Abstract

Hierarchical automata are used to model hierarchical systems. The semantics used is the Kripke structure where states are valued by atomic propositions. This structure can be large in number of states. This paper presents some heuristics to check properties expressed in LTL logic (Linear Temporal Logic). Hierarchical systems are defined in an hierarchical way by a set of subsystems by decomposing every time one or more states in a set of automata. To cope with the combinatorial explosion problem and the check of properties, we consider only the sub-systems concerned by the property to verify and we then deduct its check from it on the global system.

Keywords: Specification, Hierarchical automata, Modular verification, model-checking, Kripke structure

1. MOTIVATIONS

Our work revolves around the specification and verification of hierarchical systems from works on StateCharts [7] and Unified Modeling Language (UML) [5]. This model has, in addition, concepts such refinement of states by a set of automata, transitions which have multiple source states, multiple target states (Interlevel Transitions) and the priority between transitions.

Various techniques have been introduced to verify linear properties by model-checking on the hierarchical models [4]. In [11], the authors show how StateCharts can be translated into Promela (the specification language which uses SPIN) by using the hierarchical automata as intermediate format to verify linear properties [12].

The verification techniques used are algorithmic and are often based on the detection of cycles to show satisfaction or not of a property by making the synchronous product between the model and the negation of the property given in the form of a Büchi automaton. Properties are expressed using a LTL (Linear Temporal Logic) [9]. The main problem is in the complexity of the decision procedures and the combinatorial explosion of the number of states. A comparison of different approaches proposed to verify StateCharts by model-checking can be found in [3].

In this paper we propose to use verification on modules or sub-structures. This method can be used to verify large-sized systems. The model is cut into a set of sub-models and to determine whether a property is satisfied on each sub-model. The verification is allowing separately on each sub-models [10].

The choice of the properties and the method of cutting are two very important points. It is not always true that : if a property \( \varphi \) is satisfied on all the sub-models then it is satisfied on the whole model.

In this work, we are interesting only to a local properties describing the behavior of a part of the system and which are linked to the sub-models introduced during the process of refinement of states. We call this part of system a Kripke sub-structure, denoted \( KSS \). Each Kripke sub-structure represents a flattening part of a system represented by a set of automata associated with a state. Property \( \varphi \) is only concerned by the Kripke sub-structure \( KSS \) associated with states refined by these automata. If \( KSS \) satisfies the property \( \varphi \) then the whole Kripke structure \( KS \), associated to the hierarchical automaton, also satisfied it. Formally, for all refined state \( s \), if \( KSS_s \models \varphi \) then \( KS \models \varphi \).

This paper is organized as follows : in the second section, we present the basic concepts in particular sequential automata, hierarchical automata and their semantics with Kripke structure. In section three, we present the process
of verifying properties on each of the sub-model. Thus, we express two patterns of properties, then we give the corresponding algorithms of verification. In the section four, we plan some perspectives in this work.

Throughout this paper, we illustrate our comments by the example of the Automated Teller Machine (ATM).

2. PRELIMINARIES

2.1. Introduction

Hierarchical automaton (HA) is built as parallel and/or hierarchical composition of sequential automata. HA have maximal parallelism semantics, i.e. parallel automata execute their transitions synchronously. Hierarchical composition means mapping states of a sequential automaton to another automaton or parallel composition of automata. We start by defining the notion of sequential automata.

2.2. Sequential automata

In the following, \( s, s_0, s_1, \ldots \) denote states and \( V \) is a set of variables \( \{x_1, x_2, \ldots, x_n\} \) of type \( \text{Dom}(x_i) \), which is a finite set.

We call \( AP_V \) the set of atomic propositions over the set of variables \( V \) where \( ap \) is a formula \( x_i = d_j \), with \( x_i \in V \) and \( d_j \in \text{Dom}(x_i) \).

