Specifying and Verifying Systems of Communicating Agents in a Temporal Action Logic

Laura Giordano[1], Alberto Martelli[2], Camilla Schwind[3]

[1]Dipartimento di Informatica, Università del Piemonte Orientale, Alessandria

[2]Dipartimento di Informatica, Università di Torino, Torino

[3]MAP, CNRS, Marseille, France

Summary

- We present a logical framework for specifying and verifying systems of communicating agents.

Summary

- We present a logical framework for specifying and verifying systems of communicating agents.
- The framework is based on a Dynamic Linear Time Temporal Logic (DLTL) and provides a simple formalization of the communicative actions in terms of their effects and preconditions.

Summary

- We present a logical framework for specifying and verifying systems of communicating agents.
- The framework is based on a Dynamic Linear Time Temporal Logic (DLTL) and provides a simple formalization of the communicative actions in terms of their effects and preconditions.
- We show how to model interaction protocols based on a social approach: communication can be described in terms of changes in the social state, and protocols in terms of creation, manipulation and satisfaction of commitments among agents.

Specification

- The description of the interaction protocol and of communicative actions is given in a temporal action theory.
- Communicative actions are formalized in terms of effects and preconditions.
- Interaction protocols are specified with temporal constraints representing permissions and commitments.
- Agent programs, when known, can be formulated in DLTL as regular programs.

Verification

- Several kinds of verification problems (including the problem of compliance of agents to the protocol) can be formalized either as validity or as satisfiability problems in the temporal logic.
- Such verification tasks can be automated by translating DLTL formulas into Büchi automata, and checking for the emptiness of the language accepted by the automaton.
- We have developed a tableau-based algorithm [TIME04] for constructing a Büchi automaton from a DLTL formula "on the fly".
- The number of states of the automaton is, in the worst case, exponential in the size of the input formula. Check for emptiness can be done in linear time.

The logical framework

- A theory for reasoning about communicative actions in a multiagent system based on Dynamic Linear Time Temporal Logic ($DLTL$), an extension of LTL (the propositional linear time temporal logic).
- $DLTL$ extends LTL by strengthening the *until* operator by indexing it with the regular programs of dynamic logic. It is, essentially, a dynamic logic equipped with a linear time semantics.
- A Product Version of Dynamic Linear Time Temporal Logic ($DLTL^{\otimes}$), allows to model multiagent systems, by decorating formulas with the names of sequential agents. Synchronization is achieved by means of shared actions.

DLTL (Henriksen, Thiagarajan)

Σ be a finite non-empty alphabet of **actions**.
Σ^ω the set of infinite words on Σ.

$Prg(\Sigma)$ the set of **programs** (regular expressions)

$$Prg(\Sigma) ::= a \mid \pi_1 + \pi_2 \mid \pi_1; \pi_2 \mid \pi^*, \text{ where } a \in \Sigma$$

$[[\pi]]$ represents the set of executions of π

The set of formulas of DLTL(Σ):

$$\text{DLTL}(\Sigma) ::= p \mid \neg\alpha \mid \alpha \vee \beta \mid \alpha \mathcal{U}^\pi \beta$$

where $p \in \mathcal{P}$ are atomic propositions

DLTL

A model of DLTL(Σ) is a pair $M = (\sigma, V)$ where $\sigma \in \Sigma^\omega$ and $V : prf(\sigma) \to 2^{\mathcal{P}}$ is a valuation function.

Given a model $M = (\sigma, V)$, and a prefix τ of σ, we can define the satisfiability of a formula at τ in M. In particular:

$M, \tau \models \alpha \mathcal{U}^\pi \beta$ iff there exists $\tau' \in [[\pi]]$ such that

- $\tau\tau' \in prf(\sigma)$
- $M, \tau\tau' \models \beta$
- $M, \tau\tau'' \models \alpha$ for every prefix τ'' of τ'

DLTL

We can define derived modalities

- $\langle \pi \rangle \alpha \equiv \top \mathcal{U}^{\pi} \alpha$
- $[\pi] \alpha \equiv \neg \langle \pi \rangle \neg \alpha$
- $\bigcirc \alpha \equiv \bigvee_{a \in \Sigma} \langle a \rangle \alpha$ (next)
- $\Diamond \alpha \equiv \top \mathcal{U}^{\Sigma^*} \alpha$
- $\Box \alpha \equiv \neg \Diamond \neg \alpha$

$DLTL^{\otimes}$ (Henriksen, Thiagarajan)

- There are n *agents*, each with an alphabet Σ_i of actions.
- The alphabets are not disjoint. Actions shared by agents represent synchronized actions.
- The until operator is indexed with the name of an agent: $\mathcal{U}_i^{\pi}$
- Modal subformulas can only deal with one agent.
- In a model $M = (\sigma, V)$, σ is an infinite sequence of actions of all agents. Formulas concerning agent i are evaluated locally, i.e. by projecting σ on the actions of agent i.

