Journal of Engineering and Applied Sciences 13 (3): 557-563, 2018
ISSN: 1816-949X

Model Checking Auto-Concurrency

[1]Zine El Abidine Bounab and [2]Salim Benayoune
[1]Department of Mathematics and Informatics, Research Laboratory RELA (CS)[2],
Oum EL Bouaghi University, Oum, El Bouaghi, Algeria
[2]Laboratoire Allianstic, Groupe Efrei Paris Sud, Villejuif, France

Abstract: This study defines a new operational semantics for a subset of CSP dedicated to express concurrency. We have defined formally the translation from a specifications expressed in CSP to a true concurrency semantic model called CLTS (Concurrent Labeled Transition System). CLTS is the unification of two semantic model the multi-set labeled transition system where transition are labeled with a set of actions instead of a single action and causality semantics where the dependences between actions is considered. The main contribution of this research is to demonstrate that the algorithms for model checking suggested in the literature which are based upon the interleaving semantics can be customized easily to the true concurrency semantics for the verification of the new type of properties associated with the simultaneous execution of actions in various transitions.

Key words: True concurrency semantics, process algebra, model checking, CTL, execution, transitions

INTRODUCTION

There is an increasing need for reliable software, which is especially, critical in some areas such as distributed systems, medical equipments, highway and air traffic control, railways, electronic commerce and many others. Because of their critical character like high degree of concurrency, these application are often subjected to austere requirement reliability, aiming "zero error" quality. There are predictions that in the future the main problem for the application of information technology will not be the lack of raw computational power but our inability to develop complex systems with sufficient confidence in their correctness.

The traditional engineering techniques for validation, like simulation and testing have often proved inadequate and too expensive to avoid errors in information processing artifacts. One reason for this is that they explore only a part of the possible behavior of the system. As a result some erroneous behavior often escapes undetected. Hence the need for formal verification approaches. There are two main approaches to the verification problem:

- Proof-based methods which attempt to carry out verification at the source program level using theorem provers
- Model-based methods or model checking technique which translate the source program into a (possibly finite) model, then comparing this model with the specifications

The formal verification approach concerned by our study is based on models. The reasons for this choice is that this approach is fully automatic and has proved its efficiency in the industry for more information on the advantage of this approach we refer the reader to (Clarke *et al*., 1999).

In model checking approach the system to be verified is described using some specification language and then this specification is translated to some specific semantic model (state transition graph) and the desired properties to be verified on the model are described using temporal logic. Formally, the problem can be stated as follows: given a desired property, expressed as a temporal logic formula p and a Model M with initial state s, decide if M, s=P (Clarke *et al*., 1999).

The model-checking algorithms can be classified into global and local algorithms. Global algorithms require that the underlying transition system is completely constructed while local algorithms compute the necessary part of the transition system on-the-fly. Global algorithms typically compute the fix points in an inductive manner while the local algorithms decide the problem by depth-first-search (Clarke *et al*., 1999). In the approach,

Corresponding Author: Zine El Abidine Bounab, Department of Mathematics and Informatics, Research Laboratory RELA (CS)[2], Oum EL Bouaghi University, Oum, El Bouaghi, Algeria

the application to be verified firstly specified by means of the Process Algebra CSP. This specification will be translated using the operational semantics of CLTS to a graph called Concurrent Labeled Transition System (CLTS) to the best of our knowledge this concurrency semantic model is original. The desired properties of the system are written in CTL logic and they are verified by means of the model checking tool. The underlying algorithm of our model checking is classified as a global algorithm. A tool called CSP has been developed using CLTS semantics.

The semantics of a concurrent system can be characterized by the set of states of the system and transitions by which the system passes a state to another. In the approach presented by Mukund (1992), Mukund and Nielsen (1992) and Beek *et al.* (2007) transition can carry a set of action instead of single action. In standard LTS a transition represent the atomic execution of a single action. Consequently a transition with a set of actions represent there parallel executions. Hence, we can distinguish sequential executions and parallel executions of actions. we assume that the reader is familiar with CSP (Roscoe, 1998).

