Formal verification for STATEMATE models

Yousra HLAOUI Ben Daly
yousra.BenDalyHlaoui@esstt.rnu.tn

Research Unit of Technologies of Information and Communication
Ecole Superieure des Sciences et Techniques de Tunis
5, Avenue Taha Hussein, BP.56, Bab Menara, 1008 Tunis
Tunisia.
Fax: 00 216 71 39 11 66

Leila JEMNI BEN AYED
leila.jemni@fsegt.rnu.tn

Research Unit of Technologies of Information and Communication
Ecole Superieure des Sciences et Techniques de Tunis
5, Avenue Taha Hussein, BP.56, Bab Menara, 1008 Tunis
Tunisia.
Fax: 00 216 71 39 11 66

Abstract

We propose an approach based on automatic derivation schemes from STATEMATE model to an FNLOG specification. STATEMATE is a semi formal method that pertains to the specification and design of complex reactive systems and builds simulations and prototypes rapidly. Though STATEMATE provides rigorous specifications, these are not verifiable to ensure and guarantee the reliability of the system being developed. To fulfill this objective, a STATEMATE specification is translated into a logic-based specification language called FNLOG which allows its verification. This paper describes the translation approach, cross references between STATEMATE and FNLOG features, and the translation algorithm.

Key Words: STATEMATE, FNLOG, formal method, semi formal method, combined method, translation, verification, specification.

1 Introduction

Several approaches were proposed to provide different methods and specification languages for the development of complex reactive systems. These systems are characterized by a complexity linked with their diverse ways to react to external events. Most of these methods such as STATEMATE [9], Statecharts [9] and UML [15] use state-transition diagrams to describe complex system's behavior. They provide rigorous specifications which are not verifiable to ensure and guarantee the reliability of the system being developed. As these methods are semi formal, they are not provided with proof systems unlike formal methods. It is for the reason, semi formal and formal methods have been combined to ensure the verification of semi formal specification ([17],[18],[19],[12],[14],[13],[?]). The combination consists of transforming semi formal specification to a formal one which is verifiable by the use of the proof system; hence, enabling the verification of a semi formal graphical and comprehensive specifications. Several works were elaborated in this context such as the transformation from UML to B [13], Statecharts to FuNction LOGic based specification language called FNLOG ([17],[18],[19]) which extends Statecharts language and provides a compositional proof system for Statecharts. Additional work has been elaborated to integrate STATEMATE and FNLOG methods in a same specification approach ([12],[14]). It is within the framework which formalize the informal that our contribution is presented. This paper attempts to develop a new approach to automatically transform STATEMATE designs to FNLOG specification in order to ensure its formal verification. We explore, in fact, the usefulness of the integration of semi formal and formal methods in specifying complex reactive systems and verifying their expected properties; as well as the compatibility of STATEMATE and FNLOG languages. FNLOG [19] which is a logical specification language, was introduced for the verification of Statecharts specifying reactive system behavior and also to cover the system functional aspects and to express properties in application domain and not just in Statecharts. To ensure the verification of Statecharts by FNLOG, a semantic bridge ([17],[18],[19]) is built between their components. The STATEMATE notation allows the specification of functional and behavioral aspects with activity-Chart and Statecharts languages. Thus, the compatibility between Statecharts and FNLOG components is used and extended to STATEMATE features specifying the system functions in the translation schemes between STATEMATE and FNLOG. The proposed approach consists firstly on specifying the system with STATEMATE notation which provides clear and understandable specification and secondly on translating this specification to FNLOG notation to be verified. This translation is automatic and based on derivation schemes from STATEMATE to FNLOG specification. In the derivation schemes, all functional and behavioral STATEMATE aspects are considered such as basic and composed activities and states, events, actions and conditions with their different forms, elementary, composed and textual. We introduce also, FNLOG formulae to represent static system reactions in a given state. In the works presented in ([17],[18],[19],[12],[14]), a large range of STATEMATE concepts were considered with the exception of connectors, different type of events, conditions, actions in Statecharts and different kind of activities in activity-Charts. These are considered in our proposed approach. In addition the proposed translation schemes support all kind of textual features, implicit events, time in timeout event, scheduled action, predefined functions and particulary decomposition of activities and states. Unlike works presented in ([17],[18],[19],[12],[14]), our approach stresses on the automatization to translate a STATEM-

ATE specification into FNLOG one. To reach this goal, we developed an algorithm translating a STATEMATE specification into FNLOG one. Thus, for each kind of STATEMATE specification language component, we define transformation processes which are based on our proposed derivation schemes. To use the appropriate transformation process in the algorithm, we define a set of predicates which determine the nature of the current element to transform. To do so, we were brought about formalizing all Statecharts and activity-Charts components. In addition, the decomposition of states and activities in Statecharts and activity-Chart languages is treated by the recursive nature of the algorithm. In this paper, we present our approach translation steps and the algorithm translating STATEMATE model to FNLOG specification.

