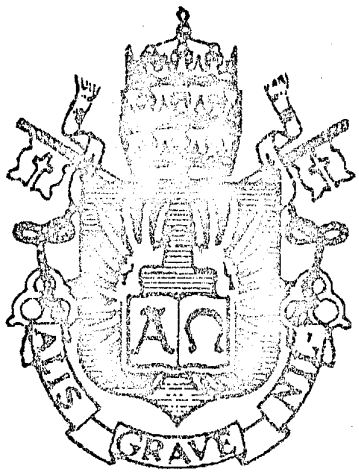

PUC

Monografias em Ciência da Computação
nº 2/92

ACCORD – Máquina Básica

Lorenzo F. G. G. M. Ridolfi
Hugo Fuks
Daniel Schwabe

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO
RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22453
RIO DE JANEIRO - BRASIL

PUC RIO - DEPARTAMENTO DE INFORMÁTICA

Monografias em Ciência da Computação, Nº 2/92
Editor: Carlos J. P. Lucena Março, 1992

ACCORD – Máquina Básica *

Lorenzo F. G. G. M. Ridolfi
Hugo Fuks
Daniel Schwabe

* Este trabalho foi patrocinado pela Secretaria de Ciência e Tecnologia da Presidência da República Federativa do Brasil.

In charge of publications:
Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC Rio — Departamento de Informática
Rua Marquês de São Vicente,225 — Gávea
22454970 — Rio de Janeiro, RJ
Brasil
Tel. +55-21-529 9386 Telex +55-21-31048 Fax +55-21-511 5645
E-mail: rosane@inf.puc-rio.br
techrep@inf.puc-rio.br (for publications only)

Resumo

Este trabalho apresenta a máquina básica do ACCORD – uma estrutura para representação de diálogos usando compromisso – que incorpora o DAC (Dialogue Action Component) e o CC (Commitment Calculus). O DAC é responsável pelo controle do diálogo, bem como pela manipulação e armazenamento de compromissos. O CC trata das consequências de se possuir compromissos, inferindo novos compromissos a partir dos compromissos assumidos por um participante de um diálogo. A máquina básica foi desenvolvida como um sistema baseado em regras, implementada em prolog.

Palavras-chave

Diálogo, Compromisso, Sistemas Baseados em Regras, Negociação, Prolog

Abstract

This work presents the basic machine of ACCORD – a framework for dialogue representation using commitment. ACCORD comprises a Dialogue Action Component (DAC) and a Commitment Calculus (CC). DAC is involved with dialogue control and with commitment storage and manipulation. CC deals with the consequences of having commitments, inferring new commitments from the ones assumed as the dialogue evolves. The basic machine was developed within a rule based system, implemented in prolog.

Keywords

Dialogue, Commitment, Rule Based Systems, Negotiation, Prolog

ACCORD - Máquina Básica

Lorenzo F. G. G. M. Ridolfi
Hugo Fuks
Daniel Schwabe

hugo@inf.puc-rio.br
Departamento de Informática
Pontifícia Universidade Católica do Rio de Janeiro
Rua Marquês de São Vicente 225
CEP 22453 - RJ - Brasil

Resumo

Este trabalho apresenta a máquina básica do ACCORD – uma estrutura para representação de diálogos usando compromisso – que incorpora o DAC (Dialogue Action Component) e o CC (Commitment Calculus). O DAC é responsável pelo controle do diálogo, bem como pela manipulação e armazenamento de compromissos. O CC trata das consequências de se possuir compromissos, inferindo novos compromissos a partir dos compromissos assumidos por um participante de um diálogo. A sua máquina básica foi desenvolvida como um sistema baseado em regras, implementada em prolog.

Palavras-Chave

Diálogo, Compromisso, Sistemas Baseados em Regras, Negociação, Prolog

1. Introdução

A negociação está presente em vários aspectos do nosso cotidiano. Normalmente, a realização de tarefas que envolvam mais de um pessoa é feita através de acordos, compromissos e consentimentos. Os relacionamentos entre grupos de pessoas ou grupos de sistemas de computadores podem ser conduzidos a partir da estruturação e coordenação dos aspectos interacionais da linguagem. Pessoas falando com pessoas é sem dúvidas a melhor maneira de estabelecer um consenso comum, trabalhar em conjunto, negociar contratos ou resolver conflitos.

Nossa proposta é fornecer uma estrutura para representação de sistemas de diálogo que seja capaz de capturar aspectos de negociação presentes em trabalho cooperativo. Uma representação baseada em lógica (ação) modal foi escolhida por sua capacidade de expressar ações, o que é fundamental para a explicitação dos aspectos interacionais do processo de negociação. Cabe lembrar que não estamos interessados no uso de fala ou de linguagem natural, apenas na definição de primitivas de conversação que guiem o processo de negociação.

A fração de conhecimento do diálogo de cada participante foi modelada através da noção de compromisso. Para se trabalhar cooperativamente é necessário se estabelecer compromissos, de forma que seja possível antecipar as ações de outros e coordená-las com as nossas ações. A noção de compromisso incorpora os aspectos visíveis envolvidos no processo de negociação.

Dentre as várias categorias de trabalho cooperativo (Ellis & Gibbs 91), nos voltamos ao suporte ao trabalho em grupo - *Teamwork* - associado a projetos de *software* (Hahn, Jarke & Rose 91). É nessa categoria que se insere a *Researh Agenda on Software Design - RASD* (Lucena, Leite, Schwabe & Fuks 91), que tem como uma de suas metas a incorporação de aspectos cooperativos em metodologias de desenvolvimento de sistemas de software.

O trabalho aqui apresentado se insere como um dos tópicos de pesquisa específicos relacionados na *RASD*, tratando da implementação da Máquina Básica do *ACCORD - A Framework for Dialogue Representation using Commitment* (Fuks 91). A implementação, em prolog, segue as idéias propostas para sistemas baseados em regras presentes em sistemas de produção. O seu objetivo é constituir um ambiente onde seja possível fazer investigações sobre o ACCORD, bem como fornecer uma interface para aplicações que desejam utilizar os conceitos da estrutura.

Esta monografia está organizada da seguinte forma. Na seção 2 apresentamos os conceitos básicos do ACCORD. A seção 3 discute os sistemas baseados em regras. A seção 4 descreve a implementação da Máquina Básica e a seção 5 a sua interface. Na seção 6 discutimos algumas extensões à Máquina Básica. Na seção 7 apresentamos as nossas conclusões. Por fim, o código de uma versão inicial do protótipo da Máquina Básica encontra-se no Apêndice.

2. Introdução ao Formalismo

Esta seção discute os conceitos básicos do ACCORD, uma estrutura para representação de sistemas de diálogo que usa a noção de compromisso para capturar alguns dos aspectos de negociação característicos de trabalho cooperativo. O leitor interessado em uma visão mais detalhada desta estrutura é referenciado para (Fuks 91).

ACCORD introduz um conjunto de formalismos que representam um diálogo através de compromissos assumidos por seus participantes no decorrer do diálogo. Cada participante possui uma carteira de compromissos que armazena os seus compromissos públicos. Esta carteira é vista por todos os participantes do diálogo. Na medida em que o diálogo evolui, o conteúdo de cada carteira de compromissos é modificado, sendo que a única maneira de se alterar o seu conteúdo é através da evolução do diálogo. Cada participante possui uma área de trabalho. Ao contrário de uma carteira de compromissos, a área de trabalho é privada, sendo usada principalmente para rascunho. O seu conteúdo é composto por sentenças particulares ao participante, das quais o mesmo não possui compromisso público.

