On Situated Reasoning in Multi-Agent Systems

Mikhail Prokopenko

Applied Artificial Intelligence Project
CSIRO Mathematical and Information Sciences
Locked Bag 17, North Ryde, NSW 1670, Australia
mikhail.prokopenko@cmis.csiro.au

Abstract

In this paper we aim to analyse relationships between different approaches to formalising interactivity in dynamic systems. Approaches developed in the framework of Reasoning about Action are mostly logic-based, rely on a centralised world model, and try to (explicitly) capture various aspects of rationality. Another methodology evolved in the field of Multi-agent Systems. It usually considers autonomous agents reacting to changes in external environment and (ideally) exhibiting emergent behaviour. We first attempt to formally define various types of situated agent architectures and encapsulate them in a hierarchical framework. We then analyse and identify domain classes and action theories corresponding to given agent architecture types. This approach can specifically assist in mapping logic theories of actions to reactive agent architectures, where ramifications are embedded in situated behaviours. The described hierarchical framework has been used in the RoboCup Simulation League domain, resulting in implementation of the Cyberoos’98 – a heterogeneous soccer team of autonomous software agents (3rd place winner of the Pacific Rim series at PRICAI-98).

Introduction


The idea that reactive behaviours can be proved to be correct with respect to a theory of actions (and in some cases can be derived from it) is relatively new. For instance, a connection between theories of actions and reactive robot control architectures based on the paradigm of situated activity is explored in (Baral and Son 1996). This approach formalises further the concept of “an action leading to a goal” defined at the representation level in the situated automata approach (Kaelbling and Rosenschein 1990) and follows the latter in relating declarative agent specifications and situated behaviours. Recently the related problem of “formally proving high-level effect descriptions of actions from low-level operational definitions” (Sandewall 1996) was addressed in the context of robotic knowledge validation. Unlike this approach, where both descriptions are expressed in logic, we do not require from an agent architecture derived from a higher-level representation to be a logic-based formalism. On the contrary, the resulting architecture may contain only reactive behaviours validated with respect to a meta-level action theory (Prokopenko and Jauregui 1997, Prokopenko et al 1998). We must stress that our approach aims not only at obtaining sound translation procedures but also (and more importantly) at analysing and identifying classes of domains corresponding to certain types of agent architectures.

More precisely, an attempt is made to select a certain class of domains and describe a procedure mapping a domain description (given as a logic theory of actions) into a behaviour-based multi-agent system. Such a conversion should ideally preserve the meaning of the domain description as compared with the multi-agent system’s dynamics. In other words, state transitions produced by behaviours of autonomous agents must be warranted by logic-based reasoning about actions and change.

Initially, we describe a hierarchical framework for situated agent architectures. Then a basic action theory describing unconstrained domains is used to derive a dynamic multi-agent system based on a simple reactive agent architecture. A more complex class of domains with logical and causal constraints is mapped into another dynamic system (based on an extended architecture), using an augmented translation procedure. Both translations are shown sound with respect to underlying action theories.

Situated Agent Architecture

In this section we define various types of situated agent architectures and analyse their formal properties. Some of the architectures are well-known – for example, variants of tropistic and hysteretic agents are discussed in (Genesereth and Nilsson 1987). We first attempt to incorporate these results in a framework suitable for situated synthetic agents. Then we try to extend the architecture, while retaining the rigour and clarity of fundamental definitions.

Environment Simulator

We define a Simulator agent as a tuple $A_s$:

$<W, P, A, E, C, \text{view}, \text{projection}, \text{send}, \text{receive}, \text{do}>$, 

where $W$ is a set of all external states, $P$ is a set of all possible partitions of $W$, $A$ is a set of situated agents, $E$ is a set of effectors, and $C$ is a communication channel type.
Function \textit{view} structures situated agent perceptions by selecting a partition of external states for each agent. In other words, it maps an agent into an external states partition and defines \textit{view}: \( A \rightarrow \mathcal{P} \).