2 STATEMATE overview

STATEMATE [9] is a graphic specification method for complex real-time reactive systems. In STATEMATE, the descriptions used to capture the system specification are organized into three views, of the system: functional, behavioral and structural. The functional view describes the system's functions, processes and activities with activity-Charts. The behavioral view describes the system's behavior over time with Statecharts. The structural one describes the subsystems, modules and the communication between them with module-Charts. Activity-charts [9] describe the system's functions, processes, or objects, also called activities. Activities may be basic or composed. A compound activity may have a control-activity as a sub-activity which is refined by a Statechart. This Statechart describes control activity parent and sisters behaviors. A control activity is considered at the same time an activity and a state. Statecharts ([8],[9]) describe the system's behavior over time, including the dynamics of activities, their control and timing behavior. A Statechart specification can be viewed as a tree of states. All other states are related by the superstate-substate property. The superstate at the top level is the specification itself. This relation imposes a natural concept of "depth" as a refinement of states. There are three types of states : AND, OR, and BASIC states. Transitions between states are specified by edges. Transitions may be labelled with a label having the form: ***Event-part [condition-part] / Action-part***.

3 FNLOG overview

FNLOG is a logic-based functional specification language, based on first-order predicate logic with arithmetic, extended by quantified temporal operators ([17],[18], [19]). An FNLOG specification is built from events and activities occurring over time, connected by logical and temporal operators [19]. Quantification over time is allowed to simplify properties description. The operators are of two types: the logical operators $(\vee, \wedge, \neg)$ and temporal operators. In FNLOG, the past-time temporal logic operators, which some of them will be described below, are used to capture relative and absolute time properties as well as the causal relationships over time.

$\odot_t$: true at time.
$\ominus_t$: true at the instant previous to t, ie at t-1
$\diamond_t$: true at some instant before t.
$\odot_{t-k}$: true k instants before time t.
$\diamond_t^{t-k}$: true at some instant in the interval [t-k,t]
$\diamond_t^{t-k}$ $\exists$ i, t-k$\leq i \leq t : \odot_t$

An event is an instantaneous occurrence of a signal, whereas an activity is a durative happening with an initial and an end instants and a finite duration between them. Conventionally, an event or an activity is called a function. Two special events are associated with every activity A: initiate-A (init-A) and terminate-A (term-A).

3.1 FNLOG specification

To specify any system using FNLOG, all the activities which form part of the system are identified. The top level specification is defined as an OR or AND of these activities. This specification is **L** which denotes the global logic-based functional specification of a system. Then: $L ::= \vee_A \odot_t (A) | \wedge_A \odot_t (A)$; Where the A's are defined by temporal formulae called *tformula*. The syntax, the semantic and temporal structure of FNLOG are presented in [19].

4 STATEMATE-FNLOG translation approach

Our approach Combining STATEMATE and FNLOG [10] consists of four steps and it is based on a systematic translation from a STATEMATE specification into an FNLOG one. The approach steps are presented by follows:

- **Step 1**: The system is specified with the activity-Charts and Statecharts STATEMATE languages and the properties expected to be checked by the system are described with FNLOG temporal formulae.

- **Step 2**: The obtained system STATEMATE specification is translated into FNLOG specification using cross referencing schemes (See section 3.1). Our main contribution concerns this step which is detailed in Fig.1.

- **Step 3**: The properties are checked from the obtained global system specification using FNLOG proof system[19].

- **Step 4**: Once the FNLOG system specification is validated, the structural view is provided using STATEMATE module-Chart language.