Esta estrutura, o ACCORD, está dividido em duas grandes partes: o DAC *(Dialogue Action Component)* e o CC *(Commitment Calculus)*. O DAC encarrega-se do processo do diálogo, gerenciando as inserções e remoções de compromissos em carteiras de compromisso. O CC, por sua vez, infere os resultados do processo de diálogo, ou seja, infere novos compromissos a partir de compromsissos armazenados em carteiras de compromissos.

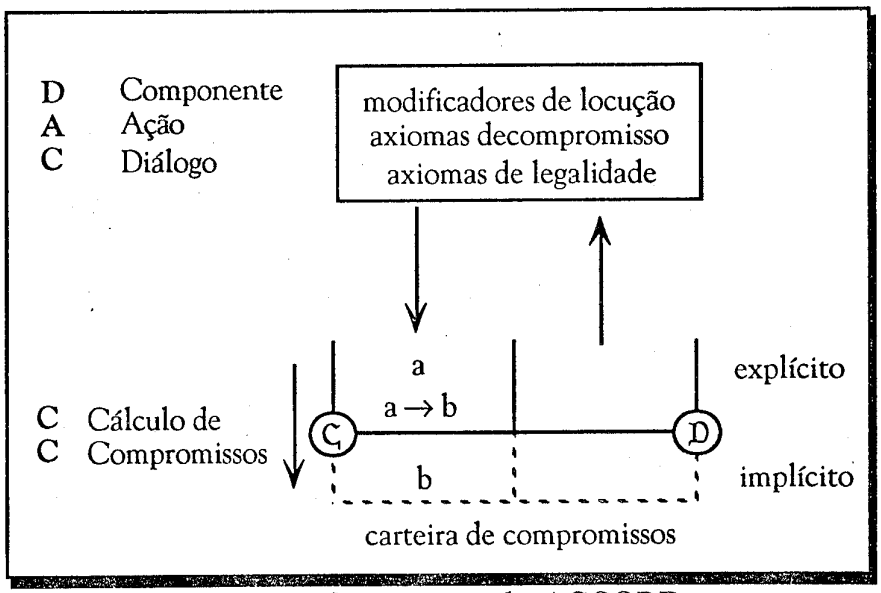

Figura 1. Organização do ACCORD

2.1. Compromisso

A noção de compromisso é central ao ACCORD, sendo empregada para representar a fração de informação de um participante sobre o diálogo. Isto é feito a partir de carteiras de compromissos, onde são armazenados os compromissos adquiridos durante o diálogo. A carteira de compromissos de cada participante é dividida em duas partes. A primeira armazena os seus compromissos, enquanto que a outra os seus compromissos que foram revogados. Esta última contém os compromissos com as quais o participante não está compromissado. As carteiras de compromissos possuem esta estrutura pois esta em muito se assemelham com a natureza de três estados da nossa noção de compromisso, que são: estar compromissado com determinada situação, não estar compromissado com determinada situação e desconhecer a existência de tal compromisso.

A existência de um compromisso sobre uma determinada situação *(state of affairs)* pressiona o comportamento do participante para que seja consistente com esta situação. Ou seja, os compromissos de um participante impõem restrições ao seu comportamento, restringindo dessa forma a sua liberdade de ação. Podemos dizer ainda que compromisso estabelece um conjunto de modos de ação, de maneira consistente com a informação disponível e o papel exercido pelo participante em seu meio. Entretanto, não estamos interessados em fornecer um significado definitivo para a noção de compromisso, apenas prover uma estrutura na qual seja possível operá-lo.

2.2. Locuções

Um participante contribui para um diálogo a partir de locuções. Uma locução é uma unidade de diálogo. Cada vez que um participante *fala*, este pronunciou uma locução. Esta consiste de um *modificador de locução* aplicada a uma sentença. As sentenças são construídas em uma linguagem proposicional que inclui negação, condicional, disjunção e conjunção. Os modificadores de locução conferem significado para uma sentença, conforme indica a tabela abaixo:

Modificador	Significado
Asserts	afirmação, declaração, defesa
Justifies	justificação, comprovação
Denials	negação, recusa, rejeição
Resolve	resolução, solução
Questions	questionamento, dúvida
Why	desafio, objeção, reclamação
Withdrawals	retração, revogação

2.3. Eventos de Diálogo

Cada contribuição de um participante em um diálogo dá origem a um *evento de diálogo*. Um diálogo é armazenado através de um *registro de eventos* que contém todos as locuções de um diálogo na ordem em que estes ocorreram. O registro de eventos guarda ainda a evolução do diálogo, já que é uma estrutura ordenada.

2.4. DAC - Dialogue Action Component

Na estrutura do ACCORD, o DAC é responsável pela inserção e remoção de compromissos nas carteiras de compromisso. Para capturar este mecanismo, definiu-se um sistema axiomático baseado em lógica modal (Goldblatt 82, Maibaum 87). Os axiomas do

DAC estão divididos em dois conjuntos distintos: axiomas de compromisso e axiomas de legalidade.

Os axiomas de compromisso definem as mudanças nas carteiras de compromisso e a validade do estado do diálogo após a elocução de uma sentença especifica (pós-condição). Um axioma de compromisso essencialmente possui este formato:

$$\text{Pré} \rightarrow [\text{AtoB, Locução}]\ \text{Pós}$$

Este é lido como: *Antes da parte* B *pronunciar* Locução *para a parte* B, *a pré-condição* Pré *é válida, enquanto que a pós-condição* Pós *é válida após a mesma locução.*

Os axiomas de legalidade, por sua vez, definem a ordem correta das sentenças em um diálogo. A sua forma geral é:

$$\text{Pré} \rightarrow [\text{AtoB, Locução}][\text{BtoA, Locução'}]\text{Pós}$$

Um axioma de legalidade é lido como: *Antes da parte* A *pronunciar* Locução *para a parte* B *e* B *pronunciar* Locução' *para a parte* A, *a pré-condição* Pré *é válida, enquanto que a pós-condição* Pós *é válida após as locuções.*

2.5. Cálculo de Compromissos

O cálculo de compromissos trata das conseqüências de se possuir compromissos nas carteiras de compromissos. A natureza aberta das carteiras de compromisso permite a observação de seu conteúdo, dando margem aos participantes anteciparem as ações dos outros. Esta característica permite também que um participante coordene as suas ações com as do outro participante.

O cálculo de compromissos é descrito por regras de inferência, a partir das quais novos compromissos podem ser inferidos do conteúdo das carteiras de compromisso. Estas regras visam a preservação da consistência das carteiras de compromisso.

A semântica do cálculo é baseada na noção de pertinência de compromissos em carteiras de compromissos. Isto é, a veracidade ou falsidade de suas regras é definida a partir da presença ou ausência de compromissos nas carteiras de compromisso. A corretude deste cálculo é um resultado trivial.

O cálculo de compromissos permite apenas uma visão estática de compromisso. Por esta razão, o cálculo encontra-se embutido no sistema de diálogo que introduz a visão dinâmica com a qual os compromissos são estabelecidos ou retirados.

