Execution of TILCO Specifications Technical Report

Pierfrancesco Bellini, Andrea Giotti, Paolo Nesi
Department of Systems and Informatics, University of Florence
Via S. Marta 3, 50139 Firenze, Italy, tel.: +39-055-4796523, fax.:+39-055-4796363
email: nesi@ingfi1.ing.unifi.it, www: http://www.dsi.unifi.it/~nesi

1 Introduction

For the specification of real-time system behaviour, the definition of temporal constraints amongst events and actions is needed [1], [2], [3]: properties of invariance, precedence amongst events, periodicity, liveness and safety conditions, etc. To this end, several different temporal logics with different degrees of expressiveness have been proposed. Some of them are based on propositional logic — e.g., PTL, TPTL, RTTL, ITL ([3]). Others are based on first or higher order logic — e.g., TRIO [4], MTL [5], interval temporal logic [6], TILCO [7].

According to [7], in which TILCO (Temporal Interval Logic with Compositional Operators) temporal logic has been presented, only a few examples of quantitative temporal logics exist. In these cases, an operator expressing the distance between time points is usually defined. First-order temporal logics that provide a metric for time usually allow quantification over the temporal domain — e.g., RTL [8], MTL [9], TRIO [4] — whereas a prohibition of this kind of quantification has been shown to be a necessary condition for the existence of feasible automated verification mechanisms [10]. All these temporal logics are based on time points rather than on intervals and provide a sharp distinction between past and future (e.g., MTL: **G**, **H**; TRIO: **Past()** and **Futr()**). In contrast, TILCO does not allow the quantification over time, presents a uniform model for time from past to future and unique operators for stating facts and events along the time axis [7], [11], [12].

The problem of executability of specifications given by means of temporal logics has often been misunderstood with that of validation. The validation of a logic specification usually consists in proving high-level properties, which are also given in the form of logical expressions, by means of theorem provers [13]. Other techniques are based on the so-called history-checking [14]. In these cases, the time ordering is not satisfied, history-checking is mainly oriented to validating specifications against off-line generated histories of system inputs and outputs.

There are at least three different definitions of executability [15]. According to concept of execution, which is closer to that used in software engineering, the execution of a specification leads to the on-line production of outputs on the basis of inputs and system status. Thus the time ordering of events must be satisfied with real-time constraints. This concept of execution is quite far from that of validation since it is supposed that the specification under execution is a verified and validated specification.

In this report, the *TILCO-Executor* algorithm for executing TILCO is presented. The meaning adopted by *TILCO-Executor* for executability consists in using the specification itself as a prototype of the real-time system, thus allowing, in each time instant, for the *on-line* generation of system outputs on the basis of current inputs and internal state and inputs and outputs history. The algorithm is capable of executing a fragment of TILCO specifications, and can be used for execution other first order temporal logics under similar restrictions. In order to simplify the execution of TILCO specifications a Basic Temporal Logic (BTL) with a lower number of operators has been introduced, this is capable of expressing all complex operators of TILCO. In the following it is briefly described how to translate TILCO to BTL. Inference rules for BTL are presented and it is described how to graphically represent the BTL specifications as Temporal Inference Networks, TIN. An algorithm for executing temporal inference network is presented and the possibility of real-time execution is highlighted.

2 Overview of TILCO

TILCO is a logic language which can be used to specify temporal constraints in either a qualitative or a quantitative way. This allows both to define ordering relationships amongst events and to express durations of facts by an absolute measure. A formalization of TILCO has been implemented in the theorem prover Isabelle/HOL, together with a set of sound axioms and deduction tactics [7], to allow the automatic verification of TILCO properties. Such properties can be expressed in TILCO, so the same language can be used to specify both general requirements, detailed behaviour of real-time systems, specification and execution. TILCO includes the concepts of typed variables and constants by providing a set of basic types and allowing the definition of new types. Finally, causal propositional TILCO specifications are executable in real-time as shown in the rest of the report.

2.1 TILCO Operators

TILCO's *temporal operators* have been added to FOL by leaving the evaluation time implicit, so the meaning of a TILCO formula is given with respect to current time. Time is discrete and linear and the temporal domain is the set of integers $\mathbb{Z}$. The minimum time interval corresponds to one instant, the current time instant is represented by 0 and positive (negative) numbers represent future (past) time instants.

The basic entity in TILCO is temporal interval, the boundaries of which can be either included or excluded by using the usual notation with squared, ("[", "]") or round ("(", ")") brackets, respectively. Time instants are regarded as closed intervals composed of a single point while symbols $+\infty$ and $-\infty$ can be used as boundaries to denote infinite intervals. Thus, TILCO allows both the specification of *facts* on intervals and *events* in time instants.

The basic TILCO *temporal operators* are:

- "**@**", universal quantification ($\forall$) over a temporal interval;
- "**?**", existential quantification ($\exists$) over a temporal interval;
- "**until**", to express that either a predicate will always be true in the future, or it will be true until another predicate will become true;
- "**since**", to express that either a predicate has always been true in the past, or it has been true since another predicate has become true.

For instance, $A \,@\, i$ is true if formula A is true in every instant of the interval i with respect to the current time instant. By denoting with t the current time instant, $(A \,@\, i)^{(t)} \iff \forall x \in i . A^{(t+x)}$ holds. The $^{(t)}$ notation has the purpose to put in evidence the formula evaluation instant and is omitted in the following, such as in several other temporal logics. This approach is called implicit time and it is used for example in RTL and TRIO [3].

Similarly, **until**(A, B) is true when evaluated in t if B will always be true in the future with respect to t or if B will be true in the interval $(t, t+x)$ and A will be true in $t+x$ with $x > 0$. This definition of **until** does not require the occurrence of A in the future, so as to correspond to the *weak until* operator defined in PTL [16]. The operators **until** and **since** express the same concept for future and past, respectively, and can be used to express ordering relationships amongst events.

Classical operators of temporal logic as eventually ($\Diamond$) and henceforth ($\Box$) can be easily obtained by using TILCO operators with infinite intervals. TILCO can be regarded as a generalization of most of the interval logics presented in literature[7], [3], with the addition of a metric for time. Tabable 1 provides some examples of TILCO formulae, accompanied by an explanation of their meaning.

$(A \,?\, [0,t]) \,@\, [0,+\infty)$	A will become true within t for each time instant in the future (response)
$(A \Rightarrow B) \,@\, [0,t]$	if A is true within t, then also B will be true at the same time
$(A \Rightarrow (B \,?\, i)) \,@\, j$	A leads to an assertion of B in i for each time instant of j
$(A \Rightarrow (B \,@\, i)) \,@\, j$	A leads to the assertion of B in the whole interval i for each time instant of j
$(A \Rightarrow (B \,@\, i)) \,?\, j$	A leads to the assertion of B in the whole interval i in at least a time instant of j

Table 1: Examples of TILCO formulae

3 From Specification to Execute of Temporal Logics

A TILCO specification is made of:

- a set of *formulae* which define system behaviour and temporal constraints (i.e., if the water level is over the danger level for three time instants then the alarm has to be switched on);
- a set of *input signals*, typed variables representing information acquired from outside (i.e., water level, temperature, etc.), the value of which can change due to external events;
- a set of *output signals*, typed variables representing information produced to outside (i.e., alarm enable, pump speed, etc.), which can be forced to assume a value by some predicate through an assignment and lead to a change in the external environment;
- a set of *auxiliary signals/internal variables*, typed variables representing the information processed internally and constituting the system state, which can be either read as input signals or forced to a value as output signals.

System inputs, outputs and internal variables can assume only one value at each time instant.

A TILCO specification can be validated to verify whether it corresponds to the system requirements. System validation is performed by proving that high-level desired properties, as safety or liveness, are satisfied by the specification of the system on the basis of *TILCO-Theory* implemented on Isabelle/HOL theorem prover. These properties can be expressed by means of other TILCO formulae, thus the same language is used to specify both the system general requirements and its detailed behaviour as shown in [13]. If during the validation phase it is found that a desired property cannot be deduced from the system specification, then the specification is incomplete. If the property must be satisfied by the system and does not contradict the specified behaviour, a new TILCO formula has to be added to the specification to express the wished property. This allows an incremental system specification by theorem proving, with an absolute degree of confidence due to the support of *TILCO-Theory* on Isabelle. This process of refinement ends when the detailed specification turns out to be an *implementation of the top-level specification.*

In this context, executability means the capability to use the system specification itself as a prototype of the real-time system, thus allowing, in each time instant, the *on-line* generation of system outputs on the basis of present inputs and internal state. Given an interpretation $\mathcal{I}_{in}$ of input signals' histories, to execute a specification formula S formally means to deduce the interpretations $\mathcal{I}_{out}$, $\mathcal{I}_{aux}$ of output and auxiliary signals' histories which satisfy S by applying a set of inference rules which will be detailed later on. When this is possible, the specification can be directly executed in real-time by means of *TILCO-Executor* without having to translate it in a programming language.

