DERIVING SAFETY MONITORS FROM FORMAL SPECIFICATIONS

J. Górski

Institute of Informatics, Technical University of Gdansk, 80–952 Gdansk, Poland

ABSTRACT. Safety is not guaranteed by the proper behaviour of the target system itself - it also depends on the properties of the plant and the environment. The paper shows how a safety monitor which supervises the assumptions under which the target system is safe, can be derived from formal specifications. The essential means are finite transition graphs which can be constructed from temporal formula by utilizing an appropriate normal form.

KEYWORDS. Safety; monitoring; temporal logic; graphs.

INTRODUCTION

One of the principles for design for safety (explicitly recognized by the EWICS TC7 guideline [EWICS'89]) recommends that the target system and the plant should be continuously monitored to intercept hazards before they convert into accidents. This way the "last chance" barriers guarding against the accidents in situations where they are likely to occur can be built into the system. In this paper we take the view that safety is the property of the whole application, which includes the target system, the plant and possibly a part of the surrounding environment. This means in particular that safety is not guaranteed by the proper behaviour of the target system itself - it also depends on the properties exhibited by the plant and the environment. In the previous papers [Gorski'87, Gorski'89] we have applied temporal logic to specify and analyse safety of an example traffic control system. In this paper we show how a safety monitor which "supervises" the assumptions under which the target system is (demonstrated to be) safe, can be formally derived from the specifications. In particular, we present algorithmic fundamentals of monitoring constraints expressed in temporal logic. The previous work in this area includes [MW'84; LS'87]. The essential means are finite transition graphs which can be constructed from temporal formula (the assertion being the subject of monitoring) by utilizing an appriopriate normal form. The graph forms the basis for constructing the safety monitor - an algorithm which analyses the subsequent states of the system. A present state can be accepted or rejected by the algorithm. Rejection means that no future continuation exists which can satisfy the assertion, i.e. the assertion has been "broken". In such case the algorithm signals SAFETY_ERROR which should initiate some extraordinary action (e.g. transition to the safe side state). The paper considers an example assertion taken from the railway crossing example [Gorski'89; Gorski'89a] which, expressed informally, says that: a train never crosses the red signal. It is easy to see that this assertion is the prerequisite for the safety argument related to railways. Moreover, this assertion is beyond the reach of the computer system which controls the crossing - the computer has no maeans to stop the train except setting the red signal. The solution is to monitor the assertion and to signal SAFETY_ERROR as soon as it fails.

TEMPORAL LOGIC

The temporal logic alphabet includes (a) constant symbols; (b) local and global variables; (c) local and global predicate symbols; (d) local and global function symbols; (e) logical constants 'True' and 'False'; (f) logical connectives: $\neg$ (negation), $\cup$ (disjunction), $\cap$ (conjunction), $==>$ (implication), $<==>$ (equivalence); (g) quantifiers: $\forall$ (universal) and $\exists$ (existential); (h) temporal operators: $\Box$ (always), $\Diamond$ (sometime), $\ominus$ (next time), $\mathcal{U}$ (until), $\mathcal{U}$ (unless), $\mathcal{P}$ (precede), $\mathcal{P}$ (before).

In the sequel we assume a fixed interpretation of constants, global predicate symbols and global function symbols over a fixed domain DOM. Let VAR denotes the set of global variables. Then, each mapping α:VAR $\longrightarrow$DOM represents valuation of global variables.

Let S denotes the set of states. Each state assignes meaning to the local symbols (variables, functions and predicates). Let Ω be the set of all finite and infinite sequences of states

and $\theta \subset \Omega$ be the set of infinite state sequences. For each infinite sequence $\sigma=s_0,s_1,..$
$\sigma[i]$ denotes the i-th element of σ; $\sigma(i,j)$, where $i \le j$, denotes the sequence $s_i,s_{i+1},..,s_j$; and $\sigma(i)$ denotes the sequence $s_i,s_{i+1},...$

The semantics of a temporal language is defined by the satisfaction (validity) relation '$\models$' which is defined below. The fact that formula F is valid for α in σ will be written '$[\sigma,\alpha]\models F$'. If F does not contain temporal operators then $[s,\alpha]\models F$ means that F is valid for α in state s.