3. Sistemas Baseados em Regras

Sistemas baseados em regras têm a sua origem fundamentada nas primeiras propostas dos sistemas de produção. Estas propostas visavam encapsular fragmentos primitivos do comportamento de processamento de informação em sistemas de simulação. Devido ao sucesso destas abordagens, observou-se que os sistemas de produção poderiam ser empregados como mecanismos gerais de programação, com todo o poder de uma máquina de Turing.

Sistemas de produção permitem que a representação do conhecimento seja feita de forma uniforme e modular. Isto é particularmente importante no desenvolvimento de programas que manipulem representações de conhecimento próprias, o que é o caso do ACCORD. O conhecimento encapsulado é completamente explícito nestes sistemas, auxiliando a verificação experimental de teorias cuja representação é fundamentada em sistemas de produção.

3.1. Sistemas de Produção Puros

Um sistema de produção puro consiste de três componentes básicos: um conjunto de regras, um banco de dados e um interpretador para as regras. Em uma concepção simplificada, uma regra é dividida em duas partes: lado direito e lado esquerdo. O conjunto de regras possui uma ordem total e o banco de dados é simplesmente um conjunto de símbolos. Nesta concepção simples, o interpretador tem a função de percorrer o lado esquerdo de cada regra do conjunto de regras até encontrar uma que se case *(match)* ao banco de dados. Os símbolos do banco de dados que foram casados são então substituídos pelo lado direito da regra. Com isto, uma regra pode ser vista como uma expressão condicional e a invocação de regras nada mais é que uma seqüência de ações encadeadas por *modus ponens*.

3.1. Sistemas de Produção em Prolog

O prolog é uma linguagem de programação que herdou seus construtores da lógica. Um programa em prolog, ou um programa lógico, é um conjunto de axiomas, ou regras que definem relacionamento entre símbolos, estes últimos armazenados sob a forma de fatos um banco de dados. Um interpretador realiza a computação de um programa lógico, deduzindo por *modus ponens* as conseqüências do programa. Estas características fazem do prolog muito semelhante a um sistema de produção, sendo a opção natural para a implementação destes sistemas.

Com isto, a implementação de um sistema de produção em prolog consiste em fazer um meta-interpretador prolog que possua o comportamento do interpretador do sistema de produção. As regras do sistemas de produção por sua vez são definidas a partir de meta-regras prolog . Finalmente, o banco de dados do sistema de produção é implementado em termos de fatos do prolog.

4. A Máquina Básica

Esta seção apresenta a máquina básica do ACCORD que incorpora o DAC e o cálculo de compromissos. Muitos dos conceitos definidos em Fuks (91), tais como registro de eventos, carteira de compromissos e os axiomas de legalidade e axiomas de compromisso são reapresentados sob a ótica de um sistema baseado em regras (vide as sub-seções 4.3 e 4.4). Com isto, os componentes do ACCORD tornam-se fatos e regras manipulados por interpretadores na máquina básica (fatos, meta-regras e meta-interpretador prolog). Durante a implementação algumas pequenas extensões foram incorporadas a proposta do ACCORD. Estas alterações encontram-se devidamente realçadas ao longo desta seção.

A máquina básica foi projetada de forma a ser extensível, uma vez que grande parte desta é implementada por intermédio de interpretadores específicos, seguindo a concepção de sistemas de produção. Esta escolha permite que mudanças em diversos de seus conceitos – regras de compromisso, de legalidade e de derivação - sejam realizadas sem que se tenha que alterar a implementação da máquina básica, fortemente baseada nos interpretadores.

4.1. Eventos de Diálogo

Um diálogo é composto por eventos de diálogo que, quando armazenados, compõem o registro de eventos do diálogo. Cada locução *dita* por um participante corresponde a um evento de diálogo. Com isto, cada evento de diálogo segue a seguinte estrutura:

$$((Origem,\ Ordem,\ Diálogo),\ (Modificador,\ [S_1,\ \ldots,\ S_n]))$$

onde *Origem* é o participante que inseriu o evento, *Ordem* indica a ordem deste evento, *Diálogo* é o diálogo ao qual o evento pertence, *Modificador* é o modificador da locução e $[S_1, \ldots, S_n]$ são as sentenças. Note que um evento está divido em duas partes: a primeira parte

contém as informações de controle *(Origem, Diálogo)* enquanto que *(Modificador, [S_1, ..., S_n])* forma a locução.

Esta estrutura, na verdade um fato prolog, é bastante semelhante a definida originalmente em Fuks (91). A diferença é a presença do item *Diálogo*, este necessário pois a máquina básica é capaz de gerenciar diversos diálogos de dois participantes concomitantes. Vide a subseção 4.6.2, para maiores detalhes.

4.2. Carteiras de Compromissos

Cada participante de um diálogo possui um carteira de compromissos que armazena os compromissos adquiridos ao longo da evolução do diálogo. As carteiras de compromissos são estruturas que não podem ser manipuladas diretamente, não havendo, portanto, operações que as atualizem explicitamente. O conteúdo de um carteira de compromissos é resultado do diálogo, somente sendo atualizado a partir de eventos de diálogos que, estes sim, têm o poder de mudar o conteúdo de carteiras de compromisso.

Cada entrada compromisso em um carteira de compromissos é armazenado na seguinte estrutura:

(Participante, Diálogo, Origem, Compromisso)

onde *Participante* indica o participante que possui o compromisso, *Diálogo* identifica o diálogo que deu origem ao compromisso, *Origem* é o participante que, em um evento de diálogo, adicionou este compromisso e, por fim, *Compromisso* é o compromisso estabelecido.

Da mesma forma do que no caso dos eventos de diálogo, a representação de um compromisso no DAC é bastante semelhante a original descrita em Fuks (91). A representação de um compromisso na máquina básica é uma adaptação da definição original. Apenas a sua sintaxe é modificada para adequar-se ao prolog, facilitando a implementação dos interpretadores. Um compromisso na máquina básica possui adicionalmente o campo *Diálogo*, necessário para o suporte a diversos diálogos concomitantes.

4.3. Regras de Legalidade

A implementação em prolog dos axiomas de legalidade, as regras de legalidade, estabelecem regras de etiqueta que governam a *forma* da interação. As regras de legalidade fornecem meios para a manutenção de diálogos legais. Na verdade, as regras de legalidade definem uma máquina de estados sobre os eventos de diálogo - mais especificamente sobre os modificadores de locução inseridos nestes, indicando as seqüências válidas de eventos de diálogo. Cada regra de legalidade possui a seguinte estrutura:

(Modificador-1, Modificador)

onde *Modificador-1* é o modificador de locução do evento anterior e *Modificador* é o modificador do evento atual. Com isto, um evento de diálogo que contenha um modificador de locução *Modificador* só será válido se o evento anterior contiver um modificador de locução *Modificador-1*.

A representação para os regras de legalidade do DAC, quando comparada àquela apresentada em Fuks (91), é bastante simples, uma vez que é uma versão reduzida da mesma. A representação em questão foi escolhida justamente pela sua simplicidade e pelo fato de ser suficiente para a implementação do interpretador para este tipo de regra.