CSP allows the specification of data and concurrency. The following grammars describe the process expressions which are going to be taken into account in the structural operational semantics and which define the subsets which can be translated to the concurrent labeled transition system semantic model. The CSP syntax taken into account is as follows:

$$p:: = stop|skip|a \rightarrow p|p[]p|p|||pp||pp;p|$$
$$p\Delta p|p/\{a\}$$

Concurrent labeled transition system

Definition 1: M being a countable set of event names , a concurrent labeled transition system or CLTS of support M is a 5-tuple $(\Omega, \Sigma, \lambda, \varphi, \eta)$.

where:

Ω = $(S, I, \delta, \alpha, \beta)$ a transition system
S = A set of state ranged over by $s_0, s_1, \ldots$
$I \epsilon S$ = An initial state
δ = A transition relation defined as $\delta: \delta \times 2^M_{fn} \rightarrow S$
α, β = two functions from $t \epsilon \delta$: α (t) is the origin of the transition and β (t) is its goal

- (Ω, Σ) a transition system labeled by an alphabet Σ which is a set of actions ranged over by a, b,...
- $\lambda: S \rightarrow 2^M_{fn}$ is function which associates to every state a finite set of causal event names present at this state corresponding to actions started their execution so that their terminations allow the start of enabled actions in this state
- $\varphi: 2^M_{fn}$ is function which associates to every transition a finite set of event names represented the parallel execution of the associated actions
- $\eta: \delta \times M \rightarrow \Sigma$ is function which associates to every event name in a transition the corresponding action

The following conditions are fulfilled:

$$\forall t1, t2 \in \delta \text{ if } \beta(t1)=\beta(t2) \Rightarrow \varphi(t1)=\varphi(t2)$$

$$\forall t1 \in \delta, \forall e \in (\lambda(\alpha(t)-\lambda(\beta t)) \Rightarrow e \notin \varphi(t2)$$

$$\forall t1 \in \delta \Rightarrow \lambda(\beta(t1)) \subseteq \varphi(t)$$

- The first condition ensuring that the resulting Kripke structure from CLTS is consistent
- The second conditions ensuring that the causal events in the state should be started in the previous transition, it means don't waiting for the termination of a not yet started action
- The third condition means that the causal completed actions of the state of source in a transition must be non-existent on this transition

MATERIALS AND METHODS

CLTS Model based logic verification

Class of properties to be verified: Several model checkers, based on interleaving semantics were developed in the literature. For our part, the use of CLTS as semantic model where the information included in the transitions represents the actions that are potentially in execution. For this fact, one can express belonging properties such that mutual exclusion in a more natural way as well as from new properties that concern actions and their parallel execution. The expression of these properties does not require the use of new logic or the introduction of new operators, since, one can use CTL temporal logic and consider actions on transitions as being atomic formulae. However, what changes is the intuition behind formulae. For examples, the formula EF(p∧q) where p and q are names of actions means that there is at least a path which leads to state where parallel execution of p and q can take place. In similar way, one can explain intuitively all the formulae of the CTL logic that may be checked using CLTS model as follows:

p∧q on the state S means that p and q can be executed in parallel in the state S of CLTS where $S = \beta(t)$ and $\{p, q\} \subseteq \varphi(t)$.

¬p in state S means that $\forall t \epsilon \delta$, $s = \beta$ (t) $p \notin \varphi$ (t), i.e., the execution of p in the state S cannot take place. EXp in state means that there is at least a path $(S_0, S_1, \ldots)$ where

p will be able to be valid in transition lead to state S_1. AXp in a state S_0 means that for any path (S_0, S_1, …) starting from sate S_0, p is executed at all the transitions lead to the next sate of S_0. E(pUq) in state means that there is at least a path (S_0, S_1, …, S_k, …) where q will be valid in the state and p will be valid in every state of this path that precedes the state S_k.

A (pUq) in state S_0 means that for any path starting from the state, S_0 there is a state in this path where q is valid and p will be valid in every preceding state.

