Modelli e metodi per la verifica dei sistemi multi-agente tramite model checking

Roberto Pasolini

21 febbraio 2011

Indice

1 Introduzione

La realizzazione di sistemi software caratterizzati da una certa complessità (come i sistemi multi-agente) richiede di seguire un processo di produzione ben pianificato. In genere questo processo comprende la *verifica*, che consiste nel controllare che il sistema sia privo di errori. Gli errori possono essere involontariamente introdotti durante qualsiasi fase del processo produttivo (analisi, progetto, implementazione, ...) e si propagano nelle fasi successive finché non sono individuati. Con l'avanzare del processo di produzione, i costi per la rimozione degli errori tendono ad alzarsi, per poi diventare potenzialmente molto alti quando il sistema è già in funzione: infatti la fase di verifica avviene prima del *deployment* del sistema, dal momento che si desidera eliminare quanti più difetti possibile prima che il sistema sia distribuito e messo in funzione.

Il metodo classico per ricercare errori nel sistema consiste nell'utilizzare i test: si individuano i casi d'uso più o meno tipici del sistema e, coerentemente ad essi, si effettuano delle esecuzioni di prova del sistema e si controlla se da queste emergono errori. I limiti di questo metodo sono l'impossibilità di collaudare in modo esauriente il sistema (in genere non si possono prevedere tutti gli scenari possibili) e la necessità di avere pronta un'implementazione del sistema, per lo meno delle parti che sono oggetto dei test.

In aggiunta ai test, un altro approcio possibile è la *verifica formale*, ovvero l'applicazione rigida di metodi matematici per modellare e analizzare il sistema. L'utilizzo di questi metodi in genere consente di rilevare falle nel sistema analizzandone un modello, che può essere utilizzato anche come base per l'implementazione del sistema vero e proprio una volta verificata la sua correttezza. Definendo formalmente e accuratamente il modello del sistema e le proprietà desiderate, attraverso tecniche di verifica automatica è possibile sapere con certezza se il sistema gode di queste proprietà.

Il *model checking* è un metodo di verifica formale che prevede l'esplorazione sistematica di tutti gli stati in cui può trovarsi il sistema durante il suo funzionamento. Tramite specifici algoritmi eseguiti da appositi programmi (*model checkers*), è possibile analizzare il modello a stati di un sistema e verificare se soddisfa delle proprietà descritte formalmente [1].

In questo lavoro, dopo alcune nozioni generali sul model checking, è illustrato un insieme di metodi, linguaggi e sistemi software proposti nella letteratura per descrivere formalmente modelli di sistemi multi-agente ed eseguirne la verifica.

2 Model checking

2.1 Modello del sistema

Per poter eseguire il model checking, il sistema va descritto in forma di un automa a stati finiti. In generale il modello utilizzato è un *sistema di transizioni*, costituito da un insieme finito S di stati e un insieme $T \subseteq S \times S$ di transizioni tra di essi. Un modello

di questo tipo, esprimibile attraverso un grafo orientato, definisce tutti gli stati in cui il sistema si può trovare e le possibili transizioni tra essi: in pratica, il sistema può passare direttamente da uno stato $s \in S$ a un altro stato $s' \in S$ se e solo se $(s, s') \in T$. In un sistema di transizioni è generalmente definito anche un insieme $I \subseteq S$ di stati iniziali, che stabilisce in quali stati può trovarsi il sistema al suo avvio[1].

In diversi casi, la descrizione del sistema comprende un insieme AP di *proposizioni atomiche* e una *funzione di etichettatura* $L : S \to 2^{AP}$. Una tupla $\langle S, I, T, L \rangle$ è detta *struttura di Kripke*. Una proposizione atomica rappresenta un fatto che si verifica in uno o più stati del sistema: $L(s)$ denota l'insieme delle proposizioni atomiche vere nello stato s; eventualmente gli stati del sistema possono fungere da proposizioni atomiche, ciascuna valida solo nello stato stesso. Le proposizioni definite si possono utilizzare per esprimere formalmente le proprietà globali che il modello deve avere.

Solitamente, la rappresentazione del sistema come automa a stati non viene data direttamente dall'utente, che descrive invece il sistema in modo più comprensibile, utilizzando un apposito linguaggio: è il model checker che si occupa di tradurre la specifica dell'utente in un sistema di transizioni da analizzare. Ad esempio, nel modello di un programma semplice, l'insieme degli stati possibili è costituito da tutte le possibili combinazioni di valori delle variabili e dello stato di esecuzione del programma: in questo caso l'utente si limita a definire le variabili, il loro dominio (i valori che possono assumere) e le istruzioni eseguite dal programma; il sistema di transizioni è poi costruito automaticamente dal model checker in base a queste informazioni. Generalmente, allo stesso modo, anche l'etichettatura degli stati con proposizioni atomiche è automatica: al posto di proposizioni di un insieme chiuso, si esprimono condizioni vere o false a seconda dello stato del sistema, ad esempio confronti tra valori e variabili (es.: `x > 0`).

Il problema principale nell'applicazione del model checking è dato dall'esplosione combinatoria dei modelli: nella pratica, il numero di stati possibili di un sistema tende ad essere molto elevato, rendendone l'analisi computazionalmente onerosa in termini di tempo impiegato e memoria necessaria. Per limitare questo problema sono stati studiati algoritmi e strutture dati ad hoc: ad esempio, come spiegato in seguito, si memorizzano stati e transizioni senza rappresentarli esplicitamente uno per uno.

Avendo a disposizione un modello del sistema in forma di automa a stati finiti, è anche possibile compiere simulazioni dello stesso: in ciascuna simulazione si parte da uno stato iniziale e si eseguono diverse transizioni attraverso altri stati, scelte tutte o in parte in modo nondetetministico. Simulando il sistema si possono avere rapidamente delle prime indicazioni sulla correttezza del modello.

[1] In alcuni casi, al posto di un insieme I di stati iniziali, si considera un unico stato iniziale s_0.

2.2 Proprietà e logica temporale

Una volta costruito il modello del sistema, per poterlo analizzare bisogna stabilire quali sono le proprietà che questo deve avere. In generale, gli algoritmi di model checking prendono in ingresso il modello del sistema e una proprietà e calcolano se questa proprietà è soddisfatta o meno dal modello; in caso di esito negativo, a volte, è fornito un esempio di esecuzione del sistema come dimostrazione.

