AUTOMATIC TRANSLATION OF MSC DIAGRAMS INTO PETRI NETS

S. Kryvyy, L. Matvyeyeva, M. Lopatina

Abstract: Development-engineers use in their work languages intended for software or hardware systems design, and test engineers utilize languages effective in verification, analysis of the systems properties and testing. Automatic interfaces between languages of these kinds are necessary in order to avoid ambiguous understanding of specification of models of the systems and inconsistencies in the initial requirements for the systems development.

Algorithm of automatic translation of MSC (Message Sequence Chart) diagrams compliant with MSC'2000 standard into Petri Nets is suggested in this paper. Each input MSC diagram is translated into Petri Net (PN), obtained PNs are sequentially composed in order to synthesize a whole system in one final combined PN. The principle of such composition is defined through the basic element of MSC language — conditions. While translating reference table is developed for maintenance of consistent coordination between the input system's descriptions in MSC language and in PN format. This table is necessary to present the results of analysis and verification on PN in suitable for the development-engineer format of MSC diagrams. The proof of algorithm correctness is based on the use of process algebra ACP. The most significant feature of the given algorithm is the way of handling of conditions. The direction for future work is the development of integral, partially or completely automated technological process, which will allow designing system, testing and verifying its various properties in the one frame.

Keywords: MSC diagram, MSC language, condition, automatic translation, Petri Net.

Introduction

While designing and developing of either software or hardware it is of vital importance to detect and remove defects in product on its early stages in order to avoid time and resource losses. Development-engineers and test engineers (verifiers) use in their work different approaches and specification languages, that eventually leads to ambiguous understanding of the same portion of a project, to inaccuracies, incompletenesses or even to the inconsistencies in the initial requirements for the development. Development-engineers usually utilize languages intended for design purposes (as VHDL, MSC, SDL, UML and so on), while test engineers (verifiers) utilize languages effective in verification and testing (languages of mathematical logics, automata theory, algebraic and net languages). The way-out of this situation is a development of automatic interfaces between languages of these kinds. Given work is devoted to the development of automatic interface between the languages MSC (Message Sequence Chart) and PN (Petri Nets). The work suggests an algorithm of automatic translation of MSC diagrams compliant with MSC'2000 language standard [ITU-TS, 2000] into Petri Nets, which allow verifying automatically a lot of properties of the system under design. The algorithm works on a certain subset of MSC'2000 language.

1. Syntactic MSC constructions

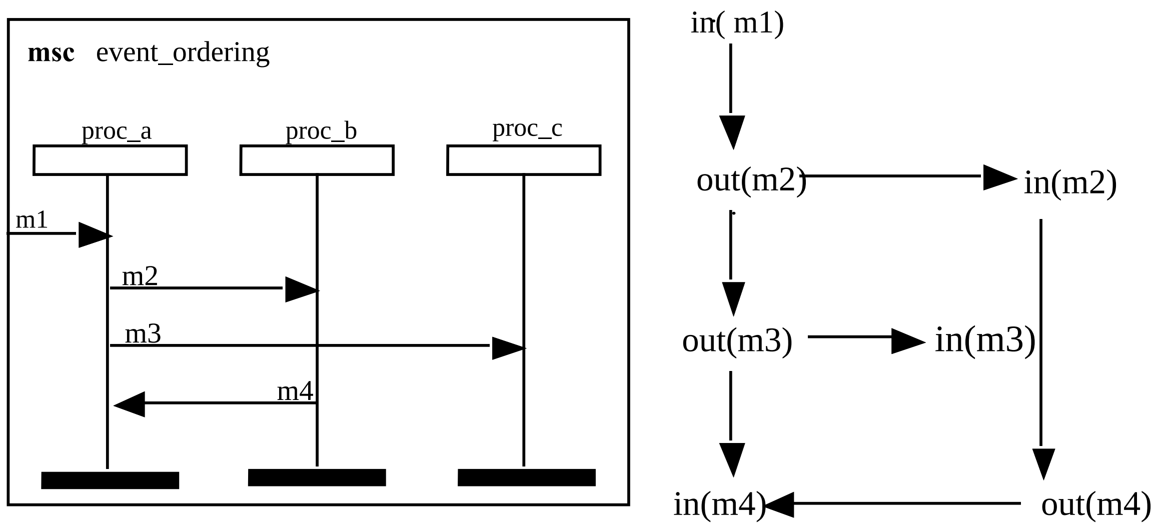
MSC is a modelling technique that uses a graphical interface, which was standardized by ITU (International Telecommunication Union, earlier CCITT). It is usually applied to applications of the telecommunication domain, since they have properties of distributed reactive real-time systems, often in combination with SDL language [Grabowski, 1991]. These very properties of the systems make an MSC with possibility of scenario describing extremely suitable as for specification so for testing purposes. This means that MSC can be applied on every stage of system development, even on the stage of test case development. MSC describes message flow between the instances, which present asynchronously communicating objects of the system or system entities like blocks, services or processes of the system. One MSC diagram describes a certain portion of system behaviour or a scenario of communication between the instances.

