9 REPRESENTACIONES BDI

Las Lógicas BDI nos permiten razonar acerca de nuestros agentes racionales, permitiendo especificar y verificar el comportamiento de estos sistemas. Sus componentes modales Intencionales, temporales y de acción, son lo suficientemente **expresivos** como para abordar creencias, deseos, intenciones y la toma de decisión en esos términos. Desafortunadamente, la **complejidad computacional** de su teoría de prueba es elevada, aunque no mayor que el de su componente temporal; y la evidencia clara de su completitud fue tardía [88].

Expresividad y Costo de las Lógicas BDI

A pesar de ello, se propusieron una serie de **arquitecturas BDI** como implementaciones de este modelo de agencia, siendo PRS [44, 47, 46, 55, 62], y su exitosa revisión conocida como dMARS [45], el caso más representativo. En estas implementaciones es común encontrarse con simplificaciones difíciles de justificar, por ejemplo, las actitudes proposicionales no son operadores modales, sino estructuras de datos; Se usan meta-planes o programas definidos por el usuario para acelerar el cómputo llevado a cabo; etc. Si consideramos además la complejidad del código asociado a estos sistemas, el resultado es una aparente falta de rigor teórico; o por lo menos una evidente **divergencia** entre el estudio formal de los agentes BDI y su implementación.

Arquitecturas BDI

Fosa teórico-práctica

Los primeros intentos para resolver este problema, consistieron en proponer una **arquitectura BDI abstracta**, como una idealización que permitiera seguir investigando las propiedades teóricas de estos agentes, atendiendo al mismo tiempo los aspectos prácticos de su implementación. Un resultado sobresaliente es la especificación de dMARS realizada por d'Inverno y col. [33]. Para ello, se optó por modelar esta arquitectura, usando el lenguaje de especificación Z [66] [1], que combina descripciones textuales del sistema especificado con esquemas que hacen uso de la teoría de conjuntos, la lógica de primer orden y el cálculo lambda. Una visión completa sobre el uso de Z para el modelado de agentes puede revisarse en el libro de d'Inverno y Luck [35]. Sin embargo, debido a su nivel de abstracción, no fue posible establecer una **correspondencia** entre la teoría del modelo, la teoría de prueba y la arquitectura abstracta propuesta. No obstante, estas herramientas son interesantes, en tanto que especificaciones precisas, muy cercanas a la implementación [2].

Arquitecturas abstractas

Teoría de Correspondencia

Siendo uno de los personajes centrales en el desarrollo de las Lógicas y las Arquitecturas BDI, Rao [89] propuso entonces una ruta alternativa hacía una solución para cerrar la fosa teórico-práctica: Partir del análisis de una implementación BDI existente, dMARS, para formalizar su semántica operacional. El resultado es $AgentSpeak(L)$, un lenguaje de programación basado en una lógica restringida de primer orden con eventos y acciones. El comportamiento de un agente está dado por su programa escrito en este lenguaje. Las creencias, deseos e intenciones no son expresiones modales, pero pueden verse como tales desde una postura Intencional:

$AgentSpeak(L)$

- El modelo del agente mismo, del ambiente en que se encuentra y de otros agentes, constituyen las **creencias** del agente.

Creencias

- Los estados a los que un agente quiere llegar, con base en sus estímulos internos y externos, puede verse como los **deseos** del agente.

Deseos

1 Z se estandarizó, obteniendo una especificación ISO, en 2002. El portal Community Z Tools (CZT) ofrece todo lo necesario para trabajar con este lenguaje de especificación: http://czt.sourceforge.net/index.html

2 De hecho, para los interesados, d'Inverno y Luck [34] han modelado en Z el formalismo que nos ocupa en este capítulo, $AgentSpeak(L)$.

- Y los planes que el agente adopta para satisfacer su estímulos, pueden verse como sus **Intenciones**. *Intenciones*

De esta forma, $AgentSpeak(L)$ puede es un lenguaje de especificación ejecutable, cuyo comportamiento es similar a las Arquitecturas BDI en sus detalles fundamentales, acercando así los aspecto teóricos de la agencia BDI, a los detalles de su implementación. Aunque esta aproximación es más cercana a la **Programación Orientada a Agentes** (AOP) propuesta por Shoham [97], debe resaltarse que se ha abandonado toda aproximación modal a la Intencionalidad. *AOP*

En el resto del capítulo describiremos formalmente la sintaxis y la semántica operacional de $AgentSpeak(L)$, tal y como se ha implementado en *Jason* [12], el conocido intérprete de este lenguaje AOP implementado en Java. Posteriormente, propondremos una aproximación modal a la Intencionalidad de los programas $AgentSpeak(L)$, con base en su semántica operacional. Tal aproximación [50] nos permitirá razonar acerca de los agentes implementados en este lenguaje de programación. *Organización* *Jason* $CTL_{AgentSpeak(L)}$

9.1 SINTAXIS

El **alfabeto** de $AgentSpeak(L)$ asume un conjunto ilimitado de símbolos de variables (Var), funciones ($Func$), predicados ($Pred$) y acciones ($Actn$). Los símbolos de las constantes, como se verá más adelante, están incluidos en los de las funciones ($Const \subseteq Func$). También forman parte del alfabeto los conectivos lógicos usuales $\{\neg, \wedge\}$; El símbolo de negación fuerte $\sim$; y algunos símbolos especiales, que incluyen $\{!, ?, +, -, :, ;, \leftarrow, : -, \top\}$. *Alfabeto*

Las **variables** se denotan mediante cadenas de caracteres que inician con una mayúscula. El símbolo $\top$ denota elementos vacíos en el lenguaje, como una secuencia vacía. Se adoptan las definiciones estándar para término, átomo y literal, pero los átomos pueden adornarse para indicar su origen –La percepción del agente (`percept`), su propia base de creencias (`self`), u otro agente (id), tal y como lo ilustra la figura9.1): *Variables*

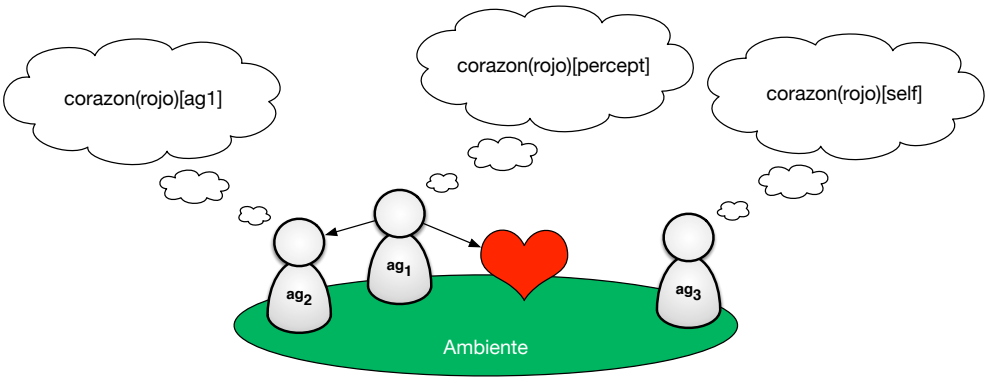

Figura 9.1: Las fuentes de un átomo en $AgentSpeak(L)$ pueden ser la percepción(ag_1), la comunicación (ag_2) o el razonamiento del propio agente (ag_3).

9.1.1 Términos, átomos y literales

Definición 9.1 (Término). *El conjunto de **términos** en $AgentSpeak(L)$ incluye las variables, las constantes y los términos compuestos:* *Término*

$$t \quad ::= \quad v \mid c \mid f(t_1, \ldots, t_n) \quad (n \geq 1) \tag{9.1}$$

donde $v \in Var$ es una variable; $c \in Const$, es una constante; y $f \in Func$ es una función de aridad $n \geq 1$, aplicada a $t_{1 \leq i \leq n}$ términos. $Const \subseteq Func$ es un conjunto de funciones de aridad cero.

Definición 9.2 (Átomo). *El conjunto de átomos en $AgentSpeak(L)$ incluye las proposiciones y los predicados, con anotaciones o sin ellas:* *Átomos*

$$\begin{aligned} at &::= p \mid p(t_1, \ldots, t_n) \quad (n \geq 1) && (9.2)\\ &\mid at[s_1, \ldots, s_m] \quad (m \geq 1) && (9.3)\\ s &::= \texttt{percept} \mid \texttt{self} \mid id && (9.4) \end{aligned}$$

donde $p \in Pred$ de aridad $n \geq 1$ es un predicado, aplicado a $t_{1 \leq i \leq n}$ términos; y si es de aridad cero, es una proposición atómica. Los átomos se adornan con al menos una anotación s, que indica su fuente de origen.

En la versión de $AgentSpeak(L)$ adoptada por *Jason*, en realidad todos los átomos tienen al menos una anotación, su fuente de origen (s). Las versiones de *at* no anotadas se incluye para tener compatibilidad con la versión original del lenguaje.

Observen que los términos compuestos y los átomos tienen la misma apariencia sintáctica, pero son de naturaleza diferente. Un término compuesto, al igual que los simples, denota un elemento del Universo de Discurso ($\mathcal{U}$), ya sea porque es una función del estilo $f : \mathcal{U}^n \mapsto \mathcal{U}$; o bien porque es una estructura de datos del estilo $f : \mathcal{U}^n$. Los átomos denotan relaciones entre elementos del universo de discurso, son predicados del estilo $p : \mathcal{U}^n \mapsto Bool$.

