ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/245107281

A Temporal Trace Language for the Formal Analysis of Dynamic Properties

Article

CITATIONS
7

READS
35

5 authors, including:

Tibor Bosse
Radboud University
258 PUBLICATIONS **2,461** CITATIONS
SEE PROFILE

Catholijn M. Jonker
Delft University of Technology
543 PUBLICATIONS **6,369** CITATIONS
SEE PROFILE

Lourens Van der Meij
28 PUBLICATIONS **927** CITATIONS
SEE PROFILE

Alexei Sharpanskykh
Delft University of Technology
126 PUBLICATIONS **1,010** CITATIONS
SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Project Modeling Human Empathy and Empathic Social Interaction View project

Project CoreSAEP: Computational Reasoning for Socially Adaptive Electronic Partners View project

All content following this page was uploaded by Catholijn M. Jonker on 29 May 2014.

The user has requested enhancement of the downloaded file.

A Temporal Trace Language for the Formal Analysis of Dynamic Properties

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

[1] Vrije Universiteit Amsterdam, Department of Artificial Intelligence,
De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands
{tbosse, lourens, treur}@cs.vu.nl
http://www.cs.vu.nl/~{tbosse, lourens, treur}
[2] Radboud Universiteit Nijmegen, Nijmegen Institute for Cognition and Information,
Montessorilaan 3, 6525 HR Nijmegen, The Netherlands
C.Jonker@nici.ru.nl

Abstract. Within many domains, among which biological and cognitive areas, multiple interacting processes occur with dynamics that are hard to handle. Current approaches to analyse the dynamics of such processes, often based on differential equations, are not always successful. As an alternative to differential equations, this paper presents the predicate logical Temporal Trace Language (TTL) for the formal specification and analysis of dynamic properties. This language supports the specification of both qualitative and quantitative aspects, and therefore subsumes specification languages based on differential equations. A special software environment has been developed for TTL, featuring both a Property Editor for building and editing TTL properties and a Checking Tool that enables the formal verification of properties against a set of traces. TTL has a number of advantages, among which a high expressivity and the possibility to define sublanguages for simulation and verification of entailment relations. TTL proved its value in a number of projects within different domains.

1 Introduction

Within many domains, among which biological and cognitive areas, multiple interacting processes occur with dynamics that are hard to handle. Modelling the dynamics of such processes poses real challenges for biologists and cognitive scientists. Currently, within the areas mentioned, differential equations are among the techniques most often used to address this challenge, with limited 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 it is advocated to take the Dynamical Systems Theory (DST, see e.g., [22]), which is also based on differential equations, as a point of departure. Many convincing examples have illustrated the usefulness of DST; however, they often only address lower-level cognitive processes such as sensory or motor processing. Areas for which a quantitative approach based on DST offers less are the dynamics of higher-level processes with mainly a qualitative character, such as reasoning, complex task performance, and certain capabilities of language processing.

As illustrated by these examples, within several disciplines it is felt that more abstract modelling techniques are required to cope with the complexity. This paper introduces the Temporal Trace Language (TTL) as such an abstract technique for the analysis of dynamic properties within complex domains. 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 practically given sets of traces. Next, in Section 3, the basic concepts of the TTL language are introduced. In Section 4 it is shown how TTL can be used to express different kinds of dynamic properties. In Section 5, it is shown how dynamic properties that are expressed in related languages can be translated into TTL. Section 6 provides an example case study in which TTL has been applied. Section 7 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 8 is a conclusion.

2 Perspective of this paper

As can be seen in the discussion about the different areas as given above, the demands for dynamic modelling approaches suitable for these areas are nontrivial. Such *desiderata for modelling languages* include:

(1) modelling at the right level of abstraction
(2) expressivity for logical relationships
(3) expressivity for quantitative relationships
(4) both discrete and continuous modelling
(5) difference and differential equations should be subsumed
(6) expressivity for dynamic properties of varying complexity, e.g., adaptivity

Moreover, analysis techniques that would be desirable concern 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. Such *desiderata for analysis techniques* include:

(a) generating traces by simulation based on quantitative, continuous variables
(b) generating traces by simulation based on qualitative, logical notions
(c) formalisation of simulated or empirical traces
(d) analysis of properties of simulated traces
(e) analysis of properties of empirical traces
(f) analysis of relationships between (e.g., global and local) properties of traces

Taken together, the desiderata gathered above are not easy to fulfill. Sometimes they may even be considered mutually exclusive. On the one hand, high expressivity is desired, but on the other hand feasible analysis techniques are demanded. To make automated support for these analyses feasible, often the strategy is followed to limit the expressivity of the modelling language, thereby compromising on the first list of desiderata. For example, the expressivity is limited to difference and differential equations as in DST (excluding logical relationships, compromising at least (2)), or to propositional modal temporal logics (excluding numerical relationships, compromising at least (3), (5), (6)). In the former case calculus can be exploited to do simulation and analysis [22], fulfilling (a) and (c) but not (b), (d), (e) and (f). In the latter case, for example, simulation can be based on a specific executable format (e.g., executable temporal logic [1], fulfilling (b) and (c) but not (a), (d), (e) and (f)) and model checking techniques can be exploited for analysis of relationships between dynamic properties, fulfilling (d) to (f), e.g., [11, 21, 23].