MSC has two syntactical representations: textual and graphical, which are in one-to-one relation according to a standard. Basic elements of the language are those which define message flow, namely, *instance*, *message*, *action*, *set*->*reset*, *set*->*time-out*, *stop*, *create* and *condition*. An example of an MSC is presented on Figure 1(a). As far as this example is of illustrative kind, it only introduces a minimal set of possible MSC constructions as instances and messages. Let's describe basic elements of the language MSC'2000, which are considered in the given work.

1.1. Instances, messages and system environment.

Instance is a basic primitive of MSC, which in graphics is presented as vertical line with its name.

Message transmissions, which are acts of communication between instances, are presented by horizontal arrows with possible curve or tilt under angle for reflecting “overtaking” or “intersection” of messages. The beginning of the vector marks a sending of the message and its ending marks receiving of the message. Evens of sending and receiving of the messages are ordered along the instances so that sending of the message always happens earlier than its receiving. There is one more rule in standard MSC’2000 for ordering events along the instances: everything located above happens earlier than that located below. A MSC diagram imposes a partial ordering on the set of events being contained. A binary relation which is transitive, antisymmetric and reflexive is called partial order. The partial ordering can be described by its connectivity graph.



(a) (b)

Figure 1

At the Picture 1(b) there is a graph, which reflects the order of events on instances in the diagram “msc event_ordering” (Figure 1(a)). Event out(m1) means sending of the message m1, in(m1) – receiving of message m1.

Environment of the system (the set of instances) is presented by the borders of MSC.

1.2. Conditions

Condition is used as for restricting or defining a set of MSC traces through indicating states of the system so for defining the composition of one MSC diagram from the several MSC diagrams. Condition can describe a global state of the system which is extended to all MSC instances existing in the time of this condition; it also can describe a state for a certain subset of MSC instances. In the first case condition is called global. Instances presenting dynamic objects can be began and finished, so far globality of a state considers dynamically changing set of instances.

Standard MSC‘2000 [ITU-TS, 2000] defines conditions of two types: setting condition and guarding condition. Conditions of the first type are those which describe the current state of the system. Conditions of the second type restrict behaviour of the MSC to execution of events in a certain part of MSC depending on the value of the given guarding condition.

Besides of composition role, conditions according to the standard [ITU-TS, 2000] are the means of events synchronization. For example, if two instances share one and the same condition, then for each message between these instance its sending and receiving events shall happen both before or both after setting of the condition. If two conditions are ordered directly sharing the common instance, or indirectly through conditions on other instances, then this order must be respected on all instances that share these two conditions.

