5
Implementando ANA

Este Capítulo apresenta como agentes projetados a partir da arquitetura descrita na Seção anterior podem ser implementados.

Para tanto, na Seção 5.1 apresentaremos uma extensão da linguagem AgentSpeak (chamada AgentSpeak(L) Normativo) e na Seção 5.2 apresentaremos uma extensão do interpretador Jason (chamado Jason Normativo).

5.1
AgentSpeak(L) Normativo

$$
\begin{aligned}
&agent ::= beliefs\ \underline{norms}\ plans\\
&beliefs ::= (literal\,'.')*\\
&\underline{norms ::= (norm)*}\\
&\underline{norm ::=< ATOM >'\ ('addressees','}\\
&\qquad\qquad\qquad\qquad\underline{activation','}\\
&\qquad\qquad\qquad\qquad\underline{deactivation','}\\
&\qquad\qquad\qquad\qquad\underline{deontic\_concept','}\\
&\qquad\qquad\qquad\qquad\underline{behavior','}\\
&\qquad\qquad\qquad\qquad\underline{rewards','}\\
&\qquad\qquad\qquad\qquad\underline{punishments)}\\
&\underline{addressees ::= names \mid roles \mid groups}\\
&\underline{names ::='\ ['[name(',' name)*]']'}\\
&name ::=< STRING >\\
&\underline{roles ::='\ ['[role(',' role)*]']'}\\
&\underline{role ::='\ role('< STRING >')'}\\
&\underline{groups ::='\ ['[group(',' group)*]']'}\\
&\underline{group ::='\ group('< STRING >')'}\\
&\underline{activation ::= context}\\
&\underline{deactivation ::= context}
\end{aligned}
$$

PUC-Rio - Certificação Digital Nº 1012680/CA

$context ::= literal$
$\quad | \underline{normative\_condition}$
$\quad | context\,'\&'\,context$
$\quad | context\,'\,|'\,context$
$\underline{normative\_condition ::='\,nc('normative\_state','\,norm')'}$
$\underline{normative\_state ::='\,FULFILLED'\,|'\,VIOLATED'}$
$\underline{\quad |'\,ACTIVATED'\,|'\,DEACTIVATED'}$
$\underline{deontic\_concept ::= OBLIGATION \mid PROHIBITION}$
$\underline{behavior ::='!'\,literal \mid action}$
$\underline{rewards ::= sanctions}$
$\underline{punishments ::= sanctions}$
$\underline{sancitons ::='\,['[sanction(','\,sanction)*]']'}$
$\underline{sanction ::= behavior \mid norm}$
$plans ::= (plan)+$
$plan ::= trigger\_event\,'\,:'\,\underline{context}\,'<-'\,body'.'$

Acima apresentamos o AgentSpeak(L) Normativo que é uma versão estendida do AgentSpeak(L), descrito na Seção 2.3 (as partes sublinhadas representam as extensões realizadas no AgentSpeak(L)).

Assim como o AgentSpeak(L), o AgentSpeak(L) Normativo possui um conjunto de crenças (representado por *beliefs*) e planos (representado por *plans*). Adicionalmente, um agente desenvolvido a partir do AgentSpeak(L) Normativo é responsável por um conjunto de normas (representado por *norms*). Agora descreveremos os principais tipos e extensões adicionadas ao AgentSpeak(L):

(*norms*) Como descrito anteriormente, um agente AgentSpeak(L) Normativo é responsável por um conjunto de normas (*norms*), onde cada norma do tipo *norm* é composta pelos seguintes componentes (a definição destes componentes é baseada na estrutura de normas apresentada na Seção 4.1.2):

- **(*identifier*)** um identificador da norma, representado pelo tipo $< ATOM >$ (ver o tipo $< ATOM >$ na Seção 2.3) ;
- **(*addressees*)** indica os responsáveis por cumprir a norma, tal componente pode ser uma lista de agentes, representada por *names* (onde cada nome do tipo *name* é uma *string* do tipo $< STRING >$ como definido na Seção 2.3), uma lista de papéis desempenhados pelos

PUC-Rio - Certificação Digital Nº 1012680/CA

agentes, representada por *roles* (onde cada papel do tipo *role* possui um identificador "role"e uma *string*), e uma lista de grupos de agentes, representada por *groups* (onde cada grupo do tipo *group* possui um identificador "group"e uma *string*);

(activation e deactivation) um context de ativação do tipo *activation* e desativação do tipo *deactivation*, ambas condições são definidas a partir de um contexto do tipo *context*. Além das condições já permitidas no contexto do AgentSpeak(L), no AgentSpeak(L) Normativo é possível definir explicitamente condições normativas, ou seja, condições que devem ser satisfeitas a partir do estado atual das normas do agente. Uma condição normativa é representada pelo tipo *normative_condition* que possui um identificador (nomeado "*nc*"), é composto pelos componentes *norm* e *normative_state*, este último podendo ser um tipo *ACTIVATED*(a norma foi ativada), *DEACTIVATED*(a norma foi desativada), *FULFILLED*(a norma foi cumprida) ou *VIOLATED*(a norma foi violada), como formalizado na Seção 4.1.2. Desta forma, as seguintes condições normativas podem ser definidas: *nc(ACTIVATED, norm)*: Verifica se a norma do tipo *norm* foi ativada;*nc(DEACTIVATED, norm)*: Verifica se a norma foi desativada; *nc(FULFILLED, norm)*: Verifica se a norma foi cumprida; e, *nc(VIOLATED, norm)*: Verifica se a norma foi violada;

(*deontic_concept*) um conceito deôntico do tipo *deontic_concept*, que pode ser uma obrigação do tipo *OBLIGATION* ou uma proibição do tipo *PROHIBITION*;

(*behavior*) um comportamento do tipo *behavior* regulado pela norma que pode esta relacionado ao atingimento de um objetivo do tipo *achievement* ou a execução de uma ação do tipo *action* (ver Seção 2.3);

(*rewards e punishments*) uma lista de recompensas do tipo *rewards* e punições do tipo *punishments*, ambas são uma lista de sanções do tipo *sanctions*, onde cada sanção do tipo *sanction* pode ser um comportamento ou uma norma.

