AToMPM Solution for the Petri Net to Statecharts Case Study

Hüseyin Ergin and Eugene Syriani
University of Alabama, Tuscaloosa AL, U.S.A.
{hergin@crimson,esyriani@cs}.ua.edu

In this paper, we present an AToMPM solution for the Petri Net to Statecharts transformation case study. AToMPM provides a convenient graphical user interface for designing domain-specific models and executing model transformations.

1 Introduction

AToMPM [2] allows one to model and execute model transformations. It provides a graphical user interface to define the metamodels of the intended formalisms, describe rules graphically as well as a control structure for model transformations, and execute step-by-step transformations on given models.

MoTif [3] rules consist of a pre-condition and a post-condition. The pre-condition pattern determines the applicability of the rule and is usually defined with a left-hand side (LHS) and optional negative application conditions (NAC). The post-condition determines the result of the rule and is defined by a right-hand side (RHS) which must be satisfied after the rule is applied. The scheduling, or the control flow, describes the order in which the rules are executed. The rule in Fig. 1 is a MoTif rule with a NAC, LHS, and RHS (from left to right). The rule at left in Fig. 1 is the exact rule in AToMPM , the rule at right is the redrawing of the same rule for this paper to save space and to make rules cleaner to see.

In the scheduling part, each rule is represented by a rule block having three ports. Conceptually, a rule receives models via the input port at the top. If the rule is successfully applied, the resulting model is output from the success port at the bottom left. Otherwise, the model does not satisfy the pre-condition and the original model is output from the fail port at the bottom right. Fig. 9 depicts an example of control flow structure to schedule MoTif rules.

This paper provides a solution to the Petri Net to Statecharts model transformation case study, whose full description can be found at [1]. In Section 2, we provide the details about the solution. In Section 3, we list the findings from the study and conclude.

2 Solution

The solution follows the steps in the description document [1]. We first define the metamodels of Petri Nets (PN) and Statecharts (SC). Then we briefly explain the rules for initialization, reduction for AND,

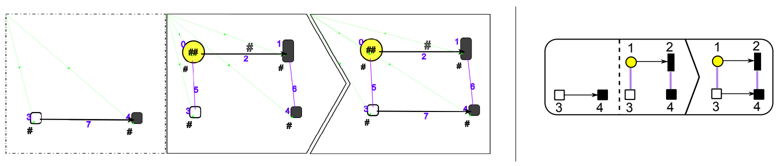

Figure 1: Original MoTif rule and its representation in this paper

Submitted to:
TTC 2013

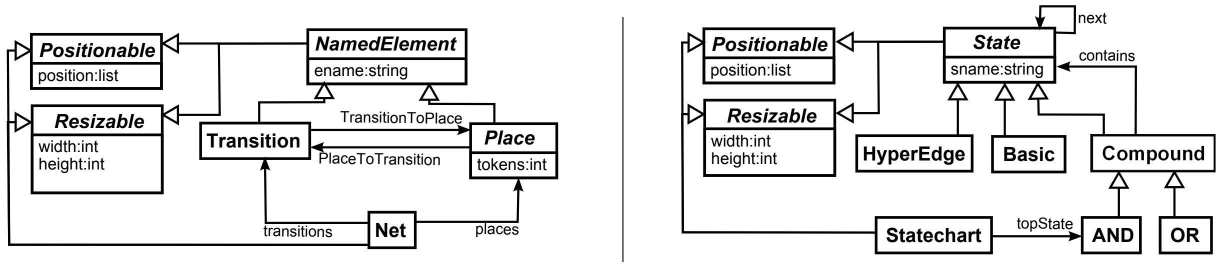

Figure 2: Petri net & Statecharts metamodels

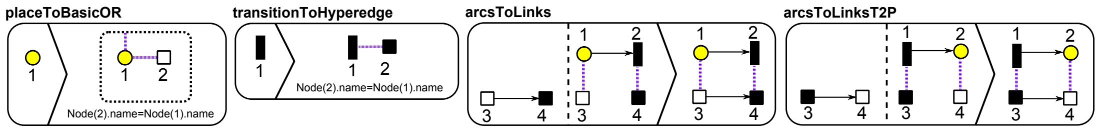

Figure 3: Initialization Phase Rules

and reduction for OR. Then we give the finishing rules which creates the main statechart and top AND state. Finally, we display the control flow of the phases and rules.

