Programa Interdisciplinar de Pós-Graduação em

Computação Aplicada

Mestrado Acadêmico

Diego Gonçalves Silva

AgentSpeak(PL)
Uma Nova Linguagem de Programação para Agentes BDI
com um Modelo Integrado de Redes Bayesianas

São Leopoldo, 2011

DIEGO GONÇALVES SILVA

AgentSpeak(PL)
Uma Nova Linguagem de Programação para Agentes BDI com um Modelo Integrado de Redes Bayesianas

Dissertação apresentada
como requisito parcial para
obtenção do título de Mestre, pelo
Programa Interdisciplinar de Pós-Graduação em
Computação Aplicada da
Universidade do Vale do Rio dos Sinos

Prof. Dr. João Gluz
Orientador

SÃO LEOPOLDO

2011

S586a Silva, Diego Gonçalves
AgentSpeak(PL): uma nova linguagem de programação para agentes BDI com um modelo integrado de Redes Bayesianas / por Diego Gonçalves Silva. – São Leopoldo, 2011.

86 f. : il. ; 30 cm.

Com: artigo “AgentSpeak(PL): a new programming language for BDI Agents with integrated Bayesian Network model”

Dissertação (mestrado) – Universidade do Vale do Rio dos Sinos, Programa Interdisciplinar de Pós-Graduação em Computação Aplicada, São Leopoldo, RS, 2011.
Orientação: Prof. Dr. João Gluz, Ciências Exatas e Tecnológicas.

1.Agentes inteligentes (software). 2.Linguagem de programação (Computadores). 3.AgentSpeak(PL) – Linguagem de programação (Computadores). 4.Rede Bayesiana. 5.Probabilidades. I.Gluz, João. II.Título.

CDU 004.89
004.43
004.72

Catalogação na publicação:
Bibliotecária Carla Maria Goulart de Moraes – CRB 10/1252

Dedico este trabalho as pessoas de bem, que com otimismo, fé e perseverança, dia após dia lutam para construir um mundo melhor e acreditam que mudar é possível.

Agradecimentos

Agradeço primeiramente a Deus, causa primeira de todas as coisas, que me acompanha dia-a-dia em minha caminhada e me dá forças para lutar em busca da realização pessoal e do sucesso.

Agradeço também, de coração, a todos aqueles que, direta ou indiretamente contribuíram para que este trabalho fosse concluído.

“É muito melhor arriscar coisas grandiosas, alcançar triunfos e glórias, mesmo expondo-se a derrota, do que formar fila com os pobres de espírito que nem gozam muito nem sofrem muito, porque vivem nessa penumbra cinzenta que não conhece vitória nem derrota”.

Theodore Roosevelt

Resumo

Quando este trabalho foi iniciado não era possível desenvolver de forma prática e direta *softwares* com agentes inteligentes onde suas crenças poderiam ser probabilidades relacionadas com seu ambiente, visto que as definições formais das linguagens disponíveis não previam tal possibilidade. Quando essa necessidade precisava ser levada em conta e implementada, devia-se lançar mão de técnicas avançadas de programação onde deveria haver a integração de ambientes de desenvolvimentos e linguagens, a fim de tornar a implementação factível.

Este trabalho teve como objetivo o desenvolvimento de uma nova linguagem de programação orientada a agentes denominada AgentSpeak(PL), baseada em AgentSpeak(L), com o intuito de agregar o conceito de crenças probabilísticas através do uso de Redes Bayesianas sendo implementada através de uma extensão da ferramenta de programação Jason.

Palavras Chave: Programação Orientada a Agentes, AgentSpeak, Jason, Lógica Probabilística, Redes Bayesianas.

Abstract

When this work was started it was not possible to develop so practical and straightforward software with intelligent agents where their beliefs could likely be related to their environment, as the settings formal language available did not foresee such a possibility. When this necessity had to be taken into account and implemented, one should resort of advanced programming techniques where there should be integration of development environments and languages in order to make the implementation feasible.

This study aimed to develop a new language agent oriented programming called AgentSpeak (PL), based on AgentSpeak (L) with the intention of creating the concept of probabilistic beliefs through the use of Bayesian networks being implemented through an extension Jason's programming tool.

Keywords: Agent Oriented Programming, AgentSpeak, Jason, Probabilistic Logical, Bayesians Networks.

Lista de figuras

FIGURA 1. Tipologia de Agentes. Adaptação de [HYACINTH, 1996]. 19
FIGURA 2. Atributos de um agente inteligente. Adaptação de [HAYES, 1995]. 20
FIGURA 3. Arquitetura BDI genérica [WOOLDRIGDE, 1999]..... 21
FIGURA 4. Exemplos de planos AgentSpeak(L) [BORDINI; et al., 2004]. 23
FIGURA 5. Gramática do AgentSpeak(L) [MOREIRA; BORDINI, 2004]. 24
FIGURA 6. Ciclo de interpretação de um programa AgentSpeak(L) [MACHADO; BORDINI, 2002]. 26
FIGURA 7. Tela principal do Jason. 28
FIGURA 8. Teorema de Bayes [ALMEIDA; VALE, 1999]..... 30
FIGURA 9. Rede Bayesiana para o domínio de alarmes contra roubo [RUSSEL; NORVIG, 2003]. 32
FIGURA 10. Rede Bayesiana e probabilidades para o domínio de alarmes contra roubos [RUSSEL; NORVIG, 2003]. 33
FIGURA 11. Tipos de inferências. 35
FIGURA 12. Tela principal do *software* BNJ [BNJ, 2010]. 36
FIGURA 13. Rede Bayesiana básica para o problema da crise [MILCH; KOLLER, 2000]. 40
FIGURA 14. Probabilidades Prévias em AgentSpeak(PL). 43
FIGURA 15. Probabilidades Prévias em AgentSpeak(PL). 43
FIGURA 16. Gramática da linguagem AgentSpeak(PL). 45
FIGURA 17. Configuração básica de um agente em AgentSpeak(PL). 48
FIGURA 18. Passos na execução de um agente. 50
FIGURA 19. Ciclo de execução de um agente..... 51
FIGURA 20. Estrutura de diretórios do *Jason*. 60
FIGURA 21. Estrutura de diretórios da pasta *jason*. 61
FIGURA 22. Estrutura de um arquivo em *JavaCC*..... 62
FIGURA 23. Estrutura de um agente em AgentSpeak(L). 62
FIGURA 24. Estrutura de um agente em AgentSpeak(PL). 63
FIGURA 25. Diretórios das novas classes inseridas no *Jason*..... 64
FIGURA 26. Diretórios das funções nativas do *Jason*..... 65

FIGURA 27. Rede Bayesiana clássica adaptada de [RUSSEL; NORVIG, 2003]. 67
FIGURA 28. Rede Bayesiana clássica adaptada de [RUSSEL; NORVIG, 2003] programada em AgentSpeak(PL) utilizando a ferramenta *JasonBayes*. 68
FIGURA 29. Rede Bayesiana com diagrama de influência e função utilidade. Adaptada de [RUSSEL; NORVIG, 2003]. 69
FIGURA 30. Implementação em AgentSpeak(PL) de uma Rede Bayesiana com diagrama de influência e função utilidade. 71
FIGURA 31. Rede Bayesiana original. Adaptada de [RUSSEL; NORVIG, 2003]. 73
FIGURA 32. Rede Bayesiana Dinâmica em cinco fatias de tempo. Adaptada de [RUSSEL; NORVIG, 2003]. 73
FIGURA 33. Rede Bayesiana Dinâmica implementada em AgentSpeak(PL). 75
FIGURA 34. Rede Bayesiana de um agente controlador de semáforo. 77
FIGURA 35. Agente controlador de semáforo implementado em AgentSpeak(PL). 78

Lista de tabelas

TABELA 1. CPT Alarme adaptado de [RUSSEL; NORVIG, 2003]. 33
TABELA 2. Tabela comparativa de trabalhos. 41
TABELA 3. Tabela de função utilidade. 70

Lista de abreviaturas

API	*Application Programming Interface*
BDI	*Beliefs, Desires, Intentions*
BNJ	*Bayesian Network tools in Java*
FRC	Função de Revisão de Crenças
IA	Inteligência Artificial
IDE	*Integrated Development Environment*
LP	Lógica Probabilística
PL	*Probabilistic Language*
RB	Redes Bayesianas
RBD	Redes Bayesianas *Dinâmicas*

Sumário

1 Introdução 12

1.1 Definição do Problema e Objetivos 12
1.2 Metodologia e Organização do Texto 14

2 Referencial Teórico 16

2.1 Agentes de *Software* 16
2.1.1 Classificação dos Agentes 17
2.1.2 Agentes Inteligentes 20
2.1.3 A Arquitetura BDI 21
2.2 Linguagem AgentSpeak(L) 22
2.2.1 Sintaxe Abstrata 23
2.2.2 Semântica Informal 25
2.3 Jason 27
2.4 Redes Bayesianas 29
2.4.1 Raciocinando Sob Incertezas 29
2.4.2 A Regra de Bayes 30
2.4.3 Redes Bayesianas 32
2.4.4 Inferência em Redes Bayesianas 34
2.4.5 *BNJ – Bayesian Network tools in Java* 35

3 Trabalhos Relacionados 37

3.1 *Integrating BDI Model and Bayesian Networks* 37
3.2 *Bayesian Logic Programs* 38
3.3 *Probabilistic Models for Agent's Beliefs and Decisions* 39
3.4 Análise dos Trabalhos Relacionados 40

4 Proposta de Trabalho 42

4.1 Introdução a AgentSpeak(PL) 43
4.2 Sintaxe e Características de AgentSpeak(PL) 45
4.3 Estruturas da Semântica de AgentSpeak(PL) 47
4.4 Arquitetura e Ciclo de Execução 51
4.5 Novas Relações e Funções 53
4.6 Novas Regras de Transição 54

5 Implementação da Linguagem AgentSpeak(PL) 60

5.1 Análise Léxica, Sintática e Semântica 61
5.2 Interpretação e Execução do Código 64

6 Aplicações de AgentSpeak(PL) 67

6.1 Caso de Estudo: Modelo de Diagnóstico Médico 67
6.2 Caso de Estudo: Diagrama de Influência 69
6.3 Caso de Estudo: Rede Bayesiana Dinâmica 72

6.4 Caso de Estudo Prático 75

7 Conclusões 79

Referências Bibliográficas 80

Apêndice A – Artigo publicado na *International Conference on Information Science and Applications 2011* 83

1 Introdução

1.1 Definição do Problema e Objetivos

O uso de sistemas baseados em agentes tem crescido de maneira rápida. Muitos problemas de difícil resolução que com paradigmas de programação antigos seriam resolvidos com certo trabalho e dispêndio de tempo, com a programação orientada a agentes podem ser construídos mais facilmente [BORDINI; et al., 2007]. A crescente complexidade das soluções que são necessárias para satisfazer os problemas que surgem faz com que o uso de agentes se popularize cada vez mais [BORDINI; et al., 2007].

Segundo [SHOHAM, 1994], um agente é uma entidade de *software* que funciona continuamente e de maneira autônoma em um ambiente em particular, frequentemente habitado por outros agentes e processos. O conceito de agente descreve a sua capacidade de atuar de forma autônoma no ambiente onde ele está inserido.

A abstração de detalhes de *hardware*, de *software* e de padrões de comunicação de tal maneira que haja uma interação direta através de interfaces entre agentes é um grande desafio a ser alcançado quando se trabalha com agentes [RUSSEL; NORVIG, 2003].

Um modelo de agentes bastante conhecido e utilizado é baseado na especificação das crenças (*beliefs*), desejos (*desires*) e intenções (*intentions*) dos agentes (modelo BDI, seguindo a terminologia adotada na literatura inglesa). O objetivo principal deste modelo é permitir o projeto e desenvolvimento de agentes utilizando noções antropomórficas, tais como estados mentais e ações [BORDINI; et al., 2007]. Estas noções e suas propriedades também podem ser estudadas e definidas através de lógicas modais que permitem analisar, especificar e verificar as características dos agentes racionais [BORDINI; et al., 2007].

Uma crença no modelo BDI é definida como uma proposição lógica: ou o agente acredita que determinado crença é verdadeira, ou acredita que não é. As ferramentas hoje disponíveis para o desenvolvimento de agentes BDI não permitem que se trabalhe com o conceito de crenças probabilísticas [GARCIA; et al., 2003], ou seja, não permitem ao agente conhecer, representar ou inferir graus de crença (ou graus de incerteza) a respeito de uma dada proposição. Um grau de crença é definido através da probabilidade subjetiva atribuída a uma dada crença. O conceito de Redes Bayesianas (RB) [PEARL, 1993; CHAN; DARWICHE, 2005] se enquadra neste cenário, permitindo a modelagem de crenças probabilísticas.

Como exemplo de uso de probabilidade na tomada de decisões na programação orientada a agentes, podemos imaginar um ambiente composto por diversos agentes que realizam mineração de dados na internet, a fim de tomar decisões potencialmente corretas sobre a compra e venda de ações na bolsa de valores. Esses agentes de *software* possuiriam em sua programação crenças com eventos probabilísticos que indicariam o grau de certeza de determinada ação. Poderíamos programar os agentes para minerar dados dos principais

jornais on-line de um determinado país. Caso a frequência de citação da palavra “petróleo” associada a “descoberta” e “dinheiro” fosse acima de 80%, isso levaria os agentes a sugerir a compra de ações de empresas petrolíferas, dada os fatos e notícias pesquisados. Caso a mineração resultasse na palavra ”petróleo” associada a “desastre” em um nível acima de 50 %, levaria o agente a sugerir a venda imediata das ações dessas mesmas empresas.

A integração entre as linguagens de agentes atuais e o conceito de crenças probabilísticas pode ser abordada de diversas formas e em diversos níveis de abstração. No nível mais abstrato a análise desta questão de integração é usualmente tratada através de Lógicas Probabilísticas (LP) [KORB; NICHOLSON, 2004] Embora, as LPs tenham efetivamente a capacidade de representar ambos os tipos de modelos, existem notórios problemas relacionados com a tratabilidade dos modelos resultantes [HALPERN; MOSES, 1992].

Outra abordagem possível, em um nível muito mais próximo da implementação, é formada pela junção *adhoc* de ambos os tipos de modelos no próprio código de programação dos agentes. Tanto agentes BDI quanto agentes *bayesianos* dependem de bibliotecas e *frameworks* de desenvolvimento, com interfaces de programação de aplicações (API) padronizadas. Assim, um agente híbrido pode ser concebido e implementado combinando chamadas em APIs de distintos modelos.

Em ambos os casos citados o conhecimento técnico envolvido torna o desenvolvimento dos agentes uma tarefa bastante complexa, muitas vezes inviabilizando projetos de médio e grande porte.

Dessa forma, uma questão importante de pesquisa relacionada ao projeto e implementação de agentes híbridos que combinem modelos BDI e modelos probabilísticos, é decidir o nível de abstração apropriado para tratar o projeto e implementação destes agentes.

Este trabalho se propôs a abordar esta questão no nível da própria linguagem de programação dos agentes, tendo como objetivo definir uma nova linguagem de programação de agentes, denominada AgentSpeak(PL) (*AgentSpeak Probabilistic Language*). A linguagem AgentSpeak(PL) é baseada na linguagem AgentSpeak(L) [BORDINI; et al., 2004], mas será capaz de suportar a representação e inferência de modelos probabilísticos de forma completamente integrada ao modelo de crenças do agente. A semântica de AgentSpeak(PL) será definida de forma a ser independente de *frameworks*, ferramentas ou bibliotecas externa de manipulação de modelos probabilísticos.

Os programas de computador são formulados a partir de uma linguagem de programação através da especificação de seu código (sintaxe) e seu processo de computação (semântica). Apesar de importante, somente a especificação da sintaxe e semântica de uma linguagem não é suficiente para se ter uma idéia clara a respeito de suas capacidades e formas de uso.

Computadores interpretam sequências de instruções específicas, mas não os textos programados em uma determinada linguagem de programação. Portanto, o texto do programa deve ser traduzido em instruções adequadas antes que possa ser processado por um computador. A tradução automatizada é realizada através de um programa chamado de compilador ou interpretador, onde o texto do programa em uma determinada linguagem de programação será traduzido (compilado ou interpretado) para ser executado em um determinado ambiente [WIRTH, 1997]. Com AgentSpeak(PL) não é diferente. Para tornar

possível sua programação foi criado em Java um ambiente de desenvolvimento e execução de agentes denominado *JasonBayes*, que foi construído como uma extensão do ambiente *Jason*, que é o ambiente original de programação de AgentSpeak(L) [JASON, 2011]. O ambiente JasonBayes permitirá que programas escritos em AgentSpeak(PL) sejam executados e testados.

1.2 Metodologia e Organização do Texto

A metodologia usada para desenvolver o trabalho seguiu a seguinte sequencia:

- Estudo do referencial teórico e literatura relativa aos temas da presente dissertação;
- Especificação de uma nova linguagem de programação orientada a agentes baseada na linguagem AgentSpeak(L) através da mudança de sua semântica formal e operacional, chamando-a de AgentSpeak(PL);
- Estudo da ferramenta Jason;
- Modificação da ferramenta Jason a fim de implementar a linguagem de programação de agentes AgentSpeak(PL);
- Realização de testes para validação dos conceitos teóricos;
- Disponibilização da nova ferramenta na Web.

O presente trabalho está organizado da seguinte forma:

Capítulo 2: será detalhado todo o referencial teórico envolvido no trabalho. A explanação partirá do conceito de Agentes de *Software*, passando pela linguagem de programação de agentes AgentSpeak(L), depois pela ferramenta Jason e por fim com a conceituação e aplicação de Redes Bayesianas;

Capítulo 3: nesse capítulo serão abordados os trabalhos relacionados. Foram reunidos três trabalhos que contemplavam o mesmo tema desta dissertação. Uma breve resenha foi desenvolvida a fim de contextualizar os assuntos bem como uma tabela comparativa a fim de elencar pontos positivos, negativos e os diferenciais;

Capítulo 4: nesse item será apresentada detalhadamente a especificação de AgentSpeak(PL);

Capítulo 5: aqui será abordado a implementação da linguagem AgentSpeak(PL);

Capítulo 6: aqui serão abordados exemplos de aplicações da programação em AgentSpeak(PL).

Por fim, teremos as conclusões acerca do assunto e possíveis trabalhos futuros na área.

2 Referencial Teórico

Nesse capítulo serão abordados os conceitos acerca dos assuntos que estão diretamente envolvidos com o presente trabalho. Inicialmente será feita uma análise dos conceitos de Agentes de *Software*, posteriormente uma análise sobre a linguagem de programação de agentes AgentSpeak(L), em seguida uma abordagem sobre a ferramenta Jason e por fim uma explanação sobre Redes Bayesianas.

2.1 Agentes de *Software*

De forma geral pode-se definir os agentes de *software* como programas que podem criar e escolher caminhos de forma autônoma. Eles podem fazer muitas atividades, sendo capazes de buscar informações na rede, administrar agendas, negociar intenções simples, etc. Entretanto, seu desenvolvimento requer um alto grau de conhecimento e sua programação é complexa. Como eles agem de forma autônoma, podem no decorrer do seu ciclo de vida, gerar ações que devido a sua natureza podem ser classificadas como imprevisíveis.

A principal diferença entre *softwares* denominados convencionais e os *softwares* de agentes está na questão da autonomia, pois um agente é capaz de comunicar-se com outros agentes e com usuários, e também reagir a mudanças no seu ambiente, sempre buscando atingir uma de suas metas.

Uma segunda diferença importante [WOOLDRIGDE; JENNINGS, 1996] é que os agentes são *softwares* situados (ou localizados) em um meio ambiente específico, capazes de interagir continuamente com este meio através de percepções e ações.

Percepções são as entradas do agente a respeito de que eventos estão ocorrendo do ambiente, e ações são as saídas do agente que se reflete em modificações no meio ambiente. O diferencial aqui é que nem todas as aplicações de *software* precisam estar situadas em algum meio para operar, mas os agentes tem esta necessidade. Esta é uma propriedade definidora do conceito de agente de *software*.

Dentro da programação de agentes, existem alguns paradigmas. Um dos paradigmas mais utilizado pelos pesquisadores da área é o modelo *BDI – Beliefs, Desires and Intentions Model*. Segundo tal modelo, um agente possui crenças, desejos e intenções. Baseado neste escopo ele interage com seu ambiente a fim de atingir suas metas. Inspecionando e interagindo com seu ambiente por tentativas de validação de suas regras. Tudo se passa como se o agente, ao encontrar uma mudança, passe a “acreditar” que algo é verdade e, em reação a essa mudança, passe a “desejar” fazer algo segundo uma “intenção” projetada. Do ponto de vista físico, um agente é um programa de computador, escrito em uma linguagem, e que

possui uma coleção de regras a verificar e ações a serem tomadas segundo uma lógica de execução.

Como ponto principal nesse paradigma de programação, podemos salientar a ideia da autonomia de decisão do agente. Com um conjunto de crenças, desejos e intenções ele interage com seu ambiente de execução e escolhe uma ou mais ações compatíveis com suas regras e crenças a fim de atingir seus objetivos.

Podemos definir um agente como sendo *stand-alone* quando é executado e opera sozinho, ou então como pertencendo a um sistema multiagente, quando faz parte de um grupo de agentes situado em um determinado ambiente e que interagem entre si. A possibilidade de um agente interagir com vários outros agentes constituindo um sistema multiagente, é uma terceira propriedade definidora a respeito do que é um agente de *software.* Nesse contexto uma aplicação de *software* pode ser concebida como sendo formada por um sistema de agentes.

Também podemos identificar a diferença entre agentes em agentes de *software* e agentes inteligentes. No contexto da Ciência da Computação agente de *software* é apenas uma aplicação de *software* que possui as características de autonomia, estar situado em ambiente e interagir com outros agentes. Os agentes inteligentes são um dos tipos possíveis de agentes de *software*, dispondo de algum grau de inteligência para sua execução, sendo capaz de raciocinar sobre os elementos percebidos e escolher a melhor forma de ação de acordo com seus conhecimentos a respeito das circunstâncias que se encontra.

As definições para agentes de *software* são inúmeras e costumam gerar controvérsias na comunidade científica, mas pode-se entender que os agentes derivam da aplicação das técnicas da IA no auxílio da realização de uma tarefa especifica [GARCIA; et al., 2003].

Segundo [RUSSEL; NORVIG, 2003], um agente pode ser como qualquer coisa que seja capaz de perceber o seu ambiente através dos seus sensores e agir sobre o mesmo com os meios que lhes foram dispostos. O mesmo estudo também caracteriza um agente racional como um agente que consegue executar uma tarefa corretamente.

2.1.1 Classificação dos Agentes

Existem vários tipos de classificação dos diversos tipos possíveis de agentes. Uma taxonomia de agentes foi proposta por [RUSSEL; NORVIG, 2003] com o intuito de mostrar as diferentes funções e complexidades que um agente pode assumir. São elas:

- **Agentes de reflexo simples:** um sistema de regras do tipo condição-ação é a base da tomada de decisão. Os sensores percebem o ambiente e escolhem a ação mais adequada para realizar uma determinada tarefa. Atuam em locais pouco abrangentes e só realizam ações quando é possível escolher a ação certa,

baseado na simples percepção do ambiente. Tem como principal característica a reatividade.

- **Agentes que rastreiam o mundo:** este modelo requer uma representação do ambiente em possíveis estados. Requer uma abstração detalhada do ambiente em que se encontram onde as mudanças sejam passíveis de serem finitamente representadas. Tem a funcionalidade de prever estados possíveis de alcançar. Com base nesta arquitetura o agente constrói uma crença acerca de qual a melhor ação a efetuar.

- **Agentes baseados em metas:** tem como característica o uso a IA para alcançar metas através da busca e o planejamento. Requer uma estrutura que divide o processo de perceber os ambientes em duas fases. O processo de raciocínio implementa a capacidade de perceber como o ambiente evolui e da capacidade de avaliar o que as ações do agente podem influenciar na evolução. A percepção se dá observando como está o ambiente agora e como ficará o ambiente se o agente executar uma determinada ação.

- **Agentes baseados na utilidade:** seleciona a melhor ação quando está diante de metas em conflito, segundo [RUSSEL; NORVIG, 2003], é constituído por quatro etapas:

    - Identificar como o ambiente está agora;

    - Como o ambiente evoluirá se for executada uma determinada ação;

    - O quanto de sucesso se pode admitir se for escolhido um determinado estado do ambiente;

    - Que ação o agente pode executar agora.

Segundo o estudo de [HYACINTH, 1996], estabeleceu-se uma classificação definindo quatro tipos de agentes baseados nas suas habilidades de cooperar, aprender e agir autonomamente. Eles são denominados por “agentes espertos, agentes colaborativos, agentes de aprendizado colaborativo e agentes de interface” [HYACINTH, 1996]. A Figura 1 mostra como esses quatro tipos utilizam as capacidades descritas acima.

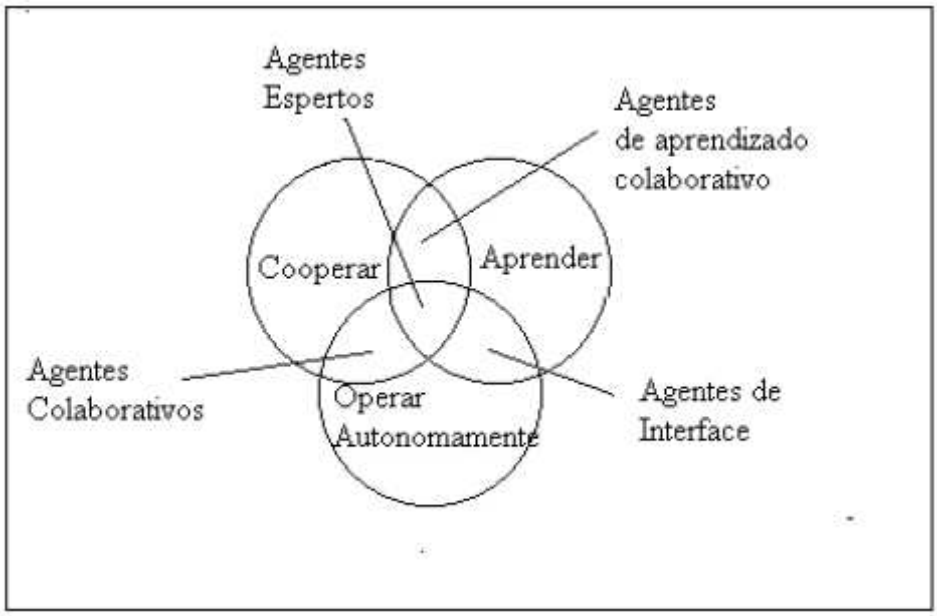


FIGURA 1. Tipologia de Agentes. Adaptação de [HYACINTH, 1996].

As características desses tipos de agentes são as seguintes:

- **Agentes colaborativos:** enfatizam autonomia e cooperação para realizar tarefas por comunicação e possivelmente por negociação com outros agentes possivelmente para atender a mutuo acordo. Esses agentes são usados para problemas distribuídos onde um único agente seria de aplicação praticamente impossível (por exemplo, controle de tráfego aéreo). É crucial para essa classe de agente a existência de uma linguagem de comunicação bem definida.

- **Agentes de interface:** são autônomos e usam aprendizado na realização de tarefas para seus usuários. Essa classe de agentes é usada para implementar assistentes, guias, auxiliares de memorização e filtros; casar e orientar ações ou comprar e vender em conforme o interesse do usuário.

- **Agentes de aprendizado colaborativo:** são processos computacionais capazes de navegar numa rede interagindo com entidades externas colhendo informações para auxilio do usuário e a ele retornando quando uma tarefa de sua responsabilidade é concluída.

- **Agentes espertos:** tem como características a busca, análise e recuperação de grandes quantidades de informação. São ferramentas que ajudam a administrar informações disponíveis em redes como a Internet [CHEONG, 1996]. Eles acessam a rede e procuram tipos particulares de informações, filtrando-as e retornando os resultados a seus usuários. São projetados para aliviar a sobrecarga de informação causada pela disponibilidade de uma grande quantidade de informações pobremente catalogadas.

- **Sistemas agentes heterogêneos:** se referem a coleções de dois ou mais agentes com diferentes arquiteturas de agente. Os agentes irão se comunicar e cooperar entre si. O requisito chave para essa interação é uma linguagem de

comunicação que permita que agentes de diferentes tipos possam se comunicar uns com os outros.

2.1.2 Agentes Inteligentes

Um agente de *software* é visto como uma construção autônoma que é capaz de atuar sem intervenção do usuário possuindo a capacidade de se comunicar com outras entidades e de monitorar e perceber o ambiente em que está inserido. Em suma, podemos elencar as seguintes afirmações para uma definição mais simples: execução autônoma; comunicação com outros agentes e/ou com o usuário e; monitoramento do estado do seu ambiente de execução [HAYES, 1995].

Um agente inteligente de *software* possui algumas características adicionais que são mostradas na Figura 2, tais como: capacidade de explorar conhecimento de domínio; tolerância à entrada de dados errada, inesperada ou estranha; uso de símbolos e abstrações; capacidade de comportamento adaptativo e orientado para uma meta; habilidade de aprender com o ambiente; operação em tempo real; capacidade de se comunicar usando linguagem natural.

Para [HAYES, 1995] os agentes inteligentes devem ser capazes de realizar as seguintes funções: perceber condições dinâmicas no seu ambiente; tomar atitudes para modificar condições no seu ambiente; raciocinar para interpretar percepções, resolver problemas, traçar inferências e determinar ações.

Para outros pesquisadores, agentes inteligentes são: “entidades de *software* que executam um conjunto de operações em beneficio de um usuário com alguma autonomia e empregam ou o conhecimento ou representações das metas e desejos do usuário” [CONTE; GILBERT, 1995].

Agente	Executa autonomamente
	Comunica-se com outros agentes ou com o usuário
	Monitora o estado do seu ambiente de execução
Agente Inteligente	Capaz de usar símbolos e abstrações
	Capaz de explorar quantidades significativas de conhecimento do domínio
Agente Verdadeiramente Inteligente	Capaz de exibir comportamento adaptativo e orientado para metas
	Capaz de aprender com o ambiente
	Tolerante a entradas erradas, inesperadas ou completamente fora de contexto.
	Capaz de comunicar usando linguagem natural

FIGURA 2. Atributos de um agente inteligente. Adaptação de [HAYES, 1995].

Para [WOOLDRIGDE; JENNINGS, 1996] não é apenas exigido autonomia, percepção e reatividade como também a definição para incluir pró-atividade, que torna necessário que o agente seja capaz de exibir comportamento dirigido por metas tomando iniciativas.

Como conclusão, podemos definir que agentes inteligentes devem possuir as características de um agente de *software* (autonomia, comunicabilidade e percepção) adicionadas às características de um agente inteligente (habilidade de manipular conhecimento e tolerar erros, raciocinar com símbolos, aprender e raciocinar em tempo real e comunicar numa linguagem apropriada).