Ejemplo 9.1. *Si asumimos que $\mathcal{U} = \mathbb{N}$, la función $suma(4,3)$ es un término que denota al 7; y la función $cons(1, cons(2, [])) = [1,2]$ también es un término que denota a la lista $[1,2]$. El predicado $mayorQue(4,3)$ es un átomo que mapea a verdadero; y el predicado $vacio([1,2])$ también es un átomo que mapea a falso.*

Definición 9.3 (Literal). *Un átomo o su negación fuerte [3] son una literal. La negación se conoce como literal negativa y el átomo no negado como literal positiva:* *Literales*

$$lit ::= at \mid \sim at \quad (9.5)$$

9.1.2 Fórmulas bien formadas y creencias

Con las definiciones anteriores podemos abordar el concepto de fórmula lógica bien formada (fbf):

Definición 9.4 (Fórmula lógica). *Las fórmulas lógicas bien formadas en $AgentSpeak(L)$ incluyen los átomos, su negación y su conjunción:* *Fórmulas lógicas*

$$fbf ::= lit \mid \neg fbf \mid fbf \wedge fbf \quad (9.6)$$

donde lit es una literal; $\neg$ denota la negación débil y $\wedge$ la conjunción.

La versión original de $AgentSpeak(L)$ no consideraba la negación fuerte, que fue introducida por *Jason*. Si esta sintaxis require compatibilidad con la versión original del lenguaje, hay que substituir las literales (lit) de la definición, por átomos (at). De esa forma, solo la negación lógica débil forma parte de las fórmulas lógicas bien formadas.

Por otra parte, *Jason* provee un operador explícito de disyunción || como parte de la las fórmulas lógicas bien formadas. En esta formalización del lenguaje hemos incluido únicamente la conjunción y la negación, asumiendo que los demás operadores lógicos pueden definirse en esos términos. Por ejemplo, $\alpha \vee \beta \equiv \neg(\neg\alpha \wedge \neg\beta)$.

9.1.3 Programa de agente

Definición 9.5 (Programa de agente). *En $AgentSpeak(L)$, un **programa de agente** es un conjunto de creencias (bs), planes (ps) y metas (gs):* *Programa de agente*

$$ag \quad ::= \quad bs \quad ps \quad gs \tag{9.7}$$

Definición 9.6 (Creencias). *Una **creencia** en $AgentSpeak(L)$ es una literal de base o una regla:* *Creencias*

$$b \quad ::= \quad lit \mid rule \tag{9.8}$$

$$rule \quad ::= \quad lit_1 \text{ :- } fbf \tag{9.9}$$

$$bs \quad ::= \quad b_1, \ldots, b_n \quad (n \geq 0) \tag{9.10}$$

tal que lit no tiene variables sin instanciar como argumentos.

La versión original de $AgentSpeak(L)$ no consideraba el uso de reglas, si se requiere compatibilidad con esa versión del lenguaje hay que eliminar la ecuación 9.9 y la referencia a *rule* en la ecuación 9.8 de la definición anterior. De igual manera, si se requiere considerar solamente la negación débil, hay que substituir literales (*lit*) por átomos (*at*).

Aunque las creencias de un agente se asemejan a los hechos y reglas de la **programación lógica**, hay que recordar que los átomos de $AgentSpeak(L)$ están anotados. *Diferencias con Prolog*

Definición 9.7 (Planes). *Un **plan** en $AgentSpeak(L)$ es una estructura compuesta por un evento disparador (te), un contexto (ct) y un cuerpo (h):* *Planes*

$$p \quad ::= \quad te : ct \leftarrow h. \tag{9.11}$$

$$ps \quad ::= \quad p_1, \ldots, p_n \quad (n \geq 1) \tag{9.12}$$

Definición 9.8 (Evento disparador). *El **evento disparador** de un plan en $AgentSpeak(L)$ consiste en agregar, o eliminar, una creencia o una meta:* *Evento disparador*

$$te \quad ::= \quad +b \mid -b \mid +g \mid -g \tag{9.13}$$

Definición 9.9 (Contexto). *El **contexto** de un plan en $AgentSpeak(L)$ puede ser vacío o una fórmula lógica bien formada:* *Contexto*

$$ct \quad ::= \quad \top \mid fbf \tag{9.14}$$

Definición 9.10 (Cuerpo). *El **cuerpo** de un plan en $AgentSpeak(L)$ es una secuencia, posiblemente vacía, de acciones, actualizaciones de creencias y/o metas:* *Cuerpo*

$$h \quad ::= \quad h1; \top \mid \top \tag{9.15}$$

$$h1 \quad ::= \quad a \mid u \mid g \mid h1; h1 \tag{9.16}$$

Definición 9.11 (Acciones). *Las **acciones** en $AgentSpeak(L)$ son llamadas a procedimientos:* *Acciones*

$$a \quad ::= A(t_1, \ldots, t_n) \quad (n \geq 0) \tag{9.17}$$

donde $A \in Actn$ es una acción aplicada a $t_{1 \leq i \leq n}$ términos.

Definición 9.12. *Las **actualizaciones de creencias** en $AgentSpeak(L)$ consisten en agregar una nueva creencia o eliminar aquellas que unifican con una literal dada:* *Actualizaciones*

$$u \quad ::= \quad +b \mid -lit \tag{9.18}$$

Definición 9.13 (Metas). *Las **metas** en $AgentSpeak(L)$ pueden ser alcanzables (!) o verificables (?):* *Metas*

$$g \quad ::= \quad !lit \mid ?lit \tag{9.19}$$

$$gs \quad ::= \quad g_1, \ldots, g_n \quad (n \geq 0) \tag{9.20}$$

La compatibilidad con la versión orignal de $AgentSpeak(L)$, requiere substituir las referencia a literal (*lit*) por átomo (*at*). El cuadro 9.1 resume la gramática de $AgentSpeak(L)$ aquí desarrollada. El ejemplo 9.2, muestra un programa $AgentSpeak(L)$ completo. *Gramática*

3 En contraposición con la negación tradicional o débil, denotada por ¬.

ag	::=	$bs \quad ps \quad gs$	
bs	::=	$b_1, \ldots, b_n$	$(n \geq 0)$
b	::=	$lit \mid rule$	$ground(lit)$
lit	::=	$at \mid \sim at$	
at	::=	$p \mid p(t_1, \ldots, t_n)$	$(p \in Pred, n \geq 1)$
	$\mid$	$at[s_1, \ldots, s_m]$	$(m > 1)$
s	::=	`percept` $\mid$ `self` $\mid id$	
$rule$	::=	lit :- fbf	
fbf	::=	$lit \mid \neg fbf \mid fbf \wedge fbf$	
t	::=	$v \mid c \mid f(t_1, \ldots, t_n)$	$(v \in Var, c \in Const, f \in Func, n \geq 1)$
ps	::=	$p_1, \ldots, p_n$	$(n \geq 1)$
p	::=	$te : ct \leftarrow h$	
te	::=	$+lit \mid -lit \mid +g \mid -g$	
ct	::=	$fbf \mid \top$	
h	::=	$h_1; \top \mid \top$	
h_1	::=	$a \mid u \mid g \mid h_1; h_1$	
a	::=	$ac(t_1, \ldots, t_n)$	$(ac \in Actn, n \geq 0)$
u	::=	$+b \mid -lit$	
gs	::=	$g_1, \cdots, g_n$	$(n \geq 0)$
g	::=	$!lit \mid ?lit$	

Cuadro 9.1: Gramática BNF de $AgentSpeak(L)$ para el lenguaje AOP *Jason*. Adaptada y extendida de Bordini, Hübner y Wooldridge [12]

9.1.4 Conjuntos, Pilas y Colas

Algunas definiciones que usaremos más adelante incluyen operaciones sobre conjuntos, pilas y colas. Para los conjuntos usaremos la notación y operaciones estándar; mientras que para pilas y colas haremos uso de las siguientes definiciones:

Definición 9.14 (Pila). *La expresión $p = [e_1 \ddagger e_2 \ddagger \ldots \ddagger e_n]$ denota una **pila** p de tamaño n. La pila vacía se denota como $\emptyset$. Se definen las siguientes operaciones sobre las pilas y sus elementos:* *Pila*

- $top(p) = e_1$.
- $pop(p) = e_1$ *y* $p = [e_2 \ddagger \ldots \ddagger e_n]$.
- $push(e, p) = [e \ddagger e_1 \ddagger e_2 \ddagger e_n]$.

Definición 9.15 (Cola). *La expresión $c = \{e_1 \ddagger e_2 \ddagger \ldots \ddagger e_n\}$ denota una **cola** c de tamaño n. La cola vacía se denota como $\emptyset$. Se definen las siguientes operaciones sobre las colas y sus elementos:* *Cola*

- $fst(c) = e_1$.
- $pop(c) = e_1$ *y* $c = \{e_2 \ddagger \ldots \ddagger e_n\}$.
- $push(e, c) = \{e_1 \ddagger e_2 \ddagger \ldots \ddagger e_n \ddagger e\}$.

Observen que $push/2$ y $pop/2$ son operaciones destructivas, las pilas y las colas son modificadas como resultado de su ejecución. Las operaciones $top/1$ y $fst/1$ no lo son.

9.2 SEMÁNTICA

La ejecución de un programa de agente $AgentSpeak(L)$ está determinado por una **se-** *Semántica operacional*

Ejemplo 9.2. *El siguiente código $AgentSpeak(L)$, tal y como se escribiría en su intérprete Jason [12]:*

```
// Agente cap04/src/asl/beto.asl

/* creencias iniciales y reglas */
factorial(1,1).
factorial(Num,Fact) :-
    Fact = Num * FactAux & factorial(Num-1,FactAux).

/* metas iniciales */
!start.

/* planes */
+!start <-
    ?factorial(5,F);
    .print("El factorial de 5 es ",F,".").
```

es un programa válido y ejecutable que computa el factorial de 5:

```
[beto] El factorial de 5 es 120.
```

Observen que el operador de conjunción se denota por &. La aritmética está predefinida, lo mismo que la acción `.print`.

mántica operacional, al estilo de las propuestas por Plotkin [86]. Estas semánticas se basan en un sistema de transiciones entre configuraciones de un programa. Una configuración puede verse como el estado de un agente en un momento dado:

Definición 9.16 (Sistema de transiciones)**.** *Un **sistema de transiciones** es una estructura $\langle \Gamma, \rightarrow \rangle$, donde Γ es un conjunto de elementos γ, conocidos como **configuraciones**, y $\rightarrow$ es una relación binaria sobre Γ, llamada **transición**.*

Sistema de transiciones

9.2.1 Configuraciones

En el caso de $AgentSpeak(L)$, una **configuración** $\gamma = \langle ag, C, M, T, s \rangle$ está compuesta por:

Configuración

- Un **programa** del agente $ag = \langle bs, ps, gs \rangle$ formado por las creencias bs, los planes ps y las metas gs del agente, tal y como se definen en la gramática del lenguaje (Ecuación 9.7) y se ilustra en el ejemplo 9.2.