In literature, there are only few executable temporal logics which can be used to build a system prototype. The execution or simulation of logic specifications with the intent of producing system outputs in the correct temporal order while meeting the temporal constraints is a quite difficult problem. The difficulty mainly depends on the computational complexity of the algorithms proposed which makes their adoption for executing logic specifications in real-time impossible. Usually, the computational complexity is $O(D^h)$ where D is the domain size of the nested quantifications which have been specified, for both time dependent and independent variables, and h is number of these quantifications.

For the design and the implementation of a temporal logic executor, two aspects have been taken into account:

- the definition of an executable semantics for common temporal logic operators based on time instant, intervals, event ordering, etc.;
- the definition and implementation of an algorithm based on the adopted semantics and allowing real-time execution of the specification.

According to these aspects it has been decided to adopt a minimum number of temporal logic operators, those strictly necessary to implement at higher level the TILCO language.

The temporal logic executor presented in this report, called TINX (Temporal Inference Network eXecutor), can execute specifications in a Basic Temporal Logic, BTL, which can be used to automatically rewrite TILCO propositional logic specifications by using simple rules. FOL specifications involving quantifications on finite domains can be rewritten in propositional logic and thus in BTL.

4 Basic Temporal Logic

Basic Temporal Logic (BTL) is a propositional logic with And ($\wedge$), Or ($\vee$) and with the Delay (∂) operator, where a true/false value is associated to each signal for each time instant.

The BTL syntax is the following:

$$\begin{aligned} formula \quad := \quad & signal \mid \neg signal \mid \\ & formula \wedge formula \mid formula \vee formula \mid \\ & \partial\, formula \mid \partial^k formula \mid (formula) \end{aligned}$$

As can be noted from the syntax the $\neg$ operator can be applied only to signals and not to complex formulae. Moreover, a BTL formula can be evaluated at an instant t with $@$ operator or asserted on the whole temporal horizon:

$$assertion := formula\,@\,t \mid \neg formula\,@\,t \mid \bowtie formula$$

where $F\,@\,t$ means that formula F is true at time instant t, $\neg F\,@\,t$ means that formula F is false at time instant t and $\bowtie F$ means that formula F is always true.
The operators are inductively defined in the following way:

$$\begin{aligned} (\neg A)\,@\,t &\iff \neg(A)\,@\,t \\ (A \wedge B)\,@\,t &\iff A\,@\,t \wedge B\,@\,t \\ (A \vee B)\,@\,t &\iff A\,@\,t \vee B\,@\,t \\ (\partial\, A)\,@\,t &\iff (\partial^1\, A)\,@\,t \\ (\partial^k\, A)\,@\,t &\iff A\,@\,(t-k) \end{aligned}$$

A BTL specification is a formula always true.

4.1 TILCO in BTL

TILCO temporal operators can be easily translated into BTL by using the following formulas:

$$\begin{aligned} A\,@\,[m,M] &\iff \bigwedge_{i=m}^{M} \partial^{-i}\, A, \\ A\,?\,[m,M] &\iff \bigvee_{i=m}^{M} \partial^{-i}\, A, \\ \mathbf{until}(A,B) &\iff A \vee (B \wedge \partial^{-1}\, \mathbf{until}(A,B)), \\ \mathbf{since}(A,B) &\iff A \vee (B \wedge \partial^{1}\, \mathbf{since}(A,B)). \end{aligned}$$

In order to automatically translate a TILCO formula to BTL some steps are needed. As a first step, the $\Rightarrow$ and $\iff$ operators are translated by using the following rewriting rules:

$$\begin{aligned} A \Rightarrow B &\mapsto \neg A \vee B \\ A \iff B &\mapsto (A \Rightarrow B) \wedge (B \Rightarrow A) \end{aligned}$$

after that, the $\neg$ operator has to be propagated down to the signals using by the following rewriting rules:

$$\begin{aligned} \neg(A \wedge B) &\mapsto \neg A \vee \neg B \\ \neg(A \vee B) &\mapsto \neg A \wedge \neg B \\ \neg(\neg A) &\mapsto A \\ \neg(A\,?\,i) &\mapsto \neg A\,@\,i \\ \neg(A\,@\,i) &\mapsto \neg A\,?\,i \end{aligned}$$

Since there is no rule to propagate $\neg$ operator for **until** and **since**, the $\neg$ **until** and $\neg$ **since** operators are translated by using the following properties:

$$\begin{aligned} \neg\,\mathbf{until}(\neg A, \neg B) &\iff A \wedge (B \vee \partial^{-1} \neg\,\mathbf{until}(\neg A, \neg B)) \\ \neg\,\mathbf{since}(\neg A, \neg B) &\iff A \wedge (B \vee \partial^{1} \neg\,\mathbf{since}(\neg A, \neg B)) \end{aligned}$$

Moreover, since the $\neg$ **until** and $\neg$ **since** operators generally are not in the form needed to use these properties, the following rewite rules can be used:

$$\begin{aligned} \neg\,\mathbf{until}(A, B) &\mapsto \neg\,\mathbf{until}(\neg(\neg A), \neg(\neg B)) \\ \neg\,\mathbf{until}(\neg A, B) &\mapsto \neg\,\mathbf{until}(\neg A, \neg(\neg B)) \\ \neg\,\mathbf{until}(A, \neg B) &\mapsto \neg\,\mathbf{until}(\neg(\neg A), \neg B) \\ \neg\,\mathbf{since}(A, B) &\mapsto \neg\,\mathbf{since}(\neg(\neg A), \neg(\neg B)) \\ \neg\,\mathbf{since}(\neg A, B) &\mapsto \neg\,\mathbf{since}(\neg A, \neg(\neg B)) \\ \neg\,\mathbf{since}(A, \neg B) &\mapsto \neg\,\mathbf{since}(\neg(\neg A), \neg B) \end{aligned}$$

Whereas **@** and **?** operators can be simply substituted by equivalent BTL expressions, the BTL definition of **since** and **until** is recursive and thus requires the introduction of new literals. For example, TILCO formula:

$$A\,\mathbf{?}\,(-3, 0) \wedge \mathbf{since}(A, \neg B) \Rightarrow \mathbf{until}(A, B)$$

after $\Rightarrow$ substitution and $\neg$ propagation becomes:

$$\neg A\,\mathbf{@}\,(-3, 0) \vee \neg\,\mathbf{since}(\neg(\neg A), \neg B) \vee \mathbf{until}(A, B)$$

then the substitution of the temporal operators (**@** , $\neg$ **since**, **until**) is made, leading to the BTL formula:

$$\begin{aligned} &((\partial^{-2} \neg A \wedge \partial^{-1} \neg A) \vee \mathit{nsince} \vee \mathit{until}) \wedge \\ &(\neg \mathit{nsince} \vee (\neg A \wedge (B \vee \partial\, \mathit{nsince}))) \wedge \\ &(\mathit{nsince} \vee (A \vee (\neg B \wedge \partial\, \neg \mathit{nsince}))) \wedge \\ &(\neg \mathit{until} \vee (A \vee (B \wedge \partial^{-1}\, \mathit{until}))) \wedge \\ &(\mathit{until} \vee (\neg A \wedge (\neg B \vee \partial^{-1} \neg \mathit{until}))) \end{aligned}$$

where signals *nsince* and *until* have been introduced to represent the $\neg$ **since** and **until** formulae.