(a) Atomic formulae (i.e. boolean term or an equation between terms). If F is an atomic formula then $[\sigma,\alpha]\models F$ iff F is true for α in the state $\sigma[0]$, i.e. $[\sigma[0],\alpha]\models F$.

(b) Logical connectives. If F, F' are formulae then so are $\neg F$, $F \cap F'$, etc. The semantics are as follows: $[\sigma,\alpha]\models\neg F$ iff not $[\sigma,\alpha]\models F$; $[\sigma,\alpha]\models F\cap F'$ iff $[\sigma,\alpha]\models F$ and $[\sigma,\alpha]\models F'$; etc.

(c) Quantification. If F is a formula, x is a global variable then $(\forall x)F$ and $(\exists x)F$ are formulae. The semantics are as follows: $[\sigma,\alpha]\models(\forall x)F$ iff for each $d \in DOM$, $[\sigma,\alpha\langle x:=d\rangle]\models F$, where $\alpha\langle x:=d\rangle$ denotes the mapping which yields d for x and for other argument equals α; $[\sigma,\alpha]\models(\exists x)F$ iff there exists $d \in DOM$ such that $[\sigma,\alpha\langle x:=d\rangle]\models F$.

(d) Temporal operators. If F,F' are formulae then so are $\Box F$, $\Diamond F$, $\ominus F$, $F\,\mathcal{U}\,F'$, $F\,\mathcal{W}\,F'$, $F\,\mathcal{P}\,F'$ and $F\,\mathcal{B}\,F'$. The semantics are as follows: $[\sigma,\alpha]\models\Box F$ iff for each $i \ge 0$, $[\sigma(i),\alpha]\models F$; $[\sigma,\alpha]\models\Diamond F$ iff there exists $i \ge 0$, such that $[\sigma(i),\alpha]\models F$; $[\sigma,\alpha]\models\ominus F$ iff $[\sigma(1),\alpha]\models F$; $[\sigma,\alpha]\models F\,\mathcal{U}\,F'$ iff there exists $i \ge 0$, such that $[\sigma(i),\alpha]\models F'$ and for each j, $0 \le j < i$, $[\sigma(j),\alpha]\models F$; $[\sigma,\alpha]\models F\,\mathcal{W}\,F'$ iff for each $i \ge 0$, $[\sigma[i],\alpha]\models F$ or there exists $i \ge 0$, such that $[\sigma(i),\alpha]\models F'$ and for each j, $0 \le j < i$, $[\sigma(j),\alpha]\models F$; $[\sigma,\alpha]\models F\,\mathcal{P}\,F'$ iff there exists $i \ge 0$, $[\sigma(i),\alpha]\models(F\cap\neg F')$ and for each j, $0 \le j < i$, $[\sigma(j),\alpha]\models\neg F'$; $[\sigma,\alpha]\models F\,\mathcal{B}\,F'$ iff for each $i \ge 0$, $[\sigma(i),\alpha]\models\neg F$ or there exists $i \ge 0$, $[\sigma(i),\alpha]\models(F\cap\neg F')$ and for each j, $0 \le j < i$, $[\sigma(j),\alpha]\models\neg F'$;

The following formulae are theorems of the temporal theory (i.e. are valid for any interpretation, any valuation α and any sequence of states σ.

$\Box P \Leftrightarrow P\ \mathcal{W}\ \mathrm{False}$. (1)
$\Diamond P \Leftrightarrow \mathrm{True}\ \mathcal{U}\ P$. (2)
$P1\ \mathcal{W}\ P2 \Leftrightarrow \Box P1 \cup (P1\ \mathcal{U}\ P2)$. (3)
$P1\ \mathcal{P}\ P2 \Leftrightarrow \neg P2\ \mathcal{U}\ (P1 \cap \neg P2)$. (4)
$P1\ \mathcal{B}\ P2 \Leftrightarrow \neg P2\ \mathcal{W}\ (P1 \cap \neg P2)$. (5)
$P1\ \mathcal{U}\ P2 \Leftrightarrow P2 \cup (P1 \cap \ominus(P1\ \mathcal{U}\ P2))$. (7)
$P1\ \mathcal{W}\ P2 \Leftrightarrow P2 \cup (P1 \cap \ominus(P1\ \mathcal{W}\ P2))$. (8)
$\ominus P1 \cap \ominus P2 \Leftrightarrow \ominus(P1 \cap P2)$. (9)