4.4. Regras de Compromisso

Os regras de compromisso definem como as carteiras de compromissos são atualizadas a partir de eventos de diálogo ou as situações de erro decorrentes de eventos de diálogo mal formulados. Cada regra possui uma pré-condição e uma pós-condição. Se a pré-condição de

uma regra for satisfeita, a pós-condição da regra é aplicada. A pós-condição pode indicar uma situação de erro ou as alterações nas carteiras de compromisso decorrentes do proferimento de uma locução. Estas regras são na verdade meta-regras prolog, cuja validação é feita a partir de um meta-interpretador específico. Um regra de compromisso possui a seguinte estrutura:

(Pré, (Modificador, Locução), Pós)

onde *Pré* é a pré-condição do regra, *(Modificador, Locução)* compõem uma sentença e, por fim, *Pós* é a pós-condição da regra.

Mais precisamente, o processamento das regras de compromisso é feito da seguinte forma. O interpretador procura todas as regras de compromisso que possuam o campo *Modificador* igual ao modificador de locução presente na locução do evento de diálogo proferida. Feito isto, são processadas as regras cuja pós-condição indique uma situação de erro. Se alguma pré-condição de uma regra que indique erro for satisfeita, o evento é rejeitado.

Em seguida, são processadas as regras que regem a atualização das carteiras de compromisso, o que é feito em duas etapas. A primeira etapa processa as regras que possuam pré-condição não vazia. O interpretador procura uma regra cuja pré-condição seja satisfeita, ou seja, que existam compromissos na carteira de compromissos dos participantes que casem *(match)* com a pré-condição. Uma vez satisfeita a pré-condição, os compromissos que casaram com a pré-condição são retirados das carteiras de compromisso dos participantes. No último passo, o interpretador aplica a pós-condições da regra que especifica os compromissos a serem inseridos nas carteiras de compromisso dos participantes.

Se nenhuma regra que possua pré-condição não vazia for aplicada, o interpretador emprega as regras que possuem pré-condição vazia, adicionando os compromissos presentes em sua pós-condição. Finalmente, se nenhuma regra de compromisso satisfizer as pré-condições, o evento de diálogo é rejeitado.

4.5. Cálculo de Compromissos

A estrutura escolhida para uma regra de derivação do cálculo de compromissos é:

(Expressão-Exp, Expressão-Imp)

onde *Expressão-Exp* é uma expressão existente (compromisso explícito) no carteira de compromissos e *Expressão-Imp* é a expressão implícita derivada (compromisso implícito).

A sintaxe utilizada para as expressões de uma regra de derivação é ligeiramente diferente da presente nas expressões de uma locução. A diferença está na presença de variáveis nas regras de derivação, justamente para que estas possam ser instanciadas nos termos das expressões de locuções.

4.6. Interpretadores

Grande parte da funcionalidade da máquina básica foi implementada através de interpretadores específicos. A validação das sentenças que compõem o diálogo, bem como a validação das regras de legalidade, regras de compromisso e o cálculo de compromissos são realizados por interpretadores específicos. A interpretação das sentenças é feita através de cláusulas DCG, as regras de legalidade através de uma máquina de estados bastante simples. Já os regras e o cálculo de compromisso, devido a sua complexidade, requerem meta-interpretadores específicos para a sua resolução, uma vez que fogem dos mecanismos tradicionais – unificação e back-tracking – do prolog.

A escolha de meta-interpretadores como mecanismo para a implementação dos regras e do cálculo de compromisso oferece liberdade ao estudo e verificação das regras de derivação

presentes no cálculo, bem como das regras de legalidade e das regras de compromisso. ACCORD é um ambiente experimental que deve suportar a investigação de novos regras de legalidade, compromisso ou de derivação, bem como a alteração dos existentes.

4.6.1. Analisador Sintático de Sentenças

A validação sintática de sentenças é feita através de um interpretador que utiliza cláusulas DCG para a sua implementação. Este tipo de cláusula foi escolhido devido a sua simplicidade, pois com este mecanismo, o analisador sintático resume-se a uma seqüência de cláusulas DCG que muito se assemelham às expressões BNF que definem as sentenças.

O efeito colateral da utilização de cláusulas DCG é sua grande ineficiência. Um interpretador baseado nestas cláusulas, juntamente com o mecanismo de unificação do prolog, é realizado de forma não determinística, a partir de tentativas de unificação feitas por intermédio de *back-tracking*. Na versão atual da máquina básica, grande parte do tempo de processamento é gasto no *matching* de cláusulas DCG. Esta abordagem singela seguramente deve ser revista em futuras extensões ou em novas versões desta implementação.

4.6.2. Máquina de Estados para Regras de Legalidade

O projeto atual do ACCORD está limitado a diálogos de dois participantes. Com isto, inclusive a representação para os regras de legalidade responsáveis pela manutenção da *etiqueta* do diálogo tornam-se simples. Neste caso, uma máquina de estados, que nada mais é do que um interpretador simples, definida sobre os modificadores de locução é suficiente. Em futuras extensões do ACCORD onde hajam diálogos com mais de dois participantes, esta abordagem deverá ser revista, já que métodos mais complexos para controle das regras de legalidade devem ser necessários.

4.6.3. Meta-Interpretador para Regras de Compromisso

As regras de compromisso definem a atualização dos carteiras de compromisso face a um evento de diálogo. Estes regras são checados por um meta-interpretador específico que se encarrega de validar as pré-condições e atualizar os carteiras de compromisso conforme prescrito nas pós-condições dos regras.

4.6.4. Meta-Interpretador para Cálculo de Compromissos

O meta-interpretador que implementa o cálculo de compromissos trata de aplicar as regras de derivação de compromissos especificadas no ACCORD. Este interpretador tenta instanciar compromissos presentes em carteiras de compromisso com a primeira parte de uma regra de derivação. Havendo instanciação, um novo compromisso é criado a partir da segunda parte da regra de derivação instanciada.

Na especificação original do ACCORD, o cálculo de compromissos só é usado a partir de sentenças que contenham o modificador de locução *resolve*. Na máquina básica, entretanto, o cálculo de compromissos também atua sobre a área de trabalho, permitindo que novos compromissos sejam inferidos sem a necessidade da elocução de uma sentença. Com isto, o cálculo de compromissos pode ser usado sem a necessidade da elocução de uma sentença *resolve*, o que resultaria em alterações na carteira de compromissos e no registro de eventos.

Esta extensão, sob a ótica de um sistema de diálogos, permite que um participante desenvolva uma linha de raciocínio sem a necessidade de expô-la através dos mecanismos de elocução de sentenças. Ou seja, o participante tem a liberdade de pensar em silêncio.

4.7. Área de Trabalho

Ao contrário de um carteira de compromissos, que é visível para todos os participantes do diálogo, a área de trabalho é restrita a um participante. Esta área contém anotações e fatos particulares, aos quais os outros participantes não tem acesso. Idealmente, podemos dizer que está área armazena todo o conhecimento do participante sobre o domínio no qual o diálogo se insere.

A área de trabalho é usada principalmente como rascunho, onde o participante pode fazer ensaios sobre os compromissos de um diálogo. Nesta área, o participante pode manipular compromissos sem alterar os seus compromissos *verdadeiros* (os da carteira de compromisso, assumidos através de diálogo) com os outros participantes. Em casos onde o participante toma parte em mais de um diálogo, a área privada serve ainda como um *clipboard*, permitindo que o usuário reúna conceitos (compromissos) dos diversos diálogos de que participa. Finalmente, a área de trabalho poderia ser acrescida de novos compromissos, inferidos através do cálculo de compromissos.

