A Logic Programming Language for Multi-Agent Systems

Stefania Costantini
Arianna Tocchio
Università degli Studi di L'Aquila
Dipartimento di Informatica

Jelia 2002,
University of Calabria

DALI: a Logic Programming language for Agents and Multi-Agent Systems

Very similar to Prolog

Easy-to-use

Logical Semantics

Novel features: different classes of events, their interaction, the interleaving of different activities, a concept of time.

Reactive Agents in Logic Programming

Kowalski & Sadri
rational and reactive agents

- observe–think–act cycle
- condition–action rules as integrity constraints
- IFF proof procedure proposed by Fung and Kowalski: actions and events as abducibles

Dell'Acqua & Sadri & Toni
rational and reactive agents plus proactivity (hybrid agents).

ConGolog: multi-agent Prolog-like language with imperative features based on situation calculus.

METATEM: reactive agents are logic programs executed asynchronously (with a notion of time), that communicate via message-passing.

Agents in Computational Logic

Belief – Desire – Intention (BDI): agents situated in a changing environment that they may want to affect

AgentSpeak(L): purely reactive logic language with external events and actions, meant to (indirectly) model BDI features in a simple way.

3APL rule-based, planning-oriented, has no concept of event.

IMPACT: very extensive approaches for Multi-Agent-Systems.

Updating Logic Programs (ULP)
represents state evolution of agents as program evolution controlled by means of the special-purpose meta-language LUPS

EPI: extension of LUPS that takes into account external events.

The DALI Project

Long-term objectives of this research:

identification and the formalization of basic patterns for

- reactivity
- proactivity
- internal “thinking”
- “memory”.

The DALI solution: different kinds of events, with a suitable treatment.

A DALI program is syntactically very close to a Prolog program.

Novel approach to the language semantics: evolutionary semantics.

DALI: Perspectives on External Events

$alarm\_clock\_rings$

Stage 1: event perceived, but no reaction yet.
Function: reasoning about what's happening.
Notation: *present event,* written $alarm\_clock\_ringsN$.

Stage 2: reaction to the event
Function: triggering an activity according to the event
Notation: *external event,* written $alarm\_clock\_ringsE$.

Stage 3: after reaction, the agent is able to remember the event
Function: reasoning about what happened in the past.
Notation: *past event,* written $alarm\_clock\_ringsP$.

DALI: Internal Events

An internal event is a conclusion reached by the agent, interpreted as an event

The agent reacts to internal events like to the external ones

Role: internal conclusions may trigger further inference.

E.g., food is finished? Possible reaction: go to buy other food.

Notation: $food\_is\_finishedI$.

Also internal events become past events.

DALI: Rules

DALI rules are in the form of Horn clauses

reactive rules syntactically emphasized.

Head of a reactive rule: an event

Body: agent's reaction to the event

Syntactic form: $E{:>}Reaction$”

$$\textit{customer\_entersE} \quad :> \quad \textit{say\_good\_morningA}, \textit{offer\_helpA}.$$

More features about External events

In the implementation, events are time-stamped, and the order in which they are "consumed "corresponds to the arrival order.

The time-stamp can be useful for introducing into the language some (limited) possibility of reasoning about time.

$$\textit{customer\_entersE} \mathbin{\textbf{:}} \textit{T} \quad :> \quad \textit{lunchtime(T)}, \textit{offer\_appetizersA}.$$

Conjunction of events in the head of a reactive rule:

$$\textit{rainE, windE} \quad :> \quad \textit{close\_windowA}.$$

Present events, and Actions

Distinction between *reasoning* about events and *reacting* to events.

$$\begin{aligned} \mathit{visitor\_arrived} &\quad \text{:-} \quad \mathit{bell\_ringsN}. \\ \mathit{bell\_ringsE} &\quad :> \quad \mathit{open\_doorA}. \end{aligned}$$

Action atoms represent actions without preconditions:
always succeed.

The action cannot properly affect the environment:
the interpreter might generate a “failure event”

Action Rules

Actions with preconditions: action atoms are defined by *action rules*.

$$\begin{array}{rcl} \textit{need\_supplyE(P)} & :> & \textit{emit\_oder(P)}. \\ \textit{emit\_oder(P)} & \text{:-} & \textit{phone\_order}A. \\ \textit{emit\_oder(P)} & \text{:-} & \textit{fax\_order}A. \\ \textit{fax\_order}A & \text{:-} & \textit{fax\_machine\_available}. \end{array}$$

Action $ActA$ becomes a past action $ActPA$.