2.1 The Metamodels

The metamodels for PN and SC are exactly like in the description document. We just added *Positionable* and *Resizable* abstract classes as super class of the elements for the sake of nice visualization in AToMPM . These abstract classes help us to relocate and resize the newly created statechart elements by using the existing locations and sizes of Petrinet elements. The metamodels are depicted in Fig. 2.

2.2 Initialization

The rules for the initialization phase are depicted in Fig. 3. In this phase, the *placeToBasicOR* rule matches all *places* to a *basic state* and an *OR state*. *transitionToHyperedge* rule matches all *transitions* to a *hyperedge*. Then all the arcs from *place* to *transition* and vice-versa creates set of *next* links between *basic states* and *hyperedges* with the help of *arcsToLinks* and *arcsToLinksT2P* rules.

The purple lines in the rules represent the traceability links of our language. With the help of these links, we do not have to compute or keep track of the *equiv* function of *places*.

2.3 AND Reduction

After the initialization phase, the transformation starts the reduction phase. AND reduction is defined to create an *AND* state for a set of *places* that are connected to the same incoming and outgoing *transitions*. The rules are depicted in Fig. 4. The *selectTransitionAnd1* rule selects a *transition* that has more than one incoming *places*. Then, the *allIncomingPlacesHaveSameIncomingTransitions* rule eliminates the *transition* if any two incoming *places* do not have same incoming *transition*. The *allIncomingPlacesHaveSameOutgoingTransitions* rule does the same job for the outgoing *transitions* of any two *places*. These three rules do not have RHS which makes them *query* or *QRule*. *QRules* are used to query information from models or find a suitable match in the model for further processing. Different types of rules are explained in Section 2.6.

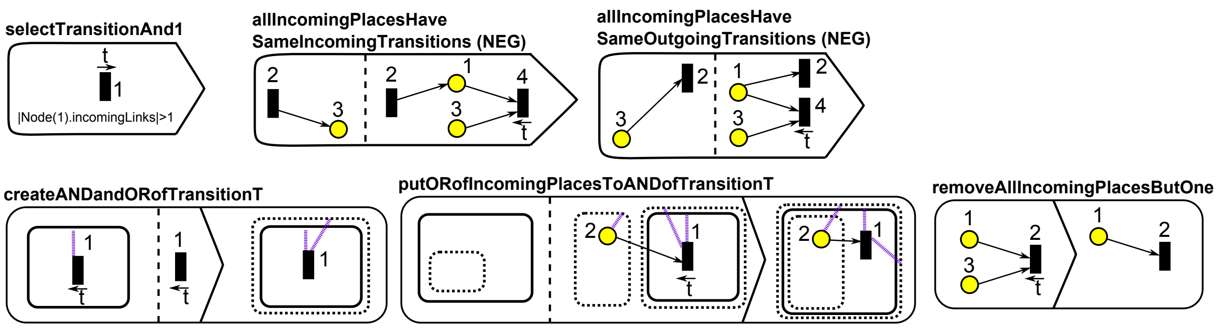
selectTransitionAnd1
|Node(1).incomingLinks|>1
allIncomingPlacesHave SameIncomingTransitions (NEG)
allIncomingPlacesHave SameOutgoingTransitions (NEG)
createANDandORofTransitionT
putORofIncomingPlacesToANDofTransitionT
removeAllIncomingPlacesButOne