(a: n) in state S_i means that n action of name a may be in execution simultaneously on the precedent transitions of S_i. This is due to the fact that every action is associated with an event name that allows distinguishing between several parallel executions of the same action at any transitions, this properties is called auto-concurrency. It is obvious that such properties cannot be expressed using interleaving models. For instance p:3 express the fact that there is three parallel execution of the action p. p:3 will be considered as being an atomic proposition; what will avoid the introduction of new operator in the considered logic.

Furthermore, if we want to specify actions incompatibility we may express that a and b are incompatible by AG¬(a∧b) which means that they will never be able to be executed concurrently. In a similar way to verify that actions can be executed concurrently may be expressed by EF(a∧b∧c, ..., ∧z) where a, b, ..., z are action names.

CLTS construction from CSP

Definition 2: The set of events names is a countable set noted M. M, N, ..., denote finite subsets of M. The elements of these sets is ranged over by x, y, ..., 2^M_{fin} is the power sets of M. for M∈ 2^M_{fin}, x∈M and a $\alpha\epsilon$ Act. Choosing an event name can be done by a deterministic manner using any function get: Act× 2^M_{fin} →M satisfying get (a, M)∈M for each M∈ 2^M_{fin}. The main condition for the function get is the avoidance of divergence in the call of process.

Definition 2: The set C of configurations is defined by induction as follows where B is the set of behavioral expression of CSP:

- ∀E∈B ∀M∈ 2^M_{fin} $[E]_M$∈C; $[-]_M$ is called the encapsulation operator
- if ε∈C∀M ∈ 2^M_{fin} $[E]_M$ then ε/M∈C
- if ε∈C and F∈B then ε; F∈C
- if ε∈C and F∈C then ε op F∈C op ∈ {[], |||, ||, Δ}

Definition 3: A configuration is canonical if it cannot be reduced by the distribution of the operation of encapsulation over CSP operators. As an example the configuration $[stop]|||[stop]_{\{x, y\}}$ is not canonical but the configuration $[stop]_{\{z\}}|||[stop]_{\{x, y\}}$ is canonical.

Proposition 1: Every canonical configuration is in one of the following forms (Where E and F are canonical configurations):

$$[stop]_M [skip]_M, [a \to E]_M, \varepsilon[]\mathcal{F}, \varepsilon \,|||\, \mathcal{F}, \varepsilon \| \mathcal{F},$$
$$\varepsilon; F, \varepsilon \Delta \mathcal{F}, \varepsilon/x$$

Given a configuration ε, if ε' is a configuration obtained from ε by the distribution of the encapsulation operator on the other operators, we denote by ε→ε' this transformation. The only cases to be studied are the configurations of the form $[G]_M$, we list the different cases of figures:

$$G = E \text{ Op } F: [G]_M \to [E]_M \text{ Op } [F]_M \to \varepsilon \text{Op} \mathcal{F} \text{ Op}$$
$$\{[], |||, ||; \Delta\}; G = E; F: [G]_M \to [E]_M; F \to \varepsilon; F$$

In what follows all configurations are assumed canonical, transforming configurations to the canonical form is assumed to be implicit.

Definition 4: The function λ: C→ 2^M_{fin} return the set of causal or dependent event names in the configuration C is defined by induction as:

$$\lambda([stop]_M) = M; \lambda([stop]_M) = M; \lambda(a \to E)_M = M$$
$$\lambda(\varepsilon op \mathcal{F}) = \lambda(\varepsilon) \cup \lambda(\mathcal{F}) op \in \{[], |||, ||, \Delta\};$$
$$\lambda(\varepsilon; \mathcal{F}) = \lambda(\varepsilon); \lambda(\varepsilon / x) = \lambda(\varepsilon)$$

Example 1:

$$\lambda([stop]_{\{x\}} |||[stop]_{\{y\}}) = \{x, y\}$$
$$\lambda([stop]\{x\} = \{x\})$$

Proposition 2: For all configurations:

$$\varepsilon[]\mathcal{F}, \lambda(\varepsilon) = (\mathcal{F})$$

Proof: We know that the configuration ε[]f behave either like ε or like $\mathcal{F}$. The only way to obtain this configuration $[E[]F]_M$ whereas $[E[]F]_M \to [E]_M$ $[]F]_M \to \varepsilon[]\mathcal{F}$ such that ε and $\mathcal{F}$ are canonical configurations with $[E]_M \to \varepsilon$ and $[F]_M \to \mathcal{F}$ because λ(ε) = M and λ $(\mathcal{F})$ = M hence, the result.

