SINTEF REPORT

SINTEF

SINTEF ICT

Address: NO-7465 Trondheim
NORWAY
Location Trondheim:
S.P. Andersens v 15
Location Oslo:
Forskningsveien 1
Telephone: +47 73 59 30 00
Fax: +47 73 59 43 02

Enterprise No.: NO 948 007 029 MVA

TITLE

A Transformational Approach to Facilitate Monitoring of High Level Policies

AUTHOR(S)

Fredrik Seehusen, Mass Soldal Lund and Ketil Stølen

CLIENT(S)

Norwegian Research Council (NCR) and European Commission (EC)

REPORT NO. A11356	CLASSIFICATION Unrestricted	CLIENTS REF. NCR: 152839 and 180052, and EC: FP6-IST-27004	
CLASS. THIS PAGE Unrestricted	ISBN 978-82-14-04433-1	PROJECT NO. 90B230, 90B245, and 403328	NO. OF PAGES/APPENDICES 76/5
ELECTRONIC FILE CODE		PROJECT MANAGER (NAME, SIGN.) Ketil Stølen [illegible]	CHECKED BY (NAME, SIGN.) Atle Refsdal [illegible]
FILE CODE	DATE 2009-03-31	APPROVED BY (NAME, POSITION, SIGN.) Bjørn Skjellaug, Research director [illegible]	

ABSTRACT

We present a method for specifying high level security policies that can be enforced by runtime monitoring mechanisms. The method has three main steps: (1) the user of our method formalizes a set of policy rules using UML sequence diagrams; (2) the user selects a set of transformation rules from a transformation library, and applies these using a tool to obtain a low level intermediate policy (also expressed in UML sequence diagrams); (3) the tool transforms the intermediate low level policy expressed in UML sequence diagrams into a UML inspired state machine that governs the behavior of a runtime policy enforcement mechanism. We believe that the method is both easy to use and useful since it automates much of the policy formalization process.

The method is underpinned by a formal foundation that precisely defines what it means that a system adheres to a policy expressed as a sequence diagram as well as a state machine. The foundation is furthermore used to show that the transformation from sequence diagrams to state machines is adherence preserving under a certain condition.

KEYWORDS	ENGLISH	NORWEGIAN
GROUP 1	ICT, modelling	IKT, modellering
GROUP 2	Design	Design
SELECTED BY AUTHOR	MDA, Information flow security,	MDA, Sikker informasjonsflyt
	Refinement, Transformation	Raffinering, Transformasjon

A Transformational Approach to Facilitate Monitoring of High Level Policies

Fredrik Seehusen, Mass Soldal Lund, and Ketil Stølen

2009

Abstract

We present a method for specifying high level security policies that can be enforced by runtime monitoring mechanisms. The method has three main steps: (1) the user of our method formalizes a set of policy rules using UML sequence diagrams; (2) the user selects a set of transformation rules from a transformation library, and applies these using a tool to obtain a low level intermediate policy (also expressed in UML sequence diagrams); (3) the tool transforms the intermediate low level policy expressed in UML sequence diagrams into a UML inspired state machine that governs the behavior of a runtime policy enforcement mechanism. We believe that the method is both easy to use and useful since it automates much of the policy formalization process.

The method is underpinned by a formal foundation that precisely defines what it means that a system adheres to a policy expressed as a sequence diagram as well as a state machine. The foundation is furthermore used to show that the transformation from sequence diagrams to state machines is adherence preserving under a certain condition.

1 Introduction

Policies are rules governing the choices in the behavior of a system [19]. We consider the kind of policies that are enforceable by mechanisms that work by monitoring execution steps of some system which we call the *target* of the policy. This kind of mechanism is called an EM (Execution Monitoring) mechanism [16].

The security policy which is enforced by an EM mechanism is often specified by a state machine that describes exactly those sequences of security relevant actions that the target is allowed to execute. Such EM mechanisms receive an input whenever the target is about to execute a security relevant action. If the state machine of the EM mechanism has an enabled transition on a given input, the current state is updated according to where the transition lands. If the state machine has no enabled transition for a given input, then the target is about to violate the policy being enforced. It may therefore be terminated by the EM mechanism.

Security policies are often initially expressed as short natural language statements. Formalizing these statements is, however, time consuming since they often refer to high level notions such as "opening a connection" or "sending an SMS" which must ultimately be expressed as sequences of security relevant actions of the target. If several policies refer to the same high level notions, or should be applied to different target platforms, then these must be reformalized for each new policy and each new target platform.

Clearly, it is desirable to have a method that automates as much of the formalization process as possible. In particular, the method should:

1. support the formalization of policies at a high level of abstraction;
2. offer automatic generation of low level policies from high level policies;
3. facilitate automatic enforcement by monitoring of low level policies;
4. be easy to understand and employ by the users of the method (which we assume are software developers).

The method we present has three main steps which accommodate the above requirements:

Step I The user of our method receives a set of policy rules written in natural language, and formalizes these using UML sequence diagrams.

Step II The user creates transformation rules (expressed in UML sequence diagrams) or selects them from a transformation library, and applies these using a tool to obtain an intermediate low level policy (also expressed in UML sequence diagrams).

Step III The tool transforms the intermediate low level policy expressed in UML sequence diagrams into state machines that govern the behavior of an EM mechanism.

There are two main advantages of using this method as opposed to formalizing low level policies directly using state machines. First, much of the formalization process is automated due to the transformation from high to low level. This makes the formalization process less time consuming. Second, it will be easier to show that the formalized high level policy corresponds to the natural language policy it is derived from, than to show this for the low level policy. The reason for this is that the low level policy is likely to contain implementation specific details which make the intention of the policy harder to understand.

The choice of UML is motivated by requirement 4. UML is widely used in the software industry. It should therefore be understandable to many software developers which are the intended users of our method. UML sequence diagrams are particularly suitable for policy specification in the sense that they specify partial behavior (as opposed to complete), i.e., the diagrams characterize example runs or snapshots of behavior in a period of time. This is useful when specifying policies since policies are partial statements that often do not talk about all aspects of the target's behavior. In addition to this, UML sequence diagrams allow for the explicit specification of negative behavior, i.e., behavior that the target is not permitted to engage in. This is useful because the only kind of policies that can be enforced by EM mechanisms are prohibition policies, i.e., policies that stipulate what the target is *not* allowed to do.

The main body if this report gives an example driven presentation of our method without going into all the technical details. In the appendices, we present the formal foundation of the method. In particular, the rest of this report is structured as follows: In Sect. 2 we describe step I of our method by introducing a running example and showing how high level security policies can be expressed with UML sequence diagrams. Sect. 3 describes step II of our method by showing how a transformation from high level to low level policies can be specified using UML sequence diagrams. Step III of our method is described in Sect. 4 which defines a transformation from (low level) sequence diagram policies to state machine policies that can be enforced by EM mechanisms. Sect. 4 discusses related work, and Sect. 5 provides conclusions and directions of future work. The formal foundation is presented in the appendices. App. A and App. B, present the syntax and the semantics of UML sequence diagrams and state machines, respectively. We also define *adherence*, i.e., what it means for a system to adhere to a sequence diagram or a state machine. Finally, App. C characterizes the transformation from high to low level sequence diagrams. App D defines the transformation from low level sequence diagrams to state machines. In App D, we also show that the transformation is adherence preserving given that a certain condition is satisfied. All proofs are given in App. E.

2 Step I: Specifying policies with sequence diagrams

In the first step of our method, the user receives a set of policy rules written in natural language. The user then formalizes these rules using UML sequence diagrams. In this section, we show how to express two security policies. The examples are taken from an industrial case study conducted in the EU project S^{3}MS [17].

As the running example of this report, we consider applications on the Mobile Information Device Profile (MIDP) Java runtime environment for mobile devices. We assume that the runtime environment is associated with an EM mechanism that monitors the executions of applications. Each time an application makes an API-call to the runtime environment, the EM mechanism receives that method call as input. If the current state of the state machine that governs the EM mechanism has no enabled transitions on that input, then the application is terminated because it has violated the security policy of the EM mechanism.

2.1 Example – specifying policy 1

The first policy we consider is

The application is not allowed to establish connections to other addresses than http://s3ms.fast.de.

This policy is specified by the UML sequence diagram of Fig. 1.

Sequence diagrams describe communication between system entities which we will refer to as *lifelines*. In a diagram, lifelines are represented by vertical dashed lines. An arrow between two lifelines represents a message being sent from one lifeline to the other in the direction of the arrow. Sequence diagrams should be read from top to bottom; a message on a given lifeline should occur

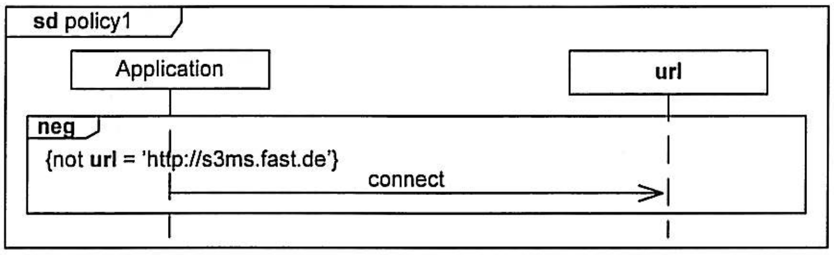


Figure 1: High level policy 1

before all messages that appear below it on the same lifeline (unless the messages are encapsulated by operators). Communication is asynchronous, thus we distinguish between the occurrence of a message transmission and a message reception. Both kinds of occurrences are viewed as instantaneous and in the following called *events*.

The two lifelines in Fig. 1 are Application representing the target of the policy, and url, representing an arbitrary address that the target can connect to. The sending of message connect from the Application to url represents an attempt to open a connection.

Expressions of the form $\{bx\}$ (where bx is a boolean expression) are called *constraints*. Intuitively, the interaction occurring below the constraint will only take place if the constraint evaluates to true.

The constraint of Fig. 1 should evaluate to true if and only if url is *not* equal to the address "http://s3ms.fast.de" (which according to the policy is the only address that the application is allowed to establish a connection to).

Interactions that are encapsulated by the neg operator specify negative behavior, i.e., behavior which the target is not allowed to engage in. Thus Fig. 1 should be read: Application is not allowed to connect to the arbitrary address url if url is different from the address "http://s3ms.fast.de".

UML sequence diagrams are partial in the sense that they typically don't tell the complete story. There are normally other legal and possible behaviors that are not considered within the described interaction. In particular, sequence diagrams explicitly describe two kinds of behavior: behavior which is *positive* in the sense that it is legal, valid, or desirable, and behavior which is *negative* meaning that it is illegal, invalid, or undesirable. The behavior which is not explicitly described by the diagram is called *inconclusive* meaning that it is considered irrelevant for the interaction in question.

We interpret sequence diagrams in terms of positive and negative *traces*, i.e., sequences of events (see App. A). When using sequence diagrams to express prohibition policies, we are mainly interested in traces that describe negative behavior. If a system is interpreted as a set of traces, then we say that the systems *adheres* to a policy if none of the system's traces have a negative trace of a lifeline of the policy as a sub-trace[1]. Thus we take the position that the target is allowed to engage in (inconclusive) behavior which is not explicitly described by a given policy. This is reasonable since we do not want to use

[1]A trace s is a (possibly non-continuous) sub-trace of trace t if s can be obtained from t by removing zero or more events from t.

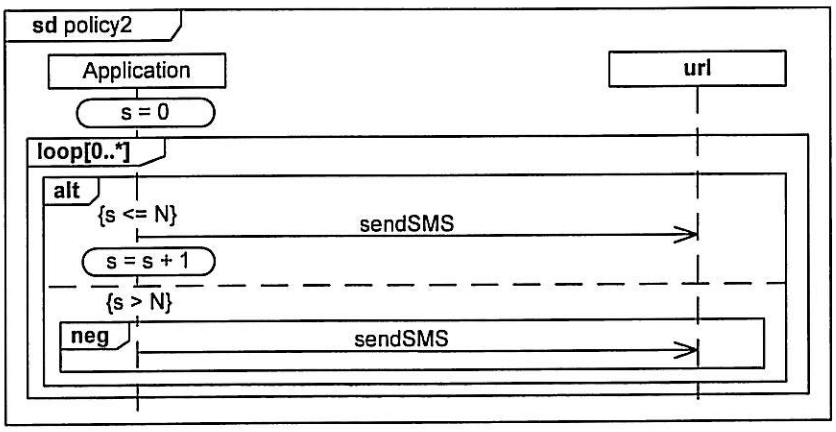


Figure 2: high level policy 2

policies to express the complete behavior of the target.

Turning back to the example, an application is said to adhere to the policy of Fig. 1 if none of its traces contain the transmission of the message connect to an address which is different from http://s3ms.fast.de.

2.2 Example – specifying policy 2

The second natural language policy is

The application is not allowed to send more than N SMS messages (where N is a natural number).

This policy is specified by the sequence diagram of Fig. 2. Again, the lifelines of the diagram are Application representing the policy target, and url, this time representing an arbitrary recipient address of an SMS message.

Boxes with rounded edges contain assignments of variables to values. In Fig. 2, the variable s is initialized to zero, and incremented by one each time the application sends an SMS. The loop operator is used to express the iteration of the interaction of its operand.

The alt-operator is used to express *alternative* interaction scenarios. In Fig. 2, there are two alternatives. The first alternative is applicable if the variable s is less than or equal to N (representing an arbitrary number). In this case, the application is allowed to send an SMS, and the variable s is incremented by one. The second alternative is applicable when s is greater than N. In this case, the application is not allowed to send an SMS as specified by the neg-operator.

An application adheres to the policy of Fig. 2 if none of its traces contain more than N occurrences of the message sendSMS.

3 Step II: Specifying transformations with sequence diagrams

In the second step of our method, the user creates new transformation rules or selects them from a transformation library. The users then employs a tool which automatically applies the transformation rules to the high level policy such that an intermediate low level policy is produced. In the following, we show how a transformation to the low level can be defined using UML sequence diagrams. An advantage of using sequence diagrams for this purpose is that the writer of the transformation rules can express the low level policy behavior using the same language that is used for writing high level policies. This will also make it easier for the user of our method to understand or modify the transformation rules in the transformation library if that should become necessary.

A *transformation rule* is specified by a pair of two *diagram patterns* (diagrams that may contain *meta variables*), one left hand side pattern, and one right hand side pattern. When a transformation rule is applied to a diagram d, all fragments of d that match the left hand side pattern of the rule are replaced by the right hand side pattern. Meta variables are bound according to the matching. A diagram pattern dp matches a diagram d if the meta variables of dp can be replaced such that the resulting diagram is syntactically equivalent to d.

In the following we illustrate the use of transformation rules by continuing the example of the previous section.

3.1 Example – specifying a transformation for policy 1

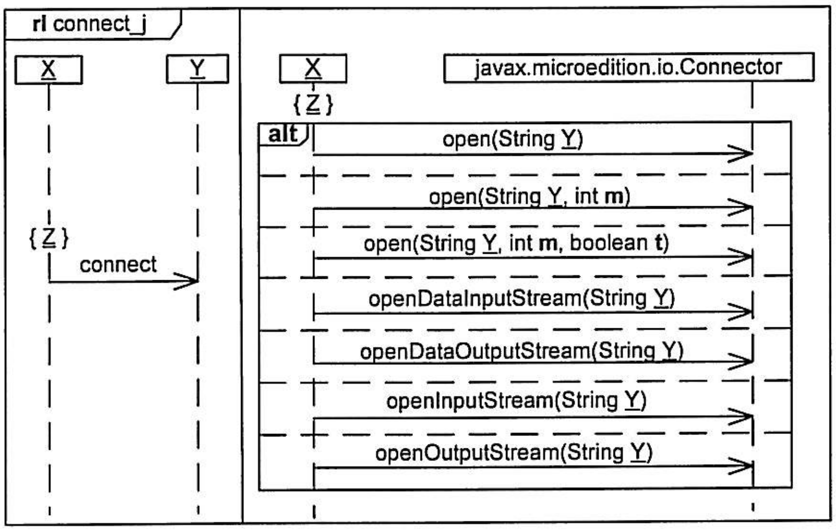
The policies described in the previous section are not enforceable since the behavior of the target is not expressed in terms of API-calls that can be made to the MIDP runtime environment. Recall the policy of Fig. 1. It has a single message connect which represents an attempt to open a connection. In order to make the policy enforceable, we need to express this behavior in terms of API-calls that can be made to the runtime environment.

Fig. 3 illustrates a transformation rule which describes how the connect message is transformed into the API-calls which can be made in order to establish a connection via the MIDP runtime environment. The diagram on the left in Fig. 3 represents the left hand side pattern of the rule, and the diagram on the right represents the right hand side pattern of the rule. In the diagram, all meta variables are underlined.

Fig. 4 shows the result of applying the rule of Fig. 3 to the policy of Fig. 1. Here we see that the message connect has been replaced by the relevant API-calls of the runtime environment.