Within the literature on analysis of properties (verification), much emphasis is put on computation of entailment relations. This essentially means checking properties on the set of all theoretically possible traces of a process. To make that feasible, expressivity of the language for these properties has to be sacrificed to a large extent. However, checking properties on a practically given set of traces (instead of all theoretically possible ones) is computationally much cheaper, and therefore the language for these properties can be more expressive. Such a set can consist of one or a number of traces, obtained empirically or by simulation. By limiting the desiderata by giving up (f), but still keeping (c) to (e), a much more expressive language for properties can be dealt with; the sorted predicate logic temporal trace language TTL described here is an example of this.

For simulation it is essential to have limitations to the language. Therefore, an *executable language* can be defined as a sublanguage of the *overall language* for analysis. Moreover, also *analysis languages* that allow analysis in the sense of (f) can be embedded in the overall language. Thus the following situation is obtained. At the top there is an expressive overall language, in our case TTL, which fulfills all of the desiderata for modelling languages, i.e., (1) to (6). Concerning the desiderata for analysis techniques, it fulfills (c) to (e), but sacrifices (a), (b) and (f). In addition, a sublanguage can be defined for execution (fulfilling (1) to (5) and (a) and (b)), and a sublanguage can be defined for analysis of relationships between properties in

the sense of (f), thereby also fulfilling (1) and (2). For the case of TTL, one of the executable sublanguages that already exist is the LEADSTO language, cf. [4]. Moreover, for the sublanguage for analysis one could think of any standard temporal logic, such as LTL or CTL, see, e.g., [2, 14]. Having the language for simulation and the languages for analysis within one subsuming language provides the possibility to have a declarative specification of a simulation model, and thus to involve a simulation model in logical analyses.

3 Basic Concepts

To describe dynamics, the notion of state is important. Dynamics will be described in the next section as evolution of states over time. The notion of state as used here is characterised on the basis of an ontology defining a set of physical and/or mental (state) properties (following, among others, [20]) that do or do not hold at a certain point in time. These properties are often called state properties to distinguish them from dynamic properties that relate different states over time. A specific state is characterised by dividing the set of state properties into those that hold, and those that do not hold in the state. Examples of state properties are 'the agent is hungry', 'the agent has pain', or 'the environmental temperature is 7° C'. Real value assignments to variables are also considered as possible state property descriptions. For example, in a DST approach based on variables x_1, x_2, x_3, that are related by differential equations over time, value assignments such as $x_1 \leftarrow 0.06$, $x_2 \leftarrow 1.84$, $x_3 \leftarrow -0.27$ are considered state descriptions. State properties are described by ontologies that define the concepts used.

3.1 Ontologies and State Properties

To formalise state property descriptions, ontologies are specified in a (many-sorted) first order logical format: an *ontology* is specified as a finite set of sorts, constants within these sorts, and relations and functions over these sorts (sometimes also called a signature). The example state properties mentioned above then can be defined by nullary predicates (or proposition symbols) such as hungry, or pain, or by using n-ary predicates (with $n \geq 1$) like has_temperature(environment, 7), or has_value (x_1, 0.06).

For a given ontology Ont, the propositional language signature consisting of all *state ground atoms* based on Ont is denoted by At(Ont). The *state properties* based on a certain ontology Ont are formalised by the propositions that can be made (using conjunction, negation, disjunction, implication) from the ground atoms and constitute the set SPROP(Ont).

In many domains, it is desirable to distinguish different *agents* that are involved in the process under analysis. Moreover, it is often useful to distinguish between the internal, external, input, and output state properties of these agents. To this end, the following different types of ontologies are introduced:

- IntOnt(A): to express *internal* state properties of the agent A
- InOnt(A): to express state properties of the *input* of agent A
- OutOnt(A)): to express state properties of the *output* of the agent, and
- ExtOnt(A): to express state properties of the *external* world (for A)

For example, the state property pain may belong to IntOnt(A), whereas has_temperature(environment, 7), may belong to ExtOnt(A). The agent input ontology InOnt(A) defines properties for perception, the agent output ontology OutOnt(A) properties that indicate initiations of actions of A within the external world. The combination of InOnt(A) and OutOnt(A) is the *agent interaction ontology*, defined by InteractionOnt(A) = InOnt(A) ∪ OutOnt(A). The *overall ontology* for A is assumed to be the union of all ontologies mentioned above: OvOnt(A) = InOnt(A) ∪ IntOnt(A) ∪ OutOnt(A) ∪ ExtOnt(A). If there is no confusion about the agent to which ontologies refer, the reference to A is left out.