Definition 5: The function ψ: C× 2^M_{fin} →C is defined as follow:

$$\psi([E1]_{M1}, M2) = [E1]_{M1 \cap M2} \psi(\varepsilon \text{Op } \mathcal{F}, M) =$$
$$\psi(\varepsilon, M) \text{ Op } \psi(\mathcal{F}, M)$$

This function is used for the elimination of terminated actions in certain configuration.

Definition 6: The function ξ: $2^{M}_{fn} \times 2^{M}_{fn} \times 2^{M}_{fn} \rightarrow 2^{M}_{fn}$ is defined as follow:

$$\xi(M1, M2, M3) = M2 / (M2 \cap (M1 / M3))$$

This function calculates the remaining parallel actions in certain configuration.

RESULTS AND DISCUSSION

Definition 1; "CLTS based structural operational semantics of CSP": The operational semantic rules are defined on configurations and the whole rules define the transition relation. The transition relation $\neg C \times 2^{M}_{fn} \times C$ is defined as the smallest relation satisfied the following rules.

Rule 1 implies that the successful termination cannot start until all actions referenced by the set M have completed execution.

Rule 2 characterizes the semantics of the prefixing operators as the start of execution of the action a depends on the termination of the actions in M only action a represented by the event name x has appeared in the causal set of the configuration E.

Rules 3a, b characterize the semantics of the operator of choice which is a direct adaptation for configurations of the inference rule defined for the structural operational semantics of interleaving of CSP behavioral expressions.

Rules 5a-c characterize the semantics of the parallel composition operator. To better explain these rules we consider the behavioral expression.

G = [a→c→d→stop] || p[b→c→e→stop] in the configuration [G]∅, actions a and b are enabled to start their execution, a possible derivation when a start first:

$$[G]_{\varnothing} \xrightarrow{\{(a,x)\}} [c \rightarrow d \rightarrow stop]_{\{x\}} \| [b \rightarrow c \rightarrow e \rightarrow stop]_{\varnothing}$$

At this state c cannot start their execution, since, it must be synchronized , so, we can launch the execution of b, we get:

$$[G]_{\varnothing} \xrightarrow{\{(x,y)\}} [c \rightarrow d \rightarrow stop]_{\{x\}} \| [b \rightarrow c \rightarrow e \rightarrow stop]_{\varnothing}$$
$$\xrightarrow{\{(a,x),(b,y)\}} [C \rightarrow d \rightarrow stop]_{\{x\}} \| [c \rightarrow e \rightarrow stop]_{\{y\}}$$

now we can execute c we have :

$$[c \rightarrow d \rightarrow stop]_{\{x\}} \| [c \rightarrow e \rightarrow stop]_{\{y\}}$$
$$\xrightarrow{\{(c,z)\}} [d \rightarrow stop]_{\{z\}} \| [e \rightarrow stop]_{\{z\}}$$

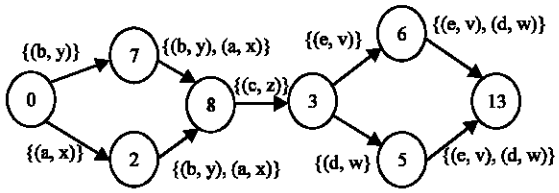

Fig. 1: CLTS corresponding to the behavioral expression G

in the state [d→stop]||[e→stop]$_{(z)}$ either d or e can start their execution after the termination of the execution of the action c represented by the event name z, a possible transition is :

$$[d \rightarrow stop]_{\{z\}} \| [e \rightarrow stop]_{\{z\}} \xrightarrow{\{(d,w)\}} [stop]_{\{w\}} \|$$
$$[e \rightarrow stop]_{\varphi} \xrightarrow{\{(d,w),(e,v)\}} [stop]_{\{w\}} \| [stop]_{\{v\}}$$

The CLTS corresponding to the derivation of the configuration [G]∅ is depicted in Fig.1.