Dependent on a current situation in the synthetic world, the \textit{Simulator} determines which particular element from a viewable partition is currently observable by every situated agent in \( A \). In other words, the \textit{Simulator} projects an external state and a situated agent into an element of the viewable partition of external states, by using projection: \( w \times A \rightarrow 2^{\mathcal{W}} \), where \( 2^{\mathcal{W}} \) is the power-set of \( \mathcal{W} \). The exact range of the \textit{projection} function is the external states partition selected by \textit{view} from the set \( \mathcal{P} \) of all possible partitions of \( \mathcal{W} \). More precisely,

\[ \forall w \in \mathcal{W}, \forall a \in A, \text{ projection}(w, a) \in \text{view}(a) \]

The projected partition element is a set of external states \( (\text{projection}(w, a) \subseteq \mathcal{W}) \), and is sent by the \textit{Simulator} to the situated agent by means of the function \textit{send}: \( A \times 2^{\mathcal{W}} \rightarrow \mathcal{C} \).

We will presume that situated agents are able to decode \( \text{projection}(w, a) \) from the input message, and respond back to the \textit{Simulator} with an effector name. The received communication is decoded by \textit{receive}: \( A \times \mathcal{C} \rightarrow E \), and the communicated effector is processed by the function \( \text{do}: E \times \mathcal{W} \rightarrow \mathcal{W} \), which maps each effector and an external state into the next state.

\textbf{Tropistic Agents}

Having defined the architecture of the \textit{Simulator} agent, we formally describe an \textit{Abstract Tropistic} agent as a tuple \( A_{\text{tr}} \)

\[ \langle C, S, E, \text{sense}, \text{tropistic-behaviour}, \text{response} \rangle, \]

where \( S \) is a set of agent sensory states, and \( C \) and \( E \) denote the same components as before. The sensory function is defined as \textit{sense}: \( C \rightarrow S \), where an element of \( C \) is expected to carry the information on \( \text{projection}(w, a) \).

Activity of the agent is characterised by \textit{tropistic-behaviour}: \( S \rightarrow E \). We do not intend here to formally define the notion of reactive planning. However, by allowing the set \( E \) to include composite effectors \( e_1; e_2 \), where \( e_1 \in E, e_2 \in E \), we can implicitly account for the case of tropistic planning - when a situated agent reacts to stimuli \( S \) with an \( n \)-length sequence of effectors. The \textit{response} function takes care of communicating the selected behaviour to the \textit{Simulator} by encoding \textit{response}: \( E \rightarrow C \).

This abstract class may not have any instances because the \textit{tropistic-behaviour} function is not implemented at this level of the hierarchy.

It is interesting at this stage to consider a very simple sub-class of the \textit{Abstract Tropistic} agent - a \textit{Clockwork} agent. This class has a specialised sensory function \textit{timer}: \( C \rightarrow S \), and does not specialise the function \textit{sense} in any other way. In other words, a \textit{Clockwork} agent is able to distinguish only between external states with different time values, having no other sensors apart from the \textit{timer}. In addition, this class specialises the \textit{tropistic-behaviour} function by introducing the \textit{command} function defined as \textit{command}: \( S \rightarrow E \). Since the only sensor available at this level is the \textit{timer}, the agent behaviour is predefined and is totally driven by time values. In other words, like a clockwork mechanism, a \textit{Clockwork} agent executes its fixed behaviour as a sequence of commands sent to the \textit{Simulator} at predefined time points. Formally, the \textit{Clockwork} agent class is defined as a tuple \( A_{cw} \)

\[ \langle C, S, E, \text{sense}, \text{timer}, \text{tropistic-behaviour}, \text{command}, \text{response} \rangle, \]

where the \textbf{bold} style indicates newly introduced functions.

The \textit{Tropistic} agent class \( A_r \) is derived from the \textit{Clockwork} agent and finally allows us to implement the \textit{tropistic-behaviour} function.

In practice, it is almost impossible to express each instantiation \( (e, s) \) of the \textit{tropistic-behaviour} function \( e = \text{tropistic-behaviour}(s) \) in terms of complete sensory states. Instead, we represent such behaviour instantiations in terms of partial sensory states. For example, the following rules, given in the form similar to control rules (Baral and Son 1996) or condition-action pairs (Kaelbling and Rosenschein 1990), describe behaviour instantiations:

\begin{verbatim}
if [SeeBall: (distance, direction) \& Far(distance)]
then turn(direction); dash(2*distance)
\end{verbatim}

The bracketed component on the left-hand side correspond to elements of \( S \) and has to be evaluated as true in order to activate effector(s) on the right-hand side. A sentence \( \alpha \) in this component specifies the set of states from \( S \) consistent with \( \alpha \). In other words, it specifies a \textit{tropistic-behaviour} as a sequence of commands sent to the \textit{Simulator}.

