Deliverable 4: Transformation Assessment

Assessment of Analysis and Transformation Techniques

Esprit Project 27439 - HIDE

*H*igh-level *I*ntegrated *D*esign *E*nvironment for Dependability

A. Bondavalli, M. Dal Cin, E. Giusti, D. Latella, I. Majzik, M. Massink, I. Mura

Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)

Consorzio Pisa Ricerche - Pisa Dependable Computing Centre (PDCC)

Technical University of Budapest (TUB)

HIDE/D4/FAU/1/v1.1

Contents

Qualitative Assessment of the Transformation from Structural UML Diagrams to Timed Petri Nets

Andrea Bondavalli CNUCE/CNR and PDCC
Majzik Istvan TUB
Ivan Mura University of Pisa and PDCC

Introduction

Though all the transformation procedure has been worked out, a prototype implementation is not available yet. Moreover no example of applying the transformation is available. Actually this transformation targets large systems and has the primary task to provide a modelling framework in which detailed behavioral models can be added as information becomes available and the need for such precision arise. It was out of the scope of the first phase of HIDE to provide such large UML specifications to exercise the transformations. This activity, which will provid feed-backs to improve and optimise (from a quantitative viewpoint) the transformation is part of the second phase of the project. Thus, at this stage, we are far from being able to provide a real quantitative assessment of the transformation. However some remarks related to the size of the resulting models and to a comparison with hand-made models for the same systems is given.

1. Adherence to UML diagrams

The basic input for the transformation from UML to timed Petri net models is represented by the standard structural UML diagrams, that is use cases, classes, objects and deployment diagrams.
Since the UML specification does not cover the aspects related to the dependability of the system, these input diagrams are enriched to allow the designer to provide the parameters needed for the definition of the dependability models.
Essentially two types of extensions are necessary: one for identifying redundancy and the other for defining the dependability parameters. The facilities to introduce such extensions are found in the standard UML language itself. They are:
*Tagged values***.**
*Constraints***.**
*Stereotypes***.**

Comments.
The number of parameters needed for the definition of the models is kept as low as possible. Default values are provided to be used whenever an explicit assignment is not performed. Also, a library of pre-defined classical fault-tolerance schemes is provided, defined by using the UML notation. On the other hand, no particular restrictions are imposed to the UML designer, apart from the controlled introduction of the redundancy into the diagrams. Therefore, we can affirm that the transformation does maintain completely the adherance to UML diagrams.

2. Semantic truthfulness of the transformation

The transformation procedure starts from the UML structural diagrams, and proceeds towards Petri net models. It is important to point out that not only the set of UML diagrams that forms the input of our transformation do not have a formal semantics, but also the specification this set provides might be incomplete or ambiguous.
Therefore, before discussing about the semantic truthfulness of the transformation, we must observe that a formal proof of correctness of the transformation itself can not be given because the original UML models do not possess a formal precise semantics. On the other side, a Petri net model does have a semantic. Thus, what we have done with the first step of the transformation has been to fix fault, propagation and repair processes in a more precise way so that the intermediate model resulting from the UML diagrams becomes a proper input for the definition of a timed Petri net dependability model.
Then a first step of the transformation is defined, which builds the intermediate model by performing an abstraction from the plethora of structural UML diagrams. The Intermediate model identifies:

1) a set of elements, which form a subset of all the entities contained in the UML specification
2) a set of relations among elements, which specifies (as far as possible with the information provided by structural diagrams) the way elements interacts.

In the intermediate model, each element is assigned an internal failure/repair process, which originates the so-called basic events. Each relation specifies a way for the propagation of basic events from one element to the other, thus defining a set of new events, called derived events. In the end, the failure of the system will be one of this derived events.
Starting from the Intermediate model, a second step of the transformation generates a timed Petri net model, as follows:

1) for each element, define the basic subnets modelling the local failure/repait processes of the element
2) for each relation, define the failure/repair propagation subnets modelling the generation of the derived events

3. Availability of a complete description of the transformation

The transformation has been specified in a precise and unhambiguous way. The algorithmic description of some steps of model generation have been provided. The computational complexity of the transformation algorithms is linear in the size of the overall number of UML diagrams composing the specification of the system.

4. Availability and performance figures of a prototype implementation

Though all the transformation procedure has been worked out, a prototype implementation is not available yet. Moreover no examples of applying the transformation are available. Actually this transformation targets large systems and has the primary task to provide a modelling framework in which detailed behavioral models can be added as information become available and the need for such precision arise. It was out of the scope of the first phase of HIDE to provide such large UML specifications to exercise the transformations. This activity, which will provid feed-backs to improve and optimise (from a quantitative viewpoit) the transformation is part of the second phase of the project.

5. Number of modelling elements wrt. the number of modelling elements of the original UML models

The following statement appears as one of the quantitative assessment targets: “The number of modelling elements produced by the transformation is of the same order of magnitude as the number of modelling elements of the original UML models”.
Despite the lack of an implementation, some arguments to support this statement can already be given. We separately consider the two steps of the transformation.
The generation of the Intermediate model is an abstraction step from the UML specification, therefore the intermediate model size is no bigger than that of the UML specification. At most, the set of nodes of the intermediate model can in a one-to-one correspondence with the set of UML entities. However, since behavioural diagrams and composite UML entities (apart stereotypes for fault-tolerance schemes) does not have a correspondent in the intermediate model, the number of nodes is in general less than the number of UML entities. Similarly, the number of relations in the intermediate model is a subset of the relations among UML entities which appear in the UML diagrams.
Notice that the fault-trees associated from the FTS nodes of the intermediate model are in fact a concise representation of the behaviour of the controller of the fault-tolerance scheme itself. This representation can be obtained from the UML statechart of the controller, with the algorithmic procedure given in Deliverable 2. The size of the fault-tree is of course proportional to the size of the controller’s statechart.
The generation of the timed Petri net produces a model with a number of modelling elements proportional to those of the elements of the intermediate model. Of course, we are here just

considering the case when no refined submodels obtained with the other HIDE transformations are included into the final dependability model.
In particular, for each node of the intermediate model, the basic subnets are generated, which for a type of node other than FTS and SYS contain:

- a number of places ranging from 3 to 6;
- a number of transitions ranging from 4 to 8;
- a number of input/output arcs ranging from 3 to 10.

These basic subnets are linked by a set of input/output arcs whose number is between 3 and 6.
Each FTS type of node and the SYS node turn out in a subnet that only contains 2 places.
For each of the U relations of the intermediate model, a failure propagation subnet is generated, which contains:

- 3 places
- 4 immediate transitions
- 7 input/output arcs

4 input/output arcs are added to the model to link the failure propagation subnet to the basic subnets previoulsy defined, and from 4 to 8 input/output arcs are added to constrain the repair.
For each of the C relations, a failure and a repair subnet are generated, which correspond to the fault-tree expressing the logic of the associated fault-tolerance scheme. The size of these propagation subnets (places and transitions) is proportional to the number of of elements (intermediate events and gates) of the fault-tree.
The failure propagation subnet thus generated is linked with 1 input and 1 output arc to each of the elements involved in the C relation, and so does the repair propagation subnet. Last, a set of input/output arcs is added to link between them the failure and repair propagation subnets. These arcs are in number proportional to the size of the fault-tree.
Therefore, the size of final dependability Petri net model is proportional to the size of the intermediate model, whose size is in turns less or equal to that of the UML specification.

6. Efficiency of the analysis of the automatically generated models.

The second target of the quantitative assessment is expressed in the Project Plan as: “For a selected set of examples, the efficiency of the analysis of the automatically generated models is within an order of magnitude of the efficiency of corresponding hand-made models.”
Since the transformation has not been implemented yet, the considerations that follow focus on the size of the Resulting Petri Net allowing to draw expected differences in the cost of the analysis performed using the transformation wrt. that of using hand-made models.

The size of the automatically generated model is minimal with respect to the basic events, in a sense no more concise timed Petri net model can be used to represent the considered failure/repair scenario.

A hand-made model could save on the modelling elements involved in the propagation processes (failure/repair), at the expenses of the modularity. Therefore, timed Petri net built by the transformation results in marking processes whose state space can be bigger than those underlying hand-made models. However, note that no timed transitions are introduced by the propagation subnets, and this implies that all the additional states found in the marking process contribute only to vanishing markings.

Thus, the definition of the reduced reachability graph for a hand-made model can be less expensive than for the a timed Petri net model generated by the transformation. On the other hand, the computational complexity to numerically solve the stochastic process represented by the reduced reachability graph does not differ for a hand-made and automatically generated model.

Qualitative and Quantitative Assessment of the Transformation from Dynamic UML Diagrams to Stochastic Petri Nets

Mario Dal Cin FAU- IMMD3

1 Qualitative Assessment

The qualitative criteria for the assessment of the transformations as defined in the Project Programme of the HIDE contract are:

QL1) Adherence to the UML-modeling techniques

QL2) Semantic truthfulness

QL3) Availability of the formal definition of the translations

We assess these criteria as follows.

QL1) Adherence to the UML-modeling techniques
The dynamic aspects of an UML-model are captured in interaction diagrams (Sequence Diagrams), Statecharts and Activity Diagrams. We referred to the realization of this aspect by these diagrams as the *Dynamic Model*. We developed transformations for all of these diagrams.