Figure 4: AND Reduction Rules

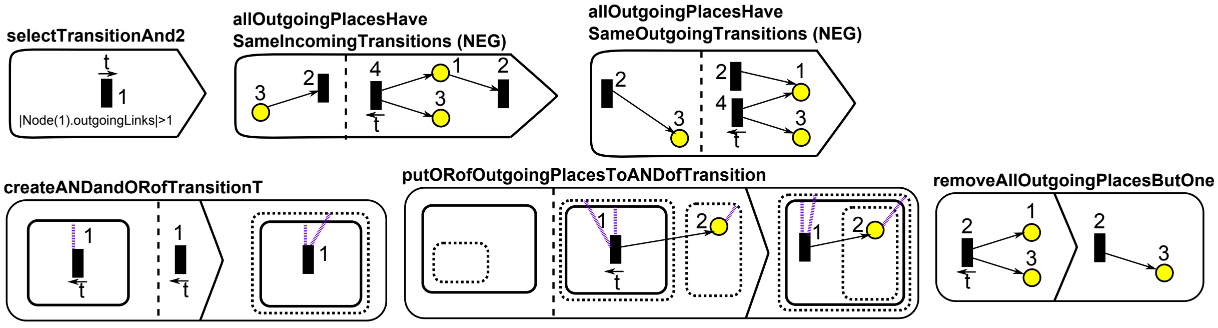
selectTransitionAnd2
|Node(1).outgoingLinks|>1
allOutgoingPlacesHave SameIncomingTransitions (NEG)
allOutgoingPlacesHave SameOutgoingTransitions (NEG)
createANDandORofTransitionT
putORofOutgoingPlacesToANDofTransition
removeAllOutgoingPlacesButOne

Figure 5: Second Version of AND Reduction Rules

After a suitable *transition* is found, it is passed to other rules. The *createANDandORofTransitionT* rule creates an inner *AND* and an outer *OR* for the selected *transition*. Then, the *putORofIncomingPlaces-ToANDofTransitionT* rule checks all incoming *places* of the selected *transition* and puts the equivalent *OR* of them to *AND* of this *transition*. Finally, all *places* are removed except one with the help of *removeAllIncomingPlacesButOne* rule.

Note the use of pivots in the different rules. A pivot identifies a model element to be bound by a rule and passed as parameter to other rules using it. Pivots are represented by a forward or backward arrow on top of the names of the element (*e.g.,* *transition* t in *selectTransitionForAnd1* rule). Forward arrow is used to set a pivot and backward arrow is used to access a pivot.

A second set of rules for AND reduction is the one which checks the *places* after a *transition*. The logic is the same. The rules are depicted in Fig. 5. These rules will find the AND reduction for the *transitions* that has more than one outgoing *places*.

2.4 OR Reduction

OR reduction is defined over a *transition* that has single incoming and outgoing *places*. The rules are depicted in Fig. 6. The *selectTransitionOr* rule selects the *transition* that satisfies this condition. Then for further conditions of OR reduction, the *noCommonTransitionOfInputAndOutputPlaces* rule ensures the incoming and outgoing *places* do not have a common incoming *transition* and the *noCommonTransitionOfInputAndOutputPlaces2* rule ensures they do not have a common outgoing transition.

We reuse the *OR* of the incoming *place* which is called ‘q’. So, the *putEachElementInORofRtoORofQ* rule puts all *basic states* and *hyperedges* in the *OR* of the outgoing *place* to the *OR* of ‘q’. Then, the

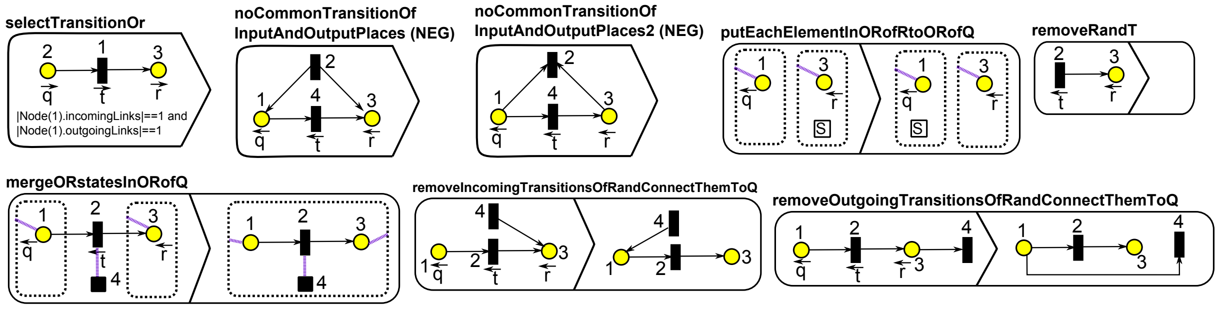
selectTransitionOr
|Node(1).incomingLinks|==1 and
|Node(1).outgoingLinks|==1
noCommonTransitionOf InputAndOutputPlaces (NEG)
noCommonTransitionOf InputAndOutputPlaces2 (NEG)
putEachElementInORofRtoORofQ
removeRandT
mergeORstatesInORofQ
removeIncomingTransitionsOfRandConnectThemToQ
removeOutgoingTransitionsOfRandConnectThemToQ