\textbf{Hysteretic Agents}

A \textit{Hysteretic} agent is defined here as a reactive agent maintaining internal state \( I \) and using it as well as sensory states \( S \) in activating effectors \( E \); i.e. its activity is characterised by \textit{hysteretic-behaviour}: \( I \times S \rightarrow E \). Again, we allow the set \( E \) to include composite effectors \( e_1; e_2 \), where \( e_1 \in E, e_2 \in E \), covering the case of hysteretic planning. A memory update function maps an internal state and an observation into the next internal state, i.e. it defines \textit{update}: \( I \times S \rightarrow I \). A \textit{Hysteretic} agent reacts to stimuli \( s \) sensed by \textit{sense}(\( I, s \)) and activates effectors \( e \) according to \textit{hysteretic-behaviour}(\( I, s \)). The agent neither has full knowledge about the state \( do(e, w) \) obtained by the \textit{Simulator}, nor reasons about the transition. The next interaction with the world may bring partial knowledge about its new state.

The \textit{Hysteretic} agent class extends its superclasses by adding the \textit{hysteretic-behaviour} and \textit{update} functions, while retaining all previously defined functions (i.e., it is a
sub-class of the Tropistic agent). So the Hysteretic agent is
defined as a tuple $A_g$

$$<C, S, E, I, \text{sense}, \text{timer}, \text{tropistic-behaviour}, \text{command}, \text{hysteretic-behaviour}, \text{update}, \text{response}>$$

Hysteretic-behaviour instantiations may be represented
in terms of partial internal and sensory states as well. For
example, the following rule describes a hysteretic
behaviour instantiation, where the effector on the right-
hand side is composite:

if [Orientation: angle] and [SeeBall: (b, dir) ^ Close(b)]
then weak_kick(angle - dir); turn(angle)

Two bracketed components on the left-hand side correspond to elements of $I$ and $S$ respectively.
An Extended Hysteretic agent $A_{e^g}$ is derived from the
Hysteretic agent. Its architecture contains two additional
communication components notify and listen, and is
defined as a tuple

$$<C, S, E, I, \text{sense}, \text{timer}, \text{tropistic-behaviour}, \text{command}, \text{hysteretic-behaviour}, \text{notify}, \text{listen}, \text{update}, \text{response}>,$$

where the communication functions are responsible for
dealing with outgoing and incoming messages exchanged
among situated agents (rather than between a Simulator and
a situated agent). The listen function is specialised from the
sense function, and notify function is a specialised
hysteretic-behaviour. The reason for introducing these
communication functions is that domain constraints may
influence internal variables of other agents or require
invocation of other agents’ actions. The distinction between
structural ramifications when “the action can affect
features of other objects than those which occur as
arguments of the action” and local ramifications involving
only “features of the argument objects” was identified in
(Sandewall 1994). For example, the following domain
constraint

$$H(t, \text{near}(x): y) \leftrightarrow H(t, \text{near}(y): x)$$

demands from a model to include the atomic formula
near(B): A, whenever it contains the atomic formula
near(A): B. Therefore, at the moment when agent A
evaluates near(A): B as true (either by sensing a new
observation, or by updating an internal variable), another
agent (B in this case) needs to be notified. If the agent B
has limited sensory capabilities (preventing, for example, a
direct sensing of near(B): A), then the communication is
the only way of ensuring a synchronous assignment.

It is worth noting that “listening” to a message is a form of
sensing, and “speaking” is a form of action (Parsons,
Sierra, and Jennings 1998). Therefore, incoming messages
can be sensed (listen-ed) by a suitable sensor, let us say,
Told: $e$, and outgoing messages can be sent by the
specialised behaviour notify activating a suitable effector,
let us say, Tell($g$, $e$), where $e$ is an effector name, and $g$ is a
name of a receiving agent. For example,

If [SeePartner: (n, d, angle) ^ SeeBall: (dist, dir) ^ NearBall(n)]
then Tell(NameOf(n), turn(angle - dir)
If [LookingForBall] and [Told: turn(x)] then turn(x)

An agent may send a message to itself. An execution of a
communicated effector modifies internal variables of the
receiving agent.