Communication

Communication: an external event can be a message from another agent, and, symmetrically, an action can consist in sending a message.

For now: no commitment to particular Agent Communication language

Attached to each event atom: the name of the senderagent

Events like $rainsE$: default indication $environment$.

Event atom (external/past) more precise:

$$Sender : Event\_Atom : Timestamp$$

Past Events: Memory

$$\begin{aligned}
\textit{sunny_weatherE} &\;:>\; \textit{open_the_windowA}.\\
\textit{rainy_weatherE} &\;:>\; \textit{close_the_windowA}.\\
\textit{open_the_windowA} &\;:\text{-}\; \textit{window_is_closed}.\\
\textit{window_is_closed} &\;:\text{-}\; \textit{close_the_windowPA}.\\
\textit{close_the_windowA} &\;:\text{-}\; \textit{window_is_open}.\\
\textit{window_is_open} &\;:\text{-}\; \textit{open_the_windowPA}.
\end{aligned}$$

Past events and actions are kept according to directives.

Amount of time / forever / terminating condition:

$$\begin{aligned}
keep\ \textit{shop_openPE} &\quad until\ \textit{shop_closed}.\\
keep\ \textit{open_the_windowPA} &\quad until\ \textit{close_the_windowA}.
\end{aligned}$$

Internal Events: Proactivity

Absolute novelty in the context of agent languages.

$$
\begin{array}{rcl}
\textit{finished(Food)} & :- & \textit{eaten(Food)}. \\
\textit{finishedI(Food)} & :> & \textit{go\_to\_buyA(Food,Where)}. \\
\textit{go\_to\_buyA(Food,bakery)} & :- & \textit{bread\_or\_biscuit(Food)}. \\
\textit{go\_to\_buyA(Food,grocery\_shop)} & :- & \textit{dairy(Food)}.
\end{array}
$$

Goals corresponding to internal events are automatically attempted from time to time (default or set by directives).

A goal that should be tried often:

$$
\begin{array}{rcl}
\textit{too\_high(Temperature)} & :- & \textit{threshold(Th), Temperature} > \textit{Th}. \\
\textit{too\_high(Temperature)I} & :> & \textit{start\_emergencyA, alert\_operatorA}.
\end{array}
$$

Procedural Semantics

Procedural semantics of DALI: extension to SLD–resolution.

A goal in DALI is a *disjunction* $G^1; G^2; \ldots; G^n$ of *component goals*.

Every G^k is a goal as usually defined in the Horn–clause language, i.e. a conjunction.

Meaning: the computation fails only if all disjuncts fail.

Procedural Semantics

The procedural behavior of a DALI agent consists of the interleaving of the following steps.

1. Trying to answer a user's query like in plain Horn–clause language.
2. Responding to either external or internal events. This means, the interpreter picks up either an external event from EV or an internal event form IV, and adds this event G^{ev} as a new query, i.e. as a new disjunct in the present goal. Thus, goal $G^1; G^2; \ldots ; G^n$ becomes $G^1; G^2; \ldots ; G^n; G^{ev}$, and G^{ev} is inserted into PV.
3. Trying to prove a goal corresponding to an internal event. The interpreter picks up an atom from EVT, and adds this atom G^{evt} as a new query, i.e. as a new disjunct in the present goal. Thus, goal $G^1; G^2; \ldots ; G^n$ becomes $G^1; G^2; \ldots ; G^n; G^{evt}$.

Declarative Semantics

Declarative semantics of DALI program P based on standard declarative semantics (Least Herbrand Model)

Modified program P_s, obtained from P by means of syntactic transformations that specify how the different classes of events are coped with.

P_s is the basis for the **evolutionary semantics**, that describes how the agent is affected by actual arrival of events.

Modelling External events

A reactive rule can be applied only if the corresponding event has happened.

How to represent that: by adding, for each event atom $p(Args)E$, the event atom itself in the body of its own reactive rule.

We transform the reactive rule

$$p(Args)E :> R_1, \ldots, R_q.$$

into the standard rule:

$$p(Args)E :\text{-}\ p(Args)E, R_1, \ldots, R_q.$$

Modelling Internal events

The reactive rule corresponding to an internal event $q(Args)I$ is allowed to be applied only if the subgoal $q(Args)$ has been proved.