Figure 6: OR Reduction Rules

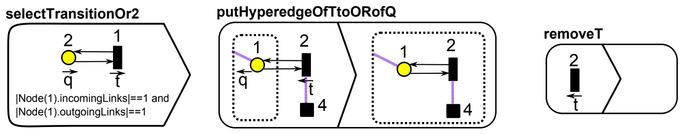

Figure 7: OR Reduction Rules for a Loop Case

mergeORstatesInORofQ rule merges the *OR* of incoming and outgoing *places* in *OR* of 'q'.

The next two rules, the *removeIncomingTransitionsOfRandConnectThemToQ* and the *removeOutgoing-TransitionsOfRandConnectThemToQ* rules, helps not to loose the existing connections of the outgoing *place* and connects them to 'q'. The *removeRandT* rule removes the *transition* and outgoing *place*.

Our *selectTransitionOr* rule will not find the matches where the incoming and outgoing *places* are the same, which produces a loop-like structure and is the situation in some of the test-cases. So, we add some extra rules for this case. The rules for this special case are depicted in Fig. 7. The *selectTransitionOr2* rule finds the matches for this situation. Since there is not a different outgoing place, all we need to do is put the *hyperedge* of *transition* to the *OR* of 'q' and remove *transition*.

2.5 Finish

The transformation starts with a *PetriNet* element with some *places* and *transitions* inside. At the end, this *PetriNet* is transformed into a *Statechart* element with another *AND* as *topState*. The rules are depicted in Fig. 8. The *createSCandAND* rule removes the *PetriNet* and creates *Statechart* with *AND*. Then, the *putOuterMostORsToTopstateAND* puts the *OR* states that are not contained in an *AND*, which means they are the outer most *OR* states, into the *topState AND*.

2.6 Control Flow

The overall control flow of the phases is depicted in Fig. 9. It starts with the *Initialization* phase, then tries

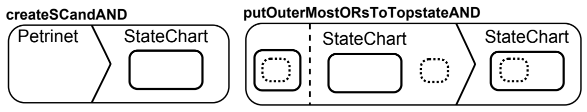

Figure 8: Finishing Rules

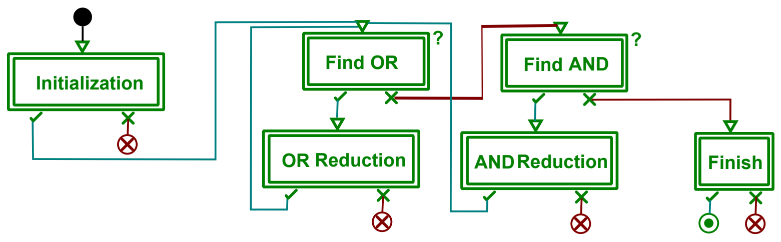

Figure 9: Overall Control Flow

to find AND or OR reductions. The reductions are applied as much as possible. If it cannot find any reduction, *Finish* rules work and finish the transformation.

The overall control flow in Fig. 9 is composed of references to expanded control flow to save space. The expanded control flow of the transformation is depicted in Appendix C. Some rule blocks are annotated, denoting a special behavior. The meaning of these rules are:

- *ARule*: is a regular atomic rule and is executed only once. It has no annotation.
- *FRule*: stands for "For all Rule". All matches are found for the input model once and this rule is applied to all matches found. It is annotated with an 'F'.
- *SRule*: stands for 'Star Rule'. It is a rule that is recursively applied on each match as long as matches are found. Therefore, the result of this rule is the cumulation of each application. It is annotated with a '*'.
- *QRule*: stands for 'Query Rule'. It is an *ARule* with no side effect since it does not have a RHS, but may still assign pivots. It is annotated with a '?'.

