture [9], is a generic agent's architecture whose ultimate feature is that the agent's specification is made in terms of Desires, Beliefs, Intentions and Expectations. The properties of these mental states are the same presented in figure 1 and an implementation of this architecture was made in Prolog [9]. The kernel of this architecture (called global agent) is made of four autonomous local agents each one corresponding to these four mental states. The behavior of the global agent emerges as a collective behavior of those local agents.

This architecture was the experimental basis from which we organized and synthetized the table of figure 1. The methodology presented in the paper is also based on this experience. This framework expands the ideas presented in [8] and [9] deeming that architecture is an intersection between the space of mental states and the space of architectures.

The IRMA [3] architecture describes how an agent selects its course of actions based on explicit representations of its perception, beliefs, desires and intentions. This architecture incorporates a number of modules as means-end reasoner, an opportunity analyzer, a filtering process, a deliberation procedure and an intention structure which is basically a time-ordered set of partial tree-structured plans. The agent's beliefs are updated by its perception.

All of these features of IRMA architecture are embedded in the Mental Sttes Framework (MSF) considering only the MS belief, desire, intention and perception. Therefore, the means-end, opportunity analyzer and filtering process module activities have correspondence with the controls in MSF.

The PRS [13] architecture is based on the assumption that the beliefs about the current state of world is explicitly represented, information for means-end reasoning i.e. about means of reaching certain world states and about options available to the agent for proceeding towards the achievement of its goals, is represented by plans. The intentions of an agent are represented implicitly by an agent's run time stack. The PRS also presents means to build plans. In the MSF the plans to accomplish goals and the procedures to build plans are embedded in the agent's system beliefs; although, none of these procedures had been presented, these features may be embedded as an object in agent's architecture implementation from the MSF. As it happens in the PRS, the MSF presents also controls to suspend, start up tasks in accordance with urgency degrees.

The INTERRAP [16] architecture like Ferguson's TouringMachines [7], is a layered architecture. This approach extends the work on BDI architectures in that it is based on the

assumption that different aspects of agent behavior, such as reactivity, deliberation and interaction, correspond to and make use of different qualities of knowledge and different mechanisms of reasoning about this knowledge. It is defined in terms of three interacting control and knowledge layers:

- A behavior-based layer that implements the reactivity capability of the agent manipulating patterns of behavior.
- A local planning layer that contains a planner and a knowledge base that contains a plan library.
- A cooperative planning layer (the highest level) that is able to generate joint plans, those satisfy the goals of a group of agents.

In the MSF-based-architectures the features obtained through the layered structure as in INTERRAP correspond to the mental states attributes. The MSF presents a general view in respect of the layered model, deeming that each type of mental state (a class in a OOP implementation) corresponds to a kind of layer.

Hence, for instance, the type Perception corresponds to a layer which include the reactivity besides beliefs production, since MSF defines that a Perception can produce either an action or a Belief; the type Belief corresponds to the planning layer (and also reasoning). The Intention corresponds to the plan execution control in order to accomplish a certain goal.

Even though some cooperative behavior can be obtained through mutual desire adoption [11], we still haven't gone further regarding joint mental states to agents interactions coordination in a team, which corresponds the INTERRAP third layer.

5 Conclusion

In the paper it is presented the Mental States Framework (MSF) in which agents' architectures are defined from a mental states space. One of the most important aspects of this framework is that it shows how mental states dynamically interact to produce other mental states (motivation) and agents' actions in the world (behavior).
Comparisons with some BDI architectures (some of which also have reactive capacity) have shown how they could be inserted in this framework context. However, it is still not possible to assure that agents' architectures, in general, can be inserted within the MSF, that is an open question, but the direction of this research points towards a methodology in which the construction of agents having complex behavior can be obtained from their mental states specification according to MSF. And, in the paper a further step towards the agents' engineering has shown that the implementation of an architecture defined in the MSF can be made by applying the Object Oriented Programming technology.

References

[1] K. Arnold and J. Gosling. The Java Programming Language, The Java Series, Sun Microsystems, 1996.

[2] M. Bratman. What is Intentions? In P. R. Cohen, J.L. Morgan and M. Pollack (eds.), Intentions in Communication, The MIT Press, Cambridge, MA, 1990.

[3] M. Bratman, D. Israel and M. Pollack. Towards an architecture for resource-bounded agents. Technical Report CSL_87-104, SRI, Stanford University, August, 1987.

[4] C. Castelfranchi. Commitments: From Individual to Groups and Organizations, Proceedings of The First International Conference on Multi-Agent Systems (ICMAS-95), 1995.

[5] P. Cohen and H. Levesque. Intention is Choice with Commitment, Artificial Intelligence, 42:213-261, 1990.

[6] P. Cohen, H. Levesque and I. Smith. On Team Formation. In Contemporary Action Theory, G. Hintikka and R. Tuomela (editors), Kluvier Academic Pub, 1997.

[7] I. A. Ferguson. TouringMachines: An Architecture for Dynamic, Rational, Mobile Agents, PhD Thesis, Computer Laboratory, University of Cambridge, UK, 1992.

[8] M. Corrêa and H. Coelho. Around the Architectural Agent Approach to Model Conversations, Proceedings of Modeling Autonomous Agents in a Multi-Agent World (MAAMAW-93), Nêuchatel, Switzerland, Springer-Verlag, 1993.

[9] M. Corrêa. The Architecture of Dialogues of Distributed Cognitive Agents, Ph. D. Thesis (in Portuguese), Federal University of Rio de Janeiro, January, 1994.

[10] M. Corrêa and H. Coelho. A Framework for Mental States and Agent Architectures. In Multi-Agents Theory and Architectures Conference, MASTA97, Coimbra, 1997.

[11] M. Corrêa, R. Viccari and H. Coelho. Dynamics in Transition Mental Activity, in Proceedings of International Conference on Multi-Agent Systems (ICMAS98), 1998.

[12] E. Frankel and M. Corrêa. A Cognitive approach to Body-Psychotherapy. The Journal of Biosynthesis, Vol 26 Nº 1, Abbotisbury Publications, April, 1995.

[13] M. Georgeff and F. Ingrand. Decision-making in embedded reasoning systems. In Proceeding of the 6th international Joint Conference on Artificial Intelligence, 1989.

[14] N. Jennings. Controlling cooperative problem solving in industrial multi-agents systems using joint intentions. Artificial Intelligence 75, 1995.

[15] N. Moussale, R. Viccari and M. Corrêa. Tutor-Student Interaction Modeling in an Agent Architecture Based on Mental States, Brazilian Symposium on AI (SBIA96), Springer, 1996.

[16] J. P. Müller. The Design of Intelligent Agents. A Layered Approach. Lecture Notes in Artificial Intelligence, Vol 1177, Springer, 1996.

[17], I. Pörn. Action Theory and Social Science. Some Formal Models, Reidel Publishing Company, Dordecht-Holland, 1974.

[18] M. Rokeach. Beliefs, Attitudes and Values, Jossey-Bass Inc, Pub., 1970.

[19] Y. Shoham. Agent-Oriented Programming, Artificial Intelligence, 60, 1993.

[20] A. Sloman. Motives Mechanisms and Emotions, In M.A.Boden (ed.) The Philosophy of Artificial Intelligence, Oxford Readings in Philosophy Series, Oxford University Press, 1990.

[21] A. Sloman. What Sort of Architecture is Required for a Human-like Agent? Invited talk at Cognitive Modeling Workshop, AAAI96, Portland Oregon, August, 1996.

[22] R. Tuomela. The Importance of Us. Stanford University Press, 1995.

[23] B. Webber, N. Badler, B. Eugenio, C. Geib, L. Levison, M. Moore. Instructions, Intentions and Expectations, University of Pennsylvania, Computer and Information Department, June, 1993.

[24] E. Werner. A Unified View of Information, Intention and Ability, In Decentralized Artificial Intelligence, Y. Demazeau and J. P. Mueller (eds.), Elsevier Science Pub. 1991.

[25] M. Wooldridge. Issues in Agent-Based Software Engineering. In P.Kandzia and M. Klush (editors), Cooperative Information Agents, Lecture Notes in Artificial Intelligence, Vol 1201, Springer, 1997.

Acknowledgment

This work was supported, in part, by the Fundação para a Ciência e Tecnologia under the PRAXIS XXI programme (SARA 2/2.1/TIT/1662/950.

The Component Based Model of the Control Knowledge in an Intelligent Tutoring Shell

Ljubomir Jerinic[1], Vladan Devedzic[2] and Danijela Radovic[3]

[1] Institute of Mathematics, University of Novi Sad,
Trg Dositeja Obradovica 4, 21000 Novi Sad, Yugoslavia
jerinic@uns.ns.ac.yu
[2] FON - School of Business Administration, University of Belgrade,
Jove Ilica 154, 11000 Belgrade, Yugoslavia
devedzic@galeb.etf.bg.ac.yu
[3] University of Kragujevac, Technical Faculty Cacak,
Svetog Save 65, 32000 Cacak, Yugoslavia
danijela@tfc.tfc.kg.ac.yu

Abstract. Issues of pragmatics and usability motivate our starting point and perspective on developing Intelligent Tutoring Systems tool. The advancement of AI methods and techniques makes understanding of ITSs more difficult, so that the teachers are less and less prepared to accept these systems. As a result, the gap between the researchers in the field of ITSs and the educational community is constantly widening. Also, the present ITSs need quite big development environments, huge computing resources and, in consequence, are expensive and hardly portable to personal computers. The paper describes an application of the component based software design methodology for realizing and arranging designing the knowledge bases and manipulating their contents for intelligent tutoring. The component based model of ITSs knowledge bases has been developed, and the concept of the control i.e. action knowledge is presented.

1 Introduction

Intelligent Tutoring Systems (ITSs) [6] are used for computer-based instruction of students in a particular domain. They have the possibilities of presenting the appropriate contents to the user (student), monitoring his/her reactions and the learning process, generating problems and tasks for the user in order to check his/her knowledge, etc. Relying on a particular model of each student, the control mechanism of an ITS can suit the tutoring process to each individual user.

Traditional ITSs are concentrated on the domain knowledge they are supposed to present and teach; hence their control mechanisms are often domain-dependent. More recent ITSs pay more attention to generic problems and concepts of the tutoring process, trying to separate architectural, methodological, and control issues from the domain knowledge as much as possible. In other words, there are interactive and

F. Moreira de Oliveira (Ed.): SBIA'98, LNAI 1515, pp. 40-49, 1998

integrated development tools for building ITSs, i.e. for helping the developer plug-in some domain knowledge and test the prototype system, and then gradually and incrementally develop the final system. Such integrated tools are often referred to as *shells* (e.g., ITS shells, or ILE shells, or IES shells, etc.), which usually require a knowledge engineer in order to be fully used, or *authoring tools*, which can be also used by human instructors who do not necessarily have knowledge engineering experience. Still more recent ITSs are developed for collaborative learning, and often use Internet/WWW/agents technology in order to provide comfortable, user-oriented, distributed learning and teaching facilities [1, 3, 5].

This paper describes how the recently developed component-based model of ITSs called **GET-BITS** [2] treats control knowledge. The model covers all essential aspects of control knowledge denoted in AI so far, such as generic tasks, control procedures, meta reasoning, inference paradigms, etc. All these aspects are covered in an object-oriented class hierarchy, reflecting the principles, structure and organization of most important elements of control knowledge, and showing how these elements are related to other types of knowledge. The model is based on a number of design patterns and class libraries developed in order to support building of intelligent systems. The next extension presented in this paper, is involving that the control knowledge is extended by adding the psychological type of the responds to some controls or questions of the user, and appropriates screen models. The control knowledge managed the domain knowledge according the student model and the teaching strategy. This component is realized by generalized and abstracted If-Then rules that control the domain knowledge represented with the knowledge network of semantically connected frames. That knowledge network represents the content to be taught in terms of lessons, concepts, rules, tasks, questions, controls, and examples and their interrelationships. The control knowledge according to the user control and pre-tested psychologically type of the users managed that network and the user-computer interaction.

2 Types of Knowledge in Intelligent Tutoring Systems

An important area in the modern intelligent software technology is the integration of the multiple knowledge sources, knowledge types, reasoning paradigms, inference models, etc. Intelligent tutoring systems may outwardly appear to be monolithic systems, but for the purposes of conceptualization and design, it is often easier to think about them as consisting of several interdependent components. In a general case, at the architectural level an ITSs is a complex system with its knowledge and problem-solving activities distributed to several modules. Previous research by Woolf [6] has identified four major components of ITSs: the student module, the pedagogical module, the domain module and the presentation module.

In our approach in designing the ITSs, we separated the domain knowledge in some parts, i.e. we have identified the fifth component, the **expert** model (Woolf includes this component as part of the domain knowledge) but we feel that it is a separate entity. The expert model is similar to the domain knowledge in that it must contain the

information being taught to the learner. However, it is more than just a representation of the data; it is a model of how someone skilled in a particular domain represents the knowledge. Most commonly, this takes the form of some kind of expert systems, i.e. one that is capable of solving problems in the domain. By using an expert model, the tutor can compare the learner's solution to the expert's solution, pinpointing the places where the learner had difficulties.

This approach leads that we must have some **explanation** module [4]. The primary purpose of the part of the knowledge base that we refer to as the explanatory knowledge is to define the contents of *explanations* and *justifications* of the ITS learning process, as well as the way they are generated. Explanatory knowledge is related to both domain knowledge and control knowledge and is often treated as a part of the other two components of knowledge bases. However, in **GET-BITS** it is treated as a distinct knowledge component, in which the knowledge about the learning process explanation, knowledge elements explanation, control strategies explanation, and other types of intelligent assistance is represented explicitly and is treated in its own right.

So, the above considerations leads that we think of an ITSs knowledge base as a logical entity composed of three related parts:

- the domain knowledge (i.e. a model of experts knowledge denoted by D_knowledge),
- the control knowledge (i.e., a model of the teaching process and a model how the system responds, C_knowledge), and,
- the third part that we call explanatory knowledge, i.e. E_knowledge.

3 Domain and Explanatory Knowledge

Domain knowledge is represented using one or more knowledge representation techniques. In the most general case, domain knowledge is a structured record of many interrelated knowledge elements. These elements describe relevant domain models and heuristics, and can vary a lot in their nature, complexity, and representation. They can be everything from simple data to instantiated knowledge primitives such as frames, rules, logical expressions, procedures, etc., and even more complex knowledge elements represented using either simple aggregations of the knowledge primitives or conceptually different techniques based on the knowledge primitives and other (but simpler) knowledge elements. Semantic nets are the best known kind of such complex knowledge elements, and are frequently used for representing deep knowledge about the problem domain.

A D_knowledge object lets other objects (communicating with it) use its public interface functions to set and get its name, get its size, access its collections of knowledge elements, compile it, etc. It should be stressed that the model strictly applies encapsulation, as a general and extremely important principle of object-oriented design: most attributes (fields) of classes and objects used in this design cannot be accessed directly. Client objects must use the server object's public interface

functions. For example, in order to read a knowledge element from a certain collection of knowledge elements belonging to the domain knowledge part of an ITSs knowledge base, the corresponding D_knowledge object's Get_K_El_Collection interface function is called. It uses the appropriate private K_El_Collection_Ptr pointer and calls interface functions of the corresponding K_elements object to retrieve the desired knowledge element.

K_element (Figure 1) is an abstract base class with a number of subclasses inheriting its properties. In the Booch notation, a solid line denotes the inheritance relationship among two classes with an arrow pointing to the base class. Abstract base classes cannot have instantiations, so only the objects of the Rule, Frame, Slot, If-clause, Signal, and other subclasses can exist in the knowledge base. Again, we assume the classes designating the knowledge primitives mentioned are derived directly from the K_element class, skipping more complex classes of knowledge elements. The model is open for extensions by less common (or even new) knowledge element types. The K_element class defines some useful common properties that are inherited by all types of knowledge elements, and including another knowledge element type in the hierarchy requires only type-specific properties to be defined. Second, all the elements of a knowledge base are treated the same way, regardless of what their type is and how common (standard) or specific they are. This means that all the common operations on elements of the knowledge base (like updating, searching, deleting, etc.) use the interface of the K_element class, and element-specific operations are provided through polymorphism in the corresponding subclasses.

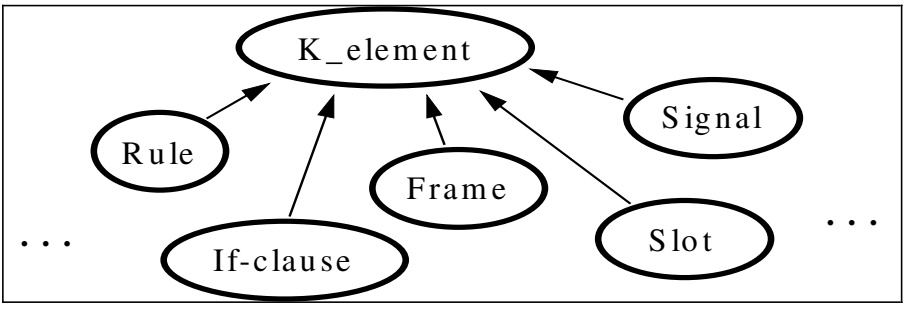


Fig. 1. The class hierarchy of the knowledge elements

Another interesting design detail in Figure 1 is the fact that certain types (classes) of knowledge elements, denoting parts of more complex elements, are at the same level of hierarchy as the corresponding aggregate elements. For example, Figure 1 suggests that If-clauses can be treated as "stand alone" knowledge elements, as well as rules. Such a design allows a particular If-clause object to be shared by several Rule objects, which is more flexible and more efficient, and can also simplify the rule compilation process.

The primary purpose of the part of the knowledge base that we refer to as the explanatory knowledge is to define the contents of explanations and justifications of the ITSs learning process, as well as the way they are generated. Explanatory

knowledge is related to both domain and control knowledge's and is often treated as a part of the other two components of knowledge bases. However, in the **GET-BITS** model it is treated as a distinct knowledge component, in which the knowledge about the learning process explanation, knowledge elements explanation, control strategies explanation, and other types of intelligent assistance is represented explicitly and is treated in its own right.

Among the knowledge elements that this component can include are:

- canned text associated with rules and other knowledge elements in the other two components,
- templates that are filled by the explanation module when required (in order to generate the full explanation text in a given situation),
- presentation functions needed for explanation of certain knowledge elements (e.g., some knowledge elements are best explained by using tables, others require the use of graphics, etc.), and
- user models necessary for generating explanations in accordance to the user's knowledge level, criteria used by the explanation module when deciding about what knowledge elements to include in the explanation and what to skip, as well as what level of details must feature the explanation.

It must be stressed that apart from the explanatory knowledge, the explanation module uses extensively the knowledge from the other two parts of the knowledge base as well when generating explanations. Therefore, the explanatory knowledge may also contain explicit descriptions of explanation control strategies. These can be specified in the form of control rules like, for example: "If the explanation type is WHY, then show:

1. the current goal, and
2. the current domain rule instantiation, and
3. the meta-rule that was last applied".

4 The Model of Control Knowledge

The reason why we explicitly denoted the new kind of the knowledge in the ITSs denoted by **control** knowledge is as follows. The control knowledge contains information, methods and procedures that are relatively independent; we could identify them and implemented them as the knowledge, i.e. the separated software modules or procedures. Also, the control knowledge could be treated as the part of the pedagogical module, i.e. the model (or models) of the teaching process. Further, the control knowledge is the part of the communication module, i.e. the model how the system responds on some controls. Beside that, it could be treated as the part of the student model, i.e. the particular or general responds of the some students actions, and at the end the part of the expert module, i.e. the connections and inter-relationships between the parts of the knowledge to be taught.

The contents of the ITSs control knowledge are abstract, explicit, and more or less domain-independent descriptions of the form how to learn some facts during the ITSs operation. In **GET-BITS** we used two approaches in representing control knowledge: generic tasks, and the ITSs meta-knowledge, presented in the form of meta-rules. Generic tasks are abstract and typical knowledge and control structures reflecting the steps and activities performed when learning process is represented by learning-by-doing. They make it possible to represent the problem of teaching strategies explicitly in the knowledge base, using appropriate control knowledge elements to express the knowledge of how the learning by problem solving task is performed. A specific problem type, structure, and problem solving strategy feature each generic task. Elementary (atomic) generic tasks, like hierarchical classification or hierarchical design by plan selection and refinement, can be used as building blocks for construction of knowledge-based systems. They can be also viewed as parts of other, more complexes, but still typical expert systems tasks (modified for ITSs) called complex generic tasks.

Meta-rules are similar to domain rules, but they contain strategically knowledge about the teaching/learning process, rather than domain heuristics. They enable the ITSs to examine its own domain knowledge during the operation and make decisions that actually direct further use of domain knowledge. As well as generic tasks, they also clearly separate control knowledge from domain knowledge and explicitly specify the system behavior at the task level. For example, if a certain problem can be solved either by applying a heuristic search or by decomposing it into sub-problems, meta-rules can specify what strategy to use given some facts about the problem or the nature of desired solution.

In the context of control knowledge we can also speak of knowledge elements and their collections, hierarchies, complexity, etc., only this time the contents and the meaning of the knowledge elements are different from that of the elements in the domain knowledge part of the knowledge base. Furthermore, some levels of complexity of control knowledge elements parallel those of domain knowledge elements. The same can be shown for explanatory knowledge. Therefore the most important abstraction used in the **GET-BITS** model is that of the abstract and general knowledge element as a universal concept.

To introduce the semantic of the control knowledge we defined the operator $\mathbf{K_n}$. The formula (i.e. the proposition) **T** interprets that operator $\mathbf{K_n}$ mean that the student denoted by the **S** knows the part of the lesson (the topic). In the **GET-BITS** model the top elements of knowledge is the model of **Lesson**, and this is basic class needed for modeling the learning process. Any lesson is consisted of one or more issues, modeled by the class **Topic**. We assume that the student must learn these issues, during mastering that lesson. The basic attributes of the lesson are:

- the name denoted by **Name**,
- the current topic denoted by CurrentTopic, the issue or the problem introduced, defined and/or explained in the present moment,
- the task or question that is currently solved (part of the **TQ** class), the level of prerequisites for the student (StudentLevel), and so on [4].

The axioms of the logic for that operator $\mathbf{K_n(S, T)}$ are inductively defined as instances of the following schemata:

- $\mathbf{M_1}$**. T** is the fact, i.e. an axiom or the truth fact, or at least an axiom of ordinary prepositional logic.
- $\mathbf{M_2}$**.** $\mathbf{K_n(S, T) => T}$
- $\mathbf{M_3}$**.** $\mathbf{K_n(S, T) => K_n(S, K_n(S, T))}$
- $\mathbf{M_4}$**.** $\mathbf{K_n(S, T=>R) => K_n(S, T) => K_n(S, R)}$
- $\mathbf{M_5}$**.** If **T** is an axiom than $\mathbf{K_n(S, T)}$ is an axiom

The closure of the above axioms under the inference rule modus ponens defines the theorems of the system. This system is very similar to those studied in modal logic. These axioms formalize the principles for control knowledge that we have discussed. $\mathbf{M_2}$ means that the fact **T** that is known by the student **S** is true. I.e. more generally, anything that is known by the student **S** is true. $\mathbf{M_2}$ means that, if somebody knows something than **S** knows that he knows it.

As the domain knowledge is represented as $\mathtt{L}(\mathbf{F, K, F_0, E})$ where **F** is the world of fact that student learn, $\mathbf{F_0}$ the initial fact state from which the lesson begin, and the **E** is a set of final state/fact and the **K** is the function for control after student **S** represented with the student model **S** learn or not learn the current fact $\mathbf{f_i}$ **F**. The control function is defined by $\mathbf{K: F \times K_n \times S \rightarrow F}$. The system made the plan of the new goal in learning according the value of $\mathbf{K_n(S, T)}$ for the current fact that student learn and the student model **S**, i.e. made the new control knowledge for that moment.

5 Example of the Knowledge Primitives for the Control

The domain, i.e., the knowledge that student must learn, is essentially a tree defined by $\mathtt{L}(\mathbf{F, K, F_0, E})$, where each node is a piece of domain knowledge and any parent-child link in the tree implies that in order for the student to be able to understand the knowledge of the parent node, he must also understand the knowledge of the child node. The representation of the knowledge domain is object-oriented. The main object classes are the lessons and the way of their presentation. Units of knowledge that can be taught, and/or repeated, and/or summarized, etc represent lessons. They are categorized according to knowledge type, for example: text, picture, simulation, more examples, and so on. The concepts have pointers, including various types of prerequisite, part-of and related-misconception links to other concepts, forming the lesson network. They have pedagogical information such as summary, motivation, examples, tasks, etc., which point to presentation. The presentation of the question or task, in the **GET-BITS** represent some task, and/or expository and/or inquisitor interactions with the student. They are composed of a task, such as a multiple-choice question, or problem solving exercise, and an environment for doing the task, such as a picture or simulation of some system. Presentation also contains the possibilities for responding to the pupil, such as hints, congratulations, elaboration's of the answer, etc. Those knowledge representations is concerning some particular and concrete type of

knowledge elements that are necessary for the realization of the Expert module in any ITS shell. Any lesson is consisted of one or more issues, modeled by the class Topic. The basic attributes of the lesson are: the name (Name), current topic (CurrentTopic, the issue or the problem introduced, defined and/or explained at the present moment), the task that is currently solved, the level of prerequisites for the student (StudentLevel), and so on.

The issues that student must learn (class Topic) are realized with separate type of knowledge elements in the **GET-BITS** system. Any topic could be introduced with text information (Information), graphically (the pictures and/or the diagram - Image), and/or with the simulation of some event (Simulation). Also, the additional or further explanation (Explanation) for that theme, or some suggestions (Hints) could be defined. The class Topic in **GET-BITS** is made for specifying and defining the issues or notions needed for lesson creation.

Abstract class TQ served for description of the common elements for the two comparable and connected classes, one for the definition of tasks or assignments (class Task) and the other class for the realization of questions or problems (class Question). That class has the fields WhyPtr, HowPtr, WhatPtr, etc. With that fields the lesson creator made their explanations for the task and/or question. The instances of that class are given to the student during the learning process. The completed definition for the classes *Lesson, TQ, Topic, Explanation, etc.* are given in [2].

The below class design shows an example of how classes are designed in the **GET-BITS** model. The class shown is the *Coaching* class. Although variants of the coaching strategy have been reported in the literature, only a single variant is included in the **GET-BITS** model so far, hence no class is derived from *Coaching*. A control object of the pedagogical module is supposed to invoke interface functions (method procedures) of an object of the *Coaching* class in order to initiate the teaching process (*Set target goal, Set known concepts*), call a teaching planner to generate sequences of teaching operators to be executed (*Generate path*), select a sequence to execute (*Select path*), execute it (*Traverse path*), compare results of the student's problem solving with those of the domain expert module (*Compare results*), update the student model accordingly (*Update student model*), etc.

Name:	*Coaching*
Visibility:	*Exported*; visible outside the enclosing class category
Cardinality:	*1*; there can be only one object as the class instance
Base class:	*Teaching strategy*; in general, a list of base classes
Derived classes:	-
Interface	
Operations:	*Set target goal, Set known concepts, Generate path, Select path, Traverse path, Compare results, Update student model,...*
Implementation	
Uses:	*Dependency graph, Teaching operator,...;* a list of classes used by this one
Fields:	*Current topic, Current goal, Target goal,*

	StudentModel_Ptr, TopicCollection_Ptr [],...
Persistency:	*Static*; disk files

The class *Control_Knowledge* is designed as follows:

Name:	*Control_Knowledge*
Visibility:	*Exported*; i.e., visible outside the enclosing class category
Cardinality:	*n*
Base classes:	- ; in general, list of class names
Derived classes:	Control_*Rule, Control_Frame, Control_If-clause, Control_Then-clause, Control_Slot,...*
Generic parameters:	-; although a base classes, K_element is not a generic
Public Interface	
Operations:	*SetName, GetName, GetSize, Create_Control_Knowledge, Delete_Control_Knowledge, Get_Control_Knowledge, Update_Control_Knowledge,...*
Implementation	
Uses:	*K_elements, Transaction,...*
Fields:	*Name, Collection_Ptr[],...*
Demons:	*TQ_Engine, Show_Questions_Ptr[], Get_Answers_Ptr[], Consult_Net, Question_Generator, Give_Congratulate*, *Calculate_the_Error,* Elaborate_Answers_Ptr[],
Persistence:	*Static* ; disk files

The control knowledge is realized by parameterizing the controls with the combination of If-Than-Control rules. Represents the TQ_Engine control knowledge for elaborating an answer for some task or question, according the theoretical results briefly given in the previous section. The idea of having a component-based ITS implies that its components will be developed independently, in (possibly) different languages, on (possibly) different machines, and using (most probably) different software development tools. Yet we have to put them all together, to make them communicate not only among themselves, but also with possibly quite different applications (like traditional databases, for example). If this is to be provided efficiently, some standards must be conformed to. Fortunately, such standards exist already. They specify standard interfaces for transparent object (component) communication; both on a single machine and in heterogeneous distributed environments.

6 Conclusion

Software components for ITS design, in the software engineering sense of the term, have started to attract increasing attention among the researchers in the field. Component-based design of ITSs can bring a number of benefits do the developers, including enhanced reusability, ease of development, modifications, and maintenance, enhanced interoperability, and further practical support for knowledge sharing

(together with ontology's). There is a large degree of correspondence between components and ontology's, and both require agreement on the vocabulary of the domain of ITS, the work that is always underway by several researchers and research groups.

The given model of component based intelligent tutoring systems, presented in the paper, allow easy and natural conceptualization and design of a wide range of ITS applications, due to its object-oriented approach. It suggests only general guidelines for developing of ITSs, and is open for fine-tuning and adaptation to particular application. ITSs developed by using this model are easy to maintain and extend, and are much more reusable than other similar systems and tools.

Further development of the **GET-BITS** model is concentrated on development of appropriate classes in order to support a number of different pedagogical strategies and the student models. The idea is that the student can have the possibility to select the teaching strategy from a pre-defined palette, thus adapting the ITSs to his/her own learning preferences. Such a possibility would enable experimentation with different teaching strategies and their empirical evaluation. Another objective of further research and development of **GET-BITS** is the support for different didactic tools that are often used in teaching.

References

1. Brusilovsky, P., Ritter, S., Schwarz, E.: Distributed Intelligent Tutoring on the Web, in: du Boulay, B., Mizoguchi, R. (eds.): "*Artificial Intelligence in Education*", IOS Press, Amsterdam/OHM Ohmsha, Tokyo, (1997), 482-489.
2. Devedzic, V., Jerinic, Lj.: Knowledge Representation for Intelligent Tutoring Systems: The GET-BITS Model, in: du Boulay, B., Mizoguchi, R. (eds.): "*Artificial Intelligence in Education*", IOS Press, Amsterdam / OHM Ohmsha, Tokyo, (1997), 63-70.
3. Hietala, P., Niemirepo, T.: Collaboration with Software Agents: What if the Learning Companion Agent Makes Errors?, in: du Boulay, B., Mizoguchi, R. (eds.): "*Artificial Intelligence in Education*", IOS Press, Amsterdam / OHM Ohmsha, Tokyo, (1997), 159-166.
4. Jerinic, Lj. and Devedzic, V.: OBOA Model of Explanation Module in Intelligent Tutoring Shell. SIGCSE Bulletin, Vol. 29, Number 3, September, ACM PRESS (1997), 133-135.
5. Stern, M.K.: The Difficulties in the Web-Based Tutoring, and Some Possible Solutions, Proceedings of the Workshop "*Intelligent Educational Systems on the World Wide Web*", Kobe Japan, (1997), 2-10.
6. Woolf, B.: AI in Education, in Shapiro, S., (ed.) "*Encyclopedia of Artificial Intelligence* , John Wiley & Sons, Inc., New York, NY, (1992), 434-444.

Modelling the MCOE Tutor Using a Computational Model

Lúcia M. M. Giraffa[1,2] Michael da C. Móra[2,1] Rosa M. Viccari[2]

[1]Instituto de Informática – PUCRS
Av. Ipiranga 6681 – prédio 16 – Porto Alegre – RS – Brasil 90610-900
{giraffa,michael}@inf.pucrs.br
[2]Curso de Pós-Graduação em Ciência da Computação – UFRGS
Av. Bento Gonçalves 9500 – Bloco IV – Porto Alegre – RS – Brasil
{giraffa,michael,rosa}@inf.ufrgs.br

Abstract: In this paper, we present the definition of the cognitive agent of the MCOE tutoring system using the agent model proposed by Móra et all in [23]. MCOE (Multi-agent Coöperative Environment) is a game modelled using a *multi-agent system* architecture composed of a society of agents who work to achieve a common goal: to assist a student to fight against pollution resulting from foreign elements (polluters) and to maintain the equilibrium of the environment. The system has two kinds of agents: reactive (designed and implemented using the object-oriented approach) and cognitive (designed with a mental state approach). The agent model used allows us both to formally define the cognitive agent and to execute it to verify the accuracy of the model.

Key words: Intelligent Tutoring Systems, Multi-agents Systems. Agent Modelling, Mental States.

1 Introduction

In recent years, a new paradigm arose for educational environments: to make more than one student interact with the same environment under artificial tutor supervision. From the educational point of view, this is not a new idea. However, it became feasible through the development of hardware and software that allow us to connect people using computer networks and related technologies. The impact of these technologies on educational software research was immediate. The researches have begun to introduce these new possibilities to improve educational environments.

Nevertheless, the student model remains the weak part of such systems. There are many factors that contribute to this weakness (hardware and software limitation, and techniques to model the student, knowledge representation, and others). However, the strongest restriction is our imprecise knowledge about the mental activities of the students during teaching/learning process. In order to build an ITS with a good student model, we must understand what is happening in the student's mind during the interaction. We need to understand the process and reproduce it in the machine. Much has been done to understand this process, according to different viewpoints: psychological, educational and computer science. The work of Gagne [6] and Moussalle [11] are examples of the such improvements.

The current technology and our limited knowledge about the human learning process still do not allow us to build the ideal ITS. At this moment, the researchers' efforts to find a computational theory that explains the human information process has not produced the

F. Moreira de Oliveira (Ed.): SBIA'98, LNAI 1515, pp. 50-60, 1998

appropriate answers. In fact, we do have paradigms that try to explain how the information is processed in the human mind. Nevertheless, those paradigms do not allow us to detail the process at the level that we needed. We have now different tools and new possibilities. The multimedia techniques and agents programming paradigm are some of these new technologies that may transform the way to design ITS.

Many systems for educational purposes have adopted the agents' paradigm to better explore the interaction and dynamic changes in teaching-learning environments. The restrictions of these systems can be overcome when we attempt to introduce the notion of co-operation in the teaching-learning process, using a multi-agents focus. Using the agent's paradigm and mental state choreography to improve the student model and the interactions between tutor and students is a new possibility. This is shown by the experiments built by Moussalle [11], where she traces some student's mental activities using that approach.

There is not, in the research community, a consensual definition of what an agent is. We adopt a *mentalistic approach,* where the term agent means a computer system that can be viewed as consisting of mental states such as beliefs, intentions, motives, expectations, obligations and so on. In this case the question of what an agent is replaced by the question of what entities can be viewed as possessing mental states. When a mentalistic approach is adopted, the central problem becomes to choose the mental states that should be used to characterise an agent. The usual solution is to adopt **B**eliefs-**D**esires-**I**ntentions (BDI) as the basic mental states to model agents. These notions and their properties are formally defined using logical frameworks that allow us to analyse, to specify and to verify rational agents, like in [4, 10](among others).

It is a general concern that there is a gap between those powerful BDI logics and practical systems. We believe that the main reason for the existence of this gap is that the logical formalisms used to define the models do not have an operational model that supports them. In [10], the authors propose a BDI model that, besides being a formal model of agents, is also suitable to be used to implement agents. In that model, the notions of belief, desires and intentions are defined using a logic formalism that is both well defined and computational. This allows us both to formally define agents in terms of their mental states and to use this formal model to verify how these agents execute. Since the model is executable, it allows us to model the tutor and students, and to verify them executing a mental states choreography.

Previous work using this approach just considered two agents and a simple testbed with a specific teaching/learning situation using *learning by example* strategy. Just one Tutor and one student were modelled and observed. The results were very interesting because the dynamics of the interaction could be traced and observed. Notice that the tendency of educational environments is to consider more than one student working in a co-operative way based on the change of the traditional educational paradigm, where the teacher is the focus, to another one, where the student is the focus (learn to learn). We strongly believe in the usefulness of this paradigm shift and of mental states metaphor to model agents (and, in particular, ITS). Therefore, our purpose in this paper is to join our previous works about the mental state choreography [8] and apply it to a more complex teaching/learning situation where more than two agents appear. We take advantage of the formal and executable aspects of our agent model to build those choreographies [10]. This is an intermediate step towards our long-term goal of achieving a real society formed by the tutor and the students. At the present stage, we can verify how co-operation between the two students evolves. But this is only simulated, as it happens out of system control.

The approach we take is to create an environment using agent's techniques, with a MAS architecture and mental states modelling integrated with a game-like interface, using simulation with the characteristics of problem solving. The purpose of the problem solving is providing a tool for students and teachers to discover the way s/he thinks and how to integrate conceptual and strategy knowledge to a specific situation (problem). Because we have computational limits we are able to model only problems, and not concepts. We designed a simulation environment not an executable environment. We intend to simulate an agent rational behaviour not a human behaviour. Although this imposes us a pedagogical restriction, it is acceptable at the present stage of our work.

The choice of Environment Education for the domain area is based on the importance to educate children to preserve the nature and learn to control pollution and preserve the planet. This is a very complex open problem with many pedagogical, psychological and modelling implications. As we said earlier we do not have a general educational and psychological theory to support strong student models based on mental states. Therefore, we can use the results obtained until now and try to reproduce them in a computer to trace the dynamic of the process in order to improve the interaction between artificial tutor and students. We would like to trace the process implicit in selecting a specific strategy.

This paper is organised as follows: in section 2, we describe the logical formalism and the agent model we use to define our tutor system. In section 3, we describe the elements that compose the MCOE system. In section 4, we present the tutor architecture we propose and that is used to build the MCOE system. In section 5, we describe a dialogue that shows the interaction of the students and the tutor, and how the mental states evolve. In section 6, we show how to formalise these mental states and how the underlying agent model may be used to execute the ITS model. Finally in section 7, we discuss some of the results we obtained, draw some conclusions and point to some future work.

2 BDI Model Using Extended Logic Programming

The formalism we are using is *logic programming extended with explicit negation* (ELP) with the ***W**ell-**F**ounded **S**emantics e**X**tended for explicit negation* (WFSX). ELP with WFSX (simply ELP, from now on) extends normal logic programs with a second negation named *explicit*, in addition to the usual *negation as failure* of normal logic programs, which is called *implicit negation* in the ELP context. This extension allows us to explicitly represent negative information (like a belief that a property *P* does not hold, or an intention that a property *P* should not hold) and increases the expressive power of the language. When we introduce negative information, we may have to deal with contradictory programs [1]. The ELP framework, besides providing the computational proof procedure for theories expressed in its language, also provides a mechanism to determine how to minimally change a logic program in order to remove contradictions. Our model benefits from these features provided by the logical formalism. As it is usually done, we focus on the formal definition of mental states and on how the agent behaves, given such mental states. But, contrasting with former approaches, our model is not only an agent specification, but it may also be executed in order to verify the actual agent behaviour, as well as it may be used as reasoning mechanism by actual agents. We depart from Bratman's analysis [2], where he states that, along with desires and beliefs, intentions is a fundamental mental state. Therefore, initially we define these three mental states and the static relations between them, namely constraints on consistency among those mental states. Afterwards, we

advance with the definition of dynamic aspects of mental states, namely how the agent chooses its intentions, and when and how it revises its intentions.

When we reason about pro-attitudes like desires and intentions, we need to deal with properties that should hold at an instant of time and with actions that should be executed at a certain time. Therefore, in order to represent them and to reason about them, we need to have a logical formalism that deals with actions and time. In this work, we use a modified version of the Event Calculus (EC) proposed in [10][1].

ELP with the EC provides the elements that are needed to model mental states. Nevertheless, the use of a higher level language would allow us to have a simpler and clearer description of actions and its effects. Therefore, we cast the ELP language and the EC primitives in a simpler syntax that is an extension to the *A* language proposed by Quaresma et all. [11]. The sentences are:

- *A causes F if* $P_1,...,P_n$ – action *A* causes property *F* if propositions P_i hold;
- *F after A if* $P_1,...,P_n$ – property *F* holds after action *A* if propositions P_i hold;
- *A occurs_in E*– action *A* occurs when event *E* occurs;
- *E preceeds E'*– event *E* occurs before event *E'*.

The language still provides means to reference beliefs that should hold in the future or that should have held in the past. In this paper, we make use only of the operator *next(P)* stating that property *P* should hold at the next instant of time. Now that we have the logical formalism set, we are able to define the BDI model.

2.1. The Agent Model

We depart from Bratman's analysis about intentions, its role in rational reasoning and how it relates to beliefs and desires. According to Bratman [2], since agents are assumed to be resource-bounded, they cannot continuously evaluate their competing beliefs and desires in order to act rationally. After some reasoning, agents have to commit to some set of choices. It is this choice followed by a commitment that characterises the intentions. Our model does not define a complete agent, but only the cognitive structure that is part of the agent model.

An *agent cognitive structure* is a tuple B, D, I, T where *B* is the set of agent's beliefs, *D* is the set of agent's desire, *I* is the set of agent's intentions and *T* is the set of time axioms, as defined above. The *desires* of the agent is a set of sentences *DES(Ag,P,Atr) if Body*, where *Ag* is an agent identification, *P* is a property and *Atr* is a list of attributes. Desires are related to the state of affairs the agent eventually wants to bring about. But desires, in the sense usually presented, does not necessarily drive the agent to act. That is, the fact of an agent having a desire does not mean it will act to satisfy it. It means, instead, that before such an agent decides what to do, it will be engaged in a reasoning process, confronting its desires (the state of affairs it wants to bring about) with its beliefs (the current circumstances and constraints the world imposes). The agent will choose those desires that are possible according to some criteria.

Beliefs constitute the agent's information attitude. They represent the information agents have about the environment and about themselves. The set *B* contains sentences describing the problem domain using ELP. An agent *A* believes a property *P* holds at a time *T* if, from *B* and *T*, the agent can deduce *BEL (Ag,P)* for the time *T*. We assume that the agent

[1] For more details on the logical formalism and the definition of the Event Calculus, refer to [10].

continuously updates its beliefs to reflect changes it detects in the environment. We assume that, whenever a new belief is added to the beliefs set, consistency is maintained.

Intentions are characterised by a *choice* of a state of affairs to achieve, and a *commitment* to this choice. Thus, intentions are viewed as a compromise the agent assumes with a specific possible future. This means that, differently from desires, an intention may not be contradictory with other intentions, as it would not be rational for an agent to act in order to achieve incompatible states. Also, intentions should be supported by the agent's beliefs. That is, it would not be rational for an agent to intend something it does not believe is possible. Once an intention is adopted, the agent will pursue that intention, planning actions to accomplish it, re-planning when a failure occurs, and so. Agents must also adopt these actions, as means that are used to achieve intentions, as intentions.

The definition of intentions enforces its rationality constraints: an agent should not intend something at a time that has already past; an agent should not intend something it believes is already satisfied or that will be satisfied with no efforts by the agent; an agent only intends something it believes is possible to be achieved, i.e., if it believes there is a course of actions that leads to the intended state of affairs. When designing an agent, we specify only the agent's beliefs and desires. It is up to the agent to choose its intentions appropriately from its desires. Those rationality constraints must also be guaranteed during this selection process [10].

Agents choose their intentions from two different sources: from its desires and as a refinement from other intentions. By definition, there are no constraints on the agent's desires. Therefore, an agent may have conflicting desires, i.e., desires that are not jointly achievable. Intentions, on the other hand, are restricted by rationality constraints (as shown above). Thus, agents must select only those desires that respect to those constraints. First, it is necessary to determine those subsets of the desires that are relevant according to the current beliefs of the agent. Afterwards, it is necessary to determine desires that are jointly achievable. In general, there may be more than one subset of the relevant desires that are jointly achievable. Therefore, we should somehow indicate which of these subsets are preferred to be adopted as intentions. This is done through a preference relation defined on the attributes of desires. According to the theory defined in [10], the agent should prefer to satisfy first the most important desires. Additionally to preferring the most important ones, the agent adopts as much desires as it can. The selection is made combining the different forms of non-monotonic reasoning provided by the logical formalism.

Once the agent adopts its intentions, it will start planning to achieve those intentions. During planning, the agent will form intentions that are relative to pre-existing ones. That is, they „refine" their existing intentions. This can be done in various ways, for instance, a plan that includes an action that is not directly executable can be elaborated by specifying particular way of carrying out that action; a plan that includes a set of actions can be elaborated by imposing a temporal order on that set. Since the agent commits to the adopted intentions, these previously adopted intentions constrain the adoption of new ones. That is, during the elaboration of plans, a potential new intention is only adopted if it is not contradictory with the existing intentions and with beliefs.

The next step is to define *when* the agent should perform all this reasoning about intentions. We argue that it is not enough to state that an agent should revise its intentions when it believes a certain condition holds, like to believe that an intention has been satisfied or that it is no longer possible to satisfy it, as this suggests that the agent needs to verify its beliefs constantly. Instead, we take the stance that it is necessary to define, along with those conditions, a mechanism that triggers the reasoning process without imposing a

significant additional burden on the agent. Our approach is to define those conditions that make the agent start reasoning about intentions as constraints over its beliefs. Recall that we assume that an agent constantly has to maintain its beliefs consistent, whenever new facts are incorporated. Whenever the agent revises its beliefs and one of the conditions for revising intentions hold, a contradiction is raised. The intention revision process is triggered when one of these constraints is violated.

3 The MCOE System

The 'MCOE' is a pedagogical game that simulates a lake with plants, and different types of fish. These elements are typical of our real world (in this case, the river in our home city and its surroundings).

The conception of the 'MCOE' was based on an ITS architecture [7, 8]. This architecture is composed of a hybrid society of agents who work to achieve a common goal: to fight against the pollution resulting from foreign elements (pollutants) and to maintain the equilibrium. We have reactive agents (bottom of the lake, micro-organisms - plankton, water, plants and three types of fish) and cognitive agents (the Tutoring agent, the ecologist, and students represented by characters).

We designed the system to be played by two students using different machines. They can be in the same physical place or not. The first student chooses a character to play using four options: Mother Nature, Mayor, Citizen, and Tourist. After that, the second student can choose one of the three remaining characters. The system defines the game configuration (foreign elements that will cause pollution) by a sorting process using a random function. The students first see a lake in equilibrium, which soon begins to show the action of the polluters. Their challenge is to maintain the equilibrium and fight the action of these foreign elements for a period of ten minutes[2].

The number of fish swimming in the lake, plants and micro-organisms, the water transparency, and the appearance of the bottom of the lake indicate the situation. There is a visual gauge (the Ecometer) that helps the user to observe the conditions of the lake. A balanced situation is shown when the gauge is full and green. As the equilibrium is lost, the colour changes gradually to yellow, finally changing to red when a dangerous situation is reached.

The system equilibrium is based on the energy level, the occasional predatory actions that can happen in the environment and the number of elements that remains in the system after the pollutants impact. The fish, plankton, and plants reproduce using a function based also on the energy level. If an agent dies (zero energy) it is removed from the system.

4 The Tutor Architecture

The elements of the Tutoring agent architecture are:

- The Tutor mental states. It is a BDI model that represents the desires and beliefs of the tutor. The knowledge about the students (beliefs about the student's beliefs and intentions) is in the tutor's beliefs;
- Knowledge about what tool to use with specific foreign elements. This is also part of the tutor's beliefs, in the formal model;

[2] The choice of ten minutes is because the game was designed to be played during classroom time, in a context of a real class situation, and a pre-defined period of time is necessary for the teacher to make her/his class plan.

- Knowledge about energy levels of scenario elements, a part of the tutor's beliefs, as well;
- Rules to help it to decide what action should be taken according to changes of the students' mental states. This is also part of the tutor's beliefs;
- Sensors for receiving data about the environment.

The kernel of the Tutor architecture is the Behavioural Decision centre (figure 1).

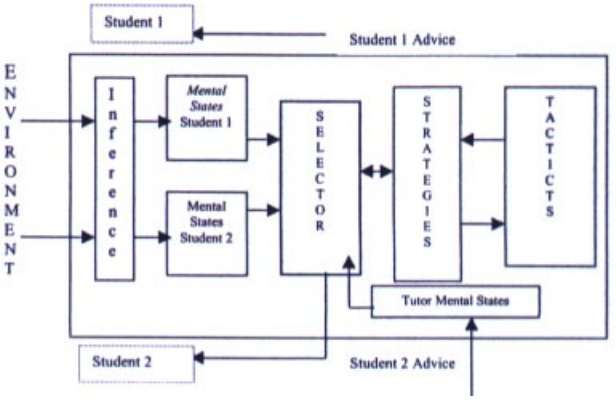


Figure 1: Behavioural Decision Centre

The tutor receives information about each student model. The information is composed by the mental states of each student, after they perform an action, and the environment energy levels. The tutor uses it to select a specific strategy with the associated tactics, according to the selector module. The Tutor itself just observes the interactions, it does not act in the environment. Its only intervention occurs when it gives advices to the student. The tutor doesn't know what will happen along the game, i.e., he doesn't know what foreign elements will appear in the scenery, neither the amount of foreign elements. The students' mental states and their actions (tool activation) are perception information the tutor receives from the environment. The system works as if the tutor hears and sees what happens in the environment. Those would be its sensorial capabilities.

The set of the students' mental states is obtained through a module of inference of attitudes, as presented in Quaresma[12]. The dialogues are seen as composed by a sequence of speech acts. In this context, to speak is to act. When agents speak they accomplish speech acts. The speech acts, according to Cohen and Perrault [3], are actions whose effects affect both the originator and the receiver. Quaresma's work offers a formal and computational model of how to infer the mental states of the participants in a dialogue. Although this is not part of the implementation of the MCOE system, it is considered in the BDC project as an explanation of how to obtain the students' mental states.

5 A Dialogue Choreography

We analyse, now, how the dialogue preceeds in a situation the simulator can create that. The interface system not allows dialogue in natural language. Due to limitation of space, we selected a small dialogue that is complex enough to show how the logical framework is used.

The system places a lot of pollution in the screen and it creates a complex situation. The foreign elements are 2 dredges, 2 Jet-Ski, 2 industrial sewer, chemical pollution and, 2 gas station. After 30 seconds the game begins. The foreign elements are placed in the scenery in a varied way, and not at the same time. This runs in order to put difficulties along the game and to promote cumulative effects through interference in the scenery.

Student1 chooses to be the mayor and student2 chooses to be the Mother Nature. Both mayor and Mother Nature are characters with plenty of resources to combat the pollution. In the beginning the system activate the dredge and the gas station. The energy level decreases: Plants –40%, Micro-organism –50%, Bottom –50%, Fish –20%, Water –20%. At this time, the Ecometer shows the energy level and the colour of some scenery elements changes.

The tutor has one and only purpose: to aid the student to maintain an adequate energy level in the environment. It believes that it may aid by sending the students messages. The contents of these messages will depend on the strategy adopted at the moment by the tutor. These desire and belief are modelled as follows

DES(tutor,verify_knowledge(Student)) if BEL(tutor,energy(ecometer,Ee)),
Ee<100.

Since this is his only desire, it is the only candidate to intention. It will be adopted as intention if the tutor believes there is a sequence of actions that it can executed. But, at this moment, it depends on knowledge about the students' mental states in order to decide what to do. Since it does not have (yet) this knowledge, it waits. But, it inserts in its beliefs triggers that will make it reconsider its options when interaction occurs. For instance, the tutor has the following belief

BEL(tutor, send_message(1)) if BEL(tutor,INT_THAT(Student, receive_aid),
BEL(tutor, energy(ecometer, Ee)), Ee > 70,
BEL(tutor, time(T)),T >8. *(1)*

meaning that if tutor believes the student expects some help, it believes the energy level is bellow 70 and the game time is above eight minutes. So, the help that it can offer is to advise the student to pay attention to the remaining game time, and select the strong tool to fight against the pollution. This belief would place as triggers

$\perp \leftarrow$ *BEL(tutor,INT_THAT(Student, receive_aid),*
BEL(tutor, energy(ecometer, Ee)),Ee > 70, BEL(tutor, time(T)),T >8 *(2)*

i.e., when the pre-conditions for his action of sending a message are satisfied, his is in condition of aiding the student. The agent models then transform the desire in an intention a satisfies it sending the message.

Once the tutor has this intention, it uses its beliefs about the environment (tool, energy level, foreign elements, scenery elements), about how to fight against pollution in the environment (different strategies on the use of the tools to eliminate pollution), about how to advise the students in order to help the students control the environment and so on.

****

(Time Point 1)

Student1: Pollution began and I will apply tool *fine*.

Mental states show the expectation „the energy level will grow up" and „student2 will collaborate".

Student2: I will *accelerate* the degradation.

Mental states show the expectation „the energy level will grow up".

Tutor: It verifies that the students didn't change their believes and the level of energy increased. It stays observing and it doesn't send any message.

None of the triggers set by the tutor are activated by this interaction. Therefore, it does not adopt any intention and remains waiting.

****

(Time Point 2)

The energy level increased but it is not 100%.

Student1: Student1 emits a message for Student2 asking what h/she intends to do.

S/he expects „to receive help from Student2" and shows the belief „Student2 will cooperate".

Student2: Student2 replies with „you should use a tool that increases as much as possible the energy level".

It adopts the intention „cooperate with Student1" and the expectation „the energy level will grow up".

Tutor: The students are co-operating and building a common strategy. As the level of energy is rising and the students maintain its basic believes and desires, the Tutor awaits. All elements have with 100% energy.

****

(Time Point 3)

Student1: Look this! Everything is going well!

Student2: Yes it is working!

Tutor: Awaits.

****

(Time Point 4)

Then appear the polluters: 1 dredge, 1 Jet-Ski and chemical polluters. Energy level at this time: Plants –40%, Micro-organism –40%, Bottom –50%, Fish –30%, Water –20%. The students notices the energy dropped in several places. S/he looks at the colour of the things in the lake.

Student1: I will *intensify* the sunshine because it will place more energy and very fast.

Mental states show the expectation „the energy level will grow up" and the belief „powerful tools increase the energy level soon".

Student2: I will use *removes the license* for everything to increase the energy level fast.

Mental states show the expectation „the energy level will grow up" and the belief „powerful tools increase the energy level soon".

Tutor: Verifies that the energy level grew and the students are not using the rule to control the number of times that the tool can be used.

The students' mental states do not shows the belief „number of tools should be controlled". The tutor emits a message of alert to the two students in this sense.

The Tutor will perform this action when the amount of some of the tools is one. Its beliefs have „number of tools should be controlled" and activates the desire „control number of tools".

In this step, the tutor just analysed the interaction. In fact, according to the agent model, the tutor just expanded its beliefs. But, these new beliefs do not activate any of the triggers. This means that none of the tutor's desires are relevant. In fact, it is not considered rational to act in order to achieve an intentions if one beliefs the intention will be satisfied without any intervention. In our case, if the tutor thinks the students is performing well, it would not be rational to give him any advice.

This is the usual behaviour when our agent model is used. The agent just maintains its set of beliefs consistent. When the change in the belief set activates some trigger, the agent will try to adopt intentions, build plans and so on. In this particular agent, the tutor, the possibility to build plans is not used. In fact, we argue that the tutor should not have a long term plan. Instead, it should have very short term plan that takes into account the history of interactions and, particularly, the last one. It is a cognitive tutor that presents a somewhat reactive behaviour. It is not a simple application of reaction rules because we take into account that history of the interaction and the expectation about the next

interaction. Besides being interesting from the computational point of view [10], it also respects the pedagogical approach being used, namely the construtivist approach.

****

(Time Point 5)

At this time, more foreign elements appear: 1 gas station and 1 industrial sewer. Total of energy after the new polluters: Plants –65%, Personal computer –35%, Bottom –65%, Fish –25%, Water –80%.

Student1: The lake is dying. I don't have more the tool *intensify* and the strongest I have is *accelerates*. I will play with this one.

Beliefs have „strong tools should be used to grow up the energy fast", „number of tool equal zero then tool unavailable", „should control the number of times that can use a tool" and „should control the time of the game".

The desire „cooperate with student" becomes an intention. There are the expectations: „Student2 will cooperate" and „the energy level will grow up".

Total of energy after this intervention: Plants –50%, Personal computer –20%, Bottom –50%, Fish –10%, Water –65%.

Student2: I will use *prohibition*. The energy level will increase.

Mental states show the beliefs „strong tools should be used to grow up the energy fast", „number of tool equal zero then tool unavailable", „should control the number of times that can use a tool" and, „should control the time of the game". The desire „cooperate with student" is selected as intention. There is the expectation „the energy level will grow up".

Total of energy in the time after this intervention: Plants –35%, Personal computer –10%, Bottom –40%, Fish –5%, Water –55%.

Tutor: Verifies that the two students are co-operating. When co-operation[3] exists the Tutor tends to reinforce their behaviour. In fact, the tutor should have a belief describing what indicates there is co-operation between students. For instance

BEL(tutor, student_co-operation) if BEL(tutor, student_exchange_messages).

BEL(tutor, student_exchange_messages) if BEL(tutor, number_message (NM)),NM > 0.

At this point, the triggers **(2)** connected with belief **(1)** set at by the at the beginning are activated. Therefore, the tutor adopts an intention (to aid the students) that is satisfied by sending a message, in this case a reinforcement message.

The dialogue proceeds in this way until the game is over. The students controlled the pollution and the Tutor aided them.

At this point, the tutor produces a description of the student's interactions. This description, with the evolution of the mental states, allows the teacher to verify how well the students understand the problem domain. The choice of inadequate tools or of low combat power in a critical situation indicates that the student still misunderstands the problem and needs a more direct orientation.

6 Conclusions and Future Work

The use of the mental states for modelling the students and tutor behaviour in ITS is based on the notion that this is the usual way to describe interaction among people. It is a

[3] Co-operation is different from collaboration. *Collaboration* is to share information without modify the received information. *Co-operation* is to share information and interfere in the received information. To act together to build something together. All the co-operation involves the collaboration.

not only a metaphor used to build a model. We try to understand what is happening in the student's mind when s/he is thinking. According to McCarthy [9], the use of mental sates is *useful* because it helps to organise the dynamics of the process and *necessary* to get the insights for this mechanism.

Initially the system MCOE was designed to use the set of mental states according to the SEM agent architecture used by Moussale [11]. However, this model does not have an operational environment available at this moment. This made us reconsider the initial proposal and we are now using the Móra, Viccari, Pereira and Coelho [10] approach. They have a formal and computational model to treat the mental states that is based on the same set of mental states as Corrêa. Therefore, we have the same theoretical foundations, we can treat the past, present and next interaction and we profit from them proximity between the description languages used by the two systems.

Although we show here a very simple and small dialogue, in order to identify better all the necessary mental states that should compose the choreography, other dialogues have been created. These dialogues represent two students with great co-operation and content domain, two students with medium co-operation and medium domain content and, two students with low co-operation level and they don't dominate the content. These three stereotypes seems to be enough to our propose. The full formalisation of these more complete dialogues is yet to be done. Nonetheless, the results we have until now are by themselves stimulating and they allow us to trace the agents' behaviour at the desired abstraction level. It also allows us to refine the algorithms and to test the set of rules that regulate the system.

7 References

[1] Alferes, J.J.; Pereira, L.M. Reasoning with Logic Programming. **Lecture Notes in Artificial Intelligence Series.** Springer-Verlag, Berlin, DE., 1996.

[2] Bratman, M.E.. What is intention? In P.R. Cohen, J.L. Morgan, and M. Pollack, editors, **Intentions in Communication.** The MIT Press, Cambridge, MA, 1990.

[3] Cohen, P.R.; Perrault, R. **Elements of a Plan-Based Theory of Speech Acts.** Cognitive Science, 3:177-212, 1979.

[4] Cohen, P.R; Levesque, H.J.. **Intention is choice with commitment**. Artificial Intelligence, 42,213--261, 1990.

[5] Frasson, C.; Mengelle,T.; Aimeur,E. Using Pedagogical Agents in a Multi-Strategic Intelligent Tutoring System. In: AI-ED97: Eighth World Conference on Artificial Intelligence in Education - Workshop V: Pedagogical Agents, 8., 1997. **Proceedings...** Kobe: Japan, 1997.

[6] Gagné, R.M. **The conditions of Learning**. Montreal: HRW, 1984.

[7] Giraffa, L.M.M; Viccari, R.M.; Self,J. Improving tutoring activities using a Multi-Agents system Architecture. Twenty-ninth SIGCSE Technical Symposium on Computer Science Education, 29,1998. **Proceedings...** Atlanta: Georgia,1998. (Student Research Contest -http://www1.acm.org/spost/gabstract.txt)

[8] Giraffa, L.M.M; Viccari, R.M. Tutor behaviour in a multi-agent ITS guided through mental states activities. ITS'98- Fourth International Conference on Intelligent Tutoring Systems. Workshop Pedagogical Agents. **Proceedings…S**an Antonio, Texas, 1998.

[9] McCarthy, J. Ascribing Mental Qualities to Machines. Menlo Park: Stanford Research Center, 1978. (Technical Report)

[10] Móra, M.C.; Lopes, J.G.; Coelho, J.G.; Viccari, R. BDI models and systems: Reducing the gap. In: Agents Theory, Architecture and Languages Workshop. **Lecture Notes on Artificial Intelligence**, Springer-Verlag, 1998.

[11] Moussale, N.M.; Viccari, R.M.; Correa, M. Intelligent Tutoring Systems Modelled through the Mental States. In: **Lectures Notes on Artificial Intelligence** - SBIA'96. Borges,D. and Kaestner,C.(Eds.). Berlin: Springer Verlag, 1996.

[12] Quaresma, P. **Inferência de atitudes em diálogos**.Lisboa: FCT/UNL, 1997. (Ph.D. Thesis -Portuguese)

From a Tridimensional View of Domain Knowledge to Multi-agent Tutoring System

Evandro de Barros Costa[1], Angelo Perkusich[2], Edilson Ferneda[3]

[1] Departamento de Informática e Matemática Aplicada
Universidade Federal de Alagoas
[2] Departamento de Engenharia Elétrica
[3] Departamento de Sistemas e Computação
Universidade Federal da Paraíba
Caixa Postal 10.105 - 58109-970 Campina Grande-PB-Brazil
Fone: +55 83 310 1137 Fax: +55 83 310 1015
ebc@fapeal.br, perkusic@dee.ufpb.br, edilson@dsc.ufpb.br

Abstract. This paper focus on results related to the design of a model of a Computer-based Interactive Learning Environment adopting a multi-agent approach. Questions related to why and how adopt this approach are addressed. We begin presenting and emphasizing the definition of both a view of the domain knowledge structure and a model for it. Then, we show the consequences of this toward the definition of a multi-agent system, leading to a complete learning environment. Related to this environment we define its characteristics and its agent model. Moreover, we discuss both the interaction model among agents and between a human learner and the system. Finally, we focus on the interactions among agents during problem solving activities, discussing the interaction language and the interaction protocols.

1 Introduction

The initiative of using computers in Education started in the 1950s. However, the use of Artificial Intelligence techniques leading to the research field of Artificial Intelligence in Education (AI-ED) begun to be used in the 1970s. One major goal of AI-ED systems is to design and develop computer tutors capable of performing as well as human tutors, thus promoting the learning. Most of the research in this field basically involves tutoring interactions between two main entities: a computer tutor and a human learner. The main idea is to improve the quality of learning. In this context, one of the central issues is the adaptability of the computer tutor to the individual needs of learners (as perceived by the tutor). That is, the possibility that the tutor provides the learner with appropriate and individualized actions in teaching/learning situations. This adaptability has been widely assumed as a major requirement to achieve an effective learning process. However, this has been a complex task to be performed by means of the computer. From now on, such task will be referred as the problem of adaptation.

In this work, we tackle in a certain way the adaptation problem. We have mainly considered issues related to the nature and quality of the interaction

F. Moreira de Oliveira (Ed.): SBIA'98, LNAI 1515, pp. 61-72, 1998.

between the learner and the system. In this perspective, we have worked on the knowledge quality of the system. From this, we define, for example, a model for the domain knowledge.

This paper focus on results related to the design of a model of a Computer-based Interactive Learning Environment adopting a multi-agent approach. In particular, we address specific questions like why and how adopt this approach. Therefore, we begin presenting and emphasizing the definition of a view on the domain knowledge structure and a model for this domain. Then, we show the consequences of this toward the definition of a multi- agent system. Then, we design the learning environment as a whole taking into account an assumed harmonic marriage between Distributed Artificial Intelligence (through a Multi-agent approach) and the notion of Cooperative Learning [8, 11]. Related to this system we define its characteristics and its agent model. Moreover, we discuss both the interaction model among agents and between a human learner and the system. Finally, we focus on the agent interactions during problem solving activities, discussing the interaction language and the interaction protocols.

The remainder of this paper is organized as follows. In Section 2 we discuss the research problem that motivated our work. In Section 3 we consider some requirements of domain knowledge that led to a multi-agent approach. In Section 4 we present a society of tutoring agents and define the agent structure. In Section 5 we present the learning environment, its architecture and its functionality. We discuss the aspects of the interactions in the learning environment are in Section 6. Finally, we present the conclusions in Section 7.

2 Research Problem

As we mentioned in the introduction, our goal is to consider the interaction process between a Learner and a System in a tutoring situation. The scheme of a learning environment is characterized by two main entities which exchange messages: a computer tutor and a human learner. Each one plays particular roles within a cooperative interaction process. In this context, a fundamental problem is how to enrich the interaction between these entities, in order to improve the learning process. Therefore, the tutor has to be able of providing individualized actions in teaching/learning situations, promoting adaptability. We aim at the facilitation of the knowledge acquisition (learning) by the learner during pedagogical activities. So, to do this the system must be designed taking into account quality questions about the knowledge and the ability to manipulate this knowledge (reasoning mechanisms) in an adequate way. For instance, the need for good solutions related to knowledge engineering is a key problems. Moreover, there is a challenge in obtaining a good way to deal with the complexity involved in the knowledge that the system must have to promote a suitable interaction with the learner. The system needs to become more perceptive with regard to the actions of a learner.

In short, to provide good conditions to support effective learning to the learners some qualities are required. These quality requirements may be seen within the state-of-the-art in AI-ED system design and include:

1. *Domain Model*: regarding domain modeling the research directions are taking into account possibilities like: deals with multiple representations [5] and permit knowledge "evolution" [8];

2. *Learner Model*: with respect to learner modeling the recent research are pointing to, at least, two main conceptions. The first one takes into account the design on distributed diagnosis [7] in problem solving situations. The second one considers the possibility of development negotiation dialogue model allowing inspectable student model [10, 1, 9];

3. *Tutoring Model*: integration of multiple pedagogical functions [5] or permits different pedagogical approaches (Contructivism, Behaviorism,$\cdots$) at the same learning environment.

Based on these requirements, we have primarily worked on the improvement of the domain knowledge (DK), taking into account a trade-off between richness and structure. Our work follows this direction, aiming to improve the system capability in interacting with the learner. In this way, we address questions about knowledge engineering that motivated a multi-agent approach.

3 A Tridimensional View of the Domain Knowledge

Assuming that the quality requirements are fundamental for the quality of the domain knowledge, we tackle aspects of domain model. We propose a particular way to consider multiple views on DK and then providing it with a suitable organization in terms of a computational structure. The DK is partitioned based on a special scheme yielding to a set of interrelated sub-domains. This partition scheme considers a dimensional view of DK and is epistemological in nature. Therefore the obtained knowledge is endowed with expertise distributed into different dimensions of DK.

Considering a goal-oriented approach, in a teaching/learning situation, we define three dimensions for DK: *context*, *depth*, and *laterality* [2, 3, 4]. The first one, the context provides multiple representations of the knowledge by leading to different interpretations. The second one, the depth, defined relatively to a particular context, provides different levels of language refinement of DK. The last dimension, the laterality, provides the knowledge support to study a specific knowledge object, that is, dependent knowledge which is considered as external prerequisites of a particular DK.

This tridimensional view provides means by which a given DK can be considered according to a contextual view, where this view can appear together with different levels of depth and laterality related to each context. Once this view is established, the DK is structured according to different contexts $(C_1, C_2, \cdots, C_N)$, where each C_i, with $1 \leq i \leq n$, corresponds to different depth levels defined by $(D_{i1}, D_{i2}, \cdots, D_{im})$. Finally, to each pair $\langle C_i, D_{ij} \rangle$, $1 \leq j \leq m$, different lateralities $(L_{ij1}, L_{ij2}, \cdots, L_{ijt})$, with $t \geq 0$, are associated.

3.1 The Model of Domain Knowledge

Definition 3.1 *A domain D submitted to a partition linked to a view of context and depth is defined by a set of sub-domains. For each pair $\langle C_i, D_{ij} \rangle$, representing a a given view on D, a sub-domain of D, here denoted by d_{ij}, is defined.*

Therefore, a domain D can be seen as composed by:

$$D = \bigcup_{j=1}^{m} d_{ij}, i = 1, \cdots, n_j$$

Definition 3.2 *Analogously, the lateral domain LD, linked with a laterality view, is defined as follows. To each view L_{ijk} define an external subdomain for D, denoted by dl_{ijk}, responsible for a lateral knowledge with respect to D, being dl_{ijk} is defined by:*

$$LD = \bigcup_{k=1}^{t} dl_{ijk}, i = 1, \cdots, n; j = 1, \cdots, m$$

Therefore, the knowledge related to a domain D, to be available in a society, here denoted by D', is defined by the union of the D sub-domains plus the linked lateral knowledge, and is given by $D' = D \cup LD$.