3.2 Different Types of States

a) A *state* for a given ontology Ont is an assignment of truth-values {true, false} to the set of ground atoms At(Ont). The *set of all possible states* for an ontology Ont is denoted by STATES(Ont). In particular, STATES(OvOnt) denotes the set of all possible *overall states*. For the agent STATES(IntOnt) is the set of all of its possible *internal states*. Moreover, STATES(InteractionOnt) denotes the set of all *interaction states*.
b) The standard satisfaction relation ⊨ between states and state properties is used: S ⊨ p means that property p holds in state S. Here ⊨ is a predicate symbol in the language, usually used in infix notation, which is comparable to the Holds-predicate in situation calculus. For a property p expressed in Ont, the set of states over Ont in which p holds (i.e., the S with S ⊨ p) is denoted by STATES(Ont, p).
c) For a state S over ontology Ont with sub-ontology Ont', a restriction of S to Ont' can be made, denoted by S|Ont'; this restriction is the member of STATES(Ont') defined by S|Ont'(a) = S(a) if a ∈ At(Ont'). For example, if S is an overall state, i.e., a member of STATES(OvOnt), then the restriction of S to the internal atoms, S|IntOnt is an internal state, i.e., a member of STATES(IntOnt). The restriction operator serves as a form of projection of a combined state onto one of its parts.

4 Expressing Dynamic Properties

To describe the (internal and external) dynamics of a process, explicit reference is made to time. Dynamic properties can be formulated that relate a state at one point in time to a state at another point in time. First, in Section 4.1 the notion of time and trace is defined. Next, in Section 4.2 it is shown in detail how dynamic properties can be expressed formally in TTL.

4.1 Time Frame and Trace

a) A fixed *time frame* T is assumed which is linearly ordered. 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.
b) A *trace* γ over an ontology Ont and time frame T is a time-indexed set of states γ_t ($t \in T$) in STATES(Ont), i.e., a mapping $\gamma : T \rightarrow STATES(Ont)$.

For the specification of dynamic properties, these definitions work fine. However, for some specific operations (such as verification), a dense time frame may cause problems, since it consists of an infinite number of time points. Therefore, in such cases *finite variability* of traces is assumed (i.e., between any two time points only a finite number of state changes occurs). This is discussed in more detail in Section 7.

Traces can be visualised, for example as in Figure 1. Here, the time frame is depicted on the horizontal axis. The different predicates of the ontology are shown on the vertical axis. A dark box on top of the line indicates that the predicate is true during that time period, and a lighter box below the line indicates that it is false. Thus, in the example of Figure 1, predicate1 is true during the whole trace, predicate2 is true from time point 2.5 to time point 4.25, and predicate3 is true from time point 2 to 3 and from 8 to 10.

The set of all traces over ontology Ont is denoted by TRACES(Ont) , i.e., TRACES(Ont) = STATES(Ont)^{T}.

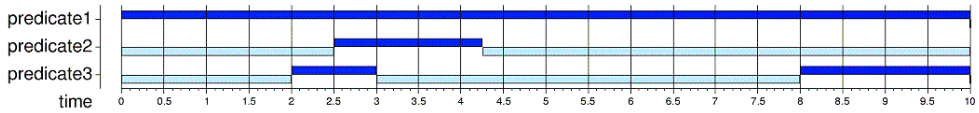

Figure 1. Example visualisation of a trace

c) A *temporal domain description* W is a given set of traces over the overall ontology, i.e., W $\subseteq$ TRACES(OvOnt). This set represents all possible developments over time (respecting the world's laws) of the part of the world considered in the application domain.

d) Given a trace γ over the overall ontology OvOnt, the *input state* of an agent A at time point t, i.e., γ_t |InOnt(A), is also denoted by state(γ, t, input(A)). Analogously, state(γ, t, output(A)) denotes the *output state* of the agent at time point t, state(γ, t, internal(A)) denotes the *internal state*, and state(γ, t, external(A)) denotes the *external world state*. If no confusion is expected about the particular agent, the reference to A can be left out, e.g., as in state(γ, t, input). Moreover, the overall state of a system (agent and environment) at a certain moment, is denoted by state(γ, t).