Clearly, the high level policy of Fig. 1 allows for an easier comparison to the natural language description than the low level policy of Fig. 4. Moreover, if the high level policy of Fig. 1 should be applied to a different runtime environment than MIDP, say .NET, then a similar transformation rule can be written or selected from a transformation library without changing the high level policy.

Notice that each message of Fig. 4 contains one or more variables that are not explicitly assigned to any value it the diagram. We call these *parameter variables*, and distinguish these from normal variables by writing them in bold-



rl connect_j
X
Y
{Z}
connect
X
{Z}
javax.microedition.io.Connector
alt
open(String Y)
open(String Y, int m)
open(String Y, int m, boolean t)
openDataInputStream(String Y)
openDataOutputStream(String Y)
openInputStream(String Y)
openOutputStream(String Y)

Figure 3: Transformation rule

face. Parameter variables are bound to an arbitrary value upon the occurrence of the message they are contained in. Parameter variables differ from normal variables that have not been assigned to a value in that parameter variables contained in a loop are bound to new arbitrary values for each iteration of the loop.

The introduction of parameter variables constitutes an extension of the UML 2.1 sequence diagram standard. This extension is necessary in order to express policies for execution mechanisms that monitor method calls where the actual arguments of the method call are not known until the method call is intercepted. Without parameter variables, one would in many cases be forced to specify all possible actual arguments that a method call can have. This is clearly not feasible.

4 Step III: Transforming sequence diagrams to state machines

In the third step of our method, the low level intermediate sequence diagram obtained from step II is automatically transformed into a set of state machines (one for each lifeline of the diagram) that govern the behavior of an EM mechanism. The state machines explicitly describe the (positive) behavior which is allowed by a system. Everything which is not described by the state machines is (negative) behavior which is not allowed. Therefore, the state machines do not have a notion of inconclusive behavior as sequence diagrams do.

The semantics of a state machine is a set of traces describing positive behavior. A system adheres to a set of state machines S if each trace described by the system is also described by a state machine in S (when the trace is restricted to the alphabet of the state machine). We define adherence like this because this

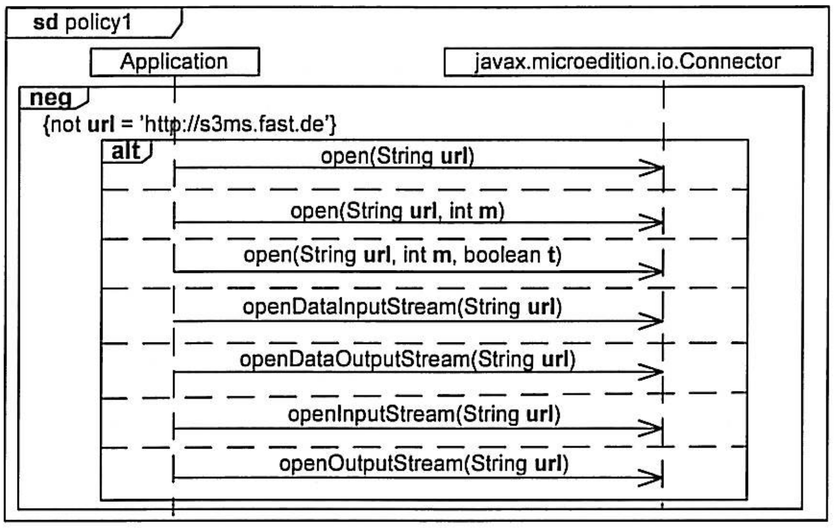

sd policy1
Application
javax.microedition.io.Connector
neg
{not url = 'http://s3ms.fast.de'}
alt
open(String url)
open(String url, int m)
open(String url, int m, boolean t)
openDataInputStream(String url)
openDataOutputStream(String url)
openInputStream(String url)
openOutputStream(String url)

Figure 4: Low level policy 1

notion of adherence is used by existing EM mechanisms (see e.g. [8, 16]) that are governed by state machines.

The transformation of step III should convert a sequence diagram into a set of state machine such that an arbitrary system adheres to the state machine set if and only if the system adheres to the sequence diagram, i.e., the transformation should be adherence preserving. As one would except, the transformation converts positive and negative behavior of the sequence diagram into positive and negative behavior of the state machines, respectively. However, the inconclusive behavior of the sequence diagram is converted to positive behavior of the state machines. This is because, by definition of adherence for sequence diagrams, a system is allowed to engage in the (inconclusive) behavior which is not described by the sequence diagram.

The only kind of policies that can be enforced by EM mechanisms are so-called prohibition policies, i.e., policies that describe what a system is *not* allowed to do. Sequence diagrams are suitable for specifying these kinds of policies because they have a construct for specifying explicit negative behavior. For this reason, the sequence diagram policies are often more readable than the corresponding state machine policies since the state machine policies can only explicitly describe behavior which is allowed by an application. In the following examples, we clarify this point and explain how the transformation from sequence diagrams to state machines works.

4.1 Example – transforming policy 2 to a state machine

Although our method is intended to transform low level intermediate sequence diagram policies to state machines, we will use the sequence diagram of Fig. 2 to illustrate the transformation process as this diagram better highlights the transformation phases than the low level intermediate policy of Fig. 4.

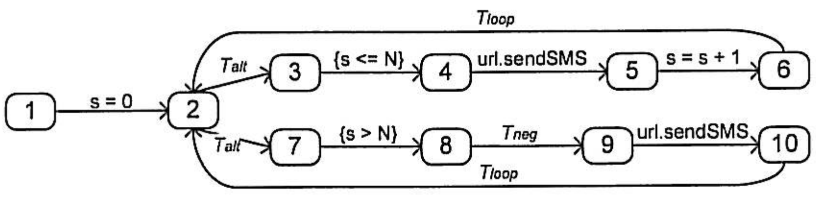


Figure 5: Projection system of policy 2

In general, the transformation from a sequence diagram yields one state machine for each lifeline of the diagram. However, in the current example, the target of the policy is the lifeline Application. Thus we only consider the transformation of this lifeline.

The transformation from a sequence diagram d with one lifeline to a state machine has two phases. In phase 1, the sequence diagram d is transformed into a state machine SM whose trace semantics equals the negative trace set of d. In phase 2, SM is inverted into the state machine SM' whose semantics is the set of all traces that do not have a trace of SM as a sub-trace.

In the following we explain the two phases by transforming a diagram describing lifeline Application in Fig. 2 into a state machine.

4.1.1 Phase 1

First, in phase 1, we transform the sequence diagram describing the lifeline Application into a state machine whose trace semantics equals the negative traces of the diagram. To achieve this, we make use of the operational semantics of sequence diagrams which is based on [13, 12] and described in App. A.

The operational semantics makes use of a so-called *projection system* for finding enabled events and constructs in a diagram. The projection system is a labeled transition system (LTS) whose states are diagrams, and whose transitions are labeled by events, constraints, assignments, and so-called silent events that indicate which kind of operation has been executed. If the labeled transition system has a transition from a diagram d to a diagram d' that is labeled by, say, event e, then we understand that event e is enabled in diagram d, and that d' is obtained by removing e from d.

To transform a diagram describing Application into a state machine describing its negative traces, we first construct the projection system whose states are exactly those that can be reached from the diagram. The result is illustrated in Fig.5 (note that the labels τ_{alt}, τ_{loop}, and τ_{neg} are silent events that correspond to the sequence diagram constructs alt, loop, and neg, respectively). Then, the projection system is transformed into a state machine describing the negative traces of the diagram. The states of this state machine are of the form (Q, mo) where Q is a set of states of the projection system and mo is a *mode* which is used to differentiate between positive and negative traces. There are two kinds of modes: *pos* (for positive) or *neg* (for negative). A state with mode *neg* leads to a final state that accepts negative traces. Thus we require that all final states have mode *neg*.

When converting the projection system into a state machine, we remove all silent events and concatenate constraints with succeeding events and assign-

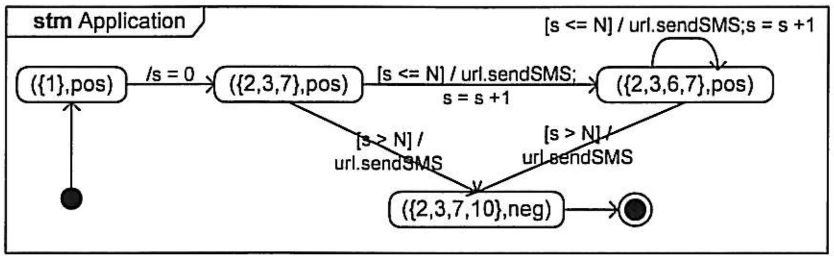


Figure 6: State machine describing negative traces for Application

ments with preceding events. In addition, the silent event τ_{neg} is used to find negative executions. That is, any execution that involves a τ_{neg} represents a negative execution that lead to states having mode *neg*.

The result of converting the projection system of Fig. 5 into a basic state machine is illustrated in Fig. 6. Here the black filled circle represents an initial state, the boxes with rounded edges represent simple states, and the black filled circle encapsulated by another circle represents a final state. Transitions of state machines are labeled by *action expressions* of the form $nm.si[bx]/ef$. Here $nm.si$ (where si denotes a signal and nm denotes the name of the state machine the signal is received from) is called an *input event*, $[bx]$ (where bx is a boolean expression) is called a *guard*, and ef is called an *effect*. An effect is a sequence of assignments and/or an output event.

The graphical notation used for specifying state machines is inspired by the UML statechart diagram notion. See App. B for more details.

The alphabet of a state machine is a set of events. When we transform a sequence diagram to a state machine, the alphabet of the state machine is the set of all events that occur in the sequence diagram.

In App.D.1.1, we formalize the transformation of phase 1. We also prove that this transformation is correct in the sense the state machine describes exactly the negative traces of the sequence diagram it is transformed from.

4.1.2 Phase 2

In phase 2, we "invert" the state machine of Fig. 6 into the state machine whose semantics is the set of all traces that do not have a trace of the state machine of Fig. 6 as a sub-trace. In general, the inversion SM' of a state machine SM has the power-set of the states in SM as its states[2] the alphabet of SM as its alphabet, all its states as its final states, and its transitions are those that are constructed by the following rule

- if Q is a state of SM', then SM' has a transition $Q \xrightarrow{act} Q \cup Q'$ where Q' represents the set of non-final states of SM that are targeted by an outgoing transition of a state in Q whose action expression contains the same event as act.

[2]Each state of SM' also needs to keep track of the transitions already visited in SM to reach that state. We postpone the discussion of this technical detail to App. D

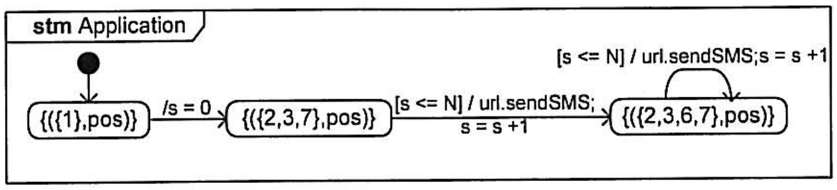


Figure 7: State machine describing positive and inconclusive traces for Application

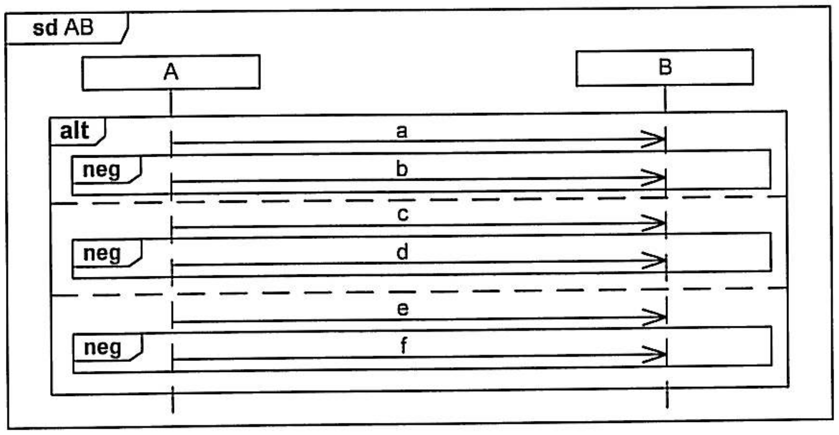


Figure 8: Composition of three sequence diagram policies

Note that the rule is slightly simplified as a lengthy presentation of all the technical details of the rule is given in App. D.1.2. In the appendix, we also show that the transformation of phase 2 correctly inverts policies that may be seen as compositions of (sub)policies with disjoint sets of variables. The state machine policy of Fig. 6, can not be seen as a composition of more than one policy. Therefore the condition of disjoint variables is trivially satisfied for this state machine.

Returning back to our running example, Fig. 7 shows the inversion of the state machine of Fig. 6. Since all states in the state machine of Fig. 7 are final, we have omitted to specify the final states.

We have that every trace that contain less than or equal to N occurrences of the sendSMS message are accepted by the state machine of Fig. 7. Indeed, this was the intended meaning of the policy.

In App.D.2, we show that the composition of the transformations of phase 1 and phase 2 is adherence preserving when the condition under which the transformation of phase 2 yields a correct inversion is satisfied.

4.2 Example – why the negation construct is useful

As noted in the beginning of this section, the sequence diagram construct for specifying explicit negative behavior is useful when specifying policies that can be enforced by EM mechanisms. In this section, we illustrate this with an

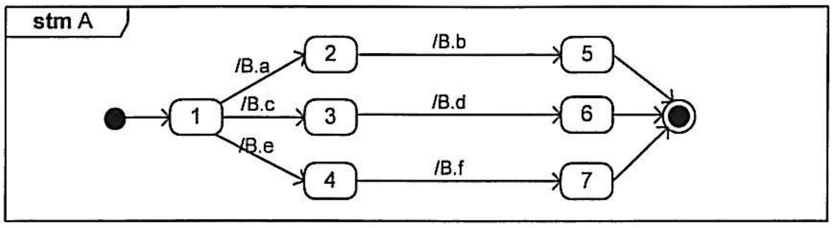


Figure 9: State machine describing the negative traces lifeline A

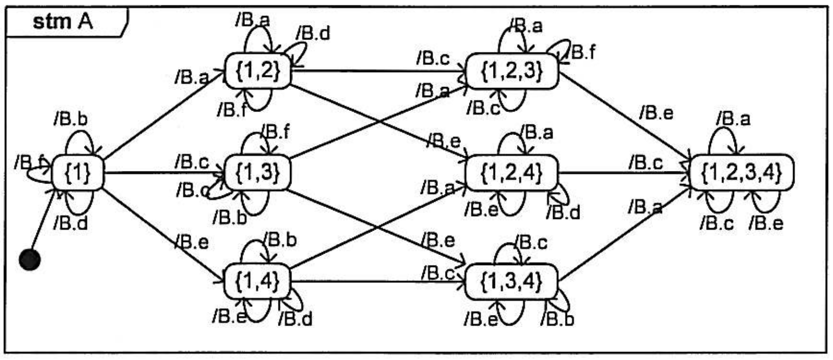


Figure 10: State machine describing the positive and inconclusive traces of lifeline A

example.

Consider the policy shown in Fig. 8. It may be seen as a composition of three policies. The upper most policy states that after the lifeline A has transmitted a, it is not allowed to transmit b. The two other policies are similar except that the messages are different.

To transform a diagram describing the lifeline A into a state machine, we first (in phase 1) construct the state machine that describes the negative traces of the diagram. The resulting state machine is shown in Fig. 9. Then, we invert the state machine of Fig. 9 to obtain the state machine of Fig. 10.

Clearly, it is more difficult to understand the meaning of the state machine policy of Fig. 10 than the sequence diagram policy of Fig. 8. The reason for this is that the state machine policy have to describe all behavior which is allowed. However, the process of inverting a state machine may lead an increase in the number of states and transitions. This shows why it is useful to have a construct for specifying negative behavior.

5 Related work

Previous work that address the transformation of policies or security requirements are [1, 2, 3, 4, 6, 7, 11, 14, 15, 21]. All these differ clearly from ours in that the policy specifications, transformations, and enforcement mechanisms are different from the ones considered in this report.

Bai and Varadharajan [1] consider authorization policies, Satoh and Yamaguchi [15] consider security policies for Web Services, Patz et. al. [14] consider policies in the form of logical conditions, and Beigi et.al. [4] focuses on transformation techniques rather than any particular kind policies. The remaining citations ([2, 3, 6, 7, 11, 21]) all address policies in the form of access-control requirements.

Of the citations above, [3] gives the most comprehensive account of policy transformation. In particular it shows how platform independent role based access control requirements can be expressed in UML diagrams, and how these requirements can be transformed to platform specific access control requirements.

The transformation of sequence diagrams (or a similar language) to state machines has been previously addressed in [5, 9, 22]. However, these do not consider policies, nor do they offer a way of changing the granularity of interactions during transformation.