5 Temporal inference with BTL

An inference process produces new information by applying inference rules to known facts. These rules generally are composed of an antecedent which represents the condition to be satisfied, and the consequent which represents the new information inferred.
For example:

$$(A \wedge B)\,\mathbf{@}\,t \vdash A\,\mathbf{@}\,t, B\,\mathbf{@}\,t$$

is an inference rule which states that if $A \wedge B$ is true at time t then the fact that A and B are true at time t

can be inferred. The inference rules chosen for BTL are the following:

$$
\begin{array}{rcll}
(A \wedge B) \,@\, t & \vdash & A \,@\, t, B \,@\, t & \downarrow \top \\
\neg(A \wedge B) \,@\, t, A \,@\, t & \vdash & \neg B \,@\, t & \downarrow \bot \\
\neg(A \wedge B) \,@\, t, B \,@\, t & \vdash & \neg A \,@\, t & \downarrow \bot \\
\neg A \,@\, t & \vdash & \neg(A \wedge B) \,@\, t & \uparrow \bot \\
\neg B \,@\, t & \vdash & \neg(A \wedge B) \,@\, t & \uparrow \bot \\
A \,@\, t, B \,@\, t & \vdash & (A \wedge B) \,@\, t & \uparrow \top \\
 & & & \\
(A \vee B) \,@\, t, \neg A \,@\, t & \vdash & B \,@\, t & \downarrow \top \\
(A \vee B) \,@\, t, \neg B \,@\, t & \vdash & A \,@\, t & \downarrow \top \\
\neg A \,@\, t, \neg B \,@\, t & \vdash & \neg(A \vee B) \,@\, t & \uparrow \bot \\
\neg(A \vee B) \,@\, t & \vdash & \neg A \,@\, t, \neg B \,@\, t & \downarrow \bot \\
A \,@\, t & \vdash & (A \vee B) \,@\, t & \uparrow \top \\
B \,@\, t & \vdash & (A \vee B) \,@\, t & \uparrow \top \\
 & & & \\
(\partial^k A) \,@\, t & \vdash & A \,@\, (t - k) & \downarrow \top \\
\neg(\partial^k A) \,@\, t & \vdash & \neg A \,@\, (t - k) & \downarrow \bot \\
A \,@\, t & \vdash & (\partial^k A) \,@\, (t + k) & \uparrow \top \\
\neg A \,@\, t & \vdash & \neg(\partial^k A) \,@\, (t + k) & \uparrow \bot \\
 & & & \\
l \,@\, t & \vdash & \neg(\neg l) \,@\, t & \rightarrow \bot \\
\neg(l) \,@\, t & \vdash & (\neg l) \,@\, t & \rightarrow \top \\
(\neg l) \,@\, t & \vdash & \neg(l) \,@\, t & \rightarrow \bot \\
\neg(\neg l) \,@\, t & \vdash & l \,@\, t & \rightarrow \top
\end{array}
$$

This set of rules have been classified in the following classes:

$\downarrow \top$ rules which propagate the truth from a parent formula to a child formula;

$\uparrow \top$ rules which propagate the truth from a child formula to a parent formula;

$\downarrow \bot$ rules which propagate the falseness from a parent formula to a child formula;

$\uparrow \bot$ rules which propagate the falseness from a child formula to a parent formula;

$\rightarrow \top$ rules which propagate the falseness of a signal to truth of the complementary signal;

$\rightarrow \bot$ rules which propagate the truth of a signal to falseness of the complementary signal;

It should be noted that for all operators, after applying a $\uparrow$ rule, a $\downarrow$ rule cannot produce new knowledge. For example if $A \,@\, t$ is true then rule $A \,@\, t \vdash (A \vee B) \,@\, t$ ($\uparrow \top$ rule) can be applied and if $\neg B \,@\, t$ is true then from $(A \vee B) \,@\, t, \neg B \,@\, t \vdash A \,@\, t$ ($\downarrow \top$ rule) can be found that $A \,@\, t$ is true, but it was already known. Moreover if we consider all the $\bot$ expressions at a given time instant these were not found using $\downarrow \bot$ rules. This can be proved by considering that all the *a priori* information not associated to a signal is known true (the specification) and the application of a $\downarrow \bot$ rule after a $\uparrow \bot$ one does not produce any new knowledge. For this reason this kind of rules can be dropped. Also the $\uparrow \top$ rules are not needed since to their result cannot be applied a $\uparrow \bot$ rule (the rule produces a $\top$ while this rule needs a $\bot$) nor a $\downarrow \top$ rule (no new information can be inferred), only another $\uparrow \top$ rule can be applied but this rules will never produce new knowledge about a signal. For this reason also the $\uparrow \top$ rules have been dropped. As a result also the $\rightarrow \top$ rules have been dropped since their result could be used only with $\uparrow \top$ rules which are nomore present.

Therefore, the final set of inference rules for BTL is:

$$
\begin{array}{rcll}
(A \wedge B) \,@\, t & \vdash & A \,@\, t, B \,@\, t & \downarrow \top \\
\neg A \,@\, t & \vdash & \neg(A \wedge B) \,@\, t & \uparrow \bot \\
\neg B \,@\, t & \vdash & \neg(A \wedge B) \,@\, t & \uparrow \bot \\
 & & & \\
(A \vee B) \,@\, t, \neg A \,@\, t & \vdash & B \,@\, t & \downarrow \top \\
(A \vee B) \,@\, t, \neg B \,@\, t & \vdash & A \,@\, t & \downarrow \top \\
\neg A \,@\, t, \neg B \,@\, t & \vdash & \neg(A \vee B) \,@\, t & \uparrow \bot \\
 & & & \\
(\partial^k A) \,@\, t & \vdash & A \,@\, (t - k) & \downarrow \top \\
\neg A \,@\, t & \vdash & \neg(\partial^k A) \,@\, (t + k) & \uparrow \bot \\
 & & & \\
l \,@\, t & \vdash & \neg(\neg l) \,@\, t & \rightarrow \bot \\
(\neg l) \,@\, t & \vdash & \neg(l) \,@\, t & \rightarrow \bot
\end{array}
$$

5.1 A graphical view of temporal inference

Given a BTL specification, a graphical view of the formula can be built from its syntax tree. For each operator the corresponding component presented in Figure 1 can be used.

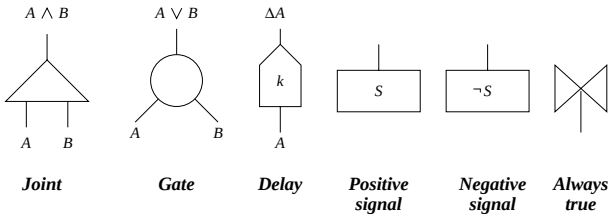

Figure 1: The basic components

For example formula:

$$\bowtie(\partial\, \neg s \wedge s \Rightarrow up)$$

states that signal up is true if at the previous time instant signal s was false and now s is true. This formula is not written in BTL since there is the imply ($\Rightarrow$) operator, but it can be translated to BTL by using the presented rewriting rules:

$$\bowtie(\partial\, s \vee \neg s \vee up)$$

The syntax tree of this formula is depicted in Figure 2.

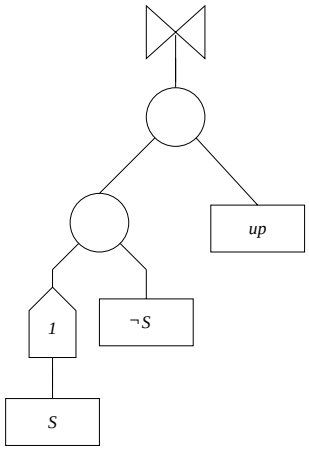

Figure 2: Syntax tree of up

A direction can be associated with the arcs connecting a parent to a sub-tree, either from the parent to the sub-tree if the sub-expression is true or from the sub-tree to the parent if the sub-expression is false. This association depends on the evaluation time instant, thus for each time instant different arc directions may exist. For instance, in the up example, the fact that s is known false at time t is represented by an arrow from the s signal to the delay component. Moreover, since the always operator means that the expression is always true there is also an arrow from the always component to the gate as depicted in Figure 3. With the association of true/false with the arc direction, the inference rules reported in the previous section can be represented in a graphical way, see Figure 4 where the black arrows represent the antecedents of the rules and the white arrows the consequents.

When returning to the up example, an arrow from the delay to the parent gate can be drawn at the following time instant by applying rule (D2) to the presented configuration. If at this time instant signal s is true, an arrow from the $\neg s$ component to its parent gate can be added. At this point, rule (G3) can be applied to draw an arrow from this gate to the upper one. Since the always component imposes an arrow from it to the upper gate then rule (G1) can be applied to draw an arrow from the gate to the up signal that can be interpreted as up being true. This is correct since s is true and it was false at the previous time instant. This deduction sequence is depicted in Figure 5. The inference rules can be applied also to the case in which up signal is known false and s signal is known true: the fact that s has to be true at the previous time instant can be derived by using these rules.