2. MSC semantics

The first version of MSC language standard defined the semantics incompletely and informally, however in ideal development of the language and its semantics shall go in parallel. Need of semantics standardization was becoming apparent, as even MSC experts could not always agreed on interpretation of the particular properties. Associated with this situation and extension of MSC language application in 1992 three new approaches of MSC semantics defining were submitted to standardization committee CCITT (now ITU-TS or Telecommunication Standardization section of the International Telecommunication Union).

The first approach was based on the theory of finite automata.

The second one was based on theory of Petri Nets using partially ordered sequences of events in the system. Given semantics is known to be extremely suitable for modelling of distributed asynchronous systems.

The third approach was based on process algebra ACP (Algebra of Communicating Processes) and interleaving model, when system is modelled by the sequence of transitions supposing that events are atomic and has no duration, and in every moment of time only one event can be executed. Semantics of interleaving simulates independence (asynchronism) among the subsystems (instances) through nondeterministic interleaving of independent parallel activities.

Every of the three approaches given for definition of MSC language semantics has its advantages and disadvantages, but the committee for standardization chose third approach to be the basis for formal definition of MSC language semantics. MSC language semantics based on process algebra [Bergstra, 1984] was defined for textual representation of MSC diagrams [ITU-TS, 1995] in expressions of process algebra that was called denotation semantics. Operational semantics is defined through addition of transitional rules to algebraic expressions. So, operational semantics is reflection of MSC-specifications into transition system. Yet it should note that operational semantics of MSC is not defined and standardized formally.

3. Algorithm of translation of MSC diagrams into Petri Net

The first progresses in defining MSC semantics basing on Petri Nets were presented in [Grabowski, 1993]. Nowadays research work on translating of MSCs into Petri Nets goes on and its results are covered, for example, in [Kluge, 2000], [Heymer, 2000], [Kluge, 2001]. However the authors develop semantics for the MSC'96 and MSC'2000 standards basing on Petri Nets, and, what’s more, elaborate on MSCs into the Petri Nets translation, an automatic translation is not considered in these papers.

3.1. Description of algorithm

The following is the formalized description of the translation algorithm.

INPUT: A set of the MSC diagrams in the basic subset of the language MSC'2000.

OUTPUT: A Petri Net adequate to the set of initial MSC diagrams.

METHOD: Translation of every MSC diagram of the input set into a Petri Net is performed in two stages in parallel with composing of corresponding to each MSC diagram Petri Nets into one final combined Petri Net (synthesis).

```
Begin
Stage 1. Building of the partial-order graph to reflect events order in the initial MSC diagrams imposed by static requirements of MSC'2000 standard [ITU-TS, 2000] (see Figure 1).
Stage 2. Translating of the partial-order graph obtained at the stage 1 into the Petri Net.
End
```

Let's give more detailed description of the algorithm’s stages and start with the description of how synthesis is proceeds.

Detailed description of the synthesis

Each input MSC diagram is being translated into PN, the PNs are sequentially "glued" in order to compose (synthesize) a whole system in one PN, while the principle of such composition is defined at the level of MSC language - through *conditions*. Let's detail semantics of the basic element of MSC language *condition* as it was described in the MSC'2000 standard.

According to the MSC'2000 standard *condition* defines a system state of instance, which it covers . A system state of the instance is interpreted not as a current global state of the whole system, but as a set of the current values of a certain subset of attributes coherent with the instance (system object/entity), another words, *condition* is a precondition of occurring the event in the system. Let's explain this statement. MSC language was created to specify systems with local interconnections among the asynchronous parallel processes. Asynchronous models are typically built on cause-and-effect relation among events rather than on clocked sequence of system state changes. Asynchronous systems also present time moments or intervals as events. So event in this model is considered to be either atomic or compound with internal structure formed from "sub-events". Thus, *condition* is a precondition of occurring the event in the system and simultaneously a synchronizing action. The steps of the synthesis are carried out according to the following rules, on the assumption of fulfilling the following requirements.