  Programa del agente

- Una **circunstancia** del agente C es una tupla $\langle I, E, A \rangle$ donde: I es el conjunto de **intenciones** $\{i_1, i_2, \ldots, i_n\}$, tal que cada $i \in I$ es una pila de planes parcialmente instanciados; E es una cola de **eventos** $\{\langle te_1, i_1 \rangle, \langle te_2, i_2 \rangle, \ldots, \langle te_n, i_n \rangle\}$, tal que cada te es un evento disparador y cada i es una intención. Si $i = \top$ se dice que el evento es externo; en caso contrario el evento es interno, es decir, generado por una intención previa; y A es el conjunto de **acciones** a ser ejecutadas por el agente en el ambiente.

  Circunstancia
  Intenciones
  Eventos
  Acciones

- M es una tupla $\langle In, Out, SI \rangle$ donde In es el **buzón** del agente, Out es una lista de mensajes a ser enviados, e SI es un registro de intenciones suspendidas [4], es decir, aquellas que esperan un mensaje de respuesta para reanudarse. Los detalles sobre la comunicación se abordarán en la sección 9.3.

  Buzón

- T es una tupla de **registros** $\langle R, Ap, \iota, \epsilon, \rho \rangle$ donde R es el conjunto de **planes relevantes** dado cierto evento; $Ap \subseteq R$ es el conjunto de **planes aplicables**, aquellos que el agente cree poder ejecutar; ι, ϵ y ρ son, respectivamente, la intención, el evento, y el plan actualmente considerados en el razonamiento del agente.

  Registros

4 También se usará para guardar las intenciones suspendidas en espera de que una acción termine de ejecutarse.

- $s \in \{SelEv, RelPl, AppPl, SelAppl, SelInt, AddIM, ExecInt, ClrInt, ProcMsg\}$ denota una **etiqueta** para la configuración. *Etiqueta*

La relación de transición $\rightarrow$ se definen en términos de un conjunto de **reglas semánticas** con la forma: *Reglas semánticas*

$$(\textbf{regla}) \quad \frac{cond}{\gamma \rightarrow \gamma'}$$

donde la configuración γ puede transformarse en una nueva configuración γ', si las condiciones expresadas en *cond* se cumplen. Para definir las reglas semánticas, adoptaremos la siguiente notación:

- Para hacer referencia al componente E de una circunstancia C, escribimos C_E. De manera similar accedemos a los demás componentes de una configuración o del programa de agente. Por ejemplo, las creencias de un agente se denotan como ag_{bs}.
- Para indicar que no hay ninguna intención siendo considerada en la ejecución del agente, se emplea $T_i = \top$. De forma similar para T_ϵ, T_ρ y demás registros de una configuración.
- Se usa $i[p]$ para denotar que p es el plan en el tope de la intención i; también se puede usar $p = top(i)$.
- Si asumimos que p es un plan de la forma $te : ct \leftarrow h$, entonces:
  - $TrEv(p) = te$ denota el evento disparador de p.
  - $Ct(p) = ct$ denota el contexto de p.
  - $Head(p) = te : ct$ denota la cabeza de p.
  - $Body(p) = h$ denota el cuerpo de p.

Finalmente, se definen las siguientes funciones de selección de eventos $S_E = pop(C_E)$, planes aplicables $S_{Ap} = pop(T_{Ap})$ e intenciones $S_I = pop(C_I)$. Observen que estas funciones de selección son **destructivas**, en el sentido que el elemento seleccionado es eliminado de la pila, cola o conjunto donde se encontraba. *Selección destructiva*

9.2.2 Consecuencia lógica, planes relevantes y aplicables

Antes de definir las reglas semánticas que componen la relación de transición de la semántica operacional de $AgentSpeak(L)$, es necesario introducir algunas definiciones auxiliares cercanas a la programación lógica, por estar asociadas a los conceptos de unificador más general y consecuencia lógica. Estos conceptos se usarán para computar conjuntos planes relevantes, planes aplicables y la respuesta a las metas verificables del agente.

Unificador más general

El problema de **unificación** se define así [57]: Dados dos términos, ¿Existe una substitución que los haga sintácticamente iguales? *Unificación*

Ejemplo 9.3. *Consideren los siguientes términos, $factorial(Num, Fact)$ y $factorial(5, F)$, si substituimos Num por 5 y $Fact$ por F, conseguimos que ambos términos sean idénticos sintácticamente. En cambio, el término $factorial(1, 1)$ no se puede unificar con $factorial(5, F)$. No hay manera de substituir 1 por 5 para lograr que ambos términos sean idénticos.*

Formalicemos primero el concepto de substitución:

Definición 9.17 (Substitución). *Una **substitución** es un conjunto finito de la forma:* *Substitución*

$$\theta = \{t_1/v_1, \ldots, t_n/v_n\}, \quad (n \geq 0)$$

*donde las v_i son variables únicas y cada t_i es un término que no incluye a v_i. La forma t_i/v_i se conoce como **ligadura** [5] de v_i y expresa que el término t_i substituye a la variable v_i. La substitución vacía se denota por ϵ, y se conoce también como **substitución identidad**.* *Ligadura* *Substitución identidad*

Ejemplo 9.4. *Los siguientes conjuntos son substituciones válidas:*

- $\theta = \{f(Y)/X, Z/Y\}$
- $\theta' = \{a/X, b/Y, Y/Z\}$

En cambio, las siguientes no lo son:

- $\theta = \{f(X)/X, Z/Y\}$
- $\theta' = \{a/X, b/Y, Y/X\}$

Las substituciones pueden verse también como funciones con dominio y rango en el conjunto de términos del lenguaje, vía su aplicación:

Definición 9.18 (Aplicación de substitución). *La **aplicación** de la substitución θ al término t se denota $t\theta$ y resulta en un nuevo término donde todas las ligaduras de la substitución se han llevado a cabo en t de forma paralela.* *Aplicación*

Ejemplo 9.5. *Sea la substitución $\theta = \{5/Num, F/Fact\}$ y el término $t = factorial(Num, Fact)$, $t\theta = factorial(5, F)$. En tanto que, $t\epsilon = factorial(Num, Fact)$, de ahí el nombre de substitución identidad.*

Las substituciones pueden ser sujeto de la operación de composición para crear nuevas substituciones:

Definición 9.19 (Composición). *La **composición** de dos substituciones, $\theta = \{t_1/v_1, \ldots, t_n/v_n\}$ y $\theta' = \{t'_1/v'_1, \ldots, t'_m/v'_m\}$, es a su vez una substitución, definida de la siguiente manera:* *Composición*

$$\theta \circ \theta' = \{t_1\theta'/v_1, \ldots, t_n\theta'/v_n\} \cup t'_i/v'_i \in \theta' \mid v'_i \text{ no ocurre en } \theta$$

Eliminando todas las ligaduras de la forma t/t.

Ejemplo 9.6. *Sean $\theta = \{g(X, Y)/W\}$ y $\theta' = \{a/X, b/Y, c/Z\}$ dos substituciones, la composición $\theta \circ \theta' = \{g(a, b)/W, a/X, b/Y, c/Z\}$.*

Con los elementos anteriores podemos formalizar el concepto de unificador y unificador más general:

Definición 9.20 (Unificador más general). *Sean t_1 y t_2 términos. Se dice que la substitución θ es un **unificador** de esos términos si $t_1\theta = t_2\theta$. Una substitución θ se dice más general que una substitución σ, si y sólo si existe una substitución τ tal que $\sigma = \theta \circ \tau$. Un unificador θ se dice el **unificador más general** (umg) de dos términos, si y sólo si es más general que cualquier otro unificador entre esos términos.* *Unificador* *Unificador más general*

Ejemplo 9.7. *Consideren los términos $f(X)$ y $f(g(Y))$, el umg de ellos es $\theta = \{g(Y)/X\}$, pero existen otros muchos unificadores, por ejemplo $\{g(a)/X, a/Y\}$. Intuitivamente, el umg de dos términos es el más simple de todos sus unificadores. Observen que $f(X)\theta = f(g(Y))\theta = f(g(Y))$.*

Definir un algoritmo para computar el *umg* no está dentro de los objetivos de este curso. Para los interesados, mi curso de Programación para la Inteligencia Artificial [6], incluye un algoritmo para ello (Secciones 6.3 y 6.4 del curso, en el archivo ia2-notas-06). Para los más interesados, Knight [57] nos ofrece una revisión del problema de unificación en las ciencias de la computación y diversos algoritmos para su cómputo.

En lo que sigue, asumiremos que el *umg* está definido, como de hecho lo está en *Jason*. Ahora bien, los términos en este lenguaje AOP están **anotados** y esto debe tomarse en cuenta al computar la unificación. En general, dos términos $t_1[a_1, \ldots, a_n]$ y $t_2[a'_2, \ldots, a'_m]$ unifican, si y sólo si (i) existe un umg θ tal que $t_1\theta = t_2\theta$ y (ii) $\{a_1, \ldots, a_n\} \subseteq \{a'_1, \ldots, a'_m\}$. Esto es, si las fuentes de t_1 están incluidas en las de t_2.