**Definition 1** An automaton \( A \) labelled over a set of variables \( V \) ranging on finite domains is defined by 5-tuple \( (S, s_0, \Sigma, \rightarrow, L) \) where :

- \( S \) is a finite set of states,
- \( s_0 \in S \) is the initial state,
- \( \Sigma \) is a set of labels (actions),
- \( \rightarrow \subseteq S \times \Sigma \times S \) is the transition relation
- \( L : S \rightarrow 2^{AP_V} \) is the state labelling function where \( AP_V \) is the set of propositions.

A path \( \sigma \) of an automaton \( A \) is a finite or an infinite sequence of states and actions \( s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \ldots \xrightarrow{a_i} s_i \xrightarrow{a_{i+1}} \ldots \) such that \( \forall i(i \geq 0 \Rightarrow s_i \xrightarrow{a_i} s_{i+1} \in \rightarrow) \). We denote \( \text{Path}(A) \) the set of paths of \( A \). The trace of a path \( \sigma \), denoted \( tr(\sigma) \), is a sequence of labels of the transitions : \( tr(\sigma) = a_1, a_2 \ldots a_i \ldots \)

We call \( cy \) a cycle of a sequential automaton \( A \) iff \( s_0 \xrightarrow{a_0} \ldots \xrightarrow{a_{n-1}} s_n \) is a finite sequence of states and actions, \( s_n = s_0 \) and for all \( i \), such as \( 0 \leq i < n, s_i \xrightarrow{a_i} s_{i+1} \in \rightarrow \).

Let \( A = (S, s_0, \Sigma, \rightarrow, L) \) be a sequential automaton, in the following, \( S_A \) denote the set of states of \( A \), \( s_0_A \) its initial state, \( \rightarrow_{A} \) to refer to the transition relation of \( A \) and \( \Sigma_A \) to refer to the actions of \( A \).

2.3. Hierarchical automata

Hierarchical automaton is a set of sequential automata which are dependent between them by a composition function \( \gamma \).

**Definition 2** \( HA \) is a tuple \( \langle F, E, \gamma \rangle \) where :

- \( F = \{A_1, \ldots, A_n\} \) is a finite set of sequential automata with distinct state i.e. : \( \forall A_1, A_2.(A_1 \in F \land A_2 \in F \land S_{A_1} \cap S_{A_2} = \emptyset) \),
- \( E = \bigcup_{A \in F} \Sigma_{A} \) is a finite set of actions,
- \( \gamma : \bigcup_{A \in F} S_{A} \rightarrow 2^{F} \) is a composition function on \( F \) where : (1) there exists a unique root automata \( A_{\text{root}} \in F \), (2) every non-root automata has exactly one ancestor state and (3) there are no cycles in \( \gamma \).

Remark : Let \( |\gamma(s)| \) represents the number of automata in \( \gamma(s) \). A state \( s \) of an automaton \( A \in F \) could be refined by only one automaton \( |\gamma(s)| = 1 \), or by a parallel composition of automata \( |\gamma(s)| > 1 \), or could be a basic state \( |\gamma(s)| = 0 \) denoted Basic(s).
For notational convenience we write \( s' \in \chi(s) \) whenever \( s' \) is a state of the direct child automaton of \( s \). Furthermore, we define \( \chi^+ \) to be the irrefflexive transitive closure, \( \chi^* \) to be reflexive transitive closure of \( \chi \). The ancestor composition function \( \gamma^{-}\leftarrow \) describes a link between a state \( s \) and its ancestor automaton.

Let \( HA = (F, E, \gamma) \) be a hierarchical automaton. The restriction of the composition function \( \gamma \) to states of some \( A \in F \) yields a sub-hierarchical automaton \( HAA = (FA, EA, \gamma_A) \) where : \( FA = F \setminus \{ A_i | S_A \cap \chi^*(S_A) = \emptyset \} \), \( EA = E \) and \( \gamma_A = \gamma | \chi^*(S_A) \). We consider \( A \) as the root automaton of \( HAA \).