Estes aspectos levaram-nos a adotar uma representação para a área de trabalho semelhante a uma carteira de compromissos. A grande diferença, além de seu caráter particular, está na permissão para livre alteração de seu conteúdo.

5. Interface

A interface da máquina básica foi projetada para atender dois modos de operação distintos. O primeiro, chamado de modo de operação normal, atinge aplicações que necessitam dos serviços básicos de manipulação de diálogos. Isto é feito através de um conjunto de operações que dão suporte a diálogos legais entre participantes. Estas operações tratam do controle, evolução e validação de diálogos.

O segundo modo de operação, o modo simulado, trata de prover uma interface aberta que permita investigações sobre o ACCORD. Este modo, além das operações do modo real, possui novas primitivas que permitem a edição livre dos componentes de um diálogo. Com isto, carteiras de compromissos, registros de eventos e todas as regras do ACCORD podem ser manipuladas livremente, sem restrições.

A máquina básica pode ainda ser acessada através de uma interface gráfica própria. Esta interface permite a visualização das carteiras de compromisso, do registro de eventos e a execução interativa das operações da máquina básica. Os dois modos de operação da máquina básica são suportados pela interface gráfica. A versão atual da máquina básica ainda não incorpora as operações de interface descrita nesta seção, apenas implementa os itens descritos na seção 4. Estas operações farão parte das próximas implementações desta que contará, inclusive, com uma interface gráfica.

5.1. Operações para Diálogo no Modo Normal

A máquina básica dispõe de operações que dão suporte a sistemas de diálogo. Estas primitivas tratam do controle, evolução e validação de diálogos no modo de operação normal. As operações para o modo simulado, por sua vez, são tratadas no próximo item.

5.1.1. Contribuir para um Diálogo

Esta operação permite que o usuário contribua para um diálogo. O usuário *fala* uma sentença que se transforma em um evento do diálogo e em compromissos para os participantes do diálogo. Se a sentença proferida ferir alguma regra de legalidade ou compromisso, é rejeitada.

5.1.2. Iniciar um Novo Diálogo

Esta operação estabelece um novo diálogo. É necessário especificar dois participantes e o modo de operação do diálogo: normal ou simulado. Uma vez o diálogo iniciado, não é possível alterar o seu modo de operação. As operações para um diálogo simulado são descritas no item 5.2. Tal qual a especificação do ACCORD, logo após o início de um diálogo os participantes possuem carteira de compromissos vazias, o que também acontece com o registro de eventos.

5.1.3. Transferir da Carteira de Compromissos para Área de Trabalho

Permite que compromissos sejam transferidos de uma carteira de compromissos para a área de trabalho, onde podem ser manipulados livremente. Esta operação em conjunto com o emprego de cálculo de compromisso e com a edição de área de trabalho (descritas a seguir) possibilita ao participante desenvolver estratégias, linhas de raciocínio ou descobrir inconsistências sem ter que se compromissar com a elocução de sentenças.

5.1.4. Calcular Compromissos na Área de Trabalho

Adiciona novos compromissos (compromissos implícitos) na área de trabalho a partir de compromissos armazenados nesta área. Em conjunto com a transferência de compromissos, permite diversas formas de manipulação de compromissos (vide item anterior).

5.1.5. Interromper um Diálogo

Interrompe um diálogo, armazenando seu registro de eventos e as carteiras de compromissos dos participantes. Na máquina básica não existe operação para encerrar um diálogo definitivamente, esta operação é coberta pela interrupção. O estado do diálogo (registro de eventos mais carteiras de compromissos) é armazenado em uma estrutura única que não pode ser editada. Isto é feito para garantir a integridade do diálogo.

5.1.6. Restabelecer um Diálogo

Esta operação restabelece um diálogo previamente interrompido. O registro de eventos e as carteiras de compromissos dos participantes são restabelecidas de acordo com o momento da interrupção, a partir de uma estrutura única não editável gerada no momento da interrupção do diálogo.

5.1.7. Desmembrar Estrutura Única

Separa os componentes de uma estrutura única, gerando arquivos texto para o registro de eventos e para as carteiras de compromisso. Com isto, é possível aproveitar o conteúdo de um diálogo normal (modo de operação normal) em diálogos simulados. Note que a máquina básica não possui nenhuma operação inversa a esta, que construa uma estrutura única. Mais uma vez, isto é feito para não ferir a integridade do diálogo.

5.1.8. Editar Área de Trabalho

Permite que compromissos sejam inseridos, alterados ou removidos de uma área de trabalho. (vide item 5.1.4).

5.1.9. Carregar uma Área de Trabalho

Adiciona compromissos á uma área de trabalho a partir de um arquivo texto.

5.1.10. Gravar uma Área de Trabalho

Armazena o conteúdo de uma área de trabalho em um arquivo texto.

5.1.11. Gravar Registro de Eventos

Armazena o conteúdo de um registro de eventos em um arquivo texto.

5.1.12. Carregar Regras de Legalidade

Carrega as regras de legalidade a partir de um arquivo texto.

5.1.13. Carregar Regras de Compromisso

Lê as regras de compromisso de um arquivo texto.

5.1.14. Carregar Regras de Derivação

Obtém as regras de derivação a partir de um arquivo texto.

5.2. Operações para Simulação

No modo simulado, além das operações do modo normal, a máquina básica possui uma série de operações que permitem alterações em carteiras de compromisso, eventos de diálogo e nas regras que definem o funcionamento da máquina básica. Estas operações não são permitidas no modo normal uma vez que têm o poder de alterar o rumo (e o histórico) de um diálogo. Este conjunto de operações tem o objetivo de auxiliar investigações sobre o ACCORD, permitindo a alteração dinâmica de suas regras e no conteúdo do diálogo sem impor restrições.

5.2.1. Retirar Contribuições de um Diálogo

Permite que o usuário retire as N últimas contribuições (eventos) de um diálogo. As carteiras de compromisso também voltam ao seu estado anterior as N contribuições.

5.2.2. Introduzir Contribuições a um Diálogo a partir de um Registro de Eventos

Através desta operação é possível inserir eventos em lote em um diálogo a partir de um arquivo texto. Esta operação só é permitida em diálogos simulados, uma vez que registros de eventos podem ser editados. Adicionalmente, esta operação valida o registro de eventos, verificando se alguma edição anterior não invalidou nenhuma regra de legalidade ou compromisso. Esta operação permite que diálogos sejam concatenados ou que uma seqüência de eventos que ocorra com freqüência seja aplicada automaticamente.

5.2.3. Editar Registro de Eventos

Permite que eventos sejam adicionados, alterados ou retirados de um registro de eventos. Esta operação, que só é permitida no modo simulado, atualiza também as carteiras de compromissos dos participantes. Da mesma forma do que a operação de introduzir contribuições a um diálogo, o registro de eventos é validado através das regras de legalidade e de compromisso.

5.2.4. Gravar Registro de Eventos

Armazena o conteúdo de um registro de eventos em um arquivo texto, onde cada linha do arquivo armazena um evento. A representação de eventos usada no arquivo texto não inclui o campo *ordem*, isto para facilitar a sua edição por editores de texto comuns. Assim, a ordem dos eventos é definida pela seqüência natural das linhas no arquivo texto.