Unificación con anotaciones

Ejemplo 9.8. *Consideremos algunos ejemplos del umg en Jason:*

- $umg(p(c)[s_1], p(c)[s_1, s_2]) = \top$ *debido a que no hay variables ligadas y las anotaciones del primer término están incluidas en las del segundo;*
- $umg(p(c)[a_1], p(c)[a_2])$ *falla debido a que la anotaciones del primer término no están incluidas en las anotaciones del segundo.*
- $umg(p(X)[a_2], p(c)[a_1, a_2, a_3]) = \{c/X\}$. *Ahora el unificador incluye una ligadura.*

Consecuencia lógica

La relación de consecuencia lógica estable que una fórmula bien formada se sigue de las creencias del agente:

Definición 9.21 (Consecuencia Lógica)**.** *La expresión $ag_{bs} \models fbf$ denota que la fórmula bien formada fbf es **consecuencia lógica** de las creencias del agente ag. Siguiendo la definición sintáctica de una fbf (Ecuación 9.6), tenemos que:*

Consecuencia lógica

- $ag_{bs} \models lit[s_1, \ldots, s_n]$, *si y sólo si, existe una literal* $lit'[s'_1, \ldots s'_m] \in ag_{bs}$, *tal que* $lit\theta = lit'$. θ *se conoce como unificador de respuesta.*
- $ag_{bs} \models \neg fbf$, *si y sólo si,* $ag_{bs} \not\models fbf$, *tal que* $\theta = \epsilon$.
- $ag_{bs} \models fbf_1 \wedge fbf_2\ \theta$, *si y sólo si,* $ag_{bs} \models fbf_1\ \theta_1$ *y* $ag_{bs} \models fbf_2\ \theta_2$. *El unificador de respuesta* $\theta = \theta_1 \circ \theta_2$.

Si consideramos a las reglas como parte de las creencias, debemos agregar el siguiente caso:

- $ag_{bs} \models lit[s_1, \ldots, s_n]$, si y sólo si, en las creencias del agente ag, existe una regla $lit'[s'_1, \ldots, s'_n] \text{ :- } fbf$ tal que: (i) $lit\theta' = lit'\theta'$ y (ii) $ag_{bs} \models fbf$ con substitución de respuesta θ''. $\theta = \theta' \circ \theta''$.

Ejemplo 9.9. $p(X)[a_1]$ *se sigue de* $ag_{bs} = \{p(t)[a_1, a_2]\}$; *pero* $p(X)[a_1, a_2]$ *no se sigue de* $ag_{bs} = \{p(t)[a_1]\}$.

La distribución de Jason incluye información complementaría sobre la unificación y las anotaciones [7]. Finalmente, definiremos una función auxiliar para obtener la substitución de respuesta cuando una fbf es consecuencia lógica de las creencias del agente:

Definición 9.22 (Substitución de Respuesta)**.** *La **substitución de respuesta** para una fbf, es un unificador θ tal que la aplicación de θ a la fbf, se sigue de las creencias del agente. Formalmente:*

Substitución de Respuesta

$$SubstResp(fbf) \quad = \quad \theta \mid ag_{bs} \models fbf\theta \tag{9.21}$$

Planes relevantes y aplicables

Con las definiciones anteriores podemos computar los conjuntos de planes relevantes y aplicables para un evento dado. Intuitivamente, los planes relevantes son aquellos que son útiles para contender con el evento; mientras que los aplicables son aquellos planes relevantes que pueden ejecutarse en ese momento, de acuerdo a las creencias del agente.

Definición 9.23 (Planes relevantes). *El conjunto de los **planes relevantes** para un evento $\langle te, i \rangle \in C_E$, está compuesto por los planes definidos para contender con tal evento. Formalmente, aquellos planes cuyo evento disparador unifica con te:*

Planes relevantes

$$PRels(te) \quad = \quad \{p\theta \mid p \in ag_{ps} \wedge \theta = umg(te, TrEv(p))\} \tag{9.22}$$

Definición 9.24 (Planes aplicables). *El conjunto de los **planes aplicables** es un subconjunto de los planes relevantes que el agente cree poder ejecutar, es decir, aquellos cuyo contexto es consecuencia lógica de las creencias del agente. Formalmente:*

Planes aplicables

$$PApls(PRels) \quad = \quad \{p\theta \mid p \in PRels \wedge ag_{bs} \models Ct(p)\theta\} \tag{9.23}$$

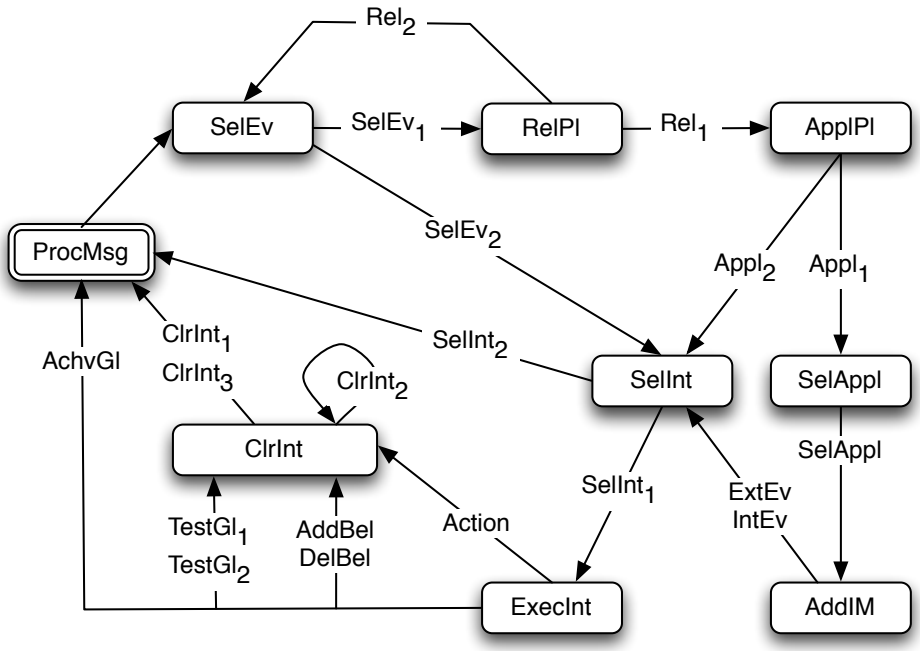

Figura 9.2: El ciclo de razonamiento de Jason. Adaptado de Guerra-Hernández, Castro-Manzano y El-Fallah-Seghrouchni [50].

9.2.3 Reglas semánticas

La relación de transición entre configuraciones de $AgentSpeak(L)$ se define mediante un conjunto de reglas semánticas que inducen el **ciclo de razonamiento** mostrado en la figura 9.2. Las etiquetas de los nodos denotan la configuración correspondiente. Los arcos están etiquetados con los identificadores de las reglas de transición que definiremos en esta sección.

Ciclo de razonamiento

El estado inicial del ciclo de razonamiento es *ProcMsg*, una configuración donde se actualizan los eventos que el agente recibe vía su percepción y la comunicación. Conceptualmente, el ciclo de razonamiento puede dividirse en dos fases, con respecto a las intenciones:

- Formación. Comprende los caminos entre *ProcMsg* y *AddIM*.
- Ejecución. Comprende los caminos entre *SelInt* y *ProcMsg*.

5 En algunos textos se usa la notación invertida $v_i \backslash t_i$, o el mapeo $t_i \mapsto v_i$.
6 http://www.uv.mx/personal/aguerra/pia/
7 /doc/tech/annotations.pdf

De forma que un agente está continuamente tratando de formar nuevas intenciones, en respuesta a los eventos percibidos, y ejecutando las formadas con anterioridad.

Los eventos percibidos son agregados a la cola C_E. Las reglas que llevan a cabo la **selección de eventos** para su procesamiento, son como sigue. Si la cola de eventos no está vacía, un evento es seleccionado y agregado al registro correspondiente de la configuración, que pasa de seleccionar eventos a computar planes relevantes:

Selección de evento

$$(\mathbf{SelEv_1}) \quad \frac{C_E \neq \emptyset, S_E = \langle te, i\rangle}{\langle ag, C, M, T, SelEv\rangle \rightarrow \langle ag, C', M, T', RelPl\rangle} \quad \text{t.q. } T'_\epsilon = \langle te, i\rangle \tag{9.24}$$

Recuerden que $S_E = pop(C_E)$, elimina el evento seleccionado de C_E, de forma que la circunstancia del agente cambia, al igual que el registro de evento seleccionado. Todas las funciones de selección tienen un comportamiento análogo a éste. En caso de que la cola de eventos esté vacía, simplemente se procede a iniciar la segunda fase del ciclo de razonamiento, seleccionando una intención para su ejecución:

$$(\mathbf{SelEv_2}) \quad \frac{C_E = \emptyset}{\langle ag, C, M, T, SelEv\rangle \rightarrow \langle ag, C, M, T, SelInt\rangle} \tag{9.25}$$

Los **planes relevantes** se computan a partir del evento seleccionado. El conjunto de planes relevantes computados se guarda en el registro correspondiente, y se procede a computar quien de ellos es aplicable:

Planes relevantes

$$(\mathbf{Rel_1}) \quad \frac{T_\epsilon = \langle te, i\rangle, PRels(te) \neq \emptyset}{\langle ag, C, M, T, RelPl\rangle \rightarrow \langle ag, C, M, T', AppPl\rangle} \quad \text{t.q. } T'_R = PRels(te) \tag{9.26}$$

Si no existen planes relevantes, el ciclo de razonamiento regresa a seleccionar otro evento:

$$(\mathbf{Rel_2}) \quad \frac{T_\epsilon = \langle te, i\rangle, PRels(te) = \emptyset}{\langle ag, C, M, T, RelPl\rangle \rightarrow \langle ag, C, M, T, SelEv\rangle} \tag{9.27}$$

El caso de los **planes aplicables** es parecido. Si hay planes aplicables, se actualiza el registro correspondiente y se procede a seleccionar uno de ellos para formar una intención:

Planes aplicables

$$(\mathbf{Appl_1}) \quad \frac{PApls(T_R) \neq \emptyset}{\langle ag, C, M, T, ApplPl\rangle \rightarrow \langle ag, C, M, T', SelAppl\rangle} \quad \text{t.q. } T'_{Ap} = PApls(T_R) \tag{9.28}$$

Puesto que al computar si un plan p es aplicable, generamos un unificador de respuesta θ, el registro T_{Ap} almacena conjuntos de pares (p, θ). Si no hay planes aplicables, se procede a seleccionar una intención para su ejecución:

$$(\mathbf{Appl_2}) \quad \frac{PApls(T_R) = \emptyset}{\langle ag, C, M, T, ApplPl\rangle \rightarrow \langle ag, C, M, T, SelInt\rangle} \tag{9.29}$$

La siguiente regla **selecciona** un plan aplicable: *Selección de plan aplicable*

$$(\textbf{SelAppl}) \quad \frac{S_{Ap} = (p, \theta)}{\langle ag, C, M, T, SelAppl\rangle \rightarrow \langle ag, C, M, T', AddIM\rangle} \quad \text{t.q. } T'_\rho = (p, \theta) \tag{9.30}$$

$AgentSpeak(L)$ distingue dos tipos de eventos: Internos y externos. Los primeros tienen su origen en una intención; mientras que los segundos lo tienen en una percepción. Esto influye la manera como se **crea** una nueva intención. Los eventos externos crean una intención totalmente nueva con el plan aplicable seleccionado: *Creación de una Intención*

$$(\textbf{ExtEv}) \quad \frac{T_\epsilon = \langle te, \top\rangle, T_\rho = (p, \theta)}{\langle ag, C, M, T, AddIM\rangle \rightarrow \langle ag, C', M, T, SelInt\rangle} \quad \text{t.q. } C'_I = C_I \cup \{[p\theta]\} \tag{9.31}$$

Los eventos internos, dada su naturaleza, apilan el plan relevante y aplicable seleccionado, en la Intención previa que generó el evento en cuestión:

$$(\textbf{IntEv}) \quad \frac{T_\epsilon = \langle te, i\rangle, T_\rho = (p, \theta)}{\langle ag, C, M, T, AddIM\rangle \rightarrow \langle ag, C', M, T, SelInt\rangle} \quad \text{t.q. } C'_I = C_I \cup \{i[p\theta]\} \tag{9.32}$$

Observen que en este caso, la intención i debe agregarse nuevamente en C_I con p en su tope. Esto tiene que ver con el concepto de **intención suspendida**, del que hablaremos al revisar las reglas para resolver las metas alcanzables. La forma exacta en que una Intención se suspende se verá más adelante, al definir las reglas de ejecución de una Intención. *Intención suspendida*

Las reglas para **seleccionar** una intención a ser ejecutada son como sigue. Si hay Intenciones en la circunstancia del agente, se selecciona una para guardarla en el registro correspondiente: *Selección Intención*

$$(\textbf{SelInt}_\textbf{1}) \quad \frac{C_I \neq \emptyset, S_I = i}{\langle ag, C, M, T, SelInt\rangle \rightarrow \langle ag, C, M, T', ExecInt\rangle} \quad \text{t.q. } T'_\iota = i \tag{9.33}$$

Si no hay Intenciones en la circunstancia del agente, se vuele al inicio del ciclo de razonamiento para actualizar los eventos percibidos por el agente:

$$(\textbf{SelInt}_\textbf{2}) \quad \frac{C_I = \emptyset}{\langle ag, C, M, T, SelInt\rangle \rightarrow \langle ag, C, M, T, ProcMsg\rangle} \tag{9.34}$$

El grupo de reglas que describiremos a continuación, expresa el efecto de la ejecución de los planes. El plan siendo ejecutado es siempre aquel que se encuentra en el tope de la intención que ha sido previamente seleccionada. Todas las reglas en este grupo terminan descartando i, por lo que otra intención puede ser seleccionada posteriormente. Las reglas se ejecutan dependiendo del componente del cuerpo del plan que se ha seleccionado.

La siguiente regla aplica cuando el primer elemento del cuerpo del plan que está en el tope de la Intención seleccionada, es una **acción**. En ese caso la acción es *Ejecución de una acción*

almacenada en el registro correspondiente:

$$(\textbf{Action}) \quad \frac{T_\iota = i[head \leftarrow a; h]}{\langle ag, C, M, T, ExecInt\rangle \rightarrow \langle ag, C', M, T', ClrInt\rangle} \tag{9.35}$$
$$\text{t.q. } C'_A = C_A \cup \{a\}, T'_\iota = i[head \leftarrow h], \quad C'_I = (C_I \backslash \{T_\iota\}) \cup \{T'_\iota\}$$

La siguiente regla aplica cuando el primer elemento del cuerpo del plan que está en el tope de la Intención seleccionada, es una **meta alcanzable**:

Ejecución de una meta alcanzable

$$(\textbf{AchvGl}) \quad \frac{T_\iota = i[head \leftarrow !at; h]}{\langle ag, C, M, T, ExecInt\rangle \rightarrow \langle ag, C', M, T, ProcMsg\rangle} \tag{9.36}$$
$$\text{t.q. } C'_E = C_E \cup \{\langle +!at, T_\iota\rangle\}, \quad C'_I = C_I \setminus \{T_\iota\}$$

Observe como la Intención que generó el evento interno es removida del conjunto de Intenciones C_I. Esto implementa la **suspención** de una Intención. Sólo cuando el curso de acción que se haya definido para procesar el evento generado haya concluído, se puede continuar con la ejecución de la intención que había sido suspendida [8].

Intención suspendida

Las **metas verificables**, se procesan mediante las siguientes dos reglas:

Ejecución meta verificable

$$(\textbf{TestGl}_\mathbf{1}) \quad \frac{T_\iota = i[head \leftarrow ?at; h], Test(ag_{bs}, at) = at\theta}{\langle ag, C, M, T, ExecInt\rangle \rightarrow \langle ag, C', M, T, ClrInt\rangle} \tag{9.37}$$
$$\text{t.q. } T'_\iota = i[(head \leftarrow h)\theta], \quad C'_I = (C_I \backslash \{T_\iota\}) \cup \{T'_\iota\}$$

$$(\textbf{TestGl}_\mathbf{2}) \quad \frac{T_\iota = i[head \leftarrow ?at; h], Test(ag_{bs}, at) = \bot}{\langle ag, C, M, T, ExecInt\rangle \rightarrow \langle ag, C', M, T, ClrInt\rangle} \tag{9.38}$$
$$\text{t.q. } C'_E = C_E \cup \{\langle -?at, T_\iota\rangle\}, \quad C'_I = C_I \backslash \{T_\iota\}$$

Observen que si la consecuencia lógica, computada por $Test/2$, **falla**; la intención en si no falla, sino que genera un evento ($TestGl_2$) para que sea procesado por un plan posteriormente. Esto introduce otra diferencia con la Programación Lógica, pensada para permitir consultas más elaboradas que las que *Test* puede computar.

Fallo meta verificable

Al igual que en dMARS [33] y a diferencia de $AgentSpeak(L)$, los agentes en Jason pueden **agregar o eliminar creencias** durante la ejecución de sus planes. Las siguientes reglas se encargan de ello:

Actualización creencias

$$(\textbf{AddBel}) \quad \frac{T_\iota = i[head \leftarrow +b; h]}{\langle ag, C, M, T, ExecInt\rangle \rightarrow \langle ag', C', M, T', ClrInt\rangle} \tag{9.39}$$
$$\text{t.q. } ag'_{bs} = ag_{bs} + b_{[self]}, \quad C'_E = C_E \cup \{\langle +b_{[self]}, \top\rangle\}, \quad T'_\iota = i[head \leftarrow h], \quad C'_I = (C_I \backslash \{T_\iota\}) \cup \{T'_\iota\}$$

8 En Jason, el operador !! evita la suspensión de las intenciones, creando una intención nueva, como si se tratase de una respuesta a un evento externo.

$$(\textbf{DelBel}) \quad \frac{T_\iota = i[head \leftarrow -at; h]}{\langle ag, C, M, T, ExecInt\rangle \rightarrow \langle ag', C', M, T', ClrInt\rangle}$$

$$\begin{aligned} \text{t.q. } ag'_{bs} &= ag_{bs} - at_{[self]}, \\ C'_E &= C_E \cup \{\langle -at_{[self]}, \top\rangle\}, \\ T'_\iota &= i[head \leftarrow h], \\ C'_I &= (C_I \backslash \{T_\iota\}) \cup \{T'_\iota\} \end{aligned} \tag{9.40}$$

Observen que aunque los eventos generados por estas operaciones tienen como origen una intención, estos son tratados como si fuesen **eventos externos**. Esto es, la intención que genero estos eventos no se suspende. *Excepción*

Para concluir con la semántica operacional de Jason se definen tres reglas más, las llamadas *clearing house rules*. $ClearInt_1$ simplemente remueve una intención del conjunto de intenciones de un agente cuando no hay más que hacer al respecto, es decir: Ya no quedan más acciones, metas o actualizaciones que ejecutar en el cuerpo del plan único en la intención. $ClearInt_3$ se encarga del caso trivial donde la limpieza no debe realizarse aún, así que el ciclo de razonamiento pasa a la configuración $ProcMsg$. La regla $ClearInt_2$ contiende con el caso donde el plan en el tope de la intención tiene un cuerpo vacío, pero la intención tiene más planes. En este caso el ciclo de razonamiento regresa a $ClrInt$ porque pueden ser necesarias más labores de limpieza. *Limpieza*

$$(\textbf{ClrInt}_\textbf{1}) \quad \frac{T_\iota = [head \leftarrow \top]}{\langle ag, C, M, T, ClrInt\rangle \rightarrow \langle ag, C', M, T, ProcMsg\rangle} \qquad \text{t.q. } C'_I = C_I \setminus \{T_\iota\} \tag{9.41}$$

$$(\textbf{ClrInt}_\textbf{2}) \quad \frac{T_\iota = i[head \leftarrow \top]}{\langle ag, C, M, T, ClrInt\rangle \rightarrow \langle ag, C', M, T, ClrInt\rangle}$$

$$\begin{aligned} \text{t.q. } C'_I = (C_I \setminus \{T_\iota\}) \ &\cup \{k[(head' \leftarrow h)\theta]\} \\ \text{si } i &= k[head' \leftarrow g; h] \\ \text{y } g\theta &= TrEv(head) \end{aligned} \tag{9.42}$$

$$(\textbf{ClrInt}_\textbf{3}) \quad \frac{T_\iota \neq [head \leftarrow \top] \wedge T_\iota \neq i[head \leftarrow \top]}{\langle ag, C, M, T, ClrInt\rangle \rightarrow \langle ag, C, M, T, ProcMsg\rangle} \tag{9.43}$$