### 2.4. Configuration

A configuration \( (C) \) denotes a global state of the hierarchical automaton. It describes which states of sequential automata of a hierarchical automaton are simultaneously active. Every sequential automaton can contribute at most one state to a configuration.

**Definition 3** Let \( S_i \) be the set of the states of all the sequential automata in \( HA \). A set \( C \subseteq S_i \) is a configuration of \( HA \) if : (1) exactly one state of the root automaton \( A_{root} \) is in the configuration : \( \exists! s.(s \in S_{A_{root}} \land s \in C) \) and (2) downward closure i.e. whenever a state is in the configuration and it is a non-basic state, each of its direct sub-automata must contribute to the configuration too : \( \forall s_A. (s \in C \land A \in \gamma(s) \Rightarrow \exists s'. (s' \in S_A \land s' \in C)) \).

For each hierarchical automaton \( HAA \), \( A \in F \), the set of all configurations of \( HAA \) is denoted by \( Conf_A \).

### 2.5. Extended transitions

In order to represent statecharts diagrams using hierarchical automata, Mikk and al. [11] have added two labels in \( HA \) which are \( sr \) (source restriction) and \( td \) (target determinator). The aim of the two labels is to preserve the information of the interlevel transitions (the multiple-source and multiple-target transitions).

Thus, the label of a transition \( t \) in a sequential automaton \( A \in F \) is described as \( t \overset{sr,td}{\rightarrow} s', \) where \( source(t) = s \), \( target(t) = s' \), \( action(t) = a \), \( sr \) is a set of configurations of the sub-hierarchical automaton \( HAA_{\gamma(s)} \) and \( td \) is a set of configurations of the sub-hierarchical automaton \( HAA_{\gamma(s')} \).

Let \( s \) and \( s' \) be two states of an automaton \( A \in F \) and \( t = s \overset{sr,td}{\rightarrow} s' \) a transition in \( A \). \( sr \) is used to determine in which configuration \( t \) is enabled and \( td \) is used to determine which states are entered simultaneously when entering the target state \( s' \). If \( s \) is refined by one or more automata and \( sr = \emptyset \), then \( t \) is enabled from all states of the sub-automata of \( s \). If \( s' \) is refined by one or more automata and \( td = \emptyset \), then target states of \( t \) are the initial states of the sub-automata of \( s \).

### 2.6. Example

Automated Teller Machine (ATM) is a computerized telecommunications device that provides the customers of a bank with access to financial transactions in a public space without the need for a human clerk or bank teller. On most modern ATMs, the customer is identified by inserting a plastic ATM card with a magnetic stripe or a plastic smartcard with a chip, that contains a unique card number and some security information, such as an expiration date. Security is provided by the customer entering a personal identification number (PIN or Code). Using an ATM, customers can access their bank accounts in order to make cash withdrawals (or credit card cash advances) and check their account balances. Customers must have a credit card valid. Initially, the ATM is idle. In this state, it may be in a state of waiting for insertion of a card or in maintenance. After insertion of a card, the ATM becomes active and authenticates certain information.

When the customer inserts his card, the ATM verify certain information concerning its validity. After validation, the customer selects an operation of retreat and the ATM processes it. Then, the customer can ask for the printing of a ticket.

We begining with a simplified model of this system. Figure 1 presents an abstract model of ATM described by an hierarchical automaton.
The Hierarchical automaton $HA = (F, E, \gamma)$ consists of three sequential automata $F = \{ATM, Active, Inactive\}$ which are related with the function of composition $\gamma = \{s_0 \rightarrow \{Inactive\} \land s_1 \rightarrow \{Active\} \cup \{s \rightarrow \emptyset\} | s \in \{s_2, s_3, s_4, s_5, s_6, s_7\}\}$. We have $\chi(s_0) = \{s_2, s_3\}$.

### 2.7. Flattening of hierarchical automata

In this section, we introduce the operational semantics of an hierarchical automaton defined as a Kripke structure [11], this allows us to verify the hierarchical systems with finite number of states by model-checking.

A Kripke structure is defined by a set of configurations with transition.