4.2 Dynamic Properties

To express dynamic properties in a precise manner, it is needed to make explicit references to time points and traces. Comparable to the approach in situation calculus, TTL is built on atoms referring to, e.g., traces, time and state properties. For example, 'in the output state of A in trace γ at time t property p holds' is formalised by state(γ, t, output(A)) $\models$ p. Throughout the remainder of this paper, these kinds of atoms will be referred to as *Holds atoms*. Based on such Holds atoms, *Dynamic Properties* can be built using the usual logical connectives and quantification (for example, over traces, time and state properties). For example, the following dynamic properties can be expressed (given both in an informal format and in TTL):

Observational belief creation

'In any trace, if at any point in time t1 the agent A observes that it is raining, then there exists a point in time t2 after t1 such that at t2 in the trace the agent A believes that it is raining'.

$\forall \gamma \in W \;\; \forall t1$
[state(γ, t1, input(A)) $\models$ observation_result(itsraining) $\Rightarrow \exists t2 \geq t1$ state(γ, t2, internal(A)) $\models$ belief(itsraining)]

Trust monotonicity

'For any two traces $\gamma 1$ and $\gamma 2$, if at each time point t the agent A's experience with public transportation in $\gamma 2$ at t is at least as good as A's experience with public transportation in $\gamma 1$ at t, then in trace $\gamma 2$ at each point in time t, the A's trust is at least as high as A's trust at t in trace $\gamma 1$'.

$\forall \gamma 1, \gamma 2 \in W$
[$\forall t$ [state($\gamma 1$, t, input(A)) $\models$ has_value(experience, v1) &
state($\gamma 2$, t, input(A)) $\models$ has_value(experience, v2) $\Rightarrow$ v1$\leq$ v2] $\Rightarrow$
$\forall t$ [state($\gamma 1$, t, internal(A)) $\models$ has_value(trust, w1) &
state($\gamma 2$, t, internal(A)) $\models$ has_value(trust, w2) $\Rightarrow$ w1$\leq$ w2]]

Instead of the term Dynamic Property, sometimes the term *TTL Formula* is used within this paper (in particular, in Section 7, where the focus is on the technical aspects of the language).

5 Relation to other Languages

In this section, TTL will be compared with a number of existing related languages. In Section 5.1 it is shown how differential equations can be modelled in TTL. In Section 5.2 it is shown how executable properties expressed in LEADSTO can be translated into TTL, and in Section 5.3 it is shown how properties expressed in standard Linear Temporal Logic (LTL) can be translated into TTL.

5.1 Expressing Dynamical Systems Theory in TTL

In this section it is shown how modelling techniques used in the Dynamical Systems approach (DST) [22], such as difference and differential equations, can be represented in TTL. In particular the discrete case is considered. An example of an application is the study of the use of logistic and other difference equations to model growth (and in particular growth spurts) of various cognitive phenomena, e.g., the growth of a child's lexicon between 10 and 17 months, cf. [13]. The logistic difference equation used is $L(n+1) = L(n) (1 + r - r L(n)/K)$. Here r is the growth rate and K the carrying capacity. This equation can be expressed in our temporal trace language on the basis of a discrete time frame (e.g., the natural numbers) in a straightforward manner:

$\forall \gamma \in W \;\; \forall t$
state(γ , t, internal) $\models$ has_value(L, v) $\Rightarrow$
state(γ , t+1, internal) $\models$ has_value(L, v (1 + r - rv/K))

The traces γ satisfying the above dynamic property are the solutions of the difference equation. In a similar manner, it is even possible to address the continuous case, i.e., to express differential equations in TTL. Due to space limitations, this case is not addressed here. See [18] for a detailed explanation how this can be done.

5.2 Expressing Executable Properties in TTL

As mentioned in Section 2, executable languages can be defined as sublanguages of TTL. An example of such a language, which was specifically designed for the simulation of dynamic

processes in terms of both qualitative and quantitative concepts, is the LEADSTO language, cf. [4]. Below, it is shown how dynamic properties expressed in LEADSTO can be translated to TTL.

The LEADSTO language enables one to model direct temporal dependencies between two state properties in states at different points in time. A specification of dynamic properties in LEADSTO format has as advantages that it is executable and that it can often easily be depicted graphically. The format of LEADSTO is defined as follows. Let α and β be state properties of the form 'conjunction of atoms or negations of atoms', and e, f, g, h non-negative real numbers. In LEADSTO the notation $\alpha \rightarrow\!\!\rightarrow_{e, f, g, h} \beta$, means:

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.*

In terms of TTL, the fact that the above statement holds for a trace γ is expressed as follows:

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

5.3 Expressing Standard Temporal Logics in TTL

As mentioned in Section 2, besides executable languages also languages often used for the verification of entailment relations can be defined as sublanguages of TTL. Examples of such languages are LTL and CTL, see, e.g., [2, 14]. In this section, it is briefly shown how dynamic properties expressed as formulae in standard temporal logics can be translated to TTL; in particular, this will be illustrated for the case of LTL. The general idea is that this can be done in a rather straightforward manner by replacing the temporal operators of LTL by quantifiers over time. For example, consider the following LTL formula:

$$G(observation\_result(itsraining) \rightarrow F(belief(itsraining)))$$

where the temporal operator G means 'for all later time points', and F 'for some later time point'. The first operator can be translated into a universal quantifier, whereas the second one can be translated into an existential quantifier. Using TTL, this formula then can be expressed, for example, as follows:

$$\forall t1\ [\ state(\gamma, t1) \models observation\_result(itsraining) \Rightarrow \exists t2 \geq t1\ state(\gamma, t2) \models belief(itsraining)\ ]$$