With regard to Statecharts, however, we only considered a subclass which we felt is most appropriate for modeling embedded systems [3, 4, 7] and which has a well defined semantics. 'An embedded system is a software-intensive collection of hardware that interfaces with the physical world. Embedded systems involve software that controls devices such as motors, actuators and displays and, in turn, is controlled by external stimuli such as sensor input, movement, and temperature changes' [3]. An example of a model of such an embedded system is provided by our demonstrator.

We introduced no new modeling element. So, adherence to the UML-modeling techniques is guaranteed.

QL2) Semantic truthfulness
It was our concern to preserve the semantics of the UML-constructs. This led us to make a few restrictions with regard to the modeling power of some of these constructs. These restrictions, however, are not essential. Thus the generated Petri nets exhibit the same

semantics as their UML-counterparts. When turning to Stochastic Petri nets we enhanced their semantics in a straightforward manner so that the resulting Generalized Stochastic Petri nets [1] specify Semi-Markov-Processes.

QL3) Availability of the formal definition of the translations
The translation of the Dynamic Model to Stochastic Perti nets is given informally but in an rigorous way.

Concerning Sequence Diagrams, Graubmann et. al. [8] indicated how to transform sequence diagrams into Petri nets, specifically into labeled occurrence nets. They provide formal specifications for transformation rules for basic sequence diagram constructs and cover also structural concepts like co-regions and sub-diagrams. When defining our transformation we followed their approach.

Activity Diagrams can be viewed as a combination of Statecharts and Petri nets. Hence, the transformation of activity diagrams to Generalized Stochastic Petri nets is straightforward.

Concerning Statecharts, we defined in an rigorous way the subclass of Guarded Statecharts [5] such that their translation to Petri nets also became straightforward. Here we employed some modeling constructs for Petri nets which are supported by the evaluation tool PANDA [2]. These constructs can be translated to standard constructs albeit in a somewhat laborious way.

Thus, the criteria QL1, OL3 and QL2 can be considered as being satisfied by the transformations.

2 Quantitative Assessment

The quantitative criteria for the assessment of the transformations as defined in the Project Programme are:

QL4) Number of modeling elements with respect to the number of modeling elements of the original UML models.

QL5) Availability and performance figures of a prototype implementation.

The translation starts from the UML-dynamic diagrams. These diagrams are exported from the UML-modeling environment and deposited in a database. The first step of the translation is to access this database and to generate a Petri net specific database. This Petri net specific database can then be accessed by the evaluation tools. Hence, it provides an open interface for evaluation tools based on Petri nets. We employed PANDA as our evaluation tool. Thus, the second step of the translation is to generate PANDA readable files.

QL4) Number of modeling elements with respect to the number of modeling elements of the original UML models.
The number of elements of the Petri net models is the same as the number of modeling elements of the original UML-models. States are transformed to places and arcs to transitions; guards of UML-Statecharts (i.e., of Guarded Statecharts) become guards of Petri net transitions. Therefor, it goes without saying that 'the complexity of the automatically generated models is within an order of magnitude of the complexity of corresponding hand-made models'.

Concerning again the Statecharts we provided a means to model, for example, sensor and actuator faults by state perturbations. This gives us the possibility to inject fault during the evaluation of the model. With this possibility the number of modeling elements doubles at most.

QL5) Availability and performance figures of a prototype implementation.
A prototype implementation based on the Production Cell example is available. This example is of manageable size albeit not a trivial one. It has been widely used as a benchmark for modeling and verification techniques. Our model example contains a full suit of UML-models: the requirement models (use case and sequence diagram), the object model, the package model, the deployment model and last but not least the dynamic model. The dynamic model has been transformed to a Kripke Structure for formal verification and to a Generalized Stochastic Petri Net for quantitative evaluation. The usefulness of formal verification was clearly demonstrated, since by formal verification we detected in short time, for example, several dead-locks in the original version of the model.

The dynamic model of the production cell contains 11 Statecharts with altogether 84 concurrent states and 95 concurrent state transitions. The Cartesian product of the individual state spaces has 265.072 Millions states of which are 375 544 reachable. Hence, problems when evaluating the model were expected.

Nevertheless, we were able to evaluate the Statechart model and we performed several evaluation runs. The main problem is not posed by the evaluation times but by the huge size of the reachability graph of the obtained Stochastic Petri Net. We were, therefore, forced to use the supercomputer (Convex Exemplar) of our University. Fortunately, our evaluation tool made it very easy to map the job onto a parallel machine [2]. Using the supercomputer considerably larger models can be analyzed.

On the other hand, the sequence diagrams of our example are quite simple and, hence, posed no problem for their evaluation. Based on the sequence diagrams, we computed performance figures such as cumulative distribution functions for the time to process a metal blank.

These evaluation experiments provided us with very useful experience and insight when dealing with and evaluating relatively large UML-models.

References

[1] M. Ajmone Marsan, G. Balbo, G. Conte, Performance Models of Multiprocessor Systems, The MIT Press 1986

[2] S. Allmaier, S. Dalibor: PANDA -- Petri net ANalysis and Design Assistant, Tools Descriptions, 9th Int. Conference on Modeling Techniques and Tools for Computer Performance Evaluation, St. Malo, 1997

[3] G. Booch, J. Rumbauch, I. Jacobson, The Unified Modeling Language User Guide, Addison Wesseley, 1998

[4] D. Harel, M. Politi. Modeling Reactive systems with Statecharts, McGraw Hill, 1998

[5] Deliverable 1 Specification of the Modeling Techniques

[6] Deliverable 5 Demonstrator

[7] Deliverable 6. Specification of the Pilot Application

[8] P. Graubmann, E. Rudolph, J. Gabrowski., Towards a Petri Net based semantics definition for message sequence charts, Siemens AG ZFE, Technical Report

Qualitative assessment of the translation from Statechart Diagrams to Kripke Structures

D. Latella - CPR PDCC and CNR Ist. CNUCE
I. Majzik - TUB
M. Massink - CNR Ist. CNUCE

The qualitative criteria for the assessment of the translation are those defined in the Project Programme of the HIDE contract. They are listed below:

QL1) Adherence to the UML-modeling techniques

QL2) Semantic truthfulness

QL3) Availability of the formal definition of the translation

The translation from statechart diagrams to Kripke Structures satisfies the requirements set by the above criteria:

QL1) Being statechart diagrams a notation defined within the UML, and being their translation into Kripke Structures a way for providing such a notation with a formal operational semantics, *adherence to the UML-modeling techniques* is guaranteed.

So the requirements of criterion $QL1$ are satisfied.

QL2) The formal proofs of some interesting properties of extended hierarchical automata and their semantic model, including the correctness of such a model with respect to the informal definition of statechart diagrams of the UML, show that the translation is sound from a semantical point of view. The proofs are given below.

This provides evidence for the fulfillment of $QL2$.

QL3) The complete formal definition of the translation from Extended Hierarchical Automata to Kripke Structures (i.e. the operational semantics of Extended Hierarchical Automata) can be found in [1] and in Deliverable 2 of Task 1.2 of the HIDE Project.

The translation from statechart diagrams to Extended Hierarchical Automata is given informally but in a rigorous way.

So also $QL3$ is fulfilled.

The fully detailed proofs of the propositions and the main theorem stated in the Deliverable 2 of Task 1.2 are given below. The reader is referred to Deliverable 2 of Task 1.2 for all relevant definitions.

In the sequel we will implicitly make reference to a generic extended hierarchical automaton $H = (F, E, \rho)$. Moreover we let $A \in F, \mathcal{C} \in \mathrm{Conf}_H, \mathcal{E} \in (\Theta E), P \in 2^{(\mathcal{T}\ H)}$ be respectively a generic automaton, a configuration, an environment and a set of transitions.

Lemma 1 *For $A, A', \bar{A}, \bar{A}' \in F$, $s \in \mathcal{S}\ H$, the following holds: (i) $A' \in \mathcal{A}\ A$ implies $\mathcal{A}\ A' \subseteq \mathcal{A}\ A$, $\mathcal{S}\ A' \subseteq \mathcal{S}\ A$, and $\mathcal{T}\ A' \subseteq \mathcal{T}\ A$. (ii) $A, A' \in (\rho s), A \neq A', \bar{A} \in (\mathcal{A}\ A), \bar{A}' \in (\mathcal{A}\ A')$ implies $\mathcal{A}\ \bar{A} \cap \mathcal{A}\ \bar{A}' = \mathcal{S}\ \bar{A} \cap \mathcal{S}\ \bar{A}' = \mathcal{T}\ \bar{A} \cap \mathcal{T}\ \bar{A}' = \emptyset$. (iii) $s \in (\mathcal{S}\ A)$ implies $\exists_1 A' \in (\mathcal{A}\ A).\ s \in \sigma_{A'}$.*

Proof We prove only (i) since (ii) and (iii) are simple consequences of the definitions of Extended Hierarchical Automata, $\mathcal{A}, \mathcal{S}, Tr$.
We first prove $A' \in \mathcal{A}\ A$ implies $\mathcal{A}\ A' \subseteq \mathcal{A}\ A$, by induction on the tree structure of F. In fact we can define a relation on F such that X is related Y iff $X \in \rho\sigma_Y$ and take its reflexive and transitive closure. This way, we get a well-founded partial order, since antisymmetry is a consequence of property (iii) in the definition of extended hierarchical automata. The bottom elements are those X such that $\rho\sigma_X = \emptyset$.