Dynamic multi-agent systems

A dynamical system can be characterised as “a system
whose state changes over time, and where effects flow
forward in time so that the non-input part of the state at one
time can only depend on its earlier states” (Sandewall 1994). The agents of the system perform actions
influencing state variables and changing the system state.

We define a dynamic multi-agent system by a set of
architecture types $A \subseteq \{ A_1, A_{cw}, A_1, A_w, A_p \}$, and a
particular value of a time parameter $t$. Given a finite set of
agents $g_k (1 \leq k \leq N)$ instantiated from the architectures in
$A$, one can construct a dynamic system $V_a$ (based on $A$)
which maps an initial state and a time value to a state.

More precisely, $V_a$ is a function $U \times R \rightarrow U$, where $U$
is the set of possible states $I_1 \times ... \times I_s \times W$ and $R$ is the set
of real numbers – assuming, without loss of generality, that
agent $g_a$ is a Simulator agent. We denote a state generated
by the dynamic system $V_a$ at the time instant $t$ as $V_a(t)$.

Action Theories for Situated Reasoning

The approach to representing operational definitions and
effect descriptions of continuous actions (Sandewall 1996)
follows a narrative time-line approach and allows us to
define continuous change, discrete discontinuities, actions
with duration, composite actions, and the distinction
between success and failure of an action. We will adopt
from (Sandewall 1996) the following notation:

$H(t, f:v)$: fluent $f$ has the value $v$ at time $t$;
$X(t,f)$: fluent $f$ is exempt from minimisation of discontinuities (the
occlusion operator);
$G(s,a)$: the action $a$ is invoked at time $s$;
$A(s,a)$: the action $a$ is applicable at time $s$;
$D_I([s,t], a)$: the action $a$ is successfully executed
over the time interval $[s,t]$;
$D_F([s,t], a)$: the action $a$ fails over the time interval $[s,t]$;
$D_R([s,t], a)$: the action $a$ is being executed during the interval $[s,t]$.

The set of axioms in (Sandewall 1996) specifies properties and
relationships of these predicates. All state variables
(fluents) in a described domain may have an argument. We
assume that multi-argument fluents can be reified and
alternatively represented by unary fluents without an
expressibility loss. In order to declare that a fluent does not
have a value we use the nil symbol: $H(t, f:nil)$ abbreviates
$\neg \exists v [H(t, f:v)]$. Another syntactic sugar is introduced for
anonymous variables: $H(t, f:_)$ abbreviates $\exists v [H(t, f:v)]$,
assuming that the sentence where the Skolem constant replaced a variable, has had no two quantifiers referring to the same variable v. By definition, $H(t, fnil) \leftrightarrow -H(t, f, \_\_)$.

The reassignment operator := will be used to abbreviate $H(t,f:v) \land X(t,f)$ as $H(t,f:=v)$.

Basic Action Theory for Unconstrained Domains

We start with simple deterministic artificial worlds where domain constraints are not defined, and therefore all action effects are direct. All initially given formulae $H(t, f:v)$ will be called observation descriptions, and all initially given formulae $G(t, a)$ will be referred to as plan descriptions.

The action success description has the following form:

$$D_s([s,t], a) \rightarrow H(t, o_\gamma)$$

(1)

where $o_\gamma$ is the post-condition of the action $a$ given at the termination time (Sandewall 1996). For example,

$$D_s([s,t], PASS(x, y, p)) \rightarrow H(t, possession(y))$$

describes successful execution of the PASS action, applicability description of which can be given as

$$A(s, PASS(x, y, p)) \leftrightarrow H(s, possession(x)) \land H(s, sustains(x,y,p))$$

An action, once invoked, continues towards a success at which instant it terminates (unless there is a qualification that forces it to fail earlier). The invocation and termination descriptions respectively are given as follows:

$$A(s, a) \land G(s, a) \rightarrow H(s, \gamma_a)$$

(2)

$$\mu_a \land D_s([s,t], a) \rightarrow D_s([s,t], a)$$

(3)

where $\gamma_a$ is the invocation condition and $\mu_a$ is one of the termination conditions.