5.2.5. Carregar Regras de Legalidade

Carrega as regras de legalidade a partir de um arquivo texto. As regras de legalidade estabelecidas antes desta operação são removidas. Esta operação inválida o conteúdo das carteiras de compromissos, que é totalmente refeito em função das novas regras de legalidade e do registro de eventos.

5.2.6. Carregar Regras de Compromisso

Lê as regras de compromisso de um arquivo texto. As regras estabelecidas antes desta operação são removidas. Esta operação inválida o conteúdo das carteiras de compromissos, que é totalmente refeito em função das novas regras de legalidade e do registro de eventos.

5.2.7. Carregar Regras de Derivação

Obtém regras de derivação a partir de um arquivo texto. As regras de derivação estabelecidas antes desta operação são removidas. Esta operação inválida o conteúdo das carteiras de compromissos, que é refeito em função das novas regras de legalidade e do registro de eventos.

5.3. Interface Gráfica

A interface gráfica da máquina usa janelas para mostrar as informações de um diálogo. Com isto, o participante pode ver simultaneamente sua carteira de compromissos, a carteira de compromissos do outro participante, o registro de eventos e, por fim, sua área de trabalho. Adicionalmente, há uma área de saída, onde tipicamente são apresentadas mensagens e, por fim, uma área de entrada onde o usuário executa comandos ou profere locuções.

A interface gráfica permite dois modos de operação: modo normal e modo simulado. O modo normal é o modo de operação onde um diálogo *normal* está sendo realizado, com dois usuários se interagindo, cada um em sua estação de trabalho. O modo simulado, por sua vez, permite apenas um participante, que assume o papel dos dois participantes. O modo simulado é útil na exploração da máquina básica, servindo, inclusive, para depuração de novos regras ou regras de derivação que tenham sido adicionadas ou modificadas.

Para que a interface gráfica funcione de forma transparente seja no modo normal ou virtual, esta foi desenvolvida sobre o sistemas de janelas X -Windows (MIT), que é independente de dispositivo gráfico e do meio de comunicação utilizado (local ou em rede). Isto é fundamental para a transparência entre os modos de operação.

6. Extensões

A máquina básica atualmente suporta apenas diálogos de dois participantes, seguindo os conceitos apresentados em Fuks (91), com pequenas modificações que visavam somente a sua implementação como um sistema baseado em regras.

Durante o desenvolvimento do protótipo da máquina básica (listado no apêndice), surgiram algumas questões que certamente direcionarão as futuras extensões deste

trabalho. Esta seção descreve estas questões e também mostra a representação de clichês, uma aplicação que utiliza as primitivas de diálogo descritas neste trabalho.

6.1. Clichê

O clichê (Laufer, Fuks & Schwabe 92) é visto como uma máquina de estados que controla eventos de diálogo. Estes eventos podem ser trocados entre os participantes do diálogo de forma a realizar clichês de diálogos característicos a objetivos determinados. Um determinado clichê restringe as possibilidades de caminhos pelos quais um diálogo pode navegar.

A noção de clichê é baseada em um conjunto de estados de conversação através dos quais o diálogo navega. A passagem de um estado para outro é efetivada através de um evento de diálogo, até que seja alcançado algum estado final. No ACCORD um evento de diálogo é composto por uma locução, pela definição de que participante emite a locução e para qual participante ela é dirigida. A locução em si é composta por um modificador de locução aplicado a uma ou mais sentenças.

O controle de clichês é feito a partir das primitivas de conversação da máquina básica. Com isto, para a máquina básica, o controle de clichês é simplesmente uma aplicação que usa as suas primitivas.

6.2. Suporte a Vários Participantes

Os axiomas de legalidade definidos em Fuks (91) definem as regras de etiqueta do diálogo de apenas dois participantes. Para que o ACCORD, por conseqüência a máquina básica, suportem diálogos de mais de dois participantes, é necessário o estudo e definição de novos axiomas de legalidade e compromisso.

6.3. Representação Clausal para Compromissos

Atualmente, a representação de compromissos da máquina básica é feita de forma direta. Isto é, o compromisso é armazenado nas carteiras de compromissos ou no registro de eventos da mesma forma do este foi proferido no diálogo, sem sofrer nenhuma alteração. Da mesma forma do que em sistemas de resolução baseados em lógica de primeira ordem, é necessário representar as nossas cláusulas (compromissos) em uma representação clausal (Casanova, Giorno & Furtado 87). Desta forma, a unificação de compromissos (*match* em se tratando de sistemas baseados em regras) torna-se computacionalmente tratável.

Para que isto seja alcançado, é necessário a definição de um algoritmo de representação clausal que atue sobre um compromisso C que produza como saída uma única representação clausal R(C). Em seguida, o cálculo de compromissos e os axiomas de comprimissos precisam ser revisados para atuarem sobre representações clausais de compromissos. Na versão atual da máquina básica esta questão não foi tratada, e por conseguinte o cálculo não está integrado à implementação do DAC. Foi oferecido apenas um exemplo (i.e. uma regra) para ilustrar as suas forma e funcionalidade.

6.4 Regras de Inconsistência

Para o devido suporte ao modificador de locução **resolve**, é necessário que se defina um conjunto de regras de *inconsistência* que represente as situações onde *ocorre inconsistência*. Estas regras seriam semelhantes as do cálculo de compromisso.

7. Conclusões

Este trabalho apresentou a máquina básica do ACCORD – uma estrutura para representação de diálogos usando compromisso. A máquina básica implementa o DAC e parcialmente o CC. O primeiro responsável pelo pelo controle de diálogo, manipulação, armazenamento de compromissos, enquanto que o segundo trata das conseqüências de se possuir compromissos, inferindo novos compromissos a partir dos compromissos assumidos por um participante de um diálogo.

A implementação é baseada em sistemas de produção, constituída de três componentes básicos: conjuntos de regras, banco de dados e interpretadores para as regras. A máquina básica possui regras que representam os axiomas de legalidade, axiomas de compromisso e as regras de derivação do Cálculo de Compromissos – conceitos do ACCORD. O banco de dados é formado pelos compromissos assumidos no decorrer do diálogo armazenados nas carteiras de compromisso de cada participante e pelo registro de eventos. A funcionalidade da máquina básica foi implementada a partir de interpretadores que, além da análise léxico-sintática de sentenças, tratam das regras de legalidade, regras de compromisso e do cálculo de compromissos. O interpretador responsável pelas regras de legalidade é bastante simples, uma vez que a versão atual do ACCORD suporta diálogos de apenas dois participantes.

A máquina básica está prevista para funcionar em dois modos de operação distintos: normal e simulado. O modo normal define uma interface de programação para aplicações *(API - Application Programming Interface)* que desejem utilizar a máquina básica e, por conseqüência, a estrutura definida pelo ACCORD. O modo simulado, permite que todas as estruturas de um diálogo - carteiras de compromissos e eventos de diálogo - sejam manipuladas sem restrições, permitindo investigações e testes na máquina básica. Finalmente, estão previstas extensões que permitam diálogos de mais de dois participantes e que adotem uma representação clausal para compromissos.

8. Referências

Casanova, M. A., Giorno, F., Furtado, A. L.: Programação em Lógica e a Linguagem Prolog, Editora Edgar Blücher Ltda, 1987.

