Specification and Verification of Dynamics in Agent Models[*]

Tibor Bosse[1], Catholijn M. Jonker[2], Lourens van der Meij[1],
Alexei Sharpanskykh[1], and Jan Treur[1]

[1] Vrije Universiteit Amsterdam, Department of Artificial Intelligence
De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands
[2] Delft University of Technology, Department of Mediametics, Man Machine Interaction Group
Mekelweg 4, 2628 CD Delft, The Netherlands
{tbosse, lourens, sharp, treur}@few.vu.nl, catholijn@mmi.tudelft.nl

Abstract

Within many domains, among which biological, cognitive, and social areas, multiple interacting processes occur among agents with dynamics that are hard to handle. This paper presents the predicate logical Temporal Trace Language (TTL) for the formal specification and analysis of dynamic properties of agents and multi-agent systems. This language supports the specification of both qualitative and quantitative aspects, and therefore subsumes specification languages based on differential equations and qualitative, logical approaches. A software environment has been developed for TTL, which supports editing TTL properties and enables the formal verification of properties against a set of traces. The TTL environment proved its value in a number of projects within different biological, cognitive and social domains.

1. Introduction

In domains such as Biology, Cognitive Science, and Social Science, the dynamics of the multiple interacting processes among different agents involved poses modelling challenges. Currently, differential equations are among the techniques most often used to address this challenge, with partial success. For example, in the area of intracellular processes, hundreds or more reaction parameters (for which reliable values are rarely available) are needed to model the processes in question. Thus, describing these processes in terms of differential equations can seriously compromise the feasibility of the model. Likewise, in the area of Cognitive Science, the Dynamical Systems Theory that is also based on differential equations (DST, see e.g., [24]), is well practiced and successful. However, the

---

[*] A preliminary (short) version of this manuscript appeared in the Proceedings of the International Conference on Intelligent Agent Technology (IAT'06)

models typically only address lower-level agent cognitive processes such as sensory or motor processing. DST has less to offer for modelling the dynamics of higher-level processes with a mainly qualitative character, such as agent reasoning, complex task performance, and certain capabilities of language processing.

For formal qualitative modelling of processes at a high level of abstraction, logic-based methods have proved useful. For example, variants of modal temporal logic [2, 16, 19, 29, 36] gained popularity in agent technology, also for modelling social phenomena. However, many of the logic-based methods lack the quantitative expressivity, needed, e.g., for modelling processes for which precise timing relations play an essential role (e.g., biological and chemical processes).

Many of the currently used analysis methods concern either quantitative or qualitative aspects of complex systems, depending on the language used for specification of the models. For example, mathematical calculus is used for analysis of quantitative models based on DST, logical proof and model-checking techniques are applied for analysis of qualitative, logic-based specifications. However, only a few of these methods address an integrative analysis of both types of aspects that often occur in complex systems; e.g., [17].

Thus, within several disciplines the need exists for general modelling and analysis techniques capable to deal with complex agent systems that comprise both quantitative and qualitative aspects. This paper introduces the Temporal Trace Language (TTL) as such a technique for the analysis of dynamic properties within complex domains such as Biology, Cognitive Science, and Social Science. In Section 2, a novel perspective is put forward for the development of such a technique, based on the idea of checking dynamic properties on given sets of traces. Section 3 describes the TTL language in detail. Examples of the application of TTL are presented in the next sections: Section 4 focuses on hybrid systems, Section 5 addresses the process of trace conditioning, Section 6 addresses the area of representation, and Section 7 discusses the application of TTL in some others domains. Section 8 describes the tools that support the TTL modelling environment in detail. In particular, the TTL Property Editor and the TTL Checker Tool are discussed. Section 9 is a conclusion.

2. Perspective of this Paper

As follows from the discussion above, the demands for dynamic modelling and analysis approaches suitable for specifying agent systems in natural domains are nontrivial. In particular, the possibility of both discrete and continuous modelling of a system at different aggregation levels is demanded. Furthermore, numerical expressivity is required for modelling systems with explicitly defined quantitative relations best presented by difference or differential equations. Moreover, for specifying qualitative aspects of a system, modelling languages should be able to express logical relationships between parts of a system.

Desiderata for analysis techniques include both the generation and formalisation of simulated and empirical trajectories or traces, as well as analysis of complex dynamic properties of such traces and relationships between such properties. A *trace* as used here represents a temporally ordered sequence of states of an agent system. Each state is characterised by a number of *state properties* that hold, for example a qualitative logical property or a numerical value that a certain state variable has. Simulated traces may be obtained by performing simulations of certain scenarios of system behaviour based on both quantitative (or continuous) and qualitative (or discrete) variables. This notion of a trace contrasts to Mazurkiewicz traces known in Theoretical Computer Science [30] that are sometimes used to analyse the behaviour of Petri nets. Mazurkiewicz traces represent restricted partial orders over algebraic structures with a trace equivalence relation.

Taken together, the desiderata for modelling languages and analysis techniques described above are not easy to fulfil. On the one hand, high expressivity is desired, on the other hand feasible analysis techniques are demanded. To provide automated support for these analyses the expressivity of the modelling language can be limited, thereby compromising the desiderata for modelling languages. For example, the expressivity may be limited to difference and differential equations as in DST (excluding logical relationships), or to propositional modal temporal logics (excluding numerical relationships). In the former case, calculus can be exploited to do simulation and analysis based on continuous variables only [33]. In the latter case, simulation is based on a specific logical executable format, which does not allow expressions involving continuous variables (e.g., executable temporal logic [2]). Another possibility is to use a number of dedicated formal languages with limited expressiveness and related to them analysis techniques for checking different particular static and dynamic aspects of a system (e.g., structural consistency of a model, dynamic aspects of execution), as proposed in the methodology for the development of correct software KORSO [13]. The languages used in this project describe different formats of system specifications, relations between them (e.g., by refinement based on proof obligations) and the temporal development of these specifications for all phases of the software life cycle. However, in order to guarantee the overall correctness of a system some properties are required to be expressed using more than one language with different types of semantics. Thus, the problem of verification across different not related proof systems arises that is not addressed in this project.

The problem of checking relationships between dynamic properties of a system, identified above as one of the desiderata for analysis techniques, is essentially the problem of justifying entailment relations between sets of properties defined at different aggregation levels of a system's representation. In general, entailment relations can be established either by logical proof procedures or by checking properties of a higher aggregation level on the set of all theoretically possible traces generated by executing a system specification that consists of properties of a lower aggregation level (i.e., by performing model checking [16, 29, 36]). To make it feasible to check relationships between dynamic properties, expressivity of the language for these properties has to be sacrificed to a large extent. However, checking properties on a given set of traces of practical size (instead of all theoretically possible ones), obtained empirically or by simulation, is computationally much cheaper.

Therefore, in that case the language for these properties can be more expressive, such as the sorted predicate logic temporal trace language TTL described in this paper. TTL fulfils all of the identified above desiderata for modelling languages and can be used both for formalisation of empirical and simulated traces and for analysis of properties on traces. Although TTL cannot be used to generate traces by simulation, an executable sublanguage of TTL, such as LEADSTO, cf. [7], may be defined for this purpose. Moreover, decidable fragments of TTL may be defined for the analysis of relationships between dynamic properties of a system.

Finally, having a language for simulation and languages for analysis within one subsuming language also opens the possibility of having a declarative specification of a simulation model, and thus to involve simulation models in logical analyses.

3. A Language to Model Agent Behaviour

The Temporal Trace Language (TTL) presented here is developed from the assumption that the dynamics of an agent system can be described as evolution of states of agents and an environment over time, as for modal temporal logics, see e.g., [2, 16, 19, 29, 36]. TTL has some similarities with situation calculus, see [34] and event calculus, see [26]. A more detailed comparison of TTL to other well-known formalisms for modelling dynamics of a system is given in Section 9. Time in TTL is assumed to be linearly ordered and depending on the application, it may be dense (e.g., the real numbers), or discrete (e.g., the set of integers or natural numbers or a finite initial segment of the natural numbers), or any other form with a linear ordering. An agent interacts with a dynamic environment via its *input* and *output* (interface) states. At its input the agent receives observations from the environment whereas at its output it generates actions that can change a state of the environment.