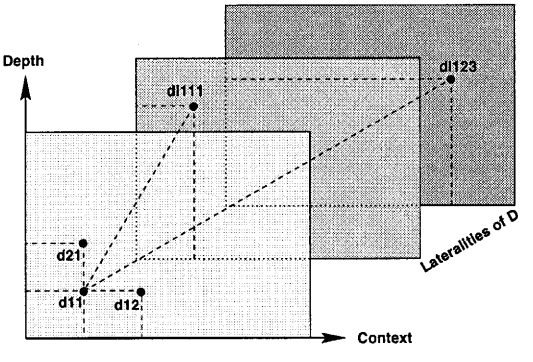


Fig. 1. Tridimensional View of the Domain Knowledge

In Figure 1, we present a target domain D composed by its d_{ij}. It is represented by points of a Cartesian plan defined by the view Context versus Depth, as illustrated in the nearest plan. In other plans are shown the lateralities chosen for D. In this Figure, there is an epistemological order, as mentioned before, where the context axis is viewed as the main. From this, depth axis is viewed related to context. Finally, laterality appears linked to an established context and depth.

In order to illustrate this tridimensional view, we present an example in the classical logic domain. In what follows, show a particular view for this domain. Let us consider$D$ = *Classical Logic* and three contexts.

Contexts: C_1 = an axiomatic view; C_2 = a Deduction Natural view; C_3 = an algebraic view.

These three systems - Axiomatic, Natural Deduction and Algebraic - are equivalents in the sense that they yield the same set of formulas, that is, the set of theorems (yielded by the two first) and tautologies or formula valid (yielded by algebraic approach).

Depth, for C1, one can obtain the following depths: D_{11}= Propositional Classical Logic (Order zero); D_{12}= Predicate Classical Logic (Order one).

Lateralities, to the pair $\langle()$C1,D11), one can define lateralities as follows: L_{111} = Set Theory ; L_{112}= Principle of Finite Induction.

4 The Multi-Agent Tutoring System

Once defined the knowledge domain model and its consequences on learner and pedagogical models defined, it becomes almost natural to adopt a multi-agent approach to design our tutoring system. Therefore, searching for techniques and functionalities of Distributed Artificial Intelligence according to a multi-agent approach. The idea is to use this approach to deal with the complexity involved in the knowledge and its manipulation. In this way, a multi-agent approach was adopted both in design and development of the learning environment model. It is very important to support tutorial interactions endowed with the requirements of quality mentioned before.

Based on the dimensional view of the DK and the quality requirements it becomes clear that it is necessary to manage a big complexity in the involved knowledge. To deal with this complexity and its consequences in the cooperative interaction process, we have adopted a multi-agent approach. This approach seems to be natural to that end by offering various benefits from knowledge engineering. Also, it offers suitable technical apparatus and methods from the field of Distributed Artificial Intelligence. With the multi-agent system introduced it will be necessary to improve the interaction capability of the system. In so doing and considering other components presented in Section 5, we design a learning environment more adequate to promote cooperative learning and also more adaptable to individual learners.

To each one of the sub-domains one builds a tutoring agent defined via the following criteria: from each d_{ij} from D, we define a tutoring agent TA_{ij}. The same is done for d_{ijk} defining LTA_{ijk}.Therefore, two kinds of agents are defined: (i) TA agents and (ii) LTA agents. Then, a TA agent has knowledge on the sub-domain d_{ij} and a LTA agent has knowledge on the domain d_{iik}. These agents are related by means of dependency relations, allowing the identification of different interactions among agents. With the multi-agent model described, we introduce a structure for the local knowledge of each agent and for the organization of the agent society, by means of a multidimensional space. This means that we obtain a centered representation on agents with regard to a domain knowledge.

4.1 Building Artificial Tutoring Agent Society

Definition 4.1 *The set of tutoring agent, TA with respect to a domain D is*

defined by:

$$TA = \bigcup_{j=1}^{m} TA_{ij}, i = 1, \cdots, n$$

Definition 4.2 *The set of Laterality Tutoring Agent LTA with respect to a domain DL is defined by:*

$$LTA = \bigcup_{k=1}^{t} LTA_{ijk}, i = 1, \cdots, n; i = 1, \cdots, m$$

Then, an agent TA_{ij} has the knowledge on the sub-domain d_{ij} and an agent LTA_{ijk} has the knowledge on the sub-domain dl_{ijk}. The union of these agents defines the Society of Artificial Tutoring Agents ($SATA$), that is $SATA = TA \cup LTA$.

4.2 General Description

The society is an open multi-agent system. It is made up of a collection of tutoring agents that may, through established protocols, cooperate among themselves. This cooperation is to achieve the teaching/learning activity to promote the learning of a certain human learner. This society is designed to be open and dynamic in the sense that it allows maintenance operations such as the entry and the exit of agents. Besides, it allows eventual modifications in the knowledge and in the inference mechanisms of an agent. Each one of the agents is an expert tutor in some subdomain, having the necessary knowledge to solve problems in this subdomain. These agents are cognitive and possess properties like autonomy, goal-oriented, social ability [6].

4.3 Agent Structure

Definition 4.3 *Let TA be a tutoring agent. Then, the basic structure of TA is defined as follows:* $TA = \langle TS, SS, DS \rangle$, *where: TS denotes a tutoring system; SS denotes a social system, and; DS is a distribution system.*

where $TS = \langle$*Mediator, Reasoner,Knowledge Bases* $\rangle$, $SS = \langle$ *SK,SM,AL, Coord, Coop, Maint*$\rangle$, and $DS = \langle$*Control, Communication*$\rangle$.

The Tutoring System (TS) is responsible for the cooperative interactions between a tutoring agent and a human learner, in learning activities. The Social System (SS) is responsible for the co-ordination of co-operative activities, reflecting the social behavior of the agent. The Distribution System (DS) executes the sending/receiving of messages through the communication environment. For the TS, *Mediator* is responsible for interpret learners' actions and identify the kind of intervention is need to do. Then, it calls the appropriate module among the *Reasoners* that are responsible for accomplishing tutorial functions. *Reasoners* is composed by a set of modules including an *expert system* and a *pedagogical module*. The *expert system* is responsible for functions like problem solving, cognitive diagnosis and explanation. The *pedagogical module* plays a role of a pedagogical instructor. *Knowledge Bases* contain knowledge about the domain

knowledge (both correct and incorrect), pedagogical knowledge and knowledge about the learner (learner model).

For the SS, SK (social-knowledge or also called in this paper acquaintance model) is the structure in which the agent explicitly represent knowledge about the expertise of the agents in the society (it is used when an agent cannot solve a goal with its own knowledge, so, it tries submitting the goal, or a part of it, to other agents. In case that the agent knows who knows an agent able to solve the goal, it sends it to this agent. Otherwise it simply broadcasts the goal, asking someone to solve it). SM (self-model) is the structure of its own capabilities, representing what the agent is able to do (it is used when an agent has to decide whether it has knowledge to solve a goal, that is a reflexive act about its own knowledge). AL (allocation) denotes a module that is responsible for selecting, based on SK, agents to starting a co-operation. *Coord* (co-ordination) is a module responsible for task synchronization. *Coop* (co-operation) is a module that is responsible for managing the co-operative activities including among other functions conflict resolution. $Maint$ (maintenance) is responsible for maintenance operation with respect to SK and SM of an agent.

For the DS, *Control* is responsible for the correct binding of messages and dialogue instances, it also decides the appropriate destination of a message depending on the type of it (co-operation or maintenance). *Communication* is responsible to interface the distribution system to a given communication environment.

5 General Description of the Interactive Learning Environment

In this section, we do a general description of MATHEMA environment. It belongs to a project aiming design and develop of a computer based Interactive Learning Environment model. Its characteristics and its multi- agent architecture are briefly described. Moreover, we also describe its functionality.

Having designed the $SATA$, we upgrade the environment by adding other elements. Conceptual completeness and software architecture are the main reasons for this upgrade. So, it is composed by six elements [2, 3]. We included in its design an entity representing the group of experts responsible for designing and maintaining $SATA$. Also, two entities for playing interface roles were added. All these elements can be found in both Definition 5.1 and shown in Figure 2.

This environment aims to engage learners in problem-solving activities with respect to a knowledge domain that the society of tutoring agent is assumed as an expert. During these activities the agents in $SATA$ are able to monitor and cooperate in favor of learners' knowledge acquisition. All of these entities communicate through exchange messages.

Definition 5.1 *The architecture of MATHEMA, denoted by (M_Arc), is defined as follows:* $M_Arc = \langle Learner, SATA, HES, IA, MA, EM \rangle$,

where HES is a Human Expert Society (working as integrated source of knowledge through maintenance operations over $SATA$), IA is the interface agent between the learner and the $SATA$ and is responsible for the selection mechanism

to choose a tutoring agent (it will become his supervisor). MA (Maintenance Agent) is the interface between HES and $SATA$ offering the necessary means for communication and maintenance of the Agent Society. EM is an external motivator representing human external entities that can motivate the learner to work in MATHEMA (may be for example: his teacher, his friends, etc.). The functional architecture of MATHEMA is shown in Figure 2. The arrows shown enhance an external view of the interactions that can take place in MATHEMA. Based on these interactions, we give an overview of MATHEMA functionality. First, let us consider a certain learner, motivated by a human external motivator, starts an interaction by means of the following steps: (1) A dialogue between the Learner and the Interface Agent is established. So, the Learner informs the IA about his objectives and intentions. (2) IA helps him in choosing his supervisor. (3) The result of (2) is the selection of an agent, named supervisor agent, which will then "responsible" for monitoring the learning activities of the learner and interaction process between the learner and the selected agent starts. (4) During this interaction, situations with different complexities may occur. Usually, these situations take place when the demand of the learner involves more than one agent (one-to-many) and occasionally involving the HES.

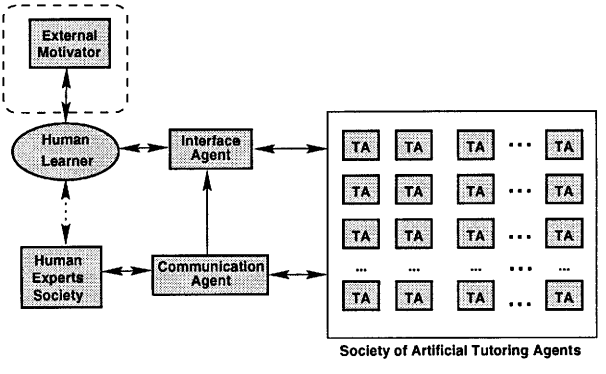


Fig. 2. The Architecture of MATHEMA

6 About the Interactions

Our work is mainly focused on three types of interactions: (1) the interaction that occurs between the human learner and the SATA through a tutoring agent at each time; (2) the interaction among tutoring agents; and (3) between SATA and human experts society. In this paper we emphasize in more details the discussion related to the interaction among tutoring agents. 6.1 Between a Human Learner and the SATA The interaction model involving a learner and an agent in the SATA is based on problem-solving situation. To this, we deal with a style of cooperative interaction where the system can make some kind of intervention. The idea is that from a problem-solving activity various pedagogical functions

can be accomplished in order to support an adaptive interaction process. These interactions can be seen through a process of message exchange in which a TA sends a certain message whose content is X to the learner and the learner answer it with a message whose content is Y. The content of X may be a pedagogical function like a problem, an explanation, a diagnosis, an instruction. On the other hand, the content of Y based in general on X, may be a solution, a query, an argumentation, a problem.

6.1 Interaction among Agents

In this interaction type, we focus on issues regarding why and how one agent needs to achieve a particular goal together with other agents. We define that this interaction process occurs as follows. First, an agent tries to solve a problem by itself. When the agent can perform alone all the needed actions in problem solving, it does so. Otherwise, it activates its social system to select one or more agents that can perform the needed action(s) in order to solve a problem which it cannot perform. Once this selection has been done, the agent starts an interaction protocol. Before detailing this interaction protocol we introduce the functional architecture for the agents.

Functional Architecture for the Agents In Figure 3 we show the functional architecture for the agents in MATHEMA by means of a hierarchical multi-level structure. As the reader can observe each one of the systems introduced in Section 4.3, namely: the Tutoring System, the Social System, and Distribution System, are embedded in this hierarchy together with a communication environment. The discussion related to the communication environment is out of the scope of this paper. In what follows we describe the functions executed in an agent that asks for co-operation and in the agent co-operating.

Asking for Co-operation: for the tutoring system, when the problem solver detects the necessity to co-operate, it submits the task, or a set of sub-tasks, to the social system through the allocation module. This module, based on the acquaintance model, determines a tuple defined by ?task,set of agents?, where the set of agents specifies the agents that may be able to co-operate in order to achieve a particular task. The set of agents may be empty, or have one or more agents. The tuple $\langle task, set\_of\_agents \rangle$ is then passed to the co-ordination module. This module provides means by which the causal dependency among tasks, in case it exists, can be kept. Also, it passes onto the co-operation module each one of the tuples. Observe that the co-operation module works over isolated tuples aiming at conflict resolution, by means of a negotiation mechanism, whether the cardinality of the set of agents is greater than one. It is important to point that one instance of dialogue of the co-operation module is created for each tuple $\langle task, set\_of\_agents \rangle$. In Section 6.2.3 we discuss the other scenarios and functionalities.

The control module in the distribution system is responsible to maintain the consistency among the messages exchanged among the different instances of dialogue of the co-operation module and communication module. The communication module abstracts details related to the communication environment.

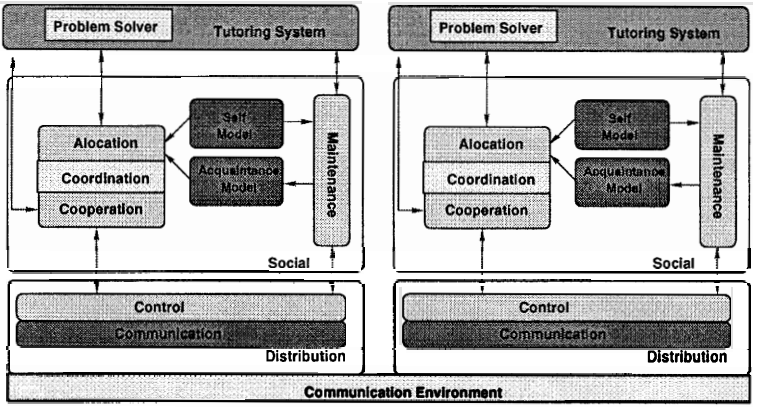


Fig. 3. Functional architecture for the agents

For instance it provides a mapping between the identifier of an agent and the machine or process executing it.

Providing Co-operation: when a message arrives in the agent providing co-operation two situations can occur. In the first one the co-operating agent directly solves a received task. The task received from the distribution system is passed onto the co-operation module in the social system, and then to the tutoring system. When the task is solved the result is sent back to the co-operation module, that returns it to the agent that asked for co-operation, through the distribution system. In the second situation it simply indicates whether a task can be solved. The steps to accomplish this are the same as those for the first case, except that instead of solving the tasks, the agent providing co-operation simply return an indication of its capability to solve the task. The distribution system performs the same functionalities as described for the case of the agent that asks for co-operation.

Social Language and Interaction Language In order to define the interaction language among the agents lets us introduce the definition of agent model and the social language. The social language aims to yield a standard vocabulary related to the elements from Social System described in Section 4.

Definition 1. Let AGM_i be the agent model for an agent TA_i, $AGM_i = \langle TS_i, SK_i, SM_i, EW \rangle$

where TS, is the Tutoring System, SK_i, is the Social Knowledge, SM_i is the self-model for the tutoring agent TA_i, EW is the external world, defined as $EW = \langle Interface\_Agent, Maintenance\_Agent \rangle$.

The social language and the interaction language were defined but are not presented in this paper due to lack of space.

Interaction Protocols In the previous section we introduced the model for the agents as well as the syntax of the interaction language in the MATHEMA

environment. The purpose of this section is to discuss the underlying protocols to promote the co-operation among each other or with the human expert society. Roughly speaking the protocols must guide the co-operation dialogues and the maintenance dialogues.

In the case of co-operation dialogues three situation have to be considered to define the protocols. The first one is a *Master Slave* situation, that is, when an agent needing co-operation knows one and only one agent capable of solving a given task. The second situation considers a scenario where an agent needing co-operation identifies a set of agents that may co-operate with it to solve a task or set of tasks. In this case, in the first step the agents capable to co-operate are queried in order to know if they can solve the task and in what conditions. After receiving the response, the agent needing co-operation chooses one of the agents to co-operate. After this, a direct contract takes place. The third situation is the one when no agent can be selected to solve a task or set of tasks. In this case a broadcast occurs. If one or more than one agent acknowledges, then the protocol resumes to one of the previous situations. Otherwise, the agent asking for co-operation executes a maintenance protocol, as discussed as follows.

In the case of maintenance dialogues the protocols must support the situation in which the society of artificial tutoring agents (SATA) cannot solve a problem or a part of it. In this case what must occur is an internal reorganization in the SATA by modifying the capabilities of one or a set of agents. Also, the protocols for maintenance dialogues must support the situations were an agent has to be introduced or removed. This is basically motivated by the fact that the SATA is constructed based upon a open multi- agent architecture.

When a new agent is introduced in the SATA, it is necessary that all other tutoring agents belonging to the SATA be aware of the capabilities of the new introduced tutor. On the other hand, when an agent has to be removed from the SATA all of the other agents must be aware of this situation in order to maintain their acquaintance models consistent with the state of the SATA. The situation of updating the knowledge of one agent can be considered as a removal of an agent followed by the introduction of another one with its knowledge updated.

6.2 Interactions Between the Agent Society and the Human Expert Society

The interactions between the agent society and the human expert society, take place when the HES needs providing a maintenance operation over $SATA$. Basically, we have considered two situations in this type of interaction. A normal situation occurs when the $SATA$ is not enough to answer a problem. Other possible situation for maintenance is that one in which HES observes the interaction process between the learner and $SATA$ and decides applying a maintenance operation over $SATA$. In order achieve a maintenance operation HES uses the maintenance agent. This agent provides the necessary means to achieve this operation through perception, communication mechanisms.

7 Final Remarks

In this paper we have presented the design of a multi-agent interactive learning environment model called MATHEMA. We specifically address and emphasize

the modeling of domain knowledge and its consequences towards a society of agents via a multi-agent approach. Regards to this society we have defined the agent structure, the co-operation model among them and their protocols. Also, we focus on the agent interactions during problem solving activities. Moreover, we discuss some aspects of the interaction between a learner and an agent in the society of artificial tutoring agents ($SATA$).

Two prototypes were developed to validate results in MATHEMA environment. Logic and Algebra were the domains used. The idea was to verify the interactions among agents in $SATA$ and the interactions between a Tutor Agent and a Learner. Now, we are working with interactions between $SATA$ and HES. In so doing, we intend to improve in $SATA$ its capability of knowledge acquisition.

References

1. S. Bull and H. Pain. "did i say what i think i said, and do you agree with me?": Inspecting and questioning the student model. In *Proceedings of AI-ED 95 - World Conference on Artificial Intelligence in Education*, Washington, DC, August 1995.
2. E. B. Costa, M.A. Lopes, and E. Ferneda. Mathema: A Learning Environment Based on a Multi-Agent Architecture. In J Wainer and A. Carvalho, editors, *Proc. of 12th Brazilian Symposium on Artificial Intelligence*, volume 991 of *Lecture Notes in Artificial Intelligence*, pages 141–150. Springer-Verlag, Campinas, Brazil, October 1995.
3. E.B. Costa and A. Perkusich. Modeling the cooperative interactions in a teaching/learning situation. In *Proc. of The Intelligent Tutoring Systems, ITS, 96, Lecture Notes in Artificial Intelligence*, Montreal, Canada, June 1996.
4. E.B. Costa and A. Perkusich. Designing a multi-agent interactive learning environment. In *Proc. of International Conference on Computers in Education, ICCE'97*, Malasya, December 1997.
5. G. Cumming and J. Self. Learner modelling in collaborative intelligent educational systems. In Peter Goodyear, editor, *Teaching Knowledge and Intelligent Tutoring*, Norwood, N.J, 1991.
6. S. Franklin and A. Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In *Third International Workshop on Agent Theories, Architectures, and Languages*, 1996.
7. S. Leman, S. Giroux, and P. A. Marcenac. A multi-agent approach to student model student reasoning process. In J Wainer and A. Carvalho, editors, *Proc. of 12th Brazilian Symposium on Artificial Intelligence*, volume 991 of *Lecture Notes in Artificial Intelligence*, pages 285–265. Springer-Verlag, Campinas, Brazil, October 1995.
8. F.M. Oliveira, R.M. Viccari, and H. Coelho. A Topological Approach to Equilibrium of Concepts. In *Proc. of the XI Brazilian Symposium on Artificial Intelligence*, Fortaleza, Brazil, 1994.
9. A. Paiva, J. Self, and R. Hartley. Externalising learner models. In *Proceedings of AI-ED 95 - World Conference on Artificial Intelligence in Education*, Washington, DC, August 1995.
10. J. Self. Bypassing the intractable problem of student modelling. In *In Proceedings of Conference on Intelligent Tutoring Systems*, Montreal, Canada, August 1988.
11. R. Slavin. *Learning to cooperate, cooperating to learn.* Plenum, New York, 1995.

A Transfer Dictionary for Words and Bigrams

Jorge Kinoshita

Escola Politécnica da Universidade de São Paulo
Departamento de Computação e Sistemas Digitais (PCS) - Brazil
email: jkinoshi@pcs.usp.br URL: http://jk.pcs.usp.br

Abstract. We present a transfer dictionary model that explains how words and bigrams can be translated from one language to another in a concise way. Translation of bigrams can be used to handle the syntactic differences between languages. The dictionary is being used in a English-Portuguese translation machine project, however, the transfer dictionary model is generic enough to be applied to other projects as well.

1 Introduction

Bilingual dictionaries from language L1 to language L2, made to humans, declares to each word (ex: *"give"*):
- one or more tags (obligatory). Ex: verb.
- one or more translations to each tag (obligatory). Ex: *"dar"* (in Portuguese).
- translation depending on other words (optional). Ex: *"give-up"* - *"desistir"*. In this example, the translation is associated to a bigram.
- idiomatic expressions based on the word (optional).

We present a dictionary model to transfer systems that is similar because it declares the same kind of information dealing with words and bigrams, although we are not yet dealing with n-grams (ex: idiomatic expressions). The difference is that our declaration must follow a strict syntax that declares both morphological and syntactic aspects (ex: words position) of L2. This paper is an evolution of [1] that studies the union of a database of bigrams and a hierarchical structure of nouns in a machine translation project. If we add some extra notation to the database of bigrams then we get the transfer dictionary that is presented here. In this paper, bigrams are 2 words that has some syntactic relationship. Many syntactic differences in L2 can be handled by translating bigrams according to the dictionary.

Our transfer system is a bit different from others.

Generally, the machine translation based on the transfer approach operates in three steps:

1) **analysis**: maps sentence *As* into a syntactic tree *Ast*.
2) **transfer**: maps the structure *Ast* into another tree *Bst*, through the use of some transfer grammar.
3) **generation**: maps the tree *Bst* into the sentence *Bs*.

The transfer grammar is a set of rules heavily dependent on the kind of structure resulted from the analysis. For instance [2]:

F. Moreira de Oliveira (Ed.): SBIA'98, LNAI 1515, pp. 73–82, 1998.

$[gov : know, subj : \$1, obj : \$2[NP]] => [gov : conhecer, subj : \$1, obj : \$2]$.
This rule specifies that if the verb *"know"* has a subject and an object of type NP (noun phrase) then it must be translated as *"conhecer"* in Portuguese (*connaitre* in French) instead of *"saber"*. This rule, in particular, can not be applied to the sub-sentence : *"... in order to know that man"* because the right side of the rule does not match the tree due to the lack of subject (*"know"* is in infinitive). This rule should focus only on two words (a bigram): the verb *"know"* and its object (ex: *"man"*). We assume that most of the transfer rules can (and should) be written focusing only on bigrams because they define a good context and tend to be translated in the same way.
We are building an English to Portuguese transfer dictionary for a machine translation project. Our translation process is:
1) **Analysis**. Link the words of an English sentence in pairs (by some dependency grammar). It is possible to associate a relation-tag (kind of linkage) to each bigram. For instance in *"know-man"*, the relation-tag is +VB_NN that means *verb-object*.
2) **Transfer**. Translate the bigrams according to the dictionary and assemble them, building the translated sentence S with Portuguese words given in radical form.
3) **Generation**. Put the words of S in their normal form based on their radical form.

2 The Transfer Dictionary

In the dictionary, we declare translation of words and bigrams. Today, the treatment of idiomatic expressions is out of this scope.

2.1 Translation of a Single Word

The translation of one word is based on its tag. The set of tags in Portuguese is different from English. For instance, in Portuguese, nouns are classified according to its genre. We created a notation that translates not only the word but also its tag. We are adopting the following symbols:
W - word. Ex: *woman*
W' - word translation. Ex: *mulher*
W^T - word tag according to the TREEBANK notation. Ex: *NN*
W'^T - tag of word translation. Ex: *NNf*. The tag W'^T can be:
(1) the same as W^T.
(2) derived from the W^T by appending some information to it. Ex: NNf is composed of NN (substantive) and f (genre feminine).
(3) different from W^T. For instance, in some cases, *MD ("Modal")* becomes a verb in Portuguese.
The declaration of single words in the dictionary follows the syntax:
$W\ W^T : Wg'$

Example: $woman\ NN\ :\ mulher.f$
where:
$Wg' = (W' \mid W'.X \mid W' :: X)$
$W'.X$ means: "Append string W^T to X generating W'^T". Ex: $mulher^T$ = NN appended to f = NNf.
$W' :: X$ means: "set X to W'^T". Ex: $can\ MD : poder :: VBP$.
Generally, words and tags in the dictionary are in their radical form. The agreement in genre, number and person for Portuguese words is made by the transfer phase and is not explained by the rules in the dictionary.

2.2 Translation of Bigrams

The translation of bigrams deals with many aspects as it is shown in the examples:
a) *yellow - man : homem - amarelo*;
b) *give - up : desistir*;
c) *like - orange : gostar de - laranja*.
A bigram is generally translated as another bigram (ex: a). In Portuguese, adjectives generally follow the noun. The adjective *"amarelo/yellow"* had its position exchanged with its head.
In example (b), *"give-up"* is translated as one single word.
In example (c), the Portuguese verb *"gostar"* is not directly linked to its object and requires the preposition *"de"*.
We propose a set of rules to deal with these facts in bigram translation. These rules can be organized in a dictionary. There are 2 kinds of entrances to bigrams:
1) *a full bigram* (ex: *give - up*). The two words of the bigram are declared.
2) *a partial bigram* that is composed of one word and a relation-tag (ex: *know - +VB_NN*).
First, we are going to explain the rule syntax in order to translate two words and then, show how this same syntax can be generalized to translate partial bigrams.

Relation Tag Notation . The two words of a bigram are in a dependency relation. For instance: in the pair *concern - with* we can say that *"with"* is subordinated to *"concern"*. We can associate a relation-tag to each bigram. The relation-tag is derived from the two morphological tags of each word. For instance, the relation-tag of (concern, with) is +VB_PREP (verb and preposition). These two tags (ex: VB and PREP) are ordered according to the sequence of words in English sentences. The sign "+" before the first tag shows that the first word is the head, otherwise the sign "-" is used. For instance, the relation tag of (big, house) is -JJ_NN. The relation-tag does not follow exactly the TREEBANK notation [6]. For instance: -NN_VBP, -NN_VBD, etc. are mapped to -NN_ROOT in order to give the notion of *"subject"*.

2.3 Translation of Full Bigrams

The translation rule syntax of a full bigram is:

$H\ H^T\ S\ RT : H'\ S'$

For instance:

$concern\ VB\ with\ +VB\_PREP : preocupar\ com$

where :
H = head-word;
H^T = tag of H according to the TREEBANK notation;
S = subordinated-word;
RT = relation-tag;
H' = translation of head-word;
S' = translation of subordinated-word.
The pattern ($H\ H^T\ S\ RT : H'\ S'$) is incremented in order to specify the following operations:
a)**Order declaration**. When a bigram is translated, the order of the 2 words can be different in Portuguese. For instance: *"Ethernet protocol"* is translated as *"protocolo Ethernet"*. The order can be specified by adding the sign "+" to S' (otherwise, the English order is maintained): S'+ means that H' comes after S' and +S' means that H' comes before S'. For instance:

$protocol\ NN\ Ethernet\ -NN\_NN : protocolo\ +Ethernet$

In the transfer phase, the order declaration imposes a movement of not only the dependent-word but also of the entire phrase bellow it. For instance: *"a very fast car"* is translated as *"um carro muito rápido"* (a car very fast). The fact that S' *"rápido"* (fast) was moved to a position after H' *"carro"* (car) made the entire phrase *"muito rápido"* (very fast) be moved after *"carro"*.
b)**suppression of S' or H'**. There are cases that S' or H' is an empty string. This is declared by the "minus" sign in the place of the translated word (generally the S'). For instance: The case of translating *"there + to be"* as *"existir"* is easily handled by the rule:

$be\ VB\ there\ -EX\_VB : existir\ -$

c)**insertion of word**. A word in Portuguese can be inserted in the translation. For instance, *"like"* is generally translated as *"gostar"* plus the insertion of the preposition *"de"* . For instance, *"like chocolate"* is declared as:

$like\ VB\ VB\_NN\ -\ chocolate : gostar < -de\ chocolate$

The notation $H' < -P$ means that P must be placed before the phrase that has S' as its head.

2.4 Translation of Partial Bigrams

Instead of writing many bigrams (know, English), (know, John), etc. and their translations, it is possible to omit the object of *"know"*:

$know\ VB\ < VB\_NN-\ : conhecer$

The general statement is:

$H\ H^T < RT : H'$ or $H\ H^T < RT : H'\ S'$ (head in focus) and

$S\ S^T > RT : S'$ or $S\ S^T > RT : H'\ S'$ (subordinated-word in focus).
For instance, in Portuguese it is common to suppress the translation of *"it"* when behaving as the subject (-NN_ROOT) of some sub-sentence. This fact is expressed as:

$it\ PRP > -NN\_ROOT : -$

The *"no translation"* **Symbol - % :** It is possible to refer to some translated word W' (H' or S') without imposing any translation. This is helpful when we want to specify some attribute of W' such as its position in the translated sentence or its morphological tag. When the "%" symbol replaces W' in a statement S, then S does not impose any translation to W. Other rules will take care of translating W. For instance: the rule ($H\ H^T < RT : H'\ S'$):

$boy\ NN < -JJ\_NN : menino\ + \%$

means: any word referred by "%" (JJ/adjective) must come after *"menino"*. It can be applied to $(yellow, boy) => (menino, amarelo)$.

2.5 Relation-Tag-Only Based Translation

Other transfer rules that does not depend on words can be created guided by the above pattern. These rules are not declared in the transfer dictionary, but are used in the transfer phase. We decided to show these rules because they were derived from the same syntax of the dictionary. The rule syntax is:

$S^T << RT : H'\ S'$

or:

$H^T >> RT : H'\ S'$

In these rules, H' and S' are generally expressed by % . For instance:

$NNP >> -NN\_NN : \%\ + \%$

Ex: *"Ethernet protocol"* becomes *"protocolo Ethernet"*.

3 Using the Dictionary in the Transfer Phase

In our machine translation project the translation is done in 3 steps:

1) **Analysis**. Each word is linked to its head (one word, the root of the sentence, does not have a head). For instance, *"the boy sold a new house"* is analyzed according to Table 1.
The first column designates the word position in the sentence and the second column its head. For instance, *"house"* is the object of *"sold"*.

2) **Transfer**. The graph of words as knots and dependency relations as arches is generally a tree. The transfer phase takes all the bigrams from the root to the leaves and translates each bigram according to the transfer dictionary. First it

Table 1. Analysis result

pos	head	tag	word
1	2	DT	the
2	3	NN	boy
3	0	VBD	sell sold
4	6	DT	a
5	6	JJ	new
6	3	NN	house

tries to translate the bigram if it is declared in the dictionary; otherwise it tries to translate H or S based on the relation-tag RT of the bigram and, then, finally it tries to make a syntax rearrangement based only on RT. The next step is to translate all words that did not fit any rule for bigrams. An example of the transfer phase output is in Table 2.

Table 2. Transfer result

pos'	pos	head	tag'	word'
1	1	2	DT	o
2	2	3	NN	menino
3	3	0	VBD3s	vender
4	4	6	DTf	um
5	6	3	NNf	casa
6	5	6	JJf	novo

The translation of bigrams is done in a top-down direction because it is easier to handle the movement of words and phrases around the head-word. The statements used from the transfer dictionary are in Table 3.

Table 3. Rules

$NN << -JJ\_NN \ : \ \% \ + \%$
$a \ DT \ : \ um$
$boy \ NN \ : \ menino$
$house \ NN \ : \ casa.f$
$new \ JJ \ : \ novo$
$sell \ VB \ : \ vender$
$the \ DT \ : \ o$

3) **Generation**. The words must be conjugated according to their tags and the result is: *"o menino vendeu uma casa nova"*.

3.1 The Transfer Phase Internals

Each knot of our syntactic tree corresponds to one word of the sentence. If we project the design of the tree in a horizontal line then we get *proj(tree)* that is a sequence of words that corresponds to the input sentence. In the literature, the transfer phase maps one syntactic tree into another tree'. If we had adopted the same approach, then we would construct tree' and work on its projection in order to get the translated sentence. Instead of this procedure, we decided to create one structure that reflects *proj(tree)* and change it in order to create *proj(tree')* that leads to the output sentence. The structure *proj(tree')* is an array where each element *proj[pos']* corresponds to one word in the translated sentence. Each position **pos'** of this array contains (see Table 2):
pos : The reference to the English word in position pos (see Table 1). Ex: for $pos' = 5$, $proj[5].pos = 6$.
word' : The Portuguese word in its radical form. Ex: $proj[3].word' = vender$. If $proj[X].word'$ is empty (due to lack of data in the transfer dictionary) then the corresponding English word is taken as word' and put in the translated sentence.
tag' : The Portuguese tag. It is generated by rules of the transfer phase (ex: *subject-verb* and *adjective-noun* agreement) and according to the declarations in the transfer dictionary. If $proj[X].tag'$ is empty, then the English tag is used. Ex: $proj[6].tag' = JJf$.
head' : The pointer to the Portuguese head of word'. The $proj[X].head'$ is generally empty for all X in the projection. When empty, the corresponding English *head* (head of $proj[X].pos$) is used for the agreement rules in the transfer phase, otherwise $proj[X].head'$ will be used. In Table 2, we do not show the head' column because it is empty. We assume, for instance, that *head'* of *"novo"* is *"casa"* (and made the agreement in genre and noun) because $proj[6].pos = 5$ and the head of 5 in the English sentence is *"house"* that corresponds to *"casa"*. We operate on the projection of tree' when applying the rules in the transfer dictionary. Initially, we set *proj(tree')* to *proj(tree)* by making $proj[i].pos = i$ for $1 \leq i \leq length(InputSentence)$. Then we apply the rules from the dictionary. The main rules that modify proj(tree') are:
a) Order declaration. When it is necessary to change the order of some bigram (H,S), then we:
1) Move the head H to the position of the subordinated-word S: $proj[H].pos$ becomes $proj[S].pos$.
2) Move S and all its descendents (the tree that has S as root) to the position of H.
b) suppression of S' or H'. It is just to remove one position of proj(tree').
c) insertion of word. In the rule:
$H\ H^T\ :\ H' < -P\ S$
it is necessary to locate X that is one position before (or after depending of H) of the boundary of the tree that has S as its root and make:
$proj[X].word' = P$
The other elements (pos, tag', head') can be filled if it is necessary to conjugate P.

Generally, P is a preposition that appears on the left of S and is not conjugated, therefore, nothing is set to the other elements.

4 Inside the Transfer System

Besides the transfer phase, the transfer dictionary has been applied in other phases of our machine translation project:
a) **parsing**. Bigrams have been applied to solve ambiguities during the analysis phase[3]. In our project, bigrams are used to solve compound names and preposition attachments.
For instance, the phrase *"data flow"* can be thought as a sequence of a noun and a verb (NNS VBP) and incorrectly analyzed, and translated as *"dados fluem"*. As technical texts have a high frequency of compound names, we decided to study and solve the compound names before the parsing process by just looking up bigrams in the transfer dictionary.
b) **text revision**. In our system, the information about the syntactic tree and its projection is not discarded. During the text revision, the user can modify the dictionary providing new information about words and bigrams. This new information is automatically applied only where it fits in the text. This process is made in a interactive way by the web interface:
1) The user reads the text, finds some wrong translation, and points to the word that is inside some sentence (using the mouse).
2) The system looks the word and its head from the syntactic tree and asks for a correction in the the word or its associated bigram, presenting all the possible entries of the transfer dictionary (word, bigram or partial bigram) in a form (even though they are not declared).
3) The user fills the form.
4) The system makes the correction in all the other sentences looking a cross-reference table.
This process speeds up the translation process because each revision (ex: a new translation for a word) can affect the entire text.

5 Conclusion

Operations such as exchanging word places, suppressing/inserting words are very common in the transfer phase. In spite of it, there are few works in the literature about this theme. For instance, [8] is an interesting work about the English to Japanese translation and presents a transfer grammar that is more complex than the dictionary presented here. The main contribution of this paper is to propose a division between general transfer rules that operates on the syntactic tree and transfer rules that operates only on bigrams. Simpler rules for bigrams were possible because, even though two words in one language can be translated as a different number of words in another language, our notation enables a bigram to be always mapped as another bigram. This approach can not cover all transfer rules. For instance, the trigram *"as well as"* requires other rules. This

formalism is being applied to our machine translation project with success. The dictionary enables a simple transfer system, for instance, it is not necessary any complex grammar for sentence generation. We do not map a syntactic tree into another one. In this approach, it is necessary to match a small tree (the rule) into the syntactic tree. Instead, we change the projection of the tree observing only bigrams, which simplifies the sentence generation.
The simplicity of the formalism suggests that it could be applied (or easily adapted) to projects that has an analysis similar to that provided by dependency grammars. The parser based on the Link Grammar [7] builds a planar graph (the words are knots, and the arches are the links between 2 related words) for each sentence that could be analyzed. In this graph, each arch has a label (our relation-tag) that provides syntactic information. The dictionary presented in this paper can be easily adapted to the Link Grammar work by the tasks:
1) transform a planar graph in a tree or forest of trees (sometimes it is necessary to cut some linkages).
2) adapt the nomenclature of relation-tags.
3) apply the dictionary as it was shown in this paper.
The project is in evolution and we intend to make the transfer dictionary work with the WordNet [5]. It is expected that it is possible to have just one translation for each meaning in WordNet.
The transfer dictionary notation works fine due to the great tendency in human languages to group related words together as it was observed in many theories ([9], [10], [4]). Some cases can not be handled by bigrams (subject-modal-verb agreement that requires a trigram, the treatment of the particle "TO", idiomatic expressions, etc.) and are treated by other modules in the transfer phase.

Acknowledgment : partially supported by FAPESP.

References

1. KINOSHITA, J.: Aspectos de Implementação de Uma Ferramenta de Auxílio à Tradução Inglês-Português. Tese de Doutorado, Departmento de Computação e Sistemas Digitais da Escola Politécnica da Univesidade de São Paulo, Brazil (1997)
2. DOUG, A.: Transfer. working paper, Department of Language and Linguistics, University of Essex, UK, (1997) 1–8
3. HINDLE,D. MATS,R.: Structural Ambiguity and Lexical Relations. Computational Linguistics, v. 19, n. 1, (1993) 103-120.
4. MELCUK, I. A. : Dependency syntax: theory and practice. New York, State University of New York Press, 1988. apud SLEATOR (1991)
5. MILLER, G.A. ET AL.: Introduction to WordNet: an on-line lexical database. Princeton University, http://www.cogsci.princeton.edu/ wn/, (1993)
6. PENN TOOLS. http://www.cis.upenn.edu/ adwait/penntools.html, (1998)
7. SLEATOR, D. D. K.; TEMPERLEY, D. : Parsing English with a Link Grammar. Pittsburgh, Canegie Mellon University, sleator@cs.cmu.edu (1991)
8. TSUTSUMI,T.: Wide-range restructuring of intermediate representations in machine translation. Computational Linguistics, v. 16, n. 2, (1990) 71–78.

9. RADFORD, A. : Transformational grammar: A first course. Cambridge, Cambridge University Press, (1989).
10. RAPOSO, E.P. : Teoria da gramática: A faculdade da linguagem. Lisboa, Editorial Caminho S.A., (1992).

Integrating Morphological, Syntactical and Semantical Aspects through Multi-Agent Cooperation

João L.T. da Silva*, Paulo R.C. Abrahão** and Vera L.S. de Lima***

PUCRS - Instituto de Informática
Av. Ipiranga, 6681 - Prédio 16, Térreo
90619-900 Porto Alegre, Brazil
{jtavares, abrahao, vera}@andros.inf.pucrs.br

Abstract. The main goal of this paper is to report advances on the use of a multi-agent systems (MAS) architecture in natural language processing (NLP), specifically for natural language interpretation. First, we define some theoretical and methodological assumptions that account for choosing a distributed architecture based on MAS. Thus, we describe a society of linguistic agents where individual components are based on Demazeau's generic agent model [1]. Each linguistic agent includes, besides linguistic knowledge, a set of procedures to manipulate it (in order to reach its goals), and mechanisms for social reasoning [8]. The paper discusses functional and implementational aspects of this society model.

1 Introduction

Lately, the construction of systems for Natural Language Processing (NLP) has faced performance limitations that restrict these systems to the specific vocabularies, the reduced grammars and to the context treated. The reasons of those limitations seem to converge to the inherent complexity of the understanding processes and generation of natural language properly. Meanwhile, the architecture traditionally used for NLP systems constitutes another point to be worked [4].

Most NLP systems are using a sequential architecture that represents the classical linguistic levels (lexical, morphological, syntactical, semantical, etc.). These systems employ huge and monolithic modules that make difficult the backtracking and the sharing of partial results among the modules.

By using a distributed architecture, we take advantage of some points such as: the parallelism in the procedural levels of the system; the distribution of memory resources like knowledge and methods; and, the interaction among the modules which are able to change information by message passing. In this way, some of

* Research assistant, CNPq/PROTEM-CC #380014/97-1.
** Research assistant, CNPq/PROTEM-CC #380156/97-0.
*** Researcher, CNPq post-doctoral at MAGMA/LEIBNIZ, Grenoble - France, #203429/86-9.

F. Moreira de Oliveira (Ed.): SBIA'98, LNAI 1515, pp. 83–92, 1998.

the shortcomings from the sequential approaches can be minimized since partial results and different points of view can be shared by those modules. Besides, as the modules have autonomy and local control over their knowledge and goals, they can dynamically establish heterogeneous communication flows in order to determine their functional roles.

The proposed architecture aims to be an open system through a description of the possible organizations for the agents running a problem solving task.

The rest of the paper is organized as follows. In section 2, we present our generic model of linguistic society, describing the agent model with its internal knowledge, goals and skills. In section 3, the linguistic agents, their functionalities and knowledge are presented. In section 4 we show aspects of the implementation of this model. Finally, in section 5, we present our conclusions.

2 Linguistic Society Model

The society is basically composed of the lexical-morphological, syntactical and semantical agents, which are the "principal" agents that the society needs in order to reach its goals, so, to build a representation for an input sentence. However, this society aims to be an open system which can receive "specialized" agents with different purposes. Then, the society is able to include "secondary" agents which can solve specific problems, like an anaphor agent, an ellipsis agent and so on. In this way, the society is not closed and may be reorganized according to the application goals. In some sense, according to [12], the former class of agents includes "regular" agents and the latter includes "transversal" agents.

2.1 Model of Linguistic Agent

NLP systems have to manipulate large knowledge bases and several types of linguistic knowledge, specially declarative and procedural ones. In this work, the declarative knowledge consists of the semantical and morpho-syntactical properties associated with the lexical items; and, the procedural knowledge involves the heterogeneous knowledge on methodologies and formalisms to dealing with a certain problem.

The model of agent proposed is based on Demazeau's [1] and Ferber's [10] definitions. Each agent has an identity, a collection of capabilities, goals and descriptions about the other agents (its acquaintances), and a behavior (an action module that reacts with incoming messages). Each incoming message triggers an action described by the corresponding specialty descriptions. The architectural model proposed (see Fig. 1) is composed by: (1) domain capabilities, knowledge and goals concerning this agent; (2) communication capabilities with other agents; (3) external descriptions - information about the other agents, which makes it possible to have mechanisms for the social reasoning; (4) actions to be performed according to changes and message exchange with the environment; (5) control of the cooperative and individual activities, making the parallel execution possible.

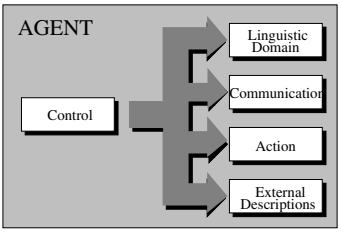


Fig. 1. Internal structure of a Linguistic Agent

2.2 The Linguistic Domain

At this level, capabilities include the execution of lexical-morphological analysis, parsing, semantical analysis and other functions necessary for the maintenance of the agent's linguistic knowledge. Grammar, dictionaries and conceptual networks are the essential attributes of each agent's specialty and, therefore, they are the agent's knowledge which is manipulated by its capabilities according to the goals that it intends to reach.

The agents goals depend on their expertise and on the domain they are associated with. Goals can be individual and/or global, and the global goals derive from the application that one intends with the society: it can be the analysis of a spoken or written text, the building of a representation for a sentence, the synthesis of sentences, automatic translation, etc. The individual goals can be exemplified as the construction of the syntactical structures, the meaning or sense representation for a sentence, disambiguation or resolution of other linguistic phenomena.

As an example of a linguistic agent, consider the following descriptions about the semantical agent:

Knowledge: a lexical database;
Capabilities:
$\text{CAP}_{(SEMANTIC)}$={SemAnalysis(*syntactic_struct),
Find_Concept (word, *concept)
FindClass(word, *Header)
Get_Sel_Binding(noun, adj, *Header)
Get_Co_Composition(verb, word , *Header)
Get_Coercion(verb, word, *Header)
Get_Has(word1, word2, *Header) }
Goals: $\text{GOAL}^{I}{}_{(SEMANTIC)}$={SemAnalysis(*syntactic_struct)=*log_represen}

2.3 The Cooperation Layer

There are two levels of implementation for the linguistic agents: the first level, the linguistic domain, has been presented in the previous section; the second level is called the cooperation layer, and is composed by the communication module and the external description which implements the social reasoning module (those modules are replicated for each agent in the society).

Thus, when we have to describe a new agent, we define its domain (the methods and knowledge that it will need) and connect it to the cooperation

layer so that it becomes an agent able to act and cooperate in the linguistic society.

Following, we describe the main modules from the cooperation layer: communication and external descriptions.

Communication. Each agent has a mailbox and a mail address for sending and receiving messages in an asynchronous mode. In order to promote the exchange of knowledge and intentions, and to make explicit a cooperative and joint action, we have defined a KQML-based communication language [3], using only those primitives that better represent intentions and cooperation, such as: *ask-if, ask-one, tell, reply, broadcast* and so on.

Using these primitives makes unnecessary to use specific protocols to request further information or to send partial results among the agents. For example, any agent can send any kind of information through the *tell* primitive:

TELL
:content "(the, art)"
:force INFORM
:sender LEXICAL-MORPH
:receiver SYNTACTICAL

If an agent needs to ask one or more agents a specific question, the *ask-one* primitive is used, which allows the agent for waiting for the answer if it wants:

ASK-ONE
:content "(Find_word(the))"
:force REQUEST
:sender SYNTACTICAL
:receiver LEXICAL-MORPH

A message parser is used to split the incoming information through message passing, and to put it in specific slots which represent the mailbox structure. With this, each agent may call a message processing module that implements the KQML-based language. This representation is very useful if we consider the possibility of the inclusion of new agents in the society.

External Description. The agents acquaintances are represented by a data structure that aggregates goals, skills and identification of the other agents in the society, i.e. an external description of the agents, that is based on Sichman's proposal [8].

Therefore, the external description of agent i (indicated as ED_i) is composed by a set of external descriptions that agent i has about each agent j (indicated as $ED_i(j)$) in the same environment:

$$ED_i = \bigcup_{j=1}^{n} ED_i(j)$$

The external description that agent i has about an agent j ($ED_i(j)$) comprises the identification of agent j (indicated as $id_i(j)$), the set of its individual goals (indicated as $GOAL_i^I(j)$) and the capabilities of j (indicated as $CAP_i(j)$):

$$ED_i = \{id_i(j),\ GOAL_i^I(j),\ CAP_i(j)\}$$

The description, as known by agent i, of agent j individual goals ($GOAL_i^I(j)$) and capabilities ($CAP_i(j)$) is represented as:

$$GOAL_i^I(j) = \bigcup_{j=1}^{n} GOAL^I(j) \text{ and } CAP_i(j) = \bigcup_{j=1}^{n} CAP_i(j).$$

For example, let us consider the description of the semantical agent. Its external description of a subsociety which was already shaped before, will look like:

$ED_{(Semantical)}$(Lex-morph) = {Lex-morph,
{Find_Word(word, *features),
Find_next_Word(word, *features)},
{Find_Word(word, *features) = nil}};

$ED_{(Semantical)}$(Syntactic) = {Syntactic,
{SyntacticAnalysis (phrase) },
{SyntacticAnalysis (phrase) = *syntactic_struct}};

We have used these descriptions in order to design a subsociety based on a situation dependency degree among agents [9]. The criterion for this formation is based on additional information that an agent needs to reach its goal. Based on this information, which is provided by the external description, a dependency action is triggered through a request in a cooperative communication protocol.

Cooperation Protocol. Because of the agents heterogeneity and in order to make the agents interactions more dynamic, we use communication protocols in this model. These protocolos were also implemented using KQML primitives.

The introduction protocol performs the inclusion of an agent in the society through the message:

TELL
:content "<the agent's capabilities>"
:force INTRODUCE
:sender ANY-AGENT
:receiver BROADCAST

In order to solve conflicts among multiple competing hypotheses during the solution of a linguistic phenomenon, we use cooperative learning protocol [6], which provides suitable actions for this approach. However, this interaction does not promote an evaluation of multiple results for the same hypothesis in our society. In ambiguity solving, for instance, we have a group of agents contributing with partial heterogeneous results. Thus, the interaction must be a consensus among the solutions proposed by cooperative agents.

Task Decomposition. In this model, task decomposition is deeply attached to the domain of knowledge involved, so Linguistics. It is even difficult to analyse task decomposition separately from knowledge decomposition.

We have decided to group knowledge associated with the linguistic levels of analysis - lexical-morphological, syntactical and semantical. This gave rise to

a coarse-grained MAS architecture, where decomposition comes from the Linguistic domain itself. Inside those linguistic level agents we find the declarative knowledge - lexical-morphological, syntactical and semantical properties of the units worked - and also the procedural knowledge - methods and formalisms - associated with a given level of treatment. Agents are all different from each other. To this first architecture, we are aggregating new agents which are able to solve linguistic phenomena, our "secondary" agents.

A sentence to be analysed by the system follows a relatively sequential flow if it doesn't present any linguistic phenomenon (so, if it is very simple), in the sequence lexical-morphology $->$ syntax $->$ semantics. Agents interaction becomes more evident when a phenomenon is found, and that is the richness of such an architecture. For the lexical categorial ambiguity solving, for instance, the syntactical agent detects the problem, and asks the morphological and the semantical agents for special tasks to be done. So this is a kind of task decomposition that is strongly cognitive: it depends on the problem (or linguistic phenomenon) to be solved, and so does the negotiation that comes with the partial results.

3 The Linguistic Agents

The aim of our linguistic agents is to participate in a society with different skills, and, for the application worked, to collaborate in the interpretation of natural language sentences (utterances). The agents in this society are able to cooperate in achieving a common goal (through communicative acts), and also to solve their own individual goals.

Following, we will describe the linguistic agents implemented.

The Morphological Agent. The morphological agent provides a component that scans sentences and searches for a string (or for the first part of a string) matching that sentence (or a part of it) in a morphological dictionary. This agent is able to accept a sentence written in Portuguese and to split it into lexical items, returning, for each item, the set of different types (and morphological variables associated) it can represent.

The morphological dictionary constitutes the agent's declarative knowledge and it is built using a Trie tree structure (Fig. 2). This structure allows to decompose the input in basic components which are linked to morphological information groups. Items can be stored as forms or as bases, depending on the linguistic connotation to be given to a certain item. If the input is a form then it is stored as a complete string of characters; otherwise, it is a base and it is stored as an invariable part which is associated with terminations.

These descriptors point to models, which contain references to possible terminations for a base. The base, attached to its termination, shall constitute a valid Portuguese lexical item (word or expression). Besides referencing to a termination, models contain information on morphological variables as number, gender, etc.

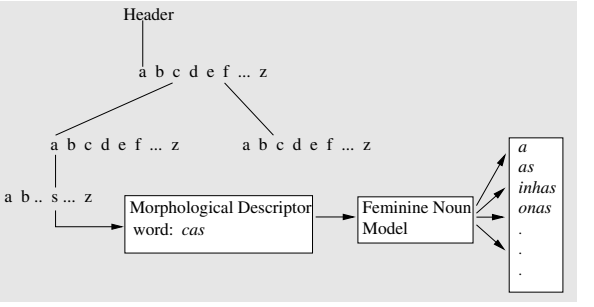


Fig. 2. Morphological Dictionary Structure

For example, the base *cas* has two descriptors - one for the model that builds the feminine noun *casa* (house) and its inflexions or derivates (plural *casas*, diminutive *casinhas*, etc), another one for the model that builds the verb *casar* (to marry) and its conjugated forms (about 56 inflexions in Portuguese).

In Portuguese language there are standard rules for changes in gender, number, degree, etc. With bases and models, it is possible to represent these changes by binding the base to the appropriate termination or groups which have identical inflexion behavior, using different models. In this way, we reduce the number of both characters and descriptors stored in the dictionary, avoiding information redundancy.

Our morphological entity, as an agent, can communicate either with applications or with other agents in the society. However, it works independently from the other agents. The morphological dictionary works as a function library on a structure set. This library was built to provide morphological information to be shared among several applications. In our proposal, the main library functions represent the agent procedural knowledge and the data structures (like the Trie tree) represent its declarative knowledge.

The Syntactical Agent. The syntactical agent is based on the Tree-Adjoining Grammars (TAG) introduced in [5], as a formalism for linguistic description. TAGs provide a way to connect the lexical items to grammar structures.

A TAG basic component is a finite set of elementary trees, each of which is a domain of locality, and can be viewed as a minimal linguistic structure. A TAG comprises two kinds of elementary trees: initial trees, which are complete structures, and auxiliary trees, that have at least one leaf symbol of the same category of the root symbol. Sentences are derived from the composition of the initial tree and any number of auxiliary trees by an operation called "adjunction". Adjunction inserts an auxiliary tree at one of the corresponding nodes of an elementary or a derived tree and moves the original root to the leaf of the same category. Recursion is provided by the auxiliary trees which can adjoin to themselves. Features, such as prepositions in verbal complements, can be associated with each node of any elementary tree.

The agent procedural knowledge is basically the syntactical analysis which splits a sentence in its basic syntactical structures using morphological knowledge. The grammar, composed by initial and auxiliary trees, together with a syntactical dictionary, constitutes the agent declarative knowledge. The aim of the syntactical dictionary is to provide information like features of verbs (e.g. complements).

The Semantical Agent. The semantical agent knowledge is based on the generative lexicon proposed by Pustejovsky [7]. The generative lexicon is characterized as a computational system involving at least four levels of representation: argument structure, event structure, qualia structure and lexical inheritance structure. The argument structure indicates the way a certain lexical item is associated with a syntactical expression. The event structure identifies a particular event type attached to a lexical item or expression. The Qualia structure has the essential attributes for an object, event and relations that define the lexical item. The inheritance structure establishes the relationships among a lexical item and other concepts in the lexicon.

The Qualia structure is a relational system that describes the semantics of lexical items. The elements that build Qualia structure include notions of the constituents, space, figure, surface, manufacture, etc, and are organized in four aspects of the meaning of a lexical item, each aspect referring to a role associated with its meaning. Those roles are known as *constituent, formal, telic and agent.*

Based on this proposal, it was implemented a semantical lexicon on a Trie tree which stores the lexical items (a lexical database which composes the semantical agent declarative knowledge). Semantical information associated with the item is stored in chained lists, in a structure called semantical descriptor.

The implementation was based on three main structures: the arguments, the events and the Qualia. Arguments and events use each one a list, and Qualia is represented using four different lists (one for each role in Pustejovsky's model).

The semantical lexicon is composed of a function library that acts on a lexical database. This library constitutes the agent procedural knowledge, which presents functions for manipulation and search of lexical meaning, as well as the operations called "generative devices" in Pustejovsky's theory (e.g. type coercion, selective binding, etc.).

4 Functional and Implementational Aspects

This model was implemented using a tool based on active objects, called DPSK+P [2], and a multi-agent execution environment, called MASENV [10], that runs on SUN workstations.

To this moment, we have already implemented the lexical-morphological, syntactical and semantical agents as linguistic ones, and the External Messages Manager agent (EMM) (Fig. 3). The first three ones implement the functionalities and knowledge described in the section 3. The EMM is responsible for the

society organization and for the management of the interface between the agents and the current application.

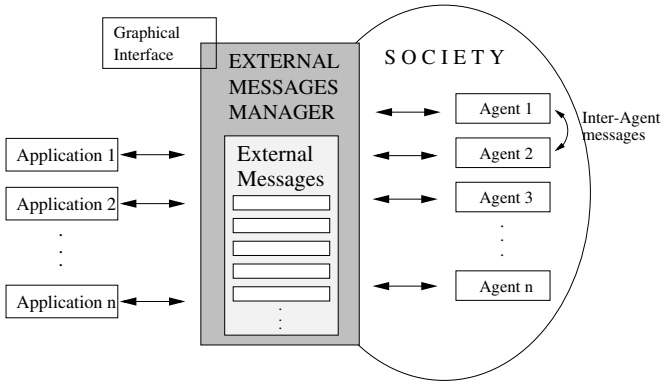


Fig. 3. The proposed society

When an application is executed, external to the system, a default message is sent to the society through the EMM. If the message is recognized by a certain agent (for instance, a syntactical analysis request is identified by a syntactical agent), a process is initialized by the society until it reaches a solution or it comes to an agrammaticality.

The EMM also feeds with information a graphical interface application. This interface is used to manage and to trace the behavior of the society, allowing for a graphical and textual view of all the events that it supports, such as message exchange, detection of unknown phenomena, external and internal services request, and so on.

Examples of applications under development are the resolution of possessive pronominal references, ellipses resolution and the categorical disambiguation.

5 Conclusion and Future Work

We have presented a generic architecture to NLP systems based on MAS. This proposal derives from previous work that has begun with studies on a subsociety to lexical disambiguating [11].

This work has been applied to interpret Portuguese language utterances through an heterogeneous linguistic information structure and a cooperative model. As a result, some less promising structures are discarded from the set of final possible solutions. The advantages concentrate on the economy of computational resources (time and space).

Further work is being conducted towards the following issues: use of this model in order to treat different linguistic phenomena (namely reference and ellipsis); additional tests to evaluate the behavior of the interactions through coop-

eration; investigation on a possible of dynamic grow of the society as lay creation of new agents with possible replication of existing ones; and, text rather than simple sentences as input for the society using, for instance, discourse agents.

This work has been carried on as part of NALAMAS project (CNPq/PROTEM - CC grant #680081/95-0).

References

1. Boissier, O., Demazeau, Y. A Distributed Artificial Intelligence View on General Purpose Vision Systems. Proceedings of Third European Workshop on Modelling Autonomous Agents in a Multi-Agent World, p. 311-330, Kaiserslautern, Elsevier Science Publishers B.V., August 5-7, 1991.
2. Cardozo, E., Sichman, J.S., Demazeau, Y. Using the active object model to implement multi-agent systems. Proceedings of Fifth IEEE International Conference on Tools with Artificial Intelligence, Boston, USA, 1993.
3. Finin, T., Fritzson, R. KQML as an agent communication language. Proceedings of Third International Conference on Information and Knowledge Management. ACM Press, November 1994.
4. Fuchs, C. Linguistique et Traitements Automatiques des Langues. Médicales Internationales, Hachette Université, 1993.
5. Joshi, A.K. How much context-sensitivity is necessary for characterizing structural descriptions: Tree adjoining grammar. In Dowty, D., Karttunen, L., Awickyy A.(Eds.), Natural Language Processing: Theorical, Computational and Psychological Perspectives. New York: Cambridge University Press, 1985.
6. Koning, J., Demazeau, Y. Collaborative Learning Using DAI Interaction Protocols in a Telecommunication Setting. ITP'95: Interacting Agents, Plodic, June 1995.
7. Pustejovsky, J. The Generative Lexicon. The MIT Press, Cambridge, Massachussetts, 1995.
8. Sichman, J.S. Du Raisonnement Social Chez les Agents: une Approche Fondée sur la Théorie de la Dépendance. Thèse de Doctorat, Institut National Polytechnique de Grenoble, France, 1995.
9. Sichman, J.S., Demazeau, Y. Exploiting social reasoning to deal with agency level inconsistency. In V. Lesser, editor, Proceedings of First International Conference on Multi-Agent Systems, pages 352-359, San Francisco, California, June 12-14 1995. AAAI Press/The MIT Press.
10. Sichman, J.S., Demazeau, Y., Boissier, O. When can Knowledge-based Systems be called Agents? Proceedings of IX Simpósio Brasileiro de Inteligência Artificial, RJ, p. 131, 1992.
11. Silva, J.L.T., Lima, V.L.S. A Lexical Categorical Disambiguation Using a Multi-Agent Systems Architecture. Proceedings of the Second International Conference on Recent Advances in Natural Language Processing, pages 330-335, Bulgary, Sept. 11-13 1997.
12. Stefanini, M.H. TALISMAN: une Architecture Multi-Agent pour l'Analyse du Franais Ecrit. These de Doctorat, Université Pierre Mendès-France, Grenoble - France, 1993.

A Massively Parallel Architecture for Natural Language Parsing - Some Results

Raul Sidnei Wazlawick[1] and José Gabriel Pereira Lopes[2]

[1] UFSC-CTC-INE Cx. P. 476 88040-900 Florianópolis, SC Brazil
Fax: +55 48 331-9770
raul@inf.ufsc.br
[2] UNL-FCT-DI Quinta da Torre 2825 Monte da Caparica, Portugal
Fax +351 1 295-4461
gpl@ssdi.di.fct.unl.pt

Abstract. This paper explores a new architecture for massively parallel natural language parsing implemented in Squeak Smalltalk. The *FastTalk* architecture is based in relatively simple operations done by actors associated with each possible syntactic category that each word might have in an input text. The FastTalk architecture differs from previous approaches by enabling the analysis of word strings in any order, and not only from left to right. Thus, this approach provides much more parallelism and additionally does not require time complex operations such as backtracking. The growth observed for the number of actors necessary to analyze a single sentence by a naive algorithm is drastically reduced by applying some heuristics derived from special characteristics of natural language. We claim that with this reduction it is possible to efficiently parse real texts and obtain every possible parse of an input.

1 Introduction

Many authors have explored parallel architectures for lexically driven natural language parsing. The Word Expert Parser [7], was probably the first attempt to associate active behavior to words in a text, such that their interaction could result in the analysis of the text. This approach has been strongly criticized, specially because it lacks some organizational principles and homogeneity [4].

A more recent approach is the ParseTalk architecture [1]. ParseTalk is based on object-oriented framework and dependency grammars. In that architecture, the objects associated to each word (*word-actors*) should send messages to each other in order to find their respective *heads* (a head is a word that is modified by another word. For example, the noun "cat" could be a head in a text for a number of modifiers, such as "beautiful", "the", "my", "white", etc.).

However, ParseTalk and the previous known approaches, are very restrictive with respect to parallelism. In ParseTalk the words are analyzed from left to right in the text. When a word-actor is activated, it must find a head among some of the previous words. For the sentence "Zenon sells this printer over-priced" [2], when the word-actor associated to the word "overpriced" is activated, it is

F. Moreira de Oliveira (Ed.): SBIA'98, LNAI 1515, pp. 93-100, 1998.

assumed that the sentence has been analyzed until that point, that is, a dependency structure for "Zenon sells this printer" has already been constructed. Then the word "overpriced" will send asynchronous messages to the actors associated with the words "printer" and "sells" in order to verify if one of them (or both) could be a valid head. Only one word-actor is enabled to search a head at a time. Every other word-actors may just wait to be searched.

This way of doing the analysis is restrictive because the parallelism exists only within the context of the analysis of one single word. This means that the text is not analyzed in parallel, because only one word is analyzed at a time.

Furthermore, the protocols for doing that search seem to be sometimes quite complex. Consider the sentence "The customer bought the silver notebook", when the fragment "The customer bought the silver" has been analyzed, the word "silver" could have been interpreted as a noun and attached to the verb "bought". But "silver", in this context, is an adjective and must be attached to the word "notebook". This is detected only when the word-actor for "notebook" is activated and does not find a corresponding head among the previous words. That starts a *backtracking protocol*, where the word "notebook" will try to find a head in the historically previous containers that hold the parse history [5]. If it is successful, then the remaining word "silver" can be correctly attached. ParseTalk architecture proposes to limit the backtracking search space, so that some sentences would not be correctly analyzed, trading completeness for efficiency [5].

The left-to-right nature of ParseTalk also implies the existence of a *prediction protocol* [2], that is invoked when it is clear that the active word must be linked to a word that is placed at its right in the text. That happens in the analysis of the sentence "Zenon sells a printer". When the actor for "a" is activated it knows that it will be linked to a word at its right, but it does not know which word it is. Then it constructs a *virtual word* that will be instantiated latter in the analysis. However, as Hahn et all said "a lot of false predictions may arise and may dramatically increase the number of ambiguous analyses." [2]. As far as we know, this problem has been only partially solved by restricting the search space by using word class hierarchy.

We aim at building a highly parallel architecture to parse natural language texts, using relatively simple operations that can eventually be efficiently performed by dedicated parallel machines. This architecture (FastTalk) is designed in a way that resembles ParseTalk (actor-based and making use of dependency grammars), but it does not require prediction or backtracking protocols. Furthermore, it enables much more parallelism, because any number word-actors can search a head at the same time.

2 Brief Description of FastTalk

Given a word, and its entries in the lexicon (one for each syntactic category the word may be associated to), FastTalk creates one word-actor for each possible entry. Word-actors will compete among themselves to construct the syntactic

structures of the input. Eventually, only word-actors that represent the correct syntactic categories of words in the context where they appear in the input will have success in this competition.

FastTalk is based on a dependency grammar [3], and is designed to work with Portuguese language. However, it could be prepared for other languages. The syntactic structures are represented by partial-order dependency relations between word-actors. If a word-actor B is *dependent* of A (represented by $B \rightarrow A$) then A is the *head* of B, and B is a *modifier* of A.

The *transitive closure of the dependency relation* is represented by $\rightarrow^+$. Then, $A \rightarrow^+ B$ iff $A \rightarrow B$ or $(\exists C) A \rightarrow C \rightarrow^+ B$. The *transitive and reflexive closure of the dependency relation* is represented by $\rightarrow^*$. Then, $A \rightarrow^* B$ iff $A = B$ or $A \rightarrow^+ B$.

A *word-actor* is an autonomous entity with the following characteristics (implemented as instance variables): (1) a *name*, that is, the associated word; (2) a natural number *pos*, representing the position of the associated word in the input text; (3) a natural number *cop*, that is used to distinguish an actor from its copies (the original actor has *cop*=0; its first copy has *cop*=1, its second copy has *cop*=2, etc.); (4) a set of *valencies for the word*, including its syntactic category, agreement information, positions where the modifiers may appear relatively to the word, and semantic concept(s) associated to; (5) a set of *dependency roles* with a name (det, subj, obj, etc.) and a set of valencies for each role; and (6) a set of *assigned dependents*, or actual modifiers, that is, the set of word-actors that have this actor as head.

None of the above characteristics change during the existence of a word-actor.

A word-actor will be represented by: $name_{pos,cop}$. The dependency relation will be also represented, for simplification purposes, by a parenthetic notation. Then, $A_{i,j} \rightarrow B_{k,l}$ will be represented by $B_{k,l}(A_{i,j})$. If additionally $B_{k,l}$ has another dependent $C_{p,q}$, the whole dependency structure will be represented by $B_{k,l}(A_{i,j}, C_{p,q})$. The names associated to each dependency role are not referenced in this paper for simplicity.

A word-actor may respond to asynchronous SEARCHHEAD messages comparing the valencies of one of its dependency roles with the valencies of the sender. If the valencies are compatible, and the respective role is free in the receiver (head), then a dependency link may be established. However, this link is never built between the sender and the receiver, but between copies of them. For example, if the actor $A_{1,0}$ send a SEARCHHEAD message to the actor $B_{2,0}$, and the valencies are compatible, then one copy of each actor is created, and the link is established between those copies, generating the structure $B_{2,1}(A_{1,1})$. If an actor that is to be copied belongs to some dependency structure, then the whole structure is copied.

The copying overhead is compensated by the simplicity of the operations involved in the analysis. That is, like in a RISC architecture, we do more operations, but simpler ones. Moreover, this read-only characteristic of the actors enables high parallelism, because the interaction among the actors does not interfere with their characteristics (instance variables), and the synchronization

may be reduced to a minimum.

In this process of search, some actors will never find a head. Those actors may be destroyed in the end of the analysis, or as soon as they are detected, by sending them a KILLYOURSELF message. When an actor receives that message, it destroys itself and its dependents.

3 Neighbors and Prospects

The *left border* of a word-actor $A_{i,j}$ is a word-actor $B_{k,l}$ such that $B_{k,l} \rightarrow^* A_{i,j}$, and there is no $C_{p,q}$ such that $C_{p,q} \rightarrow^* A_{i,j}$, and $p < k$. The *right border* of a word-actor $A_{i,j}$ is a word-actor $B_{k,l}$ such that $B_{k,l} \rightarrow^* A_{i,j}$, and there is no $C_{p,q}$ such that $C_{p,q} \rightarrow^* A_{i,j}$, and $p > k$.