Base case:
Trivial.

Induction step:
If $A' \in \rho\sigma_A$ then $\mathcal{A}\ A' \subseteq \mathcal{A}\ A$ easily follows from the definition of $\mathcal{A}$. Suppose then there exists A'' such that $A' \in \mathcal{A}\ A''$ and $A'' \in \rho\sigma_A$.

$A'' \in \rho\sigma_A \wedge A' \in \mathcal{A}\ A''$

$\Rightarrow$ {Induction Hypothesis($\mathcal{A}\ A''$ strictly smaller than $\mathcal{A}\ A'$)}

$A'' \in \rho\sigma_A \wedge \mathcal{A}\ A' \subseteq \mathcal{A}\ A''$

$\Rightarrow$ {definition of $\mathcal{A}$}

$\mathcal{A}\ A'' \subseteq \mathcal{A}\ A \wedge \mathcal{A}\ A' \subseteq \mathcal{A}\ A''$

$\Rightarrow$ {Set Theory}

$\mathcal{A}\ A' \subseteq \mathcal{A}\ A$

The proof for $\mathcal{S}\ A' \subseteq \mathcal{S}\ A$ whenever $A' \in \mathcal{A}\ A$ follows:

$\mathcal{S}\ A'$
$=$ {definition of $\mathcal{S}$}

$\bigcup_{A'' \in \mathcal{A}\ A'} \sigma_{A''}$

$\subseteq$ {$\mathcal{A}\ A' \subseteq \mathcal{A}\ A$ (see above); Set Theory}

$\bigcup_{A'' \in \mathcal{A}\ A} \sigma_{A''}$

$=$ {definition of $\mathcal{S}$}

$\mathcal{S}\ A$

The proof for $\mathcal{T}$ is similar to that for $\mathcal{S}$.

□

Lemma 2 *For $A, A' \in F$, $s \in \sigma_A$, $s' \in \sigma_{A'}$ the following holds: $s' \in \mathcal{S}\ (\rho s) \Rightarrow A' \in (\mathcal{A}\ A)$.*

Proof

$s' \in \mathcal{S}\ (\rho s)$

$\Rightarrow$ {definition of image}

$\exists \bar{A} \in \rho s.\ s' \in (\mathcal{S}\ \bar{A})$

$\Rightarrow$ {Lemma 1 (iii)}

$\exists \bar{A} \in \rho s.\ \exists_1 \bar{A}' \in \mathcal{A}\ \bar{A}.\ s' \in \sigma_{\bar{A}'}$

$\Rightarrow$ {$s' \in \sigma_{A'}$ and state sets of different automata disjoint implies $A' = \bar{A}'$}

$\exists \bar{A} \in \rho s.\ A' \in (\mathcal{A}\ \bar{A})$

$\Rightarrow$ {$\bar{A} \in \rho s \Rightarrow \bar{A} \in (\mathcal{A}\ A)$; Lemma 1 (i)}

$\exists \bar{A} \in \rho s.\ A' \in (\mathcal{A}\ \bar{A}) \wedge (\mathcal{A}\ \bar{A}) \subseteq (\mathcal{A}\ A)$

$\Rightarrow$ {Logics}

$A' \in (\mathcal{A}\ A)$ □

Proposition 1 *Relation $\preceq$ is a partial order.*

Proof Transitivity easily follows from Lemmata 1 and 2, antisymmetry follows from property (iii) of ρ. □

Lemma 3 *For $s, s', \bar{s} \in (\mathcal{S}\ H)$ the following holds:* $(s \preceq \bar{s}) \wedge (s' \preceq \bar{s}) \Rightarrow (s \preceq s') \vee (s' \preceq s)$

Proof Suppose w.l.g. there is no s'' such that $s \prec s'' \prec \bar{s}$ or $s' \prec s'' \prec \bar{s}$ and assume $s \neq s'$.

$(s \prec \bar{s}) \wedge (s' \prec \bar{s})$

$\Rightarrow$ {definition of $\preceq, \mathcal{S}, \mathcal{A}$ }

$\exists A \in F.\ \bar{s} \in \sigma_A \wedge A \in (\rho\ s) \cap (\rho\ s')$

$\Rightarrow$ {definition of extended hierarchical automaton (ii)}

FALSE □

Lemma 4 *For all $A, A' \in F$ and all $s \in \sigma_A$, $s' \in \sigma_{A'}$ the following holds:* $s \preceq s' \Rightarrow (\mathcal{S}\ A) \cap (\mathcal{S}\ A') \neq \emptyset$

Proof The assert trivially holds for $s = s'$.

$s \prec s'$

$\Rightarrow$ {definition of $\prec$}

$s' \in \mathcal{S}\ (\rho s)$

$\Rightarrow$ {Lemma 2}

$A' \in \mathcal{A}\ A$

$\Rightarrow$ {Lemma 1 (i)}

$(\mathcal{S}\ A') \subseteq (\mathcal{S}\ A)$

$\Rightarrow$ $\{s^0_{A'} \in (\mathcal{S}\ A')\}$

$(\mathcal{S}\ A') \cap (\mathcal{S}\ A) \neq \emptyset$ □

Lemma 5 *For all* $s, s' \in \mathcal{S}\ H$ *the following holds:* $s\ ||\ s' \Rightarrow s \not\preceq s'$

Proof

$(s\ ||\ s') \wedge (s \preceq s')$

$\Rightarrow$ {definition of $||$}

$(\exists \bar{s} \in (\mathcal{S}\ H), A, A' \in (\rho \bar{s}).\ A \neq A' \wedge s \in \mathcal{S}\ A \wedge s' \in \mathcal{S}\ A') \wedge (s \preceq s')$

$\Rightarrow$ {Lemma 1 (ii)}

$(\exists A, A' \in F.\ s \in \mathcal{S}\ A \wedge s' \in \mathcal{S}\ A' \wedge \mathcal{S}\ A \cap \mathcal{S}\ A' = \emptyset) \wedge (s \preceq s')$

$\Rightarrow$ {Lemma 1 (iii)}

$(\exists A, A' \in F.\ \exists_1 \bar{A} \in (\mathcal{A}\ A), \bar{A}' \in (\mathcal{A}\ A').\ s \in \sigma_{\bar{A}} \wedge s' \in \sigma_{\bar{A}'} \wedge \mathcal{S}\ A \cap \mathcal{S}\ A' = \emptyset) \wedge (s \preceq s')$

$\Rightarrow$ {Lemma 1 (i)}

$(\exists A, A', \bar{A}, \bar{A}' \in F.$

$(\mathcal{S}\ \bar{A}) \subseteq (\mathcal{S}\ A) \wedge (\mathcal{S}\ \bar{A}') \subseteq (\mathcal{S}\ A') \wedge s \in \sigma_{\bar{A}} \wedge s' \in \sigma_{\bar{A}'} \wedge \mathcal{S}\ A \cap \mathcal{S}\ A' = \emptyset) \wedge (s \preceq s')$

$\Rightarrow$ {Set Theory}

$(\exists \bar{A}, \bar{A}' \in F.\ s \in \sigma_{\bar{A}} \wedge s' \in \sigma_{\bar{A}'} \wedge \mathcal{S}\ \bar{A} \cap \mathcal{S}\ \bar{A}' = \emptyset) \wedge (s \preceq s')$

$\Rightarrow$ {Lemma 4}

$(\exists \bar{A}, \bar{A}' \in F.\ \mathcal{S}\ \bar{A} \cap \mathcal{S}\ \bar{A}' = \emptyset \wedge \mathcal{S}\ \bar{A} \cap \mathcal{S}\ \bar{A}' \neq \emptyset$

$\Rightarrow$ {Logics}

FALSE □

Lemma 6 *For* $S, S' \subseteq \mathcal{S}\ H$ *sets of* pairwise orthogonal states $S \preceq^s S' \wedge S' \preceq^s S$ *implies* $S = S'$.

Proof We show $S \subseteq S'$, the proof for $S' \subseteq S$ being similar.