2.1.3 A Arquitetura BDI

As mais importantes arquiteturas de agentes inteligentes deliberativos são baseadas em um modelo de cognição fundamentado em três principais atitudes mentais: as crenças, os desejos, e as intenções. A fundamentação filosófica para esta concepção de agentes vem do trabalho de [DENNET, 1989] sobre sistemas intencionais e de [BRATMAN, 1987] sobre raciocínio prático.

De forma esquemática, a arquitetura BDI genérica está apresentada na Figura 3, conforme proposto em [WOOLDRIGDE, 1999].

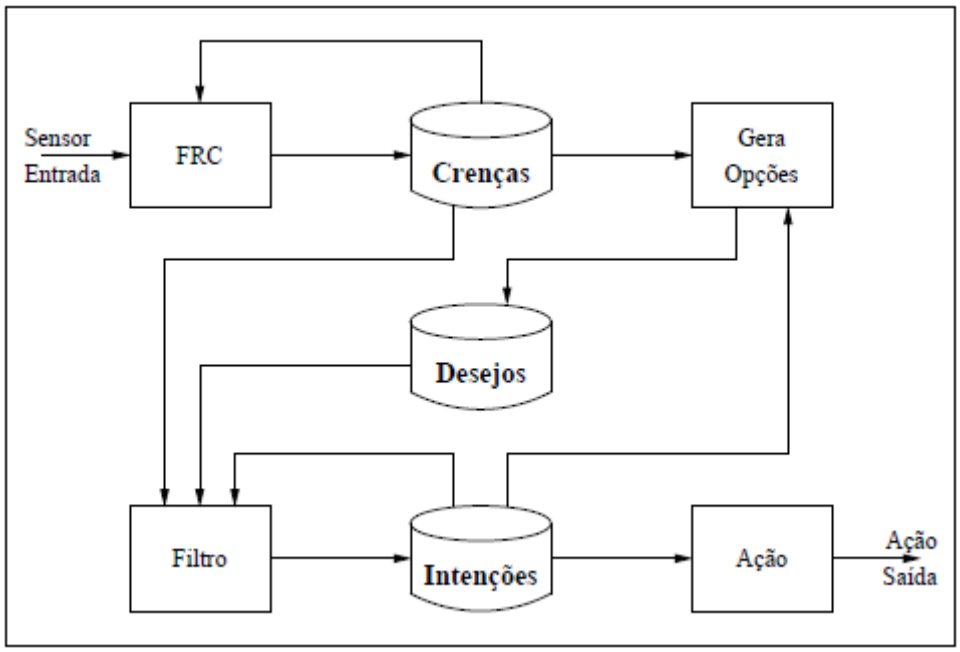


FIGURA 3. Arquitetura BDI genérica [WOOLDRIGDE, 1999].

Esta arquitetura de agente está estruturada da seguinte forma:

- **Crenças:** representam aquilo que o agente sabe sobre o estado do ambiente e dos agentes naquele ambiente (inclusive sobre si mesmo).

- **Desejos:** representam estados do mundo que o agente quer atingir. Em tese, desejos podem ser contraditórios.

- **Intenções:** representam sequências de ações específicas que um agente se compromete a executar para atingir determinados objetivos.

- **FRC:** recebe a informação dos sensores do ambiente e, consultando as crenças anteriores do agente, atualiza essas crenças para que elas reflitam o novo estado do ambiente.

- **Gera Opções:** verifica quais as novas alternativas de estados a serem atingidos, que são relevantes para os interesses particulares daquele agente. A atualização dos objetivos se dá de duas formas: as observações do ambiente possivelmente determinam novos objetivos do agente, e a execução de intenções de mais alto nível pode gerar a necessidade de que objetivos mais específicos sejam atingidos.

- **Filtro:** decidir qual curso de ações específico será usado para alcançar os objetivos atuais do agente e atualizar o conjunto de intenções do agente, com base nas crenças e desejos atualizados e nas intenções já existentes. Esse processo realizado pela função Filtro é normalmente chamado de *deliberação*.

- **Ação:** com o conjunto de intenções já atualizado, a escolha de qual ação específica, entre aquelas pretendidas, será a próxima a ser realizada pelo agente no ambiente. Nos casos em que não é necessário priorização entre múltiplas intenções, a escolha pode ser simples. Entretanto, alguns agentes podem precisar usar escolha de intenções baseadas em critérios mais complexos para garantir que certas intenções sejam priorizadas em relação a outras em determinadas circunstâncias.

2.2 Linguagem AgentSpeak(L)

A linguagem AgentSpeak(L) foi projetada para a programação de agentes BDI na forma de sistemas de planejamento reativos (*reactive planning systems*). Ela foi primeiramente apresentada em [ANAND, 1996] e é uma extensão natural de programação em lógica para a arquitetura de agentes BDI, que representa um modelo abstrato para a programação de agentes e tem sido a abordagem predominante na implementação de agentes inteligentes [WOOLDRIGDE, 1999]. Um agente AgentSpeak(L) corresponde à especificação

de um conjunto de crenças que formarão a base de crenças inicial e um conjunto de planos [BORDINI; et al., 2004].

AgentSpeak(L) distingue dois tipos de objetivos: *objetivos de realização* (*achievement goals*) e *objetivos de teste* (*test goals*). Objetivos de realização e teste são predicados, tais como crenças e expressam que o agente quer alcançar um estado no ambiente onde o predicado associado ao objetivo é verdadeiro. Um objetivo de teste retorna a unificação do predicado de teste com uma crença do agente, ou falha caso não seja possível à unificação com nenhuma crença do agente.

```
+concert(A,V) : likes(A)
   ← !book_tickets(A,V).

+!book_tickets(A,V) : ¬busy(phone)
   ← call(V);
      ...;
      !choose_seats(A,V).
```

FIGURA 4. Exemplos de planos AgentSpeak(L) [BORDINI; et al., 2004].

A Figura 4 apresenta exemplos de planos AgentSpeak(L). O primeiro plano especifica que ao anúncio de um concerto a ser realizado pelo artista A no local V (do Inglês *venue*), correspondendo a adição de uma crença concert(A,V) como consequência da percepção do ambiente. Se for o caso de o agente gostar do artista A, então o agente terá como objetivo a reserva dos ingressos para esse concerto. O segundo plano especifica que ao adotar o objetivo de reservar ingressos, se for o caso de a linha telefônica não estar ocupada, então o agente pode executar o plano que consiste de: executar a ação básica de fazer contato telefônico com o local do concerto V, seguido de um determinado protocolo de reserva de ingressos (indicado por '...'), e que termina com a execução de um subplano para a escolha de acentos em eventos desse tipo naquele local.

2.2.1 Sintaxe Abstrata

A especificação de um agente *ag* em AgentSpeak(L) deve ser feita de acordo com a gramática apresentada na Figura 5. Em AgentSpeak(L), um agente é especificado por um conjunto de crenças *bs* (*beliefs*) correspondendo à base de crenças inicial do agente, e um conjunto de planos *ps* que forma a biblioteca de planos do agente.

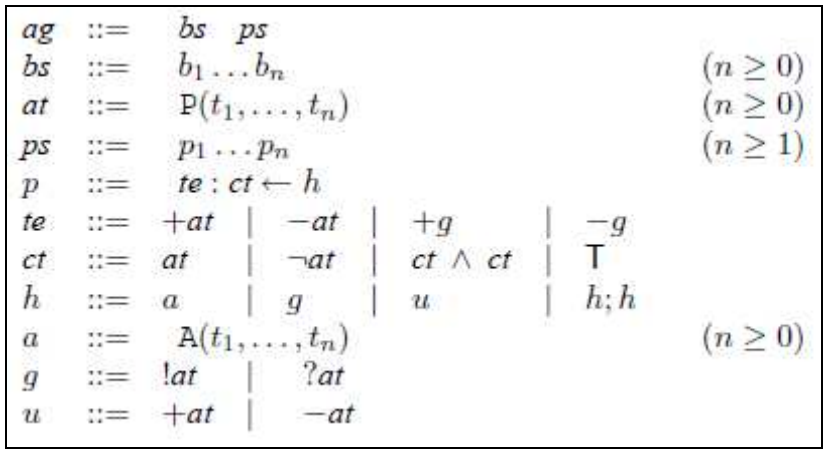


FIGURA 5. Gramática do AgentSpeak(L) [MOREIRA; BORDINI, 2004].

As fórmulas atômicas *at* da linguagem são predicados onde P é um símbolo predicativo e t_1, ... ,t_n são termos padrão da lógica de primeira ordem. Chamamos de *crença* uma fórmula atômica *at* sem variáveis e *b* é meta-variável para crenças. O conjunto inicial de crenças de um programa AgentSpeak(L) é uma sequência de crenças *bs*.

Um plano em AgentSpeak(L) é dado por *p* acima, onde *te* (*triggering event*) é o evento ativador, *ct* é o contexto do plano (uma conjunção de literais de crença) e *h* é uma sequência de ações, objetivos ou atualizações de crenças. A construção *te* : *ct* é dita a *cabeça* do plano, e *h* o *corpo* do plano. O conjunto de planos de um agente é dado por *ps* como uma lista de planos.

Um evento ativador *te* corresponde à adição/remoção de crenças da base de crenças do agente (+*at* e -*at*, respectivamente), ou à adição/remoção de objetivos (+*g* e -*g*, respectivamente). O agente possui um conjunto de *ações básicas* que utiliza para atuar sobre o ambiente. Ações são referidas por predicados usuais com a exceção de que um símbolo de ação A é usado no lugar do símbolo predicativo. Objetivos *g* podem ser objetivos de realização (!*at*) ou de teste (?*at*). Finalmente, +*at* e -*at* (no corpo de um plano) representam operações de atualização (*update*) da base de crença *u*, através da adição ou remoção de crenças respectivamente.

Note que uma fórmula *!g* no corpo de um plano gera um evento cujo evento ativador é *+!g*. Portanto, planos escritos pelo programador que tenha um evento ativador que possa ser unificado com *+!g* representam alternativas de planos que devem ser considerados no tratamento de tal evento. Planos com evento ativador do tipo +*at* e -*at* são utilizados no tratamento de eventos que são gerados quando crenças são adicionadas ou removidas (tanto como consequência da percepção do ambiente, como devido a alterações de crenças explicitamente requisitadas no corpo de um plano). Eventos ativadores do tipo -*!g* são usados para o tratamento de falhas de planos, e eventos ativadores do tipo +?*g* e -?*g* não são utilizados na implementação atual da linguagem.

2.2.2 Semântica Informal

Um interpretador abstrato para a linguagem AgentSpeak(L) precisa ter acesso à base de crenças e à biblioteca de planos, e gerenciar um conjunto de *eventos* e um conjunto de *intenções*. Seu funcionamento requer três *funções de seleção*: a função de seleção de eventos (*SE*) seleciona um único evento do conjunto de eventos; outra função (*SAp*) seleciona uma "opção" (um plano aplicável) entre o conjunto de planos aplicáveis para um evento selecionado; e a terceira função (*SI*) seleciona uma intenção do conjunto de intenções. As funções de seleção são específicas para cada agente, sendo responsáveis por parte significativa do comportamento do agente.

Duas estruturas importantes para o interpretador abstrato são o conjunto de eventos e o conjunto de intenções. *Intenções* são cursos de ações com os quais um agente se compromete para tratar certos eventos. Cada intenção é uma pilha de planos parcialmente instanciados.

Eventos causam o início da execução de planos que tem eventos ativadores relevantes. Eventos podem ser *externos*, quando originados da percepção do ambiente (adição ou remoção de crenças resultantes do processo de revisão de crenças); ou *internos*, quando gerados pela execução de planos do agente (um subobjetivo em um plano gera um evento do tipo "adição de objetivo de realização"). No último caso, o evento é acompanhado da intenção que o gerou (o plano escolhido para aquele evento será colocado no topo daquela intenção). Eventos externos criam novas intenções representando diferentes focos de atenção na atuação do agente no ambiente.

Na Figura 6 podemos ver o fluxo de funcionamento de um interpretador AgentSpeak(L), onde conjuntos de crenças, eventos, planos e intenções são representados por retângulos. Losangos representam a seleção de um elemento de um conjunto. Círculos representam alguns dos processos envolvidos na interpretação de programas AgentSpeak(L).

A cada ciclo de interpretação de um programa AgentSpeak(L), a lista de eventos é atualizada com o resultado do processo de revisão de crenças. Assume-se que as crenças são atualizadas pela percepção e que sempre que houver mudanças na base de crenças do agente, isso implica na inserção de um evento no conjunto de eventos. Essa função de revisão de crenças não é parte de um interpretador AgentSpeak(L), mas é um componente que deve estar presente na arquitetura geral do agente (implementações de interpretadores AgentSpeak(L) tipicamente fornecem uma função de revisão de crenças simples utilizada como *default*).

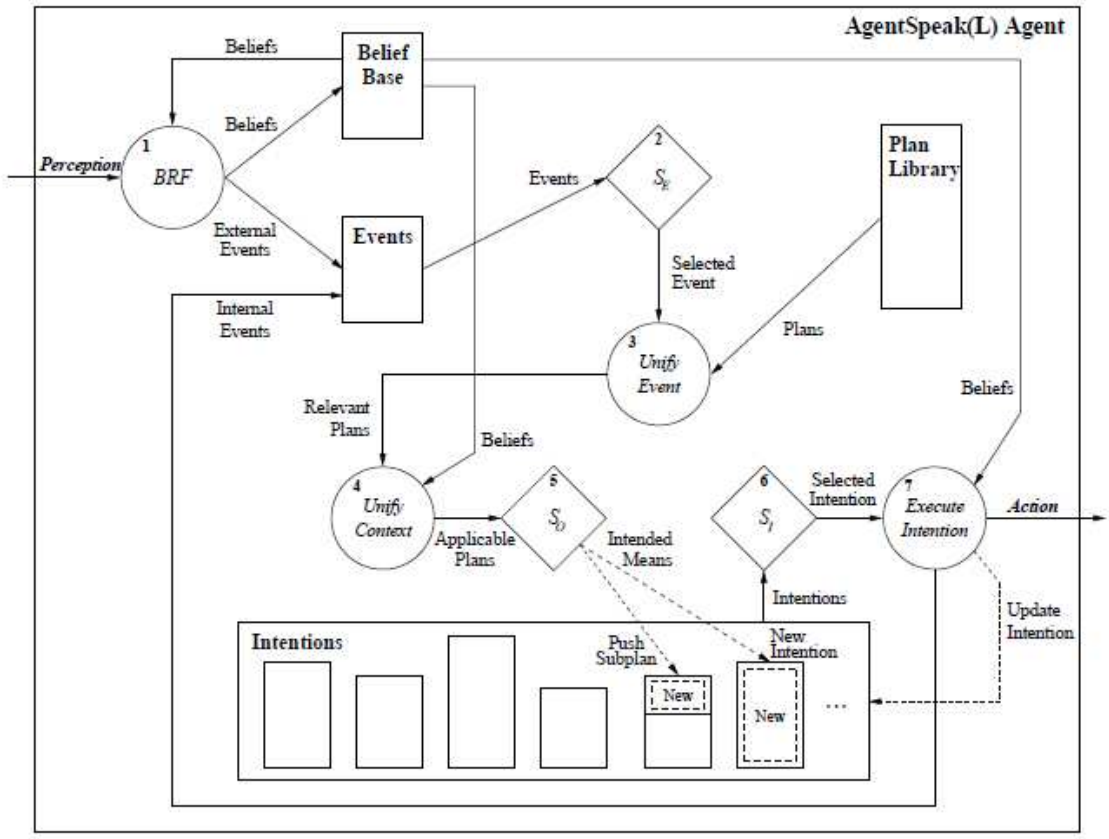


FIGURA 6. Ciclo de interpretação de um programa AgentSpeak(L) [MACHADO; BORDINI, 2002].

Depois de *SE* selecionar um evento, o interpretador AgentSpeak(L) tem que unificar aquele evento com eventos ativadores nos planos presentes na biblioteca de planos. Isso gera um conjunto de todos os *planos relevantes* para o evento escolhido. Pela verificação dos contextos de planos que seguem logicamente das crenças do agente, AgentSpeak(L) determina o conjunto de *planos aplicáveis* (planos que podem ser usados, na situação presente, para tratar o evento selecionado naquele ciclo).

Entre os planos do conjunto de planos aplicáveis, um único plano aplicável que se torna o *meio pretendido* para o tratamento daquele evento, e coloca o plano no topo de uma intenção existente (se o evento for interno), ou cria uma nova intenção no conjunto de intenções (se o evento for externo, gerado por percepção do ambiente), definindo um novo "foco de atenção" do agente. Nesse estágio, resta apenas a seleção de uma única intenção para ser executada no ciclo. A função *SI* seleciona uma intenção do agente (uma das pilhas de planos parcialmente instanciados que se encontram dentro do conjunto de intenções, cada uma representando um dos focos de atenção do agente). No topo dessa intenção existe uma instância de um plano da biblioteca de planos, e a fórmula no início do corpo do plano é executada. Isso implica em uma ação básica a ser realizada pelo agente no ambiente, na geração de um evento interno (se a fórmula selecionada for um objetivo de realização) ou na execução de um objetivo de teste (através do acesso à base de crenças). Caso a fórmula seja um objetivo de realização, simplesmente um evento do tipo "adição de objetivo de realização" é adicionado ao conjunto de eventos, acompanhado da intenção que gerou o evento. Essa intenção tem que ser removida do conjunto de intenções, pois ela fica suspensa até que o evento interno seja escolhido pela função *SE*. Quando uma instância de plano for escolhida como meio pretendido para tratar este evento, o plano é colocado no topo da pilha de planos

daquela intenção, e ela é retornada para o conjunto de intenções (podendo novamente ser selecionada por *SI*).

Se a fórmula a ser executada é a realização de uma ação básica ou a execução de um objetivo de teste, a fórmula deve ser removida do corpo da instância de plano que se encontra no topo da intenção selecionada. No caso da execução de um objetivo de teste, a base de crenças será inspecionada para encontrar um átomo de crença que unifica com o predicado de teste. Se uma unificação for possível, instanciações de variáveis podem ocorrer no plano parcialmente instanciado; após isto, o objetivo de teste pode ser removido do conjunto de intenções, pois já foi realizado. No caso de uma ação básica a ser executada, o interpretador simplesmente informa ao componente da arquitetura do agente que é responsável pela atuação sobre o ambiente qual ação é requerida, podendo também remover a ação do conjunto de intenções. Quando todas as fórmulas no corpo de um plano forem removidas (tiverem sido executadas), o plano é removido da intenção, tal como o objetivo de realização que o gerou, se esse for o caso, é removido do início do corpo do plano abaixo daquele na pilha de planos daquele foco de atenção. O ciclo de execução termina com a execução de uma fórmula do corpo de um plano pretendido, e AgentSpeak(L) começa um novo ciclo, com a verificação do estado do ambiente após a ação do agente sobre ele, a geração dos eventos adequados, e continuando a execução de um ciclo de raciocínio do agente como descrito acima.

2.3 Jason

Jason [BORDINI; et al., 2007] é uma plataforma de desenvolvimento de sistemas multiagentes baseada em um interpretador para uma versão estendida da linguagem AgentSpeak(L) e também oferece uma série de extensões que são necessárias para o desenvolvimento de tais sistemas.

Ele é implementado em Java e está disponível em código aberto. Sua primeira versão operacional foi liberada 2004. A Figura 7 mostra a tela principal do *software*.

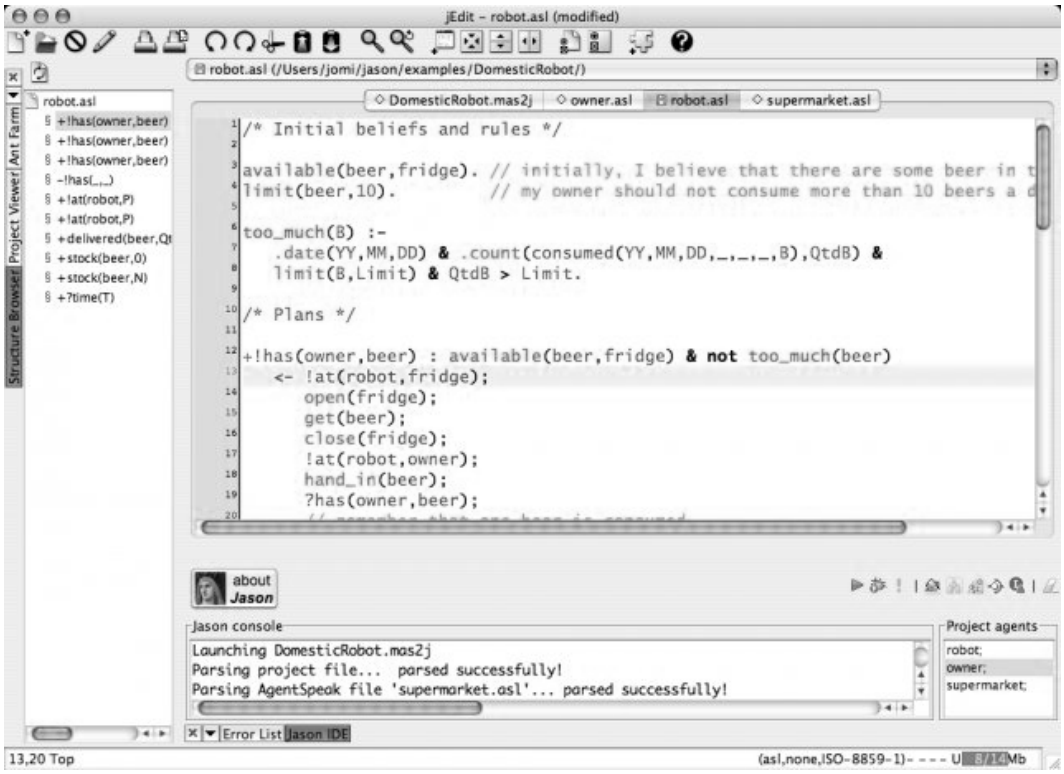


FIGURA 7. Tela principal do Jason.

A plataforma, genericamente, contém as seguintes características [BORDINI; et al., 2004]:

- **Distribuição**: a plataforma torna fácil a definição dos agentes que participam do sistema e também determinar em quais máquinas cada um vai rodar, se a distribuição for necessária. Atualmente, dois tipos de infraestrutura estão disponíveis: um que executa todos os agentes na mesma máquina, e outra que permitem a distribuição usando SACI.

- **Ambientes**: sistemas multiagentes normalmente serão implantados em algum ambiente do mundo real. Fornece suporte para desenvolvimento de ambientes, que são programados em Java ao invés de uma linguagem de agentes.

- **Customização**: os programadores podem customizar as funções de seleção, funções de confiança e arquitetura geral do agente (incluindo percepções, ações, revisão de crenças e comunicação entre agentes).

- **Extensibilidade da linguagem e código legado**: a extensão do AgentSpeak disponibilizada com o Jason tem uma construção chamada “ações internas”. Permite a extensibilidade da linguagem diretamente pelas ações internas definidas pelo usuário, que também é uma maneira simples de invocar código legado dentro do alto nível de raciocínio do agente de uma forma elegante. Além de ações definidas pelo usuário, Jason vem com uma biblioteca interna de ações de nível fundamental.

- **IDE (*Integrated Development Environment***): Jason é distribuído com uma IDE que oferece uma interface gráfica para o gerenciamento de projetos (sistemas multiagentes), com edição do código-fonte dos agentes, e o funcionamento do sistema. Outra ferramenta fornecida como parte do IDE permite ao usuário inspecionar os estados dos agentes quando o sistema está em execução.

2.4 Redes Bayesianas

Os estudos em inteligência artificial podem ser divididos em duas grandes áreas: o desenvolvimento de sistemas que agem como humanos e o desenvolvimento de sistemas que agem racionalmente.

Dentro do contexto dos sistemas que agem racionalmente, duas abordagens principais podem ser utilizadas: raciocínio lógico e raciocínio probabilístico. O raciocínio lógico pondera sobre o conhecimento prévio a respeito do problema e, sobre esta base de conhecimento retira suas conclusões. Esta abordagem pode não ser útil em situações onde não se conhece previamente todo o escopo do problema, para estes casos, o raciocínio probabilístico surge como uma boa opção.

Um sistema que possa atuar em situações de incerteza deve ser capaz de atribuir níveis de confiabilidade para todas as sentenças em sua base de conhecimento, e ainda, estabelecer relações entre as sentenças. As Redes Bayesianas [PEARL, 1993] oferecem uma abordagem para o raciocínio probabilístico que engloba teoria de grafos, para o estabelecimento das relações entre sentenças e ainda, teoria de probabilidades, para a atribuição de níveis de confiabilidade.

2.4.1 Raciocinando Sob Incertezas

“A principal vantagem de raciocínio probabilístico sobre raciocínio lógico é fato de que agentes podem tomar decisões racionais mesmo quando não existe informação suficiente para se provar que uma ação funcionará” [CHARNIAK, 1991].

Alguns fatores podem condicionar a falta de informação em uma base de conhecimento, os principais são:

- **Impossibilidade**: Em alguns casos, o trabalho exigido para a inserção de todos os antecedentes ou consequentes que configurem uma base de conhecimento onde quaisquer inferências a respeito do domínio do problema podem ser efetuadas, pode ser muito oneroso.

- **Ignorância Teórica**: Em alguns casos não se possui o conhecimento de todo domínio do problema.

Lidar com falta de informação significa lidar com incertezas. Nestes ambientes é necessário utilizar conectivos que manipulem níveis de certeza e não apenas valores booleanos, verdadeiro (1) e falso (0). Assim, seria possível, por exemplo, tratar uma expressão do tipo: “Eu tenho probabilidade 0.8 de fazer um bom trabalho de IA” ou “A probabilidade de um trabalho de IA ser bom é 0.5” ou “A probabilidade de um bom trabalho de IA tirar A é 0.9”. Com isto, uma possível questão seria: “Quais são as minhas chances de tirar A?”. Para responder a questão é necessário se estabelecer uma regra que combine as três probabilidades. Em outras palavras, é necessária uma função que retorne um valor dadas as probabilidades, 0.8, 0.5 e 0.9. Comparativamente poderíamos dizer que lógica probabilística estende lógica proposicional, no sentido que, probabilidade 1 representa verdadeiro e probabilidade 0 representa falso.

2.4.2 A Regra de Bayes

Antes de definirmos o que é a Regra de Bayes e sua aplicação, precisamos definir probabilidades condicionais e incondicionais.

O conceito de probabilidade condicional envolve dois eventos, A (com P(A) > 0) e B em uma situação em que se sabe que o evento A ocorreu. Logo, a probabilidade condicional é definida como a análise da ocorrência de um evento B dado à ocorrência de um evento A. Na probabilidade incondicional não há dependência de variáveis, ou seja, deve ser analisada a probabilidade de um evento A ocorrer, sem levar em consideração qualquer outro evento.

A Regra de Bayes ou Teorema de Bayes é obtido diretamente da aplicação da regra do produto na álgebra da teoria de probabilidades, sendo mostrado na Figura 8, em variáveis aplicáveis para o caso em estudo [ALMEIDA; VALE, 1999]:

$$f(\mathbf{s},\mathbf{b}|\mathbf{x}) = \frac{f(\mathbf{x}|\mathbf{s},\mathbf{b}) f_0(\mathbf{s},\mathbf{b})}{\int f(\mathbf{x}|\mathbf{s},\mathbf{b}) f_0(\mathbf{s},\mathbf{b}) d\mathbf{s} d\mathbf{b}}$$

FIGURA 8. Teorema de Bayes [ALMEIDA; VALE, 1999].

Para aplicação da Regra de Bayes precisamos de três termos: uma probabilidade condicional e duas incondicionais. Vamos considerar um exemplo de diagnostico médico:

“um médico sabe que a meningite causa torcicolo em 50% dos casos. Porém, o médico também conhece algumas probabilidades incondicionais que dizem que, um caso de meningite atinge 1/50000 pessoas e, a probabilidade de alguém ter torcicolo é de 1/20.”.

Considere T e M, respectivamente, como a probabilidade incondicional de um paciente ter torcicolo e a probabilidade incondicional de um paciente ter meningite. Assim:

$P(T|M) = 0.5$ (probabilidade de ter torcicolo tendo meningite)

$P(M) = 1/50000$

$P(T) = 1/20$

Aplicando a Regra de Bayes:

$P(M|T) = (P(T|M)P(M))/P(T) = (0.5 \times 1/50000)/(1/20) = 0.0002$

Ou seja, é esperado que apenas 1 em 5000 pacientes com torcicolo tenha meningite. Note que mesmo tendo torcicolo uma alta probabilidade nos casos de meningite (0.5), a probabilidade de um paciente ter meningite continua pequena, devido ao fato de a probabilidade incondicional de torcicolo ser muito maior que a probabilidade de meningite.

Uma argumentação válida surge do fato de que o médico poderia também possuir a probabilidade incondicional P(M|T), a partir de amostras de seu universo de pacientes, da mesma forma que P(T|M), evitando o cálculo realizado anteriormente. Porém, imagine que um surto de meningite aflija o universo de pacientes do médico em questão, aumentando o valor de P(M). Caso P(M|T) tenha sido calculado estatisticamente a partir de observações em seus pacientes, o médico não terá nenhuma ideia de como este valor será atualizado (visto que P(M) aumentou). Entretanto, caso tenha realizado o cálculo de P(M|T) em relação aos outros três valores (como demonstrado) o médico verificará que P(M|T) crescerá proporcionalmente em relação a P(M).

Considere agora que um paciente pode estar com problema de coluna C, dado que está com torcicolo: $P(C|T) = (P(T|C)P(C))/P(T)$.

Utilizando P(M|T) podemos calcular a probabilidade relativa de C em M dado T, ou, em outras palavras, a marginalização de M e C. Considerando que $P(T|C) = 0.8$ e $P(C) = 1/1000$. Então:

$P(M|T)/P(C|T) = (P(T|M)P(M))/(P(T/C)P(C)) = (0.5 \times 1/50000)/(0.8 \times 1/1000) = 1/80$

Isto é, Problema de coluna C é 80 vezes mais comum que meningite M, dado torcicolo.

2.4.3 Redes Bayesianas

Matematicamente, uma Rede Bayesiana é uma representação compacta de uma tabela de conjunção de probabilidades do universo do problema [PEARL, 1993; RUSSEL; NORVIG, 2003]. Por outro lado, do ponto de vista de um especialista, Redes Bayesianas constituem um modelo gráfico que representa de forma simples as relações de causalidade das variáveis de um sistema.

Uma Rede Bayesiana consiste do seguinte:

- Um conjunto de variáveis e um conjunto de arcos ligando as variáveis.
- Cada variável possui um conjunto limitado de estados mutuamente exclusivos.
- As variáveis e arcos formam um grafo dirigido sem ciclos (DAG).
- Para cada variável A que possui como pais B_1, ..., B_n, existe uma tabela P(A| B_1, ..., B_n).

Repare que, caso A não possua um pai, a tabela de probabilidades é reduzida para uma probabilidade incondicional P(A). Uma vez definida a topologia da rede, basta especificar as probabilidades dos nós que participam em dependências diretas, e utilizar estas para computar as demais probabilidades que se deseje.