Ellis, C.A., Gibbs, S.J. & Rein, G.L.: *Groupware – Some Issues and Experiences*; Communications of the ACM, Vol.24, nº.1, pp 39-58, January 1991.

Fuks, H.: ACCORD – A Framework for Dialogue Representation using Commitment; Monografias em Ciência da Computação nº 24/91, Departamento de Informática, PUC-RJ, Dezembro 1991.

Goldblatt, R.: Axiomatising the Logic of Computer Programming. XI; Lectures Notes in Computer Science V130, Springer-Verlag,1982.

Hahn, U., Jarke, M. & Rose, T.: *Teamwork Support in a Knowledge-Based Information Systems Environment*, IEEE Transactions on Software Engineering, Vol.17, nº.5, pp 467-482, May 1991.

Laufer, C., Fuks, H., Schwabe, D.: ACCORD – Representação de Clichês de Conversação para Cooperação, Monografias em Ciência da Computação nº 3/92, Departamento de Informática, PUC-RJ, Março 1992.

Lucena, C. J. P., Leite, J., Schwabe, D. & Fuks, H.: A Research Agenda for Software Design; Monografias em Ciência da Computação nº 29/91, Departamento de Informática, PUC-RJ, Março 1991.

Maibaum, T. S E.: A logic for the Formal Requirements Specification of Real-Time/Embedded Systems. FOREST internal report, Imperial College, London, 1987.

Apêndice. Código da Máquina Básica em Prolog

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%    Valid Locution Modifiers
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

modifier(questions).
modifier(asserts).
modifier(withdraws).
modifier(denies).
modifier(justifies).
modifier(resolve).
modifier(why).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%    Valid sequences derived from the Legality Axioms
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

seq(asserts).
seq(question).
seq(why).
seq(withdraws).

seq(asserts,asserts).
seq(asserts,questions).
seq(asserts,resolve).
seq(asserts,why).
seq(asserts,withdraws).

seq(denies,asserts).
seq(denies,questions).
seq(denies,resolve).
seq(denies,why).
seq(denies,withdraws).

seq(justifies,asserts).
seq(justifies,questions).
```

```
seq(justifies,resolve).
seq(justifies,why).
seq(justifies,withdraws).

seq(questions,asserts).
seq(questions,denies).
seq(questions,withdraws).

seq(resolve,asserts).
seq(resolve,withdraws).

seq(why,justifies).
seq(why,resolve).
seq(why,withdraws).

seq(withdraws,asserts).
seq(withdraws,questions).
seq(withdraws,resolve).
seq(withdraws,why).
seq(withdraws,withdraws).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%    Commitment Rules
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

commitment_rule(asserts,([],[(origin,c,current),(destin,c,current)])).
commitment_rule(asserts,([(origin,c,current)],error)).

commitment_rule(questions,([],[])).
commitment_rule(questions,([(origin,c,current)],error)).

commitment_rule(withdraws,([(origin,c,current)],[(origin,d,current)])).
commitment_rule(withdraws,([],[(origin,d,current)])).
commitment_rule(withdraws,([(origin,d,current)],error)).

commitment_rule(why,([],[(origin,c,why(current)),(destin,c,current)])).
commitment_rule(why,([(origin,c,current),(destin,c,current)],[(destin,c,curre
nt),(origin,c,why(current))])).
```

```
commitment_rule(justifies,([(destin,c,why(last)),(origin,c,last)],
[(destin,c,why(last)),(destin,c,implies_(current,last)),(destin,c,current),(o
rigin,c,last),(origin,c,implies_(current,last)),(origin,c,current)])).


commitment_rule(denies,([],[(origin,c,not_(current)),(destin,c,not_(current))
])).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%    Last event from dialogue
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

last_event(0,null).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%    Establishing a participant relationship
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

other_participant(a,b).
other_participant(b,a).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%    Initializing the Dialogue
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

initialise :-
      abolish(event/5),
      abolish(commitment_store/3),
      abolish(last_event/2),
      assert(commitment_store(a,c,null)),
      assert(commitment_store(a,d,null)),
      assert(commitment_store(b,c,null)),
      assert(commitment_store(b,d,null)),
      assert(last_event(0,null)).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%    Allowing a participant to speak
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

speak(Speaker,Audience,String) :-
      can_talk(Speaker,Audience), !,
      parse(String,Locution), !,
      Locution =.. [Modifier,Expression], !,
      check_legality(Modifier), !,
      alter_commitment(Speaker,Audience,Modifier,Expression), !,
      add_event(Speaker,Audience,Modifier,Expression).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%    Checking if the turn-taking is being preserved
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

can_talk(_,_) :-
      last_event(0,null), !.

can_talk(Speaker,Audience) :-
      last_event(N,_),
      other_participant(Speaker,Audience),
      event(N,Audience,_,_,_).

can_talk(_,_) :-
      write('This participant cannot talk in this momment'),
      !, fail.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%    Checking if the Legality Axioms are being preserved
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
check_legality(Modifier) :-
      last_event(0,_),
      seq(Modifier).

check_legality(Modifier) :-
      last_event(N,_),
      event(N,_,_,Previous,_),
      seq(Previous,Modifier).

check_legality(Modifier) :-
      write(Modifier),
      write(' is an invalid modifier in this momment'),
      !, fail.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%    Altering the Commitment Stores
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

alter_commitment(Speaker,Audience,Modifier,Statement) :-
      check_commitment_error(Speaker,Audience,Modifier,Statement), !,
      check_commitment_healthy(Speaker,Audience,Modifier,Statement).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%    Checking for Commitment Axioms
%%    with errors as pos-conditions
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

check_commitment_error(Speaker,Audience,Modifier,Statement) :-
      obtain_commitment_error_rules(Modifier,Rules),
      check_error_rules(Speaker,Audience,Statement,Rules).

%% check_commitment_error(_,_,_,_).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%
%%    Checking for rules with errors as
%%    pos-conditions in a rule list
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

check_error_rules(_,_,_,[]) :- !.

check_error_rules(Speaker,Audience,Statement,[Rule|Tail]) :-
      check_error(Speaker,Audience,Statement,Rule),
      check_error_rules(Speaker,Audience,Statement,Tail).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%    Checking for rules with errors as pos-conditions
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

check_error(Speaker,Audience,Statement,Rules) :-
      obtain_condition(Statement,Rules,Unified_Rules),
      match_error_rule(Speaker,Audience,Unified_Rules),
      write('There is a commitment rule that prohibits this sentence in this
moment'),
      !, fail.

check_error(_,_,_,_).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%    Matching a rule with error as a pos-condition
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

match_error_rule(Speaker,Audience,[(Party,Type,Condition)|_]) :-
      obtain_party(Speaker,Audience,Party,Person),
      commitment_store(Person,Type,Condition).

match_error_rule(Speaker,Audience,[_|Tail]) :-
      match_error_rule(Speaker,Audience,Tail).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%    Checking for healthy s-o-a's in the commitment axioms
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

check_commitment_healthy(Speaker,Audience,Modifier,Statement) :-
      obtain_commitment_condition(Modifier,(Pre,Pos)),
      obtain_condition(Statement,Pre,Pre_Condition),
      match_commitment_pre_condition(Speaker,Audience,Pre_Condition),
      retract_pre_condition(Speaker,Audience,Pre_Condition),
      obtain_condition(Statement,Pos,Pos_Condition),
      apply_pos_condition(Speaker,Audience,Pos_Condition).