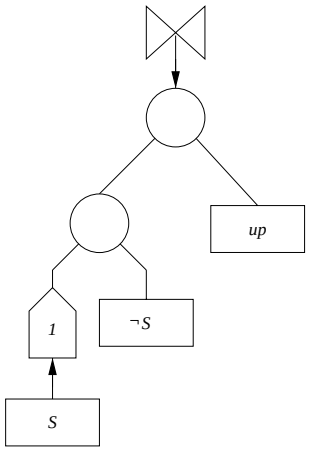

Figure 3: The up example, arrow configuration

6 Temporal inference networks

A temporal inference network is built from the syntax tree of a specification by applying the simplifications reported in Figure 6 to eliminate the $\bowtie$ component. Simplifications (1a) and (1b) are based on the following properties:

$$\forall t.(A \wedge B)\,@\,t \iff (\forall t.A\,@\,t) \wedge (\forall t.B\,@\,t),$$
$$\forall t.(\partial^k A)\,@\,t \iff (\forall t.A\,@\,(t-k)) \iff (\forall t.A\,@\,t),$$

which permit the propagation of $\bowtie$ to the lower levels.

Simplification (1c) has been derived by considering the two $\downarrow \top$ inference rules of $\vee$ operator:

$$(A \vee B)\,@\,t, \neg A\,@\,t \;\vdash\; B\,@\,t$$
$$(A \vee B)\,@\,t, \neg B\,@\,t \;\vdash\; A\,@\,t$$

When $(A \vee B)\,@\,t$ is true for all t, if sub-expression A is false then B is true. Thus, if there is an arrow from A to the gate then there is an arrow from the gate to B, and the same property holds in reverse order (from B to A), as reported in Figure 4. The same behaviour is accomplished by eliminating the $\bowtie$ component and directly connecting sub-tree A to sub-tree B.

Other substitutions to eliminate signal components are reported in Figure 7. Simplification (2a) permits to substitute two signal nodes of the same type (either asserted or negated occurrences of the same signal) with only one signal node, with the same behaviour. If the signal (asserted or negated) is false, an arrow is drawn from the signal node to the joint node, and the application of rule (J1) permits to derive the correct directions of arrows for the two connections. On the other hand if a signal (asserted or negated) is derived as true from upper levels, the signal node is derived as true by using rule (J2) or (J3). This substitution rule allows to replace all the signal nodes of the same type with only one signal node.

Simplification (2b) permits to substitute two complementary signal nodes (one asserted and one negated) with a arc connecting its parents. In fact, an arrow from the asserted to the negated node can be drawn if the signal is known true or in the opposite direction if it is known false. Moreover, if the signal is derived as true then an arrow from the asserted parent to the negated one can be drawn or in the opposite direction if the signal is derived as false. In this way a signal is represented in the network by an arc labeled both with the signal name and with an arrow in the middle meaning the direction to be given when the signal is known true.

If for a signal s there are present only asserted or negated nodes then tautology $(s \vee \neg s)$ has to be added to the specification to allow the substitution for s. By applying these simplifications the resulting graph has only the three basic types of nodes (joint, gate and delay), and can be executed by using the defined inference rules.

Since in the temporal inference network of the *up* example the signal *up* is present only as asserted, tautology $up \vee \neg up$ has to be added to the BTL specification:

$$\bowtie((\partial\, s \vee \neg s \vee up) \wedge (up \vee \neg up))$$

In the up left part of Figure 8 the syntax tree of this formula is reported, together with the steps showing the application of the simplifications (1a, 1c, 2b, 2a, 2b).

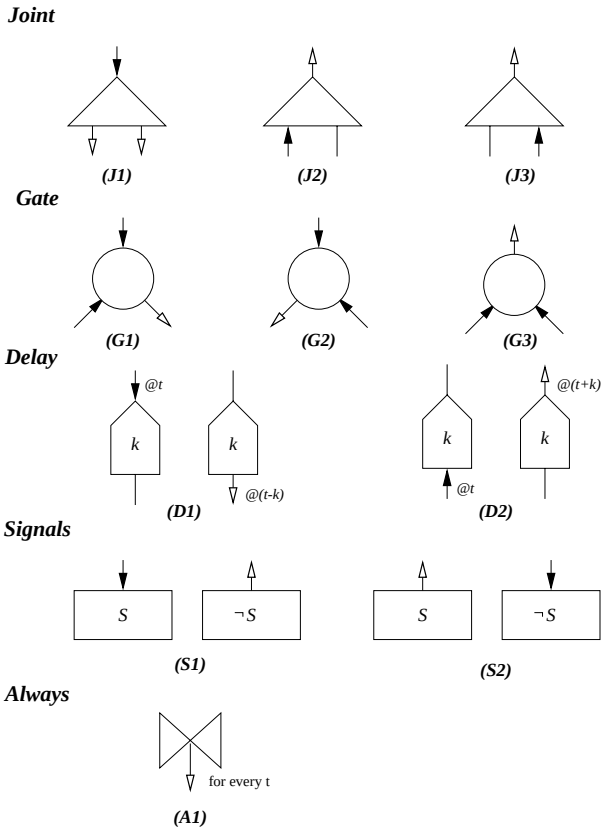

Figure 4: Graphical inference rules

6.1 Sketch of the execution algorithm

A temporal inference network $\mathcal{N}$ is formally defined as:

$$\mathcal{N} =< N, k, p, l, r >$$

where:

- N is the set of nodes;
- $k : N \rightarrow \{\wedge, \vee, \partial^k\}$ is a function which returns for each node its type;
- $p : N \rightarrow N$ is a function which returns for each node its parent;
- $l : N \rightarrow N$ is a function which returns for each node its left son;
- $r : N \rightarrow N$ is a function which returns for each node its right son;

An arrow from node n_1 to node n_2 of the network is represented as a couple (n_1, n_2), but at each time instant the network can have different arrow directions, thus each arrow has a time instant. The fact that there is an arrow from node n_1 to node n_2 at time t is represented by the 3-ple (n_1, n_2, t) or in a more readable way $(n_1, n_2)\,@\,t$. Such an element is also called an event.

The set $T = [0, T_{max}] \subset \mathbb{Z}$ represents the temporal horizon. The known information of a network is a subset of $D = N \times N \times T$, whereas the inference process can be seen as a sequence of subsets of D, $\Phi_0 \subset \Phi_1 \subset \ldots \subset \Phi_n \subseteq D$. The elements of Φ_{k+1}/Φ_k represent the new information which has been deduced from Φ_k by using the inference rules or acquired from outside via inputs.

A general schema for the algorithm is:

repeat forever
  acquire information from outside;
  if there is information to be processed
    select an event to be processed;
    infer new events from the event selected;

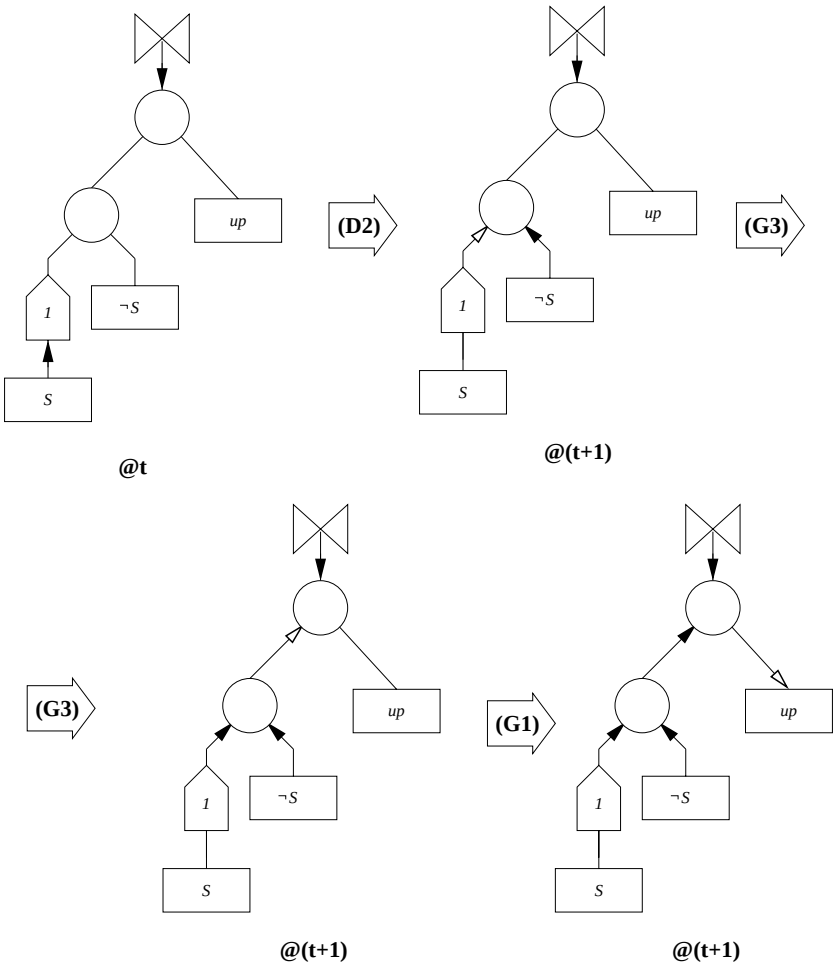

Figure 5: The up example execution

end if
add new information (inferred or acquired)
to information to be processed;
end repeat