Conforme o exemplo mostrado em [RUSSEL; NORVIG, 2003], considere o seguinte domínio, relacionado a alarmes contra roubos: “Você possui um novo alarme contra ladrões em casa. Este alarme é muito confiável na detecção de ladrões, entretanto, ele também pode disparar caso ocorra um terremoto. Você tem dois vizinhos, João e Maria, os quais prometeram telefonar-lhe no trabalho caso o alarme dispare. João sempre liga quando ouve o alarme, entretanto, algumas vezes confunde o alarme com o telefone e também liga nestes casos. Maria, por outro lado, gosta de ouvir música alta e às vezes não escuta o alarme.” Este domínio pode ser representado como apresenta a Figura 9.

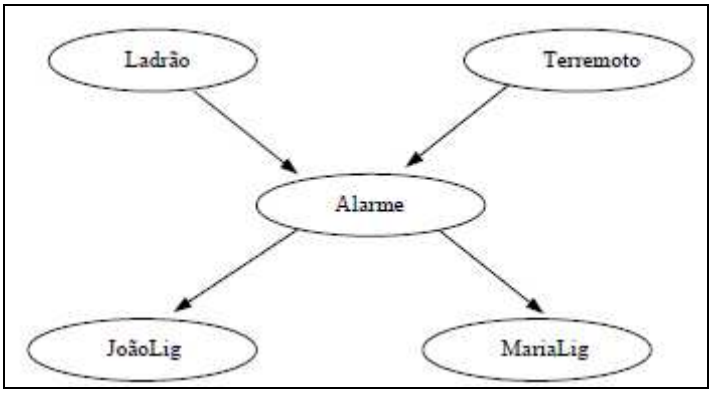


FIGURA 9. Rede Bayesiana para o domínio de alarmes contra roubo [RUSSEL; NORVIG, 2003].

Repare que a rede não possui nós indicando que Maria está ouvindo música, ou que o telefone está tocando e atrapalhando o entendimento de João. Estes fatos estão implícitos, associados à incerteza relacionada pelos arcos *Alarme→ JoãoLig* e *Alarme→MariaLig*, pois, poderia ser muito dispendioso (ou até mesmo impossível) se obter tais probabilidades, ou ainda, tais informações poderiam não ser relevantes. Assim, as probabilidades devem resumir as condições em que o alarme pode falhar (falta de bateria, campainha estragada, etc.) e ainda, as condições em que João e Maria podem falhar (não estava presente, não ouviu o alarme, estava de mau-humor, etc.). Desta forma, o sistema pode lidar com uma vasta gama de possibilidades, ao menos de forma aproximada.

Uma vez definida a topologia, é necessário se definir a tabela de probabilidades condicionais (CPT) para cada nó. Cada linha na tabela contém a probabilidade condicional para cada caso condicional dos nós pais. Um caso condicional é uma possível combinação dos valores para os nós pais. Para a variável aleatória *Alarme* temos a Tabela 2.

Ladrão	Terremoto	*P(Alarme|Ladrão,Terremoto)*	
		Verdadeiro	Falso
Verdadeiro	Verdadeiro	0.95	0.050
Verdadeiro	Falso	0.95	0.050
Falso	Verdadeiro	0.29	0.71
Falso	Falso	0.001	0.999

TABELA 1. CPT Alarme adaptado de [RUSSEL; NORVIG, 2003].

A Figura 10 apresenta a Rede Bayesiana para o domínio de alarmes contra roubos e suas probabilidades condicionais.

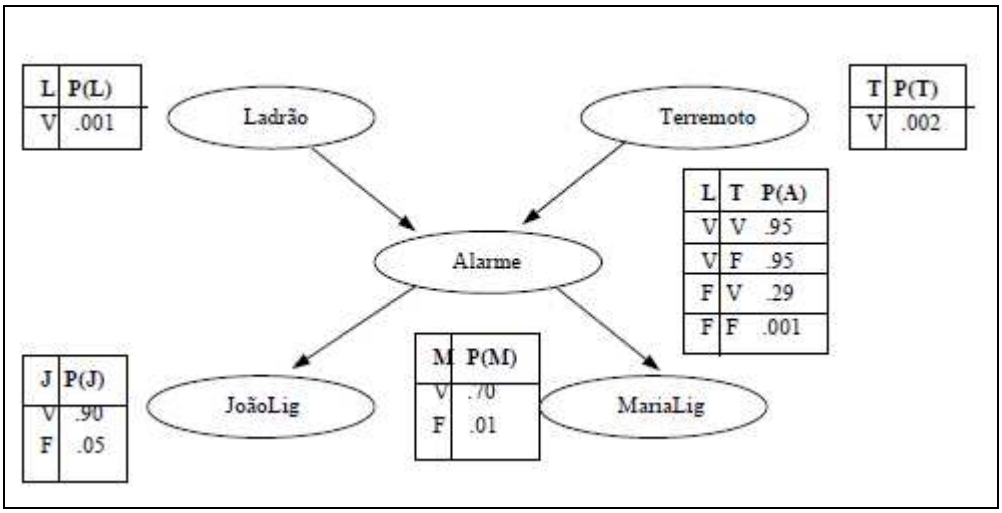


FIGURA 10. Rede Bayesiana e probabilidades para o domínio de alarmes contra roubos [RUSSEL; NORVIG, 2003].

2.4.4 Inferência em Redes Bayesianas

A tarefa básica de uma inferência probabilística é computar a distribuição de probabilidades posterior para um conjunto de variáveis de consulta, dada uma variável de evidência. Para o exemplo mostrado na Figura 10, *Ladrão* constitui uma boa variável de consulta e *JoãoLig*, *MariaLig* seriam boas variáveis de evidência. Dado que temos a evidência que João e Maria fizeram uma ligação, qual a probabilidade de haver um ladrão na casa e não ter acontecido um terremoto?

Neste caso queremos descobrir P (*Ladrão* = *Verdadeiro* | *JoãoLig* = *Verdadeiro*, *MariaLig* = *Verdadeiro*), ou, P (L | J ∧ M). Entre a evidência e a variável que queremos descobrir temos ainda a variável Alarme. Portanto vamos considerar a probabilidade que João fez uma ligação dado que o alarme tocou (P (J | A)) e também que Maria fez uma ligação dado que o Alarme tocou (P (M | A)). Também temos que considerar a probabilidade de Alarme ser verdadeiro quando a variável Ladrão for verdadeira e a variável Terremoto for falsa, ou P (A | L ∧ ¬T).

Temos então:

$$P (L | J \wedge M) = P (J | A) . P (M | A) . P (A | L \wedge \neg T) . P (L) . P (\neg T)$$

$$P (L | J \wedge M) = 0,9 . 0,7 . 0,95 . 0,001 . 0,998$$

$$P (L | J \wedge M) = 0,000597303 \approx 0,0597 \%$$

Através do resultado descobrimos que há uma probabilidade de aproximadamente 0,06 % de haver um ladrão na casa e não ter acontecido um terremoto, dado que João e Maria fizeram uma ligação.

Várias outras inferências podem ser realizadas sobre Redes Bayesianas para:

- **Diagnósticos**: Dos efeitos para as causas. Dado *JoaoLig*, P(*Ladrão*|*JoaoLig*)
- **Causas**: De causas para efeitos. Dado *Ladrão*, P(*JoaoLig*| *Ladrão*)
- **Inter causais**: Entre causas de um efeito comum. Dado *Alarme*, P(*Ladrão*|*Alarme*) e dado *Terremoto*, P(*Ladrão*|*Alarme*,*Terremoto*).

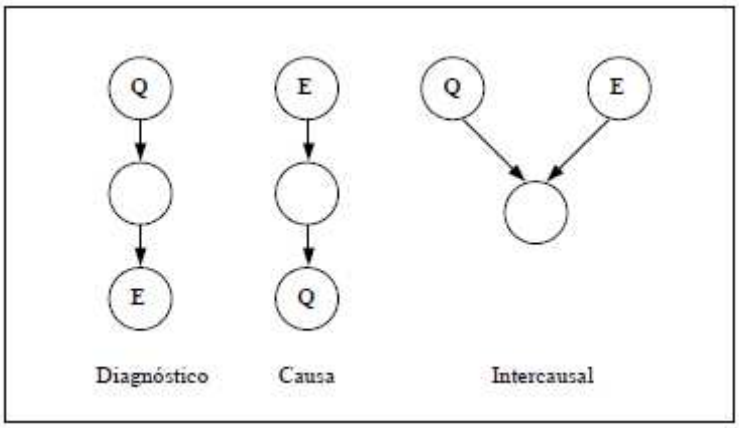


FIGURA 11. Tipos de inferências.

E, além de consultas a partir de evidências, Redes Bayesianas podem ser utilizadas para:

- Tomar decisões baseadas em probabilidades.
- Decidir quais evidências adicionais devem ser observadas a fim de se obter informações úteis do sistema.
- Analisar o sistema a fim de buscar os aspectos do modelo que possuem maior impacto sob as variáveis de consulta.
- Explicar os resultados de uma inferência probabilística ao usuário.

2.4.5 *BNJ – Bayesian Network tools in Java*

Para montar, calcular e inferir as Redes Bayesianas envolvidas nos agentes programados através da linguagem proposta por este trabalho foi necessário a utilização de ferramentas de apoio, que visaram aumentar o desempenho das modificações implementadas e diminuir o tempo de processamento dos agentes criados. Uma das ferramentas utilizadas foi o BNJ (*Bayesian Network tools in Java*). Através dele foi possível criar e inferir em Redes Bayesianas de forma prática e rápida, obtendo resultados satisfatórios.

BNJ é um *software* para modelagem gráfica de Redes Bayesianas. Ele foi desenvolvido em Java e é o*pen-source.* O público alvo do BNJ são pesquisadores e desenvolvedores que necessitam desenvolver, testar e inferir modelos gráficos de probabilidade baseado em Redes Bayesianas [BNJ, 2010].

Este *software* também permite a criação de Redes Bayesianas através de arquivos textos, o que permite utilizá-lo em conjunto com outras ferramentas e/ou integrá-lo a outros

softwares a fim de possibilitar o uso das variáveis de uma determinada rede probabilística em algum outro sistema de dados.

O BNJ oferece mais de dez opções para algoritmos de inferência, o que dá mais flexibilidade ao desenvolver na hora de escolher o melhor caminho a tomar e o melhor resultado frente aos cálculos realizados pelos motores de inferência.

Na Figura 12 é mostrada a tela principal do BNJ, onde é possível ter o controle para criar, deletar e manipular os nós de uma determinada Rede Bayesiana.

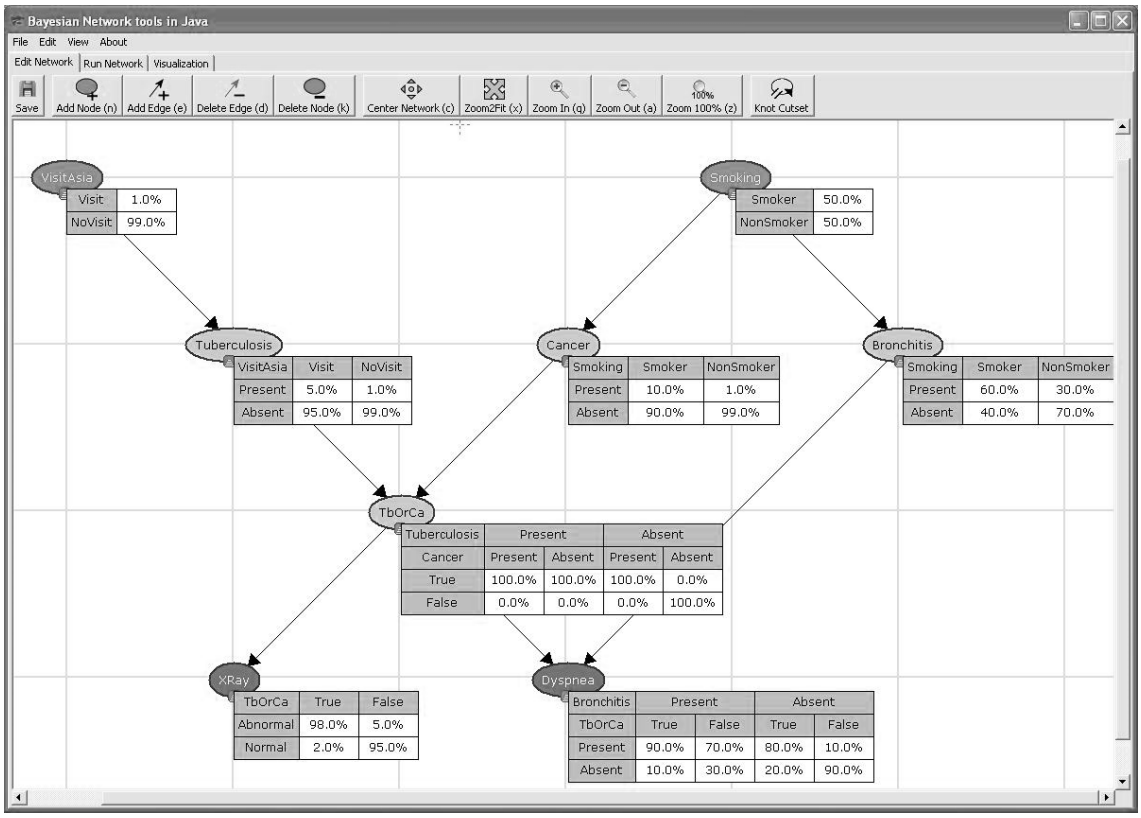


FIGURA 12. Tela principal do *software* BNJ [BNJ, 2010].

3 Trabalhos Relacionados

Neste capítulo será realizada uma análise de trabalhos da área de agentes de *software* e redes probabilísticas frente a proposta desta dissertação. A ideia é fazer um levantamento sobre o contexto científico atual na área de estudo e verificar o diferencial deste trabalho em relação aos demais a fim de garantir que a proposta desta dissertação seja reconhecida como uma contribuição científica pela comunidade acadêmica.

Os critérios para escolhas dos artigos foram baseados nos assuntos envolvidos com este trabalho. Basicamente o filtro continha Agentes de *Softwares,* Redes Probabilísticas e Linguagens e Ferramentas de Programação. Um resumo de cada artigo é apresentado juntamente com comentários, contrapondo como as ideias expostas por esta dissertação.

Ao final deste capítulo será apresentada uma tabela comparativa, a fim de elencar as principais diferenças entre o presente trabalho e os artigos explanados onde o objetivo é corroborar as ideias aqui propostas como sendo factíveis e de extrema utilidade para a comunidade de programação orientada a agentes.

Neste capítulo serão mostrados de forma resumida três trabalhos relacionados ao assunto [FAGUNDES, 2007], [KERSTING; RAEDT, 2000] e [MILCH; KOLLER, 2000] deste estudo que de alguma forma contribuem para o desenvolvimento do projeto.

3.1 *Integrating BDI Model and Bayesian Networks*

No trabalho [FAGUNDES, 2007] é dito que as linhas de pesquisa da Inteligência Artificial têm proposto abordagens para a resolução de inúmeros problemas complexos do mundo real, incluindo o paradigma orientado a agentes que provê agentes autônomos, capazes de perceber os seus ambientes, reagir de acordo com diferentes circunstâncias e estabelecer interações sociais com outros agentes de *software* ou humanos. Também é afirmado que as Redes Bayesianas fornecem uma maneira de representar graficamente as distribuições de probabilidades condicionais e permitem a realização de raciocínios probabilísticos baseados em evidências. O objetivo deste trabalho é a integração do modelo de agentes BDI (*Belief-Desire-Intention*) e das Redes Bayesianas, utilizando uma abordagem baseada em ontologias para representar o conhecimento incerto dos agentes.

No trabalho foi desenvolvida uma ontologia para representar a estrutura das Redes Bayesianas. Esta ontologia tinha como principal objetivo permitir a interoperabilidade ente os agentes compatíveis com a arquitetura proposta. No entanto, a ontologia também facilitaria o entendimento necessário para abstrair os estados mentais e processos cognitivos dos agentes através de elementos das Redes Bayesianas. Uma vez construída a ontologia, a mesma foi

integrada com a arquitetura BDI. Através da integração do modelo BDI com as Redes Bayesianas foi obtida uma arquitetura cognitiva de agentes capaz de deliberar sob incertezas.

A integração é composta da abstração dos estados mentais através de elementos das Redes Bayesianas, seguida da especificação do processo deliberativo. Finalmente, foi desenvolvido um estudo de caso, que consistiu na aplicação da arquitetura proposta no Agente Social, um componente de um portal educacional multiagente. A linguagem OWL (*Web Ontology Language*) [W3C, 2011] foi usada para codificar as ontologias demonstradas ao longo do texto.

3.2 *Bayesian Logic Programs*

Para [KERSTING; RAEDT, 2000], uma Rede Bayesiana especifica a distribuição de probabilidade sobre um conjunto fixo de variáveis discretas aleatórias. Redes Bayesianas provêm uma extensão probabilística elegante da lógica proposicional. Entretanto, as Redes Bayesianas herdam também as limitações da lógica proposicional. Essas limitações motivaram o desenvolvimento de mecanismos de representação de conhecimento empregando lógica de primeira ordem, como lógica de programação e a linguagem Prolog. O trabalho explora a teoria para construção de extensões de primeira ordem para Redes Bayesianas, tais como: programas probabilísticos lógicos, relação de Redes Bayesianas e modelos relacionais de probabilidade.

Programas de lógica *bayesiana* consistem em duas partes. A primeira parte é a lógica que consiste em um conjunto de cláusulas *bayesianas* que capturam a estrutura qualitativa do domínio e sua base. A segunda parte é a quantitativa que consiste na informação quantitativa sobre o domínio e o emprego de noções da tabela de probabilidade condicional e regras de combinação.

Cada programa de lógica *bayesiana* especifica uma Rede Bayesiana proposicional que pode ser adquirida através de um mecanismo de inferência de Rede Bayesiana. A estrutura da Rede Bayesiana segue a semântica da lógica de programação, onde aspectos quantitativos são manipulados nas tabelas de regras de probabilidades condicionais e regras combinacionais.

O estudo introduziu os modelos de programação lógica em Redes Bayesianas e também alguns *frameworks*. Cada *framework* foi descrito e visto ponto a ponto, de forma a permitir que uma análise minuciosa fosse realizada para efeitos comparativos de cunho teórico.

3.3 *Probabilistic Models for Agent's Beliefs and Decisions*

Segundo [MILCH; KOLLER, 2000], quando um sistema inteligente interage com um agente, ele frequentemente precisa saber as crenças desses agentes, bem como seus processos de tomada de decisão. O problema central em muitos domínios é a predição, isto é, descobrir o que os agentes irão ou poderão fazer no futuro. Desde que um agente tome suas decisões baseadas em suas crenças e suas preferências, descobrir seu estado mental é essencial para fazer algumas predições.

O autor supõe, por exemplo, que esta desenvolvendo um sistema para ajudar analistas e policiais a descobrir crises internacionais. Em um exemplo de cenário, tem que o Iraque adquiriu Antrax e começou a desenvolver mísseis capazes de levar o Antrax até o Oriente Médio. Há uma vacina que os EUA esta administrando em suas tropas. Mas por razões éticas, os EUA ainda não terminaram os estudos controlados da efetividade da vacina. O Iraque, por outro lado, esta tentando alguns testes de suas novas armas. O propósito do Iraque é desenvolver um míssil equipado com Antrax para atirar contra a força aérea americana na Turquia e na Arábia Saudita. Entretanto, para o Iraque desenvolver tal arma, ele precisará utilizar fábricas de armamentos antigas, que podem ser vistas pelos satélites americanos. O autor quer o sistema inteligente saiba responder as seguintes perguntas: "Se nós observamos que o Iraque adquiriu Antrax, qual é a probabilidade da vacina ser efetiva?" e "O Iraque acredita que se eles começaram a desenvolver um míssil para levar o Antraz, os EUA irão saber que eles adquiriram Antrax?".

Segundo [MILCH; KOLLER, 2000], a fórmula epistêmica clássica dos agentes, não permite que eles quantifiquem uma determinada fórmula, ou eles sabem ou não sabem. A lógica probabilística das crenças retira essa limitação. Entretanto, um agente com uma base de crenças probabilísticas requer a análise de todos os estados possíveis das variáveis do domínio. Essa análise é exponencial e computacionalmente infinita de se obter. A proposta é utilizar inferência epistêmica probabilística, para não utilizar a enumeração exponencial do número de estados do agente.

Em muitos domínios os agentes não só observam passivamente o mundo a sua volta e constroem crenças, eles também tomam decisões e ações. Em muitos sistemas de descobertas probabilísticas existentes, especialistas definem as condições de distribuição de probabilidade que descrevem os passos de cada possível ação do agente, dada uma lista de variáveis relevantes para o processo de tomada de decisão. Mas essa técnica torna necessário que os especialistas entendam como os agentes tomam suas decisões e isso é um processo complexo. Para contornar isto a ideia é modelar o processo de decisão de um agente através de um diagrama de influencia e converte-lo em uma Rede Bayesiana. Isso permite descobrir o modelo de tomada de decisão sobre o curso das ações, e usar essas ações para alcançar conclusões sobre aspectos não observados no mundo.

Na lógica epistêmica probabilística (PEL) proposta, foi assumida que os agentes possuem uma prioridade de distribuição de probabilidade comum sobre os estados do mundo, e a distribuição de probabilidade local de um determinado estado é igual à distribuição global condicionada no conjunto de estados que o agente considera possível.

As Redes Bayesianas provêm uma representação compacta de um espaço complexo de probabilidades. Podem-se fazer representações simples de modelos de classes PEL com essas redes. A Figura 13 mostra o exemplo da crise, citado anteriormente.

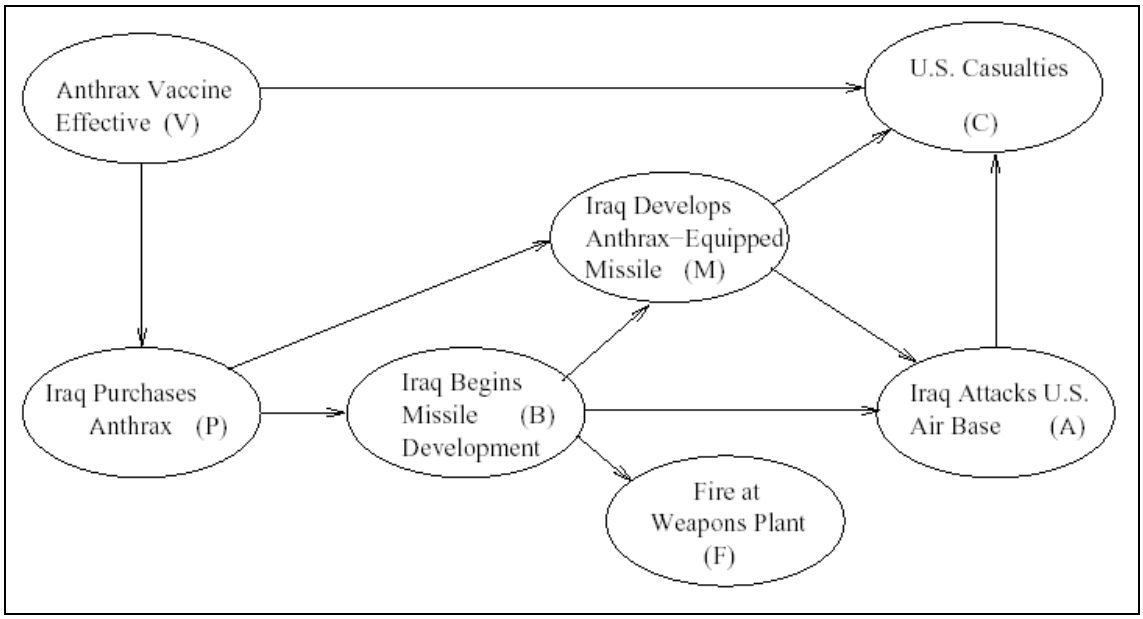


FIGURA 13. Rede Bayesiana básica para o problema da crise [MILCH; KOLLER, 2000].

3.4 Análise dos Trabalhos Relacionados

No artigo [FAGUNDES, 2007] é apresentado o uso de programas lógicos baseados em Redes Bayesianas, onde variáveis são calculadas sobre outras variáveis, encadeando probabilidades e obtendo estimativas sobre os dados. Tudo isso em uma estrutura de ontologias implementada e integrada através de ferramentas de programação existentes. O artigo [KERSTING; RAEDT, 2000] mostra uma lógica probabilística aplicada a agentes, onde as crenças e decisões de cada agente podem ser tomadas levando em consideração o resultado de um cálculo probabilístico. Por último, o artigo [MILCH; KOLLER, 2000] apresenta um estudo de caso de uma linguagem probabilística existente, mostrando exemplos e comprovando-os de forma teórica.

Os três estudos se relacionam com os assuntos tratados no presente trabalho. A partir desses assuntos foi elaborada uma Tabela Comparativa 3, a fim de visualizar as diferenças entre os trabalhos e a presente pesquisa. Na última linha são apresentadas as principais características da presente dissertação em relação aos demais trabalhos.

Trabalho	Suporte ao modelo BDI	Uso de Redes Bayesianas	Uso de frameworks	Linguagem de programação	Ambiente integrado de programação
Integrating BDI Model and Bayesian Networks	X	X	X		
Bayesian Logic Programs		X	X		
Probabilistic Models for Agent's Beliefs and Decisions	X	X			
Proposta de AgentSpeak(PL)	X	X	X	X	X

TABELA 2. Tabela comparativa de trabalhos.

Os três trabalhos estão claramente ligados aos assuntos tratados na dissertação, podendo aprimorar o processo de pesquisa, indicando caminhos a serem percorridos para atingir um método prático e bem fundamentado de programação orientada a agentes baseados em crenças probabilísticas.

Na primeira coluna da tabela apresentada é focado no suporte ao modelo BDI para projeto e implementação de agentes de *software*. Podemos observar que, exceto o trabalho de [KERSTING; RAEDT, 2000] focado nas questões de programação em lógica, todos os demais trabalhos incluindo a presente dissertação suportam os conceitos do modelo BDI e programação baseada em agentes. Todos os trabalhos também suportam a representação de modelos probabilísticos através de Redes Bayesianas, convergindo com a proposta de trabalho atual. Em se tratando de uso de *frameworks*, os trabalhos 1 e 2 usam algum tipo de ferramentas para fazer a integração entre a programação de agentes e o uso das crenças probabilísticas juntamente com as Redes Bayesianas.

O principal diferencial da presente dissertação pode ser observado nas últimas duas colunas. Nenhum dos trabalhos estudados propõe uma nova linguagem de programação que integre todos os conceitos (programação de agentes BDI e modelos probabilísticos) em uma só ferramenta. Tampouco fornecem um ambiente integrado de programação para facilitar o desenvolvimento de agentes de *software* com base de crenças probabilísticas.

A ausência destas características nos trabalhos estudados foi um dos fatores motivadores do presente trabalho, pois ao mesmo tempo em que os trabalhos relacionados ajudam a corroborar a teoria da proposta, também lançam o desafio de criar uma linguagem, bem como um ambiente de desenvolvimento, capaz de implementar e testar agentes de *softwares* com crenças probabilísticas e Redes Bayesianas.

4 Proposta de Trabalho

Na programação orientada a agentes, cada agente possui sua base de crenças, desejos e intenções. Cada crença dentro do banco de dados é única e possui somente N estados que não podem alterar seus valores. Isso impossibilita ações baseadas em probabilidades, que poderiam permitir um comportamento mais flexível para determinado agente, de forma a ele atingir seus objetivos de forma mais rápida e ágil.

O presente trabalho teve como objetivo desenvolver uma forma de permitir ao programador, conceber um banco de dados de crenças baseado em probabilidades. Ao concluir este estudo e implementação, tornou-se possível através de uma nova linguagem de programação criar redes probabilísticas, possibilitando que agentes possam tomar decisões baseadas em variáveis com probabilidades que estão continuamente mudando conforme o ambiente em que se encontram. Esta nova linguagem de programação foi chamada de AgentSpeak(PL), ou *AgentSpeak Probability Language*.

As redes probabilísticas que serão usadas na base de crenças do agente estão baseadas nas Redes Bayesianas, que formam redes clássicas de probabilidades de variáveis. O suporte a Diagramas de Influência [RUSSEL; NORVIG, 2003], que permitem a inclusão de variáveis de decisão e funções de utilidade a uma Rede Bayesiana, também está previsto em AgentSpeak(PL)

Inicialmente será apresentado um resumo das características de AgentSpeak(L) que é a linguagem BDI em que AgentSpeak(PL) está baseada.

Logo após, será mostrado o estudo e a implementação da linguagem AgentSpeak (PL), incluindo sua sintaxe e semântica.

No segundo momento será mostrada a semântica formal sobre as novas funções acrescidas à linguagem de programação de agentes existente AgentSpeak(L). Posteriormente, será mostrada a implementação da semântica através da linguagem de programação Java, a fim de permitir a integração na ferramenta de desenvolvimento de sistemas multiagentes Jason.

No final do trabalho, foi obtida uma ferramenta capaz de interpretar códigos de sistemas multiagentes com crenças puramente absolutas, puramente probabilísticas ou então mistas. Isso abriu um novo horizonte na programação orientada a agentes.

4.1 Introdução a AgentSpeak(PL)

A representação de Redes Bayesianas em AgentSpeak(PL) é feita por meio do operador funcional '%' que permite associar probabilidade subjetiva (*bayesianas*) às crenças de um agente. Informalmente este operador equivale ao operador *P()* da Teoria das Probabilidades. Tanto em AgentSpeak(L) quanto em AgentSpeak(PL), crenças são representadas por predicados completamente instanciados (*grounded*). Assim expressões utilizadas dentro do operador '%' devem ser compostas apenas de predicados completamente instanciados.

O método de representação de Redes Bayesianas em AgentSpeak(PL) assume que todas as variáveis *bayesianas* de uma dada Rede Bayesiana sejam representadas por predicados unários de *AgentSpeak(PL).* Os estados relacionados à variável são definidos por termos literais da linguagem.

No caso do exemplo da Figura 11, as variáveis *Ladrão*, *Terremoto*, *Alarme*, *JoãoLig* e *MariaLig*, seriam representadas, respectivamente, pelo predicado *ladr(x)*, *tmot*(*x*), *alrm*(*x*), *jlig*(x) e *mlig*(*x*), com *x*={*yes*,*no*}. As probabilidades prévias de *Ladrão* e *Terremoto* são definidas de acordo com a Figura 14.

```
% ladr(yes)    = 0.001 .
% ladr(no)     = 0.999 .
% tmot(yes)    = 0.002 .
% tmot(no)     = 0.998 .
```

FIGURA 14. Probabilidades Prévias em AgentSpeak(PL).