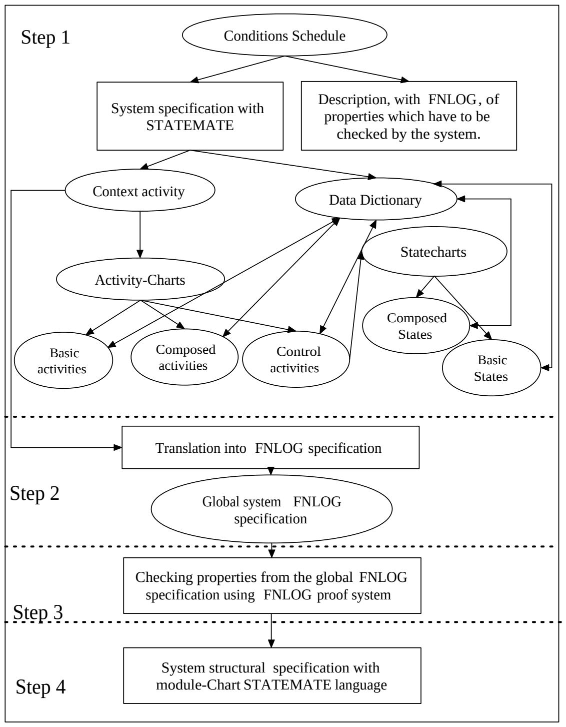

Step 1
Conditions Schedule
System specification with STATEMATE
Description, with FNLOG, of properties which have to be checked by the system.
Context activity
Data Dictionary
Statecharts
Activity-Charts
Composed States
Basic activities
Composed activities
Control activities
Basic States
Translation into FNLOG specification
Step 2
Global system FNLOG specification
Checking properties from the global FNLOG specification using FNLOG proof system
Step 3
System structural specification with module-Chart STATEMATE language
Step 4

Figure 1: The translation Steps

As the translation is systematic, we define for Statechart and activity-Charts special translation processes into FNLOG specification. Starting with the top level of activity-Charts, each activity of each level is translated into FNLOG formulae and then refined in a top down method using other simpler activities which themselves are translated into FNLOG specification. Once the bottom level of the activity-Charts is reached, the FNLOG formulae of this activities level will substitute their FNLOG specification, denoted by **L**, appearing in their parents FNLOG formulae. The substitution or recomposition process will continue until achieving the top level of the Activity charts (See Fig.1).

5 Translation algorithm

In this section, we present the translation algorithm which has as input the context diagram. This diagram is refined by activity-Charts belonging to the different levels of abstraction of the STATEMATE specification. To get a global FNLOG specification of the system, we have to transform all of these activity-Charts as well as Statecharts modelling behaviors of activity-Charts activities using two procedures : *translate-Activity* and *translate-Statechart*. These procedures are based on translation patterns which match for a given activity or state, the correspondent FNLOG specification. To apply the right translation pattern, we test the kind of activity or state by using predicates belonging to the predicates set presented in details in [10]. We define some of them in the e following:

$$\textit{composed}(\text{Ac}) = \left| \begin{array}{l} \textbf{True} \text{ if } \Omega_{Ac} \neq \emptyset \\ \textbf{false} \text{ if } \Omega_{Ac} = \emptyset \end{array} \right.$$

Tests if an activity Ac is composed or not. Ω_{Ac} is the set of sub-activities Ac.

$$\textit{have-actrl}(\text{Ac}) = \left| \begin{array}{l} \text{True if Ac} \\ \text{has a control activity} \\ \text{False else} \end{array} \right.$$

Tests if an activity Ac has a control activity as sub-activity.

$$\textit{or-state}(\text{S}) = \left| \begin{array}{l} \text{True if } \Sigma_S \neq \emptyset \\ \text{and S is the disjunction of} \\ \text{states} \in \Sigma_S \\ \text{False else} \end{array} \right.$$

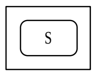
Tests if the state S is an Or-State. Σ_S is the set of sub-states of S.

5.1 STATEMATE and FNLOG features cross references

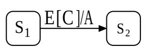
STATEMATE and FNLOG provide some similarities. They both use events and activities as primitives. In this section we introduce mapping relations from STATEMATE models to FNLOG formulae. We propose to match each concept and element of the STATEMATE language with an other of the FNLOG language. Thus, as a durative happening, a STATEMATE activity, state or action are represented with an FNLOG activity. A STATEMATE event is an instantaneous occurring over time, is represented by an FNLOG event.

If an activity A is alive at an instant t, it is represented by $\odot_t$(A) which is an FNLOG formula. If, in STATEMATE, an activity A is composed with sub-activities A1 and A2 then, in FNLOG, this corresponds to the following formula: $\odot_t(A)=\odot_t(A_1)\wedge\odot_t(A_2)$.

For each composed or basic STATEMATE state corresponds a composed or a basic activity in FNLOG.