Una proprietà Φ, riferita ad un modello $\mathcal{M}$, esprime in modo formale e inambiguo un fatto riferito ad un generico stato del sistema: se questo fatto si verifica in uno stato $s \in S$, si dice che *s soddisfa* Φ ($\mathcal{M}, s \models \Phi$ o, in breve, $s \models \Phi$). Più in generale, si può dire che il modello $\mathcal{M}$ soddisfa la proprietà Φ ($\mathcal{M} \models \Phi$ o, in breve, $\models \Phi$) se e solo se tutti i suoi stati iniziali la soddisfano.

I fatti esprimibili dalle proprietà si riferiscono allo stato presente e agli stati futuri che il sistema attraversa nella sua esecuzione. L'evoluzione di un sistema a partire da uno stato s_0 può essere rappresentata da un *percorso* $\pi = s_0 s_1 s_2 \ldots$, definito come una sequenza finita o infinita di stati dell'insieme S tale che $(s_i, s_{i+1}) \in T \ \forall i \geq 0$.

Per esprimere le proprietà del sistema si possono utilizzare diverse forme di *logica temporale*, un'estensione della logica proposizionale con operatori modali che consentono di riferirsi al comportamento futuro del sistema. La logica proposizionale di base, come linguaggio, è definito dalla seguente grammatica.

$$\Phi ::= \text{vero} \mid a \mid \neg\Phi \mid \Phi \wedge \Phi$$

a rappresenta una proposizione atomica in un insieme AP che, a seconda del contesto, può essere vera o falsa, vero rappresenta una frase vera in ogni caso, $\neg\Phi$ è vera se e solo se Φ è falsa e $\Phi_1 \wedge \Phi_2$ è vera se e solo se sia Φ_1 che Φ_2 sono vere. Da questi operatori è possibile derivare tutti gli altri utilizzati tipicamente nella logica (come la disgiunzione $\vee$). Se si considera un modello etichettato con un insieme di proposizioni atomiche AP, un suo stato s soddisfa una formula del tipo a ($s \models a$) con $a \in AP$ se e solo se lo stesso stato è etichettato con a ($a \in L(s)$).

Gli operatori aggiunti a questi nella logica temporale dipendono dalla concezione del tempo futuro che si considera: in una visione lineare, ad ogni istante presente o futuro corrisponde un singolo istante successivo; nella visione ramificata, invece, il tempo si può dividere in più percorsi, per cui per ogni istante si considerano tutti i possibili istanti successivi. A seconda dell'assunzione che si fa, si utilizzano linguaggi diversi con potere espressivo diverso.

2.2.1 Linear Temporal Logic

La LTL (*Linear Temporal Logic*) [6] è un formalismo in cui il tempo futuro è considerato lineare. Una proprietà LTL esprime una specifica che può essere soddisfatta da un percorso $\pi = s_0 s_1 s_2 \ldots$: uno stato s_0 soddisfa una proprietà Φ se e solo se tutti i possibili percorsi che partono da s_0 la soddisfano. La grammatica utilizzata per descrivere

una formula LTL è la seguente.

$$\Phi ::= \text{vero} \mid a \mid \neg\Phi \mid \Phi \wedge \Phi \mid \bigcirc\Phi \mid \Phi \cup \Phi$$

Dagli operatori dati si possono derivare altri due operatori molto usati, $\Diamond$ e $\Box$, definiti con $\Diamond\Phi = \text{vero} \cup \Phi$ e $\Box\Phi = \neg\Diamond\neg\Phi$. Dato un generico percorso $\pi = s_0 s_1 s_2 \ldots$, il significato degli operatori temporali si può descrivere formalmente come segue.

$$\begin{aligned}
\pi \models \bigcirc\Phi &\Leftrightarrow s_1 \models \Phi \\
\pi \models \Phi \cup \Psi &\Leftrightarrow \exists n \geq 0 : (s_i \models \Phi \; \forall i < n) \wedge (s_n \models \Psi) \\
\pi \models \Diamond\Phi &\Leftrightarrow \exists n \geq 0 : s_n \models \Phi \\
\pi \models \Box\Phi &\Leftrightarrow s_i \models \Phi \; \forall i \geq 0
\end{aligned}$$

In pratica, dato un generico stato $s \in S$:

- $s \models \bigcirc\Phi$ è vera se e solo se allo stato s segue sempre uno stato in cui si verifica Φ;
- $s \models \Phi \cup \Psi$ è vera se e solo se il sistema dallo stato s in ogni caso possibile si trova o si troverà in futuro in uno stato in cui si verifica Ψ e fino ad allora attraverserà solo stati in cui si verifica Φ;
- $s \models \Diamond\Phi$ è vera se e solo se il sistema dallo stato s in ogni caso possibile si trova o si troverà in futuro in uno stato in cui si verifica Φ;
- $s \models \Box\Phi$ è vera se e solo se il sistema, arrivato nello stato s, si trova e si troverà infinitamente in stati in cui si verifica Φ.

Un esempio di proprietà che si può esprimere in LTL è la *fairness*, che si ha quando è garantito che una o più condizioni "ogni tanto" si verifichino nel sistema: ad esempio un algoritmo di scheduling di processi è considerato *fair* se garantisce che tutti i processi continuino ad essere eseguiti e non ce ne sia neanche uno lasciato in sospeso per sempre. Se una condizione Φ si verifica "infinitamente spesso", si può dire che il modello soddisfa la proprietà LTL $\Box\Diamond\Phi$.

2.2.2 Computation Tree Logic

La CTL (*Computation Tree Logic*) [3], al contrario della LTL, ha una concezione ad albero del tempo futuro: in ogni istante in cui non c'è un unico stato successivo possibile, la linea temporale si ramifica. Percorrendo questo albero dal nodo radice che rappresenta lo stato di partenza, si possono seguire vari percorsi: in CTL si possono esprimere proprietà che si verificano se e solo se delle condizioni come quelle della LTL sono vere in alcuni (almeno uno) dei percorsi possibili o in tutti i percorsi possibili. La grammatica utilizzata in CTL è la seguente.

$$\Phi ::= \text{vero} \mid a \mid \neg\Phi \mid \Phi \wedge \Phi \mid \exists\vartheta \mid \forall\vartheta$$

$$\vartheta ::= \bigcirc\Phi \mid \Phi \cup \Phi$$

Anche in questo caso si possono introdurre gli operatori $\Diamond$ e $\Box$: $\exists\Diamond\Phi = \exists(\text{vero} \cup \Phi)$, $\forall\Diamond\Phi = \forall(\text{vero} \cup \Phi)$, $\exists\Box\Phi = \neg\forall\Diamond\neg\Phi$, $\forall\Box\Phi = \neg\exists\Diamond\neg\Phi$.