Probabilidades condicionais são definidas usando o operador '|'. Assim as probabilidades condicionais para *P*(*Alarme*|*Ladrão*,*Terremoto*), *P*(*JoãoLig*|*Alarme*) e *P*(*MariaLig*|*Alarme*) são representadas em AgentSpeak(PL), conforme a Figura 15.

```
% alrm(yes)    | ladr(yes)    & tmot(yes)    =  0.95 .
% alrm(no)     | ladr(yes)    & tmot(yes)    =  0.05 .
% alrm(yes)    | ladr(yes)    & tmot(no)     =  0.95 .
% alrm(no)     | ladr(yes)    & tmot(no)     =  0.05 .
% alrm(yes)    | ladr(no)     & tmot(yes)    =  0.29 .
% alrm(no)     | ladr(no)     & tmot(yes)    =  0.71 .
% alrm(yes)    | ladr(no)     & tmot(no)     =  0.001 .
% alrm(no)     | ladr(no)     & tmot(no)     =  0.999 .

% jlig(yes)    | alrm(yes)    = 0.9 .
% jlig(yes)    | alrm(no)     = 0.1 .
% jlig(no)     | alrm(yes)    = 0.05 .
% jlig(no)     | alrm(no)     = 0.95 .

% mlig(yes)    | alrm(yes)    = 0.7 .
% mlig(yes)    | alrm(no)     = 0.3 .
% mlig(no)     | alrm(yes)    = 0.01 .
% mlig(no)     | alrm(no)     = 0.99 .
```

FIGURA 15. Probabilidades Prévias em AgentSpeak(PL).

Antes de o agente entrar em execução todos as declarações na seção de crenças que começam com o operador probabilístico ‘*%*()’ são agrupadas num conjunto único de declarações, denominado de *conjunto de crenças probabilísticas* do agente, que é analisado para verificar se corresponde a uma Rede Bayesiana corretamente construída.

Se o conjunto de crenças probabilísticas realmente constituir uma Rede Bayesiana, então é feita a uma busca no conjunto restante de crenças do agente de possíveis evidências (prévias) para o modelo probabilístico. De maneira geral evidências são crenças não probabilísticas que correspondem a possíveis estados das variáveis da Rede Bayesiana. Caso alguma evidência seja encontrada então a probabilidade do estado correspondente será definida como 1.0.

Considerando que os estados correspondentes para cada variável *bayesiana* devem ser mutuamente exclusivos, é de responsabilidade do modelador do sistema evitar a inconsistência do conjunto de evidências prévias, não declarando duas (ou mais) evidências contraditórias. Caso isto ocorra uma delas será escolhida não-deterministicamente como evidência inicial.

Também é possível adicionar evidências probabilísticas, correspondentes a atribuições de valores de probabilidade para variáveis da Rede Bayesiana. Isso é feito através de comandos na forma ‘*+%b(s)=r*’, onde *b* identifica um variável da Rede Bayesiana e *s* um estado desta variável, ao qual é a atribuída a probabilidade *r*. No exemplo que vem sendo utilizado para ilustrar AgentSpeak(PL), o agente poderia registrar a informação que houve um grande aumento na probabilidade prévia de ocorrer roubos, usando o comando ‘*+%ladr(yes)=0.1*’.

A estrutura formada pelo conjunto de crenças probabilísticas, adicionada do conjunto de evidências, formará o *modelo probabilístico* do agente. O estado inicial deste modelo é definido pelo conjunto de crenças probabilísticas e de evidências encontradas na base inicial de crenças. Na inicialização do agente, este estado é repassado ao módulo responsável pelo processo de inferência probabilístico.

O estado do modelo probabilístico será alterado cada vez que o conjunto de evidências sofrer alguma modificação, ou seja, cada vez que uma nova crença correspondente a um evento básico da RB seja adicionada ou excluída da base de crenças do agente. Quando isto ocorrer, o motor de inferência *bayesiano* executará um processo de propagação de evidências que recalculará as probabilidades das variáveis. Os valores recalculados para as probabilidades de cada evento básico são armazenados para futuras consultas, continuando a ser válidos até que ocorra alguma outra alteração no conjunto de evidências. O conjunto destes valores forma o terceiro elemento da estrutura que representa o estado atual do modelo probabilístico do agente.

Quanto o agente está em execução pode-se usar o operador probabilístico ‘*%*(*b*(*s*))’ para consultar o valor atual de probabilidade do estado *s* da variável *b*. Assim, no exemplo que está sendo usado para ilustrar AgentSpeak(PL) pode-se consultar a probabilidade P(*Alarme=V*) de ocorrência de um alarme, usando a expressão '%(*alrm*(*yes*))'. Estas consultas podem ocorrer tanto na condição de aplicação quanto no próprio corpo de um plano do agente.

Variáveis de decisão e funções de utilidade também podem ser utilizadas em AgentSpeak(PL), estendendo o modelo probabilístico do agente com um modelo de decisão e

um modelo de utilidade. As variáveis de decisão do agente são representadas por equações na forma '*%pb = ?.*', adicionadas ao final da definição da Rede Bayesiana. As funções de utilidade são definidas em AgentSpeak(PL) através do operador '$' que permite associar utilidades aos estados das variáveis da Rede Bayesiana.

4.2 Sintaxe e Características de AgentSpeak(PL)

A linguagem AgentSpeak(PL) é uma extensão da linguagem AgentSpeak(L) definida em [BORDINI; et al., 2007]. A gramática abstrata da linguagem AgentSpeak(PL) é apresentada na Figura 16.

$$
\begin{array}{l}
ag ::= bs\ \ bn\ \ dm\ \ um\ \ pvs\ \ ps \\
bs ::= b_1 \ldots b_n \ \ (n \geq 0) \\
bn ::= bnq_1, \ldots, bnq_n \ \ (n \geq 0) \\
bnq ::= \%pb\ = r \mid \%pb\ \ '|'\ \ pb_1 \wedge \ldots \wedge pb_n\ = r \ \ (n > 0) \\
dm ::= dq_1, \ldots, dq_n \ \ (n \geq 0) \\
dq ::= \%pb\ = ? \\
um ::= ufq_1, \ldots, ufq_n \ \ (n \geq 0) \\
ufq ::= \$uf = r \ \ (n \geq 0) \\
pvs ::= pvq_1 \ldots pvq_n \ \ (n \geq 0) \\
pvq ::= \%pb\ = r \\
ps ::= p_1 \ldots p_n \ \ (n \geq 0) \\
p ::= te : ct \leftarrow h \\
te ::= +at \mid -at \mid +pat \mid -pat \mid +g \mid -g \\
ct ::= ct_1 \mid \mathcal{T} \\
ct_1 ::= at \mid \neg at \mid pat \mid \neg pat \mid ct_1 \wedge ct_1 \\
h ::= h_1 ; \mathcal{T} \mid \mathcal{T} \\
h_1 ::= a \mid g \mid u \mid h_1 ; h_1 \\
at ::= \mathrm{P}(t_1, \ldots, t_n) \ \ (n \geq 0) \ \ \mid \mathrm{P}(t_1, \ldots, t_n)[s_1, \ldots, s_m] \ \ (n \geq 0, m > 0) \\
pat ::= pat_1 \mid pat_1[s_1, \ldots, s_m] \ \ (m > 0) \\
pat_1 ::= \%\mathrm{P}(t) = x \mid \%\mathrm{P}(t) = r \mid \%\mathrm{P}(t) \neq r \mid \\
\qquad \%\mathrm{P}(t) < r \mid \%\mathrm{P}(t) \leq r \mid \%\mathrm{P}(t) > r \mid \%\mathrm{P}(t) \geq r \\
s ::= percept \mid self \mid id \\
a ::= \mathrm{A}(t_1, \ldots, t_n) \ \ (n \geq 0) \\
g ::= !at \mid ?at \mid !pat \mid ?pat \\
u ::= +b \mid -at \mid +pvq \mid -pat
\end{array}
$$

FIGURA 16. Gramática da linguagem AgentSpeak(PL).

Na gramática acima os símbolos P, A, *t*, e *b* são os elementos léxicos originais de AgentSpeak(L), representando, respectivamente, os predicados, ações, termos lógicos e

crenças literais da linguagem. Os elementos *pb*, *uf*, *r* e *x* são os novos elementos léxicos de AgentSpeak(PL), representando, respectivamente, crenças probabilísticas literais, funções de utilidade, valores numéricos (constantes) reais e variáveis reais.

As principais modificações na gramática de AgentSpeak(L) são as seguintes:

- Possibilidade de inclusão de um modelo de crenças probabilísticas *bn*, composto da especificação de uma Rede Bayesiana [RUSSEL; NORVIG, 2003].

- Possibilidade de inclusão de um modelo de decisão *dm* e de um modelo de utilidade *um*, que permitem a expressão de *Diagramas de Influência* [RUSSEL; NORVIG, 2003].

- Possibilidade de inclusão de uma base de evidências probabilísticas *pvs*, composto de uma lista de atribuições de probabilidades para as crenças probabilísticas literais.

- Possibilidade de utilização de eventos de *trigger* *+pat* e *-pat* baseados em crenças probabilísticas atômicas (ou literais).

- Possibilidade de utilização de objetivos de atingimento (*achievement goals*) !*pat* e de teste ?*pat* também baseados em crenças probabilísticas atômicas.

- Possibilidade de utilização de ações *+pvq* e *-pvq* de atualização da base de estimativas de crenças probabilísticas.

Com essas modificações o agente passa a poder conter, na sua base de crenças, de uma Rede Bayesiana *bn* que representa o *modelo de crenças probabilísticas (*ou apenas *modelo probabilístico)* do agente. A Rede Bayesiana é definida através de um conjunto de equações probabilísticas bnq_1, ..., bnq_n que especificam as tabelas de probabilidade prévias para as variáveis (nós da rede) sem pais (%*pb* = *r*), ou de probabilidade condicionada para as variáveis que possuem pais (%*pb* : $\%pb_1 \wedge .. \%pb_n = r$).

Na sua forma mais elementar, uma crença probabilística *%pb* atribui uma probabilidade para um estado possível de uma variável da Rede Bayesiana. Em AgentSpeak(PL) as variáveis da Rede Bayesiana são representadas através de predicados unários P(*t*). O nome do predicado define o nome da variável, enquanto que o termo *t* identifica os possíveis estados que a variável pode assumir. Assim os termos *%pb* sempre tem a forma *%P(t)*, onde o predicado unário *P* define o nome da variável *bayesiana* e o termo literal *t* identifica o estado desta variável.

Note que estes estados são mutuamente exclusivos por definição, ou seja, não é possível que uma variável assuma ao mesmo tempo dois estados distintos. Assim, uma crença probabilística literal *%pb* sempre identifica um estado único de uma variável da Rede Bayesiana.

O modelo probabilístico do agente também pode conter uma base de evidências probabilísticas *pvs*, que é formada por uma lista de estimativas (atribuições) de valores de probabilidade previamente conhecidos determinados estados das variáveis da Rede Bayesiana. A base *pvs* é um componente opcional do modelo probabilístico do agente. Quando definida, ela atribui, através de equações na forma *%pb = r*, valores prévios de probabilidade para determinados estados de variáveis da Rede Bayesiana.

Além do modelo probabilístico, AgentSpeak(PL) também permite a utilização de variáveis de decisão e funções de utilidade para que o agente possa tomar decisões com base na utilidade (ou custo) das suas crenças probabilísticas.

Para tanto, é possível adicionar um modelo de decisão *dm* e um *modelo de utilidade um*, que transformam a Rede Bayesiana do agente em um Diagrama de Influência [RUSSEL; NORVIG, 2003]. O modelo de decisão *dm* contém uma lista das variáveis de decisão do agente. Essa lista é formada por equações na forma *%pb = ?*. Cada uma das equações *%pb = ?* indica que a probabilidade (prévia) atribuída a um dados estado de uma variável (crença) probabilística ainda não está definida (decidida). Essas variáveis de decisão podem ser vinculadas ao modelo probabilístico do agente, ou apenas ao modelo de utilidade. Para vincular ao modelo probabilístico, basta utilizar as variáveis de decisão como pais de outras variáveis da Rede Bayesianas, ou seja, condicionar variáveis da Rede Bayesiana a variáveis de decisão. A vinculação ao modelo de utilidade é feita pela atribuição de utilidades para os estados da variável de decisão.

A escolha de um particular estado de uma variável de decisão depende do modelo de utilidade do agente. O mecanismo de inferência probabilístico é capaz de decidir, com base no modelo probabilístico e no modelo de utilidade qual dos estados da variável de decisão que maximiza a utilidade. Nesse caso, será atribuída a probabilidade total (1.0) para o estado que maximizar a utilidade e probabilidade nula (0.0) para os demais estados desta variável.

O modelo de utilidade *mu* de AgentSpeak(PL) é formado por um conjunto de definições de funções de utilidade. Uma função de utilidade é definida através de uma equação *%uf = r*, onde *r* é um valor real que atribui uma utilidade (se positivo) ou custo (se negativo) para uma determinada combinação de estados do modelo probabilístico. O termo *%uf* identifica a função de utilidade e a combinação particular de estados das variáveis da Rede Bayesiana. Esses termos tem o formato *%id(pb1, ..., pbn)*, onde *id* é o identificador (nome) da função de utilidade e *pb1, ..., pbn* é uma lista de crenças probabilísticas literais que define uma combinação possível de estados de variáveis probabilísticas da Rede Bayesiana.

4.3 Estruturas da Semântica de AgentSpeak(PL)

A semântica formal da linguagem original AgentSpeak(L) foi definida através de uma semântica operacional baseada em regras para um sistema de transição de configurações. Neste trabalho foi utilizada a mesma técnica empregada em AgentSpeak(L), sendo definida uma extensão da semântica operacional capaz de tratar das novas construções linguísticas de AgentSpeak(PL).

Uma semântica operacional é apenas um conjunto de regras que define as transições que podem ocorrer entre as *configurações* de um dado agente. A configuração de um agente é uma estrutura que contém todas as informações relevantes para a operação do agente. Formalmente essa configuração é uma quíntupla mostrada na Figura 17.

$$\langle ag, C, M, T, s \rangle$$

FIGURA 17. Configuração básica de um agente em AgentSpeak(PL).

O componente *ag* é um programa em AgentSpeak(PL) composto de todos os elementos previstos na gramática de AgentSpeak(PL). Assim, *ag* é dividido nos subcomponentes: ag_{bs}, ag_{bn}, ag_{ds}, ag_{um}, ag_{pvs} *e* a_{gps} que contém, respectivamente, as crenças não probabilísticas do agente, modelo probabilístico (Rede Bayesiana), variáveis de decisão da rede, modelo de utilidade, a base de evidências probabilísticas e o conjunto de planos do agente.

O componente *C* indica a *circunstância* do agente, possuindo a estrutura $\langle I, E, A, P \rangle$, onde C_I= *{i, i , . . . }* define o conjunto de intenções do agente, C_E=*{(te, i), (te, i), . . . }* define o conjunto de eventos do agente (*te* é o evento de trigger e *i* sua intenção associada), C_A *é o* conjunto de ações do agente e C_P é o conjunto de crenças probabilísticas derivadas da Rede Bayesiana do agente.

Os subcomponentes C_I, C_E e C_A, são utilizados na definição da semântica operacional original de AgentSpeak(L), detalhes sobre suas características e utilização podem ser encontrados em [BORDINI; et al., 2004].

O subcomponente C_P, foi definido apenas para AgentSpeak(PL). Este subcomponente contém a atribuição (estimativa) de probabilidade para cada uma das crença probabilísticas previstas na Rede Bayesiana, e é sempre mantido atualizado de acordo com a estrutura da rede e com as evidências disponíveis até o momento. Da mesma forma que a base de evidências probabilísticas ag_{pvs}, o subcomponente C_P é formado por uma lista de atribuições de probabilidade $\%P_1(t_{11}) = r_{11}$, $\%P_1(t_{12}) = r_{12}$, ..., $\%P_n(t_{n1}) = r_{n1}$,..., $\%P_n(t_{nm}) = r_{nm}$, onde todos os termos t_{ij} devem ser termos literais, correspondentes aos estados possíveis de cada variável P_i. Porém, diferente de ag_{pvs}, o subcomponente C_P atribui uma probabilidade para cada estado possível de cada variável da Rede Bayesiana, definindo a atribuição corrente de probabilidades do agente.

O subcomponente C_P deve ser atualizado quando novas evidências são agregadas ou excluídas da base de crenças do agente. Isso pode acontecer em duas possibilidades: quando uma nova crença literal não probabilística *b* é adicionada (+*b*) ou excluída (-*at*) da base de crenças não probabilísticas do agente, ou quando a própria base de evidências probabilísticas ag_{pvs} é alterada diretamente através de ações *+pbeq e -pat*. Nos dois casos a base é atualizada de acordo, e logo após o subcomponente é atualizado pelo recálculo da distribuição de probabilidades da Rede Bayesiana executado pelo motor de inferência.

No caso de uma crença não probabilística ser adicionada (+*b*) a base de crenças do agente, se ela for formada por um predicado unário (*b=P(t)*) que tenha exatamente o mesmo formato (que possa ser unificada) com um estado de uma variável da Rede Bayesiana, então, ag_{pvs} é atualizado de forma que a probabilidade atribuída a *P(t)* seja 1 (*%P(t)=1)*, e que a probabilidade atribuída aos demais estados da variável *P* seja 0. Se a crença estiver sendo retirada da base (-*at*), então qualquer atribuição feita a variável *bayesiana* correspondente na base ag_{pvs} é excluída desta base.

Outra possibilidade é a atualização direta da base ag_{pvs} através de ações *+pvq* e *-pat*. Caso seja uma adição de estimativa (*+pvq*) então deve ser fornecida a sequência completa: $+\%P(t_1)=r_1$; $+\%P(t_2)=r_2$; ...; $+\%P(t_n)=r_n$, de atribuição de probabilidades para todos os estados possíveis $t_1, t_2, ..., t_n$ da variável *P*. Para as exclusões de evidências, tanto no caso de crenças não-probabilísticas quanto probabilísticas, então são eliminadas da base ag_{pvs} atual todas as estimativas correntes de probabilidades para essas crenças, sendo substituídas pelas probabilidades prévias que podem ser inferidas diretamente da Rede Bayesiana.

Em termos de programação dos planos do agente, as crenças probabilísticas podem ser utilizadas tanto como eventos de trigger associados aos planos, quanto nas condições de contexto dos planos. Em ambos os casos é possível utilizar usar expressões como $\%P(t)<r$ ou $\%P(t)>r$, para testar o valor de probabilidade atribuído a *%P(t)*. Também se pode utilizar variáveis lógicas nos termos que representam os estados das variáveis da Rede Bayesiana, ou variáveis numéricas em expressões como $\%P(t)=x$, para recuperar o valor da probabilidade atribuído a *%P(t)*.

Objetivos de atingimento (achievement goals) !pat podem ser usados nos planos dos agentes, tendo exatamente a mesma semântica que no caso de AgentSpeak(L). Os objetivos de teste *?pat*, por sua vez, são testados exatamente como as crenças probabilísticas usados nas condições de contexto dos planos.

O componente *M*, definido como $\langle In, Out, SI \rangle$, armazena as informações do processamento de comunicação do agente, onde M_{In} contém as mensagens de entrada, M_{Out} as mensagens de saída e M_{SI} guarda as intenções que estão suspensas devido ao processamento de alguma mensagem.

O componente *T*, com estrutura $\langle R, Ap, \iota, \varepsilon, \rho \rangle$, contém as informações temporárias do agente: T_R é um conjunto que armazena os planos relevantes do evento sendo processado, T_{Ap} é o conjunto de planos aplicáveis para este evento, enquanto que os demais componentes armazenam, respectivamente, a intenção, evento e plano sendo processados.

Os componente *M* e *T* e seus respectivos subcomponentes não sofreram alterações na definição de AgentSpeak(PL), possuindo as mesmas características definidas para a semântica operacional de AgentSpeak(L) Mais detalhes sobre estes elementos podem ser obtidos em [BORDINI; et al., 2004].

Por fim o componente *s* indica o passo atual no ciclo de raciocínio do agente. O ciclo de raciocínio é formado por uma série de passos de execução que começam a ser executados quando o agente é inicializado e é processado continuamente até o agente ter seu processo de raciocínio interrompido. A Figura 18 mostra os possíveis passos dentro do ciclo de raciocínio, bem como quais são as transições possíveis de um passo para outro.

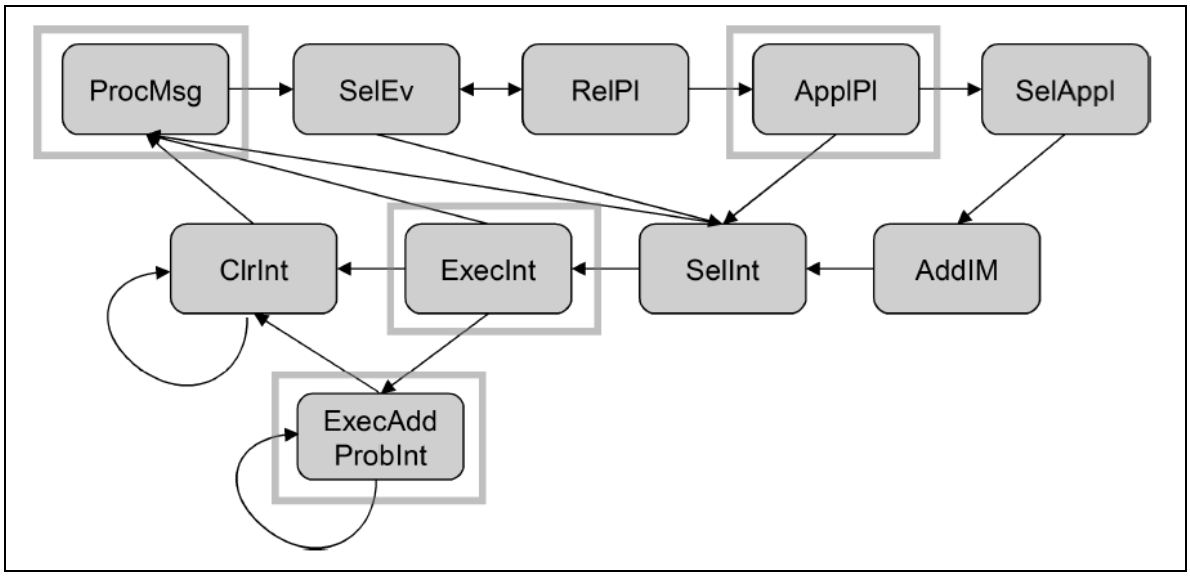


FIGURA 18. Passos na execução de um agente.

Os passos marcados foram alterados em relação à semântica operacional de AgentSpeak(L). O passo **ExecAddProbInt** é novo, sendo necessário para especificar a semântica da inclusão de crenças probabilísticas atômicas. A seguir é apresentada uma breve descrição das funções executadas em cada passo:

- **ProcMsg**: processa mensagem direcionada ao agente;
- **SelEv**: seleciona um evento do conjunto de eventos;
- **RelPl**: retorna todos os planos relevantes;
- **ApplPl**: verifica quais planos são aplicáveis;
- **SelAppl**: seleciona um plano em particular;
- **AddIM**: adiciona uma nova intenção ao conjunto de intenções;
- **SelInt**: seleciona uma intenção;
- **ExecInt**: executa a intenção selecionada;
- **ClrInt**: limpa a intenção que pode ter sido concluída no passo anterior.
- **ExecAddProbInt**: passo que garante que a adição de uma crença probabilística seja feita de forma atômica.

Foi necessária a inclusão do novo passo **ExecAddProbInt** porque as crenças probabilísticas a respeito do estado de uma variável da Rede Bayesiana são formadas por várias equações de atribuição de probabilidade, mas a base *pvs* somente assume um estado consistente após todas as equações de atribuição de uma dada variável terem sido adicionadas, este passo extra garante que essas adições sejam feitas de forma atômica.

4.4 Arquitetura e Ciclo de Execução

O processo de raciocínio do agente é definido por uma série de regras de transição de configurações, que definem como o agente pode passar de um passo do ciclo a outro. Essas regras de transição dependem de um conjunto de funções adicionais, capazes de executar operações como a buferização das crenças (função), o processo de revisão de crenças (função *BRF*), a seleção de eventos (função S_E), planos (função S_O) e intenções (função S_I) do agente. A Figura 19 mostra a arquitetura geral do ciclo de execução, identificando como os componentes da configuração se relacionam com as funções de transição. Na figura já estão indicadas as principais modificações devidas a AgentSpeak(PL). Para maiores detalhes sobre a arquitetura definida na Figura 19, consulte [BORDINI; et al., 2007].

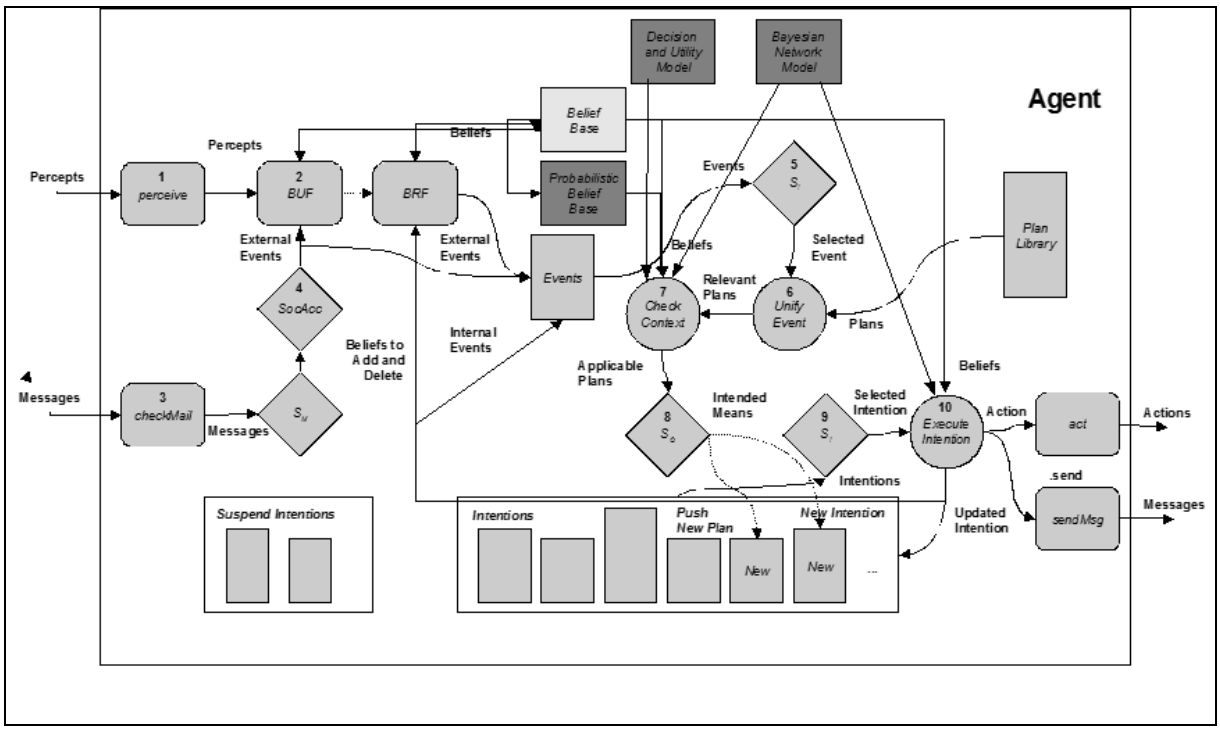


FIGURA 19. Ciclo de execução de um agente.

O objetivo de AgentSpeak(PL) é ser uma extensão de AgentSpeak(L). Assim foram mantidos os principais componentes e elementos da arquitetura dos agentes AgenteSpeak(L). As novas bases são itens adicionais à arquitetura já existente, ou seja, a base de crenças lógicas (ou não probabilísticas) continuará existindo de forma normal e independente das outras bases.

Os principais elementos adicionados para suportar as funcionalidades de AgentSpeak(PL) foram os novos modelos e bases de conhecimentos probabilísticos previstos pela gramática desta linguagem. As bases de conhecimento *Bayesian Network Model Probabilistic Belief Base* e *Decision and Utility Model*, representam, respectivamente o modelo de Rede Bayesiana ag_{bn}, a base de evidências probabilísticas ag_{pvs} do agente e a combinação do modelo de decisão ag_{dm} com o modelo de utilidade ag_{um} do agente. O

componente *Probabilistic Belief Base* também incorpora a distribuição corrente de probabilidades C_P do modelo probabilístico do agente. Também são mostradas as dependências que existem entre essas novas bases e os demais componentes da arquitetura.

O tratamento das novas crenças probabilísticas do modelo *bayesiano* será feito através de novas regras de transição de configuração e de modificações de algumas regras que já foram definidas para AgentSpeak(L).

Os passos que necessitam de alteração, incluem várias regras de transição de configuração. A seguir são apresentas as regras que compõem cada um desses passos, e são indicadas quais regras foram modificadas ou incluídas.

(1) Passo **ApplPl**:

- Regras que foram alteradas na semântica de AgentSpeak(PL):

  - **Appl1':** identifica quais planos são aplicáveis para um dado evento, testando se a condição de contexto do plano é verdadeira, foi modificada para permitir que crenças probabilísticas também sejam testadas na condição de contexto.

  - **Appl2':** trata o caso em que não há planos aplicáveis, modificada pela mesma razão de **Appl1'**.

(2) Passo **ExecInt**:

- Regras que não foram alteradas pela semântica de AgentSpeak(PL):

  - **Action**: executa uma ação explícita no corpo do plano do agente;

  - **AchvGl**: registra um novo objetivo de atingimento interno ao agente;

  - **TestGl1**: testa se um objetivo de teste já foi alcançado;

  - **TestGl2**: gera uma evento interno para testar o objetivo de teste;

  - **ClrInt1**, **ClrInt2**, **ClrInt2**: finalizam a execução de uma intenção;

- Regras que foram alteradas na semântica de AgentSpeak(PL):

  - **AddBel':** adiciona uma nova crença não-probabilística a base de crenças do agente, foi modificada porque a crença não-probabilística pode estar relacionada a uma variável da Rede Bayesiana.

  - **DelBel':** exclui uma nova crença não-probabilística a base de crenças do agente, foi modificada pela mesma razão de **AddBel'**.

- Novas regras da semântica de AgentSpeak(PL):

  - **TestProbGl1:** equivale a TestGl1, apenas que para objetivos de teste relacionados a crenças probabilísticas.

- **TestProbGl2:** mesmo caso de TestGl2, aplicado a teste de crenças probabilísticas.

- **AddProbBel1:** inicia o processo de adição de uma crença probabilística atômica na base ag_{pvs}.

- **DelProbBel:** exclui uma crença probabilística da base ag_{pvs}.

(3) Passo **ExecAddProbInt**:

- Novas regras da semântica de AgentSpeak(PL):

- **AddProbBel2:** executa a adição de mais uma equação de atribuição de probabilidade a base ag_{pvs}.

- **AddProbBel3:** finaliza o processo de adição das equações que definem a crença probabilística e efetua o processo de recálculo da Rede Bayesiana ag_{bn} e atualização da distribuição corrente de probabilidades C_P.

4.5 Novas Relações e Funções

Essas novas regras necessitaram de novas definições e funções para sua especificação. Em particular, foi necessário definir a relação de consequência lógica entre a base de evidências probabilísticas ag_{pvs} ou da base C_P que contém a distribuição corrente de probabilidades e as fórmulas atômicas probabilísticas pat_1 definidas na gramática de AgentSpeak(PL).