The requirements:

1. We only consider a subset of MSC'2000, including the elements: instances, message inputs and outputs, setting conditions in textual representation. It is supposed, that all input diagrams are syntactically correct and satisfy the static requirements of the MSC'2000 standard. For example, for each event of a message sending the diagram shall have a pair event — a message consumption.
2. Naming of instances is complete and exact.
3. Naming of conditions is complete and exact, meaning of the conditions do not influence synthesis in any way.
4. Each instance shall have initial and final conditions either local or, probably, shared by several instances. The given requirement is not referred to a case of creation and termination of the instance within the scope of the given diagram.

The condition is called *initial*, if the diagram has not any event or condition which precedes it, and *final*, if there is no event or condition after the given condition. Not only singular but also multiple initial and final conditions are possible, in logic they can be represent as conjunction of all initial conditions, and, correspondingly, conjunction of all final conditions.

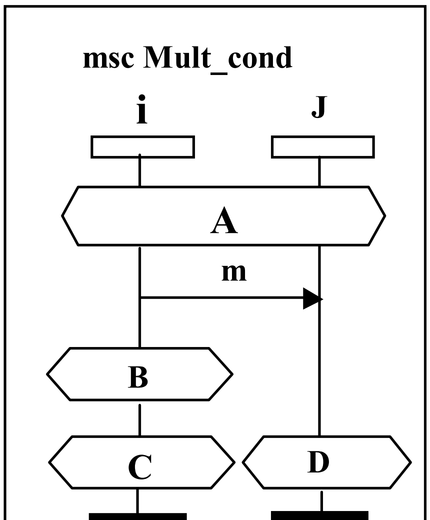


Figure 2

For example, at Figure 2 conditions B and C represent a multiple final condition, D — singular final condition or simply final condition, A is an initial condition.

5. Each diagram from the input set shall satisfy the following: every initial and final condition of MSC shall cover a minimum one instance, on which an event occurs (in given case, either sending or consumption of the message).

6. "Gluing" is forced and carried out according to the following rules.

The Rules:

1. If the first MSC diagram has a final condition, which corresponds (has the same name) to the initial condition of the second MSC diagram, and these conditions cover the same set of instances in the both diagrams, than the first and the second diagrams can be "glued" together via the given condition regardless of the fact that this condition is either global or local for the diagrams. For example, at Figure 3 msc Z is a synthesis or composition of msc X and msc Y. The dashed line shows places of "gluing".