(*plans*) Assim como o AgentSpeak(L), um plano (representada por *plan*) no AgentSpeak(L) Normativo é composto por uma condição de invocação (representado por *trigger_event*), um contexto que define as condições que devem ser satisfeitas para que o plano possa ser executado (representado por *context*) e uma sequência de objetivos para serem atingidos

PUC-Rio - Certificação Digital Nº 1012680/CA

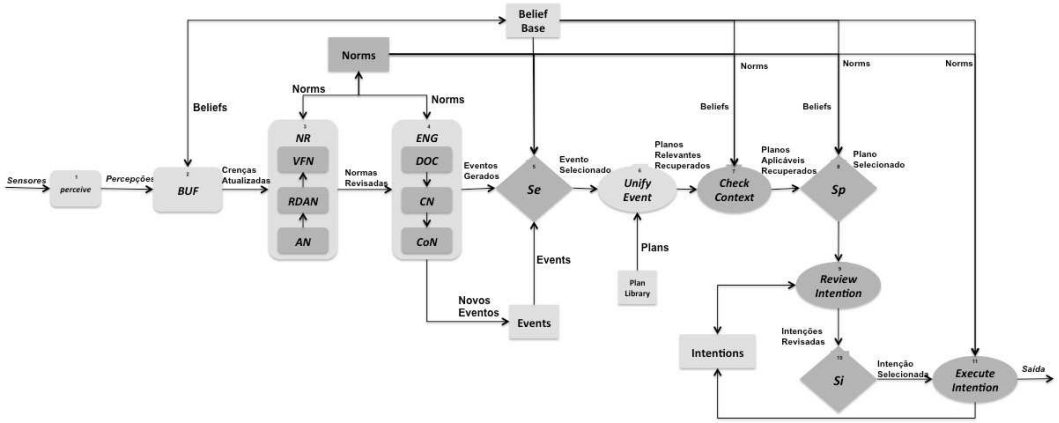

Figura 5.1: Jason Normativo

e ações para serem executadas (representado por *body*). A principal diferença entre planos no AgentSpeak(L) e no AgentSpeak(L) Normativo esta no contexto do plano, dado que no AgentSpeak(L) Normativo é possível definir explicitamente condições normativas, como as descritas anteriormente.

5.2 Interpretador Jason Normativo

Figura 5.1 apresenta o interpretador Jason Normativo que é uma extensão do interpretador Jason descrito na Seção 2.4 (as partes escurecidas representam as extensões realizadas no Jason). A versão normativa do Jason apresentada na Figura 5.1 fornece um conjunto de funcionalidades normativas que possibilitam interpretar um agent AgentSpeak(L) Normativo e lidar com as questões normativas.

Um agente Jason Normativo funciona como segue: ele percebe informações do ambiente através de seus sensores, suas crenças são atualizadas utilizando a função *Belief Update*(BUF), a função *Normative Reviewer*(NR) é responsável por realizar as seguintes tarefas: (*Adopting Norms*-AN-) verifica se novas normas percebidas são endereçadas ao agente e adiciona tais normas ao conjunto de normas adotadas. Esta tarefa implementa a formalização apresentada no esquema *AdoptingNorms* descrito na Seção 4.3.1; (*Reviweing (De)Activated Norms*-RDAN-) atualiza o conjunto de normas ativadas e desativadas, considerando que algumas se tornam ativas e outros inativas devido às percepções, ativação, desativação, cumprimento, ou violação de outras normas. Esta tarefa implementa a formalização apresentada no esquema *ReviewDeActivatedNorms* descrito na Seção 4.3.1; e (*Verifying Fulfillment Norms*-VFN-) verifica o cumprimento e violação das normas, considerando que algumas po-

PUC-Rio - Certificação Digital Nº 1012680/CA

dem ter sido cumpridas e outras violadas devido ao funcionamento do agente, esta tarefa implementa a formalização apresentada no esquema *VerifyFulfillmentNorms* descrito na Seção 4.3.1.

Após rever as normas, a função *Events Normative Generator*(ENG) é responsável por: (*Detect and Overcome Conflicts*-DOC) checar e superar os conflitos entre normas. Esta tarefa implementa a formalização apresentada no esquema *DetectOvercomeConflicts* descrito na Seção 4.3.2;(*Choosing Norms*-CN-) selecionar as normas não conflitantes que o agente tem interesse em cumprir ou violar. Esta tarefa implementa a formalização apresentada no esquema *ChoosingNorms* descrito na Seção 4.3.2; e (*Compliance with Norms*-CoN-) adicionar novos eventos a base de eventos do agente levando em consideração as decisões normativas tomadas, esta tarefa é inspirada na formalização apresentada no esquema *ComplianceNorms* descrito na Seção 4.3.2;

O evento com maior prioridade é selecionado pela função *Selecting Event*-Se-(o funcionamento desta função é inspirado no processo de seleção de objetivos descrito na Seção 4.3.3) e é unificado, utilizando o processo *Unify Event*, com as condições de invocação dos planos gerando um conjunto de *planos relevantes*. O contexto de tais planos são verificados de acordo com as crenças e normas do agente, utilizando o processo *Check Context*, gerando um conjunto de *planos aplicáveis* (o funcionamento deste processo é inspirado na função *genapplplans* descrita na Seção 4.3.3).

O plano aplicável com maior importância é escolhido pela função *Selecting Plan*-Sp-(o funcionamento desta função é inspirado no processo de seleção de planos descrito na Seção 4.3.3) para ser executado. As intenções são atualizadas a partir do plano selecionado e levando em consideração que a importância das intenções já existentes podem ter modificado dado que algumas normas podem ter sido ativadas, desativadas, cumpridas ou violadas. A execução de tal processo é baseada no esquema *ReviewIntentions* apresentando na Seção 4.3.3.

A função *Selecting Intention*-Si-(o funcionamento desta função é inspirado no processo de seleção de intenções descrito na Seção 4.3.4) seleciona a intenção de maior importância para ser executada. Tal intenção é executada e monitorada pelo processo (*Execute Intention*), assim como discutido na Seção 4.3.4.

Nas Seções seguintes descreveremos cada uma das funcionalidades normativas.