$\exists$ e $\forall$ sono rispettivamente il quantificatore esistenziale di percorsi e quello universale, entrambi sono seguiti da una formula riferita a un generico percorso: $s \models \exists\vartheta$ è vera se e solo se da s parte almeno un percorso per cui si verifica ϑ, mentre $s \models \forall\vartheta$ è vera se e solo se ϑ è vera in tutti i percorsi possibili che partono da s. Gli operatori $\bigcirc$, $\cup$, $\Diamond$ e $\Box$ hanno lo stesso significato che hanno in LTL.

LTL e CTL hanno un'espressività diversa: ci sono proprietà che possono essere espresse in entrambi i formalismi e altre che possono essere espresse in uno solo dei due (l'uno o l'altro a seconda dei casi). Ad esempio la proprietà LTL $\Diamond(\Phi \wedge \bigcirc\Phi)$ ("prima o poi il sistema passa per due stati consecutivi in cui si verifica Φ") non ha una formulazione equivalente in CTL, mentre la proprietà CTL $\forall\Box\exists\Diamond\Phi$ ("è sempre possibile che nel sistema prima o poi si verifichi Φ") non ha una formulazione equivalente in LTL. Per avere un formalismo in grado di esprimere tutte le proprietà che si possono formulare da almeno uno tra LTL e CTL, è stato introdotto CTL*, una variante di CTL in cui gli operatori della logica proposizionale ($\neg$, $\wedge$ e tutti quelli derivabili da essi) possono essere applicati anche alle formule di percorso (come in LTL).

2.3 Tecniche e algoritmi tipici

Per implementare algoritmi per il model checking, è necessario innanzitutto definire le strutture dati in cui memorizzare il modello del sistema. Gli algoritmi hanno bisogno di una rappresentazione del sistema come automa a stati, ma il numero complessivo di stati che può assumere un sistema è potenzialmente molto elevata: per questo si utilizzano strutture dati che consentono di memorizzare gli insiemi degli stati e delle transizioni senza enumerarli uno per uno. Una struttura dati utilizzata tipicamente è l'OBDD (*Ordered Binary Decision Diagram*), che in generale è utilizzato per valutare una funzione del tipo $B^n \rightarrow B$ con $B = \{0, 1\}$. Assegnando ad ogni stato una codifica binaria, è possibile memorizzare in un OBDD la funzione $S \times S \rightarrow B$ che associa a una coppia di stati (s, s') il valore 1 se e solo se ad s' è tra i possibili stati successivi a s ($(s, s') \in T$). Allo stesso modo, per ogni proposizione atomica $a \in AP$, si può definire una funzione $S \rightarrow B$ che indica se a è vera o no in ciascun stato.

In generale, gli algoritmi di model checking accettano in ingresso un modello $\mathcal{M}$ e una proprietà Φ espressa in un linguaggio specifico (come CTL) e restituiscono un insieme $Sat(\Phi)$ degli stati del modello che soddisfano la proprietà data. Una volta ottenuto l'insieme $Sat(\Phi)$ si può dire se Φ è soddisfatta da un qualsiasi stato s ($s \models \Phi \Leftrightarrow s \in Sat(\Phi)$) e/o dal sistema ($\mathcal{M} \models \Phi \Leftrightarrow I \subseteq Sat(\Phi)$). Per ogni operatore del linguaggio utilizzato, è definita una procedura specifica per trovare l'insieme di stati che soddisfano una condizione espressa con quell'operatore: ad esempio, per

determinare l'insieme $Sat(\exists \bigcirc \Phi)$, una volta determinato $Sat(\Phi)$, vanno cercati gli stati s tali che $\exists s' \in S : (s, s') \in T \wedge s' \in Sat(\Phi)$.

3 Verifica di *interpreted systems*

Per eseguire il model checking di sistemi multi-agente, è opportuno innanzitutto definire un formalismo con cui modellare un sistema multi-agente. In generale, un sistema multi-agente è modellabile come un automa a stati finiti, ma esprimere in modo intuitivo le proprietà desiderate del sistema nei linguaggi logici visti finora (LTL e CTL) può risultare difficoltoso. Per questo sono stati introdotti dei modelli formali specifici per la descrizione di sistemi multi-agente e linguaggi logici per esprimere proprietà sui modelli di questi sistemi. Un esempio relativamente semplice di questi formalismi è quello degli *interpreted systems* [4], in cui praticamente sono i singoli agenti ad essere modellati come automi a stati, in modo simile a quanto avviene solitamente quando si vogliono modellare sistemi concorrenti.

3.1 Modello

Un sistema è composto da un insieme Ag di agenti. Considerando ciascun agente $i \in Ag$ singolarmente, ognuno di essi è contraddistinto da un insieme L_i di stati locali possibili, un insieme Act_i di azioni che può compiere e un protocollo $P_i : L_i \to 2^{Act_i}$ che associa ad ogni stato le azioni possibili nello stesso. In un sistema multi-agente gli agenti "vivono" in un ambiente, il quale può essere modellato esso stesso come un agente: L_E, Act_E e P_E denotano rispettivamente gli stati, le azioni e il protocollo dell'ambiente.

Dalla combinazione dei possibili stati locali degli n agenti di un sistema e del loro ambiente, si ottiene l'insieme $S = L_1 \times L_2 \times \ldots \times L_n \times L_E$ dei possibili stati globali del sistema; allo stesso modo, assumendo che tutti gli agenti e l'ambiente compiano le loro azioni in sincronia tra loro, si denota con $Act = Act_1 \times Act_2 \times \ldots \times Act_n \times Act_E$ l'insieme delle possibili combinazioni di azioni che possono compiere. Per ciascun agente, l'evoluzione del suo stato locale è data da una funzione $t_i : L_i \times L_E \times Act \to L_i$, mentre l'evoluzione dello stato dell'ambiente è data da $t_E : L_E \times Act \to L_E$: da queste funzioni, si può ottenere la funzione $t : S \times Act \to S$ che descrive l'evoluzione dello stato globale del sistema nel suo insieme.