The only paper that we are aware of that considers UML sequence diagrams for policy specification is [20]. However, in that paper, transformations from high- to low level policies or transformation to state machines is not considered. The paper argues that sequence diagrams must be extended with customized expressions for deontic modalities to support policy specification. While this is true in general, this is not needed for the kind of prohibition policies that can be enforced by EM mechanisms.

6 Conclusions

We claim that it is desirable to automate as much as possible of the process of formalizing security policies. To this end we have presented a method which (1) supports the formalization of policies at a high level of abstraction, (2) offers automatic generation of low level policies from high level policies, and (3) facilitates automatic enforcement by monitoring of low level policies. Enforcement mechanisms for the kind of policies considered in this report have been developed in the S^{3}MS EU project[17]. Thus the method fulfills the first three requirements that were presented in the Sect. 1. Empirical investigation of whether the method satisfies the fourth requirement, namely that it should be easy to understand by software developers, is beyond the scope of this report. However, we have used UML as a policy language, and using UML for specifying policies, we believe, is not much harder than using UML to specify software systems (in particular, since we focus on execution monitoring and do not have to take other modalities than prohibition into consideration).

In the appendices, we provide a formal foundation for our method. In particular, we define the semantics of sequence diagrams and state machines, and we precisely define what it means that a system adheres to a sequence diagram policy as well as a state machine policy. We also formalize the transformation from high to low level sequence diagrams, and the transformation from sequence diagrams to state machines. Finally, we prove that the transformation from sequence diagram policies to state machine policies is adherence preserving under a certain condition. All examples of this report satisfies this condition.

Previous work in the literature has addressed policy transformation, but differ clearly from ours in that the policy specifications, transformations, and enforcement mechanisms are different.

Acknowledgement

The research of this report has partly been funded by the Research Council of Norway project SECURIS (152839/220) and DIGIT (180052/S10), and partly by the EU through the S^3MS (contract no. 27004) project.

References

[1] Y. Bai and V. Varadharajan. On transformation of authorization policies. *Data and Knowledge Engineering*, 45(3):333–357, 2003.

[2] D. Basin, J. Doser, and T. Lodderstedt. Model driven security for process-oriented systems. In *Proc. of the 11th ACM Symposium on Access Control Models and Technologies (SACMAT'03)*, pages 100–109. ACM, 2003.

[3] D. Basin, J. Doser, and T. Lodderstedt. Model driven security: From UML models to access control infrastructures. *ACM Transactions on Software Engineering Methodologies*, 15(1):39–91, 2006.

[4] M. Beigi, S. B. Calo, and D. C. Verma. Policy transformation techniques in policy-based systems management. In *Proc. of the 5th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2004)*, pages 13–22. IEEE Computer Society, 2004.

[5] Y. Bontemps, P. Heymans, and P. Schobbens. From live sequence charts to state machines and back: A guided tour. *IEEE Transactions on Software Engineering*, 31(12):999–1014, 2005.

[6] C. C. Burt, B. R. Bryant, R. R. Raje, A. M. Olson, and M. Auguston. Model driven security: unification of authorization models for fine-grain access control. In *Proc. of the 7th International Enterprise Distributed Object Computing Conference (EDOC'03)*, pages 159–173. IEEE Computer Society, 2003.

[7] E. Fernández-Medina and M. Piattini. Extending OCL for secure database development. In *Proc. of the 7th International Conference on The Unified Modelling Language: Modelling Languages and Applications (UML'04)*, volume 3273 of *Lecture Notes in Computer Science*, pages 380–394. Springer, 2004.

[8] I. Ion, B. Dragovic, and B. Crispo. Extending the java virtual machine to enforce fine-grained security policies in mobile devices. In *Proc. of the 23rd Annual Computer Security Applications Conference (ACSAC'07)*, pages 233–242. IEEE Computer Society, 2007.

[9] J. Klose and H. Wittke. An automata based interpretation of live sequence charts. In *Proc. of the 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'01)*, pages 512–527. Springer, 2001.

[10] L. Lamport. How to write a long formula. *Formal Aspects of Computing*, 6(5):580–584, 1994.

[11] T. Lodderstedt, D. Basin, and J. Doser. SecureUML: A UML-based modeling language for model-driven security. In *Proc. of the 5th International Conference on The Unified Modeling Language (UML'02)*, volume 2460 of *Lecture Notes in Computer Science*, pages 426–441. Springer, 2002.

[12] M. S. Lund. *Operational analysis of sequence diagram specifications.* PhD thesis, Faculty of Mathematics and Natural Sciences, University of Oslo, 2008.

[13] M. S. Lund and K. Stølen. A Fully General Operational Semantics for UML 2.0 Sequence Diagrams with Potential and Mandatory Choice. In *Proc. of the 14th International Symposium on Formal Methods (FM'06)*, volume 4085 of *Lecture Notes in Computer Science*, pages 380–395. Springer, 2006.

[14] G. Patz, M. Condell, R. Krishnan, and L. Sanchez. Multidimensional security policy management for dynamic coalitions. In *Proc. of the DARPA Information Survivability Conference and Exposition (DISCEX II'01)*, volume 02. IEEE Computer Society, 2001.

[15] F. Satoh and Y. Yamaguchi. Generic security policy transformation framework for ws-security. In *Proc. of the 2007 IEEE International Conference on Web Services (ICWS'07)*, pages 513–520. IEEE Computer Society, 2007.

[16] F. B. Schneider. Enforceable security policies. *ACM Transactions on Information and System Security*, 3(1):30–50, 2000.

[17] Security of Software and Services of Mobile Systems (S^{3}MS). *IST-FP6-STREP-27004, www.s3ms.org (accessed 21. August 2008).*

[18] F. Seehusen, B. Solhaug, and K. Stølen. Adherence preserving refinement of trace-set properties in STAIRS: exemplified for information flow properties and policies. *Journal of Software and Systems Modeling*, 8(1):45–65, 2009.

[19] M. Sloman. Policy driven management for distributed systems. *Journal of Network and Systems Management*, 2:333–360, 1994.

[20] B. Solhaug, D. Elgesem, and K. Stølen. Specifying Policies Using UML Sequence Diagrams–An Evaluation Based on a Case Study. In *Proc. of the 8th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'07)*, pages 19–28. IEEE Computer Society, 2007.

[21] B. Vela, E. Fernández-Medina, E. Marcos, and M. Piattini. Model driven development of secure XML databases. *SIGMOD Record*, 35(3):22–27, 2006.

[22] J. Whittle and J. Schumann. Generating statechart designs from scenarios. In *Proc. of the 22nd international conference on Software engineering (ICSE'00)*, pages 314–323. ACM, 2000.

A UML sequence diagrams

In this section, we first present the syntax (Sect. A.1) and semantics (Sect. A.2) of UML sequence diagrams. Then, in Sect. A.3, we define what it means for a system to adhere to a sequence diagram policy.

A.1 Syntax

We use the following syntactic categories to define the textual representation of sequence diagrams:

ax	$\in$	**AExp**	arithmetic expressions
bx	$\in$	**BExp**	boolean expressions
sx	$\in$	**SExp**	string expressions

We let **Exp** denote the set of all arithmetic, boolean, and string expressions, and we let ex range over this set. We denote the empty expression by ϵ. We let **Val** denote the set of all values, i.e., numerals, strings, and booleans (`t` or `f`) and we let **Var** denote the set of all variables. Obviously, we have that $\mathbf{Val} \subset \mathbf{Exp}$ and $\mathbf{Var} \subset \mathbf{Exp}$.

Every sequence diagram is built by composing atoms or sub-diagrams. The atoms of a sequence diagram are the *events*, *constraints*, and the *assignments*. These constructs are presented in Sect. A.1.1, while the syntax of sequence diagrams in general is presented in Sect. A.1.2. Finally, in Sect. A.1.3, we present syntax constraints for sequence diagrams.

A.1.1 Events, constraints, and assignments

The atoms of a sequence diagram are the *events*, *constraints*, and the *assignments*. An event is a pair (k, m) of a kind k and a message m. An event of the form $(!, m)$ represents a transmission of message m, whereas an event of the form $(?, m)$ represents a reception of m. We let **E** denote the set of all events:

$$\mathbf{E} \overset{\text{def}}{=} \{!, ?\} \times \mathbf{M} \tag{1}$$

where **M** denotes the set of all messages.

On events, we define a kind function $k.\_ \in \mathbf{E} \rightarrow \{!, ?\}$ and a message function $m.\_ \in \mathbf{E} \rightarrow \mathbf{M}$:

$$k.(k, m) \overset{\text{def}}{=} k \qquad m.(k, m) \overset{\text{def}}{=} m \tag{2}$$

Messages are of the form (l_t, l_r, si) where l_t represents the transmitter lifeline of the message, l_r represents the receiver lifeline of the message, and si represents the signal of the message. We let **L** denote the set of all lifelines, and **SI** denote the set of all signals. The set **M** of all messages is then defined by

$$\mathbf{M} \overset{\text{def}}{=} \mathbf{L} \times \mathbf{L} \times \mathbf{SI} \tag{3}$$

On messages, we define a transmitter function $tr.\_ \in \mathbf{M} \rightarrow \mathbf{L}$ and a receiver function $re.\_ \in \mathbf{M} \rightarrow \mathbf{L}$:

$$tr.(l_t, l_r, si) \overset{\text{def}}{=} l_t \qquad re.(l_t, l_r, si) \overset{\text{def}}{=} l_r \tag{4}$$

We let the transmitter and receiver functions also range over events, $tr.\_, re.\_ \in \mathbf{E} \rightarrow \mathbf{L}$:

$$tr.(k,m) \stackrel{\text{def}}{=} tr.m \qquad re.(k,m) \stackrel{\text{def}}{=} re.m \tag{5}$$

We define a lifeline function $l.\_ \in \mathbf{E} \rightarrow \mathbf{L}$ that returns the lifeline of an event and a function $l^{-1}.\_ \in \mathbf{E} \rightarrow \mathbf{L}$ that returns the inverse lifeline of an event (i.e., the receiver of its message if its kind is transmit and the transmitter of its message if its kind is receive):

$$l.e \stackrel{\text{def}}{=} \begin{array}{ll} tr.e & \text{if } k.e = ! \\ re.e & \text{if } k.e = ? \end{array} \qquad l^{-1}.e \stackrel{\text{def}}{=} \begin{array}{ll} tr.e & \text{if } k.e = ? \\ re.e & \text{if } k.e = ! \end{array} \tag{6}$$

A signal is a tuple $(nm, ex_1, \ldots, ex_n)$ where nm denotes the signal name, and $ex_1, \ldots, ex_n$ are the parameters of the signal. We usually write $nm(ex_1, \ldots, ex_n)$ instead of $(nm, ex_1, \ldots, ex_n)$. Formally, the set of all signals is defined

$$\mathbf{SI} \stackrel{\text{def}}{=} \mathbf{Nm} \times \mathbf{Exp}^* \tag{7}$$

where A^* yields the set of all sequences over the elements in the set A.

A signal may contain special so-called *parameter variables* that are bound to values upon the occurrence of the signal. Parameter variables are similar to free normal variables (normal variables that have not explicitly been assigned to a value). However, they differ in that parameter variables contained in a loop will be assigned to new values for each iteration of the loop.

A parameter variable is a pair (vn, i) consisting of variable name vn and an index i (this is a natural number). When a sequence diagram is executed, the index of a parameter variable contained in a loop will be incremented by one for each iteration of the loop. This is to ensure that the parameter variable is given a new value when the loop is iterated. Hence, the index of a parameter variable is only used for bookkeeping purposes during execution, and it will never be explicitly specified in a graphical diagram.

In a graphical sequence diagram, parameter variables are distinguished from normal variables by writing the parameter variables in boldface. The index of a parameter variable in a graphical sequence diagram is always initially assumed to be zero.

The set of all parameter variables **PVar** is defined

$$\mathbf{PVar} \stackrel{\text{def}}{=} \mathbf{VN} \times \mathbb{N} \tag{8}$$

where **VN** is the set of all variable names and $\mathbb{N}$ is the set of all natural numbers. We assume that

$$\mathbf{PVar} \subset \mathbf{Var} \tag{9}$$

A *constraint* is an expression of the form

$$\mathsf{constr}(bx, l)$$

where bx is a boolean expression and l is a lifeline. Intuitively, interactions occurring after a constraint in a diagram will only take place if and only if the boolean expression of the constraint evaluates to true. We denote the set of all constraints by **C** and we let c range over this set.

An *assignment* is an expression of the form

$$\mathsf{assign}(x, ex, l)$$

where x is a normal variable, i.e, $x \in \mathbf{Var} \setminus \mathbf{PVar}$, ex is an expression, and l is a lifeline. Intuitively, the assignment represents the binding of expression ex to variable x on lifeline l. We let $\mathbf{A}$ denote the set of all assignments and we let a range over this set.

We define the function $l.\_ \in \mathbf{A} \cup \mathbf{C} \rightarrow \mathbf{L}$ which yields the lifeline of an assignment or constraint as follows

$$l.\mathsf{constr}(bx, l) \overset{\text{def}}{=} l \qquad l.\mathsf{assign}(x, ex, l) \overset{\text{def}}{=} l \tag{10}$$

We denote by $\mathbf{E}^l$, $\mathbf{C}^l$, and $\mathbf{A}^l$, the set of all events, constraints, and assignments with lifeline l, respectively, i.e.,

$$\mathbf{E}^l \overset{\text{def}}{=} \{e \in \mathbf{E} \mid l.e = l\} \quad \mathbf{C}^l \overset{\text{def}}{=} \{c \in \mathbf{C} \mid l.c = l\} \quad \mathbf{A}^l \overset{\text{def}}{=} \{a \in \mathbf{A} \mid l.a = l\} \tag{11}$$

A.1.2 Diagrams

In the previous section, we presented the atomic constructs of a sequence diagram. In this section, we present the syntax of sequence diagrams in general.

Definition 1 (Sequence diagram) *Let e, bx, l, x, and ex denote events, boolean expressions, lifelines, variables, and expressions, respectively. The set of all syntactically correct sequence diagram expressions $\mathbf{D}$ is defined by the following grammar:*

$$\begin{aligned} d \ ::= \ & \mathsf{skip} \mid e \mid \mathsf{constr}(bx, l) \mid \mathsf{assign}(x, ex, l) \mid \mathsf{refuse}\,(d) \mid \mathsf{loop}\langle 0..* \rangle\,(d) \mid \\ & d_1\,\mathsf{seq}\,d_2 \mid d_1\,\mathsf{alt}\,d_2 \mid d_1\,\mathsf{par}\,d_2 \end{aligned}$$

The base cases implies that any event (e), skip, constraint ($\mathsf{constr}(bx, l)$), or assignment ($\mathsf{assign}(x, ex, l)$) is a sequence diagram. Any other sequence diagram is constructed from the basic ones through the application of operators for negation ($\mathsf{refuse}\,(d)$), iteration ($\mathsf{loop}\langle 0..* \rangle\,(d)$), weak sequencing ($d_1\,\mathsf{seq}\,d_2$), choice ($d_1\,\mathsf{alt}\,d_2$), and parallel execution ($d_1\,\mathsf{par}\,d_2$).

We define some functions over the syntax of diagrams. We let the function $eca.\_ \in \mathbf{D} \rightarrow \mathbb{P}(\mathbf{E} \cup \mathbf{C} \cup \mathbf{A})$ return all events, constraints, and assignments present in a diagram. The function is defined as follows

$$\begin{array}{lll l} eca.\alpha & \overset{\text{def}}{=} \{\alpha\} & \text{for } \alpha \in \mathbf{E} \cup \mathbf{C} \cup \mathbf{A} \\ eca.\mathsf{skip} & \overset{\text{def}}{=} \emptyset & \\ eca.(\mathsf{op}(d)) & \overset{\text{def}}{=} eca.d & \text{for } \mathsf{op} \in \{\mathsf{refuse}, \mathsf{loop}\langle 0..* \rangle\} \\ eca.(d_1\,\mathsf{op}\,d_2) & \overset{\text{def}}{=} eca.d_1 \cup eca.d_2 & \text{for } \mathsf{op} \in \{\mathsf{seq}, \mathsf{alt}, \mathsf{par}\} \end{array} \tag{12}$$

Note that we henceforth let α denote an arbitrary event, constraint, or assignment, i.e., $\alpha \in \mathbf{E} \cup \mathbf{C} \cup \mathbf{A}$.