5.2.1

PUC-Rio - Certificação Digital Nº 1012680/CA

Algoritmo 1 *Adopting Norms*

Require: Normas Sensoriadas, representadas por *sensednorms*
Require: Normas Adotadas, representadas por *adoptednorms*
Require: Nome do Agent, representado por *name*
Require: Papeis assumidos pelo Agent, representado por *roles*
Require: Grupos que o Agent faz parte, representado por *groups*

```
1: for all Norm sn in sensednorms do
2:   if !exist(sn, sensednorms)) then
3:     if isAddressees(name,roles,groups,sn.addressees) then
4:       adoptednorms.add(sn)
5:     end if
6:   end if
7: end for
```

Função *Normative Reviewer*

Esta função atualiza o conjunto de normas adotadas, ativadas e desativadas, além de verificar as normas que foram cumpridas ou violadas a partir da realização das atividades: *Adopting Norms*, *Reviewing (De)Activated Norms* e *Verifying Fulfillment Norms*. Tais atividades serão apresentadas em mais detalhes nas próximas Seções.

Adopting Norms

A atividade *Adopting Norms* é responsável por ajudar o agente a reconhecer suas responsabilidades através da adoção de normas que especificam tais responsabilidades, a realização desta tarefa é guiada pela formalização apresentada no esquema *AdoptingNorms* descrito na seção 4.3.1, e o Algoritmo 1 apresenta a operacionalização desta fomalização. A operação têm início a partir do conjunto de normas seonsoriadas pelo agente, para cada norma sensoriada as seguintes verificações são realizadas: (i) se a norma sensoriada ainda não existe na base de normas adotadas (ver linha 2); e (ii) se a norma sensoriada é endereçável ao agente (ver linha 3). se tais condições são satisfeitas, a norma sensoriada é adicionada ao conjunto de normas adotadas (ver linha 4). A verificação se uma norma sensoriada já existe na base de normas adotadas é realizada utilizando a função *exist* que recebe uma norma e um conjunto de normas, e retorna *true* se a norma existe no conjunto de normas. Já a verificação se o agente é responsável pela norma sensoriada é realizada utilizando a função *isAddressees* que recebe como entrada o nome do agente, os papéis assumidos por ele e os grupos que o mesmo pertence, e o campo *Addressees* da norma, e é satisfatível se o nome, papel ou grupo do agente é um dos endereçadores da norma.

PUC-Rio - Certificação Digital Nº 1012680/CA

Algoritmo 2 *Reviewing (De)Activated Norms*

Require: Base de crenas do agente, representada por *bels*
Require: Normas Adotadas, representadas por *adoptednorms*
Require: Normas Ativadas, representadas por *acnorms*
Require: Normas Desativadas, representadas por *deacnorms*
1: **for all** Norm ad *in* adoptednorms **do**
2: **if** *verifyContext*(ad.activation, bels, acnorms, deacnorms) **then**
3: *acnorms.add*(ad)
4: **end if**
5: **end for**
6: **for all** Norm deac *in* acnorms **do**
7: **if** *verifyContext*(deac.deactivation, bels, acnorms, deacnorms) **then**
8: *acnorms.remove*(deac)
9: *deacnorms.add*(deac)
10: **end if**
11: **end for**

Reviewing DeActivated Norms

Esta atividade checa e atualiza o conjunto de normas ativadas e desativadas, a realização desta tarefa é guiada pela formalização apresentada no esquema *ReviewDeActivatedNorms* descrito na seção 4.3.1 e a sua operacionalização é realizada pelo Algoritmo 2 que funciona como segue:

Verificação das Normas Ativadas: O contexto de ativação de cada norma adotada pelo agente é verificado utilizando a função *verifyContext* na linha 2, se o contexto de ativação é satisfeito, a norma é adicionada ao conjunto de normas ativadas na linha 3. O funcionamento da função *verifyContext* tem como base a formalização da função *verifiContext* apresentada na Seção 4.3.1 e sua operacionalização será melhor explicada na Seção 5.2.4 .

Verificação das Normas Desativadas: O contexto de desativação de cada norma ativada é verificado utilizando a função *verifyContext* na linha 7, se o contexto de desativação é satisfeito, a norma ativada é removida do conjunto de normas ativadas na linha 8 e é adicionada ao conjunto de normas desativadas na linha 9.

Verifying Fulfillment Norms

A atividade *Verifying Fulfillment Norms* checa e atualiza o conjunto de normas cumpridas ou violadas, a realização desta atividade é guiada pela formalização apresentada no esquema *VerifyingFulfillmentNorms* descrito na Seção 4.3.1 e a sua operacionalização é realizada pelo Algoritmo 3 como segue:

PUC-Rio - Certificação Digital Nº 1012680/CA

Algoritmo 3 *Verifying Fulfillment Norms*

Require: Normas Ativadas, representadas por *activatednorms*
Require: Normas Desativadas, representadas por *deactivatednorms*
Require: Uma *Map* indicando se um norma foi cumprida ou violada, tal *Map* é baseada na relacao *normfulfillment* apresentado na Secao 4.2
Require: Uma *Map* indicando se o comportamento regulado pela norma foi realizado, tal *Map* é baseada na relacao *behaviorrealization* apresentada na Secao 4.2

```
1:  for all Norm ac in activatednorms do
2:    if behaviorrealization.get(ac) == realized then
3:      if ac.deonticconcept = OBLIGATION then
4:        normfulfillment.put(ac, FULFILLED)
5:      end if
6:      if ac.deonticconcept = PROHIBITION then
7:        normfulfillment.put(ac, VIOLATED)
8:      end if
9:    end if
10: end for
11: for all Norm deac in deactivatednorms do
12:   if behaviorrealization.get(deac) == not_realized then
13:     if deac.deonticconcept = OBLIGATION then
14:       normfulfillment.put(deac, VIOLATED)
15:     end if
16:     if deac.deonticconcept = PROHIBITION then
17:       normfulfillment.put(deac, FULFILLED)
18:     end if
19:   end if
20: end for
```

Verificação do Cumprimento das Normas Ativadas Para cada norma ativada é verificado se a mesma teve seu comportamento realizado (linha 2). Em caso afirmativo, podemos ter as seguintes opções:

- Se a norma é uma obrigação, ela foi cumprida e o seu *status* de cumprimento é atualizado para *FULFILLED* (linhas 3 e 4);
- Se a norma é uma proibição, ela foi violada e o seu *status* de cumprimento é atualizado para *VIOLATED* (linhas 6 e 7).