A seconda dei possibili stati iniziali $I \subseteq S$ del sistema, si ha un insieme $W \subseteq S$ di stati raggiungibili (detti anche *mondi possibili*), mentre dalla funzione t si ottiene l'insieme $T = \{(w, w') \in W \times W \mid \exists a \in Act, t(w, a) = w'\}$ delle possibili transizioni tra gli stati. Per ogni agente i si può ricavare una relazione di *accessibilità epistemica* $\sim_i \subseteq W \times W$ tale che $w \sim_i w'$ se e solo se ad entrambi gli stati globali w e w' corrisponde uno stesso stato locale dell'agente: ciascuna $\sim_i$ in pratica è una relazione d'equivalenza che indica quali stati globali del sistema appaiono identici dal punto di vista dell'agente

i, in quanto ad essi corrisponde un unico stato locale dell'agente. Anche in questo caso, dato un insieme AP di proposizioni atomiche che rappresentano dei fatti che si verificano in alcuni stati, si può definire una funzione $L : W \to 2^{AP}$ che associa a ciascuno stato le proposizioni vere nello stesso.

Si può quindi considerare il modello di un sistema composto da n agenti individuato da una tupla $\langle W, T, \sim_1, \sim_2, \ldots, \sim_n, I, L\rangle$. Questo modello è in pratica una struttura di Kripke con l'aggiunta delle relazioni $\sim_i$, sulle quali si possono esprimere ulteriori proprietà rispetto a quelle definite nelle logiche temporali di base.

3.2 Computation Tree Logic of Knowledge

La CTLK (*Computation Tree Logic of Knowledge*) [4] è una forma di logica temporale estesa con operatori che esprimono assunzioni riguardanti la *conoscenza* degli agenti, ovvero riguardo a ciò che gli agenti sanno con certezza del sistema. Nella sua forma più basilare, la CTLK è definita dalla seguente grammatica, dove a indica una generica proposizione atomica dell'insieme AP e i indica un generico agente del sistema.

$$\Phi ::= \text{vero} \mid a \mid \neg\Phi \mid \Phi \wedge \Phi \mid EX\Phi \mid EG\Phi \mid E(\Phi U \Phi) \mid K_i\Phi$$

Dagli operatori $\neg$ e $\wedge$ si possono derivare gli altri operatori logici classici, mentre dagli altri operatori si possono derivare $F\Phi = \text{vero}\ U\ \Phi$, $AX\Phi = \neg EX\neg\Phi$ e $AG\Phi = \neg EF\neg\Phi$. Il significato dei simboli X, U, F, G, E e A equivale rispettivamente a quello dei simboli $\bigcirc$, $\cup$, $\Diamond$, $\Box$, $\exists$ e $\forall$ nella CTL. Gli operatori primitivi possono essere definiti formalmente come segue.

$$\begin{aligned}
w &\models \text{vero} \\
w \models a &\quad\Leftrightarrow\quad a \in L(w) \\
w \models \neg\Phi &\quad\Leftrightarrow\quad \neg(w \models \Phi) \\
w \models \Phi \wedge \Psi &\quad\Leftrightarrow\quad w \models \Phi \wedge w \models \Psi \\
w_0 \models EX\Phi &\quad\Leftrightarrow\quad \exists \pi = w_0 w_1 \ldots : w_1 \models \Phi \\
w_0 \models EG\Phi &\quad\Leftrightarrow\quad \exists \pi = w_0 w_1 \ldots : w_i \models \Phi\ \forall i \geq 0 \\
w_0 \models E(\Phi U \Psi) &\quad\Leftrightarrow\quad \exists \pi = w_0 w_1 \ldots, n \geq 0 : w_n \models \Psi \wedge w_i \models \Phi\ \forall i < n
\end{aligned}$$

Il principale operatore introdotto nella CTLK è K: intuitivamente, $K_i\Phi$ significa "l'agente i sa che Φ è vera". Dal momento che un agente $i \in Ag$ non ha una conoscenza completa dello stato globale del sistema, ma solo del suo stato locale, $w \models \Phi$ non implica necessariamente $w \models K_i\Phi$: un agente i non sa che il sistema si trova nello stato w, sa con certezza invece che si trova in un generico stato w' tale che $w \sim_i w'$, ovvero in un qualsiasi stato che percepisce nello stesso modo in cui percepirebbe w. Per questo, $w \models K_i\Phi$ è vera se e solo se Φ è vera in tutti gli stati $w' \in W$ tali che $w \sim_i w'$: in pratica un agente ha la certezza di qualcosa solo se questa si verifica in tutti gli stati del

sistema in cui sa che potrebbe trovarsi.

$$w \models K_i\Phi \Leftrightarrow \forall w' \in W, w \sim_i w' : w' \models \Phi$$

Solitamente, oltre a quelli già visti, si introducono in CTLK altri operatori che esprimono assunzioni riguardo ad un insieme di agenti $\Gamma \subseteq Ag$. Uno di questi è E_Γ: $E_\Gamma\Phi$ significa "tutti gli agenti in Γ sanno che Φ è vera". Formalmente, $w \models E_\Gamma\Phi$ è vera se e solo se Φ si verifica in tutti gli stati w' che sono in relazione con w considerando l'unione di tutte le relazioni di accessibilità degli agenti in Γ.

$$w \models E_\Gamma\Phi \Leftrightarrow w \models \bigwedge_{i\in\Gamma} K_i\Phi \Leftrightarrow \forall w' \in W, (w, w') \in \bigcup_{i\in\Gamma} \sim_i : w' \models \Phi$$

3.2.1 Conoscenza comune

Un insieme $\Gamma \subseteq Ag$ di agenti ha una cosiddetta *conoscenza comune* di un fatto Φ quando ciascun agente sa che Φ è vero, sa che qualunque altro agente dell'insieme sa che Φ è vero, sa che qualunque altro agente dell'insieme sa ciò, e così via. Un agente può essere a conoscenza di ciò che un altro agente sa in quanto si presuppone che ciascun agente conosca i possibili stati S del sistema e le relazioni di accessibiltà $\sim_i$ di tutti gli agenti in esso: tuttavia la conoscenza di un agente, anche quella relativa alla conoscenza degli altri, dipende dallo stato locale dell'agente, che dipende dallo stato globale del sistema.