$s \in S$

$\Rightarrow$ {definition of $\preceq^s$; $S \preceq^s S'$}

$\exists s' \in S'.\ s \preceq s'$

$\Rightarrow$ {definition of $\preceq^s$; $S' \preceq^s S$}

$\exists s' \in S'.\ s \preceq s' \wedge \exists s'' \in S.\ s' \preceq s''$

$\Rightarrow$ {Transitivity of $\preceq$}

$s \preceq s''$

$\Rightarrow$ {Lemma 5; $s, s'' \in S$}

$s = s''$

$\Rightarrow$ {$s \preceq s' \preceq s''$; $s = s''$}

$s = s'$

$\Rightarrow$ {$s' \in S'$}

$s \in S'$ □

Lemma 7 *For $t, t' \in (\mathcal{T}\ H)$ the following holds: $(SRC\ t)\ ||\ (SRC\ t')$ implies $\neg(t \# t')$.*

Proof

$(SRC\ t)\ ||\ (SRC\ t')$

$\Rightarrow$ {Commutativity of $||$ follows directly from its definition}

$(SRC\ t)\ ||\ (SRC\ t') \wedge (SRC\ t')\ ||\ (SRC\ t)$

$\Rightarrow$ {Lemma 5}

$(SRC\ t) \not\preceq (SRC\ t') \wedge (SRC\ t') \not\preceq (SRC\ t)$

$\Rightarrow$ {Logics}

$\neg((SRC\ t) \preceq (SRC\ t') \vee (SRC\ t') \preceq (SRC\ t))$

$\Rightarrow$ {definition of $\#$}

$\neg(t \# t')$ □

Lemma 8 *For $t, t' \in (\mathcal{T}\ H)$ the following holds: $(SRC\ t)\ ||\ (SRC\ t')$ implies $\pi t \not\sqsubseteq \pi t'$.*

Proof

$(SRC\ t)\ ||\ (SRC\ t') \wedge \pi t \sqsubseteq \pi t'$

$\Rightarrow$ {Lemma 7; $(SRC\ t)\ ||\ (SRC\ t')$ implies $t \neq t'$}

$\neg(t\#t') \wedge t \neq t' \wedge \pi t \sqsubseteq \pi t'$

$\Rightarrow$ {definition of Priority Schema}

$\neg(t\#t') \wedge t\#t'$

$\Rightarrow$ {Logics}

FALSE $\square$

Proposition 2 *$(PWO, \preceq^s, f)$ is a priority schema.*

Proof It follows directly from the definition of $\preceq^s$ and from Lemma 6 that $(PWO, \preceq^s)$ is a partial order. Moreover the co-domain of f is PWO since for any t $(ORIG\ t)$ is pairwise orthogonal by definition. We show that $\forall t, t' \in (\mathcal{T}\ H).\ (f\,t \preceq^s f\,t') \wedge t \neq t' \Rightarrow t\#t'$:

$f\,t \preceq^s f\,t'$

$\Rightarrow$ {definition of f}

$ORIG\ t \preceq^s ORIG\ t'$

$\Rightarrow$ {definition of $\preceq^s$}

$\forall s \in ORIG\ t.\ \exists s' \in ORIG\ t'.\ s \preceq s'$

$\Rightarrow$ {$\forall x \in ORIG\ y.\ SRC\ y \preceq x$}

$\forall s \in ORIG\ t.\ \exists s' \in ORIG\ t'.\ s \preceq s' \wedge SRC\ t \preceq s \wedge SRC\ t' \preceq s'$

$\Rightarrow$ {Logics}

$\exists s \in ORIG\ t, s' \in ORIG\ t'.\ s \preceq s' \wedge SRC\ t \preceq s \wedge SRC\ t' \preceq s'$

$\Rightarrow$ {$\preceq$ transitive}

$\exists s' \in (\mathcal{S}\ H).\ SRC\ t \preceq s' \wedge SRC\ t' \preceq s'$

$\Rightarrow$ {Lemma 3}

$(SRC\ t \preceq SRC\ t') \vee (SRC\ t' \preceq SRC\ t)$

$\Rightarrow$ {definition of $\#$}

$t\#t'$ $\square$

Proposition 3 *For $A \in F$ and $A' \in \rho_A\ \sigma_A$: $\mathcal{C} \in \mathrm{Conf}_A \wedge \mathcal{C} \cap \sigma_{A'} \neq \emptyset \Rightarrow \mathcal{C} \cap \mathcal{S}\ A' \in \mathrm{Conf}_{A'}$*

Proof The assert easily follows from the definitions. $\square$

The following proofs are carried out by induction either on the length of the derivation for proving $A \uparrow P :: (\mathcal{C}, \mathcal{E}) \xrightarrow{L} (\mathcal{C}', \mathcal{E}')$ [3] or on the structure of the subset of F affected by $\mathcal{C}$. With respect to structural induction, let $F_{\mathcal{C}}$ be the set $\{A \in F \mid \mathcal{C} \cap \sigma_A \neq \emptyset\}$. It is easy to define a relation on $F_{\mathcal{C}}$ such that X is related Y iff $s \in \mathcal{C} \cap \sigma_Y$ and $X \in (\rho_Y s)$. Notice that since $\mathcal{C}$ is a configuration, for each A in $F_{\mathcal{C}}$ there is a unique state $s \in \sigma_A \cap \mathcal{C}$. The transitive and reflexive closure of such a relation is a well-founded partial order, since antisymmetry is a consequence of property (iii) in the definition of extended hierarchical automata. and the bottom elements are those X such that $\rho\sigma_X = \emptyset$ [2].

In the sequel we shall often abbreviate

$$\bigcup_{A' \in \left(\bigcup_{s \in \sigma_A} (\rho_A s)\right)} \cdots$$

with the more compact (although not strictly type-correct) $\bigcup_{A' \in (\rho\sigma_A)} \ldots$.

Proposition 4 *For all $L \in 2^{(\mathcal{T}\ H)}, \mathcal{C}', \mathcal{E}'$ the following holds:*
$A \uparrow P :: (\mathcal{C}, \mathcal{E}) \xrightarrow{L} (\mathcal{C}', \mathcal{E}') \Rightarrow ((\mathcal{C}' \in \mathrm{Conf}_A) \wedge (\mathcal{E}' \in (\Theta E)))$.

Proof By induction on the length d of the derivation for $A \uparrow P :: (\mathcal{C}, \mathcal{E}) \xrightarrow{L} (\mathcal{C}', \mathcal{E}')$.

Base case ($d = 1$):
If the derivation has length 1, then only the Progress Rule or the Stuttering Rule could have been applied. In both cases the assert follows directly from the consequent of the rule.
Induction step ($d > 1$):
In this case the Composition Rule must have been applied in the last step of the derivation. Each $\mathcal{C}_j$ belongs to a step-transition which is proven by a sub-derivation of length strictly less than d, so, by the Induction Hypothesis, it is a configuration of A_j; consequently $\{s\} \cup \bigcup_{j=1}^{n} \mathcal{C}_j \in \mathrm{Conf}_A$. □

Lemma 9 *For all $L \in 2^{(\mathcal{T}\ H)}, t \in L$ the following holds:*
$A \uparrow P :: (\mathcal{C}, \mathcal{E}) \xrightarrow{L} \Rightarrow \nexists t' \in P.\ \pi t \sqsubset \pi t'$

Proof By induction on the length d of the derivation for $A \uparrow P :: (\mathcal{C}, \mathcal{E}) \xrightarrow{L} (\mathcal{C}', \mathcal{E}')$.

Base case ($d = 1$):
If the derivation has length 1, then only the Progress Rule or the Stuttering Rule could have been applied. In the second case the assert follows trivially since $L = \emptyset$. In the first case the assert follows from (2) of the Progress Rule.
Induction step ($d > 1$):
In this case the Composition Rule must have been applied in the last step of the derivation. If $\bigcup_{j=1}^{n} L_j = \emptyset$ then the assert follows trivially. Suppose then that $\bigcup_{j=1}^{n} L_j \neq \emptyset$, and, w.l.g., let $t \in L_j$ with $A_j \uparrow P' :: (\mathcal{C}, \mathcal{E}) \xrightarrow{L_j} (\mathcal{C}_j, \mathcal{E}_j)$, according to (3) of the Composition Rule. The following steps prove the assert:

$t \in L_j \wedge A_j \uparrow P' :: (\mathcal{C}, \mathcal{E}) \xrightarrow{L_j} (\mathcal{C}_j, \mathcal{E}_j)$

$\Rightarrow$ {Induction Hypothesis}

$\nexists t' \in P \cup LE_A\ \mathcal{C}\ \mathcal{E}.\ \pi t \sqsubset \pi t'$

$\Rightarrow$ {Set Theory}

$\nexists t' \in P.\ \pi t \sqsubset \pi t'$ $\square$

Lemma 10 *(i)* $E_A\ \mathcal{C}\ \mathcal{E} \subseteq (\mathcal{T}\ A)$*; (ii)* $E_A\ \mathcal{C}\ \mathcal{E} = LE_A\ \mathcal{C}\ \mathcal{E} \cup \left(\bigcup_{A' \in \rho\sigma_A} E_{A'}\ \mathcal{C}\ \mathcal{E}\right)$

Proof

(i):

$t \in E_A\ \mathcal{C}\ \mathcal{E}$

$\Rightarrow$ {definition of E_A}

$t \in \bigcup_{A' \in (\mathcal{A}\ A)} LE_{A'}\ \mathcal{C}\ \mathcal{E}$

$\Rightarrow$ {definition of $LE_{A'}$}

$t \in \bigcup_{A' \in (\mathcal{A}\ A)} \delta_{A'}$

$\Rightarrow$ {definition of $\mathcal{T}\ A'$}

$t \in \bigcup_{A' \in (\mathcal{A}\ A)} \mathcal{T}\ A'$

$\Rightarrow$ {Lemma 1 (i); Set Theory}

$t \in \mathcal{T}\ A$

(ii):

$E_A\ \mathcal{C}\ \mathcal{E}$

$=$ {definition of E_A}

$\bigcup_{A' \in (\mathcal{A}\ A)} LE_{A'}\ \mathcal{C}\ \mathcal{E}$

$=$ {definition of $\mathcal{A}\ A$; Set Theory}

$LE_A\ \mathcal{C}\ \mathcal{E} \cup \left(\bigcup_{A' \in (\mathcal{A}\ A) \setminus \{A\}} LE_{A'}\ \mathcal{C}\ \mathcal{E}\right)$