Proof of DLTL formulas

- The satisfiability problem for DLTL can be solved in deterministic exponential time, as for LTL, by constructing for each formula $\alpha \in DLTL(\Sigma)$ a Büchi automaton $\mathcal{B}_\alpha$ such that the language of ω-words accepted by $\mathcal{B}_\alpha$ is non-empty if and only if α is satisfiable.
- There is a one to one correspondence between models of the formula (infinite sequences of actions) and infinite words accepted by $\mathcal{B}_\alpha$.
- The validity of a formula α can be verified by constructing the Büchi automaton $\mathcal{B}_{\neg\alpha}$ for $\neg\alpha$: if the language accepted by $\mathcal{B}_{\neg\alpha}$ is empty, then α is valid, whereas any infinite word accepted by $\mathcal{B}_{\neg\alpha}$ provides a counterexample to the validity of α.

Automaton construction

A *Büchi automaton* over Σ is a tuple $\mathcal{B} = (Q, \rightarrow, Q_{in}, F)$ where:

- Q is a finite nonempty set of states;
- $\rightarrow \subseteq Q \times \Sigma \times Q$ is a transition relation;
- $Q_{in} \subseteq Q$ is the set of initial states;
- $F \subseteq Q$ is a set of accepting states.

Let $\sigma \in \Sigma^{\omega}$. Then a run of $\mathcal{B}$ over σ is a map $\rho : prf(\sigma) \rightarrow Q$ such that:

- $\rho(\varepsilon) \in Q_{in}$
- $\rho(\tau) \xrightarrow{a} \rho(\tau a)$ for each $\tau a \in prf(\sigma)$

A run is *accepting* iff if contains infinitely many times an accepting state.
$\mathcal{L}(\mathcal{B})$ is the language of ω-words accepted by $\mathcal{B}$.

Action theory (AI*IA 2001)

An action theory can be given by defining
Action laws

$$\Box(\alpha \rightarrow [a]\beta)$$

Precondition laws

$$\Box(\alpha \rightarrow [a]\bot)$$

Causal laws

$$\Box([a]\alpha \rightarrow [a]\beta)$$

Persistency is modelled by a completion construction (Reiter).

Interaction protocols

We adopt a social approach, where communicative actions affect the "social state" of the system, rather than the internal states of the agents. The social state records the social facts, like the permissions and the commitments of the agents, which are created and modified in the interactions among them.

This protocol does not require the rigid specification of all the allowed action sequences, e.g. by means of finite state diagrams.

The action theory can provide a high level specification of the protocol.

Specifying protocols

In our action theory the effects of communicative actions will be modeled by *action laws*.

Permissions, which determine when an action can be taken by each agent, can be modeled by *precondition laws*.

Commitment policies, which rule the dynamic of commitments, can be described by *causal laws* which establish the causal dependencies among fluents.

The specification of a protocol can be further constrained through the addition of suitable *temporal formulas*.

The agents' programs can be modeled by making use of complex actions (*regular programs*).

Contract net

The Contract Net protocol begins with an agent (the manager) broadcasting a task announcement (call for proposals) to other agents viewed as potential contractors (the participants). Each participant can reply by sending either a proposal or a refusal. The manager must send an accept or reject message to all those who sent a proposal. When a contractor receives an acceptance it is committed to perform the task.

Contract net

We assume to have only two **agents**: M, the manager, and P, the participant.

Actions sent by M: $cfp(T)$, $accept$, $reject$
Actions sent by P: $refuse$, $propose$
Fluents: $task, replied, proposal, acc\_rej$

$task$ is a *functional fluent* ($task = T$)

Commitments

Commitments are special fluents

- **base-level commitments**: $C(ag_1, ag_2, \alpha)$ (agent ag_1 is committed to agent ag_2 to bring about α)
- **conditional commitments**: $CC(ag_1, ag_2, \beta, \alpha)$ (agent ag_1 is committed to agent ag_2 to bring about α, if the condition β is brought about)

In the Contract Net we have the commitments:

$C(P, M, replied)$ and $C(M, P, acc\_rej)$

and conditional commitments

$CC(P, M, task \neq nil, replied)$ and
$CC(M, P, propose, acc\_rej)$.