Ad esempio, presupponendo che il sistema si trovi in uno stato w in cui un agente 1 sa che p è vera ($w \models K_1 p$), un agente 2 potrebbe ignorare che p sia vero oppure saperlo e, in quest'ultimo caso, potrebbe anche sapere che l'agente 1 lo sa. La conoscenza di p da parte dell'agente 2 ($K_2 p$) dipende solo da $\sim_2$, mentre se l'agente 2 sappia o no che l'agente 1 sa p ($K_2K_1 p$) dipende anche da $\sim_1$: tutti gli stati w' percepiti dall'agente 2 come w ($w \sim_2 w'$) devono soddisfare la condizione $K_1 p$, per cui, perché $w \models K_2K_1 p$, sia vera, p deve verificarsi in tutti gli stati w'' tali che $w \sim_2 w' \sim_1 w''$.

Generalizzando l'esempio, considerando un insieme $\Gamma \subseteq Ag$ di agenti, la condizione $w \models K_iK_j\Phi\ \forall i, j \in \Gamma$ ("ciascun agente in Γ sa che se stesso e tutti gli altri sanno Φ") si verifica negli stati w'' tali che $w \sim_i w' \sim_j w''\ \forall i, j \in \Gamma, w' \in W$. Applicando il principio ricorsivamente, si dimostra che nello stato w gli agenti dell'insieme Γ hanno la conoscenza comune di una proprietà Φ se e solo se questa è valida in tutti gli stati in relazione con w nella chiusura transitiva[2] dell'unione delle relazioni di accessibilità degli agenti in Γ: in CTLK ciò si scrive $w \models C_\Gamma\Phi$.

$$w \models C_\Gamma\Phi \Leftrightarrow \forall w' \in W, (w, w') \in \left(\bigcup_{i\in\Gamma} \sim_i\right)^+ : w' \models \Phi$$

Il significato dell'operatore C_Γ può essere descritto intuitivamente in modo ricorsivo tramite la proprietà sempre valida $C_\Gamma\Phi \Leftrightarrow E_\Gamma(\Phi \wedge C_\Gamma\Phi)$.

[2] La chiusura transitiva R^+ di una relazione $R : S \times S$ è il più piccolo insieme tale da contenere R ($R^+ \supseteq R$) e godere della proprietà transitiva.

3.2.2 Conoscenza distribuita

La *conoscenza distribuita* di un insieme $\Gamma \subseteq Ag$ di agenti consiste nella "combinazione" della conoscenza di tutti gli agenti dell'insieme. Ad esempio, se un agente 1 sa Φ e un altro agente 2 sa $\Phi \rightarrow \Psi$, si può dire che 1 e 2 hanno una conoscenza distribuita del fatto che Ψ sia vero. In CTLK si può introdurre l'operatore D_Γ per rappresentare questo concetto: $D_\Gamma\Phi$ si verifica se e solo se dalla conoscenza distribuita degli agenti dell'insieme Γ si dimostra che Φ è vero. Formalmente, $w \models D_\Gamma\Phi$ è vera se e solo se Φ si verifica in tutti gli stati w' che sono in relazione con w nell'intersezione di tutte le relazioni di accessibilità degli agenti in Γ.

$$w \models D_\Gamma\Phi \Leftrightarrow \forall w' \in W, (w, w') \in \bigcap_{i\in\Gamma} \sim_i : w' \models \Phi$$

3.3 MCMAS

MCMAS [5] è un model checker simbolico specifico per sistemi multi-agente. I sistemi sono modellati tramite il linguaggio ISPL (*Interpreted Systems Programming Language*), che riprende il modello formale dei sistemi interpretati citato sopra, mentre le proprietà da verificare sono espresse in CTLK. MCMAS, implementato in C++, utilizza gli OBDD, gestiti tramite la libreria CUDD, per memorizzare il modello, applicando alcune ulteriori ottimizzazioni.

MCMAS dispone di un'interfaccia a linea di comando (accessibile eventualmente anche tramite un apposito plug-in per la piattaforma Eclipse): passando in ingresso un file ISPL contenente il modello e le proprietà su cui eseguire il model checking, MCMAS legge il file, costruisce le strutture dati OBDD in base alla descrizione del modello e le utilizza per verificare se il sistema gode delle proprietà date; per ogni proprietà non verificata, può essere generata una traccia dell'esecuzione del sistema che dimostra come questa non si verifichi. È anche possibile eseguire MCMAS in modalità interattiva per eseguire una simulazione del modello: l'utente può scegliere uno stato iniziale ed esplorare le possibili evoluzioni del sistema.

3.3.1 Descrizione del sistema

Per descrivere un sistema multi-agente nel linguaggio ISPL utilizzato da MCMAS, sono innanzitutto descritti i singoli agenti presenti nel sistema (incluso l'ambiente, se necessario); ciascuno di essi è descritto principalmente in quattro punti.

Variabili e stati locali. A ciascun agente è associato un insieme di variabili locali ad esso, per ciascuna delle quali è specificato un dominio: le combinazioni possibili dei valori di queste variabili costituiscono i possibili stati locali che l'agente può assumere. Se nel sistema è definito un ambiente con delle variabili osservabili, l'agente ha la conoscenza del valore di queste variabili oltre che delle proprie,

per cui il suo stato locale dipende anche da questi valori (nonostante l'agente non possa modificarli direttamente).

Azioni. Ciascun agente ha un insieme di azioni che può compiere.

Protocollo. Il protocollo di un agente associa ad ogni suo stato l'insieme delle azioni possibili nello stesso; si può assegnare un insieme di azioni ad un insieme di stati individuato da una condizione sui valori delle variabili definite. Ad esempio si può dichiarare che in tutti gli stati in cui una variabile a vale 0 sono possibili le azioni α e β.

Funzione di evoluzione. La funzione di evoluzione di un agente descrive come il suo stato interno cambia in base allo stato e alle azioni compiute nel sistema. Questa funzione è descritta assegnando uno stato successivo a una condizione basata sul valore delle variabili locali dell'agente e di quelle osservabili dell'ambiente e sulle azioni eseguite da tutti gli agenti nel sistema. Dalle funzioni di evoluzione di ciascun agente, MCMAS ricava la funzione di evoluzione globale del sistema.

La descrizione di un sistema comprende poi l'insieme dei possibili stati iniziali, le proposizioni atomiche con le loro condizioni di verità, i gruppi di agenti (utilizzati in alcuni operatori CTLK), le condizioni di *fairness* (è possibile "costringere" il sistema a far verificare ogni tanto determinate condizioni) e le proprietà CTLK su cui si vuole eseguire il model checking.