This process can be detailed in the following algorithm, where:

- $\Omega_0 \subset D$ is the set of initial knowledge;
- $\omega()$ is a function which for each evaluation returns a subset of D representing information acquired from outside;
- $\lambda(e \,@\, t, S) \subset D$ is a function which determines the events derivable from $e \,@\, t$ on the basis of knowledge $S \subset D$;
- variable $\Phi \subset D$ is the set of available knowledge;
- variable $\Delta \subset D$ is the set of acquired information which has not been processed yet;
- variable Ψ is the set of new knowledge found during current iteration.

$\Phi \leftarrow \Omega_0$; // initial info
$\Delta \leftarrow \Omega_0$; // initial info which has to be processed
repeat forever
$\Psi \leftarrow \omega() \setminus \Phi$; // acquire new info from outside
if $\Delta \neq \emptyset$ **then**// if there is something to process...
choose $e \,@\, t \in \Delta$ **with the minimum** t;
$\Psi \leftarrow \Psi \cup \lambda(e \,@\, t, \Phi \setminus \Delta) \setminus \Phi$;
// determine the new info inferred by $e \,@\, t$
$\Delta \leftarrow \Delta \setminus \{e \,@\, t\}$;

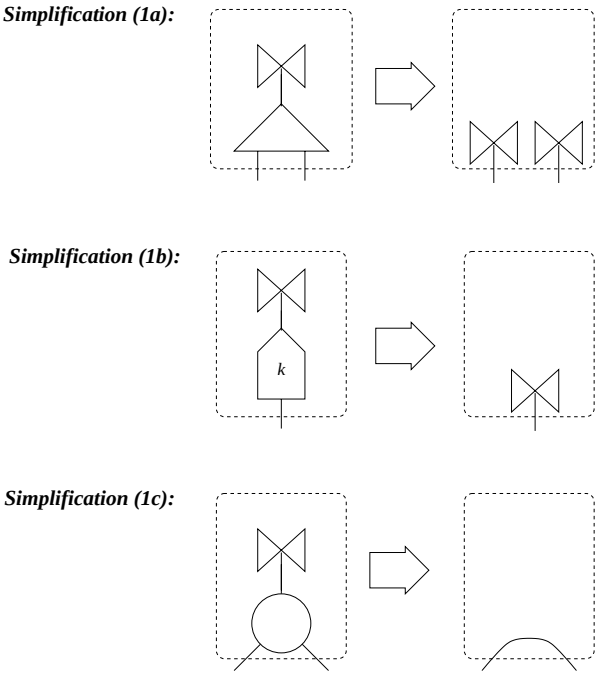

Figure 6: Simplifications to eliminate $\bowtie$

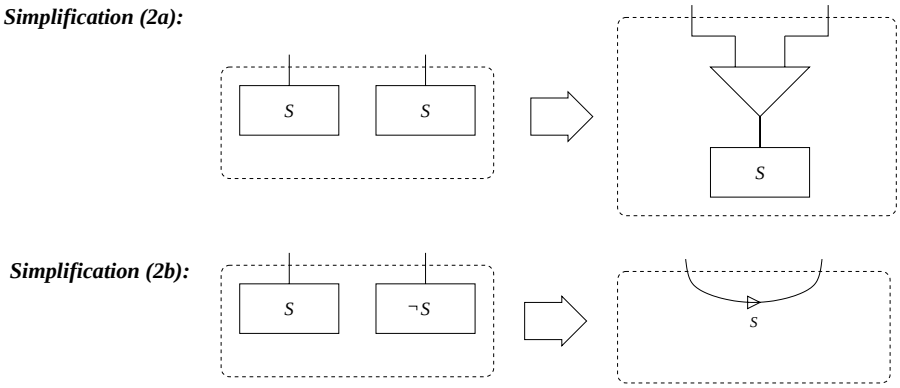

Figure 7: Simplifications to eliminate signals

// drop $e \,@\, t$ from info to be processed
endif
$\Delta \leftarrow \Delta \cup \Psi$;// add the new info as to be processed
$\Phi \leftarrow \Phi \cup \Psi$; // add the new info to the current knowledge
endrepeat

The function $\lambda((n_i, n_o) \,@\, t, \Phi)$ can be computed by using the temporal inference rules presented in the previous section.

To obtain an evaluation of the amount of steps needed to produce the whole information for a single time instant, the inference function λ has to be causal. This means that it infers events in the present or in the future with respect to event $e \,@\, t$:

$$\forall e \,@\, t, S, f \,@\, t'. \;\; f \,@\, t' \in \lambda(e \,@\, t, S) \Rightarrow t \leq t'$$

With this assumption, and by assuming that all the events in each sample of ω share the same non-decreasing time, an amount of steps equal to the amount of network arcs N_a is needed in the worst case to drop from Δ all the events with time τ, where τ is the minimum time of the events in $\Delta \cup \omega()$. In fact, at each step the algorithm chooses an event with time not less than τ (neither λ nor ω can introduce in Δ events with time previous than τ) and if an arc has been processed for a time instant it will be never reprocessed for the same time instant. Thus it is possible to deduce that in the worst case all the arcs of the network are processed.