Verificação do Cumprimento das Normas Desativadas Para cada norma desativada é verificado se a norma não teve seu comportamento realizado (linha 12). Em caso afirmativo, podemos ter as seguintes opções:

- Se a norma é uma obrigação, ela foi violada e e o seu *status* de cumprimento é atualizado para *VIOLATED* (linhas 13 e 14);
- Se a norma é uma proibição, ela foi cumprida e o seu *status* de cumprimento é atualizado para *FULFILLED* (linhas 16 e 17);

PUC-Rio - Certificação Digital Nº 1012680/CA

5.2.2 Função *Events Normative Generator*

Esta função detecta e soluciona conflitos entre normas, seleciona as normas não conflitantes que devem ser cumpridas ou violadas, e atualiza a base de eventos de acordo com as decisões normativas tomadas. Para tanto as seguintes atividades são realizadas: *Detect and Overcom Conflicts*, *Choosing Normas* e *Compliance with Norms*.

Detect and Overcome Conflicts

A atividade *Detect and Overcome Conflicts* é responsável por detectar e superar conflitos entre normas, a realização desta atividade é guiada pela formalização apresentada no esquema *DetectOvercomeConflicts* descrito na seção 4.3.2 e a sua operacionalização é realizada pelo Algoritmo 4 que verifica se existem conflitos entre normas, como descrito na linha 3.

Em caso afirmativo, se a *influência deôntica* da primeira norma (verificada utilizando a função *deonticInfluence* que implementa a definição *deonticInfluence* descrita na Seção 4.3.2) mais a importância das recompensas por cumprir a primeira norma (a importância das recompensas é calculada utilizando a função *rewardsImportance* que implementa a função *rewardsImportance* formalizada na Seção 4.3.2) e a importância das puniões por violar a segunda norma (a importância das puniões é calculada pela função *punishmentImportance* que implementa a definição *punishmentsImportance* descrita na Seção 4.3.2) é maior do que a importância por receber as recompensas por cumprir a segunda norma mais a importância por receber as punições por violar a primeira norma (como descrito na linha 4). Se tal condição é satisfeita, a primeira norma é selecionada para ser cumprida e o seu *status* de seleção é atualizado para *TOBE_FULFILLED*, linha 5, a segunda norma é selecionada para ser violada e o seu *status* de seleção é atualizado para *TOBE_VIOLATED*, linha 6. Caso contrário, a primeira norma é selecionada para ser violada, linha 8, e a segunda norma para ser cumprida, linha 9.

Choosing Normas

A atividade *Choosing Normas* é responsável por selecionar quais normas não conflitantes serão cumpridas ou violadas, a realização desta atividade é guiada pela formalização apresentada no esquema *ChoosingNorms* descrito na seção 4.3.2 e a sua operacionalização é realizada pelo Algoritmo 5 que verifica quais das normas ativadas não é conflitante (linha 2), tal verificação é realizada utilizando a função *nonConflicting*. Em caso afirmativo, se a influência deôntica da norma mais as recompensas por cumprir a norma é maior do que as punições

PUC-Rio - Certificação Digital Nº 1012680/CA

Algoritmo 4 Detect and Overcome Conflicts

Require: Normas Ativadas representadas por *activatednorms*
Require: Uma *Map* representando a relacao *normselection* apresentada na Secao 4.2

```
1:  for all Norm ac1 in activatednorms do
2:    for all Norm ac2 in activatednorms; ac2 != ac1 do
3:      if detectConflict(ac1, ac2) then
4:        if deonticInfluence(ac1)+
          rewardsImportance(ac1, activatednorms) +
          punishmentsImportance(ac2, activatednorms) >=
          rewardsImportance(ac2, activatednorms) +
          punishmentsImportance(ac1, activatednorms) then
5:          normselection.put(ac1, TOBE_FULFILLED)
6:          normselection.put(ac2, TOBE_VIOLATED)
7:        else
8:          normselection.put(ac1, TOBE_VIOLATED)
9:          normselection.put(ac2, TOBE_FULFILLED)
10:       end if
11:     end if
12:   end for
13: end for
```

Algoritmo 5 Choosing Norms

Require: Normas Ativadas representadas por *activatednorms*
Require: Uma *Map* representando a relao *normselection* apresentada na Secao 4.2

```
1: for all Norm ac in activatednorms do
2:   if nonConflicting(ac) then
3:     if deonticInfluence(ac)+ rewardsImportance(ac, activatednorms)
       >= punishmentsImportance(ac, activatednorms) then
4:       normselection.put(ac, TOBE_FULFILLED)
5:     else
6:       normselection.put(ac, TOBE_VIOLATED)
7:     end if
8:   end if
9: end for
```

por violar a norma (linha 3). Então, a norma é selecionada para ser cumprida (linha 4). Caso contrário, a norma é selecionada para ser violada (linha 6).

Compliance with Norms

Esta atividade é responsável por efetivar as decisões normativas tomadas, adicionando novos eventos a base de eventos do agente. Como discutido na Seção 2.3, agentes implementados utilizando Jason Normativo reagem a eventos cujo *trigger* podem estar representando a adição de objetivos. Desta forma, novos eventos são gerados para representar o desejo do agente em realizar comportamentos regulados por normas de obrigação selecionadas para serem cumpridas ou normas de proibição selecionadas para serem violadas.

A realização desta atividade é baseada na formalização apresentada no esquema *ComplianceNorms* descrito na seção 4.3.2 e operacionalizada pelo

PUC-Rio - Certificação Digital Nº 1012680/CA

Algoritmo 6 Compliance with Norms

Require: Normas Ativadas representadas por *activatednorms*
Require: Uma *Map* representando a relao *normselection*

```
1: for all Norm ac in activatednorms do
2:   if !existBehavior(ac.behavior) then
3:     if ( ac.deonticconcept = OBLIGATION and
       normselection.get(ac) = TOBE_FULFILLED ) or
       ( ac.deonticconcept = PROHIBITION and
       normselection.get(ac) = TOBE_VIOLATED ) then
4:       makeEvent(ac.behavior)
5:     end if
6:   end if
7: end for
```

Algoritmo 6 que verifique se já não existe um comportamento que efetive a decisão normativa, ver linha 2, se não existe e uma norma de obrigação foi selecionada para ser cumprida ou uma norma de proibição foi selecionada para ser violada. Então, um novo evento cujo trigger representa a adição do comportamento regulado pela norma é gerado e adicionado a base de eventos. A criação do novo evento é realizada utilizando a função *makeEvent* que recebe o comportamento regulado pela norma como entrada e adiciona um evento na base de eventos do agente, cujo *trigger* represente o tipo de comportamento a ser realizado.