An action failure is defined by a failure description and by a failure effects description:

$$\delta_a \land D_s([s,t], a) \land \neg D_s([s,t], a) \rightarrow D_s([s,t], a)$$

(4)

$$\delta_a \land D_s([s,t], a) \land D_s([s,t], a) \rightarrow H(t, \tau_a)$$

(5)

where $\tau_a$ is the failure post-condition.

We will denote the described theory of actions as $T_1 = \langle D, T \rangle$, where all domain axioms compose $D$, and $M$ is a specific minimisation policy. The chronological minimisation of discontinuities in piecewise continuous fluents should, in general, be complemented by maximisation of action duration (Sandewall 1996).

Basic Translation

In this section we employ a basic translation $T_r: D \Rightarrow V_s H$ from a domain described by the theory of actions $T_1$ to a dynamic multi-agent system based on the Simulator and Hysteretic agent architectures $S_H = \{A_s, A_w\}$.

We introduce a set $G$ of agent sub-classes (derived from elements of $S_H$) and assign all domain descriptions in $D$ to corresponding classes. Formally, an assignment relation $P \subset G \times D$ is defined such that $\forall d \in D, \exists g \in G, (g, d) \in P$.

For all agent classes $g \in G$, the translation $T_r$ introduces appropriate sensor, effector and internal variables names $tr(f)$, and processes all (assigned to $g$) descriptions $d$ such that $(g, d) \in P$. In particular, the following steps are performed:

- for all plan descriptions $G(t, a)$ produce a behaviour instantiation

  $\text{if } [tr(\theta_\gamma)] \text{ and } [\text{Timer: } t] \text{ then } start_a$

  where $\theta_\gamma$ is the applicability condition of action $a$, Timer is a sensor, $t$ denotes a current time reading; the effector $start_a$ executes the translated invocation condition $tr(\gamma_a)$ (2) and initiates the $\text{Started}_a$ variable as $\text{Started}_a; t$;

- for all observation descriptions $H(t, f:v)$ produce an update and/or sense instantiation(s)

  $\text{if } [\text{Timer: } t] \text{ then } [tr(\theta_\gamma)]$

- for all termination descriptions (3) produce a behaviour instantiation

  $\text{if } [\text{Started}_a; s \land \neg tr(\theta_\gamma)] \text{ and } [\text{Timer: } t] \text{ then } stop_a$

  where $a$ is the action in $D_s([s,t], a)$; $\theta_\gamma$ is its post-condition; the effector $stop_a$ executes the translated success condition $tr(\gamma_a)$ (1) and sets the $\text{Started}_a$ variable as $\text{Started}_a; nil$;

- for all failure descriptions (4) produce a behaviour instantiation

  $\text{if } [\text{Started}_a; s \land \neg tr(\theta_\gamma) \lor \theta_\gamma] \text{ and } [\text{Timer: } t] \text{ then } halt_a$

  where $a$ is the action in $D_s([s,t], a)$; $\theta_\gamma$ and $\tau_a$ are its post-condition and failure post-condition respectively; the effector $halt_a$ executes the translated failure effects description $tr(\tau_a)$ (5) and sets the $\text{Started}_a$ variable as $\text{Started}_a; nil$.

The described theory of actions $T_1 = \langle D, M \rangle$ provides a validation criterion for the dynamic system $V_s H$. Although the time in $T_1$ is continuous we can, nevertheless, validate all non-auxiliary atomic formulae $tr(f); \forall in $V_s H$ at the time instant $t$ if $H(t, f:v)$ is entailed by a consistent theory $T_r$. An auxiliary formula $\text{Started}_a; s (s \neq nil)$ is valid in $V_s H$ if $s < t \land D_s([s,t], a)$ is entailed by a consistent theory $T_r$. Similarly, an auxiliary formula $halt_a; nil; s < t \land (\text{Started}_a; s \lor D_s([s,t], a)))$. It is easy to verify that the following soundness proposition is true.

Proposition 1. Given a deterministic unconstrained domain $D$ described by a consistent action theory, there exists an assignment relation $P \subset G \times D$ for a set of agent class names $G$, such that the translation $T_r: D \Rightarrow V_s H$ produces a dynamic system (based on $\{A_s, A_w\}$) where all atomic formulae are valid.