$=$ {definition of $\mathcal{A}\ A$}

$LE_A\ \mathcal{C}\ \mathcal{E} \cup \left(\bigcup_{A' \in \bigcup_{A'' \in \rho_A\sigma_A} (\mathcal{A}\ A'')} LE_{A'}\ \mathcal{C}\ \mathcal{E}\right)$

$=$ {Set Theory}

$LE_A\ \mathcal{C}\ \mathcal{E} \cup \left(\bigcup_{A'' \in \rho_A\sigma_A} \bigcup_{A' \in (\mathcal{A}\ A'')} LE_{A'}\ \mathcal{C}\ \mathcal{E}\right)$

$=$ {definition of $E_{A''}$}

$LE_A\ \mathcal{C}\ \mathcal{E} \cup \left(\bigcup_{A'' \in \rho_A\sigma_A} E_{A''}\ \mathcal{C}\ \mathcal{E}\right)$ $\square$

Lemma 11 *For $s \in \mathcal{C} \cap \sigma_A$ and $A_j \in \rho_A s$ the following holds: $E_{A_j}\ \mathcal{C}\ \mathcal{E} = (E_A\ \mathcal{C}\ \mathcal{E}) \cap (\mathcal{T}\ A_j)$*

Proof

Part 1 - $E_{A_j}\ \mathcal{C}\ \mathcal{E} \subseteq (E_A\ \mathcal{C}\ \mathcal{E}) \cap \mathcal{T}\ A_j$.

$t \in E_{A_j}\ \mathcal{C}\ \mathcal{E}$

$\Rightarrow$ {Lemma 10 (i); Set Theory}

$t \in (E_{A_j}\ \mathcal{C}\ \mathcal{E}) \cap (\mathcal{T}\ A_j)$

$\Rightarrow$ {definition of E_{A_j}}

$t \in \left(\bigcup_{A' \in (\mathcal{A}\ A_j)} LE_{A'}\ \mathcal{C}\ \mathcal{E}\right) \cap (\mathcal{T}\ A_j)$

$\Rightarrow$ {Lemma 1 (i)}

$t \in \left(\bigcup_{A' \in (\mathcal{A}\ A)} LE_{A'}\ \mathcal{C}\ \mathcal{E}\right) \cap (\mathcal{T}\ A_j)$

$\Rightarrow$ {definition of E_A}

$t \in (E_A\ \mathcal{C}\ \mathcal{E}) \cap (\mathcal{T}\ A_j)$

Part 2 - $(E_A\ \mathcal{C}\ \mathcal{E}) \cap \mathcal{T}\ A_j \subseteq E_{A_j}\ \mathcal{C}\ \mathcal{E}$.

$t \in (E_A\ \mathcal{C}\ \mathcal{E}) \cap \mathcal{T}\ A_j$

$\Rightarrow$ {definition of E_A}

$t \in \left(\bigcup_{A' \in (\mathcal{A}\ A)} LE_{A'}\ \mathcal{C}\ \mathcal{E}\right) \cap (\mathcal{T}\ A_j)$

$\Rightarrow$ {definition of $\mathcal{A}\ A$;Set Theory}

$t \in \left(LE_A\ \mathcal{C}\ \mathcal{E} \cup \left(\bigcup_{A' \in (\mathcal{A}\ A_j)} LE_{A'}\ \mathcal{C}\ \mathcal{E}\right) \cup \left(\bigcup_{A' \in \bigcup A'' \in (\rho_A \sigma_A \setminus \{A_j\})(\mathcal{A}\ A'')} LE_{A'}\ \mathcal{C}\ \mathcal{E}\right)\right) \cap (\mathcal{T}\ A_j)$

$\Rightarrow$ {$\delta_A \cap \mathcal{T}\ A_j = \emptyset \Rightarrow (LE_A\ \mathcal{C}\ \mathcal{E}) \cap \mathcal{T}\ A_j = \emptyset$}

$t \in \left(\left(\bigcup_{A' \in (\mathcal{A}\ A_j)} LE_{A'}\ \mathcal{C}\ \mathcal{E}\right) \cup \left(\bigcup_{A' \in \bigcup A'' \in (\rho_A \sigma_A \setminus \{A_j\})(\mathcal{A}\ A'')} LE_{A'}\ \mathcal{C}\ \mathcal{E}\right)\right) \cap (\mathcal{T}\ A_j)$

$\Rightarrow$ {Lemmata 10 and 1 (ii) imply $(LE_{A'}\ \mathcal{C}\ \mathcal{E}) \cap (\mathcal{T}\ A_j) = \emptyset$ for $A' \in (\mathcal{A}\ A'')$ with $A'' \in (\rho_A \sigma_A \setminus \{A_j\})$ }

$t \in \left(\bigcup_{A' \in (\mathcal{A}\ A_j)} LE_{A'}\ \mathcal{C}\ \mathcal{E}\right) \cap (\mathcal{T}\ A_j)$

$\Rightarrow$ {Lemmata 10 and 1 (i) and $A' \in (\mathcal{A}\ A_j)$ imply $(LE_{A'}\ \mathcal{C}\ \mathcal{E}) \cap (\mathcal{T}\ A_j) = LE_{A'}\ \mathcal{C}\ \mathcal{E}$}

$t \in \bigcup_{A' \in (\mathcal{A}\ A_j)} LE_{A'}\ \mathcal{C}\ \mathcal{E}$

$\Rightarrow$ {definition of E_{A_j}}

$t \in E_{A_j}\ \mathcal{C}\ \mathcal{E}$ $\square$

Remark 1 *An obvious consequence of the above lemma is that* $E_{A_j}\ \mathcal{C}\ \mathcal{E} \subseteq E_A\ \mathcal{C}\ \mathcal{E}$

Theorem 1 *For all* $L \subseteq (\mathcal{T}\ A)$, $A \uparrow P :: (\mathcal{C}, \mathcal{E}) \xrightarrow{L}$ *if and only if* L *is a maximal set, under set inclusion, which satisfies all the following properties: (i)* L *is conflict-free, i.e.* $\forall t, t' \in L.\ \neg t \# t'$*); (ii) all transitions in* L *are enabled in the current status, i.e.* $L \subseteq E_A\ \mathcal{C}\ \mathcal{E}$ *; (iii) there is no transition outside* L *which is enabled in the current status and which has higher priority than a transition in* L*, i.e.* $\forall t \in L.\ \nexists t' \in E_A\ \mathcal{C}\ \mathcal{E}.\ \pi t \sqsubset \pi t'$*; and (iv) all transitions in* L *respect* P*, i.e.* $\forall t \in L.\ \nexists t' \in P.\ \pi t \sqsubset \pi t'$

Proof We prove the two implications separately.
Part 1 - direct implication: if L is a maximal set which satisfies properties (i) to (iv) then $A \uparrow P :: (\mathcal{C}, \mathcal{E}) \xrightarrow{L}$. We proceed by induction on the structure of $F_{\mathcal{C}}$.

Base case ($\rho_A\ s = \emptyset$):
If $L = \emptyset$ then the conditions for the Stuttering Rule are fulfilled, if instead $L \neq \emptyset$ then the conditions for the Progress Rule are fulfilled and the assert follows.

Induction step ($\rho_A\ s \neq \emptyset$):
There are two cases depending on $\exists t \in L.\ t \in \delta_A$:

Case 1 - $\exists t \in L.\ t \in \delta_A$
Notice first of all that, in order for L to be conflict free, in this case it must also be $L = \{t\}$, as it follows from the definitions of configuration and of conflicting transitions. Moreover, the hypothesis on L make the conditions for the application of the Progress Rule fulfilled for t and the assert follows.

Case 2 - $\nexists t \in L.\ t \in \delta_A$
Let $L_j = L \cap (\mathcal{T}\ A_j)$ with $\{A_1, \ldots, A_n\} = \rho s$. We prove that each L_j is a maximal set which satisfies (i) to (iv) w.r.t. A_j and $P \cup LE_A\ \mathcal{C}\ \mathcal{E}$. We first prove that L_j satisfies (i) to (iv).