5.2.3 *Selecting Event*

Similar a um agente Jason, um agente Jason Normativo possui a sua disposição uma lista de eventos a serem gerenciados e um deles precisa ser selecionado. Tal seleção é realizada tomando como base o processo de seleção de objetivos apresentado na Seção 4.3.3, onde cada cada objetivo possui uma prioridade associada que é definida levando em consideração a motivação do agente em atingir tal objetivo, e que o cumprimento de tal objetivo pode levar ao cumprimento e violação de um conjunto de normas.

Baseado nesta idéia, a atividade *Selecting Event* é realizada considerando que cada evento tem uma prioridade associada que é definida a partir da prioridade do *trigger* que o compõe. Lembre-se da Seção 2.3, que um *trigger* pode está representando uma crença ou um objetivo. Desta forma, definimos a prioridade de um *trigger* tomando como base o elemento que ele está representando como descrito no Algoritmo 7):

– Se o *trigger* está relacionado a crenças, o Algoritmo 7 retornará zero. Pois, sua prioridade será definida a partir da ordem com que tais eventos

PUC-Rio - Certificação Digital Nº 1012680/CA

foram percebidos pelo agente como descrito em (d'Inverno et al. 2004).

– Se o *trigger* está representando um objetivo (representado no Algoritmo 7 como o tipo GOAL), então a prioridade do *trigger* é definida inspirada na formalização da função *goalPriority* apresentada na Seção 4.3.3, onde a prioridade de um objetivo é definida levando em consideração a motivação do agente em atingir tal objetivo objetivo mais a importância das recompensas por cumprir um conjunto de normas de obrigação e a importância das punições por violar um conjunto de normas de proibições ao atingir tal objetivo.

  A motivação do agente para atingir um objetivo é calculada utilizando a função *motivation*, que é similar a função *motivatioin* definida na Seção 4.1.1, e a importância por receber um conjunto de recompensas ou punições é avaliada utilizando a função *behaviorNormativeInfluence* cuja implementação é realizada tomando como base a formalização da função *behaviorNormativeInfluence* definida na Seção 4.3.3 .

Após definir a prioridade de cada evento, o Algoritmo 8 é executado a fim de selecionar o evento de maior prioridade. Primeiro, é verificado se o evento no topo da lista de eventos é composto por um *trigger* representando uma crença (representada pelo tipo *BELIEF* no Algoritmo 8), então, tal evento é selecionado para ser lidado (ver linhas de 2 à 4), dado que a sua prioridade é definida somente considerando a ordem com que tal evento foi percebido, como descrito anteriormente. Caso contrário, o evento, cujo *trigger* está representando um objetivo com a maior prioridade será selecionado (ver linhas de 5 à 10).

PUC-Rio - Certificação Digital Nº 1012680/CA

5.2.4 *Check Context*

Após selecionar o evento de maior prioridade o próximo passo é encontrar os planos relevantes. Esta etapa funciona como no Jason Original e é realizada pelo processo *UnifyEvent* representado na Figura 5.1.

Após encontrar os planos relevantes, é necessário verificar quais deles são aplicáveis. Para tanto, o processo *Check Context* já provido pelo Jason foi estendido a fim de permitir a verificação explicita de condições normativas.

Sendo assim, no Jason Normativo o processo *Check Context* passou a trabalhar como apresentado no Algoritmo 9 (este algoritmo é uma operacionalização da formalização da função *VerifyContext* apresentada na Seção 4.3.1). Que funciona a partir da execução dos seguintes passos:

Algoritmo 7 triggerPriority(trigger, activatednorms)

Require: *Trigger* de um evento, representado pelo tipo *trigger*
Require: Conjunto de Normas Ativadas, representado por *activatednorms*

```
1: if trigger type GOAL then
2:    priority = motivation(trigger.goal) +
      behaviorNormativeInfluence(trigger.goal, activatednorms)
3:    return priority
4: end if
5: return 0
```

Algoritmo 8 selectEvent(events, activatednorms)

Require: Uma sequncia de eventos, representada por *events*
Require: Conjunto de Normas Ativadas, representado por *activatednorms*

```
1: event = events.get(0)
2: if event.trigger type BELIEF then
3:    return event
4: end if
5: for all i = 1 until events.size do
6:    if triggerPriority(events.get(i).trigger, activatednorms) >
      triggerPriority(event.trigger, activatednorms) then
7:       event = events.get(i)
8:    end if
9: end for
10: return event
```

Algoritmo 9 verifyContext(context, beliefs, acnorms, deacnorms)

Require: Um contexto, representada por *context*
Require: Um conjunto de crencas, representada por *bels*
Require: Normas Ativadas, representadas por *acnorms*
Require: Normas Desativadas, representadas por *deacnorms*

```
1: if context type beliefcontext then
2:    return logialConsequence( context, bels)
3: end if
4: if context type normativeconditioncontext then
5:    return normativeConsequence(context, acnorms, deacnorms)
6: end if
7: if context type and then
8:    return verifyContext(context.left(), bels, acnorms, deacnorms) &
      verifyContext(context.right(), bels, acnorms, deacnorms)
9: end if
10: if context type or then
11:    return verifyContext(context.left(), bels, acnorms, deacnorms) |
       verifyContext(context.right(), bels, acnorms, deacnorms)
12: end if
```

PUC-Rio - Certificação Digital Nº 1012680/CA

Algoritmo 10 normativeConsequence(ncs, bels, actns, deacns)