Esta definição é similar a definição da relação de consequência lógica entre uma fórmula atômica não probabilística de AgentSpeak(L) e a base de crenças ag_{bs}, que é repetida aqui para conveniência do leitor:

Definição 1. Uma fórmula atômica não probabilística at_1, com anotações $[s_{11}, s_{12}, ..., s_{1n}]$ é uma consequência lógica de um conjunto ag_{bs} de fórmulas não-probabilísticas literais, denotado por $ag_{bs} \models at_1[s_{11}, s_{12}, ..., s_{1n}]$, se e somente se existir uma fórmula $at_2\,[s_{21}, s_{22}, ..., s_{2m}] \in ag_{bs}$ tal que (i) $at_1\theta = at_2\theta$, para um unificador mais geral θ e (ii) $\{s_{11}, s_{12}, ..., s_{1n}\} \subseteq \{s_{21}, s_{22}, ..., s_{2m}\}$.

A relação de consequência lógica para crenças probabilísticas é definida como segue:

Definição 2. Uma fórmula atômica probabilística pat_1, com anotações $[s_{11}, s_{12}, ..., s_{1n}]$ é uma consequência lógica de um conjunto *pvs* de estimativas de crenças probabilísticas, denotado por $pvs \models pat_1[s_{11}, s_{12}, ..., s_{1n}]$, se e somente se, para o termo probabilístico $\%p(t) \in pat_1$ existir uma equação $\%pb_k = r_k\,[s_{21}, s_{22}, ..., s_{2m}] \in pvs$ tal que, para um dado um unificador mais geral θ, são satisfeitas as seguintes condições:

(*a*) $\%p(t)\theta = \%pb\theta$,

(*b*) $\{s_{11}, s_{12}, ..., s_{1n}\} \subseteq \{s_{21}, s_{22}, ..., s_{2m}\}$,

(*c*) se pat_1 é da forma $\%p(t) = x$, então $r_k = x\theta$, e

(*d*) se pat_1 é da forma $\%p(t) = r$, $\%p(t) \neq r$, $\%p(t) < r$, $\%p(t) \leq r$, $\%p(t) > r$, ou $\%p(t) \geq r$ então, respectivamente, $r_k = r$, $r_k \neq r$, $r_k < r$, $r_k \leq r$, $r_k > r$, ou $r_k \geq r$.

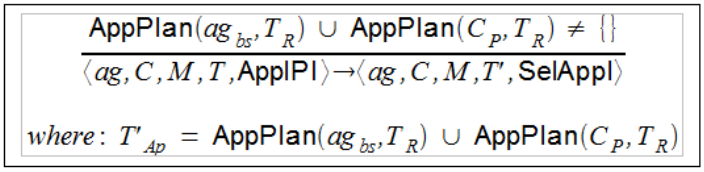
Além dessa definição as novas regras de transição farão das seguintes funções diretamente vinculadas ao processo de inferência probabilístico e a avaliação dos modelos de utilidade e decisão:

- A função *AddProbEvds(b, ag_{bn}, ag_{pvs})* atualiza a base de evidências ag_{pvs}, caso a crença não-probabilística *b* sendo adicionada a base de crenças do agente, corresponder a um dos estados *t* de alguma variável *P* da Rede Bayesiana ag_{bn} do agente. Nesse caso, conforme definido anteriormente, ag_{pvs} é atualizado de forma que a probabilidade atribuída a *P(t)* seja 1 (*%P(t)=1*), e que a probabilidade atribuída aos demais estados da variável *P* seja 0.
- A função *DelProbEvds(b, ag_{bn}, ag_{pvs})* é similar a *AddProbEvds(b, ag_{bn}, ag_{pvs})*, também fazendo a atualização da base de evidências ag_{pvs}, mas nesse caso se a crença não-probabilística *b* estiver sendo excluída da base de crenças do agente. Se o predicado *P* de *b* corresponder alguma variável *P* da Rede Bayesiana ag_{bn} do agente, então qualquer atribuição feita para esta variável *bayesiana* é excluída de ag_{pvs}.
- A função *RecalcProbModel(ag_{bn}, ag_{dm}, ag_{um}, ag_{pvs})* faz o recálculo da distribuição de probabilidades do modelo probabilístico ag_{bn} em função das evidências coletadas até o momento ag_{pvs}. Esse processo de recálculo leva em conta se existe um modelo de utilidade ag_{um} e decisão ag_{dm} associados à Rede Bayesiana do agente, ou seja, na prática ele deve ser implementado por um motor de inferência capaz de lidar com Diagramas de Influência e não apenas Redes Bayesianas. Apesar disso não existem restrições quanto ao algoritmo específico utilizado no processo de recálculo.

4.6 Novas Regras de Transição

As novas regras são apresentadas a seguir:

Appl1': essa regra identifica quais planos são aplicáveis para um dado evento, testando se a condição de contexto do plano é verdadeira. Verifica crenças absolutas e crenças probabilísticas.

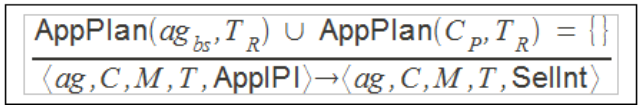


$$\frac{\mathsf{AppPlan}(ag_{bs}, T_R) \cup \mathsf{AppPlan}(C_P, T_R) \neq \{\}}{\langle ag, C, M, T, \mathsf{ApplPl}\rangle \rightarrow \langle ag, C, M, T', \mathsf{SelAppl}\rangle}$$

$$where:\ T'_{Ap} = \mathsf{AppPlan}(ag_{bs}, T_R) \cup \mathsf{AppPlan}(C_P, T_R)$$

A condição de **Appl1'** foi estendida de forma a permitir que expressões probabilísticas como '%(*b*(*a*))=*r*' possam ser usadas nas condições de contexto dos planos, assim é necessário verificar também *AppPlan*(C_P, T_R) que testa as condições de contexto probabilísticas de T_R na distribuição corrente de probabilidades C_P.

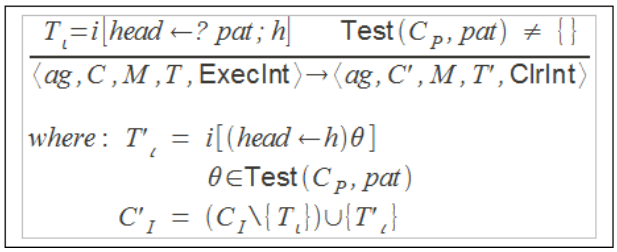
Appl2': trata o caso em que não há planos aplicáveis tanto para crenças probabilísticas quanto para crenças não probabilísticas.



$$\frac{\mathsf{AppPlan}(ag_{bs}, T_R) \cup \mathsf{AppPlan}(C_P, T_R) = \{\}}{\langle ag, C, M, T, \mathsf{ApplPl}\rangle \rightarrow \langle ag, C, M, T, \mathsf{SelInt}\rangle}$$

A condição **Appl2'** foi modificada a fim de atender quando não há planos aplicáveis, sejam eles probabilísticos ou não. Dessa forma o agente passa diretamente ao passo *SelInt* em seu ciclo de execução.

TestProbGl1: testa se um objetivo de teste relacionado a uma crença probabilística foi alcançado.

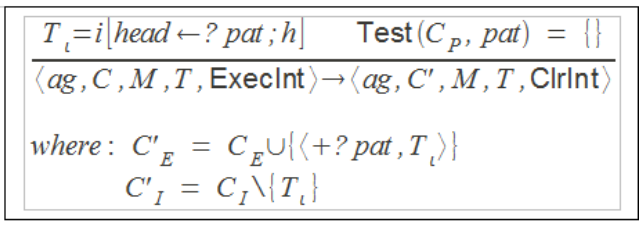


$$\frac{T_\iota = i[head \leftarrow ?\ pat; h] \qquad \mathsf{Test}(C_P, pat) \neq \{\}}{\langle ag, C, M, T, \mathsf{ExecInt}\rangle \rightarrow \langle ag, C', M, T', \mathsf{ClrInt}\rangle}$$

$$where:\ T'_\iota = i[(head \leftarrow h)\theta]$$
$$\theta \in \mathsf{Test}(C_P, pat)$$
$$C'_I = (C_I \setminus \{T_\iota\}) \cup \{T'_\iota\}$$

A regra **TestProbGl1** foi adicionada e através da função *Test*(C_p, *pat*) verifica se nas condições de contexto do plano um determinado objetivo *pat* foi atingido dentro de uma distribuição de probabilidades C_P.

TestProbGl2: gera um evento interno para testar o objetivo de teste relacionado a uma crença probabilística.

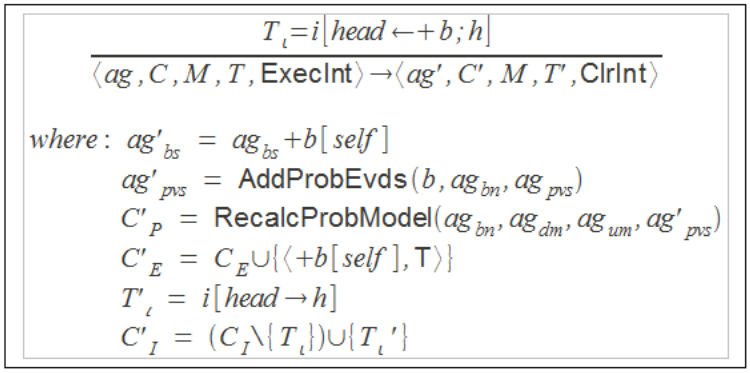


$$\frac{T_\iota = i[head \leftarrow ?\ pat; h] \qquad \mathsf{Test}(C_P, pat) = \{\}}{\langle ag, C, M, T, \mathsf{ExecInt}\rangle \rightarrow \langle ag, C', M, T, \mathsf{ClrInt}\rangle}$$

$$where:\ C'_E = C_E \cup \{\langle +?\ pat, T_\iota\rangle\}$$
$$C'_I = C_I \setminus \{T_\iota\}$$

A regra **TestProbGl2** tem a função de gerar um evento de testes *+?pat*, para verificar um determinado objetivo relacionado a uma crença probabilística *pat*. O resultado será atribuído a C'_E.

AddBel': adiciona uma crença não probabilística a base de crenças do agente.

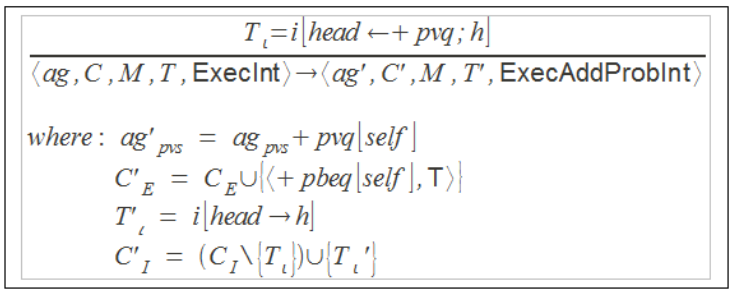


$$\frac{T_\iota = i[head \leftarrow + b ; h]}{\langle ag, C, M, T, \mathsf{ExecInt}\rangle \rightarrow \langle ag', C', M, T', \mathsf{ClrInt}\rangle}$$

$$\begin{aligned} where:\ ag'_{bs} &= ag_{bs} + b[self] \\ ag'_{pvs} &= \mathsf{AddProbEvds}(b, ag_{bn}, ag_{pvs}) \\ C'_P &= \mathsf{RecalcProbModel}(ag_{bn}, ag_{dm}, ag_{um}, ag'_{pvs}) \\ C'_E &= C_E \cup \{\langle +b[self], \mathsf{T}\rangle\} \\ T'_\iota &= i[head \rightarrow h] \\ C'_I &= (C_I \setminus \{T_\iota\}) \cup \{T_\iota'\} \end{aligned}$$

A regra **AddBel'** foi modificada em relação a regra original **AddBel** de forma a permitir primeiro a atualização da base de evidências probabilísticas através da função *AddProbEvds(b, ag_{bn}, ag_{pvs})* que verifica se a nova crença *b* corresponde a algum estado de variável da rede ag_{bn}. Posteriormente a regra **AddBel'** efetua o recálculo (essencialmente aplica o processo de propagação de evidências) da distribuição de probabilidades atual da Rede Bayesiana (e dos modelos de utilidade e decisão, se eles existirem). A função *RecalcProbModel(ag_{bn}, ag_{dm}, ag_{um}, ag_{bn}, ag'_{pvs})* é responsável por esse recálculo. O resultado dessa função é atribuído a C'_P que conterá a nova distribuição de probabilidades.

AddProbBel1: inicia o processo de adição de uma crença probabilística atômica a base de crenças do agente.

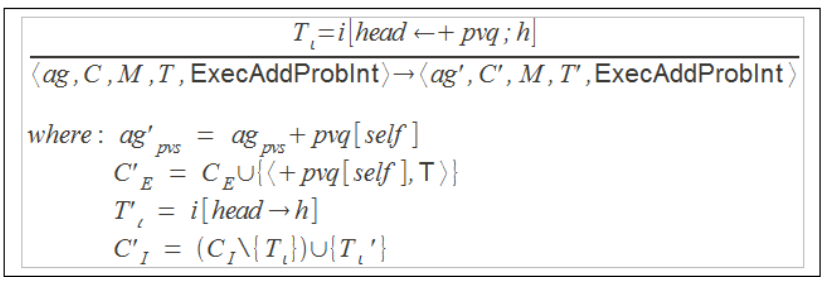


$$\frac{T_\iota = i[head \leftarrow + pvq ; h]}{\langle ag, C, M, T, \mathsf{ExecInt}\rangle \rightarrow \langle ag', C', M, T', \mathsf{ExecAddProbInt}\rangle}$$

$$\begin{aligned} where:\ ag'_{pvs} &= ag_{pvs} + pvq[self] \\ C'_E &= C_E \cup [\langle + pbeq[self], \mathsf{T}\rangle] \\ T'_\iota &= i[head \rightarrow h] \\ C'_I &= (C_I \setminus [T_\iota]) \cup [T_\iota'] \end{aligned}$$

Na regra **AddProbBel1** o ciclo de execução do agente começa a inclusão de uma nova crença probabilística *pvq* a sua base de crenças *ag'$_{pvs}$*. Essa nova crença pode conter mais de uma atribuição de probabilidade definida por *pbeq*, dado o número de evidências de uma determinada variável.

AddProbBel2: executa a adição de mais uma equação de atribuição de probabilidade relacionado a crença a ser adicionada a base de crenças do agente.



$$\frac{T_\iota = i[head \leftarrow + pvq ; h]}{\langle ag, C, M, T, \mathsf{ExecAddProbInt}\rangle \rightarrow \langle ag', C', M, T', \mathsf{ExecAddProbInt}\rangle}$$

$$\begin{aligned} where:\ ag'_{pvs} &= ag_{pvs} + pvq[self] \\ C'_E &= C_E \cup \{\langle + pvq[self], \mathsf{T}\rangle\} \\ T'_\iota &= i[head \rightarrow h] \\ C'_I &= (C_I \setminus \{T_\iota\}) \cup \{T_\iota'\} \end{aligned}$$

Na sequencia a regra **AddProbBel2** trata as atribuições intrínsecas a uma determinada crença *pvq*. A cada atribuição uma probabilidade é adicionada a base de crenças *ag'$_{pvs}$*.

AddProbBel3: finaliza o processo de adição das equações que definem a crença probabilística e efetua o processo de recálculo da Rede Bayesiana do agente atualizando a distribuição de probabilidades.

$$\frac{T_\iota \neq i[head \leftarrow + pvq ; h]}{\langle ag, C, M, T, \mathsf{ExecAddProbInt}\rangle \rightarrow \langle ag, C', M, T, \mathsf{ClrInt}\rangle}$$

$$where: \; C'_P = \mathsf{RecalcProbModel}(ag_{bn}, ag_{dm}, ag_{um}, ag_{pvs})$$

Por fim com a regra **AddProbBel3**, após as crenças probabilísticas terem sido adicionadas, a função *RecalcProbModel(ag_{bn}, ag_{dm}, ag_{um}, ag_{bn}, ag_{pvs})* é executada, a fim de recalcular as probabilidades da Rede Bayesiana com suas novas evidencias.

DelBel': exclui uma crença não-probabilística da base de crenças do agente.

$$\frac{T_\iota = i[head \leftarrow - b ; h]}{\langle ag, C, M, T, \mathsf{ExecInt}\rangle \rightarrow \langle ag', C', M, T', \mathsf{ClrInt}\rangle}$$

$$\begin{aligned} where: \; ag'_{bs} &= ag_{bs} - b[self] \\ ag'_{pvs} &= \mathsf{DelProbEvds}(b, ag_{bn}, ag_{pvs}) \\ C'_P &= \mathsf{RecalcProbModel}(ag_{bn}, ag_{dm}, ag_{um}, ag'_{pvs}) \\ C'_E &= C_E \cup \{\langle -b[self], \mathsf{T}\rangle\} \\ T'_\iota &= i[head \leftarrow h] \\ C'_I &= (C_I \setminus \{T_\iota\}) \cup \{T_\iota'\} \end{aligned}$$

A regra **DelBel'** foi modificada para remover as evidencias relacionadas com a crença excluída através da função *DelProbEvds(b, ag_{bn}, ag_{pvs})* e realizar o recálculo da Rede Bayesiana através da função *RecalcProbModel(ag_{bn}, ag_{dm}, ag_{um}, ag_{bn}, ag'$_{pvs}$).*

DelProbBel: exclui uma crença probabilística da base de crenças do agente.

$$\frac{T_\iota = i[head \leftarrow - pat ; h]}{\langle ag, C, M, T, \mathsf{ExecInt}\rangle \rightarrow \langle ag', C', M, T', \mathsf{ClrInt}\rangle}$$

$$\begin{aligned} where: \; ag'_{pvs} &= ag_{pvs} - pat[self] \\ C'_P &= \mathsf{RecalcProbModel}(ag_{bn}, ag_{dm}, ag_{um}, ag'_{pvs}) \\ C'_E &= C_E \cup \{\langle -pat[self], \mathsf{T}\rangle\} \\ T'_\iota &= \{i[head \leftarrow h]\} \\ C'_I &= (C_I \setminus \{T_\iota\}) \cup \{T_\iota'\} \end{aligned}$$

A regra **DelProbBel** foi adicionada para remover as crenças probabilísticas *pat* do agente e realizar o recálculo da Rede Bayesiana através da função *RecalcProbModel(ag_{bn}, ag_{dm}, ag_{um}, ag_{bn}, ag'$_{pvs}$).*

Tell': mensagem recebida de outros agentes a fim de adicionar crenças probabilísticas e não probabilísticas na base do agente recebedor. Em caso de crenças probabilísticas a Rede Bayesiana é recalculada.

$$\frac{\begin{array}{c} S_{\mathcal{M}}(M_{In}) = \langle mid, id, Tell, Bs\rangle \\ (mid, i) \notin M_{SI}(\text{for any intention } i) \\ \mathsf{SocAcc}(id, Tell, Bs) \end{array}}{\langle ag, C, M, T, \mathsf{ProcMsg}\rangle \rightarrow \langle ag', C', M', T, \mathsf{SelEv}\rangle}$$

$$\begin{array}{l} where: \; M'_{In} = M_{In} \setminus \{\langle mid, id, Tell, Bs\rangle\} \\ \quad \text{and for each } b \in Bs: \\ \quad ag'_{bs} = ag_{bs} + b[id] \\ \quad ag'_{pvs} = \mathsf{AddProbEvds}(b, ag_{bn}, ag_{pvs}) \\ \quad C'_{P} = \mathsf{RecalcProbModel}(ag_{bn}, ag_{dm}, ag_{um}, ag'_{pvs}) \\ \quad C'_{E} = C_{E} \cup \{\langle +b[id], \mathsf{T}\rangle\} \end{array}$$

A regra **Tell'** foi modificada de forma a permitir primeiro a atualização da base de evidências probabilísticas através da função *AddProbEvds(b, ag_{bn}, ag_{pvs})* que verifica se a nova crença *b* corresponde a algum estado de variável da rede ag_{bn}. Posteriormente é recalculada a distribuição de probabilidades atual da Rede Bayesiana através da função *RecalcProbModel(ag_{bn}, ag_{dm}, ag_{um}, ag_{bn}, ag'_{pvs})*. Sua execução está associada a comunicação e troca de mensagens entre agentes.

TellRepl': resposta enviada ao agente que fez o pedido prévio de inclusão da crença probabilística ou não probabilística.

$$\frac{\begin{array}{c} S_{\mathcal{M}}(M_{In}) = \langle mid, id, Tell, Bs\rangle \\ (mid, i) \in M_{SI}(\text{for any intention } i) \\ \mathsf{SocAcc}(id, Tell, Bs) \end{array}}{\langle ag, C, M, T, \mathsf{ProcMsg}\rangle \rightarrow \langle ag', C', M', T, \mathsf{SelEv}\rangle}$$

$$\begin{array}{l} where: \; M'_{In} = M_{In} \setminus \{\langle mid, id, Tell, Bs\rangle\} \\ \quad M'_{SI} = M_{SI} \setminus \{(mid, id)\} \\ \quad C'_{I} = C_{I} \cup \{i\} \\ \quad \text{and for each } b \in Bs: \\ \quad ag'_{bs} = ag_{bs} + b[id] \\ \quad ag'_{pvs} = \mathsf{AddProbEvds}(b, ag_{bn}, ag_{pvs}) \\ \quad C'_{P} = \mathsf{RecalcProbModel}(ag_{bn}, ag_{dm}, ag_{um}, ag'_{pvs}) \\ \quad C'_{E} = C_{E} \cup \{\langle +b[id], \mathsf{T}\rangle\} \end{array}$$

Untell': mensagem recebida de outros agentes a fim de remover crenças probabilísticas e não probabilísticas na base do agente recebedor. Em caso de crenças probabilísticas a Rede Bayesiana é recalculada.

$$\frac{S_{\mathcal{M}}(M_{In}) = \langle mid, id, Untell, ATs\rangle \quad (mid, i) \notin M_{SI}(\text{for any intention } i) \quad \text{SocAcc}(id, Untell, ATs)}{\langle ag, C, M, T, \text{ProcMsg}\rangle \rightarrow \langle ag', C', M', T, \text{SelEv}\rangle}$$

$$\begin{aligned}
&where: \; M'_{In} = M_{In} \setminus \{\langle mid, id, Untell, ATs\rangle\} \\
&\quad \text{and for each } b \in \{at\,\theta \mid \theta \in \text{Test}(ag_{bs}) \wedge at \in ATs\} \\
&\quad ag'_{bs} = ag_{bs} - b[id] \\
&\quad ag'_{pvs} = \text{DelProbEvds}(b, ag_{bn}, ag_{pvs}) \\
&\quad C'_P = \text{RecalcProbModel}(ag_{bn}, ag_{dm}, ag_{um}, ag'_{pvs}) \\
&\quad C'_E = C_E \cup \{\langle -b[id], \text{T}\rangle\}
\end{aligned}$$

A modificação da regra **Untell'** visa permitir a remoção de crenças probabilísticas através da função *DelProbEvds(b, ag_{bn}, ag_{pvs})* da base de crenças de uma agente. Posteriormente é recalculada a distribuição de probabilidades atual da Rede Bayesiana através da função *RecalcProbModel(ag_{bn}, ag_{dm}, ag_{um}, ag_{bn}, ag'_{pvs})*. Sua execução está associada a comunicação e troca de mensagens entre agentes.

UntellRepl': resposta enviada ao agente que fez o pedido prévio de remoção da crença probabilística ou não probabilística.

$$\frac{S_{\mathcal{M}}(M_{In}) = \langle mid, id, Untell, ATs\rangle \quad (mid, i) \in M_{SI}(\text{for any intention } i) \quad \text{SocAcc}(id, Untell, ATs)}{\langle ag, C, M, T, \text{ProcMsg}\rangle \rightarrow \langle ag', C', M', T, \text{SelEv}\rangle}$$

$$\begin{aligned}
&where: \; M'_{In} = M_{In} \setminus \{\langle mid, id, Untell, ATs\rangle\} \\
&\quad M'_{SI} = M_{SI} \setminus \{(mid, id)\} \\
&\quad C'_I = C_I \cup \{i\} \\
&\quad \text{and for each } b \in \{at\,\theta \mid \theta \in \text{Test}(ag_{bs}) \wedge at \in ATs\} \\
&\quad ag'_{bs} = ag_{bs} - b[id] \\
&\quad ag'_{pvs} = \text{DelProbEvds}(b, ag_{bn}, ag_{pvs}) \\
&\quad C'_P = \text{RecalcProbModel}(ag_{bn}, ag_{dm}, ag_{um}, ag'_{pvs}) \\
&\quad C'_E = C_E \cup \{\langle -b[id], \text{T}\rangle\}
\end{aligned}$$

5 Implementação da Linguagem AgentSpeak(PL)

AgentSpeak(PL) foi implementada como uma extensão da ferramenta *Jason* [JASON, 2011], resultando em uma nova versão desta ferramenta denominada *JasonBayes*. A ferramenta *Jason* original foi desenvolvida em Java e permite desenvolver agentes utilizando a linguagem AgentSpeak(L). A diretriz principal da implementação foi realizar alterações pontuais no código fonte existente a fim de deixar a ferramenta compatível com ambas as linguagens.

O primeiro passo foi analisar a estrutura de diretórios do código fonte do aplicativo, e verificar a função de cada arquivo. A Figura 20 mostra a estrutura de arquivos do *Jason*.

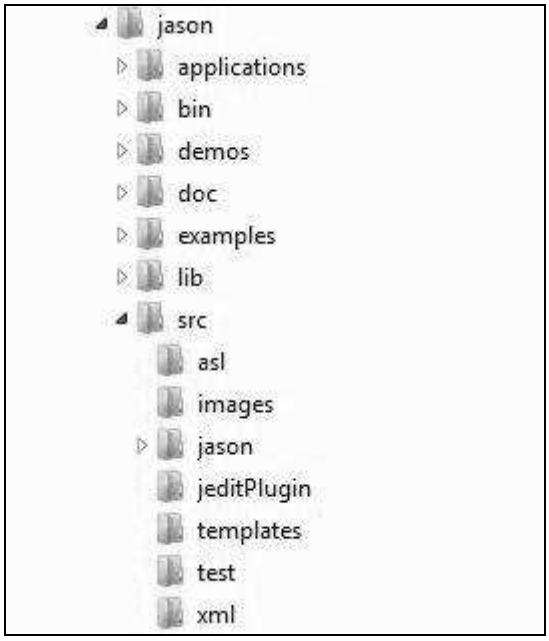


FIGURA 20. Estrutura de diretórios do *Jason*.

O diretório *src* é o que contém os arquivos em Java, que após a compilação permitem que a ferramenta seja executada. Uma característica interessante é que o *Jason* foi concebido como uma extensão da ferramenta *JEdit*, utilizando grande parte de seu código a fim de permitir a edição e manipulação de arquivos dos agentes envolvidos no projeto.

Dentro do diretório *src* existe outra estrutura de diretórios. O mais importante deles é o diretório *jason*, que contém todos os arquivos fontes em Java que integram o *Jason* com o *JEdit* e implementam a linguagem AgentSpeak(PL). A Figura 21 mostra a estrutura dessa pasta.

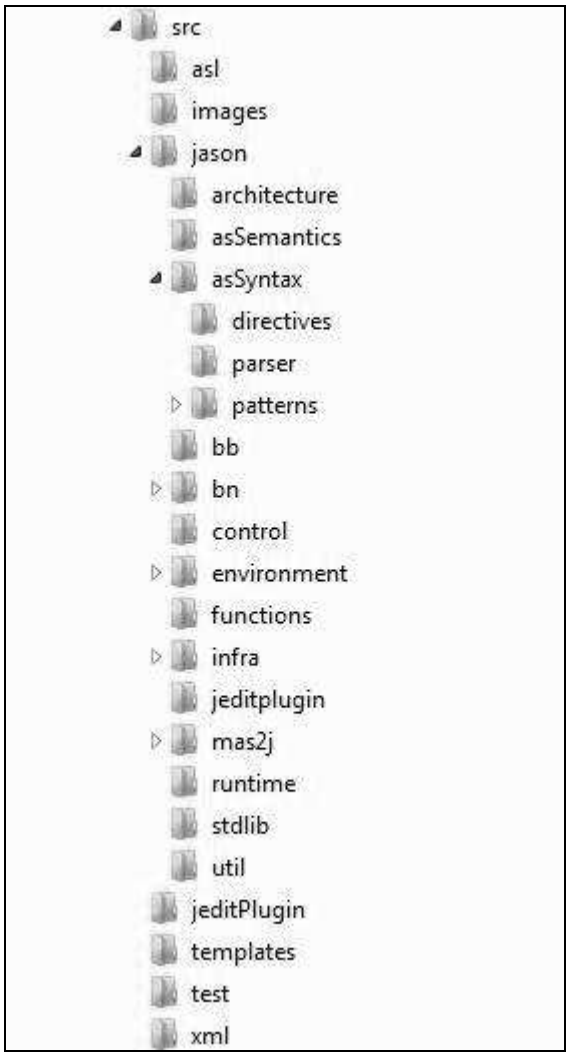


FIGURA 21. Estrutura de diretórios da pasta *jason*.

O diretório *jason* foi fundamental para o desenvolvimento de AgentSpeak(PL), porque contém os fontes que foram alterados para suportar essa linguagem.

Seguindo a estratégia usual de desenvolvimento de compiladores e interpretadores de linguagens de programação o processo de implementação de AgentSpeak(PL) foi dividido na análise léxica, sintática e semântica das novas construções da linguagem, seguido do processo de interpretação e execução destas novas construções.

5.1 Análise Léxica, Sintática e Semântica

A primeira tarefa no processo de transformação da ferramenta *Jason* na *JasonBayes* foi fazer com que a *Jason* aceitasse e reconhecesse a sintaxe de atribuição de probabilidades

as variáveis e seus eventos. Para isso foi necessário identificar em quais arquivos era realizada a análise léxica. Na implementação da ferramenta *Jason*, o diretório *asSyntax* contém os arquivos que correspondentes ao analisadores léxico, sintático e semântico de AgentSpeak(L). O subdiretório *parser* é responsável pelos os arquivos que realizam a análise léxica. Tanto o analisador léxico quanto o analisador sintático de AgentSpeak(L) foram desenvolvidos com o auxílio da ferramenta *JavaCC*. Essa ferramenta gera códigos fontes em Java tendo como entrada arquivos estruturados como mostrado na Figura 22.