TRANSITION GRAPHS

Nontemporal formula is a formula built from atomic formulae (i.e. either a booelan term or an equation between terms) using logical connectives or quantification. Let BF be a set of nontemporal formulae which we assume as given (basic formulae). A propositional formula over BF is a nontemporal formula built from formulae in BF using logical connectives only. We denote the set of propositional formulae over BF by PL(BF).

A propositional temporal formula over BF is a temporal formula built from formulae in BF using logical connectives and temporal operators only. We denote the set of propositional formulae over BF by PTL(BF).

We say that the formula F' is in disjunctive normalform over BF, abbreviated $F \in DNF(BF)$, if

$$F = \bigcup_k (W_k \cap \ominus(P_k)), \qquad (10)$$

where
- each W_k is a conjunction of basic formulae or their negations,
- each P_k is a conjunction of propositional temporal formulae $P_{k,i}$ or $\neg P_{k,i}$ such that $P_{k,i}$ is either a basic formula or bound by a temporal operator.

Parts equivalent to true may be missing.

It can be proven ([LS'87]) that every formula $F \in PTL(BF)$ can be transformed into an equivalent formula $F' \in DNF(BF)$.

Let $F \in DNF(BF)$, i.e. F is of the form (10). Let σ be a sequence of states and α be a valuation for global variables in F. Then, the following conditions are equivalent:
$[\sigma,\alpha] \models F$;
$[\sigma(1),\alpha] \models \bigcup_k P_k$,
for those k where $[\sigma,\alpha]\models W_k$.

From the above it results that to check F in σ it is enough to check P_k on the tail sequence $\sigma(1)$, for those k for which W_k holds in the first state $\sigma[0]$ (note that W_k is a nontemporal formula).

Let F be a formula, α be a valuation for variables in F and δ be a finite sequence of states, i.e.
$\delta=s_0,s_1,..,s_n$.
We will say that F is possibly valid in δ iff there exists some continuation $s_{n+1},s_{n+2},...$ such that (infinite) sequence $\sigma=\delta^*(s_{n+1},s_{n+2},..)$ satisfies F. Intuitively, possible validity is the goal which should be maintained while monitoring safety. We can divide a system behaviour into two parts: a finite initial sequence of states which have already happened (and are known) and an infinite sequence of future (unknown) states. The safety monitor is a mechanism which deals with the known states only. the mechanism is represented by a transition graph $T=(G,\tau,\mu,M_0)$, where

- $G=(N,E)$ is a directed graph with nodes N and edges $E \subseteq N \times N$;
- $\tau: N \longrightarrow PTL(BF)$ is a node labelling function;
- $\mu: E \longrightarrow PL(BF)$ is an edge labelling function;
- $M_0 \in 2^N$ is an initial marking.

Behaviour of a transition graph for a given valuation α is a sequence of markings generated by a marking function M, by applying it repeatitivly to the initial marking M_0. For a given transition graph T, the marking $M(j,\delta)$, at step j for a state sequence $\delta \in \Omega$, is given by a (partial) function $M: NAT \times \Omega \longrightarrow 2^N$, which is given by the following inductive definition:

$M(0,\delta)=M_0$,
$M(i+1,\delta)=TR(M(i,\delta),\delta[i])$ if $i+1 \le$ length of δ, else undefined,

where TR: $2^N \times S \longrightarrow 2^N$ is defined as follows

$TR(MM,s)=\{Node' \in N \mid (\exists Node \in MM)(Node,Node') \in E \cap [s,\alpha] \models \mu(Node,Node')\}$.