A word-actor $A_{i,j}$ with right border $B_{k,l}$ is a *left neighbor* of an actor $C_{p,q}$ with left border $D_{r,s}$ iff $k + 1 = r$. A word-actor $A_{i,j}$ with left border $B_{k,l}$ is a *right neighbor* of an actor $C_{p,q}$ with right border $D_{r,s}$ iff $k = r + 1$.

A *headless actor* is an actor $A_{i,j}$ that does not depend of any other actor, that is, there is no actor $B_{k,l}$ such that $A_{i,j} \rightarrow B_{k,l}$. Only headless actors may send SEARCHHEAD messages.

When a headless actor $A_{i,j}$ receives an ACTIVATE asynchronous message, it sends asynchronous SEARCHHEAD messages to its prospects. An actor $B_{k,l}$ is a *prospect* for an actor $A_{i,j}$ iff the following conditions hold simultaneously: (1) $B_{k,l}$ must be a neighbor of $A_{i,j}$; (2) $A_{i,j}$ must *be attracted by* $B_{k,l}$ (the relation "*be attracted by*" is defined between word categories. For example, determiners and adjectives are attracted by nouns; nouns are attracted by prepositions and verbs, prepositions are attracted by nouns and verbs, etc.); and (3) $A_{i,j}$ must not have had send a SEARCHHEAD message to $B_{k,l}$ yet.

The first condition assures that the messages will be send only locally. That means that an actor will not send a message to a great number of actors, but only to those that are its neighbors. This condition, however, is not restrictive when the head is not contiguous to its modifier. For example, consider the input "the nice cat sleeps". The modifier "the" is not contiguous to the head "cat", but it can be attached. Initially, the actor $the_{1,0}$ have no prospects to send a SEARCHHEAD message, because its only neighbor ($nice_{2,0}$) is an adjective, and determiners are not attracted by adjectives. Thus, the actor $the_{1,0}$ will not send any message until some modification in the neighborhood justifies it. Eventually, the actor $nice_{2,0}$ will send a SEARCHHEAD message to $cat_{3,0}$ and the structure $cat_{3,1}(nice_{2,1})$ is created. When the actor $cat_{3,1}$ is created, it informs this fact to its neighbors $the_{1,0}$ and $sleeps_{4,0}$. This means that the actor $the_{1,0}$ will try to send a message again to its prospects. At this time, this actor does have a prospect: $cat_{3,1}$.

The second condition is useful to avoid sending innocuous messages. For example, a verb does not need to send a SEARCHHEAD message to a determiner, and so on.

The third condition assures that actor $A_{i,j}$ will send a SEARCHHEAD message to an actor $B_{k,l}$ only once. That makes sense because the characteristics of the

actors does not change with time. This condition can be used to proof that the searching protocol stops for any sentence of finite size.

A headless actor $A_{i,j}$ is activated in two situations:

1. when $A_{i,j}$ is created (be it a copy or the original actor);
2. when a new dependency structure is created and some actor in this structure is neighbor of $A_{i,j}$.

In the FastTalk architecture, every asynchronous message is placed in a queue. The messages wait in this queue until a processor is available to perform it. Thus, the number of available processors has a great influence in the performance of the system.

It can happen that an actor is neighbor of two or more new dependency structures created almost simultaneously. In this case, this actor could receive two or more ACTIVATE messages. But, in this case, the queue maintains only the oldest one, because when the actor is activated, it search all prospects and not only the sender of the ACTIVATE message. Each message is removed from the queue immediately before it is performed. Then if a new change occurs while the actor is processing the ACTIVATE message, it is enabled to search the neighborhood again.

4 A Short Example

Consider the sentence "the nice cat sleeps", with no syntactically ambiguous words. As every actor is activated at the same time and run in parallel, the order of the construction of the syntactic structures of this sentence is mostly unpredictable.

There is no space for describing the whole process, but when the system stops, the following structures would exist:

1. The original actors: $the_{1,0}$, $nice_{2,0}$, $cat_{3,0}$, and $sleeps_{4,0}$;
2. Some intermediary partial structures: $cat_{3,1}(nice_{2,1})$, $sleeps_{4,1}(cat_{3,2})$, $cat_{3,3}(the_{1,1}, nice_{2,2})$, and $sleeps_{4,2}(cat_{3,4}(nice_{2,3}))$; and
3. The complete parse of the sentence: $sleeps_{4,3}(cat_{3,5}(the_{1,2}, nice_{2,4}))$.

At this time, a KILLYOURSELF message can be sent to every actor except that/those which represent the complete parse of the sentence.

As this proliferation of intermediary partial structures is the main source of inefficiency of this kind of algorithm, we are trying to reduce it to a minimum, using some heuristics that are discussed in the following section.

5 Some Heuristics

In order to obtain better results, some heuristics concerning the characteristics of natural language were used.

5.1 Word barriers

A *word barrier* is a set of syntactic categories such that words associated to them may not be activated and do not respond to SEARCHHEAD messages during a given stage of the analysis. The analysis with word barriers has two stages. In the first stage the word barrier set consists in the following categories: verb, preposition, adverb, comma and period. At this stage, only actors belonging to other syntactic classes may search (or be searched) for a head. As result, only non-recursive NP's (a NP that does not have another NP in its dependency structure) are built. The PP's and VP's are not built at this stage. Then, when the system stops, every partially built NP structure is destroyed, reducing the number of actors that survive to the next stage. In the second stage, the barrier is composed only by the period, and every headless actor is reactivated. Then, the verbs and prepositions will be combined directly with complete NP's, instead of partially built ones. In the example given in section 4, this heuristic would have avoided the creation of the intermediary structures $sleeps_{4,1}(cat_{3,2})$, and $sleeps_{4,2}(cat_{3,4}(nice_{2,3}))$. This may not have much significance for small sentences, but when we consider big sentences with lots of verbs and prepositions, the results are impressive.

5.2 Necessary Roles

The heuristic of *necessary roles* (NR) consists of do not allowing some words to search a head unless they have already one or more specific roles filled.

For example, in Portuguese a preposition may never appear in a sentence without governing a NP. And this NP must appear always immediately at the right of the preposition. Thus, sentences where the preposition appears as in "whom did john give the book to?" are never used in Portuguese.

Prepositions, conjunctions, relative pronouns and commas, must have always a complement attached in order to search for a head. Verbs, nouns and adjective that subcategorize NP's and/or PP's, must have those complements attached in order to search a head. For example, many intransitive verbs may search a head only if the subject role is filled. Bi-transitive verbs of category VERBC123 , may search a head only after the subject, direct object and indirect object are attached.

5.3 Incremental elimination of substructures

If a word-actor discovers that it is not productive anymore, than it can send a KILLYOURSELF message to itself. However, the detection of such situations can be sometimes tricky. For example, at a first glance, an article could be destroyed after a copy of it is attached to a noun. For instance, $the_{1,0}$ could be destroyed after the creation of a structure like $mexican_{2,1}(the_{1,1})$. However, this is not valid if the input sentence is "the Mexican airplane...". But if the original actor is destroyed after attached to "Mexican", then a backtracking process would be necessary to return to the previous situation. We chose to avoid this

backtracking process by applying this heuristic only when the words involved are not syntactically ambiguous.

5.4 Pos-Tagging

Some ambiguities may be eliminated with pos-tagging techniques. For example, the word "a", in Portuguese can be either article, pronoun, or preposition, depending on the context. However, if the word "a" is immediately on the left of an article, than "a" can not be an article itself. In this case, the word-actor associated to the article "a" may be destroyed. A good set of such rules for Portuguese ambiguous words is described by [6]. This heuristic combined with the incremental elimination of substructures is very powerful to eliminate non-productive actors before they start creating non-productive partial structures.

6 Discussion

The fact that a word-actor never changes its characteristics, has enabled us to implement an architecture where SEARCHHEAD messages can be executed asynchronously by all active actors at the same time, and not only by one actor as in previous approaches.

The heuristics that are being used have reduced significantly the number of actors necessary to parse a sentence, so that it does not reach a prohibitive amount of time and space for real-world texts.

The next table shows a comparison among five algorithms incrementally including each of the heuristics. The first algorithm (N) is the naive approach. The second (WB) uses only the heuristics of word barriers. The third algorithm (NR) uses word barriers and necessary roles. The fourth algorithm (IES) uses word barriers, necessary roles and incremental elimination of substructures. The last algorithm uses all heuristics.

We have measured the number of agents created (measure of space), and also the number of sends of SEARCHHEAD messages (measure of time). The real time elapsed is also measured, but as the prototype run on an interpreted language, it has significance only relatively to each algorithm. Those times may be heavily reduced by using a compiled and optimized language.

This experiment shows the results of the analysis of a medium size 30-word sentence, taken from the web. Many other examples may be found in [8].

	N	WB	NR	IES	PT
actors created	79.219	16.653	8476	1.490	679
searchHead sent	15.007	2.877	1.309	319	128
time (sec.)	6.489	877	340	46	17

The number of actors is exponentially related only to the size (complexity) of a single sentence. But it is linearly related to the number of sentences in a text. If we assume that the average size of the sentences does not grow with the size of

a text, then we can conclude that the number of actors will grow linearly when compared to the number of words in a large text. This means that if a text with 100.000 words takes k seconds to be analyzed, then a text with 200.000 words would take only $2k$ seconds to be analyzed. Also, as the activation messages are sent to every word in the text at the same time and queued, this means that, for large texts, when the number of processors is duplicated, the time of analysis is reduced to a half.

Some phenomena are still under consideration. That includes handling discontinuous structures, unknown words, and a preference model to produce in the first place the most probable parsing if few processors are available. Reports on those subjects are left for future papers.

References

1. Hahn, U., Schacht, S., Bröker, N.: Concurrent, Object-Oriented Dependency Parsing: The ParseTalk Model. *International Journal on Man-Machine Studies* 41 (1994) 179-222
2. Hahn, U., Neuhaus, P. and Broeker, N.: Message-Passing Protocols for Real-World Parsing – An Object-Oriented Model and its Preliminary Evaluation. *Proc. Int'l Workshop on Parsing Technology.* (1997) 101-112
3. Heringer, H.J., Lima, J.P.: *Palavra Puxa Palavra - Comunicação e Gramática Dependência.* ICALP: Lisboa, Portugal (1987)
4. Hirst, G.: Semantic Interpretation and Ambiguity *Artificial Intelligence.* 34 (1988) 131-177
5. Neuhaus, P., Hahn, U.: Trading off Completeness for Efficiency - The ParseTalk Performance Grammar Approach to Real-World Text Parsing. *IX Florida Artificial Intelligence Research Symposium.* Key West, U.S.A. (1996) 60-65
6. Pacheco, H.C.F.: *Uma Ferramenta de Auxílio à Redação.* Doc. Thesis. UFMG. (1996) 49pp.
7. Rieger, C., Small, S.: Toward a Theory of Distributed Word Expert Natural Language Parsing. *IEEE Transactions on Systems, Man and Cybernetics.* SMC-11 (1981) 43-51
8. Wazlawick, R. S.: *FastTalk: An Actor-Based Approach to Natural Language Parsing. Technical Report on a Post-Doctoral Research.* Lisboa, Portugal, 1998.

Planning and Learning: Put the User in the Loop

José Luís Ferreira and Ernesto Jorge Costa

Dept. Engenharia Informática da Universidade de Coimbra
Pólo II – Pinhal de Marrocos
3030 Coimbra - Portugal
{zeluis, ernesto}@dei.uc.pt

Abstract. Integrating different aspects of learning in planning systems has been one of the most promising trends towards solving the problem of complexity and scalability that affects the planning systems. Most of the times, the user has been put aside this integration, as most approaches consider the existence of perfect models of the domain to act as an oracle for the ability of the system to learn, either by observation or practice. We have decided to put the user in the learning loop, in a mixed initiative approach to online learning. The user will propose problems and guide the system through the space of possible alternatives to solve it, therefore guiding learning. To generate learning episodes, the system also performs its own experimentation, relying on the user to classify the resulting scenarios, allowing some otherwise impossible learning approaches. The user is not a perfect model, so mistakes can occur, during this interaction. We have to be prepared to cope with these imperfections, and recover from user induced errors.

1. Introduction

Intelligent planning is a knowledge intensive task, and most planners need a complete knowledge model of the world they are supposed to interact with. Building such domain knowledge is a complex task, in itself, and learning can be used in many different ways as an attempt to solve this problem. Many learning systems can learn some form of control knowledge [1, 10, 11], and compile experience into directly usable knowledge. Other systems can learn the action models, based on the existence of a perfect domain model, as we can find within the Prodigy research group [2, 15].

Most systems don't rely on directly interacting with a user. Although this is recognised as an important research subject, not many researchers have focused on this particular subject. Instructo-Soar [6], is built over Soar [9], and its purpose is to teach the system new procedures, based in existing primitive actions, using what they call flexible instruction, in natural language. A fundamental difference concerns the learning purposes. We want our system to learn the primitive actions that Instructo-Soar is supposed to know already. Our system knows nothing, at the beginning. We are specifically interested in acquiring the action models, by interacting with the expert at the knowledge level, in a mixed initiative strategy. The interaction refers the

F. Moreira de Oliveira (Ed.): SBIA'98, LNAI 1515, pp. 101-110, 1998

objects of the world, and the knowledge acquisition is based in the system's ability to acquire general usable operators from specific instructions and other contextually situated dialogs. This allows incremental acquisition of knowledge, either through assisted experimentation, by planning or by observing the expert's solutions. We include some specific tools for assisted debugging and pay some attention to the problem of imperfect interaction, with the purpose of allowing the revision of the solutions and correction of the acquired knowledge.

Our research concerns other aspects of the integration of learning and planning. To give a general idea of the system as a whole, we first give a general perspective of our interests in integrating learning and planning. We then focus on the particular problem of acquiring the operators. Limitations, achievements, and our perspectives to continue with the system's development are also discussed. We conclude with an overview of some other approaches and their differences and similarities to ours. Finally, the work to be done is presented.

2. A General Framework for Learning and Planning

A plan is a set of actions that, if executed in some defined order, transforms the current state of the world into one that satisfies the defined planning goals. Planning involves knowledge about the possible actions, the objects of the world, their mutual relations and restrictions, interactions among each set of possible interacting actions. Previous failures and successes can also be used to guide search for solutions. In this context, we can see several possibilities of learning, either to acquire domain knowledge or to be used throughout the planner's actual application, to improve domain and control knowledge through experience. Leaning and planning can be combined in several different situations: learning action preferences; determining better goal ordering; deciding on whether or not to apply an operator (execute an action) or first try to solve an unsolved sub-problem; choosing the best objects to manipulate first. Experience can provide many different insights into the problem. Learning macro-operators [3], domain constraints [7], control rules [1], operator refinement [15], are some of the aspects that have been approached to find better plans. Instructo-Soar [6] in fact learns what is most known as macro-operators, although they refer to them as new operators. It also learns control rules, that specify preference or rejection criteria for an operator in certain scenarios.

Our work concerns several of these aspects, as we describe in [5]. We particularly focus on the possibilities of interacting with an expert in the context of solving planning problems, to acquire most of the knowledge the system needs. By observing how the user solves the problems, by asking the correct questions and by assisted experimentation, the system can aggregate different types of knowledge that will eventually model the domain and allow it to efficiently solve any future problem. The system acquires domain knowledge, in the form of operator description and constraints between attribute coexistence. It also acquires macro-operators, that can be used to implement goal ordering and preference. The macro-operators define a hierarchy, built on the basis of the dependencies among the individual actions in a plan, and also on

the achievement of partial goals. Efficient use of this hierarchy can reduce the search for solutions to new problems, as experience can be brought to use and drive the search process to faster and better solutions. The idea of a hierarchical decomposition of the whole problem into sets of co-operative actions, gave rise to an interesting approach to case based reasoning that produces creative solutions based in known ones, that has been successfully applied in the domain of musical creation [12].

3. The Basic Assumptions

A problem consists of an initial state and the planning goals. The initial state is described by three subsets of attributes: the enumeration of the objects, the static or unchangeable domain attributes and the changeable attributes. The system is supposed to have no knowledge of the domain at the beginning. Nevertheless, while it is used, the operators will be acquired so, for the sake of completeness, the system can begin with some initial operators, but it acts as if they can be incorrectly described. The description of a state must contain every verified attribute, and no negated ones. The initial state therefore needs to be completely described. The operators are described by preconditions and post-conditions, but also by conditional effects, defining what we call side contexts. Operators can have multiple side contexts, and each of them is independent of each other. A side context, as its name suggests, needs not be verified in every situation of operator applicability. However, if verified, it will necessarily be taken into account for the effects of the operator. Attributes in the effects are negated if they become false after the execution of the action.

The system learns by observing the proposed solutions, but also by experimentation, which results in a more effective interaction for learning purposes. To cope with the user's imperfections, we assume that the interaction is normally correct. Whenever something suggests an error, the user is asked for confirmation, and the convenient correction is made, if necessary. The user is intended to instruct the system in a step by step approach, from the initial to the final state. So, interaction between the system and the user is always situated in some particular context, and instructions refer to the current and the resulting state. Whenever no operator can be applied, or the applicable ones are useless, regarding planning goals, the user is asked to give the best action to be taken in the current state. In such cases, a new operator is acquired, or an existing one is refined.

4. Brief Summary of the Learning Strategy

The approach to acquiring an operator's description is based in the way an operator is used. When an operator is applied, its preconditions must all be true in the current state. Some of these conditions will be destroyed by the execution of the action, which justifies their inclusion as negated effects. The positive effects are introduced in the new state after execution of action. Some attributes are not changed by the action, therefore appear in both the current and the new states. Now consider we are trying to

learn the operator's description, based on a particular episode of action execution. We can use a kind of abduction reasoning by just thinking in the reverse direction. The episode refers the current state and the new state, and also the particular objects involved in the action. We can therefore easily obtain the positive and negative effects, determining the differences between the new state and the current state, and vice-versa, respectively. The negated effects must be considered preconditions of the operator, because we can not destroy something that doesn't exist. Other conditions are acquired by looking into the current state and determining what are the attributes that refer the objects involved both in action and in the effects. So, the operator's description in that particular context is built. Of course, by doing so, we can pick some context dependent attributes, others that don't really matter for the operator, and in general, we are limited by the particular application context. As a result of this kind of reasoning, we observe that the first description of an operator normally contains several inaccuracies:

- preconditions are either over specific or over general: they do not cope with exceptions, they refer attributes that do not matter, they include side contexts;
- the episode is normally too context dependent; even if side contexts apply, a single episode can not guarantee the separation of normal contexts from side contexts;
- the learning context is over specific;

We will show how the different forms of operator refinement take place, with the help of a trace of a session with the system. The example concerns the widely known problem of the towers of Hanoi, which we describe as:

domain objects: ((peg a)(peg b)(peg c)(hand h)(disk d1)(disk d2)(disk d3))
unchangeable attributes: ((lt d1 d2)(lt d2 d3)(lt d1 d3)) ;;lt = smaller
changeable attributes: ((at d1 a)(at d2 a)(at d3 a)(top a d1)(on d1 d2)(on d2 d3)(empty h))
planning goals: ((at d1 c)(at d2 c)(at d3 c))

Fig. 1. The problem description

5. Different Aspects of Learning: A Practical Session

The only thing the system knows, when it is first presented the problem, is the description shown in Fig. 1. The user is therefore asked for the good action.

this is current state: (empty h)(on d2 d3)(on d1 d2)(top a d1)(at d3 a)(at d2 a)(at d1 a)
what is the best action now?*(pick d1 a h)*
what is the purpose of using (pick d1 a h)?
((holding h d1)(not(empty h))(not (at d1 a))(not (on d1 d2))(not (top a d1))(top a d2))
after applying (pick d1 a h), we achieve:
(holding h d1)(top a d2)(on d2 d3)(at d3 a)(at d2 a)
is it correct? [y/*]:*y*

Once the definition of the resulting state is accepted by the user, the system can learn the description of the operator. It is normally incomplete, and needs to be refined. Future application of the operator will be used to achieve the necessary refinement. The first description of the operator PICK is:

action: (pick x3 x4 x5)
conditions: (hand x5)(disk x3)(peg x4)(at x3 x4)(top x4 x3)(empty x5)
side conditions: (disk x2)(on x3 x2)(at x2 x4)(lt x3 x2)
post conditions: (holding x5 x3)(not (empty x5))(not (at x3 x4))(not (top x4 x3))
side effects: (not (on x3 x2))(top x4 x2)
learning context: (disk x1)(lt x3 x1)(at x1 x4)

Fig. 2. An operator description

In Fig. 2, we can see the description of the learned operator. It has been acquired from a single application example, yet it is correctly described. The only unnecessary component that the operator keeps is the learning context that remembers the fact that there was another disk at the original peg, bigger than the one that was picked. The learning context refers some characteristics describing the learning episodes that can later be used to assist in the operator's refinement, particularly in the case of an over general first description of the operator. Also notice that the operator correctly contains a side context (side conditions and side effects), although this particular side context is verified in the learning situation. Due to the system's learning bias: the objects described in the action (D1, A and H, in the example), define a learning focus, all other objects and their properties will be first considered as defining side contexts. Side contexts are used in the operator's application, if they are verified, while learning context is not, therefore learning context is just a way of remembering the circumstances of operator acquisition. Operator refinement through specialisation uses side contexts and learning context, moving them to the operator's preconditions.

6. Acquiring Domain Constraints

When the system learns a new operator, it also acquires domain constraints that will be used in the learning phases that follow. Domain constraints are acquired by looking into the changes that occur in the world when one operator is applied. If two attributes refer the same objects, and one becomes true when the other becomes false, then it is assumed that they cannot occur simultaneously. They are said to be in mutual contradiction [5]. Similarly, although symmetrically, Kelleher [7] learns relations concerning attribute coexistence. Such constraints can not be proven to be true, yet our experience shows that our models usually refer such pairs of attributes. Examples are (empty, full), (occupied, free), and many others. Their use has shown to improve the quality of the interaction between user and system, allowing to infer changes in the world, when others are known. These learned constraints facilitate the definition of changes that occur when an action is executed. The user will validate their use, by

accepting or not the proposed state, allowing the system to upgrade its beliefs concerning the current set of constraints. In the example, the acquired constraints are:
- being empty contradicts hold something: ((holding x_5 x_3), (empty x_5))
- being held contradicts being at some peg: ((holding x_5 x_3), (at x_3 x_4))
- being held contradicts being on top of any peg: ((holding x_5 x_3), (top x_4 x_3))

Now the system knows how to pick objects, but the hand holds disk d1. So, there is nothing the system can do now. The user is once again asked for assistance.

what is the best action now? (put d1 c h)
after applying (put d1 c h), we achieve:
(empty h)(at d1 c)(top a d2)(on d2 d3)(at d3 a)(at d2 a)
is it correct? [y/*]:*n*
describe the changes to the state *((top c d1))*
after applying (put d1 c h), we achieve:
(top c d1)(empty h)(at d1 c)(top a d2)(on d2 d3)(at d3 a)(at d2 a)
is it correct? [y/*]:*y*

The system always tries to explain the purposes of executing a particular action given by the user. It looks into the goals and tries to find a justification for some action to take place. In the case of the previous step, it assumes that one of the goals, (AT D1 C), is a good justification for the use of putting the disk D1 in Peg C. This is partially incorrect, because this goal is not to be achieved now. However, this allows the system to reason something more, based in the constraints that it acquired in the first step. Because the disk will be in a peg, it will not be hold, if the acquired constraints are correct. By the same reasoning, if nothing is hold by the hand, then it is empty. And the state is proposed. Unfortunately, the system still doesn't know that when it puts some disk down, it will be the topmost one in the peg. The user must make the adequate corrections, and the first description of the operator put is acquired. It is far from being correct. But future use will allow its refinement.

action: (put x8 x9 x10)
conditions: (hand x10)(disk x8)(peg x9)(holding x10 x8)
post conditions: (top x9 x8)(not (holding x10 x8))(empty x10)(at x8 x9)
learning context: (disk x7)(disk x6)(lt x8 x7)(lt x8 x6)

Fig. 3.. An incompletely described operator

7. Experimentation: The Key to Operator Refinement

Experimentation occurs whenever the system tries to apply the operator in some context. This generates negative application episodes, which the user would never propose. In every episode, the system applies the operator to the current state, given its current description, and asks the user to classify the result as:
- correctly applied, meaning the operator is applicable and its effects are correct;
- incorrectly applied, meaning the effects needs to be corrected;
- not applicable, which means the operator can not be applied, and has to be refined;

The user will guide the system throughout the solution plan, until the goals are reached. As interaction proceeds, the operators will normally be refined, and eventually corrected.

Continuing with our example, the system proposes to pick disks d1 or d2, and the user indicates to pick d2. Then, it tries to put d2 on d1.

---

can I apply (put d2 c h) and obtain:
(at d2 c)(empty h)(top c d2)(top a d3)(at d1 c)(top c d1)(at d3 a)
is it correct?[y/n/c/b/?]:*n*

This is incorrect, because disk d2 can not be put over disk d1. The operator needs to be specialised, since it is over general. Specialisation of the operator can be achieved by inclusion of facts in the preconditions that prevent the operator from being applicable in the current state.

The system tries to find these conditions, by looking into the learning context, the side contexts or the state that results from the operator's application, in this order. If something exists in the learning context that disables the application of the operator in the current situation, the corresponding context is moved to the preconditions of the operator, and the operator will therefore no longer be applicable. Remember that the learning context refers to the previous application contexts of the operator. Otherwise, if there is any side context that is not verified, its inclusion in the preconditions of the operator will also inhibit the application of the operator. For that to happen, the particular side context would have to be verified in each previous application of the operator, allowing it to be promoted to normal application context.

If none of the previous solutions can be used, which is the case here, the system will try to justify the non applicability of the operator by looking into the effects of the operator, therefore in the next state. If something exists in the following state that prevents the operator from being applied, the system defines a negated context that, if verified, inhibits the operator's application. Once again, the user's assistance is the key to the solution.

---

were it applicable, would the state be correct?[y/*]:*n*
describe the changes: *((not (top c d1))(on d2 d1))*
this would be the state:
(on d2 d1)(at d2 c)(empty h)(top c d2)(top a d3)(at d1 c)(at d3 a)
is it correct?[y/*]:*y*

The operator is refined, to include the information resulting from the interaction with the user, by addition of a negative context.

action: (put x14 x15 x16)
conditions: (disk x14)(peg x15) (hand x16) (holding x16 x14)
(not (disk x13) (top x15 x13)(at x13 x15)(lt x13 x14))
post conditions: (empty x16)(top x15 x14)(at x14 x15)(not (holding x16 x14))
learning context: (disk x12)(disk x11)(lt x14 x12)(lt x14 x11)

Fig. 4. Learning a negative context

If the problem is not in the next state, the conditions for the non applicability of the operator have to be found in the current one. Two minimal solutions are used to cor-

rect the operator: either by adding a condition that is not verified in the current state but should, or by adding a negated condition that is verified in the current state. For this purpose, the system considers potential positive contexts, anything that consistently appears in all previous contexts but not in the current state, and potential negative contexts everything that is verified in the current context and has not been in the past. If no previous contexts are known, the system will have an empty set of potential positive additions and a set of potential negative conditions consisting of each attribute verified in the current context that does not belong to the current operator preconditions. The user is asked to give a solution, which will be checked by the system. If the given solution solves the applicability error, then it will be added to the operator's description. Since the operator has been corrected, its application can be reconsidered. If there are other possibilities to apply the operator that do not conflict with the new description, they are proposed. Once again, the system uses all possible situations to experiment new applications of the known operators. The results are worthy, as in the situation that follows.

can I apply (put d2 a h) and obtain:
(at d2 a)(empty h)(top a d2)(top a d3)(at d1 c)(top c d1)(at d3 a)
is it correct?[y/n/c/b/?]:*c* ;;the operator is applicable, but its effects are not correct
describe the changes to the state *((not (top a d3))(on d2 d3))*
after applying (put d2 a h),we achieve:
(on d2 d3)(at d2 a)(empty h)(top a d2)(at d1 c)(top c d1)(at d3 a)
is it correct? [y/*]:*y*

Now, the operator is applicable, but the effects are not correct. The user is asked for the changes, and a side context is learned, as follows.

action: (put x20 x21 x22)
conditions: (hand x22)(disk x20)(peg x21) (holding x22 x20)
(not (at x19 x21) (top x21 x19) (lt x19 x20) (disk x19))
side conditions: (top x21 x17)(at x17 x21)(lt x20 x17)(disk x17)
post conditions:(top x21 x20)(empty x22)(at x20 x21)(not (holding x22 x20))
side effects:(not (top x21 x17))(on x20 x17)
learning context: (disk x18) (lt x18 x20)

Fig. 5. Learning a side context

The operator is now correct. Concluding the solution eliminates the learning context. To test the operators without the user's assistance, we have implemented a planning system, following Prodigy's search strategy, as described in [14]. It doesn't use control rules, and uses a strategy for eagerly applying, but it serves our purposes: learned operators can be used to solve new problems.

8. Related Work

Disciple [8] is a learning apprentice that uses a multi-strategy approach to acquire problem decomposition rules, and incrementally refine a domain theory, by interacting

with a user in a way that is very similar to our approach. Disciple does not learn operators, describing elementary actions, but ways of decomposing problems into simpler ones, till the elementary actions are reached. Our purpose is to learn, by a similar interaction with the user, the elementary actions, and the macro-operators that define decomposition rules, by experience.

Observer [15], also does automatic acquisition of planning operators, by observing expert solution traces and by practice. Solution traces contain episodes of action execution, described by the state and the changes that occur after execution. Experimentation does not rely on the user, but uses a perfect model of the domain to validate the current solution to a problem, therefore to refine the operators. To acquire the preconditions of an operator, it uses an inductive technique, taking solution traces and experiments as examples, closely following Mitchell's version spaces [13]. The main difference with our approach concerns the way the expert is used as a teacher. In our approach, the expert is used to both give the solution traces, and to validate experimentation. Because of the domain constraints, the acquisition of the operator's effects tends to be less dependent on the user's answer, as operators are being acquired. We also use positive and negative examples, to refine the operators 's preconditions. We always generalise objects to variables. Observer uses a type hierarchy, therefore it can generalise differently. It learns negated preconditions, always based in near misses, so it can only learn one negated condition. We extend this possibility a little further.

Instructo-Soar also bases its learning efforts in an interaction with a user. It differs from ours in the learning purposes. They want to learn how to aggregate elementary actions into more complex procedures. We want to learn these elementary actions. Our work is also based in the notion of situated explanation, but we only deal with some aspects of the possible solutions to the lack of explanation. Particularly, we always rely on the user. We allow the system to conduct some experiments, to complete its knowledge, while Instructo-Soar uses its complete but elementary knowledge to reason about the world and eventually solve the problem. Then, it learns new operators. We call these macro-operators.

Experiment [2] uses experimentation to refine operators that are initially incomplete. It can not start with an empty set of knowledge, and also relies on a perfect model of the domain, to define learning episodes. Experiment learns from general to specific, while we can learn in both directions, by generalisation or specialisation of the operator's description.

9. Conclusions and Future Work

Most of our experiments have been done with toy applications, and some effort must be conducted to deal with more world-related applications. The results have been satisfactory, and even when trying different modelling approaches, the operators can be acquired. A single planning problem normally allows acquiring and refining most of the operators involved, due to the experimentation capabilities of the system. In spite of the toy aspect of the used domains, we have tried to create most different learning scenarios and have been successful with the approach. One of the major

limitations is the fact that the system doesn't work with numbers, so it can not correctly work with (particularly, learn) domains where characteristics of objects are better described in terms of a number. This has to be improved. Also, the system does not learn operators that need to refer to universally quantified variables to be correctly described. This is important, even if one can normally find different ways of modelling a domain that do not necessitate this characteristic. The application of the system to more domains will certainly allow us to discover other learning perspectives.

10. References

1. Borrajo, D and Veloso, M, 1994, *Incremental Learning of Control Knowledge for Nonlinear Problem Solving*, ECML94, pp 64-82.
2. Carbonell, J and Gil, Y, 1990, *Learning by Experimentation: The Operator Refinement Method*, In R. S. Michalsky and Y. Kodratoff (eds) Machine Learning: An Artificial Intelligent Approach, Vol III, Morgan Kaufmann, pp 191-213.
3. Cheng, P and Carbonell, J, 1986, *The FERMI System: Inducing Iterative Macro-operators from Experience*, AAAI86, pp 490-495.
4. Ferreira, J L and Costa, E, 1993, *Learning Operators While Planning*. *Proceedings of the Sixth Portuguese Conference on Artificial Intelligence*, pp 311-323.
5. Ferreira, J L and Costa, E, 1994, *Opportunities for Learning in Planning*, *Proceedings of the XI Brazilian Symposium on Artificial Intelligence*, pp 321-334.
6. Huffman, S and Laird, J, 1996, *Flexible Instructable Agents*, Journal of Applied AI Research, vol 3, pp 271-324.
7. Kelleher, G and Cohn, A G, 1992, *Automatically Synthesising Domain Constraints from Operator Descriptions*, ECAI92, pp 653-655.
8. Kodratoff, Y and Tecuci, G, 1991, *DISCIPLE-I: Interactive Apprentice System in Weak Theory Fields*. IJCAI91.
9. Laird, J E, Newell, A, & Rosenbloom, P S, 1987, *Soar: An architecture for general intelligence*. Artificial Intelligence, 33 (1), 1—64.
10. Langley, P and Allen, J, 1991, *The Acquisition of Human Planning Expertise*, Proceedings of the 8th International Workshop on Machine Learning, pp 80-84.
11. Leckie, C and Zuckerman, I, 1993, *An Inductive Approach to Learning Search Control Rules for Planning*, IJCAI93, pp1100-1105.
12. Macedo, L, Pereira, F, Grilo, C and Cardoso, A, 1996, *Plan Cases as Structured Networks of Hierarchical and Temporal Related Case Pieces*, in EWCBR96.
13. Tom M. Mitchell. 1978, *Version Spaces: An Approach to Concept Learning*. PhD Thesis, Stanford University.
14. Veloso, M, Carbonell, J, Pérez, A, Borrajo, D, Fink, E and Blythe, J, 1995, *Integrating Planning and Learning: The PRODIGY Architecture*, Journal of Experimental and Theoretical Artificial Intelligence, 7 (1), pp 81-120.
15. Xuemei Wang. 1996, *Learning Planning Operators by Observation and Practice*, PhD Thesis, School of Computer Science, Carnegie Mellon University (CMU-CS-96-154).

Redundant Covering with Global Evaluation in the RC1 Inductive Learner

Alneu de Andrade Lopes[1]

alneu@ncc.up.pt
http://www.ncc.up.pt/~alneu

Pavel Brazdil[2]

pbrazdil@ncc.up.pt
http://www.ncc.up.pt/~pbrazdil

LIACC - Laboratório de Inteligência Artificial e Ciências de Computadores
Universidade do Porto,
Rua Campo Alegre 823, 4150 Porto, Portugal

Abstract. This paper presents an inductive method that learns a logic program represented as an ordered list of clauses. The input consists of a training set of positive examples and background knowledge represented intensionally as a logic program. Our method starts by constructing the explanations of all the positive examples in terms of background knowledge, linking the input to the output arguments. These are used as candidate hypotheses and organized, by relation of generality, into a set of hierarchies (forest). In the second step the candidate hypotheses are analysed with the aim of establishing their effective coverage. In the third step all the inconsistencies are evaluated. This analysis permits to add, at each step, the best hypothesis to the theory. The method was applied to learn the past tense of English verbs. The method presented achieves more accurate results than the previous work by Mooney and Califf [7].

1 Introduction

Inductive Logic Programming (ILP) is a sub-area of Machine Learning whose objective is to induce clausal theories from given background knowledge and sets of positive and negative examples. Further requirements usually include the following. The induced theory should be able to classify new examples accurately; the induced rules should be as comprehensible as possible; the learning system should be efficient enough to be used on large example sets. The existing algorithms try to optimize the tradeoff between these requirements.

[1] Supported by CNPq - Conselho Nacional de Desenvolvimento Científico e Tecnológico, Brazil; Support acknowledged of Praxis XXI and FEDER.
[2] Support gratefully acknowledged of Praxis XXI, FEDER and Programa de Financiamento Plurianual de Unidades de I&D.

F. Moreira de Oliveira (Ed.): SBIA'98, LNAI 1515, pp. 111-120, 1998.

In this paper we present an algorithm named RC1 (rule- and case-based system) which uses a modified version of the standard covering method using a richer structure for its hypothesis space. This approach provides better accuracy and leads to more comprehensible theories. These aspects compensate for certain loss of efficiency that can arise when dealing with large training sets.

This paper is organized as follows. In section 2 we describe the general covering algorithm, focus on motivation for developing a new method. In Section 3 we describe the RC1 algorithm and its application to a problem of past tense of English verbs. In Section 4 we present the experiment results on this dataset. In Section 5 we comment on a future work and then present the conclusions.

2 Problems of Standard Covering Approach

Covering algorithms can be considered a family of methods for learning rule sets [6]. Algorithms in this family, such as CN2 [2], FOIL, FOIDL [7] and SKILit [4], typically learn one rule at a time. The strategy consists of processing the given examples in a sequential manner. Whenever a rule is generated, the examples that are covered are removed. This process is iterated on the remaining examples as many times as necessary to learn a disjunctive set of rules that together cover all the given positive examples.

We note that sequential covering may lead to local optima. The algorithm selects an uncovered example and attempts to generate the best hypothesis from this standpoint. This is why usually different rule sets are returned when the examples are reordered. In consequence the error rate usually also varies, producing a component of error which is often referred to as *variance*.

Although in general the problem of local optima cannot be totally avoided, certain steps can be taken to minimize the problem. One possibility is to generate several alternative explanations, which can give rise to alternative hypotheses. It is possible to perform global evaluation, taking into account all available examples, before selecting a particular alternative. This permits to overcome the limitations of the greedy approach inherent in the standard covering method and make better, statistically based, decisions. These are the basic lines followed in RC1.

3 Redundant Covering with Global Evaluation in RC1

The objective of this method is to learn a logic program represented as an ordered list of clauses. The input consists of a training set of positive examples E^+ and background knowledge *BK*, represented intensionally as a logic program. The method starts by constructing the explanations of all the positive examples in terms of background knowledge, linking the input to the output arguments. These are used to generate candidate hypotheses that are organized, by relation of generality, into a set of hierarchies (a forest). The system does not evaluate the individual hypotheses as

they are being generated. The evaluation is done in the subsequent step. The candidate hypotheses are analyzed to establish their overall generality (effective coverage etc). In the third step all the inconsistencies are evaluated. This analysis permits to add, at each step, the best hypothesis to the theory. The RC1 algorithm is summarized in the following figure.

1. Initialize the final theory (to empty theory)
2. Construct Candidate Hypotheses
   - For each positive example, do:
     - Generate explanations and convert them into candidate hypotheses;
     - Add them to the existing structure (a forest).
3. Global Generality Analysis
   - Determine coverage and effective coverage of each candidate hypothesis.
4. Perform Consistency Analysis
   - Identify potentially redundant hypotheses
   - While some non-redundant candidate hypotheses exist, do:
     - Select the best among the most general candidate hypotheses;
     - Add the hypothesis selected to the final theory;
       (at the beginning of the ordered list of clauses)
     - Eliminate inconsistent candidate hypotheses (from the forest);
     - Update the set of potentially redundant hypotheses.
5. Output the final theory

Fig. 3.1 - The RC1 Algorithm.

The method is described in more detail later. As it was applied to the problem of learning the past tense of English verbs, this particular domain of application is described first.

The Problem of Generating the Past Tense of English Verbs: The problem of learning the past tense of English verbs is interesting, because there are various "final theories" that could in principle be generated by a learning system. The input consists of positive examples, such as:

e_1 : past([e, m, i, t],[e, m, i, t, t, e, d]).
e_2 : past([e, a, t],[a, t, e]).
e_3 : past([a, c, c, e, p, t],[a, c, c, e, p, t, e, d]). Etc.

Other examples used are shown later in Fig. 3.4. The *BK* used here was:

```
past(Present, Past) :-
    ending(EndPresent, EndPast),
    split(Present, Radix, EndPresent),
    split(Past, Radix, EndPast).

split( [X|Y], [X], Y).
split( [X|Y], [X|W],Z) :-
    split(Y,W,Z).
```

The predicate *ending(..)* relates the termination of verbs in its present tense forms (*EndPresent*) with the termination of its past tense (*EndPast).* The predicate *split*(*A, B, C*) divides list *A* in two lists, *B* and *C.* So, for example, *split*([*e, m, i, t*], *B, C*)

produces alternative answers shown in Fig. 3.2a. The task is to achieve a final theory composed of an ordered set of clauses *ending(..)* such as the one shown in Fig. 3.2b.

B=[],	C=[e, m, i, t];
B=[e],	C=[m, i, t];
B=[e, m],	C=[i, t];
B=[e, m, i],	C=[t];
B=[e, m, i, t],	C=[]

Fig. 3.2a - Results of ?-split([e,m,i,t], B, C).

```
c8: ending( [f, i, g, h, t], [f, o, u, g, h, t]) :- !.
c7: ending( [e, a, t], [a, t, e] ) :- !.
c6: ending( [b, e, a, t], [b, e, a, t] ) :- !.
c5: ending( [o, t], [o, t, t, e, d] ) :- !.
c4: ending( [i, t], [i, t, t, e, d] ) :- !.
c3: ending( [s, t], [s, t] ) :- !.
c2: ending( [t], [t, e, d] ) :- !.
c1: ending( [], [t, e, d] ) :- !.
```

Fig. 3.2b - Example of a final theory.

3.1 Construction of Candidate Hypotheses

In this section we describe how candidate hypotheses are generated and organized by the relation of generality, giving rise to a structure in the form of a *forest*.

The system proceeds by analyzing the positive examples given one by one. Let us see what happens when one particular example has been encountered. The first task is to generate one or more *explanations*. These are structures of the form $\langle h_j, BK, e_i \rangle$, such that $h_j \cup BK \vdash e_i$, where h_j represents a particular hypothesis, e_i an example belonging to the set of positive examples E^+, and BK is the background knowledge. Whenever e_i can be deduced from $h_j \cup BK$, we say that $h_j \cup BK$ *explains* e_i. So, for instance, *ending*([*i,t*],[*i,t,t,e,d*]) can be considered as one possible explanation for the example *past*([*e,m,i,t*],[*e,m,i,t,t,e,d*]). This can be expressed as:

ending([*i,t*], [*i,t,t,e,d*]) $\cup$ *BK* $\vdash$ *past*([*e,m,i,t*],[*e,m,i,t,t,e,d*]).

Considering our previous case, the explanation can be represented as:

$\langle$*ending*([*i,t*], [*i,t,t,e,d*]), *BK*, *past*([*e,m,i,t*],[*e,m,i,t,t,e,d*])$\rangle$

Explanations can be generated with the help of inductive or deductive methods. The first category includes inductive learning systems, like Progol, SKILit [4], among others. The other group includes methods that attempt to generate possible explanation from other more general forms of knowledge using deductive methods. As our aim is to discuss the methods useful for the analysis of explanations, we do not go into details of any particular method. However, it is interesting that in our domain we can generate all explanations by simply posing a query of the form:

?-past(Present, Past), split(Present, Radix, EndPresent), split(Past, Radix, EndPast), *write*(*ending*(*EndPresent, EndPast*)), *fail, nl*.

So for instance, for the examples *past*([*e,m,i,t*], [*e,m,i,t,t,e,d*]) and *past*([*a,l,l,o,t*], [*a,l,l,o,t,t,e,d*]) we would obtain four different explanatory hypotheses to each (giving rise to four different alternative explanations), which are shown in the following figure.

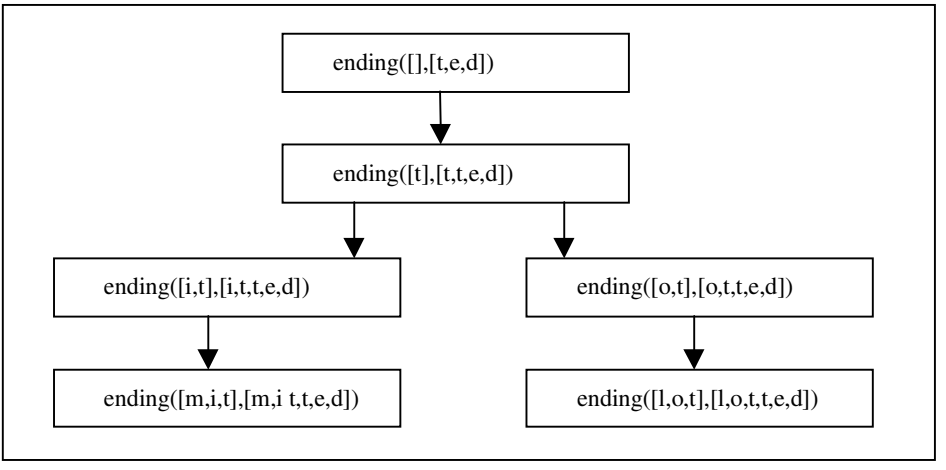


Fig. 3.3 - Alternative Hypotheses Ordered by Generality for *past([e,m,i,t],[e,m,i,t,t,e,d])* and *past([a,l,l,o,t],[a,l,l,o,t t,e,d])* after merging.

We note that the hypotheses generated can be ordered by generality. For instance, $h_1 = ending([],[t,e,d])$ is more general than $h_2 = ending([t],[t,t,e,d])$. This is because h_2 can be re-represented as h_2' = *decomp*([*t*],*t*,[]), *decomp*([*t,t,e,d*],t,[*t,e,d*]), *ending*([],[*t,e,d*]) and h_1 *θ-subsumes* h_2'. Here the predicate *decomp*(*A,B,C*) decomposes list *A* into the first element *B* and the remaining list *C*. In general we will use *θ-subsumption* to determine whether one hypothesis is more general than another.

Definition: Let n_j and n_k be two hypotheses in the hypothesis space. Then n_j is ***more-general-than-or-equal-to*** n_k ($n_j \geq_g n_k$) iff n_j θ-subsumes n_k.

When we get to processing two or more examples, each giving rise to a particular set of hypotheses ordered by generality. Let us consider, for instance, that having processed example *past*([*e,m,i,t*],[*e,m,i,t,t,e,d*]) we have encountered *past*([*a,l,l,o,t*], [*a,l,l,o,t,t,e,d*]). This produces the hypotheses that can be merged into a tree, such as shown in Fig. 3.3.

As the processing progresses we get a collection of hierarchies similar to the ones shown, which is referred to as a *forest.*

3.2 Global Generality Analysis

After the alternative candidate hypotheses have been generated, the system proceeds with global analysis of all alternative hypotheses. The aim is to determine how general each hypothesis is, by looking at how many examples it covers. Here we will consider coverage of positive examples only, which can be defined as the number of positive examples correctly covered by a given hypothesis

Definition: The ***coverage*** of hypothesis h_j with respect to background knowledge *BK* and training data E^+ is equal to the number of examples in the set $\{ e_i \mid (e_i \in E^+) \wedge (h_j \cup BK \vdash e_i)\}$.

Another fundamental concept is effective coverage, which expresses the net contribution of going down one level following the links in the forest. Effective coverage is thus the number of positive examples covered by the hypothesis minus the number of examples covered by its more specific versions in the forest. A more precise definition is given below.

Definition: The ***effective coverage*** of hypothesis h_j with respect to background knowledge *BK* and training data E^+ is the coverage of hypothesis h_j minus the sum of coverage of all hypotheses h_j' immediately below h_j in the tree ($h_j >_g h_j$').

The results of this stage are captured in Fig. 3.4. It shows the forest generated by the system in which each hypothesis is characterized by two numbers - coverage end effective coverage.

3.3 Consistency Analysis

Having generated the candidate hypotheses the system carries out a consistency analysis (step 4 of Fig. 3.1).

This method differs from other usual approaches in which specific (but correct) clauses are preferred first. Our aim is to start with good general candidates and then to minimize its incorrect coverage by selecting more specific rules. But let us see various steps of this phase in more detail.

Identify Potentially Redundant Hypotheses: In this step the topmost hypothesis at each hierarchy in the forest are identified. All the hypotheses bellow are considered as potentially redundant.

Selecting the Best Among the Most General Candidate Hypotheses: First, let us see how we select the most general hypotheses. This is done by examining the forest and returning T, the set of topmost elements. Let us consider the forest shown in Fig. 3.4. On this domain the method will return the following hypotheses as T: ⟨*ending*([],[*t,e,d*]), *5,0*⟩, ⟨*ending*([],[*e,d*]), *3,0*⟩, ⟨*ending*([],[]), *3,0*⟩. The elements in this set are characterized using coverage and effective coverage. The system uses effective coverage as the first criterion to selection, and in case of tie-up, it considers coverage. In our case above, as all three hypotheses have the same effective coverage (0), the first hypothesis is selected, because it covers 5 examples, while the other two alternatives cover only 3. Consequently, the clause *ending*([],[*t,e,d*]) is added to the final theory and eliminated from the forest. As this is the first clause that was handled, the final theory contains one clause after this step. More details concerning this method are given in Section 3.4.

Eliminating Inconsistent Candidate Hypotheses: The following discussion is oriented towards deterministic solutions, where each particular input value is expected to give *exactly one* output value. Then inconsistent hypotheses can be simply identified as those that generate different outputs for the same input. Having selected a hypothesis (say *h*), the system analyzes the subset T with the objective of identifying all inconsistent hypotheses (with *h*).

Let us consider again the forest shown in Fig. 3.4. We observe that the hypotheses *ending*([],[*e,d*] and *ending*([],[]) are inconsistent with the chosen hypothesis (i.e.

ending([],[*t,e,d*]). In consequence they are eliminated from the forest. Here we see that:

ending([], [*t,e,d*]) ∪ *BK* ⊢ *past*([*e,m,i,t*],*Y*)θ_1, where $\theta_1 = (Y = [e,m,i,t,t,e,d])$

ending([], [*e,d*]) ∪ *BK* ⊢ *past*([*e,m,i,t*],*Y*)θ_2, where $\theta_2 = (Y = [e,m,i,t,e,d])$

and observe that the substitutions returned differ, that is $\theta_1 \neq \theta_2$. In general, the notion of inconsistency can be defined as follows:

Definition: Let e_i be a positive example of the form $\langle x, c(x) \rangle$, where x represents the inputs and $c(x)$ the given output. Let e_i^* be an example obtained from e_i by replacing the output by a variable. Then hypotheses h and h' are ***inconsistent*** iff:

$$h \cup BK \vdash e_i^* \; \theta_1 \text{ and } h' \cup BK \vdash e_i^* \; \theta_2 \text{ and } \theta_1 \neq \theta_2$$

More details concerning the elimination of inconsistencies step are given in Section 3.4.

Update the Potentially redundant Hypotheses: In each iteration, after selecting the best hypothesis, the remaining hypotheses in the set of topmost elements are eliminated. This elimination implies that their immediate descendants are no longer potentially redundant. So they have to be reconsidered in the next iteration.

When there are no more non-redundant hypotheses in the forest we obtain the final theory. The final theory constructed in this manner has already been shown in Fig. 3.2b. Note that clauses c6, c7 and c8 represent cases. We note that instead of using, say c8: *ending*([*f,i,g,h,t*],[*f,o,u,g,h,t*]), we could have used the more general version *ending*([*i,g,h,t*],[*o,u,g,h,t*]). Our criterion here is that if a clause covers just one example, we prefer to store a case.

3.4 Consistency Analysis in Detail

In this section we present a detailed description of the method of consistency analysis. Before this we describe some preliminary steps.

First, the system attaches a certain level to each element in the forest. These levels are related to generality relation of the hypotheses at each tree in the forest (Fig. 3.4). The levels are processed starting with the most general level (level 1).

Second, the system needs to identify all potentially redundant hypotheses in the forest. These are all the candidate hypotheses different from the most general ones (set T). They are marked as potentially redundant.

When one of several inconsistent hypotheses is selected, the examples covered specifically by the other hypotheses are stored as new cases. Supposing we have reached level 2 and are dealing with the following inconsistent hypotheses ⟨*ending*([*t*],[*t,t,e,d*]), *5,0*⟩, ⟨*ending*([*t*],[*t,e,d*]), *3,1*⟩, ⟨*ending*([*t*],[*t*]), *3,1*⟩, while the second one has been selected on the basis of its effective coverage. One of the inconsistent hypotheses that was left behind is ⟨*ending*([*t*],[*t*]), *3,1*⟩, which covers the case *past*([*b,e,a,t*], [*b,e,a,t*]). As this is effective coverage (this case is covered by ⟨*ending*([*t*],[*t*]), *3,1*⟩, but not for its more specific descendant), this case is stored as a learned case. It could be considered as exception to the chosen hypothesis.

In addition, whenever a hypothesis has been selected from incompatible set, the remaining ones are used to identify their immediate descendants. They are updated to

non-redundant. This way, for instance, the hypothesis ⟨*ending*([*s,t*],[*s,t*]), *2,2*⟩ is now considered non-redundant, as well as ⟨*ending*([*i,t*],[*i,t,t,e,d*]), *5,0*⟩ and ⟨*ending*([*o,t*], [*o,t,t,e,d*]), *5,0*⟩, which are immediate descendants of ⟨*ending*([*t*],[*t,t,e,d*]), *5,0*⟩. Since their parent hypotheses have been eliminated. When there are no more non-redundant hypotheses, all cases learned till then are appended to the ordered list of learned rules and we get the final theory.

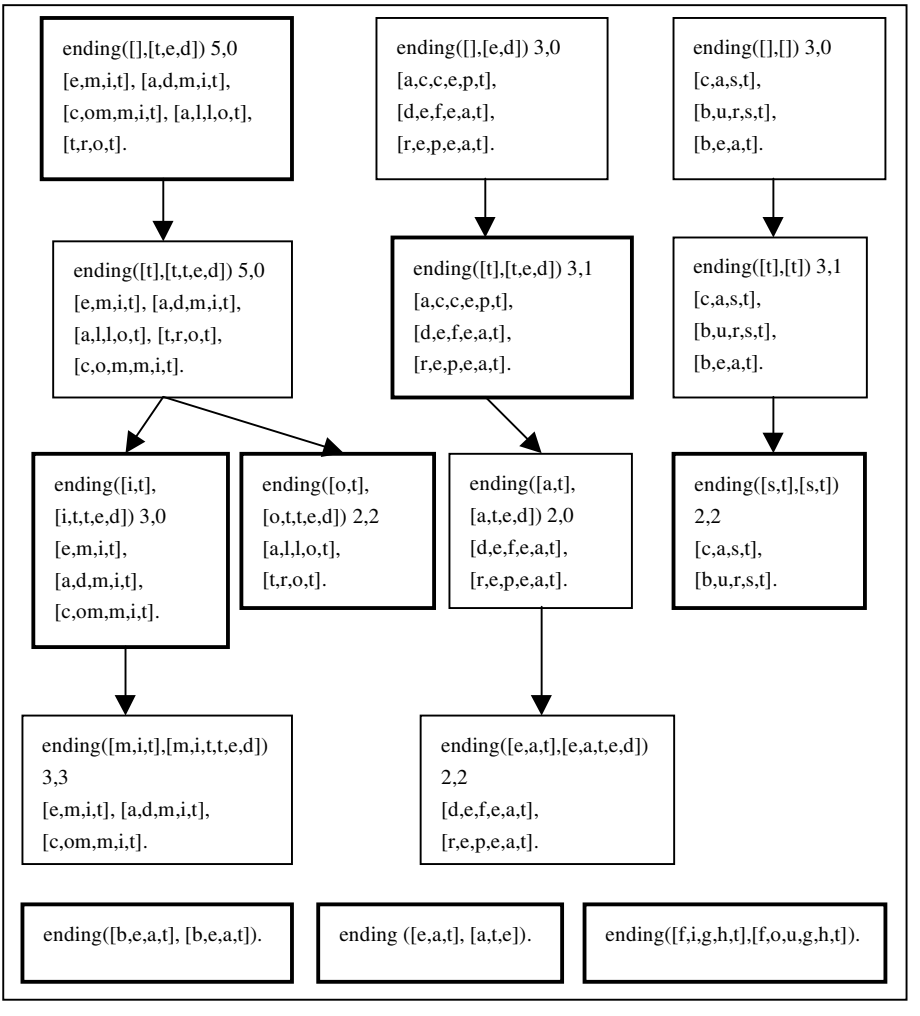


Fig. 3.4 - The Forest of Hypotheses. Each box contains the hypothesis qualified with coverage and effective coverage. All examples covered are also shown. The boxes in bold represent the hypotheses which were selected into the final theory. The bottom part of the figure contains boxes without links to other parts of the diagram. As no accepted explanation was found for the examples in question, they are represented as cases.

4 Experimental Results

In the experiments we use the past tense data (1392 cases) used by Mooney and Califf [7]. This data was split to provide training sets with 25, 50, 100, 250, 500, and 1000 cases. For the training sets consisting of up to 250 cases, the remaining portion was used to generate a test set with 500 cases. For training sets with 500 and 1000 cases, the rest of the available data, i.e., 892 and 392 cases, respectively, was used for testing. The results were averaged over 5 trials. The results are shown in Fig. 4.1. For comparison the figure also shows the results of Foidl [7] which uses a standard covering method to generate the theory.

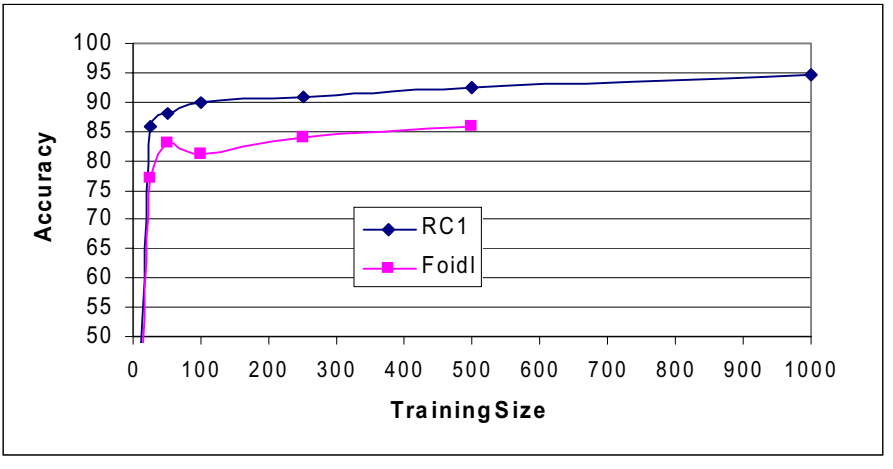


Fig. 4.1 - Accuracy on Past Tense Data.

5 Discussion and Conclusion

In our view the method (RC1) achieves good results due to its mechanism of selecting rules. RC1 starts by selecting good general rules. However, general rules tend to incorrectly cover a large number of examples (negative examples). So, the RC1 tries to minimize this incorrect coverage by selecting new more specific rules that correctly cover them. These new rules may themselves incorrectly cover other examples. Therefore the process iterates until all examples are correctly covered.

Note that the rules that are introduced to minimize incorrect coverage can be redundant relative to the previously selected ones. The power of the method seems to be due those redundant rules. However plausible this assumption is, this could be verified. We could apply, for instance, the MDL principle to carry out experiments with pruning to see how this affects performance.

We note that the final theory is composed of cases without explanation, cases partially explained (exceptions to a rule), and rules. It is interesting to consider exactly why and how the cases can arise. There are three different possibilities.

The first one involves opting for a case in preference to a rule. We use rather a pragmatic criterion to decide when to do that. We simply consider how many examples the rule covers and if it is just one example, a case is preferred. This is probably in line with MDL criterion, but more work could be done to evaluate this better. Cases can also arise whenever no other explanation is found (besides the trivial explanation that is the case itself). For instance, the example *past*([*e, a, t*],[*a, t, e*]) is explained by *ending*([*e, a, t*],[*a, t, e*]). It is questionable whether we should store *ending*([*e, a, t*],[*a, t, e*]) or just *past*([*e, a, t*],[*a, t, e*]). Cases can also be generated during the consistency analysis. Whenever there are incompatible hypotheses, one of them is chosen and added to the final theory, while the examples effectively covered by the other hypotheses remain as cases. These cases represent exceptions to the chosen rule.

Considering that cases play quite an important role, it would seem useful to explore and integrate better Case-based Reasoning with rule induction, following the previous work in [5]. This seems promising, since the case explanation seems to provide useful semantic information (rather than usual syntactic measures) for case retrieval and adaptation.

In conclusion, we have presented an algorithm that exploits the information concerning alternative hypotheses, organized by generality into a forest, in the search for final theory. The method is applicable in situations where this information can easily be generated. The global analysis enables the system to exploit this information (particularly that of effective coverage), and in consequence, produce a final theory that has good predictive power. Experimental results have shown that this method achieves better results than Foidl of Mooney and Califf [7] that uses a standard covering method.

Acknowledgments: We would like to thank James Cussens, Alípio Jorge, and Luc Dehaspe for their useful comments.

References

[1] Clark, P. A Comparison of Rule and Exemplar-Based Learning Systems. *Machine Learning, Meta Reasoning and Logics,* Pavel B. Brazdil, Kurt Konolige (eds.), Kluwer Academic Publishers, pp. 159-186, 1989.

[2] Clark, P.; Niblett T. The CN2 Induction Algorithm. *Machine Learning 3*, Kluwer Academic Publishers, pp. 261-283, 1989.

[3] Golding, A.R.; Rosenbloom, P.S. Improving Accuracy by Combining Rule-Based and Case-Based Reasoning. *Artificial Intelligence 87,* pp. 215-254, 1996.

[4] Jorge, A.; Brazdil, P. Architecture for Iterative Learning of Recursive Definitions. *Advances of Inductive Logic Programming*, L. de Raedt (ed.), IOS Press, Amsterdam, 1996.

[5] Lopes, A. A. Using Inductive Learning in Case-Based Reasoning (in Portuguese). Proceedings of Brazilian Symposium on Artificial Intelligence - SBIA 96 (student session), 1996.

[6] Mitchell, T. M.; *Machine Learning*. The McGraw-Hill Companies, Inc, 1997.

[7] Mooney, R.J. Induction of First-order Decision Lists: Results on Learning the Past Tense of English Verbs. *Journal of Artificial Intelligence Research 3,* pp. 1-24, 1995.

Towards Integrating Hierarchical Censored Production Rule(HCPR) Based Systems and Neural Networks

Kamal Kant Bharadjwa [1] and Jose Demisio Simoes da Silv [2]

[1] LAC/INPE, S.J.Campos, SP, Brazil
Email : drkkb@lac.inpe.br
[2] LAC/INPE, S.J.Campos, SP, Brazil
Email : demisio@lac.inpe.br

Abstract. Over the past few years , researchers have successfully developed a number of systems that combine the strength of the symbolic and connectionist approaches to Artificial Intelligence. Most of the efforts have employed standard production rules, **IF**<condition> **THEN** <action> as underlying symbolic representation. In this paper we have suggested Variable Precision Neural Logic (VPNL) networks as an attempt towards integrating Hierarchical Censored Production Rule(HCPR) based system and neural networks. A HCPR has the form :

Decision (**If** precondition)
(**Unless** censor_conditions)
(**Generality** general_information)
(**Specificity** specific_information))

which can be made to exhibit variable precision in the reasoning such that both certainty of belief in a conclusion and its specificity may be controlled by the reasoning process. Also it is proposed how Dempster-Shafer uncertainty calculus can be incorporated into the inferencing process of VPNL networks. The proposed hybrid system would have numerous applications where decision must be taken in real time and with uncertain information - examples range from medical expert systems for operating rooms to domestic robots and with its capability of modeling wide range of human-decision making behavior, the system would have applications such as bond trading, currency option trading.

1 Introduction

Over the last few years there has been an increasing amount of research that can be considered a hybrid of the symbolic and connectionist approaches. There are a large number of ways ([5], [7], [9], [13], [14], [15], [16]) to combine symbolic and connectionist Artificial Intelligence. Shavlik [13] presented a framework that encompasses the approaches of several different research groups. This frame work views the combination of symbolic and neural learning as a three stage process :

---

[1]On leave from the School of Computer & Systems Sciences, JNU, New Delhi, India. The research of the first auther is supported by the Brazilian foundation CNPq under Grant No. 381777/97-9.

F. Moreira de Oliveira (Ed.): SBIA'98, LNAI 1515, pp. 121-130, 1998.

(I) the insertion of symbolic information into a neural network, thereby (partially) determining the topology and initial weight settings of a network.,
(II) the refinement of this network using a numeric optimization method such as backpropogation, possibly under the guidance of symbolic knowledge, and
(III) the extraction of symbolic rules that accurately represent the knowledge contained in a trained network.

An excellent effort has been made by Towell and Shavlik [16] towards developing KBANN (Knowledge- Based artificial neural Networks). Mahoney and Mooney [7] developed RAPTURE (Revising Approximate Probabilistic Theories Using Repositories of Examples) system for revising probabilistic knowledge bases that combines connectionist and symbolic learning methods.

A neural network named as the Neural Logic Network (NEULONET) has been developed by Tan et. al [14] that combines strength of rule-based expert systems and neural networks. The NEULONET based on a 3-valued logic allows a great multitude of logical operation which models a wide range of human-decision making behaviors ([11], [12]).

Most of these approaches have used standard production rules (**<IF** *Condition* **THEN** *Action* >) as the symbolic knowledge representation. The major shortcomings of an ordinary logic based reasoning system is that you cannot tell much about the way you want it to perform its task. As an extension of standard production rule, Michalski and Winston[8] have suggested the *Censored Production Rule* (CPR) as an underlying representational and computational mechanism to enable logic based systems to exhibit variable precision in which certainty varies while specificity stays constant. A CPR has the form , **If** *A* **Then** *B* **Unless** *C*, where *C* (Censor) is the exception condition. Such rules are employed in situations in which the conditional statement '**If** *A* **Then** *B*' holds frequently and the assertion *C* holds rarely. By using a rule of this type we are free to ignore the Censor (exception) condition, when the resources needed to establish its presence are tight or there simply is no information available as to whether it holds or does not hold. As time permits the Censor condition *C* is evaluated establishing the conclusion B with higher certainty if *C* does not hold or simply changing the polarity of *B* to ~ *B* if *C* holds.

To address various problems and shortcomings with CPRs system , Bharadwaj and Jain [1] have introduced a concept of Hierarchical Censored Production Rule (HCPR). A HCPR is a CPR augmented with specificity and generality information which can be made to exhibit variable precision in the reasoning such that both certainty of belief in a conclusion and its specificity may be controlled by the reasoning process. Such a system has numerous applications in situations where decision must be taken in real time and with uncertain information. Examples range from medical expert systems for operating rooms to domestic robots. Further, HCPRs system that support various symbolic [6] and Genetic based machine learning (GBML) [2] would be very useful in developing Knowledge based systems with learning capabilities.

As an extension of NEULONET a Variable Precision Neural Logic(VPNL) network is proposed in this paper that combines the strengths of Hierarchical Censored Production Rules (HCPRs) system and neural networks. Also it is proposed how Dempster-Shafer uncertainty calculus can be incorporated into the proposed scheme.

2 Variable Precision Neural Logic (VPNL) Network

Following [14], a NEULONET as shown in Fig. 1(a) contains a set of input nodes and an output node. Every node can take one of the three ordered pair activation values ,(1,0) for true, (0,1) for false and (0,0) for don't know-so called the 3-valued NEULONET. Every edge is also associated with an ordered pair weight (α, β) where α and β are real numbers. The rule of activation is defined as follows :

$$Act(Q) = \begin{cases} (1,0) & if\ \ Net(Q) >= \lambda \\ (0,1) & if\ \ Net(Q) <= -\lambda \\ (0,0) & otherwise \end{cases} \tag{1}$$

where λ is the threshold , always set to 1, unless otherwise indicated. Where Net(Q) is defined by the rule of propagation :

$$Net(Q) = \sum_{i=1}^{n} (a_i \alpha_i - b_i \beta_i) \tag{2}$$

Neulonets representing basic logical operations are shown in Fig. 1(b) - (e).

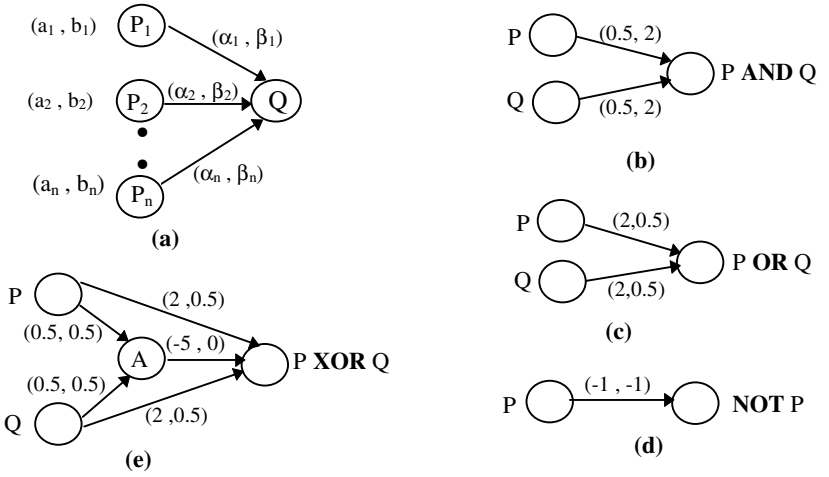


Fig. 1. (a) A basic Neural Logic Network - a Neulonet.
(b) - (e) Logic Operations.

2.1 Hierarchical Censored Production Rules(HCPRs) to Network Translation

Firstly censored production rules(CPR) are considered and thereafter translation scheme is extended to HCPRs. A CPR, **IF** *P* **THEN** *D* **UNLESS** *C* has the interpretation that if *P* is true and censor condition C is not true then D is true; and if P is true and censor condition C holds then D is not true. The Table 1 represents this interpretation. An equivalent network representation of the CPR that incorporates the above logical operation D $\oplus$ C, is called, Variable Precision Neural Logic

(VPNL)network and its graphical representation is given in Fig. 2. As an example consider the following Censored Production Rule (CPR) :

flies(x) **IF** *is -bird(x)* **UNLESS** *is-special-bird (x)* **or** *is-in-unusual-condition (x).*
is-special-bird (x) **IF** *is-penguin (x)* **or** *is-ostrich (x)* **or** *is-domestic-turkey (x).*
is-in-unusual-condition (x) **IF** *is-dead (x)* ***or*** *is-sick (x)* **or** *has-broken-wings (x).*

The above rule can be evaluated with the following two interpretations:
(i) censor conditions are ignored or both the censor conditions are evaluated at once.
(ii) censor conditions are ignored or censor conditions are evaluated one by one as the time permits

Table 1.