Agent states

An agent state at a certain point in time as used here is an indication of which of the state properties of the agent and its environment (e.g., observations and actions) are true (hold) at that time point. For specifying state properties for the input, output, internal, and external states of an agent A, state ontologies, named IntOnt(A), InOnt(A), OutOnt(A), and ExtWorldOnt respectively, are used which are specified by a number of sorts, sorted constants, variables, functions and predicates (i.e., a signature in order-sorted predicate logic; e.g., [34]).

State properties

State properties are formulae constructed using a standard multi-sorted first-order predicate language based on such ontologies. For example, a state property expressed as a predicate pain may belong to IntOnt(A), whereas the atom has_temperature(environment, 7) may belong to ExtWorldOnt.

Sorts and atoms for dynamic properties

To characterize the dynamics of the agent and the environment, *dynamic properties* relate properties of states at certain points in time.

To enable reasoning about the dynamic properties of arbitrary systems the language TTL includes special sorts, such as:

TIME	a set of linearly ordered time points
STATE	a set of all state names of an agent system
TRACE	a set of all trace names; a trace or a trajectory can be thought of as a timeline with for each time point a state
STATPROP	a set of all state property names
PART	a set of all names for parts of agents and the world (e.g., inputs, outputs, internals), to which state properties are related.

Throughout the paper, variables such as t, $t1$, $t2$, t', t'' stand for variables of the sort TIME; and variables such as γ, $\gamma1$, $\gamma2$ stand for variables of the sort TRACE.

A state of an agent is related to a state property via the satisfaction relation

$\models$: STATE x STATPROP

formally defined as a binary infix predicate (or by holds as a binary prefix predicate in the software environment). For example,

“in the output state of agent A in trace γ at time t property p holds”

is formalised by

state(γ , t, output(A)) $\models$ p.

Here function symbols are used such as:

state: TRACE x TIME x PART → STATE
output: AGENT → PART
input: AGENT → PART
internal: AGENT → PART

If the indication of an agent aspect is not essential, the third argument is left out: $\text{state}(\gamma, t) \models p$, thus using a function

$$\text{state}: \quad \text{TRACE} \times \text{TIME} \rightarrow \text{STATE}$$

Both $\text{state}(\gamma, t, \text{output}(A))$ and p are terms of the TTL language. TTL terms are constructed by induction in a standard way for sorted predicate logic from variables, constants and function symbols typed with TTL sorts.

Dynamic properties

Dynamic properties are expressed by TTL-formulae inductively defined by:

(1) If v_1 is a term of sort STATE, and u_1 is a term of the sort STATPROP, then $\text{holds}(v_1, u_1)$ is an atomic TTL formula.

(2) If τ_1, τ_2 are terms of any TTL sort, then $\tau_1 = \tau_2$ is an atomic TTL formula.

(3) If t_1, t_2 are terms of sort TIME, then $t_1 < t_2$ is an atomic TTL formula.

(4) The set of well-formed TTL-formulae is defined inductively in a standard way based on atomic TTL-formulae using boolean propositional connectives and quantifiers.

For example, the dynamic property

> 'in any trace γ, if at any point in time t1 agent A observes that it is dark in the room, whereas earlier a light was on in this room, then there exists a point in time t2 after t1 such that at t2 in the trace γ agent A switches on a lamp'

is expressed in formalised form as:

$$\begin{aligned}&\forall t1\ [\ [\ \text{state}(\gamma, t1, \text{input}(A)) \models \text{observed}(\text{dark\_in\_room})\ \&\\ &\exists t0{<}t1\ [\ \text{state}(\gamma, t0, \text{input}(A)) \models \text{observed}(\text{light\_on})]\\ &\Rightarrow\ \exists t2 \geq t1\ \text{state}(\gamma, t2, \text{output}(A)) \models \text{performing\_action}(\text{switch\_on\_light})\ ]\end{aligned}$$

Within TTL the following abbreviation is used for summation:

$$\Sigma_{k:S}\ \text{case}(\varphi,\ v1,\ v2) = v$$

Here for any formula φ, the expression $\text{case}(\varphi, v1, v2)$ indicates the value $v1$ if φ is true, and $v2$ otherwise. The formula as mentioned is an abbreviation for a formula involving conjunctions over subsets $\{k_1, \ldots, k_n\}$ of sort S of known size N:

$$\bigwedge_{n=1,\ldots N}\ \bigwedge_{i=1,\ldots,n}\ \varphi(k_i)\ \wedge\ \forall k{:}S\ [\ [\ \bigwedge_{i=1,\ldots,n}\ k \neq k_i\ ] \Rightarrow \neg\ \varphi(k)\ ] \Rightarrow\ v = n{*}v1 + (N-n){*}v2$$

In applications this abbreviation is very useful. Within the software environment special facilities have been implemented to evaluate such statements.

As TTL uses order-sorted predicate logic as a point of departure, it inherits the standard semantics of this variant of predicate logic. That is, the semantics of TTL is defined in a standard way, by interpretation of sorts, constants, functions and predicates, and a variable assignment. However, in addition the semantics involves some specialised aspects. As a number of standard sorts are present, the elements of these sorts are limited to instances of specified terms in these sorts, as is usual, for example, in logic programming semantics. For example, for the sort TIME it is assumed that in its semantics its elements consist of the time points of the fixed time frame chosen. Moreover, for the sort TRACE, it is assumed that in its semantics its elements consists of a (limited) number of elements named by constants. Furthermore, for the sort STATPROP for state properties it is assumed that in its semantics its elements consist of the set of terms denoting the propositions built in a chosen state language (this is called reification). A full description of the technical details of TTL's semantics is beyond the scope of the current paper. For this purpose, see [35].

By executing dynamic properties traces can be generated and visualised, for example as in Figure 1. Here, the time frame is depicted on the horizontal axis. The names of predicates are shown on the vertical axis. A dark box on top of the line indicates that the predicate is true during that time period.

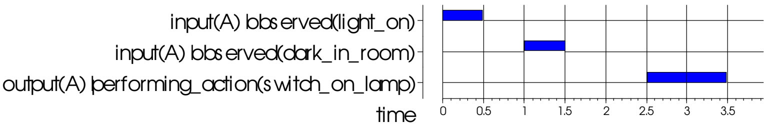


Figure 1. Example visualisation of a trace

LEADSTO as an Executable Sublanguage of TTL

The executable language LEADSTO (cf. [7]) can be defined as a sublanguage of TTL in the following manner. The LEADSTO expression

$$\alpha \twoheadrightarrow_{e,\,f,\,g,\,h} \beta$$

means informally (see Figure 2):

> If state property α holds for a certain time interval with duration g,
> then after some delay (between e and f) state property β will hold for a certain time interval of length h.

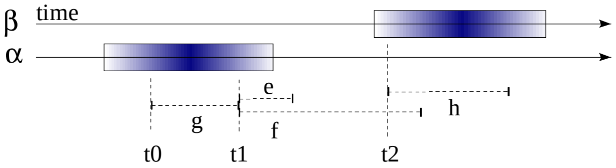


Figure 2. The timing relationships for a LEADSTO expression

An example dynamic property in the LEADSTO format is:

$$\text{observes(agent\_A, food\_present)} \twoheadrightarrow_{2, 3, 1, 1.5} \text{believes(agent\_A, food\_present)}$$

This LEADSTO expression can be formally defined in TTL as follows:

$$\forall t1: [\forall t\ [t1-g \leq t < t1 \Rightarrow \text{state}(\gamma, t) \models \alpha] \Rightarrow \exists d\ [e \leq d \leq f\ \&\ \forall t'\ [t1+d \leq t' < t1+d+h \Rightarrow \text{state}(\gamma, t') \models \beta]$$

The TTL language and its supporting software environment have been applied in research projects addressing different topics in biological, cognitive and social domains, such as cell behaviour, the circulatory system in mammals, ant colony behaviour, human reasoning, conditioning, consciousness, psychotherapy, organisation behaviour, artificial societies, and philosophy of mind. The main research goal in these projects was to analyse the behavioural dynamics of the agents involved (e.g., [6, 8, 9, 10, 11, 12]). TTL was used to formalise dynamic properties of these processes at a high level of abstraction. Next, such properties were automatically checked against simulated or empirical traces. Examples of the application of TTL in different areas are presented in the next sections.