The above definition formalizes the intuitive stepwise processing of the transition graph. A node Node' is marked at step i+1 iff there exists a node Node marked at step i and there exists an edge from Node to Node', such that its label holds in the state $\delta[i]$, for the valuation α. If none of the edge labels is valid in $\delta[i]$, the TR function yields an empty set (which can be interpreted as error).

For a given transition graph T, let $M(j,\delta)$ be the marking at step j for a state sequence δ. Then, actual formula, denoted $F_T(i,\delta)$, is defined as the disjunction of the labels of all nodes marked by $M(j,\delta)$.

We say that a transition graph T accepts a finite sequence δ of length i iff $M(i,\delta)$ is a nonempty set.

Let F be a propositional temporal formula. The set of basic formulae BF_F contains exactly all minimal nontemporal constituents of F that occur as operands of logical connectives outside quantifications. So each formula in BF_F is either an atomic formula of predicate logic or is bound by a quantifier $\forall$ or $\exists$. Obviously, F itself belongs to $PTL(BF_F)$. The transition graph T_F associated with F is constructed as follows.

Algorithm 1: TRANSITION GRAPH CONSTRUCTION

```
Assume that T_F is empty;
Assume that NEW set is empty;
Insert a node labelled F into T_F. The
   initial marking M_0 contains exactly
   this node. Insert this node to the NEW set
REPEAT
  FOR EACH Node∈NEW
     DO
      transform the node label τ(Node) into
       DNF(BF_F). Let us denote this
       transformed formula by f(Node);
      FOR EACH constituent (W∩eP) of f(Node)
        (fill up missing parts W or P with
        TRUE)
       DO
       IF no node in T_F is labelled with P
         THEN insert a node labelled
              P into T_F and insert it to
              the NEW set;
       create an edge from Node to the node
         labelled P and label this edge by W;
       IF several edges occur between two
         nodes
         THEN replace them by one edge
              labelled with the dis-
              junction of the given edges;
       OD;
       remove Node from NEW;
     OD;
UNTIL  NEW is an empty set.
```

It can be proven that Algorithm 1 terminates [LS'87].

MONITORING SAFETY ASSERTIONS

The basic result (proven in [Saake'85]) establishes that for every $i \in NAT$ and with respect to a fixed valuation α, F is valid in an infinite state sequence σ iff T_F accepts the prefix $\sigma(0,i-1)$ and the actual formula $F_T(i,\sigma)$ is valid in the tail sequence $\sigma(i)$. Saying it in another words, the necessary condition for validity of F is that T_F accepts $\sigma(0,i-1)$.

The safety monitor has to interpret the transition graph for every valuation of free global variables in F. Therefore it maintaines the set $M_F(\alpha)$ of marked nodes, for each valuation α. Initially, $M_F(\alpha)$ equals the initial marking of the transition graph T_F. The algorithm which is executed for every subsequent state s is given below:

Algorithm 2: GENERAL SAFETY MONITOR

```
FOR EACH safety condition F
 FOR EACH valuation α
 DO
  NEXT:={};
  FOR EACH node Node∈M_F(α)
   FOR EACH outgoing edge from Node to Node'
    IF [s,α] ⊨μ(Node,Node')
      THEN NEXT:=NEXT+{Node'};
  IF NEXT=∅ THEN SAFETY_ERROR
            ELSE M_F(α):=NEXT;
 OD.
```

SAFETY_ERROR means that the read-in sequence of states has not been accepted and consequently, there is no possibility that F is valid.

As we can observe, safety monitoring has been reduced to testing static conditions (i.e. the conditions which can be checked by looking at the present state only). Such tests are built into the control structure of GENERAL SAFETY MONITOR and the underlying transition graph. If a formula F contains free global variables then the number of tests is multiplied by the number of possible valuations. In some cases, it is not acceptable for practical purposes (note that value domains of the free variables in F can be infinite and in this case the GENERAL SAFETY MONITOR algorithm diverges).