Rule 6a, b is direct translation of the sequencing operator of processes. in fact, rule 6a implies that the behavior of ε; F is that of ε as long as ε has not successfully complete its execution. Whereas the rule 6b states that once ε finished successfully, the process F can start its execution only when the operation skip of the process ε has totally finished, this fact is represented by the configuration [F]δ.

The Rules 7a-c give the semantics of the interrupt operator. To better understand the mechanism of interruption, it is necessary to distinguish the different strategies of interruption which may be deemed in the presence of the non-atomicity of actions which are:

The first strategy is to stop the process, including actions that are underway. The second strategy is to prevent the execution of the actions that have not started while leaving the current actions continues their execution.

The third strategy is to authorize the interruption only when actions underway has finished. The drawback of that strategy is that it may never be able to interrupt the process in the case where there is an overlap between the beginning and ending of actions.

This is the second strategy that was taken into consideration. In the semantics of CLTS, the set of causal events associated with states is of vital importance because it represents the actions that are started execution and may still be trying to run in this state and other action are waiting for it . Therefore, when one of the process ε or f finishes running, either voluntarily or after an interruption, it is replaced in the resulting configuration by the process that no longer offers new actions but track any action that may still be in execution. This implies,

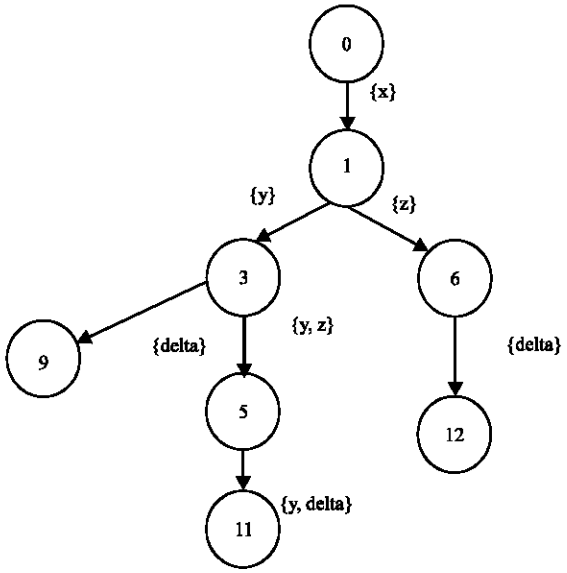

Fig. 2: CLTS corresponding to the behavioral expression G of interruption

under assumption that actions are not atomic that the interruption of a process only affects actions that have not yet started their execution; furthermore current executed actions will terminate their execution (Fig. 2) for illustration consider the example:

$$G = a \rightarrow [b \rightarrow SKIP \Delta c \rightarrow SKIP]$$

The initial configuration is G∅ from this configuration we can derive:

$$G_{\varnothing} \xrightarrow{\{(a,x)\}} [b \rightarrow SKIP]_{\{x\}} \Delta [c \rightarrow SKIP]_{\{x\}}$$

In the case the action a finished its execution and left process execute b without interruption using the Rule 7a, we get:

$$[b \rightarrow SKIP]_{\{x\}} \Delta [c \rightarrow SKIP]_{\{x\}} \xrightarrow{\{(b,y)\}} [SKIP]_{\{y\}} \Delta [c \rightarrow SKIP]_{\{\}}$$

In the other case when the action a finished its execution and right process execute c and interrupt the left process using the Rule 7c, we get:

$$[b \rightarrow SKIP]_{\{x\}} \Delta [c \rightarrow SKIP]_{\{x\}} \xrightarrow{\{(c,z)\}} [SKIP]_{\{z\}}$$

For the case the right process interrupt the left process after its execution of the action b and before its termination, using the Rule 7c, we get:

$$[SKIP]_{\{y\}} \Delta [c \rightarrow skip]_{()} \xrightarrow{\{(b,y),(c,z)\}} [STOP]_{\{y\}} \Delta [SKIP]_{\{z\}}$$