4. Modelling and Analysis of Hybrid Systems in TTL

Hybrid systems incorporate both continuous and discrete components. The dynamics of the former can be described by differential equations, those of the latter can be represented by finite-state automata. Both continuous and discrete dynamics of components influence each other. In particular, the input to the continuous dynamics is the result of some function of the discrete state of a system; whereas the input of the discrete dynamics is determined by the value of the continuous state.

A modelling method for hybrid systems should be capable of expressing both quantitative and qualitative properties of the system and integrating them into one model. TTL satisfies this requirement. Qualitative aspects of systems can be directly expressed by logical TTL properties, which essentially describe temporal relations between system states occurring over time. Quantitative aspects represented by systems of differential equations can be expressed in TTL using discrete or dense time frames in the following manner. As an example, Euler's method, see [32], for solving differential equations is modelled in TTL. Euler's method approximates a differential equation $dy/dt = f(y)$ with the initial condition $y(t_0)=y_0$ by a difference equation $y_{i+1}=y_i+h*f(y_i)$ ($i \geq 0$ is the step number and $h>0$ is the integration step size). This equation can be modelled in TTL in the following way:

$$\forall\gamma\ \forall t\ \forall v: \text{VALUE}\ \text{state}(\gamma, t) \models \text{has\_value}(y, v) \Rightarrow \text{state}(\gamma, t+1) \models \text{has\_value}(y, v + h \bullet f(v))$$

States specify the respective values of y at different time points and the difference equation is modelled by a transition rule from the current to the successive state. The traces γ satisfying the above dynamic property are the solutions of the difference equation. More precise and stable numerical approximation methods (e.g., Runge-Kutta, dynamic step size, see [32]) can be expressed in TTL in a similar manner.

The obtained TTL specification of a complex system can be analysed by means of dedicated techniques as described in Section 8.

5. Analysis of Trace Conditioning in TTL

The example considered in this section illustrates how TTL can be used for the analysis of continuous models of complex systems. This example is taken from [6]. In that paper, TTL is used to analyse the temporal dynamics of trace conditioning. In general, research into conditioning is aimed at revealing the principles that govern associative learning. An important issue in conditioning processes is the adaptive timing of the conditioned response to the appearance of the unconditioned stimulus. This feature is most apparent in an experimental procedure called *trace conditioning*. In this procedure, a trial starts with the presentation of a *warning stimulus* (S1, comparable to a conditioned stimulus). After a blank interval, called the *foreperiod*, an *imperative stimulus* (S2, comparable to an unconditioned stimulus) is presented to which the participant responds as fast as possible. The *reaction time* to S2 is used as an estimate of the conditioned state of preparation at the moment S2 is presented. In this case, the conditioned response obtains its maximal strength, here called *peak level*, at a moment in time, called *peak time*, that closely corresponds to the moment the unconditioned stimulus occurs.

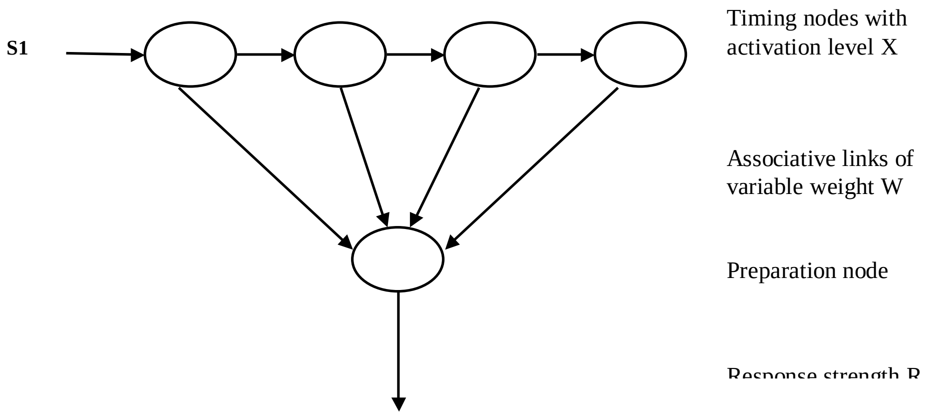


Figure 3. Structure of Machado's conditioning model

Machado [28] developed a basic model that describes the dynamics of these conditioning processes in terms of differential equations. The structure of this model is shown in Figure 3. The model posits a layer of *timing nodes* and a single *preparation node*. Each timing node is connected both to the next (and previous) timing node and to the preparation node. The connection between each timing node and the preparation node (called *associative link*) has an adjustable weight associated to it. Upon the presentation of a warning stimulus, a cascade of activation propagates through the timing nodes according to a regular pattern. Owing to this regularity, the timing nodes can be likened to an internal clock or pacemaker. At any moment, each timing node contributes to the activation of the preparation node in accordance with its activation and its corresponding weight. The activation of the preparation node reflects the participant's preparatory state, and is as such related to reaction time. The weights reflect the state of conditioning, and are adjusted by learning rules, of which the main principles are as follows. First, *during* the foreperiod extinction takes place, which involves the decrease of weights in real time in proportion to the activation of their corresponding timing nodes. Second, *after* the presentation of the imperative stimulus a process of reinforcement takes over, which involves an increase of the weights in accordance with the current activation of their timing nodes, to preserve the importance of the imperative moment. Machado describes the more detailed dynamics of the process by a mathematical model (based on linear differential equations), representing the (local) temporal relationships between the variables involved. For example,

$$dX(t,n)/dt = \lambda X(t,n-1) - \lambda X(t,n)$$

expresses how the activation level of the n-th timing node $X(t+dt,n)$ at time point t+dt relates to this level $X(t,n)$ at time point t and the activation level $X(t,n-1)$ of the (n-1)-th timing node at time point t. Similarly, as another example,

$$dW(t,n)/dt = -\alpha X(t,n)W(t,n)$$

relates the n-th weight $W(t+dt,n)$ at time point t+dt to this weight $W(t,n)$ at time point t and the activation level $X(t,n)$ of the n-th timing node at time point t.

In [6], a number of dynamic properties relevant for trace conditioning have been formalised in TTL. These properties were taken from the existing literature on conditioning, such as [27], in which they were mainly expressed informally. TTL turned out useful to express these properties in a formal manner. An example of such a property (taken from [27], p.372) is given below, both in informal, semi-formal and in formal notation:

Global Hill Preparation

Informal: ‘The state of conditioning implicates an increase and decay of response-related activation as a critical moment is bypassed in time’.

Semi-formal: ‘In trace γ, if at $t1$ a stimulus $s1$ starts, then the preparation level will increase from $t1$ until $t2$ and decrease from $t2$ until $t1 + u$, under the assumption that no stimulus occurs too soon (within u time) after $t1$.’ Formal:

$$\mathrm{has\_global\_hill\_prep}(\gamma{:}\mathrm{TRACE},\ t1,t2{:}\mathrm{TIME},\ u{:}\mathrm{INTEGER}) \equiv$$
$$\forall t',t''{:}\mathrm{TIME}\ \forall p',p''{:}\mathrm{REAL}$$
$$[\ \mathrm{state}(\gamma, t1) \models \mathrm{stimulus\_occurs}\ \ \&\ \neg\ \mathrm{stimulus\_starts\_within}(\gamma, t1, t1+u)\ \ \&$$

$$
\begin{array}{l}
\mathrm{state}(\gamma,\ t') \models \mathrm{preparation\_level}(p')\ \ \&\ \mathrm{state}(\gamma,\ t'') \models \mathrm{preparation\_level}(p'') \\
\Rightarrow\ [t1 \leq t' < t'' \leq t2\ \ \&\ \ t'' \leq t1 + u\ \ \Rightarrow\ \ p' < p''\,]\ \ \& \\
\quad [t2 \leq t' < t'' \leq t1 + u \qquad\qquad \Rightarrow\ \ p' > p''\,]\,]
\end{array}
$$

Here, stimulus_starts_within is defined as follows:

$$
\begin{array}{l}
\mathrm{stimulus\_starts\_within}(\gamma{:}\mathrm{TRACE},\ t1{,}t2{:}\mathrm{TIME}) \equiv \\
\qquad \exists t{:}\mathrm{TIME}\ [\ \mathrm{state}(\gamma,\ t) \models \mathrm{stimulus\_occurs}\ \ \&\ t1 < t < t2\ ]
\end{array}
$$

These (and various similar) properties were automatically verified using the TTL checker tool against a number of (empirical and simulation) traces. Among these properties were also properties that compare different traces, such as:

> ‘the conditioned response takes more time to build up and decay and its corresponding asymptotic value is lower when its corresponding critical moment is more remote from the warning signal.’ (cf. [27])

Such properties cannot be expressed, for example, in modal temporal logics, just like familiar properties such as ‘exercise improves skill’, expressing that the more intensive a training history, e.g., of an athlete, the better the skill will be.