(i):

$L_j \subseteq L$

$\Rightarrow$ $\{L$ sat. (i)$\}$

$L_j sat.(i)$

(ii):

$t \in L_j$

$\Rightarrow$ {definition of L_j}

$t \in L \cap (\mathcal{T}\ A_j)$

$\Rightarrow$ $\{L$ sat. (ii)$\}$

$t \in (E_A\ \mathcal{C}\ \mathcal{E}) \cap (\mathcal{T}\ A_j)$

$\Rightarrow$ {Lemma 11}

$t \in E_{A_j}\ \mathcal{C}\ \mathcal{E}$

(iii) By contradiction:

$t \in L_j \wedge t' \in E_{A_j}\ \mathcal{C}\ \mathcal{E} \wedge \pi t \sqsubset \pi t'$

$\Rightarrow$ $\{L_j \subseteq L$ and Remark 1$\}$

$t \in L \wedge t' \in E_A\ \mathcal{C}\ \mathcal{E} \wedge \pi t \sqsubset \pi t'$

$\Rightarrow$ $\{L$ sat. (iii) w.r.t $A\}$

FALSE

(iv) By contradiction:

Suppose $t \in L_j \wedge t' \in (P \cup LE_A\ \mathcal{C}\ \mathcal{E}) \wedge \pi t \sqsubset \pi t'$. Either $t' \in P$ or $t' \in LE_A\ \mathcal{C}\ \mathcal{E}$:

Case 1 - $t' \in P$

$t \in L_j \wedge t' \in P \wedge \pi t \sqsubset \pi t'$

$\Rightarrow$ $\{L_j \subseteq L\}$

$t \in L \wedge t' \in P \wedge \pi t \sqsubset \pi t'$

$\Rightarrow$ $\{L$ sat. (iv) w.r.t $P\}$

FALSE

Case 2 - $t' \in LE_A\ \mathcal{C}\ \mathcal{E}$

$t \in L_j \wedge t' \in LE_A\ \mathcal{C}\ \mathcal{E} \wedge \pi t \sqsubset \pi t'$

$\Rightarrow$ $\{L_j \subseteq L\}$

$t \in L \wedge t' \in LE_A\ \mathcal{C}\ \mathcal{E} \wedge \pi t \sqsubset \pi t'$

$\Rightarrow$ {Lemma 10 (ii)}

$t \in L \wedge t' \in E_A\ \mathcal{C}\ \mathcal{E} \wedge \pi t \sqsubset \pi t'$

$\Rightarrow$ $\{L$ sat. (iii) w.r.t $A\}$

FALSE

Now we prove that L_j is maximal. We proceed again by contradiction. Suppose there exists L'_j such that $L_j \subset L'_j$ and L'_j satisfies (i) to (iv) w.r.t. A_j and $P \cup LE_A\ \mathcal{C}\ \mathcal{E}$. Let $L' = L'_j \cup \bigcup_{k=1,k\neq j}^{k=n} L_k$. Clearly $L \subset L'$ and in the following we show that L' would satisfy (i) to (iv) w.r.t. A and P.

(i):

TRUE

$\Rightarrow$ $\{L'_j$ sat. (ii) w.r.t. A_j;Lemma 10 (i)$\}$

$L'_j \subseteq (\mathcal{T}\ A_j)$

$\Rightarrow$ {Composition Rule; definition of $\mathcal{T}$, ||; Lemma 7}

$\forall t \in L'_j, t' \in L_k.\ \neg(t \# t')$

$\Rightarrow$ {definition of L'; L'_j and L_k sat. (i)}

$L' sat. (i)$

(ii):

$L' \subseteq E_A\ \mathcal{C}\ \mathcal{E}$ since L'_j sat. (ii) w.r.t. A_j and Lemma 10 (ii) applies; moreover $\bigcup_{k \neq j} L_k \subseteq L$ and L sat. (i) w.r.t. A.

(iii):

First notice that

1. $\forall t \in L_k.\ \nexists t' \in E_A\ \mathcal{C}\ \mathcal{E}.\ \pi t \sqsubset \pi t'$ for $k \neq j$ since $L_k \subseteq L$ which sat. (iii) w.r.t. A.
2. $\forall t \in L'_j.\ \nexists t' \in E_{A_j}\ \mathcal{C}\ \mathcal{E}.\ \pi t \sqsubset \pi t'$ since L'_j sat. (iii) w.r.t.A_j
3. $\forall t \in L'_j.\ \nexists t' \in E_{A_k}\ \mathcal{C}\ \mathcal{E}.\ \pi t \sqsubset \pi t'$ for $k \neq j$ because of Lemma 8
4. $\forall t \in L'_j.\ \nexists t' \in LE_A\ \mathcal{C}\ \mathcal{E}.\ \pi t \sqsubset \pi t'$ since L'_j sat. (iv) w.r.t. $P \cup LE_A\ \mathcal{C}\ \mathcal{E}$.

Points (1) to (4) above, together with Lemma 10 (ii) prove that L' satisfies (iii) w.r.t. A.

(iv):

$\forall t \in L'.\ \nexists t' \in P.\ \pi t \sqsubset \pi t'$ since L sat. (iv) w.r.t. P and L'_j sat. (iv) w.r.t. $P \cup LE_A\ \mathcal{C}\ \mathcal{E}$.
All the above shows that L' would satisfy (i) to (iv) w.r.t. A and P and $L \subset L'$ which would contradict the hypothesis on L.

So, by the Induction Hypothesis $\bigwedge_{j=1}^{n} A_j \uparrow P \cup LE_A\ \mathcal{C}\ \mathcal{E} :: (\mathcal{C}, \mathcal{E}) \xrightarrow{L_j} (\mathcal{C}_j, \mathcal{E}_j)$. Moreover, if $\bigcup_{j=1}^{n} L_j = \emptyset$ then $L = \emptyset$ which, by hypothesis on L implies in turn $\forall t \in LE_A\ \mathcal{C}\ \mathcal{E}.\ \exists t' \in P.\ \pi t \sqsubset \pi t'$. This means that the Composition Rule can be applied and the assert is proven.

Part 2 - reverse implication: if $A \uparrow P :: (\mathcal{C}, \mathcal{E}) \xrightarrow{L}$ then L is a maximal set which satisfies properties (i) to (iv). We proceed by induction on the length d of the derivation for $A \uparrow P :: (\mathcal{C}, \mathcal{E}) \xrightarrow{L} (\mathcal{C}', \mathcal{E}')$.

Base case ($d = 1$):
If the derivation has length 1, then only the Progress Rule or the Stuttering Rule could have been applied. In both cases the assert follows directly from the conditions of the rules.
Induction step ($d > 1$):
In this case the Composition Rule must have been applied in the last step of the derivation. By the Induction Hypothesis, every L_j satisfies (i) to (iv) w.r.t. A_j and $P \cup LE_A\ \mathcal{C}\ \mathcal{E}$. We show that $L = \bigcup_{j=1}^{n} L_j$ is a maximal set which satisfies (i) to (iv) w.r.t. A and P.

We first show that L satisfies (i) to (iv) w.r.t. A and P.

(i): L sat. (i) since each L_j sat. (i) by Induction Hypothesis and for $i \neq j$ if $t \in L_i$ and $t' \in L_j$, then $\neg(t \# t')$ because of $(SRC\ t \parallel SRC\ t')$ and Lemma 7.

(ii):

TRUE

$\Rightarrow$ $\{L_j$ sat. (ii) w.r.t. $A_j\}$

$\bigwedge_{j=1}^{n} L_j \subseteq E_{A_j}\ \mathcal{C}\ \mathcal{E}$

$\Rightarrow$ $\{$Remark 1; $L = \bigcup_{j=1}^{n} L_j\}$

$L \subseteq E_A\ \mathcal{C}\ \mathcal{E}$

(iii)By contradiction:

Suppose w.l.g. $t \in L_j$ and $t' \in E_A\ \mathcal{C}\ \mathcal{E}$ with $\pi t \sqsubset \pi t'$. By Lemma 10 (ii) either $t' \in LE_A\ \mathcal{C}\ \mathcal{E}$ or $t' \in E_{A_k}\ \mathcal{C}\ \mathcal{E}$ for some k such that $A_k \in \rho s$.

Case 1 - $t' \in LE_A\ \mathcal{C}\ \mathcal{E}$

$t \in L \wedge t' \in LE_A\ \mathcal{C}\ \mathcal{E} \wedge \pi t \sqsubset \pi t'$

$\Rightarrow$ $\{$Composition Rule$\}$

$t \in L_j \wedge t' \in LE_A\ \mathcal{C}\ \mathcal{E} \wedge A_j \uparrow P \cup LE_A\ \mathcal{C}\ \mathcal{E} :: (\mathcal{C}, \mathcal{E}) \xrightarrow{L_j} \wedge \pi t \sqsubset \pi t'$

$\Rightarrow$ $\{$Lemma 9$\}$

FALSE

Case 2 - $t' \in E_{A_k}\ \mathcal{C}\ \mathcal{E}$ for some k such that $A_k \in \rho s$.

Notice first that it cannot be $k = j$ since L_j is conflict free and $\pi t \sqsubset \pi t'$ implies $t \# t'$ by definition of priority schema. On the other hand, it cannot be $k \neq j$ since $\pi t \sqsubset \pi t'$ would violate Lemma 8.