D (Decision)	C (Censor)	D ⊕ C
(1,0)	(1,0)	(0,1)
(1,0)	(0,1)	(1,0)
(1,0)	(0,0)	(1,0)
(0,1)	(1,0)	(0,1)
(0,1)	(0,1)	(0,1)
(0,1)	(0.0)	(0,1)
(0,0)	(1,0)	(0,1)
(0,0)	(0,1)	(0,0)
(0,0)	(0,0)	(0,0)

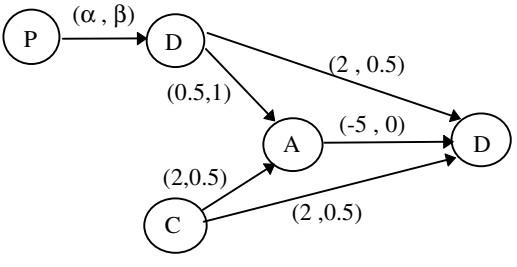


Fig. 2. VPNL network representation of CPR : IF P Then D Unless C

The Fig. 3 gives the VPNL network representation of the above "Bird" example under the interpretation (ii)..

Hierarchical Censored Production Rule (HCPR) is an augmented CPR with general and specific information and has the following form [1] :

Decision (**If** precondition)
(**Unless** censor_conditions)
(**Generality** general_information)
(**Specificity** specific_information)

A HCPRs-tree is a collective and systematical representation of all the related HCPRs about a given problem domain. Here from 'collective' means the set of all rules or definitions related to a most generalized concept in the knowledge base. A HCPR in the HCPRs-tree is the smallest and simplest chunk of knowledge that may be created, modified, or removed without directly affecting the other HCPRs in the ,

A few related HCPRs are given below and it is shown how they are linked in a HCPR-tree structure, as depicted in Fig. 4. This represents a rule-base to find answers to queries of the type " What is X doing?" when supplied with relevant input data.

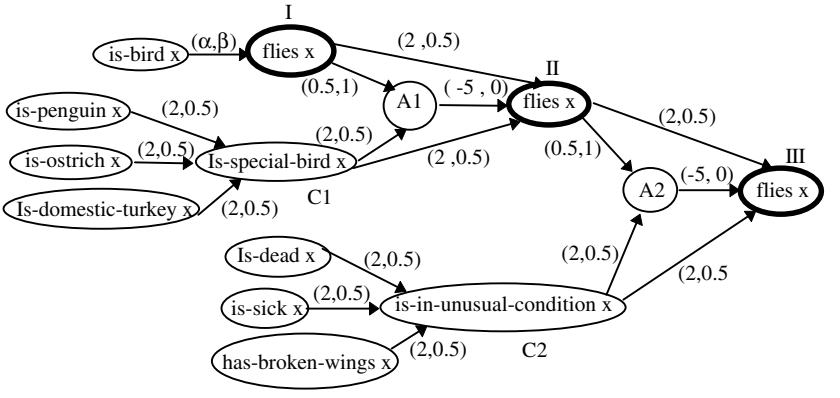


Fig. 3. Bird Example - node I indicates decision when censors are ignored, node II indicates the decision when only is-special -bird (x) is considered and node II corresponds to the case when is-in-unusual-condition (x) is also being considered.

{level 0}
Is_in_city(X,Y)
If [Lives_in_city(X,Y)]
Unless [Is_on_tour(X)]
Generality [ ]
Specificity [Is_at_home(X), Is_outside_home(X)]

{level 1}
Is_at_home(X)
If [Time(night)]
Unless[Is_doing_overtime(X), Works_in_night_shift(X)]
Generality [Is_in_city(X,Y)]
Specificity[ ]

Is_outside_home(X)
If [Time(day)]
Unless [Is_ill(X) , Watching_TV(X), Bad_weather]
Generality [Is_in_city(X,Y)]
Specificity[Is_working_outdoor(X), Is_entertaining_outdoor(X)]

{level 2}
Is_working_outdoor(X)
If [Day(working)]
Unless [National_holiday, Is_unemployed(X)]
Generality [Is_outside_home(X)]
Specificity [ ]

Is_entertaining_outdoor(X)
If [Day(Sunday)]
Unless [Met_an_accident(X)]
Generality[Is_outside_home(X)]
Specificity[ ]

Fig. 5 illustrates the VPNL network representation showing the connections between two VPNL nets corresponding to any two nodes in the HCPR-tree at the $level_i$ and $level_{i+1}$. For example D_i = X is_outside_home and D_{i+1} = X is_working_outdoor

would represent the connection between the VPNL nets corresponding to two nodes at level1 and level2 in the HCPR-tree shown in Fig. 4.

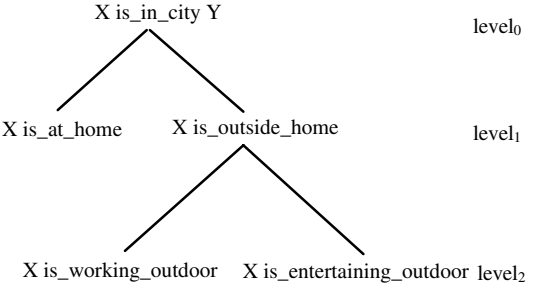


Fig. 4. A HCPRs-tree

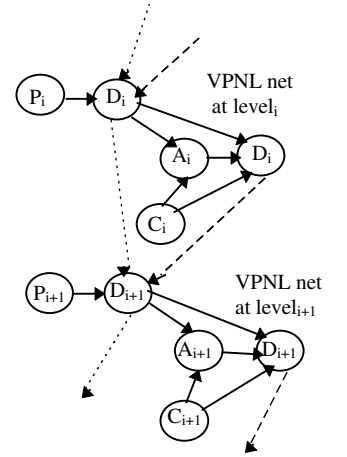


Fig. 5. VPNL network representation of HCPR-tree Hierarchy
.....censors are ignored
- - - - - censors are being considered

3 Refinement Training

The system needs to recognize two distinct types of HCPRs / VPNL network elements. The first type is "definitional" (fundamental principles of domain expert) that are not to be changed during the training phase and the second type is "nondefinitional" (rules based on heuristics) that are subjected to change during training [11], [16].

In VPNL networks various logical operations are represented by arc weights and that provides a means of modifying the rules by adjustment of weights. For heuristic rules weights may be initially assigned based on some intuition but later fine tuned by training examples. During training , when a training example is presented to the system, it will chain up an inference tree from the input nodes to the desired goal node. Relevant VPNL-net rules are selected from the knowledge base depending on the input variables in the training example. After applying the activation and propagation rules, the activation state of the network output node is compared to the desired value in the training example. If the network is not able to derive at the desired conclusion , the error is backpropagated , and the VPNL nets in the inference tree will have their connection weights adjusted[11]. The major distinctions from the refinement training for standard production rules and the HCPRs are :

(I) Each individual VPNL net corresponding to a node in the HCPRs tree can undergo refinement training.

(II) Level to level propagation would follow heuristic rules ([1], [3]).

(III) Refinement training would be required for different interpretations for the same knowledge base i.e. cases for which censors are ignored and when censors are being considered etc. need to be taken into account during training.

4 Dempster-Shafer Uncertainty Calculus

In this section we have suggested an alternative inference scheme to incorporate Dempster-Shafer uncertainty calculus into the VPNL networks.

Dempster-Shafer belief calculus serves as a natural generalization of probability concepts. However, because the belief calculus is broader in scope, it can be used to represent features of uncertainty that are difficult or impossible to represent using probability concepts. For example, the belief calculus provides a natural way to represent ignorance about a situation, and it allows us to draw a distinction between lack of belief and disbelief [3].

Table 2.

D (decision)	C (censor)	D ⊕ C
(1,1)	(1,1)	(0,0)
(1,1)	(0,0)	(1,1)
(1,1)	(0,1)	(1,1)
(0,0)	(1,1)	(0,0)
(0,0)	(0,0)	(0,0)
(0,0)	(0.1)	(0,0)
(0,1)	(1,1)	(0,0)
(0,1)	(0,0)	(0,1)
(0,1)	(0,1)	(0,1)

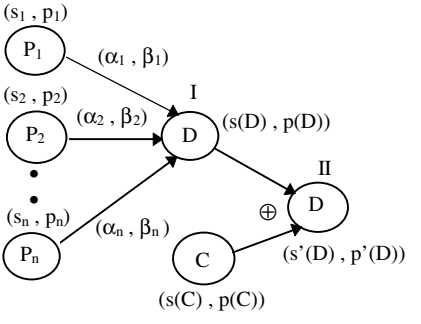


Fig. 6. An alternative inference scheme with node values as Shafer-intervals.

The belief in a proposition A is represented by a Shafer interval [s(A) . .p(A)] , where

$$s(A) = support(A) = \sum_{A_i \subseteq A} m(A_i)$$

$$p(A) = plausibility(A) = 1 - s(\sim A)$$

The uncertainty of a proposition is then defined as the width of the interval : u(A) = p(A) - s(A). In terms of Shafer intervals True, False and Unknown would be represented as [1..1], [0 ..0] and [0 ..1] respectively. The Table 2 gives the representation of the logical operation D ⊕ C. The activation and propagation formulas (1) and (2) are modified accordingly.

Under the alternative inference scheme , apart from the usual activation value, each node P_i will carry another ordered pair (s_i,p_i) denoting Shafer interval associated with the nodal values. After the usual propagation and activation , the Shafer interval associated with the output node are computed as under:

(i) Total uncertainty of the incoming nodes is computed as

$$U = \sum_{i=1}^{n} (p_i - s_i)$$

(ii) Using the product terms $s_i * \alpha_i \; and \; -(1-p_i) * \beta_i, \forall i$, F (evidence favorable to D) and A (evidence against D) are computed as under :
F = sum of all the positive terms,
A = Absolute value of the sum of all the negative terms
(iii) Support (D) and plausibility(D) are computed as (node marked I in Fig. 6):

$s_D = F / S, \quad p_D = (1 - A / S), where \; S = U + F + A$
(iv) If $s_D + p_D >= T_D(threshold)$ then $s_D \leftarrow 1 \; and \; p_D \leftarrow 1$
(v) In order to compute Shafer interval for the node D (marked II in Fig. 6) the (s_D, p_D) values computed at step (iii) are combined (operation $\oplus$) with the Shafer interval (s_C,p_C) associated with the censor condition , using the increment/decrement formulae given below.

Computation of Increment/Decrement : Assume that [s(D)..p(D)] represents the Shafer interval for the decision for which censor conditions are completely ignored. As time permits , a censor with Shafer interval [s(C)..p(C)] is considered and Incre/Decre computed using the following criterion :
[0..0] $\subseteq$ [s(C)..p(C)] $\subset$[0..1] $\Rightarrow$Increment, [s(C)..p(C)] = [0..1] $\Rightarrow$No effect,
and [0..1] $\subset$ [s(C)..p(C)] $\subseteq$ [1..1] $\Rightarrow$ Decrement
such that if [s(C)..p(C)] = [0..0] (False)
$\Rightarrow Incre_{s(D)}(0) = 1 - s(D))(maximum)$

$Incre_{p(D)}(0) = 1 - p(D))(maximum),$

if [s(C)..p(C)] = [0..1] (Unknown) $\Rightarrow Incre = 0, Decre = 0,$
and if [s(C)..p(C)] = [1..1] (True)
$\Rightarrow Decre_{s(D)}(1) = s(D) \quad (maximum)$

$Decre_{p(D)}(1) = p(D) \quad (maximum)$

Based on the above criterion the following heuristic formulae are proposed for the computation of Incre/Decre (in fabricating the formulae the Shafer interval [s(C)..p(C)] is mapped to the set of real numbers, R, using the mapping $[s(C)..p(C)] \rightarrow s(C) + p(C)/10$).

$$Incre_{s(D)}[s(C)] = [(1 - s(D) * Factor * (1 - 10 * \{s(C) + p(C)/10\})],$$

$$Incre_{p(D)}[p(C)] = [(1 - p(D) * Factor * (1 - 10 * \{s(C) + p(C)/10\})],$$

$$Decre_{s(D)}[s(C)] = [s(D) * \{(s(C) + p(C)/10) - 0.1\}],$$

$$Decre_{p(D)}[p(C)] = [p(D) * \{(s(C) + p(C)/10) - 0.1\}]$$

$$, where \; 0 \le Factor \le 1,$$

$$if \; Factor = 0 \Rightarrow all \; censor \; conditions \; are \; unknown,$$

$$= 1 \Rightarrow all \; censor \; condition \; are \; known$$

$$\in (0..1) \Rightarrow some \; censor \; conditions \; are \; unknown$$

The above Increment/Decrement formulae give rise to appropriate formulae for level to level propagation in the HCPR-tree [3]. Fig. 7 shows the computations under the alternative inference scheme

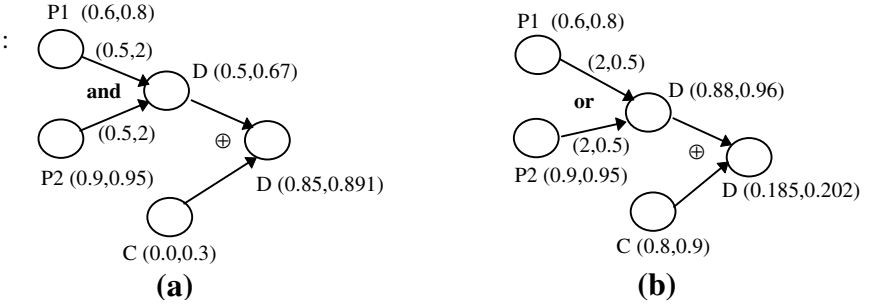


Fig.7. Examples illustrating computation under alternative inference scheme
(a) case of INCRE (increment)
(b)case of DECRE (decrement)

5 Conclusion

As an attempt towards integrating variable precision logic based semantic structure of HCPRs system and learning capability of neural networks we have proposed Variable Precision Neural Logic (VPNL) networks.

In order to incorporate Dempster-shafer uncertainty calculus into the proposed system a scheme is suggested that employs heuristic formulas for the computation of increment/decrement for the [s(D)..p(D)] depending on the censor conditions being evaluated. Performance of the proposed scheme is illustrated through examples. Detailed implementation of the proposed hybrid system is being accomplished.

VPNL networks that supports variable precision logic would have numerous applications where decision must be taken in real time and with uncertain information-examples range from medical expert systems for operating rooms to domestic robots. Further VPNL networks being capable of modeling wide-range of human-decision making behaviors would have applications such as bond trading[11], currency option trading[12] etc.

One of the important research direction would be development of efficient/accurate schemes for handling uncertainty in such hybrid systems([7], [12], [16]). It is to be seen how fuzzy logic [10], certainty factors approach[1] , inherent parallelism in HCPRs system [4] and temporal logic can be incorporated into the proposed system..

References

1. Bharadwaj, K.K., Jain, N.K.: Hierarchical Censored Production Rules (HCPRs) System, Data and Knowledge Engineering, vol.8, (North Holland), pp. 19-34, 1992.
2. Bharadwaj, K.K., Hewahi, N.M.,.Brandao, M.A.: Adaptive Hierarchical Censored Production Rule-Based System : A Genetic Algorithm Approach, *13th Brazilian*

Symposium on Artificial Intelligence, SBIA'96, Curitiba, Brazil, October 1996, *Lecture Notes in Artificial Intelligence* 1159, Eds. Dibio L. Borges, Celso A.A.Kaestner ,Springer-Verlag, pp81-90 .

3. Bharadwaj, K.K., Neerja, Goel, G.C.: Hierarchical Censored Production Rules (HCRPs) Systems Employing the Dempster Shafer Uncertainty Calculus, *Information and Software Technology*, Butterworth-Heinemann Ltd.(U.K.) Vol 36 No., p155-164(1994)

4. Bharadwaj, K.K., Varshneya, R.: Parallelization of Hierarchical Censored Production Rules (HCPRs) System" *Information and Software Technology*, Butterworth- Heinemann(UK), vol.27,No.8,pp453-460,1995.

5. Gallant, S.I.: Connectionist Expert Systems, *CACM,* vol.31, No.2 (1988).

6 Jain, N.K., Bharadwaj, K.K.: Some Learning Techniques in Hierarchical Censored Production Rules (HCPRs) System ,*International Journal of Intelligent Systems,* John Wiley & Sons , Inc.,vol.13, (1997).

7. Mahoney, J.J., Mooney, R.J.: Combining Connectionist and Symbolic Learning to Refine Certainty Factor Rule Bases, Connectionist Science, Vol.5, No.3&4, pp339, 1993.

8. Michalski, R.S., Winston, P.H.: Variable Precision Logic,"*Artificial Intelligene*, vol.29, pp. 121-145, North Holland, 1986.

9. Narazki, H.,.Ralescu, A.L.: A connectionist Approach for Rule-Based Inference Using an Improved Relaxation Method*, IEEE Trans. on Neural Networks,* vol.3, No. 5, pp. 741-751 (1992).

10. Neerja, Bharadwaj, K.K.: Fuzzy Hierarchical Censored Production Rules System, *International Journal of Intelligent Systems,* John wily & sons (New York) Vol.11, No.1, (1996).

11. Quah, T.S., Tan, C.L., Raman, K.S., Srinivasan, B.: Towards Integrating Rule-Based Expert Systems and Neural Networks, *Decision Support Systems,* 17 (1996).

12. Quah, T.S., Tan, C.L., Teh, H.H., Srinivasan, B.: Utilizing a Neural Logic Expert System in Currency Option Trading, in *Hybrid Intelligent System Applications* by Cognizant Communication Corporation, Eds. Jay Liebowitz, 1996.

13. Shavlik, J.W.: Combining Symbolic and Neural Learning, *Machine Learning* 14, pp321-331 (1994).

14. Tan, C.L., Quah, T.S., Teh, H.H.:An Artificial Neural Network that Models Human Decision Making, *Computer*, March1996, pp64-70.

15. Tirri, H. Implementing Expert system Rule Conditions by Neural Networks., *New generation Computing*, 10 ,pp. 55-71(1991).

16. Towell, G.W., Shavlik, J.W.: Knowledge-based Artificial Neural Networks, *Artificial Intelligence* 70, pp119-165(1993).

Goal-Directed Reinforcement Learning Using Variable Learning Rate

Arthur P. de S. Braga, Aluizio F. R. Araújo

Universidade de São Paulo, Departamento de Engenharia Elétrica,
Av. Dr. Carlos Botelho, 1465, 13560-250, São Carlos, SP, Brazil
{arthurp, aluizioa}@sel.eesc.sc.usp.br

Abstract. This paper proposes and implements a reinforcement learning algorithm for an agent that can learn to navigate in an indoor and initially unknown environment. The agent learns a trajectory between an initial and a goal state through interactions with the environment. The environmental knowledge is encoded in two surfaces: the reward and the penalty surfaces. The former deals primarily with planning to reach the goal whilst the latter deals mainly with reaction to avoid obstacles. Temporal difference learning is the chosen strategy to construct both surfaces. The proposed algorithm is tested for different environments and types of obstacles. The simulation results suggest that the agent is able to reach a target from any point within the environment, avoiding local minimum points. Furthermore, the agent can improve an initial solution, employing a variable learning rate, through multiple visits to the spatial positions.

1 Introduction

The motivation of this work is to propose and implement an agent that learns a task, without help from a tutor, in a limited and initially unknown environment. The agent is also capable of improving its performance to accomplish the task as time passes. Such a task is concluded when the agent reaches an initially unknown goal state. This problem is defined by Koening and Simmons [8] as a goal-directed reinforcement learning problem (GDRLP) and it can be divided in two stages. The goal-directed exploration problem (GDEP) involves the exploration of the state space to determine at least one viable path between the initial and the target states. The second phase uses the knowledge previously acquired to find the optimal or sub-optimal path.

Reinforcement learning (RL) is characterized by an agent that extracts knowledge from its environment through trial-and-error [6] [11]. In this research, learning is accomplished through a RL method that takes into consideration two distinct appraisals for the agent state: the reward and penalty evaluations. On one hand, a reward surface, modified whenever the agent reaches the target, indicates paths the agent can follow to reach the target. On the other hand, the penalty surface, modified whenever the agent reaches an obstacle, is set up in a way that the agent

F. Moreira de Oliveira (Ed.): SBIA'98, LNAI 1515, pp. 131-140, 1998.

can get rid of obstacles on its way. Thus, the reward surface deals basically with planning [12] [14] whilst the penalty surface deals mainly with reactive behavior [4] [13]. The composition of the two surfaces guides the agent decisions avoiding the occurrence of any local minimum. The two surfaces are constructed employing temporal difference (TD) as the RL learning strategy.

The tests consist in simulating a point robot within an indoor environment. Initially the robot does not have any information about environment and about the goal state. Hence, the main objective of this work is to propose an algorithm that allows the robot to reach the target from any position within the environment through an efficient path. The tests involve obstacles with different levels of complexity. The results suggest that the target is reached from any position within the environment and the chosen path is improved during the second stage of the problem solution. Despite the final trajectory is better than that one initially generated, the final path is not necessary the very best one. Moreover, the improvement takes very long to be reached.

This paper is structured as follows. The next section presents the task of this work. Section 3 introduces the RL algorithm used. The exploration stage is treated in Section 4 and the improvement of the resulting paths is discussed in Section 5. Finally, Section 6 presents the conclusions.

2 Navigation in an Indoor and Initially Unknown Environment

The proposed environment is based on the physical structure of our laboratory: a room divided into two compartments with a single passage between them (Figure 1.a). In this figure, the circles represent initial training states and the "X" represents a particular target. After the learning phase, the robot should reach the target from any point within the environment. The algorithm was also tested for different shapes of obstacles immersed into a single-room in order to evaluate how well the algorithm generalizes its achievements. These tests are based on the experiments proposed by Donnart and Meyer [5] and Millán [9]. The different environments are the following: a free room, a room with a barrier, a room with an U obstacle and a labyrinth.

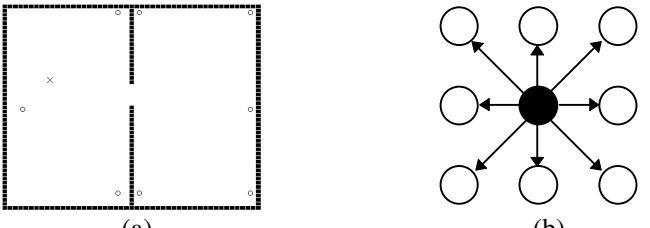


Fig. 1. (a) Sketch of the chosen environment. (b) A particular state (in black) and its neighborhood: circles represent states and arrows are possible actions

All simulated environments are taken as a discrete and limited space state in which each state has eight neighbor states (Figure 1.b). An agent action is a movement from a given state to one of its neighbors.

3 Using TD to Acquire Knowledge about the Environment

The navigation problem can be viewed as a decision tree in which each node corresponds to a possible state s. The search in this tree is heuristically guided by the evaluations for any possible action a the agent can take from the state s. The agent has to learn how to evaluate the actions from scratch in autonomous way using only environmental feedback. Hence, RL is a suitable approach to solve GDRLP.

In RL, the agent receives a reinforcement signal r from its environment that quantifies the instantaneous contribution of a particular action in the state s to reach the goal state. Then, the evaluation of a state s under a policy π, denoted by $V^{\pi}(s)$, estimates the total expected return starting the process from such a state and following that mentioned policy [11]. The learning process takes into consideration three types of situations: free states, obstacle states, and goal state. A free state is defined as a state in which the agent does not reach the goal (goal state) or an obstacle (obstacle state). Thus, Araújo and Braga [1] proposed the following definition for the reinforcement signal:

$$r(s_t, a, s_{t+1}) = \begin{cases} +1, & \text{goal state;} \\ -1, & \text{obstacle state;} \\ 0, & \text{free state.} \end{cases} \quad (1)$$

This representation is derived from the two most common types of representations found in the literature: the goal-reward representation [10] and the action-penalty representation [3]. The former indicates the states to guide the agent towards the goal. Consequently, it is a very sparse representation because, very often, there is only a single target. The latter is less sparse than the first representation because the occurrence of obstacles is much more common than the presence of goals. Even though, this representation does not direct the agent towards the target.

The learning strategy for solving this GDRLP is the temporal difference (TD) as used by Barto *et al.* [2]. This technique is characterized for being incremental and for declining a model of world. The chosen learning rule is the following [11]:

$$\Delta V_t^{\pi}(s_t) = \alpha\left[r_{t+1} + \gamma V_t^{\pi}(s_{t+1}) - V_t^{\pi}(s_t)\right] \quad (2)$$

where: $V^{\pi}(s_t)$ is the evaluation of the state s; $\Delta V^{\pi}(s_t)$ is the updating value of the evaluation of the state s_t; s_t is the state at time t; r_{t+1} is the feedback from the environment at time $t+1$; α is the learning rate ($0 \le \alpha \le 1$); γ is the discount factor ($0 \le \gamma \le 1$).

The agent learns the navigation task accumulating knowledge into two surfaces: a reward and a penalty one. The reward surface does not explicitly map obstacles, it

is modified following the identification of the goal in the environment. The penalty surface holds information that indicates the proximity between the agent and any obstacle. The composition of both surfaces guides the robot towards the target. The learning algorithm is shown in Figure 2.

```
Initialize with zeros all states in both surfaces;
Repeat
    Take the agent to the initial position;
    Repeat
        Select an action a according to the policy π;
        Execute the action a and take the agent to its new state s_{t+1};
        If the new state is an obstacle
            Update the penalty evaluation for s_t according to (2).
        Store the current state in the current trajectory;
    While the agent does not find an obstacle or a target;
    If the new state is a target
        Update the reward evaluation for the whole memorized trajectory
        according to (2);
        Delete the last trajectory from the agent memory.
While the agent does not find the target for a pre-determined number of times.
```

Fig. 2. The learning algorithm in procedural English.

The agent policy π is inspired in the potential field methods [7]:

- In 85% of the attempts the algorithm chooses the action with larger combined value (reward + penalty);
- In 10% of the attempts the algorithm chooses the action that takes the agent to the opposite state with the largest penalty;
- In 5% of the attempts the algorithm chooses the action with the largest reward.

The timing to update each one of the surfaces is different. The penalty surface is modified whenever the agent changes its state and the algorithm takes into consideration only the current state and its successor. The changes follow every single visit to a particular state. The reward surface is updated whenever the target is encountered. The modification involves all the states the agent visited to reach the goal. This trajectory is stored by the agent while it tries to reach the goal.

The algorithm above was implemented in MATLAB® where the graphic user interface (GUI) permitted to visualize some behaviors of the agent. The two next sections give the details about the results obtained in the two stages of GDRLP.

4 Solving GDEP

The objective of the first stage of GDRLP is to explore the workspace to generate trajectories, not necessarily the shortest ways, between the initial states and the goal state. The algorithm is tested in five different environments: layout of our laboratory, free room, room with barrier, room with U obstacle and labyrinth.

The learning stage generates surfaces as illustrated in Figures 3.a and 3.b. The reward surface assigns values between zero and one to each visited state in the environment. These values represent the proximity to the goal state. The penalty surface maps the obstacles positions that the agent tried to visit. The penalty evaluations range from -1 to zero and such values represent the proximity to obstacles. Note that the reward surface is smoother than the penalty one.

Figure 3.c shows a number of trajectories obtained after initial exploration of the first environment. Within the explored area, the trajectory generated by the agent is a viable connection between an initial point and the target but, generally, it is not the best path. The results shown in Figure 3.c illustrate that certain parts of the trajectories are repeated for different starting points. Thus, when leaving from a nontrained initial point, the agent moves along the state space to reach a point that had been visited during the training stage. From that point onwards, the robot follows the trajectory to which that point belongs, the agent stops its movement after reaching the goal.

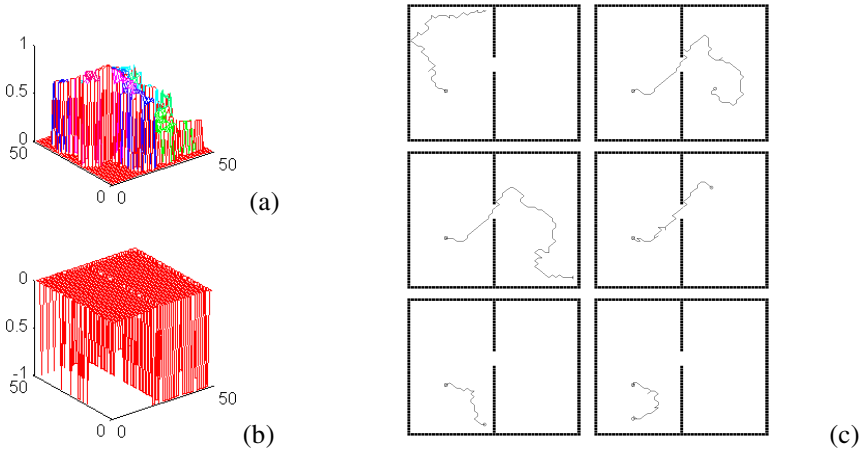


Fig. 3. (a) A reward surface learned by the agent. (b) A penalty surface learned by the agent. (c) Trajectories obtained with the TD algorithm: the left-hand side shows initial trained points and the right-hand side initial nontrained points. For all trajectories the goal state is located at the center of the left-hand side part of the environment.

Figures 4.c and 4.f show agent paths in one-room environment with and without a barrier. The corresponding reward surfaces for visited states in both cases (Figures 4.a and 4.d) are similar, even so the penalty surfaces are significantly different. Figure 4.e shows many more details about the obstacles than Figure 4.b because in the first one the agent had to work further to find a viable path.

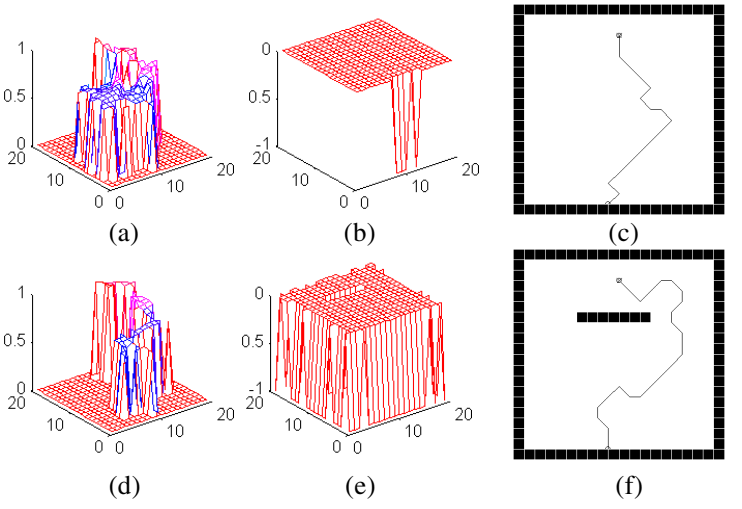


Fig. 4. Reward surface, penalty surface and obtained trajectory on a agent in a: (a)-(c) free single room; (d)-(f) single room with a barrier. The target is at the uppermost point of the path.

More complex environments are also tested, as presented in Figures 5.a and 5.b. Note that the agent has successfully reached the target in both cases.

Fig. 5. Two paths found by the agent considering: (a) An U obstacle with the initial position inside the U, (b) A labyrinth with the initial position in the right lowest corner of the figure.

Based on the proposed algorithm, the agent produces trajectories to reach the goal. However, if the evaluations are not updated, the trajectories remain be the same ones, and no decrease in the number of visited states occurs. A mechanism to suitably modify the reward surface to accomplish the second stage of GDRLP is presented in the next section.

5 Solving the Second Stage of GDRLP

At this point, the agent already has coded into its two surfaces the information that allows it to navigate appropriately along the areas visited during the exploration

stage. As mentioned before, the evaluation of each state is updated only once. If that constraint is lifted, the multiple visitation originates “isles” in the reward surface (Figure 6.a). As a consequence, the agent does not have suitable reward evaluation in the area between isles then the agent is arrested in a loop as sketched in Figure 6.b. The penalty surface, with its local update scheme, does not have significant change in this extra training.

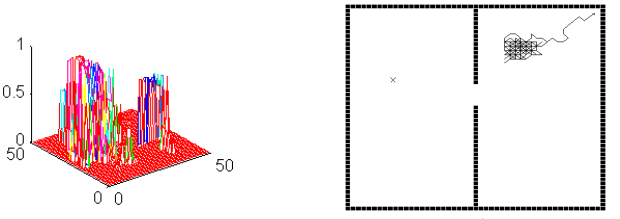


Fig. 6. (a) “Isles” generated in the reward surface. (b) An agent get stuck due to multiple visitation during the training stage.

The trajectory pursued by the agent during the training stage (Figure 7.b) explains the occurrence of isles in the reward surface. A particular state, visited a number of times in a same trajectory, has its evaluation modified proportionally to the number of visits. Thus, very often this evaluation becomes incoherent with those in the neighborhood presenting significant differences between them. Hence, one can conclude that the modifications in the reward surface should become smother in the current visit than the change in the previous visit in order to avoid surfaces as showed in Figure 7.a.

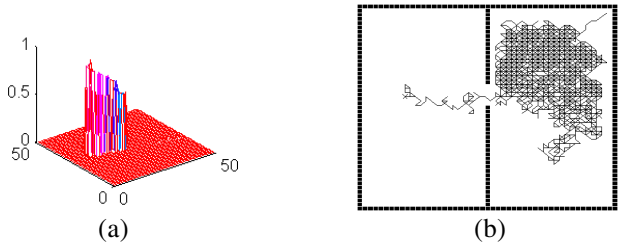


Fig. 7. (a) The reward surface for learning with multiple visitation. (b) The corresponding trajectory to reach the goal.

In sum, the constrain of updating just one time the evaluation of each state guarantees the solution of GDEP. However, this restriction does not allow the agent to alter the learned trajectory in a way to get a better solution. The possible solution is to adopt a variable learning rate in the algorithm.

The learning rate determines the size of the step to update the evaluations during the training stage. It is adopted a variable learning rate per state that is initially high and diminishes as function of the number of visits. The decreasing of the learning rate should be abrupt to avoid the occurrence of the isles in the reward surface. The expression used for the learning rate to update the reward surface is:

$$\alpha(s) = \alpha_2 \left(\frac{\alpha_1}{\alpha_2} \right)^{\frac{k(s)-1}{k(s)}} \quad (3)$$

where[1]: $\alpha(s)$ is the learning rate of state s; α_1 is the minimum value of all learning rates ($\alpha_1 = 0.1$); α_2 is the maximum value of all learning rates ($\alpha_2 = 0.9998$); k(s) is the number of visits to state s in trials that reached the goal.

The choice of an action is based on a composition of the reward and the penalty. Thus, the policy [2] becomes:

- In 60% of the attempts the algorithm chooses the action with the largest combined value (reward + penalty);
- In 10% of the attempts the algorithm chooses the action that takes the agent to the opposite state with the largest penalty;
- In 5% of the attempts the algorithm chooses the action with the largest reward;
- In 25% of the attempts the algorithm chooses the action randomly.
- In case of tie the choice is random.

The first three components of the policy provides the agent with a tendency to follow the growth of reward evaluation and to avoid the growth of penalty evaluation. The fourth component, however, adds a random component to the agent behavior to allow variations in the learned paths. The randomness allows to find shorter trajectories as long as the number of training cycles[3] increases.

During the training cycles, the penalty surface changes slightly, i.e., the learning basically occurs in the reward surface. Following the increase in the number of training cycles, the reward surface maps more states and becomes smoother (Figure 8).

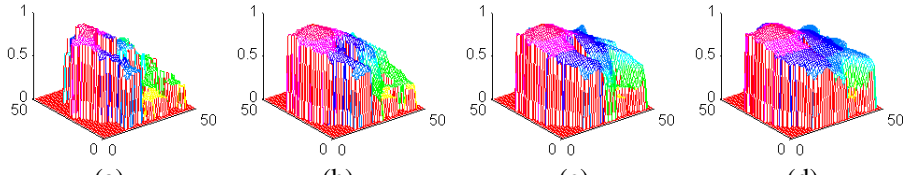


Fig. 8. A reward surface after: (a) 1 training cycle, (b) 100 training cycles, (c) 1000 training cycles and (d) 5574 training cycles.

The trajectories in Figure 9 exemplify the improvement of the agent performance (Figure 8). During these tests, the policy becomes that one explained in Section 3.

---

[1] The parameters adopted in the learning rate calculus are chosen heuristically.

[2] The distribution used in the policy tries an tradeoff between a determinist and a random behavior and it was selected heuristically.

[3] In simulations, a training cycle corresponds to the random selection and training of the eight initial points and a final state as in Figure 1.a.

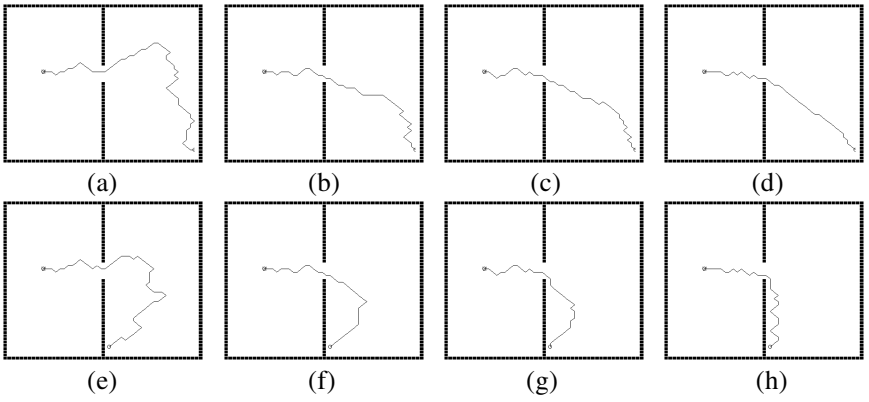


Fig. 9. (a)-(d) Improvement of the agent performance between an initial point and the goal state and (e)-(h) Improvement of the agent performance between an initial point and the goal after 1, 100, 1000 and 5574 training cycles. The final point is placed at the left-hand side of the room.

Figure 10 shows that after ten thousand training cycles the trajectories in other environments are also better then before, however they are not the shortest paths.

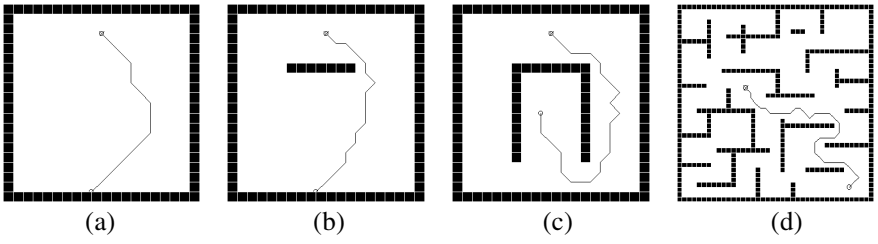


Fig. 10. (a)-(d) Trajectories followed by the agent after 10000 stages in the environments presented in Figures 4 and 5.

6 Conclusions

This paper proposed an agent that uses reinforcement learning methods to guarantee its autonomy to navigate in its state space. The reinforcement signal from the environment is encoded into the reward and penalty surfaces to endow the agent with the ability to plan and to behave reactively. The agent solves GDRLP in which an exploration stage finds a solution to the proposed task and after that the agent performance is improved following further training.

The agent performance is assessed using three out of four evaluations methods proposed by Wyatt *et al*. [15]. The internal estimates of the performance, embodied in the two surfaces, are able to represent the action repercussions. The qualitative analysis of the robot behavior shows that it improves its performance with further training. Finally, the quantitative external analysis of performance results in shorter paths since when the robot finds its goal for the first time.

The agent solves GDRLP, even so the time to reach a sub-optimal solution is quite long. It is necessary a great amount of training cycles to generate trajectories better than the early ones. The authors are working for learning strategies that dynamically modify the policy, so the randomness in agent behavior decreases with the increase of visitations to states of the environment.

References

1. Araújo, A.F.R., Braga, A.P.S.: Reward-Penalty Reinforcement Learning Scheme for Planning and Reactive Behavior. In Proceedings of IEEE International Conference on Systems, Man, and Cybernetics (1998).
2. Barto, A. G., Sutton, R. S. and Anderson, C. W.: Neuronlike adaptive elements that can solve difficult learning control problems. IEEE Transactions on SMC. V. 3, N. 5, pp: 834-846. (1983).
3. Barto, A. G.; Bradtke, S. J. and Singh, S. P.: Learning to act using real-time dynamic programming. Artificial Intelligence. V. 73, N. 1, pp: 81-138. (1995).
4. Brooks, R. A.: A robust layered control system for a mobile robot. IEEE Journal of Robotics and Automation. V. RA-2, N. 1, pp: 14-23. (1986).
5. Donnart, J. -Y. and Meyer, J. -A. Learning reactive and planning rules in a motivationally autonomous animat. IEEE Transactions on SMC, V. 26, N. 3, pp: 381-395. (1996).
6. Kaelbling, L. P., Littman, M. L. and Moore, A. W. Reinforcement learning: A survey. Journal of Artificial Intelligence Research, V. 4, pp: 237-285. (1996).
7. Khatib, O. Real-time obstacle avoidance for manipulators and mobile robots. In Proceedings of IEEE International Conference on Robotics Automation. St. Louis, MO. pp: 500-505. (1995).
8. Koenig, S. and Simmons, R. G. The effect of representation and knowledge on goal-directed exploration with reinforcement learning algorithms. Machine Learning, V. 22, pp: 227-250. (1996).
9. Millán, J. del R. Rapid, safe, and incremental learning of navigation strategies. IEEE Transactions on SMC, V. 26, pp: 408-420. (1996).
10. Sutton, R. S. Integrated architectures for learning, planning and reacting based on approximating dynamic programming. In Proceedings of International Conference on Machine Learning. pp: 216-224. (1990).
11. Sutton, R.S. and Barto, A. An Introduction to Reinforcement Learning. Cambridge, MA: MIT Press, Bradford Books. (1998).
12. Thrun, S., Moeller, K. and Linden, A. Planning with an Adaptive World Model. In Advances in Neural Information Processing Systems (NIPS) 3, D. Touretzky, R. Lippmann (eds.), Morgan Kaufmann, San Mateo, CA. (1991).
13. Whitehead, S. D. and Ballard, D. H. Learning to perceive and act by trial and error. Machine Learning, V. 7, pp: 45-83. (1991).
14. Winston, P. H. Artificial Intelligence. Reading, UK: Addison Wesley. (1992).
15. Wyatt, J., Hoar, J. and Hayes, G. Design, analysis and comparison of robot learners, accepted for publication in Robotics and Autonomous Systems: Special Issue on quantitative methods in mobile robotics, Ulrich Nehmzow, Michael Recce and David Bisset (eds). (1998).

On the Relations between Acceptable Programs and Stratifiable Classes*

F. Protti and G. Zaverucha

COPPE - PESC, Universidade Federal do Rio de Janeiro
{fabiop,gerson}@cos.ufrj.br

Abstract. Acceptable programs [4] play an important role when studying termination of logic programs, in the sense that this class matches the class of non-floundering left terminating programs. Moreover, various ways of defining semantics coincide for acceptable programs. In this paper, we study inclusion relations between this class and *stratifiable* classes, e.g. locally stratified [19,15], weakly stratified [14], effectively stratified [6], and support-stratified programs [9]. In particular, we show that every acceptable program is weakly stratified.
Keywords: Logic programs, acceptable programs, stratified programs

1 Introduction

In the field of logic programming, two aspects have been extensively studied. On the one hand, several classes of logic programs that guarantee the existence of a unique stable model have been defined. To name just a few, locally stratified programs [19,15], weakly stratified programs [14], effective stratified programs [6], and support-stratified programs [9]. The class of odd-cycle free programs [9] is also worth mentioning, although it contains programs that may have more than one stable model. On the other hand, a theoretical basis for studying termination of logic programs with the Prolog selection rule has been investigated. Acyclic programs [7] and acceptable programs [4] are left terminating [18,**?**]. The latter one matches the class of non-floundering left terminating programs, and various ways of defining semantics coincide for it. Acceptable programs are interesting not only from a logic programming theorectical point of view, but also because of their applications in neural networks that implement logic programming [12] and propositional inductive logic programming [11].

The definition of acceptable programs uses the concept of level mapping (a function from ground atoms to natural numbers) [7], which has a direct correspondence with acyclicity. Intuitively, a program P is acceptable if it has an "acyclic part" which guarantees its termination. More specifically, if for some level mapping and for every ground rule, the level of the head is greater than the level of atoms in a certain prefix of the body. This prefix is determined by the model I of P considered.

* This work is part of the project LOGIA / ProTeM - CC / CNPq. The authors are partially financially supported by CNPq.

F. Moreira de Oliveira (Ed.): SBIA'98, LNAI 1515, pp. 141–150, 1998.

In this work, we study inclusion relations between the classes above, derived from those two approaches (stability and termination). The paper is organized as follows. Section 2 contains preliminary definitions. Section 3 defines all the classes of logic programs dealt with here. In Section 4, we investigate inclusion relations between what we conventioned to call *stratifiable* classes. In particular, we show examples relating support-stratified/odd-cycle free programs and weakly/effectively stratified programs. Section 5 is devoted to revisit the universe of stratifiable classes, in order to localize acceptable programs in it. This is done by means of some examples and an interesting result (Theorem 20), which states that every acceptable program is weakly stratified (the converse is not true). The last section contains our conclusions, where we conjecture a relation involving the concepts of acceptability, weakly stratifiability and left termination.

2 Preliminaries

Many definitions, terminologies and notations here come from [9] and [1]. A *program* P is a finite set of first order rules of the form

$$\alpha \leftarrow l_1 \wedge l_2 \wedge \ldots \wedge l_n$$

where $n \geq 0$, α is an atom, and l_i is a literal. If r is a rule, $head(r)$ is the atom in its head, and $body(r)$ the set of literals (*subgoals*) in its body. We define, for a set R of rules, $head(R) = \{head(r)\}_{r \in R}$ and $body(R) = \cup_{r \in R} body(r)$.

We denote by $base(P)$ the Herbrand base of P, P_{ground} the program obtained by instantiating the variables in P in all possible ways, and P_{comp} the Clark's completion of P (which − informally − strengthens the program by interpreting implications as equivalences). We say that P_{ground} consists of *ground rules*. If L is a set of literals, then $\neg L = \{\neg l \mid l \in L\}$. An element $\neg\beta$ of the set $\neg base(P)$ is called an *assumption*. A set H of assumptions constitutes a *hypothesis*.

Let H be a hypothesis and P a program. We define P_H^+ as the positive ground program obtained from P as follows: a) delete the assumptions in H from P_{ground}; b) from the resulting program after applying step a), delete all rules containing negative subgoals. We will then say that H *supports* an atom α in P (denoted by $H \rightarrow \alpha$) if $P_H^+ \models \alpha$. We denote by $H^{\rightarrow}$ the set of atoms H supports in P. Also, we say that H *minimally supports* α (denoted by $H \rightarrow_{min} \alpha$) if no subset of H supports α. A hypothesis H *attacks* another hypothesis H' in a program P (denoted by $H \gg H'$) if $H^{\rightarrow} \bigcap \neg H' \neq \emptyset$. In other words, H supports an atom whose negation is in H'. We also say that H *minimally attacks* H' (denoted by $H \gg_{min} H'$) if it is a minimal hypothesis such that $H \gg H'$.

An *interpretation* for a program P is a set I of literals such that $I \subseteq base(P) \cup \neg base(P)$ and $I \cap \neg I = \emptyset$. We define $I^+ = I \cap base(P)$ and $I^- = I \cap \neg base(P)$. An interpretation I is *total* if $I^+ \cup \neg I^- = base(P)$. If $\alpha \in I$, we write $I \models \alpha$ or $I \not\models \neg\alpha$. If $\neg\alpha \in I$, we write $I \not\models \alpha$ or $I \models \neg\alpha$. An interpretation I for a program P is said *supported* if $I = I^- \cup (I^-)^{\rightarrow}$.

An interpretation I is a *(partial) model* for a program P if $P \cup I$ is consistent. We also say that I is a *stabel model* if I is supported and total. Another important kind of model is the *well-founded model*, whose definition can be found in [20].

A *least model* for a program P is one with minimum number of positive literals. It is well known that every positive logic program has a least model, and this model is unique.

Let P be a logic program. The *precedence graph* D_P *for* P has as vertex set the atoms ocurring in P. Its edge set contains signed edges, and it is defined in the following way: for every rule whose head is α and whose body contains the positive (resp. negative) literal l, there is a *positive* (resp. *negative*) edge from l to α in D_P.

The *ground precedence graph* G_P *for* P is defined in a similar way. Its vertex set consists of the ground atoms ocurring in P_{ground}, and for every ground rule whose head is α and whose body contains the positive (resp. negative) ground literal l, there is a *positive* (resp. *negative*) edge from l to α in G_P.

The *minimal attack graph* $A_P = (V, E)$ *for* P is the directed graph with vertex set $V = \{$ hypothesis $H \mid$ there is α in P_{ground} s.t. $H \rightarrow_{min} \alpha\}$, and edge set $E = \{(H_1, H_2) \mid H_1 \gg_{min} H_2\}$.

Given a program P, a *level mapping* is any function *level* from $base(P)$ to the natural numbers. This function is extended to ground negative literals by setting $level(\neg\alpha) = level(\alpha)$.

A *stratification (local stratification)* of a program P is a partition $P = P_1 \cup \ldots \cup P_n$ of the set of rules (ground rules) of P such that:
a) there are no positive edges from the literals in $body(P_{i+1} \cup \ldots \cup P_n)$ to the atoms in $head(P_i)$ in the graph D_P (G_P), for $i = 1 \ldots n-1$;
b) there are no negative edges from the literals in $body(P_i \cup \ldots \cup P_n)$ to the atoms in $head(P_i)$ in the graph D_P (G_P), for $i = 1 \ldots n$.

In this definition, P_1 may be empty. When some relations are undefined, it is assumed that they are defined in P_1 by the empty set of clauses.

3 Classes of programs

Now we are able to define the classes focused in this work.

Definition 1. [7] A program P is called *acyclic* if there is a level mapping *level* such that for every rule $\alpha \leftarrow l_1 \wedge l_2 \wedge \ldots \wedge l_m$ in P_{ground} we have $level(\alpha) > level(l_i)$, for $i = 1 \ldots m$.

Definition 2. [19,15] A program is called *stratified* if it admits a stratification.

Definition 3. [13,17] A program P is called *call-consistent* if its dependency graph D_P has no cycle with an odd number of negative edges.

In default theory, stratified programs correspond to ordered default theories [10], and call-consistent programs correspond to even default theories [16,8]. Stratifiability guarantees the existence of at least one stable model, but one can show that call-consistency is sufficient for it.

Definition 4. [19,15] A program is called *local stratified* if it admits a local stratification.

Locally stratified programs guarantee the existence of unique stable models. Przymusinska and Przymusinski [14] propose the concept of *weak stratification*, an adaptation of local stratification to deal with programs that are not locally stratified but still have a unique stable model. The reader can find the necessary concepts for the definition in the cited paper (we omit them here because of lack of space).

Definition 5. [14] A logic program P is *weakly stratified* if it has a weakly perfect model and all of its strata S_k consist only of trivial components or, equivalently, when all of its layers L_k are positive logic programs. In this case, we call the set of program's strata the *weak stratification* of P.

Another proposal is the concept of *effective stratification* due to Bidoit and Froidevaux, which extends weak stratification and is closely related with well-founded models.

Definition 6. [6] A program P is *effectively stratified* if its well-founded model is total.

Support-stratified and *odd-cycle free* programs are also classes of interest. Every support-stratified program has a unique stable model, and every odd-cycle free program has at least one stable model. We define these classes in terms of the minimal attack graph.

Definition 7. [9] A program P is *support-stratified* if A_P is acyclic.

Definition 8. [9] A program P is *odd-cycle free* if every cycle in A_P is of even length.

Finally, we define the class of *acceptable programs* [4], a generalization of acyclic programs. Denote by Neg_P the set of atoms ocurring in negative literals in a ground program P, Neg_P^* the set of atoms in P on which the atoms in Neg_P depend, and P^- the program consisting of all rules of P whose head is contained in Neg_P^*. Let us call *well-behaved negation model* a model I of P with the property that I restricted to Neg_P^* is a model of $(P^-)_{comp}$.

Definition 9. [4] A program P is *acceptable* if there exist a level mapping *level* for P and a well-behaved negation model I of P such that for every ground rule $\alpha \leftarrow l_1 \wedge l_2 \wedge \ldots \wedge l_n$ in P_{ground} the following implication holds for $i = 1 \ldots n$:

$$if\ I \models l_1 \wedge l_2 \wedge \ldots \wedge l_{i-1}\ then\ level(\alpha) > level(l_i).$$

4 Inclusion relations involving stratifiable classes

Theorem 10. (see [1])
a) Every acyclic program is locally stratified, and the latter class is strictly larger than the former.
b) Every stratified program is locally stratified, and the latter class is strictly larger than the former.

In the universe of datalog programs (programs without function symbols), the class of acyclic programs is properly contained in the class of stratified programs. In the general case, however, there exist acyclic programs which are not stratified.

Theorem 10 is in fact a corollary of the following straightforward result (see, for instance, [3]), which shows a way of characterizing acyclic and (locally) stratified programs presented via associated graphs.

Theorem 11. *Let P be a logic program. Then:*
i) P is acyclic iff G_P has no cycles.
ii) P is stratified iff D_P has no cycle with one or more negative edges.
iii) P is locally stratified iff G_P has no cycle with one or more negative edges.

In what follows, we present some known results relating stratifiable classes.

Theorem 12. [13,17] *Every stratified program is call-consistent, and the latter class is strictly larger than the former.*

Theorem 13. [9] *Every call-consistent program is odd-cycle free, and the latter class is strictly larger than the former.*

In fact, the result of Theorem 12 is an easy consequence of the definitions. More interesting is the fact that odd-cycle free programs are more general than call-consistent programs (Theorem 13), even though slightly.

Theorem 14. [9] *Every locally stratified program is support-stratified, and the latter class is strictly larger than the former.*

Theorem 15. [9] *Every support-stratified program is odd-cycle free, and the latter class is strictly larger than the former.*

Theorem 16. [14] *Every locally-stratified program is weakly stratified, and the latter class is strictly larger than the former.*

Theorem 17. [6] *Every weakly stratified program is effectively stratified, and the latter class is strictly larger than the former.*

Now, by means of examples, we analyze interactions involving weakly stratified, effectively stratified, support-stratified, and odd-cycle free programs.

Example 1. Not every weakly stratified program is odd-cycle free.

$b \leftarrow .$
$c \leftarrow \neg a \wedge \neg b$
$d \leftarrow \neg c$
$a \leftarrow \neg d$

The bottom stratum $S(P)$ of the program P above consists only of the minimal component $\{b\}$. The bottom layer is given by $L(P) = \{b \leftarrow .\}$ and has the least model $M = \{b\}$. The reduced program P/M is

$d \leftarrow \neg c$
$a \leftarrow \neg d$

By applying two more steps of the construction, we conclude that P is weakly stratified. However,

$$\{\neg a, \neg b\} \gg_{min} \{\neg c\} \gg_{min} \{\neg d\} \gg_{min} \{\neg a, \neg b\}$$

is an odd cycle in A_P, that is, P is not odd-cycle free.

As a corollary of this example, not every weakly stratified program is suport-stratified, not every effectively stratified program is support-stratified, and not every effectively stratified program is odd-cycle free. □

Example 2. Not every support-stratified program is weakly stratified.

$b \leftarrow a$
$a \leftarrow b$
$c \leftarrow \neg a$
$a \leftarrow \neg c \wedge b$

It is easy to see that the program P above is support-stratified. However, the bottom layer $L(P)$ of P is P itself, and P is not a positive program. Therefore, P is not weakly stratified.

As a corollary of this example, not every odd-cycle free program is weakly stratified. □

Example 3. Not every odd-cycle free program is effectively stratified.

$b \leftarrow \neg a$
$c \leftarrow \neg b$
$d \leftarrow \neg c$
$a \leftarrow \neg d$

The cycle $\{\neg a\} \gg_{min} \{\neg b\} \gg_{min} \{\neg c\} \gg_{min} \{\neg d\} \gg_{min} \{\neg a\}$ is of even length. Therefore, P is odd-cycle free. However, P is not effectively stratified, since its well-founded model is empty. □

As far as we know, it still remains an open question to decide whether the class of support-stratified programs is contained in the class of effectively stratified programs.

5 Localizing Acceptable programs

In this section, we show theorems and examples illustrating the relationships between acceptable programs and the classes focused in Section 4.

Theorem 18. [4] *Every acyclic program is acceptable, and the latter class is strictly larger than the former.*

Example 4. Not every locally stratified program is acceptable.

$p \leftarrow q$
$q \leftarrow p$
$r \leftarrow .$

Clearly, the program above is locally stratified. Moreover, it is not acceptable, since for any level mapping f we cannot have $f(p) > f(q)$ and $f(q) > f(p)$.

As a corollary of this example, the classes of support-stratified programs, odd-cycle free programs, weakly stratified programs and effectively stratified programs cannot be contained in the class of acceptable programs. It is also easy to see that not every stratified program is acceptable. □

Example 5. Not every acceptable program is locally stratified.
Let H be an acyclic graph. Consider the program P below:

$move(a, b) \leftarrow .$ (for all edges (a, b) in H)
$win(x) \leftarrow move(x, y) \wedge \neg win(y)$

As shown in [4], P is acceptable. However, P is not locally stratified, since any ground rule of the form $win(a) \leftarrow move(a, a) \wedge \neg win(a)$ implies the existence of a cycle with a negative edge in G_P.

As a corollary of this example, not every acceptable program is stratified. □

Example 6. Not every acceptable program is odd-cycle free.

$a \leftarrow .$
$q \leftarrow \neg a \wedge \neg p$
$r \leftarrow \neg a \wedge \neg q$
$p \leftarrow \neg a \wedge \neg r$

Let us show that the program P above is acceptable. Recall from the definition that $Neg_P = \{a, p, q, r\}$ and $Neg_P^* = base(P)$. Thus, $P^- = P$. Consider $I = \{a, \neg p, \neg q, \neg r\}$. Notice that I is a model of $(P^-)_{comp}$. Consider also a level mapping $level$ such that

$$level(p) = level(q) = level(r) > level(a).$$

Observe that the implication

$$I \models l_1 \wedge l_2 \wedge \ldots l_{i-1} \Rightarrow level(\alpha) > level(l_i)$$

holds for all rules $\alpha \leftarrow l_1 \wedge l_2 \wedge \ldots l_n$ of P.

On the other hand, P is not odd-cycle free because of the existence of the cycle $\{\neg a, \neg p\} \gg_{min} \{\neg a, \neg q\} \gg_{min} \{\neg a, \neg r\} \gg_{min} \{\neg a, \neg p\}$ in A_P.

As a corollary of this example, not every acceptable program is support-stratified. □

The next theorem relates acceptable programs and effectively stratified programs. It is an easy corollary of a result of Apt and Pedresci [4].

Theorem 19. *Every acceptable program is effectively stratified, and the latter class is strictly larger than the former.*

Proof. In [4] it is shown that if a program P is acceptable, then the well-founded model of P is total. Therefore, P is effectively stratified. Example 4 shows an effectively stratified program which is not acceptable. □

In fact, a stronger result can be proved:

Theorem 20. *Every acceptable program is weakly stratified, and the latter class is strict larger than the former.*

Proof. Let P be an acceptable program with respect to a level mapping *level* and to a model I of $(P^-)_{comp}$. We claim that:

(1) all components of the bottom stratum $S(P)$ of P are trivial. Otherwise, suppose that there exists a minimal component $C \subseteq S(P)$ such that the atoms in C form a cycle passing through a negative edge. Since C is minimal, the bottom layer $L(P)$ of P has a subset R of rules satisfying $head(R) = C$ and $body(R) \subseteq C \cup \neg C$ (that is, the rules in R consist only of atoms belonging to C).

Let us write a rule $r \in R$ in the following way:

$$r : \alpha^r \leftarrow l_1^r \wedge l_2^r \wedge \ldots \wedge l_{n_r}^r$$

Since P is acceptable, we have $level(\alpha^r) > level(l_1^r), \forall r \in R$. Let α_1^r be the atom corresponding to literal $l_1^r, \forall r \in R$. Since $head(R) = C$, there is a rule $s \in R$ such that $\alpha^s = \alpha_1^r$. Therefore,

$$level(\alpha^r) > level(l_1^r) = level(\alpha^s) > level(l_1^s) = level(\alpha_1^s).$$

Let $p \in R$ such that $\alpha_p = \alpha_1^s$, and continue this process. Since $body(R) \subseteq C \cup \neg C$, notice that the sequence

$$level(\alpha^r) > level(\alpha^s) > level(\alpha^p) > \ldots$$

has repeated elements, a contradiction.

(2) if the bottom layer $L(P)$ of P is not empty, then the reduced program P/M, where M is the least model of $L(P)$, is also acceptable. In what follows, we present an argument supporting this claim. Note that the only way to destroy the acceptability of P/M is to remove from some rule $\alpha \leftarrow l_1 \wedge l_2 \wedge \ldots \wedge l_n$ in P a subgoal l_i for which $M \models l_i$, $I \models l_1 \wedge l_2 \wedge \ldots \wedge l_{i-1}$, and $I \not\models l_i$. Suppose that this happens. We have two cases:

(2.1) l_i is a positive literal. Then, since $l_i \in C$ and $l_i \in M$, we have that $P \models l_i$, i. e., l_i can be proved. This leads to a contradiction (it is impossible that $I \not\models l_i$).

(2.2) l_i is a negative literal. Let us write $l_i = \neg\beta$. Then $M \not\models \beta$ and $I \models \beta$. Since I is a model of $(P^-)_{comp}$, we have that β can be proved. This leads to another contradiction (it is impossible that $M \not\models \beta$).

By (1) and (2), inductively, it is possible to construct a weakly perfect model for P: every P_k is acceptable and every S_k is not empty and consists only of trivial components, whenever P_k is non-empty. That is, P is indeed weakly stratified. Example 4 shows that acceptable programs are properly included in weakly stratified programs. □

Figure 1 shows the inclusion relations involving the classes discussed in this paper. Arrows mean proper inclusions. The dashed arrow indicates a possibility.

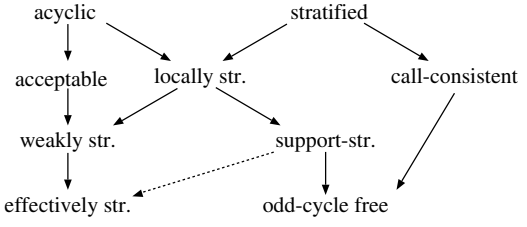


Fig. 1. Inclusion relations involving stratifiable classes and acceptable programs.

6 Conclusions

In this work, we presented a study on inclusion relations between some classes of programs derived from two research approaches in logic programming (stability and termination). First, we studied (Section 4) how stratifiable classes are related. A contribution of Section 4 was to show, by means of examples, the interactions involving weakly stratified, effectively stratified, support-stratified, and odd-cycle free programs. There still remains the possibility that support-stratified programs are effectively stratified. Finally, in Section 5 we localized acceptable programs in the universe of stratifiable classes. The main contribution of this work was to show that every acceptable program is weakly stratified, unifying the two approaches mentioned above.

At this point, we may conjecture a stronger relation involving acceptable programs and weakly stratified programs. The definition of weakly stratifiability does not include, of course, any kind of order of computation as in acceptability. However, we believe that acceptable programs are exactly weakly stratified programs for which it is possible to set an order of computation inside the rules so that they become left terminating according to the Prolog selection rule.

References

1. K. R. Apt and R. Bol. Logic programming and negation: a survey. *Journal of Logic Programming* 19-20 (1994) 9-71.

2. K. R. Apt, H. Blair, and A. Walker. Towards a theory of declarative knowledge. In: J. Minker, ed. *Foundations of Deductive Databases and Logic Programming* (Morgan Kaufmann, Los Altos, CA, 1988), pp. 89-142.
3. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*, Addison-Wesley, 1995.
4. K. R. Apt and D. Pedresci. Reasoning about termination of pure prolog programs. *Information and Computation* 106 (1993) 109-157.
5. M. Bezem. Characterizing termination of logic programs with level mappings. In: E. L. Lusk and R. A. Overbeek, eds. *Proceedings of the North American Conference on Logic Programming*, MIT Press, Cambridge, MA, 1989.
6. N. Bidoit and C. Froidevaux. Negation by default and unstratifiable logic programs. *Theoretical Computer Science* 78 (1991) 85-112.
7. L. Cavedon. Continuity, consistency, and completeness properties for logic programs. In G. Levi and M. Martelli, Eds., *Proceedings of the Sixth International Conference on Logic Programming*, pp. 571-584, MIT Press, Cambridge, 1989.
8. Y. Dimopoulos, V. Magirou and C. H. Papadimitriou. On kernels, defaults and even graphs. Technical Report MPI-I-93-264, Max-Planck-Institut für Informatik, Germany.
9. Y. Dimopoulos and A. Torres. Graph theoretical structures in logic programs and default theories. *Theoretical Computer Science* 170 (1996) 209-244.
10. D. W. Etherington. Formalizing nonmonotonic reasoning systems. *Artificial Intelligence* 31 (1987) 41-85.
11. A. Garcez, G. Zaverucha, and V. Silva. Applying the Connectionist Inductive Learning and Logic Programming System to Power System Diagnosis. *Proceedings of the IEEE / INNS International Conference on Neural Networks (ICNN-97)*, 9-12 June 1997, Houston, Texas, USA, Vol.1, pp.121-126.
12. S. Holldobler and Y. Kalinke. Toward a new massively parallel computational model for logic programming. In: *Workshop on Combining Symbolic and Connectionist Processing (ECAI'94)* (1994) pp. 1-14.
13. K. Kunen. Signed data dependencies in logic programming. *J. Logic Programming* 7 (1989) 231-245.
14. H. Przymusinska and T. C. Przymusinski. Weakly perfect model semantics for logic programs. In: *Proc. Logic Programming Conf.*, Seattle (1988) 1106-1120.
15. T. Przymusinski. On the declarative semantics of logic programs with negation. In: J. Minker, ed. *Foundations of Deductive Databases and Logic Programming* (Morgan Kaufmann, Los Altos, CA, 1988), pp. 193-216.
16. C. H. Papadimitriou and M. Sideri. Default theories that always have extensions. *Artificial Intelligence* 69 (1994) 347-357.
17. T. Sato. Completed logic programs and their consistency. *J. Logic Programming* 9 (1990) 33-44.
18. L. Sterling and Shapiro. *The Art of Prolog*, MIT Press, Cambridge, MA, 1986.
19. A. Van Gelder. Negation as failure using tight derivations for general logic programs. In J. Minker, Ed., *Foundations of Deductive Databases and Logic Programming*, pp. 149-176, Morgan Kaufmann, Los Altos, CA, 1988.
20. A. Van Gelder, R. Roth, and J. S. Schlipf. Unfounded sets and well founded semantics for general logic programs. In: *Proc. ACM SIGACT-SIGMOD Symp. on Principles of Database Systems* (1988) 221-230.

An Adaptation of Dynamic Slicing Techniques for Logic Programming*

Wamberto Weber Vasconcelos**
Dept. of Statistics & Computing
State University of Ceará
Ceará, Brazil

Marcelo A. T. Aragão***
Artificial Intelligence Laboratory
Federal University of Ceará
Ceará, Brazil

Abstract. Program slicing is a process by means of which subparts of a program with a collective meaning are automatically obtained. When the slicing takes into account the actual execution of the program, it is said to be *dynamic*. We show how some techniques for dynamic slicing of procedural languages can be adapted to logic programming.
Keywords. logic program slicing, program analysis & understanding.

1 Introduction

Program slicing is a technique for obtaining subparts of a program with a collective meaning. It was originally proposed by Weiser [19,20], in the context of procedural languages. When we slice a program, we want to automatically obtain portions of the program responsible for specific computations.

Program slicing is a useful technique for program development and reuse. Slicing can be particularly useful when debugging programs [19]. Starting from the command line where the bug was noticed, we obtain all the other commands potentially related to the problem, leaving out of the analysis the parts that do not affect the bugged part.

We have investigated the adequacy of some slicing techniques, originally proposed for procedural languages, for slicing logic programs. We aim at obtaining *executable* slices, i.e., the slices must reproduce parts of the original computations. This is in contrast with methods whose slices produced are simply subsets of commands from the original program and do not necessarily comprise an executable program. If we consider program *sum_count/3* below,

```
sum_count([],0,0) ←
sum_count([X|Xs],S,C) ←
    sum_count(Xs,SXs,CXs),
    S is SXs + X,
    C is CXs + 1
```

to sum (second argument) and count (third argument) the elements of a list (first argument), three of its executable slices are

```
sum_count([],_,_) ←
sum_count([_|Xs],_,_) ←
    sum_count(Xs,_,_)
```

```
sum_count([],0,_) ←
sum_count([X|Xs],S,_) ←
    sum_count(Xs,SXs,_),
    S is SXs + X
```

```
sum_count([],_,0) ←
sum_count([_|Xs],_,C) ←
    sum_count(Xs,_,CXs),
    C is CXs + 1
```

The underscore symbols "_" stand for *anonymous variables* [5,13], i.e. place holders for arguments which are not important for the computation. Those argument positions which are not relevant to the slice are replaced with anonymous

* Work partially carried out while the first author was a Visiting Researcher at the Institut für Informatik, Universität Zürich, Switzerland.
** Partially sponsored by CNPq grant no. 352850/96-5. Email: wvasconcelos@acm.org.
*** Sponsored by CNPq grant no. 380559/96-0. Email: aragao@lia.ufc.br.