6. Analysis of Representational Content in TTL

The analysis of notions of representational content of *internal* mental state properties is well-known in the literature on Cognitive Science and Philosophy of Mind. In a nutshell, the question in this literature is ‘what does it mean for an agent to have a mental state’, or ‘what information does the mental state represent’? Usually this question is answered by taking a relevant internal mental state property m and identifying a *representation relation* that indicates in which way m relates to properties in the external world or the agent’s interaction with the external world; cf. [4, 21, 24]. The common idea is that the occurrence of the internal (mental) state property m at a specific point in time is related (by a representation relation) to the occurrence of other state properties, at the same or at different time points. Such a representation relation then describes in a precise manner what the internal state property m represents. To define a representation relation, the *causal-correlational* approach is often discussed in the literature in Philosophy of Mind. However, although from a technical viewpoint such a relation is easy to define, this approach has a number of severe limitations and problems; cf. [21, 24]. Two approaches that are considered to be more promising are the *interactivist* approach [4, 23] and the *relational specification* approach [24]. The relational specification approach to representational content is based on a specification of how a representation relation relates the occurrence of an internal state property to properties of states distant in space and time; cf. [24, pp. 200-202]. This notion of representation relation is widely applicable. From a technical viewpoint this requires specification of a more complex temporal relation involving multiple time points, thus in many nontrivial cases a temporal language with high expressivity is needed. The following Section 6.1 illustrates how such complex temporal relations can be represented in TTL. In Section 6.2 it is demonstrated how collective representational content for a group of agents can be defined by using arithmetical operations in TTL expressions.

6.1 Representational Content for an Agent with Intensive Interaction

As a first example, from a paper about representational content for the mental state of an agent that intensively interacts with the environment [10] is discussed. This case study involves the processes to unlock a front door that sticks. Between the moment that the door is reached and the moment that the door unlocks the following reciprocal interaction takes place:

- the agent puts rotating pressure on the key,
- the door lock generates resistance in the interplay,
- the agent notices the resistance and increases the rotating pressure,
- the door increases the resistance,
- and so on, without any result.
- finally, after noticing the impasse the agent changes the strategy by at the same time pulling the door and turning the key, which unlocks the door.

To model the example the following internal state properties are used:

s1	sensory representation for being at the door
s2(r)	sensory representation for resistance r of the lock
p1(p)	preparation for the action to turn the key with rotating pressure p (without pulling the door)
p2	preparation for combined pulling the door and turning the key
c	state for having learnt that turning the key should be combined with pulling the door

The interactions between agent and environment are defined by the following sensor and effector states:

o1	observing being at the door
o2(r)	observing resistance r
a1(p)	action turn the key with rotating pressure p (without pulling the door)
a2	action turn the key while pulling the door

In addition, the following state properties of the world are used:

arriving_at_door	the agent arrives at the door
lock_reaction(r)	the lock reacts with resistance r
door_unlocked	the door is unlocked
d(mr)	resistance threshold mr of the door (indicating that the door will continue to resist until pressure mr or more is used)
max_p(mp)	maximal force on the key that can be exercised by the agent

The following property expresses the representational content of the internal state property c, with respect to the past process (the *backward* representational content, for short).

> ‘In any trace γ, internal state c occurs iff in the past once observation o1 occurred, then action a1(1), then o2(1), then a1(2), then o2(2), then a1(3), and finally o2(3)’.

This is formalised in TTL by:

$$\begin{array}{l}
\forall t1,t2,t3,t4,t5,t6,t7\ [\ t1 \le t2 \le t3 \le t4 \le t5 \le t6 \le t7 \\
\quad \&\ \mathrm{state}(\gamma, t1, \mathrm{input}) \models o1 \\
\quad \&\ \mathrm{state}(\gamma, t2, \mathrm{output}) \models a1(1)\ \ \&\ \ \mathrm{state}(\gamma, t3, \mathrm{input}) \models o2(1) \\
\quad \&\ \mathrm{state}(\gamma, t4, \mathrm{output}) \models a1(2)\ \ \&\ \ \mathrm{state}(\gamma, t5, \mathrm{input}) \models o2(2) \\
\quad \&\ \mathrm{state}(\gamma, t6, \mathrm{output}) \models a1(3)\ \ \&\ \ \mathrm{state}(\gamma, t7, \mathrm{input}) \models o2(3) \\
\quad \Rightarrow \exists t8 \ge t7\ \mathrm{state}(\gamma, t8, \mathrm{internal}) \models c\ ] \\
\&\ \forall t8\ [\ \mathrm{state}(\gamma, t8, \mathrm{internal}) \models c \Rightarrow \\
\quad \exists t1,t2,t3,t4,t5,t6,t7\ \ t1 \le t2 \le t3 \le t4 \le t5 \le t6 \le t7 \le t8 \\
\quad \&\ \mathrm{state}(\gamma, t1, \mathrm{input}) \models o1 \\
\quad \&\ \mathrm{state}(\gamma, t2, \mathrm{output}) \models a1(1)\ \ \&\ \ \mathrm{state}(\gamma, t3, \mathrm{input}) \models o2(1) \\
\quad \&\ \mathrm{state}(\gamma, t4, \mathrm{output}) \models a1(2)\ \ \&\ \ \mathrm{state}(\gamma, t5, \mathrm{input}) \models o2(2) \\
\quad \&\ \mathrm{state}(\gamma, t6, \mathrm{output}) \models a1(3)\ \ \&\ \ \mathrm{state}(\gamma, t7, \mathrm{input}) \models o2(3)\ ]
\end{array}$$

6.2 Collective Representational Content for a Group of Agents

The idea of representational content can also be applied to a group of agents. In such a case, instead of defining the representational content of an internal mental state of one agent, one should look for the representational content of some *shared extended mental state* of a group of agents (such as observed, for example, for the role of pheromones in ant colonies). As a result, an extension of the analysis of notions of representational content to *external* state properties is needed. Moreover, for the case of external mental state properties that are *shared*, a notion of *collective* representational content is needed. As a result, the question to be answered then becomes ‘what information does a shared extended mental state (e.g., a level of pheromones at a certain location) represent for the group’?

This question can only be answered by considering quantitative information, such as certain *levels* assigned to a shared extended mental state property p; in this case a mental state property is involved that is parameterised by a number: it has the form p(r), where r is a number, denoting that p has level

r. This differs from the above in that now the following aspects have to be modelled: (1) joint creation of p: multiple agents together bring about a certain level of p, each contributing a part of the level, (2) by decay, levels may decrease over time, (3) behaviour may be based on a number of state properties with different levels, taking into account their relative values, e.g., by determining the highest level of them. For the ants example, for each choice point multiple directions are possible, each with a different pheromone level; the choice is made for the direction with the highest pheromone level (ignoring the direction the ant just came from).

To address the *backward* case (i.e., the case of joint creation of a mental state property in the past process), the representation relation involves a summation of droppings of pheromones over multiple agents at different time points. Moreover a decay rate r with $0 < r < 1$ is used to indicate that after each time unit only a fraction r is left.

The ants example (which was taken from [9]) concerns multiple agents (the ants), each of which has input (to observe) and output (for moving and dropping pheromones) states, and a physical body which is at certain positions over time, but no internal mental state properties (they are assumed to act purely by stimulus-response behaviour). The following formalised state properties were used to model this example:

pheromones_at(e, i)	the pheromone level at edge e is i
is_at_location_from(a, l, e)	ant a is at location l coming from edge e
is_at_edge_from_to(a, e, l1, l2)	ant a is at edge e, heading from location l1 to location l2
connected_to_via(l1, l2, e)	edge e connects locations l1 and l2

Note that in some of the state properties the direction of an ant is incorporated (e.g., ant a is at location l coming from e, ant a is at edge e to l2 coming from location l1). This direction is meant to relate to the orientation of the ant's body in space, which is a genuine state property; but for convenience this is expressed by referring to the past or future states involved.