Let us consider a sequence σ, and a formula F. Consider the transition graph as shown in Fig.1. Assume that the initial marking $M_0=\{Node1\}$ and that $\neg W(x) \Longrightarrow W'(x)$ holds. It is easy to note that for every valuation α and every $i \geq 0$, if for each j, $0 \leq j \leq i$, $[\sigma[j],\alpha] \models \neg W(x)$ then $\sigma(0,i)$ is accepted by T_F. Consequently, instead of monitoring every valuation, it is enough to consider only those valuations α for which for some j, $0 \leq j \leq i$, $[\sigma[j],\alpha] \models W(x)$. Then, such α should be monitored until, for some $k \geq j$, $M_F(k,\alpha)=M_0$ (i.e. we come back to the initial marking). The modified safety monitor, which exploits the above observation is given below.

Algorithm 3: SPECIFIC SAFETY MONITOR

```
FOR EACH safety condition F
 DO
  FOR EACH α such that [s,α] ⊨W(x)
    DO
     IF ¬(α∈ALPHA) THEN
        ALPHA:=ALPHA+(α);
        M_F(α):=M_0;
    OD;
  FOR EACH valuation α∈ALPHA
  DO
   NEXT:={};
   FOR EACH  Node∈M_F(α)
    FOR EACH outgoing edge from Node to Node'
     IF [s,α] ⊨μ(Node,Node')
        THEN NEXT:=NEXT+{Node'};
```

```
    IF NEXT=() THEN SAFETY_ERROR;
    IF NEXT=M0 THEN ALPHA:=ALPHA-(α)
        ELSE MF (α):=NEXT;
  OD;
 OD.
```

In the above algorithm, ALPHA is a set of valuations which are explicitly monitored. The algorithm is executed for each subsequent state s of σ. Before the algorithm is executed the first time, ALPHA is an empty set. It is assumed that ALPHA preserves its value between subsequent activations of the algorithm (is a sort of static variable).

A RAILWAY CROSSING SAFETY MONITOR

In this section we will apply the above theory to the example taken from [Gorski 89; Gorski 89a]. A computer system controls the railway crossing by receiving signals from sensors and changing the lights. Formal proof of safety of the system requires some additional assumptions which represent the guaranteed properties of the problem domain. One of those guarantees says that a train never passes a red signal. In terms of the model of system architecture this assumption can be expressed as follows

(∀n)(RED ∩ TCOUNT=n ==> TCOUNT=n ⊔ GREEN),
where RED is a local variable which denotes a current colour of the rail light signal, TCOUNT is a local variable which counts signals from the train position sensor (counts trains entering the crossing area) and GREEN=¬RED. The assertion says that if the light signal is red then TCOUNT will not change unless the signal changes to green.

In order to monitor this property we will apply Algorithm 1 to the following formula (11)
□(RED ∩ TCOUNT=n ==> TCOUNT=n ⊔ GREEN),
where n is a free global variable.

Step 1.
Let N denotes the set of nodes of the transition graph T_F. Initially N={}.

Step 2.
Let us introduce the following denotations:

P:RED ∩ TCOUNT=n ==> TCOUNT=n ⊔ GREEN, (12)
Q: RED ∩ TCOUNT=n, (13)
R: TCOUNT=n, (14)
W: GREEN. (15)

Then, P=(Q ==> R ⊔ W) and the formula (11) can be represented as
□P. (16)

By (1) we can convert (16) to
P ⊔ False. (17)

In this step we insert a node Node1 labelled
τ(Node1) = P ⊔ False
to the set N, i.e. N={Node1}.

Step 3.
We transform (17) into DNF. This transformation is shown by the sequence of formulae given below:

From (17), by (8) we obtain
False ∪ P∩ ⊖(P ⊔ False). (18)

And then, by (12-15) we convert (18) to
(¬Q ∪ (R ⊔ W))∩ ⊖(P ⊔ False), (19)
which can be further converted to
(¬Q∩ ⊖(P ⊔ False)) ∪ (20)
((R ⊔ W)∩ ⊖(P ⊔ False)),
and then, by virtue of (8), to
(¬Q∩ ⊖(P ⊔ False)) ∪ (21)
((W ∪ R∩⊖(R ⊔ W))∩ ⊖(P ⊔ False)),
and from this, by (9) we obtain
(¬Q∩ ⊖(P ⊔ False)) ∪ (W∩ ⊖(P ⊔ False)) ∪
(R∩⊖((R ⊔ W)∩(P ⊔ False))). (22)