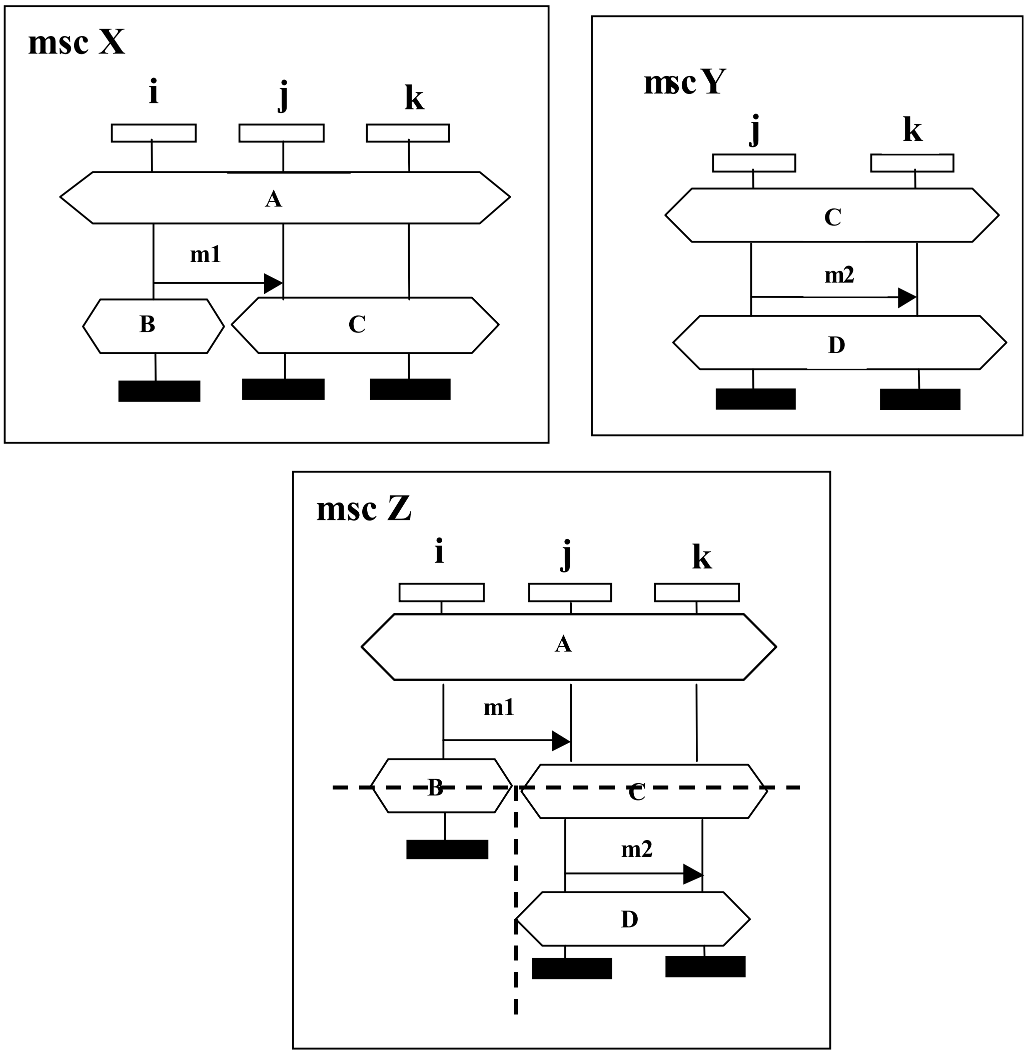


Figure 3

This rule also assumes the opportunity to "glue" by condition not only two diagrams in sequence, but also to "glue" together more than two diagrams at once ("multiple gluing"), this means that "gluing" is a non-deterministic alternative composition in opposite to the delayed choice operator for alternatives. As a result of gluing we receive an MSC diagram presenting a set of possible alternative traces in the system. Let's consider, for example, Figure 4. There are two possible continuations of the diagram msc X: it is msc Y and msc Z. And the choice is made when the condition C occurs and it is not delayed until the events presented on msc Z (msc Result 1) and on msc Y (msc Result 2) begin to differ from each other, namely, before passing of the messages m and k.

2. The following is a very important rule for "gluing" of MSC diagrams by overlapped and multiple conditions:

a) conditions occur simultaneously and are equal to the conjunction of these conditions,

b) the order of their enumeration in the diagrams is of no importance because of simultaneous occurrence of conditions,

c) these conditions can glue together independently from each other.

Let's consider an example at Figure 5. Each of MSC diagrams X1, X2, X3 is a possible continuation of msc X. The conditions B and C are overlapped in msc X and msc X3.

All requirements and rules mentioned above do not contradict the MSC'2000 standard, and only specify it. Composing of one common Petri Net is performed in accordance with semantics of the synthesis of MSC diagrams described above. A table of initial and final conditions of the initial MSCs plays a key role in the synthesis, as it contains all information necessary for composing. It is formed on the stage 1 during sequential processing of MSCs. Thus, the gluing is performed along with translation of MSCs applying the table of initial and final conditions.