3 Conclusion

We applied the transformation to the eleven test-cases and successfully see the output Statecharts. Since AToMPM provides a graphical user interface, we were not able to run the transformation on the benchmark models. Writing an adapter that reads the ecore models and automate their creation in AToMPM requires also dealing with the geometrical dispositions of the Petri net elements, which we plan to incorporate soon. Furthermore, AToMPM does not yet provide a headless environment and runs completely online. Therefore the weak performance experienced is due to the GUI and network overheads. Hence this solution focuses on the expressiveness and usability power of modeling and transforming in AToMPM rather than its performance. The manual to reproduce the test-cases in an online AToMPM instance is provided in Appendix A. We also add another appendix for the simulation of both systems in Appendix B.

References

[1] Pieter Van Gorp & Louis M. Rose: *The Petri-Nets to Statecharts Transformation Case*. Available at `http://planet-sl.org/community/_/ttc/ttc2013/cases/PetriNetsToStateCharts`.

[2] Raphael Mannadiar (2012): *A Multi-Paradigm Modelling Approach to the Foundations of Domain-Specific Modelling*. Ph.d. thesis, McGill University.

[3] Eugene Syriani & Hans Vangheluwe (2011): *A Modular Timed Model Transformation Language*. Journal on Software and Systems Modeling 11, pp. 1–28.

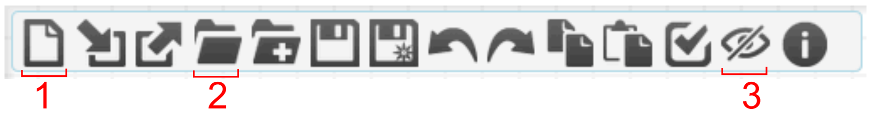

Figure 10: Main Toolbar

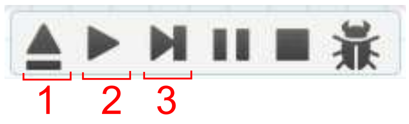

Figure 11: Transformation Controller Toolbar

A Appendix: Manual to Reproduce Test Cases

In this section, we introduce how to use AToMPM and how to reproduce test-cases. AToMPM can be reached via the SHARE virtual machine *XP-TUe_PN2SC_AToMPM.vdi*. AToMPM is a browser-based application. You will find the shortcut on desktop named *AToMPM Shortcut*. It opens AToMPM environment automatically. The user name and password is *ttc* if requested.

AToMPM has two main toolbars. First toolbar is used for basic file manipulation and depicted in Fig. 10. We only give meanings of the necessary buttons for the sake of simplicity.

1. **New:** opens a new AToMPM instance in a new window.
2. **Load:** opens a popup box and lets you select a model to load.
3. **Toggle Visibility:** toggles the visibility of any formalism. For example, one can easily hide Petri Net elements or Statechart elements using this button.

Second toolbar is transformation controller and depicted in Fig. 11. Again we only give meanings of the necessary buttons.

1. **Load:** opens a popup box and lets you select a transformation to be loaded.
2. **Play:** executes all the rules in the transformation automatically.
3. **Step:** executes the rules one by one.

The test-cases stay in this folder: **/Formalisms/TTC2013/testcases/**

The main transformation to load is: **/Formalisms/TTC2013/pn2scTrafo_New/T_ALL.model**

There will be a Chrome window open with all testcases and transformations loaded and ready to execute, you can just press *play* button in the transformation toolbar.

If that is not the case, you can load any test-case using the main toolbar and select the main transformation using the transformation toolbar and see the execution of it by pressing *play*. If the test-case is too big and you want to see all, just play with your browsers' zoom level.

You can re-play any testcase by simply loading it again from main toolbar, and choosing the transformation. You can open a new instance anytime by just pressing *New* button in main toolbar.