The function $ll.\_ \in \mathbf{D} \rightarrow \mathbb{P}(\mathbf{L})$ returns all lifelines of a diagram:

$$ll.d \overset{\text{def}}{=} \bigcup_{\alpha \in eca.d} \{l.\alpha\} \tag{13}$$

We denote by $\mathbf{D}^l$, the set of all diagrams with only one lifeline l, i.e.,

$$\mathbf{D}^l \overset{\text{def}}{=} \{d \in \mathbf{D} \mid ll.d = \{l\}\} \tag{14}$$

The function $msg.\_ \in \mathbf{D} \rightarrow \mathbb{P}(\mathbf{M})$ returns all the messages of a diagram:

$$msg.d \overset{\text{def}}{=} \bigcup_{e \in (eca.d \cap \mathbf{E})} \{m.e\} \tag{15}$$

The projection operator $\pi_{\_}(\_) \in \mathbf{L} \times \mathbf{D} \rightarrow \mathbf{D}$ that projects a diagram to a lifeline is defined

$$\begin{array}{lll} \pi_l(\alpha) & \overset{\text{def}}{=} \alpha & \text{if } l.\alpha = l \\ \pi_l(\alpha) & \overset{\text{def}}{=} \mathsf{skip} & \text{if } l.\alpha \neq l \\ \pi_l(\mathsf{skip}) & \overset{\text{def}}{=} \mathsf{skip} & \\ \pi_l(\mathsf{op}\, d) & \overset{\text{def}}{=} \mathsf{op}(\pi_l(d)) & \text{for } \mathsf{op} \in \{\mathsf{refuse}\,,\mathsf{loop}\langle 0..^*\rangle\,\} \\ \pi_l(d_1\, \mathsf{op}\, d_2) & \overset{\text{def}}{=} \pi_l(d_1)\, \mathsf{op}\, \pi_l(d_2) & \text{for } \mathsf{op} \in \{\,\mathsf{seq}\,,\,\mathsf{alt}\,,\,\mathsf{par}\,\} \end{array} \tag{16}$$

We let $var \in (\mathbf{Exp} \cup \mathbf{M}) \rightarrow \mathbb{P}(\mathbf{Var})$ be the function that yields the variables in an expression or the variables in the arguments of a signal of a message. We lift the function to diagrams as follows

$$\begin{array}{ll} var(d) & \overset{\text{def}}{=} \bigcup_{m \in msg.d} var(m) \cup \bigcup_{\mathsf{constr}(bx,l) \in eca.d \cap \mathbf{C}} var(bx) \cup \\ & \bigcup_{\mathsf{assign}(x,ex,l) \in eca.d \cap \mathbf{A}} (\{x\} \cup var(ex)) \end{array} \tag{17}$$

A.1.3 Syntax constraints

We impose some restrictions on the set of syntactically correct sequence diagrams $\mathbf{D}$. We describe four rules which are taken from [12]. First, we assert that a given event should syntactically occur only once in a diagram. Second, if both transmitter and the receiver lifelines of a message are present in a diagram, then both the transmit event and the receive event of that message must be in the diagram. Third, if both the transmit event and the receive event of a message are present in a diagram, then they have to be inside the same argument of the same high level operator. The constraint means that in the graphical notion, messages are not allowed to cross the frame of a high level operator or the dividing line between the arguments of a high level operator. Fourth, the operator refuse is not allowed to be empty, i.e., to contain only the skip diagram.

The four rules described above are formally defined in [12]. These rules ensure that the operational semantics is sound and complete with the denotational semantics of sequence diagrams as defined in [12]. In this report, we define nine additional rules and we say that a diagram d is *well formed* if it satisfies these:

SD1 The variables of the lifelines of d are disjoint.

SD2 All parameter variables of d have index 0.

SD3 If m is a message in d, then the arguments of the signal of m must be distinct parameter variables only.

SD4 The first atomic construct of each lifeline in d must be an assignment (not a constraint or an event).

SD5 All parameter variables that occur inside a loop in d do not occur outside that loop.

SD6 All loops in d must contain at least one event.

SD7 No two events in d contain the same parameter variables.

SD8 For each lifeline in d, each constraint c must be followed by an event e (not an assignment or a constraint). In addition, the parameter variables of c must be a subset of the parameter variables of e.

SD9 For each lifeline in d, the parameter variables of an assignment must be a subset of parameter variables of each event that proceeds it on the lifeline. If the assignment has no proceeding events on the lifeline, then the assignment cannot contain parameter variables.

SD10 All variables in d (except for the parameter variables) must explicitly be assigned to a value before they are used.

The purpose of the syntax constraints is to ensure that the sequence diagram can be correctly transformed into a state machine.

Note that any graphical sequence diagram can be described by a textual diagram that satisfies conditions **SD1** - **SD4**.

To obtain a diagram that satisfies **SD1** and **SD2** we have to rename variables on each lifeline and set the index of all parameter variables to zero. To obtain a diagram that satisfies condition **SD3** we convert arguments (that are not parameter variables) of the signal of an event into constraints proceeding the event. For instance, the diagram

$$(!, (l_t, l_r, msg(ex)))$$

– which does not satisfy **SD3** because ex might not be a parameter variable – can be converted into the diagram

$$\mathsf{constr}(px{=}ex, l_t)\ \mathsf{seq}\ (!, (l_t, l_r, msg(px))) \qquad \text{for some } px \in \mathbf{PVar}$$

which does satisfy **SD3**. Here $px{=}ex$ is a boolean expression that yields true if and only if px is equal to ex.

If a sequence diagram d does not satisfy condition **SD4**, then a dummy assignment can be added to start of each lifeline in d that assigns some value to a variable that is not used in d.

A.2 Semantics

In this section, we define the operational semantics of UML sequence diagrams based on the semantics defined in [12]. The operational semantics tells us how a sequence diagram is executed step by step. It is defined as the combination of two labeled transition systems, called the *execution system* and the *projection system*.

These two systems work together in such a way that for each step in the execution, the projection system updates the execution system by selecting an enabled event to execute and returning the state of the diagram after the execution of the event.

A.2.1 The projection system

The projection system is used for finding enabled events at each step of execution. The projection system (as well as the execution system) is formally described by a labeled transition system (LTS).

Definition 2 (Labeled transition system (LTS)) *A labeled transition system over the set of labels LE is a pair $(\mathcal{Q}, \mathcal{R})$ consisting of*

- *a (possibly infinite) set $\mathcal{Q}$ of states;*
- *a ternary relation of $\mathcal{R} \subseteq (\mathcal{Q} \times LE \times \mathcal{Q})$, known as a transition relation.*

We usually write $q \xrightarrow{le} q' \in (\mathcal{Q}, \mathcal{R})$ if $(q, le, q') \in \mathcal{R}$, or just $q \xrightarrow{le} q'$ if $(\mathcal{Q}, \mathcal{R})$ is clear from the context. If $s = \langle le_1, le_2, \ldots, le_n \rangle$, we write $q \xrightarrow{s} q'$ for $q \xrightarrow{le_1} q_1 \xrightarrow{le_2} q_2 \cdots \xrightarrow{le_n} q'$. For the empty sequence $\langle\rangle$, we write $q \xrightarrow{\langle\rangle} q'$ iff $q = q'$.

To define the projection system, we make use of a notion of structural congruence which defines simple rules under which sequence diagrams should be regarded as equivalent.

Definition 3 (Structural congruence) *Structural congruence over sequence diagrams, written $\equiv$, is the congruence over $\mathbf{D}$ determined by the following equations:*

1. $d\,\mathsf{seq}\,\mathsf{skip} \equiv d$, $\mathsf{skip}\,\mathsf{seq}\,d \equiv d$
2. $d\,\mathsf{par}\,\mathsf{skip} \equiv d$, $\mathsf{skip}\,\mathsf{par}\,d \equiv d$
3. $\mathsf{skip}\,\mathsf{alt}\,\mathsf{skip} \equiv \mathsf{skip}$
4. $\mathsf{loop}\langle 0..{*}\rangle\,(\mathsf{skip}) \equiv \mathsf{skip}$

The projection system is an LTS whose states are pairs $\Pi(L, d)$ consisting of a set of lifelines L and a diagram d. If the projection system has a transition from $\Pi(L, d)$ to $\Pi(L, d')$ that is labeled by, say event e, then we understand that e is enabled in diagram d, and that d' is obtained from d by removing event e. Whenever the high level construct alt, refuse, or loop is enabled in a diagram, the projection system will produce a so-called silent event that indicates the kind of construct that has been executed. For instance, each state of the form $\Pi(L, \mathsf{refuse}\,(d))$ has a transition to $\Pi(L, d)$ that is labeled by the silent event τ_{refuse}.

The set of lifelines L that appears in the states of the projection system is used to define the transition rules of the weak sequencing operator seq. The weak sequencing operator defines a partial order on the events in a diagram; events are ordered on each lifeline and ordered by causality, but all other ordering of events is arbitrary. Because of this, there may be enabled events in both the left and the right argument of a seq if there are lifelines present in the right argument of the operator that are not present in the left argument. The set of lifelines L is used to keep track of which lifelines are shared by the arguments of seq, and which lifelines only occur in the right argument (but not the left) of seq.

The following definition of the projection system is based on [12].

Definition 4 (Projection system) *The projection system is an LTS over*

$$\alpha_\tau \in \{\tau_{refuse}, \tau_{alt}, \tau_{loop}\} \cup \mathbf{E} \cup \mathbf{C} \cup \mathbf{A}$$

whose states are

$$\Pi(\_,\_) \in \mathbb{P}(\mathbf{L}) \times \mathbf{D}$$

and whose transitions are exactly those that can be derived by the following rules

$$\frac{-}{\Pi(L,\alpha) \xrightarrow{\alpha} \Pi(L,\mathsf{skip})}\text{if } l.\alpha \in L \qquad \frac{-}{\Pi(L,\mathsf{refuse}\,(d)) \xrightarrow{\tau_{refuse}} \Pi(L,d)}\text{if } ll.d \cap L \neq \emptyset$$

$$\frac{-}{\Pi(L,d_1\,\mathsf{alt}\,d_2) \xrightarrow{\tau_{alt}} \Pi(L,d_i)}\text{if } ll.(d_1\,\mathsf{alt}\,d_2) \cap L \neq \emptyset \text{ for } i \in \{1,2\}$$

$$\frac{\Pi(ll.d_1 \cap L, d_1) \xrightarrow{\alpha_\tau} \Pi(ll.d_1 \cap L, d_1')}{\Pi(L,d_1\,\mathsf{seq}\,d_2) \xrightarrow{\alpha_\tau} \Pi(L,d_1'\,\mathsf{seq}\,d_2)}\text{if } ll.d_1 \cap L \neq \emptyset$$

$$\frac{\Pi(L \setminus ll.d_1, d_2) \xrightarrow{\alpha_\tau} \Pi(L \setminus ll.d_1, d_2'))}{\Pi(L,d_1\,\mathsf{seq}\,d_2) \xrightarrow{\alpha_\tau} \Pi(L,d_1\,\mathsf{seq}\,d_2')}\text{if } L \setminus ll.d_1 \neq \emptyset$$

$$\frac{\Pi(L,d_1) \xrightarrow{\alpha_\tau} \Pi(L,d_1')}{\Pi(L,d_2) \xrightarrow{\alpha_\tau} \Pi(L,d_2')}\text{if } d_1 \equiv d_2 \text{ and } d_1' \equiv d_2'$$

$$\frac{\Pi(ll.d_1 \cap L, d_1) \xrightarrow{\alpha_\tau} \Pi(ll.d_1 \cap L, d_1')}{\Pi(L,d_1\,\mathsf{par}\,d_2) \xrightarrow{\alpha_\tau} \Pi(L,d_1'\,\mathsf{par}\,d_2)} \qquad \frac{\Pi(ll.d_2 \cap L, d_2) \xrightarrow{\alpha_\tau} \Pi(ll.d_2 \cap L, d_2')}{\Pi(L,d_1\,\mathsf{par}\,d_2) \xrightarrow{\alpha_\tau} \Pi(L,d_1\,\mathsf{par}\,d_2')}$$

$$\frac{-}{\Pi(L,\mathsf{loop}\langle 0..*\rangle\,(d)) \xrightarrow{\tau_{loop}} \Pi(L,\mathsf{skip}\,\mathsf{alt}\,(d\,\mathsf{seq}\,\mathsf{loop}\langle 0..*\rangle\,(d)))}\text{if } ll.d \cap L \neq \emptyset$$

For more explanation of the rules of the projection system, the reader is referred to [12].

The projection system of Def. 4 is based on [12] where parameter variables are not taken into consideration. Recall that each parameter variable is bound to a new value upon the occurrence of the event it is contained in. This has the consequence that parameter variables occurring inside a loop are bound to new values for each iteration of the loop. Thus to modify the projection system of Def. 4 to take this into account, we only need to modify the rule for $\mathsf{loop}\langle 0..*\rangle$ (the last rule of Def. 4). To simulate the fact that parameter variables are bound to new values in each iteration of the loop, we let the projection system rename all parameter variables by incrementing their index for each iteration of the loop. Formally, we make use of the function $ipv(\_) \in \mathbf{PVar} \rightarrow \mathbf{PVar}$ that increments the index of a parameter variable by one, i.e.,

$$ipv((vn,i)) = (vn, i+1)$$

The function is lifted to diagrams such that $ipv(d)$ yields the diagram obtained from d by incrementing all its parameter variables by one. The revised projection system is now given by the following definition.

Definition 5 (Revised projection system) *The revised projection system that handles parameter variables is the LTS over*

$$\alpha_\tau \in \{\tau_{refuse}, \tau_{alt}, \tau_{loop}\} \cup \mathbf{E} \cup \mathbf{C} \cup \mathbf{A}$$

whose states are

$$\Pi'(\_,\_) \in \mathbb{P}(\mathbf{L}) \times \mathbf{D}$$

and whose transitions are exactly those that can be derived by the rules of Def. 4 except for rule for $\mathsf{loop}\langle 0..*\rangle$ *which is redefined as follows:*

$$\frac{-}{\Pi'(L,\mathsf{loop}\langle 0..*\rangle\,(d)) \xrightarrow{\tau_{loop}} \Pi'(L,\mathsf{skip}\,\mathsf{alt}\,(d\,\mathsf{seq}\,\mathsf{loop}\langle 0..*\rangle\,(ipv(d))))}\text{if } ll.d \cap L \neq \emptyset$$

A.2.2 Evaluation and data states

In order to define the operational semantics of sequence diagrams, we need to describe how the data states change throughout execution. In this section, we present some auxiliary functions that are needed for this purpose.

An expression $ex \in \mathbf{Exp}$ is closed if $var(ex) = \emptyset$. We let **CExp** denote the set of closed expressions, defined as:

$$\mathbf{CExp} \stackrel{\mathsf{def}}{=} \{ex \in \mathbf{Exp} \mid var(ex) = \emptyset\}$$

We assume the existence of a function $eval \in \mathbf{CExp} \rightarrow \mathbf{Val} \cup \{\bot\}$ that evaluates any closed expression to its value. If an expression ex is not well formed or otherwise cannot be evaluated (e.g., because of division by zero), then $eval(ex) = \bot$. The evaluation function is lifted to signals, messages, and events such that $eval(si)$, $eval(m)$, $eval(e)$ evaluate all expressions of signal si, message m, and event e, respectively. For example, we have that

$$eval(msg(1+2, 4-1)) = msg(eval(1+2), eval(4-1)) = msg(3,3)$$

If an expression ex in signal si is not well formed, i.e., $eval(ex) = \bot$, then $eval(si) = \bot$. If e is an event (k, m) and si the signal of m, then we also have that $eval(m) = \bot$ and $eval(e) = \bot$.

Let $\sigma \in \mathbf{Var} \rightarrow \mathbf{Exp}$ be a mapping from variables to expressions. We denote such a mapping $\sigma = \{x_1 \mapsto ex_1, x_2 \mapsto ex_2, \ldots, x_n \mapsto ex_n\}$ for distinct $x_1, x_2, \ldots, x_n \in \mathbf{Var}$ and for $ex_1, ex_2, \ldots, ex_n \in \mathbf{Exp}$. If $ex_1, ex_2, \ldots, ex_n \in \mathbf{Val}$ we call it a data state. We let Σ denote the set of all mappings and $\widehat{\Sigma}$ denote the set of all data states. We use the same convention for the set of all events $\mathbf{E}$, and denote by $\widehat{\mathbf{E}}$, the set of all events whose signals have only values as arguments.

The empty mapping is denoted by $\emptyset$. $\mathcal{D}om(\sigma)$ denotes the domain of σ, i.e.,

$$\mathcal{D}om(\{x_1 \mapsto ex_1, x_2 \mapsto ex_2, \ldots, x_n \mapsto ex_n\}) \stackrel{\mathsf{def}}{=} \{x_1, x_2, \ldots, x_n\}$$

We let $\sigma[x \mapsto ex]$ denote the mapping σ except that it maps x to ex, i.e.,

$$\{x_1 \mapsto ex_1, \ldots, x_n \mapsto ex_n\}[x \mapsto ex] \stackrel{\mathsf{def}}{=} \begin{cases} \{x_1 \mapsto ex_1, \ldots, x_n \mapsto ex_n, x \mapsto ex\} \\ \text{if } x \neq x_i \text{ for all } i \in \{1, \ldots, n\} \\ \{x_1 \mapsto ex_1, \ldots, x_i \mapsto ex, \ldots, x_n \mapsto ex_n\} \\ \text{if } x = x_i \text{ for some } i \in \{1, \ldots, n\} \end{cases}$$

We generalize $\sigma[x \mapsto ex]$ to $\sigma[\sigma']$ in the following way:

$$\sigma[\{x_1 \mapsto ex_1, \ldots, x_n \mapsto ex_n\}] \stackrel{\mathsf{def}}{=} \sigma[x_1 \mapsto ex_1] \cdots [x_n \mapsto ex_n]$$

The mapping is lifted to expressions such that $\sigma(ex)$ yields the expression obtained from ex by simultaneously substituting the variables of ex with the expressions that these variables map to in σ. For example, we have that $\{y \mapsto 1, z \mapsto 2\}(y + z) = 1 + 2$. We furthermore lift σ to signals, messages, and events such that $\sigma(si)$, $\sigma(m)$, and $\sigma(e)$ yields the signal, message, and event obtained from si, m, and e, respectively, by substituting the variables of their expressions according to σ.