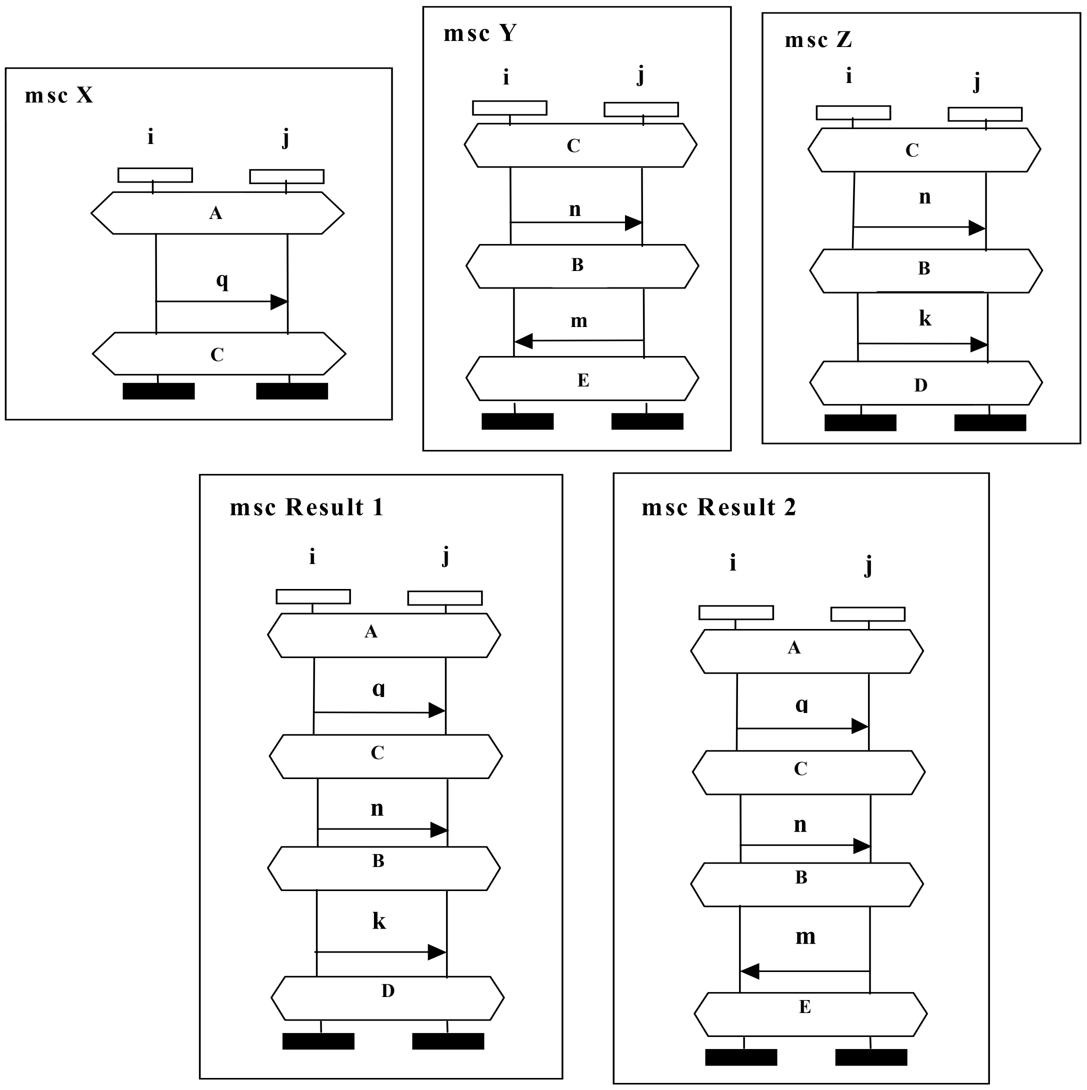


Figure 4

Detailed description of stage 1. Partial-order graph for system events is built according to the following rules:

1. Directed edges of the graph set an order of the events.

2. Events of sending and consumption of messages and conditions correspond to the graph nodes. It is necessary to emphasize, that a condition is also represented by an event, namely, event of synchronization. Moreover the graph defines nodes of two types: a graph node which denotes intermediate condition or event of message consumption/sending, and a graph node which denotes initial or final conditions. They differ in the way of translation into the elements of Petri Net during the second stage.