4 Verifica di sistemi basati sul modello BDI

Nel modello di agente descritto sopra, si è considerato per ciascuno un insieme di stati generici: per facilitare la descrizione del comportamento degli agenti, può essere utile definire un modello in base al quale definire quali sono gli stati possibili e le transizioni tra essi. Una caratteristica importante degli agenti è l'*autonomia*, intesa come la capacità di scegliere in modo indipendente come comportarsi in risposta agli stimoli esterni e di agire anche in loro assenza (proattività). Per realizzare ciò, agli agenti sono dati degli stati mentali e un metodo di "ragionamento pratico" simile a quello umano: questo approcio è seguito nel modello software BDI (*Belief-Desire-Intention*) [8], utilizzato frequentemente nei sistemi multi-agente.

Nel modello BDI, ciascun agente è dotato di un insieme di *credenze* (*Beliefs*), basate sull'osservazione dell'ambiente circostante, e di *desideri* (*Desires*). Ciascun agente sceglie un insieme di desideri compatibili tra loro e li assume come *obiettivi* (*goals*): per perseguire i suoi obiettivi, un agente deve pianificare le azioni da compiere perché questi si realizzino. Analizzando una libreria predefinita di *piani*, che "spiegano" all'agente come agire sul mondo circostante per raggiungere determinati scopi, dal processo di ragionamento dell'agente sono generate delle *intenzioni* (*Intentions*), costituite da stack di piani le cui azioni portano progressivamente al raggiungimento di un obiettivo

prestabilito. I piani possono diventare intenzioni dell'agente in risposta a eventi interni, come l'assunzione di un nuovo obiettivo, o esterni, come l'arrivo di un messaggio inviato da un altro agente.

4.1 Modello

Come nel caso precedente, per modellare un sistema (composto da un insieme Ag di agenti), si utilizza la concezione del tempo secondo cui il futuro è ramificato in base alle diverse scelte che gli agenti possono compiere. In particolare, il passaggio da uno stato ad un altro di un sistema è causato da un *evento*: gli eventi primitivi sono quelli che fanno passare il sistema da uno stato ad un altro specifico tra i possibili successivi e possono essere concatenati per considerare eventi "composti" che collegano invece stati non adiacenti. In alcuni contesti si considera che gli eventi, considerati come azioni eseguite dagli agenti, possono avere successo o fallire (senza che l'agente sappia a priori l'esito).

In questo modello si possono considerare esplicitamente più alberi temporali, ciascuno dei quali è costituito da stati dell'agente e rappresenta un *mondo possibile*, inteso come uno dei possibili stati in cui si trovano tutte le parti del sistema esterne all'agente. Uno stato s può essere presente negli alberi temporali di più di un mondo: per questo lo stato "completo" di un agente in un sistema, ovvero quello ottenuto considerando sia il suo stato mentale che le condizioni dell'ambiente circostante, dipende sia dal mondo w che dallo stato s. Una coppia del tipo (w, s) è detta *situazione* e può essere scritta con la notazione s_w.

Si considera un agente i in una situazione s_w: l'agente sa con certezza che si trova nello stato s, ma non può essere sicuro che si trovi nel mondo w perché questo, dal suo punto di vista, appare identico ad altri. Ad s_w si può associare un insieme di mondi *accessibili per credenza*, che l'agente crede essere possibili in quella situazione; in pratica la relazione di accessibilità per credenza è simile a quella di accessibilità epistemica tra stati nel formalismo degli interpreted systems presentato in precedenza.

Come discusso sopra, ciascun agente ha dei desideri in base ai quali, selezionandone alcuni che siano compatibili tra loro e che l'agente considera possibili da soddisfare, si pone degli obiettivi: fissati questi, in una data situazione s_w, si può definire un insieme di mondi *accessibili per obiettivi*, che consistono in alberi temporali che portano in qualsiasi caso a raggiungerli. Dato che, in qualsiasi mondo in cui l'agente pensa di potersi trovare, deve esistere almeno una sequenza di eventi che porta al raggiungimento degli obiettivi, in ogni situazione s_w, per ciascun mondo v accessibile per credenze (ritenuto possibile), deve esistere ed essere accessibile per obiettivi un suo *sottomondo*, ovvero un mondo descritto da un albero temporale che sia "contenuto" in quello di v.

Dagli obiettivi di un agente scaturiscono le sue intenzioni: queste devono essere compatibili con gli obiettivi stessi, così come questi ultimi devono essere compatibili con ciò che l'agente ritiene possibile. I mondi *accessibili per intenzioni* in una situazione s_w rappresentano ciò che l'agente ha deciso di provare a realizzare: in modo simile

alla relazione tra credenze e obiettivi, per ciascun mondo v accessibile per obiettivi, deve esistere ed essere accessibile per intenzioni un suo sottomondo.

Ricapitolando, per ogni mondo che un agente ritiene possibile (accessibile per credenze), sono effettuate delle scelte che restringono l'albero temporale in modo da ottenere un corrispondente sottomondo accessibile per obiettivi, da cui si seleziona a sua volta un sottoalbero che costituisce il suo sottomondo accessibile per intenzioni, che delinea le azioni che l'agente proverà a compiere.

Conformemente a quanto detto, un agente di un sistema può essere descritto formalmente da una tupla $\langle W, E, S, T, L, \mathcal{B}, \mathcal{G}, \mathcal{I} \rangle$, dove

- W è l'insieme di tutti i mondi possibili,
- E è l'insieme di tutti gli eventi primitivi,
- S è l'insieme di tutti gli stati,
- $T \subseteq S \times S$ è la relazione di transizione tra gli stati,
- $L : W \times S \rightarrow 2^{AP}$ è la funzione di etichettatura degli stati con proposizioni atomiche,
- $\mathcal{B}, \mathcal{G}$ e $\mathcal{I}$ sono sottoinsiemi di $W \times S \times W$ che definiscono, per ciascuna situazione, quali sono i mondi accessibili rispettivamente per credenze, per obiettivi e per intenzioni.

Un mondo $w \in W$ è caratterizzato da un insieme $S_w \subseteq S$ degli stati contenuti nel suo albero temporale e una relazione di transizione $T_w \subseteq T$ tra gli stessi. Se si considerano formalmente gli eventi con due esiti possibili (successo o fallimento), si introducono anche le funzioni $\mathcal{S}_w$ e $\mathcal{F}_w$ di tipo $S_w \times S_w \rightarrow E$, che associano rispettivamente il buon esito o il fallimento di un evento ad ogni possibile transizione tra stati.