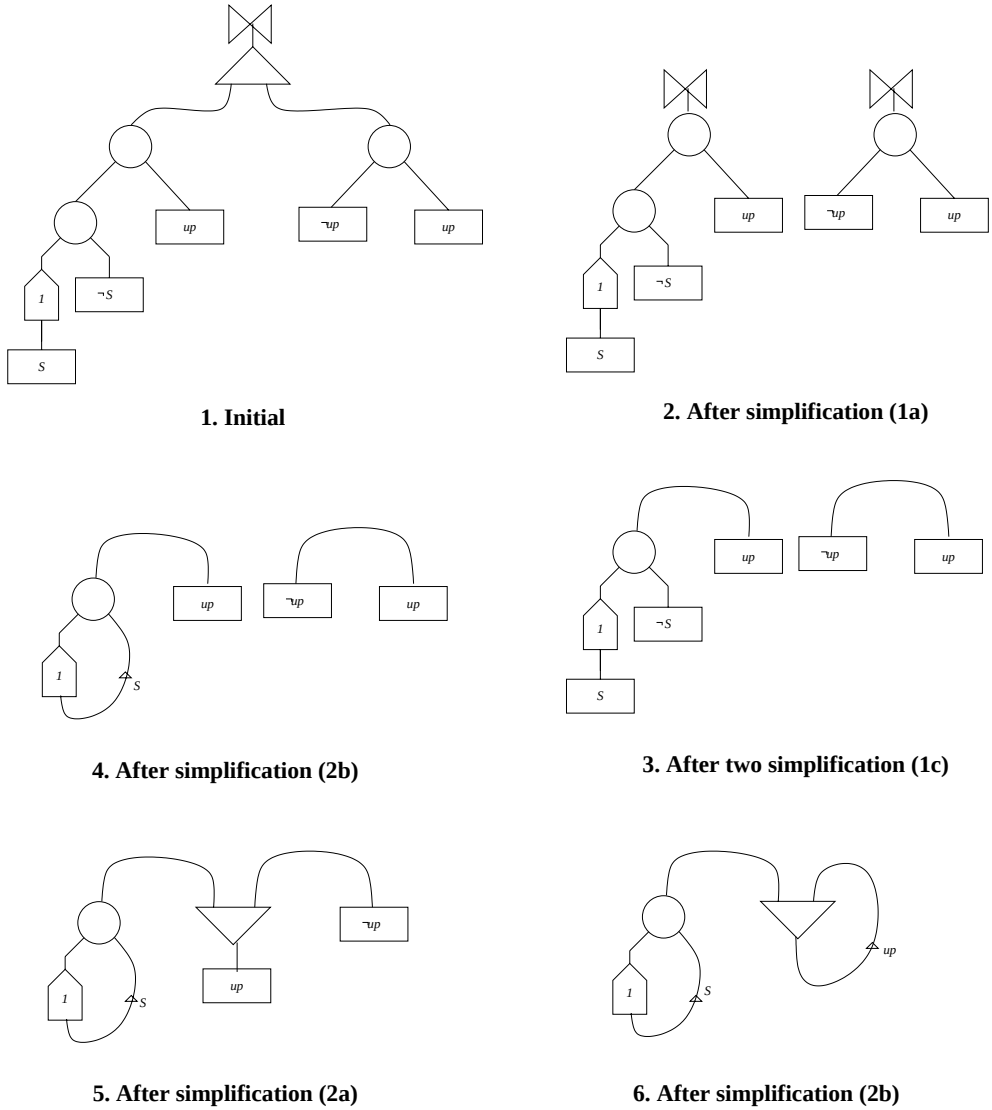

Figure 8: Simplifications of the up example

By assuming that the time of the events in $\omega()$ is not more than the minimum time of the events in Δ, meaning that all the events referencing previous time instants have already been processed, and by using appropriate data structures to store Φ and Δ, the asimpthotical complexity for a single inference step (the "repeat" body) can be evaluated as $O(D_{max} + \text{card } I)$, where D_{max} is the maximum of the absolute value of delays present in the network and card I is the size of $\omega()$, which is equal to the number of inputs. In this case, the maximum time needed to infer all the outputs for a single time instant is an:

$$O((D_{max} + \text{card } I)\ N_a)$$

6.2 Real-time execution

The execution algorithm has been developed even for on-line execution, when continous input signals are sampled and outputs are produced at a given rate. If the specification is strictly causal, meaning that current values of outputs and internal variables depend only on present and past samples of inputs and internal variables, then at each discrete time instant the outputs for the next time instant can be deduced by using the inference process as depicted in Figure 9. However, the time comprised between two consecutive input acquisitions or output productions may not be enough to deduce the information about outputs.

As showed in the previous section, if the inference function is causal at most N_a inference steps are needed to produce all the inferrable information for time t, where N_a is the number of network arcs. The maximum

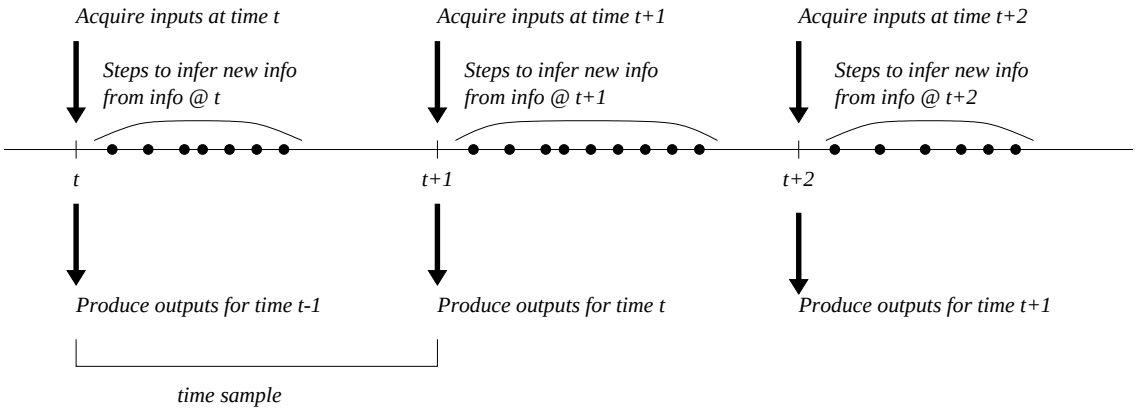

Figure 9: Real-time execution

time needed for one step of the algorithm can be evaluated depending on the underlaying hardware and the inference network structure (D_{max}), so that the maximum time required to produce the outputs for an instant can be found by multiplying this time duration for the amount of arcs of the inference network. If this time is less then the time sample then real-time execution can be accomplished:

$$T_s > T_{max}\ N_a$$

where T_s is the duration of the time sample and T_{max} is the maximum time needed to execute an inference step.

In order to have a more accurate evaluation of the time needed, T_{max} can be split in two times, $T_{input}+T_{infer}$, where T_{input} is the time needed to add inputs to set Δ and T_{infer} the one needed to infer information. By considering that, during execution, for each time sample there is only one step in which new inputs are acquired, then condition on the time sample becomes:

$$T_s > T_{input} + T_{infer}\ N_a$$

The condition presented do not change the asymphtotical complexity of the algorithm as evaluated in the previous section, but may be useful when the evaluation of condition $T_s > T_{max}\ N_a$ fails.

7 An Example

The selected example is a control system by which a fictitious but dangerously unstable reaction evolves within a control field under the supervision of a human operator. Therefore, the interacting components are:

- The reactor
- The human operator
- The control system, composed by:
  - The field control subsystem, charged to manage field monitoring, feeding and cooling
  - The reaction control subsystem, charged to manage reaction monitoring, feeding, ignition and extinction
  - The supervisor subsystem, charged with the operational condition evaluation and the communication with the operator

A scheme summing up the relationships amongst these components is shown in figure 10. The signals appearing hereinafter have following meaning:

- Reactor interface:
  - *sense_field*: sensors measure the field presence
  - *feed_field*: field generators are feeded
  - *sense_overheat*: field temperature is above danger level

- *cool_field*: field cooling systems are on
- *sense_reaction*: sensors measure the reaction presence
- *feed_reaction*: reagents are let in inside the reactor
- *ignite_reaction*: reaction ignition system is on
- *extinguish_reaction*: emergency extinction system is on

- Human operator interface:
  - *switch_on*: operator requests the process activation
  - *engine_halt*: process is stopped on the ground of precautionary measure
  - *alert*: situation requests the operator's attention
  - *acknowledge*: operator realized the situation and process can go on

- Interface amongst subsystems:
  - *engine_on*: supervisor subsystem requests process activation
  - *field_on*: field is steadily on
  - *field_fault*: field does not stabilize although it is feeded
  - *field_nominal*: field temperature is steadily within working limits
  - *field_overheat*: field temperature steadily exceeds danger level
  - *reaction_off*: reaction is steadily off
  - *reaction_fault*: reaction does not stabilize although it is feeded
  - *reaction_persistence*: reaction keeps on although it is not feeded
  - *warning*: there is a pre-alert condition