Detailed description of stage 2. Rules for the second stage of translation (translation of the graph into the Net) are the following:

1. Each directed edge of the partial-order graph is translated into a place of Petri Net.

2. An arrow of each directed edge of the graph corresponds to an arrow of Petri Net that directs tokens' flow in Petri Net.

3. A graph node of the first type (denoted intermediate conditions and events of an message input (consumption) and message output (sending)) is translated into a transition of Petri Net.

4. A graph node of the second type (denoted final and initial conditions) is translated into combination of transition and place of Petri Net. This place is a joint element for "gluing" into the common Petri Net (representing input system as a whole) .

While translating reference table is developed for maintenance of consistent coordination between the input system's descriptions in MSC language and in Petri Net format. This table is necessary to present the results of analysis and verification on Petri Net in suitable for the development-engineer format of MSC diagrams.

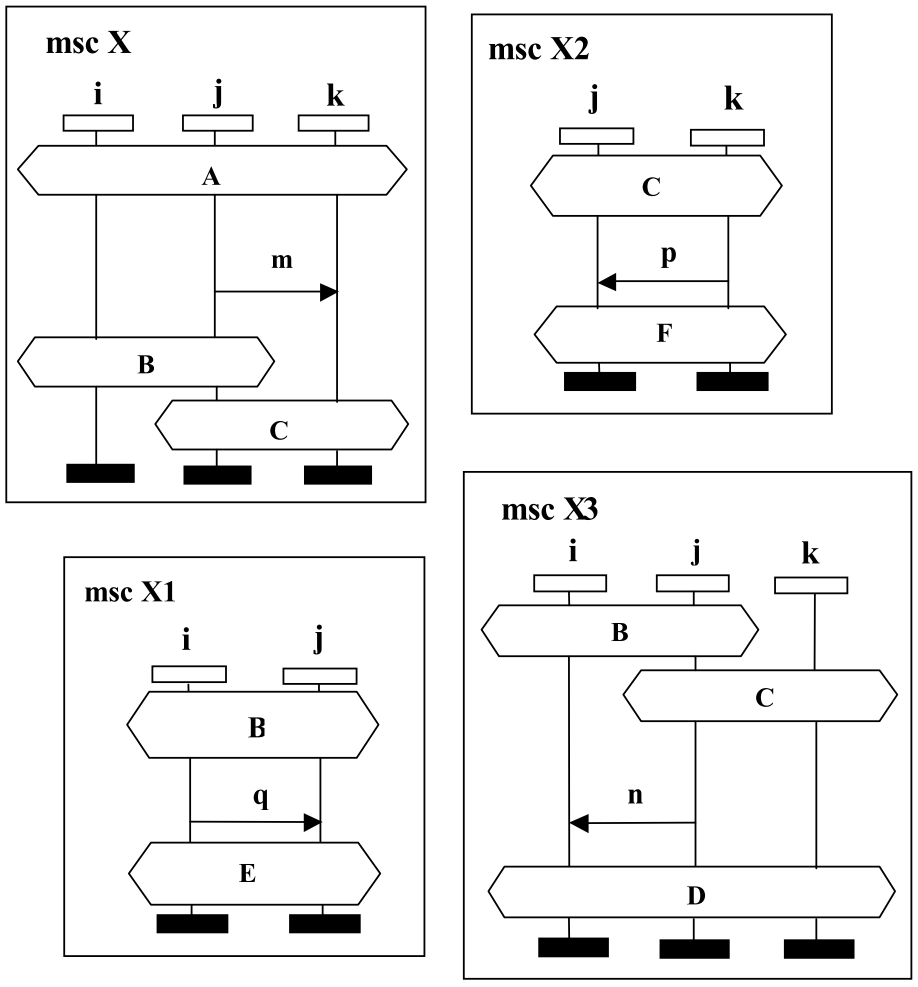


Figure 5

Unfortunately, limited size of the paper does not allow to place here the proof of algorithm correctness even in the reduced version. We note only that the proof of algorithm correctness is based on the use of process algebra ACP.