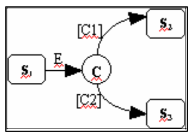
STATEMATE	**FNLOG**
	$\odot_n(\texttt{S}) = \odot_{n_1}(\texttt{init-S}) \wedge \odot_{n_2}(\texttt{term-S}) \wedge (\texttt{n}_1 < \texttt{n} < n_2)$

A transition is labelled with event[condition]/action. The event corresponds to an event in FNLOG language or to an event expression when it is composed. Whereas, the condition is represented by an FNLOG logic expression. The action is an activity in FNLOG. If at an instant t a transition is triggered, that means, at the same instant its relative event E_i occurs, its conditions Ci holds and the action Ai performs.

STATEMATE	**FNLOG**
	$\odot_n(\texttt{T}_i) = \odot_n(\texttt{E}_i) \wedge \odot_n(\texttt{C}_i) \wedge \odot_n(\texttt{A}_i)$

The other forms of events and cross references between different kinds of actions and conditions and FNLOG formulae are presented in details in [10].

A composed transition is labelled with the used connector which relates different components of a transition. We distinguish c-connector, s-connector, Junction, termination connector, H-connector). the use of such of such connectors allows the reduction of transitions number in a STATEMATE model. We give, in this paper, only correspondent FNLOG formulae for C-connector. Consider CT as a composed transition of the form CT = E[C1] or E[C2]. With E[C1] and E[C2] are transition labels. The correspondent FNLOG formula is given as follow :

STATEMATE	**FNLOG**
	$\odot_n(\texttt{CT}) = (\odot_n(\texttt{E}) \wedge \odot_n(\texttt{C}_1)) \vee (\odot_n(\texttt{E}) \wedge \odot_n(\texttt{C}_2))$

5.2 Proposed algorithm

We give, in the following, the main algorithm, detailing the translate-activity procedure and sketching only the main steps of Translate-Statechart procedure. We define the integer `num` as variable containing the number of the last definition in FNLOG specification formulae. All the unmentioned procedures are detailed in [10].

```
Algorithm translation STATEMATE-FNLOG
begin
    translate-activity(main activity, num)
End.

Procedure translate-activity(Ac:activity, VAR num:integer)
Begin
    $L_{Ac}=\odot_n(Ac)$ $(\forall i \geq 0)$ /*Ac FNLOG specification*/
    If have-comp-sup(Ac) Then
      If react-mini-spec(Ac)$\neq 0$ Then
        Translate-mini-spec-reactive(Ac,react-mini-spec(Ac))
      Else
        Translate-mini-spec-proc-like(Ac,proc-like-mini-spec(Ac))
      EndIf
    Endif
    If composed(Ac) Then
      $\Omega_{Ac} \leftarrow$ sub-activity(Ac)
      If have-actrl(Ac) Then
        $L_{Ac} \leftarrow L_{Ac} \wedge L_{Ac\_CTRL}$
        Add to the specification $L_{Ac}$ the following definitions
        /*the definition numbering starts with num+1 */
        num+1.$\odot_n(Ac) = \odot_n(Ac\_CTRL)$
        num+2. $\odot_n(init - Ac) = \odot_n(init - Ac\_CTRL)$
        num+3.$\odot_n(term - Ac\_CTRL) = \odot_n(term - Ac)$
        num+4.$\odot_n(term - Ac) = \bigwedge_i \odot_n(term - Ac_i)$
        /*$Ac_i \in \Omega_{Ac}$: Ac sub-activities set.*/
        num $\leftarrow$num+4.
        Translate- Statechart(Ac_CTRL, num, $L_{Ac\_CTRL}$)
        $\Omega_{Ac} \leftarrow \Omega_{Ac}/Ac\_CTRL$
      Else
        $L_{Ac} \leftarrow L_{Ac} \wedge_i L_{Ac_i}$
        The definition set will be:
        num+1.$\odot_n(Ac) = \wedge_{i=1,p} \odot_n (Ac_i)$ ,$\forall Ac_i \in \Omega_{Ac}$
        num+2.$\odot_n(Ac) \rightarrow \odot_{n_1}(init - Ac) \wedge \odot_{n_2}(term - Ac) \wedge$
        $(n_1 < n < n_2)$
        num+3.$\odot_n(init - Ac) \rightarrow \wedge_{i=1,p} \odot_n (init - Ac_i)$
        num+4.$\odot_n(term - Ac) \rightarrow \wedge_{i=1,p} \odot_n (term - Ac_i)$
        num$\leftarrow$ num+4
      EndIf
      While $\Omega_{Ac} \neq \emptyset$ Do
          Ac-curent$\leftarrow Ac_i$
          $\Omega_{Ac} \leftarrow \Omega_{Ac}/Ac_i$
          translate-activity($Ac_i$, num)
      EndWhile
      Substitute the $L_{Ac_i}$, in $L_{AC}$, expression with their
      generated formulae
    EndIf
End.
```