In the case the left process finish its execution before interruption, using the Rule 7b we have:

$$[SKIP]_{\{y\}} \Delta [c \rightarrow SKIP]_{()} \xrightarrow{(\delta)} [STOP]_{(\delta)}$$

This operational semantics take in consideration auto-concurrency due to the notion of events associated to actions. The function get is defined based on the set M this last is defined based on the context of the prefix operator which is the basic constructor for the generation of the state space (transition relation), from an implementation point of view this set can be passed as parameter to the semantics rules and represent the parallel events. Furthermore this way can reduce the temporal complexity of state space generation and avoid complex computation for getting this information from transitions. The event δ represent the successful termination after the execution of the SKIP action. The function η(x) return the action α associated to the event x, in our implementation is just the function first. The last semantic rule 8 is the semantic of call for process which is similar to the α reduction of λ calculus.

Case study: In this study, we present the railway level crossing example, the problem of this example is defined as follow: One road and one railway line cross each other and as a usual there is a gate which can be lowered to prevent cars crossing the railway. If the gate is raised, then cars can freely cross the track. A train can cross the road regardless of whether the gate is up or down. There is a safety and a liveness property that can be checked in this model which are:

- Mutual exclusion: there should never be a train and a car on the cross point at the same time
- Starvation free: whenever a car or a train approaches the crossing they should eventually be able to cross

CSP specification: System railway (car-run, car approach, car-enter, car-leave, train-run, train-approach, train-enter, train-leave,gate-raise,gate-lower, crash, crash2) ::=
vehicule(crash, car-run, car-approach, car-enter, car-leave, train-run, train-approach, train-enter, train-leave
)|||
gate (gate-raise, gate-lower)
where,
Process gate (a,b) ::= b->a->gate (a,b) EndProc
Procces svehicule (crash, car-run, car-approach, car-enter, car-leave, train-run,train-approach, train-enter, train-leave) ::=car (crash, car-run, car-approach, car-enter, car-leave)
|[crash]|
train (crash, train-run, train-approach, train-enter, train-leave) EndProc
Process car (crash, car-run, car-approach, car-enter, car-leave) ::=
(car-run- > car-approach- > car-enter >car-leave >car (crash, car-run, car-approach, car-enter, car-leave))
[](crash- > STOP) EndProc
Process train (crash, train-run, train-approach, train-enter, train-leave) ::=
(train-run- > train-approach- > train-enter- > train-leave- >
train (crash, train-run, train-approach, train-enter, train-leave))
[] (crash- > STOP) EndProc
EndSystem

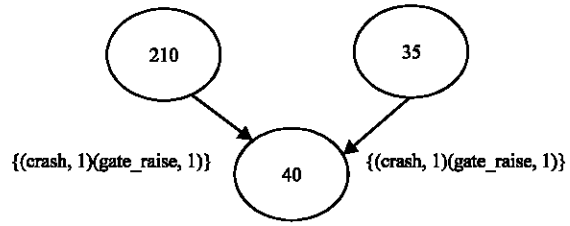

Fig. 3: CLTS corresponding to the specification of the railway system state 40 violate the mutual exclusion properties

This specification of the railway system can be found in (https://github.com/bzine19/model-checking-csp/tree/code) it is the exact translation of the specification from RSL (Lizeth Tapia and Chris George, 2008) to CSP. After compiling this specification using the tool, the CLTS corresponding to the specification of the railway system has 240 transitions and 78 states.

Mutual exclusion for our examples when there is no crash, since, this action is the synchronization of the process car and the process train. We can specify this property using CTL logic as follow: AG¬(crash).

Here, we can see that we have specifying the properties on actions with natural way. After the verification using our model checking tool we have founding that the system violate the mutual exclusion properties and as a counter example we have the state number 40 which has the crash action in execution, see the part of the system depicted in Fig. 3.

For the case of the starvation property we can express it by CTL formulae for example for the case of car as follow: car-approach⇒AF (car-enter).