Conclusion

Summing up, we note, that the algorithm of an automatic translation, presented in the paper, considers only the subset of MSC language, therefore extending of this subset to the maximum or even up to the whole MSC language is an obvious direction of the further research. The most significant feature of the given algorithm is the way of handling of conditions, since the literature indicates this problem in translation process as the most difficult. What is also important is obtaining of necessary experience for developing analog translators for the languages: SDL, UML, etc. The ultimate goal of this research is development of integral, partially or completely automated technological process, which will allow designing system, testing and verifying its various properties in the one frame.

Bibliography

[Grabowski, 1993] J. Grabowski, P. Graubmann, E. Rudolph. Towards a Petri Net Based Semantics Definition for Message Sequence Charts, In O.Fergemand and A.Sarma, editors, SDL'93 Using Objects, Proceedings of the 6th SDL Forum, Darmstadt, 1993. Elsevier Science Publishers B.V.

[Grabowski, 1991] J. Grabowski, P. Graubmann, E. Rudolph. Towards an SDL-Design-Methodology Using Sequence Cart Segment, SDL'91 Evolving Methods - O.Fergemand and A.Sarma, editors, North-Holland, 1991.

[CCITT, 1992]. CCITT Recommendation Z.120: Message Sequence Chart (MSC). Geneva,1992.

[Reniers, 1995] M. A. Reniers.Static semantics of Message Sequence Charts. In SDL'95 with MSC in CASE, Proceedings of the Seventh SDL Forum, Oslo, 1995. Elsevier Science Publishers B.V.

[ITU-TS, 1995] ITU-TS Recommendation Z.120 Annex B: Algebraic semantics of Message Sequence Charts. ITU-TS, Geneva, 1995.

[Котов, 1984] В.Е.Котов Сети Петри. М.:Наука, 1984, 157 стр.

[ITU-TS, 2000] ITU-TS Recommendation Z.120: Message Sequence Chart (MSC). ITU-TS, Geneva, 2000.

[Belina, 1992] F. Belina. SDL Methodology Guidelines, CCITT, Geneva, May 1992.

[Bergstra, 1984] J.A. Bergstra, J.W. Klop. Process Algebra for Synchronous Communication, Inf.&Control 60,pp.109-137,1984.

[Mauw, 1995] S. Mauw, M.A. Reniers. Thoughts on the meaning of conditions. Experts meeting SG10, St.Petersburg TD9016, ITU-TS,1995.

[Kluge, 2000] O. Kluge. Time in Message Sequence Charts Specifications and How to Derive Stochastic Petri Nets. In: Proceedings of the Third International Workshop on Communication Based Systems (CBS3), Berlin, March 2000.

[Heymer, 2000] S. Heymer. A Semantic for MSC based on Petri-Net Components, SIIM Technical Report A-00-12, Informatik-Berichte Humboldt-Universitдt zu Berlin, June 2000.

[Kluge, 2001] O. Kluge, J. Padberg, H. Ehrig. Modeling Train Control Systems: From Message Sequence Charts to Petri Nets, Technische Universitдt Berlin,http://cs.tu-berlin.de/SPP/index.html, 2001

Authors information

Sergiy Kryvyy - Institute of cybernetics, NAS of Ukraine, Pr. Glushkova, 40, Kiev-03187, Ukraine; e-mail: krivoi@i.com.ua

Lyudmila Matvyeyeva - Institute of cybernetics, NAS of Ukraine, Pr. Glushkova, 40, Kiev-03187, Ukraine; e-mail: luda@iss.org.ua

Mariya Lopatina - Institute of cybernetics, NAS of Ukraine, Pr. Glushkova, 40, Kiev-03187, Ukraine; e-mail: marichkay@mail.ru