It follows immediately that a consistent action theory describing a (trivial) domain with only plan descriptions
can provide a validation criterion for a dynamic multi-agent system $V_{s,CW}$ based on the Simulator and Clockwork agent architecture $S_{CW} = \{A_s, A_{cw}\}$.

**Extended Action Theory**

The extended action theory allows us to reason about ramifications and interactions. Typically, indirect changes (ramifications) are non-monotonically derived as consequences of domain constraints. For example,

$$H(t, \text{near}(x): y) \land H(t, \text{near}(x): z) \rightarrow H(t, \text{near}(y): z)$$

This reassignment constraint uses the occlusion operator $X(t,f)$ and excludes the indirect effects from the law of inertia. This effectively specifies the direction of the dependency and makes the latter look like a "causal rule" producing necessary ramifications (McCain and Turner 1995, Gustaffson and Doherty 1996).

Another form of ramifications describes an interaction when one continuous action triggers another:

$$\lambda_i \land D_i([s,t], a) \land \neg D_i([s,t], a) \rightarrow G(t, b) \quad (6)$$

where each $\lambda_i$ represents an interaction condition, and $b$ is another action invoked by occurrences of $\lambda_i$ during the execution of the action $a$. For example,

$$H(t, \text{see}\_\text{opponent}(x): z) \land H(t, \text{near}(x): z) \land H(t, \text{see}\_\text{partner}(x): y) \land D_i([s,t], \text{Dribble}(x, d)) \land \neg D_i([s,t], \text{Dribble}(x, d)) \rightarrow G(t, \text{PASS}(x, y, \text{distance}(x, y)))$$

$G(t, b)$ is the only specified effect of the interaction. Therefore other effects of the action $b$ (defined in its success, failure, and/or interaction descriptions) can be viewed as ramifications of this interaction. They do not have to be specified explicitly with every such interaction and are supposed to be implied indirectly. Possible preconditions for the action invocation are checked by the applicability description $A(t, b)$. In general, any expression of the form

$$\lambda_i \rightarrow G(t, b) \quad (7)$$

can be considered as a (trivial) interaction description.

Thus at least two ways to address the ramification problem in a logic characterising piecewise continuous change can be observed: by defining constraints and by specifying interaction descriptions for continuous actions.

We will denote the described theory of actions as $T = <D, M>$, where domain axioms composing $D$ may include domain constraints and interaction descriptions.

**Extended Translation**

The Reasoning about Action tradition proposes to use domain constraints and/or causal laws separated from action specifications in order to derive indirect effects of an action. In a multi-agent framework, a similar solution can be achieved by embedding indirect effects in situated behaviours of autonomous reactive agents.

An extended translation $Tr_e: D \Rightarrow V_{s,EH}$ from a domain described in the theory of actions $T$, to a dynamic system based on the Simulator and Extended Hysteretic agent architectures $S_{EH} = \{A_e, A_{eh}\}$ translates every domain constraint into a number of reassignment (causal) constraints and introduces additional required names. In particular, it adds the following steps to the translation $Tr_e$:

- for all interaction descriptions (6) produce a notify behaviour instantiation
  - if $[\text{Started}_a \land \text{tr}(\neg \tau_a)]$ and $[\text{tr}(\lambda_a) \land \text{Timer}: t]$
    - then $\text{Tell}(gb, \text{b\_start})$
  and a listen behaviour instantiation for an agent $g_a$
    - if $[\text{tr}(\varphi_a)]$ and $[\text{b\_start}](b) \land \text{Timer}: t$
    - then $\text{b\_start}$

- for each causal constraint $H(t, \alpha) \rightarrow H(t, f:= v)$ produce a notify behaviour instantiation
  - if $[\text{tr}(\alpha) \land \neg (\text{tr}(f): v)]$
    - then $\text{Tell}(g_{eff}, \text{assign}\_\#)$
  and a listen behaviour instantiation for an agent $g_{s\_eff}$
    - if $[\text{Timer}: t]$
    - then $\text{assign}\_\#$

where a (sequentially numbered) $assign\_\#$ effect executes $[\text{tr}(f): v]$ and $g_{s\_eff}$ is the agent receiving the communication.

It is worth noting that domain constraints and interaction descriptions are translated into situated behaviours of the same structure, thus allowing to uniformly embed possible ramifications.