9.3 ACTOS DE HABLA

Vieira y col. [107] han propuesto una semántica extendida de $AgentSpeak(L)$ para implementar un lenguaje de comunicación basado en Actos de habla, para Jason. Debido a una literatura más abundante, pero sobre todo por razones históricas, se optó por adoptar KQML en esta tarea. La semántica de KQML fue propuesta por Labrou y Finin [61], con base en los trabajos de Perrault y Allen [83] y Cohen y Levesque [26]. La idea central consiste en especificar operadores tipo STRIPS que definen las las **condiciones** preparatorias, posteriores y de satisfacción de cada performativa. *KQML* *Condiciones KQML*

Ejemplo 9.10. *Para tell tendríamos:*

- *Condiciones preparatorias:*
  - $Pre(S) : bel(S, X) \wedge know(S, want(R, know(R, bel(S, X))))$
  - $Pre(R) : intend(R, know(R, bel(S, X)))$
- *Condiciones posteriores:*
  - $Pos(S) : know(S, know(R, bel(S, X)))$
  - $Pos(R) : know(R, bel(S, X))$
- *Satisfacción:*
  - $know(R, bel(S, X))$

donde las condiciones preparatorias se leen así: el emisor del mensaje S cree X y sabe que el receptor del mensaje R quiere saber si ese es el caso; mientras que R intenta saber si ese es el caso, si S cree X. El resto de las condiciones puede leerse de manera similar.

No es de extrañar que la implementación y verificación de tal implementación sean difíciles de lograr. La abstracción Intencional oculta el nivel de diseño en tal especificación. Afortunadamente, en el caso de $AgentSpeak(L)$ extendido, podemos recurrir a las anotaciones para definir una semántica de los actos de habla. En lo que sigue, asumiremos la sintaxis y la semántica operacional definidas previamente y las extenderemos como sigue.

9.3.1 Sintaxis

Asumiremos la existencia de una acción especial predefinida para enviar mensajes: `.send(Id,Ilf,Cnt)`, donde Id es la identidad del destinatario, ilf es la fuerza ilocutoria del mensaje y Cnt su contenido.

9.3.2 Semántica

En realidad, los **mensajes** tienen como soporte una tupla $\langle mid, dest, fil, cnt \rangle$, donde: *Mensajes*

- mid es un **identificador** único del mensaje, generado automáticamente.
- $dest$ es el nombre del agente **destinatario**.
- $fil \in \{Tell, Untell, Achieve, Unachieve, TellHow, UntellHow, AskOne, AskAll\}$ es la **fuerza ilocutoria** del mensaje.
- cnt es el **contenido** del mismo. Dependiendo de la fuerza performativa del mensaje, su contenido puede ser: una fórmula atómica (at), o un conjunto de fórmulas atómicas (ATs), o una creencia (b), es decir una literal de base, o un conjunto de creencias (Bs), o un conjunto de planes (PLs).

Toda configuración de la semántica operacional provee un mecanismo **asíncrono** para el envío y recepción de mensajes con su fuerza performativa. Recuerden que la **circunstancia** del un agente incluye la estructura M, un conjunto de **buzones** que incluye uno de de **entrada** M_{In}, uno de **salida** M_{Out} y uno de **intenciones suspendidas** M_{SI}. En realidad, el buzón de intenciones suspendidas guarda pares de la forma $\langle mid, i \rangle$, donde mid es el identificador del mensaje que causó que la intención i fuera suspendida en espera de la respuesta solicitada. Los mensajes se almacenan en un buzón y uno de ellos es procesado cada vez que el agente inicia un ciclo de razonamiento. *Buzones*

Cuando un agente envía una **solicitud de información**, se ha optado por suspender la intención asociada hasta recibir la respuesta solicitada. Esto garantiza que el *Solicitud de información*

agente tenga la información solicitada antes de continuar ejecutando el cuerpo del plan que incluye la solicitud de información. Se asume que la información recibida pasa a formar parte de las **creencias** del agente, por lo que una meta verificable es necesaria para poderla usar posteriormente.

A continuación definimos dos reglas semánticas para ejecutar la acción interna `.send`. Estas reglas pueden verse como especializaciones de la regla *Action* que ejecuta acciones en los planes de los agentes y tienen prioridad sobre ésta. La primer regla contiende con solicitudes de información y por tanto, suspende intenciones:

Envío de mensajes

$$(\textbf{ExecActSndAsk}) \quad \frac{\begin{array}{c} T_\iota = i[head \leftarrow .send(id, ilf, cnt); h], \\ ilf \in \{AskOne, AskAll, AskHow\} \end{array}}{\langle ag, C, M, T, ExecInt\rangle \rightarrow \langle ag, C', M', T, ProcMsg\rangle} \tag{9.44}$$

$$\begin{array}{rl} \text{Donde:} & M'_{Out} = M_{Out} \cup \{\langle mid, id, ilf, cnt\rangle\} \\ & M'_{SI} = M_{SI} \cup \{(mid, i[head \leftarrow h])\} \\ & C'_I = C_I \backslash \{T_\iota\} \end{array}$$

Cuando la fuerza performativa es de otro tipo, simplemente se forma el mensaje en el buzón de salida; pero el siguiente paso en el razonamiento en ese caso es *ClrInt*, en lugar de pasar directamente a *ProcMsg*:

$$(\textbf{ExecActSnd}) \quad \frac{\begin{array}{c} T_\iota = i[head \leftarrow .send(id, ilf, cnt); h], \\ ilf \notin \{AskOne, AskAll, AskHow\} \end{array}}{\langle ag, C, M, T, ExecInt\rangle \rightarrow \langle ag, C', M', T, ClrInt\rangle} \tag{9.45}$$

$$\begin{array}{rl} \text{Donde:} & M'_{Out} = M_{Out} \cup \{\langle mid, id, ilf, cnt\rangle\} \\ & C'_I = (C_I \backslash \{T_\iota\}) \cup \{i[head \leftarrow h]\} \end{array}$$

Siempre que un mensaje nuevo es enviado, asumimos que el sistema crea un nuevo identificador *mid* único. Más adelante veremos que al contestar a un mensaje, el mismo *mid* se mantiene en la respuesta obtenida, de forma que un agente pueda saber que cierto mensaje le llegó en respuesta a un mensaje suyo.

Necesitamos una nueva función de **selección** $\mathcal{S}_M$ que opere sobre los buzones. Analogamente a las otras funciones de selección en Jason, la función regresa el primer mensaje en el buzón que recibe como parámetro. Necesitamos también una función booleana $SocAcc(id, ilf, at)$ donde ilf es la fuerza ilocutoria del mensaje del agente id con contenido proposicional at, que determina si el mensaje es **socialmente aceptable** en un contexto dado. Estas funciones pueden ser reprogramadas al igual que las otras funciones de selección del lenguaje.

Función de selección

Socialmente aceptable

A continuación definimos la regla asociada a la recepción de un mensaje cuya fuerza performativa es *Tell*, como en un informe no solicitado:

Tell

$$(\textbf{Tell}) \quad \frac{\begin{array}{c} \mathcal{S}_M(M_{In}) = \langle mid, id, Tell, Bs\rangle, \\ \forall i\ (mid, i) \notin M_{SI} \\ SocAcc(id, Tell, Bs) \end{array}}{\langle ag, C, M, T, ProcMsg\rangle \rightarrow \langle ag', C', M', T, SelEv\rangle} \tag{9.46}$$

$$\begin{array}{rl} \text{Donde:} & M'_{In} = M_{In} \backslash \{\langle mid, id, Tell, Bs\rangle\} \\ & \forall b\ b \in Bs : ag'_{bs} = ag_{bs} + b[id] \\ & C_{E'} = C_E \cup \{\langle +b[id], \top\rangle\} \end{array}$$

La otra posibilidad es recibir el informe con fuerza performativa *Tell*, en respuesta a una solicitud previa (existe una intención suspendida asociada al *mid*):

$$\begin{array}{rl} & \mathcal{S}_M(M_{In}) = \langle mid, id, Tell, Bs\rangle, \\ & \exists i\ (mid, i) \in M_{SI}, \\ (\textbf{TellRepl}) & \dfrac{SocAcc(id, Tell, Bs)}{\langle ag, C, M, T, ProcMsg\rangle \rightarrow \langle ag', C', M', T, SelEv\rangle} \end{array}$$

$$\begin{array}{rl} \text{Donde: } & M'_{In} = M_{In} \backslash \{\langle mid, id, Tell, Bs\rangle\} \\ & M'_{SI} = M_{SI} \backslash \{(mid, i)\} \\ & C'_I = C_I \cup \{i\} \\ & \forall b\ b \in Bs : ag'_{bs} = ag_{bs} + b[id] \\ & C_{E'} = C_E \cup \{\langle +b[id], \top\rangle\} \end{array} \tag{9.47}$$