(iv) By contradiction:

$t \in L \wedge t' \in P \wedge \pi t \sqsubset \pi t'$

$\Rightarrow$ $\{$Composition Rule and Set Theory$\}$

$\exists Lj.\ t \in L_j \wedge t' \in P \cup LE_A\ \mathcal{C}\ \mathcal{E} \wedge \pi t \sqsubset \pi t'$

$\Rightarrow$ $\{L_j$ sat. (iv) w.r.t. $P \cup LE_A\ \mathcal{C}\ \mathcal{E}\}$

FALSE

Now we prove that L is maximal. We proceed again by contradiction. Suppose there exists L' such that $L \subset L'$ and L' satisfies (i) to (iv) w.r.t. A and P. First observe that $L' \subseteq \bigcup_{j=1}^{n} (\mathcal{T}\ A_j)$ in fact:

TRUE

$\Rightarrow$ $\{L \subseteq L'$, L' sat. (i), and L' sat. (ii)$\}$

$L' \cap (LE_A\ \mathcal{C}\ \mathcal{E}) = \emptyset \wedge L' \subseteq E_A\ \mathcal{C}\ \mathcal{E}$

$\Rightarrow$ $\{$Lemma 10 (ii)$\}$

$L' \subseteq \bigcup_{j=1}^{n} E_{A_j}\ \mathcal{C}\ \mathcal{E}$

$\Rightarrow$ {Lemma 10 (i)}

$L' \subseteq \bigcup_{j=1}^{n} (\mathcal{T}\ A_j)$

Let, for $k = 1, \ldots n$ $L'_k = L' \cap (\mathcal{T}\ A_k)$. Obviously there must exist a j such that $L_j \subset L'_j$ since $\bigcup_{j=1}^{n} L_j = L \subset L'$. We show that L'_j satisfies (i) to (iv) w.r.t. A_j and $P \cup LE_A\ \mathcal{C}\ \mathcal{E}$ which contradicts the Induction Hypothesis (maximality of L_j).

(i):

$L'_j \subseteq L'$

$\Rightarrow$ {L' sat. (i)}

$L'_j\ sat.\ (i)$

(ii):

$t \in L'_j$

$\Rightarrow$ {definition of L'_j}

$t \in L' \cap (\mathcal{T}\ A'_j)$

$\Rightarrow$ {L' sat. (ii)}

$t \in (E_A\ \mathcal{C}\ \mathcal{E}) \cap (\mathcal{T}\ A_j)$

$\Rightarrow$ {Lemma 11}

$t \in E_{A_j}\ \mathcal{C}\ \mathcal{E}$

(iii) By contradiction:

$t \in L'_j \wedge t' \in E_{A_j}\ \mathcal{C}\ \mathcal{E} \wedge \pi t \sqsubset \pi t'$

$\Rightarrow$ {$L'_j \subseteq L'$ and Remark 1}

$t \in L' \wedge t' \in E_A\ \mathcal{C}\ \mathcal{E} \wedge \pi t \sqsubset \pi t'$

$\Rightarrow$ {L' sat. (iii)}

FALSE

(iv) By contradiction:

Suppose $t \in L'_j \wedge t' \in (P \cup LE_A\ \mathcal{C}\ \mathcal{E}) \wedge \pi t \sqsubset \pi t'$. Either $t' \in P$ or $t' \in LE_A\ \mathcal{C}\ \mathcal{E}$:

Case 1 - $t' \in P$

$t \in L'_j \wedge t' \in P \wedge \pi t \sqsubset \pi t'$

$\Rightarrow$ {$L'_j \subseteq L'$}

$t \in L' \wedge t' \in P \wedge \pi t \sqsubset \pi t'$

$\Rightarrow$ {L' sat. (iv) w.r.t P}

FALSE

Case 2 - $t' \in LE_A\ \mathcal{C}\ \mathcal{E}$

$t \in L'_j \wedge t' \in LE_A\ \mathcal{C}\ \mathcal{E} \wedge \pi t \sqsubset \pi t'$

$\Rightarrow$ $\{L'_j \subseteq L'\}$

$t \in L \wedge t' \in LE_A\ \mathcal{C}\ \mathcal{E} \wedge \pi t \sqsubset \pi t'$

$\Rightarrow$ {Lemma 10 (ii)}

$t \in L' \wedge t' \in E_A\ \mathcal{C}\ \mathcal{E} \wedge \pi t \sqsubset \pi t'$

$\Rightarrow$ $\{L'$ sat. (iii) w.r.t $A\}$

FALSE

So L is maximal. □

References

[1] D. Latella, M. Massink, and I. Majzik. A Simplified Formal Semantics for a Subset of UML Statechart Diagrams. Technical Report HIDE/T1.2/PDCC/5/v1, ESPRIT Project n. 27439 - High-Level Integrated Design Environemnt for Dependability HIDE, 1998. Available in the HIDE Project Public Repository (https://asterix.mit.bme.hu:998/).

[2] Z. Manna, S. Ness, and J. Vuillemin. Inductive methods for proving properties of programs. *Communications of the ACM*, 16(8), 1973.

[3] R. Milner. *Communication and Concurrency*. Series in Computer Science. Prentice Hall, 1989.

Quantitative assessment of the translation from Statechart Diagrams to Kripke Structures

E. Giusti - CPR PDCC
D. Latella - CPR PDCC and CNR Ist. CNUCE
M. Massink - CNR Ist. CNUCE

1 Rationale

During the first phase of the HIDE Project the formal definition of the translation from UML statechart diagrams to Kripke Structures have beed fully developed (see Deliverable 2) and its correctness has been formally proven (see Deliverable 4).

Such a definition is to be considered as a specification for the translation itself, to which any implementation of such a translation needs to conform.

No implementation of the translation has been developed during the first phase of HIDE for which a conformance proof is available.

So no definite quantitative assessment which refers to implementation issues, like memory usage or execution time can be done at this stage of development of the work.

The major consideration which can be done at this moment is that the translation from statechart diagrams to Kripke Structures is a way for providing a formal semantics for UML statechart diagrams within the HIDE environment. The definition of the semantics is a necessary pre-requisite for any formal verification activity and the formal definition of the possible behaviours (i.e. the semantics) is the minimal information one needs for understanding the statechart diagram.

2 Outline of an Extended Hierarchical Automata to PROMELA translation

At a more concrete level, it is worth mentioning that there is work ongoing in PDCC for implementing a translation from Extended Hierarchical Automata to the PROMELA language [1]. The implementation is still in progress. The implementation language we chose is ML, due to its sound mathematical basis which hopefully will help in developing a conformance proof.

PROMELA is the modeling language of the SPIN model-checker, which is a widely used tool specially due to the various state compression and efficiency enhancement techniques it includes.

The main idea of our implementation effort is to translate an extended hierarchical automaton into a PROMELA model which, when executed by SPIN, generates as statespace essentially the Kripke Structure which is the semantic model, via our operational semantics, of the extended hierarchical automaton.

Experiments with classical statecharts [2] show that the translation is feasible and the results, in terms of memory usage and execution time, are encouraging.

The above facts bring us to formulate a strong conjecture that a PROMELA representation of extended hierarchical automata, as we are defining it can be a very efficient one.

We conclude this section showing what would be the result of applying the PROMELA translation to the extended hierarchical automaton of Fig. 1

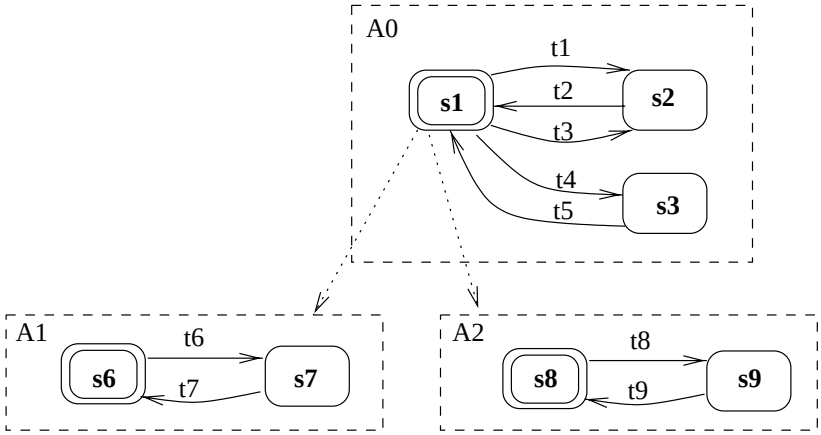

Figure 1: Example of an Extended Hierarchical Automaton

The code is given below. There is a distinct boolean variable Sn for each state sn. Moreover, set P (see deliverable 2) is represented by its characteristic function coded as 9 boolean variables, one per each transition. Variable Pn is referred to transition tn. Transition tn is fired iff the corresponding variable $Firen$ evaluates to 1. The program essentially marks those transitions which are enabled and respect the priority constraints. This labeling is made via variables $Seln$. On the basis of those transition which are enabled and which respect priority (i.e. those tn such that $Seln$ is 1) and using the PROMELA non-deterministic construct, the transitions to be fired are selected (i.e. their corresponding variable $Firen$ is set to 1). The fact that transition i has lower priority than transition j is coded via function SM, i.e SM(i,j) evaluates to 1. Otherwise SM(i,j) evaluates to 0. Finally, no parallelism is used in the PROMELA model since, given the absence of synchronization and the fact that there is no particular order in which parallel automata have to be considered, such parallelism would only result in an increase of the state space.

Finally, for the sake of simplicity and limited to the present example, we have chosen to represent the event queue as an integer. This is because we know, a priori, that the queue will never contain more than one item in this example.