In precedenza, si è definita la relazione secondo cui un mondo w' è sottomondo di w ($w' \sqsubseteq w$) se e solo se l'albero temporale di w' è contenuto dentro a quello di w con le stesse proprietà (transizioni, etichette e relazioni di accessibilità): formalmente, ciò avviene se e solo se sono verificate le condizioni $S_{w'} \subseteq S_w$, $T_{w'} \subseteq T_w$ e $\forall s \in S_{w'} : L(w', s) = L(w, s), (w', s, v) \in \mathcal{B} \Rightarrow (w, s, v) \in \mathcal{B}, (w', s, v) \in \mathcal{G} \Rightarrow (w, s, v) \in \mathcal{G}, (w', s, v) \in \mathcal{I} \Rightarrow (w, s, v) \in \mathcal{I}$.

4.2 Estensione della CTL e $\mathcal{LORA}$

Dato il modello di un agente descritto nel modo discusso sopra, è possibile esprimere proprietà riferite ad esso attraverso la CTL. Per esprimere proprietà riguardanti lo stato mentale dell'agente, è necessario estendere la logica con operatori opportuni. La grammatica riportata sotto è quella della CTL con l'aggiunta di tre operatori di stato.

$$\Phi ::= \text{vero} \mid a \mid \neg\Phi \mid \Phi \wedge \Phi \mid \exists\vartheta \mid \forall\vartheta \mid B\Phi \mid G\Phi \mid I\Phi$$

$$\vartheta ::= \bigcirc\Phi \mid \Phi \cup \Phi$$

Nel contesto di un modello $\mathcal{M}$ del tipo discusso sopra (rappresentante un singolo agente), una formula di stato Φ è riferita ad una situazione s_w, ovvero ad una combinazione di un mondo w ed uno stato s appartenente al suo albero temporale (non solo ad uno stato, come nei formalismi precedenti): il significato degli operatori CTL è, prendendo come riferimento il solo albero temporale di w, lo stesso descritto in precedenza.

Gli operatori B, G e I sono basati sulle relazioni di accessibilità dell'agente in analisi: una situazione s_w (mondo w, stato s) soddisfa la proprietà $B\Phi$, $G\Phi$ o $I\Phi$ se e solo se Φ è soddisfatta, nello stesso stato s, in tutti i mondi v che sono accessibili dalla situazione s_w rispettivamente per credenza ($(w, s, v) \in \mathcal{B}$), per obiettivo ($(w, s, v) \in \mathcal{G}$) o per intenzione ($(w, s, v) \in \mathcal{I}$). Formalmente, il significato dell'operatore B si definisce nel modo seguente (le definizioni per G e I sono analoghe).

$$\mathcal{M}, w, s \models B\Phi \Leftrightarrow \forall v \in W, (w, s, v) \in \mathcal{B} : \mathcal{M}, v, s \models \Phi$$

In pratica, $s_w \models B\Phi$ significa che l'agente crede che Φ sia vera nella situazione s_w: ciò è vero se Φ si verifica nello stato s in tutti i mondi che ritiene possibili nella stessa. Allo stesso modo $G\Phi$ significa che l'agente ha come obiettivo (a lungo termine) fare in modo che Φ sia vera e $I\Phi$ significa che l'agente ha assunto come intenzione di rendere Φ vera.

Operatori uguali o simili a quelli presenti in questa logica si possono ritrovare in $\mathcal{LORA}$ (*Logic Of Rational Agents*) [10], una logica del prim'ordine specifica per esprimere proprietà sui sistemi multi-agente (non solo su un singolo agente). In particolare, sono definiti gli operatori Bel i Φ, Des i Φ e Int i Φ, con cui asserire che un agente i rispettivamente creda, desideri e intenda fare in modo che Φ sia verificata. Considerando formalmente un insieme di azioni che gli agenti possono compiere, è definito l'operatore di percorso Happens α ad indicare l'esecuzione imminente dell'azione α.

4.3 MABLE

MABLE [9] è un linguaggio di programmazione specifico per sistemi multi-agente. Tramite questo linguaggio è possibile descrivere un sistema multi-agente specificando da quali agenti è composto e qual'è il loro comportamento: per far ciò il linguaggio fornisce i costrutti classici dei linguaggi di programmazione e altri specifici per gestire lo stato mentale degli agenti e farli comunicare tra loro. Dato un sistema descritto in MABLE, è possibile esprimerne delle proprietà in un sottoinsieme di $\mathcal{LORA}$ denominato $\mathcal{MORA}$. La verifica di un modello MABLE è eseguita attraverso il model checker

SPIN: una volta scritta la specifica del sistema e le proprietà da verificare, un apposito compilatore riscrive la prima in PROMELA, il linguaggio utilizzato da SPIN, e le seconde in LTL, in modo che possano essere interpretate dallo stesso.

4.3.1 Linguaggio e semantica

La specifica di un sistema multi-agente in MABLE è data principalmente da un insieme di *programmi* e un insieme di agenti, per ciascuno dei quali è specificato quale dei programmi definiti ne delinea il comportamento. Ciascun programma è composto da una sequenza di istruzioni, che possono contenere i classici costrutti di selezione (`if ... then ... else`, ...) e iterazione (`while`, `for`, ...).

In MABLE si possono dichiarare variabili di tre tipi: le variabili *locali* costituiscono lo stato interno di un agente e sono private, mentre alle variabili *condivise* e a quelle *globali* possono accedere tutti gli agenti. La differenza tra gli ultimi due tipi è data dal fatto che ciascun agente è sempre aggiornato sul valore effettivo delle variabili condivise, mentre per ciascuna variabile globale il valore creduto vero è l'ultimo che è stato impostato dall'agente stesso oppure che è stato letto tramite un'apposita istruzione `observe`.

Nelle condizioni dei costrutti `if` e `while`, è possibile utilizzare, oltre ai classici operatori di confronto (esempi: `x==5`, `y>0`), gli operatori modali di $\mathcal{LORA}$ sullo stato mentale degli agenti (esempio: "se l'agente *i* crede che `x==5` ..."). Il costrutto `if ... then ... else` può inoltre essere completato dalla clausola `unsure`, con cui specificare un blocco di istruzioni da eseguire se l'agente, in base alle sue credenze, non sa con certezza se la condizione data sia vera o falsa.