However, 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 to LTL is the property 'Trust Monotonicity' expressed in Section 4.2. This property cannot be expressed in LTL since it involves the comparison of two different traces ($\gamma 1$ and $\gamma 2$ in this case). This shows that for example LTL can be considered a proper sublanguage of TTL, i.e., a sublanguage not equal to TTL. Similar observations can be made for other well-known temporal logics such as CTL.

6 Applications

The TTL language and its supporting software environment have been applied in a number of research projects in different domains. In general, the research goal in these projects was to analyse the behavioural dynamics of agents in different domains (e.g., [6, 7, 10, 19]). 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. In this section, for an example application, the formalisation of dynamic properties will be discussed.

The example given in this section is taken from [9]. In that paper, the notion of *representational content* for mental states is discussed. It is explored for a case study how representational content can be defined by means of so-called representation relations: expressions that relate an agent's mental state to an entire history (or future) of other states. For the case study, a domain was chosen that is generally viewed as a more complex case: a situation where the agent-environment interaction takes the form of an extensive reciprocal interplay in which both the agent and the environment contribute to the process in a mutual dependency. The specific scenario addressed involves an agent that tries to unlock a front door that sticks. After some interaction between agent and environment (i.e., the agent increases the rotating pressure, but continues to observe resistance of the lock), the agent finally changes the strategy by at the same time pulling the door and turning the key, which unlocks the door.

To model this example, a specific ontology was created, consisting of a number of relevant state properties for the domain, among which the following:

- o1: the agent observes being at the door
- a1(p): the agent turns the key with rotating pressure p
- o2(r): the agent observes resistance r of the lock
- c: the agent has learnt that turning the key is not the right strategy

Based on this ontology, a simulation model has been created by formally specifying the dynamics between the state properties. Using the LEADSTO simulation software [4], a number of simulation traces (similar to the one depicted in Figure 1) have been generated on the basis of this model. After that, representation relations have been formally specified in TTL for each of the internal state properties involved in the model. An example of such an expression is given below, both in semi-formal and in formal notation:

Representational Content of c

'Internal state c occurs iff in the past once o1 occurred, then a1(1), then o2(1), then a1(2), then o2(2), then a1(3), and finally o2(3)'.

$\forall t1,t2,t3,t4,t5,t6,t7$ [$t1 \leq t2 \leq t3 \leq t4 \leq t5 \leq t6 \leq t7$
& state(γ, t1, input) $\models$ o1
& state(γ, t2, output) $\models$ a1(1) & state(γ, t3, input) $\models$ o2(1)
& state(γ, t4, output) $\models$ a1(2) & state(γ, t5, input) $\models$ o2(2)
& state(γ, t6, output) $\models$ a1(3) & state(γ, t7, input) $\models$ o2(3)
$\Rightarrow \exists t8 \geq t7$ state(γ, t8, internal) $\models$ c]
& $\forall t8$ [state(γ, t8, internal) $\models$ c $\Rightarrow$
$\exists t1,t2,t3,t4,t5,t6,t7$ $t1 \leq t2 \leq t3 \leq t4 \leq t5 \leq t6 \leq t7 \leq t8$
& state(γ, t1, input) $\models$ o1
& state(γ, t2, output) $\models$ a1(1) & state(γ, t3, input) $\models$ o2(1)
& state(γ, t4, output) $\models$ a1(2) & state(γ, t5, input) $\models$ o2(2)
& state(γ, t6, output) $\models$ a1(3) & state(γ, t7, input) $\models$ o2(3)]

Finally, these expression were automatically verified (using the TTL checker tool, see Section 7.2) against the generated simulation traces, thereby validating the representation relations.

7 Tools

This section presents the software environment[1] that was built to support the process of specification and automated verification of dynamic properties. Basically, this software environment consists of two closely integrated tools: the Property Editor and the Checker Tool. To explain how these tools work, Section 7.1 describes more details of the TTL language from an implementation perspective. Next, Section 7.2 describes the actual operation of the tools. Finally, Section 7.3 discusses some implementation details of the Checker Tool.