**Definition 4 Enabled transition.** Let $HA = (F, E, \gamma)$ be an hierarchical automaton, $A$ an automaton in $F$, $C$ a configuration in $Conf_A$ and $a$ an action in $E$. We say that the transition $t \overset{def}{=} s \xrightarrow{sr, a, td} s'$ of automaton $A$ is enabled from configuration $C$, denoted $\text{enable}_{(C)}(t)$, if the state $s$ is in the configuration $C$ and $C$ is a configuration in $sr$.

If $sr = \emptyset$ then the transition $t$ is enabled from all the configurations which contain the state $s$.

**Definition 5 Kripke structure.** Let $HA = (F, E, \gamma)$ be an hierarchical automaton, Kripke structure associated to $HA$ is denoted by $KS = (Conf, C_0, \rightarrow_K, E, LK)$ where:

- $Conf$ is a set of configurations of $HA$,
- $C_0$ is the initial configuration,
- $E$ is a set of actions,
- $LK : Conf \rightarrow 2^{AP}$ such as $LK(C) = \cup_{s_i \in C} L(s_i)$ and $\rightarrow_K \subseteq Conf \times 2^E \times Conf$ is the transition relation of $KS$. 

---

**FIGURE 1:** Example of $HA_1$
Let $t \overset{d e f}{=} C \xrightarrow{c} C'$ be a transition, $t \in \rightarrow_K$ if there is a transition $t \overset{d e f}{=} s \xrightarrow{sr,td} s'$ of automaton $A$ enabled from the configuration $C$, the state $s'$ is in the configuration $C'$ and $td \subseteq C'$ where $(td = \emptyset \text{ and } C_{0A} \subseteq C')$.

$$\begin{align*}
C_{0A} & \text{ is the initial configuration of the hierarchical sub-automaton } HA_A. \\
\end{align*}$$

### 2.8. Refinement of hierarchical automata

In this section, we present a refinement relation between two hierarchical automata which is inspired from refinement relation between transition systems [14]. The basic states of hierarchical model can be decomposed by one or several parallel automata. It allows to see in detail the behavior of system in these states.

Let $HA_1 = (F_1, E_1, \gamma_1)$ be the specification of an hierarchical system over a set of variables $V$. Subsequently $HA_1$ and $KS_1$ indicate the abstract system and $HA_2$ and $KS_2$ indicate their refinement. Refinement consists in developing the basic states of an abstract system, into a set of new automata denoted $A_\tau$. The transitions of $A_\tau$ are designed by $\tau$. We consider $A_1 = (S_1, s_{01}, \Sigma_1, \rightarrow_1, L_1)$ an automaton of $HA_1$ and $A_2 = (S_2, s_{02}, \Sigma_2, \rightarrow_2, L_2)$ an automaton of $HA_2$.

The principal characteristics of our refinement are:

- introduction of a gluing invariant $I_{12}$ which expresses the relation between the variables of $V_1$ and those of $V_2$,
- no new deadlocks introduced,
- no livelocks introduced by new actions.

In example of ATM, We will detail the specification of two states Validation and Impression of hierarchical automaton $HA_1$. Concerning the validation of the card, the customer can enter twice the code. If the first input is good, the machine treats his operation. Otherwise, he can input the code a second time. In case of a mistake, the card can be ejected or cancelled. In this level of refinement, it means decomposing the state $s$ by the automaton $TestCode$ and the state $s'$ by three automata ($Ticket$, $Cards$, $Money$). The figure 2 represent the second level of refinement of the refined hierarchical automaton of abstract ATM.

- The variable $Code$ allows to indicate the state of test of the code, $TestCode \in \{Entry1, Entry2, Test1, Test2, Good\}$.
- The variable $T$ denote the state of the ticket printing, $T \in \{Impression, EndImpression, \}$.
- $C$ indicate the card position , $C \in \{In, Out\}$.
- The variable $B$ indicates the state of the distribution of moneys , $B \in \{Distribution, endDistribution\}$.