Translation of a definitional domain constraint produces several causal constraints, where a right-hand side fluent is partially defined in terms of the left-hand side. It is well known that some fluents cannot be fully defined in terms of their definitional counterparts. For example, the domain constraint

$$H(t, \text{free}(x)) \leftrightarrow \neg \exists y [H(t, \text{near}(x): y)]$$

fully defines the propositional fluent $\text{free}(x)$ in terms of the fluent $\text{near}(x)$. However, the latter is not uniquely defined in terms of the former. Nevertheless, in order to produce causal constraints, it is sufficient to employ nil and _ values:

- $H(t, \text{near}(x): \text{nil}) \rightarrow H(t, \text{free}(x): \text{True})$
- $H(t, \text{near}(x): \_ ) \rightarrow H(t, \text{free}(x): \text{False})$
- $H(t, \text{free}(x): \text{True}) \rightarrow H(t, \text{near}(x): \text{nil})$
- $H(t, \text{free}(x): \text{False}) \rightarrow H(t, \text{near}(x): \_ )$

The last constraint leaves open the question what object is
near x, but does not justify any unsound formulae.

The extended theory of actions \( T_s = \langle D, M_s \rangle \) provides a validation criterion for the system \( V_{\text{str}} \). All atomic formulae \( \text{tr}(f) : v \) in \( V_{\text{str}}(f) \) are validated as in \( V_{\text{str}}(f) \). Analogously, the following soundness proposition is true.

**Proposition 2.** Given a deterministic domain \( D \) described by a consistent action theory, there exists an assignment relation \( P \subseteq G \times D \) for a set of agent class names \( G \), such that the translation \( \text{Tr}_2: D \Rightarrow V_{\text{str}} \) produces a dynamic system (based on \( \{A_s, A_t\} \)) where all atomic formulae are valid.

Sometimes the translation \( \text{Tr}_s \) may produce a multi-agent system based on the Simulator and Hysteretic agent architectures \( S_H = \{A_s, A_t\} \). It occurs when all messages are communicated internally within an agent. In other words, interactions and domain constraints are defined in terms of internal variables. The class of action domains where the translation yields these particularly simple results is the class of domains with local ramifications:

**Corollary 2.** Given a deterministic domain with local ramifications \( D \) described by a consistent action theory, there exists an assignment relation \( P \subseteq G \times D \) for a set of agent class names \( G \), such that the translation \( \text{Tr}_2: D \Rightarrow V_{\text{str}} \) produces a dynamic system (based on \( \{A_s, A_t\} \)) where all atomic formulae are valid.

Furthermore, if a deterministic temporal projection domain with local ramifications is described only by plan and trivial interaction descriptions (7), then its translation produces a multi-agent system based on the Simulator and Tropistic agent architectures \( S_T = \{A_s, A_t\} \). The sense function in \( A_s \) captures all trivial interaction conditions, the tropistic-behaviour implements all trivial interaction descriptions, and the command invokes all (pre-)planned actions, producing timed response. For such trivial domains (described only by plans and trivial interactions) we immediately obtain the following

**Corollary 3.** Given a trivial deterministic domain with trivial local ramifications \( D \) described by a consistent action theory, there exists an assignment relation \( P \subseteq G \times D \) for a set of agent class names \( G \), such that the translation \( \text{Tr}_2: D \Rightarrow V_{\text{str}} \) produces a dynamic system (based on \( \{A_s, A_t\} \)) where all atomic formulae are valid.

**Conclusions**

The obtained results can be generalised by translating broader classes of action domains into more complex agent architectures. Ideally, any extended translation \( \text{Tr}_2: D \Rightarrow V_{\text{str}} \) must satisfy the important soundness property: state transitions produced by a dynamic multi-agent system \( V_s \) are sound with respect to reasoning warranted by an action theory \( T_s \). For instance, action theories capturing goal-oriented behaviour axiomatised in (Sandewall 1997) may be used to validate process-oriented agent architectures.

The intention is to consider a generic class of systematic models \( \langle T, \text{Tr}, V \rangle \), where each instance of an action theory \( T \) provides a validation criterion for a dynamic system \( V \), and the translation \( \text{Tr} \) is sound. Such systematic models would support uniform specifications of synthetic agents and facilitate a rigorous comparative analysis of different architectures and their ranges of applicability with respect to provably correct logics.

**References**