Based on these formalised state properties, the following backward representation property is expressed for the ants example (in mathematical terms):

Backward Collective Representation Relation

There is an amount v of pheromone at edge e, if and only if there is a history such that at time point 0 there was ph(0, e) pheromone at e, and for each time point k from 0 to t a number dr(k, e) of ants was present at e, and $v = ph(0, e) * r^t + \Sigma_{k=0}^{t} dr(t-k, e) * r^k$

A formalisation of this property in the logical language TTL is as follows:

$$\forall t{:}\mathrm{TIME}\ \forall e{:}\mathrm{EDGE}\ \forall v{:}\mathrm{VALUE}\ \ state(\gamma, t) \models pheromones\_at(e, v) \Leftrightarrow$$
$$\Sigma_{k=0}^{t}\ \Sigma_{a=ant1}^{ANTS}\ case([\ \exists l,l1{:}\mathrm{LOCATION}\ state(\gamma, k) \models is\_at\_edge\_from\_to(a, e, l, l1)\ ], 1, 0) * r^{t-k} = v$$

Recall that for any formula f, the expression case(f, v1, v2) indicates the value v1 if f is true, and v2 otherwise.

The *forward* case (i.e., with respect to the future process) involves a behavioural choice that depends on the relative levels of multiple mental state properties. This makes that at each choice point the representational content of the level of one mental state property is not independent of the level of the other mental state properties involved at the same choice point. Therefore it is only possible to provide representational content for the combined mental state property involving all mental state properties involved in the behavioural choice. For the ants example the following forward representation relation property is specified according to the relational specification approach.

Forward Collective Representation Relation

If at time t1 the amount of pheromone at edge e1 (connected to location 1) is maximal
with respect to the amount of pheromone at all other edges connected to that location l,
except the edge that brought the ant to the location,
then, if an ant is at that location l at time t1,
then the next edge the ant will be at some time t2 > t1 is e1.

If at time t1 an ant is at location 1 and
for every ant arriving at that location 1 at time t1,
the next edge it will be at some time t2 > t1 is e1,
then the amount of pheromone at edge e1 is maximal with respect to the amount of pheromone
at all other edges connected to that location l, except the edge that brought the ant to the location.

A formalisation of this property in TTL is as follows:

∀t1:TIME,l,l1:LOCATION,e1,e2:EDGE,i1:REAL
[e1≠e2 &
state(γ, t1) |= connected_to_via(l, l1, e1) &
state(γ, t1) |= pheromones_at(e1, i1) &
[∀l2:LOCATION≠l1, e3:EDGE≠e2 [state(γ, t1) |= connected_to_via(l, l2, e3) ⇒
∃i2:REAL [0≤i2<i1 & state(γ, t1) |= pheromones_at(e3, i2)]]
⇒ ∀a:ANT [state(γ, t1) |= is_at_location_from(a, l, e2) ⇒
∃t2:TIME>t1 state(γ, t2) |= is_at_edge_from_to(a, e1, l, l1) &
[∀t3:TIME t1<t3<t2 ⇒ is_at_location_from(a, l, e2)]]]]

∀t1:TIME, l,l1:LOCATION,e1,e2:EDGE
[e1≠e2 &
state(γ, t1) |= connected_to_via(l, l1, e1) &
∃a:ANT state(γ, t1) |= is_at_location_from(a, l, e2) &
∀a:ANT [state(γ, t1) |= is_at_location_from(a, l, e2) ⇒
∃t2:TIME>t1 state(γ, t2) |= is_at_edge_from_to(a, e1, l, l1) &
[∀t3:TIME t1<t3<t2 ⇒ is_at_location_from(a, l, e2)]]]

$\Rightarrow \exists$i1:REAL [state(γ, t1) |= pheromones_at(e1, i1) &

[$\forall$l2:LOCATION$\neq$l1, e3$\neq$e2 [state(γ, t1) |= connected_to_via(l, l2, e3)

$\Rightarrow \exists$i2:REAL [0$\leq$i2$\leq$i1 & state(γ, t1) |= pheromones_at(e3, i2)]]]]

For more details about this case, see [9].

7. Application of TTL in Other Areas

Besides the areas discussed above, TTL has been applied in many other domains as well. In order to give an extended overview in limited space, below a number of TTL formulae used in other domains are presented (both in informal and formal notation):

7.1 Analysis of human reasoning

In [12] a common human reasoning pattern has been analysed: reasoning by assumption. The process of reasoning by assumption involves three important sub-processes: *assumption determination*, *observation result prediction*, and *assumption evaluation*. See Figure 4 for an overview of the model. In this figure, the rounded rectangles denote different components of the model where the different sub-processes take place (including the external world, which is used to observe the relevant predictions made). The arrows indicate information flow.

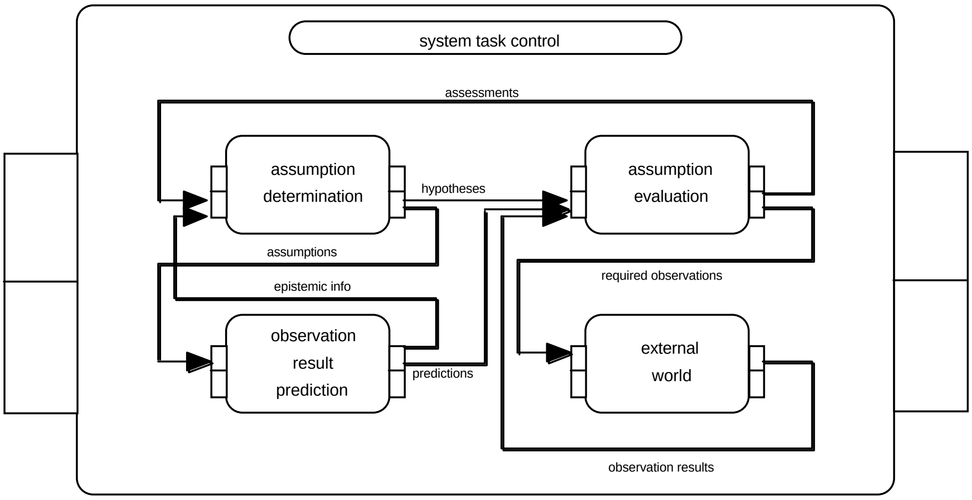


Figure 4. Model for Reasoning by Assumption

From this domain of human reasoning, as an illustration the following property is taken from [12].

Proper Rejection Grounding

'In any trace γ, if an assumption is rejected, then earlier on there was a prediction for it that did not match the corresponding observation result'.

$$\begin{aligned}
&\forall t{:}\text{TIME}\ \forall A{:}\text{INFO\_EL}\ \forall S1{:}\text{SIGN}\\
&\quad \text{state}(\gamma, t) \models \text{rejected}(A,S1) \Rightarrow\\
&\qquad [\ \exists t'{:}\text{TIME}{\leq}t\ \exists B{:}\text{INFO\_EL}\ \exists S2,S3{:}\text{SIGN}\\
&\qquad \text{state}(\gamma, t') \models \text{prediction\_for}(B, S2, A, S1)\ \&\ \text{state}(\gamma, t') \models \text{observation\_result}(B, S3)\ \&\ S2 \neq S3\ ]
\end{aligned}$$

7.2 Learning Behaviour of Aplysia

Aplysia Californica is a sea hare that is often used to do experiments. It is able to learn on the basis of classical conditioning. In this section, a simplified description is given of this learning behaviour (viewed from an external perspective), based on [18], pp. 155-156.

Behaviour before learning phase

Initially the following behaviour is shown:

- a tail shock leads to a response (contraction)
- a light touch on its siphon is insufficient to trigger such a response

Learning phase

Now suppose the following experimental protocol is undertaken. In each trial the subject is touched lightly on its siphon and then, shocked on its tail (as a consequence it responds).

Behaviour after a learning phase

It turns out that after a number of trials (three in the example) the behaviour has changed:

- the animal also responds (contracts) on a siphon touch.