La regla asociada a recibir un mensaje con fuerza performativa *Untell* que no responde a una solicitud previa, es: *Untell*

$$\begin{array}{rl} & \mathcal{S}_M(M_{In}) = \langle mid, id, Untell, ATs\rangle, \\ & \forall i\ (mid, i) \notin M_{SI}, \\ (\textbf{Untell}) & \dfrac{SocAcc(id, Tell, ATs)}{\langle ag, C, M, T, ProcMsg\rangle \rightarrow \langle ag', C', M', T, SelEv\rangle} \end{array}$$

$$\begin{array}{rl} \text{Donde: } & M'_{In} = M_{In} \backslash \{\langle mid, id, Tell, ATs\rangle\} \\ & \forall b.\ b \in \{at\theta\ |\theta \in Test(ag_{bs}, at) \wedge at \in ATs\} : \\ & \quad ag'_{bs} = ag_{bs} - b[id] \\ & C_{E'} = C_E \cup \{\langle -b[id], \top\rangle\} \end{array} \tag{9.48}$$

Observen que el contenido de estos mensajes es una fórmula atómica por lo que puede incluir variables para las cuales será necesario computar sus posibles valores mediante unificación (*Test*). En el caso que el emisor del mensaje sea la única fuente de la creencias, esta es removida de las creencias del receptor. En otro caso, solo la fuente en la etiqueta es eliminada. En caso de que el mensaje se de en respuesta a una solicitud previa, la regla que aplica es la siguiente:

$$\begin{array}{rl} & \mathcal{S}_M(M_{In}) = \langle mid, id, Untell, ATs\rangle \\ & \exists i\ (mid, i) \in M_{SI} \\ (\textbf{UntellRepl}) & \dfrac{SocAcc(id, Untell, ATs)}{\langle ag, C, M, T, ProcMsg\rangle \rightarrow \langle ag', C', M', T, SelEv\rangle} \end{array}$$

$$\begin{array}{rl} \text{Donde: } & M'_{In} = M_{In} \backslash \{\langle mid, id, Tell, ATs\rangle\} \\ & M'_{SI} = M_{SI} \backslash \{(mid, i)\} \\ & C'_I = C_I \cup \{i\} \\ & \forall b.\ b \in \{at\theta\ |\theta \in Test(ag_{bs}, at) \wedge at \in ATs\} : \\ & \quad ag'_{bs} = ag_{bs} - b[id] \\ & C_{E'} = C_E \cup \{\langle -b[id], \top\rangle\} \end{array} \tag{9.49}$$

En el contexto social adecuado, es posible que un agente reciba un mensaje pidiéndole satisfacer una meta, por ejemplo, cuando el emisor tiene poder sobre el receptor. La regla que regula la recepción de este tipo de mensajes es:

$$(\textbf{Achieve}) \quad \frac{\mathcal{S}_M(M_{In}) = \langle mid, id, Achieve, at\rangle \quad SocAcc(id, Achieve, at)}{\langle ag, C, M, T, ProcMsg\rangle \rightarrow \langle ag, C', M', T, SelEv\rangle} \tag{9.50}$$

$$\begin{aligned} \text{Donde:} \quad & M'_{In} = M_{In} \backslash \{\langle mid, id, Achieve, at\rangle\} \\ & C'_E = C_E \cup \{+!at, \top\} \end{aligned}$$

Observen que la recepción de un mensaje socialmente aceptable con fuerza ilocutoria *Achieve* crea una nueva pila en las intenciones, al igual que lo hacen los eventos externos. Esto puede usarse como un mecanismo de focus de atención. Para los mensajes de fuerza ilocutoria *Unachieve* se usa la acción interna estándar `.drop_desire`.

Una extensión interesante de Jason a los actos de habla tradicionales, es la consideración del concepto de *know-how* [101], lo cual implica que la librería de planes de un agente no es estática –puede agregar planes que otro agente le comunica y eliminar planes a petición de otro agente. La recepción de mensajes con fuerza ilocutoria *TellHow* se regula por la siguiente regla:

$$(\textbf{TellHow}) \quad \frac{\mathcal{S}_M(M_{In}) = \langle mid, id, TellHow, PLs\rangle \quad \exists i\ (mid, i) \in M_{SI} \quad SocAcc(id, TellHow, PLs)}{\langle ag, C, M, T, ProcMsg\rangle \rightarrow \langle ag', C', M', T, SelEv\rangle} \tag{9.51}$$

$$\begin{aligned} \text{Donde:} \quad & M'_{In} = M_{In} \backslash \{\langle mid, id, TellHow, at\rangle\} \\ & M'_{SI} = M_{SI} \backslash \{(mid, i)\} \\ & C'_I = C_I \cup \{i\} \\ & ag'_{ps} = ag_{ps} \cup PLs \end{aligned}$$

La regla para los mensajes con fuerza ilocutoria *UntellHow* es muy similar:

$$(\textbf{UntellHow}) \quad \frac{\mathcal{S}_M(M_{In}) = \langle mid, id, UntellHow, PLs\rangle \quad SocAcc(id, UntellHow, PLs)}{\langle ag, C, M, T, ProcMsg\rangle \rightarrow \langle ag', C', M', T, SelEv\rangle} \tag{9.52}$$

$$\begin{aligned} \text{Donde:} \quad & M'_{In} = M_{In} \backslash \{\langle mid, id, UntellHow, at\rangle\} \\ & M'_{SI} = M_{SI} \backslash \{(mid, i)\} \\ & ag'_{ps} = ag_{ps} - PLs \end{aligned}$$

Para terminar de ilustrar la semántica de los actos de habla, consideremos la requisición de información mediante un mensaje con fuera ilocutoria *AskOne*:

$$(\textbf{AskOne}) \quad \frac{\mathcal{S}_M(M_{In}) = \langle mid, id, AskOne, \{at\}\rangle \quad SocAcc(id, AskOne, \{at\})}{\langle ag, C, M, T, ProcMsg\rangle \rightarrow \langle ag, C, M', T, SelEv\rangle}$$

$$\text{Donde:} \quad M'_{In} = M_{In} \backslash \{\langle mid, id, AskOne, at\rangle\}$$

$$M'_{Out} = \begin{cases} M_{Out} \cup \{\langle mid, id, Tell, AT\rangle\}, & \\ AT = \{at\theta\}, \theta \in Test(ag_{bs}, at) & \text{Si } Test(ag_{bs}, at) \neq \{\} \\ M_{Out} \cup \{\langle mid, id, Untell, \{at\}\rangle\}\} & \text{En otro caso} \end{cases} \tag{9.53}$$

La respuesta a un mensaje con fuera ilocutoria *AskAll*, a diferencia de *AskOne* regresa todas las creencias que unifica con el átomo solicitado, mientras que éste último solo regresa el átomo en cuestión o un mensaje de fuerza ilocutoria *Untell* con el átomo como contenido.

9.4 $CTL_{AgentSpeak(L)}$

La semántica operacional de $AgentSpeak(L)$ provee una especificación clara, precisa y computable del ciclo de razonamiento de los agentes BDI, pero ha perdido en el camino precisamente los operadores Intencionales de creencia, deseo e intención. Es por ello que propusimos $CTL_{AgentSpeak(L)}$ [49, 50], una lógica con operadores Intencionales y temporales, cuya semántica se basa en la semántica operacional de $AgentSpeak(L)$. *$CTL_{AgentSpeak(L)}$*

9.4.1 Sintaxis

Las fbfs de $CTL_{AgentSpeak(L)}$ incluyen expresiones intencionales y temporales:

- Si $at \in AgentSpeak(L)$, entonces at, $\mathsf{BEL}(at)$, $\mathsf{DES}(at)$, $\mathsf{INTEND}(at)$ son fbf **intencionales** de $CTL_{AgentSpeak(L)}$. *Fbfs Intencionales*
- Las fbf de **estado** son: *Fbfs de estado*
  - s1 Toda fbf intencional es una fbf de estado.
  - s2 Si ϕ y ψ son fbf de estado, $\phi \wedge \psi$ y $\neg\phi$ lo son.
  - s3 Si ϕ es una fbf de camino, entonces $\mathsf{E}\phi$ es una fbf de estado.
- Las fbf de **camino** son: *Fbfs de camino*
  - p1 Toda fbf de estado es una fbf de camino.
  - p2 Si ϕ y ψ son fbf de camino, $\neg\phi$, $\phi \wedge \psi$, $\bigcirc\phi$, $\Diamond\phi$ y $\phi\ \mathsf{U}\ \psi$ son fbf de camino.

De forma que podemos expresar enunciado como que intentaré en todo futuro posible eventualmente ir a Paris: $\mathsf{INTEND}(\mathsf{A}\Diamond ir(paris))$, tal y como lo hacíamos con las lógicas BDI_{CTL}.

9.4.2 Semántica de los operadores BDI

Para definir la semántica de los operadores intencionales necesitaremos una función auxiliar, que dada una intención nos regrese el conjunto de fbf atómicas que son sujeto de una **meta alcanzable**: *Metas alcanzables*

$$
\begin{aligned}
agls(\top) &= \{\},\\
agls(i[p]) &= \begin{cases} \{at\} \cup agls(i) & \text{if } p = +!at : ct \leftarrow h,\\ agls(i) & \text{otherwise} \end{cases}
\end{aligned}
$$

Observen que se trata de una función recursiva. Las metas alcanzables de una intención vacía son el conjunto vacío. Si la intención no es vacía y el evento disparador del plan en su tope es una meta alcanzable, entonces se guarda la fbf atómica correspondiente para agregarla a las fbf que se colectarán en el resto de la intención.