In the following, **S** denotes the current state,$\forall i \geq 0$, $\mathbf{T}_{1i} \in$ I denote in transitions set of **S** state and $\mathbf{T}_{2i} \in$ O out transitions set.

Procedure *Translate-Statechart*

Step 1 Provide the FNLOG specification corresponding to the considered state as if it is basic using the following pattern:

```
$L_S = \odot_n(S)$ With:
1. $\odot_n(S)=\odot_{n_1}(init\text{-}S) \wedge \odot_{n_2}(term\text{-}S) \wedge (n_1 < n < n_2)$
2. $\odot_n(init\text{-}S) = \bigvee_{T_{1i} \in I} \odot_n(T_{1i})$
3. $\odot_n(term\text{-}S) = \bigwedge_{T_{2i} \in O} \odot_n(T_{2i})$
4. $\odot_n(T_j) = \odot_n(E_j) \wedge \odot_n(C_j) \wedge \odot_n(A_j)$
with $T_j \in O \cup I$ and $E_j/C_j/A_j$ transition label.
```

Go to the step 2.

Step 2 Test if the current state is composed using *composed* predicate. In the affirmation go to step 4. In the negation, test if it exists other states non transformed belonging to the same level of abstraction of the Statecharts description of the current state. In the affirmation, choose one the states which becomes a current state and go to the stage 1 to transform it. In the case with all states of all abstraction levels have been transformed, go to step 5.

Step4 According to the nature of the current state resulting from (Or-State or And-State predicates) modify its FNLOG specification provided in step1 using one of the patterns presented below:

And-state pattern

$$\boxed{L_S = L_{S_{basic}} \bigwedge_{S_i \in \Sigma_S} L_{S_i})}$$

Or-state pattern

$$\boxed{L_S = L_{S_{basic}} \bigwedge(\bigvee_{S_i \in \Sigma_S} L_{S_i})}$$

Choose, arbitrarily, one of its sub-states and consider it as current state then go to stage 1 to achieve its transformation.

Step5 If the current abstraction level of the specification is the first one then go to step 7. Else substitute the **L** specifications states with their formulae in their **L** specification parent state. Go to step 6.

Step6 The current abstraction level of specification becomes the level of the parent. If it doesn't remain other states non transformed in the current level, go to step 5. Else choose one of the non transformed state and go to step 1.

Step7 End of the procedure.

6 Illustration

In this section, the proposed approach and the transformation algorithm are illustrated over an example of train crossing system [**?**]. The system is composed of warning components which forbid conductors to cross the rail by detecting an arriving train. The warning component is the light signal, while the forbidding component is represented by the barrier. When a train, arriving from any direction, comes in the train detection zone the light signal has to be set up after 9s of the train detection, the barrier has to be down to prohibit the conductors to cross the rails. Once the train is far away from the detection zone, the barrier becomes up and the light signal is off. The system has to guarantee that it won't be a collision between cars and train or cars and barrier.

In the following, STATEMATE specification of the above system will be translated to FNLOG specification in order to verify system properties which are described with FNLOG formulae.

6.1 Step 1:Required properties description in FNLOG and system specification in STATEMATE

6.1.1 Property description

(P1): The barrier is never down if the light signal is not ON (to avoid collisions between barrier and cars at night). The correspondent FNLOG formula is: $\neg(\neg \bigodot_n(Down) \wedge \bigodot_n(ON))$

(P2): The light signal is never OFF only if it doesn't exist any train in the detection zone. In FNLOG, it will be: $\neg(\neg \bigodot_n(OnState) \wedge \bigodot_n(train - arrives))$

(P3): The barrier don't have to be down only after 9s that the light signal is ON. The FNLOG formula is : $\bigodot_n(OffState) \Rightarrow \bigodot_n^{n-9}(OnState)$

These proprieties have to be verified by the following STATEMATE specification after translation into FNLOG specification.