Lo stato mentale di un agente può essere manipolato esplicitamente (tramite le istruzioni `assert` e `retract` analoghe alle omonime clausole Prolog) e influenzato dalle comunicazioni con altri agenti. Le istruzioni `send` e `receive` servono rispettivamente a inviare e attendere la ricevezione di un messaggio ed hanno come argomenti il tipo di messaggio (o di *atto di comunicazione*), l'agente destinatario o mittente (uno solo) e una condizione che funge da contenuto. Le condizioni necessarie per l'invio di un qualsiasi tipo di messaggio e l'effetto della sua ricezione sullo stato mentale di un agente è definito in un file esterno al programma.

Si considera ad esempio un tipo di messaggio con cui un agente *m* (mittente) informa un agente *d* (destinatario) che una condizione Φ è vera: è possibile definire una semantica dell'azione "informare" secondo la quale l'agente *d* si fida dell'agente *m* e gli crede (per cui l'effetto del messaggio è Bel *d* Φ, "*d* crede Φ") oppure una secondo la quale l'agente *d* si limita a considerare che l'agente *m* ha intenzione di fargli credere Φ (Bel *d* Int *m* Bel *d* Φ).

4.4 AgentSpeak

Il termine AgentSpeak è utilizzato per indicare un insieme di linguaggi di programmazione largamente utilizzati per la programmazione ad agenti basati sul modello BDI. Di seguito sono riassunte la sintassi e la semantica di AgentSpeak(L) [7], sul quale sono basate diverse varianti utilizzate nella pratica.

Per rappresentare le proprietà del sistema e le credenze degli agenti, AgentSpeak utilizza termini del prim'ordine (del tipo $p(t_1, t_2, \ldots)$) e formule con gli operatori $\neg$ e $\wedge$. Gli obiettivi sono di due tipi: $!t$ indica che l'agente vuole far sì che t (termine del prim'ordine) si verifichi, mentre $?t$ indica che l'agente vuole sapere se t è vero. Le azioni compiute da un agente sono dettate da una libreria di piani: ciascun piano è costituito da un evento di attivazione, che consiste nell'aggiunta o nella rimozione di una credenza o un obiettivo, un contesto, che rappresenta delle credenze che l'agente deve avere per eseguire il piano, e un corpo, che consiste in una sequenza di azioni. Un azione può consistere nella modifica delle credenze, nell'aggiunta di un nuovo obiettivo o in un intervento sull'ambiente circostante (azione esterna).

Il comportamento di un agente segue un ciclo descritto di seguito. Per prima cosa tutti gli eventi osservabili generati internamente (modifica del proprio stato mentale) o esternamente (osservazione di cambiamenti nell'ambiente) sono inseriti in un insieme E, dal quale ne viene selezionato uno in base ad una funzione $\mathcal{S}_e$ e rimosso. Un'ulteriore funzione $\mathcal{S}_o$ seleziona una tra le opzioni disponibili, che consistono nei piani dell'insieme P i cui eventi di attivazione sono unificabili con l'evento selezionato e i cui contesti sono compatibili con le credenze dell'agente. Dal piano selezionato e dalle relative unificazioni si ottiene un'intenzione, in forma di uno stack di piani parzialmente istanziati, che è aggiunta ad un insieme I. A questo punto con una funzione $\mathcal{S}_i$ una di queste intenzioni è selezionata e la prima azione (aggiornamento dello stato interno o azione sull'esterno) del piano in cima allo stack è eseguita e rimossa dallo stesso.

4.4.1 CASP

CASP (*Checking AgentSpeak Programs*) [2] è un insieme di strumenti mirati al model checking di sistemi costituiti da agenti implementati in AgentSpeak. Per la precisione, il linguaggio riconosciuto da CASP è detto AgentSpeak(F) ed è una variante di AgentSpeak(L) tale da consentirne la verifica automatica.

Il motivo principale per cui non è possibile eseguire la verifica su un generico modello in AgentSpeak(L) è la necessità di avere un numero finito di stati. Per soddisfare questo requisito del model checking, sono introdotti alcuni parametri che rappresentano il massimo numero di elementi di AgentSpeak in diversi contesti: ad esempio uno di questi è M_{Bel}, che rappresenta il massimo numero di termini che l'insieme di credenze di un agente può contenere. Altre limitazioni di AgentSpeak(F) sono l'impossibilità di utilizzare termini del prim'ordine l'uno dentro l'altro e variabili non istanziate in alcuni contesti. Come in altre varianti di AgentSpeak, in questa sono introdotte del-

le *azioni interne*, denotate dal carattere "." iniziale, utilizzate in questo caso per la comunicazione tra agenti.

Le proprietà da verificare sul sistema sono espresse tramite una versione semplificata di $\mathcal{LORA}$, in modo che possano essere tradotte in formule LTL. Nella loro versione originale, gli strumenti di CASP possono riscrivere il modello nel linguaggio PROMELA e le proprietà in LTL, in modo che possano essere gestite poi dal model checker SPIN. In seguito, è stata introdotta la possibilità di ottenere una traduzione del modello in Java, più semplice rispetto a quella in PROMELA per via della maggiore potenza espressiva, che può essere analizzata tramite model checker specifici per Java come JPF (*Java PathFinder*).

Riferimenti bibliografici

[1] C. Baier, J.P. Katoen, et al. *Principles of Model Checking*. The MIT Press, 2008.

[2] R.H. Bordini, M. Fisher, W. Visser, and M. Wooldridge. Verifying multi-agent programs by model checking. *Autonomous Agents and Multi-Agent Systems*, 12(2):239–256, 2006.

[3] E. Clarke and E. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. *Logics of Programs*, pages 52–71, 1982.

[4] R. Fagin, JY Halpern, Y. Moses, and MY Vardi. Reasoning about knowledge, 1995.

[5] A. Lomuscio, H. Qu, and F. Raimondi. MCMAS: A model checker for the verification of multi-agent systems. In *Computer Aided Verification*, pages 682–688. Springer, 2009.

[6] A. Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science*, pages 46–57. IEEE, 1977.

[7] A. Rao. AgentSpeak (L): BDI agents speak out in a logical computable language. *Agents Breaking Away*, pages 42–55, 1996.

[8] A.S. Rao and M.P. Georgeff. Modeling rational agents within a BDI-architecture. *Readings in agents*, pages 317–328, 1997.

[9] M. Wooldridge, M.P. Huget, M. Fisher, and S. Parsons. Model checking for multiagent systems: The MABLE language and its applications. *International Journal of Artificial Intelligence Tools*, 15(2):195–226, 2006.

[10] M.J. Wooldridge. *Reasoning about rational agents*. The MIT Press, 2000.