F. Moreira de Oliveira (Ed.): SBIA'98, LNAI 1515, pp. 151-160, 1998.

variables, an approach introduced in [11]. Each of the slices above is a fully operational program computing parts of the original program. The left slice shown above performs the same list processing (traversal) as the original program; the middle slice sums the elements of the list; and the right slice counts the elements of the list.

We have adapted some of the ideas for slicing programs in procedural languages and proposed a flexible framework for logic programs. We regard our approach as "flexible" because it can be customised to any logic language, provided we have its computation rules and information on system predicates. This article contains part of the work described in [17]. We refer the reader to that reference for more details.

In the next Section we survey some of the work on program slicing, mostly for procedural languages, but we also study and compare existing research on logic program slicing. In Section 3 we outline our proposed method and propose a criterion for logic program slicing. In Section 4 we introduce our alternative proposal to represent the meaning of logic programs — this is made necessary since the same logic program construct may have different meanings depending on the way it is executed. Section 5 describes our proposal for representing different kinds of dependences within logic programs. These dependences are necessary for the appropriate removal of those unrelated (with respect to given criteria) parts of a program. In Section 6 we put forth our notion of *relevance* for logic program constructs. Section 7 describes our proposed slicing algorithm, making use of our framework. In Section 8 we draw conclusions and make final remarks on the work described here.

In this paper, variables are denoted as x, y, predicate symbols as p^n, q^n (where n is the arity predicate), constants as a, b, and terms as s, t.

2 Program Slicing

Weiser originally proposed in [19] the notion of program slicing for procedural languages, giving psychological evidence that programmers, consciously or not, break apart large programs into smaller coherent pieces. Functions, subroutines, and so on, are all *contiguous* portions of the original program with particular ends which are easily observed. However, another form of program pieces can be detected and used by experienced programmers when analysing a program: these are sets of statements related by the flow of data, not necessarily contiguous but scattered through the program.

A program slice can be seen more generally as those parts of a program that (potentially) affect the values computed at some point of interest, referred to as the slicing criterion [14]. When the slicing technique gathers information on the data- and control flow of the program taking into account an actual and specific execution (or set of executions) of it then it is said to be *dynamic*, otherwise (only statically available information being used) it is said *static* [14]. When the slicing is performed as a backward traversal of the program, starting from the statement referred to in the criterion and going back to the beginning (gathering data- and control flow information and testing the relevance of the statements in the process), then it is said to be *backward* slicing, otherwise (when slicing is performed from the beginning of the program to its end) it is said to be *forward*.

2.1 Criteria for Program Slicing

Slicing of programs is performed with respect to some criterion. Weiser [19] proposes as a criterion the number i of a command line and a subset V of program variables. According to this criterion, a program is analysed and its commands are checked for their relevance to command line i and those variables in V; a command is relevant if it tests or assigns values to variables in V or to variables used (directly or not) to test values to variables in V.

2.2 Logic Program Slicing

Slicing of logic programs has only recently been addressed. In [15,16] we are presented with a first attempt at grams. In that work, the slices obtained are (possibly non-contiguous) portions of an original Prolog program responsible for some of the computations, and they may not necessarily be executable. The slices detected are called *programming techniques*: they correspond to the pieces of code used by programmers in a systematic way when developing their programs [3].

In [6] we are presented with a static slicing method, to be offered as part of a debugging tool. The method, however, is not aimed at producing slices (i.e., subsets of commands) of the original program. Rather, it is used to slice the *proof-tree* of the logic program, eliminating those parts which are not related to the point of the program where a bug has been detected.

In [11] we are presented with a backward slicing algorithm to produce executable slices of Prolog programs. A useful contribution of that work is the introduction of *argument abstraction*: when analysing the relevance of a subgoal the algorithm may decide that certain argument are not relevant and should be left out. Irrelevant arguments are replaced with anonymous variables, thus *partially* keeping a program statement.

2.3 Why is Slicing of Logic Programs Difficult?

The commands of procedural languages have a unique mapping between their syntax and the computations they perform. This is not always the case with logic languages where some constructs may have many different possible executions. Determining the actual (or possible) computations of a program is necessary when slicing it. Slicing of logic programs is more sophisticated since it must account for this diversity of meanings for the same program.

Another distinctive feature of logic languages concerns the nature of variables. Variables in logic languages do not store values: they are references to constructs of the language (constants, other variables or complex constructs combining them). Although the operations on variables are given a logic-theoretic reconstruction in logic programming [2,9], they can be equated with assignments, tests and aliasing of procedural languages [12]. However, such operations are performed on the associated pointers rather than on the variables themselves. Variables and their operation play a fundamental role in slicing techniques; these, however, are geared towards variables of procedural languages.

3 A Method for Slicing Logic Programs

We have developed a method for slicing logic programs. It incorporates ideas of some dynamic slicing techniques geared towards procedural languages. Our method starts with a logic program Π and a criterion $\mathcal{C}$ (defined in Subsection 3.1), and initially obtains the meaning of Π (described in Section 4); by "meaning" here we refer to a procedural interpretation of the program. Using our representation for the meaning of the program we build a term dependences graph (Section 5). Using *both* the represented meaning of the program and the graph, the slicing itself takes place (Section 7). A slice Π' is a variant of Π in which subgoals may have been removed, argument positions may have been replaced with anonymous variables, and the predicates that are not relevant have been deleted.

3.1 Criteria for Logic Program Slicing

We have extended the usual syntax of logic programs in order to enable the unambiguous reference of any literal of a program. We have adopted a simple identification system by means of which each subgoal of a program is assigned a label $\langle i, j \rangle, i \geq 1, j \geq 0$, where i is the order in which the clause appears in the source program, and j is the order of the subgoal within its clause: the head goal has order 0 and the body goals have order 1,2, and so on. Without loss of generality, we will assume that our logic programs have this unique identification for each subgoal. To provide for a homogeneous treatment, we will assume that a query $G_1, \ldots, G_n$ has positions $\langle 0, i \rangle, 1 \leq i \leq n$, respectively, for its literals.

In order to refer in a unique way to any term of a literal within a logic program, we will employ the concept of *argument position* (or simply *position*) adapted from [4]:

Definition 1. Let $\langle c, i \rangle$ be the identification of the $i-th$ literal $p_j^r(t_1, \ldots, t_r)$ of the c-th clause in Π, then its k-th *argument position* (or simply *position*) is uniquely defined within Π by the tuple $\langle c, i, p_j^r, k \rangle$.

We denote positions as α and β. We can now define criteria:

Definition 2. A *criterion* $\mathcal{C}$ is the pair $\langle (G_0, \ldots, G_n)\theta, P \rangle$ where $G_0, \ldots, G_n$ is a sequence of subgoals, θ is a (possibly empty) term substitution and P is a set of argument positions of Π.

The slices of program *sum_count/3* shown previously, for instance, were obtained with respect to criteria $\langle \langle 0, 1, \texttt{sum\_count([1,5],A,B)} \rangle \emptyset, \{\alpha_1, \alpha_2, \alpha_3\} \rangle$, $\alpha_i = \langle 0, 1, \texttt{sum\_count}^3, i \rangle, 1 \leq i \leq 3$, respectively, from left to right.

4 Mode-Annotated Logic Programs

We have proposed a means to formally represent the *procedural* meaning of programs, i.e., the way the program is *executed*. Proof-trees [2, 9] and traces [13], have been proposed for this purpose, concentrating on dynamic aspects of program execution, showing the progress of the computation in terms of the unifications taking place. However, in those representations when a unification takes place we lose information on what terms *used* to be. We propose a means to incorporate the procedural meaning of logic programs in their syntax. Our proposal consists of representing the instantiation mode of variables before and after each subgoal, in explicit *mode annotations*.

According to this approach, we relate to each variable a mode describing its instantiation status: "**f**" for *free* variables; "**g**" for *ground* variables, *i.e.* variables bound to constants or (composed) terms with ground subterms only; and "**i**" for *instantiated* variables, *i.e.* variables *not free* (i.e., instantiated to partially ground terms). Copies of subgoals with similar tokens associated with their variables can be safely discarded, since they do not add any relevant information to the representation of relationships among terms. The computations of logic program constructs are represented by the changes taking place in their variables, recorded in mode-annotations:

Definition 3. A *mode-annotation* θ is a finite (possibly empty) set of pairs x/m where x is a variable and $m \in \{\mathbf{f}, \mathbf{g}, \mathbf{i}\}$ is the *instantiation mode* of x.

The subgoals of a program execution have two mode-annotations associated to each of them, one recording the instantiation status of variables *before* the subgoal execution, and another *after* it:

Definition 4. A *mode-annotated* goal $\widetilde{G}$ is the triple $\theta\, G\, \theta'$ where θ, θ' are mode-annotations.

Our goal mode-annotations should enable us to detect *changes* that have happened to the instantiation of variables. The detection and formalisation of changes allow us to represent the term dependences of a program. We employ the definition of annotated subgoals to define mode-annotated executions:

Definition 5. A *mode-annotated execution* $\widetilde{E}$ is a finite (possibly empty) sequence $\widetilde{G}_0, \ldots, \widetilde{G}_n$ of mode-annotated subgoals $\widetilde{G}_i, 0 \leq i \leq n$.

The mode-annotated execution(s) of a program can be obtained by augmenting a meta-interpreter, as suggested in [16], with failures and backtracking (sets of executions) elegantly incorporated to the mode-annotated program execution. We show below the mode-annotated execution for query `sum_count([1,5],A,B)` using program *sum_count/3* of Section 1 (with literal positions):

{A/**f**,B/**f**}	⟨0, 1, `sum_count([1,5],A,B)`⟩	{A/**g**,B/**g**}
{X/**f**, Xs/**f**, S/**f**, C/**f**}	⟨2, 0, `sum_count([X|Xs],S,C)`⟩	{X/**g**, Xs/**g**, S/**f**, C/**f**}
{Xs/**g**, SXs/**f**, CXs/**f**}	⟨2, 1, `sum_count(Xs,SXs,CXs)`⟩	{Xs/**g**, SXs/**g**, CXs/**g**}
{X_1/**f**, Xs_1/**f**, S_1/**f**, C_1/**f**}	⟨2, 0, `sum_count([`X_1`|`Xs_1`],`S_1`,`C_1`)`⟩	{X_1/**g**, Xs_1/**g**, S_1/**f**, C_1/**f**}
{Xs/**g**, SXs/**f**, CXs/**f**}	⟨2, 1, `sum_count(Xs,SXs,CXs)`⟩	{Xs/**g**, SXs/**g**, CXs/**g**}
{ }	⟨1, 0, `sum_count([],0,0)`⟩	{ }
{S/**f**, SXs/**g**, X/**g**}	⟨2, 2, `S is SXs + X`⟩	{S/**g**, SXs/**g**, X/**g**}
{C/**f**, CXs/**g**}	⟨2, 3, `C is CXs + 1`⟩	{C/**g**, CXs/**g**}

Subgoals are shown in the order they appeared during the execution, however, as we shall see below, this ordering is not relevant for the slicing. Subscripts are used to denote different instances of the same variable. To reduce the amount of displayed information, only the instantiations of those variables in the subgoal are shown, rather than the complete set of instantiations for the whole program. All subgoals with goal number 0 are head goals, and whenever they appear we can infer that a match between a subgoal and the head goal of a clause has taken place. When $\theta_i\, G\, \theta'_i$ matches head goal $\theta_j\, H\, \theta'_j$, θ_i and θ_j have the modes of variables *before* the matching, θ'_j has the modes of variables of H *after* the matching and θ'_i the modes of the variables of G after its complete execution. Each time a subgoal matches the head of a clause this is recorded but only one

record is kept for each goal/head goal match. In the example above, subgoal $\langle 2,1\rangle$ matches head goal $\langle 2,0\rangle$ twice, but since the modes of variables are consistently the same, only one record is kept.

5 Term Dependences Graph

Our representation of the procedural meaning of logic programs above enables us to detect and formalise the relationships between terms of a program. We propose that a graph be used to represent two important kinds of information, *viz.* the *data flow* among terms, that is, how data are passed on from one term to another, and the *control flow* of a program, that is, those points of a program where a computation affecting the flow of execution is performed:

Definition 6. A *term dependences graph* $\mathcal{G}$ is a finite (possibly empty) set of edges $x \Leftarrow t$ and nodes *test*(t).

Edges depict the flow of data and *test* nodes represent those terms tested during the program execution. Our graph is built by examining the program execution: each mode-annotated subgoal gives rise to a (possibly empty) set of edges presenting the information extracted from its computations. Given a mode-annotated execution, its graph is the union of all sub-graphs of its subgoals.

Unifications and system predicates whose possible meanings are known in advance are employed to build the graph: any computation of a logic program is actually carried out by unification and system predicates only [1]. We thus need to know, in advance, the set **S** of system predicates offered by the language and their possible meanings. We have defined a framework to represent the meaning of system predicates in terms of edges in the data- and control flow graphs. This framework can be customised to any language.

The terms of a program must be uniquely identified. This is required to correctly map a term onto its argument position. In program *sum_count/3*, it is not possible to tell the argument position of "0" for it is found in two different places. To solve this problem, we propose that each term of a program be assigned a unique label – labels, however, play no role whatsoever in the program execution. Program *sum_count/3* is thus rewritten as

$\langle 1,0,$`sum_count(`t_1`,`t_2`,`t_3`)`$\rangle \leftarrow$
$\langle 2,0,$`sum_count([X|Xs],S,C)`$\rangle \leftarrow$
    $\langle 2,1,$`sum_count(Xs,SXs,CXs)`$\rangle$,
    $\langle 2,2,$`S is SXs + X`$\rangle$,
    $\langle 2,3,$`C is CXs + `$t_4\rangle$

5.1 Data Flow between Terms

The meaning of system predicates can be detected by examining the unifications that took place and are formalised as edges of the graph. An edge $x \Leftarrow t$ indicates that a computation made data flow from t onto x. We have formalised the relationship between mode-annotated subgoals and their associated edges/nodes through relation $graph_G$ mapping a mode-annotated goal to its set of edges/nodes, given some restrictions. Below we show one possible use of the unification predicate *=/2*. It depicts the edges associated with the aliasing that takes place when two free variables x and y are unified: links between them are

created, in both directions, representing the potential for data flow:

$$graph_G(\theta\langle c, l, x = y\rangle\theta', \{x \Leftarrow y, y \Leftarrow x\}) \leftarrow x/\mathbf{f} \in \theta \ \wedge \ y/\mathbf{f} \in \theta$$

Similar definitions would be required for all system predicates: since the same predicate can be used in different contexts, all possible behaviours should be explicitly dealt with at this point.

5.2 Control Flow Information

The control flow of logic programs is also appropriately captured with our mode-annotated program execution. We regard as control flow those computations that change or define the flow of execution of the program. A node *test*(t) indicates that a computation performed a test making use of t. We show below the *=/2* predicate performing a test between x and t:

$$graph_G(\theta\langle c, l, x = t\rangle\theta', \{test(x), test(t)\}) \leftarrow x/m \in \theta \ \wedge \ m \neq \mathbf{f}$$

The unification conditions must hold to ensure that a test is being performed, otherwise the computation is not a test but an assignment and is dealt with by one of the formulae for data-flow computations. The mode-annotated execution shown above yields the following graph:

θ_0	$\langle 0, 1, \texttt{sum\_count(t}_1\texttt{,A,B)}\rangle$	θ'_0	$\texttt{A} \Leftarrow \texttt{S}, \texttt{S} \Leftarrow \texttt{A}, \texttt{B} \Leftarrow \texttt{C}, \texttt{C} \Leftarrow \texttt{B},$
θ_1	$\langle 2, 0, \texttt{sum\_count([X\|Xs],S,C)}\rangle$	θ'_1	$\texttt{X} \Leftarrow \texttt{t}_1, \texttt{Xs} \Leftarrow \texttt{t}_1,$
θ_2	$\langle 2, 1, \texttt{sum\_count(Xs,SXs,CXs)}\rangle$	θ'_2	$\texttt{SXs} \Leftarrow \texttt{S}_1, \texttt{S}_1 \Leftarrow \texttt{SXs}, \texttt{CXs} \Leftarrow \texttt{C}_1, \texttt{C}_1 \Leftarrow \texttt{CXs},$
θ_3	$\langle 2, 0, \texttt{sum\_count([X}_1\texttt{\|Xs}_1\texttt{],S}_1\texttt{,C}_1\texttt{)}\rangle$	θ'_3	$\texttt{X}_1 \Leftarrow \texttt{Xs}, \texttt{Xs}_1 \Leftarrow \texttt{Xs},$
θ_4	$\langle 2, 1, \texttt{sum\_count(Xs,SXs,CXs)}\rangle$	θ'_4	$test(\texttt{Xs}), \texttt{SXs} \Leftarrow \texttt{t}_3, \texttt{CXs} \Leftarrow \texttt{t}_4,$
θ_5	$\langle 1, 0, \texttt{sum\_count(t}_2\texttt{,t}_3\texttt{,t}_4\texttt{)}\rangle$	θ'_5	$test(\texttt{t}_2),$
θ_6	$\langle 2, 2, \texttt{S is SXs + X}\rangle$	θ'_6	$\texttt{S} \Leftarrow \texttt{SXs}, \texttt{S} \Leftarrow \texttt{X},$
θ_7	$\langle 2, 3, \texttt{C is CXs + t}_5\rangle$	θ'_7	$\texttt{C} \Leftarrow \texttt{CXs}, \texttt{C} \Leftarrow \texttt{t}_5$

The term dependences graph could be proposed using argument positions as nodes. However, it proves to be simpler and more natural to describe the phenomena of program executions using the program's terms.

6 Relevance of Argument Positions

Our approach to logic program slicing consists of analysing the argument positions of a program and testing their *relevance* to one of the argument positions of the criterion. Our definition of relevance is stated in terms of argument positions but uses the term dependences graph, however, it is possible to obtain all those terms occupying a certain argument position, by scanning each subgoal:

Definition 7. Relation $terms(\alpha, \widetilde{\mathcal{E}}, \{t_0, \ldots, t_n\})$ holds iff terms $\{t_0, \ldots, t_n\}$ occupy argument position α in $\widetilde{\mathcal{E}}$

The notion of relevance described here was originally proposed in [16] and we have adapted it to deal with argument positions in program executions:

Definition 8. Given $\widetilde{\mathcal{E}}$ and $\mathcal{G}$, α is relevant to β if one of the cases below holds:

$$relevant(\alpha, \beta, \widetilde{\mathcal{E}}, \mathcal{G}) \leftarrow relevant(\alpha, \gamma, \widetilde{\mathcal{E}}, \mathcal{G}) \ \wedge \ relevant(\gamma, \beta, \widetilde{\mathcal{E}}, \mathcal{G})$$

$$relevant(\alpha, \beta, \widetilde{\mathcal{E}}, \mathcal{G}) \leftarrow terms(\alpha, \widetilde{\mathcal{E}}, \{\ldots t_\alpha \ldots\}) \ \wedge \ terms(\beta, \widetilde{\mathcal{E}}, \{\ldots t_\beta \ldots\}) \ \wedge \ t_\beta \Leftarrow t_\alpha \in \mathcal{G}$$

$$relevant(\alpha, \beta, \widetilde{\mathcal{E}}, \mathcal{G}) \leftarrow terms(\alpha, \widetilde{\mathcal{E}}, \{\ldots t_\alpha \ldots\}) \ \wedge \ test(t_\alpha) \in \mathcal{G}$$

The first formula states that α is relevant to β if there is an intermediate argument position γ which α is relevant to, and also is relevant to β. This case formalises the transitivity of the *relevant* relationship and amounts to finding out whether there are sequences of relevant intermediate terms.

The second formula exploits the immediate relevance between two argument positions: α is relevant to β if there are terms t_α and t_β occupying, respectively, positions α and β, such that an edge in $\mathcal{G}$ states that data flowed from t_α to t_β.

The third formula holds if there is a node $test(t_\alpha)$ in $\mathcal{G}$ such that t_α is in α. This means that α satisfies the relationship if a term in α is used in a computation that may define or alter the flow of execution. In such circumstances, the β position with respect to which α is being tested is not taken into account. Through this relationship all argument positions whose terms have been employed in computations that define or may alter the flow of control are relevant.

If all term dependences have been detected and appropriately represented, then it is easy to see that the *relevance* relation above will correctly hold when α is relevant to β, since the terms associated with these positions will have their data- and control flow edges in the graph.

7 An Algorithm for Logic Program Slicing

We have devised an algorithm for automatically obtaining a slice of a given a logic program Π with respect to a criterion $\mathcal{C}$. The slice output by the algorithm is another logic program Π' obtained by examining Π and checking the relevance of each argument position α with respect to each argument position β of the criterion.

We show in Figure 1 our proposed algorithm. Steps 1 and 2 collect infor-

input program $\Pi = \{C_1, \ldots, C_m\}$ and criterion $\mathcal{C} = \langle (G_0, \ldots, G_n)\theta, P \rangle$
output program $\Pi' = \{C'_1, \ldots, C'_m\}$
begin
1. obtain mode-annotated execution $\widetilde{\mathcal{E}}$ for $\Pi \vdash (G_0, \ldots, G_n)\theta$
2. obtain term dependences graph $\mathcal{G}$ for $\widetilde{\mathcal{E}}$
3. **for** each subgoal $\widetilde{G}$ in $\widetilde{\mathcal{E}}$ **do for** each α in $\widetilde{G}$ **do for** each β in P **do**
   if $relevant(\alpha, \beta, \widetilde{\mathcal{E}}, \mathcal{G})$ **then** α is marked **relevant**
4. **for** each subgoal $G = \langle c, l, p(t_0, \ldots, t_n) \rangle$ in Π **do for** each t_i **do**
   begin
   if there is a **relevant** α, $terms(\alpha, \widetilde{\mathcal{E}}, \{\ldots t_i \ldots\})$, **then** $t'_i = t_i$ **else** $t'_i = \_$
   if there is $t'_i \neq \_$ **then** $G' = \langle c, l, p(t'_0, \ldots, t'_n) \rangle$ **else** $G' = true$
   end
5. **for** each clause $C_i = H \leftarrow G_1, \ldots, G_n$ in Π **do**
   if $H' = true$ **then** $C'_i = true$ **else** $C'_i = H' \leftarrow G'_1, \ldots, G'_n$

end

Fig. 1: Slicing Algorithm for Logic Programs

mation on the dynamics of the program (its execution and term dependences graph). Step 3 examines the annotated subgoals of the program execution and performs a relevance analysis of each of its argument positions α with respect to $\beta \in P$: if α is found to be relevant to at least one β, then it is marked **relevant**;

subsequent relevance tests will not change its status (an argument position is preserved if it is relevant to *at least* one argument position of the criterion). Step 4 checks each subgoal in Π and obtains a new sliced version G': terms t_i whose argument positions are **relevant** are preserved; those terms whose argument positions were not marked relevant are replaced with an anonymous variable "_"; if all terms of a subgoal have been replaced with anonymous variables then the corresponding sliced subgoal is a *true* predicate which always succeeds. Step 5 assembles the clauses of Π' using the sliced subgoals: if the head goal H has been sliced as *true* then the body will be empty; if H is not *true* then the body is assembled using the corresponding sliced subgoals.

8 Conclusions and Directions of Research

We have studied and compared existing techniques for slicing programs of procedural and logic languages and proposed a framework to slice logic programs. We regard our proposal as a flexible *framework*, rather than a method specific to a particular logic language. The mode-annotated execution concept can be adapted to any logic language; the term dependency graph can be easily obtained if the meaning of the system predicates (built-ins) is available; finally, alternative definitions for *relevance* can be supplied. The slicing algorithm can be seen as being parameterised by all these components: depending on them different slices can be obtained. Our framework scales up naturally, being able to cope with actual programs of arbitrary sizes and complexity. A number of original techniques were devised and used:

- *Representation for procedural meaning of logic programs*: this idea was originally pursued in [16] but we have extended it to cope with complete programs. Our proposal conveniently handles backtracking and failures.
- *Definition of an expressive criterion*: criteria for slicing programs of procedural languages are not easily adapted to logic programs due to special features of the latter. We have defined an expressive notion of criterion for slicing logic programs.
- *A framework to detect and represent data- and control flow*: our term dependences graph can be seen as a model for program dependences as those proposed in [7] and [8]. An interesting feature of our framework (not presented in this paper due to space limitations) is its ability to naturally handle input-output system predicates.
- *Definition of relevance for argument positions*: we have defined a notion of relevance among the argument positions of a program. The notion of relevance is the essence of the slicing algorithm. An extra contribution is that within our method it is possible to slice a logic program with respect to its side-effects (again, due to space limitation we were not able to dwell on this).
- *Slicing algorithm*: its correctness and termination hinge on the test for relevance among argument positions.

We have implemented prototypical versions of the algorithm and all related techniques. We would like to continue our work along the following lines of research:

- *Tutoring and Debugging*: upon clicking on an argument position of a program users could be presented with those other argument positions that are related and hence may cause undesired behaviour in the program (loops or failures) if modified.
- *Program Reuse*: logic program slices can be made into partially developed programs (or *program techniques* [3]) for subsequent use in a program editor [10]. In particular, a slice can be converted into a program transformation schema, as in [18], and applied to existing programs, conferring extra functionalities on them.
- *Theoretic Implications*: we want to investigate the logic-theoretic relationships between a program and its slices.

References

1. H. Aït-Kaci. *Warren's Abstract Machine.* MIT Press, USA, 1991.
2. K. R. Apt. *From Logic Programming to Prolog.* Prentice-Hall, Great Britain, 1997.
3. A. W. Bowles, D. Robertson, W. W. Vasconcelos, M. Vargas-Vera, and D. Bental. Applying Prolog Programming Techniques. *Int'l Journal of Human-Computer Studies*, 41(3):329–350, Sept. 1994.
4. J. Boye, J. Paaki, and Małuszyński. Synthesis of Directionality Information for Functional Logic Programs. In *LNCS, Vol. 724.* Springer-Verlag, 1993.
5. W. F. Clocksin and C. S. Mellish. *Programming in Prolog.* Springer-Verlag, 1987.
6. T. Guymóthy and J. Paaki. Static Slicing of Logic Programs. In *Proc. 2nd. Int'l Workshop on Automated and Algorithmic Debugging*, IRISA, France, 1995.
7. S. Horwitz, T. Reps, and D. Binkley. Interprocedural Slicing Using Dependence Graphs. *ACM Trans. on Program. Lang. & Systems*, 12(1), Jan. 1990.
8. D. Jackson and E. J. Rollins. A New Model of Program Dependences for Reverse Engineering. In *Proc. SIGSOFT'94 (New Orleans, Louisianna).* ACM, 1994.
9. J. W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, 1993.
10. D. Robertson. A Simple Prolog Techniques Editor for Novice Users. In *3rd Annual Conf. on Logic Progr.* Springer-Verlag, 1991.
11. S. Schoenig and M. Ducassé. A Backward Slicing Algorithm for Prolog. In *LNCS, Vol. 1145.* Springer-Verlag, 1996.
12. R. W. Sebesta. *Concepts of Programming Languages.* Addison-Wesley, 1996.
13. L. Sterling and E. Shapiro. *The Art of Prolog.* MIT Press, 1986.
14. F. Tip. A Survey of Program Slicing Techniques. *Journal of Prog. Lang.*, 3(3), 1995.
15. W. W. Vasconcelos. A Method of Extracting Prolog Programming Techniques. Technical Paper 27, Dept. of Art. Intell., Edinburgh Univ., 1994.
16. W. W. Vasconcelos. *Extracting, Organising, Designing and Reusing Prolog Programming Techniques.* PhD thesis, Dept. of Art. Intell., Edinburgh Univ., 1995.
17. W. W. Vasconcelos and M. T. Aragão. Slicing Logic Programs. Tech. Report, 1998. Available from authors.
18. W. W. Vasconcelos and N. E. Fuchs. Prolog Program Development via Enhanced Schema-Based Transformations. Research Paper 776, Dept. of Art. Intell., Edinburgh Univ., 1995.
19. M. Weiser. Programmers Use Slices When Debugging. *CACM*, 25(7), 1982.
20. M. Weiser. Program Slicing. *IEEE Trans. on Soft. Eng.*, SE-10(4), 1984.

Argumentative and Cooperative Multi-agent System for Extended Logic Programming*

Iara de Almeida Móra[1], and José Júlio Alferes[1,2]

[1] CENTRIA, Univ. Nova de Lisboa
2825 Monte da Caparica, Portugal
{idm,jja}@di.fct.unl.pt
[2] Dept. Matemática, Univ. de Évora
R. Romão Ramalho 59
7000 Évora, Portugal
jja@dmat.uevora.pt

Abstract. The ability to view extended logic programs as argumentation systems opens the way for the use of this language in formalizing communication among reasoning computing agents in a distributed framework. In this paper we define an argumentative and cooperative multi-agent framework, introducing credulous and sceptical conclusions. We also present an algorithm for inference and show how the agents can have more credulous or sceptical conclusions.

1 Introduction

The notion of autonomous agent is used to denote the fact that an agent has the ability to decide for itself which goals it should adopt and how these goals should be achieve. In most agents applications the autonomous components need to interact with one another given the inherent interdependencies which exist between then. The predominant mechanism for managing these interdependencies at run-time is negotiation. Negotiation is a process that takes place between two or more agents which communicate to try and to come to a mutually acceptable agreement on some matter to achieve goals which they cannot, or prefer not to, achieve on their own. Their goals may conflict – in which case the agents have to bargain about which agent achieve which goal – or the agents may depend upon each other to achieve the goals.

It is our conviction that the use of a particular argumentation system provides the negotiation mechanism. Argument-based systems analyze defeasible, or non-monotonic reasoning in terms of the interactions between arguments to get an agreement of a goal. Non-monotonicity arises from the fact that arguments can be defeated by stronger (counter)arguments. The intuitive notions of argument, counterargument, attack and defeat have natural counterparts in the real world, which makes argument-based systems suitable for multi-agents

* Thanks to Luis Moniz Pereira and Michael Schroeder for their valuable comments. The work is financially supported by the Portuguese JNICT and Brazilian CAPES.

F. Moreira de Oliveira (Ed.): SBIA'98, LNAI 1515, pp. 161–170, 1998.

applications. The ability to view extended logic programs as argumentation systems opens the way for the use of this language in formalizing communication among reasoning computing agents in a distributed framework. Argumentation semantics in extended logic programming has been defined in [3,4,8,10] for an agent, determining the agent's beliefs by an internal argumentation process. Intuitively, argumentation semantics treats the evaluation of a logic program as an argumentation process, where a goal G holds if all arguments supporting G cannot be attacked. Thus, logic programming can be seen as a discourse involving attacking and counter-attacking arguments.

Dung [3,4] defines whether an argument is acceptable or not with respect to a set of arguments. Prakken and Sartor [8] follow Dung's proposal and introduce a third kind of argument status, defining when an argument is justified, overruled or defeasible wrt. a set of arguments. Both argumentation frameworks have different methods to obtain credulous or sceptical results. Móra, Schroeder and Alferes's [10] modify Prakken and Sartor's proposal and define a more credulous argumentation framework. While Dung uses argumentation to define a declarative semantics for extended logic programs (ELP), Prakken and Sartor's work is driven by applications in legal reasoning. Both Dung's and Prakken and Sartor's work refers to argumentation in a single agent. In [10,6] we generalize their work to a multi-agent setting. In this paper we present a negotiation framework more flexible than the previous ones that allows the agents to obtain conclusions in a more credulous or sceptical way, depending on the application's goals.

1.1 Extended Logic Programming

In the sequel we use the language of Extended Logic Programming, a generalization of Normal Logic Programming with explicit negation [5]:

Definition 1 (Extended Logic Program). *An* extended logic program *(ELP) is a (possibly infinite) set of ground rules*[1] *of the form* $L_0 \leftarrow L_1, \ldots, L_l, not\ L_{l+1}, \ldots, not\ L_m$ $(0 \leq l \leq m)$ *where each* L_i *is an objective literal* $(0 \leq i \leq m)$*. If* $n = 0$ *then the rule is called a* fact *and the arrow symbol is omitted. An objective literal is either an atom* A *or its explicit negation* $\neg A$*. Literals of the form not* L *are called default literals. As usual* L_0 *is called the* head*, and* $L_1, \ldots, not\ L_n$ *the* body *of the rule.*

For this language, various declarative semantics have been defined, e.g. the answer-sets semantics [5] (which is a generalization of the stable models semantics of normal logic programs), the well-founded semantics with explicit negation (WFSX) [7], and the well-founded semantics with "classical" negation [9]. Both of the last two are generalization of the well-founded semantics of normal programs. However, the former, unlike the latter, considers the so-called coherence requirement relating the two form of negation: *"if L is explicitly false then L must*

[1] For simplicity, we use non-grounded rules in some examples. These rules simply stand for its ground version, i.e. for the ground rules obtained by substituting in all possible ways each of the variables by elements of the Herbrand Universe.

be assumed false by default". A paraconsistent extensions of WFSX ($WFSX_p$) has been defined in [1]. In it, unlike in the others mentioned above, contradictory information is accepted and deal with by the semantics. For further details on the subject of extended logic programs, their semantics, applications, and implementation see [2].

1.2 Further Motivation

A negotiation process is started by an agent that should prove some goal to answer a request from another agent, or to satisfy some internal decision. Intuitively, we see the negotiation process as a dialogue. A negotiative dialogue is an exchange of arguments between two (or more) agents, where each alternately presents arguments attacking the previous proposed arguments. The agent who fails to present (counter)arguments looses the negotiation. A process of negotiation is started by an agent who puts forward an initial proposal, i.e. an argument for some goal that should be proven or achieved. Then the receipts agents evaluate the proposal. Arguments can be unacceptable either by having conflicting goals or by having incomplete information. In the former case, the recipient must construct a counterargument to attack the initial proposal (argumentation). In the latter, the try to construct an argument in favor of the initial proposal (cooperation). When the proposed argument is acceptable, the recipient just agree with the initial proposal. The negotiation process continues until an argument or a counterargument is acceptable to all the parties involved, or until the negotiation breaks down without an agreement.

An interesting situation that can happen in a multi-agent negotiation is when an argument proposed to an agent causes contradiction in its knowledge (i.e. the agent concludes both L and $\neg L$). Consider now a variation of the well known Tweety example where an agent Ag_1 knows that "Tweety is a penguin", "Birds normally fly" and "Penguins do not fly", and another agent, Ag_2, knows that "Tweety is a bird". The agents' knowledge can be represented by logic programs and the sequence of rules can be seen as arguments by the corresponding agents:

$$Ag_1 = \{p(t); f(X) \leftarrow b(X), not\ ab(X); \neg f(X) \leftarrow p(X)\}\ \ Ag_2 = \{b(t)\}$$
$$Ag_1\colon [\neg f(t) \leftarrow p(t); p(t)]\ (A_1)\ \ Ag_1\colon [p(t)]\ (A_2)$$
$$Ag_1\colon [f(t) \leftarrow b(t); b(t)]\quad (A_3)\ \ Ag_2\colon [b(t)]\ (A_4)$$

Then a negotiation for "Tweety is a bird" starts by Ag_2 proposing to Ag_1 the argument A_4. Ag_1 has no counterargument for A_4 but Ag_1 cannot agree with $b(t)$ because its addition to Ag_1's knowledge would cause a contradiction (Ag_1 would conclude both $f(t)$ and $\neg f(t)$). In those an argument is not accepted if it has some hypotheses that can be defeated by an agent. This is not the case here, where the argument that is not acceptable does not even have hypotheses. We can be more flexible in this assumption by defining sceptical or credulous agents. Sceptical agents neither agree with a goal L nor a goal $\neg L$, even if one of these is a fact. Credulous agents may agree with L even if other agents (or itself) also agree with $\neg L$. Then, if Ag_1 is a credulous agent, it will accept the

fact that "Tweety is a bird". If the agent is sceptical, it will not agree with that fact. Note that this situation happens because there is no evidence "Tweety is an abnormal bird" to defeat "Tweety flies since it is a bird and *there is no evidence* that it is an abnormal bird". In this paper we do not consider a "belief revision" approach to solve situations when there is no agreement between agents. In such approach, the contradiction between fly and $\neg fly$ could be solved if Ag_1 assumes the objective literal ab.

Next we define an argumentative and cooperative multi-agent framework by combining and extending the proposal of [6]. Finally, we present an algorithm for inference and show how the agents can have more credulous or sceptical conclusions.

2 Multi-agent Negotiation

A multi-agent system is a set of several extended logic programs, each corresponding to an individual agent with an individual view of the world. Agents negotiate by exchanging parts of several independent or overlapping programs and should obtain consensus concerning inference of literals:

Definition 2 (Multi-agent System). *Let Ag_i be extended logic programs, where $1 \leq i \leq n$. The set $\mathcal{A} = \{Ag_1, \ldots, Ag_n\}$ is called a* **M**ulti-**A**gent **S**ystem.

An agent Ag negotiates, i.e. argues and cooperates, by using arguments. An argument is a sequence of ground rules chained together, based on facts and where all negative literals are viewed as hypotheses and assumed to be true. Our negotiation framework has two kinds of arguments, strong arguments and weak arguments. A weak argument is more liable to be attacked than a strong one. Intuitively, a strong argument A_L (for some literal L) can only be attacked by "proving" the complement of a (negative) hypothesis assumed in A_L. A weak argument A'_L (for some literal L) can also be attacked by "proving" the explicit complement[2] of a conclusion of A'_L. For example, if "Tweety flies because it is a bird, non-abnormal birds fly, and I assume that Tweety is not abnormal" is a strong argument, it can be only attacked by "proving" that Tweety is abnormal. If it were a weak argument, it could also be attacked by "proving" that Tweety does not fly.

Definition 3 (Complete Strong and Weak Arguments). *Let Ag be an extended logic program. A* strong *(resp.* weak*)* argument *for L is a finite sequence A^s_L (resp. A^w_L) $= [r_n; \ldots; r_m]$ of rules r_i of the form $L_i \leftarrow Body_i$ (resp. $L_i \leftarrow Body_i, not\ \neg L_i$) such that $L_i \leftarrow Body_i \in Ag$ and (1) for every $n \leq i \leq m$, and every objective literal L_j in the body of r_i there is a $k < i$ such that L_j is the head of r_k; (2) L is the head of some rule of A^s_L (resp. A^w_L); and (3) no two distinct rules in the sequence have the same head. We say A_L is an argument for L if it is either a strong or a weak argument for L. An argument A_L is a* sub-argument *of the argument $A_{L'}$ iff A_L is a prefix of $A_{L'}$.*

[2] By the explicit complement of an objective literal L we mean $\neg L$, if L is an atom, or A if $L = \neg A$.

However, if an agent Ag has no complete knowledge of a literal L then Ag has just a partial argument for L:

Definition 4 (Strong and Weak Partial Argument). *Let Ag be an extended logic program. A* strong *(resp.* weak*)* partial argument *of L is a finite sequence PA^s_L (resp. PA^w_L) $= [r_n; \ldots; r_m]$ of rules r_i of the form $L_i \leftarrow Body_i$ (resp. $L_i \leftarrow Body_i, not\ \neg L_i$) such that $L_i \leftarrow Body_i \in Ag$ and (1) for every $n < i \leq m$, and every objective literal L_j in the body of r_i there is a $k < i$ such that L_j is the head of r_k; (2) L is the head of some rule of PA^s_L (resp. PA^w_L); and (3) no two distinct rules in the sequence have the same head. We say PA_L is a partial argument for L if it is either strong or weak partial argument for L.*

Example 1. Based on the example presented on section 1.2, the set $\mathcal{A} = \{Ag_1, Ag_2\}$ is a multi-agent system such that $Ag_1 = \{p(t); f(X) \leftarrow b(X), not\ ab(X); \neg f(X) \leftarrow p(X)\}$ and $Ag_2 = \{b(t)\}$. The set of arguments[3] of Ag_1 is $Arg_1 = \{[p], [p \leftarrow not\ \neg p], [p; \neg f \leftarrow p], [p \leftarrow not\ \neg p; \neg f \leftarrow p, not\ f]\}$, i.e. $\{A^s_p, A^w_p, A^s_{\neg f}, A^w_{\neg f}\}$, and of Ag_2 is $Arg_2 = \{[b]; [b \leftarrow not\ \neg b]\}$, i.e. $\{A^s_b, A^w_b\}$, and the partial arguments for f are $PA^s_f = \{[f \leftarrow b, not\ ab]\}$ and $PA^w_f = \{[f \leftarrow b, not\ ab, not\ \neg f]\}$.

An argument can be attacked by contradicting (at least) one hypothesis in it. Since arguments assume some negative literals (hypotheses), another complete argument for its complement attacks the former. In this respect, our notion of attack differs from the one of Prakken and Sartor [8] and of Dung [3]. In theirs, an argument attacks another argument via rebut or undercut [8] (resp. RAA- or ground-attack in [3]). The difference depends on whether the attacking argument contradicts a conclusion[4], in the former, or an assumption of another argument[5], in the latter. Since our argumentation framework has two kinds of arguments, the strong and the weak, the first kind of attack is not needed. Simply note that rebut are undercut attacks to weak arguments. Moreover, partial arguments of an agent may be completed with the help of arguments of other agents (i.e. agents may *cooperate* to complete arguments). Such partial arguments can also be used to attack other arguments.

Definition 5 (Direct Attack). *Let A_L and $A_{L'}$ be strong or weak arguments, A_L* attacks *$\{A_{L'}\}$ iff (1) A_L is an argument for L and $A_{L'}$ is an argument with assumption not L, i.e. there is a $r : L_0 \leftarrow L_1, \ldots, L_l, not\ L_{l+1}, \ldots, not\ L_m \in A_{L'}$ and a j such that $L = L_j$ and $l + 1 \leq j \leq m$; or (2) A_L is a partial argument for L, $A_{L'}$ has the assumption not L, and there exists a sequence of arguments $A_{L_1}, \ldots, A_{L_n}$ such that $A_L + A_{L_1} + \ldots + A_{L_n}$[6] is a complete argument for L.*

[3] Assuming that the Herbrand universe only has one individual, we suppress t from the literals, i.e. f stands for $f(t)$.

[4] An argument A_L rebuts/RAA-attacks $A_{\neg L}$, and vice-versa.

[5] An argument A_L undercuts/g-attacks $A_{L'}$ if it contains the default literal *not* L.

[6] By $A + B$ we mean the concatenation of arguments A and B.

Note that the second point of the definition may be viewed as cooperation among some agents with the purpose of another agent. Not all arguments make sense. For example, it does not make sense to have an argument which assumes some hypothesis and, at the same time, concludes its negation. The notions of coherent argument and conflict-free set of arguments formalize what we mean by arguments and sets of arguments that "make sense":

Definition 6 (Coherent, Conflict-free). *An argument A_L is* coherent *if it does not contain sub-arguments attacking each other, i.e. there are no two sub-arguments in A_L, $A_{L'}$ and $A_{L''}$, such that $A_{L'}$ is a complete argument for a literal L' and $A_{L''}$ is an argument with assumption not L'. A set of arguments Args is called* conflict-free *if no two arguments in Args attack each other.*

This attack definition does not completely foresees the cases where an argument causes conflict among arguments, as shown by the example below. This is the motivation for the definition of indirect attacks.

Example 2. Continuing example 1, Arg_1 is a conflict-free set of (partial and complete) arguments, $\{A^s_p, A^w_p, A^s_{\neg f}, A^w_{\neg f}, PA^s_f, PA^w_f\}$. Suppose that agent Ag_2 proposes the argument A^w_b to Ag_1. Intuitively we saw that A^w_b *indirectly attacks* the subset of arguments $S = \{PA^w_f, A^w_{\neg f}\}$ of Ag_1. Note that PA^w_f is a partial argument because it has no sub-argument for b, but $PA^w_f + A^w_b$ is now an argument for f, i.e. $A^w_f = [b \leftarrow not\ \neg b; f \leftarrow b, not\ ab, not\ \neg f]$.

Definition 7 (Indirect Attack). *Let Args and $Args'$ be conflict-free sets of arguments, and $A_{L'}$ be an argument in $Args'$.* $A_{L'}$ indirectly attacks *a subset S of Args iff $S \cup \{A_{L'}\}$ is not conflict-free. An argument A_L attacks a set of arguments S iff A_L directly or indirectly attacks S.*

To defeat a set of arguments S, an agent needs an argument A attacking at least one of the arguments in S that, in turn, is not itself attacked by arguments in S. This, unless the set S contains some incoherent argument, in which case nothing else is needed to defeat S.

Definition 8 (Defeat). *Let Args be a set of arguments, S be a subset of Args, and A_L and $A_{L'}$ be arguments.* $A_{L'}$ defeats *S iff (1) $A_{L'}$ directly attacks S and there is no $A_L \in S$ that attacks $\{A_{L'}\}$; or (2) $A_{L'}$ is empty and there is some incoherent argument $A_L \in S$.*

An argument is acceptable if it can be defended against all attacks, i.e. all arguments that attack it can be defeated. Since we have defined strong and weak arguments, the definition of acceptable (and corresponding characteristic function) [4] can be generalized by parameterizing the kinds of arguments:

Definition 9 ($Acceptable^p_o$). *An argument A_L is $acceptable^p_o$ wrt. a set of arguments Args iff each o argument $A_{L'}$ attacking A_L is defeated by a p argument in Args.*

Definition 10 (Characteristic Function). *Let Ag be an extended logic program in $\mathcal{A}$, Args be the set of arguments of $\mathcal{A}$ and $S \subseteq Args$. The* characteristic function *of Ag and S is $F_{Ag}(S) = \{A_L \in S \mid A_L$ is $acceptable^p_o$ wrt. $S\}$.*

2.1 The Status of an Argument and Agent's Conclusions

The status of an argument is determined based on all ways in which arguments interact. It takes as input the set of all possible arguments and their mutual defeat relations, and produces as output a split of arguments into three classes: arguments with which a dispute can be 'won', 'lost', and arguments which leave the dispute undecided. The "weakest link" principle defines that an argument cannot be justified unless all of its sub-arguments are justified. We define the losing, or "overruled" arguments as those that are attacked by a justified argument and, finally, the undeciding, or "defeasible" arguments as those that are neither justified nor overruled.

Definition 11 (Justified, Overruled and Defeasible). *Let Ag be an extended logic program in $\mathcal{A}$, $Args$ be the set of arguments of $\mathcal{A}$, A_L be an argument of Ag, and F_{Ag} be the characteristic function of Ag and $Args$ then (1) A_L is* justified *iff it is in the least fixpoint of F_{Ag} (called $JustArgs$); (2) A_L is* overruled *iff it is attacked by a justified argument; and (3) A_L is* defeasible *iff it is neither justified nor overruled.*

Based on the status of the arguments we are able to determine the possible values for these conclusions. Note that with weak and strong arguments, a given literal might have more that one conclusion. If both (weak and strong) arguments for L have the same status, i.e. both are justified, overruled or defeasible, then both conclusions for L also coincide. If the strong and the weak argument do not coincide and the values are inconsistent, i.e. the strong argument is justified and the weak is overruled, then a credulous agent would conclude that the literal is justified, and a sceptical one that is defeasible. Otherwise, a credulous agent assumes the value of the strong argument and a sceptical one assumes the value of the weak argument.

Definition 12 (Sceptical and Credulous Conclusion). *Let A_L be an argument for L, C_L^s be a sceptical conclusion for L and C_L^c a credulous one, and C_L be either C_L^s or C_L^c then (1) if A_L is justified (resp. overruled, defeasible) then C_L is justified (resp. overruled, defeasible); (2) if A_L^s is justified and A_L^w is overruled then C_L^c is justified and C_L^s is defeasible; (3) if A_L^s is justified (resp. defeasible) and A_L^w is defeasible (resp. overruled) then C_L^c is justified (resp. defeasible) and C_L^s is defeasible (resp. overruled).*

Example 3. The rule $r_1 : r(X) \leftarrow not\ d(X)$ states that "Someone is republican if it is not democrat" and the rule $r_2 : d(X) \leftarrow not\ r(X)$ states that "Someone is democrat if it is not republican". The fact $f_1 : \neg d(john)$ states that "John is not a democrat". The set of arguments, with obvious abbreviations, is $Args = \{[\neg d], [\neg d \leftarrow not\ d], [d \leftarrow not\ r], [d \leftarrow not\ r, not\ \neg d], [r \leftarrow not\ d], [r \leftarrow not\ d, not\ \neg r]\}$, i.e. $\{A^s_{\neg d}, A^w_{\neg d}, A^s_d, A^w_d, A^s_r, A^w_r\}$. The arguments $A^s_{\neg d}$ and A^s_r are *justified*, A^s_d is *overruled*, and the *defeasible* ones are $A^w_{\neg d}$, A^w_d and A^w_r. Then the credulous conclusion (wrt. $Args$) for $\neg d$ and r is both are justified, and d is overruled. The sceptical conclusion (wrt. $Args$) assumes all literals as defeasible.

Continuing the example 1, the credulous conclusion (wrt. $Args$) for b, p, f, $\neg f$ is that all are justified. However, the sceptical conclusion (wrt. $Args$) is that b and p are justified, and f and $\neg f$ are defeasible.

3 Proof Proposal for an Argument

Based on Praken and Sartor's dialogue proposal [8], we propose a (credulous and sceptical) proof for a justified argument as a dialogue tree where each branch of the tree is a dialogue. As seen in section 2, to attack others' arguments, an agent may need to cooperate in order to complete its partial arguments. To account for this, we start by defining a cooperative dialogue. Intuitively, in cooperative dialogues other agents provide arguments for the still incomplete argument. The process stop either if no agent can provide the required arguments or if all arguments are completed.

Definition 13 (Cooperative Dialogue). *Let $\mathcal{A}$ be a MAS, Ag be an agent in $\mathcal{A}$, $A_L = [r_n; \ldots; r_m]$ be a weak or strong partial argument of Ag, and Inc be the set of objective literals L_j in the body of r_n. A* cooperative dialogue *for A_L in Ag is a finite nonempty sequence of moves, $move_i^c = (Ag_k, A_{L_i})$ $(i > 0)$ such that (1) $move_1^c = (Ag, A_L)$. (2) If A_L is a weak (resp. strong) partial argument then A_{L_i} is a weak (resp. strong) argument. (3) If $i \neq j$ then $S_i \neq S_j$. (4) Let $move_i^c = (Ag_i, [r_n; \ldots; r_m])$ then (a) if there exists an $Ag_k \in \mathcal{A}$ with an argument for some $L \in Inc$, $move_{i+1}^c = (Ag_k, [r_{n_k}; \ldots; r_{m_k}; r_n; \ldots; r_m])$; otherwise, (b) $move_i^c$ is the last move in the dialogue. A cooperative dialogue of an agent Ag for a partial argument A_L is successful if its last move (Ag_f, A_{L_f}) is a (complete) argument. Then we call A_{L_f} its* result.

An (argumentative) dialogue is made between a proponent P and opponents O, and the root of the dialogue tree is an argument for some literal L. An (argumentative) move in a dialogue consists of an argument attacking the last move of the other player. The required force of a move depends on who states it. Since the proponent wants a conclusion to be justified, a proponent's argument has to be defeating while, since the opponents only want to prevent the conclusion from being justified, an opponent move may be just attacking. Based on our definition of credulous and sceptical conclusions, we define a (flexible) dialogue that allow us to obtain conclusions in a more credulous or sceptical way. It depends on how the players, i.e. a proponent P and opponents O, move their arguments. A $dialogue_o^p$ is a dialogue between a proponent that supports p conclusions and an opponent that supports o conclusions; p and o represent a strong (s) or a weak (w) conclusion:

Definition 14 ($dialogue_o^p$). *Let $\mathcal{A}$ be a MAS, Ag be an agent in $\mathcal{A}$, S be a subset of arguments of Ag, and p (resp. o) be a weak or a strong conclusion from the proponent (resp. opponent). A $dialogue_o^p$ for an argument A_L of an agent Ag is a finite nonempty sequence of (argumentative) moves $move_i^a = (Role_i, Ag_i, S_i)$ $(i > 0)$, where $Role_i \in \{P, O\}$ and $Ag_i \in A$, such that (1) $move_1^a = (P, Ag, A_L)$.*

(2) $Role_i = P$ *iff* i *is* odd*; and* $Role_i = O$ *iff* i *is* even. *(3) If* $Player_i = P$ *then every* $A_{L_i} \in S_i$ *supports* p *conclusions; otherwise, they support* o *conclusions. (4) If* $Role_i = Role_j = P$ *and* $i \neq j$*, then* $S_i \neq S_j$*. (5) Every* $A_{L_i} \in S_i$ *are either arguments of* Ag_i *or the* result *of a successful cooperation dialogue for partial arguments of* Ag_i*. (6) If* $Role_i = P$ $(i > 1)$*, then every* $A_{L_i} \in S_i$ *are minimal (wrt. set inclusion) arguments defeating some* $A_{L'} \in S_{i-1}$*. (7) If* $Role_i = O$*, then* S_i *attacks some* $A_{L'} \in S_{i-1}$*.*

A dialogue tree considers all ways in which an agent can attack an argument:

Definition 15 (Dialogue Tree). *Let* $\mathcal{A}$ *be a MAS,* Ag *be an agent in* $\mathcal{A}$*,* S *be a set of arguments, and* A_L *be an argument of* Ag*. A* dialogue tree *for* A_L *is a finite tree of moves* $move_i^a = (Role, Ag_k, S_i)$ *such that each branch is a* $dialogue_o^p$ *for* A_L*, and if* $Role = P$ *then the children of* $move_i^a$ *are all defeaters of an argument* $A_{L'} \in S_i$*. An agent wins a dialogue tree iff it wins all branches of the tree, and an agent wins a* $dialogue_o^p$ *if another agent cannot counter-argue.*

Proposition 1. *Let* Ag *and* Ag_k *be agents, and* A_L *be an argument of* Ag*. An argument* A_L *is* justified *iff* Ag *wins the dialogue tree starting with* A_L*; otherwise,* A_L *is overruled or defeasible. An argument* A_L *is overruled when an opponent* Ag_k *moves at least one justified argument; otherwise,* A_L *is defeasible.*

In the extended version of this article [7] we continue by showing how the players should negotiate, i.e. how they should move their arguments in order to obtain more credulous or sceptical conclusions, and what is correspondence with other semantics. We define which kind of (strong or weak) argument the proponent should move to justify a credulous or a sceptical conclusion, and what kind of (strong or weak) opponent's argument the proponent should defeat to win a negotiation, and compare the results with others. Lack of space prevents us from presenting here the full results, but a brief overview of those is presented below.

To justify a sceptical conclusion, the proponent has to justify both arguments for L. However, it is sufficient to justify only the weak argument. To win the argumentation, the proponent has to defeat all opponent's arguments, i.e. to defeat the strong and the weak ones. Nevertheless the proponent should defeat just the opponent's weak argument. Then both players have to move weak arguments to get a sceptical conclusion. The $dialogue_o^p$ has the same results as Prakken and Sartor's dialogue proposal [8].

The dialogue is an paraconsistent (and credulous) semantics if a player concludes L and if it is also capable of concluding $\neg L$. To obtain a credulous conclusion, the proponent has to justify a single argument in order to prove the conclusion. Thus, it is sufficient for it to justify the strong argument. Similarly to the sceptical proof, the proponent only has to defeat the weak argument. The results of $dialogue_w^s$ are the same as those obtained by the $WFSX_p$ semantics [1] on the union of all agents' programs. Finally, if both players, the

[7] In "http://www-ssdi.di.fct.unl.pt/ idm/fullpaper/mora-ExtSBIA98.ps.gz".

proponent and the opponent, support strong conclusions the semantics coincides with the WFS with "classical negation" defined in [9].

4 Conclusion and Further Work

In this paper we propose a negotiation framework for a multi-agent system, based on argumentation and cooperation. We define strong and weak arguments, simplify the notion of ground- and RAA-attack [3] (resp. undercut and rebut [8]), propose a new concept of attack (indirect attacks), and introduce the credulous and the sceptical conclusions for an agent's literal. Finally, we present inference algorithms to obtain credulous and sceptical conclusions and we show the relation of these algorithms are similar to Prakken and Sartor's argumentation [8], and also to the known semantics, $WFSX$ [7] and $WFSX_p$ [1].

We intend now to define a multi-agent system where the agents have several degrees of credulity, from agents that always accept new information to agents that accept the new one only if it is not incompatible with agent knowledge. We also intend to introduce the capability of the agents to revise their knowledge as consequence of internal or external events. In case of no agreement in a negotiation process the agents would be able to discuss (or negotiate again) how and when they got their knowledge, and try to find a way to get an agreement.

References

1. J. J. Alferes, C. V. Damásio, and L. M. Pereira. A logic programming system for non-monotonic reasoning. *Journal of Automated Reasoning*, 14(1):93–147, 1995.
2. J. J. Alferes and L. M. Pereira. *Reasoning with Logic Programming.* (LNAI 1111), Springer, 1996.
3. P. M. Dung. An argumentation semantics for logic programming with explicit negation. In *10th ICLP*, pages 616–630. MIT Press, 1993.
4. P. M. Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, 77(2):321–357, 1995.
5. M. Gelfond and V. Lifschitz. Logic programs with classical negation. In Warren and Szeredi, editors, *7th ICLP*, pages 579–597. MIT Press, 1990.
6. I. A. Móra and J. J. Alferes. Argumentation and cooperation for distributed extended logic programs. In J.Dix and J.Lobo, editors, *7th International Workshop on Nonmonotonic Reasoning (NMRW)*, 1998.
7. L. M. Pereira and J. J. Alferes. Well founded semantics for logic programs with explicit negation. In *ECAI*, pages 102–106. John Wiley & Sons, 1992.
8. H. Prakken and G. Sartor. Argument-based extended logic programming with defeasible priorities. *Journal of Applied Non-Classical Logics*, 1997.
9. T. Przymusinski. Extended stable semantics for normal and disjunctive programs. In Warren and Szeredi, editors, *7th ICLP*, pages 459–477. MIT Press, 1990.
10. M. Schroeder, I. A. Móra, and J. J. Alferes. Vivid agents arguing about distributed extended logic programs. In Ernesto Costa and Amilcar Cardoso, editors, *Progress in Artificial Intelligence, 8th Portuguese Conference on Artificial Intelligence (EPIA97)*, pages 217–228. (LNAI 1323) Springer. Coimbra, Portugal, 1997.

Modelling Credulity and Skepticism through Plausibility Measures*

Berilhes Borges Garcia[1,2], José Gabriel Pereira Lopes[1], and Paulo Quaresma[1,3]

[1] Centro de Inteligência Artificial (CENTRIA)
Departamento de Informática, Universidade Nova de Lisboa, 2825, Portugal
{bbg,gpl,pq}@di.fct.unl.pt
[2] Departamento de Informática, Universidade Federal do Espírito Santo, Brazil
[3] Departamento de Matemática, Universidade de Évora, Portugal

Abstract. In this paper we show how recently observed facts and plausible explanations for them can be accommodated or not in the mental state of an agent. This depends on whether the epistemic surprise degree associated with them is below or above a surprise limit that the agent is willing to accept. This limit is related to the credulity or skepticism degree the agent exhibits. The proposed approach also is sensitive to context known by the agent (its mental state), so that it allows for a given agent to refuse (or accept) a new evidence in a specific context, while it accepts (or refuses) the same belief in a different context. This approach is completely innovative as agents become more flexible in their interaction with the external world.

1 Introduction

The Webster dictionary defines skepticism as being: '1 : an attitude of doubt or a disposition to incredulity either in general or toward a particular object 2 a : the doctrine that true knowledge or knowledge in a particular area is uncertain b : the method of suspended judgement, systematic doubt, or criticism characteristic of skeptics 3 : doubt concerning basic religious principles (as immortality, providence, and revelation)'. Thus skepticism implies unwillingness to believe without conclusive evidence, requiring compelling evidence before we believe. The opposite of the skeptical attitude is called everything from credulity, a relatively kind term, to outright foolishness and self-delusion.

However one can easily observe that the list of behaviours an individual can present cover the whole spectrum from the purest ingenuousness to the most ingrained skepticism. In spite of this apparent consensus, concerning the diversity of an intelligent agent's possible behaviours, practically every research results (Quaresma's [9] is an exception) on autonomous agents assume an extremely naive posture (credulous); i.e. the agent accepts every belief proposed by the external environment or by any other existing agents, with which it interacts.

* This work has been carried out in the framework of project DIXIT, funded by programme Praxis XXI, contract 2/2.1/TIT/1670/95, and has been supported by the Ph.D. grant awarded by CNPq-Brazil to Berilhes Garcia, Proc. 200562-95.9.

F. Moreira de Oliveira (Ed.): SBIA'98, LNAI 1515, pp. 171-180, 1998.

This extremely naive posture weakens the agent's rational behaviour. It drives it to non-natural behaviours and into a lot of non-coherent situations. Allowing the modelling the two extreme poles of the spectrum of possible behaviours, i.e. pure ingenuousness and ingrained skepticism, the work of Quaresma and Lopes [10] constitutes a progress in relation the previous approaches. However it is characterized by being highly rigid, leading the agent to behave always in the same manner in every context and does not allow the modelling of intermediate behaviours between the two extremes. Thus, in the approach by Quaresma and Lopes, a naive agent always accepts to modify its beliefs in function of the new evidence if it believes its interlocutor is sincere, while a skeptical agent never accepts to modify them.

The main goal of this paper is to propose a new framework which allows to model the whole range of an agent's possible behaviours, and at the same time has into account the context known by the agent. It also allows for a given agent to refuse (or accept) a proposed belief in a specific context, while it accepts (or refuses) the same belief in a different context, possibly due to the evolution of the previous context by adding some new evidence.

In this paper will take as central the notion of plausibility degree presented by Garcia and Lopes[5], and on top of that we will define what does it mean to say that a certain agent is very credulous, or any other gradation with relation to the agent credulity, or skepticism. Intuitively we will say, for example, that an agent is very credulous when it accepts to modify its beliefs in order to accommodate a new observation despite the fact that the most plausible explanation for the observation is very surprising, or in other words it has very little plausibility.

Previous paragraph highlights another basic aspect of our proposal the notion of explanation for the newly observed fact. According to our proposal, when an agent observes a new fact ϕ, it should look for an explanation for this fact, before changing its beliefs. If it is possible to find an explanation α for the newly observed fact, it should additionally determine the plausibility degree for this explanation. If this plausibility degree is above a threshold the agent is willing to accept, it should then change its beliefs. This change is done in such way that both ϕ and α are incorporated in agent's belief set as well as the logical consequences of the new fact and of its explanation.

On the other hand, if the agent can not determine an explanation, its beliefs remain unaltered; this means that the agent assumes a cautious position. Of course this can be changed, but the agent should assume an inquisitive posture (similar to ELISA [11]) asking its interlocutor for some kind of explanation. But this will be worked in the future. This scenario is equivalent to the situation where the agent prefers the information in background to the newly observed fact.

This paper is structured in the following way: next section introduces a motivational example and the formalism used to describe our domain. Section 3 presents concisely our proposal for incorporating new evidence in an agents's knowledge base. This proposal is based on the semi-qualitative notion of plausibility and in the abductive explanation for the newly observed fact. Section 4

demonstrates how can the framework previously presented be used for modelling several types of behaviour an agent can manifest. In the final section we draw the main conclusions and directions for further research.

2 Domain Descriptions

The specific knowledge an agent possesses about a certain context, T, will be represented by a pair (K_D, E), where K_D denotes the background generic knowledge about the domain the agent has and E represents the contingential knowledge, i.e. the knowledge that is likely to vary from case to case and along the time axis. We also refer to the pair $T = (K_D, E)$ as the agent's known context or its knowledge base.

Let $\mathcal{L}$ be a set of arbitrary ground literals (an atom a or its explicit negation, $\neg a$). By a schema $\lambda(X)$, where X is a tuple of free variables and constants, we mean the set of all ground instances $\lambda(a)$ presents in $\mathcal{L}$, where a is a tuple of ground terms, and every free variable in X has been substitued by a ground term.. The background knowledge K_D will be represented by means of a set of default clauses[1]: $\alpha_i \rightsquigarrow \beta_i$. Each default $\alpha_i \rightsquigarrow \beta_i$ is interpreted as a defeasible rule, which can be understood, as stating: "If α_i holds then normally / typically β_i holds". Where β_i is a schema or its negation. And α_i is a formula constructed from a set of schemas and from the connectives $\vee$, $\wedge$ and $\neg$. $\rightsquigarrow$ is a meta-connective, meaning normally / typically. The additional symbol $\perp$ represents logical falsity.

Example 1. Take an agent A, having the following specific knowledge, regarding a certain domain:

$$K_D = \left\{ \begin{array}{l} d_1 : cs(X) \rightsquigarrow \neg int(X, lp) \\ d_2 : cs(X) \rightsquigarrow \neg int(X, lin) \\ d_3 : int(X, ai) \rightsquigarrow int(X, lp) \\ d_4 : int(X, ai) \rightsquigarrow \neg int(X, lin) \\ d_5 : int(X, ai) \rightsquigarrow cs(X) \\ d_6 : int(X, pr\_cl) \rightsquigarrow int(X, lin) \\ d_7 : int(X, pr\_cl) \rightsquigarrow int(X, ai) \end{array} \right\} \quad (1)$$

Rules d_i represent the following facts: (d_1) A believes that computer science (cs) students are normally neither interested in learning logic programming (lp), (d_2) nor linguistics (lin), (d_3) students interested in artificial intelligence (ai) are normally interested on learning logic programming, (d_4) but typically are not interested in learning linguistics, (d_5) students interested in artificial intelligence are normally students from computer science, (d_6) students interested in doing their final course project on computational linguistics (pr_cl) are typically interested on learning linguistics and (d_7) are normally interested in artificial intelligence.

[1] For simplicity reasons and space in this paper we only consider domains described exclusively by default rules.

Assume now that the agent A also knows that b is a computer science student, which can be represented by the contingential knowledge: $E = \{cs(b)\}$.

If after a while A gets evidence in favor of the fact that b is interested in studying linguistics, which is represented by formula $\phi = int(b, lin)$.

Given that A believes in $\neg int(X, lin)$, should A change its beliefs in order to incorporate this new evidence ϕ? In a general way all the approaches for modelling autonomous agents assume that the answer to this question is positive, regardless of the behaviour characteristics of the agent (that is if the agent is skeptical, not very naive, etc.) and of the context known by the agent. However we conjecture in this work that an agent should prefer to maintain his beliefs in background and question the validity of the new evidence, when the most plausible explanation for this new evidence is above the plausibility limit that the agent is willing to accept.

Assume that an agent A is a credulous agent and that the explanation most plausible for $\phi = int(b, lin)$ is $\alpha = \{int(b, pr\_cl)\}$. This explanation is very surprising (the definition of epistemic surprise and the mechanism to measure it are presented in section 3, however greater details can be found in [5] and [4]) given the context known by agent. In this case we conjecture that the agent's new belief set should incorporate not only the logic consequences of the new fact but also the logic consequences of its explanation. So that the belief set of A is equal to: $BS_1 = \{cs(b), int(b, lin), int(b, ai), int(b, pr\_cl)\}$.

Assume now that an agent $A^{'}$ is a bit less credulous than A. It has also the same background knowledge (1) and the same contingential knowledge $E = \{cs(b)\}$ as A. What will be the epistemic state of $A^{'}$ after it observed the new evidence $\phi = int(b, lin)$? Given that the context T known by two agents is identical we are led to conclude that the more plausible explanation $\alpha = \{int(b, pr\_cl)\}$ to A is also the more plausible explanation to $A^{'}$. Furthermore suppose that the degree of epistemic surprise of $\alpha = \{int(b, pr\_cl)\}$ is above of the surprise threshold that $A^{'}$ is willing to accept. Therefore we propose that the belief set of $A^{'}$ should be:

$$BS_2 = \left\{cs(b), \neg int(b, lp), \neg int(b, lin), goal\left(A^{'}, know\_if\left(int\left(b, pr\_cl\right)\right)\right)\right\} \quad (2)$$

Where $goal(A^{'}, know\_if(b, int(b, pr\_cl)))$ represents the fact that $A^{'}$ has got this new goal that will lead it to find out if b is interested in doing his final project course in computational linguistics. In this way the belief set of $A^{'}$ after the observation of ϕ differs from the set of beliefs it possessed before, given the fact that it has now adopted the goal for obtaining additional information in order to clarify the situation. We can understand this state of beliefs as being an unstable and transitory state, in which the agent sets up a new goal for getting confirmation for the most plausible explanation for ϕ, in order to enable it to incorporate (or reject) new evidence ϕ.

The examples shown above demonstrate that the epistemic state of an agent after the observation of a new evidence ϕ depends on its behavioural tendency, that is if it is very credulous, credulous, etc. However this does not tell the whole story. Another essential aspect is the context known by the agent, that is its

background knowledge and its contingential knowledge. The previous example also stresses the central role played by the *most plausible* explanation for a new fact.

3 Plausibility Degree and Abductive Explanations

From the description of the background knowledge K_D of an agent A we use the specificity relations among the default conditionals to establish preference relations between the interpretations an agent foresees as possible. The determination of the specificity relations is made using system Z [7] and the set K_D is partitioned into mutually exclusive subsets $K_{D0}, K_{D1}, ..., K_{Dn}$. Two rules belonging to a subset K_{Di} are equally specific. If they belong different subsets they have different specificities. The higher the i value the higher the specificity. For space reasons we will not detail the partitioning process. Thus readers should consult [7], [6] for more details.

Example 2. The partition of K_D of example (1) is: $K_{D0} = \{d_1,\ d_2\};\ K_{D1} = \{d_3, d_4, d_5\};\ K_{D2} = \{d_6, d_7\}$

The the specificity of the default d_m; written $Z(d_m)$, is i iff $d_m \in K_{Di}$. In the previous example we have $Z(d_1) = Z(d_2) = 0,\ Z(d_3) = Z(d_4) = Z(d_5) = 1$ and $Z(d_6) = Z(d_7) = 2$.

A relationship can be established between the default ranking function $Z(.)$ and a function $k(.)$ that ascribes to each interpretation w of $T = (K_D, E)$ an ordinal representing the normality of this interpretation. An interpretation is a truth assignment to the elements of $\mathcal{L}$ present in $K_D \cup E$.