The formula (22) is in DNF. From it we see that another node labelled
τ(Node2) = (R ⊔ W)∩(P ⊔ False),
is inserted to N, i.e. N={Node1,Node2} and the following edges are introduced:
from Node1 to Node1 labelled by ¬Q, (23)
from Node1 to Node1 labelled by W, (24)
from Node1 to Node2 labelled by R. (25)

Now, let us consider Node2 and convert its label to DNF. This is shown by the following sequence of formulae:

We start with
(R ⊔ W)∩(P ⊔ False).
Then, by (8) we obtain
(W ∪ R∩⊖(R ⊔ W))∩ (P∩⊖(P ⊔ False)), (26)
and then
(W∩P∩⊖(P ⊔ False)) ∪
(R∩P∩⊖((R ⊔ W)∩(P ⊔ False))). (27)

From this and (11)-(14) we have
(W∩(¬Q ∪ (R ⊔ W))∩⊖(P ⊔ False)) ∪ (28)
(R∩(¬Q ∪ (R ⊔ W))∩⊖((R ⊔ W)∩(P ⊔ False))),
and further conversions are shown below
(W∩¬Q∩⊖(P ⊔ False)) ∪ (29)
(W∩ (R ⊔ W)∩⊖(P ⊔ False)) ∪
(R∩¬Q∩⊖((R ⊔ W)∩(P ⊔ False))) ∪
(R∩(R ⊔ W)∩⊖((R ⊔ W)∩(P ⊔ False))),

(W∩¬Q∩⊖(P ⊔ False)) ∪ (30)
(W∩ (W ∪ R∩⊖(R ⊔ W))∩⊖(P ⊔ False)) ∪
(R∩¬Q∩⊖((R ⊔ W)∩(P ⊔ False))) ∪
(R∩(W ∪ R∩⊖(R ⊔ W))∩⊖((R ⊔ W)∩
(P ⊔ False))),

(W∩¬Q∩⊖(P ⊔ False)) ∪ (31)
(W∩ ⊖(P ⊔ False)) ∪
(W∩R∩⊖(R ⊔ W)∩⊖(P ⊔ False)) ∪
(R∩¬Q∩⊖((R ⊔ W)∩(P ⊔ False))) ∪
(R∩W∩⊖((R ⊔ W)∩(P ⊔ False))) ∪
(R∩(R∩⊖(R ⊔ W))∩⊖((R ⊔ W)∩
(P ⊔ False))),

(W∩¬Q∩⊖(P ⊔ False)) ∪ (32)
(W∩ ⊖(P ⊔ False)) ∪
(R∩¬Q∩⊖((R ⊔ W)∩(P ⊔ False))) ∪
(R∩W∩⊖((R ⊔ W)∩(P ⊔ False))) ∪
(R∩⊖((R ⊔ W)∩(P ⊔ False))).

From (32) we have that the following edges are introduced:
from Node2 to Node1 labelled by W∩¬Q, (33)
from Node2 to Node1 labelled by W, (34)
from Node2 to Node2 labelled by R∩¬Q, (35)
from Node2 to Node2 labelled by R∩W, (36)
from Node2 to Node2 labelled by R. (37)

The edges (23), (24) are replaced by
μ(Node1,Node1)=¬Q ∪ W.
After substitution of (13) and (15) and some conversions we obtain
μ(Node1,Node1) = (GREEN ∪ TCOUNT≠n).

After substitution of (14) to (25) we obtain
μ(Node1,Node2) = (TCOUNT=n).

The edges (33), (34) are replaced by

$W \cap \neg Q \cup W$, which is equivalent to W. After substitution of (15) we obtain
μ(Node2,Node1) = GREEN.

The edges (35),(36),(37) are replaced by $R \cap \neg Q \cup R \cap W \cup R$, which is equivalent to R. After substitution of (14) we obtain
μ(Node2,Node2) = (TCOUNT=n).