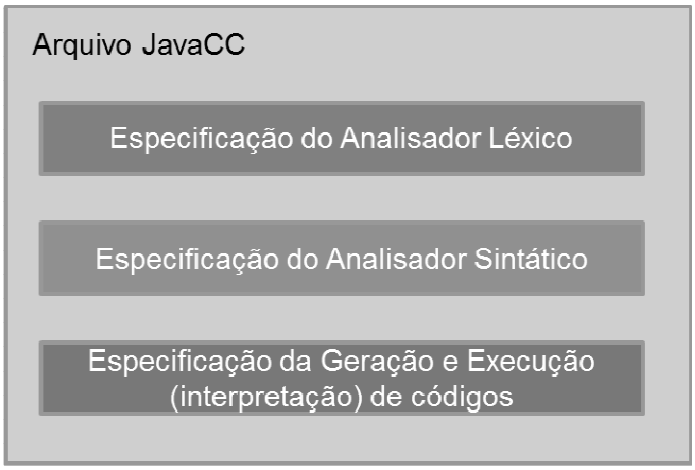


FIGURA 22. Estrutura de um arquivo em *JavaCC*.

O arquivo *AS2JavaParser.jcc* contém a especificação da análise léxica e sintática do *Jason* para AgentSpeak(L). Nesse arquivo estão identificados todos os literais e estruturas de comandos que a linguagem AgentSpeak(L) aceita. A estrutura geral da sintaxe de um código de agente AgentSpeak(L) tem uma organização tal como mostrada na Figura 23.

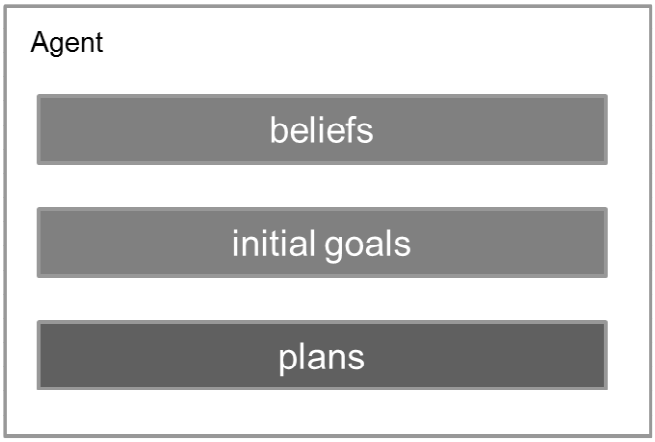


FIGURA 23. Estrutura de um agente em AgentSpeak(L).

Para suportar as novas construções de AgentSpeak(PL) o arquivo *AS2JavaParser.jcc* foi modificado de forma a permitir que na estrutura padrão de um agente, sejam incluídos os itens *beliefsprob* e *utils*, que tem o objetivo de definir, respectivamente, as declarações que formam a Rede Bayesiana do agente e as declarações que especificam funções de utilidade e variáveis de decisão associadas a essa rede, conforme mostra a Figura 24.

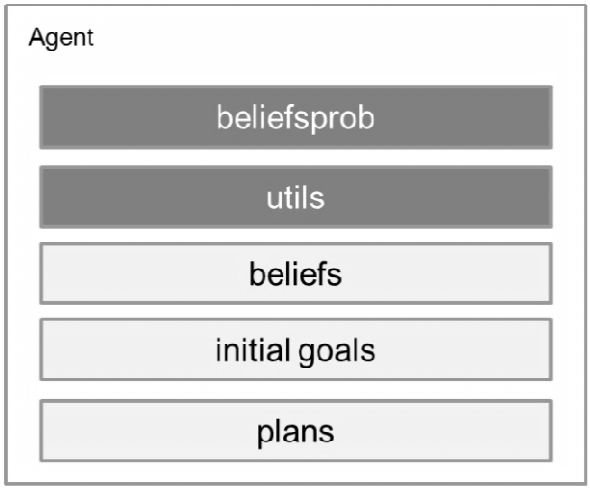


FIGURA 24. Estrutura de um agente em AgentSpeak(PL).

Abaixo destes itens foi implementado o código em *JavaCC* que aceita os *tokens* e estruturas sintáticas que permitem as declarações da Rede Bayesiana e das funções de utilidade, tal como definido no capítulo 4 (ver gramática de AgentSpeak(PL) definida na Figura 16).

Durante o processo de interpretação do código fonte de um agente *Jason*, todas as construções de AgentSpeak(L) são executadas através de funções puramente implementadas em Java. Esta mesma diretriz foi adotada na implementação de AgentSpeak(PL) no novo ambiente *JasonBayes*. Assim, as funções correspondentes as construções de AgentSpeak(PL) foram implementadas em duas classes chamadas de *bn.java* (para as variáveis e eventos probabilísticos da Rede Bayesiana) e *ut.java* (para as funções de utilidade), incluídas na estrutura de diretórios do *Jason*, a fim de serem compiladas juntamente com a ferramenta, conforme mostra a Figura 25.

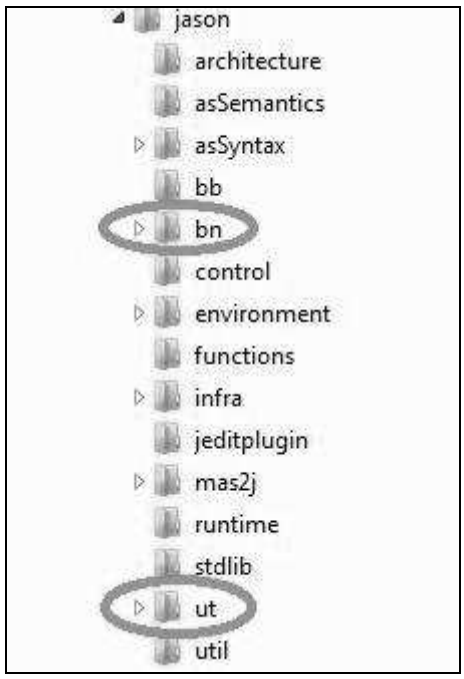


FIGURA 25. Diretórios das novas classes inseridas no *Jason*.

A classe *bn.java*, implementa funções que traduzem o código fonte AgentSpeak(PL) de uma Rede Bayesiana em um arquivo XML que representa essa rede em um formato compatível com diversos motores de inferência de Redes Bayesianas, como o Hugin [HUGIN, 2011], o *JavaBayes* [JAVABAYES, 2011] e o BNJ [BNJ, 2010], que está sendo usado na implementação do protótipo de AgentSpeak(PL). Na medida em que o analisador sintático e léxico é executado, as funções da classe *bn.java* são chamadas de forma a acrescentar as variáveis probabilísticas da rede, incluindo seus estados e valores em um arquivo XML que será posteriormente utilizado como entrada para o motor de inferência de Redes Bayesianas.

A classe *ut.java* contém funções que trabalham de forma similar a *bn.java*, transformando as construções correspondentes as funções de utilidade e variáveis de decisão em estruturas XML incorporadas no mesmo arquivo XML que contém a Rede Bayesiana.

No final do processo de análise léxica e sintática, é feita a análise semântica da Rede Bayesiana, das funções de utilidade e das variáveis de decisão. Nesta análise, o arquivo XML é submetido ao motor de inferência para uma verificação se não existe nenhum tipo de erro semântico na Rede Bayesiana ou nas funções de utilidade. São identificados nesta fase, erros como, a criação de um laço nas probabilidades condicionais da rede, a falta de alguma variável da Rede Bayesiana, a falta de definição da probabilidade de algum evento, ou a definição de uma função de utilidade sobre uma variável inexistente na Rede Bayesiana.

Caso algum erro seja detectado, esse erro é informado ao usuário e o processo de análise e interpretação do código fonte é interrompido. Como resultado do processo de análise inicial da Rede Bayesiana pelo motor de inferência também é obtida a distribuição marginal de probabilidades para todas as variáveis da rede, quando nenhuma evidência ainda é conhecida.

5.2 Interpretação e Execução do Código

A análise léxica, sintática e semântica das bases de crenças do agente, que inclui, no caso de AgentSpeak(PL), a Rede Bayesiana e as funções de utilidade do agente, é executada uma etapa antes da execução (interpretação) do restante do código fonte, que contém os planos do agente. Durante essa fase inicial é feito apenas a conferência de sintaxe e semântica, resultando no armazenamento temporário dessas informações que serão posteriormente usadas para que o agente possa tomar decisões em seus planos.

Após o término das análises iniciais, o Jason passa a executar o ciclo de vida dos agentes conforme mostrado na Figura 19 do capítulo 4. Dentro do ciclo de vida de um agente podem ocorrer várias etapas, conforme o que está programado em seu código fonte. Dentro dessas etapas, duas novas funções necessitaram ser implementadas para suportar as construções da linguagem AgentSpeak(PL). A primeira corresponde função “%()” que

executa a consulta de probabilidade marginal da Rede Bayesiana. A segunda corresponde a função “$” para consultar uma função utilidade previamente definida.

Essas funções foram implementadas na forma de função nativa do *Jason*, tal qual funções matemáticas como a *math.pi*, que retorna o valor de PI. Essas funções nativas ficam armazenadas dentro do diretório *functions*, conforme mostra a Figura 26.

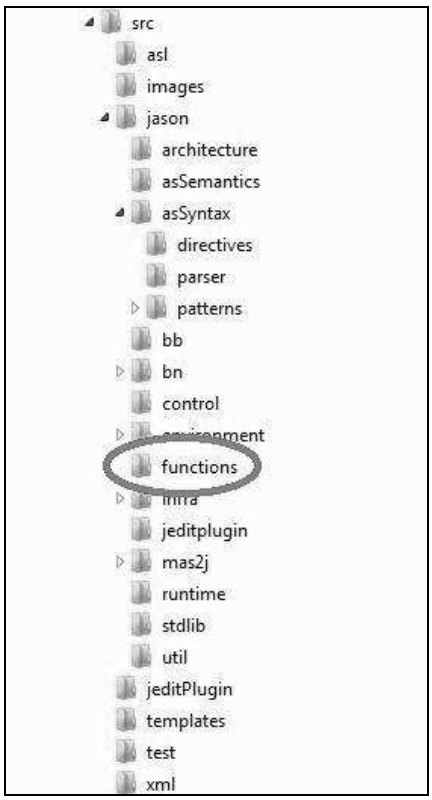


FIGURA 26. Diretórios das funções nativas do *Jason*.

A função “%” foi implementada em uma classe Java chamada *pb.java*, onde seus métodos seriam invocados a fim de consultar e alterar a estrutura da rede probabilística. Da mesma forma a função “$” foi implementada em uma classe chamada *ub.java*, e seus métodos manipulariam as funções de utilidades baseadas nas variáveis e seus eventos declaradas no topo do código dos agentes.

Para que ambas novas funções fossem aceitas, foi necessária a inclusão dos caracteres “%” e “$” como *tokens* dentro do analisador léxico e sintático. Especificamente dentro da subfunção *pred*, que define os predicados aceitos pela linguagem definida. Como as funções foram implementadas de forma nativa, nenhuma outra alteração na estrutura do analisador foi necessária.

Mesmo a descrição da Rede Bayesiana e sua posterior inferência sendo processos distintos e ocorrerem respectivamente na análise léxica, semântica e sintática e no ciclo de vida de um agente, também é possível realizar alterações nas redes probabilísticas ao longo do ciclo de vida de um agente de *software*. Essas alterações são feitas diretamente nos arquivos XML que armazenam as variáveis e suas evidências permitindo que as probabilidades sejam atualizadas e alteradas a critério do próprio agente ou se seu programador.

A cada alteração implementada, o código do Jason foi recompilado e testado, a fim de validar as mudanças e verificar a compatibilidade com AgentSpeak(L). Por fim foi obtido uma versão estável da nova versão do *Jason*, rebatizada como *JasonBayes*, que permite a programação e o desenvolvimento de projetos de agentes tanto em AgentSpeak(L) quanto em AgentSpeak(PL).

6 Aplicações de AgentSpeak(PL)

Após concluir a definição formal da nova linguagem e de implementá-la através do *JasonBayes*, foram escolhidos alguma cenários de aplicação e modelos probabilísticos retirados de textos introdutórios clássicos a respeito do assunto, em particular [RUSSEL; NORVIG, 2003].

Estes exemplos serão apresentados e posteriormente programados em AgentSpeak(PL), a fim de validar a nova linguagem e sua ferramenta de programação. Visando mostrar a gama de opções que a linguagem AgentSpeak(PL) pode oferecer, vários casos de estudo foram escolhidos: um modelo probabilístico relativamente simples a respeito de diagnóstico médico, um modelo probabilístico de decisão representado através de um Diagrama de Influência [RUSSEL; NORVIG, 2003], que combina Rede Bayesianas com funções de utilidade e variáveis de decisão e por fim um modelo do comportamento dinâmico de um sistema, representado através de uma Rede Bayesiana dinâmica. Ao final é apresentado um caso de estudo sobre uma aplicação prática de um modelo de sincronização de semáforos.

6.1 Caso de Estudo: Modelo de Diagnóstico Médico

Na Figura 27, é mostrado um exemplo clássico de Rede Bayesiana sobre como diagnosticar tuberculose, câncer, bronquite e doenças, com base em evidências de que o paciente tenha viajado para a Ásia, fuma ou está estressado [RUSSEL; NORVIG, 2003].

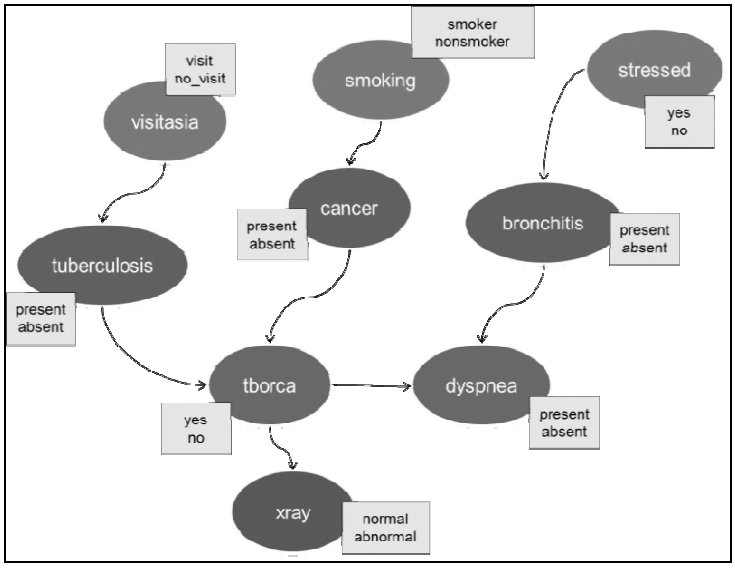


FIGURA 27. Rede Bayesiana clássica adaptada de [RUSSEL; NORVIG, 2003].

Através da Figura 28, podemos ver a programação do agente de *software* que implementa a Rede Bayesiana mostrada na figura anterior.

```
// Probabilistc Beliefs

// Standard Syntax for probabilistic beliefs
%visitasia(visit)                                       = 0.01 .
%visitasia(no_visit)                                    = 0.99 .
%smoking(smoker)                                        = 0.5 .
%smoking(nonsmoker)                                     = 0.5 .
%stressed(yes)                                          = 0.8 .
%stressed(no)                                           = 0.2 .
%tuberculosis(present)    | visitasia(visit)            = 0.05 .
%tuberculosis(present)    | visitasia(no_visit)         = 0.01 .
%tuberculosis(absent)     | visitasia(visit)            = 0.95 .
%tuberculosis(absent)     | visitasia(no_visit)         = 0.99 .

// Compact Notation (partial)
%cancer(present)          | smoking                     = [0.1, 0.01].
%cancer(absent)           | smoking                     = [0.9, 0.99].
%bronchitis(present)      | stressed                    = [0.6, 0.3].
%bronchitis(absent)       | stressed                    = [0.4, 0.7].
%tborca(yes)              | tuberculosis   & cancer     = [1.0, 1.0, 1.0, 0.0].
%tborca(no)               | tuberculosis   & cancer     = [0.0, 0.0, 0.0, 1.0].

// Compact Notation (full)
%xray(abnormal, normal)   | tborca                      = [0.98, 0.05, 0.02, 0.95].
%dyspnea(present, absent)| tborca         & bronchitis  = [0.9, 0.7, 0.8, 0.1, 0.1, 0.3, 0.2, 0.9].

// Initial Belief
plan1.

// Plans
+plan1: true <-
        math.randon(pvisitasia);
        math.randon(psmoke);
        +%visitasia(visit) = pvisitasia;
        +%visitasia(no_visit) = 1 - pvisitasia;
        +%smoking(smoker) = psmoke;
                            +%smoking(nonsmoker) = 1 - psmoke.

// Plans activated by probabilistic beliefs
+%cancer(present) > 0.9: true            <- .println ("Cancer present!").
+%bronchitis(present) > 0.9: true        <- .println ("Bronchitis present!").
+%tuberculosis(present) > 0.9: true      <- .println ("Tuberculosis present!").
```

FIGURA 28. Rede Bayesiana clássica adaptada de [RUSSEL; NORVIG, 2003] programada em AgentSpeak(PL) utilizando a ferramenta *JasonBayes*.

A implementação de AgentSpeak (PL) no *JasonBayes* suporta uma notação compacta, o que permite definir um modelo de Rede Bayesiana com um código-fonte muito menor do que a sintaxe padrão definida na Figura 28, mas pode sempre ser formalmente expandida para essa sintaxe padrão. No exemplo, é utilizada inicialmente a sintaxe padrão para definir a distribuição de probabilidade das variáveis *visitasia*, *smoking*, *stressed* e *tuberculosis*. Depois, é utilizada a notação compacta parcial para definir variáveis de *cancer*, *bronchitis* e *tborca*. Finalmente, a notação compacta completa é utilizada para definir as demais variáveis.

Depois do modelo probabilístico, é definido *plan1*, que é a única crença não-probabilística do agente. Essa crença não-probabilística permite executar o plano inicial do agente, que gera probabilidades aleatórias, as quais serão atribuídas as variáveis *visitasia* e *smoking*. Note que a função *math.random* irá gerar um número aleatório entre 0 e 1 e a soma dos eventos de uma variável deve ser igual a 100%. Como os valores são gerados aleatoriamente, em algum momento a probabilidade do evento *present* da variável *cancer* será maior que 0,9, ou 90%. Neste momento, o plano associado a este evento dessa variável irá apresentar uma mensagem informando que o câncer está presente. A mesma condição pode ser testada para as variáveis *bronchitis* e *tuberculosis*.

Este é um exemplo que tem como objetivo mostrar alguns aspectos práticos da Programação em AgentSpeak (PL). Naturalmente, a complexidade do agente de *software* pode ser aumentado, bem como com o seu domínio de aplicabilidade. Por exemplo, no caso de sistemas multiagentes é possível alterar as estimativas de probabilidades através da comunicação de evidências ocorridas entre agentes.

6.2 Caso de Estudo: Diagrama de Influência

Como segundo caso de estudo, temos um diagrama de influência que combina uma Rede Bayesiana com variáveis de decisão (nós retangulares) e funções de utilidade (nós losango). As variáveis de decisão representam as possíveis ações a serem tomadas por um agente dadas às circunstâncias das variáveis e eventos envolvidos. A função de utilidade define os ganhos (se a utilidade for positiva) ou perdas (se a utilidade for negativa) que poderão ocorrer em função de possíveis decisões e também de eventuais variáveis probabilísticas. A Figura 29 mostra um exemplo adaptado de [RUSSEL; NORVIG, 2003] onde a decisão a ser tomada definirá o local a ser instalado um aeroporto.

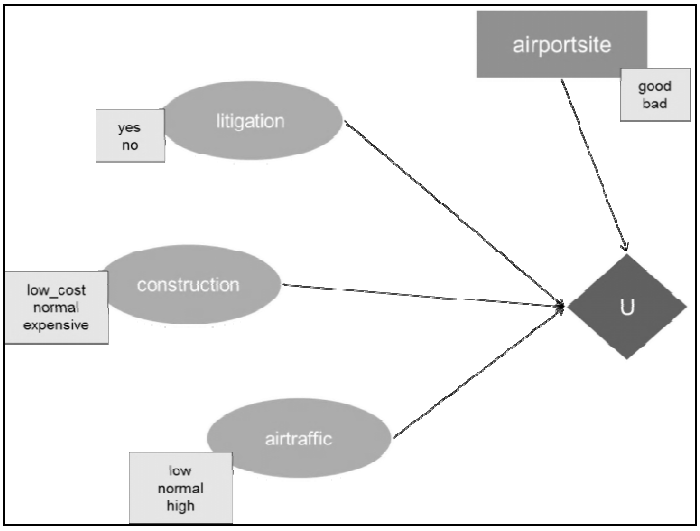


FIGURA 29. Rede Bayesiana com diagrama de influência e função utilidade. Adaptada de [RUSSEL; NORVIG, 2003].

Nesse diagrama os nós ovais representam as variáveis probabilísticas. O diagrama modela um processo de decisão a respeito de onde instalar um aeroporto. Podem existir dúvidas sobre o custo de construção (*construction*), o nível de tráfego aéreo (*airtraffic*) e o potencial para ocorrência de processos judiciais (*litigation*) a respeito de uma localização em particular. Cada nó desse tipo tem associado a ele uma distribuição condicional que é indexada pelo estado do nó pai. Os retângulos representam pontos de decisão onde há uma possibilidade de ação. Neste caso, a ação *airportsite* pode assumir um valor diferente para cada local em consideração. Por fim o losango representa a função de utilidade *U* tem como

pais todas as variáveis que descrevem o resultado que afetam diretamente a utilidade. A Tabela 4 mostra os valores da função utilidade *U*.

airportsite	litigation	construction	airtraffic	U(U|airportsite)
good	yes	low_cost	low	-20
good	yes	low_cost	normal	-34
good	yes	low_cost	high	3
good	yes	normal	low	4
good	yes	normal	normal	70
good	yes	normal	high	30
good	yes	expensive	low	10
good	yes	expensive	normal	1000
good	yes	expensive	high	230
good	no	low_cost	low	-23
good	no	low_cost	normal	45
good	no	low_cost	high	100
good	no	normal	low	2000
good	no	normal	normal	-450
good	no	normal	high	-79
good	no	expensive	low	34
good	no	expensive	normal	78
good	no	expensive	high	67
bad	yes	low_cost	low	50
bad	yes	low_cost	normal	489
bad	yes	low_cost	high	-62
bad	yes	normal	low	34
bad	yes	normal	normal	-20
bad	yes	normal	high	-233
bad	yes	expensive	low	333
bad	yes	expensive	normal	400
bad	yes	expensive	high	32
bad	no	low_cost	low	3095
bad	no	low_cost	normal	-398
bad	no	low_cost	high	2096
bad	no	normal	low	2965
bad	no	normal	normal	-948
bad	no	normal	high	-323
bad	no	expensive	low	494
bad	no	expensive	normal	982
bad	no	expensive	high	9

TABELA 3. Tabela de função utilidade.

Com os valores definidos pode-se então programar o código em AgentSpeak(PL) conforme mostra a Figura 30.

```
// Utility Function

// Probabilistic beliefs
% litigation(yes)                = 0.5 .
% litigation(yes)                = 0.5 .
% construction(low_cost)         = 0.1 .
% construction(normal)           = 0.7 .
% construction(expensive)        = 0.2 .
% aitraffic(low)                 = 0.6 .
% aitraffic(normal)              = 0.3 .
% aitraffic(high)                = 0.1 .

// Standard Syntax for utility function
$ U ( airportsite(good)  , litigation(yes) , construction(low_cost) , airtraffic(low) )     = -20 .
$ U ( airportsite(good)  , litigation(yes) , construction(low_cost) , airtraffic(normal) )  = -34 .
$ U ( airportsite(good)  , litigation(yes) , construction(low_cost) , airtraffic(high) )    = -3 .
$ U ( airportsite(good)  , litigation(yes) , construction(normal) , airtraffic(low) )       = 4 .
$ U ( airportsite(good)  , litigation(yes) , construction(normal) , airtraffic(normal) )    = 70 .
$ U ( airportsite(good)  , litigation(yes) , construction(normal) , airtraffic(high) )      = 30 .
$ U ( airportsite(good)  , litigation(yes) , construction(expensive) , airtraffic(low) )    = 10 .
$ U ( airportsite(good)  , litigation(yes) , construction(expensive) , airtraffic(normal) ) = 1000 .
$ U ( airportsite(good)  , litigation(yes) , construction(expensive) , airtraffic(high) )   = 230 .
$ U ( airportsite(good)  , litigation(no) , construction(low_cost) , airtraffic(low) )      = -23 .
$ U ( airportsite(good)  , litigation(no) , construction(low_cost) , airtraffic(normal) )   = 45 .
$ U ( airportsite(good)  , litigation(no) , construction(low_cost) , airtraffic(high) )     = 100 .
$ U ( airportsite(good)  , litigation(no) , construction(normal) , airtraffic(low) )        = 2000 .
$ U ( airportsite(good)  , litigation(no) , construction(normal) , airtraffic(normal) )     = -450 .
$ U ( airportsite(good)  , litigation(no) , construction(normal) , airtraffic(high) )       = -79 .
$ U ( airportsite(good)  , litigation(no) , construction(expensive) , airtraffic(low) )     = 34 .
$ U ( airportsite(good)  , litigation(no) , construction(expensive) , airtraffic(normal) )  = 78
$ U ( airportsite(good)  , litigation(no) , construction(expensive) , airtraffic(high) )    = 67 .
$ U ( airportsite(bad)   , litigation(yes) , construction(low_cost) , airtraffic(low) )     = 50.
$ U ( airportsite(bad)   , litigation(yes) , construction(low_cost) , airtraffic(normal) )  = 489 .
$ U ( airportsite(bad)   , litigation(yes) , construction(low_cost) , airtraffic(high) )    = -62 .
$ U ( airportsite(bad)   , litigation(yes) , construction(normal) , airtraffic(low) )       = 34 .
$ U ( airportsite(bad)   , litigation(yes) , construction(normal) , airtraffic(normal) )    = -20 .
$ U ( airportsite(bad)   , litigation(yes) , construction(normal) , airtraffic(high) )      = -233 .
$ U ( airportsite(bad)   , litigation(yes) , construction(expensive) , airtraffic(low) )    = 333 .
$ U ( airportsite(bad)   , litigation(yes) , construction(expensive) , airtraffic(normal) ) = 400 .
$ U ( airportsite(bad)   , litigation(yes) , construction(expensive) , airtraffic(high) )   = 32 .
$ U ( airportsite(bad)   , litigation(no) , construction(low_cost) , airtraffic(low) )      = 3095.
$ U ( airportsite(bad)   , litigation(no) , construction(low_cost) , airtraffic(normal) )   = -398 .
$ U ( airportsite(bad)   , litigation(no) , construction(low_cost) , airtraffic(high) )     = 2096 .
$ U ( airportsite(bad)   , litigation(no) , construction(normal) , airtraffic(low) )        = 2965 .
$ U ( airportsite(bad)   , litigation(no) , construction(normal) , airtraffic(normal) )     = -948 .
$ U ( airportsite(bad)   , litigation(no) , construction(normal) , airtraffic(high) )       = -323 .
$ U ( airportsite(bad)   , litigation(no) , construction(expensive) , airtraffic(low) )     = 494 .
$ U ( airportsite(bad)   , litigation(no) , construction(expensive) , airtraffic(normal) )  = 982 .
$ U ( airportsite(bad)   , litigation(no) , construction(expensive) , airtraffic(high) )    = 9 .

// Initial Belief
plan1.

// Plans activated by probabilistic beliefs and utility function
+airportsite(good): $U(airportsite(good)) > 2000
        <- .println ("Excellent location to construct one airport!").
+airportsite(bad): $U(airportsite(bad)) < -340
        <- .println ("Find another location to construct your airport!").
```

FIGURA 30. Implementação em AgentSpeak(PL) de uma Rede Bayesiana com diagrama de influência e função utilidade.

Após a definição dos valores probabilísticos e da função utilidade temos uma crença inicial chamada *plan1*. Essa crença serve para iniciar o ciclo de vida do agente. Dois planos foram definidos. O primeiro será ativado quando o local do aeroporto for considerado bom, baseado nos valores definidos previamente e na alteração dos valores das variáveis. Quando este plano for ativado, significa que frente ao cenário instantâneo, o local escolhido para construção do aeroporto é bom. Já no outro plano acontece ao contrário, e é ativado quando o cenário é desfavorável, aconselhando a escolher outro local para construção. Considera-se para a tomada de decisões, neste caso, não somente as probabilidades, mas também um peso

atribuído a cada uma delas que representa o diagrama de decisão e função utilidade. Dada uma determinada distribuição probabilística dos eventos pertencentes as variáveis do agente, o cálculo da distribuição de probabilidades resultará em um valor absoluto que será comparado com outro valor pré-determinado, resultando em um tomada de decisão.

6.3 Caso de Estudo: Rede Bayesiana Dinâmica

No terceiro exemplo é mostrado a implementação de uma Rede Bayesiana Dinâmica (RBD), que se expande ao longo do tempo dado um conjuntos de variáveis e estados. Essa expansão ocorre com o objetivo de prever uma possível ação de determinado agente, alterando os valores futuros dos eventos e verificando os resultados das funções de utilidades.

Uma RBD é uma Rede Bayesiana que representa um modelo de probabilidade temporal. Em geral, cada fatia de tempo de uma RBD pode ter qualquer número de variáveis de estados. Para simplificar, podemos assumir que as variáveis e suas ligações são exatamente replicadas fatia a fatia de tempo e que a RBD representa uma cadeia de Markov de primeira ordem, onde cada variável pode ter só os pais na sua própria fatia de tempo ou na fatia imediatamente anterior [RUSSEL; NORVIG, 2003].

Cada modelo oculto de Markov pode ser representado como uma RBD com uma única variável de estado e uma única variável de evidências. Podemos dizer que cada cadeia de Markov pode ser uma RBD e cada RBD pode ser traduzida em uma cadeia de Markov. A diferença entre elas é que, pela decomposição do estado de um sistema complexo em suas variáveis, a RBD é capaz de tirar proveito de escassez no modelo de probabilidades temporais. Supondo, por exemplo, que uma RBD tenha 20 variáveis de estados booleanos, cada qual tendo três pais da fatia de tempo anterior. O modelo de transição RBD terá 20 x 23 = 160 probabilidades, enquanto que a cadeia de Markov correspondente terá 220 estados e, portanto, 240, ou cerca de um trilhão de probabilidades na matriz de transição [RUSSEL; NORVIG, 2003].

Para ilustrar as características das RBDs será utilizado o seguinte exemplo, retirado de [RUSSEL; NORVIG, 2003]: suponha que você é o guarda de segurança em alguma instalação subterrânea secreta. Você quer saber se está chovendo hoje, mas seu único acesso ao mundo exterior ocorre a cada manhã, quando você vê o diretor chegar com, ou sem um guarda-chuva. Para cada dia *t*, o conjunto *Et* contém, assim, uma única variável de evidência *Ut* (se o guarda-chuva aparece), e o conjunto *Xt* contém uma única variável de estado *Rt* (se estiver chovendo) .