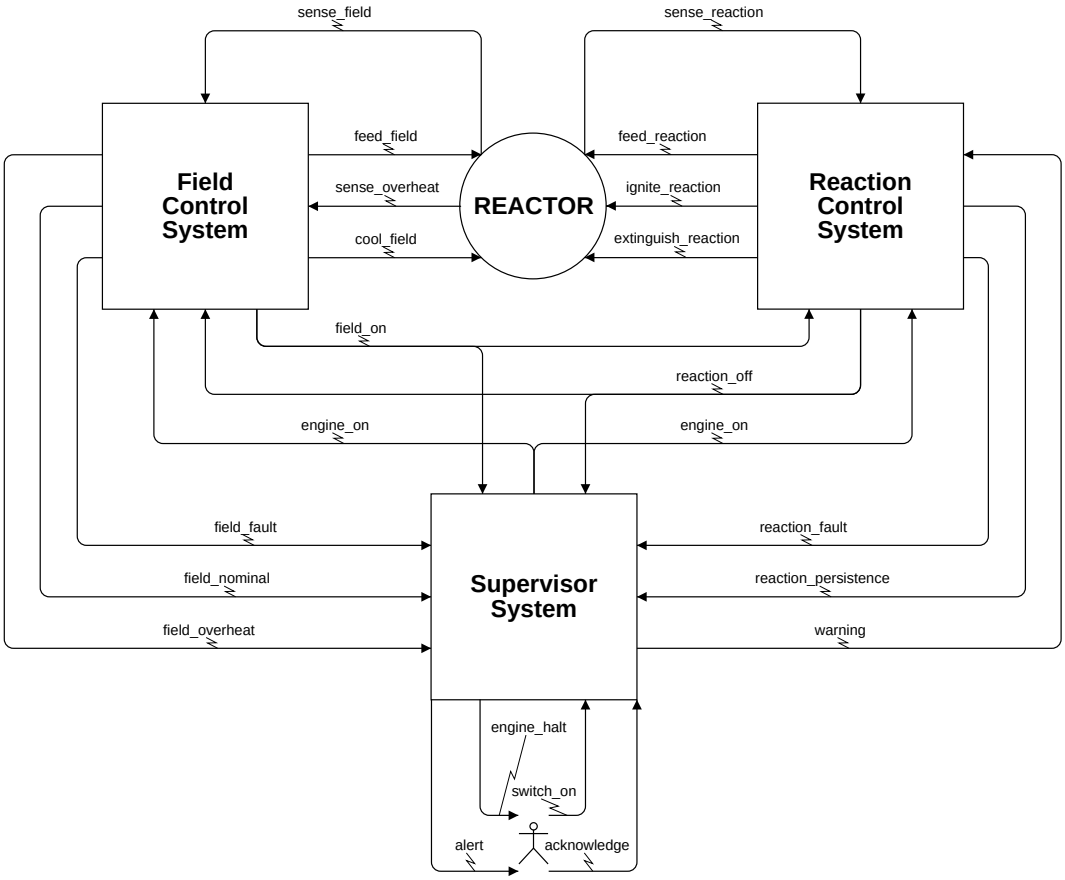

Figure 10: A concise outline of the control system

If, on the one hand, the behaviour of both reactor and operator is not modelled herein, on the other hand the control system is fully defined by the specification which follows:

- Field control subsystem:

$$
\begin{aligned}
\mathit{field\_on} &\iff \mathit{sense\_field} \,@\, [-3, 0] \\
\mathit{field\_nominal} &\iff (\neg \mathit{sense\_overheat}) \,@\, [-3, 0] \\
\mathit{field\_overheat} &\iff \mathit{sense\_overheat} \,@\, [-3, 0] \\
\mathit{field\_fault} &\iff \mathit{feed\_field} \,@\, [-12, -1] \wedge \neg \mathit{field\_on} \\
\mathit{feed\_field} &\iff \mathit{engine\_on} \vee \neg \mathit{reaction\_off} \\
\mathit{cool\_field} &\iff \mathit{since}((\partial \neg \mathit{field\_overheat} \wedge \mathit{field\_overheat}) \,?\, [-20, 0], \neg \mathit{field\_nominal})
\end{aligned}
$$

- Reaction control subsystem:

$$
\begin{aligned}
\mathit{reaction\_off} &\iff (\neg \mathit{sense\_reaction}) \,@\, [-2, 0] \\
\mathit{reaction\_on} &\iff \mathit{sense\_reaction} \,@\, [-2, 0] \\
\mathit{reaction\_fault} &\iff \mathit{feed\_reaction} \,@\, [-9, -1] \wedge \neg \mathit{reaction\_on} \\
\mathit{reaction\_persistence} &\iff (\neg \mathit{feed\_reaction}) \,@\, [-9, -1] \wedge \neg \mathit{reaction\_off} \\
\mathit{feed\_reaction} &\iff \mathit{engine\_on} \wedge \mathit{field\_on} \wedge \neg \mathit{warning} \\
\mathit{ignite\_reaction} &\iff \mathit{feed\_reaction} \wedge \mathbf{since}(\mathit{reaction\_off}, \neg \mathit{reaction\_on}) \\
\mathit{extinguish\_reaction} &\iff \mathbf{since}(\neg \mathit{engine\_on} \wedge \mathit{reaction\_persistence}, \neg \mathit{reaction\_off})
\end{aligned}
$$

- Supervisor subsystem:

$$
\begin{aligned}
\mathit{engine\_on} &\iff \mathit{switch\_on} \wedge (\partial \neg \mathit{switch\_on} \vee \mathit{switch\_on}) \,@\, [-10, -1] \wedge \neg \mathit{engine\_halt} \\
\mathit{engine\_halt} &\iff \mathbf{since}(\mathit{field\_fault} \vee \mathit{reaction\_fault} \vee \mathit{general\_fault}, \mathit{switch\_on}) \\
\mathit{warning} &\iff \neg \mathit{field\_nominal} \vee \mathit{reaction\_persistence} \vee \mathit{danger} \,?\, [-15, 0] \\
\mathit{danger} &\iff (\mathit{field\_overheat} \wedge \mathit{reaction\_persistence}) \vee \mathit{emergency} \\
\mathit{emergency} &\iff \neg \mathit{field\_on} \wedge \neg \mathit{reaction\_off} \\
\mathit{alert} &\iff \mathbf{since}(\partial \neg \mathit{danger} \wedge \mathit{danger}, \neg \mathit{acknowledge} \wedge \mathit{switch\_on}) \\
\mathit{general\_fault} &\iff (\mathit{danger} \wedge \mathit{alert}) \,@\, [-35, 0] \vee \mathit{emergency}
\end{aligned}
$$

When restricting oneself to the reaction control subsystem analysis, one can notice that the reaction is considered stable if it is measured as either absent or present for three consecutive instants. From this evaluation, nine instants are the greatest interval allowed to get a steadily present reaction in a constant feeding condition, afterwards an unsettled problem is notified to the supervisor system. The reaction is ignited only if it is feeded and for all the time needed to grant its transition from stable absence condition to stable presence one. On the other hand, possible further fluctuations which do not bring back to the starting condition do not cause interventions as to bound control activity. The feeding is granted only provided that the supervisor subsystem requests the process activation, the field is steadily on and there is no warning condition. The persistence of residual reaction traces after nine instants of interrupted feeding is considered anomalous and for that reason it is notified to the supervisor subsystem. If the latter does not request the process prosecution, the emergency shutdown system is activated and kept on until a stable reaction absence condition is reached.

A graphical representation of the network corresponding to the described specification appears in figure 11, while an execution example of the overall system is shown in figure 12.

By observing the signals' histories which appear in the example you can notice that the ignition happened in a regular way after the reactor activation request, but then an overheating condition occurred, which caused the reaction feeding shutdown and the cooling systems' activation. Since the reaction persists even in absence of feeding, the alert starts ringing and the operator is careful to turn it off by using the suitable signal. In fact, rather than do anything else, he prefers requesting the reactor shutdown, which happens by activating the emergency extinction system.