In the actual code below, the comments often refer to the specific premisses of the operational semantics rules (e.g. PROG. 1 is condition (1) of the progress rule).

```
bit S1, S2, S3, S6, S7, S8, S9;
bit P1, P2, P3, P4, P5, P6, P7, P8, P9;
bit Sel1, Sel2, Sel3, Sel4, Sel5, Sel6, Sel7, Sel8, Sel9;
bit Fire1, Fire2, Fire3, Fire4, Fire5, Fire6, Fire7, Fire8, Fire9;
```

```
#define SM(x,y)  \
(x==3 & y==1) || \
(x==3 & y==4) || \
(x==3 & y==6) || \
(x==3 & y==7) || \
(x==3 & y==8) || \
(x==3 & y==9)

# define no_event 0
# define a1 1
# define a2 2
# define e1 3
# define e2 4
# define f1 5
# define f2 6
# define r1 7
# define r2 8

int Ev

proctype STEP()
{
do
::
atomic
{
/* BEGIN PROG. FOR A0 */
/* BEGIN SELECTION TEST FOR TRANSITION 1 */
Sel1=
/* BEGIN PROG. 1 */
S1 & S6 & (Ev==r1)
/* END PROG. 1 */
&
/* BEGIN PROG. 2 */
!(P3 & SM(1,3))&
!(P4 & SM(1,4))&
!(P6 & SM(1,6))&
!(P7 & SM(1,7))&
!(P8 & SM(1,8))&
!(P9 & SM(1,9))
&
!((Ev==e1) & SM(1,3))&
!(S8 & (Ev==r2) & SM(1,4))
&
!(S6 & (Ev==e1) & SM(1,6))&
!(S7 & (Ev==f1) & SM(1,7))&
!(S8 & (Ev==e2) & SM(1,8))&
```

```
!(S9 & (Ev==f2) & SM(1,9))
/* END PROG. 2 */
/* END SELECTION TEST FOR TRANSITION 1 */
;
/* BEGIN SELECTION TEST FOR TRANSITION 2 */
Sel2=
/* BEGIN PROG. 1 */
S2 & (Ev==a1)
/* END PROG. 1 */
/* NO PROG. 2 */
/* END SELECTION TEST FOR TRANSITION 2 */
;
/* BEGIN SELECTION TEST FOR TRANSITION 3 */
Sel3=
/* BEGIN PROG. 1 */
S1 & (Ev==e1)
/* END PROG. 1 */
&
/* BEGIN PROG. 2 */
!(P1 & SM(3,1))&
!(P4 & SM(3,4))&
!(P6 & SM(3,6))&
!(P7 & SM(3,7))&
!(P8 & SM(3,8))&
!(P9 & SM(3,9))
&
!(S6 & (Ev==r1) & SM(3,1))&
!(S8 & (Ev==r2) & SM(3,4))
&
!(S6 & (Ev==e1) & SM(3,6))&
!(S7 & (Ev==f1) & SM(3,7))&
!(S8 & (Ev==e2) & SM(3,8))&
!(S9 & (Ev==f2) & SM(3,9))
/* END PROG. 2 */
/* END SELECTION TEST FOR TRANSITION 3 */
;
/* BEGIN SELECTION TEST FOR TRANSITION 4 */
Sel4=
/* BEGIN PROG. 1 */
S1 & S8 & (Ev==r2)
/* END PROG. 1 */
&
/* BEGIN PROG. 2 */
!(P1 & SM(4,1))&
!(P3 & SM(4,3))&
!(P6 & SM(4,6))&
!(P7 & SM(4,7))&
```

```
!(P8 & SM(4,8))&
!(P9 & SM(4,9))
&
!(S6 & (Ev==r1) & SM(4,1))&
!((Ev==e1) & SM(4,3))
&
!(S6 & (Ev==e1) & SM(4,6))&
!(S7 & (Ev==f1) & SM(4,7))&
!(S8 & (Ev==e2) & SM(4,8))&
!(S9 & (Ev==f2) & SM(4,9))
/* END PROG. 2 */
/* END SELECTION TEST FOR TRANSITION 4 */
;
/* BEGIN SELECTION TEST FOR TRANSITION 5 */
Sel5=
/* BEGIN PROG. 1 */
S3 & (Ev==a2)
/* END PROG. 1 */
/* NO PROG. 2 */
/* END SELECTION TEST FOR TRANSITION 5 */
/* END PROG. FOR A0 */
;
/* BEGIN COM. FOR A0 */
/* BEGIN UPDATING P WITH ACTIVE TRANSITIONS OF A0 */
P1= S1 & S6 & (Ev==r1);
P2= S2 & (Ev==a1);
P3= S1 & (Ev==e1);
P4= S1 & S8 & (Ev==r2);
P5= S3 & (Ev==a2)
/* END UPDATING P WITH ACTIVE TRANSITIONS OF A0 */
;
/* BEGIN PROG. FOR A1 */
/* BEGIN SELECTION TEST FOR TRANSITION 6 */
Sel6=
/* BEGIN PROG. 1 */
S6 & (Ev==e1)
/* END PROG. 1 */
&
/* BEGIN PROG. 2 */
!(P1 & SM(6,1))&
!(P3 & SM(6,3))&
!(P4 & SM(6,4))
/* END PROG. 2 */
/* END SELECTION TEST FOR TRANSITION 6 */
;
/* BEGIN SELECTION TEST FOR TRANSITION 7 */
Sel7=
```

```
/* BEGIN PROG. 1 */
S7 & (Ev==f1)
/* END PROG. 1 */
&
/* BEGIN PROG. 2 */
!(P1 & SM(7,1))&
!(P3 & SM(7,3))&
!(P4 & SM(7,4))
/* END PROG. 2 */
/* END SELECTION TEST FOR TRANSITION 7 */
/* END PROG. FOR A1 */
;
/* BEGIN PROG. FOR A2 */
/* BEGIN SELECTION TEST FOR TRANSITION 8 */
Sel8=
/* BEGIN PROG. 1 */
S8 & (Ev==e2)
/* END PROG. 1 */
&
/* BEGIN PROG. 2 */
!(P1 & SM(8,1))&
!(P3 & SM(8,3))&
!(P4 & SM(8,4))
/* END PROG. 2 */
/* END SELECTION TEST FOR TRANSITION 8 */
;
/* BEGIN SELECTION TEST FOR TRANSITION 9 */
Sel9=
/* BEGIN PROG. 1 */
S9 & (Ev==f2)
/* END PROG. 1 */
&
/* BEGIN PROG. 2 */
!(P1 & SM(9,1))&
!(P3 & SM(9,3))&
!(P4 & SM(9,4))
/* END PROG. 2 */
/* END SELECTION TEST FOR TRANSITION 9 */
/* END PROG. FOR A2 */
;
/* BEGIN SELECTING TRANSITIONS TO FIRE */
if
::Sel1->Fire1=1
::Sel2->Fire2=1
::Sel3->Fire3=1
::Sel4->Fire4=1
::Sel5->Fire5=1
```

```
::(Sel6||Sel7||Sel8||Sel9)->
  if
  ::Sel6->Fire6=1
  ::Sel7->Fire7=1
  ::else->skip
  fi;
  if
  ::Sel8->Fire8=1
  ::Sel9->Fire9=1
  ::else->skip
  fi
::else->skip
fi
/* END SELECTING TRANSITIONS TO FIRE */
;
/* BEGIN FIRING SELECTED TRANSITIONS */
if ::Fire1->S1=0;S2=1;S3=0;S6=0;S7=0;S8=0;S9=0;Ev=a1 ::else->skip fi;
if ::Fire2->S1=1;S2=0;S3=0;S6=1;S7=0;S8=1;S9=0;Ev=r2 ::else->skip fi;
if ::Fire3->S1=0;S2=1;S3=0;S6=0;S7=0;S8=0;S9=0;Ev=no_event ::else->skip fi;
if ::Fire4->S1=0;S2=0;S3=1;S6=0;S7=0;S8=0;S9=0;Ev=a2 ::else->skip fi;
if ::Fire5->S1=1;S2=0;S3=0;S6=1;S7=0;S8=0;S9=1;Ev=e1 ::else->skip fi;
if ::Fire6->S6=0;S7=1;Ev=f1 ::else->skip fi;
if ::Fire7->S6=1;S7=0;Ev=r1 ::else->skip fi;
if ::Fire8->S8=0;S9=1;Ev=e1 ::else->skip fi;
if ::Fire9->S8=1;S9=0;Ev=no_event ::else->skip fi
/* END FIRING SELECTED TRANSITIONS */
;
/* BEGIN CLEANING-UP AUXILIARY STRUCTURES */
P1=0;P2=0;P3=0;P4=0;P5=0;P6=0;P7=0;P8=0;P9=0;
Sel1=0;Sel2=0;Sel3=0;Sel4=0;Sel5=0;Sel6=0;Sel7=0;Sel8=0;Sel9=0;
Fire1=0;Fire2=0;Fire3=0;Fire4=0;Fire5=0;Fire6=0;Fire7=0;Fire8=0;Fire9=0
/* END CLEANING-UP AUXILIARY STRUCTURES */
}
od
}

init
{
/* BEGIN SETTING INITIAL CONFIGURATION */
S1=1;S2=0;S3=0;S6=1;S7=0;S8=1;S9=0;
/* END SETTING INITIAL CONFIGURATION */
Ev=e2;
atomic{run STEP()}
}
```

References

[1] G Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295, 1997.

[2] E. Mikk, Y. Lakhnech, and M. Siegel. Hierarchical automata as model for statecharts. In R. Shyamasundar and K. Euda, editors, *Third Asian Computing Science Conference. Advances in Computing Sience - ASIAN'97*, volume 1345 of *Lecture Notes in Computer Science*, pages 181–196. Springer-Verlag, 1997.