O intervalo entre as frações de tempo também depende do problema. Para o monitoramento do diabetes, um intervalo de tempo pode ser de uma hora ao invés de um dia. Neste exemplo, consideramos um intervalo fixo finito, o que significa que os tempos podem ser rotulados por números inteiros. Supondo que a sequência do estado comece em t = 0; e que a prova começa a chegar em t = 1. O guarda-chuva é representado pelos estados das variáveis *R0, R1, R2*; e variáveis de evidências *Ul, U2,...*

A inferência em Redes Bayesianas Dinâmicas ocorre da mesma forma que em Redes Bayesianas normais. Dada uma sequência de observações em uma fatia de tempo discreto e finito, pode-se construir a representação completa da Rede Bayesiana de uma RBD, replicando as fatias até que a rede seja grande o suficiente para acomodar as observações desejadas. Essa técnica é chamada de *Unrolling*. Uma vez que a técnica de *Unrolling* seja aplicada em uma RBD, pode-se usar qualquer algoritmo de inferência disponível [RUSSEL; NORVIG, 2003].

Conforme mostra a Figura 31, observamos a Rede Bayesiana em sua forma original.

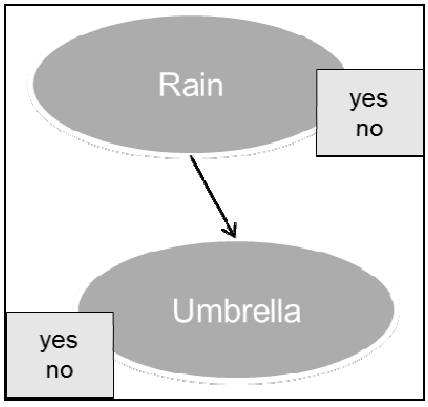


FIGURA 31. Rede Bayesiana original. Adaptada de [RUSSEL; NORVIG, 2003].

Como a rede é tratada de forma dinâmica, podemos visualiza-la na forma temporal como mostra a Figura 32.

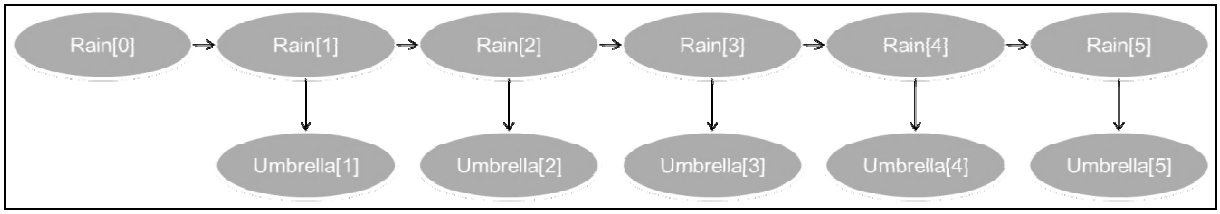


FIGURA 32. Rede Bayesiana Dinâmica em cinco fatias de tempo. Adaptada de [RUSSEL; NORVIG, 2003].

A programação em AgentSpeak(PL) de RBDs com um tamanho fixo, como a RBD do exemplo da Figura 32 com 5 fatias de tempo, é exatamente igual aos exemplos vistos anteriormente.

Entretanto, para permitir o tratamento de RBDs cujo tamanho seja determinado durante o tempo de execução do agente, está previsto em AgentSpeak(PL) o suporte a comandos de atualização da estrutura da Rede Bayesiana em tempo de execução.

Estes comandos não foram definidos na semântica formal de AgentSpeak(PL), assim aqui será feita apenas uma introdução informal das capacidades destes comandos. A Rede Bayesiana do agente poderá ser alterada, através de ações de na estrutura desta rede, que podem ter as seguintes formas:

(a) ‘+%%$c(x)=p$’

(b) ‘+%%$c(x) \mid e_1(y_1)\&e_2(y_2)\&...\&e_n(y_n)=p$’

(c) ‘-%$c(x)=p$’

(d) ‘-%%$c(x) \mid e_1(y_1)\&e_2(y_2)\&...\&e_n(y_n)=p$’

(e) ‘-+%%$c(x)=p$’

(f) ‘-+%%$c(x) \mid e_1(y_1)\&e_2(y_2)\&...\&e_n(y_n)=p$’

(g) ‘+-%%’

As ações nos formatos ‘+%%...’ e ‘-%%...’, respectivamente, adicionam ou eliminam uma probabilidade prévia ou condicional, enquanto que ações no formato ‘-+%%...’ alteram o valor uma probabilidade prévia ou condicional pré-existente.

A consistência do modelo probabilístico é garantida, evitando-se que as ações de alterações sejam aplicadas imediatamente. Uma alteração consistente do modelo probabilístico irá requerer a execução de várias ações de alteração, até que todas as novas probabilidades prévias e condicionais estejam corretas. Para evitar problemas de inconsistência e geração de eventos probabilísticos espúrios, as ações de alteração que um dado agente executa não são aplicadas ao modelo atual, ficando pendentes até que uma ação de atualização do modelo probabilístico, que tem a forma ‘+-%%’ de uma ação de alteração “nula” é executada. A ação de atualização ‘+-%%’ aplica todas as ações pendentes de alteração de forma monolítica e de uma única vez, antes que alguma nova evidência seja agregada a base ou que alguma ação de consulta de valores de probabilidades seja efetuada. Após a execução de ‘+-%%’, a lista de alterações pendentes é limpa, podendo ser preenchida através de novas operações de alteração do modelo probabilístico.

Note que há uma diferença importante entre a adição de uma evidência ao modelo probabilístico e a alteração da probabilidade prévia associada a esta evidência. Evidências relacionadas ao modelo probabilístico podem ser incorporadas a base de crenças do agente pelo próprio agente através de ações ‘*-+b(x)*’ em qualquer instante. Da mesma forma evidências podem ser excluídas por meio de ações ‘*-b(x)*’. Supondo uma variável $b(x)$ com eventos básicos $x \in \{a,b\}$, então, embora a adição de uma evidência ‘*-+b(a)*’ tenha o mesmo efeito, em se tratando de cálculo de probabilidades, que a modificação do modelo quantitativo desta variável feita pelos comandos ‘*+%%b(a)=1.0;+%%b(b)=0.0;+%%;*’, existe uma diferença importante em termos de efeitos colaterais no modelo probabilístico do agente. No caso da alteração de evidências, se posteriormente houver uma exclusão da evidência ‘*-b(a)*’ então o modelo prévio atribuído à variável *b(x)* volta a valer, por outro lado, os comandos ‘*+%%b(a)=1.0;+%%b(b)=0.0;+%%;*’ alteram efetivamente o modelo quantitativo de *b(x)*, estes são agora aos valores prévios para a distribuição de probabilidade de *b(x)*.

Usando os comandos de atualização da Rede Bayesiana, então a RBD da Figura 32 pode ser implementada em AgentSpeak(PL) de acordo como mostrado na Figura 33.

```
// RB inicial da RBD
// Define o modelo de sensores da RBD
// Aqui sera usada apenas para referencia
// de valor inicial de probabilidade
% rain([yes,no])= [0.5, 0.5].
% umbrella([yes,no]) | rain (yes)= [0.5, 0.5].
% umbrella([yes,no]) | rain (no) = [0.5, 0.5].

// Initial Belief
plan1.

// Plans
+plan1: true <-
   // Cria a RBD com 5 slices de tempo. Usa os comandos dinamicos de alteracao
   // da rede bayesiana do agente para gerar os timeslices

   // RB do Timeslice 0
   +%%rain_0(yes)= %rain(yes).
   +%%rain_0(no)= %rain(no).

   // RB do Timeslice 1.
   +%%rain_1([yes,no]) | rain(yes)= [0.5, 0.5].
   +%%rain_1([yes,no]) | rain(no) = [0.5, 0.5].
   +%% umbrella_1([yes,no]) | rain_1 (yes)= [0.5, 0.5].
   +%% umbrella_1([yes,no]) | rain_1 (no) = [0.5, 0.5].


   // RB do Timeslice 2.
   +%%rain_2([yes,no]) | rain_1(yes)= [0.5, 0.5].
   +%%rain_2([yes,no]) | rain_1(no) = [0.5, 0.5].
   +%% umbrella_2([yes,no]) | rain_2 (yes)= [0.5, 0.5].
   +%% umbrella_2([yes,no]) | rain_2 (no) = [0.5, 0.5].


    // Gera os demais timeslices
   ...


   // Apos gerar o ultimo timeslice atualiza a RBD e
   // computa o proximo estado, no caso o timeslice 4:
   +-%%;


   // Nesse ponto a RBD esta' completa e pode ser usada pelo agente
   ...
```

FIGURA 33. Rede Bayesiana Dinâmica implementada em AgentSpeak(PL).

6.4 Caso de Estudo Prático

Inúmeras outras aplicações podem ser simuladas e implementadas através da AgentSpeak(PL). Como exemplo prático de uso pode-se citar a solução dos problemas de tráfego de veículos em uma determinada cidade. No contexto urbano atual, a cada dia que passa observarmos um incremento no número de veículos que trafegam nas metrópoles, o que acarreta um trânsito lento e aumento do tempo de percurso dos trajetos. Um fator agravante para este cenário é o número elevado de semáforos em cruzamentos, onde cada um deles age de maneira independente e com temporizações nem sempre ideais.

Para solucionar esse problema, órgãos de trânsito têm utilizado sincronismos simples entre semáforos, ou seja, com base em horas, minutos e segundos, cada um deles permanecendo um determinado tempo verde ou vermelho. Esse procedimento tem-se mostrado eficiente até certo ponto, pois quase nunca o fluxo é constante em determinada hora, minuto e segundo.

Para resolver essa situação de forma mais abrangente, um exemplo de aplicação da linguagem AgentSpeak(PL) seria programar um sistema multiagente que realize a gestão de vários cruzamentos através de um conjunto de semáforos, contribuindo assim para o aumento da fluidez do trânsito. Desta forma, a aplicação levaria em conta os semáforos de cada cruzamento onde há maior fluxo de veículos bem como os locais onde existem sequências de semáforos de maneira a garantir uma melhor fluidez do trânsito.

O sistema seria constituído por agentes, em que cada um seria responsável pela gestão de um semáforo, com o objetivo de minimizar o tempo médio de espera dos veículos que estariam diretamente dependentes dele.

O agente apenas poderia decidir se deveria mudar o estado atual do semáforo, utilizando para tomar esta decisão os seguintes aspectos:

- Fluxo de tráfego em cada direção
- Tempo do sinal verde numa dada direção
- Informação dos agentes vizinhos
- Decisões mais recentes do próprio agente.

O sistema seria constituído por vários agentes que se comunicam entre si, de forma a terem consigo informações referentes ao número de carros que se encontram ou nas proximidades do semáforo correspondente ao agente, ou nos semáforos adjacentes, de forma a poder tomar uma decisão.

Esse contexto facilitaria a *Onda Verde* (procedimento onde os semáforos de uma determinada via ficam verdes em sequencia, permitindo ao veículo realizar o trajeto sem efetuar novas paradas). Com uma base de crenças probabilísticas, cada agente ou um determinado grupo de agentes poderia analisar as condições baseado em probabilidades, e realizar eventos tais como: não executar a *Onda Verde*, executá-la em toda via executá-la em alguns trechos, etc.

A Figura 34 mostra um exemplo compacto de uma Rede Bayesiana que corresponde à base de crenças de um agente controlador de semáforo.

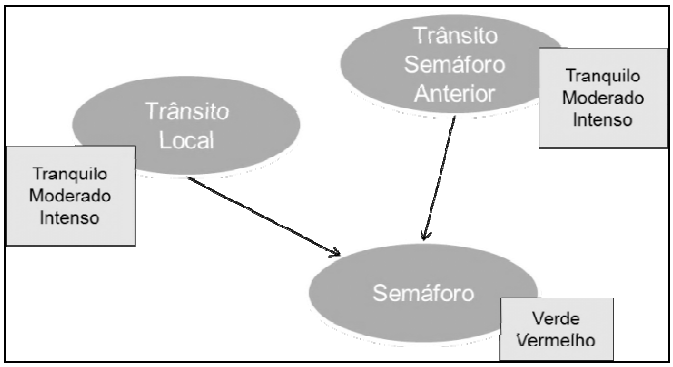


FIGURA 34. Rede Bayesiana de um agente controlador de semáforo.

Esta Rede Bayesiana representa um único agente que controlaria um único semáforo de dois tempos (um cruzamento). O nó de *Trânsito Semáforo Anterior* refere-se ao semáforo/agente anterior ao semáforo atual em uma via de mão única e é alimentado por tal agente. Os dados do nó Trânsito Local referem-se ao trânsito local ao semáforo e é alimentado por sensores localizados na rua. Frente a estas probabilidades têm-se as estimativas do nó Semáforo, que aumentaria ou diminuiria os tempos do sinal verde e vermelho.

O código da implementação deste agente em AgentSpeak(PL) é mostrado na Figura 35.

```
// Crencas probabilisticas
% transitolocal([Tranquilo,Moderado,Intenso])= [1.0, 0.0, 0.0].
% transitosemaforoanterior([Tranquilo,Moderado,Intenso])= [0.3, 0.7, 0.0].
% semaforo([verde, vermelho]) | transitolocal(Tranquilo) & transitosemaforoanterior(Tranquilo) =
[0.5, 0.5].
% semaforo([verde, vermelho]) | transitolocal(Tranquilo) & transitosemaforoanterior(Moderado) =
[0.6, 0.4].
% semaforo([verde, vermelho]) | transitolocal(Tranquilo) & transitosemaforoanterior(Intenso) =
[0.7, 0.2].
% semaforo([verde, vermelho]) | transitolocal(Moderado) & transitosemaforoanterior(Tranquilo) =
[0.6, 0.4].
% semaforo([verde, vermelho]) | transitolocal(Moderado) & transitosemaforoanterior(Moderado) =
[0.7, 0.3].
% semaforo([verde, vermelho]) | transitolocal(Moderado) & transitosemaforoanterior(Intenso) =
[0.8, 0.2].
% semaforo([verde, vermelho]) | transitolocal(Intenso) & transitosemaforoanterior(Tranquilo) =
[0.7, 0.3].
% semaforo([verde, vermelho]) | transitolocal(Intenso) & transitosemaforoanterior(Moderado) =
[0.8, 0.2].
% semaforo([verde, vermelho]) | transitolocal(Intenso) & transitosemaforoanterior(Intenso) =
[0.9, 0.1].

// Crenca inicial
plan1.

// Planos

// Inicializacao do agente
+plan1: true <-
   // Inicializa variaveis
   tempoverde = 30;
   tempovermelho = 30;
   / Ativa ciclo de vida do agente
   ciclo.

// Ciclo de vida do agente
+ciclo: true <-
   // Le sensores e atualiza RB
   .lesensores();
   // Verifica tempos e manipula sinal verde e vermelho
   .controlasemaforo();
   // Se auto ativa para ficar em looping
   ciclo.

// Recebeu probabilidades novas do agente/semaforo anterior
+transitolocal(E)[source(agenteanterior)]: true <-
   // Recalcula RB
   +-%%;
   // Envia mensagem para o semaforo da frente com novo valor
   .send(proximoagente,tell,%transitolocal(E));

// Sensores locais alteraram valores da probabilidade
+%transitolocal(E): true <-
   // Recalcula RB
   +-%%;
   // Envia mensagem para o semaforo da frente com novo valor
   .send(proximoagente,tell,%transitolocal(E));

// Rede foi atualizada. Aumenta ou diminui tempos do verde
+%semaforo(verde): true <-
   if (%semaforo(verde) > 0.5) { tempoverde = tempoverde + 10; }
   else { tempoverde = tempoverde - 10; }

// Rede foi atualizada. Aumenta ou diminui tempos do vermelho
+%semaforo(vermelho): true <-
   if (%semaforo(vermelho) > 0.5) { tempovermelho = tempovermelho + 20; }
   else { tempovermelho = tempovermelho - 20; }

// fim
```

FIGURA 35. Agente controlador de semáforo implementado em AgentSpeak(PL).

7 Conclusões

Haja a vista a proposta inicial deste trabalho de desenvolver uma nova linguagem de programação de agentes com base de crenças probabilísticas e também desenvolver uma ferramenta de desenvolvimento que permitisse o uso de tal linguagem, o trabalho foi desenvolvido em sua totalidade atingindo os objetivos propostos.

As etapas cumpridas no processo de desenvolvimento passaram inicialmente pelo contexto teórico, sendo desenvolvida uma nova semântica informal e formal que atendesse as necessidades da linguagem. Posteriormente um ambiente de programação foi desenvolvido a fim de permitir que a nova linguagem AgentSpeak (PL) pudesse ser implementada e utilizada na programação de agentes bem como oferecer uma solução completa e compacta para quem quisesse implementar projetos de sistemas multiagente baseados em crenças probabilísticas. Essa ferramenta foi disponibilizada de uma forma aberta de maneira a ser utilizada tanto para fins educacionais quando fins comerciais, visando cobrir lacunas que até então existiam na programação orientada a agentes.

Alguns exemplos de uso da linguagem AgentSpeak (PL) integrada ao JasonBayes foram implementados, com o intuito de validar o trabalho desenvolvido e mostrar a forma prática com que agentes com crenças probabilísticas podem ser criados.

O desenvolvimento de uma nova linguagem de programação de agentes para construção de agentes com crenças probabilísticas permite que novos cenários sejam desenvolvidos e testados de forma a facilitar estudos e projeções nos mais diversos ambientes e áreas, tornando-se uma ferramenta mais multidisciplinar.

Com esta implementação será possível fazer o uso de redes probabilísticas dentro da base de dados de crenças dos agentes, fortalecendo o conceito de programação orientada a agentes e unindo tecnologias que antes convergiam individualmente. A linguagem AgentSpeak(PL) e o ambiente *JasonBayes* oferecem uma contribuição efetiva neste sentido. Uma primeira versão deste trabalho mostrando a especificação de AgentSpeak(PL) sem o suporte para os modelos de decisão e de utilidade, foi aceito na *International Conference on Information Science and Applications* de 2011, apoiada pela IEEE, que se realizará em Jeju, na Coréia do Sul de 26 a 29 de abril de 2011. Este artigo está incluído em anexo.

É importante salientar, entretanto, que este ainda é um trabalho em evolução. Há uma necessidade de novas abstrações e novas linguagens de programação que integrem cada vez mais os conceitos probabilísticos e de sistemas multiagente.

Um desafio para o futuro seria desenvolver uma ferramenta de compilação para programação orientada a agentes, a fim de desenvolver agentes capazes de serem executados em diferentes plataformas, tais como microcontroladores embarcados. Diferentemente da execução de agentes em uma plataforma de desenvolvimento onde eles são manipulados pelo ambiente em que foram programados, um compilador permitiria a execução de um agente independente, sem a necessidade de um ambiente de execução ou um interpretador.

Referências Bibliográficas

ALMEIDA, F. M. L.; VALE, M. A. B.; **Extracting Signal Limits from Samples with Multiple Independent Backgrounds.** Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, Capítulo de livro, Elsevier, 438, n. 2, p. 511-522, 1999.

ANAND S. R.; **AgentSpeak(L): BDI agents speak out in a logical computable language.** MAAMAW '96 Proceedings of the 7th European workshop on Modelling autonomous agents in a multi-agent world : agents breaking away: agents breaking away, Artigo, Springer-Verlag, New York, Inc. Secaucus, NJ, USA, 1996.

BNJ; **http://bnj.sourceforge.net/** acessado em 05/2010.

BORDINI, R. H.; FISHER, M.; VISSER, W.; WOOLDRIDGE, M.; ***Verifiable multi-agent programs. Programming Multi-Agent Systems.*** Capítulo de livro, Berlin / Heidelberg, Springer, v. 3067, p. 72-89, 2004.

BORDINI, R. H.; WOOLDRIDGE, M.; HÜBNER, J. F.; ***Programming Multi-Agent Systems in AgentSpeak using Jason.*** Livro, John Wiley & Sons, 2007.

BRATMAN, M. E.; ***Intentions, Plans and Practical Reason.*** Livro, 208 p., Center for the Study of Language and Inf, 1987.

CHAN, H.; DARWICHE, A.; **On the revision of probabilistic beliefs using uncertain evidence.** Artificial Intelligence archive, Journal, v. 163 Issue 1, Elsevier Science Publishers Ltd. Essex, UK, March 2005.

CHARNIAK, E.; ***Bayesians Networks without Tears.*** *AI Magazine archive, Journal*, American Association for Artificial Intelligence Menlo Park, CA, USA, v. 12 Issue 4, Winter 1991.

CHEONG, F. C.; **Internet Agents - Spiders, Wanderers, Brokers and Bots.** Internet agents: spiders, wanderers, brokers, and bots, Livro, New Riders Publishing Indianapolis, IN, USA 1996.

CONTE, R.; GILBERT, N.; **Artificial Societies: the Computer Simulation of Social Life. Artificial Societies:** The Computer Simulation of Social Life, Livro, Taylor & Francis, Inc. Bristol, PA, USA, 1995**.**

DENNET, D. C.; ***The Intentional Stance*****.** Library of Congress Cataloging-in-Publication Data, Livro, MIT Press, MA, USA, 1989.

FAGUNDES, M. S.; ***Integrating BDI Model and Bayesian Networks***. Dissertação de Mestrado, UFRGS, 2007.

GARCIA, A. F.; DE LUCENA, C. J. P.; ZAMBONELLI, F.; OMICINI, A.; CASTRO, J.; ***Software Engineering for Large-Scale Multi-Agent Systems, Research Issues and Practical Applications.*** Livro, v. 3914, XIV, 255 p., Softcover, 2003.

HALPERN, J.; MOSES, Y.; A guide to completeness and complexity for modal logics of knowledge and belief. Artificial Intelligence, Amsterdan, n. 52, p. 311-379, 1992.

HAYES, B.; ***An Architecture for Adaptive Intelligent System.*** *Artificial Intelligence archive, Journal*, v. 72 Issue 1-2, Elsevier Science Publishers Ltd. Essex, UK, January, 1995.

HUGIN; **http://www.hugin.com/** acessado em 02/2011.

HYACINTH, S. N.; ***Software Agents: An Overview.*** *Knowledge Engineering Review*, Artigo, v. 11, n. 3, pp.1-40, Cambridge University Press, 1996.

JASON; **http://jason.sourceforge.net/Jason/Jason.html** acessado em 02/2011.

JAVABAYES; **http://www.cs.cmu.edu/~fgcozman/Research/JavaBayes/Home/** acessado em 02/2011.

KERSTING, K.; RAEDT, L. D.; ***Bayesian Logic Programs*.** Relatório técnico, Albert-Ludwigs University at Freiburg, 2000.

KORB, K. B., NICHOLSON, A.E.; ***Bayesian Artificial Intelligence.*** The Knowledge Engineering Review archive, *Journal*, v. 19 Issue 3, Cambridge University Press New York, NY, USA, 2004.

MACHADO, R.; BORDINI, R. H.; ***Running AgentSpeak(L) agents on SIM_AGENT.*** *Intelligent Agents*, Capítulo de livro, v.2333, p. 158-174, Springer Berlin / Heidelberg, 2002.

MILCH, B.; KOLLER, D.; **Probabilistic Models for Agents' Beliefs and Decisions.** UAI '00 Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence, Artigo, Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 2000.

MOREIRA, A. F.; BORDINI, R. H.; **An operational semantics for a BDI agentoriented programming language.** Declarative Agent Languages and Technologies, Capítulo de livro, v. 2990, p. 1270-1270, Springer Berlin / Heidelberg 2004.

PEARL, J.; **Belief Networks Revisited.** Artificial Intelligence, Amsterdam, n. 59, p. 49-56, 1993.

RUSSEL, S.; NORVIG, P.; ***Artficial Intelligence: A Modern Approach.*** Livro, Pearson Education, 2003.

SHOHAM, Y.; ***Agent-oriented programming.*** *Artificial Intelligence, Journal*, v. 60 Issue 1, Elsevier Science Publishers Ltd. Essex, UK, 1994.

W3C; **http://www.w3.org/** acessado em 02/2011.

WIRTH, N.; ***Compiler Construction.*** Livro, v. 32 Issue 2, ACM New York, NY, USA, 1997.

WOOLDRIGDE, M.; JENNINGS, N. R.; **Intelligent Agents - Theories, Architectures, and Languages.** ECAI'96 Workshop (ATAL), Livro, v. 1193, 401 p., Budapest, Hungary, 1996.

WOOLDRIGDE, M.; Multiagent Systems - A Modern Approach to Distributed Artificial Intelligence. Livro, MIT Press Cambridge, MA, USA, 1999.

Apêndice A – Artigo publicado na *International Conference on Information Science and Applications 2011*

AgentSpeak(PL)

A New Programming Language for BDI Agents with Integrated Bayesian Network Model

Diego Gonçalves Silva
Programa de Pós-Graduação em Computação Aplicada
Universidade do Vale do Rio dos Sinos - UNISINOS
São Leopoldo, Brasil
diego.goncalves.silva@gmail.com

João Carlos Gluz
Programa de Pós-Graduação em Computação Aplicada
Universidade do Vale do Rio dos Sinos - UNISINOS
São Leopoldo, Brasil
jcgluz@unisinos.br

Abstract— **Within the current BDI paradigm of agent oriented programming, it is not possible to develop a practical and straightforward software when agent's beliefs are based on probabilistic knowledge related to their environment, because the logical base of the programming language do not allow this possibility. Usually, to develop this kind of hybrid agent, it is necessary to make use of advanced programming techniques, which combine and integrate different development environments and programming languages in order to represent the logical part and the probabilistic part of the model, to make implementation feasible. This work presents a new agent-oriented programming language called AgentSpeak(PL), based in AgentSpeak(L), a classical BDI programming language. AgentSpeak(PL) integrates the concept of probabilistic beliefs through the use of Bayesian Networks, to core BDI programming concepts. The language is implemented through an extension of the Jason programming environment.**

Keywords: Programming Languages, Agents, Bayesian Networks, Probability, AgentSpeak, Jason

INTRODUCTION

The use of agent-based systems has grown rapidly. There are several advantages with the use of agents to solve problems. In particular, the increasing complexity of current problems makes the use of agents to enter mainstream computing [1].

An agent is a software entity, which works continuously and unattended in a particular environment, often inhabited by other agents and processes [2]. The concept of agent describes its ability to act autonomously in environment where it belongs.

The abstraction of hardware, software and communication details, in a way that allows the direct interaction among agents [3] is a great challenge to be achieved when designing and developing agent-based systems.

A model of agents well known and used is the BDI (Beliefs - Desires – Intentions) model. The main objective this model is to allow the characterization of agents using anthropomorphic notions, such as mental states and actions [1]. These notions and their properties can also be studied and defined by modal logics that support the formal analysis, specification and verification of the characteristics of rational agents [1].

In the BDI model, a belief is defined as a two-state logical proposition: or the agent believes that a certain event is true or believes that it is false. Today, programming languages and tools available for development BDI agents do not work with the concept of probabilistic beliefs [4], i.e. they do not allow agents to understand, infer or represent degrees of belief (or degrees of uncertainty) about a given proposition. A degree of belief is defined by the subjective probability assigned to a particular belief. The concept of Bayesian Networks (BN) [5] fits in this scenario, allowing the modeling of probabilistic beliefs.

As an example of using probability for decision-making in agent oriented programming, it is possible to imagine an environment composed of several agents perform data mining on the Internet in order to make decisions potentially correct about buying and selling shares on the stock exchange. These software agents possess beliefs in their programming with probabilistic events that indicate the degree of certainty certain stock. The agents could be programed to data mine from major on-line journals in a given country. If the frequency of citation of the word "oil" associated with the "discovery" and "money" it is above 80%, this would lead the agents to suggest buying stocks of oil companies, given the circumstances of the events and news surveyed. If the mining would result in the word "oil" associated with "disaster" in a level above 50%, then this fact would lead the agent to suggest the immediate sale of the stocks of those companies.

There are some alternatives to develop an agent able to work in the scenario presented above. The integration between

the current agent programming languages and the concept of belief probabilities can be approached in several ways and at different levels of abstraction.

The more abstract level to address this issue of integration is usually treated by Probabilistic Logics [6]. Although the Probabilistic Logics have effectively ability to represent both logical beliefs about probabilistic beliefs, there are notorious problems related to the tractability of the resulting models [6].

Another possible approach, at a level much closer to the implementation, it is an *ad-hoc* junction of both kind of models in the actual programming code of the agents. Both BDI and BN agent programming environments rely on libraries and development frameworks, with a standard Application Programming Interface (API). Thus, a hybrid agent can be designed and implemented by combining calls from different sets of APIs, each one from distinct programming environments.

In both cases cited, the technical knowledge involved makes development of agents a complex task, often turning medium and large projects unfeasible enterprises. This paper aims to address this programming issue in an intermediate abstraction level. The goal is to define a new agent programming language called AgentSpeak(PL), which is capable to supports BN representation and inference in an integrated model of the beliefs, and plans of agent. AgentSpeak(PL) is based on language AgentSpeak(L) [7], inheriting from it all BDI programming concepts.

There are some works that adopt a similar approach. The work [8] explores the theory for building first order BN extensions, including probabilistic logic programs. According to [8], Bayesian logic programs consist of two parts: the logic part, which consists of a set of Bayesian clauses that capture the qualitative structure of the domain and its base, and the quantitative part that provides the quantitative information about the domain through the use of conditional probability table and rules combination. Each Bayesian logic program specifies a propositional BN that can be handled by an BN inference engines. The structure of BN follows the semantics of logic programming, where the quantitative aspects are handled in the tables of conditional probability rules and combinatorial rules.

According to [9], when an intelligent system interacts with an agent, it often needs to know the beliefs of these agents and their decision-making processes. The central problem in many areas is the prediction, that is, how to find out what the agents will or may do in the future. Since an agent takes its decisions based on their beliefs and their preferences, finding out his mental state is essential to make some predictions. The classic model of epistemic agents, does not allow them to quantify a certain knowledge, or they know it or they do not know. Probabilistic logic beliefs remove this limitation. However, an agent with a base of probabilistic beliefs requires the analysis of all possible states of domain variables. This analysis is computationally exponential. The proposal of [9] is to use probabilistic epistemic inference, to avoid the exponential explosion. The Probabilistic Epistemic Logic (PEL) proposal assumes that the agents have a priority of the probability distribution on the common states of the world, and that the probability distribution of a particular state is equal to the global distribution conditioned on the set of states that the agent considers possible. BN provide a representation a compact complex space of probabilities, allowing simple representations of PEL class models PEL.

For [10], there is a great progress in the representation of the logic of knowledge and in representing the knowledge of probability. There are numerous attempts to develop formalisms that allow represent both kind of knowledge. In this context, the language P-Log is presented. P-Log is a probabilistic logic programming language that combines both types of logic programming: knowledge representation and probability discovery.

These works cover a reasonable portion of the research into programming languages that integrate logic and probabilities. However, compared with AgentSpeak(PL) proposal, it is possible to observe some important differences.

First, the goal of these proposals is focused on aspects of logic programming. AgentSpeak(PL), on the other hand, despite using similar concepts for the treatment of logical probabilistic models, is focused on the unification of the established paradigms of agent programming (the BDI model) with abstractions and concepts derived from probabilistic BN model. The ultimate goal is to achieve a unified BDI-BN programming model that is transparent to the programmer.

Another major difference, and also a big problem to the studied systems, is that none of the studies cited offer a tool for testing software that can simulate the theory shown in order to validating it. This was an important motivating factor for the present work, because, in our opinion, only with an operational implementation of a new language it is really possible to evaluate the value and perspective brought by the language.