It means each time when a car is approaching and want to cross the railway it will be able to do it in the future. This property is satisfied by the specification. The main difference between our approach and the approach presented by Tapia and George (2008) is that in there technique of verification they use the refinement approach where the system is specified by CSP and the properties also are specified by CSP process. All equations are discussed as:

$$[skip]_M \overset{\{\delta\}}{} [stop]_{\{\delta\}} \tag{1}$$

$$[a \to E]_M \overset{\{x\}}{} [E]_{(x)} \quad x = get(a, M) \tag{2}$$

$$\frac{\varepsilon \overset{M}{} \varepsilon'}{\varepsilon[]\mathcal{F} \overset{M}{} \varepsilon'} \tag{3}$$

$$\frac{\mathcal{F} \overset{M}{} \mathcal{F}'}{\varepsilon[]\mathcal{F} \overset{M}{} \mathcal{F}'} \tag{4}$$

$$\frac{\varepsilon \overset{M}{} \varepsilon', M' = \lambda(\varepsilon')}{\varepsilon \| \mathcal{F} \overset{M^*}{} \varepsilon' \| \mathcal{F}'} \tag{5}$$

$$\mathcal{F}' = \Psi(F, \xi(M1, M2, M')),$$
$$M1 = \lambda(\varepsilon), M2 = \lambda(\mathcal{F}),$$
$$M'' = \lambda(\mathcal{F}') \cup \lambda(\varepsilon')$$

$$\frac{\varepsilon \overset{M}{} \varepsilon', M' = \lambda(\varepsilon')}{\mathcal{F} \| \varepsilon \overset{M^*}{} \mathcal{F}' \| \varepsilon'} \tag{6}$$

$$\mathcal{F}' = \Psi\, \mathcal{F}, \xi(M1, M2, M'),$$
$$M1 = \lambda(\varepsilon), M2 = \lambda(\mathcal{F}),$$
$$M'' = \lambda(\mathcal{F}') \cup \lambda(\varepsilon')$$

$$\frac{\varepsilon \overset{\{x\} \cup M1}{} \varepsilon', \mathcal{F} \overset{\{x\} \cup M2}{} \mathcal{F}'}{\varepsilon \| \mathcal{F} \overset{M^*}{} \varepsilon' \| \mathcal{F}'} \tag{7}$$

$$a \in \alpha E \cap \alpha F, a = \eta(x)$$
$$M' = \lambda(\mathcal{F}') \cup \lambda(\varepsilon')$$

$$\frac{\varepsilon \overset{M}{} \varepsilon', M' = \lambda(\varepsilon')}{\mathcal{F} \| \varepsilon \overset{M^*}{} \mathcal{F}' \| \varepsilon'} \tag{8}$$

$$F' = \Psi(\mathcal{F}, \xi(M1, M2, M')),$$
$$M1 = \lambda(\varepsilon), M2 = \lambda(\mathcal{F}),$$
$$M'' = \lambda(\mathcal{F}') \cup \lambda(\varepsilon')$$

$$\frac{\varepsilon \overset{M}{} \varepsilon', M' = \lambda(\varepsilon')}{F \| \varepsilon \overset{M^*}{} F' \| \varepsilon'} \tag{9}$$

$$F' = \Psi(\mathcal{F}, \xi(M1, M2, M')),$$
$$M1 = \lambda(\varepsilon), M2 = \lambda(\mathcal{F}),$$
$$M'' = \lambda(\mathcal{F}') \cup \lambda(\varepsilon')$$

$$\frac{\varepsilon \overset{M}{} \varepsilon', \delta \notin M}{\varepsilon; F \overset{M}{} \varepsilon'; F} \tag{10}$$

$$\frac{\varepsilon \overset{(\delta)}{} \varepsilon'}{\varepsilon; F \overset{(\delta)}{} \varepsilon'; F_{(\delta)}} \tag{11}$$

$$\frac{\varepsilon \overset{M}{} \varepsilon', \delta \notin M}{\varepsilon \Delta \mathcal{F} \overset{M,'}{} \varepsilon' \Delta \mathcal{F}'} \tag{12}$$

$$M' = \lambda(\mathcal{F}') \cup \lambda(\varepsilon')$$
$$F' = \Psi(\mathcal{F}', \xi(M1, M2, M)),$$
$$M1 = \lambda(\varepsilon), M2 = \lambda(\varepsilon')$$