Require: Um conjunto de condicoes normativas, representada por *ncs*
Require: Normas Ativadas, representadas por *acns*
Require: Normas Desativadas, representadas por *deacns*
Require: Uma *Map* indicando se um norma foi cumprida ou violada, tal *Map* baseada no relacionamento *normfulfillment* apresentado na Seo

```
1:  sat = true
2:  for all NormativeCondition nc in ncs do
3:    if !( nc.normativestate = ACTIVATED and
      existNorm(nc.norm, acns)) then
4:      sat = false
5:    end if
6:    if !( nc.normativestate = DEACTIVATED and
      existNorm(nc.norm, deacns)) then
7:      sat = false
8:    end if
9:    if !( nc.normativestate = FULFILLED and
      normfulfillment.get(nc.norm) == FULFILLED) then
10:     sat = false
11:   end if
12:   if !( nc.normativestate = VIOLATED and
      normfulfillment.get(nc.norm) == VIOLATED) then
13:     sat = false
14:   end if
15: end for
16: return sat
```

– Primeiro, é verificado se o contexto está relacionado somente a condições que devem ser satisfeitas a partir das crenças do agente (ver linha 1), em caso afirmativo, a função *logicalConsequence* é utilizada para verificar se o contexto é consequência lógica das crenças do agente (ver linha 2). A função *logicalConsequence* já é disponibilizada pelo Jason original e nenhum extensão foi realizada;

– Segundo, é verificado se o contexto está relacionado a condições normativas (ver linha 4), em caso afirmativo, a função *normativeConsequence* é utilizada para verificar se o contexto é satisfeito a partir do estado atual das normas do agente (ver linha 5). Para realizar tal verificação a formalização da função *NormativeConsequence* descrita na Seção 4.3.1 foi implementada como descrito no Algoritmo 10. Que recebe um conjunto de condições normativas e o conjunto de normas ativadas e desativadas do agente, e retorna *true* se todas as condições são satisfeitas. Onde cada condição normativa é verificada como segue:

  – Se o estado normativo especificado na condição normativa é do tipo *ACTIVATED*, o contexto é satisfeito se a norma especificada na condição normativa existe na base de normas ativadas (ver linhas de 3-5) ;

PUC-Rio - Certificação Digital Nº 1012680/CA

Algoritmo 13 planImportance(plan, acnorms)

Require: Plano, representado por *plan*
Require: O conjunto de normas ativadas, representado por *acnorms*
1: **return** triggerPriority(plan.trigger, acnorms) +
bodyMainImportance(plan.body) +
bodyNormativeInfluence(plan.body, acnorms)

- Se o estado normativo especificado na condição normativa é do tipo *DEACTIVATED*, o contexto é satisfeito se a norma especificada na condição normativa existe na base de normas desativadas (ver linhas de 6-8);
- Se o estado normativo especificado na condição normativa é do tipo *FULFILLED*, o contexto é satisfeito se o *status* de cumprimento da norma especificada na condição normativa é igual a *FULFILLED* (ver linhas de 9-11);
- Se o estado normativo especificado na condição normativa é do tipo *VIOLATED*, o contexto é satisfeito se o *status* de cumprimento da norma especificada na condição normativa é igual a *VIOLATED* (ver linhas de 12-14);

– O último caso lida com a situação onde existe os dois tipos de contexto.

5.2.5 Selecting Plan

Após encontrar os planos relevantes e aplicáveis, ainda podemos ter um conjunto de planos aplicáveis, logo, é necessário escolher um deles para se tornar uma intenção. A fim de capacitar o agente com a habilidade necessária para tomar tal decisão, nos baseamos na formaliação da função *selectPlan* descrita na seção 4.3.3, onde o plano de maior importância é selecionado e a importância de cada plano é avaliada considerando a prioridade da condição de invocação do plano (como descrito na formalização da função *goalPriority* apresentada na Seção 4.3.3), a importância de realizar os comportamentos que compõem o corpo do plano (como formalizado na função *bodyMainImportance* apresentada na Seção 4.3.3) e a importância em cumprir ou violar um conjunto de normas ao realizar os comportamentos que constituem o corpo do plano (como formalizado na função *bodyNormativeInfluence* apresentada na Seção 4.3.3).

A partir desta discussão e dado que a condição de invocação de um plano no *AgentSpeak(L)* é representada por um *trigger*. A importância de um plano no *AgentSpeak(L)* é avaliada como apresentado no Algoritmo 13. Onde a importância de um plano é definida a partir da execução das seguintes funções:

PUC-Rio - Certificação Digital Nº 1012680/CA

Algoritmo 11 bodyMainImportance(body)

```
Require: Corpo do Plano, representado por body
1: for all i = 1 until body.body_formulas.size do
2:   if body.body_formula(i) type GOAL then
3:     bmi = bmi + motivation(body.body_formula(i))
4:   end if
5:   if body.body_formula(i) type ACTION then
6:     bmi = bmi + satisfaction(body.body_formula(i))
7:   end if
8: end for
9: return bmi
```

Algoritmo 12 bodyNormativeInfluence(body, acnorms)

```
Require: Corpo do Plano, representado por body
Require: O conjunto de normas ativadas, representado por acnorms
1: for all i = 1 until body.body_formulas.size do
2:   bni = bni +
     behaviorNormativeInfluence(body.body_formula(i), acnorms)
3: end for
4: return bni
```

(***triggerPriority***) A função *triggerPriority* já descrita anteriormente, avalia a prioridade do *trigger* que compõe a condição de invocação do plano;

(***bodyMainImportance***) A função *bodyMainImportance*, descrita no Algoritmo 11, avalia a importância de atingir os objetivos e ações que compõem o corpo do plano e funciona semelhante a formalização *bodyMainImportance* descrita na Seção 4.3.3. Onde os componentes que constituem o corpo do plano (no *AgentSpeak(L)* tais componentes são chamados *body_formula*) são percorridos e a importância por realizá-los é avaliada tomando como base a formalização *behaviorImportance* descrita na Seção 4.3.3 . Ou seja, se o *body_formula* é um objetivo, então a importância é igual a motivação do agente em atingir tal objetivo (ver linhas de 2-4) e se o *body_formula* é uma ação, então a importância é igual a satisfação do agente em exexutar tal ação (ver linhas de 5-7).