A.2.3 Execution system and trace semantics for sequence diagrams

The execution system of the operational semantics tells us how to execute a sequence diagram in a step by step manner. Unlike the projection system, the execution system keeps track of the *communication medium* and *data states* in addition to the diagram state. Thus a state of the execution system is a triple consisting of a communication medium, diagram, and data state:

$$\mathbf{AXS} \stackrel{\text{def}}{=} \mathbf{B} \times \mathbf{D} \times \widehat{\Sigma}_T$$

Here $\widehat{\Sigma}_T$ denotes the set of *total* data states, i.e., the set of all data states σ satisfying

$$\mathcal{D}om(\sigma) = \mathbf{Var}$$

We assume a communication model where each message has its own channel from the transmitter to the receiver, something that allows for message overtaking. The communication medium keeps track of messages that are sent between lifelines of a diagram, i.e., the messages of transmission events are put into the communication medium, while the messages of receive events are removed from the communication medium.

It is only necessary to keep track of the communication between those lifelines that are present in a sequence diagram; messages received from the environment (i.e., from lifelines not present in a diagram) are always assumed to be enabled.

The states of the communication medium are of the form (M, L) where M is a set of messages and L is a set of lifelines under consideration, i.e., the lifelines that are not part of the environment. The set of all communication medium states $\mathbf{B}$ is defined by

$$\mathbf{B} \stackrel{\text{def}}{=} \mathbb{P}(\mathbf{M}) \times \mathbb{P}(\mathbf{L}) \tag{18}$$

We define two functions for manipulating the communication medium: $add, rm \in \mathbf{B} \times \mathbf{M} \to \mathbf{B}$. The function $add(\beta, m)$ adds the message m to the communication medium β, while $rm(\beta, m)$ removes the message m from the communication medium β. We also define the predicate $ready \in \mathbf{B} \times \mathbf{M} \to \mathbb{B}ool$ that for a communication medium β and a message m yields true if β is in a state where it can deliver m, and false otherwise. Formally, we have

$$\begin{array}{lcl} add((M,L),m) & \stackrel{\text{def}}{=} & (M \cup \{m\}, L) \\ rm((M,L),m) & \stackrel{\text{def}}{=} & (M \setminus \{m\}, L) \\ ready((M,L),m) & \stackrel{\text{def}}{=} & tr.m \notin L \vee m \in M \end{array} \tag{19}$$

We are now ready to define the execution system for sequence diagrams.

Definition 6 (Execution system) *The execution system is an LTS whose states are*

$$\mathbf{B} \times \mathbf{D} \times \widehat{\Sigma}_T$$

whose labels are

$$\{\tau_{refuse}, \tau_{alt}, \tau_{loop}, \tau_{assign}, \mathtt{t}, \mathtt{f}, \bot\} \cup \mathbf{E}$$

and whose transitions are exactly those that can be derived from the following rules

$$\frac{\Pi'(ll.d, d) \xrightarrow{\tau} \Pi'(ll.d, d')}{[\beta, d, \sigma] \xrightarrow{\tau} [\beta, d', \sigma]} for\ \tau \in \{\tau_{refuse}, \tau_{loop}, \tau_{alt}\}$$

$$\frac{\Pi'(ll.d, d) \xrightarrow{(!,m)} \Pi'(ll.d, d')}{[\beta, d, \sigma] \xrightarrow{(!,eval(\sigma(m)))} [add(\beta, eval(\sigma(m))), d', \sigma]}$$

$$\frac{\Pi'(ll.d, d) \xrightarrow{(?,m)} \Pi'(ll.d, d')}{[\beta, d, \sigma] \xrightarrow{(?,m')} [rm(\beta, m'), d', \sigma]} \textit{if } ready(\beta, m') \land eval(\sigma(m)) = m'$$

$$\frac{\Pi'(ll.d, d) \xrightarrow{\mathsf{assign}(x,ex,l)} \Pi'(ll.d, d')}{[\beta, d, \sigma] \xrightarrow{\tau_{assign}} [\beta, d', \sigma[x \mapsto eval(\sigma(ex))]]}$$

$$\frac{\Pi'(ll.d, d) \xrightarrow{\mathsf{constr}(bx,l)} \Pi'(ll.d, d')}{[\beta, d, \sigma] \xrightarrow{eval(\sigma(bx))} [\beta, d', \sigma]}$$

See [12] for more details on the rules of the execution system.

The trace semantics of a sequence diagram is a pair consisting of a positive trace set and a negative trace set. The traces of a diagram d are obtained by recording the events occurring on the transitions of the execution system when executing d until it is reduced to a skip (which means that the diagram cannot be further executed).

To distinguish negative from positive traces, we make use of the silent event τ_{refuse}. That is, if a transition labeled by τ_{refuse} is taken during execution, then this means that a negative trace is being recorded. Otherwise the trace is positive.

Definition 7 (Trace semantics) *The trace semantics of d, written $[\![d]\!]$, is then defined by*

$$\begin{aligned}
[\![d]\!] \overset{\text{def}}{=} (&\{s|_{\mathbf{E}} \in \widehat{\mathbf{E}}^* \mid \\
&\quad \exists \beta \in \mathbf{B} : \exists \sigma, \sigma' \in \widehat{\Sigma}_T : \\
&\quad\; [(\emptyset, ll.d), d, \sigma] \xrightarrow{s} [\beta, \mathsf{skip}, \sigma'] \land s|_{\{\tau_{refuse}, \mathtt{f}, \perp\}} = \langle\rangle\}, \\
&\{s|_{\mathbf{E}} \in \widehat{\mathbf{E}}^* \mid \\
&\quad \exists \beta \in \mathbf{B} : \exists \sigma, \sigma' \in \widehat{\Sigma}_T : \\
&\quad\; [(\emptyset, ll.d), d, \sigma] \xrightarrow{s} [\beta, \mathsf{skip}, \sigma'] \land s|_{\{\tau_{refuse}, \mathtt{f}, \perp\}} \in \{\tau_{reuse}\}^+\})
\end{aligned}$$

Note that the projection function | takes a set A and a sequence s and yields the sequence $s|_A$ obtained from s by removing all elements not in A. Note also that A^+ denotes the set of sequences of A with at least one element, i.e., $A^+ \overset{\text{def}}{=} A^* \setminus \{\langle\rangle\}$.

A.3 Policy adherence for sequence diagrams

In this section, we define what it means for a system to adhere to a policy expressed by a sequence diagram.

A system (interpreted as a set of traces of events) adheres to a sequence diagram policy if none of the traces of the system has a negative trace of a lifeline in the sequence diagram as a sub-trace. A trace $s = \langle e_1, \ldots, e_n \rangle$ is a sub-trace of t, written $s \lhd t$, iff

$$s_1 \frown \langle e_1 \rangle \frown \cdots \frown s_n \frown \langle e_n \rangle \frown s_{n+1} = t \tag{20}$$

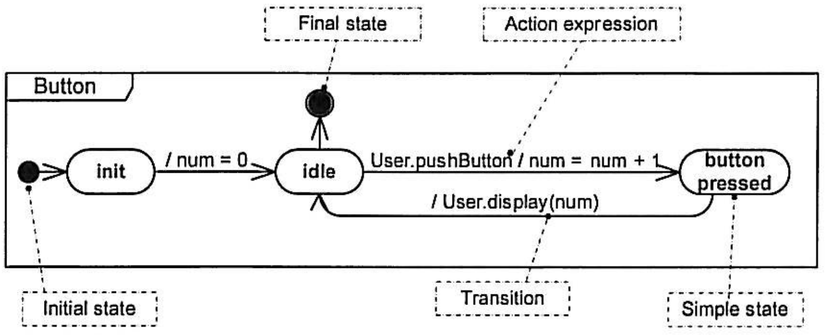


Figure 11: Example of a state machine

for some $s_1, \ldots, s_{n+1} \in \mathbf{E}^*$. See [18] for a more precise definition.

We first formally define adherence for diagrams consisting of a single lifeline.

Definition 8 (Policy adherence of single lifeline sequence diagrams) *Let d be a single lifeline diagram, i.e., $d \in \mathbf{D}^l$ for some lifeline l, and let Φ denote the traces of a system. Then the system adheres to the policy d, written $d \rightarrow_{da} \Phi$, iff*

$$(s \in H_{neg} \wedge t \in \Phi|_{\mathbf{E}^l}) \implies \neg(s \lhd t) \quad \textit{for } [\![d]\!] = (H_{pos}, H_{neg})$$

Note that the projection operator $\_|\_$ is lifted to sets of sequences such that $\Phi|_A$ yields the set obtained from Φ by projecting each sequence of Φ to A, i.e., $\Phi|_A = \{s|_A \mid s \in \Phi\}$.

Adherence for general sequence diagrams (i.e., sequence diagrams that may contain more than one lifeline) is captured by the following definition.

Definition 9 (Policy adherence of sequence diagrams) *Let d be a sequence diagram, i.e., $d \in \mathbf{D}$ and let Φ denote the traces of a system. Then the system adheres to the policy d, written $d \rightarrow_{dag} \Phi$, iff*

$$\pi_l(d) \rightarrow_{da} \Phi \qquad \textit{for all } l \in ll.d$$

B State machines

In this section, we define the syntax and the semantics of UML inspired state machines. We also define what it means for a system to adhere to a policy expressed as a state machine.

B.1 Syntax

As illustrated in Fig. 11, the constructs which are used for specifying state machines are *initial state*, *simple state*, *final state*, *transition*, and *action expression.*

A state describes a period of time during the life of a state machine. The three kinds of states, *initial state*, *simple states*, and *final states*, are graphically represented by a black circle, a box with rounded edges, and a black circle encapsulated by another circle, respectively.

A *transition* represents a move from one state to another. In the graphical diagrams, transitions are labeled by *action expressions* of the form

$$nm.si[bx]/ef$$

Here the expression *nm.si* where *nm* is a state machine name and *si* is a signal is called an *event trigger*. The expression $[bx]$ where bx is a boolean expression is called a *guard*, and ef is called an *effect*. Intuitively, the action should be understood as follows: when signal *si* is received from a state machine with name *nm* and the boolean expression bx evaluates to true, then the effect ef is executed. An effect is a sequence of assignments and/or an output expression of the form *nm.si* representing the transmission of signal *si* to the state machine with name *nm*.

We will henceforth consider action expressions that contain at most one event. In our formal representation of state machines, we will therefore use action expressions of the form (e, bx, sa) where e is an input or output event, bx is a boolean expression (the guard) and sa is a sequence of assignments of the form $((x_1, ex_1), \dots, (x_n, ex_n))$. Formally, the set of all action expressions w.r.t. to the set of events E is defined by

$$\mathbf{Act}_E \overset{\text{def}}{=} (E \cup \{\epsilon\}) \times \mathbf{BExp} \times (\mathbf{Var} \times \mathbf{Exp})^*$$

Note that the event is optional in an action. An action without an event is of the form (ϵ, bx, sa). We will henceforth use e_ϵ to denote an arbitrary event or an empty expression, i.e., e_ϵ denotes a member of $\mathbf{E} \cup \{\epsilon\}$

The alphabet of a state machine is a set of events containing signals whose arguments are distinct parameter variables. We require that all events in the alphabet are distinct when two events e and e' are considered equal if they have the same name and the same number of arguments.

To make this more precise, we let $\mathbf{E}_{pv}$ denote the set of all events whose signals contain distinct parameter variables only, i.e.,

$$\begin{array}{l} \forall (k, (nm_t, nm_r, st(ex_1, \dots, ex_n))) \in \mathbf{E} : \\ \quad \wedge\, ex_1 \in \mathbf{PVar} \wedge \dots \wedge ex_n \in \mathbf{PVar} \\ \quad \wedge\, \forall i, j \in \{1, \dots, n\} : \\ \qquad i \neq j \implies ex_i \neq ex_j \\ \quad \Leftrightarrow (k, (nm_t, nm_r, st(ex_1, \dots, ex_n))) \in \mathbf{E}_{pv} \end{array} \tag{21}$$

Note that the formula is written in a style suggested by Lamport [10]. Here, the arguments of a conjunction may be written as an aligned list where $\wedge$ is the first symbol before each argument. A similar convention is used for disjunctions. Also, indentation is sometimes used instead of parentheses.

We write $e = e'$ if events e and e' have the same kind, transmitter, and receiver and their signals have the same name and the same numbers of arguments, i.e.,

$$\begin{array}{l} (k, (nm_t, nm_r, st(ex_1, \dots, ex_j))) = (k', (nm'_t, nm'_r, st'(ex'_1, \dots, ex'_k))) \\ \Leftrightarrow k = k' \wedge nm_t = nm'_t \wedge nm_r = nm'_r \wedge st = st' \wedge j = k \end{array} \tag{22}$$

We are now ready to define the syntax of state machines precisely.

Definition 10 (State machines) *A state machine is a tuple* $(\mathcal{E}, \mathcal{Q}, \mathcal{R}, q_I, \mathcal{F})$ *consisting of*

- *an alphabet* $\mathcal{E} \subseteq \mathbf{E}_{pv}$ *where* $e, e' \in \mathcal{E} \implies \neg(e = e')$;
- *a set of states* $\mathcal{Q}$;
- *a transition relation* $\mathcal{R} \subseteq \mathcal{Q} \times \mathbf{Act}_{\mathcal{E}} \times \mathcal{Q}$;
- *an initial state* $q_I \in \mathcal{Q}$;
- *a set of final states* $\mathcal{F} \subseteq \mathcal{Q}$

The set of all state machines is denoted by **SM**.

We define the functions for obtaining the alphabet, states, transitions, initial state, and final states of a state machine:

$$\begin{array}{lcl}
alph((\mathcal{E}, \mathcal{Q}, \mathcal{R}, q_I, \mathcal{F})) & \overset{\text{def}}{=} & \mathcal{E} \\
states((\mathcal{E}, \mathcal{Q}, \mathcal{R}, q_I, \mathcal{F})) & \overset{\text{def}}{=} & \mathcal{Q} \\
trans((\mathcal{E}, \mathcal{Q}, \mathcal{R}, q_I, \mathcal{F})) & \overset{\text{def}}{=} & \mathcal{R} \\
init((\mathcal{E}, \mathcal{Q}, \mathcal{R}, q_I, \mathcal{F})) & \overset{\text{def}}{=} & q_I \\
final((\mathcal{E}, \mathcal{Q}, \mathcal{R}, q_I, \mathcal{F})) & \overset{\text{def}}{=} & \mathcal{F}
\end{array}$$

B.1.1 Syntax constraints

We impose one restriction on the set of syntactically correct action expressions $\mathbf{Act}_E$; the parameter variables of the guard and the assignment sequence must be a subset of the parameter variables of the event (if the event is present in the action):

$$(e, bx, sa) \in \mathbf{Act}_E \implies (pvar(bx) \cup pvar(sa)) \subseteq pvar(e) \qquad (23)$$

where *pvar* yields the set of all parameter variables in an expression, assignment sequence, or an event.

We define four syntax rules for state machines, and we say that a state machine SM is *well formed* if it satisfies these rules:

SM1 The initial state of SM has zero ingoing transitions.

SM2 The initial state of SM has exactly one outgoing transition, and the action expression of this transition does not contain an event or a guard.

SM3 Each transition of SM (except the initial transition) is labeled by an action expression that contains an event.

SM4 All variables (except parameter variables) in SM must be explicitly assigned to a value before they are used.

B.2 Semantics

In this section, we define the semantics of state machines. First we define the execution graph of a state machine, then we define the traces obtained by executing a state machine.