7.1 Details of the TTL language

To enter TTL formulae in the correct format, the TTL Property Editor provides a graphical interface. The user fills in templates and builds up formulae by selecting building blocks from a menu. TTL specifications may also be supplied as plain text. Several definitions are used:

- A *TTL specification* consists of a number of user-defined property definitions and sort definitions. A property definition consists of a header (someprop(v1:s1, v2:s2), property name and formal arguments) and a body. The body is a TTL formula.
- A *TTL formula* is built from basic 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)).
- *Basic TTL formulae* are user-defined properties, Holds atoms, *predefined mathematical properties* (e.g. term1 = term2, term1 > term2) and built-in properties. The semantics of a user-defined property occurring in some TTL formula is one of substitution, not some kind of logic programming (recursion of properties is not allowed).
- *Holds atoms* are introduced in Section 4.2, e.g. state(trace1, t, output(ew)) $\models$ a1 $\wedge$ a2 .
- *Built-in properties* are complex properties encoded into the implementation language.
- TTL formula elements contain *terms* at various places: as restrictions on range variables, as actual parameter values in sub properties, within Holds atoms, and so on. Terms are "Prolog terms" (e.g., fn(t1,t2) , n1, t1 + t3, 1.3). Variables in terms are represented as X:sort1. Terms that are mathematical operations are evaluated, so the operands must be of an appropriate type.

Furthermore, the language has a number of built-in *sorts* for integer, real and range of integers (sorts integer, real, between(i1:integer,i2:integer)). Sorts may be defined by enumerating their elements. There are predefined sorts for the set of all states (sort STATE) and the set of all loaded traces (sort TRACE, the temporal domain description set W introduced in Section 4.1).

TTL formulae usually contain variables referring to time, specifically to time for a state property. In case a dense time frame is used, this may cause problems for the verification process, because an infinite number of time points must be considered. To deal with this problem, in the TTL tools *finite variability* of traces is assumed. This assumption states that between any two time points only a finite number of state changes occurs. Thus, when a property

[1] The software can be downloaded (in combination with the LEADSTO software [4]) from: http://www.cs.vu.nl/~wai/TTL.

is checked against a set of traces, the software determines time-intervals during which all atoms occurring in the property are constant in all traces. A built-in sort interval enumerates these disjoint time intervals. Values of this sort are ordered. A number of primitives are introduced to translate between interval values and time values:

- begin(i:interval) refers to the first time point of interval i.
- end(i:interval) refers to the last time point of interval i.
- interval(t:time) refers to the interval in which time point t occurs.
- time(i:interval) refers to a time point that occurs in interval i.

For an example in which one of these primitives is used, see the following Holds atom:

state(γ:TRACE, time(i:interval), internal) ⊨ a.

Moreover, libraries of predefined properties and functions are available, some generic, others for specific application domains.

7.2 Operation

As mentioned earlier, the TTL software environment comprises 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 the TTL language. It was designed in particular for less experienced users. By means of graphical manipulation and filling in of forms a TTL specification may be constructed (see Figure 2 for an impression). The Checker Tool can be used to check automatically whether a TTL formula holds for a set of traces. Operation of the tools involves three separate actions:

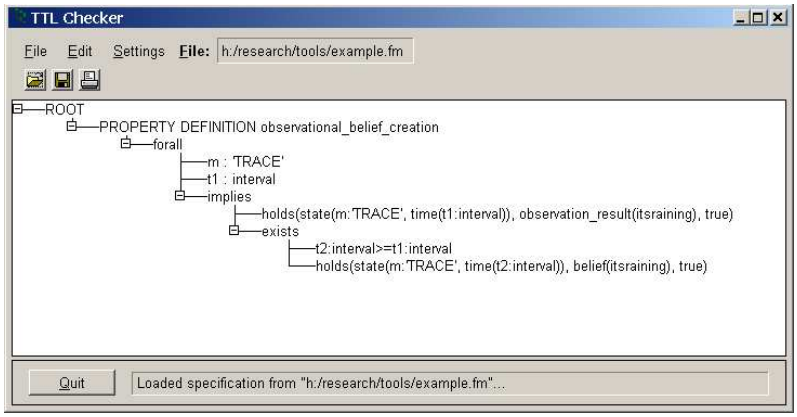

Figure 2. The TTL Checker with Trace Loader

1. Loading, editing, and saving a set of TTL properties and user-defined sorts with the Property Editor (shown in Figure 2).
2. Activating the Trace Manager (not shown in Figure 2): loading and inspecting traces that will be checked and that will constitute the set of traces, the elements of sort TRACE (see section 7.1). Sources of traces can be both results of simulations such as output from the LEADSTO simulation software (see [4]) and empirical traces.
3. Checking a property as shown in Figure 2 against a set of loaded traces. The property is compiled (see Section 7.3 for details) and checked, and the result is presented to the user.

In addition to the above, the TTL Checker has facilities for systematically loading traces and checking properties without user interaction. The software runs on Windows, Solaris and Linux platforms.