(***bodyNoramtiveInfluence***) A função *bodyNoramtiveInfluence*, descrita no Algoritmo 12, avalia a influência normativa sobre o corpo do plano e funciona semelhante a formalização *bodyNoramtiveInfluence*, onde os comportamento que constituem o corpo o plano são percorridos e a influência das normas sobre cada comportamento é avaliada utilizando a função *bodyNormativeInfluence* que funciona como a formalização *bodyNormativeInfluence* descrita na Seção 4.3.3.

Finalmente, o plano com maior importância é retornado como descrito no Algoritmo 14.

PUC-Rio - Certificação Digital Nº 1012680/CA

Algoritmo 14 selectPlan(plans, acnorms)

Require: Lista de Planos, representada por *plans*
Require: O conjunto de normas ativadas, representadas por *acnorms*

```
1: plan = plans.get(0)
2: for all i = 1 until plans.size do
3:    if planImportance(plans.get(i), acnorms) >
      planImportance(plan, acnorms) then
4:       plan = plans.get(i)
5:    end if
6: end for
7: return plan
```

Algoritmo 15 intentionImportance(int, acnorms)

Require: Inteno representada por *int*
Require: O conjunto de normas ativadas, representadas por *acnorms*

```
1: ii = 0
2: for all i = 1 until int.size do
3:    ii = ii + planImportance(int(i), acnorms)
4: end for
5: return ii
```

PUC-Rio - Certificação Digital Nº 1012680/CA

5.2.6 *Review Intention*

Como discutido na Seção 4.3.3, as intenções de um agente precisam ser revistas e para capacitar um agente *Jason Normativo* com a habilidade de rever suas intenções consideramos que cada intenção tem uma importância associada, então, as intenções de maior importância devem ser mantidas na base de intenções.

Onde a importância de uma intenção é definida, como descrito no Algoritmo 15, como a soma da importância dos planos que compõem a intenção, tal como definido pela formalização *intentionImportance* descrita na Seção 4.3.3. Note que como no Jason, uma intenção no Jason Normativo é uma seqüência de planos.

Antes de revisarmos o conjunto de intenções, é necessário ressaltar que assim como em ANA, no *Jason Normativo* uma intenção pode assumir um *status* de três: *active*, *suspended* e *active_in_execution*. Se a intenção é *suspended* ou *active_in_execution* não pode ser revista dado que o seu processo de execução já foi iniciado. No entanto, se a intenção é *active*, a mesma ainda não foi selecionada para ser executada e pode ser revista.

Desta forma, assim como descrito no esquema *ReviewIntention* descrito na Seção 4.3.3, o primeiro passo do processo de revisão de intenções (ver linhas de 1-5 no Algoritmo 16) é rever as intenções *active* levando em consideração a *importância* associada a cada uma delas.

Algoritmo 16 *Review Intention*

Require: Lista de Intencoes representada por *intentions*
Require: Lista de Eventos representada por *events*
Require: O conjunto de crencas do agente, representadas por *bels*
Require: Normas Ativadas, representadas por *acns*
Require: Normas Desativadas, representadas por *deacns*
Require: Uma *Map* representando o *status* da intencao, representada por *intentionstatus*

```
1: for all Intention int in intentions do
2:   if intentionsstatus.get(int) = ACTIVE then
3:     int = getNewIntention(int.get(0).trigger, bels, acns, deacns)
4:   end if
5: end for
6: event = selectEvent(events, activatednorms)
7: newintention = getNewIntention(event.trigger, bels, acns, deacns)
8: if undefined(event) then
9:   intentions.add(newintention)
10:  intentionstatus.put(newintention, ACTIVE)
11: end if
12: if defined(event) then
13:  oldintention = getOldIntention(event)
14:  intentions.update(oldintention, newintention)
15:  intentionstatus.put(newintention, ACTIVE_IN_EXECUTION)
16: end if
```

Para tanto, assim como no esquema *ReviewIntention*, uma função *getNewIntention* é definida. Tal função recebe o *trigger* da condição de invocação do plano no topo da intenção, a biblioteca de planos, o conjunto de crenças e o conjunto de normas e retorna o plano de maior importância cuja condição de invocação é composta pelo mesmo *trigger* que foi passado como entrada.

Depois de revisar as intenções *active*, seguimos os passos descritos no esquema *ReviewIntention*, onde foi definido que se um objetivo é definido, ou seja, ele foi gerado a partir da execução de uma intenção, o plano selecionado para atingir tal objetivo é adicionado a intenção que gerou tal objetivo e o *status* de tal intenção é atualizado para *active_in_execution*. Mas, se o objetivo é indefinido, ele não possui uma intenção associada, o plano selecionado para atingir tal objetivo é adicionado como uma nova intenção na base de intenções do agente e o *status* de tal intenção é atualizado para *active*.

Baseado no discutido acima, no Jason Normativo as intenções são atualizadas tomando como base o evento selecionado para ser gerenciado (representado por *event* no Algiritmo 16 na linha 6) e o plano selecionado para lidar com tal evento (representado por *newintention* no Algiritmo 16 na linha 7) , como segue:

1. Se o evento selecionado é indefinido (ver linha 8), ou seja, ele foi gerado a partir da execução de uma intenção, o plano selecionado para lidar com tal evento é adicionado como uma nova intenção a base de intenções do

PUC-Rio - Certificação Digital Nº 1012680/CA

Algoritmo 17 selectIntention(intentions, acnorms)

Require: Lista de Intencoes, representada por *intentions*
Require: O conjunto de normas ativadas, representadas por *acnorms*

```
1: intention = intentions.get(0)
2: for all i = 1 until intentions.size do
3:    if intentionImportance(intentions.get(i), acnorms) >
      intentionImportance(intention, acnorms) then
4:       intention = intentions.get(i)
5:    end if
6: end for
7: return intention
```

agente (ver linha 9) e o *status* de tal intenção é atualizado para *active*(ver linha 10);

2. Se o evento selecionado é definido (ver linha 12), ou seja, ele foi gerado a partir da execução de uma intenção, o plano selecionado para lidar com tal evento é adicionado a intenção que gerou tal evento (ver linha 14) e o *status* de tal intenção é atualizado para *active_in_execution*(ver linha 15). A intenção que gerou tal evento é obtida utilizando a função *getOldIntention* (ver linha 13).