6.1.2 STATEMATE specification

The system is specified with STATEMATE. The specification starts with the context diagram (Figure.3) which is refined with the activity-Chart represented by Figure.4. This activity-Chart contains two sub-activities: *Train-Cross* and *Bar-Sig*. Each of them has a control activity as sub-activity: *Train-Cross-ctrl* and *Bar-Sig-ctrl*. In this paper, we will only present the associated Statecharts to the *Train-Cross-ctrl* control activity (Figure.5), because we will focus only on it, in the translation illustration, by translating *Preventing* and *OnState* states into FNLOG.

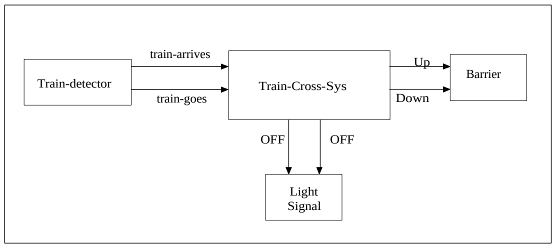


Figure 2: Train crossing system's context diagram

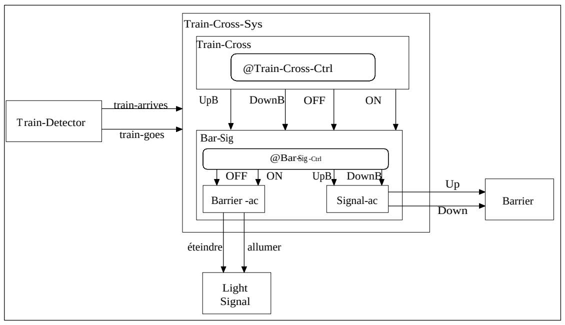


Figure 3: Train crossing system's activity-Chart

6.2 Step2: *Transforming the STATEMATE specification of cross train system to FNLOG specification*

The transformation step is started by transforming, to FNLOG, the activity representing the context diagram, then the Train-crossing system activity-Chart and finally the train-Crossing control activity Statechart and the Barrier-signal control activity Statechart. As

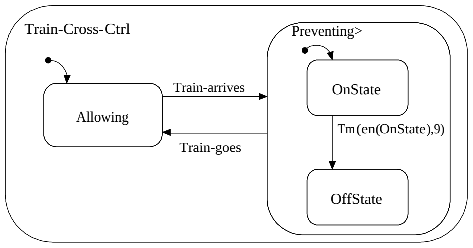


Figure 4: Train-Cross-ctrl activity's Statecharts

mentioned above, we will only present in this section the translation of *Preventing* and *OnState* states into FNLOG.

6.2.1 Translation Preventing state into FNLOG

Apply, on *Preventing* state,the procedure *Translate-Statechart* which translate a STATEMATE state into FNLOG formulae.

1. Preventing state is considered as basic state.
The correspondent specification FNLOG will be:

$\mathrm{L}_{\texttt{Preventing}} = \bigodot_n(\texttt{Preventing})$ with

$1.\bigodot_n(\texttt{Preventing}) = \bigodot_{n_1}(\texttt{Preventing}) \bigwedge \bigodot_{n_2}(\texttt{Preventing}) \bigwedge(\texttt{n1<n<n2})$

$2.\bigodot_n(\texttt{init-Preventing}) = \bigodot_n(\texttt{Train-arrives})$

$3.\bigodot_n(\texttt{term-Preventing}) = \bigodot_n(\texttt{Train-goes})$

2. Test if this state is an And-State by evaluating the predicate *and-state*(`Preventing`). In this case, *and-state*(`Preventing`)= false. Preventing is not an And-state.

3. Test if *Preventing* state is an Or-state by evaluating the predicate *OR-state*(`Preventing`). *OR-state*(`Preventing`)=true, Preventing is the disjunction of the states `OnState` and `OffState`. $\Sigma_{\texttt{Preventing}} = \{\texttt{OnState,OffState}\} \neq \emptyset$

The FNLOG specification of **Preventing** state , $\mathrm{L}_{\texttt{Preventing}}$, will be:

$\mathrm{L}_{\texttt{Preventing}} = \bigodot_n(\texttt{Preventing}) \wedge (L_{\texttt{OnState}} \vee L_{\texttt{OffState}})$

4. Continue with the transformation of the substates of **Preventing** state. Choose randomly a current state from the set substates. $\mathrm{S}_{current} \leftarrow \texttt{OnState}$

$\Sigma_{\texttt{Preventing}} \leftarrow \Sigma_{\texttt{Preventing}} \setminus \{\texttt{OnState}\} = \{\texttt{OffState}\}$.