The execution graph of a state machine SM is an LTS whose states are pairs $[q, \sigma]$ where q is a state of SM and σ is a data state. The transitions of the execution graph are defined in terms of the transitions of SM. That is, if

SM has a transition from q to q' that is labeled by (e, bx, sa) and the signal of e has no arguments, then the execution graph has a transition from $[q, \sigma]$ to $[q, \sigma']$ that is labeled by e provided that the guard bx is evaluated to true under state σ. Here, the data state σ' is equal to σ except that the variables of sa are assigned to new values as specified by sa. To make this precise, we define the function $as2ds \in \widehat{\Sigma} \times (\mathbf{Var} \times \mathbf{Exp})^* \to \widehat{\Sigma}$ which takes a data state σ and an assignment sequence sa and yields a new updated data state. Formally,

$$\begin{array}{lcl} as2ds(\sigma, ()) & \overset{\text{def}}{=} & \sigma \\ as2ds(\sigma, (x, ex) \frown sa) & \overset{\text{def}}{=} & as2ds(\sigma[x \mapsto eval(\sigma(ex))], sa) \end{array} \tag{24}$$

If the signal of event e contains arguments, i.e., parameter variables, then these variables are bound the new arbitrary values. In this case, the guard bx and the event e are evaluated under some data state $\sigma[\sigma'']$ where σ'' is an arbitrary mapping whose domain equals the parameter variables of e.

Definition 11 (Execution graph of state machines) *The execution graph of state machine $SM = (\mathcal{E}, \mathcal{Q}, \mathcal{R}, q_I, \mathcal{F})$, written $EG(SM)$, is the LTS over $\{\epsilon\} \cup \mathbf{E}$ whose states are*

$$\mathcal{Q} \times \widehat{\Sigma}_T$$

and whose transitions are exactly those that can be derived from the following rule

$$\frac{q \xrightarrow{(e_\epsilon, bx, sa)} q' \in \mathcal{R}}{[q, \sigma] \xrightarrow{eval(\sigma[\sigma'](e_\epsilon))} [q', as2ds(\sigma[\sigma'], sa)]} \text{if } eval(\sigma[\sigma'](bx)) = \mathrm{t} \wedge \mathcal{D}om(\sigma') = pvar(e_\epsilon)$$

The trace semantics of a state machine is the set of sequences obtained by recording the events occurring in each path from the initial state to a final state of the state machine.

Definition 12 (Trace semantics of state machines) *The trace semantics of a state machine $SM = (\mathcal{E}, \mathcal{Q}, \mathcal{R}, q_I, \mathcal{F})$, written $[\![SM]\!]$, is defined by*

$$\begin{array}{lcl} [\![SM]\!] & \overset{\text{def}}{=} & \{s|_{\mathbf{E}} \in \widehat{\mathbf{E}}^* \mid \\ & & \quad \exists q' \in \mathcal{F} : \exists \sigma, \sigma' \in \widehat{\Sigma}_T : \\ & & \quad\quad [q_I, \sigma] \xrightarrow{s} [q', \sigma'] \in EG(SM)\} \end{array}$$

B.3 Policy adherence for state machines

In this section, we define what it means for a system to adhere to a policy expressed as a state machine. We also define what it means for a system to adhere to a set of state machines.

Intuitively, a system S adheres to a state machine policy SM if all execution traces of S (when restricted to the alphabet of SM) are described by SM. This is formally captured by the following definition.

Definition 13 (Policy adherence for a state machine) *Let SM be a state machine defining a policy and let Φ denote the traces of a system. Then the system adheres to SM, written $SM \to_{sa} \Phi$, iff*

$$\Phi|_E \subseteq [\![SM]\!]$$

where $E \overset{\text{def}}{=} \{e \in \widehat{\mathbf{E}} \mid e' \in alph(SM) \wedge e = e'\}$.

Adherence for a set of state machines is precisely captured by the following definition.

Definition 14 (Policy adherence for a set of state machines) *Let SMS be a set of state machines and let Φ denote the traces of a system. Then the system adheres to SMS, written $SMS \rightharpoonup_{sag} \Phi$, iff*

$$SM \rightharpoonup_{sa} \Phi \qquad \textit{for all } SM \in SMS$$

C Specifying transformations using sequence diagrams

In this section, we show how transformations can be expressed in terms of sequence diagrams.

C.1 Transformation specifications

A *transformation specification* is a set of *mapping rules.* A mapping rule is a pair (dp, dp') consisting of a left hand side *sequence diagram pattern* dp and a right hand side sequence diagram pattern dp'. A sequence diagram pattern is a sequence diagram whose atoms (events, constraints, and assignments) may contain *meta variables.*

We let **MVar** denote the set of all meta variables and we let mv range over this set. Events that may contain meta variables are called *event patterns.* The set **EP** of all event patterns is defined

$$\mathbf{EP} \stackrel{\text{def}}{=} \mathbf{K} \times ((\mathbf{L} \cup \mathbf{MVar}) \times (\mathbf{L} \cup \mathbf{MVar}) \times (\mathbf{SIP} \cup \mathbf{MVar}))$$

Here **SIP** denotes the set of all *signal patterns.* This set is defined by

$$\mathbf{SIP} \stackrel{\text{def}}{=} (\mathbf{NM} \cup \mathbf{MVar}) \times (\mathbf{ExpP})^*$$

where **ExpP** is an expression that may contain meta variables (in addition to normal variables and parameter variables).

A *constraint pattern* is an expression of the form

$$\mathsf{constr}(bxp, l)$$

where bxp is a boolean expression that may contain meta variables. We let **CP** denote all constraint patterns.

An *assignment pattern* is an expression of the form

$$\mathsf{assign}(x, exp, l)$$

where exp is an expression that may contain meta variables. We let **AP** denote the set of all assignment patterns.

Definition 15 (Sequence diagram pattern) *The set of sequence diagram patterns* **DP** *is defined by the following syntax*

$$\begin{array}{rcl} dp & ::= & mv \mid ep \mid cp \mid ap \mid \mathsf{refuse}\,(dp) \mid \mathsf{loop}\langle 0..^*\rangle\,(dp) \mid \\ & & dp_1 \;\mathsf{seq}\; dp_2 \mid dp_1 \;\mathsf{alt}\; dp_2 \mid dp_1 \;\mathsf{par}\; dp_2 \end{array}$$

A sequence diagram pattern is either a meta variable (mv), an event pattern (ep), a constraint pattern (cp), an assignment pattern (ap), or the composition of one or more diagram patterns.

C.2 Transformation

In this section, we define the function induced by a transformation specification. Intuitively, when a transformation specification ts is applied to a sequence diagram d, all fragments of d that *match* a left hand side pattern of a mapping rule in ts are replaced by the right hand side pattern of the mapping rule.

Matching is defined in terms of a substitution $sub \in \mathbf{MVar} \rightarrow (\mathbf{D} \cup \mathbf{Exp})$ that replaces meta variables by diagrams or expressions. Any substitution sub is lifted to diagram patterns such that $sub(dp)$ yields the diagram obtained from dp by simultaneously replacing all meta variables in dp by diagrams or expressions according to sub. The set of all substitution is denoted by $\mathcal{S}ub$.

A diagram pattern dp *matches* a diagram d if there is a substitution sub such that

$$sub(dp) = d$$

We say that the domain of a mapping rule (dp, dp'), written $\mathcal{D}om((dp, dp'))$, is the set of all diagrams that can be matched by its left hand side pattern, i.e.,

$$\mathcal{D}om((dp, dp')) \overset{\text{def}}{=} \{d \in \mathbf{D} \mid \exists sub \in \mathcal{S}ub : sub(dp) = d\}$$

To ensure that transformation specifications induce functional transformations, we require that the mapping rules of a transformation specification must have disjoint domains, i.e., each transformation specification ts must satisfy the following constraint

$$\begin{array}{l} \forall (dp_1, dp'_1) \in ts : \forall (dp_2, dp'_2) \in ts : \\ \quad (dp_1, dp'_1) \neq (dp_2, dp'_2) \implies \mathcal{D}om((dp_1, dp'_1)) \cap \mathcal{D}om((dp_2, dp'_2)) = \emptyset \end{array}$$

Definition 16 (Function induced by a transformation specification) *The function $T_{ts} \in \mathbf{D} \rightarrow \mathbf{D}$ induced by transformation specification ts is defined as follows*

```
if $sub(dp) = d$ for some $(dp, dp') \in ts$ and $sub \in \mathcal{S}ub$
  then $T_{ts}(d) = sub(dp')$
else if $d = d_1 \, \mathsf{op} \, d_2$ for some $d_1, d_2 \in \mathbf{D}$ and $\mathsf{op} \in \{\mathsf{seq}, \mathsf{alt}, \mathsf{par}\}$
  then $T_{ts}(d) = T_{ts}(d_1) \, \mathsf{op} \, T_{ts}(d_2)$
else if $d = \mathsf{op}(d_1)$ for some $d_1 \in \mathbf{D}$ and $\mathsf{op} \in \{\mathsf{loop}\langle 0..* \rangle, \mathsf{refuse}\}$
  then $T_{ts}(d) = \mathsf{op}(T_{ts}(d))$
else
  $T_{ts}(d) = d$
```

D From sequence diagrams to state machines

In this section, we define the transformation from sequence diagrams to state machines. In general, the transformation of a sequence diagram yields a set of state machines, i.e., one state machine for each lifeline in the sequence diagram.

The main requirement to the transformation is that is should be adherence preserving.

Definition 17 (Adherence preservation) *Let $T \in \mathbf{D} \rightarrow \mathbb{P}(\mathbf{SM})$ be a transformation from sequence diagrams to sets of state machines. Then T is adherence preserving if for every system with traces Φ and sequence diagram policy d, the system adheres to d if and only if it adheres to $T(d)$, i.e.,*

$$d \rightarrow_{dag} \Phi \Leftrightarrow T(d) \rightarrow_{sag} \Phi$$

We first, in Sect. D.1, define the transformation from a sequence diagram with only one lifeline to a state machine. Then, in Sect. D.2, we define the transformation from (general) sequence diagrams to state machine sets. We show that this transformation is adherence preserving for policies that are composed of sub-policies with disjoint sets of variables.

D.1 From single lifeline diagrams to state machines

The transformation from a single lifeline diagram d to a state machine has two phases. In phase 1, the sequence diagram d is transformed into a state machine SM whose trace semantics equals the negative trace set of $[\![d]\!]$. In phase 2, SM is inverted into the state machine SM' whose semantics is the set of all traces that do not have a trace of SM as a sub-trace.

Definition 18 (Single lifeline sequence diagram to basic state machine) *The transformation $d2p \in \mathbf{D} \rightarrow \mathbf{SM}$ from single lifeline diagrams to state machines is defined by*

$$d2p \stackrel{\text{def}}{=} ph2 \circ ph1$$

where $ph1$ and $ph2$ represent phase 1 and 2 (as defined below).

D.1.1 Phase 1

In phase 1, the sequence diagram d is transformed into a state machine SM that describes the negative traces of d. The state machine SM corresponds to the projection system induced by d. That is, if the projection system has a transition $\Pi(ll.d, d) \xrightarrow{e} \Pi(ll.d, d')$, then SM has a transition $q \xrightarrow{e} q'$ where q and q' correspond to $\Pi(ll.d, d)$ and $\Pi(ll.d, d')$, respectively. However, some transitions of the projection system are truncated. In particular,

- all silent events are removed, e.g., if $\Pi(ll.d, d) \xrightarrow{\tau_{alt}} \Pi(ll.d, d') \xrightarrow{e} \Pi(ll.d, d'')$, then SM has a transition $q \xrightarrow{e} q''$;
- constraints are concatenated with succeeding events and assignments concatenated with preceding events, e.g., if $\Pi(ll.d, d) \xrightarrow{\mathsf{constr}(bx,l)} \Pi(ll.d, d_1) \xrightarrow{e} \Pi(ll.d, d_2) \xrightarrow{\mathsf{assign}(x,ex,l)} \Pi(ll.d, d_3)$, then SM has a transition $q \xrightarrow{(e,bx,((x,ex)))} q_3$.

To define this precisely, we introduce the notion of experiment relation.

Definition 19 (Experiment relations) *The relations $\Rightarrow$, $\overset{\alpha}{\Rightarrow}$, and $\overset{s}{\Rightarrow}$ for any $\alpha \in (\mathbf{E} \cup \mathbf{C} \cup \mathbf{A})$ and $s \in (\mathbf{E} \cup \mathbf{C} \cup \mathbf{A})^*$ are defined as follows*

1. *$q \Rightarrow q'$ means that there is a sequence of zero or more transitions from q to q' that are labeled by silent events, i.e., $q \xrightarrow{\langle \tau_1, \ldots, \tau_n \rangle} q'$ for $\tau_1, \ldots, \tau_n \in \{\tau_{alt}, \tau_{refuse}, \tau_{loop}\}$;*

2. $q \overset{\alpha}{\Rightarrow} q'$ *means that* $q \Rightarrow q_1 \xrightarrow{\alpha} q_2 \Rightarrow q'$ *for some states* q_1 *and* q_2;

3. *if* $s = \langle \alpha_1, \alpha_2, \ldots, \alpha_n \rangle$, *then* $q \overset{s}{\Rightarrow} q'$ *means that* $q \overset{\alpha_1}{\Longrightarrow} q_1 \overset{\alpha_2}{\Longrightarrow} q_2 \cdots \overset{\alpha_n}{\Longrightarrow} q'$.

To obtain correct action expressions for transitions, we define the function $c2g \in \mathbf{C}^* \rightarrow \mathbf{BExp}$ for converting a sequence of constraints into a guard and the function $a2ef \in \mathbf{A}^* \rightarrow (\mathbf{Var} \times \mathbf{Exp})^*$ for converting a sequence of sequence diagram assignments into a sequence of state machine assignments. More precisely, these functions are defined as follows

$$\begin{array}{lcl} c2g((\mathsf{constr}(bx_1, l), \ldots, \mathsf{constr}(bx_n, l))) & \overset{\text{def}}{=} & conj(bx_1, \ldots, bx_n) \\ a2ef((\mathsf{assign}(x_1, ex_1, l), \ldots, \mathsf{assign}(x_n, ex_n, l))) & \overset{\text{def}}{=} & ((x_1, ex_1), \ldots, (x_n, ex_n)) \end{array} \tag{25}$$

where $conj \in \mathbf{BExp}^* \rightarrow \mathbf{BExp}$ yields the conjunction of a sequence of boolean expressions. For the empty sequence, $conj$ yields true, i.e., $conj(()) = \mathtt{t}$. We use the function $conj$ instead of expressing the conjunction directly because we have not defined the notation for boolean expressions in $\mathbf{BExp}$ since this is not important in this report.

To distinguish negative from positive traces, we make use of the τ_{refuse} silent event. That is, any execution that involves a τ_{refuse} represents a negative behavior. Otherwise the execution represents positive behavior.

Definition 20 (Positive and negative experiment relations) *The relations* $\overset{s}{\Rightarrow}_{pos}$ *and* $\overset{s}{\Rightarrow}_{neg}$ *for any* $s \in (\mathbf{E} \cup \mathbf{C} \cup \mathbf{A})^*$ *are defined as follows*

1. $q \overset{s}{\Rightarrow}_{neg} q'$ *means that* $q \overset{t}{\Rightarrow} q_1 \xrightarrow{\tau_{refuse}} q_2 \overset{u}{\Rightarrow} q'$ *for some states* q_1 *and* q_2 *and some traces* t *and* u *such that* $s = t \frown u$

2. $q \overset{s}{\Rightarrow}_{pos} q'$ *means that* $q \overset{s}{\Rightarrow} q'$ *and not* $q \overset{s}{\Rightarrow}_{neg} q'$

Since the goal of phase 1 is to construct a state machine SM that describes the negative traces of a sequence diagram, each final state of SM should accept a negative trace. To distinguish these final states from those that accept positive traces, we let each state of SM have one of two modes: *pos* and *neg*. If a state has mode *pos*, then this means that a positive trace is being recorded when this state is entered. If a state has mode *neg*, then a negative trace is being recorded when the state is entered.

Even though we shall restrict attention to well formed sequence diagrams, we cannot in general let the alphabet of the state machine be equal to the set of all events in the sequence diagram, because this set may not satisfy the requirement that all events in the alphabet must be distinct up to argument renaming (see Def. 10). To ensure that a correct alphabet is constructed, we make use of the function $\psi \in \mathbf{PVar} \rightarrow \mathbf{PVar}$ that renames parameter variables. We lift the function to signals with parameter variables as arguments as follows:

$$\psi(st(pv_1, pv_2, \ldots, pv_n)) \overset{\text{def}}{=} st(\psi(pv_1), \psi(pv_2), \ldots, \psi(pv_n))$$

To ensure that the renaming function does not change the meaning of a signal, we require that ψ does not rename two distinct parameter variables of a signal into the same parameter variable, i.e., we require

$$\begin{array}{l} \forall i, j \in \{1, \ldots, n\} : \\ \quad (\psi(st(pv_1, \ldots, pv_n)) = st(pv'_1, \ldots, pv'_n) \wedge pv_i \neq pv_j) \implies pv'_i \neq pv'_j \end{array} \tag{26}$$

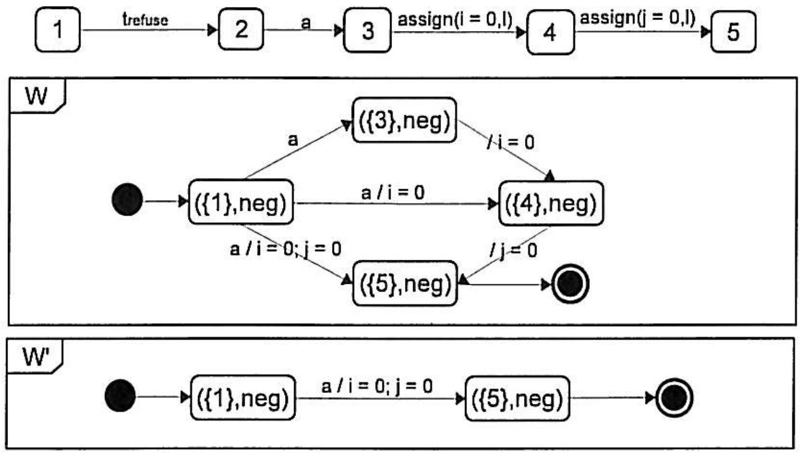


Figure 12: State machines W and W' are obtained by transformation without and with condition *Last*, respectively.