5.2.7 *Selecting Intention*

A atividade *Selecting Intention* é responsável por selecionar a intenção de maior importância para ser executada, a realização desta atividade é guiada pela formalização da função *selectIntention* descrita na Seção 4.3.4, e o Algoritmo 17 é responsável por executar tal passo.

5.2.8 *Execute Intention*

Após atualizar as intenções, o agente Jason Normativo inicia o processo de execução e monitoramento da execução de suas intenções. O Algoritmo 18 é utilizado para selecionar uma intenção. A variável, *selectedint*, é a intenção selecionado. A variável *executingplan* representa o plano de execução (ou seja, primeiro plano) da intenção e a variável *executingbehavior* é o comportamento de execução (ou seja, primeiro comportamento) neste plano. Este processo é baseado na formalização do esquema *SelectIntention* apresentado na Seção 4.3.4 .

O processo restante é descrito no Algoritmo 19. Assim como em ANA, durante a execução de uma intenção no *Jason Normativo*, duas possibilidades podem surgir devido aos diferentes comportamentos que constituem o plano no

PUC-Rio - Certificação Digital Nº 1012680/CA

Algoritmo 18 SelectIntention

Require: Lista de Intencoes representada por *intentions*
Require: O conjunto de normas ativadas, representadas por *activatednorms*

```
1: selectedint = selectIntention(intentions, activatednorms)
2: executingplan = selectedintention.get(0)
3: executingbehavior = executingplan.body.get(0)
4: executeMonitorIntention(selectedint, executingplan, executingbehavior)
```

Algoritmo 19 executeMonitorIntention(selectedint, execplan, execbeh)

Require: Normas ativadas, representadas por *activatednorms*
Require: Normas cujo comportamento regulado foi realizado, representadas por *behaviorrealization*
Require: O conjunto de intencoes, representadas por *intentions*
Require: Intencao selecionada, representada por *selectedint*
Require: Plano em execucao, representada por *execgplan*
Require: Comportamento a ser realizado, representada por *execbeh*

```
1: if execbeh type GOAL then
2:    events.add(makeEvent(execbeh))
3:    intentionstatus.put(selectedint, SUSPENDED)
4:    SelectIntention()
5: end if
6: if execbeh type ACTION then
7:    actions.add(execbeh)
8:    norm = existNorm(execbeh, activatednorms)
9:    if norm != null then
10:      behaviorrealization.add(norm)
11:   end if
12:   execplan.remove(execbeh)
13:   selectedint.update(executingplan)
14:   executeMonitorIntention(selectedint, execplan, execplan.body.get(0))
15: end if
16: if execplan.body.size = 0 and selectedint.size > 1 then
17:   norm = existNorm(execplan.trigger, activatednorms)
18:   if norm != null then
19:      behaviorrealization.add(norm)
20:   end if
21:   nextplan = selectedint.get(2)
22:   selectedint.update(nextplan)
23:   executeMonitorIntention(selectedint, nextplan, execplan.body.get(0))
24: end if
25: if execplan.body.size = 0 and selectedint.size = 1 then
26:   intentions.remove(selectedint)
27:   SelectIntention()
28: end if
```

PUC-Rio - Certificação Digital Nº 1012680/CA

topo da intenção, estas variam dependendo se o comportamento é um objetivo ou uma ação. Primeiro, se o comportamento é um objetivo, um novo evento é gerado para atingir tal objetivo. Este evento é adicionado ao conjunto de eventos a serem gerenciados pelo agente. Neste caso, o *status* da intenção em execução deve ser definido como *suspended* e uma nova intenção deve ser selecionada (ver linhas de 1-5).

Segundo, se o comportamento é uma ação, ela é adicionada ao conjunto de ações a serem executadas. Se existe uma norma regulando tal ação. Então, tal norma é adicionada ao conjunto de normas cujo comportamento regulado foi realizado (como também definido no esquema *PostAction* apresentado na Seção 4.3.4). Assim como em ANA, no Jason Normativo este passo é importante para determinar quais normas de ação foram cumpridas ou violadas, conforme mostrado na Seção 4.3.1. Agora, o comportamento em execução será explicitamente removido do corpo do plano em execução (similarmente ao esquema *RemoveBehavior* na Seção 4.3.4), a intenção será atualizada e o processo de execução inicia novamente (ver linhas de 6-15) .

Como descrito nas linhas de 16-24, se, após o comportamento é removido, não existem comportamentos no plano atual (ou seja, $executingplan.body.size = 0$), mas ainda há mais planos na pilha de intenção ($selectedintetion.size > 1$), o plano atual está terminado e o próximo plano na intenção está pronto para ser executado. Neste caso, observe que um plano foi executado completamente e se o objetivo que constitui o *trigger* da condição de invocação do plano é regulado por uma norma, isto significa que o comportamento regulado pela norma foi realizado. Este passo é similar ao processo descrito no esquema *AchievePlanOnly* definido na Seção 4.3.4 .

Finalmente, tal como descrito no esquema *AchieveIntention* descrito na Seção 4.3.4, se após a realização de um comportamento, o plano foi executado completamente e não há mais planos na intenção (ou seja, $executingplan.body.size = 0$ e $selectedintention.size = 1$), a intenção foi realizada e pode ser removida do conjunto de intenções do agente (ver linhas de 25-28).

5.2.9 Considerações Finais

Este Capítulo apresentou uma extensão da linguagem AgentSpeak e do interpretador Jason que possibilita a implementação de ANA.

o AgentSpeak(L) foi estendido através da adição de um novo tipo representando as normas do agente. E o tipo *context* já disponibilizado pelo AgentSpeak(L) foi estendido para possibilitar a definição explicita de condições

PUC-Rio - Certificação Digital Nº 1012680/CA

normativas.

O interpretador Jason foi estendido através da adição de funcionalidades normativas e algumas funções já existentes foram estendidaas para que o seu funcionamento levasse em consideração as normas do sistema.

Um importante aspecto a ser observado neste Capítulo foi o direto mapeamento entre a formalização apresentada no Capítulo 4 e as funções do Jason Normativo. Tal mapeamento pode ser melhor visualizado através do conjunto de algoritmos apresentados.

PUC-Rio - Certificação Digital Nº 1012680/CA