B Appendix: Simulation

We also created a rule-based Petri Net simulator and a highlighter for the statecharts as the Petri Net is being simulated. The rules and the control flow of the rules are depicted in Fig. 12. The simulator first

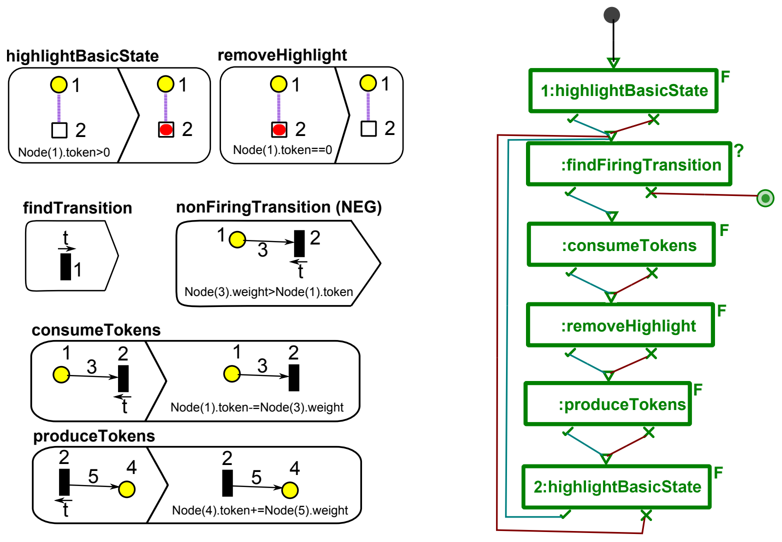

Figure 12: Rules and Control Flow for Simulation

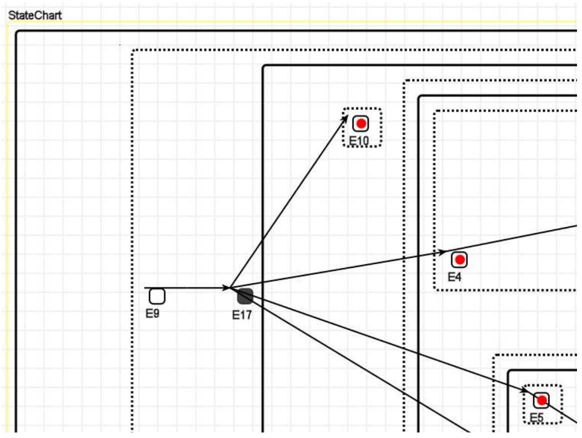

Figure 13: A Screenshot from Simulation

highlights the *basic states* by using the *highlightBasicState* rule if the corresponding *place* has a token. Then, the *findTransition* and *nonFiringTransition* rules check for a suitable *transition* which has tokens on all incoming *places*. The *consumeTokens* rule reduces the token number from incoming *places* and the *removeHighlight* rule removes the highlight from *basic states* if the corresponding *place* has no token. Finally, the *produceTokens* increases the token number of all outgoing *places* and next rule highlights the *basic states* again. This control flow loops until no more suitable transition found.

A screenshot from simulation is depicted in Fig. 13. The underlying Petri Net is being simulated and the basic states in statechart is highlighted as in the figure. The statechart in the figure is from testcase1.

C Appendix: Expanded Control Flow

Fig. 14 depicts the expanded control flow of the transformation.