We lift ψ to expressions, events, and actions in the obvious way. Furthermore, we lift ψ to event sets and transition sets as follows:

$$\begin{aligned} \psi(E) &\overset{\text{def}}{=} \{\psi(e) \in \mathbf{E}_{pv} \mid e \in E\} \\ \psi(R) &\overset{\text{def}}{=} \{q \xrightarrow{\psi(act)} q' \mid q \xrightarrow{act} q' \in R\} \end{aligned}$$

We are now ready to define the transformation of phase 1.

Definition 21 (Phase 1) *The transformation $ph1 \in \mathbf{D} \to \mathbf{SM}$ which takes a single lifeline sequence diagram d and yields a state machine describing the negative traces of d is defined by*

$$ph1(d) = (\psi(eca.d \cap \mathbf{E}), \mathcal{Q}, \psi(\mathcal{R}), (\{d\}, pos), \{(Q, neg) \in \mathcal{Q} \mid \mathsf{skip} \in Q\})$$

where

$$\mathcal{Q} = \mathbb{P}(\mathbf{D}) \times \{pos, neg\}$$

$\psi \in \mathbf{PVar} \to \mathbf{PVar}$ *renames parameter variables such that*

$$\forall e, e' \in \psi(eca.d \cap \mathbf{E}) : \neg(e = e')$$

and transition relation $\mathcal{R}$ is defined by the following formula

$$
\begin{array}{lll}
\texttt{let} & Last(d') & \overset{\text{def}}{=} \lor \exists ec \in (\mathbf{E} \cup \mathbf{C}) : \exists d'' \in \mathbf{D} : \Pi(ll.d, d') \xrightarrow{ec} \Pi(ll.d, d'') \\
& & \phantom{\overset{\text{def}}{=}} \lor d' = \mathsf{skip} \\
& St(Q, t, mo) & \overset{\text{def}}{=} \{d'' \in \mathbf{D} \mid \\
& & \qquad d' \in Q \land \Pi(ll.d, d') \overset{t}{\Rightarrow}_{mo} \Pi(ll.d, d'') \land Last(d'')\}
\end{array}
$$

$$
\begin{array}{ll}
\texttt{in} & \forall tc \in \mathbf{C}^* : \forall e \in eca.d \cap \mathbf{E} : \forall ta \in \mathbf{A}^* : \\
& \quad \forall (Q, mo) \in \mathcal{Q} : \forall mo' \in \{pos, neg\} : \\
& \qquad \land\, St(\{d\}, ta, mo') \neq \emptyset \\
& \qquad\quad \Leftrightarrow (\{d\}, pos) \xrightarrow{(\epsilon, \epsilon, ta)} (St(\{d\}, ta, mo'), mo') \in \mathcal{R} \\
& \qquad \land\, St(Q, tc \frown \langle e \rangle \frown ta, pos) \neq \emptyset \\
& \qquad\quad \Leftrightarrow (Q, mo) \xrightarrow{(e, c2g(tc), a2ef(ta))} (St(Q, tc \frown \langle e \rangle \frown ta, pos), mo) \in \mathcal{R} \\
& \qquad \land\, St(Q, tc \frown \langle e \rangle \frown ta, neg) \neq \emptyset \\
& \qquad\quad \Leftrightarrow (Q, mo) \xrightarrow{(e, c2g(tc), a2ef(ta))} (St(Q, tc \frown \langle e \rangle \frown ta, neg), neg) \in \mathcal{R}
\end{array}
$$

The predicate *Last* (in Def. 21) ensures that the longest possible sequence of assignments is selected. For instance, the condition ensures that the following sequence diagram

$$\mathsf{refuse}\,(\mathsf{a}\ \mathsf{seq}\ \mathsf{assign}(\mathsf{i} = 0, \mathsf{l})\ \mathsf{seq}\ \mathsf{assign}(\mathsf{j} = 0, \mathsf{l}))$$

is transformed into the state machine W' in Fig. 12, and not the state machine W of Fig. 12. Note that the projection system consisting of those states that can be reached from the sequence diagram is illustrated at the top of Fig. 12.

Each state of the state machine constructed in phase 1 consists of a set of diagrams Q rather than a single diagram which is used by the projection system. This is to reduce nondeterminism in the constructed state machine. To see how this works, consider the LTS labeled A illustrated on the left hand side of Fig. 13. If we convert this into a state machine by removing silent events without merging states, we would obtain the state machine B shown in the middle of Fig. 13 (note that we have omitted to specify the modes of states in the figure). Clearly, this state machine is nondeterministic. However, if we merge states 3 and 4, we obtain the state machine C (on the right hand side of Fig. 13) which is deterministic.

Lemma 1 *Let d be a well formed single lifeline sequence diagram, then the state machine $ph1(d)$ describes the negative traces of d, i.e.,*

$$[\![\, ph1(d) \,]\!] = H_{neg} \qquad \textit{for } [\![\, d \,]\!] = (H_{pos}, H_{neg})$$

D.1.2 Phase 2

In phase 2, the state machine obtained from phase 1 is inverted into a state machine SM' whose semantics is the set of all traces that do not have a trace of SM as a sub-trace. This notion of inversion is captured by the following definition.

Definition 22 (Inversion) *State machine SM' is an inversion of state machine SM, written $inv(SM, SM')$, iff*

$$alph(SM) = alph(SM')$$

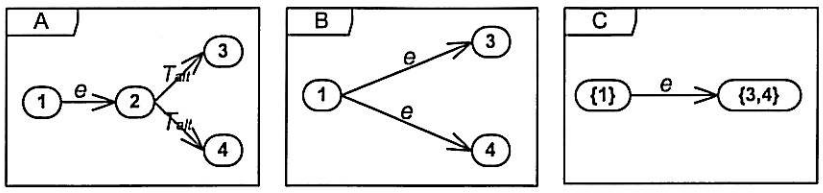


Figure 13: Machines A and B are nondeterministic while C is deterministic.

and for all $s \in \{e \in \widehat{\mathbf{E}} \mid \exists e' \in alph(SM) : e = e'\}^*$

$$(\forall t \in [\![SM]\!] : \neg(t \lhd s)) \quad \Leftrightarrow \quad s \in [\![SM']\!]$$

To explain how the transformation of phase 2 works, we first define the transformation for state machines whose transitions each contain exactly one event (whose signal has zero arguments) and no guards or assignments.

Definition 23 (Phase 2 - Preliminary definition 1) *The transformation* $ph2' \in \mathbf{SM} \rightarrow \mathbf{SM}$ *which yields the inversion of state machines whose transitions each contain exactly one event (whose signal has zero arguments) and no guards or assignments is defined by*

$$ph2'((\mathcal{E}, \mathcal{Q}, \mathcal{R}, q_I, \mathcal{F})) \overset{\text{def}}{=} (\mathcal{E}, \mathbb{P}(\mathcal{Q}), \mathcal{R}', \{q_I\}, \mathbb{P}(\mathcal{Q}))$$

where the transition relation $\mathcal{R}'$ *is defined by the formula*

```
let  $St(Q,e) \overset{\text{def}}{=} \{q' \in \mathcal{Q} \mid \exists q \in Q : q \xrightarrow{e} q' \in \mathcal{R}\}$

in   $\forall Q \in \mathbb{P}(\mathcal{Q}) :$
       $\forall e \in \mathcal{E} :$
         $St(Q,e) \cap \mathcal{F} = \emptyset \Leftrightarrow Q \xrightarrow{e} Q \cup St(Q,e) \in \mathcal{R}'$
```

The rule for generating transitions ensures that previously visited states are "recorded". To see why this is needed, consider the state machine P on the left hand side of Fig. 14. The set of traces described by it is

$$\{\langle \mathsf{a}, \mathsf{a} \rangle, \langle \mathsf{b}, \mathsf{b} \rangle\}$$

The inversion of P is the state machine P' shown on the right hand side of Fig. 14, i.e., $ph2'(\mathsf{P}) = \mathsf{P'}$. All states of P' are final, thus we have omitted to specify the final states in the figure. The trace semantics of P' is the set

$$\{\langle\rangle, \langle \mathsf{a} \rangle, \langle \mathsf{b} \rangle, \langle \mathsf{a}, \mathsf{b} \rangle, \langle \mathsf{b}, \mathsf{a} \rangle\}$$

Initially, both a and b are enabled. However, if a has occurred, then only b is enabled (if we assume that the alphabet of the state machine is $\{\mathsf{a}, \mathsf{b}\}$). Similarly if b has occurred, then only a is enabled. If both a and b have occurred, then no events are enabled. The final states of P are used to find those events that should *not* be enabled in P'. For instance, consider the transition $\{1\} \xrightarrow{\mathsf{a}} \{1,2\}$ in P'. Here the state 1 is "collected" because we need to make sure that b is not enabled after b has occurred in state {1,2}. Since 1 is collected, the occurrence

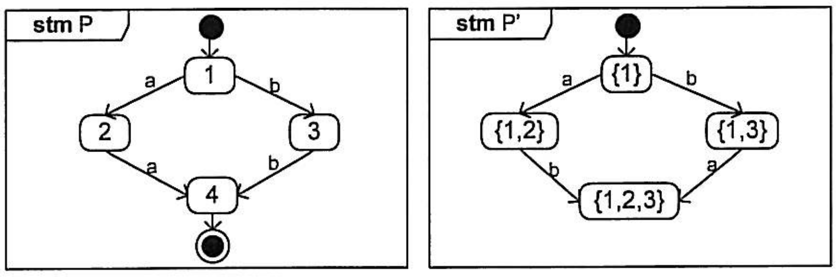


Figure 14: State machine P and its inversion P'

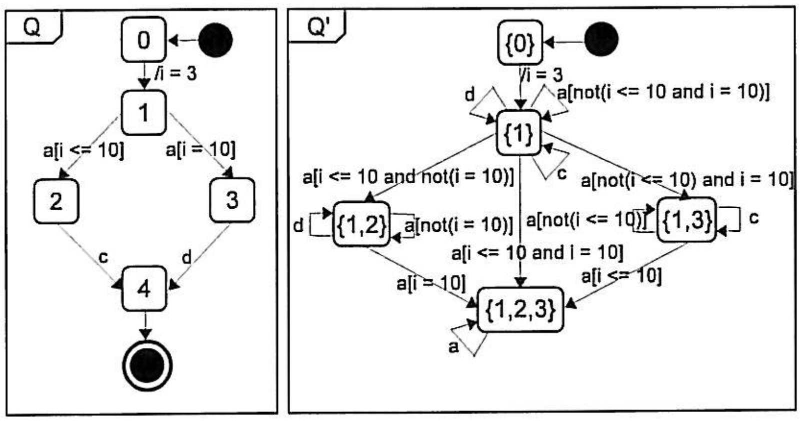


Figure 15: State machine Q and its inversion Q'

of b in state {1,2} leads to state {1,2,3} and b is not enabled in this state because the occurrence of b in state 3 leads to a final state in P.

The following lemma shows that the transformation of phase 2 is correct for simple state machines, i.e., state machines with no guards or assignments.

Lemma 2 *Let SM be a state machine whose transitions each contain exactly one event (whose signal has zero arguments) and no guards or assignments. Then $ph2'(SM)$ is an inversion of SM if $\langle\rangle \notin [\![SM]\!]$, i.e.,*

$$inv(SM, ph2'(SM))$$

The transformation of phase 2 is more complicated for state machines whose transitions contain guards. To see this, consider the state machine Q depicted on the left hand side of Fig.15. Inverting this state machine according to the transformation of Def. 23 would not work because the transformation does not take the guards into consideration. A correct inversion of Q is given by the state machine Q' depicted on the right hand side of Fig.15. Here we see that the transitions

$$1 \xrightarrow{(\mathsf{a}, bx_1, \epsilon)} 2 \quad \text{and} \quad 1 \xrightarrow{(\mathsf{a}, bx_2, \epsilon)} 3$$

of state machine Q (where bx_1 denotes i <= 10 and bx_2 denotes i = 10) have been converted into the transitions

$$\begin{array}{ll} \{1\} \xrightarrow{(\mathsf{a},\mathsf{not}(bx_1 \text{ or } bx_2),\epsilon)} \{1\} & \{1\} \xrightarrow{(\mathsf{a},\mathsf{not}(bx_1) \text{ and } bx_2,\epsilon)} \{1,3\} \\ \{1\} \xrightarrow{(\mathsf{a},bx_1 \text{ and } \mathsf{not}(bx_2),\epsilon)} \{1,2\} & \{1\} \xrightarrow{(\mathsf{a},bx_1 \text{ and } bx_2,\epsilon)} \{1,2,3\} \end{array}$$

In general, if a state machine SM has transitions

$$q \xrightarrow{(e,bx_1,sa_1)} q_1, q \xrightarrow{(e,bx_2,sa_2)} q_2, \ldots, q \xrightarrow{(e,bx_n,sa_n)} q_n$$

where $q_1, \ldots, q_n$ are not final states, and its inversion SM' has a state $\{q\}$, then for each set of indexes $Ix \subseteq \{1, \ldots, n\}$, the inversion SM' has a transition

$$\{q\} \xrightarrow{(e,bx \text{ and } bx',sa)} \{q\} \cup Q$$

where bx is the conjunction of those guards bx_i that have an index in Ix (i.e., $i \in Ix$), bx' is the negation of the disjunction of the guards that do *not* have an index in Ix, sa is the concatenated sequence of assignment sequences sa_i that have an index in Ix, and Q is the set of states q_i that have an index in Ix.

Note that for the special case where $Ix = \emptyset$, then bx should be equal to true. In addition, for the special case where $Ix = \{1, \ldots, n\}$, then bx' should be equal to true.

To make this more precise, we make use of the function $\_ \circledcirc \_ \in \mathbb{P}(\mathbb{N}) \times A \to A$ (where $\mathbb{N}$ denotes the set of all natural numbers, and A denotes the set of all sequences), that for a set of indexes Ix and a sequence s, yields the sequence obtained from s by removing all elements whose index is not in Ix, e.g.,

$$\{1,3,6\} \circledcirc (a,b,c,d,e) = (a,c) \qquad \text{and} \qquad \{2,4,5\} \circledcirc (a,b,c,d,e) = (b,d,e)$$

In addition, we let *list* be a function that turns a set into a list, *set* be a function that turns a list into a set (according to some total ordering on the elements in the set), and *flatten* be the function that flattens a nested sequence, e.g.,

$$\begin{array}{lcl} list(\{a,b,c\}) & = & (a,b,c) \\ list(\{b,a,c\}) & = & (a,b,c) \\ set((a,b,a,c)) & = & \{a,b,c\} \\ flatten((a,(b,c),(),f))) & = & (a,b,c,f) \end{array}$$

We also need functions on boolean expressions. As before, we let the function *conj* yield the conjunction of a sequence of boolean expressions. We also define the function $disj \in \mathbf{BExp}^* \to \mathbf{BExp}$ which yields the disjunction of a sequence of boolean expressions. For the empty sequence, *disj* yields false, i.e., $disj(()) =$ f. Finally, we let $neg \in \mathbf{BExp} \to \mathbf{BExp}$ be the function that yields the negation of a boolean expression.

We now revise the definition of the transformation of phase 2 in light of the above discussion.