The resulting transition graph is shown in Fig.2. The corresponding safety monitor algorithm (derived from Algorithm 3) is given below.

Algorithm 4: RAILWAY CROSSING MONITOR

```
TYPE
  NODE=(Node1,Node2);
  VALUATION = RECORD
                VALUE:INTEGER;
                MARKING: SET OF NODE;
              END;
VAR
ALPHA: SET OF VALUATION; := {};

PROCEDURE   SAFETY_MONITOR(TCOUNT:INTEGER;
                           RED:BOOLEAN);
 VAR
  NEXT: SET OF NODE;
  NEW: BOOLEAN;

 BEGIN
(L1)  NEW := TRUE;
      FOR EACH a∈ALPHA
       NEW:=(NEW AND a.VALUE≠TCOUNT));
      IF NEW THEN
       ALPHA := ALPHA + (TCOUNT,{Node1});

(L2)  FOR EACH a∈ALPHA
       BEGIN
(L3)   NEXT := {};

(L4)   IF Node1∈a.MARKING THEN
         BEGIN
         IF NOT RED AND  TCOUNT≠a.VALUE
           THEN NEXT := NEXT + {Node1};
         IF TCOUNT=a.VALUE
           THEN NEXT := NEXT + {Node2};
         END;

(L5)   IF Node2∈a.MARKING THEN
         BEGIN
         IF TCOUNT=a.VALUE
           THEN NEXT := NEXT + {Node2};
         IF NOT RED
           THEN NEXT := NEXT + {Node1};
         END;

(L6)   IF NEXT = {}  THEN SAFETY_ERROR;
(L7)   IF NEXT = {Node1}
          THEN ALPHA := ALPHA - {a}
          ELSE a.MARKING := NEXT;
       END;
 END {SAFETY_MONITOR}.
```

To show that the above algorithm is implementable we have to assure that the cardinality of ALPHA does not increase infinitely. We will show that the cardinality of ALPHA is less or equal 2. Let alpha and tcount denote the values assumed by ALPHA and TCOUNT at the call and let $NEXT_a$ denotes the NEXT set computed for a given a, inside the loop L2. First, let us observe that for each a∈alpha, a.VALUE≤tcount. It becomes clear if we note that TCOUNT counts trains entering the crossing, i.e. in subsequent calls its value can not decrease, and initially ALPHA is empty.

Let us consider a given call. Then, at the first entry to the loop (L2), we have one of the following situations:
(a) ALPHA=alpha and there exists a∈ALPHA such that a.VALUE=tcount (in which case, at point (L6) we will have Node2∈$NEXT_a$) and for all b∈ALPHA and b≠a, b.VALUE≠tcount (and consequently, at point (L6) we will have $NEXT_b$={Node1} or $NEXT_b$={}, depending on if RED is False or True);
(b) ALPHA=alpha+{(tcount,{Node1})} (in which case, at point (L6) we will have $NEXT_a$={Node2}) and for all b∈alpha, b.VALUE≠tcount (which implies that at point (L6) we will have $NEXT_b$={Node1} or $NEXT_b$={}, depending if RED is False or True).

It is easy to see that in lines (L6), (L7) the only valuation a which is not removed from ALPHA is that for which a.VALUE=tcount. For all others, either they are removed in instruction (L7) or the SAFETY_ERROR is being signalled in instruction (L6). Consequently on normal (i.e. without SAFETY_ERROR) return ALPHA contains exactly one element. And because the only instruction which can increase ALPHA is (L1), we can conclude that ALPHA never contains more then 2 elements.

CONCLUSION

The paper presented a theoretical background for monitoring safety assertions. For expressing and analyzing safety we have utilized temporal logic. It has been shown that monitoring safety assertions can essentially be reduced to local tests in states. The tests can be determined systematicly and are represented by a transition graph. The monitoring procedure follows paths through the graph by maintaining sets of marked nodes. It has been shown how the monitor performance can be improved for some classes of assertions. The applicability of the theory has been demonstrated by building a safety monitor for an example assertion taken from the railway signalling world.