$\Sigma_{Preventing} = \{\text{OffState}\} \neq \emptyset$

6.2.2 Translation of the state OnState

Apply the procedure *Translate-Statechart* on *OnState*

1. Consider the state *OnState* as basic state. Its FNLOG specification is provided as follow: $\mathrm{L}_{\texttt{OnState}} = \bigodot_n(\texttt{OnState})$ with:

$4.\bigodot_n(\texttt{OnState}) = \bigodot_{n_1}(\texttt{init-OnState}) \bigwedge \bigodot_{n_2}(\texttt{term-OnState}) \bigwedge(\texttt{n1<n<n2})$

$\mathrm{I}_{\texttt{OnState}} = \{\texttt{default}\}$

So we add the following definition: $5.\bigodot_n(\texttt{init-Preventing}) \rightarrow \bigodot_n(\texttt{init-OnState})$

$\mathrm{O}_{\texttt{OnState}} = \{\texttt{Tm(en(OnState),9)}\}$.

So we add the following definition: $6.\bigodot_n(\texttt{term-OnState}) = \bigodot_n(\texttt{Tm(in(OnState),9)})$

`Tm(en(OnState),9)` is a composed event, thus we add to the above FNLOG specification the following definition: $7.\bigodot_n(\texttt{Tm(en(OnState),9)}) = \bigodot_{n-9}(\texttt{OnState})$

2. The predicates *Or-state*(`OnState`) and *Or-state*(`OnState`) are evaluated to false and $\Sigma_{\texttt{OnState}} = \emptyset$. Thus, `OnState` is not a composed state. So, the transformation of the state *OnState* is achieved. The correspondent FNLOG specification is:

$\mathrm{L}_{\texttt{OnState}} = \bigodot_n(\texttt{OnState})$ with:

$4.\bigodot_n(\texttt{OnState}) = \bigodot_{n_1}(\texttt{init-OnState}) \bigwedge \bigodot_{n_2}(\texttt{term-OnState}) \bigwedge(\texttt{n1<n<n2})$

$5.\bigodot_n(\texttt{init-Preventing}) \rightarrow \bigodot_n(\texttt{init-OnState})$

$6.\bigodot_n(\texttt{term-OnState}) = \bigodot_n(\texttt{Tm(en(OnState),9)})$

$7.\bigodot_n(\texttt{Tm(en(OnState),9)}) = \bigodot_{n-9}(\texttt{OnState})$

2. Substitute $\mathrm{L}_{\texttt{OnState}}$ with its formula $\bigodot_n(\texttt{OnState})$ in $\mathrm{L}_{\texttt{Preventing}}$ FNLOG formula which will changed to:

$$\mathrm{L}_{\texttt{Preventing}} = \bigodot_n(\texttt{Preventing}) \wedge (L_{\texttt{OnState}} \vee L_{\texttt{OffState}})$$

6.3 Step3: Properties verification

All of the three properties have been proofed in our work with FNLOG deduction. In this section we present only the proof of the property P2.

(P2): $\bigodot_n(\texttt{train-arrives}) \rightarrow \bigodot_n(\texttt{OnState})$
Def 2 : $\bigodot_n(\texttt{init-Preventing}) = \bigodot_n(\texttt{Train-arrives})$
That means:
$\bigodot_n(\texttt{init-Preventing}) \rightarrow \bigodot_n(\texttt{Train-arrives})$ and
$\bigodot_n(\texttt{Train-arrives}) \rightarrow \bigodot_n(\texttt{init-Preventing})$ **(1)**
Def 5 : $\bigodot_n(\texttt{init-Preventing}) \rightarrow \bigodot_n(\texttt{init-OnState})$ **(2)**
(1) and **(2)**: $\bigodot_n(\texttt{Train-arrives}) \rightarrow \bigodot_n(\texttt{init-OnState})$ **(3)**
FNLOG Axiom : $\bigodot_n(\texttt{init-OnState}) \rightarrow \bigodot_n(\texttt{OnState})$ **(4)**
(3) and **(4)** : $\bigodot_n(\texttt{Train-arrives}) \rightarrow \bigodot_n(\texttt{OnState})$. **(P2)** verified.