How to represent that: we transform each reactive rule for internal events:

$$q(Args)I :> R_1, \ldots, R_q.$$

into the standard rule:

$$q(Args)I :\text{-}\ q(Args), R_1, \ldots, R_q.$$

Modelling Actions

Given a rule of the form

$$B \text{ :- } D_1, \ldots, D_h, A_1, \ldots, A_k. \;\; h \geq 1, k \geq 1$$

whenever the conditions $D_1, \ldots, D_h$ of the above rule are true, the action atoms should become true as well (given their preconditions, if any).

How to represent that: for every action atom A, with action rule

$A \text{ :- } C_1, \ldots, C_s. \;\; s \geq 1$

we modify this rule into:

$A \text{ :- } D_1, \ldots, D_h, C_1, \ldots, C_s.$

If A has no defining clause, we instead add clause:

$A \text{ :- } D_1, \ldots, D_h.$

Agent evolution according to events

Program P_s is actually affected by the events, by means of subsequent syntactic transformations.

Declarative semantics of agent program P at a certain stage: declarative semantics of the version of P_s at that stage.

$P_0 = \langle P_s, [] \rangle$ (initially no event has happened).

$P_n = \langle Prog_n, Event\_list_n \rangle$,

where $Event\_list_n$ is the list of the n events that have happened, and $Prog_n$ is the current program

$Prog_n$ obtained from P_s step by step by means of a *transition function* Σ.

Program snapshot at step n

$$P_n = \Sigma(P_{n-1}, E_n)$$

Definition 1 *The* transition function Σ *is defined as follows.*

$$\Sigma(P_{n-1}, E_n) = \langle \Sigma_P(P_{n-1}, E_n), [E_n | Event\_list_{n-1}] \rangle$$

where

$$\Sigma_P(P_0, E_1) = \Sigma_P(\langle P_s, [] \rangle, E_1) = P_s \cup E_1 \cup E_{1N}$$

$$\Sigma_P(\langle Prog_{n-1}, [E_{n-1}|T] \rangle, E_n) =$$

$$\{\{Prog_{n-1} \cup E_n \cup E_{nN} \cup E_{n-1PE}\} \setminus E_{n-1N}\} \setminus E_{n-1}$$

Program Evolution

Definition 2 *Let* P_s *be a DALI program, and* $L = [E_n, \ldots, E_1]$ *be a list of events. Let* $P_0 = \langle P_s, [] \rangle$ *and* $P_i = \Sigma(P_{i-1}, E_i)$ *The list* $\mathcal{P}(P_s, L) = [P_0, \ldots, P_n]$ *is the* program evolution *of* P_s *with respect to* L*.*

Notice that $P_i = \langle Prog_i, [E_i, \ldots, E_1] \rangle$, where $Prog_i$ is the program as it has been transformed after the ith application of Σ.

Model Evolution

Definition 3 *Let* P_s *be a DALI program,* L *be a list of events, and* PL *be the* program evolution *of* P_s *with respect to* L*. Let* M_i *be the Least Herbrand Model of* $Prog_i$*. The sequence* $\mathcal{M}(P_s, L) = [M_0, \ldots, M_n]$ *is the* model evolution *of* P_s *with respect to* L*, and* M_i *the* instant model at step i *.*

Evolutionary Semantics

Models the history of the events received by the agents, and of the effect they have produced on it.

Definition 4 *Let* P_s *be a DALI program,* L *be a list of events. The* evolutionary semantics $\mathcal{E}_{P_s}$ *of* P_s *with respect to* L *is the couple* $\langle \mathcal{P}(P_s, L), \mathcal{M}(P_s, L) \rangle$.

Theorem 1 *Let* P_s *be a DALI program,* $L = [E_n, \ldots, E_1]$ *be a list of events and* P_n *be the program snapshot at step* n*. DALI resolution is correct and complete with respect of* P_n*.*

The evolutionary semantics can be extended to DALI multi-agent programs, by considering the evolutionary semantics of all agents involved.

Remarks

Main objective in the design of DALI: understanding whether modelling agents in pure logic programming was possible, and to which extent.

Syntax and semantics: very close to the Horn clause language

For practical applications we are integrating into the language:

- an agent communication language
- primitives for coordination and cooperation
- meta-programming facilities
- specific features for planning: Answer Set Programming

DALI in Practice

DALI is fully implemented

DALI is being applied:

- Component-based Software Engineering
- Network Security