The link between these two functions is made considering the normality of an interpretation as being equal to specificity the of the higher order default that it violates[2] added with one. Thus, the ranking of an interpretation w in which a student b of computer science, $cs(b)$, is interested on learning logic programming , $int(b, lp)$, is $k(w) = 1$, since the specificity of the higher order default violated by this interpretation is 0 (this interpretation violates the default $cs(X) \rightsquigarrow \neg int(X, lp)$, whose specificity is zero). However it should be stressed that we are interested in measuring how much surprised an agent would be by finding an interpretation that would satisfy ψ given that its mental state already satisfies ϕ. This degree of surprise may be defined as being $k(\psi/\phi)$ [7], and it equals the difference between $k(\psi \wedge \phi)$ and $k(\phi)$, or written otherwise:

$$k(\psi/\phi) = k(\psi \wedge \phi) - k(\phi) \tag{3}$$

It is assumed that the degree of plausibility of a formula ψ since the agent already knows ϕ, $Pl(\psi/\phi)$, is 'inversely' proportional to the degree of surprise of ψ, $k(\psi/\phi)$, i.e. given two propositions α and β we will say that α is at least

[2] A default $\alpha \rightsquigarrow \beta$ is verified in an interpretation if both α and β are satisfied, and it is violated in an interpretation if α is satisfied but β is not.

as plausible as β given the contingential knowledge E, $Pl(\beta/E) \preceq Pl(\alpha/E)$, iff $k(\alpha/E) \leq k(\beta/E)$.

Based on this plausibility degree concept, Garcia and Lopes [5] define a ζ-translation of the agent's knowledge base $T = (K_D, E)$ into an extended logic program, with two types of negation: explicit negation and negation by default. This translation plays two roles. In first place it provides a efficient method for computing the logical consequences of the agent's known context. Furthermore, it also allows, on a meta-logic level, to measure the degree de plausibility of a formula. For more details see [5] and [4]. The semantics of this program is given by the Well Founded Semantics eXplicit negation (WFSX) [1].

An extended logic program P is a set of rules of the following kind:

$$L_0 \leftarrow L_1 \wedge ... \wedge L_m \wedge not\ L_{m+1} \wedge ... \wedge not\ L_n \tag{4}$$

Where $0 \leq m \leq n$ and $0 \leq i \leq n$ Each L_i is an objective literal. An objective literal is an atom A or its explicit negation $\neg A$. The symbol *not* represents negation-as-failure and $not\ L_i$ is a default literal. Literals are objective literals or default literals and $\neg\neg A \equiv A$.

Example 3. For example one considers the background knowledge K_D defined by (1). The ζ-translation of this knowledge is equal to the following program P_D:

$$\begin{aligned}
\neg int(X, lp) &\leftarrow cs(X) \wedge not\ int(X, lp) \wedge not\ ab0.\\
\neg int(X, lin) &\leftarrow cs(X) \wedge not\ int(X, lin) \wedge not\ ab0.\\
ab0 &\leftarrow int(X, ai).\\
int(X, lp) &\leftarrow int(X, ai) \wedge not\ \neg int(X, lp) \wedge not\ \ ab1.\\
\neg int(X, lin) &\leftarrow int(X, ai) \wedge not\ int(X, lin) \wedge not\ ab1.\\
cs(X) &\leftarrow int(X, ai) \wedge not\ \neg cs(X) \wedge not\ ab1.\\
ab1 &\leftarrow int(X, pr_cl).\\
int(X, lin) &\leftarrow int(X, pr_cl) \wedge not\ \neg int(X, lin) \wedge not\ ab2.\\
int(X, ai) &\leftarrow int(X, pr_cl) \wedge not\ \neg int(X, ai) \wedge not\ ab2.\\
lev0 &\leftarrow not\ \neg lev0 \wedge not\ lev1 \wedge not\ lev2 \wedge not\ lev3.\\
lev1 &\leftarrow cs(X) \wedge not\ \neg int(X, lp) \wedge not\ \neg lev1 \wedge not\ lev2 \wedge not\ lev3.\\
lev1 &\leftarrow cs(X) \wedge not\ \neg int(X, lin) \wedge not\ \neg lev1 \wedge not\ lev2 \wedge not\ lev3.\\
lev2 &\leftarrow int(X, ai) \wedge not\ int(X, lp) \wedge not\ \neg lev2 \wedge not\ lev3.\\
lev2 &\leftarrow int(X, ai) \wedge not\ \neg int(X, lin) \wedge not\ \neg lev2 \wedge not\ lev3.\\
lev2 &\leftarrow int(X, ai) \wedge not\ cs(X) \wedge not\ \neg lev2 \wedge not\ lev3.\\
lev3 &\leftarrow int(X, pr_cl) \wedge not\ int(X, lin) \wedge not\ \neg lev3\\
lev3 &\leftarrow int(X, pr_cl) \wedge not\ int(X, ai) \wedge not\ \neg lev3
\end{aligned} \tag{5}$$

Where the literal lev_j represents the normality degree, so that the higher value of the index i, the lower a model's normality will be.

We will refer the Well Founded Model (WFM) of extended logic program P_D obtained by means of ζ-translation of $T = (K_D, E)$ by $WFM_{K_D,E}$. It takes into account the implicit specificity relations in agent's knowledge base and it is a maximally normal model, i.e. minimally surprising, which satisfies E. The obtained logic program can be enlarged in order to allow the agent determine the possible explanations for the new evidence, according to [3] and [8]. For more details see [5]. Once the agent has determined a possible explanation α for a new fact ϕ, it is able to evaluate the plausibility degree associated to this fact ϕ and its explanation α. All it needs to do is to evaluate the difference between the normality of $WFM_{K_D,E}$ and the normality of WFM that results from the assimilation of α and ϕ, $WFM_{K_D,E\cup\alpha\cup\phi}$, in other words $k(WFM_{K_D,E\cup\alpha\cup\phi}) - k(WFM_{K_D,E})$. This can be done in a simple way through the difference that exists between the indexes of the literals lev_i's present in $WFM_{K_D,E\cup\alpha\cup\phi}$ and $WFM_{K_D,E}$.

To summarize so far, we are able to determine the degree of plausibility resulting from the assimilation of a fact ϕ and its explanation α. However, our objective is to determine the degree of plausibility of a new fact ϕ. How can this be done? The simple answer to this question is to assume the most plausible explanation for α, i.e. minimally surprising.

Definition 1 (ϕ's plausibility degree). *Let* $||T \wedge \phi|| = \{\alpha_1, \alpha_2, ..., \alpha_m\}$ *be a set of explanations for* ϕ *in* T *, thus* $k(\phi/E) = \min_{\alpha_i \in ||T\wedge\phi||} k(WFM_{K_D,E\cup\alpha_i\cup\phi}) - k(WFM_{K_D,E})$.

4 Modelling Behaviours

In the previous section we briefly presented a framework that allows an agent A to evaluate the plausibility degree of a new information ϕ, given that it already knows a context (mental state) $T = (K_D, E)$. Next step establishes the connection between this plausibility measure and the types of behaviour an agent may exhibit. Initially take two agents, A and A', that possess the same background knowledge K_D. Suppose also that both have the same evidential context E, that is, both know the same context $T = (K_D, E)$.

Assume now that both obtain simultaneously a new evidence ϕ, which is inconsistent with the beliefs presently retained by them. Observe that we can state that the most plausible explanation α found by A will also be the most plausible explanation found by A', given that both agents know the same context T. Taking into account what has been said in section 3 we can conclude that both agents have the same degree of epistemic surprise in relation to the new evidence ϕ.

Being so, if agent A modifies its beliefs, in order to incorporate the new evidence and its most plausible explanation α, and if A' prefers to engage in an investigation process to guarantee more plausibility for the most plausible explanation α for ϕ; how can we justify that both agents show differentiated behaviours? A possible answer, advocated by us in this paper, is that both

agents possess different limits of tolerance to epistemic surprise. And that an agent only accepts a new evidence ϕ if the degree of surprise associated to this new evidence is situated below the limit of epistemic surprise that it is willing to accept. Thus, the greater the limit of epistemic surprise an agent has, the higher its capability for accepting a new evidence ϕ will be, that is the more credulous the agent is, and consequently is the smaller level of skepticism it will have.

Proposition 1 (Acceptance of new evidence ϕ). *A new evidence ϕ is only accepted by an agent A in a context $T = (K_D, E)$ iff $k(\phi/E)$ is below the surprise limit it is willing to accept.*

Example 4. Assume that two agents A e A' have the same background knowledge K_D defined by (1). Given that the contingential knowledge the agents know is $E = \{cs(b)\}$ suppose that the new evidence $\phi = \{int(b, lin)\}$ is observed. The abductive framework P_A obtained, according to [5], for both agents, in this situation is equal to:

$$P_A = \left\langle \begin{array}{c} \{P_D \cup cs(b) \leftarrow\}, \{cs(b), int(b, ai), int(b, pr\_cl)\}, \\ \{\bot \leftarrow not\ int(b, lin)\} \end{array} \right\rangle \tag{6}$$

Where P_D is the extended logic program showed in (5), $\{cs(b), int(b, ai), int(b, pr\_cl)\}$ is the abductive literals set and $\{\bot \leftarrow not\ int(b, lin)\}$ denotes the integrity constraints set that must be satisfied. In this case, the abductive answer is $\{int(b, pr\_cl)\}$. And the plausibility, or better, its epistemic surprise, is $k(int(b, pr\_cl)/E) = 2$, para both agents. Assuming that the surprise threshold of A is 2, we would conclude that the new context known for A is equal to:

$$T_1^A = \{K_D, E \cup \phi \cup int(b, pr\_cl)\} \tag{7}$$

We can see that the conjunction of P_D (the ζ-translation of K_D) with $\{cs(b) \leftarrow\ , int(b, pr\_cl) \leftarrow\ , int(b, lin) \leftarrow\ \}$ derives the following belief set:

$$BS_A = \{cs(b), int(b, lin), int(b, ai), int(b, pr\_cl), ab0, ab1, lev2\} \tag{8}$$

which corresponds to advocated intuition. Furthermore if we assume that the surprise threshold of A' is equal to 1, so that in accordance with the proposition (1) the epistemic state of A' does not include the new evidence ϕ. However in this work, we conjecture that agent A' shall acquire a new goal for obtaining an explicit confirmation for the most plausible explanation it has found for the newly observed fact. The main justification for this position is that the observation of ϕ has an informative value that should not be dismissed by the agent. Therefore we believe that the new context known for A' is equal to:

$$T_1^{A'} = \left\{K_D, E \cup goal\left(A', know\_if\left(int\left(b, pr\_cl\right)\right)\right)\right\} \tag{9}$$

Again we can see that the conjuction of P_D with $\left\{cs(b) \leftarrow\ ,\ goal\left(A', know\_if\left(int\left(b, pr\_cl\right)\right)\right) \leftarrow\ \right\}$ derives the following belief

set:

$$BS_{A'} = \left\{ \begin{array}{c} cs(b), \neg int(b, lin), \neg int(b, lp), \\ goal\left(A', know\_if\left(int\left(b, pr\_cl\right)\right)\right), lev0 \end{array} \right\} \quad (10)$$

Therefore, we can model the various types of behaviour an agent may present by fixing different values for its limit of epistemic surprise. The only restriction to be imposed refers to the relation of order (rank), so that for any two agents A and $A^{'}$, if A has a higher threshold than $A^{'}$, then A is more credulous than $A^{'}$. In other words, A is more credulous than $A^{'}$ iff $thr\left(A\right) > thr(A^{'})$, where $thr\left(X\right)$ represents the limit of epistemic surprise acceptable by agent X. Another point to be stressed concerns the role played by the context in the process of assimilation of new evidence. Consider again agent $A^{'}$ from the previous example. Assume now that his evidential knowledge is equal to $E^{'} = \{cs(b), int(b, ai)\}$ and it obtains the same evidence $\phi = \{int(b, lin)\}$ as in the previous example. Observe, however, that the plausibility degree of ϕ given $E^{'}$ is equal to 1, $k\left(\phi/E^{'}\right) = 1$. So, according to proposition (1) the new context known by $A^{'}$ is:

$$T_1^{A^{'}} = \left\{K_D, E^{'} \cup \phi \cup int(b, pr\_cl)\right\} \quad (11)$$

In this way, we can conclude that the contingential knowledge E held by an agent before the observation of a new evidence ϕ is a determinant factor in the epistemic state that results from this observation.

5 Conclusion

Along this paper we presented a framework that is able to model various types of behaviour an agent may exhibit. This framework relies on two basic notions: the notion of abductive explanation for a fact recently observed and the concept of degree of surprise associated to this new evidence. After, we introduced the notion that a new evidence can only be accepted by the agent, without major questioning, if the degree of surprise of this new evidence is below the limit of epistemic surprise that the agent is willing to accept. Thus, through the assignment of differentiated values to this limit, we can model various types of behaviour.

This framework has two main characteristics. First, it can model various types of behaviour an agent may exhibit. This characteristic, in our opinion, is extremely important when we consider the possibility of a computational agent interacting with other agents, who may convey intentionally or unintentionally erroneous information. In any of these situations, the agent should possess auto-protection mechanisms for its own beliefs. One possible mechanisms is to adopt skeptical positions in relation to its interlocutors. However, we also expect the agent to be sensitive to the context, so that the acceptance, or rejection, of new evidence is conditioned by the context known by the agent (by its mental state). In this way, the agent has a more flexible behaviour, and so it does not necessarily

accepts or rejects always a new evidence, but it evaluates the convenience of the new information regarding the context already known. This flexibility is the second main characteristic of our proposal.

Presently we are investigating how this proposal may be applied in the context of advising dialogues, where the process of transmission of beliefs between the interlocutors is mediated by the framework we presented here. And when an autonomous agent receives a statement whose degree of plausibility is below its acceptance level, it should try to engage itself in pro-active attitudes in order to clarify the situation. However some questions remain open, namely the ones related to the nature of the surprise limit and how this can be determined.

All the examples that were presented in this paper were experimented with the latest version of the program REVISE [2], a programming system in extended logic for the revision of belief base. This program is based in top-down derivation procedures for WFSX (Well Founded Semantics with eXplicit negation) [1].

References

1. J. J. Alferes and L. M. Pereira. *Reasoning with Logic Programming.* Springer-Verlag: Heidelberg, Germany, 1996.
2. C. V. Damásio, L. M. Pereira, and M. Schroeder. Revise progress report. In *Workshop on Automated Reasoning: Bridging the Gap between Theory and Practice*, University of Sussex, Brighton, Apr. 1996.
3. K. Eshghi and R. A. Kowalski. Abduction compared with negation by failure. In M. Levi, Giorgio; Martelli, editor, *Proceedings of the 6th International Conference on Logic Programming (ICLP '89)*, pages 234–254, Lisbon, Portugal, June 1989. MIT Press.
4. B. B. Garcia and G. P. Lopes. Incorporating specificity in extended logic programms for belief resion. In *Proceedings of the 11th International FLAIRS Conference*, pages 215–219, Miami, USA, May 1998. AAAI Press.
5. B. B. Garcia and G. P. Lopes. Introducing plausibility measures to the process of belief revision through extended logic programs. In *Proceedings of 13th European Conference on Artificial Intelligence - ECAI 98*, pages 378–382, Brighton, UK, Aug. 1998. John Wiley & Sons.
6. M. Goldszmidt and J. Pearl. Qualitative probabilities for default reasoning, belief revision, and causal modeling. *Artificial Intelligence*, 84(1-2):57–112, Dec. 1996.
7. J. Pearl. System Z: A natural ordering of defaults with tractable applications to nonmonotonic reasoning. *TARK: Theoretical Aspects of Reasoning about Knowledge*, 3, 1990.
8. L. M. Pereira, J. J. N. Aparício, and J. J. Alferes. Nonmonotonic reasoning with well founded semantics. In K. Furukawa, editor, *Proceedings of the 8th International Conference on Logic Programming*, pages 475–489. MIT Press, 1991.
9. P. Quaresma. *Inference of Attitudes in Dialogue situation.* PhD thesis, Department of Computer Science, Universidade Nova de Lisboa, 1997.
10. P. Quaresma and J. G. Lopes. Unified logic programming approach to the abduction of plans and intentions in information-seeking dialogues. *Journal of Logic Programming*, 24(1–2):103–119, July/Aug. 1995.
11. J. Weizenbaum. ELIZA-a computer program for the study of natural language comunication between man and machine. *Communications of the ACM*, 9(1):36–44, 1966.

Fuzzy Temporal Categorical and Intensity Information in Diagnosis

Jacques Wainer[1] and Sandra Sandri[2]

[1] Instituto de Computação
UNICAMP, Brazil
wainer@dcc.unicamp.br
[2] LAC
INPE, Brazil
sandri@lac.inpe.br

Abstract. This paper proposes a way of incorporating fuzzy temporal reasoning within diagnostic reasoning. Disorders are described as an evolving set of necessary and possible manifestations. Fuzzy intervals are used to model ill-known moments in time (e.g. the beginning and end of a manifestation) and intensity of manifestations (e.g. "high" fever). The paper discusses several measures of consistency between a disorder model and the patient data, and defines when the manifestations presented by the patient can be explained by a disorder.

1 Introduction

Temporal information and temporal reasoning are important aspects of diagnostic reasoning [9,8,2,13], specially in some domains, such as, for example, the diagnostics of infectious diseases.

That is the case, for instance, of the intoxication caused by ingestion of poisonous mushrooms of the species *Amanita phalloides, A. virosa*, and *A. verna* [10, Chap. 81]. It always causes abdominal cramps and nausea within 6 to 24 h from ingestion, lasting for up to 24 h, followed by a period of 1 to 2 days of no symptoms, finally followed by hepatic and renal failure (leading to death in 50% of the cases). This is called the model of the disease.

Faced with a case in which the patient has ingested mushrooms, felt abdominal cramps and nausea, but has not yet shown symptoms of renal and hepatic failure, one should not rule out intoxication by the *Amanita* family without verifying whether there has been enough time for those symptoms to develop.

The main goal of this paper is to answer the following questions:

- when is the model of a disease consistent with the case information and to what degree. For example, can we state that the model of intoxication with *Amanita* is consistent with the case of a patient who suffered from nausea and abdominal cramps for three days and then showed signs of renal failure and loss of sensation in the limbs two days later.

F. Moreira de Oliveira (Ed.): SBIA'98, LNAI 1515, pp. 181–190, 1998.

- when is the disease model categorically consistent the case information, that is, have all necessary symptoms in the model occurred (provided that they had had enough time to occur). If a patient had nausea and abdominal cramps for one day, then showed signs of renal failure two days later, but did not present signs of hepatic failure, can we consider that the model and the case are categorically consistent.

- when is a single disorder capable of explaining all symptoms of the present in the case. In the example of the patient that exhibited nausea and abdominal cramps, followed by renal failure and loss of sensation on the limbs, and given that the doctor had not made any test for hepatic failure, does poisoning by *Amanita* explain all symptoms, or in other words, is poisoning by *Amanita* a possible diagnostic.

- if we consider that a particular disease explains all the patient symptoms, what else do we know about other manifestations that the patient may have had, or will develop in the future.

In our model all temporal information is modeled by fuzzy sets, since most of the time we are dealing with information furnished by human beings, which are usually tainted with vagueness. For instance, a patient will normally tell that the interval between ingestion and cramps was "around 4 to 5.5 hours". On the other hand, a doctor would hardly discard the hypothesis of ingestion of *Amanita* if the patient has developed abdominal cramps exactly 25 hours after ingesting some kind of mushroom, instead of the expected 24 hours.

The use of fuzzy sets allow us moreover to obtain a degree of consistency between a model and a case. For instance, the 25 hours that elapsed between ingestion and cramps, although beyond the specified 24 hours, is not too far apart. But if it is known that this interval was precisely 36 hours, one is much more inclined to state the inconsistency. Thus one would like to have a concept of "degree of consistency," such that if the elapsed time between ingestion and cramps in the case is precisely known and is between 6 to 24 hours, then one can fully state the consistency of case and model, and this degree decreases the further away from the interval the case information is.

The next section brings definitions about fuzzy sets, defines what is the model of a disease and what is the information for a case. Section 3 provides answers to some of the questions raised above: when is the case temporally consistent with the model, when is the case categorically consistent with the model, and when is the model an explanation for the case. It also includes a subsection on intensity consistency. Section 4 brings some current research material about dealing with manifestations which have a known frequency of occurrence inside a given disorder. Finally section 5 discusses the limits of the model proposed and future work.

2 Basic Definitions

2.1 Fuzzy Intervals

A fuzzy set A in Θ [4] is characterized by a membership function $\mu_A : \Theta \to [0, 1]$, such that $\exists x \in \Theta, \mu_A(x) = 1$. The height of a fuzzy set A is calculated as $h(A) = \sup_{x \in \Theta} \mu_A(x)$, and A is said to be normalized when $h(A) = 1$.

Let A and B be fuzzy sets in Θ. The sum $A \oplus B$, the subtraction $A \ominus B$ and the negation $-A$ are respectively characterized by membership functions [4]:

- $\mu_{A \oplus B}(z) = sup_{\{(x,y)/z=x+y\}} min(\mu_A(x), \mu_B(y))$,
- $\mu_{A \ominus B}(z) = sup_{\{(x,y)/z=x-y\}} min(\mu_A(x), \mu_B(y))$.
- $\mu_{-A}(z) = \mu_A(-z)$.

In this work, a fuzzy set A such that μ_A is convex will be called a fuzzy interval. An interval will be positive if Θ is the real line, and $\forall x < 0, \mu(x) = 0$.

In some cases we will assume that the fuzzy interval is trapezoidal, as in figure 1. In that case, the interval will be represented by a 4-tuple $\langle a, b, c, d \rangle$.

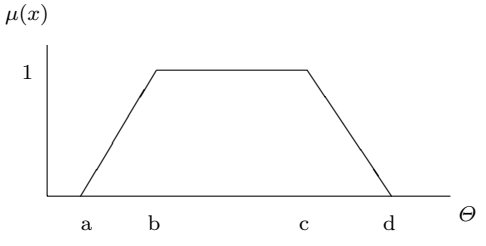


Fig. 1. A trapezoidal fuzzy interval

A fuzzy set $A = \langle a_1, a_2, a_3, a_4 \rangle$, such that $a_1 = a_2$ and $a_3 = a_4$ is a convex crisp set, and will sometimes be denoted by $A = \langle a_1, a_3 \rangle$, throughout this paper.

For a trapezoidal interval $A = \langle a_1, a_2, a_3, a_4 \rangle$ the range $[a_2, a_3]$, where $\mu_A(x) = 1$, will be called core. The range $[a_1, a_4]$, where $\mu_A(x) > 0$, will be called support.

For two trapezoidal intervals $A = \langle a_1, a_2, a_3, a_4 \rangle$, and $B = \langle b_1, b_2, b_3, b_4 \rangle$, the $\oplus$ and $\ominus$ operations are simply $A \oplus B = \langle a_1 + b_1, a_2 + b_2, a_3 + b_3, a_4 + b_4 \rangle$, and $A \ominus B = \langle a_1 - b_4, a_2 - b_3, a_3 - b_2, a_4 - b_1 \rangle$.

Throughout this paper, we shall make use of four particular fuzzy intervals. Let θ be a moment in Θ. The fuzzy intervals describing the possibility of an event occurring *at any time*, *exactly at* θ, *after* θ, and *before* θ are respectively defined as:

- $I_{\text{anytime}} = A$, such that $\forall x \in \Theta, \mu_A(x) = 1$,
- $I_{=\theta} = A$, such that $\mu_A(x) = 1$, in $x = \theta$, and $\mu_A(x) = 0$, otherwise.
- $I_{\geq\theta} = A$, such that $\forall x \in \Theta$, if $x \geq \theta, \mu_A(x) = 1$, and $\mu_A(x) = 0$, otherwise.
- $I_{\leq\theta} = A$, such that $\forall x \in \Theta$, if $x \leq \theta, \mu_A(x) = 1$, and $\mu_A(x) = 0$, otherwise.

We also use θ_0 to denote the present moment, and define $I_{beforenow} = I_{\leq\theta_0}$ and $I_{afternow} = I_{\geq\theta_0}$.

Finally, we will define that an interval A is tighter than an interval B (or, informally narrower) if $\mu_A(x) \leq \mu_B(x)$ for all $x \in \Theta$. If A and B are trapezoidal, then $A = \langle a_1, a_2, a_3, a_4\rangle$ is tighter than $B = \langle b_1, b_2, b_3, b_4\rangle$, iff $a_1 \geq b_1$, $a_2 \geq b_2$, $a_3 \leq b_3$, and $a_4 \leq b_4$.

2.2 The Knowledge Base

The knowledge base for a fuzzy temporal/categorical diagnostic problem is the information about disorders and how they evolve. The knowledge base is given by the tuple $\langle \Theta, D, M, N, P, V, T\rangle$ where:

- Θ is a time scale.
- D is the set of disorders.
- M the set of manifestations.
- N is the necessary effects function that associates to each disorder d_l a set $M_L \subseteq M$ of manifestations that d_l *necessarily* causes. That is, if $N(d_1) = \{m_4, m_5, m_7\}$ then it is not possible to have the disorder d_1 without having eventually the symptoms m_4, m_5 and m_7.
- P is the possible effects function that associates to each disorder d_l a set $M_L \subseteq M$ of manifestations that d_l *may* causes.
- We will define the derived function E, effects of a disorder, as $E(d) = N(d) \cup P(d)$.
- V associates to each disorder a set of (instantaneous) events. These events will be used to describe the evolution of the disorder. Among the events in $V(d_l)$ it must be included events that correspond to the beginning of all manifestations in $E(d_l)$. Furthermore, $V(d_l)$ can include events that correspond to the end of some of the manifestations in $E(d_l)$ and can also include other, non-observable events. For example, a common non-observable event in infectious diseases is the infection itself.
- $\mathcal{T}$ is a function that associates to *some* pairs of events $e_i, e_j \in V(d_l)$ a fuzzy temporal interval $\mathcal{T}(d_l)(e_i, e_j) = \pi$ which states that (according to the model for the disorder d_l) the elapsed time between the event represented by e_j and the event represented by e_i must be within the fuzzy temporal interval π.

Together, $V(d_l)$ and $\mathcal{T}(d_l)$ can be better understood in terms of a graph of events. The events in $V(d_l)$ are the nodes of the graph and if $\mathcal{T}(d_l)$ is defined for the pair of events (e_i, e_j) then there is a directed arc from e_j to e_i and the value in the arc is $\mathcal{T}(d_l)(e_i, e_j)$. We will call such interpretation as the temporal graph of the disorder.

2.3 Case Information

For a particular diagnostics problem, one needs, besides the knowledge base about the disorders, a particular case. The case information should describe the

manifestations that the patient is suffering and have suffered from, temporal information about those when those symptoms started and ended, and information about manifestations that the diagnostician knows are not present in the patient.

Information about a given case is modeled by a tuple $Ca = \langle M^+, M^-, \mathrm{EV}^+, \mathrm{TIME}^+, \theta_0 \rangle$, where

- M^+ is the set of manifestations known to be or to have been present in the case.
- M^- is the set of manifestations known to be absent from the case.
- EV^+ is a set of events for which one has temporal information. Among the events in EV^+ are the ones that represent the beginning of each manifestation in M^+. Events representing the end of the manifestations in M^+ may also belong to the set EV^+.
- TIME^+ is a function that associates to each event $e \in \mathrm{EV}^+$ a fuzzy temporal interval that represents the possible moments in which that event happened.
- θ_0 is the moment of the diagnosis.

For instance, in our example, we could have a piece of information such as "the patient had nausea (m_2), starting 24 hours before the consultation, which lasted for about 2 or 3 hours, and he is sure he did not have abdominal cramps (m_1)". We would then have $M^+ = \{m_2\}$, $M^- = \{m_1\}$ and $\mathrm{EV}^+ = \{m_2^b, m_2^e\}$. If we consider that the consultation happened at time 0, the temporal information above would be translated as $\mathrm{TIME}^+(m_2^b) = \langle -24, -24 \rangle$ and $\mathrm{TIME}^+(m_2^e) = \langle -22, -21 \rangle$. Of course, all the temporal information can be given in terms of fuzzy intervals.

3 Consistency between a Model and a Case

3.1 Minimal Network

Before discussing measures of consistency between a disorder model and a patient case we will present the concept of a minimal network [11]. Given a set of intervals among some events, the minimal network is way to compute the intervals between any two of those events, so that those computed intervals are as tight as possible.

The minimal network for each disorder d_l can be computed by the algorithm below, the Floyd-Warshall algorithm, which computes the shortest path for every pair of nodes [3, Chap.26]. We assume that the events in $V(d_l)$ are arbitrarily numbered, and that $|V(d_l)| = n$. The algorithm computes the values t_{ij} with is the interval between events e_i and e_j in the minimal network for a particular disorder d_l.

```
1    for i = 1 to n do
2        for j = 1 to n do
3            if i = j then t_ii = I_=0
4            else if T(d_l)(e_i, e_j) is defined then t_ij = T(d_l)(e_i, e_j)
5            else if  T(d_l)(e_j, e_i) is defined then t_ij = -T(d_l)(e_j, e_i)
```

6 **else** $t_{ij} = I_{\text{anytime}}$
7 **for** k = 1 **to** n **do**
8 **for** i = 1 **to** n **do**
9 **for** j = 1 **to** n **do**
10 $t_{ij} = t_{ij} \cap (t_{ik} \oplus t_{kj})$

We will define a function $\mathcal{T}^*(d_l)(e_i, e_i)$ which returns the value of t_{ij} in the minimal network for disorder d_l. We will abbreviate $\mathcal{T}^*(d_l)(e_i, e_i)$ as $\pi_l(e_i, e_j)$, and if the disorder is clear by the context, we will not use the superscript.

In terms of the graph analogy of V and $\mathcal{T}$, the minimal network computes the transitive closure of the graph, considering that if there is an arc from e_j to e_i with value $\pi_l(e_i, e_j)$, then there should be an arc from e_i to e_j with value $-\pi_l(e_i, e_j)$ (line 5 in the algorithm).

3.2 Temporal Consistency

In evaluating the temporal consistency between the case and the model, one needs to compare the elapsed time between the events in the case (the events in EV^+) and the corresponding fuzzy intervals as specified in the model.

The fuzzy temporal distance between the real occurrences of any two events e_i and e_j is computed as $\text{DIST}^+(e_i, e_j) = \text{TIME}^+(e_j) \ominus \text{TIME}^+(e_i)$.

In order to verify how well these two events fit with the model of a particular disorder d_l we must compare $\text{DIST}^+(e_i, e_j)$ with $\pi_l(e_i, e_j)$ if both e_i and e_j belong to that disorder model. The temporal distance between two events e_i and e_j, taking into account both the model and the case information is given by $\text{DIST}^t(e_i, e_j) = \text{DIST}^+(e_i, e_j) \cap \pi_l(e_i, e_j)$. The degree of consistency between d_l and the case information in relation to the pair of events e_i and e_j is then the height of $\text{DIST}^t(e_i, e_j)$, i.e. $h(\text{DIST}^t(e_i, e_j))$.

Finally, the temporal consistency degree of the disorder d_l is defined as:

- $\alpha(d_l) = \inf_{e_i, e_j \in \text{EV}^+, V(d_l)} h(\text{DIST}^t(e_i, e_j))$

3.3 Categorical Consistency

Categorical consistency between model and case refers to the fact that a necessary manifestation of a disorder must happen, if the patient is suffering from that disorder. If the case does not have a manifestation then no disorder that considers that manifestation necessary can be a possible diagnostic, or be part of a possible diagnostic. But categorical inconsistency is tightly bound with temporal reasoning. In fact we can only state that a case is categorically inconsistent with the model if a necessary manifestation has not occurred and there has been enough time for it to happen.

One can say that a manifestation m_i had had enough time to occur in d_l if

- there exists an event e_j, which was supposed to happen after the start of m_i, and that event has already occurred;

– or there exists an event e_j, which was supposed to happen before the start of m_i, and that event did happen as expected, but the expected elapsed time between the event and the start of m_i has already expired.

Categorical consistency can be calculated as temporal consistency if we assume that all necessary manifestations that have not yet occurred will start sometime after the moment of consultation. If, because of either the two reasons above, there is other temporal information that states that this event should have already started, the temporal consistency index of the disorder will reflect it. Thus, with the initialization

– $\forall m_i \in M^- \cap N(d_l), \mathrm{TIME}^+(m_i^b) = I_{\mathrm{afternow}}$,

the temporal consistency index $\alpha(d_l)$, will reflect both the temporal and the categorical consistency. We will call this combined temporal and categorical index as $\alpha_{ct}(d_l)$.

3.4 Intensity Consistency

In some disorders, it is important to quantify the intensity with which some of its manifestations occur. For instance, let us suppose a given disease is characterized by strong fever at some time during its development; in this case, it is reasonable to suppose that that disorder will be the less plausible, the lower the temperature of the patient.

We use a fuzzy set S_{m_i/d_l} to model the intensity with which m_i is expected to occur in d_l. Each fuzzy set S_{m_i/d_l} is defined on its particular domain Ω_{m_i}. For manifestations m_i for which intensity is not a relevant matter in d_l, S_{m_i/d_l} is constructed as $\forall x \in \Omega_{m_i}$, $\mu_{S_{m_i/d_l}}(x) = 1$. When the intensity can be quantified by a precise constant x^* in Ω_{m_i}, S_{m_i/d_l} is constructed as $\mu_{S_{m_i/d_l}}(x) = 1$, if $x = x^*$, $\mu_{S_{m_i/d_l}}(x) = 0$, otherwise.

In the same way, the case information contains for each node $m_i \in M^+$ a fuzzy set $S^+_{m_i}$, defined in Ω_{m_i}, describing the intensity with which that manifestation occurred in that particular case.

The consistency of the intensity of a manifestation m_i, in relation to a disorder d_l, is quantified as follows: $\beta(m_i, d_l) = h(S_{m_i/d_l} \cap S^+_{m_i})$.

Finally, for a disorder d_l its intensity consistency is given by

– $\beta(d_l) = \inf_{m_i \in E(d_l)} \beta(m_i, d_l)$.

3.5 Diagnostic Explanation

In this paper we assume that every explanation, or better diagnostic explanation, is a single disorder that is temporal, categorical, and intensity consistent with the symptoms and explains all symptoms present in the case. Thus d_l is a diagnostic for the case $Ca = \langle M^+, M^-, \mathrm{EV}^+, \mathrm{TIME}^+, \theta_0, \mathrm{INT}^+ \rangle$, if

– $\alpha_{ct}(d_l) > 0$
– $\beta(d_l) > 0$
– for all $m_i \in M^+$, $m_i \in E(d_l)$.

4 Possibilistic Vertices

So far, we have assumed that experts can give positive opinion about a manifestation m_i being necessary or only possible in the context of a given disorder, denoted as $m_i \in N$ and $m_i \in P$ respectively. Let us now suppose that for some of the possible, but not necessary manifestations, the expert is also capable of giving a frequency with which it occurs in the disorder.

For instance, he may say that "when the patient is suffering from d_l, m_i occurs 10% of the time", which is a way of saying that it is possible for m_i to occur in d_l, but that this is rarely the case. Or else, he may say that "m_i occurs 90% of the time", which means that m_i is not only possible but is very frequentely present in d_l. In the following we outline how our framework could be extended to deal with this kind of information.

Let $f_{m_i^+/d_l} \in [0,1]$ denote the frequency with which m_i is present when disorder d_l occurs. The frequency with which m_i is absent given that d_l occurs, is denoted by $f_{m_i^-/d_l} = 1 - f_{m_i^+/d_l}$.

When $m_i \in N$ for a given d_l, ie m_i is necessary in d_l, we have $f_{m_i^+/d_l} = 1$. On the other hand, when $m_i \in P$ for a given d_l, ie m_i is possible but not necessary in d_l, we have $0 < f_{m_i^+/d_l} < 1$. If a manifestation m_i is considered to be impossible to happen in d_l then we have $f_{m_i^-/d_l} = 1$.

Let $\pi_{m_i^+/d_l} \in [0,1]$ (respec. $\pi_{m_i^-/d_l} \in [0,1]$) denote the possibility degree that m_i is present (respec. absent) when disorder d_l occurs. These values are such that $\max(\pi_{m_i^+/d_l}, \pi_{m_i^-/d_l}) = 1$ and are obtained using the following procedure, adapted for the dichotomic case from [7]:

if $f_{m_i^+/d_l} > f_{m_i^-/d_l}$ **then** $\{\pi_{m_i^+/d_l} = 1, \pi_{m_i^-/d_l} = 2 * f_{m_i^-/d_l}\}$
else $\{\pi_{m_i^+/d_l} = 2 * f_{m_i^+/d_l}, \pi_{m_i^-/d_l} = 1\}$

For instance, when m_i is always present whenever d_l occurs, we have $\pi_{m_i^+/d_l} = 1$ and $\pi_{m_i^-/d_l} = 0$, whereas when m_i never happens in d_l, we have $\pi_{m_i^+/d_l} = 0$ and $\pi_{m_i^-/d_l} = 1$. In the example given above, in which m_i happens in 90% of the cases in d_l, we have $\pi_{m_i^+/d_l} = 1$ and $\pi_{m_i^-/d_l} = .2$. Alternatively, when m_i happens in 10% of the cases in d_l, we have $\pi_{m_i^+/d_l} = .2$ and $\pi_{m_i^-/d_l} = 1$.

Now, we have to compare this information with the patient data, given by M^+ and M^-. Also this kind of information could be uncertain, although not frequentist in nature. For instance, a patient may believe that m_i happened rather than the other way around. We will however assume here that the patient data is still supplied in terms of M^+ and M^-.

Using M^+ and M^- provided by the patient, for each m_i in d_l, we obtain the possibility of manifestation having or not occurred, denoted by $\pi_{m_i^+}$ and $\pi_{m_i^-}$, in the following manner:

if $m_i \in M^+$ **then** $\{\pi_{m_i^+} = 1, \pi_{m_i^-} = 0\}$
else if $m_i \in M^-$ **then** $\{\pi_{m_i^+} = 0, \pi_{m_i^-} = 1\}$

else $\{\pi_{m_i^+} = 1, \pi_{m_i^-} = 1\}$

The compatibility of the occurrence of m_i in d_l is given by

- $\psi(m_i, d_l) = \max[min(\pi_{m_i^+/d_l}, \pi_{m_i^+}), min(\pi_{m_i^-/d_l}, \pi_{m_i^-})]$

and the overall occurrence compatibility of d_l is given by:

- $\psi(d_l) = \inf_{m_i \in M} \psi(m_i, d_l)$

Therefore a disorder will have low occurrence compatibility degree when the patient presents manifestations considered to be rare in d_l, and also when if he does not present manifestations considered to be frequent in d_l. This scheme is equivalent to that presented in [6].

Using the original M^+, M^-, N and P, the other indices (temporal, categorical...) remain unchanged, and $\psi(d_l)$ can be seen as only an additional information for the physician. However, they could be used to affect the other indices, but this remain an issue for future investigation.

5 Conclusions and Future Work

This work presented a model to include fuzzy temporal information, categorical information, and (fuzzy) intensity information within a diagnostic framework. We provided answers to the following questions: when is the temporal information in the case consistent with a disorder model, when is the case categorically consistent with the model, and how information about intensity can be included. We have also outlined how to treat "fuzzy" categorical information, making it possible to model pieces of information furnished by a medical expert such as "in disorder d_l, manifestation m_i is very likely to occur" or "in disorder d_l, m_i will seldom occur".

In this paper we are not concerned on how the temporal information about the disorder model ($\mathcal{T}(d_l)$) is obtained (see [1] for details about this problem). Also, we are not concerned on how the temporal information about the case (TIME$^+$) is obtained, and assume here that the information about the case is internally consistent. [11] discusses this problem, and proposes a method to evaluate the consistency of the information and the most precise intervals for the occurrence of the events using minimal networks. Other researchers have also discussed similar issues [5]. The approach presented here yields only possibilistic compatibility degrees, but could be modified to obtain also entailment degrees, as in [11,12].

This work extends a previous work by the authors [14]. In [14] it is assumed that the temporal graph of the disorder is a tree. In order to calculate the temporal consistency of the case and the model, the case information was propagated towards the root; any temporal inconsistency would result in conflicting intervals for the root. We have also been studying, once a diagnostic has been made, how can one make forecasts about future manifestations, past manifestations that

have not been tested for, and so on, based not only on what the disorder model predicts but also based on how fast the case is progressing [15].

In the future, we intend to exploit the case in which the set of all manifestations presented by the patient can only be explained by a set of disorders, rather than by a single disorder, as addressed here. When all disorders in an explanation do not have any common manifestations, it seems that the theory above could be generalized by calculating the consistency indices for each disorder and attributing global consistency indices to the explanation as the minimum of the disorder's indices. However, it is not yet clear what should be done when a set of disorders explaining the manifestations have manifestations in common.

References

1. S. Barro, R. Marin, J. Mira, and A.R. Paton. A model and a language for the fuzzy representation and handling of time. *FSS*, 61:153–175, 1994.
2. L. Console and P. Torasso. Temporal constraint satisfaction on causal models. *Information Sciences*, 68:1–32, 1993.
3. T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.
4. D. Dubois and H. Prade. *Possibility Theory: an approach to computerized processing of uncertainty*. Plenum Press, 1988.
5. D. Dubois and H. Prade. Processing fuzzy temporal knowledge. *IEEE Trans. on S.M.C.*, 19(4), 1989.
6. D. Dubois and H. Prade. Fuzzy relation equations and causal reasoning. *Fuzzy Sets and Systems*, pages 119–134, 1995.
7. D. Dubois, H. Prade, and S. Sandri. On possibility/probability transformations. In R. Lowen and M. Roubens, editors, *Fuzzy Logic: State of the Art*, pages 103–112. Kluwer, 1993.
8. I. Hamlet and J. Hunter. A representation of time for medical expert systems. In J. Fox, M. Fieschi, and R. Engelbrecht, editors, *Lecture Notes in Med. Informatics*, volume 33, pages 112–119. Springer-Verlag, 1987.
9. W. Long. Reasoning about state from causation and time in a medical domain. In *Proc. of the AAAI 83*, 1983.
10. G. L. Mandell, R. G. Douglas, and J. E. Bennett, editors. *Principles and practice of infectious diseases*. Churchill Livingstone, 4rd edition, 1995.
11. L. Vila and L. Godo. On fuzzy temporal constraint networks. *Mathware and Soft computing*, 3:315–334, 1994.
12. L. Vila and L. Godo. Possibilistic temporal reasoning on fuzzy temporal constraints. In *Proc. IJCAI'95*, 1995.
13. J. Wainer and A. Rezende. A temporal extension to the parsimonious covering theory. *Artificial Intelligence in Medicine*, 10:235–255, 1997.
14. J. Wainer and S. Sandri. A fuzzy temporal/categorical extension to the parsimonious covering theory. In *The Seventh Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU'98)*, Paris, 1998. To be published.
15. J. Wainer and S. Sandri. Fuzzy temporal/categorical information in diagnosis. *Special Issue on Intelligent Temporal Information Systems in Medicine, JIIS*, 1998. Submitted.

Experiments on a Memory Structure Supporting Creative Design

Paulo Gomes, Carlos Bento

Centro de Informática e Sistemas da Universidade de Coimbra
Departamento de Engenharia Informática
Polo II – Pinhal de Marrocos
3030 Coimbra – Portugal
{pgomes | bento}@ dei.uc.pt

Abstract. Creative Design is an activity that involves complex cognitive tasks like analogy and concept composition. This makes the memory structure an important aspect of a Case-Based Reasoning Creative Design System. The indexing structure must support case retrieval based on abstract features and partial case retrieval. This allows the system to perform two important case adaptations: analogy and case composition. In this paper we present an index structure suitable for case retrieval for three adaptation strategies: mutation, composition and thematic abstraction. This memory structure also aids the designer to define and elaborate the design problem. We present the experimental results obtained with CREATOR a case-based design system that uses the memory structure presented in this paper.

1 Introduction

The creative capabilities of the human mind can generate new and surprising artifacts. The processes used for creative generation have been the focus of research in the Artificial Intelligent and Cognitive Science fields [2] [3] [5]. In this paper we present a memory structure capable of supporting creative design processes.

Besides rules and knowledge about existing components, designers often use their experience about past designs to solve new problems [11]. This makes design a task involving a huge amount of experiences and domain knowledge. The computational model used within our framework integrates Case-Based Reasoning (CBR) and Model-Based Reasoning (MBR). In design tasks these reasoning mechanisms complement each other. While CBR is suited for synthesis tasks, MBR is adequate for analysis tasks [14].

Design usually deals with three different levels of knowledge: functional, structural and behavioral knowledge. Functional knowledge represents 'what is the design for' and is associated with the design problem. The designer usually defines the problem in terms of functional requirements. The design structure comprises

F. Moreira de Oliveira (Ed.): SBIA'98, LNAI 1515, pp. 191-200, 1998.

components and relations between them, representing 'what the design is'. Behavioral knowledge connects functional and structural knowledge explaining how the structure achieves the design functions.

In creative design the understanding of how devices work is important, allowing analogies and other cognitive processes to be performed [12]. Thus a case representation needs to possess the three levels of knowledge described above. Within our framework we use the Structure-Behavior-Function (SBF) formalism proposed by Goel [7] to represent design cases.

In order for a CBR system to use functional knowledge in the problem definition, the memory structure must be able to index cases by their functional properties. We present an indexing scheme based on design functions. This memory structure allows also the retrieval of cases that partially match the new problem for composition of design parts, which is also an important adaptation process in creative design.

One important aspect in systems for creative design support is the capability of proposing design alternatives, so that the designer can become aware of important issues in the design problem specification [6]. The memory structure presented here allows the exploration of the case library for mutation, composition and thematic abstraction, searching for cases relevant to the new problem.

One of the most important reasoning mechanisms in creative design is analogy [2]. This process comprises transfer of knowledge between different cases, either within the same domain (intra-domain analogy) or between different domains (inter-domain analogy). Analogy involves a process of abstraction from a source situation to the target case. In case-based reasoning systems the support of this kind of reasoning implies the capability to index cases by abstract features. The proposed indexing structure exhibits this capability by allowing thematic abstraction retrieval.

Sycara and Navinchandra [13] performed another important work in this area. They proposed a thematic abstraction hierarchy of influences as indexes for case retrieval. Case representation in KRITIK [7] makes use of Structure-Behavior-Function (SBF) models. Through the use of these models, cases are indexed by the functionalities they exhibit. KRITIK's successor, IDEAL, indexes design cases by structural features, in addition to functionalities [1]. IM-RECIDE [10] focus on the exploration of the case library, in order to provide alternative solutions to the user. The case library is divided into different search spaces. Cases in each space have specific properties in common, which allows searching for design alternatives by incremental exploration of the case library.

In the next section we present our case-based model for creative processes. The case representation is explained in section 3 and is necessary to understand the memory structure presented in section 4. Experimental results are shown in section 5. Finally conclusions and remarks are presented in section 6.

2 A Case-Based Model for Creative Design Processes

Creative design processes involve the exploration of the design space defined by the possible designs in the working domain [3][5]. By searching areas in the design space

not yet explored, the possibility of finding creative solutions increases. In a case-based system, the design space is implicitly defined by the cases in the case library, and space exploration is performed using adaptation mechanisms.

We implemented a creative design system called CREATOR. The central idea of CREATOR is to help the designer by suggesting cases related to the new problem. Besides this goal, there is another important task, which is to help the designer in the exploration of the functional space corresponding to the problem definition and elaboration. Figure 1 shows the main steps in CREATOR reasoning.

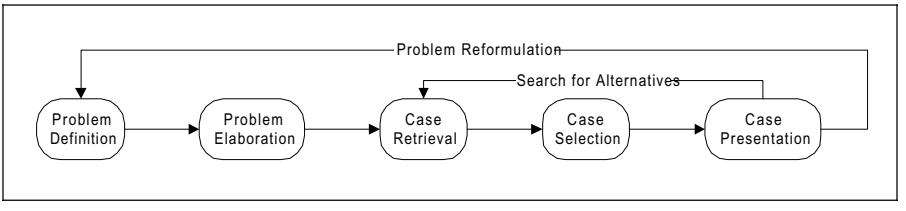


Fig. 1. CREATOR reasoning steps.

The first step is the definition of the problem. In this step the designer uses the memory structure to specify the functional requirements of the problem [8]. CREATOR provides the user with a graphical editor for selection of the required functions.

If the problem defined by the designer is not completely specified, the system uses an algorithm to elaborate the problem. This can result in several alternative problem specifications. The resulting problem specifications give the designer alternative views on the problem, thus allowing him to explore the functional design space searching for relevant functions.

After the selection of the problem specification by the user, the system uses the retrieval algorithms to search the case library. CREATOR has three different retrieval algorithms. The algorithm that will be applied depends on the adaptation strategy that will take place (in future work an adaptation and verification module will be added to the system providing it with the capability of generating solutions).

The three different adaptation strategies are mutation, composition and thematic abstraction [9]. When the designer selects one retrieval algorithm he is also choosing which type of case transformations he wants to perform. So, if the retrieval algorithm for mutation is selected, cases are retrieved taking into account this kind of adaptation strategy.

The result from the retrieval phase is a set of cases, which are going to be ranked by the metrics associated to the retrieval algorithm that is used, resulting in an ordered set of cases. This set is presented to the designer for inspection. After browsing through the cases suggested by the system, the user can reformulate the problem, or ask for more alternative cases.

3 Case Representation

Within our framework, design episodes are represented in the form of SBF models [7]. These models are based on the component-substance ontology developed by

Bylander and Chandrasekaran [4]. A case comprises three distinct parts: problem specification, solution and explanation. The problem specification comprises a set of high level functions (HLFs) and functional specifications (FSs) that the new design must have. The case solution describes the design structures. The explanation is represented in terms of sequences of transitions between behavioral states.

While high level functions are abstract functions used to help the user specifying the design problem, functional specifications are detailed functions described by behavior states. A design problem is a tree of functions, where leaves are lists of functional specifications and the other nodes in the tree represent high level functions (see Figure 2). This taxonomy of functions makes the task of specifying the design requirements more flexible and easy to perform. Two function lists define high level functions: a HLF list and a FS list. A functional specification is a schema that represents the initial behavior state, the final behavior state, a pointer to the internal causal behavior of the design, inputs from the external environment and structural constraints.

The design solution is a partonomic hierarchy of device structures. Each structure can be viewed as a set of connected sub-structures, where substances can flow through structures.

A case explanation is the causal behavior of the design in terms of directed acyclic graphs (DAGs). The nodes of a DAG represent behavioral states and the edges represent state transitions. A behavioral state can be composed of one or more substance schemas. A substance schema characterizes the location, properties, and property values of a substance. The state transition schema represents constraints associated with the transition between behavior states.

4 Memory Structure

The memory structure comprises two interconnected substructures: a tree of functions and a graph of cases. The functional tree comprises HLFs and FSs and has two purposes. One is to provide help for the problem specification and elaboration phases. The other role is in retrieving the starting case. The starting case is the case with the higher number of FSs in common with the new problem. This case is used as a starting point in the graph of cases for exploration of the episode memory, making possible for the search mechanism to incrementally retrieve similar neighbor cases.

The tree of HLFs and FSs comprises three different types of nodes and three types of links (see Figure 2). Nodes are of type: HLF class, HLF instance, and LF node (a LF node comprises a list of HLFs and/or FSs). Connections between nodes are of type: *Ako*, *Isa*, and *Comp* links. An *Ako* link represents that a HLF class is a kind of a HLF superclasse. An *Isa* link means that a HLF instance is a HLF class. A *Comp* link connects a list of functionalities (HLFs and/or FSs) to a HLF instance. This kind of connection represents a partonomic relation between a HLF instance and the list of HLFs and FSs in the LF node.

In the graph of cases, nodes represent episodes and edges represent functional differences between cases. In Figure 2 *Case1* is connected to *Case2* by the link

Differences1-2. This link represents the functional differences between *Case1* and *Case2* problem descriptions.

A difference link is created only when the cases that it connects have at least one FS in common. A difference link connecting two cases, comprises three parts:

- The set of FSs that belong to the first case but do not belong to the second one.
- The set of FSs that belong to the second one but do not belong to the first one.
- The set of differences in the FSs common to both cases.

Case links are automatically generated, when a new case is stored in the case library.

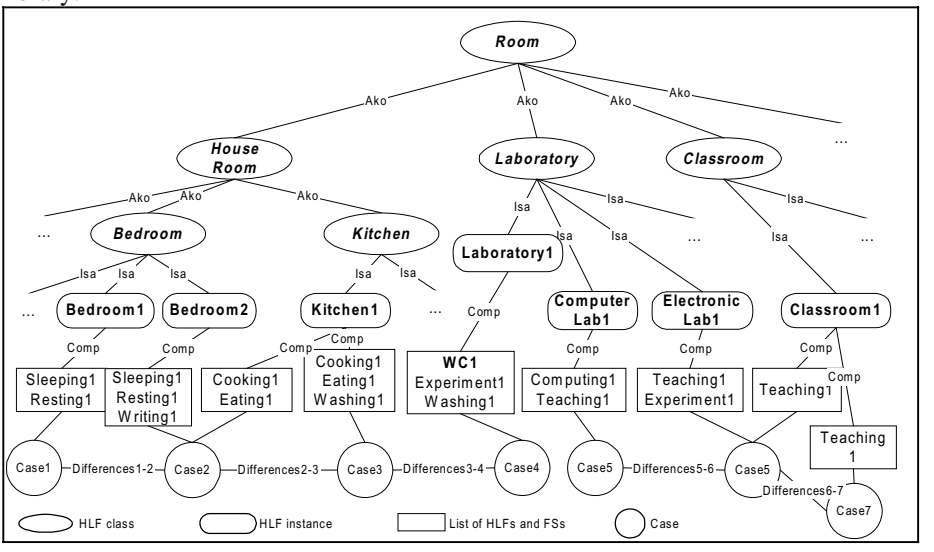


Fig. 2. Example of a memory structure in the domain of room configuration.

While the functional tree allows accuracy in the retrieval of the initial case, the case network allows a gradual search of the case library to be performed with temporal efficiency. Besides these two advantages the functional tree permits thematic abstractions based on functional knowledge, and partial matching.

5 Experiments

Experiments were done to measure the system behaviour and performance in searching the case library. The three retrieval algorithms were used to test the memory structure characteristics. Ten different problems were used as probes to search relevant cases in memory. A score was given to each case in the library in regard to the problem tested. In this way, associated to each case there were ten different scores. The score represents the similarity between the new problem and the case problem description. Ten different runs of the system (each one relating to one test problem) were done. The user scored cases retrieved by the system accordingly to

the previous defined scores. Each run comprises several iterations and for each iteration it is calculated the average and standard deviation of the cases score. Several statistical results were produced and are presented in the next subsections.

Two different domains were used: design of electric circuits and design of digital circuits. The library for the first domain comprises cases representing electrical circuits used in rooms, like illumination circuits or sound systems. Cases in this library are conceptually apart from each other. This means that there are few cases with common HLFs, thus being far away from each other in the functional tree. The case library has twenty cases covering uniformly the design space.

The digital circuit library comprises fifty cases and describes circuits comprising logic gates implementing logical functions. A problem in this domain comprises a set of logical functions that the design must implement. The solution is a set of connected logic gates. This case library has cases conceptually close, in the sense that most of the cases have common HLFs. This type of case library provides a more clustered coverage of the design spaces.

5.1 Results in the Electric Circuit Domain

The results obtained for each retrieval algorithm are presented in Figures 3, 4 and 5, respectively, retrieval algorithm for mutation, composition and thematic abstraction. The graphs show the average and standard deviation of the score obtained for the retrieved cases in each iteration.

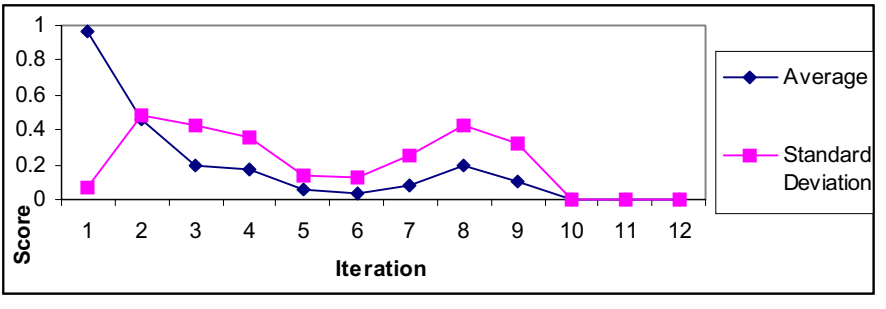


Fig. 3. Scores obtained by the retrieved cases using the retrieval algorithm for mutation in the domain of electric circuits.

As can be depicted from the graphs, the retrieval algorithms start retrieving the most similar cases to the new problem. Then the algorithms gradually get further away from the area defined by the new problem. This is the expected behavior, since the algorithms start exploring the case network, gradually searching for cases more distant from the initial cases.

In Figure 3 we can see a significant increase in the score average. This happened because the retrieval algorithm found a more relevant case or set of cases. A question that can be made is why weren't these cases found before? The answer is, because the search space defined by the case descriptions is not uniform or regular. This is also

due to the partially indexing scheme that was used, allowing the cases to be indexed by distant features in the functional space.

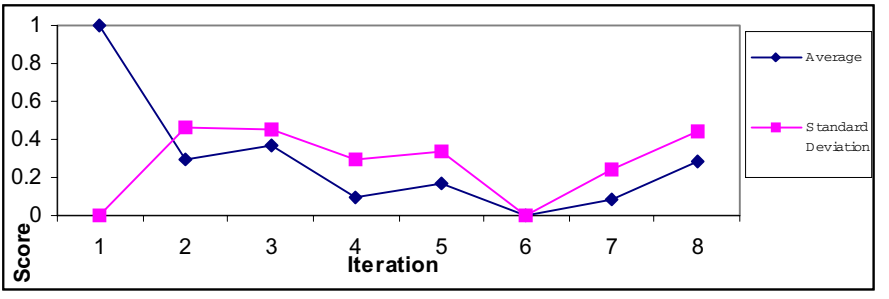


Fig. 4. Scores obtained by the retrieved cases using the retrieval algorithm for composition in the domain of electric circuits.

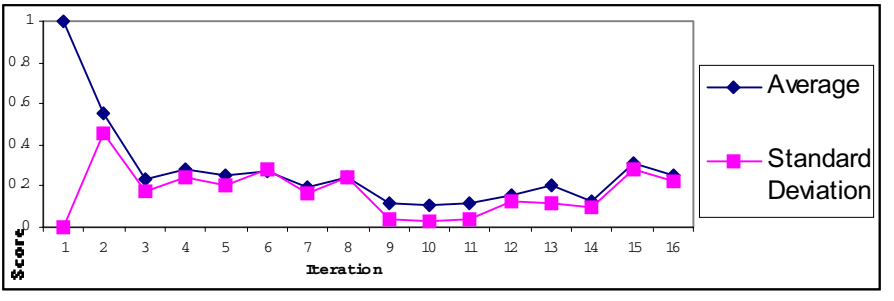


Fig. 5. Scores obtained by the retrieved cases using the retrieval algorithm for thematic abstraction in the domain of electric circuits.

Similar situations happen in Figures 4 and 5. We must stress that the retrieval algorithm for composition searches for sets of cases instead of cases alone. This is due to the adaptation strategy, which is to combine different parts of cases in order to generate a new solution. As can be seen in the graph, this algorithm searches exhaustively the case library in seven iterations while the other algorithms take at least twelve iterations. This algorithm is aided by the partially indexing scheme, which allows it to retrieve cases that have at least one relevant part. In contrast, the other two algorithms compute the case similarity to the new problem taking the case as a whole piece.

The retrieval algorithm for thematic abstraction has a similar behavior to the first algorithm. From the three algorithms this is the one that makes a more uniform search in the case library, due to the regular distribution of cases in memory.

An important aspect of these experiments is that the first iteration always takes more time when compared with other iterations. The reason for this is that in the first iteration the tree of HLFs and FSs is searched in order to find the initial case. Once this case is selected the algorithms only have to search cases using the difference links

in the case network, thus being less time consuming. If taking into account that a real system may work with thousands of cases, this characteristic is very important for temporal efficiency.

5.2 Results in the Digital Circuit Domain

The results obtained in the design of digital circuits are presented in Figures 6 and 7. This experiment shows that the retrieval algorithm for thematic abstraction is not the best retrieval algorithm for this kind of library. This happens because most of the cases in this library have common HLFs, so the retrieval algorithm for thematic abstraction that works in the HLF level of similarities can not work properly. This algorithm can not identify functional differences among the cases and all the cases in the library are retrieved in the first three iterations. This is an expected result because the cases in the library are conceptually near to each other and the retrieval algorithm for thematic abstraction can not make abstractions.

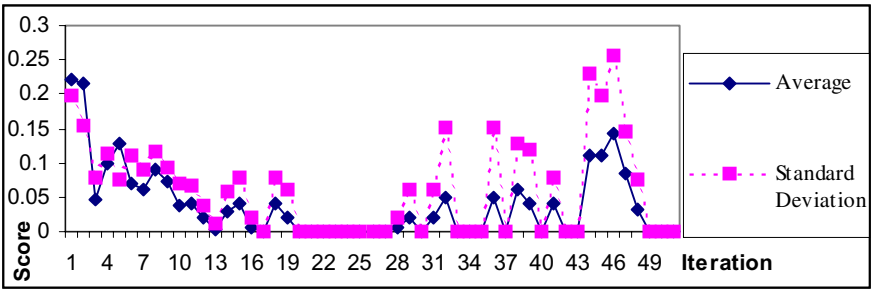


Fig. 6. Scores obtained by the retrieved cases using the retrieval algorithm for mutation in the domain of digital circuits.

From Figure 6 we can see that the type of the domain is very important to the performance of the retrieval algorithm for mutation. The peaks observed in iterations 29, 33, 36, 41 and 44 to 48 correspond to relevant cases that were found. When the algorithm begins searching, it starts by the initial case which can have only some required FSs, but not all of them. The FSs that were missing are present in the cases found in the latter iterations, explaining the peaks in the graph. If a new problem comprises a set of FSs that are not present in the cases of the same neighborhood these peaks appear. This happens because the FSs are spread across the case network. If the FSs are present in cases near by (in the case network) then the retrieval algorithm does not show these kinds of irregularities. One important thing is that similarity among cases in this library depends more on the FSs than on the HLFs. This aspect is more relevant in this domain than in the electric circuit library.

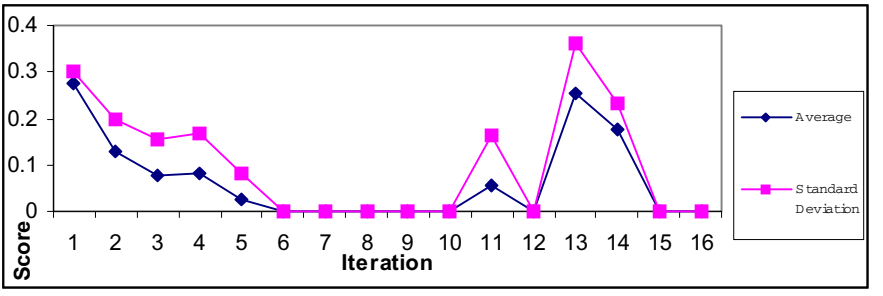


Fig. 7. Scores obtained by the retrieved cases using the retrieval algorithm for composition in the domain of digital circuits.

The retrieval algorithm for composition presents similar effects to the previous algorithm. The algorithm found new cases comprising FSs common to the new problem in iterations 11, 13 and 14. This rises the score a lot because what matters in this retrieval algorithm is how well can a set of cases be complementary in regard to the new problem, so when a case with a FSs not present in the explored cases is found the case scores goup. Again this is a characteristic of the case library and does not depend on the algorithm.

6 Conclusion

We presented a memory structure capable of producing efficient case retrieval and exploring the case library for generation of design alternatives at the same time. This memory structure is based on a case representation capable of dealing with functional, behavioral and structural knowledge enabling the exploration in these three different search spaces.

Because the memory structure supports three different retrieval methods, the performance of each retrieval algorithm depends on the characteristics of the case library, as can be seen from the results of the previous section. While the retrieval algorithms for mutation and composition search the case library using the FSs similarities stored in the case network links, the retrieval algorithm for thematic abstraction uses the tree of functions to compute the HLF similarity between cases, thus searching at a different conceptual level.

In the experiments conducted with CREATOR three main conclusions can be drawn. The first one is that the case network allows an efficient retrieval of several alternative cases. Another conclusion is that indexing cases by their functions enables partial retrieval, thus being an appropriate indexing scheme for retrieving cases for composition. This indexing scheme also demonstrated to be adequate for retrieving cases for mutation. The tree of functions allows thematic abstraction retrieval based on functionalities. The memory structure also aids the user specifying the problem and is also used for elaboration of the problem.

All these conclusions can be explained in terms of the conceptual distance between the problem and the cases used for achieving the solution. This is what exploration of the case library means: searching for design alternatives in cases further away from the problem. The risk of finding bizarre solutions increases, but the chance of finding a novel and possibly original solution is also higher.

7 References

1. Bhatta, S., and Goel, A. 1993. Model-Based Learning of Structural Indices to Design Cases. In *Proceedings of IJCAI-93, Workshop on Reuse of Designs: An Interdisciplinary Cognitive Approach*.
2. Bhatta, S. and Goel, A. 1997. An Analogical Theory of Creativity in Design. In *Proceedings of the International Conference on Case-Based Reasoning (ICCBR 97)*, Providence - Rhode Island, EUA.
3. Boden, M. 1994. What is Creativity? In Boden, M. (Ed.) *Dimensions of Creativity*. The MIT Press.
4. Bylander,T., and Chandrasekaran, B. 1985. Understanding Behavior Using Consolidation. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*.
5. Gero, J. 1994. Computational Models of Creative Design Processes. In Dartnall, T. (Ed.), *Artificial Intelligence and Creativity*. Kluwer Academic Publishers.
6. Gero, J., and Maher, M., 1993. *Modelling Creativity and Knowledge-Based Creative Design*. Lawrence Erlbaum Associates, Sydney.
7. Goel, A., 1992. Representation of Design Functions in Experience-Based Design. *Intelligent Computer Aided Design*. D. Brown, M. Waldron, H. Yosnikawa (Eds.), Elsevier Science Publishers.
8. Gomes, P. and Bento, C. 1997. A Case-Based Approach for Elaboration of Design Requirements. In *Proceedings of the International Conference on Case-Based Reasoning (ICCBR 97)*, Providence - Rhode Island, EUA.
9. Gomes, P. and Bento, C. 1997. A Retrieval Method for Exploration of a Case Memory. In *Proceedings of the Portuguese Conference on Artificial Intelligence* (*EPIA 97*), Coimbra - Portugal.
10. Gomes, P., Bento, C., Gago, P., and Costa, E. 1996. Towards a Case-Based Model for Creative Processes. In *Proceedings of the 12th European Conference on Artificial Intelligence* (*ECAI 96I*). John Willey & Sons.
11. Reich, Y. 1991. *Design* Knowledge Acquisition: Task Analysis and a Partial Implementation. *Knowledge Acquisition: An International Journal of Knowledge Acquisition for Knowledge-Based System* 3(3): 234-254.
12. Simina, M. and Kolodner, J. 1997. Creative Design: Reasoning and Understanding. In *Proceedings of the International Conference on Case-Based Reasoning (ICCBR 97)*, Providence - Rhode Island, USA.
13. Sycara, K., and Navinchandra, D., 1993. Case Representation and Indexing for Innovative *Design* Reuse. In *Proceedings of the Workshop of the 13th International Joint Conference on Artificial Intelligence*, France.
14. Tong, C., and Sriram, D. Eds. 1992. *Artificial Intelligence in Engineering Design*, Vol. I. Academic Press.

Real Time Variable Precision Logic Systems

Nabil M. Hewahi
Department of Computer Science, The Islamic University
Gaza, Palestine
E-mail:nhewahi@mail.iugaza.edu

Abstract

Real time systems are time constraints systems, they should work very effectively and accurately within the given time for the system response. One important system that works within the time constraints is a rule based system based on Variable Precision Logic (VPL). VPL systems may be based on variable certainty or variable specificity or both of them. In variable certainty systems, the more is the given time, the more certain are the answers. In variable specificity systems, the more is the given time, the more specific are the answers. In this paper, we propose an algorithm for a real time system based on Hierarchical Censored Production Rule-Based (HCPR) System which takes care of both variable certainty and variable specificity as well. The algorithm distributes the time necessary for every HCPR-Tree and its related rules (HCPRs) involved in the inference and forces the system to respond within the time constraints. The proposed algorithm takes care of the required certainty of the system. It combines a tradeoff between the time constraints and the system required certainty and between the time constraints and the specificity of the conclusions given enough information. The main principle of the algorithm is; the lower is the level of the HCPR, the less is the time needed to check it, that is, the more general is our conclusion, the more certain we are. A general idea to parallelize the algorithm is provided.

1 Introduction

VPL systems have numerous real live application in situations where decisions must be made in real-time and uncertain information [3],[4],[5],[6],[8],[9],[10],[11],[12],[15]. Variable precision logic is concerned with problems of reasoning with incomplete information. It deals with both the problem of time-constrained reasoning and that of reasoning efficiently with exceptions. VPL offers mechanisms for handling tradeoffs between the precision of inferences and computational efficiency of deriving them [15]. Specificity and certainty are the two aspects of precision. A system that gives more certain answers, given more time, is called a variable certainty system. A system that gives more specific answers, given more time, is called a variable specificity system. Thus, in general, a variable precision system is a system that exhibits either variable specificity or variable certainty or some tradeoff between the two. There can be various combinations of having a tradeoff between certainty and specificity where specificity and certainty are inversely related [15].

To enable a logic-based system to exhibit variable precision in which certainty varies while specificity stays constant, Michalski and Winston [15] proposed the Censored Production Rule (CPR) as an extension of standard production rule (IF <premise> THEN <decision>). The form of CPR is as follows:

IF <premise> THEN <decision> UNLESS <censors>
and can be written

$$P \rightarrow D \lfloor C$$

F. Moreira de Oliveira (Ed.): SBIA'98, LNAI 1515, pp. 201-208, 1998.

where P is the premise, D is the decision, and C is the censor.