7 Conclusion

In this paper, we have presented a systematic translation schemes from STATEMATE models to FNLOG specification. We have considered all features of STATEMATE such as activities, states, connectors, static reactions, activity in state, compound events, compound conditions composed actions and predefined functions. We have also proposed a solution dealing with decomposition of activities and states in a STATEMATE models. Cross references, translation patterns have been developed to be used in our proposed translation algorithm. Once, a STATEMATE specification is translated into a FNLOG notation, the specification becomes verifiable by using the FNLOG axiomatic [19]. Our future focus shall consists of validating transformation rules and developing a tool supporting proposed transformation rules to ensure the systematic verification of required properties.

References

[1] B.Berard, M.Bidoit, F.Laroussine, A.Petit et P.Schnoebelen. *Vérification des logiciels-Techniques et outils du model-Cheking.* Vuibert, Paris, 1999.

[2] R.Bossow and W.Grieskamp. "A Modular Framework for the Integration of Heterogenous Notations and Tools". Proc. Of the 1st Intl. Conference on integrated Formal Methods-IFM00'. Springer-Verlag. London. (2000).

[3] S.Dupuy, Y.Lerdu et M.Chabre-peccoud. "Translating the OMT dynamique model into object-Z". Proceeding of 11th International Conference of Z users. Berlin, Germany. (1998).

[4] A.B.Ferrentino and H.D.Mills. "State machines and their semantics in software engineering". Proc. IEEE COMPSAC'77 Conference(1977) pp. 242-251.

[5] H.Fkih, L.Jemni Ben Ayed, and S.Murz." Transformation des Specification B en des Diagrammes UML". In AFADL: Approches Formelles dans l'Assistance au developpement des logiciels. France.(2004). P 131–145

[6] H.Fkih, L.Jemni Ben Ayed, and S.Murz. "Transformation of B Specifications into UML Class Diagrams and State Machines". The 21st Annual ACM Symposium on Applied Computing. Dijon, France. (2006).

[7] A.Hammad and al. "From a B Specification to UML Statechart Diagrams. "C.George and H.Miao(Eds.):ICFEM 2002,LNCS 2495,pp.511-522, 2002.Spring-Verlag Berlin Heidelberg. (2002).

[8] D.Harrel: "Statecharts. A visual Formalism For Complex System." Communicated by A.Pnueli. Received December 1984. Revised July 1986. Sci-ence of Computer Programming 8 (1987) 231-274. North-Holland.

[9] D.Harrel and M.Politi. *Modeling reactive Sys-tems with Statecharts. the STATEMATE Ap-proach.* McGraw-Hill Books. Software Develop-ment Series.USA.

[10] Y.Hlaoui. *Transformation automatique d'un modele STATEMATE en une specification FNLOG pour la verification.* Januray 2006.

[11] A.Idani and Y.Ledru. "Dynamic Graphical UML Vues from Formal B Spacification". Journal of Information and Software Technology. (2005).

[12] L.Jemni, O.Mosbahi and S.Ben Ahmed. "A Specification and Validation Method Based on STATEMATE and FNLOG". MOSIM'03: Organisation and Conduct of Industry Activities and Ser-vives. (2003). Toulouse- France.

[13] H.Ledang and J. Souquières." Modeling class operations in B : a case study on the pump component". Technical Report A01-R-011, Laboratoire Lorrain de Recherche en Informatique et ses Applications. (2001).

[14] O.MosbahiL, L.Jemni, S.Ben Ahmed and J.Jaray. "A Specification and Validation Technique Based on STATEMATE and FNLOG". 4th international conference on Formal methods and Software Ingeneering ICFEM,LNCS, Vol 2495, Chine October 2002.

[15] Object Modeling Group. *Uinified Modelling Language Specification*, version 1.4. (2001).

[16] E.Sekerinski et R.Zurob."Translating Statecharts to B". In M.Butler, L.Petre, and K.Sere, editors, Proc of the 3rd International Conference on Integrated Formal Methods (IFM'02), volume 2335 of Lecture Notes in Computer Science, pages 128-144, Finland, 2002.Springer-Verlag.

[17] A.Sowmya and S.Ramesh. "Extending Statecharts with Temporal logic".SCSE Report 9401. School of Computer Science and Engineering the university of New South Wales.(1994).

[18] A.Sowmya and S.Ramesh. "Extending Statecharts with Temporal logic." IEEE Transactions on Software Engineering, Vol.24, NO. 3. (1998).

[19] A.Sowmya and S.Ramesh. "A Semantics-Preserving Transformation of statecharts to FNLOG" . Proc.14th IFAC , Seoul, Korea. (1997).