Rules for commitments

We introduce some reasoning rules for commitments:

$$\Box(\bigcirc\alpha \rightarrow \bigcirc\neg C(i,j,\alpha))$$
$$\Box(\bigcirc\alpha \rightarrow \bigcirc\neg CC(i,j,\beta,\alpha))$$
$$\Box((CC(i,j,\beta,\alpha) \wedge \bigcirc\beta) \rightarrow \bigcirc(C(i,j,\alpha) \wedge \neg CC(i,j,\beta,\alpha)))$$

A commitment (or a conditional commitment) to bring about α is cancelled when α holds, and a conditional commitment $CC(i,j,\beta,\alpha)$ becomes a base-level commitment $C(i,j,\alpha)$ when β has been brought about.

Action laws

$$\Box[cfp(T)]task = T$$
$$\Box[cfp(T)]CC(M, P, proposal, acc\_rej)$$
$$\Box[accept]acc\_rej$$
$$\Box[reject]acc\_rej$$
$$\Box[refuse]replied$$
$$\Box[propose](replied \wedge proposal)$$

Permissions

The permissions to execute communicative actions are represent by precondition laws.

$$\Box(\neg proposal \vee acc\_rej \rightarrow [accept]\bot)$$
$$\Box(\neg proposal \vee acc\_rej \rightarrow [reject]\bot)$$
$$\Box(task = nil \vee replied \rightarrow [refuse]\bot)$$
$$\Box(task = nil \vee replied \rightarrow [propose]\bot)$$
$$\Box(task \neq nil \rightarrow [cfp(T)]\bot).$$

We will call $Perm_i$ (permissions of agent i) the set of all the precondition laws of the protocol pertaining to the actions of which agent i is the sender.

Initial state and commitments

The *initial state* of the protocol:

$$\{task = nil, \neg replied, \neg proposal,$$
$$CC(P, M, task \neq nil, replied), C(M, P, task \neq nil)\}$$

We can express the condition that the commitment $C(i, j, \alpha)$ has been fulfilled by the following constraint:

$$\Box(C(i, j, \alpha) \rightarrow \Diamond\alpha)$$

We will call Com_i the set of constraints of this kind for all commitments of agent i. Com_i states that agent i will fulfill all the commitments of which he is the debtor.

Specifying a protocol

The protocol can be described for each agent as:

- **Action and causal laws** Π (suitably completed to cope with persistency $Comp(\Pi)$)
- **Initial state** $Init$
- **Permissions** $Perm_i$ for each agent i (precondition laws)
- **Commitment conditions** Com_i for each agent i

Domain description

A domain description D, can be defined by the formula:

$$D = (Comp(\Pi) \wedge Init \wedge \bigwedge_i (Perm_i \wedge Com_i))$$

The runs of the system according the protocol are the linear models of $Comp(D)$. Observe that in these protocol runs all permissions and commitments have been fulfilled.

Note also that all sequences of actions accepted by the Büchi automaton obtained from the above formula represent all correct executions of the given protocol.

Verification

Given an execution history describing the interactions of the agents, we want to verify the compliance of that execution with the protocol.

This verification is carried out at runtime.

We are given a history $\tau = a_1, \ldots, a_n$ of the communicative actions executed by the agents, and we want to verify that the history τ is the prefix of a run of the protocol, that is, it respects the permissions and commitments of the protocol.

This problem can be formalized as satisfiability of the formula

$$(Comp(\Pi) \wedge Init \wedge \bigwedge_i (Perm_i \wedge Com_i)) \wedge < a_1; a_2; \ldots; a_n > \top$$

(where i ranges on all the agents involved in the protocol).

Verification

Proving a property φ of a protocol.

This can be formulated as the validity of the formula

$$(Comp(\Pi) \wedge Init \wedge \bigwedge_i (Perm_i \wedge Com_i)) \rightarrow \varphi$$

Observe that, to prove the property φ, all the agents are assumed to be compliant with the protocol.