$$\frac{\varepsilon \overset{(\delta)}{\rightarrow} \varepsilon'}{\varepsilon \Delta \mathcal{F} \overset{(\delta)}{\rightarrow} \varepsilon'} \quad (13)$$

$$\frac{\varepsilon \overset{M}{\rightarrow} \varepsilon', \delta \notin M}{\varepsilon \Delta \mathcal{F} \overset{M'}{\rightarrow} \varepsilon' \Delta \mathcal{F}'} \quad (14)$$

$$M' = M3 \cup M2,$$
$$M3 = \lambda(\varepsilon) / ((\lambda(\varepsilon) \cap \lambda(\mathcal{F})) / \lambda(\mathcal{F}')$$

$$\frac{P ::= E, E_M \overset{M'}{\rightarrow} \mathcal{F}}{[P]_M \overset{M'}{\rightarrow} \mathcal{F}} \quad (15)$$

CONCLUSION

In this study, we have shown an operational manner for generating concurrent labeled transition system from a subset of the process algebra CSP with a few examples we have shown the subtleties of this semantics which can verify auto-concurrency, a similar research has been done in the literature Costa and Courtiat (1993) and Saidouni and Courtiat (1993) this last is state based semantics versus our semantics which is a transition based semantics for more information on this subject we refer the reader to Smith (2013), Hansen *et al.* (2003) and Nicola and Vaandrager (1990). Another things to mention is the way the properties are verified in our approach the properties can be verified directly by model checking versus the approach of the behavioral equivalence between the properties and the semantic model (Tapia and George, 2008), it means the refinement approach. For perspective we will enhance this model with real time concepts thanks to the event notion which can be naturally transformed to clocks and mimic timed automata.

REFERENCES

Beek, T.M.H., A. Fantechi, S. Gnesi and F. Mazzanti, 2007. An action/state-based model-checking approach for the analysis of communication protocols for service-oriented applications. Proceedings of the International Workshop on Formal Methods for Industrial Critical Systems, July 1-2, 2007, Springer, Berlin, Germany, pp: 133-148.

Clarke, E.M., O. Grumberg and D. Peled, 1999. Model Checking. MIT Press, Cambridge, Massachusetts, Pages: 309.

Costa, D.R.C. and J.P. Courtiat, 1993. A Causality-Based Semantics for CCS. In: NAPAW 92 Workshops in Computing, Purushothaman S. and A. Zwarico (Eds.). Springer, London, England, UK., pp: 200-215.

Hansen, H., H. Virtanen and A. Valmari, 2003. Merging state-based and action-based verification. Proceedings of the Third International Conference on Application of Concurrency to System Design, June 20, 2003, IEEE, Guimaraes, Portugal, ISBN:0-7695-1887-7, pp: 150-156.

Mukund, M. and M. Nielsen, 1992. CCS, locations and asynchronous transition systems. Proceedings of the International Conference on Foundations of Software Technology and Theoretical Computer Science, December 18-20, 1992, Springer, New Delhi, India, pp: 328-341.

Mukund, M., 1992. Transition system models for concurrency. Masters Thesis, Aarhus University, Aarhus, Denmark.

Nicola, D.R. and F. Vaandrager, 1990. Action versus state based logics for transition systems. Proceedings of the Semantics on Systems of Concurrent Processes, April 23-27, 1990, Springer, Posay, France, pp: 407-419.

Roscoe, A.W., 1998. The Theory and Parctice of Concurrency. Prentice-Hall, Upper Saddle River, New Jersey, ISBN:9780136744092, Pages: 565.

Saidouni, D.E. and J.P. Courtiat, 1993. A comparison of the semantics of maximality and causality of basic LOTOS. Toulouse Cedex, ?Occitanie, France.

Smith, J., 2013. State/event based versus purely action or state based logics. Allen Institute, Ithaca, New York, USA.

Tapia, L. and C. George, 2008. Model checking concurrent RSL with CSPM and FDR2. Masters Thesis, The United Nation University, Shibuya, Japan.