The premise is a conjunction of literals; the decision is a single literal; and the censor is a disjunction of literals.

Censored production rules embody both object level and control level information. The object level information resides in the fact that censors represent exceptions. Since exceptions are by definition false most of the time, we have certain expectations concerning the character of inferences made with such rules. These expectations may be used to control the inferences.

To understand the implication of CPR, Michalski and Winston [15] presented a quantitative definition for it where two parameters Υ and δ have been introduced. A CPR is then written:

$$P \rightarrow D \lfloor (C_1 \vee UNK) : \Upsilon, \delta$$

where Υ = prob (D \ P), certainty of $P \rightarrow D$ when it is not known whether ($C_1 \vee UNK$) holds. The implication $P \rightarrow D$ is certainty 1 when $C_1 \vee UNK$ is known to be false. When δ = prob ($D \backslash P \ \& \neg C_1$), it is the certainty that $P \rightarrow D$ when $\neg C_1$ is true.

Obviously the a priori certainty of $\neg (C_1 \vee UNK)$ must be equal to or smaller than the a priori certainty that $\neg$ UNK. Therefore, $\Upsilon \leq \delta$. Note that $\delta = 1$ if it is certain that there are no conditions in the censor other than C_1.

Bharadwaj and Jain [4] extended the CPR of Michalski and Winston by introducing two new operators to represent more specific and general information, and defined a hierarchical censored production rule (HCPR) of the form:

(< A	{concept/decision/action/head}
IF B[b1,b2,.....,bm]	{preconditions (AND conditions)}
UNLESS C[c1,c2,....,cn]	{censor conditions (OR conditions)}
GENERALITY [G]	{general information}
SPECIFICITY S[a1,a2,.....,ak]	{specific information}: Υ, δ>)

The general information G in an HCPR is the clue about the next general concept related to the concept A in hierarchy. Hence general information is useful in the backward chaining of inference. The specificity information S is the clue about the next set of more specific concepts (goals,decision, or actions) in a knowledge base: which are the most likely to be satisfied after successful execution of that HCPR. The set of more specific information has XOR relation between its members (only one decision in the set is true at a time). Hence, specific information is useful in forward chaining of inference. The symbols Υ, δ have their meaning as explained by Michalski and Winston.

A few related HCPRs are given below, and it is shown how they are linked in an HCPR-tree structure as depicted in Figure 1 (the numerical values of Υ, δ have been omitted for our purpose).

relevent input data .

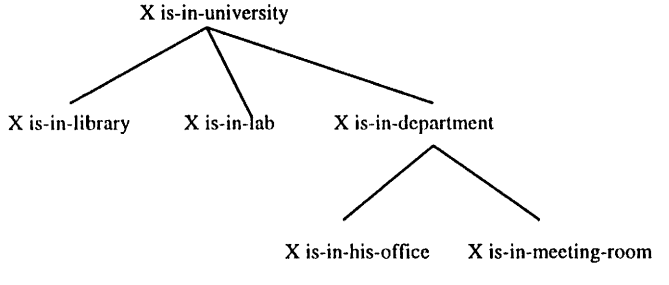


Figure 1. HCPR-tree

{level 1}

X is-in-university
IF [working-day , 9am<= time<=5 pm]
UNLESS [X is-ill,urgent-work]
GENERALITY [ ]
SPECIFICITY [X is-in-library,X is-in-his-office,X is-in-dept]

{level 2}

X is-in-library
IF [10am <=time<=11am]
UNLESS [urgent-meeting,busy-in-lab,extra-class]
Generality [X is-in-university]
SPECIFICITY[]

X is-in-lab
IF [3pm <=time <= 5pm]
UNLESS [urgent-meeting]
GENERALITY [X is-in-university]
SPECIFICITY []

X is-in-department
IF [8 <= time < 10,]
UNLESS []
GENERALITY [X is-in-university]
SPECIFICITY [X is-in-his-office, X is-in-meeting-room]

{level 3}

X is-in-his-office
IF [8 <=time < = 9]
UNLESS []
GENERALITY [X is-in-his-office]
SPECIFICITY []

X is-in-meeting-room
IF [day-is-monday, 9 <= time <=11]
UNLESS []
GENERALITY [X is-in-his-office]
SPECIFICITY []

Here, any HCPR is a more specific case of its parents. Thus, the root HCPR represents the most general concept. Once this is verified, we can descend to its children for more specific information, depending on our requirement and resources available. If any HCPR we reach is blocked, either due to one of the preconditions being false or one of the censor conditions being true, there is no need for further search of its children.

2 The proposed algorithm

The proposed algorithm is an extension of the algorithm proposed by Hewahi [10] for real time variable certainty systems where the required time necessary for every rule to fire is computed taking in to account the system time constraints and the required system certainty factor. In the case of variable certainty and variable specificity, the following factors will affect the algorithm

1) Time constriants 2) Certainty of the system conclusion
3) How much we want to be specific in our decisions.

We assume that, the time constraint is given and is denoted by **tcons**, the system should consider the given time constraints and the time that might be lost during the algorithm computation. Let us consider that the on average time for the algorithm is denoted by **at**, then the system should respond within

$$T = tcons - at \qquad (1)$$

The required overall certainty of the system conclusion is given and denoted by **SCF, where $0.5 \leq SCF \leq 1$**. This means, we want the system to respond within the time constraint with an attempt to achieve the **SCF** (if possible) and the required specificity.

Let us now consider the following terms:

T: Actual response time, assumed to be enough to fire at least the premise parts of HCPRs.
V: Average time for every HCPR-tree to get fired as a whole (upto the last level that can be reached)
nHCPR_tree: The maximum number of HCPR-trees that might participate in the inference for any given problem. This can be defined initially and depends on the behavior of the system.

L(HCPR-tree-j): The length of the jth HCPR-tree, it is the number of levels in the HCPR-tree .

SP: The specificity in percentage required for the system decisions. This is provided by the user or the system which works under the real time system.

MS(HCPR_tree_j): The maximum level of specificity that can be reached for the jth HCPR-tree.

HCPRij_time : The time specified to the HCPR at the ith level of the jth HCPR_tree (only one specific HCPR among specificities will satisfy the conditions) conditions to fire.

CF(HCPRij): The certainty factor of the HCPR at ith level of the jth HCPR-tree.

m: is the threshold certainty factor for the system . The default value is 0.5.

Ct(HCPR_tree_j) : The time consumed out of V in the process of inference for the jth HCPR_tree up to the moment t.

Ct_HCPRij : The time consumed out of HCPRij_time at the moment t.

Before getting in to the algorithm, the following formulaes are defined.

The average time for every HCPR-tree to get fired:

$$V = T / nHCPR\_Tree \qquad (2)$$

The maximum level that might be reached if possible:

$$MS(HCPR\_tree\_j) = \lceil L(HCPR\_tree\_j) * (SP/100) \rceil \qquad (3)$$

we shall use only MS instead of MS(HCPR-tree-j).

The time that might be necessary to fire certain HCPR taking into account the level of the HCPR and the system certainty threshold.

$$HCPRij\_time = (V / (m * \sum_{K=1}^{MS} k)) * (MS-i+1) * (MS*m)$$

which will give

$$HCPRij\_time = (V / \sum_{K=1}^{MS} k)) * (MS-i+1) * MS \qquad (4)$$

This formulae would give more time to upper levels of the HCPR-tree to be as certain as possible from the derived conclusion with reasonable certainty (≥ m) before going to the lower levels.

Any HCPR involved in the solution path of the conclusion should have a certainty factor not less than m. Accordingly the certainty of every HCPR might be calculated as below:

$$CF(HCPRij) = \begin{cases} 1.0-(1.0-m)^{(i/MS)} & 1 \le i < MS \\ m & MS \le i < L(HCPR\_tree\_j) \end{cases} \quad (5)$$

The formulae ensures that the last level checked will have always a certainty factor of m and if m is equal to 1, the certainty factor for all the levels that can be reached would be 1. The formulae may produce a certainty factor which is more than SCF and this will give the opportunity to be more certain if time permits (even more than what we require). This can be seen as, the higher is the HCPR level, the more is the certainty factor. When we go down, the certainty degrades till the maximum level that can be reached should not have certainty below m.

In general, we notice that the higher is the HCPR, the more is the time specified for checking it. This would allow the system to reach the specified certainty of the HCPR. This is true because whenever we take a general decision we are more certain and whenever we take a more specific answer, we are less certain .

The algorithm is then can be summerized as below:

REPEAT

1. Check the root HCPR of the chosen HCPR-tree to be fired (HCPR-trees are chosen from the matching list).
2. Store all HCPRs that have computed certainty factor more than SCF in an array X.
3. Try to satisfy the certainty factor of the chosen HCPR according to formulae (5) within the given time according to formulae (4).
4. IF the given time is not enough to reach the computed certainty (as in (5)) but the certainty achieved is equal to or above m, then
   a. Store the HCPR number along with its related HCPR-tree number. The last censor checked if any, the achieved certainty and and the HCPR required certainty are also stored. All of these are in an array A.
   b. Go down along the HCPR-tree to the next level of specificity and find the matched specificity.
   c. Go to 2 unless MS is reached or V is over.
5. IF the computed (required) certainty according to (5) is achieved before the end of the given time, then
   a. Compute the remaining time out of the time allocated to the HCPR

$$Rt = (HCPRij\_time)-(Ct\_HCPRij) \quad (6)$$

   b. The remaining time **Rt** will be distributed to the remaining HCPRs in the HCPR-tree to give the new allocated time for the remaining HCPRs. The same idea is applicable also here, that is, the higher is the level of the HCPR, the more is the allocated time to it. Assume f = i+1 (next level)

$$new(HCPRfj\_time) = old(HCPRfj\_time) + (Rt / \sum_{K=f}^{MS} k)) * (MS-f+1) * MS \quad (7)$$

   This step is important to be done before proceeding in the inference because if the case in (4) Occurs again without changing the time of all the levels, a time computation error will occur.

   c. Go to 2 unless MS is reached or V is over.
6. As a worst case, if the time allocated to a certain HCPR is not enough to achieve the m certainty factor and the HCPR computed certainty is above m, then

a. Take more time till at least m is achieved.
b. Store the HCPR number along with its related HCPR-tree number, the last censor checked if any, the achieved certainty and the required certainty in the array B.
c. Compute the newly allocated time for the next levels of specificity. Assume that f =i+1

$$\text{new(HCPRfj\_time)} = ((\text{V-Ct(HCPR\_tree\_j)}) / \sum_{K=f}^{MS} k) * ((\text{MS-f+1})* \text{MS}) \quad (8)$$

d. Go to 2 unless MS is reached or V is over.
7. For every HCPR_tree, if more time is available (V is not over) and the certainty along the HCPR_tree is satisfied then
   a. The remaining time can be used for other HCPR_trees

$$\text{Time\_left(HCPR\_tree)} = \text{V-Ct(HCPR\_tree\_j)} \quad (9)$$

   b. Compute new V for the remaining HCPR-trees

$$\text{new(V)} = \text{old(V)} + \text{Time\_left(HCPR-tree)} / (\text{nHCPR\_tree} - k) \quad (10),$$

   where k is the number of HCPR-trees checked.
   c. Go to 2 unless MS is reached or V is over.
8. If some time is still available after checking all the HCPR_trees, check the HCPRs stored in array A then the HCPRs stored in array B.
9. For every checked HCPR_tree, if there was no enough time (within V) to reach the required specificity and still some time is available after checking all the HCPR_trees, the remaining time could be used to check those unchecked specificites related to various HCPR_trees. This can be done at random to give equal opportunity to all HCPR_trees.
10. After applying all the previous steps and still there is more time available, check array X to test more HCPRs that have certainty factor more than SCF. This will allow us to be more certain than SCF if the time permits. For every HCPR in X, the certainty achieved and the last censor checked should be known.

3 A proposal to modify the algorithm for parallel HCPR system

Like standard production rules, CPR and HCPR also suffer from slow execution speeds, which limit their utility in research setting as well as real time applications. Many parallelization sources have been tried. Most of these attempts are concerned with parallelization on match phase within the rule [1],[2],[7] and parallelization of premises associated with the rule [13],[14],[16],[17],[18],[20]. Some other parallelization attempts are concerned with HCPRs[6].

Bharadwaj and Varshneya [6] have exploited the inherent hierarchical structure of the HCPRs, representing various levels of specificities in a conclusion to develop a parallel model which could be used to enhance its performance. The model has been simulated for shared memory architecture. The model presented does not incorporate the parallelization of censors.

In our discussion, we stress on time distribution to the HCPRs to get the conclusion within the time constraints. Many factors can be considered.

1. Firing all the independent HCPRs from different HCPR_trees at the same time.
2. All the specificities of any HCPR are mutually exclusive to each other, implying that only one of them can be true at any instance. This allows the specificities to be evaluated synchronously.
3. The HCPR premises and censors for the same HCPR can be checked simultaneously . This can happen within the given time for checking the HCPR.

As a conclusion of the above points, we might say that all independent HCPRs along with their censor conditions can be checked and fired at the same time but still each HCPR makes use only of its allocated time according to its level and its required certainty factor.

The above discussion would lead us to the following

a. Checking all the independent HCPRs from different HCPR-trees at the same time.
b. Assume that the maximum time needed to check any HCPR is t, this means any n independent HCPRs (from different HCPR-trees) would be checked within t units of time. Any way, the time and certainty factor allocated for every HCPR can be computed as explained in the previous algorithm.
c. Assume that a maximum of r stages of t units each are needed to get the system conclusion. This will always maintain

$$T \geq rt$$

d. This means, the end of t units indicates the end of a stage out of r stages. Each stage means firing all independent HCPRs within its time.

   We should satisfy the SCF for every HCPR_tree within its t units separately, thereby, the conclusion will satisfy the required SCF.

e. The following steps would be repeated until the inference is over
f. check all independent HCPRs that are in the matching list.
g. For every HCPR in the matching list, SCF is tried to be achieved within t. That is by trying to achieve the certainty of every HCPR according to formulae (5).
h. If t is over and the required certainty of some HCPRs could be achieved, then ignore these HCPRs and thereby no more inference will occur in the HCPR_trees where those HCPRs are related.
i. If the certainty factor of all HCPRs fired at the same stage achieved before the end of t, then the new t would be

$$\mathbf{new(t)= tlt/(r-i),}$$

where **i** is the current stage number and **tlt (time left)** is the remaining time of T which can be expressed

$$\mathbf{tlt=T-tct,}$$

where **tct** is the time consumed out of T till the moment t.

4 Conclusion

An algorithm that combines a tradeoff between time constraints, the required conclusion specificity and the system required certainty factor for variable precision logic systems is presented. The algorithm attempts to satisfy the required certainty factor of the system along with the possibility of achieving the level of the required specificity as time permits, otherwise the algorithm sends the results with the achieved certainty and specificity (type of the conclusion is based on the level) that could be achieved . The algorithm gives more opportunity to higher levels to be more certain than the lower levels by allocating more time for checking the higher level HCPRs than the lower level HCPRs. This procedure coup with the theory of VPL. If time permits, the system would respond with a certainty higher than the required system certainty factor. A general idea towards a modified version of the algorithm to be suitable for parallel HCPRs system is also provided. The main principle is based on executing several HCPRs at the same time, various kinds of parallelizations can be considered. Some of the interesting future research directions would be (1) Implementation of the algorithm and checking its behavior with various applications (2) developing and improving the idea given for parallelization to give a complete algorithm that can be suitable with any parallelism source (strategy of the parallelism used) and coup with HCPR structure (4) What are the necessary changes that could be done on the algorithm if multiprocessors are used.

References

1. Acharya, A and Tame, M 'Production systems on message passing computers: Simulation results and analysis ' Proc. Int. Conf. On Parallel Processing, 1989.
2. Bahar, E, Barachini, F, Doppelbauer, J, Grabner, H, Kasparec, F, Mandle, T and Mistelberger, H 'A parallel production system archi-tecture ' J.Parallel and Distributed computing Vol. 13,pp 456-462, 1991.
3. Bharadwaj, K K,Hewahi, N M, and Brando, M A, 'Adaptive hierarchical censored production rule-based system : a genetic algorithm approach' in S.L. Borges and C.A.A.Kaestner (Eds.), Lecture notes in artificial intelligence, Vol.1159, Advances in artificial intelligence, the Brazilian symposium on artificial intelligence SBIA'96, pp 81-90, Spriger-Verlag, 1996.
4. Bharadwaj, K K and Jain, N K, 'Hierarchical censored production rules (HCPRs) system', Data and Knowledge Engineering, North-Holland Vol.8,pp 19-34,1992.
5. Bharadwaj, K K, Neeraj and Goel, G C, 'Hierarchical censored production rules system employing Dempster-shafer uncertainty calculus', Information and Software Technology, Vol. 3, pp 155-164, 1994.
6. Bharadwaj, K K, and Varshneya, R, 'Parallelization of hierarchical censored production rules', Information and Software Technology, 37(8), pp 453-460,1995.
7. Forgy, C L,'RETE: a fast algorithm for the many patterm/many objects match problem', Artificial Intelligence, Vol. 19, no.1, pp 17-37, September, 1982.
8. Haddaway, 'P implementation and experiments with a variable precision logic inference system', Intelligent System Group, Department of Computer Science, University of Illnois.
9. Hewahi, N M , ' Genetic algorithms approach for adaptive hierarchical censored production rule based system', Ph.D. Thesis, School of Computer and Systems Sciences, Jawaherlal Nehru University, New Delhi,India, February 1994.
10. Hewahi, N M, 'Real time variable certainty systems', 4th congress on Expert Systems, Mexico, 4-8 March, 1998.
11. Hewahi, N M and Bharadwaj, K K, 'Bucket brigade algorithm for hierarchical censored production rule-based system', Inter. Journal of Intelligent Systems, Vol. 11, pp 197-225,1996.
12. Jain N K,' Variable Precision Logic: a Hierarchical censored production rules system', M.Tech. Dissertation, School of Computer and Systems Sciences, Jawaherlal Nehru University, New Delhi, India, January 1989.
13. Kuo, S and Moldovan, D,' Implementation of multiple rule firing production systemson hypercube', Proc. 9th Nat. Conf. On AI, AAAI,pp 304-309, 1991.
14. Laster, B P, 'The art of parallel programming', Printice-Hall, 1993.
15. Michalski, R S and Winston, P H, 'Variable precision logic', Artificial Intelligence, Vol. 29, pp 121-145, 1986.
16. Neiman, D E, 'Control issues in parallel rule-firing production systems', Proc. 9th Nat. Conf. On AI, AAAI, pp 310-316, 1991.
17. Oflazer, k,' Highly parallel execution of production systems: a model, algorithms and architecture ', New Generation Computing, Vol. 10, pp 287-313,1992.
18. Tout, K R and Evans, D J, ' A parallel expert system using a backward chaining strategy', Lecture Notes in Computer Science 634, pp 713-718, September 1992.
19. Willson, H, Kalp, D, Newell, a et al., ' The effectiveness of task-level parallelism for production systems', J. Parallel and Distributed Computing, Vol. 13, pp 395-411, 1991.
20. Xu, J and Hwang, K ,' Mapping rule-based systems on multicomputers using simulated annealing', J. Parallel and Distributed Computing, Vol. 13, pp 442-455, 1991.

Strong Conditional Logic

Cláudia Nalon and Jacques Wainer

Institute of Computing
State University of Campinas
Caixa Postal 6176 - CEP: 13.083-970 - Campinas - SP - Brazil
Fax: +55 19 289 3115 r. 251
{nalon, wainer}@dcc.unicamp.br

Abstract. This paper presents the semantic of SCL, a strong conditional logic that has the desirable properties of standard conditional logics but also deals with modes of reasoning that are hard for these logics like inheritance and irrelevance. The approach taken here produces fewer models (and then best results), even when considering strenghted conditional logics.

1 Introduction

The properties that a non-monotonic logic should exhibit are usually described in two different ways. A set of general properties of the entailment relation of non-monotonic logic, which are almost consensually assumed to be desirable properties, are described in [8]. [8] defines a nonmonotonic entailment relation, and thus a nonmonotonic logic, as cumulative, preferential, or rational according to an increasing set of properties that the entailment relation should satisfy.

The other set of properties is described as examples of desirable inferences. These examples, or better, modes of reasoning, are specificity, inheritance and irrelevance. In order to describe these modes of reasoning, let us suppose the following knowledge base:

1. Mammals usually live on land
2. Mammals usually give birth to live infants
3. Whales are mammals
4. Whales usually do not live on land

Specificity states that more specific information should override more general information. Thus given 1, 3, and 4 above, and given that something is a whale, we should conclude that it does not live on land. **Inheritance** states that (default) properties of a superclass should be inherited by the subclass, if there is no reason to believe the contrary. Thus from 1, 2, and 3 above, and given that something is a whale we should conclude that it gives birth to live infants. Finally, **irrelevance** states that an irrelevant property (or proposition) should not affect the conclusions that would be obtained if that information was not present. Thus from 1 above, and given that something is a brown mammal, we should conclude that it lives on land.

F. Moreira de Oliveira (Ed.): SBIA'98, LNAI 1515, pp. 209–218, 1998.

To this list of properties we would like to include another generic mode of reasoning, which for the lack of a better name, we will call ambiguity. **Ambiguity** states that if the knowledge base "licenses" the inference of both a formula and its negation and no specificity aspects apply to those two possible inferences, then nothing can be concluded about the formula. The term "licenses" is not defined, but it can be understood through an example. If we add to our knowledge base above:

5 Flying animals usually do not give birth to live infants (they lay eggs)
6 Bats are mammals and they fly
7 Mammals are animals

Thus given 2, 5, 6, and 7 then we should not conclude anything about how bats give birth to their infants. There is a line of reasoning, or argumentation for bats giving birth to live infants since they are mammals and mammals give birth to live infants. On the other hand, there is also a line of reasoning or argumentation for bats not giving birth to live infants since they are also flying animals and flying animals usually lay eggs. Thus no conclusion about the bat's birth method should be inferred.

Delgrande [5] characterized nonmonotonic systems into two groups. A **strong system** is one where some undesirable inferences may be obtained. Classic nonmonotonic systems such as default logics [15], circumscription [11], autoepistemic logics [12], and non-monotonic modal logics [10] fall into this group. These systems usually do not treat specificity within the logic itself, some of them do not deal with ambiguity, and some of them do not satisfy even the weaker set of desirable properties in [8]. A **weak system**, on the other hand, is one which satisfy [8] general properties on the entailment relation (either cumulativity or preference or rationality) but leave out some of the other desirable properties. Conditional logics such as N [3] and CT4 [1], among others, are systems that fall into this group. Weak systems usually deal with specificity and ambiguity within the logic itself, but do not deal correctly with irrelevance and inheritance.

Two lines of research have been pursued by nonmonotonic logic researchers: weakening strong systems (for example [2]) or strengthening weak systems (for example [1], [4], [5], and [7]). The logic developed herein, called **Strong Conditional Logic (SCL)**, follows the second approach.

The definition of entailment from a set of formula $\mathcal{K}B$ in a conditional logic (or any standard logic) involves all models that satisfy the set of formulas $\mathcal{K}B$. That is, γ will be an entailment of $\mathcal{K}B$, written $\mathcal{K}B \models \gamma$, if all models that satisfy $\mathcal{K}B$ also satisfy γ. Or to put in a different perspective, we will say that γ is a conclusion from $\mathcal{K}B$ if all models in $X = \{M | M \models \mathcal{K}B\}$ satisfy γ. But because it is a weak system, the entailment of conditional logics can not capture all forms of desirable non-monotonic reasoning. In order to strengthen the entailment relation one needs to make the set X smaller. The smaller the set X, more formulas will be satisfied in all models of X and thus more conclusions could be derived from $\mathcal{K}B$. In the limit, if the set X is empty, then all formulas are satisfied in the models of X and all conclusions can be derived (which is certainly too strong).

The essence of our approach is to define a way of construction the models that will belong to the set X. Since these constructed models will be fewer than the models that satisfy (in a traditional way) the formulas $\mathcal{K}B$, the resulting logic will be stronger than standard conditional logics.

2 Strong Conditional Logic

2.1 Language

SCL system is based on a propositional logic extended with a conditional binary operator ($\rightarrow$). We will denote the set of propositional symbols of the language as $\mathcal{P}$.

In our approach we make the distinction between the set of rules (conditional and strict), which we call the knowledge base ($\mathcal{K}B$), and the set of facts, which we call evidences ($\mathcal{E}$). The knowledge base is further separated into the strict rules (that do not contain the conditional operator) $\mathcal{S}$ and the conditional rules $\mathcal{C}$. Conditional rules are formulas which contain only one conditional operator and it is the formula's main connective. That is, we do not allow nesting of conditional operators.

To the point that it is possible to say that there is a standard conditional logic semantics, our constructed models will be somewhat similar to standard conditional logic models: a model is a set of possible worlds and an accessibility relation among these worlds denoted by $<$. The accessibility relation is to be interpreted as less exceptional, that is $x < y$ if x is less exceptional (more normal) then y. The relation $<$ is transitive and antisymmetric, but some other constraints could be imposed to it.

A SCL-model will be a pair $\langle W, < \rangle$, where W is the set of worlds that satisfy the strict rules, and $<$ is a preference relation among *some* of the worlds in W.

Strict rules are assumed to be necessary and thus only worlds that satisfy all the strict rules will be used to construct the models. Thus if $\mathcal{P}$ is the set of propositional symbols in the language, then we define the set of possible worlds W as $W = \{w | w \in 2^{\mathcal{P}} \text{ and } w \models_{PL} \mathcal{S}\}$, where $\models_{PL}$ is propositional satisfability.

The $<$ relation should have the minimal properties for a preference relation, that is, it is transitive and antisymmetric (and therefore non-reflexive). An important aspect of the model is that there may be *isolated worlds*, that is worlds that are not comparable (by the $<$ relation) to any other world in W. Since all models have the same set of worlds, each model can be identified with the preference relation it defines, thus from now on, when we mention a model we will be referring to the preference relation of that model.

The construction of the SCL-models can be seen as an incremental specification of $<$ relation for each model, but at the intermediary steps, the partially specified relation may not satisfy transitivity or antisymmetry. These intermediary specifications of the preference relation can be represented in two ways, as a set of restrictions and as a set of pairs of worlds.

Definition 1. *A* **restriction** *is relation between a world and a set of worlds, denoted by* $x \dot{<} \{y_1, \ldots, y_m\}$. *It stands for* $x < y_j$, *for all* $j = 1, \ldots, m$.

We will say that a pair of worlds $\langle a, b\rangle$ belongs to a restriction $r = x\dot{<}\{y_1, \ldots y_m\}$ if, and only if, $a = x$ and $b \in \{y_1, \ldots y_m\}$. Similarly we will say that a pair of worlds $\langle a, b\rangle$ belongs to a set of restrictions $R = \{r_1, r_2, \ldots, r_n\}$ if, and only if, $\langle a, b\rangle$ belongs to one of the restrictions $r_i \in R$.

2.2 Constructing the Pre-Models

The standard definition for a conditional rule such as $\alpha \to \beta$ to be satisfied in a model is when either $\neg\alpha$ is true in all worlds of the model, or when there is a world in which $\alpha \wedge \beta$ is true and in all accessible, more normal worlds, $\alpha \supset \beta$ is true. For our purposes, it is more interesting to look of what the truth of a conditional $\alpha \to \beta$ entails for the less normal worlds: there exists at least one $\alpha \wedge \beta$-world which precedes each $\alpha \wedge \neg\beta$-world.[1]

Definition 2. *The* **set of restrictions defined by a conditional rule** $\alpha \to \beta$ *is* $R_{[\alpha\to\beta]} = \{\omega_1\dot{<}\Psi, \omega_2\dot{<}\Psi, \ldots \omega_m\dot{<}\Psi)$ *where* $\omega_1, \omega_2, \ldots, \omega_m$ *are all the worlds in* W *that satisfy* $\alpha \wedge \beta$ *and* $\Psi = \{\upsilon \mid \upsilon \in W, \upsilon \models_{PL} \alpha \wedge \neg\beta\}$.

As we will see later, in a SCL-model one can have more than an $\alpha\wedge\beta$-world to be less exceptional than all $\alpha \wedge \neg\beta$-worlds. However, the definition of a restriction establishes that *exactly one world* satisfying $\alpha\wedge\beta$ is more normal than *all worlds* satisfying $\alpha\wedge\neg\beta$, which is a strengthening of the definition of the conditional in standard conditional logics.

Informally, the first step in constructing SCL-models for the knowledge base is to pick one restriction from each set of restrictions defined by each conditional rule in C, and collect them into a set of restrictions which we call a pre-model. Formally we define a pre-model as:

Definition 3. *Given* $\mathcal{C} = \{c_1, \ldots, c_n\}$*, and for each* c_i *there is a set of restrictions* $R_i = \{r_{i1}, r_{i2} \ldots r_{im_i}\}$ *defined by* c_i*. A* **pre-model** *is a set of restrictions* $PM = \{r_{1x_1}, r_{2x_2}, \ldots, r_{nx_n}\}$ *where each of the* $r_{ix_i} \in R_i$.

For instance, suppose $\mathcal{C} = \{c_1, c_2\}$, where $R_1 = \{a\dot{<}\{b, c\}\}$ and $R_2 = \{d\dot{<}\{f, g\}, e\dot{<}\{f, g\}\}$ are the sets of restrictions defined by c_1 and c_2, respectively. Then, we will have two pre-models: $PM_1 = \{a\dot{<}\{b, c\}, d\dot{<}\{f, g\}\}$ and $PM_2 = \{a\dot{<}\{b, c\}, e\dot{<}\{f, g\}\}$.

From now on, it is convenient to start working with sets of pairs of worlds instead of sets of restrictions. We will accomplish this change of point of view when defining extended pre-models.

Definition 4. *The* **extended pre-model** PM_i^e *of a pre-model* PM_i *is the transitive closure of the set* $\{\langle x, y\rangle | \langle x, y\rangle$ *belongs to* $PM_i\}$.

Definition 5. *An extended pre-model* PM_i^e *is* **inconsistent** *iff there exists a world* x *such that* $\langle x, x\rangle \in PM_i^e$*. It is* **consistent** *otherwise.*

[1] We do not consider the case in which α is false in all worlds of the model. However, in this case there is no $\alpha\wedge\neg\beta$-world, and a conditional $\alpha \to \beta$ is satisfied vacuously.

2.3 Constructing SCL-Models

Let $\mathcal{PM} = \{PM_1^e, PM_2^e, \ldots, PM_m^e\}$ be the set of all consistent extended pre-models of the theory $\mathcal{KB}$. The set of non-contested pairs (NCP) is defined as

Definition 6. *The* **set of non-contested pairs (NCP)** *is defined as* $NCP = \{\langle x, y\rangle |$ *there is some* $PM_i^e \in \mathcal{PM}$ *such that* $\langle x, y\rangle \in PM_i^e$ *and there is no* $PM_j^e \in \mathcal{PM}$ *such that* $\langle y, x\rangle \in PM_j^e\}$.

Thus, NCP are pairs of worlds that are demanded by some consistent extended pre-model, and that order is not contested by any of the other consistent pre-models. The intuition is that the NCP correspond to a global information that can be added to each extended pre-model.

Definition 7. *The* **SCL-model** M_i *(or better the preference relation of a SCL-model) corresponding to an consistent extended pre-model* PM_i^e *is the transitive closure of* $PM_i^e \cup NCP$*, provided that the resulting transitive closure does not contain a pair* $\langle x, x\rangle$ *for some* $x \in W$.

At this step, usually, the number of models is smaller than the number of consistent extended pre-models, since it could occur that $PM_i^e \cup NCP = PM_j^e \cup NCP$, for $i \neq j$.

Finally the definition of entailment:

Definition 8. *Let* α *and* β *be propositional formulas. The knowledge base* $\mathcal{KB}$ *and the evidence* α *entails* β*, written* $\mathcal{KB}, \alpha \models \beta$*, iff for all* $M_i = \langle W, <_i\rangle$ *SCL-models of the knowledge base, for all worlds* $\omega \in W$*, if* $\omega \models_{PL} \alpha \wedge \neg\beta$ *there is a world* $\upsilon \in W$ *such that* $\upsilon <_i \omega$ *and* $\upsilon \models_{PL} \alpha \wedge \beta$.

2.4 An Example

As an example, let us see how irrelevance, a common problem for conditional logics, is treated in SCL.

Example 1. Given the knowledge base $\mathcal{KB} = \langle \mathcal{C}, \mathcal{S}\rangle$, and $\mathcal{C} = \{A \to B\}, \mathcal{S} = \{\}$, we want to conclude, among other things, that B can be inferred from the the evidence $A \wedge C$. The set of propositional symbols of the language of this theory is $\{A, B, C\}$. The possible worlds are:

ω	0	1	2	3	4	5	6	7
A	0	0	0	0	1	1	1	1
B	0	0	1	1	0	0	1	1
C	0	1	0	1	0	1	0	1

The set of strict rules is empty, therefore, $W = \{\omega_0, \ldots, \omega_7\}$. The set of restrictions defined by $A \to B$ is $R_{[A\to B]} = \{\omega_6 \dot{<} \{\omega_4, \omega_5\}, \omega_7 \dot{<} \{\omega_4, \omega_5\}\}$.

Each restriction in $R_{[A \to B]}$ defines a pre-model. Then, the two extended consistent pre-models are $PM_1^e = \{\langle \omega_6, \omega_4 \rangle, \langle \omega_6, \omega_5 \rangle\}$ and $PM_2^e = \{\langle \omega_7, \omega_4 \rangle, \langle \omega_7, \omega_5 \rangle\}$.

The non-contested pairs are the union of both pre-models, that is $NCP = \{\langle \omega_6, \omega_4 \rangle, \langle \omega_6, \omega_5 \rangle, \langle \omega_7, \omega_4 \rangle, \langle \omega_7, \omega_5 \rangle\}$, and thus there is a single SCL-model, which is the NCP itself. It can be represented graphically as the figure below, where the most normal worlds are to the right, and worlds in the same group can be seen as "equally normal" worlds in the sense that they all have the same set of better and worse (in the $<$ sense) worlds. The group consisting of worlds $\omega_0, \omega_1, \omega_2$, and ω_3 are isolated worlds, neither more nor less exceptional than any other world.

$$M = \begin{pmatrix} \omega_4 \\ \omega_5 \end{pmatrix} \longrightarrow \begin{pmatrix} \omega_6 \\ \omega_7 \end{pmatrix}$$
$$\begin{pmatrix} \omega_0 \ \omega_1 \\ \omega_2 \ \omega_3 \end{pmatrix}$$

From this model we will have (1) $\mathcal{K}B, A \models B$; (2) $\mathcal{K}B, A \wedge \neg B \not\models B$; (3) $\mathcal{K}B, A \wedge C \models B$ (4) $\mathcal{K}B, A \wedge \neg C \models B$ and (5) $\mathcal{K}B, C \not\models B$.

In each case, the proof of the entailment rests on showing that for all worlds in which the evidence is true and the conclusion false, there is a more normal world in which both are true. In (1), ω_4 and ω_5 satisfy the evidence and falsify the conclusion, but are more exceptional then ω_6 which satisfy both the evidence and the conclusion. In (2), the same worlds ω_4 and ω_5 satisfy the evidence and falsifies the conclusion, but there is no more normal world (in fact there is no world at all) that satisfies both the evidence and the conclusion. In (3), a typical example of irrelevance, the world ω_5 is less normal than the worlds ω_7. In (4) case, the world ω_4 is less normal than the world ω_6. And finally in (5), the isolated worlds play their role: the worlds that satisfy the evidence and falsify the conclusion are ω_1 and ω_5, but they are not all more exceptional that the worlds that satisfy both evidence and conclusion, because ω_1 is not more or less exceptional than any other world. [2]

3 Further Examples

Besides the properties of specificity, inheritance, irrelevance, and ambiguity, there are other desirable examples of inferences that are interesting and that can be performed in SCL. We list some of them, in the following format: $\langle$ conditional rules, strict rules$\rangle$, evidence and conclusion.

The two examples below deal with basic nonmonotonicity properties:

1. $\langle \{A \to B\}, \{\} \rangle, A \models B$

[2] In fact, from C, we can conclude nothing at all, but C (i.e., $\mathcal{K}B, C \not\models A$, $\mathcal{K}B, C \not\models \neg A$, $\mathcal{K}B, C \not\models B$ and $\mathcal{K}B, C \not\models \neg B$).

2. $\langle\{A \to B\}, \{\}\rangle, A \wedge \neg B \not\models B$

The next two examples deal with ambiguity:

3. $\langle\{A \to B, C \to \neg B\}, \{\}\rangle, A \wedge C \not\models B$
4. $\langle\{A \to B, C \to \neg B, D \to B\}, \{\}\rangle, A \wedge C \not\models B$

The standard example of specificity:

5. $\langle\{A \to C, B \to \neg C\}, \{B \supset A\}\rangle, B \models \neg C$

The next two examples deal with irrelevance. The first is irrelevance of evidence, the second irrelevance of conditional rules.

6. $\langle\{A \to B\}, \{\}\rangle, A \wedge C \models B$
7. $\langle\{A \to B, C \to D\}, \{\}\rangle, A \wedge E \models B$

Inheritance can be seen as a form of transitivity. There are three cases of transitivity to consider: conditional to material, conditional to conditional and material to conditional (the standard example of inheritance).

8. $\langle\{A \to B\}, \{B \supset C\}\rangle, A \models C$
9. $\langle\{A \to B, B \to C\}, \{\}\rangle, A \models C$
10. $\langle\{B \to C\}, \{A \supset B\}\rangle, A \models C$

Furthermore in SCL, transitivity is carried as far as possible, that is, if one of the links in the transitivity chain would lead to inconsistency, only this inference is not performed.

11. $\langle\{A \to B, B \to C, C \to \neg A\}, \{\}\rangle, A \models C$

Finally, SCL also correctly deals with combinations of the examples above, for example, the combination of inheritance (10) and specificity (5).

12. $\langle\{A \to C, B \to \neg C, B \to D\}, \{A \supset B\}\rangle, A \models C \wedge D$

4 Theorems

Besides the example of reasoning in the previous section, we are able to prove some generic theorems about the logic SCL. The proofs of the theorems are in an accompanying technical report by the authors [13]. In the following, α, β and γ are propositional formulas; $L(X)$, where X is a propositional formulas set, is the language of X; $L(\alpha)$ is the language of α.

The first theorem is related to ability of inferencing new conditional rules from the old ones in SCL. This is not possible in traditional nonmonotonic logics (as in [15]), where semantics is based on extensions. The ability to infer conditionals is an important part of weak systems since it allows one to define when two knowledge bases are equivalent, when a conditional rule is superfluous to a knowledge base, and so on.

Theorem 1. *Given a theory* $T = \langle \mathcal{K}B, \{\alpha\}\rangle$*, if* $\mathcal{K}B, \alpha \models \beta$*, then* $\mathcal{K}B \models \alpha \rightarrow \beta$*.*

Theorem 1 does not assure that the set of models will not be modified by addition of a new rule. When a new conditional rule is added to a theory, the set of SCL-models will be probably different from the original one. However, new conditional rules can be obtained from semantics structure.

The next theorem relates SCL set of properties for nonmonotonic systems.

Theorem 2. *SCL is preferential.*

It was not proved yet if SCL is a rational system or not. [8], [6] and [9] argue that rational systems, better than a preferential one, can capture a set of desirable nonmonotonic inferences. This is not a consensus among researchers (see [7], for example).

Preferential systems can capture interesting modes of reasoning (e.g., specificity and ambiguity), althought they can not treat irrelevance and inheritance. But, we can prove a generic result about irrelevance

The next theorem is a generalization of the property of a weak form of irrelevance. We prove that one can add an irrelevant evidence without changing the original conclusion. For this theorem, irrelevance means a formula that uses a propositional symbols, which are not in original language.

Theorem 3. *Let* $T = \langle \mathcal{K}B, E\rangle$ *be a theory, where* $E = \{\alpha\}$*.* L_T *is the language of* T*; and* W*, the set of worlds that satisfy the strict rules in* T*. If* $\mathcal{K}B, \alpha \models \beta$*, then* $\mathcal{K}B, \alpha \wedge \gamma \models \beta$*, where* γ *is a propositional formula (or its negation) and if* $\alpha \in L(\gamma)$ *then* $\alpha \notin L_T$*.*

It was not proved that SCL treats a general form of inheritance. However, several examples were tested and results were correct in all cases. For instance:

1. From $\mathcal{K}B = \{\mathrm{M} \rightarrow \mathrm{T}, \mathrm{M} \rightarrow \neg\, \mathrm{O}, \mathrm{B} \rightarrow \mathrm{M}, \mathrm{B} \rightarrow \neg\, \mathrm{T}\}$, we can obtain $\mathrm{M} \models \mathrm{T}$, $\mathrm{M} \wedge \mathrm{B} \models \neg\, \mathrm{T}$, and $\mathrm{B} \models \neg\, \mathrm{O}$.
2. From $\mathcal{K}B = \{\mathrm{E} \rightarrow \mathrm{A}, \mathrm{A} \rightarrow \mathrm{T}, \mathrm{E} \rightarrow \neg\, \mathrm{T}\}$, we can obtain $\mathrm{E} \models \mathrm{A}$, and $\mathrm{E} \wedge \mathrm{A} \models \neg\, \mathrm{T}$.
3. From $\mathcal{K}B = \{\mathrm{P} \supset \mathrm{B}, \mathrm{B} \rightarrow \mathrm{F}, \mathrm{B} \rightarrow \mathrm{W}, \mathrm{P} \rightarrow \neg\, \mathrm{F}\}$, we can obtain $\mathrm{P} \models \neg\, \mathrm{F}$, and $\mathrm{P} \models \mathrm{W}$.

5 Related Work

As mentioned before, SCL follows in the tradition of strengthening a weak system. Some of the other logics developed within this approach were the strengthening of the logic N [4], the logic CO* [1], the conditional entailment [7] and the preference-based approach in [5]. Let us discuss just the last one.

The idea presented in [5] is based on the notion of preferential model structures [16]. The approach expresses preference in terms of orderings on worlds. Let L be a propositional language augmented with a binary operator $\rightarrow$. Sentences of L are interpreted in terms of a model $M = \langle W, E, P\rangle$, where W is a set (of

worlds), E is a reflexive, transitive, forward-connected[3] accessibility relation on worlds; and P is a mapping of atomic sentences and worlds onto $\{0,1\}$. The approach deals with the full set of mappings of the (finite) set of atomic sentences $\mathbf{P}$ onto $\{0,1\}$, thus we have $W = \{f \mid f : \mathbf{P} \rightarrow \{0,1\}\}$

Starting from this weak system, Delgrande defines specificity ordering on formulas, given a default theory T and $A, B \in L$ as: $A \prec_T B$ iff $T \models A \vee B \rightarrow \neg B$ and $T \models \neg\Box\neg A$. And $A \preceq_T B$ if it is not the case that $B \prec A$.

Defaults in T provides a basic preference notion on worlds. A conditional $A \rightarrow B$ prefers ω_1 to ω_2 if its material counterpart is true in ω_1 and is false in ω_2. The set of rules that prefer ω_1 to ω_2 is $Pref(\omega_1, \omega_2) = \{A \rightarrow B \in T \mid A \rightarrow B$ prefers ω_1 to $\omega_2\}$

Given a full specificity ordering[4] , Delgrande defines a *preference ordering* $P = \langle W, < \rangle$: For ω_1 and $\omega_2 \in W$, we have $\omega_1 < \omega_2$ iff

1. $Pref(\omega_1, \omega_2) \neq \emptyset$
2. for every $C \rightarrow D \in Pref(\omega_2, \omega_1)$ there is some $A \rightarrow B \in Pref(\omega_1, \omega_2)$ such that $C \preceq A$ and it is not the case that $A \preceq C$.

Thus, one can conclude B as a *preferential default inference* from A in theory T, if for every preference ordering, for every ω_2 where $\omega_2 \models A \wedge \neg B$ there is a ω_1 where $\omega_1 \models A \wedge B$, and $\omega_1 < \omega_2$.

This approach handles specificity, inheritance and irrelevance, but does not deal adequately with ambiguity. If there are two conditionals and one can not state a specificity relation between their antecedents, then the ordering one obtains cannot express the ambiguity of the knowledge base. In particular it does not deal correctly with example 4 section 3. The approach also does not deal with example 11.

6 Conclusions

This paper has presented the semantics of SCL, a conditional logic that besides retaining the good properties of weak systems also correctly deals with types of inferences that are usually associated with strong systems, such as inheritance and irrelevance. SCL differs from other recent approaches to strengthening conditional logics in many aspects:

- These approaches have an underlying conditional logic. For example [5] uses an underlying conditional logic to define the notion of specificity between two formulas. [1] uses the underlying logic CT4 to define irrelevance. SCL does not have an underlying conditional logic used to define specificity or irrelevance.
- More importantly, SCL deals with examples that are problematic for the other approaches. In particular the approach taken in [5], which is in the same spirit of our approach, fails to deal with some forms of ambiguity (example 4 in section 3) and transitivity with blocking (example 11 in section 3).

[3] If $E\omega_1\omega_2$ and $E\omega_1\omega_3$ then $E\omega_2\omega_3$ or $E\omega_3\omega_2$

[4] If $B_1 \vee B_2 \prec A$ and is not the case that $B_1 \prec A$ then $B_2 \prec A$.

In parallel with the theoretical work presented here, we have implemented a SCL theorem prover for small knowledge bases based on model-checking. The theorem prover has two components. The **model constructor** performs the costly computation of creating the SCL models, but runs off-line, that is only once for each knowledge base. The second component, the **model checker**, runs on-line and answers the queries by performing the much faster computation of reading in the models stored by the off-line component and performing a model checking for the evidence and conclusions.

References

1. Boutilier, C. *Conditional Logics of Normality: A Modal Approach.* Artificial Intelligence, vol. 68, pp. 87-154, 1994.
2. Brewka, G. *Cumulative Default Logic: In Defense of Nonmonotonic Inference Rules.* Artificial Intelligence, vol. 50 (2), pp.183-205, 1991.
3. Delgrande, James P. *A First-Order Conditional Logical for Prototypical Properties.* Artificial Intelligence, vol. 33 (1), pp. 105-130, 1987.
4. Delgrande, J.P. *An Approach to Default Reasoning Based on a First-Order Conditional Logic: Revised Report.* Artificial Intelligence, vol. 36 (1), pp. 63-90, 1988.
5. Delgrande, James P. *A preference Approach to Default Reasoning: Preliminary Report*, American Association for Artificial Intelligence Conference, Seattle, WA, July, 1994.
6. Freund, M., Lehmann,D. Morris, P. *Rationality, Transitivity and Contraposition.* Artificial Intelligence, vol. 52, pp.191-203, 1991.
7. Geffner,H., Pearl, J. *Conditional Entailment: Bridging Two Approaches To Default Reasoning.* Artificial Intelligence, vol. 53, pp. 209-244, 1991.
8. Kraus, S., Lehmann,D., Magidor, M. *Nonmonotonic Reasoning, Preferential Models and Cumulative Logics.* Artificial Intelligence, vol. 44, pp.167-207, 1990.
9. Lehmann, D., Magidor,M. *What Does a Conditional Knowledge Base Entail?* Artificial Intelligence, vol. 55 (1), pp. 1-60, 1992.
10. Marek, W., Truszczynski, M. *Autoepistemic Logic.* Journal of the ACM, 38:588-619, 1991.
11. McCarthy, J. *Circumscription - A Form Of Nonmonotonic Reasoning.* Artificial Intelligence, vol. 13, pp. 27-39, 1980.
12. Moore, R. C. *Semantical Considerations on Nonmonotonic Logic.* Artificial Intelligence, vol. 25 (1), pp. 75-94, 1985.
13. Nalon, C. *Lógica Condicional Forte.* (Dissertação de Mestrado). Universidade Estadual de Campinas. São Paulo, 1997.
14. Pearl, J. *System Z: A natural ordering of defaults with tractable applications to nonmonotonic reasoning.* In Proceedings of the Third Conference on Theoretical Aspects of Reasoning About Knowledge, pp. 121-135, Pacific Grove, Ca., 1990.
15. Reiter, R. *A logic for default reasoning.* Artificial Intelligence, vol. 13 (1,2), pp. 81.132, 1980.
16. Shoham, Y. *Reasoning About Change: Time and Causation from the Standpoint of Artificial Intelligence.* MIT Press, 1988.

Computing Aesthetics

Penousal Machado[1], Amílcar Cardoso[2]

[1] Instituto Superior de Engenharia de Coimbra
[2] Departamento de Engenharia Informática da Universidade de Coimbra
{machado, amilcar}@dei.uc.pt

Abstract. Aesthetic judgment is one of the most valued human characteristics, being regarded as a remarkable display of intelligence. Popular knowledge tells us that it is purely cultural. In this paper, we present a brief theory according to which aesthetics depends on biological and cultural issues, namely on visual image processing. We present an implementation of this theory and the experimental results achieved. The tests were made with two purposes in mind, validating the theory and using it as a basis for the automatic generation and evaluation of images.

1 Introduction

The work presented in this paper, is part of an ongoing research project (NEvAr), whose goal is the development of computer programs capable of creating artworks [5]. The main field of study and application is the field of visual arts.

We can divide AI applications to the field of Arts in three categories: (1) "Intelligent" tools; (2) Systems performing "art understanding" tasks (e.g. musical analysis); (3) C*onstructed artists,* which "are supposed to be capable of creating aesthetically meritorious artworks on their own, with minimal human intervention" [10].

In our quest for the creation of constructed artist, we identified a set of features that such a system should exhibit [5]. One of these is the ability to make aesthetic judgements. Consequently, we tried to answer two basic questions, namely: "Can this be achieved?" and "How can it be achieved?".

The success of constructed artists is greater in music then in visual arts [4], this can be explained by the higher quantity of data required by image handling [4], but this explanation seems a little shallow. The fact is that music theory is more developed and quantitative than theory in visual arts [4]. In music we can "write down" a piece, in visual arts we have neither the alphabet nor the grammar to do so. In other words, we also lack a notation system. In music there is a clear distinction between composer and interpreter, in visual arts these two different roles are mixed.

To our knowledge, there was only one attempt to create a system (in the field of visual arts) with the ability to make aesthetic judgements [1]. In this system a Neural Network makes the evaluation of the images. In the field of music, the approach to the

F. Moreira de Oliveira (Ed.): SBIA'98, LNAI 1515, pp. 219-228, 1998.

problem of creating a constructed artist is radically different. The generation of new musical pieces is, usually, guided by knowledge (i.e. by a musical theory), and the evaluation of the pieces, when it exists, is also based on a musical theory. There was no attempt in [1] to support the system in an underlying aesthetic theory, we consider this to be a factor that hindered the success of the system.

In this paper, we will focus on aesthetics and on how to compute aesthetic value. The paper has the following structure: The second section pertains to the origins of art and aesthetic judgment: we give a short biological explanation to the devotion of humans to art and to how natural evolution favored the appearance of art. We also make a brief outline of an aesthetic theory that serves as basis, and is part of our ongoing research. In section three, we present an implementation of this theory. Some experimental results are referred in the fourth section and, finally, in section five, we draw some conclusions and talk about further work.

2 Art & Aesthetics

If we ask someone why he/she likes a certain painting, they will usually talk about the emotions or feelings triggered by the artwork, the combination of colors, or, even more frequently, we will get the intriguing answer: “I just like it.”. It is extremely difficult to find a consensual definition for what is Art.

Art and Aesthetics are different, yet highly related, fields. We can say, to a certain point, that Aesthetics is a subset of the Arts, and we can define it as the study of the form in itself, striped from its meaning.

From our point of view, the assessment of an artwork is influenced by two factors:

- The “content” of the artwork, which relates to what is represented by the artwork. If we consider Art as a form of communication, then “content” is what is communicated.
- The “visual aesthetic value” of the artwork, that is related to color combination, composition, shape, etc. We are talking about the form of the artwork, thus, how “content” is represented.

By assuming this point of view, we aren’t creating a false dichotomy between “form” and “content”. We are aware of the fact that these factors aren’t completely independent. Consequently, the value of an artwork rests on these factors and their interactions. It is possible to have an artwork that is visually pleasing, but whose content is displeasing. In fact, many art styles rely on the mixed feelings caused by this discrepancy (e.g. many of S. Dali’s paintings).

If we restrict ourselves to the field of Aesthetics, these factors gain independence because, as stated before, Aesthetics focuses in the “form”. In other words, an image can have a high aesthetic value, independently from its content, and even if it is deprived of content. We don’t mean that content isn’t important, we just mean it is not indispensable from the Aesthetics point of view.

It seems clear that the way in which content influences the value of a given artwork depends, mainly, on cultural issues. From our point of view, visual aesthetic value, is

connected to visual image perception and processing, thus being mainly: biological, hardwired, and therefore universal. We aren't saying that it is coded in our genes, we just mean that what we are, and the way our visual image perception system works, makes us favor certain images to others. The remainder of this section is dedicated to the support of the previous statement.

2.1 The Origins of Art

We will start by focusing on the origins of art and try to show how Natural Evolution could favor the appearance of art. Natural selection should favor the fittest individuals in a population, so why should a seemingly useless activity as art be favored by it?

To the vast majority of the animals, the struggle for survival takes all their time. Only in their infancy they have time to playing and games. The same happened to the primitive man. Only from a certain point in history man begun to have spare time.
The appearance of art may be explained by the necessity of using other forms of communication other than gesture or speech. Although this explanation is usually accepted, there are, other explanations that can be considered complementary. The coordination of hand movements had a great influence on human evolution. When a prehistoric man devoted himself to painting he was not doing a fruitless activity: by painting he was also training and improving his motor coordination. These two factors would give him an immediate survival advantage.

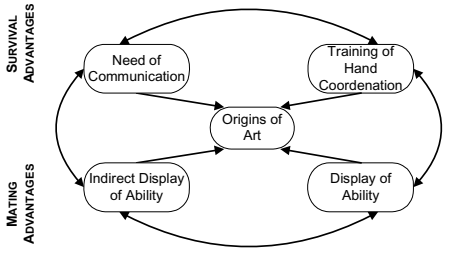


Fig. 1. Set of explanations that can justify the appearance of art.

We also have to consider the mating advantages since animals tend to choose the most fit individual they can for mating. By making an artwork, he is making a direct display of ability; furthermore he is showing that has time to spend on activities that aren't vital for his immediate survival, and thus making an indirect display of ability. In our opinion, this set of explanations gives reasonable justification for the devotion of man to art, but they don't justify why we find certain images beautiful, aesthetically pleasing or artistic.

The analysis of children's drawings allows us to observe the development of aesthetic, showing that the earliest stages of development are common to all children and that there is a universal imagery shared across cultural boundaries. Kellogg [8] recognizes five basic stages in pictorial representation: Scribbles, Diagrams,

Combines, Aggregates, Pictorials. Usually, the first pictorial image is the human figure. This image is, always constructed in the same, rather puzzling way. It begins with an empty circle; in the next step bubbles are added to the inside of the circle; the bubbles are gradually transformed in eyes, mouth and nose; afterwards hair is included; Some of the hairs get longer, until they are transformed into arms and legs. Somewhere between the ages of 6 and 12, this universal imagery begins to disappear, probably, due to educational influence [8].

It seems safe to say that visual aesthetic judgment is not particular to humans. In fact, we share this ability with other species of animals. Experiments with chimpanzees show that they follow the same steps of development of human infants. The first stages of development are similar, however, Chimpanzees aren't able to go beyond the phase of the circle into the phase of the filled circle. They also never seem to be able to create a pictorial image. Nevertheless, their paintings show that the brain of a chimpanzee is capable of making simple aesthetic judgments. Morris [8] found six common principles between chimpanzee and human art: Self-Rewarding Activity, Compositional Control, Calligraphic Differentiation, Thematic Variation, Optimum Heterogeneity and Universal Imagery.

These factors lead us to conclude that the assessment of the visual aesthetic value of an image is directly connected to the visual image perception system. This conclusion is also supported by a wide variety of experiments. One instance of such is the well-known, perfect rectangle experiment. When asked to pick the most beautiful rectangle from a list, most people will choose the rectangle with the "Golden Proportions" 1:1.618. The explanation for this is related to the range of our visual field [8]. Another example would be the difference between horizontal/vertical and diagonal lines. The diagonal line has a higher visual intensity. The periphery of the eye is very sensitive to diagonals, so it calls for complete attention from the viewer. That is why traffic signs designed to warn of hazards are diamond shaped, using diagonals. These differences must be taken into consideration, when composing an artwork, so that the desired "balance" is achieved.

Let us consider how the visual image perception system works.

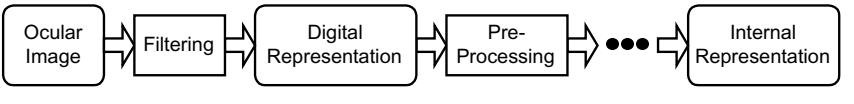


Fig. 2. Steps of visual image perception.

The process of transformation of the ocular image into its digital representation is well known. From this representation, the brain constructs internal representations, retaining only certain aspects of the image. The way in which this process works is still source of much debate. The idea that there is a "pre-processing" of the digital image (shape and contour detection, color analysis, depth and movement analysis, etc), and that "recognition" and subsequent transformation to internal representations is made based on the results of this "pre-processing", is usually accepted [11].

From the previous statement, we can say that there is a difference between image complexity, and internal representation complexity; furthermore a complex image isn't necessarily difficult to (pre)process. To clarify our previous statement, consider the following analogy: a fractal image is usually complex, and highly detailed; yet it can be compactly described by a simple mathematical formula. Therefore, we can say that there is a difference between image complexity, and internal representation complexity; furthermore a complex image isn't necessarily difficult to (pre)process.
In the book The Society of Mind, Minsky associates the concepts of fashion and style to the mental work necessary to process images: "…why do we tend to choose our furniture according to systematic styles or fashions? Because familiar styles make it easier for us to recognize and classify the things we see. If every object in a room were distracting by itself, our furniture might occupy our minds too much… It can save a lot of mental work…" [6].

If we accept this explanation, we are lead to conclude that simpler, easier to process images have higher aesthetic value than complex ones. Or, in other words, low processing complexity implies high aesthetic value. Following this idea, we come to the conclusion that a completely blank image has higher aesthetic value than any artwork, since it is certainly easier to process.[1]

Abraham Moles [7] sustains a quite different position, namely, that aesthetic value is directly connected image complexity, and bases his measure of image complexity on information's theory, e.g. image complexity is directly connected to the unpredictability of the "pixels" in the image.

Minsky's examples are related to office furniture. When we are in an office we don't want to be distracted, we want to work; when we are admiring an artwork, we want to be distracted, that's probably why we usually have, in our offices, a painting to look at when we want to distract ourselves.

In our opinion, aesthetic visual value depends on two factors: (1) Processing Complexity (the lower, the better); (2) Image Complexity (the higher, the better). This seems contradictory, but, as we said before, a complex image isn't necessarily difficult to process. Thus, images that are simultaneously visually complex and easy to process are the images that have higher aesthetic value. Our state of mind influences how we value these factors, if we are tired, we will probably give more weight to processing simplicity. Returning to the fractal example, fractal images are usually complex, however, the propriety of self-similarity makes these images easier to process, which gives an explanation to why we usually find fractal images beautiful. Another important characteristic of fractal images is that they have several levels of detail, characteristic that can also be found in many artworks, (e.g. Kandinsky's paintings). This strikes us as very important, especially if we notice that the act of "seeing" isn't instantaneous, it spans through a (sometimes-long) period of time. When we look briefly at such an artwork, we are automatically able to recognize its main shapes, if we give it more attention we will increasingly discover more detail. This makes the image easy to process, and thus less distracting when we don't want to

[1] This may not be as absurd as it seems considering paintings like "White on White" from Malevitch.

give it attention; simultaneously, when we want to give it attention, we will always find enough detail to "fill" our minds. If the image had only one level of detail, it would probably make it either difficult to process rapidly or with little complexity. This would hinder the generality of the artwork in the sense that our willingness to look at it would largely depend on our state of mind. Thus, it is important to preserve a high Image Complexity / Processing Complexity ratio through all the period of seeing.

3 Implementing the Theory

On the previous section, we have outlined an aesthetics theory. We stated that: the aesthetic value of an artwork is directly connected to Image Complexity (IC) inversely connected to Processing Complexity (PC).

In order to test our theory, we must devise estimates for IC and PC. Having these estimates, we could develop a formula to evaluate the aesthetic value, e.g.:

$$\frac{IC^{a}}{PC^{b}} \tag{1}$$

In which, *a* and *b*, can be used to change the importance given to each factor.

We have also stated that the IC/PC ratio should be maintained high trough all the period of seeing. By practical reasons, that will be explained later, we were forced to consider that IC is constant. Consequently, we must estimate this factor, based, solely, in the differences between PC at distinct moments in time. Our formula will then become:

$$\frac{IC^{a}}{(PC)^{b} * \left(\frac{PC(t_1) - PC(t_0)}{PC(t_1)} \right)^{c}} \tag{2}$$

Since we now have, $PC(t_0)$ and $PC(t_1)$, we will replace PC by the multiplication of $PC(t_0)$ and $PC(t_1)$, hoping that this will give a better estimate of Processing Complexity. Therefore we arrive to the following formula:

$$\frac{IC^{a}}{(PC(t_0) * PC(t_1))^{b} * \left(\frac{PC(t_1) - PC(t_0)}{PC(t_1)} \right)^{c}} \tag{3}$$

With *a*, *b* and *c* yielding the relative importance of each factor. Now, in order to test our theory we "only" have to devise estimates for IC and PC. This formula is only one instance of the ones that could be constructed based on the presented theory.

3.1 The Estimates

How can we estimate IC and PC? If we had an implementation of the visual image perception system, we could use this to estimate, at least, PC. Since we don't have this, we had to resort to other methods. One of the main steps of image perception involves transforming a digital image into some internal representation. During this transformation, some (probably lots) of detail is lost, and only certain aspects of the image are retained. This led us to establish a metaphor between this process and *lossy* image compression (e.g. *jpg* compression). In this type of compression method, the image is coded with some error, we can specify the amount of detail kept and thus, indirectly, the error resulting from the compression and the compression ratio.

Jpg compression works as follows[2]: Transformation of the image into frequencies, using the discrete cosine transform (DCT); Quantization of the resulting coefficient matrix; Lossless compression of the results (e.g. by Huffman coding). The process of quantization is the only step in which detail is lost. The process of quantization affects (mainly) the high frequencies, which can, usually, be discarded without significant loss in image quality. The amount, and quality (i.e. the error involved), of the compression achieved by this method, depends on the predictability of the pixels in the image. Considering a definition of image complexity similar to the one proposed in [7] we could use this type of compression to estimate IC. Therefore, our IC estimate will be:

$$\frac{RMS\_Error}{Compression\_Ratio} \quad (4)$$

resulting from the *jpg* compression of the image.

In the last years, several new compression methods have appeared, including fractal image compression and wavelets. Some of these methods can be compared favorably with *jpg*. Fractal image compression method has grabbed our attention, due to some of its characteristics, namely:

- Suppose that you compress an image with Fractal Image Compression and *jpg* compression. Suppose, also, that the *rms* errors resulting from this compression are equal. If you ask someone to choose the image with higher quality they will choose (almost always) the image resulting from fractal image compression [2].
- The images resulting from fractal image compression don't have a fixed size. They can be reproduced at sizes greater than the original, resulting in what has been called a "fractal zoom". The images resulting from fractal zoom have better quality than the ones produced by standard zoom techniques [2].
- The basic idea of fractal image compression is exploring the self-similarities present in an image.

This set of features led us to conclude that fractal image compression is, somehow, closer to the way in which humans process images than other types of compression (e.g. *jpg*). Accordingly, we will use fractal image compression for estimating PC.

---

[2] For a detailed description of the *jpg* standard see, e.g. [9].

As we have seen before, we need to take these estimates at different moments in time. The idea is that the amount of detail perceived by a person when looking at an image increases over time. The amount of detail kept can be specified in both compression methods. Therefore, we can vary this amount to simulate estimates for IC and PC at different points in time.

At this point, we encountered some implementation problems. We wanted to use high compression ratios, since we think that the visual image perception system must also perform "high compressions". Furthermore, we wanted to use similar compression ratios for our IC and PC estimates. Unfortunately, although fractal image compression works well at high compression ratios, *jpg* compression doesn't, and the images get too degraded. To deal with this problem, we decided to use only one IC estimate, i.e. we only vary PC over time. Fractal image compression is a slow process, so, we decided to measure it, only, in two points in time (t_0 and t_1). The amount of detail kept in PC(t_1) is significantly larger than in PC(t_0). We can consider, to some degree, that the image resulting from the compression in PC(t_1) can be produced as follows: divide the original image in four equal parts; enlarge each of the resulting blocks so it reaches the same size as the original image; perform fractal image compression in each of these blocks, using for each the same parameters that in PC(t_0). This can be seen as a person concentrating at each of the quadrants of the image.

In the IC estimate, we used a compression ratio close to the one used in PC(t1).

Before describing our experimental results, we want to stress some points to avoid misinterpretations. We don't intend to say that the visual image perception system works by fractal image encoding, or that image complexity is the IC estimate. We merely think that we can use the results of jpg and fractal image compression as rough estimates of IC and PC. In other words, when image complexity increases, jpg compression tends to perform worse; Fractal image compression takes advantage of some of the characteristics of the images (i.e. self-similarities). The visual image perception system also takes advantages of these (and possibly many others) characteristics. Thus, the results from fractal image compression are closer to the ones of visual image perception than the results of jpg compression.

4 Experimental results

The next logical step was to test our formula, for which we used the TDA test (Test of Drawing Appreciation) [3]. This is a psychological test that can be applied to individuals or groups. Its objective is to assess the aptitude to evaluate artistic structures. The test tries to estimate the level in which an individual recognizes and reacts to basic principles of aesthetic order: unity, dominance, variety, balance, continuity, rhythm, symmetry and proportion [3]. The test is composed by 90 items. Each item consists in a pair or triad of similar drawings in which one of the drawings corresponds to the fundamental principles while the other violate one or more of these principles. The task of the individual being evaluated is to chose, for each item, the drawing that she/he likes more. The average scores in this test are 45.680 (for

professionals of several activities) and 55.6897 (for Fine Arts graduates) correct answers [3]. Answering randomly to the test would result in an average of 43.47. Notice that the difference between this result and the ones achieved by a random population is very small, 2.21, which gives an idea of the toughness of this test.

Each of the images in the TDA test was scanned, and our program computed the IC, PC(t_0) and PC(t_1) estimates. Using these estimates, the program calculated the results of the previously stated formula. For each test item, the program "chose", as answer, the image yielding the higher result.

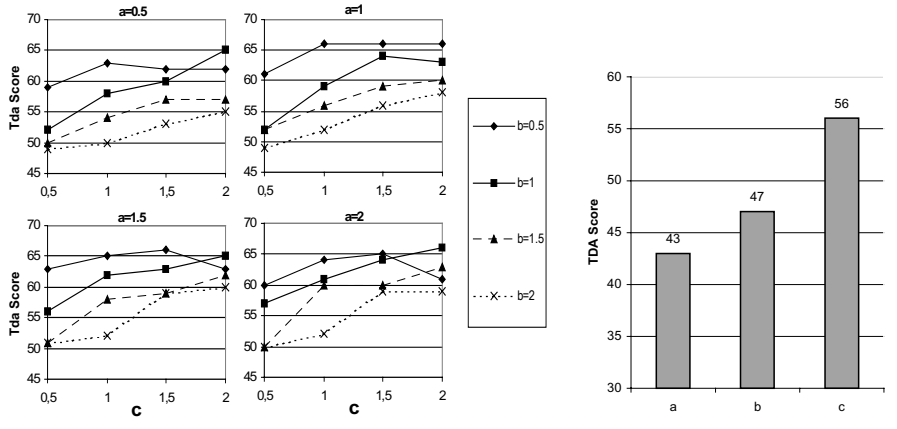


Fig. 3. Results from TDA test, using all the factors in the formula (left) and using only one factor (right).

Fig.3 shows the results achieved by our program, in the TDA test. For values of a, b and c ranging from 0.5 to 2. Our program's TDA score varies from 49 to 66 and the average score is 58.6. Thus, our minimum score is larger than the average achieved by humans (45.680), and our average score is larger than the one achieved by fine arts graduates (55.6897). Using only one factor of the formula (i.e. with the corresponding parameter different from zero and the other two equal to zero) we also get interesting results. Using the IC or PC factors by themselves gives low TDA scores (43 and 47). Using the (PC(t0)-PC(t1)) factor alone gives a TDA score of 56. These results are consistent with the presented theory, i.e. neither Image Complexity nor Processing Complexity, by themselves, are insufficient to assess Aesthetic Value.

5 Conclusions and Further Work

In this paper, we presented a brief theory of aesthetics. This theory considers the biological roots of Art and Aesthetics, stating that the assessment of an artwork depends on two factors: content and form. Furthermore, we claim that aesthetic value

is connected to the image-processing task of the brain. We relate aesthetic value with image complexity and with processing complexity. We devised a formula and estimates, in order to test our theory. The experimental results achieved in the TDA test were surprisingly good, especially if we consider the roughness of the estimates. The presented formula is based on the theory, which doesn't mean that there aren't other formulas/methods that could express our theory equally well (or better).

We believe that our theory is an approximation to the truth. We also believe that basing the development of a constructed artist on a consistent aesthetic theory will increase the quality of the results.

In spite of the results achieved on the TDA, it is important to say that we can't, directly, use our formula to guide the generation process of a constructed artist. The images in each test item are quite similar, both in style and complexity. Our formula showed to be effective discriminating between these images. However, it falls short when comparing images with significant style/complexity differences. The problem is attached with the $PC(t_0)$-$PC(t_1)$ factor that can take values very close to zero, and thus, increase the result value enormously. We are currently developing a test set with images of dissimilar styles. It is our belief that with some "minor" changes the formula will be able to pass the test.