The new actions are:

- $TestCode$, to test the security code of card,
- $Essai2$, second try of code,
- $CodeV$, to validate the code,
- $Error$, the code is an error,
- $Imprime$ to print the ticket,
- $ejectCard$, to eject the card,
- $distribBillets$, distribution of moneys

The transition $s_1 \overset{annulation}{\rightarrow} s_0$ is enabled from all states of automata $Active$ and $TestCode$, but the transition $s_1 \overset{endOperation}{\rightarrow} s_0$ is enabled only from the configuration $(s_7, s_9, s_{11}, s_{13})$. The transition $s_4 \overset{V_{code}}{\rightarrow} s_5$ is enabled only from the state $s_{17}$.

To be in the context, we remind you the refinement and semantics of hierarchical automata, see figure 3.
Heuristics to Verify LTL Linear Properties of Hierarchical Systems

FIGURE 2: Refinement of ATM by decomposition of states

3. Kripke Sub-Structures and Properties

Some properties can be verified only on the refined system, it is the case of new properties relative to the new automata introduced by the refinement. For example, the property: the process of selection comes after validation of the code cannot be verified on the abstract model.

In this paper we are concerned with the verification of new properties expressed in LTL\(^1\) on hierarchical systems. In this section, we present an approach to modular verification of a class of properties.

The method of cutting induced by the refinement of states allows us to verify effectively that: if a property is verified on all the sub-models then it is also on the whole model of the system.

3.1. The KSS Associated with the new automata

Let \( H A_1 = (F_1, E_1, \gamma_1) \) and \( H A_2 = (F_2, E_2, \gamma_2) \) be two extended hierarchical automata as \( H A_2 \) is obtained from \( H A_1 \) by refined states. Let \( K S_2 = (Conf_{f2}, C_{02}, \rightarrow_{K2}, E_2, LK_2) \) the Kripke structure associated to \( H A_2 \). Let \( S_r \) the set of the refined states by the new automata. We associate for each state \( s \) of \( S_r \) a Kripke sub-structure, denoted \( KSS_s \), of whole Kripke structure \( K S_2 \).

The Kripke sub-structure \( KSS_s \) represents the flattening of the sub-automata of the state \( s \) and the outgoing transitions of \( s \) belonging to the ancestors automata of \( s \). \( KSS_s \) is defined by a subset of configurations \( K S_2 \) and a set of transitions between these configurations.

\(^1\)The definition of the logic can be found in [9, 13]
Heuristics to Verify LTL Linear Properties of Hierarchical Systems

Definition 6 Let $s$ a state of $S_T$. The Kripke sub-structure $KSS_s = (\text{Conf}_s, C_0s, \rightarrow_K,s, LK_2)$ associated to $s$ is defined as:

- $\text{Conf}_s$ is a subset of configurations of $KS_s$, as: $C$ is a configuration in $\text{Conf}_s$ if the state $s$ is in the configuration $C$ and is a configuration obtained from a state $s$.
- $C_0s$ is the initial configuration such as all the initial states of sub-automata of $s$ owned to $C_0s$.
- $E_2$ is a set of actions.
- $LK_2 : \text{Conf}_s \rightarrow 2AP_V$ is the function of the decor configurations,
- $\rightarrow_K,s \subseteq \rightarrow_K$ is the transition relation of $KSS_s$.
- $\rightarrow_K,s$ is a subset of $\rightarrow_K$. $\overset{a}{\rightarrow}_s C_i \rightarrow C_j$ is a transition in $\rightarrow_K,s$ iff $t \in \rightarrow_K, C_i \in \text{Conf}_s$ and $C_j \in \text{Conf}_s$.

The Kripke sub-structure $KSS$ associated to the hierarchical automaton $HA_2$ is represented in figure 4.