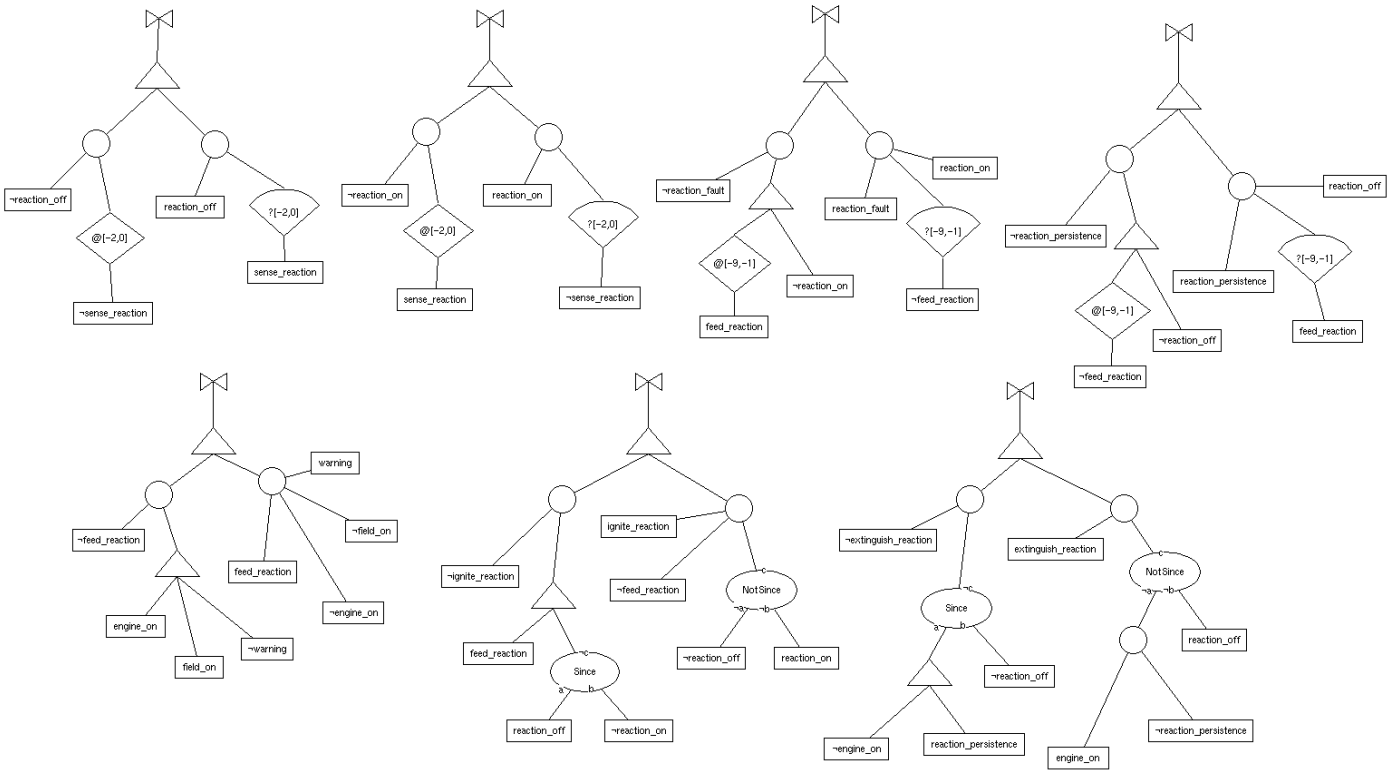

Figure 11: Network corresponding to the reaction control subsystem

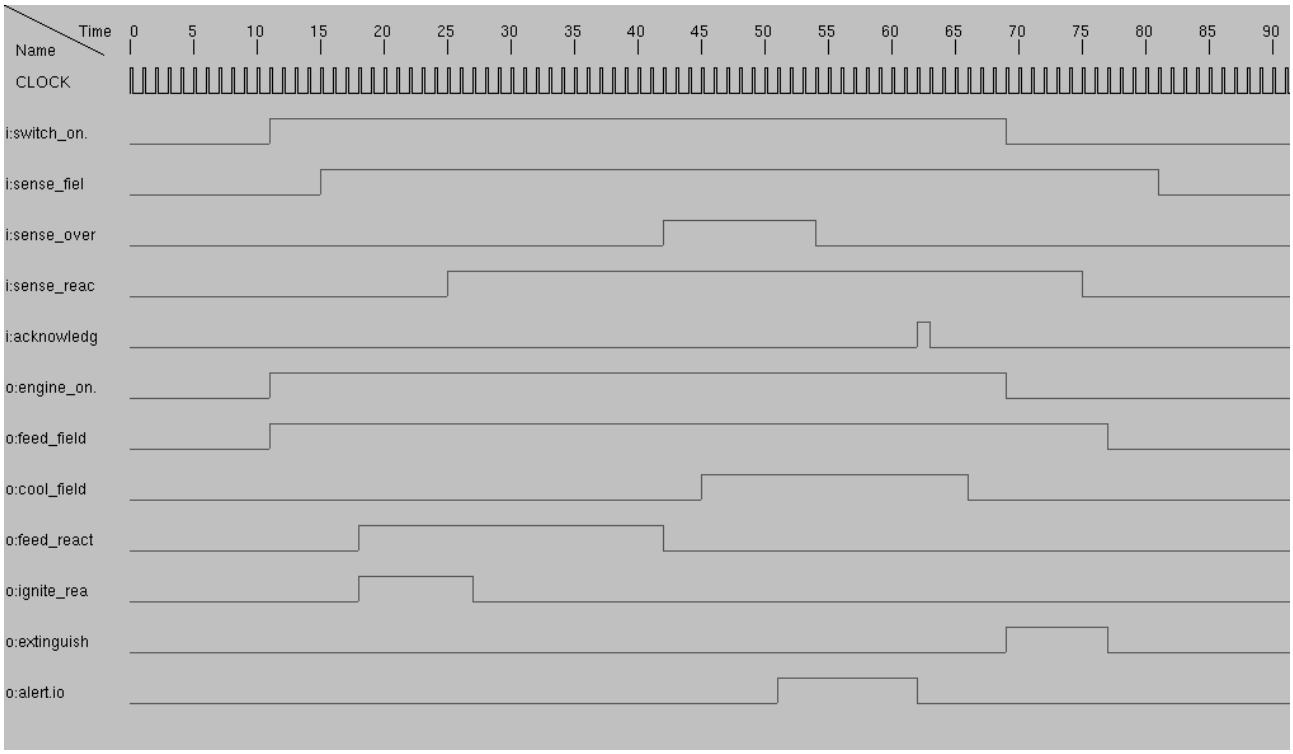

Figure 12: An execution example

8 Conclusions

This report has described the executable model for TILCO specifications, based on Basic Temporal Logic (BTL) and Temporal Inference Networks (TIN). TILCO specifications can be executed after having been transformed in BTL and represented by TIN networks. Each TIN network presents a set of nodes on which the inference rules are applied to produce the outputs on the basis of inputs, and thus to execute in real-time the specification. The TIN engine has been implemented in C++ and can be used under both Linux and Windows platforms. It permits also the integration of complex, time dependent behaviours in Visual C++ applications, where the use of traditional programming languages turns out to be difficult and very time consuming.

The work proposed in this article has been partially supported by MIUR (Italian Ministery of University and Research), with QUACK project in which several other universities are involved.

References

[1] G. Bucci, M. Campanai, and P. Nesi, "Tools for specifying real-time systems," *Journal of Real-Time Systems*, vol. 8, pp. 117–172, March 1995, YES.

[2] A. D. Stoyenko, "The evolution and state-of-the-art of real-time languages," *Journal of Systems and Software*, pp. 61–84, April 1992, YES.

[3] P. Bellini, R. Mattolini, and P. Nesi, "Temporal logics for real-time system specification," *ACM Computing Surveys*, vol. 31, no. 4, December 2000, YES.

[4] C. Ghezzi, D. Mandrioli, and A. Morzenti, "Trio, a logic language for executable specifications of real-time systems," *Journal of Systems and Software*, vol. 12, no. 2, pp. 107–123, May 1990, YES.

[5] R. Koymans, *Specifying Message Passing and Time-Critical Systems with Temporal Logic*, Number 651. Lecture Notes in Computer Science, Springer-Verlag, 1992.

[6] J. F. Allen and G. Ferguson, "Actions and events in interval temporal logic," Tech. Rep., University of Rochester Computer Science Department, TR-URCSD 521, Rochester, New York 14627, July 1994.

[7] R. Mattolini and P. Nesi. An interval logic for real-time system specification. *IEEE Trans. Soft. Eng.*, 27(3):208–227, March 2001.

[8] F. Jahanian and A. K.-L. Mok, "Safety analysis of timing properties in real-time systems," *IEEE Transactions on Software Engineering*, vol. 12, no. 9, pp. 890–904, Sept. 1986.

[9] R. Koymans, "Specifying real-time properties with metric temporal logic," *Real-Time Systems Journal*, vol. 2, pp. 255–299, 1990.

[10] R. Alur and T. A. Henzinger, "Real-time logics: Complexity and expressiveness," Tech. Rep., Dept. of Comp. Science and Medicine STAN-CS-90-1307, Stanford University, Stanford, California, USA, March 1990.

[11] P. Nesi P. Bellini, "Tilco-x: an extension of tilco temporal logic," in *Proc. of the 7th "IEEE International Conference on Engeneering of Complex Computer Systems",ICECCS'01.*

[12] P. Nesi P. Bellini, "Communicating tilco: a model for real-time system specification," in *Proc of the 7th "IEEE International Conference on Engeneering of Complex Computer Systems",ICECCS'01.*

[13] R. Mattolini and P. Nesi, "Using tilco for specifying real-time systems," in *Proc. of the 2nd IEEE International Conference on Engineering of Complex Computer Systems ICECCS'96*, Montreal, Canada, Oct. 1996, IEEE Press.

[14] M. Felder and A. Morzenti, "Validating real-time systems by history-checking trio specifications," *ACM Transactions on Software Engineering and Methodology*, vol. 3, no. 4, pp. 308–339, Oct. 1994.

[15] M. Fisher and R. Owens, "An introduction to executable modal and temporal logics," in *Executable Modal and Temporal Logics: Proceedings of the IJCAI '93 Workshop, Chamberry, France, August 1993*, M. Fisher and R. Owens, Eds. 1995, pp. 1–20, Lecture Notes in Artificial Intelligence, Springer Verlag LNCS 897.

[16] M. Ben-Ari, *Mathematical Logic for Computer Science*, Prentice Hall, New York, 1993.

Contents