Acknowledgments

We would like to thank CISUC (Centro de Informática e Sistemas da Universidade de Combra) for supporting our research.

References

1. Baluja, S., Pomerlau, D., Todd, J., *Towards Automated Artificial Evolution for Computer-Generated Images*. Connection Science, 6(2), 1994, pp. 325-354.
2. Fisher, Y. (Ed.), *Fractal Image Compression - Theory and Application*, Springer-Verlag, 1994.
3. Graves, M., *Test of Drawing Appreciation*, The Psychological Corporation, 1977.
4. Kurzweil, R. *The Age of the Intelligent Machines*, 1990.
5. Machado, P., Cardoso, A., *Model Proposal for a Constructed Artist*, World Multi-Conference on Systemics Cybernetics and Informatics - SCI'97.
6. Minsky, M., *The Society of Mind*. Simon & Schuster, 1986.
7. Moles, A., *Art et ordinateur*, Casterman, 1971.
8. Morris, D., *The Biology of Art*, 1962.
9. Nelson, M, Gailly, J.-L., *The Data Compression Book*. M&T Books. 95.
10. Spector, L., Alpern, A., *Criticism, Culture, and the Automatic Generation of Artworks*, AAAI-94, MIT Press.
11. Vignaux, G., *Les Sciences Cognitives*, Éditions La Découverte, 1992.

Manipulator Robots Using Partial-Order Planning

José Helano Matos Nogueira

Sebastião Cícero Pinheiro Gomes[1]

Statistic and Computing Department
State University of Ceará - UECE
Fortaleza, CE - Brazil

e-mail: helano@lcc.uece.br

Abstract. Intelligent task planning executed by manipulators robots and mainly its integration with the development of efficient controllers responsible for the execution of these tasks, constitute a non completely well solved problem, for sure needing more researches. In this way, it was created two computational tools: NONLinear Intelligent Planner (NONLIP), and the Simulator with Interface for Manipulators (SIM). Good results have been achieved in a general way for all simulated situations. This permit to conclude that the proposed formulations can be implemented in a practical device since the simulations use realistic models validated experimentally.

1. Introduction

Several researches have been achieved with the purpose of provide some way of intelligence in robots [1, 2, 5, 6]. However, it has been verified that the complex task planning to be performed for manipulators robots constitute an open problem. That is verified mainly due to absence of an appropriated methodology for automated generating of performed plans and also due to difficulties that appear at the moment of group these plans with the conception of effective control synthesis. One of the prepositions in this work consists of indicate how to integrate an intelligent task planner with a control system, developed in a manner to guarantee a powerful accomplishment of these tasks. Hence, it is been proposed an intermediate stage between the planning of the task and the control. This new stage is the automatic generation of trajectories that will be used by the manipulator. It was created specifically to facilitate the integration of the planner with the control law.

In addition, this work proposes a mechanism for automatic generation of intelligent plans into manipulator robots, and alternative ways of how to project the

[1] Professor at DMAT, FURG University, Rio Grande - RS, Brazil.

F. Moreira de Oliveira (Ed.): SBIA'98, LNAI 1515, pp. 229-238, 1998.

controller in order that these plans have been realized with success in this sort of robots. The technique of intelligent planning can find out a plan to solve a task of the same way that a person tries to solve the problem. Nevertheless, case the steps of a problem solution over the real world can not be canceled or ignored, the planning becomes important, and so useful, since that the steps of solution over the real world sometimes are irreversible, but their simulation into computer are not. Therefore, it is possible surmount constraints of the real world searching a complete solution.

To achieve all the proposed objectives, it was created two computational tools: an NONLinear Intelligent Planner (NONLIP) implemented in *PROLOG* and the Simulator with Interface for Manipulator (SIM) implemented in *DELPHI*. The computational system NONLIP was created using advanced artificial intelligence techniques as means-ends analysis, automatic detection of reduced solutions and unproductive plans, least-commitment and heuristic search. In a formal way, NONLIP has a mechanism of knowledge representation and deductive inference based on first-order logic with the purpose to give soundness to the system. Issues of the tasks planning, trajectories generation, dynamic modeling and synthesis of control law can be observed into SIM system. SIM has a user-friendly interface facilitating the project of controllers. Therefore, SIM generates the trajectories of control from the information obtained into NONLIP allowing the visualization of realized simulations upon dynamic models (actuators and structures) in a quite realistic and already validated experimentally form [3], fact that makes easy the project of efficient controllers.

2. Plan Generation

A *plan* is composed in a set of actions that will be executed by a manipulator robot. The *plan generation process* in an automatic way consists of split the task in a sequence of steps that solve the modeled problem. Analyzing the features of the problem some inadequate steps of the solution can be ignored or canceled. If an unproductive path is detected by the planner, another path can be explored, backtracking to the point of last choice. The NONLinear Intelligent Planner (NONLIP) uses as input a set of operator (Op_i) and a problem. The problem is defined by a initial state (S_i), and a final state (S_f) that represents the objectives. Moreover, must have a knowledge base about the domain of application. This knowledge base is composed by facts and rules that map the real world knowledge in a computational model. See Figure 1.

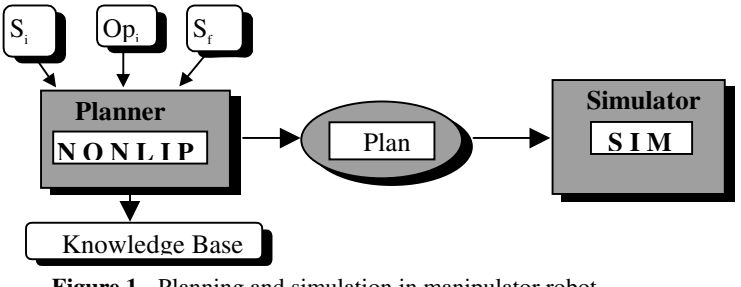


Figure 1 - Planning and simulation in manipulator robot

2.1 Knowledge Representation Using Actions

The world over what the plan is defined is called *application domain* or *planning world*. Nevertheless, any application in this planning world has the *frame problem*. However, the technique used to avoid the frame problem into NONLIP is to represent only the effects that will be modified from the action in the actual state — *locality effect*.

The planning technique can be applied on a large quantity of task-domains. However, to become easy the comparison of the various domains, it is recommended to consider all of them in only one domain, complex enough to show the necessity of each mechanism, but at the same time, simple to provide easy examples to be analyzed. Therefore, the manipulator on blocks world will be presented as example of application domain. In this domain the manipulator robot is instructed to transport blocks from an origin configuration (initial state) to a final configuration (goal state). The problem is to form a plan in the blocks world, that is, to specify a sequence of actions for restacking blocks to achieve a particular configuration.

Relationships in the blocks domain are:
on(Block, Object) and free(Block)

Each available action is defined in terms of their preconditions and effects. More precisely, each action is specified by:

(1) *Precondition*: this condition must be satisfied to the action be executed.
(2) *Addition List*: list of relationships that has the conditions that becomes true after the execution of the action.
(3) *Removal List*: list of relationships that is removed by the action.

The effects are defined by the predicates *add(Action, Relationship)* and *remove(Action, Relationship)*. In the first predicate — *add*, the argument *Relationship* represents a list of relationships that is added by the execution of *Action*; in the second predicate — *Remove*, represent a list of relationships that is removed by the execution of *Action*.

In the blocks world there is only one possible action: *move(Block,From,To)*, where *Block* is the block to be removed, *From* is the actual position, *To* is the new position of *Block*. Hence, to plan in the blocks world is a traditional exercise of planning. Meanwhile, it is easy to find real applications with this same sort of planning: container robots in charge terminals; mobile robots; assembly robots.

3. NONLinear Intelligent Planner (NONLIP)

All planners develop plans in stages, usually starting with the empty plan and ending (if successful) with final plan. The final plan is usually, but not always, a linear sequence of actions. In between the empty initial plan and the final one, planners maintain *partial plans*, which are repeatedly extended, refined, and otherwise modi-

fied. The term 'linear planners' refers to planners whose partial plans in these intermediate stage are linear sequence of actions. Early AI planners were all linear.

In linear planners, any partial plans under development consists of a linear sequence of actions. AI researchers recognized early on that the linear sequences might be an unnecessary limitation. For example, suppose a construction robot has a goal of placing a container at an elevated level. One plan would be to build a ramp, build a platform, and use the ramp to place a container on the platform. Obviously, both the platform and the ramp must be built before the container is moved, but it is unimportant (as far this goal goes) which is built first. Why then not delay this decision until a later point in planning, where it might make difference ? If we do delay the decision, we end up with only a partial order among the plan steps and, hence, a partial-order plan.

Delaying the order among plan steps is a form of *least-commitment* planning. Another form of the same principle exists, and it has to do with instantiation of variables. Suppose a two-handed robot needs to grasp a given object with its hand. Rather than commit to using one hand or the other, why not delay this commitment, so that if a future planning step calls for the use of a particular hand simultaneously with the grasping then at that point the other hand will be selected for the grasping. It should be added that, although least commitment is a natural notion, the extend to which it speeds up planning is still a matter of debate.

One notion investigated in connection with planning, and emphasized in the context of partial-order planning, is that of *planner completeness*. A planner is complete if it is guaranteed to produce a solution, if one exists. NONLIP maintains a symbol table, which is a mapping of plan-step names to actual operators. A partial ordering on the plan-step names is represented separately in the form of *safety conditions* and *causal links*. A safety condition is a statement that a particular instance of one action precedes another particular instance of some (possible identical) action. A typical safety condition could be "Step 5 (in which the robot picks up a block b) precedes Step2 (in which the robot unstacks blocks c from d)." We use the symbol $\prec$ to denote a safety condition, as in 'Step 5 $\prec$ Step 2'. A casual link is a strong relation between pairs of plan-step names; it not only implies an ordering of the two steps in the plan, but also supplies logical information about why the planner decided to impose the ordering. Specifically, a casual link between two steps means that some precondition p of the later step is caused by (i.e., appears in the add-list) the earlier step. The casual link is labeled not only with the two steps it relates, but with this precondition, as 'Step5 $\xrightarrow{p}$ Step2'.

Before we explain the operation of NONLIP, it is convenient to introduce two special steps: *before* and *end*, and two rules:

```
rule(begin, [ ], initial_state, [ ]).
rule(end, goal, [ ], [ ]).
```

The step *begin* is defined to precede all other steps that will ever exist in the plan, and the step *end* to follow them.

NONLIP starts with an initial partial plan; the partial plan may be the null one, which consists only of the steps *begin* and *end*. NONLIP then repeatedly refines and modifies its partial plan, until an acceptable plan is achieved. A plan is *acceptable* if it satisfies the following condition:

> for every precondition p of every plan step S2 there is another plan step S1 such that S1 $\xrightarrow{p}$ S2
> and such that
> for any other step S3 in the plan with p in either its addition or removal lists
> (a so-called thread),
> either S3 $\prec$ S1 or S2 $\prec$ S3.

In order to achieved this condition, the operations available to NONLIP are the additional of a safety condition, the addition of a causal link between two steps already in the plan, and the addition of a plan step. The operation of NONLIP is as follows:

Partial-Order Plan Procedure (Plan: a partial plan)

1) If Plan is acceptable, return Plan.

2) If the safety conditions in Plan are inconsistent then fail and backtracking.

3) If there is a causal link S $\xrightarrow{p}$ W in Plan and a thread V to this causal link in Plan such that neither V $\prec$ S nor W $\prec$ V are in Plan, then add either V $\prec$S or W $\prec$ V to Plan and go to 1

4) Otherwise, there must be a step W in Plan with precondition p such that there is no causal link of the form S $\xrightarrow{p}$ W in Plan. In this case do one of the following:

(a)Find a step S in Plan that adds p and add S $\xrightarrow{p}$ W to Plan.
(b)Find an operator O that add p, create a new entry S in the symbol table which maps to O, and add S $\xrightarrow{p}$ W to Plan.

5) Return to 1

The Prolog implementation below uses the following representation:

The symbol table is a list of (Name, Op) pairs.

The safety condition is represented by a list of (A, B) pairs, interpreted as A precedes B.

The causal links are represented by a list of (S, P, W) triples, where S and W are steps names, and P is the condition which is in the addition list of S and precondition list of W.

The procedure above contains a number of choice points. The implementation of NONLIP opts for systematicity over efficiency. Systematicity is achieved by conducting an iterative-deepening search, exhaustively checking all plans of less than n steps before increasing n. The result is a planner that is complete.

4. Operation Module

From this section, we analyze the operation module, that is, the plans were already generated by NONLIP, the actions which will be performed by the manipulator are consequently known. Taking as example the block assembling, the position coordinates of each block are known on the initial, intermediate if any , and final states, and all this positions are inside the manipulator useful working volume. Considering the manipulator on its initial repose configuration, the first action of the plan to be performed by the manipulator should be moving the block *c*, which is on block *a,* to the table. The second should be put *b* (initially over *a*) over *c* the third and last should be put *a* (initially on the table) on *b*. Then the manipulator can return to its initial configuration. Upon each action performed by the manipulator, only are known the initial and final positions of the extremity, becoming then necessary to get continuous spatial trajectories, connecting these positions, trajectories which must be followed as faithfully as possible. This will be possible only if the control system project is efficient enough. After the plan generation, made of actions, these information are passed to the trajectories generator, which after generate inform them to the control law, which in turn use the fundamental information of the manipulator dynamic model. The control law sends them the torque to the manipulator actuators (mostly motor gear), and receives the observations coming from the sensors (generally, rotor angular velocities and gear output axes angular positions).

There are two main operation modes: planning trajectories *off* and *on line.* In first case, the physical problem is kept unchanged for a long time and so, the task planning will be performed only one time. Intelligent task planning and the consequent trajectories generation would be valid until a new change on the physical problem occurs. This could be the situation, for example, component assembling without changing the initial and final states, and also, without any changes inside the manipulator working useful volume. The second could be the apposite case, on which many variations may occur on referred positions or eventual obstacles positions in the useful volume. Follow the information of how the Trajectories Generator, Dynamic Model and Control Law blocks were got.

4.1 Trajectories Generator

As explained before, each action planned in intelligent form indicates which manipulator extremity must leave from a position $\left(x_i, y_i, z_i\right)$ on time t_i to a final position $\left(x_f, y_f, z_f\right)$ on time t_f. The manipulator has *n* degrees of freedom, so it is possible from the inverse cinematic relations, to get the *n* angular shifts relative to each manipulator link, or else:

$$\theta_{ki} = f\left(x_i, y_i, z_i\right) \quad \text{and} \quad \theta_{kf} = f\left(x_f, y_f, z_f\right) \tag{1}$$

where k ranges from 1 to n. The aim is to get the trajectories $\theta_k(t)$, $\dot{\theta}_k(t)$, and $\ddot{\theta}_k(t)$, which connect the angular initial and final positions already known (equations 1). On this article, new trajectories are proposed (equations 2, 3 and 4), developed aiming to increase the used control law performance:

$$\theta_k(t) = \left(\theta_{kf} - \theta_{ki}\right)\left(1 - e^{-\left(\frac{t}{t_f - t}\right)}\right) + \theta_{ki}; \tag{2}$$

$$\dot{\theta}_k(t) = \left(\theta_{kf} - \theta_{ki}\right)e^{-\left(\frac{t}{t_f - t}\right)}\left(\frac{t_f}{\left(t_f - t\right)^2}\right) \tag{3}$$

$$\ddot{\theta}_k(t) = \frac{\left(\theta_{kf} - \theta_{ki}\right)\left(-\frac{1}{\left(t_f - t\right)} - \frac{t}{\left(t_f - t\right)^2}\right)e^{-\left(\frac{t}{t_f - t}\right)}t_f}{\left(t_f - t\right)^2} + \frac{2\left(\theta_{kf} - \theta_{ki}\right)e^{-\left(\frac{t}{t_f - t}\right)}t_f}{\left(t_f - t\right)^3} \tag{4}$$

where $k=1,...n$ (number of links), θ_{kf} and θ_{ki} are the angular initial and final positions respectively, while t_f correspond to the time spent to hit the final position from the initial (considering initial time of each trajectory as null).

The central idea, which originated these trajectories was based on continuity of successive derivatives of the angular position, allowing then soft trajectories that facilitate the developing and implementation of control laws and also, generate control torque curves without discontinuities, avoiding existence of undesirable vibrations during the manipulator movement.

4.2 Dynamic Modeling

It was idealized a manipulator with three links for performing the tests, as shown in the following figure:

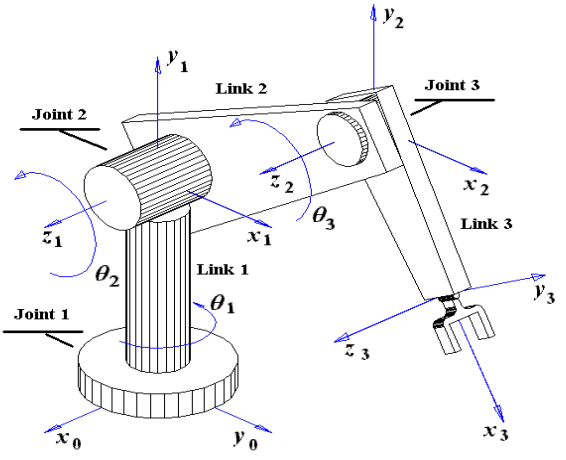


Figure 2 - Manipulator with three links idealized for simulating.

The modeling process of rigid bodies articulated chain as the manipulator on figure 2 is already well-known and do not present any problem. In general, the Euler-Lagrange formalism is used, and this need the kinetic and potential energies of the system. Beyond the joints and payload masses, the link masses were also considered, and also including the rotational inertia of each mass of the system. The differential matrix equation representing the dynamic manipulator model is on the following form:

$$\left[I\left(\vec{\theta}\right)\right]\ddot{\vec{\theta}} + \vec{f}\left(\vec{\theta},\dot{\vec{\theta}}\right) + \left[f_v\left(\vec{\theta},\dot{\vec{\theta}},\vec{T}_m\right)\right]\dot{\vec{\theta}} + \vec{g}\left(\vec{\theta}\right) + [K]\vec{\theta} = \vec{T}_m \tag{5}$$

where $\vec{\theta} = \left(\theta_{r1}\ \theta_1\ \theta_{r2}\ \theta_2\ \theta_{r3}\ \theta_3\right)^T$, θ_{r1}, θ_{r2} and θ_{r3} are the rotor angular positions 1, 2 and 3 respectively. The vector of motor torques is on the form : $\vec{T}_m = \left(T_{m1}\ \ 0\ \ T_{m2}\ \ 0\ \ T_{m3}\ \ 0\right)^T$, where T_{m1}, T_{m2} and T_{m3} are the torque applied on the rotors 1, 2 and 3. The other terms that appear on the equation (5) are with the definition: $\left[I\left(\vec{\theta}\right)\right] \Rightarrow$ system inertia matrix; $\vec{f}\left(\vec{\theta},\dot{\vec{\theta}}\right) \Rightarrow$ vector with the Coriolis - centrifugal torques; $\left[f_v\left(\vec{\theta},\dot{\vec{\theta}},\vec{T}_m\right)\right] \Rightarrow$ diagonal matrix with the variable viscous friction coefficients [4]; $\vec{g}\left(\vec{\theta}\right) \Rightarrow$ vector of gravitational torques; $[K] \Rightarrow$ matrix with elastic constants of the drive joints;

The equation (5) indicates that the equations of the three drive joints dynamics were increased to the structural model and so, adding the rotor vibrations modes, with all the consequent non-linear friction considered in models (see [4] to more details about drive joint dynamic modelling).

4.3 Control Law

The most used control technique are based on the compensation of the non-linear coupling dynamics from the control torques. To perform the simulations shown in this paper, a control law was designed using inverse dynamics. It is question of non-linear control, multivariable, projected specifically to have good results with the trajectories introduced in section 4.1. Hence, the control torques can be defined as the form:

$$\vec{T}_m = \vec{f}\left(\vec{\theta}, \dot{\vec{\theta}}\right) + \vec{g}\left(\vec{\theta}\right) + \vec{T}_c \tag{6}$$

where

$$\vec{T}_c = \left[I\left(\vec{\theta}\right)\right]\left\{\ddot{\vec{\theta}}_c + \left[K_d\right]\left(\dot{\vec{\theta}}_c - \dot{\vec{\theta}}\right) + \left[K_p\right]\left(\vec{\theta}_c - \vec{\theta}\right) + \left[K_i\right]\int_0^t \left(\vec{\theta}_c - \vec{\theta}\right) d\tau\right\} \tag{7}$$

The equation (7) is specific to the case of a proportional, integral and derivative control (PID), which $\left[K_p\right]$, $\left[K_d\right]$ and $\left[K_i\right]$ are the gains matrix, and $\vec{\theta}_c$ corresponds to the reference angular positions vector. Figure 3 presents a simulation when the global model (actuators and structure, equation(5)) is considered.

The simulation corresponds to the following situation:

manipulator links	initial position (deg.)	final position (deg.)	final time (s)
1 ($Rot. \Rightarrow \theta_1$)	0	60	1
2 ($Rot. \Rightarrow \theta_2$)	0	-50	1
3 ($Rot. \Rightarrow \theta_3$)	0	45	1

The final time corresponds to the imposed trajectory time (t_f in equations 2, 3 and 4), in order to the final position to be hit from the initial. The simulation time was 1.2 *s*. The trajectories then were generated using this information and the controller got in action, sending the control torques to the drive joints.

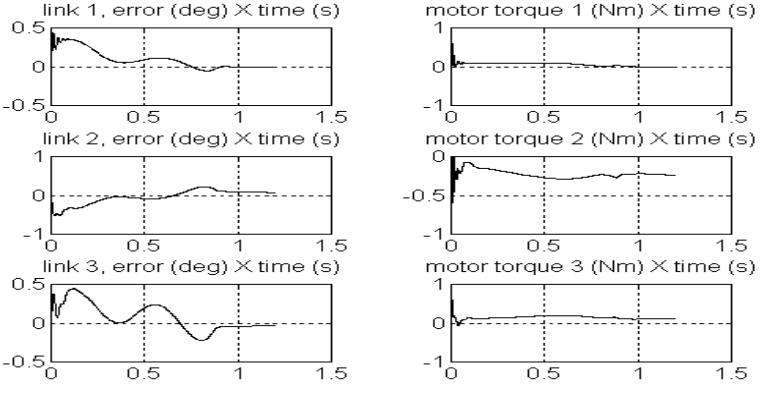


Figure 3 - Simulation showing the timing evolution of position errors

As illustrated in figure 3, the errors that appear on trajectories tracking are really small, so the errors after hitting final positions can be despised. It must be reminded that the dynamic model used is realist enough, mainly because the model used to the actuators considered non-linear dynamic, and it was experimentally proved, providing good results in all evaluated situations [3, 4]. Another important information is that the control law was developed aiming the result quality maintenance, independent of the amplitude of final position references.

5. Conclusion

This work presents an efficient and productive way for intelligent task planning and its application in manipulator robots. With this purpose, two computational tools was created: NONLinear Intelligent Planner (NONLIP) and the Simulator with Interface for Manipulators (SIM). The use of these tools had fundamental importance because with them it has been possible simulate manipulator robots that realize optimized actions conducting to productive paths for solution of these tasks. The nonlinear intelligent planner, NONLIP, generates plans in an automated way with use of artificial intelligent techniques applied in robotics. Also, with NONLIP it was possible detects new solutions for inefficient tasks and the same time proposes new solutions. Control law and the reference trajectories propose in this work ware tested in the SIM and they were elaborated in the form to give the same quality in terms of precision, stability, absence of undesirable vibrations, and answer time independently of the level of the amplitudes in terms of finals positions commanded. This result is fundamental for manipulators that destine to realize non repetitive tasks, for example from a planning that use artificial intelligence techniques.

References

1. Allen B. P.; Case-based reasoning: Business Applications; Communications of ACM; 37(3):40-42; March 1994.
2. Baral C., Son T. C.; Relating Theories of Actions and Reactive Robot Control; AAAI; Workshop on Reasoning About Actions, Planning and Robot Control; 1996.
3. Gomes S. C. P., Chrétien, J. P.; Dynamic Modeling and Friction Compensated Control of a Robot Manipulator Joint; In IEEE Robotics and Automation Conference; Nice; France; p 1460-1466; 1992.
4. Gomes S. C. P.; Modeling of Friction in Drive Joints of Manipulator Robots; In XIII Brazilian Congress of Mechanic Engineering; CD-ROM; Belo Horizonte; Brazil; 1995.
5. Nogueira J. H. M., Gomes S. C. P; Intelligent Task Planning; In proceedings of the International Workshop on Intelligent Robots; Brasília - DF; Brazil; 1997.
6. Nogueira J. H. M., Gomes S. C. P.; Automatic Plan Generation for Manipulator Robots; In proceeding of the I Brazilian Artificial Intelligent Meeting (ENIA97); Brasília - DF; Brazil; 1997.
7. Nogueira J. H. M; Applications of Plan Recognition; In proceedings of the XX National Congress on Applied Mathematics; Gramado - RS; Brazil; September; 1997.

Darwinci: Creating Bridges to Creativity

Francisco C. Pereira[1], Penousal Machado[2], Amílcar Cardoso[1]

[1]Departamento de Engenharia Informática da Universidade de Coimbra
{camara, amilcar}@dei.uc.pt
[2]Instituto Superior de Engenharia de Coimbra
machado@dei.uc.pt

Abstract. This paper presents Darwinci, a system that generates new ideas, using a multi-domain knowledge base composed by musical and drawing structures. Its theoretical background comes from the theory of Divergent Production as stated in [7], Genetic Algorithms and metaphor interpretation. Rather than attempting to model human thought, Darwinci's main goal is to make cross-domain transfer of ideas, in order to create something new, thus becoming a good proposal for creative problem solving.

1 Introduction

It is known and stated that some of our most brilliant ideas come up when our mind "wonders" around something that has nothing to do with the problem we are trying to solve. Due to the complexity of the world and to its representation in computers, the elaboration of a "divagation" machine has never appeared, as far as we know, in AI literature. However, several psychologists agree that this capacity of "wondering" is very important in solving problems that involve some degree of creativity [3].

The work presented here is part of an ongoing research project, centered on an important theory of creativity: J.P. Guilford's theory of "divergent production". More specifically, this paper presents an implementation of what Guilford calls "the ability to generate variety and amount of information". Darwinci uses a genetic algorithm (GA) to produce new ideas, supported by a multi-domain knowledge base. Each domain will be composed of two types of information: a set of tree-like structures representing "individuals" (e.g. musical pieces, drawings), a set of conceptual nets to assert knowledge about it. This representation can be used with a variety of domains, including Music and Visual Arts (our experimental domains). As will be shown, we find several conceptual and structural connections among these forms of art.

The paper is organised as follows: Section 2 is dedicated to a small survey and dissertation on the study of creativity. We believe that the importance of this area touches not only psychology but also AI. In sections 3 and 4, we will describe our previous work in Music and Visual Arts. The inter-crossing between the two domains will be addressed in section 5. Finally, in section 6, we draw some generic considerations about Darwinci, and point to some unexplored aspects in this field.

F. Moreira de Oliveira (Ed.): SBIA'98, LNAI 1515, pp. 239-248, 1998.

2 Studying Creativity

The study of creativity has motivated research from various viewpoints like the creative product; the creative process; the creative environment; the creative person [3]. Psychologists like [7], [5] have dived into the search for answers to questions like: How do we create novel ideas? Why are some people more creative than others? Are there any special traits directly related to this capacity? How to measure it?

In AI many relevant examples have already emerged, be it related to analogy [6], metaphor and linguistics [15], [6], neural networks [8], or creativity modeling and studying [2]. Whether in search for new problem-solvers, for ways of understanding cognition or programs that create interesting things, the motivations behind these studies touch this ability of ours: creativity.

Without making claims on how do we create or what is a creative product, our interest is the computational application of theories of creativity in order to create new ideas.

The work we present in this paper is connected to some ideas of Guilford [7]. In his theory - Structure of Intellect -, divergent production is considered as the ability to "generate variety and amount of information; most involved in creative potential"[3]. More specifically, Michael [11] states that this can be achieved by recurring to the process of "transform recall", in which the same information is applied to different contexts (as if one is trying to apply an old puzzle piece into a new one). According to Guilford, creativity is directly related to our capacity to interconnect and correlate apparently divergent ideas, i.e., to see similarity where, by default, there seems to be none (e.g. the famous example of the self-biting snake and the benzene molecule of Kekulé's dream). Computationally, this can only be achieved with a high degree of flexibility in the representation and some tendency to divagate through unrelated areas, which means that we must enable associations between concepts from various areas. A program able to do it must accept an open representation (in the sense that it must allow different domains) and have a mechanism to operate on the knowledge itself without loosing semantics.

Our approach to computational production of (as possible creative) ideas is mainly based on: divergent production and operations on knowledge.

We have chosen two domains in which we have some experience and where creativity is relatively unconstrained (in opposition to Natural Language, for example). These are Music and Visual Arts. We have already developed systems under these two subjects: SICOM [12] and NevAr [10].

In both works, structure and hierarchy play an important role and we believe that, at least in upper levels of abstraction, one can surely try this cross-domain divergent production between Music and Visual Arts. Parallels between these two forms of expression are neither few nor new. It's common sense that there is strong intersection between the several forms of art. In fact, history tells us that a given style is not associated to a single form of art, its general abstract characteristics are spread through various artistic and geographic areas. Maybe due to this, specific concepts of each form of art are often applied to others, yielding new and fertile metaphors. Concepts like "colour", "texture" and "contour" are often present in Music. Likewise,

"rhythm", "dissonance", are used in Visual Arts. Other concepts like harmony, motive, tension or dynamics are equally used in both domains. As we do not want to play paintings or to paint musical pieces, we will not "force" correspondence between those concepts.

As will be seen bellow, the connections between domains are generated through a mechanism derived from the work on metaphor of [15] and [6]. In the next two sections, we will describe previous work on Music and Visual Arts, switching gradually to the main scope of the paper.

3 Creating Music

The analysis and composition of Music by computers has already some history. Its state-of-the-art, by now, comprehends several works, both in analysis [14] and in composition [4].

Following ideas from [4] and [9], our method of generation of music is based on the reusability of previous ideas. Although in former work [12] we were interested in creating entire pieces, by now and for the present purpose, we are focussing on the creation of chord sequences with rhythm. A piece of music can be represented by a hierarchical structure corresponding to its harmonic, melodic and rhythmic analysis. Creating a new musical piece, consequently, can be seen as constructing such an organisation. So, these chords sequences come out from the construction of a tree-like structure, much in the same shape as in [9] where, analytically, one can represent an entire music with a hierarchical structure defined by grouping, time-span and metric rules. In the next paragraphs, we will describe briefly some aspects of our musical system (SICOM), which generated new pieces through the use of Case-Based Reasoning [12].

SICOM used pre-elaborated analysis of music coded as trees, with non-hierarchical links between nodes, used for explaining relations among them (see fig.1).

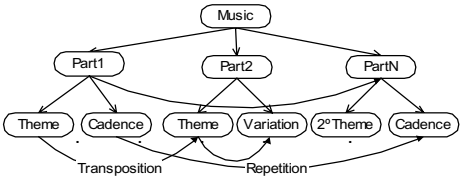


Fig 1. A Musical Structure

In the act of producing new structures, the system used these links as "suggestions" with associated strength (for example, Repetition may be strong and Transposition may be weak, in fig.1) to reduce the search space and to keep some coherency throughout the piece.

To make a synthesis of the SICOM's process of generation, the algorithm takes four steps: (1) Search for an acceptable node in the memory; (2) Apply it and, if

needed, adapt it; (3) Spread its "suggestions"; (4) If no more nodes are expected, finish, else go to 1.

The last condition is simply accomplished by structural links, i.e., when choosing a node, its structural connections to descendants are also spread, becoming strong suggestions. If there are no more of these, the piece is finished. Since explaining SICOM in detail is not the scope of this paper, we redirect the reader to [12] for further details. For present purposes, we retain some of this system's characteristics, namely:

- Representation – Although applying some changes in order to approach closer to Lerdhal and Jackendoff's theory enhancing concepts like "tension" and "relax", the basics of the former representation are kept.
- Domain adaptation – While developing SICOM, we have built some adaptation algorithms. Although highly biased by the musical idioms we worked with, we keep the main ideas.
- Some theoretical background – We have already applied some ideas from Guilford [7] in SICOM. Then, it was centred mainly on work on the similarity metric. Presently, we are following different paths to explore these ideas.

4 Creating Images

The basic idea for our approach to the generation of images is based on an ongoing research project [10]. The idea is using a genetic algorithm to generate aesthetically pleasing images, which has already been shown to be effective [13][1]. The main problem of this approach lies on the assignment of fitness values to the individuals. Generally, the fitness is supplied by the user (e.g. [13]). In [10][1], the evaluation of the individuals is made through artificial neural networks.

Our approach to the "evaluation problem" is radically different, and will be described in the next section. For the scope of this section, let's consider that evaluation is made by a black box. Apart from this, our algorithm is very close to the one presented in [10]. The differences are related to representation, resulting in minor changes to the crossover and mutation algorithms.

4.1 Overview of the algorithm

One of the innovative aspects of the algorithm presented in [10] is the use of background knowledge. This is achieved in two different ways:

1. The initial population does not need to be random. Any set of images, including famous artworks, can, potentially, be used as initial population.
2. It maintains a knowledge base (KB) of images, which are not subject of selection. The images in this KB can be selected for mating with others from the current population or between themselves. The KB can be initialized with any set of images we choose. Additionally, individuals generated by the system can be added to the KB.

The use of a KB has additional benefits, namely, the increase of population diversity and preventing the early lost of remarkable individuals [10].

Let's make a brief overview of the generation algorithm: we start with a working set of images built by the KB and the current population. The individuals in this set are already evaluated.

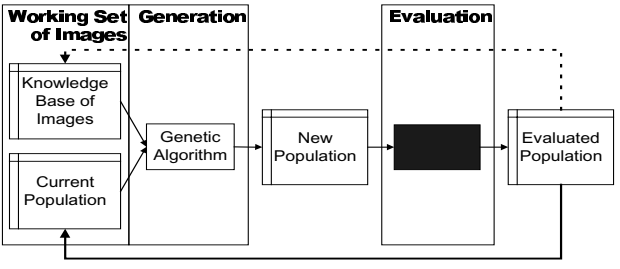


Fig. 3. The generation algorithm

From the working set, the GA creates a new population through genetic recombination of the selected individuals (probabilistic selection based on fitness). The images of this new population are evaluated, resulting in an evaluated population, which becomes the current population. Additionally, images that have an aesthetic value higher than *c*, or superior to the average of the population by a value *d*, are added to the KB. A new working set is achieved and the process is repeated.

4.2 Image representation

The representation issue is one of the most important ones in this type of system. One common approach [13][10][1] is to represent the images through mathematical functions. This representation has several drawbacks. We are limited by the representational power of the used formulas, and there is no procedure for converting a generic image to this type of representation. This creates a great problem to us since we want to use artworks made by human artists. A further problem results from the fact that we don't have any explicit representation of the basic building blocks of the image nor of its relations. Considering these problems, we chose a different representation, which is quite similar to the one used in the music domain.

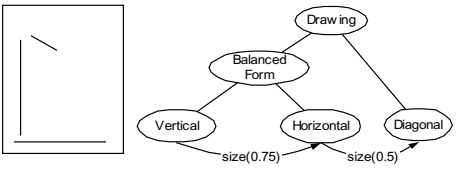


Fig. 4. A simple drawing and its representation.

This representation is still tree-like, allowing us to use the same type of crossover and mutation operators. In our previous work, the crossover of two individuals consisted in the exchange of sub-trees between these individuals. Our new crossover operator also exchanges sub-trees. However, since we have relationship links, these can be broken during crossover. When this happens, the algorithm searches for new relations so that the broken links can be reestablished. The same happens during mutation.

5 Mixing Structures

The organisation of Darwinci corresponds to the following diagram:

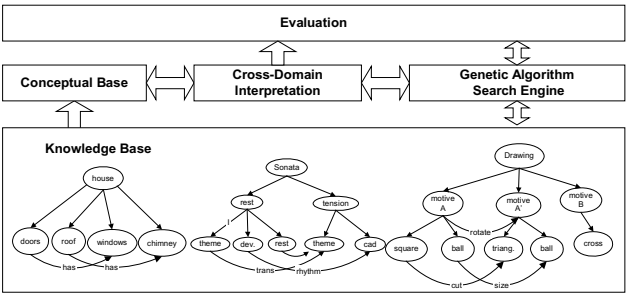


Fig. 5. General architecture of Darwinci

The basic blocks of Darwinci can be described as follows:

Knowledge Base – Set of structures from a variety of domains. Its representation is similar to the one described in previous sections.

Conceptual Base – Composed by the conceptual graphs describing domain knowledge, organized according to an implicit ontology (easily extractable by following "isa" arcs). These graphs allow inter-relations among any pair of concepts.

Genetic Algorithm Search Engine – This algorithm is similar to the one described in the previous section. The only difference lies in the fact that we are dealing with individuals from different domains. This, however, doesn't imply radical changes to the algorithm, since the representation is kept among domains. Therefore, the main difference lies in the fitness function. In Darwinci, the Evaluation module determines the fitness values. During mutation and crossover, some of the previously existing relation links may be broken. The algorithm tries to re-establish links through the search of new relations, which must be coherent with the conceptual. This search is made in a conservative way; i.e. it tries to maintain the previous relations with the minimum amount of change. This is made through the use of the Cross-Domain Interpretation Module.

Cross-Domain Interpretation – This is one of the key blocks of Darwinci. When interpreting a structure, this module establishes cross-domain links between the

domains in question; i.e. it establishes a metaphor between the domains [15] and [6]. The individual will be interpreted according to this metaphor (e.g. “colour gradient is harmonic progression”). It creates correspondences that control the translation of the structure to a specific domain. The nodes of a given structure are all interpreted in accordance to the same metaphor. In other words, when using a metaphor (like “harmony is color contrast”), the system is supposed to create new connections, (like “chord is color”, “Dominant Major is Blue”) and use them to translate nodes.

Evaluation – This module is responsible for the assignment of fitness, which relies on the coherency of the individual with the conceptual nets present in the system. We can test coherency with respect to several domains (e.g. music, painting) or we can “direct” the search to a specific domain. To assess coherency, the Evaluation Module calls the Cross-Domain Interpretation. Following the current metaphor links, it finds concepts that can evaluate coherence (suppose, in music, the “Sonata form” net, which would have a set of concepts that characterise it: “has_many cadences”, “Middle_Section is Dominant”, etc.) and uses it to calculate the fitness.

6 An Example

In this section, we show a small example of Darwinci. Suppose the conceptual base corresponds to fig. 6 and that we have an initial knowledge base as in fig. 7.

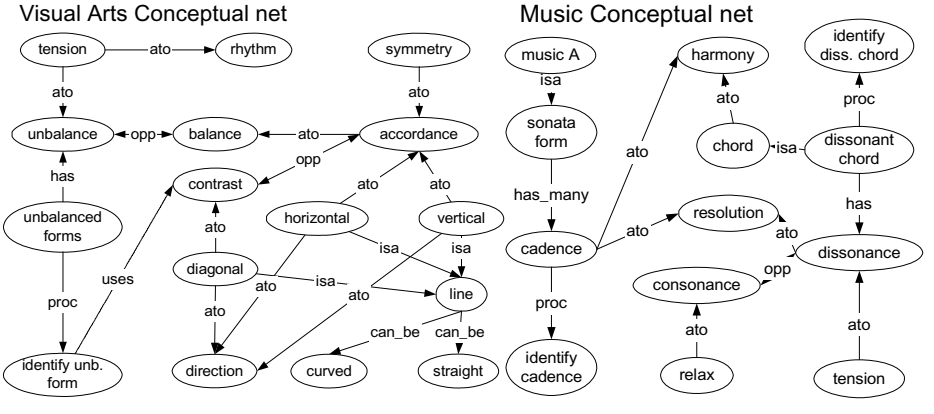


Fig. 6. Musical and Visual Arts Conceptual nets. “ato”–associated to; “ako”–a kind of; “isa”–is a; ”proc”-procedure; “opp”–Opposite

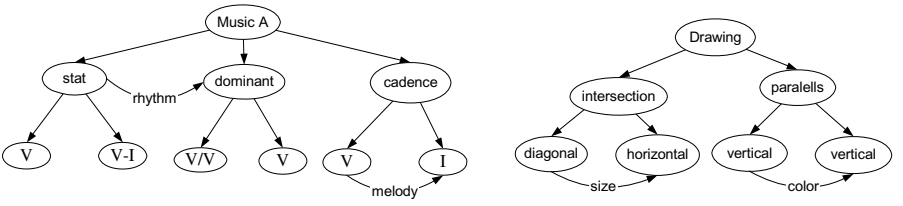


Fig. 7. Two structures of Music and Drawings

The GA starts by selecting a random set of individuals from the knowledge base. Let's assume it picked the two structures from fig. 7, and that the following individuals were created through recombination:

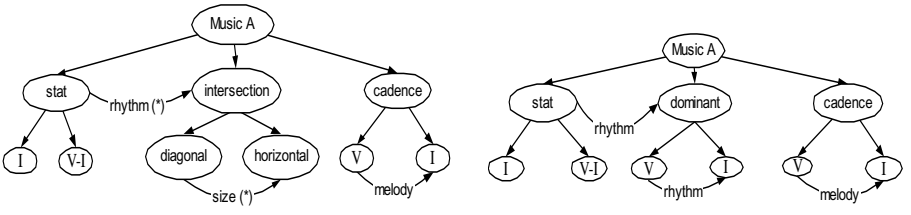


Fig. 8. On the left, the resulting structure, before inter-domain "translation". (*) Broken link. On the right, the final structure

In the following step, the cross-domain interpretation, Darwinci establishes new "metaphorical" links between conceptual nets. (fig. 9).

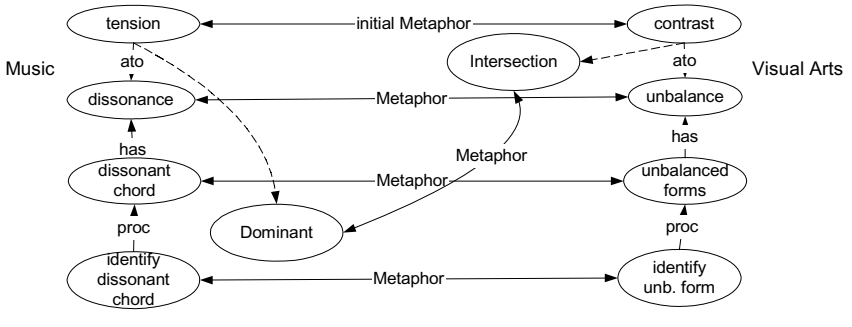


Fig 9. The metaphor links have been spread across both nets

As can be verified, Darwinci has (in this example) an initial metaphor between "tension" in music and "contrast" in visual arts. Departing from this, it creates a

structure mapping that will be used as the base for interpretation. Following the systematicity principle [15], some of the remaining "translations" are generated (assuming a generated metaphor link between "main degrees" and "direction", the result could be "Horizontal = I", "Vertical = IV" and "Diagonal = V").

Still in the cross-domain interpretation phase, the program tries to re-establish links. One of the ways to do this is simply by reapplying it to the new situation. Suppose this is what happened to the first broken link (rhythm). The other way to re-establish links is by following metaphoric links. Let's believe it would follow them and detect a link between "rhythm" and "size". Then it would be recovered with success. The new structure is presented in Fig. 8 (right).

The next phase is the evaluation. This critical step is also supported by the metaphor links. As can be seen, the concept "has_many cadence" is associated to "Sonata", which is associated to "Music A". According to this, Darwinci executes the procedure associated to "cadence", counting the number it "succeeds". This, with other concepts (like "dominant in middle_section") will help the program to evaluate the fitness.

Following this algorithm, the program is expected to arrive at some interesting solutions. Some of these steps are not yet implemented, but the basic ideas were shown. For the sake of simplicity, this example is highly biased (there can be many unfortunate interpretations and crossover results) but, with the help of the GA, Darwinci is expected to get solutions like the one above sooner or later.

7 Conclusions and Further Work

This paper presented the overall architecture and theoretical background behind a work on computational creativity. As in other works, it is natural that implementations and further investigations mutate the original idea gradually until getting the final result. But we think this is a rather mature framework at least in what concerns to the idea of using a GA to create a multiplicity of structures and using "metaphorical" connections between conceptual webs to make cross-domain interpretation. The authors have already achieved some confidence on working on the generation of musical and drawing structures in previously published work.

Some parts of this work, namely the conceptual nets, need an extensive search in order to achieve an acceptable degree of coherency and completeness. In order to do it, and to allow relatively low complexity, we must assume that there must be a limited set of relations among concepts. This will introduce some bias in the cross-domain interpretation, but we hope to keep some expressiveness on these links.

Presently, the cross-domain interpretation and the conceptual base modules are being implemented. We are following previous work by [15] and [6]. After the complete interpretation of this system, we envisage the following two further steps: the application to other domains (like story plots and dance) and the design of a learning module (at the level of the conceptual base module).

Without aiming to obtain results of excellence in the creation of artworks, we hope to shed some light on the search for creative production in Artificial Intelligence.

Acknowledgements

We would like to thank CISUC (Centro de Informática e Sistemas da Universidade de Combra) for supporting our research. The first author is sponsored by a grant PRAXIS XXI BD 9611/96.

References

1. Baluja, S.; Pomerlau, D.; Todd, J.; 1994*, Towards Automated Artificial Evolution for Computer-Generated Images*. Connection Science, 6(2), 1994, pp.325-354.
2. Boden, Margaret; 1990. *The Creative Mind: Myths & Mechanisms*. Basic Books.
3. Brown, R. T. ; 1989. *Creativity: what to are we to measure?* In Glover, J.; Ronning, R. and Reynolds, C., Handbook of Creativity, Plenum Press, New York.
4. Cope, D.; 1996; *Experiments in Musical Intelligence*. Vol.12 of the collection: The Computer Music and Digital Audio Series. A-R Editions.
5. De Bono, E.; 1986. *El pensamiento lateral*. Manual de la Creatividad. Barcelona, Espanha: Padóis.
6. Gentner, D.;1983. *Structure Mapping: A Theoretical Framework for Analogy*. Cognitive Science, 7(2), pp 155-170.
7. Guilford, J.P. 1967. *The Nature of Human Intelligence*. New York: McGraw-Hill.
8. Holyoak, K. J. and P. Thagard. (1989). *Analogical Mapping by Constraint Satisfaction*, Cognitive Science 13, pp 295-355.
9. Lerdhal, F. and Jackendoff, R. 1983. *A Generative Theory of Tonal Music*. Cambridge, Mass.: MIT Press.
10. Machado, P.; Cardoso, A.; 1997. *Model Proposal for a Constructed Artist*. In Proceedings of World Multiconference on Systemics, Cybernetics and Informatics, SCI'97. Caracas.
11. Michael, W.B.; 1977. *Cognitive and affective components of creativity in mathematics and the physical sciences*. In J.C. Stanley, W.C. George & C.H. Solano(Eds.), The gifted and the creative: A fifty-year perspective. Baltimore: Johns Hopkins University Press.
12. Pereira, F.; Grilo, C; Macedo, L; Cardoso, A.; 1997. Composing Music with CBR. In Proceedings of the First International Conference on Computational Models of Creative Cognition. Dublin.
13. Sims, K.; 1991. *Artificial Evolution for Computer Graphics*. SIGGRAPH'91, Computer Graphics, Vol. 25, Nr.4, pp. 319-328, ACM.
14. Smaill, A.; Wiggins, G.; and Harris, M. 1993. *Hierarchical Music Representation*. In Computers and the Humanities, vol. 27.
15. Veale, T. ; Keane, M. T.; 1994. *Metaphor and Memory: Symbolic and Connectionist Issues in Metaphor Comprehension*, in the Proc. of the European Conference on Artificial Intelligence Workshop on Neural and Symbolic Integration, Amsterdam

Scheduling to Reduce Uncertainty in Syntactical Music Structures

Miguel Ferrand and Amílcar Cardoso

Centro de Informática e Sistemas da Universidade de Coimbra
Polo II, Pinhal de Marrocos, 3030 Coimbra, Portugal
{mferrand,amilcar}@dei.uc.pt

Abstract. In this paper, we focus on the syntactical aspects of music representation. We look at a music score as a structured layout of events with intrinsic temporal significance and we show that important basic relations between these events can be inferred from the topology of symbol objects in a music score. Within this framework, we propose a scheduling algorithm to find consistent assignments of events to voices, in the presence of uncertain information. Based on some experimental results, we show how we may use this approach to improve the accuracy of an Optical Music Recognition system.

1 Introduction

Constraint satisfaction problems (CSP) are central to many AI applications and extensive research has been directed to develop new representations, formalisms and to improve or develop new problem solving algorithms [12]. Some of these formalisms, in particular the ones related to temporal reasoning, have been embraced by computer music research on the issues of representing time and structure in music [6,7]. This work will focus on the syntactical aspects of music. We look at a music score as a structured two-dimensional layout of symbols representing events with intrinsic temporal significance, where relative temporal constraints are inferred from topological relations between events. Allen's interval algebra [1] offered a very flexible representation for temporal events, enabling the retrieval of all possible relations between any pair of intervals. However for some specific problems, reasoning with such a representation may lead to unnecessary complexity.

In the first part of this paper we describe a syntactical representation for musical events along with some consistency properties of the model, as we outline the type problems we are addressing. Then we present a scheduling algorithm that uses this representation to find consistent event-to-stream assignments in multi-voice music excerpts in the presence of uncertain information, and we show some experimental results. Finally we discuss how we can improve the performance of an Optical Music Recognition (OMR) system, based on this approach and results.

F. Moreira de Oliveira (Ed.): SBIA'98, LNAI 1515, pp. 249-258, 1998.

2 Representing music events

Representing music with computers is not a trivial issue. Some general purpose approaches have been proposed, but authors often found it more convenient to design and adapt music representations to their specific needs, being those of music composition, analysis, or performance [2,5,7], pursuing a balance between expressiveness and computational efficiency.

As perceived by a reader, a sheet of printed music reveals a layout of graphical symbols that represent music events. These symbols may be represented in a two-dimensional diagram from where topological relations may be inferred using domain specific spatial analysis. As depicted in Fig. 1, events may appear sequentially or superimposed in a music score. The *superimposed* relation relates to the pitch axis while the *sequential* relation relates to the temporal axis of the music layout.

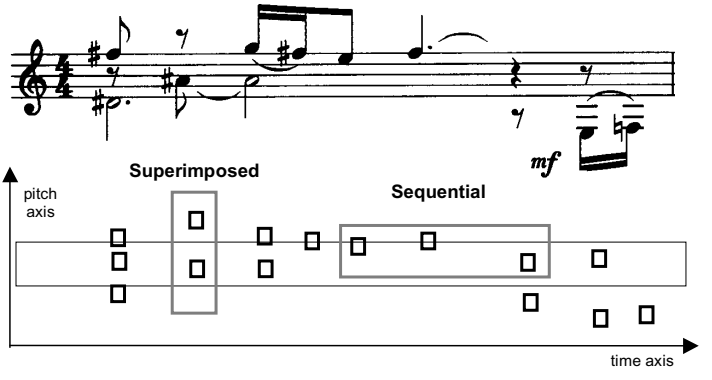


Fig. 1. Original score image and event diagram, showing superimposed and sequential events along the pitch and physical time axis

Based on these two topological relations, some important temporal properties may be inferred and defined for every pair of events:

Definition 1: Sequential events - Any two events are sequential, iif they have different starting moments along the physical time axis.

$$sequential(e_i,e_j):\ follows(e_i,e_j) \Leftrightarrow precedes(e_j,e_i)$$

Definition 2: Superimposed events - Two events are superimposed, iif they have the same temporal starting moment.

$$superimposed(e_i,e_j):\ \ above(e_i,e_j) \Leftrightarrow below(e_j,e_i)$$

The *above* and *below* relations emphasise the topology expressed in the music layout but without loosing the intrinsic temporal significance. Based on these elementary relations, we describe in the next section some properties of our model that comprise the syntax of music writing.

2.1 Streams of events

A music piece is usually composed of sequences of related notes representing monophonic lines or voices. Several of these voices or *streams*, as designated in [2], may exist concurrently in a music score. All events within a music excerpt can be associated with a unique stream. Sequential events may eventually (but not necessarily) belong to the same stream, while superimposed events never share a stream.

Property 1: All events in a stream are sequential - Given any stream of events S, then

$$\forall\, e_i, e_j \in S,\ e_i \neq e_j \Rightarrow \mathit{precedes}(e_i, e_j) \vee \mathit{follows}(e_i, e_j)$$

Since events are naturally sequentially related, streams may be represented by before-after chains of events, as suggested by Kwong [9], each pointing its successor and predecessor. Durations may then be assigned and propagated through these chains to determine the relative starting and ending moments for all events.

An important assumption about our model is that no temporal gaps may exist in streams, which means that all atomic components of a stream must be explicitly represented being either a note or a rest.

Property 2: Tightness of a stream - For any stream S of finite duration,

$$\mathit{duration}\{S\} = \sum_{e_i \in S} \mathit{duration}\{e_i\} \qquad (1)$$

Some of the properties presented earlier, may be extended to represent and relate larger cells of a music piece. Whole measures may be treated as events and may be related by preceding and succeeding relations. In multi-part pieces several superimposed staffs can be viewed as streams and may be related by above or below relations (i.e. as suggested by extended measure bars across staffs). This hierarchical nature of the representation enables us to partition large music pieces, by establishing additional constraints between the resulting smaller excepts.

3 Reasoning With the Representation

We are addressing the problem of achieving a consistent event-to-stream assignment, for a given set of events. This is a complex task and usually entails a semantic analysis of a piece. However we argue that, under some circumstances, it is possible to find solutions for this problem, based only on the syntax of music notation and on the topological relations between the events, described in the previous sections.

3.1 Consistency

Concurrent *streams* may be connected or "bonded", by *above* or *below* relations, whenever superimposed events occur (see Fig. 2). These bonds set important temporal constraints that will be used for synchronism and consistency checking during the process of assigning events to streams. This is explained by the following property:

Property 3: Stream bonds - Given two streams formed by sequential events, respectively $e_i...e_{i+p \text{ and }} e_j...e_{j+q}$, such that $\forall\ e_k,e_{k+1}$: *precedes*(e_k,e_{k+1}), then when

$$superimposed(e_i,e_j) \wedge superimposed(e_{i+p},e_{j+q}) \Rightarrow \sum_{n=i}^{+p-1} duration(e_n) = \sum_{n=j}^{J+q-1} duration(e_n) \quad (2)$$

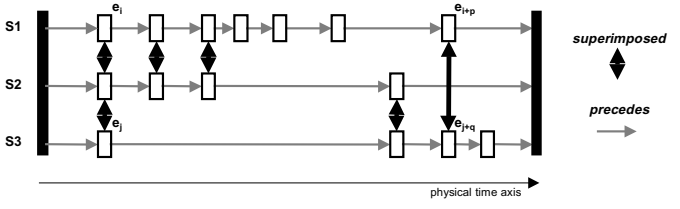


Fig. 2. Stream assignment for the excerpt of Fig. 1

Bel [2] has presented a similar consistency property to infer the missing implicit relations between events in a polymetric structure. He also demonstrated that when we have a complete set of events with all known durations, stream assignments may be solved using only precedence and simultaneity relations. We will follow this same principle making use of the topological relations as the base for stream separation.

3.2 Uncertainty about events

Within the scope of the present work, two types of uncertainty which affect events were considered. In the first case an event may have an ambiguous duration and a list of possible durations may be associated with the event. In the second case some events may not belong to any of the streams being considered, and then its presence may not be consistent within the given event set. These events will be represented just like in the first case, but one of the possible durations will be null.

If all non-null event durations are found to be inconsistent and null becomes the only remaining duration labeling, the event is neglected. We will refer to these events as "false" events.

3.3 The Scheduling Algorithm

The problem we are dealing with may be compared to the classical processor/task scheduling problem. Events will be viewed as the tasks to schedule, and streams will

be the processor's execution queues. Just like processor tasks, events have precedence constraints with respect to the remaining events. Additional complexity is introduced by events with ambiguous durations or "false" events.

The scheduling algorithm was implemented in Prolog and is described in Fig. 3.

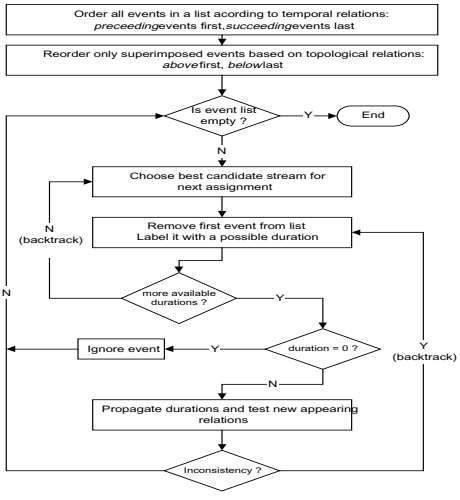


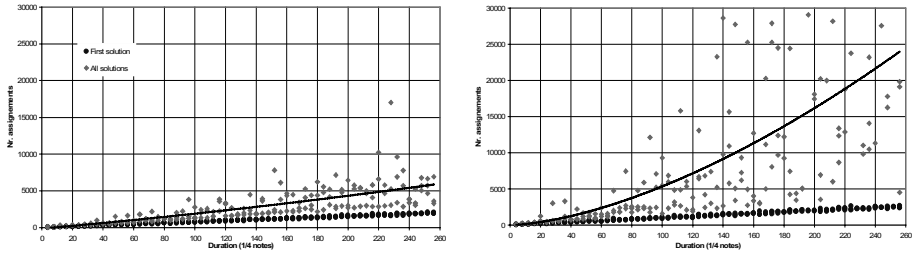
Fig. 3. The scheduling algorithm

Some heuristics are used to find the best candidate stream for each assignment: first the stream with lower current duration is picked. If two or more streams have equal current duration then conflict detection is performed between the event about to be scheduled and the last events assigned to the possible eligible streams. Finally, if all streams are equally eligible to receive the next event then the first available stream is simply chosen for assignment. All the durations available for an event are tried before assigning it to another stream. The pre-orderings described in the two pre-processing steps of the algorithm will allow most conflicts to be detected and solved locally and thus avoiding large backtracking steps. This is confirmed by the experimental results, presented in the next section.

4 Experimental Results

In order to test the behavior of the scheduling algorithm in the presence of uncertain information, we used a random event generator to create test stream excepts of varying durations. The generator was configured to produced an average of 2 events for every quarter note or the equivalent for other time division units. Ambiguous durations and "false" events were then randomly introduced in all test sets at rates ranging from 0 to 20% of the total number of events. We established for these experiments a fixed number of 3 streams.

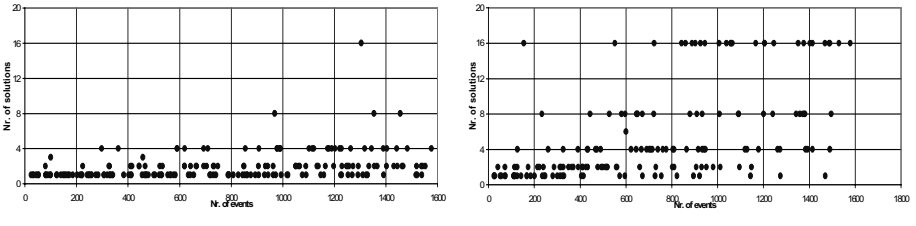
In Fig. 4 we show the variation in the number of assignments necessary to obtain the first and then all the remaining solutions, as the duration of the test samples increases. In Fig. 4 a) and b) event data was affected with ambiguous durations at rates of 10% and 20% respectively. This 10% increase, had a considerable impact on the number of assignments required to find all consistent solutions, especially for the samples with longer duration. To retrieve all solutions the algorithm terminates in polynomial-space complexity. In spite of this, a first consistent solution is found in a number of assignments nearly proportional to the duration of the test sample.



a) 10% ambiguous events b) 20% ambiguous events

Fig. 4. Number of event assignments vs duration for 3-stream excerpts

It can be seen that in both cases, for durations below 60 ¼ notes, the number of assignments required to search the whole solution space approaches the one needed to obtain the first solution. This is mainly due to the small number of possible solutions within the given set of events. In Fig. 5 we plot the number of solutions produced by the same test sets of Fig. 4 a) and b); this time the number of events of each excerpt is presented along the horizontal axis.



a) 10% ambiguous events b) 20% ambiguous events

Fig. 5. Number of solutions produced vs number of events for 3-stream excerpts

As expected, there is an average reduction in the number of solutions for the sets with lower rate of ambiguous information. For the sake of legibility some very occasional larger number of solutions were truncated in the charts. Most additional solutions result from the existence of pairs of events that can alternate durations consistently within the same stream, which explains the dominance of even numbers of multiple solutions.

5 Improving the recognition rate of an OMR system

All automatic recognition systems are evaluated by their rate of successful recognition. Usually most of the missing or incorrectly recognised information, results from the poor quality of the original scanned image or from loss of information or aliasing resulting from image segmentation and processing phases. The high degree of data abstraction carried by these systems, transforming image pixels into symbolic data, makes it difficult to supply all the contextual details that surround missing or misprinted information, to the later stages of recognition. The human reader is usually able to detect these inconsistencies, just by looking at the page and analysing the surrounding information.

In order to produce a final output some decisions must be taken by the recognition algorithms that often make use of threshold numbers to rate and classify graphical primitives or symbols [4]. These thresholds are usually derived for every new score image, based on various music notation features [3]. However, wrong threshold decisions are often difficult to revert so it seems crucial to give the OMR system the ability to describe the degree of confidence of the recognition decisions, and allowing re-classification of graphical features during the remaining recognition process [8].

In Fig. 6 we illustrate how this may be achieved. In the first case (Fig. 6-a) as a result of a single threshold decision, the symbol is misrecognized although the parameter value being checked by the recognition algorithm is barely beyond the line of decision. In the second case (Fig. 6-b), two confidence thresholds were defined for each symbol; since the parameter now stands in the undefined zone, the system will consider two possible symbols, that will be used later to evaluate alternative scenarios.

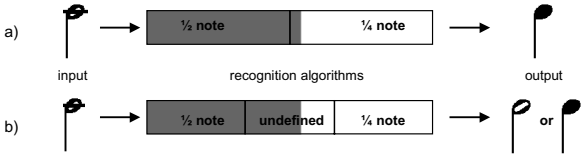


Fig. 6. Uncertain information generated by recognition algorithms

The same principle is used to indicate "false" events; if a symbol is not confidently recognized, the system labels it with an additional null duration indicating that the remaining labels might not be trustable.

RIEM [10], an OMR system that was developed prior to this work, will be our source of information. It will generate sets of events with associated attributes, during the recognition process. Events are described as predicates of the type:

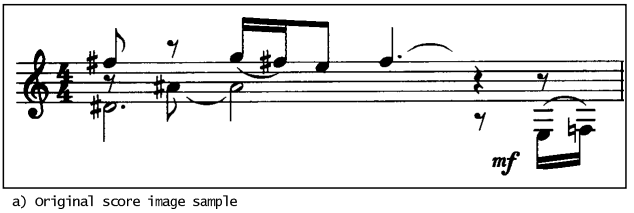
```
event(id, x, y, durations, type, beam, stem)
```

id - unique event identification number
x,y - spatial coordinates of event graphical center
durations - list of possible event durations
type - type of event (note or rest)
beam - note beam number (if note is beamed with other notes)
stem - indicates up/down direction of note stem (if any)

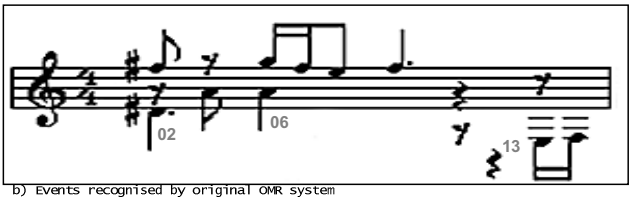
The type, stem and beam fields are used to further reconstruct the score image. The beam field is also used for consistency checking since beamed events should in principle be assigned to the same stream. The two-dimensional space of the music score image will provide the necessary information to pre-process event information: the relations inferred from the relative positioning of events are extracted with the aid of proximity comparisons, using image dependent measurement units (ex: distance between two consecutive staff lines or average width of note heads).

5.1 An example

In Fig. 7 we follow a small example of the behavior of the improved version of RIEM. In Fig. 7-a we have the original image sample read by RIEM. The results of the preliminary recognition process are illustrated in Fig. 7-b, and they correspond to the output of the original recognition algorithms.



a) Original score image sample



b) Events recognised by original OMR system

```
event(01,0198,0139,p,[0.50],0, 0).
event(02,0202,0197,n,[1.50,3.00],0,-1).
event(03,0208,0109,n,[0.50],0,+1).
event(04,0328,0157,n,[0.50],0,-1).
event(05,0332,0071,p,[0.50],0, 0).
event(06,0456,0159,n,[2.00,1.00],0,-1).
event(07,0462,0099,n,[0.25],2,+1).
event(08,0544,0107,n,[0.25],2,+1).
event(09,0630,0119,n,[0.50],2,+1).
event(10,0758,0109,n,[1.50],0,+1).
event(11,0942,0217,p,[0.50],0, 0).
event(12,0944,0141,p,[1.00],0, 0).
event(13,1006,0305,p,[1.00,0],0,0).
event(14,1068,0255,n,[0.25],3,-1).
event(15,1074,0135,p,[0.50],0, 0).
event(16,1158,0243,n,[0.25],3,-1).
```

c) Event descriptions with uncertain information generated by the new OMR system

```
S1: [--03--][--05--][07][08][--09--][----------10----------][--15--]
S2: [--01--][--04--][--------------06--------------][------12------]
S3: [----------------------02----------------------][--11--][14][16]
```

d) Stream assignment diagram of unique solution for the generated event set

Fig. 7. Processing stages of the improved RIEM system

In Fig. 7-c we have the list of event descriptions generated by the new recognition algorithms. Events number 2 and 6 have ambiguous durations and a false event labeled with number 13 was generated with a null duration. This list of event predicates is then processed by the scheduling algorithm, already presented in the previous section, to perform the stream separation. In this small example a single solution was consistent with the given event set. The corresponding stream diagram (Fig. 7-d) shows that both events number 2 and 6 had their durations corrected, and event number 13 was rejected from the final solution. Naturally not all cases may be completely corrected, like this one. The worst cases are those where no consistent solution is found by the scheduling algorithm; the system will then have to recall the preliminary recognition results, to produce the final output.

6 Conclusions and Future Work

In this paper we have focused on the syntactical aspects of music representation. We have presented a scheduling algorithm to achieve stream separation when uncertain information is associated with the events. It was shown that for relatively small sets of events, it is feasible to integrate this algorithm in an OMR system, to improve its effective recognition rate by correcting durations of events or by removing "false" events that result from misrecognized symbols.

It is unlikely that more than a full score line is ever presented for processing so the number of events to schedule each time is fairly reduced. Assuming a realistic rate (say below 15%) of events affected by uncertain durations, the number of possible solutions may be kept low and so will the complexity of the scheduling process. In the cases where more than one solution is produced, it may be possible to discriminate the real solution with the aid of some domain dependant heuristics.

The complexity of this problem is introduced mainly by the uncertain information, and possible absent events of unknown duration. For the present being we have ignored the existence of missing events but it would be important to try to guess and locate these ocurrences. We this in mind, a new formalization for this problem, and a new problem-solver based on an Assumption-Based Truth Maintenance System (ATMS) is currently under development. We hope to diagnose events responsible for inconsistencies more efficiently, while making assumptions about possible absent events.

With this work we expect to contribute to the integration of Artificial Intelligence techniques in traditional automatic recognition systems. We find this a challenging task since some compromise must exist between accuracy and time usefulness of the solutions provided. Also, this requires that such systems be prepared to handle new data representations, and most important of all, be able to recognize and describe uncertainty.

Acknowledgements

We would like to thank CISUC (Centro de Informática e Sistemas da Universidade de Coimbra) for supporting our research.

References

1. Allen, James F.: "Maintaining Knowledge about Temporal Intervals", *Communications of the ACM*, 26, 832-843, 1983.
2. Bel, B.: "Symbolic and Sonic Representations of Sound-Object Structures", in *Understanding Music with AI: Perspectives on Music Cognition*, MIT press, 64-109, 1992.
3. Carter, Nicholas P.: "A Generalized Approach to Automatic Music Scores", CCRMA Report, Stanford University, December, 1993.
4. Fujinaga, Ichiro: "Optical Music Recognition using projections", MA dissertation, McGill University, Montreal, 1988.
5. Harris, Mitch; Smail, Alan; Wiggins, Graint: "Representing Music Symbolically", Research paper 562, Dept. of Artificial Intelligence, Edinburgh, 1991.
6. Honing, Henkjan: "Issues in the Representation of Time and Structure in Music", *Contemporary Music Review*, 9, 221-239, 1993.
7. Huron, David: "Design Principles in Computer Based Music Representation", *Computer Representations and Models in Music*, Academic Press, 4-38, 1992.
8. Kato, Hirokazo; Inokuchi, Seiji: "A recognition system for printed piano music using musical knowledge and constraints". in Structured Document Image Analysis, ed Springer-Verlag, pp. 435-455. Eds. H.S. Baird, H. Bunke, K.Yamamoto, 1992.
9. Kwong, Charlie: "Representing time", *Artificial Intelligence*, A&W, 189-205, 1987.
10. Leite, João; Ferrand, Miguel; Cardoso, Amilcar: "RIEM – a system for Recognition and Interpretation of Music Writing" (in portuguese), internal report RI-DEI-001-98, Dept. Engenharia Informatica, Universidade de Coimbra, 1998.
11. van BeeK, Peter: "Reasoning about Qualitative Temporal Information", *Proceedings of AAAI*, 728-734, 1991.
12. Valdés-Péres, R.E.: "The Satisfiability of Temporal Constraint Networks", *Proceedings of the 6th National Conference on A.I.*, 256-260, 1987.

Author Index