check_commitment_healthy(_,_,_,_) :-
      write('There is no commitment rule for this sentence'),
      !,fail.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%    Obtaining commitment rules
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

obtain_commitment_condition(Modifier,(Pre,Pos)) :-
      commitment_rule(Modifier,(Pre,Pos)),
      Pre \== [].

obtain_commitment_condition(Modifier,([],Pos)) :-
      commitment_rule(Modifier,([],Pos)).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%    Obtaining commitment error rules
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

obtain_commitment_error_rules(Modifier,Pre_list) :-
```

```
      findall(Pre,commitment_rule(Modifier,(Pre,error)),Pre_list).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%    Obtaining a condition
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

obtain_condition(_,[],[]) :- !.

obtain_condition(Current,[(Party,Type,C)|T],[(Party,Type,Condition)|Tail]) :-
      last_event(_,Last),
      unify_condition(Current,Last,C,Condition),
      obtain_condition(Current,T,Tail).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%    Unifying a condition
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

unify_condition(Current,Last,current,Current) :- !.

unify_condition(Current,Last,last,Last) :- !.

unify_condition(Current,Last,C,Condition) :-
      C =.. [Operator,T],!,
      unify_condition(Current,Last,T,Term),
      Condition =.. [Operator,Term].

unify_condition(Current,Last,C,Condition) :-
      C =.. [Operator,T1,T2],!,
      unify_condition(Current,Last,T1,Term1),
      unify_condition(Current,Last,T2,Term2),
      Condition =.. [Operator,Term1,Term2].


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
```

```
%%    Matching the pre-conditions
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

match_commitment_pre_condition(_,_,[]) :- !.

match_commitment_pre_condition(Speaker,Audience,Pre) :-
      find_commitment(Speaker,Audience,Pre).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%    Finding the commitments that matches pre-conditions
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

find_commitment(_,_,[]) :- !.

find_commitment(Speaker,Audience,[Condition|Tail]) :-
      obtain_commitment(Speaker,Audience,Condition),
      find_commitment(Speaker,Audience,Tail).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%    Obtaining the commitment that matches pre-condition
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

obtain_commitment(Speaker,Audience,(Party,Type,Expression)) :-
      obtain_party(Speaker,Audience,Party,Person),
      commitment_store(Person,Type,Expression).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%    Applying the pos-conditions
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

apply_pos_condition(_,_,[]) :- !.
```

```
apply_pos_condition(Speaker,Audience,[(Party,Type,Condition)|Tail]) :-
      obtain_party(Speaker,Audience,Party,Person),
      assert_commitment_store(Person,Type,Condition),
      apply_pos_condition(Speaker,Audience,Tail).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%    Retracting pre-conditions
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

retract_pre_condition(_,_,[]) :- !.

retract_pre_condition(Speaker,Audience,[(Party,Type,Condition)|Tail]) :-
      obtain_party(Speaker,Audience,Party,Person),
      retract(commitment_store(Person,Type,Condition)),
      retract_pre_condition(Speaker,Audience,Tail).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%    Asserting commitments in the commitment stores
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

assert_commitment_store(Party,Type,Condition) :-
      not commitment_store(Party,Type,Condition),
      assert(commitment_store(Party,Type,Condition)).

assert_commitment_store(_,_,_).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%    Adding an event to the event list
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

add_event(Speaker,Audience,Modifier,Statement) :-
```

```
        last_event(Old,_),
        abolish(last_event/2),
        N is Old + 1,
        assert(event(N,Speaker,Audience,Modifier,Statement)),
        assert(last_event(N,Statement)).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%    Obtaining a party
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

obtain_party(Speaker,_,origin,Speaker) :- !.
obtain_party(_,Audience,destin,Audience) :- !.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%    Parsing a string and generating an Expression
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

parse(String,Expression) :-
        list_text(List,String),
        lexical(Tokens,List,[]),
        locution(Expression,Tokens,[]).

parse(_,_) :-
        write('There is an parsing error in this sentence'),
        !, fail.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%    Performing a Lexical analysis in a String
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

lexical([],[],[]).
```

```
lexical([close]) -->
      ")".

lexical(Tokens) -->
      " ",
      lexical(Tokens).

lexical([open|Tokens]) -->
      "(",
      lexical(Tokens).

lexical([close|Tokens]) -->
      ")",
      lexical(Tokens).

lexical([Token|Tokens]) -->
      token(Token),
      " ",
      lexical(Tokens).

lexical([Token,close|Tokens]) -->
      token(Token),
      ")",
      lexical(Tokens).

lexical([Token,open|Tokens]) -->
      token(Token),
      "(",
      lexical(Tokens).

lexical([Token]) -->
      token(Token).

token(Token) --> letters(List),
      { list_text(List,String),
       string_term(String,Token) }.

letters([X]) -->
      letter(X).

letters([Letter|Tail]) -->
      letter(Letter),
```

```
      letters(Tail).

letter(X) --> [X],
      { X > 96,
       X < 123 } .


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%    Recognizing a Locution
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

locution(Locution) -->
      modifier(Modifier),
      statement(Statement),
      { Locution =.. [Modifier,Statement] }.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%    Recognizing a Locution Modifier
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

modifier(X) -->
      [X],
      { modifier(X) }.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%    Recognizing a Statement
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

statement(Statement) -->
      term1(Term1),
      [implies],
      statement(State_aux),
      { Statement =.. [implies_,Term1,State_aux] }.
```

```
statement(Statement) -->
      term1(Statement).

term1(Term1) -->
      term2(Term2),
      [or],
      term1(Term_aux),
      { Term1 =.. [or_,Term2,Term_aux] }.

term1(Term1) -->
      term2(Term1).

term2(Term2) -->
      term3(Term3),
      [and],
      term2(Term_aux),
      { Term2 =.. [and_,Term3,Term_aux] }.

term2(Term2) -->
      term3(Term2).

term3(Term3) -->
      [not],
      variable(T),
      { Term3 =.. [not_,T] }.

term3(Term3) -->
      variable(Term3).

term3(Term3) -->
      [open],
      statement(Term3),
      [close].

variable(X) -->
      [X],
      { atom(X) }.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
```

```
%%    Generating Commitments from the Commitment Store
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

calculate(Commitment_List, Commitment) :-
      cc_rule(C_List, Commitment),
      sub_set(C_List,Commitment_List).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%    Checking for inconsistencies in the Commitment Stores
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

inconsistent(Commitment_List,Commitment) :-
      calculate(Commitment_List,Result),
      inconsistent_rule(Result,Commitment).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%    Checking for sub-set relationships
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

sub_set([],_) :- !.

sub_set([H|Tail],List) :-
      member(H,List),
      sub_set(Tail,List).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%    Checking for membership
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

member(H,[H|_]) :- !.
member(H,[_|Tail]) :-
```

```
      member(H,Tail).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%    Inconsistency rules (example)
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

inconsistent_rule((X,C_or_D),(not_(X),C_or_D)).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%    Calculus Rules (example)
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

cc_rule([(C_or_D,implies_(X,Y)),(C,_or_DY)],(C_or_D,X)).
```