The Kripke sub-structure $KSS_{s_4}$ associated to the state $s_4$ consists of 08 configurations and 13 transitions. Transitions labelled by the action annulation are transitions of the ancestor $ATM$, outgoing of the state $s_4$. The transition labelled by the action $V code$ is a transition of the father automaton $Active$, outgoing of the state $s_4$. Other are transitions of the child automaton $Ticket, Card$ and $Moneys$ child of $s_7$.

3.2. Patterns of properties

We propose two patterns of properties which can be verified only on Kripke sub-structures associated with fathers states of the new automata. Patterns of properties on $KSS$ are as:

- $\square (p \Rightarrow q)$,
- $\square (p \Rightarrow \Diamond q)$.

A property related to the explosion of the basic state is as follows:

Proposition 1 Let $s$ be a state of $HA_2$ and $KSS_s$ the Kripke sub-structure associated to $s$. If $s \models p$ then $KSS_s \models \square p$.

The $KSS$ deco configurations produced by the new automata of $s$ contains that of the father state $s$.

Let $s$ a state of $S_T$ and $F_T$ the set of sub-automata. We denote $AP_T$ the set of atomic propositions on variables of $F_T$, $p$ is a proposition expressed on $AP_T$ ((see the figure 5) and $q$ is a proposition expressed on $AP_V$, where $AP_V$ is the set of atomic propositions on variables of $HA_2$.

Figure 5 illustrates the atomic propositions $p$ in $KSS$ associated with new automata.
Heuristics to Verify LTL Linear Properties of Hierarchical Systems

FIGURE 4: KSS associated to states $S_4$ and $S_7$

At semantics level of $H A_2$, The proposition $p$ are present only in the configurations which belong to the $KSS_s$ associated with the state $s$. That is that, there are no other configurations in $K S_2 \setminus K S_{s}$, which contain the proposition $p$ (see the figure 5).

**Proposition 2** All Kripke sub-structures $K S S$ which are not concerned by $p$, satisfy $\varphi : K S S_s \models \varphi \land K S_2 \setminus K S_s \models \square \neg p$

We express a class of properties which we are verifying in a modular way.

**Theorem : 1** Let $p$ and $q$ be two propositions with $p$ a proposition expressed on $A P$. Let $\varphi$ a LTL formula as $\forall (p \Rightarrow \diamond q), \forall (p \Rightarrow \lozenge q)$. If all Kripke sub-structures satisfy $\varphi$, then $K S_2$ satisfies $\varphi$.

The proof uses results of the work presented in [1]. It is necessary to show at first that $K S S$ well cover the space of states of $K S_2$. The first formula is not a problem particularly because transitions are labelled by $p$ and $q$ in $K S S$. The second formula of liveliness reflects the fact that there should not have cycles that take control indefinitely. In other words, there is always an exit of cycle which takes a priority transition to reach a state valued by $q$.

The property "Operation selection comes after the validation code" is expressed in LTL in the following form:

$$ \varphi = \square (\text{Code} = \text{Error} \implies \lozenge (\text{ATM} = \text{Inactive})) $$

This property is to be verified only on the Kripke sub-structure $K S S_4$ because $\text{Code} = \text{Error}$ is an atomic proposition expressed only on the variables of the new automaton $T e s t C o d e$. 
Heuristics to Verify LTL Linear Properties of Hierarchical Systems

Figur 5: Illustration of \( p \) and \( q \) in \( K_{S_2} \) and in \( KSS \).

Others \( KSS \) are not concerned by \( \varphi \) and thus we can conclude that \( \varphi \) is satisfied on \( K_{S_2} \) of the ATM. The verification of these properties can be done using the model-checking. As we least wish to use it, we exploit the refinement of states to make the check by means of simple algorithms which are linear in number of states.

3.3. Algorithms

In this section, we propose algorithms to verify the two following properties on \( KSS : \Box(p \Rightarrow \Diamond q), \Box(p \Rightarrow \Diamond q) \).