Note that, to characterise behaviour, there is a difference between the *learned* behaviour (which is simply an *adapted* stimulus-response behaviour) and the *learning* behaviour, which is a form of *adaptive* behaviour, no stimulus-response behaviour. To analyse the latter type of behaviour in a study of Aplysia's adaptive processes [11], the following property was specified in TTL:

Aplysia learning

'In any trace γ, if a siphon touch occurs, and at three different earlier time points t1, t2, t3, a siphon touch occurred, directly followed by a tail shock, then the animal will contract'.

$$\begin{aligned}
&\forall t\ \ [\ \text{state}(\gamma, t) \models \text{siphon\_touch}\ \ \&\\
&\exists t1, t2, t3, t4, t5, t6\\
&t1{<}t2\ \&\ t2{<}t3\ \&\ t3{<}t4\ \&\ t4{<}t5\ \&\ t5{<}t6\ \&\ t6{<}t\ \&\\
&\text{state}(\gamma, t1) \models \text{siphon\_touch}\ \ \&\ \text{state}(\gamma, t2) \models \text{tail\_shock}\ \ \&\\
&\text{state}(\gamma, t3) \models \text{siphon\_touch}\ \ \&\ \text{state}(\gamma, t4) \models \text{tail\_shock}\ \ \&\\
&\text{state}(\gamma, t5) \models \text{siphon\_touch}\ \ \&\ \text{state}(\gamma, t6) \models \text{tail\_shock}\ \ ]\\
&\Rightarrow\ \exists t7\ \ t7{\geq}t\ \&\ \text{state}(\gamma, t7) \models \text{contraction}
\end{aligned}$$

7.3 Behaviour of an Ant Colony

In the analysis [8] of the behaviour of an ant colony [5], one of the properties specified was the global property of successfulness of getting food in the nest. Below this is expressed in the sense that at least one ant drops food at the nest location at least at one point in time.

Food Delivery Successfulness

'In any trace γ, there is at least one ant that brings food back to the nest'.

$\exists$t:TIME $\exists$a:ANT $\exists$l:LOCATION $\exists$e:EDGE
state(γ, t) $\models$ is_at_location_from(a, l, e) & state(γ, t) $\models$ nest_location (l) & state(γ, t) $\models$ to_be_performed(a, drop_food)

8. Software Environment

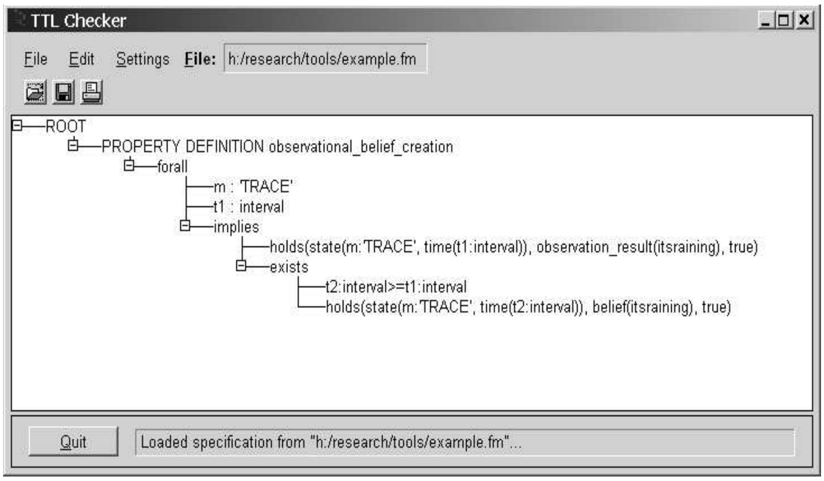
This section presents the software environment[1] that was built in SWI-Prolog to support the process of specification and automated verification of dynamic properties on a limited set of traces. Basically, this software environment consists of two closely integrated tools: the Property Editor and the Checker Tool.

The Property Editor provides a user-friendly way of building and editing properties in TTL. By means of graphical manipulation and filling in forms a TTL specification can be constructed. TTL specifications may also be provided as plain text. When a TTL specification is created, the Checker Tool can be used to verify automatically whether a TTL property from the specification holds for a given set of traces. User interaction with the tools involves three separate actions:

1. Loading, editing, and saving a TTL specification in the Property Editor (see Figure 5).
2. Loading and inspecting traces to be checked by activating the Trace Manager.
3. Checking a property against a set of loaded traces by the Checker Tool. The property is compiled and checked, and the result is presented to the user. If a property is not satisfied by a set of traces, then a counter-example is provided to the user, which identifies the cause of failure.

---

[1] The software can be downloaded from the following URL: http://www.cs.vu.nl/~wai/TTL.



```
ROOT
    PROPERTY DEFINITION observational_belief_creation
        forall
            m : 'TRACE'
            t1 : interval
            implies
                holds(state(m:'TRACE', time(t1:interval)), observation_result(itsraining), true)
                exists
                    t2:interval>=t1:interval
                    holds(state(m:'TRACE', time(t2:interval)), belief(itsraining), true)
```

```
Loaded specification from "h:/research/tools/example.fm"...
```

Figure 5. The TTL Checking Environment

Note that the traces that are loaded in step 2. can be either traces produced by simulations (see [7]) or empirical traces. Empirical traces provided to the TTL Checker may be obtained by formalising empirical data from log-files produced by information systems or from results of experiments. Within the area of Requirements Engineering methods have been described to aid the modeller in formalisation of scenarios (which can be considered as informal traces) and requirements. For example, in [20] it is described how requirements and scenarios initially formulated informally in natural language and/or graphical elements can be restructured into a more standard structured natural language format, which then can be reformulated more easily in a formal language. The structured language form makes the keywords used for state properties, the input and output references and the temporal succession relations more explicit. More precisely, for scenarios, a structured semi-formal representation is obtained by performing the following:

- explicitly distinguish *input and output* concepts in the scenario description
- define (domain) *ontologies* for the input and output information
- represent the temporal structure described implicitly in a sequence of events.

As an illustration, a simple example in informal representation is the following:

> The temperature and pressure are high.
> A red alert is generated and the heater is turned off.

This can be reformulated into a more structured form as follows.

- input: temperature is high, pressure is high
- output: red alert, situation is explosive
- input: situation to be resolved
- output: heater is turned off

Note that here an implicit intermediate state property (the meaning of the red alert) is made explicit. A formalisation can be made by using formal ontologies for the concepts used, and by formalising the relationships. More precisely, formalisation of a scenario on the basis of a structured semi-formal representation is achieved by:

- choosing *formal ontologies* for the input and output information
- formalisation of the *temporal structure*

This results in a formal temporal trace γ for the scenario.

state(γ, 1, input(S)) |= high(temperature)
state(γ, 1, input(S)) |= high(pressure)
state(γ, 2, output(S)) |= red_alert
state(γ, 3, input(S)) |= explosive_situation_to_be_resolved
state(γ, 4, output(S)) |= turn_off(heater)

For a more extensive discussion about the transition from informal to formal, see [20].

The following sections provide more details on implementation of the software environment. In particular, Section 8.1 describes the implementation of the TTL Editor and Section 8.2 discusses the verification procedure underlying the TTL checker.

8.1 Implementation of the TTL Editor

A *TTL specification* constructed in the TTL Property Editor consists of a number of user-defined property definitions and sort definitions. A property definition consists of a header (property name and arguments, i.e., prop_name(v1:s1, v2:s2)) and a body (a TTL formula). Arbitrary sorts may be defined by enumerating their elements.

A *TTL formula* is constructed from atomic TTL formulae by conjunction, (Formula1 and Formula2), disjunction (Formula1 or Formula2), negation (not Formula), implication and quantification (forall ([v1:s1, v2:s2], Formula), exists ([v1:s1, v2:s2 < term2], Formula)).

Atomic TTL formulae correspond to user-defined properties, *holds atoms* (e.g., holds(state(trace1, t, output(ew)), a1 ∧ a2) or state(trace1, t, output(ew)) ⊨ a1 ∧ a2), mathematical expressions (e.g. term1 = term2, term1 > term2) and built-in properties (i.e., complex properties encoded into the implementation language).

All TTL formulae are constructed from *terms* that are implemented as Prolog terms (e.g., fn(t1,t2) , n1, t1 + t3, 1.3). Constants, variables and functions from which terms are constructed should be typed with appropriate sorts. For example, each variable should be declared as variable_name: sort. The software supports a number of built-in sorts, among which sorts for integer, real and range of integers (i.e., sorts integer, real, between(i1:integer,i2:integer)), the sort for the set of all states (STATE) and the sort for the set of all traces (TRACE). Furthermore, libraries with predefined general purpose and domain-specific sorts and functions are available for creating terms.