Definición 9.25 (Semántica Intencional)**.** *La semántica de los operadores BDI es como sigue:*

$$
\begin{aligned}
\mathsf{BEL}_{\langle ag,C\rangle}(\phi) &\equiv ag_{bs} \models \phi\\
\mathsf{INTEND}_{\langle ag,C\rangle}(\phi) &\equiv \phi \in \bigcup_{i \in C_I} agls(i) \vee \bigcup_{\langle te,i\rangle \in C_E} agls(i)\\
\mathsf{DES}_{\langle ag,C\rangle}(\phi) &\equiv \langle +!\phi, i\rangle \in C_E \vee \mathsf{INTEND}(\phi)
\end{aligned}
$$

Si el agente ag y su circunstancia C son explícitos, escribimos simplmenete $\mathsf{BEL}(\phi)$, $\mathsf{DES}(\phi)$ e $\mathsf{INTEND}(\phi)$. De forma que, se dice que un agente ag **cree** la fbf atómica ϕ, si ésta es consecuencia lógica de sus creencias. Se dice que un agente **intenta** ϕ, si ésta fbf atómica es sujeto de una meta alcanzable en las intenciones, tanto activas como suspendidas, del agente. Se dice que un agente **desea** la fbf ϕ, si ésta fbf atómica es intendada o existe un evento donde la fbf es sujeto de una meta alcanzable.

Creer

Intentar

Desear

9.4.3 Semántica de los operadores temporales

Como suele ser el caso, la semántica de los operadores temporales requiere de un modelo basada en una estrcutura de Kripke.

Definición 9.26 (Estructura de Kripke $AgentSpeak(L)$)**.** *Sea $K = \langle S, R, V\rangle$ es una **estructura de Kripke** definida sobre el sistema de transición (Γ), donde:*

- *S es un conjunto de configuraciones $\langle ag, C, M, T, s\rangle$.*
- *$R \subseteq S^2$ es una relación serial t.q. para todo $(c_i, c_j) \in R$, $(c_i, c_j) \in \Gamma$.*
- *V es una función de evaluación sobre los operadores intencionales y temporales, por ej., $V_{\mathsf{BEL}}(c, \phi) \equiv \mathsf{BEL}_{\langle ag,C\rangle}(\phi)$ en la configuración $c = \langle ag, C, M, T, s\rangle$, etc.*

Le expresión $x = c_0, \ldots, c_n$ denota un **camino**, una secuencia de configuraciones t.q. para todo $c_i \in S$, $(c_i, c_{i+1}) \in R$. La expresión x^i denota el sufijo del camino x a partir de la configuración c_i.

Definición 9.27 (Semántica temporal)**.** *La semántica de las fbf de estado es como sigue:*

$$
\begin{aligned}
K, c_i \models \mathsf{BEL}(\phi) &\Leftrightarrow \phi \in V_{\mathsf{BEL}}(c_i, \phi)\\
K, c_i \models \mathsf{INTEND}(\phi) &\Leftrightarrow \phi \in V_{\mathsf{INTEND}}(c_i, \phi)\\
K, c_i \models \mathsf{DES}(\phi) &\Leftrightarrow \phi \in V_{\mathsf{DES}}(c_i, \phi)\\
K, c_i \models \mathsf{E}\phi &\Leftrightarrow \exists x^i \exists c_{j \geq i} \in x^i \text{ t.q. } K, c_j \models \phi\\
K, c_i \models \mathsf{A}\phi &\Leftrightarrow \forall x^i \exists c_{j \geq i} \in x^i \text{ t.q. } K, c_j \models \phi
\end{aligned}
$$

La semántica de las fbf de camino es como sigue:

$$
\begin{aligned}
K, c_i \models \phi &\Leftrightarrow K, x^i \models \phi, \text{cuando } \phi \text{ es una fbf de estado}\\
K, c_i \models \bigcirc\phi &\Leftrightarrow K, x^{i+1} \models \phi\\
K, c_i \models \Diamond\phi &\Leftrightarrow \exists c_{j \geq i} \in x^i \text{ t.q. } K, c_j \models \phi\\
K, c_i \models \phi \,\mathsf{U}\, \psi &\Leftrightarrow (\exists c_{k \geq i} \text{ t.q. } K, x^k \models \psi \wedge\\
&\qquad \forall c_{i \leq j < k} \quad K, x^j \models \phi) \vee (\exists c_{j \geq i} \quad K, x^j \models \phi).
\end{aligned}
$$

Observen que la semántica del operador "hasta" (U) corresponde a su versión débil (ψ puede no ocurrir nunca). Las definiciones de corrida, satisfacción y válidez son las comunes.

Definición 9.28 (Corrida). *Dada una configuración inicial c_0, la **corrida** K^0_Γ denota una secuencia de configuraciones $c_0, c_1, \ldots$ t.q. $\forall i \geq 1, c_i = \Gamma(c_{i-1})$.* *Corrida*

Definición 9.29 (Satisfacción). *Dada una corrida K^0_Γ, se dice que la propiedad $\phi \in CTL_{AgentSpeak(L)}$ se **satisface** si $\forall i \geq 0, K^0_\Gamma, c_i \models \phi$.* *Satisfacción*

Definición 9.30 (Validez). *Una propiedad $\phi \in CTL_{AgentSpeak(L)}$ es **válida**, si $K^0_\Gamma, c_0 \models \phi$ para toda configuración inicial c_0.* *Validez*

9.4.4 Propiedades BDI de $AgentSpeak(L)$

Bordini y Moreira [13] utilizaron el segmento BDI de $CTL_{AgentSpeak(L)}$ para investigar sus propiedades con respecto a la tesis de asimetría, encontrando que $AgentSpeak(L)$ no es equivalente a ninguna de las lógicas estudiadas por Rao y Georgeff. El cuadro 9.2 muestra que axiomas satisface $AgentSpeak(L)$, comparándo el resultado con otras lógicas BDI.

	AT1	AT2	AT3	AT4	AT5	AT6	AT7	AT8	AT9
Realismo fuerte	✓		✓	✓		✓	✓		✓
Realismo	✓	✓		✓	✓		✓	✓	
Realismo débil	✓	✓	✓	✓	✓	✓	✓	✓	✓
$AgentSpeak(L)$		✓	✓	✓		✓		✓	✓

Cuadro 9.2: La tesis de asimetría en $AgentSpeak(L)$ y otras Lógicas BDI.

Nuestro interés [49, 50] era saber que tipo de estrategia de compromiso siguen los agentes $AgentSpeak(L)$ que implementa Jason. Por tanto, primero se abordó el siguiente problema: ¿Es válido el axioma de posposición finita?

$$\mathsf{INTEND}(\phi) \rightarrow \mathsf{A}\Diamond(\neg\mathsf{INTEND}(\phi))$$

Prueba: Asumamos $K, c_0 \models \mathsf{INTEND}(\phi)$, entonces dada la definición de INTEND, existe un plan $p \in C_I \cup C_E$ con la cabeza $+!\phi$ en c_0. p es finito por definición. Siguiendo $ClrInt_X$ el plan es abandonado por éxito o fracaso. □

Los agentes $AgentSpeak(L)$ **no satisfacen** la propiedad de compromiso ciego:

$$\mathsf{INTEND}(\mathsf{A}\Diamond\phi) \rightarrow \mathsf{A}\Diamond\mathsf{BEL}(\phi)$$

Prueba: Asumamos una configuración inicial c_0, donde $ag = \langle\{b(t_1)\}, \{+b(t_1) : \top \leftarrow p(t_2). \quad +!p(t_2) : \top \leftarrow +b(t_3).\}$. Se obtendrá una configuración c' donde $C_I = \{[+!p(t_2) : \top \leftarrow \quad +b(t_3).]\}$ y $C_E = \{\}$ t.q. $K, c' \models \mathsf{INTEND}(p(t_2))$ y $K, c' \not\models \mathsf{BEL}(p(t_2))$. En la siguiente configuración c'', $K, c'' \models \neg\mathsf{INTEND}(p(t_2))$ y $K, c'' \not\models \mathsf{BEL}(p(t_2))$. □

Un agente $AgentSpeak(L)$ **satisface** una forma limitada del compromiso racional: Gu

$$\mathsf{INTEND}(\mathsf{A}\Diamond\phi) \rightarrow \mathsf{A}(\mathsf{INTEND}(\mathsf{A}\Diamond\phi)\ \mathsf{U}\ \neg\mathsf{BEL}(\mathsf{E}\Diamond\phi))$$

Prueba: Siguiendo la demostración de posposición finita, en los casos de fallo el agente satisface eventualmente $\bigcirc\neg\mathsf{INTEND}(\phi)$ debido a Rel_2 –Para un evento $\langle te, i[+!\phi : c \leftarrow h.]\rangle$ no hubo planes relevantes y la intención asociada es abandonada. □

Se trata de una forma limitada de compromiso racional porque no hay representación explícita de la razón de abandono, aunque ésta se podría aprender.

9.5 LECTURAS Y EJERCICIOS SUGERIDOS

Rao [89] introduce $AgentSpeak(L)$ y su semántica operacional, la primera parte de este capítulo se basa en este artículo. Bordini, Hübner y Wooldridge [12] es el libro de texto oficial sobre *Jason* y la programación de SMA con este lenguaje. La semántica operacional presentada en la segunda parte de este capítulo se basa en este libro. Plotkin [86] ofrece una introducción a las semánticas operacionales estructurales y su pertinencia. d'Inverno y col. [33] nos ofrecen una especificación de dMARS, el predecesor de $AgentSpeak(L)$ donde pueden revisarse las principales ideas detras de la programación BDI. Aunque esta especificación no está basada en una semántica operacional, es de utilidad para comprender las extensiones provistas por *Jason* y otras posibles extensiones a implementar.

Ejercicios

Ejercicio 9.1. *En el ejemplo 9.2 identifique con precisión los elementos del programa, de ser posible, argumentando con las reglas sintácticas correspondientes. Por ejemplo:* `factorial(1,1)` *es una creencia, de acuerdo a la ecuación 9.8, etc.*

Ejercicio 9.2. *En el ejemplo 9.4 explique las razones por las cuales las substituciones son válidas o no.*

Ejercicio 9.3. *Desarrolle paso a paso la composición de substituciones del ejemplo 9.6.*