These algorithms take in input a Kripke sub-structure \( KSS \) and two atomic propositions \( p \) and \( q \). They use a transversal graphs visiting all the configurations of \( KSS \). The first algorithm is simple and is to visit the successor configurations of a given configuration. The second algorithm uses a procedure of cycles detection (the strongly connected components algorithm by Tarjan) to consider the priority of transitions (transitions of the father automaton). We could use a transversal graph by depth to visit cycles and see if for each cycle detected, there is a transition priority.

3.3.1. To verify if \( KSS_{s} \models \Box(p \Rightarrow \Diamond q) \)

/* This algorithm requires a transversal graph by depth of the reachable configurations of \( KSS \). It's test if target configurations of outgoing transitions of the configuration \( C \in C_{satisf}\varphi \) satisfy \( q \)? */

\[
\begin{align*}
C_{satisf}\varphi & := \{ C \models p \}; \\
Satisfaction & := true; // boolean : true if there is satisfaction \\
& \text{For each transition } C \xrightarrow{a} C' \in \rightarrow KSS, \text{ do} \\
& \quad Satisfaction := Satisfaction \land C' \models q; \\
& \text{End For}
\end{align*}
\]

If \( Satisfaction = true \), this means that the property is true on \( KSS \). If \( Satisfaction = true \) on all \( KSS \), then the property is satisfied on \( K_{S_2} \). On the other hand, if it is false on a \( KSS \), then it is false on \( K_{S_2} \).

This algorithm works for any structure of Kripke, it uses a function successor to visit the reachable configurations of a given configuration. A transversal graphs in width is enough. The complexity is of the order \( O(|\rightarrow|) \) where \( \rightarrow \) is the set of transitions of \( KSS \).

3.3.2. To verify if \( KSS_{s} \models \Box(p \Rightarrow \Diamond q) \)

/* For each path \( \sigma \) in a cycle \( c_{y} \) there is an outgoing old transition.*/

\[
\begin{align*}
C_{satisf}\varphi & := \{ C \models p \}; \\
Satisfaction & := true; // boolean : true if there is a satisfaction \\
H_{s2} & := \text{Set of the priorities transitions} \\
& \text{For each cycle } c_{y} = C_{i} \ldots C_{j} \text{ reached from } C \text{ Do} \\
& \quad \text{If } C_{k,i \leq k \leq j} \in C_{satisf}\varphi \text{ Then} \\
& \quad \quad Satisfaction := Satisfaction \land (\forall C_{i \leq n \leq j} \rightarrow C_{m} \in H_{s2} \land C_{m} \in C_{satisf}\varphi) \\
& \quad \quad \lor (\exists C_{k,i \leq k \leq j} \in C_{satisf}\varphi \land \exists C_{m,i \leq n \leq j} \rightarrow C_{m} \in H_{s2} \land C_{m} \notin C_{satisf}\varphi)); \\
& \quad \text{End For}
\end{align*}
\]
Heuristics to Verify LTL Linear Properties of Hierarchical Systems

If Satisfaction = true, this means that the property is true on KSS. If Satisfaction = true on all KSS, then the property is satisfied on KS2. Conversely, if Satisfaction = false, it does not necessarily mean that the property is false, it is possible that it is true on the whole Kripke structure KS. The complexity is in order to O(| | + |Conf|) where → is the set of transitions of KSS, and |Conf| is the set of configurations of KSS.

4. CONCLUSION

We have proposed a class of properties, describing the behaviour introduced during the process of refinement. These properties are checked locally on a part of the refined Kripke structure. This is to prove that if a property ϕ is satisfied on the concerned Kripke sub-structures, then it will be satisfied on the whole refined Kripke structure. The properties to be verified locally concern only the new introduced automata.

The new properties, expressed on the new automata, can be also verified by model-checking on small-sized modules. We showed that a simple transversal graphs of the reachable configurations of the refined Kripke structure is enough for deciding on the satisfaction of a property ϕ on the whole system.

We intend to extend our technique to deal with a larger class of temporal properties, because our objective is to exploit the concept of hierarchy without using the model-checking or possibly use it on the sub-models with reasonable size.

REFERENCES