8.2 Verification by the TTL Checker

After a TTL property is specified in the Editor and traces being loaded by the Trace Manager, the Checker Tool may be used to determine if the considered property holds on the loaded traces. To perform such verification an algorithm has been developed.

The verification algorithm is a backtracking algorithm that systematically considers all possible instantiations of variables in the TTL formula under verification. However, not for all quantified variables in the formula the same backtracking procedure is used. Backtracking over variables occurring in holds atoms is replaced by backtracking over values occurring in the corresponding holds atoms in traces under consideration. Since there are a finite number of such state atoms in the traces, iterating over them often will be more efficient than iterating over the whole range of the variables occurring in the holds atoms. Formulae that contain variables quantified over infinite sorts not occurring in a holds atom cannot be checked by the TTL Checker.

As time plays an important role in TTL-formulae, special attention is given to continuous and discrete time range variables. Because of the finite variability property of TTL traces (i.e., only a finite number of state changes occur between any two time points), it is possible to partition the time range into a minimum set of intervals within which all atoms occurring in the property are constant in all traces. Quantification over continuous or discrete time variables is replaced by quantification over this finite set of time intervals.

In order to increase the efficiency of verification, the TTL formula that needs to be checked is compiled into a Prolog clause. Compilation is obtained by mapping conjunctions, disjunctions and negations of TTL formulae to their Prolog equivalents, and by transforming universal quantification into existential quantification. Thereafter, if this Prolog clause succeeds, the corresponding TTL formula holds with respect to all traces under consideration.

The complexity of the algorithm has an upper bound in the order of the product of the sizes of the ranges of all quantified variables. However, if a variable occurs in a holds atom, the contribution of that variable is no longer its range size, but the number of times that the holds atom pattern occurs (with different instantiations) in trace(s) under consideration. The contribution of an isolated time variable is the number of time intervals into which the traces under consideration are divided.

The specific optimisations discussed above make it possible to check realistic dynamic properties with reasonable performance. In particular, checking the property ‘Aplysia Learning’ given in Section 7.2 (involving eight different time points) against a single trace with three state atoms occurring in the verified formula and 28 changes of atom values over time takes 0.76 sec. on a regular PC. With the increase of the number of traces with similar complexity as the first one, the verification time grows linearly: for 3 traces - 3.9 sec., for 5 traces - 6.59 sec. However, the verification time is polynomial in the number of isolated time range variables occurring in the formula under verification.

9. Conclusion

This paper presents the predicate logical Temporal Trace Language (TTL) for the formal specification and analysis of dynamic properties of cognitive agent models. Although the language has a logical foundation, it supports the specification of both qualitative and quantitative aspects,and subsumes specification languages based on differential equations. TTL allows for explicit reference to time points and time durations, which enables modelling of the dynamics of continuous real-time phenomena. Furthermore, more specialised languages can be defined as a sublanguage of TTL. For the purpose of simulation, the executable language LEADSTO has been developed [7]. For verification of properties, different decidable fragments of predicate logic (e.g., [1]) can be defined as sublanguages of TTL.

The use of the temporal trace language TTL has a number of practical advantages. In the first place, it offers a well-defined language to formulate relevant dynamic relations in practical domains, with standard first order logic semantics. It has a high expressive power. For example, the possibility of explicit reference to *time points* and *time durations* enables modelling of the dynamics of continuous real-time phenomena, such as sensory and neural activity patterns in relation to mental properties (cf. [33]). Also difference and differential equations can be expressed. These features go beyond the expressive power available in standard linear or branching time temporal logics.

Furthermore, the possibility to quantify over traces allows for specification of *more complex adaptive behaviours*. As within most temporal logics, reactiveness and pro-activeness properties are specified. In addition, in our language also properties expressing different types of adaptive behaviour can be expressed. For example a property such as

> ‘exercise improves skill’

which is a relative property in the sense that it involves the comparison of two alternatives for the history. Another property of this type is trust monotonicity:

> ‘For any two traces $\gamma1$ and $\gamma2$, if
> initially in trace $\gamma2$ A’s trust is at least as high as A’s trust at t in trace $\gamma1$, and
> at each time point t agent A’s experience with public transportation in $\gamma2$ at t is at least as good as A’s experience with public transportation in $\gamma1$ at t, then in trace $\gamma2$ at each point in time t, A’s trust is at least as high as A’s trust at t in trace $\gamma1$’.

$$\begin{array}{l}
\forall \gamma 1, \gamma 2 \\
[\forall w1, w2{:}\mathrm{VALUE}\ [\ \mathrm{state}(\gamma 1, 0) \models \mathrm{has\_value}(\mathrm{trust}, w1)\ \&\ \\
\mathrm{state}(\gamma 2, 0) \models \mathrm{has\_value}(\mathrm{trust}, w2)\ ] \Rightarrow\ w1 \leq w2\ ]\ \&\ \\
[\forall t, \forall v1, v2{:}\mathrm{VALUE}\ [\ \mathrm{state}(\gamma 1, t) \models \mathrm{has\_value}(\mathrm{experience}, v1)\ \&\ \\
\mathrm{state}(\gamma 2, t) \models\ \mathrm{has\_value}(\mathrm{experience}, v2)\ ] \Rightarrow\ v1 \leq v2\ ]]] \Rightarrow \\
[\forall t, \forall w1, w2{:}\mathrm{VALUE}\ [\ \mathrm{state}(\gamma 1, t) \models \mathrm{has\_value}(\mathrm{trust}, w1)\ \&\ \\
\mathrm{state}(\gamma 2, t) \models \mathrm{has\_value}(\mathrm{trust}, w2)\ ] \Rightarrow\ w1 \leq w2\ ]]]]]
\end{array}$$

Thus, different alternative histories can be represented and compared in TTL, whereas in standard forms of temporal logic it is not possible. Similarly, the kind of relative or comparative properties put forward in [22], such as ‘the more south on the northern hemisphere, the higher the trees’, as properties lacking an explanation in terms of a cause and its effects, can be expressed since our language allows comparison of different traces and different (local) restrictions within traces.

The possibility to define restrictions to *local languages for parts* of a system or the world is also an important feature. For example, the distinction between internal, external and input and output languages is crucial, and is supported by the language TTL, which also entails the possibility to quantify over system parts; this allows for specification of system modification over time. This possibility allows to consider traces in which ‘brain, body and world’ are modelled in an integrative manner, and to focus on one of these aspects in the context of the overall trace [14, 15].

Finally, since state properties are used as first class citizens in the temporal trace language, it is possible to explicitly refer to them, and to quantify over them, enabling the specification of what are sometimes called *second-order properties*, which are used in part of the philosophical literature (e.g., [25]) to express functional roles related to mental properties or states.

TTL has some similarities with the situation calculus [34] and the event calculus [26], which are two well-known formalisms for representing and reasoning about temporal domains. However, a number of important syntactic and semantic distinctions exist between TTL and both calculi. In particular, the central notion of the situation calculus - a situation - has different semantics than the notion of a state in TTL. That is, by a situation is understood a history or a finite sequence of actions, whereas a state in TTL is associated with the assignment of truth values to all state properties (a “snapshot” of the world). Moreover, in contrast to the situation calculus, where transitions between situations are described by actions, in TTL actions are in fact properties of states.

Moreover, although a time line has been recently introduced to the situation calculus [34], still only a single path (a temporal line) in the tree of situations can be explicitly encoded in the formulae. In contrast, TTL provides more expressivity by allowing explicit references to different temporally ordered sequences of states (traces) in dynamic properties (e.g., the trust monotonicity property). Other examples of such properties, where different histories are compared are given in Section 5 above on trace conditioning.

In contrast to the event calculus, TTL does not employ the mechanism of events that initiate and terminate fluents. Events in TTL are considered to be functions of the external world that can change states of components, according to specified properties of a system. Furthermore, similarly to the situation calculus, also in the event calculus only one time line is considered.

TTL can also be related to temporal languages that are often used for verification (e.g., propositional temporal logic (PTL) and linear-time logic (LTL) [3, 16, 19]). The general idea of translation of a LTL formula into a TTL expression is rather straightforward: by replacing the temporal operators of LTL by quantifiers over time. E.g., the following LTL formula

$$\boldsymbol{G}(\text{observation\_result(itsraining)} \rightarrow \boldsymbol{F}(\text{belief(itsraining)}))$$

where the temporal operator ***G*** means 'for all later time points', and ***F*** 'for some later time point' is translated into the following TTL expression:

$$\forall t1 \; [\; \text{state}(\gamma, t1) \models \text{observation\_result(itsraining)} \Rightarrow$$
$$\exists t2 > t1 \; \text{state}(\gamma, t2) \models \text{belief(itsraining)} \;]$$

Note that the translation is not bi-directional, i.e., it is not always possible to translate TTL expressions into LTL expressions. An example of a TTL expression that cannot be translated into LTL is again the property of trust monotonicity.