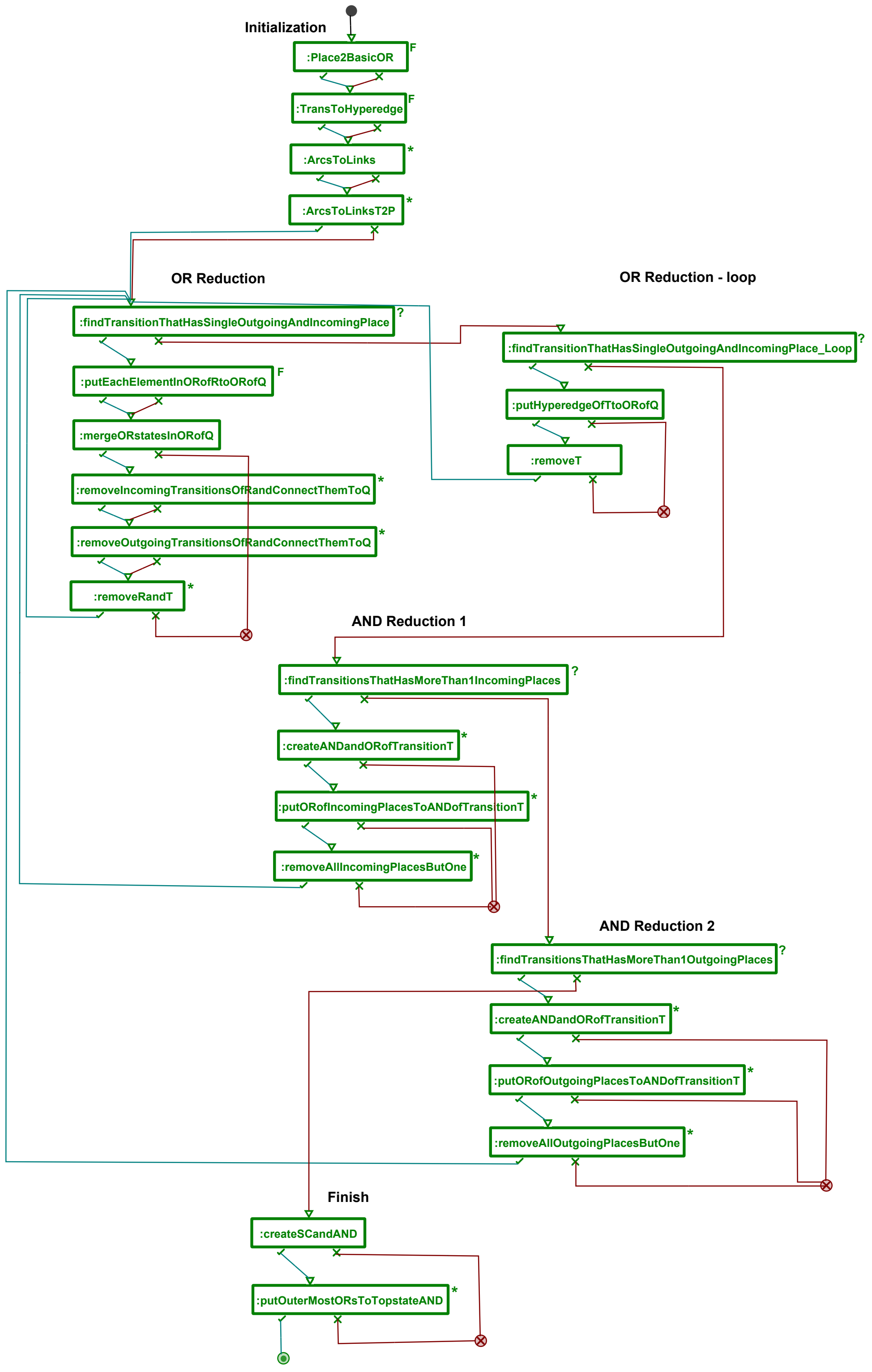
Initialization
:Place2BasicOR
F
:TransToHyperedge
F
:ArcsToLinks
*
:ArcsToLinksT2P
*
OR Reduction
:findTransitionThatHasSingleOutgoingAndIncomingPlace
?
:putEachElementInORofRtoORofQ
F
:mergeORstatesInORofQ
:removeIncomingTransitionsOfRandConnectThemToQ
*
:removeOutgoingTransitionsOfRandConnectThemToQ
*
:removeRandT
*
OR Reduction - loop
:findTransitionThatHasSingleOutgoingAndIncomingPlace_Loop
?
:putHyperedgeOfTtoORofQ
:removeT
AND Reduction 1
:findTransitionsThatHasMoreThan1IncomingPlaces
?
:createANDandORofTransitionT
*
:putORofIncomingPlacesToANDofTransitionT
*
:removeAllIncomingPlacesButOne
*
AND Reduction 2
:findTransitionsThatHasMoreThan1OutgoingPlaces
?
:createANDandORofTransitionT
*
:putORofOutgoingPlacesToANDofTransitionT
*
:removeAllOutgoingPlacesButOne
*
Finish
:createSCandAND
:putOuterMostORsToTopstateAND
*

Figure 14: Full Control Flow with Expanded Phases