The specification of a new programming language usually requires the proper formal foundation. In this work we follow the guidelines of AgentSpeak(L) [1] and define the operational semantics of AgentSpeak(PL), as an extension of AgentSpeak(L) operational semantics.

The formal specification, although important, it is not sufficient, by itself, to allow that software and agents be developed based on it. To do so, it is necessary that programming tools also be created. Thus we developed the JasonBayes agent programming environment. JasonBayes was developed in Java [11] as an extension of the Jason [12] agent development environment.

The following sections present the formal syntax and semantics of AgentSpeak(PL), and an example of use of this language.

AGENTSPEAK(PL) DEFINITION

A. *Syntax*

The language AgentSpeak(PL) is an extension of language AgentSpeak(L) defined in [1]. The abstract syntax is shown in Figure 1. Grammar's symbols P, A, *t*, and *b* are original lexical elements of AgentSpeak(L), representing, respectively, the

predicates, actions, and logical terms, and logical beliefs of this language. The *pb*, *r* and *s* elements are the new lexical items of AgentSpeak(PL), representing respectively, literal probabilistic beliefs, constant numerical values (probabilities) and numerical variables.

The main changes in AgentSpeak(L) grammar were:

- Inclusion of *bn*, the probabilistic belief model of the agent, consisting of the specification of a BN.
- Inclusion of *pbs*, a probability belief estimation base, composed of a list of assignments of probabilities values to probabilistic beliefs.
- Inclusion of *+pat* and *–pat* events/triggers based on atomic probabilistic beliefs.
- Inclusion of *!pat* achievement goal, and *?pat* test also based on atomic probabilistic beliefs.
- Inclusion of *+pbeq* and *-pbeq* actions, able to update the probability belief estimation base.

$$
\begin{array}{l}
ag ::= bs\ \ bn\ \ pbs\ \ ps \\
bs ::= b_1 \ldots b_n \ \ (n \geq 0) \\
bn ::= bneq_1, \ldots, bneq_n \ \ (n \geq 0) \\
bneq ::= \%pb = r \mid \%pb : pb_1 \wedge \ldots \wedge pb_n = r \ \ (n > 0) \\
pbs ::= pbeq_1 \ldots pbeq_n \ \ (n \geq 0) \\
pbeq ::= \%pb = r \\
ps ::= p_1 \ldots p_n \ \ (n \geq 0) \\
p ::= te : ct \leftarrow h \\
te ::= +at \mid -at \mid +pat \mid -pat \mid +g \mid -g \\
ct ::= ct_1 \mid \mathcal{T} \\
ct_1 ::= at \mid \neg at \mid pat \mid \neg pat \mid ct_1 \wedge ct_1 \\
h ::= h_1 ; \mathcal{T} \mid \mathcal{T} \\
h_1 ::= a \mid g \mid u \mid h_1 ; h_1 \\
at ::= \mathrm{P}(t_1, \ldots, t_n) \ \ (n \geq 0) \ \mid \mathrm{P}(t_1, \ldots, t_n)[s_1, \ldots, s_m] \ \ (n \geq 0, m > 0) \\
pat ::= pat_1 \mid pat_1[s_1, \ldots, s_m] \ \ (m > 0) \\
pat_1 ::= \%\mathrm{P}(t) = x \mid \%\mathrm{P}(t) = r \mid \%\mathrm{P}(t) \neq r \mid \\
\qquad \%\mathrm{P}(t) < r \mid \%\mathrm{P}(t) \leq r \mid \%\mathrm{P}(t) > r \mid \%\mathrm{P}(t) \geq r \\
s ::= percept \mid self \mid id \\
a ::= \mathrm{A}(t_1, \ldots, t_n) \ \ (n \geq 0) \\
g ::= !at \mid ?at \mid !pat \mid ?pat \\
u ::= +b \mid -at \mid +pbeq \mid -pat
\end{array}
$$

Abstract Syntax of AgentSpeak(PL).

With these modifications the agent can add to its beliefs a *bn* probabilistic model, which is represented by a BN. The BN is defined by a set of probabilistic equations $bneq_1$, ..., $bneq_n$ specifying the prior probability tables for the variables (network nodes) without parents (*%pb = r*), or conditional probability for the variables that have parents (*%pb : %pb₁ ∧ .. %pbₙ = r*).

In its most elementary form, a probabilistic belief must correspond to an assignment of some probability value for a possible state of a BN variable. In AgentSpeak(PL) all BN variables are represented by unary predicates P(*t*). The name of the predicate defines the variable name, while the term *t* identifies the possible states that the variable can assume. Note that these states are mutually exclusive by definition, i.e. it is not possible to assume two states at same time. Thus an equation of the form %P(*t*) = *r*, where *t* is a literal term and *r* is a real constant, is sufficient to define a probabilistic belief. In addition to simple equations in AgentSpeak(PL) is also possible to use expressions like %P(*t*) < *r* or %P(*t*) > *r*, to test the probability value assigned to %P(*t*).

Besides the BN model (which is fixed) the agent's belief base also includes *pbs*, a dynamic probability estimation base, which contains the current probability assignments for the beliefs in the BN model. The *pbs* base is formed by a list of probability assignments $\%P_1(t_{11}) = r_{11}$, $\%P_1(t_{12}) = r_{12}$, ..., $\%P_n(t_{n1}) = r_{n1}$,..., $\%P_n(t_{nm}) = r_{nm}$, where all terms t_{ij} must be literal terms, corresponding to possible states of each variable P_i. Thus the *pbs* base assigns a probability to each possible state of each BN variable.

The *pbs* base is updated when new evidence is aggregate to agent's beliefs base. A new evidence can be a non-probabilistic literal belief +P(*t*) formed by a unary predicate that is exactly the same format (which can be unified) that a BN variable. In this case, if the non-probabilistic belief is being added to the base, the *pbs* list is updated so that the probability assigned to P(*t*) is 1 (%P(*t*) = 1), and that the probability assigned to other states of variable P is 0. Another possibility is to update the *pbs* base directly through actions *+pbeq* and *-pat*. If adding an estimate (*+pbeq*) then it should be provided the complete sequence: $+\%P(t_1) = r_1$; $+\%P(t_2) = r_2$; ...; $+\%P(t_n) = r_n$, of probability assignments for all possible states t_1, t_2, ..., t_n of the variable P. The exclusion of evidences, both in the case of probabilistic or non-probabilistic beliefs, is executed in two steps: first is eliminated from current *pbs* all assignments of probabilities for the excluded beliefs, then this assignments are replaced by the prior probabilities that can be inferred directly from the Bayesian Network.

In terms of programming of agent's plans, the probabilistic beliefs can be used both as trigger events associated with the plans, or in the context conditions of the plans. In both cases it is possible to use expressions like %P(*t*) < *r* or %P(*t*) > *r*, to test the probability value assigned to %P(*t*). It is also possible to use logical variables in terms that represent the states of the BN variables, or numeric variables in expressions like %P(*t*) = *x*, to recover the value of the probability assigned to %P(*t*).

Achievement goals *!pat* can be used in the plans of agents, having exactly the same semantics of AgentSpeak(L). The objectives test *?pat*, in turn, are tested exactly as used in probabilistic beliefs context conditions of the plans.

B. Semantics

AgentSpeak(L) language was designed for programming BDI agents in form of reactive planning systems. It was first presented in [7], and is a natural extension of logic programming for BDI agent architecture, which represents an abstract model for agent programming and has been the predominant approach in the implementation of intelligent agents [13]. An AgentSpeak(L) agent corresponds to the specification of a set of beliefs that form the basis of initial

beliefs and a set of plans [11]. AgentSpeak(L) distinguishes two types of goals: achievement goals and test goals. Both, achievement goals and test goals are predicates, such as beliefs. They express that the agent wants to reach a state in the environment where the goal is associated with the predicate true. A test goal returns a unification of the predicate test with a belief agent, or fails if it is not possible unification with any belief.

```
+concert(A,V) : likes(A)
   ← !book_tickets(A,V).

+!book_tickets(A,V) : ¬busy(phone)
   ← call(V);
      ...;
      !choose_seats(A,V).
```

Examples of AgentSpeak(L) plans (source [11]).

Figure 2 shows examples of AgentSpeak(L) plans. The first plan specifies that the announcement of a concert to be held in place V by the artist A, corresponding to addition of a belief *concert(A,V)* as a consequence of the perception of the change in the environment. If the agent likes the artist, then it will have the goal to book the tickets for this concert. The second plan specifies that, to adopt the goal of reserving tickets when the phone line is not busy, then the agent should execute the following plan: first performs the basic action to make telephone contact with the concert venue V, then follows a certain protocol to reserve tickets, and ends with the implementation a sub-plan for the choice of seats.

The formal semantics of AgentSpeak(L) was defined by a rules-based operational semantics, which specified a transition relation between configurations. In this paper we used the same technique defining an extension of the operational semantics able to deal with new constructions of AgentSpeak(PL).

In this context, the operational semantics is just a set of rules that define what transitions may occur between configurations of a given agent. The configuration of an agent is a structure that contains all information relevant to the operation of the agent. Formally, this configuration is a quintuple shown in Figure 3.

$$\langle ag\ , C\ , M\ , T\ , s \rangle$$

Basic setup of an agent in AgentSpeak(PL).

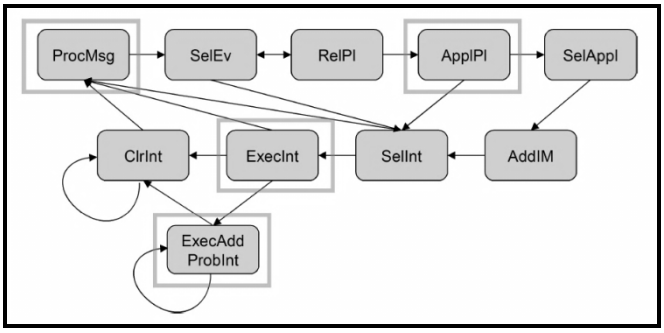
The *ag* component is a program in AgentSpeak(PL) composed of all elements provided in the grammar of AgentSpeak(PL). Thus, *ag* is divided into four subcomponents: ag_{bs}, ag_{bn}, ag_{pbs} and ag_{ps} containing, respectively, the non-probabilistic beliefs of the agent, its BN model, the probabilistic beliefs derived from the BN (or used as evidences) and the set of plans of the agent. The subcomponent ag_{pbs} contains estimates of probability for each probabilistic belief defined in the BN model, always kept updated according to the network structure and the evidence available so far.

The component *C* indicates the circumstance of the agent, having the structure $\langle I, E, A \rangle$, where *I = {i, i , . . . }* defines the agent's set of intentions, *E = {(te, i), (te, i), . . . }* defines the agent's set of events (*te* is the trigger event associated with *i* intention) and *A* is the agent's set of actions.

The *M* component, defined as $\langle In, Out, SI \rangle$, stores the information of agent communication processing, where *In* contains the incoming messages, *Out* the outgoing messages and *SI* stores the intentions that were suspended due to processing of any message.

The component *T*, with structure $\langle R, Ap, \iota, \varepsilon, \rho \rangle$, contains the temporary information of agent: *R* is a set that stores the relevant plans of the event being processed, *Ap* is the set of applicable plans for this event, while the other components store, respectively, the intent, the event and the plan are being processed.

Finally, the component *s* indicates the current step in the cycle of the agent's reasoning. The reasoning cycle consists of a series of execution steps that begin to be executed when the agent is initialized and continue to be executed until the agent stops its reasoning. Figure 4 shows the possible steps within the reasoning cycle, and what are the possible transitions from one step to another.



Reasoning Cycle of AgentSpeak(PL).

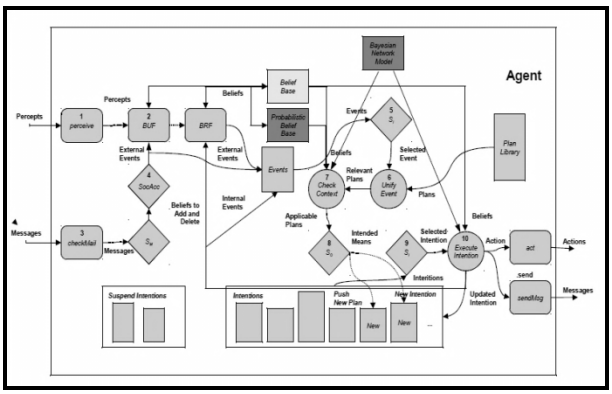
The steps marked in red have been changed in relation to operational semantics of AgentSpeak(L). Step **ExecAddProbInt**, marked in blue, is new, and it is necessary to specify the semantics of probabilistic atomic beliefs inclusion. Below is presented a description of the functions performed at each step:

- **ProcMsg**: process messages directed to the agent;
- **Selev**: select an event from the set of events;
- **RelPl**: return all relevant plans;
- **ApplPl**: check what plans apply;
- **SelAppl**: select a particular plan;
- **AddIM**: add a new intention to the set of intentions;
- **SelInt**: select an intention;
- **ExecInt**: execute the selected intention;

- **ClrInt**: clear intentions completed in the previous step.
- **ExecAddProbInt**: ensure that the addition of a probabilistic belief be done atomically.

It was necessary to include the new step **ExecAddProbInt** because the update of the probabilities of the BN variables could be formed by several assignments, one for each state of the variable. However, the base *pbs* only assumes a consistent state after the probability assignments of all states of a given variable were executed. This extra step ensures that the additions to be made atomically.

The reasoning process of the agent is defined by a series of configuration transition rules, which define how the agent can move from one step of the cycle to another. These transition rules depend on a number of additional functions, capable of performing operations such as buffering of beliefs (function), the process of belief revision (function *BRF*), the event selection function (S_E), the plan selection function (S_O) and the intention selection function (S_I). The Figure 5 shows the overall architecture of the reasoning cycle, identifying how the configuration components are related to the transition functions. The diagram already shows the main modifications due to AgentSpeak(PL). For details on the architecture defined in Figure 5, see [1].



Execution cycle of an agent in AgentSpeak(PL).

AgentSpeak(PL) is an extension of AgentSpeak(L), thus were kept the key components and elements of the architecture of AgentSpeak(L). The new bases are additional items to the existing architecture. The base of not-probabilistic beliefs continues to be defined and used as in AgentSpeak(L), independent of other bases added for AgentSpeak(PL).

The main elements added to support the features of AgentSpeak(PL) were the new probabilistic knowledge bases specified by the grammar of the language. The knowledge bases *Bayesian Network Model* and *Probabilistic Belief Base*, represent, respectively the *bn* Bayesian model and the *pbs* probability estimation base defined by AgentSpeak(PL) grammar. Also are shown in the diagram the dependencies that exist between these new bases and other components of the architecture.

The new (or modified) rules of the operational semantics require some new definitions and functions to be properly specified. In particular, it is necessary to define the logical consequence relation between the *pbs* probability estimation base, and pat_1 probabilistic atomic formulas defined in the grammar of AgentSpeak(PL).

This definition is similar logical consequence relation between a non-probabilistic atomic formula of AgentSpeak(L) and the *bn* belief base [1]:

Definition 1. A non-probabilistic atomic formula at_1, with annotations $[s_{11}, s_{12}, ..., s_{1n}]$ is a logical consequence of a set *bs* of non-probabilistic literal formulas, denoted by $bs \models at_1[s_{11}, s_{12}, ..., s_{1n}]$, if and only if there is a formula $at_2\ [s_{21}, s_{22}, ..., s_{2m}] \in bs$ such that (i) $at_1\theta = at_2\theta$, for a most general unifier θ and (ii) $\{s_{11}, s_{12}, ..., s_{1n}\} \subseteq \{s_{21}, s_{22}, ..., s_{2m}\}$.

The relation of logical consequence for probability beliefs is defined as follows:

Definition 2. A probabilistic atomic formula pat_1, with annotations $[s_{11}, s_{12}, ..., s_{1n}]$ is a logical consequence of a set *pbs* of probability estimations for beliefs, denoted by $pbs \models pat_1[s_{11}, s_{12}, ..., s_{1n}]$, if and only if, for the probabilistic term $\%p(t) \in pat_1$ exist an equation $\%pb_k = r_k\,[s_{21}, s_{22}, ..., s_{2m}] \in pbs$ such that there is a most general unifier θ, which satisfies the following conditions:

1) $\%p(t)\theta = \%pb\theta$

2) $\{s_{11}, s_{12}, ..., s_{1n}\} \subseteq \{s_{21}, s_{22}, ..., s_{2m}\}$

3) if pat_1 has the form $\%p(t) = x$, then $r_k = x\theta$

4) if pat_1 has the form $\%p(t) = r$, $\%p(t) \neq r$, $\%p(t) < r$, $\%p(t) \leq r$, $\%p(t) > r$, *ou* $\%p(t) \geq r$ then, respectively, $rk = r$, $rk \neq r$, $rk < r$, $rk \leq r$, $rk > r$, or $rk \geq r$.

In addition to this definition the new transition rules make use of *RecalcBN* function, which makes the recalculation of the probability distribution of the BN. In practical terms, is this function that incorporates BN inference engine.

The transition rules that form each of reasoning cycle steps are presented in the following items.

5) Step **ApplPl**

All rules from this step were changed in the semantics of AgentSpeak(PL):

Appl1':

$$\frac{\mathsf{AppPlan}(ag_{bs}, T_R) \cup \mathsf{AppPlan}(ag_{pbs}, T_R) \neq \{\}}{\langle ag, C, M, T, \mathsf{ApplPl}\rangle \rightarrow \langle ag, C, M, T', \mathsf{SelAppl}\rangle}$$

$$where:\ T'_{Ap} = \mathsf{AppPlan}(ag_{bs}, T_R) \cup \mathsf{AppPlan}(ag_{pbs}, T_R)$$

This rule identifies which plans are applicable for a given event, testing whether the plan context condition is true. The rule was modified to allow probabilistic belief to be tested in the context condition.

Appl2':

$$\frac{\mathsf{AppPlan}(ag_{bs}, T_R) \cup \mathsf{AppPlan}(ag_{pbs}, T_R) = \{\}}{\langle ag, C, M, T, \mathsf{ApplPl}\rangle \rightarrow \langle ag, C, M, T, \mathsf{SelInt}\rangle}$$

This rule handles the case when there are no applicable plans. It was modified for the same reason of **Appl1'**.

6) Step **ExecInt**

In the case of step **ExecInt**, the following rules do not needed to be changed by the semantics of AgentSpeak(PL):

- **Action**: performs an explicit action in the body plan of the agent;
- **AchvGl**: registers a new goal of achieving internal to the agent;
- **TestGl1**: tests if a test goal has been achieved;
- **TestGl2**: generates an internal event to test the test goal;
- **ClrInt1**, **ClrInt2**, **ClrInt3**: finish the execution of an intention;

The rules **AddBel'** and **DelBel'** where changed by the semantics of AgentSpeak(PL):

AddBel':

$$\frac{T_i = i[head \leftarrow +b; h]}{\langle ag, C, M, T, \mathsf{ExecInt}\rangle \rightarrow \langle ag', C', M, T', \mathsf{ClrInt}\rangle}$$

$$\begin{aligned} where:\ ag'_{bs} &= ag_{bs} + b[self] \\ ag'_{pbs} &= \mathsf{RecalcBN}(ag'_{bs}, ag_{bn}, ag_{pbs}) \\ C'_E &= C_E \cup \{\langle +b[self], \mathsf{T}\rangle\} \\ T'_i &= i[head \rightarrow h] \\ C'_I &= (C_I \setminus \{T_i\}) \cup \{T_i'\} \end{aligned}$$

The rule **AddBel'** adds a new non-probabilistic belief to the agent belief base. It was changed because the non-probabilistic belief may be related to a BN variable.

DelBel':

$$\frac{T_i = i[head \leftarrow -b; h]}{\langle ag, C, M, T, \mathsf{ExecInt}\rangle \rightarrow \langle ag', C', M, T', \mathsf{ClrInt}\rangle}$$

$$\begin{aligned} where:\ ag'_{bs} &= ag_{bs} - b[self] \\ ag'_{pbs} &= \mathsf{RecalcBN}(ag'_{bs}, ag_{bn}, ag_{pbs}) \\ C'_E &= C_E \cup \{\langle -b[self], \mathsf{T}\rangle\} \\ T'_i &= i[head \rightarrow h] \\ C'_I &= (C_I \setminus \{T_i\}) \cup \{T_i'\} \end{aligned}$$

This rule excludes a new non-probabilistic belief from the agent belief bases, and was changed for the same reason **AddBel'**.

Some new rules were necessary to specify the operational semantics of step **ExecInt** in AgentSpeak(PL):

TestProbGl1:

$$\frac{T_i = i[head \leftarrow ?pat; h] \quad \mathsf{Test}(ag_{pbs}, pat) \neq \{\}}{\langle ag, C, M, T, \mathsf{ExecInt}\rangle \rightarrow \langle ag, C', M, T', \mathsf{ClrInt}\rangle}$$

$$\begin{aligned} where:\ T'_i &= i[(head \rightarrow h)\theta] \\ & \theta \in \mathsf{Test}(ag_{pbs}, pat) \\ C'_I &= (C_I \setminus \{T_i\}) \cup \{T'_i\} \end{aligned}$$

This rule is equivalent to **TestGl1**, just for test goals formed by probabilistic beliefs.

TestProbGl2:

$$\frac{T_i = i[head \leftarrow ?pat; h] \quad \mathsf{Test}(ag_{pbs}, pat) = \{\}}{\langle ag, C, M, T, \mathsf{ExecInt}\rangle \rightarrow \langle ag, C', M, T, \mathsf{ClrInt}\rangle}$$

$$\begin{aligned} where:\ C'_E &= C_E \cup \{\langle +?pat, T_i\rangle\} \\ C'_I &= C_I \setminus \{T_i\} \end{aligned}$$

Same function of **TestGl2**, but applied the test of probabilistic beliefs.

AddProbBel1:

$$\frac{T_i = i[head \leftarrow +pbeq; h]}{\langle ag, C, M, T, \mathsf{ExecInt}\rangle \rightarrow \langle ag', C', M, T', \mathsf{ExecAddProbInt}\rangle}$$

$$\begin{aligned} where:\ ag'_{pbs} &= ag_{pbs} + pbeq[self] \\ C'_E &= C_E \cup \{\langle +pbeq[self], \mathsf{T}\rangle\} \\ T'_i &= i[head \rightarrow h] \\ C'_I &= (C_I \setminus \{T_i\}) \cup \{T_i'\} \end{aligned}$$

Rule that starts the process of adding an atomic probabilistic belief to the *pbs* base.

DelProbBel:

$$\frac{T_i = i[head \leftarrow -pbeq; h]}{\langle ag, C, M, T, \mathsf{ExecInt}\rangle \rightarrow \langle ag', C', M, T', \mathsf{ClrInt}\rangle}$$

$$\begin{aligned} where:\ ag'_{pbs} &= \mathsf{RecalcBN}(ag_{bs}, ag_{bn}, ag_{pbs} - pat[self]) \\ C'_E &= C_E \cup \{\langle -pat[self], \mathsf{T}\rangle\} \\ T'_i &= \{i[head \rightarrow h]\} \\ C'_I &= (C_I \setminus \{T_i\}) \cup \{T_i'\} \end{aligned}$$

Rule that excludes a probabilistic belief from *pbs* base.

7) Step **ExecAddProbInt**

This new step in the operational semantics of AgentSpeak(PL) required the following new transition rules:

AddProbBel2:

$$\frac{T_i = i[head \leftarrow +pbeq; h]}{\langle ag, C, M, T, \mathsf{ExecAddProbInt}\rangle \rightarrow \langle ag', C', M, T', \mathsf{ExecAddProbInt}\rangle}$$

$$\begin{aligned} where:\ ag'_{pbs} &= ag_{pbs} + pbeq[self] \\ C'_E &= C_E \cup \{\langle +pbeq[self], \mathsf{T}\rangle\} \\ T'_i &= i[head \rightarrow h] \\ C'_I &= (C_I \setminus \{T_i\}) \cup \{T_i'\} \end{aligned}$$

This rule implements the addition of one more probability assignment equation into the *pbs* base.

AddProbBel3:

$$\frac{T_i \neq i[head \leftarrow +pbeq; h]}{\langle ag, C, M, T, \mathsf{ExecAddProbInt}\rangle \rightarrow \langle ag', C, M, T, \mathsf{ClrInt}\rangle}$$

$$where:\ ag'_{pbs} = \mathsf{RecalcBN}(ag_{bs}, ag_{bn}, ag_{pbs})$$

The rule **AddProbBel3** finalizes the process of adding the equations that define the probabilistic belief, updating the *pbs* base through the BN recalculation process.

EXAMPLE OF USE

After the formal definition of the new language, the next step was to modify JASON, the AgentSpeak(L) programming environment, to generate a new programming environment able to support AgentSpeak(PL). This new environment is being preliminarily called *JasonBayes*. All changes in Jason were made according to the operational semantics of the new language in order to support the new functionality of AgentSpeak(PL), without losing the full compatibility with AgentSpeak(L). As an example of validation of this programming tool, Figure 6 shows the code of a software agent with probabilistic beliefs.

```
// Probabilistc Beliefs

// Standard Syntax for probabilistic beliefs
%visitAsia(visit)                                 = 0.01 .
%visitAsia(no_visit)                              = 0.99 .
%smoking(smoker)                                  = 0.5 .
%smoking(nonsmoker)                               = 0.5 .
%stressed(yes)                                    = 0.8 .
%stressed(no)                                     = 0.2 .
%tuberculosis(present) | visitAsia(visit)         = 0.05 .
%tuberculosis(present) | visitAsia(no_visit)      = 0.01 .
%tuberculosis(absent) | visitAsia(visit)          = 0.95 .
%tuberculosis(absent) | visitAsia(no_visit)       = 0.99 .

// Compact Notation (partial)
%cancer(present) | smoking                        = [0.1, 0.01].
%cancer(absent) | smoking                         = [0.9, 0.99].
%bronchitis(present) | smoking                    = [0.6, 0.3].
%bronchitis(absent) | smoking                     = [0.4, 0.7].
%tborca(yes) | tuberculosis & cancer              = [1.0, 1.0, 1.0, 0.0].
%tborca(no) | tuberculosis & cancer               = [0.0, 0.0, 0.0, 1.0].

// Compact Notation (full)
%xray(abnormal, normal) | tborca                  = [0.98, 0.05, 0.02, 0.95].
%dyspnea(present, absent) | tborca & bronchitis = [0.9, 0.7, 0.8, 0.1, 0.1,
0.3, 0.2, 0.9].

// Initial Belief
plan1.

// Plans
+plan1: true <-
            math.randon(PVAsia);
            math.randon(PSmoke);
            +%visitAsia(visit) = PVAsia;
            +%visitAsia(no_visit) = 1 - PVAsia;
            +%smoking(smoker) = PSmoke;
            +%smoking(nonsmoker) = 1 - PSmoke.

// Plans activated by probabilistic beliefs
+%cancer(present) > 0.9: true      <- .println ("Cancer present!").
+%bronchitis(present) > 0.9: true  <- .println ("Bronchitis present!").
+%tuberculosis(present) > 0.9: true <- .println ("Tuberculosis present!").
```

Example of an agent in AgentSpeak(PL).

In Figure 6, is shown the AgentSpeak(PL) code for a BN model that implements a classical Bayesian diagnostic example about how to diagnose cancer, tuberculosis and bronchitis diseases, based on evidences that the patient had traveled to Asia, smokes or is stressed [14].

The JasonBayes implementation of AgentSpeak(PL) supports a *compact notation*, which allows to define a BN model with a lot fewer source code than the standard syntax defined in Figure 1, but can always be formally expanded to this standard syntax. In the example, is initially used the standard syntax to define the probability distribution of the variables *visitAsia*, *smoking*, *stressed* and *tuberculosis*. After, it is used the partial compact notation to define variables *cancer*, *bronchitis* and *tborca*. Finally, the full compact notation is used to define the other variables.

After the probabilistic model, it is defined *plan1*, which is the sole non-probabilistic belief of the agent. This non-probabilistic belief enables initial plan of the agent, which generates random probabilities, which will be assigned *visitAsia* and *smoking* variables. Note that the function *math.random* will generate a random number between 0 and 1 and the events sum must be 100%. Because values are generated randomly, at some point the probability of the event present in variable *cancer* will be greater than 0.9, or 90%. At that moment, the plan associated with this variable will present a message stating that cancer is present. The same condition can be tested for *bronchitis* and *tuberculosis* variables.

This is a simple example that aims to show some practical aspects of programming in AgentSpeak(PL). Naturally, the complexity of the example can be increased as well with its applicability domain. For instance, in the case of multiagent systems it is possible to change the probability estimations based on communication of evidences that occurred between agents.

8) CONCLUSIONS

The main objective of this work was to create a new agent programming language called AgentSpeak(PL) and to provide a programming tool that could be used for software development purposes, in order to cover gaps detected in agent oriented programming.

The creation of a new agent programming language, powerful enough to support the design and building of agents with probabilistic beliefs models fully integrated with BDI concepts, surely can enable the development and test of new hybrid applications for complex scenarios and multidisciplinary domains. This, in turn, will facilitate studies and projections on the different environments and areas, allowing even more multidisciplinary projects.

With this work will be possible to use probabilistic knowledge within agents' belief bases, strengthening the concept of agent oriented programming and linking technologies that before evolved individually.

9) REFERENCES

[1] Bordini, R. H., Wooldridge, M., Hübner, J. F., “Programming Multi-Agent Systems in AgentSpeak using Jason”, 2007.

[2] Shoham, Y., “Agent-oriented programming. Artificial Intelligence”, 1992.

[3] Russell, S., Norvig, P., “Artficial Intelligence: A Modern Approach”, 1995.

[4] Garcia, A. F., de Lucena, C. J. P., Zambonelli, F., Omicini, A., Castro, J., “Software Engineering for Large-Scale Multi-Agent Systems, Research Issues and Practical Applications”, 2003.

[5] Chan, H., Darwiche, A., “On the revision of probabilistic beliefs using uncertain evidence”, 2005.

[6] Korb, K. B., A. E. Nicholson, A. E., “Bayesian Artificial Intelligence”, 2003.

[7] Anand S. R., “AgentSpeak(L): BDI agents speak out in a logical computable language”, 1996.

[8] Kerting, K., Raedt, L. D., “Bayesian Logic Programs”, 2000.

[9] Milch, B., Koller, D., “Probabilistic Models for Agents' Beliefs and Decisions”, 1999.

[10] Baral, C., Hunsaker, M., “Using the Probabilistic Logic Programming Language P-log for Causal and Counterfactual Reasoning and Non-naive Conditioning”, 2007.

[11] Bordini, R. H., Vieira, R., “Linguagens de Programação Orientadas a Agentes: uma introdução baseada em AgentSpeak(L)”, 2003.

[11] http://jason.sourceforge.net/Jason/Jason.html em 11/2010.

[12] Wooldridge, M., “Reasoning about Rational Agents”, 2000.

[13] Lauritzen S, Spiegelhalter D., “Local Computation with Probabilities on Graphical Structures and their Application to Expert Systems”, 1988.