7.3 Implementation Details of the Checker

This section describes the algorithm used by the Checker Tool in detail. First, a number of introductory remarks are made:

- The Checker Tool was built specifically for the process of checking TTL formulae against traces. Here, a trace consists of a finite number of state atoms, changing a finite number of times. This has the following consequences:
  - Using intervals instead of (continuous) time in TTL formulae will improve performance of the checking process (by simplifying quantification over time). Nevertheless, both options are possible.
  - Other quantification variables will often refer to arguments of state atoms. There are a finite number of such state atoms. Iterating over values occurring in the traces will often be faster than iterating over all possible values of some variable.
- Checking may involve iteration over many values. Therefore, efficient coding is important. Compiling the formula that needs to be checked into code in the implementation language will improve performance (compared to interpretation).
- Checking may involve frequent access to values of state atoms. For acceptable performance, it is important to assure efficient access to state atoms specific to the formula that is checked.

The implementation is in Prolog (SWI-Prolog, the graphical user interface uses XPCE). A query to check some TTL formula against all loaded traces is compiled into a Prolog clause, which will succeed if the formula holds. The compilation proceeds as follows:

1. Fast access to state atoms is ensured: all atoms occurring in state properties within the TTL formula are gathered. Then, the set of all traces is analysed to determine the time intervals where all those atoms are constant. An index is built for fast access to all those atom values.
2. The TTL formula is compiled into Prolog: the formula is translated by mapping conjunction, disjunction and negation onto Prolog equivalents and by transforming universal quantification into existential quantification. For every variable occurring in the property, information about whether it is bound is maintained. If the first occurrence of some variable in a conjunction is in a Holds atom, then this variable becomes bound by code that binds the variable to successive matching Holds atoms; in a following element of the conjunction, the value may be used in expressions and evaluations in other members of the conjunction. If a variable is not bound by such an occurrence, but should be bound (because it appears in some mathematical operation or comparison), the variable must be bound by generating binding code to bind the variable to successive elements of the variable sort. If the sort is infinite, an error message is generated.

The specific optimizations discussed above make it possible to check realistic dynamic properties with reasonable performance. For an impression of the performance: checking the simply property ‘Observational belief creation’ (see Section 4.2) against a single trace takes less than a second on a regular Personal Computer. Checking more complex properties may

take longer. For example, property ‘Representational Content of c’ (involving eight different time points, see Section 6) took about three minutes to check.

8 Conclusion

Within many domains, among which biological and cognitive areas, multiple interacting processes occur with dynamics that are hard to handle. Current approaches to analyse the dynamics of such processes are often based on differential equations. However, for a number of applications these approaches have serious limitations. For example, within Cognitive Science, approaches based on differential equations are not particularly suitable to model higher-level processes with mainly a qualitative character. To deal with these limitations, this paper presents the predicate logical Temporal Trace Language (TTL) for the formal specification and analysis of dynamic properties. 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.

To support the formal specification and analysis of dynamic properties, a special software environment has been developed for TTL. This environment features both a dedicated Property Editor for building and editing TTL properties and a Checking Tool that enables the formal verification of properties against a set of traces, for example obtained from experiments or simulation. Although this form of checking is not as exhaustive as model checking (which essentially means checking properties on the set of all theoretically possible traces), in return, this makes it possible to specify more expressive properties. Furthermore, more specialised languages can be defined as a sublanguage of TTL. For the purpose of simulation, the executable language LEADSTO has been developed [4]. For the verification of entailment relations, standard temporal languages such as LTL and CTL (see, e.g., [2, 14]) can be defined as sublanguages of TTL.

As mentioned above, TTL 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. [22]. 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, in TTL it is possible to quantify over traces, see e.g. property ‘Trust Monotonicity’ in Section 4.2.

To conclude, the approach proved its value in a number of research projects in different domains. It has been used to analyse behavioural dynamics of agents in cognitive science (e.g., human reasoning [10], creation of consciousness [7], diagnosis of eating disorders [3]), biology (e.g., cell decision processes [17], the dynamics of the heart [8]), social science (e.g., organisation dynamics including organisational change [16], incident management [15]), and artificial intelligence (e.g., verification of Multi-Agent Systems [19], design processes [6], ant colony behaviour [5]).