REFERENCES

[EWICS'89] Guidelines for design computer systems for safety. In: Dependability of critical computer systems 2, F.J. Redmill (Ed.), Elsevier Applied Science, (to be published in 1989).

[FB'86] M. Fisher and H. Barringer: Program logics - a short survey. Techn. Rep. UMCS-86-11-1, University of Manchester, 1986.

[Gorski'87] J. Gorski: Formal support for development of safety related systems. Proc. SARSS'87, B.K. Daniels (Ed.), Elsevier Applied Science, 1987, pp. 14-28.

[Gorski'89] J. Gorski: Formal approach to development of critical computer applications. Proc. HICSS-22, IEEE Computer Society Press, 1989.

[Gorski'89a] J. Gorski: Problems of specification and analysis of safety critical computer systems - application of temporal logic. Techn. Rep. (in Polish), Institute of Informatics, Techn. University of Gdansk, 1989.

[LS'87] U. W. Lipeck and G. Saake: Monitoring dynamic integrity constraints

based on temporal logic. Inform. Systems, vol. 12, no. 3, 1987, pp. 255–269.

[MW'84] Z. Manna and P. Wolper: Synthesis of communicating processes from temporal logic specifications. ACM TOPLAS, vol. 6, no. 1, 1984, pp. 68–93.

[Saake'85] G. Saake: Construction of transition graphs from temporal formulae for integrity monitoring in databases. Diploma Thesis, Techn. University of Braunschweig, 1985.

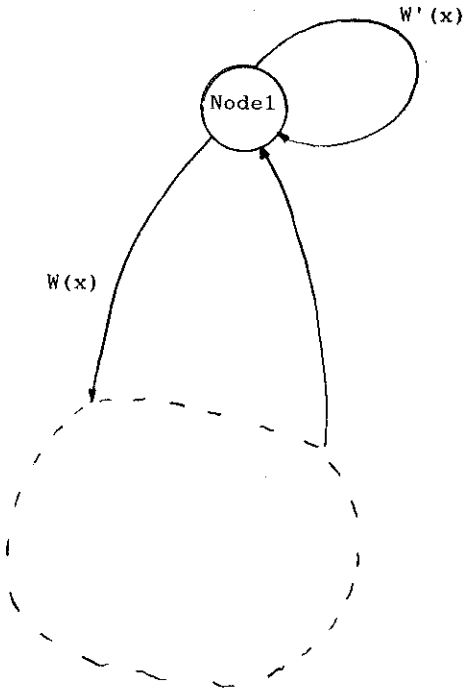

Fig. 1.

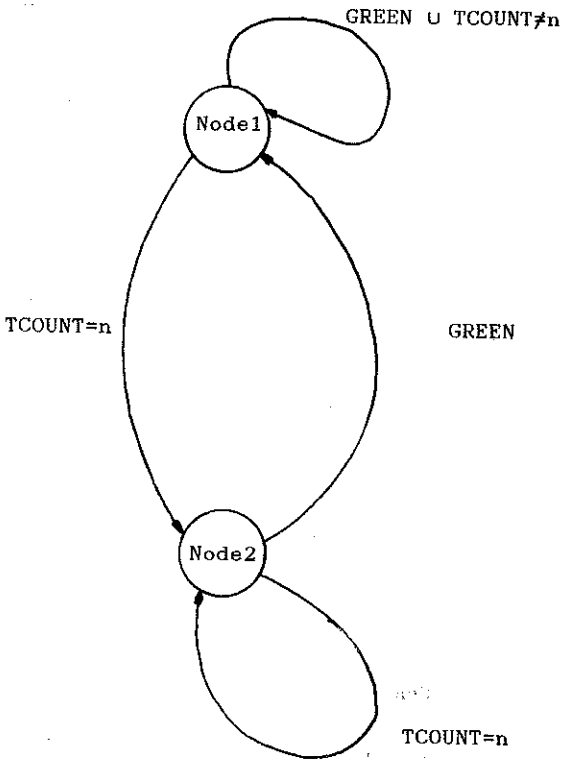

Fig. 2.