Verification

Verify that an agent is compliant with the protocol, given the program executed by the agent itself.
We can specify the behavior of an agent by making use of complex actions (regular programs). The following program π_P describes the behavior of the participant:

$$[\neg done?; ((cfp(T); eval\_task; (\neg ok?; refuse; exit+ \\ ok?; propose))+ \\ (reject; exit)+ \\ (accept; do\_task; exit))]^*; done?$$

The state of the agent is obtained by adding to the fluents of the protocol, the local fluents: $done$, $exit$, and ok. The local actions are $eval\_task$, do_task and $exit$. Furthermore, $done?$ and $ok?$ are test actions.

Domain description

The program of the participant can be specified by a domain description $Prog_P = (\Pi_P, \mathcal{C}_P)$, where Π_P is a set of action laws describing the effects of the private actions of the contractor, for instance:

$$\Box[exit]done$$
$$\Box(task = t1 \rightarrow [eval\_task]ok)$$
$$\Box(task = t2 \rightarrow [eval\_task]\neg ok)$$

and, $\mathcal{C}_P = \{\langle \pi_P \rangle \top, \neg done, \neg ok\}$ contains the constraints on the initial values of fluents ($\neg done, \neg ok$) as well as the formula $\langle \pi_P \rangle \top$ stating that the program of the participant is executable in the initial state.

Compliance verification

The property that the participant is compliant with the protocol, i.e. that all executions of program π_P satisfy the specification of the protocol, cannot be proved by considering only the program π_P.

The correctness of the property depends on the behavior of the manager. For instance, if the manager begins with an $accept$ action, the participant will execute the sequence of actions $accept; do\_task; exit$ and stop, which is not a correct execution of the protocol.

We have to take into account also the behavior of the manager. Since we don't know its internal behavior, we will assume that the manager respects its public behavior, i.e. that it respects its permissions and commitments in the protocol specification.

Compliance verification

The verification that the participant is compliant with the protocol can be formalized as a validity check. Let $D = (\Pi, \mathcal{C})$ be the domain description describing the protocol, as defined above. The formula

$$(Comp(\Pi) \wedge Init \wedge Perm_M \wedge Com_M \wedge Comp(\Pi_P) \wedge \mathcal{C}_P) \rightarrow (Perm_P \wedge Com_P)$$

is valid if in all the behaviors of the system, in which the participant executes its program π_P and the manager (whose internal program is unknown) respects the protocol specification (in particular, its permissions and commitments), the permissions and commitment of the participant are also satisfied.

Contract Net with N participants

The formulation of the protocol can be extended by introducing a fluent $replied(i), \ldots$ and an action $refuse(i), \ldots$ for each participant i. We assume action $cfp(T)$ to be shared by all agents (broadcast by the manager).

The precondition laws for $accept(i)$ and $reject(i)$ must be modified so that these actions will be executed only after all participants have replied to the manager, i.e.:

$$\Box((\neg proposal(i) \vee acc\_rej(i) \vee \bigvee_{j=1,N} \neg replied(j)) \rightarrow [accept(i)]\bot)$$

and the same for $reject(i)$.

Rigid protocol

Contract Net is a *rigid protocol*, where the behavior of the agents involved is defined step by step. It can be represented by an AUML diagram or finite state automaton.
We can represent it in DLTL as a regular program (automaton) instead of using permissions and commitments:

$$cfp(M, all); (refuse(i, M)+ \\ (propose(i, M); (reject(M, i)+ \\ accept(M, i); inform(i, M, Done(i, task)))))$$

The specification of the protocol must also give the meaning of the communicative actions used in the protocol (domain description), by describing their effects on the social state and by defining the condition under which they are executable in a state.

Model checking

In principle, with DLTL we do not need to use model checking, because programs and domain descriptions can be represented in the logic itself. However this can be rather inefficient.
Given a domain description D and a property φ to be proved, we can apply a model checking approach as follows:

- derive from D a Kripke structure (the model), which directly corresponds to a Büchi automaton where all the states are accepting, and which describes all possible computations,
- derive from $\neg\varphi$ a Büchi automaton,
- take the product of the two automata, and check it for emptiness of the accepted language.

To summarize

Our approach provides a unified framework for describing different aspects of multi-agent systems using $DLTL$.

- Programs are expressed as regular expressions,
- (communicative) actions can be specified by means of action and precondition laws,
- properties of social facts can be specified by means of causal laws
- commitments and temporal properties can be expressed by means of the *until* operator.