Furthermore, TTL also allows expressivity provided by different extensions of PTL. In particular, the extended temporal logic (ETL) [37] provides a possibility to express any property definable by a regular expression on sequences of states, which cannot be expressed in PTL. Due to the fact that the syntax of TTL provides quantifiers, predicates, and arithmetic functions, such properties can be also expressed in TTL. For example, the property "a given proposition p has to be true in every even state of a sequence" can be expressed in TTL as follows: $\forall t \; \text{state}(\gamma, 2\bullet t) \models p$.

To support the formal specification and analysis of dynamic properties in TTL, special software tools (the Property Editor and the Checker Tool) have been developed. The Property Editor has an intuitive graphical interface for building and editing TTL properties, and the Checker Tool employs an efficient algorithm for the formal verification of properties against a limited set of traces. Although this form of checking is not as exhaustive as model checking (which essentially means checking properties on the set of all traces generated by model execution), in return, it allows more expressivity in specifying properties.

The TTL environment has been tested and proved its value in a number of projects within different domains; e.g., [6, 8, 9, 10, 11, 12]). During this work, the TTL environment has been further developed to provide automated support.

References

1. Andreka, H., Nemeti, I., and van Benthem, J. (1998). Modal Languages and Bounded Fragments of Predicate Logic. Journal of Philosophical Logic, 27(3): 217-274, 1998.

2. Barringer, H., M. Fisher, D. Gabbay, R. Owens, & M. Reynolds (1996). The Imperative Future: Principles of Executable Temporal Logic, Research Studies Press Ltd. and John Wiley & Sons.

3. Benthem, J.F.A.K., van (1983). The Logic of Time: A Model-theoretic Investigation into the Varieties of Temporal Ontology and Temporal Discourse, Reidel, Dordrecht.

4. Bickhard, M.H. (1993). Representational Content in Humans and Machines. Journal of Experimental and Theoretical Artificial Intelligence, 5, 1993, pp. 285-333.

5. Bonabeau, J. Dorigo, M. and Theraulaz, G. (1999). Swarm Intelligence: From Natural to Artificial Systems. Oxford University Press, New York.

6. Bosse, T., Jonker, C.M., Los, S.A., Torre, L. van der, and Treur, J. (2005). Formalisation and Analysis of the Temporal Dynamics of Conditioning. In: Mueller, J.P. and Zambonelli, F. (eds.), *Proceedings of the Sixth International Workshop on Agent-Oriented Software Engineering, AOSE'05*, pp. 157-168.

7. Bosse, T., Jonker, C.M., Meij, L. van der, and Treur, J. (2005). LEADSTO: a Language and Environment for Analysis of Dynamics by SimulaTiOn. In: Eymann, T., et al. (eds.), *Proc. of the Third German Conference on Multi-Agent System Technologies, MATES'05*. LNAI, vol. 3550. Springer Verlag, pp. 165-178

8. Bosse, T., Jonker, C.M., Schut, M.C., and Treur, J. (2005). Simulation and Analysis of Shared Extended Mind. *Simulation Journal* (Transactions of the Society for Modelling and Simulation), vol. 81, 2005, pp. 719 - 732.

9. Bosse, T., Jonker, C.M., Schut, M.C., and Treur, J., Collective Representational Content for Shared Extended Mind. *Cognitive Systems Research Journal*, vol. 7, 2006, pp. 151-174.

10. Bosse, T., Jonker, C.M., and Treur, J. (2005). Representational Content and the Reciprocal Interplay of Agent and Environment. In: Leite, J., Omincini, A., Torroni, P., and Yolum, P. (eds.), *Proc. of the Second Int. Workshop on Declarative Agent Languages and Technologies, DALT'04*. LNAI, vol. 3476. Springer Verlag, pp. 270-288.

11. Bosse, T., Jonker, C.M., and Treur, J. (2006). An Integrative Modelling Approach for Simulation and Analysis of Adaptive Agents. In: *Proc. of the 39th Annual Simulation Symposium.* IEEE Computer Society Press, pp. 312-319. Extended version to appear in *Advances in Complex Systems Journal*, 2007.

12. Bosse, T., Jonker, C.M., and Treur, J. (2006). Formalization and Analyisis of Reasoning by Assumption. *Cognitive Science Journal*, vol. 30, issue 1, pp. 147-180.

13. Broy, M., and Jahnichen, S. (1995). KORSO: Methods, Languages, and Tools for the Construction of Correct Software - Final Report. LNCS, vol. 1009. Springer Verlag.

14. Clark, A. (1997). Being there: Putting brain, body and world together again. Cambridge, MA: MIT Press.

15. Clark, A. (2001). Reasons, robots and the extended mind. Mind & Language, 16, 121–145.

16. Clarke, E.M., Grumberg, O., and Peled, D.A. (2000). Model Checking. MIT Press.

17. Davoren, J.M., Nerode, A., 2000. Logics for Hybrid Systems. Proceedings of the IEEE, vol. 88 no. 7, pp. 985-1010.

18. Gleitman, H. (1999). *Psychology*. W.W. Norton & Company, New York.

19. Goldblatt, R. (1992). Logics of Time and Computation, 2nd edition, CSLI Lecture Notes 7.

20. Herlea Damian, D.E., Jonker, C.M., Treur, J., and Wijngaards, N.J.E., Integration of Behavioural Requirements Specification within Compositional Knowledge Engineering. *Knowledge-Based Systems Journal*, vol. 18, 2005, pp. 353-365.

21. Jacob, P. (1997). *What minds can do? Intentionality in a non-intentional world*. Cambridge: Cambridge University Press.

22. Jackson, F., and Pettit, P. (1990). Program Explanation: A General Perspective. *Analysis*, vol. 50, pp. 107-117.

23. Jonker, C. M., & Treur, J. (2003). A temporal-interactivist perspective on the dynamics of mental states. Cognitive Systems Research, 4, 137–155.

24. Kim, J. (1996). *Philosophy of Mind.* Westview Press.

25. Kim, J. (1998). *Mind in a Physical world: an Essay on the Mind-Body Problem and Mental Causation.* MIT Press, Cambridge, Mass.

26. Kowalski, R., and Sergot, M. (1986). A logic-based calculus of events, New Generation Computing, 4: 67-95.

27. Los, S.A. and Heuvel, C.E, van den. (2001). Intentional and Unintentional Contributions to Nonspecific Preparation During Reaction Time Foreperiods. *Journal of Experimental Psychology: Human Perception and Performance*, vol. 27, pp. 370-386.

28. Machado, A. (1997). Learning the Temporal Dynamics of Behaviour. *Psychological Review*, vol. 104, pp. 241-265.

29. Manna, Z., and Pnueli, A. (1995). Temporal Verification of Reactive Systems: Safety. Springer Verlag.

30. Mazurkiewicz. A. (1987) Trace Theory. In: Advances in Petri nets II: applications and relationships to other models of concur-rency, LNCS vol. 255, pages 279-324. 1987. Springer.

31. Manzano, M., (1996). Extensions of First Order Logic, Cambridge University Press, 1996.

32. Pearson, C.E. (1986). Numerical Methods in Engineering and Science. CRC Press.

33. Port, R.F., Gelder, T. van (eds.) (1995). Mind as Motion: Explorations in the Dynamics of Cognition. MIT Press, Cambridge, Mass.

34. Reiter, R. (2001). *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems,* Cambridge MA: MIT Press.

35. Sharpanskykh, A. and Treur, J. (2005). Verifying Interlevel Relations within Multi-Agent Systems: Formal Theoretical Basis, Technical Report TR-1701AI. VU Amsterdam, 2005. http://hdl.handle.net/1871/9777

36. Stirling, C. (2001). Modal and Temporal Properties of Processes. Springer Verlag.

37. Wolper, P. (1983). Temporal logic can be more expressive. *Information and Control*, vol. 56(1-2), pp. 72-99.