References

1. 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.
2. 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.
3. Bosse, T., Delfos, M.F., Jonker, C.M., and Treur, J. (2003). Analysis of Adaptive Dynamical Systems for Eating Regulation Disorders. *Proceedings of the 25th Annual Conference of the Cognitive Science Society, CogSci 2003*. Mahwah, NJ: Lawrence Erlbaum Associates, Inc.
4. 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*. Lecture Notes in Artificial Intelligence, vol. 3550. Springer Verlag, pp. 165-178
5. Bosse, T., Jonker, C.M., Schut, M.C., and Treur, J. (2004). Simulation and Analysis of Shared Extended Mind. *Simulation Journal (Transactions of the Society for Modelling and Simulation)*. To appear, 2005. Short version in: Davidsson, P., Gasser, L., Logan, B., and Takadama, K., (eds.), *Proceedings of the First Joint Workshop on Multi-Agent and Multi-Agent-Based Simulation (MAMABS'04)*, pp. 191-200.
6. Bosse, T., Jonker, C.M., and Treur, J. (2004). Analysis of Design Process Dynamics. In: R. Lopez de Mantaras, L. Saitta (eds.), *Proceedings of the 16th European Conference on Artificial Intelligence, ECAI'04*, pp. 293-297.
7. Bosse, T., Jonker, C.M., and Treur, J. (2005). Modelling Body, Emotion, and Core Consciousness. In: Proc. of the Symposium on Next Generation Approaches to Machine Consciousness: Imagination, Development, Intersubjectivity, and Embodiment. Society for the Study of Artificial Intelligence and the Simulation of Behaviour, SSAISB, pp. 95-103.
8. Bosse, T., Jonker, C.M., and Treur, J. (2004). Organisation Modelling for the Dynamics of Complex Biological Processes. In: G. Lindemann, D. Moldt, M. Paolucci, B. Yu (eds.), *Regulated Agent-Based Social Systems, Proc. of the International Workshop on Regulated Agent-Based Social Systems: Theories and Applications, RASTA'02*. Lecture Notes in AI, vol. 2934. Springer Verlag, pp. 92-112.
9. 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*. Lecture Notes in Artificial Intelligence, vol. 3476. Springer Verlag, pp. 270-288.
10. Bosse, T., Jonker, C.M., and Treur, J. (2005). Reasoning by Assumption: Formalisation and Analysis of Human Reasoning Traces. *Cognitive Science Journal*. In press. Short version in: Mira, J., Alvarez, J.R. (eds.), *Proc. of the First International Work-conference on the Interplay between Natural and Artificial Computation, IWINAC'05*. Lecture Notes in Artificial Intelligence, vol. 3561. Springer Verlag, pp. 430-439.
11. Clarke, E.M., Grumberg, O., and Peled, D.A. (2000). *Model Checking*. MIT Press.
12. Gamboa, R., and Kaufmann, M. (2001). Nonstandard Analysis in ACL2. Journal of Automated Reasoning, vol. 27, pp. 323-351.
13. Geert, P. van (1995). Growth Dynamics in Development. In: (Port and van Gelder, 1995), pp. 101-120.

14. Goldblatt, R. (1992). *Logics of Time and Computation*, 2nd edition, CSLI Lecture Notes 7.
15. Hoogendoorn, M., Jonker, C.M., Popova, V., Sharpaskykh, A., Xu, L. (2005). Formal Modelling and Comparing of Disaster Plans. In: Carle, B., and Walle, B. van de, (eds.), *Proceedings of the Second International Conference on Information Systems for Crisis Response and Management ISCRAM '05*, pp. 97-107.
16. Hoogendoorn, M., Jonker, C.M., Schut, M., and Treur, J. (2004). Modelling the Organisation of Organisational Change. In: Giorgini, P., and Winikoff, M., (eds.), *Proceedings of the Sixth International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS'04)*, 2004, pp. 29-46.
17. Jonker, C.M., Snoep, J.L., Treur, J., Westerhoff, H.V., and Wijngaards, W.C.A. (2002). Putting Intentions into Cell Biochemistry: An Artificial Intelligence Perspective. *Journal of Theoretical Biology*, vol. 214, pp. 105-134.
18. Jonker, C.M., and Treur, J. (2003). A Temporal-Interactivist Perspective on the Dynamics of Mental States. *Cognitive Systems Research Journal*, vol. 4, 2003, pp. 137-155. Short version in: C. Castelfranchi and W.L. Johnson (eds.), *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems, AAMAS'02*. ACM Press, pp. 865-872.
19. Jonker, C.M., and Treur, J. (2002). Compositional Verification of Multi-Agent Systems: a Formal Analysis of Pro-activeness and Reactiveness. *International Journal of Cooperative Information Systems*, vol. 11, pp. 51-92.
20. Kim, J. (1998). Mind in a Physical world: an Essay on the Mind-Body Problem and Mental Causation. MIT Press, Cambridge, Mass.
21. Manna, Z., and Pnueli, A. (1995). *Temporal Verification of Reactive Systems: Safety*. Springer Verlag.
22. Port, R.F., Gelder, T. van (eds.) (1995). *Mind as Motion: Explorations in the Dynamics of Cognition*. MIT Press, Cambridge, Mass.
23. Stirling , C. (2001). *Modal and Temporal Properties of Processes*. Springer Verlag.

View publication stats