Definition 24 (Phase 2 - preliminary definition 2) *The transformation $ph2'' \in \mathbf{SM} \rightarrow \mathbf{SM}$ which yields the inversion of well formed state machines is defined by*

$$ph2''((\mathcal{E}, \mathcal{Q}, \mathcal{R}, q_I, \mathcal{F})) \stackrel{\text{def}}{=} (\mathcal{E}, \mathbb{P}(\mathcal{Q}), \mathcal{R}', \{q_I\}, \mathbb{P}(\mathcal{Q}))$$

where the transition relation $\mathcal{R}'$ is defined by the following two rules:

$$q_I \xrightarrow{(\epsilon,\epsilon,sa)} q' \in \mathcal{R} \Leftrightarrow \{q_I\} \xrightarrow{(\epsilon,\epsilon,sa)} \{q'\} \in \mathcal{R}'$$

and

$$\begin{array}{lll}
\texttt{let} & Vi(Q,e) & \stackrel{\text{def}}{=} \{q \xrightarrow{(e',bx',as')} q' \in \mathcal{R} \mid q \in Q \wedge e = e'\} \\
 & Vi(Q,e,Ix) & \stackrel{\text{def}}{=} set(Ix \mathbin{\text{Ⓣ}} list(Vi(Q,e)) \\
 & St(Q,e,Ix) & \stackrel{\text{def}}{=} \{q \in \mathcal{Q} \mid \exists q' \xrightarrow{(e',bx',as')} q'' \in Vi(Q,e,Ix) : q = q''\} \\
 & Ga(Q,e,Ix) & \stackrel{\text{def}}{=} list(\{bx \in \mathbf{BExp} \mid \\
 & & \qquad \exists q' \xrightarrow{(e',bx',as')} q'' \in Vi(Q,e,Ix) : bx = bx'\}) \\
 & Ga'(Q,e,Ix) & \stackrel{\text{def}}{=} conj((conj(Ga(Q,e,Ix), \\
 & & \qquad neg(disj(Ga(Q,e,\mathbb{N} \setminus Ix)))))) \\
 & As(Q,e,Ix) & \stackrel{\text{def}}{=} \{as \in (\mathbf{Var} \times \mathbf{Exp})^* \mid \\
 & & \qquad \exists q' \xrightarrow{(e',bx',as')} q'' \in Vi(Q,e,Ix) : as = as'\} \\
 & As'(Q,e,Ix) & \stackrel{\text{def}}{=} flatten(list(As(Q,e,Ix))) \\
\texttt{in} & \multicolumn{2}{l}{\forall Q \in \mathbb{P}(\mathcal{Q}) :} \\
 & \multicolumn{2}{l}{\quad \forall e \in \mathcal{E} :} \\
 & \multicolumn{2}{l}{\qquad \forall Ix \in \mathbb{P}(\mathbb{N}) :} \\
 & \multicolumn{2}{l}{\qquad\quad St(Q,e,Ix) \cap \mathcal{F} = \emptyset \Leftrightarrow} \\
 & \multicolumn{2}{l}{\qquad\quad\quad Q \xrightarrow{(e,Ga'(Q,e,Ix),As'(Q,e,Ix))} (Q \cup St(Q,e,Ix)) \in \mathcal{R}'}
\end{array}$$

The transformation $ph2''$ does not always yield the correct inversion of a state machine. For instance, consider the state machine W depicted on the left hand side of Fig. 16. It describes two traces: one trace consisting of 9 occurrences of a, and one trace consisting of 9 occurrences of b. Applying the phase 2 transformation $ph2''$ to W yields the state machine W' depicted on the right hand side of Fig. 16. Note that we have not depicted final states (since all states are final) or transitions whose guards always evaluate to false and that redundancy in the boolean expressions of the guards have been removed, e.g., i $\leq$ 10 and true is written i $\leq$ 10.

The state machine W' rejects traces consisting of 10 or more occurrences of a or b. For instance, the trace t consisting of 5 occurrences of a and 5 occurrences of b is rejected by the state machine W'. However, no trace of W is a sub-trace of t. Hence, W' is not a correct inversion of W. The reason for this is that the two possible executions of W, resulting from the branch in state 2, *share* the variable i, i.e., the variable is used in a condition/guard of one execution and assigned to a value in another execution. In the example, this causes W' not to be a correct inversion of W.

In general, to ensure that $ph2''$ yields the correction inversion SM' of a state machine SM, we must require that all guards encountered when executing SM' must evaluate to the same values as the "corresponding" guards encountered when executing SM. We say that a transformation is *side effect free* for SM if this requirement is satisfied. This is precisely captured by the following definition.

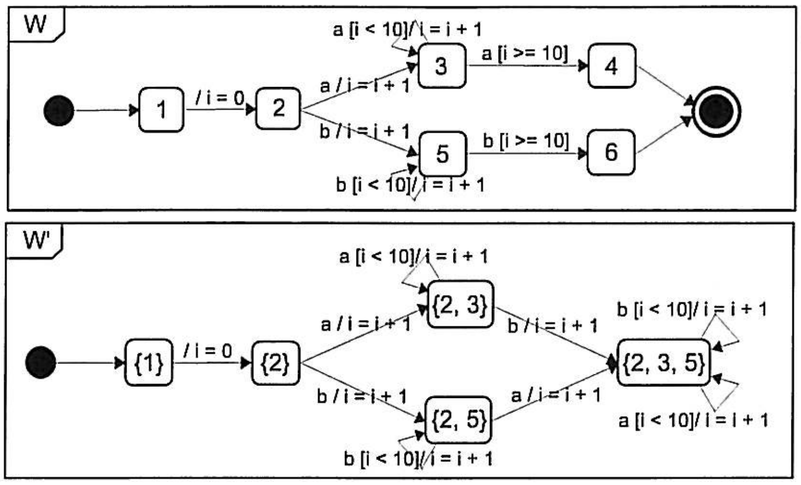


Figure 16: State machine W and its inversion

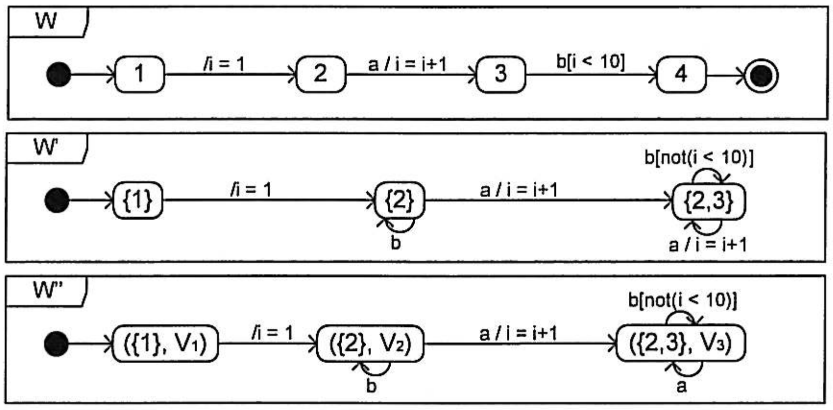


Figure 17: State machine W, its incorrect inversion $ph2''(\mathsf{W}) = \mathsf{W'}$, and its correct inversion W''

Definition 25 (Side effect free) *Let $SM = (\mathcal{E}, \mathcal{Q}, \mathcal{R}, q_I, \mathcal{F})$, $tr \in$ SM $\rightarrow$ SM, and $tr(SM) = SM' = (\mathcal{E}', \mathcal{Q}', \mathcal{R}', q_I', \mathcal{F}')$. Then transformation tr is side* *effect free for SM iff*

$$\begin{array}{l}
\forall s,t \in (\widehat{\mathbf{E}} \cup \{\epsilon\})^* : \forall (e,bx,as) \in \mathbf{Act}_{\widehat{\mathbf{E}}} : \\
\quad \forall q_1, q_2 \in \mathcal{Q} : \forall q_1' \in \mathcal{Q}' : \\
\quad\quad \forall \sigma_I, \sigma_1, \sigma_I', \sigma_1' \in \widehat{\Sigma}_T : \\
\quad\quad\quad \wedge\, [q_I, \sigma_I] \xrightarrow{s} [q_1, \sigma_1] \in EG(SM) \\
\quad\quad\quad \wedge\, [q_I', \sigma_I'] \xrightarrow{t} [q_1', \sigma_1'] \in EG(SM') \\
\quad\quad\quad \wedge\, q_1 \xrightarrow{(e,bx,as)} q_2 \in \mathcal{R} \\
\quad\quad\quad \wedge\, comp([\![SM]\!], s, t) \\
\quad\quad\quad \Longrightarrow \sigma_1 \cap (var(bx) \times \mathbf{Exp}) = \sigma_1' \cap (var(bx) \times \mathbf{Exp})
\end{array}$$

where the predicate $comp(\_,\_,\_) \in \mathbb{P}(\mathbf{E}^*) \times \mathbf{E}^* \times \mathbf{E}^* \rightarrow \mathbb{B}$ *is defined* $comp(T,s,t) \overset{\text{def}}{=} s \lhd t \wedge pr(T,s) \neq \emptyset \wedge \neg(\exists s' \in pr(T,s) : s \sqsubset s' \wedge s' \lhd t)$ *where the function* $pr \in \mathbb{P}(\mathbf{E}^*) \times \mathbf{E}^* \rightarrow \mathbb{P}(\mathbf{E}^*)$ *is defined by* $pr(T,s) \overset{\text{def}}{=} \{s \frown s' \in T \mid \neg(\exists t \in T : t \sqsubset s \frown s')\}$.

Lemma 3 *Let SM be a well formed state machine such that* $\langle\rangle \notin [\![SM]\!]$ *and* $ph2''$ *be side effect free for SM, then* $ph2''(SM)$ *is an inversion of SM, i.e.,*

$$inv(SM, ph2''(SM))$$

It is possible to define an alternative version of the transformation of phase 2 for which the side effect free condition is less restrictive. In particular, we observe that *ph2''* may generate unnecessary guards and assignment sequences for transitions corresponding to inconclusive behavior. As an example, consider the state machine W in Fig.17. It describes the trace ⟨a, b⟩. The result of applying transformation *ph2''* to W is depicted by state machine W' in Fig.17 (i.e., *ph2''*(W) = W'). Note that we have not illustrated final states (since all states are final) or transitions whose guards always evaluate to false, and that boolean expressions have been simplified. The state machine W' is not a correct inversion of W since b is enabled after a has occurred 10 times. The problem is that the reflexive transition

$$\{2,3\} \xrightarrow{(a/i = i + 1)} \{2,3\}$$

in W' describing inconclusive behavior, contains the (unnecessary) assignment i = i + 1) since W has the transition

$$2 \xrightarrow{(a/i = i + 1)} 3$$

which has been previously visited to reach the state {2, 3}.

A solution to the problem of the current example, is the let each state of the inverted state machine record all transitions that are previously visited in order to reach that state. The previously visited transitions can then be disregarded when generating reflexive transitions corresponding to inconclusive behavior.

State machine W'' in Fig.17 shows how this would work in the current example. Here V_1, V_2, V_3 are sets of previously visited transitions of W defined by

$$V_1 = \emptyset \qquad V_2 = \{1 \xrightarrow{/i = 1} 2\} \qquad V_3 = \{1 \xrightarrow{/i = 1} 2, 2 \xrightarrow{a/i = i + 1} 3\}$$

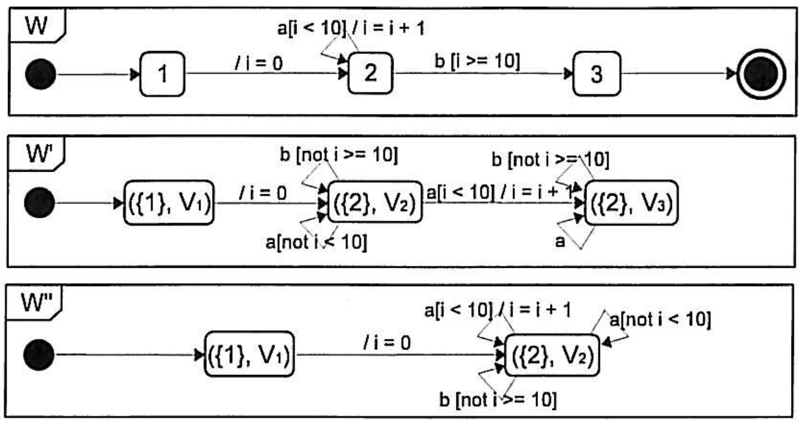


Figure 18: State machine W and its (incorrect) inversion W' and (correct) inversion W''

Now, when generating transitions for a in state $(\{2,3\}, \mathsf{V}_3)$, we disregard the set V_3 of previously visited transitions. Thus we get

$$(\{2,3\}, \mathsf{V}_3) \xrightarrow{\mathsf{a}} (\{2,3\}, \mathsf{V}_3)$$

and by definition of inversion (Def. 22) we have that W'' is a correct inversion of W.

The solution proposed above may not work for state machines that contain loops. For instance, consider the state machine W of Fig.18. It describes the trace containing 9 occurrences of a followed by b. In other words, the policy states that b is not allowed to occur after a has occurred 9 times. If we use the transformation $ph2''$ to invert W and record previously visited transitions as described above, we get state machine W' of Fig.18. Here we have that

$$\mathsf{V}_1 = \emptyset \qquad \mathsf{V}_2 = \{1 \xrightarrow{/\mathsf{i{=}0}} 2\} \qquad \mathsf{V}_3 = \{1 \xrightarrow{/\mathsf{i{=}0}} 2, 2 \xrightarrow{\mathsf{a[i < 10]/i{=}i + 1}} 2\}$$

The state machine W' is not a correct inversion of W since it allows the occurrence of b after a has occurred more than 9 times. In this case, adding the transition $2 \xrightarrow{(\mathsf{a},\ \mathsf{i} < 10, \mathsf{i} = \mathsf{i} + 1)} 2$ into the set of previously visited transitions, and thereby disregarding its transitions, is incorrect, because the transition is in a loop and may therefore be visited several times.

A solution to the problem is to remove the transitions of a loop from the set of visited transitions each time the loop is iterated. To achieve this, we can remove the outgoing transitions of each state that is entered from the set of previously visited transitions. In the current example, we would then obtain the state machine W'' of Fig.18 which is a correct inversion of W.

We are now ready to give the final definition of the transformation of phase 2.

Definition 26 (Phase 2) *The transformation $ph2 \in \mathrm{SM} \to \mathrm{SM}$ which yields the inversion of well formed state machines is defined by*

$$ph2((\mathcal{E}, \mathcal{Q}, \mathcal{R}, q_I, \mathcal{F})) \stackrel{\text{def}}{=} (\mathcal{E}, (\mathbb{P}(\mathcal{Q}) \times \mathbb{P}(\mathcal{R})), \mathcal{R}', (\{q_I\}, \emptyset), (\mathbb{P}(\mathcal{Q}) \times \mathbb{P}(\mathcal{R})))$$

where the transition relation $\mathcal{R}'$ is defined by the following two rules:

$$q_I \xrightarrow{(\epsilon,\epsilon,sa)} q' \in \mathcal{R} \Leftrightarrow (\{q_I\},\emptyset) \xrightarrow{(\epsilon,\epsilon,sa)} (\{q'\},\emptyset) \in \mathcal{R}'$$

and

$$\begin{array}{lll}
\texttt{let} \; Vi(Q) & \overset{\text{def}}{=} & \{q \xrightarrow{(e',bx',as')} q' \in \mathcal{R} \mid q \in Q\} \\
Vi(Q,e,V) & \overset{\text{def}}{=} & \{q \xrightarrow{(e',bx',as')} q' \in \mathcal{R} \mid q \in Q \wedge e = e'\} \setminus V \\
Vi(Q,e,Ix,V) & \overset{\text{def}}{=} & set(Ix \circledS list(Vi(Q,e,V)) \\
St(Q,e,Ix,V) & \overset{\text{def}}{=} & \{q \in \mathcal{Q} \mid \\
 & & \exists q' \xrightarrow{(e',bx',as')} q'' \in Vi(Q,e,Ix,V) : q = q''\} \\
Ga(Q,e,Ix,V) & \overset{\text{def}}{=} & list(\{bx \in \mathbf{BExp} \mid \\
 & & \exists q' \xrightarrow{(e',bx',as')} q'' \in Vi(Q,e,Ix,V) : bx = bx'\}) \\
Ga'(Q,e,Ix,V) & \overset{\text{def}}{=} & conj((conj(Ga(Q,e,Ix,V), \\
 & & neg(disj(Ga(Q,e,\mathbb{N} \setminus Ix,V)))))) \\
As(Q,e,Ix,V) & \overset{\text{def}}{=} & \{as \in (\mathbf{Var} \times \mathbf{Exp})^* \mid \\
 & & \exists q' \xrightarrow{(e',bx',as')} q'' \in Vi(Q,e,Ix,V) : as = as'\} \\
As'(Q,e,Ix,V) & \overset{\text{def}}{=} & flatten(list(As(Q,e,Ix,V)))
\end{array}$$

$$\begin{array}{l}
\texttt{in} \; \forall (Q,V) \in \mathbb{P}(\mathcal{Q}) \times \mathbb{P}(\mathcal{R}) : \forall e \in \mathcal{E} : \\
\quad \forall Ix \in \mathbb{P}(\mathbb{N}) : \\
\quad\quad St(Q,e,Ix,V) \cap \mathcal{F} = \emptyset \Leftrightarrow \\
\quad\quad\quad (Q,V) \xrightarrow{(e,Ga'(Q,e,Ix,V),As'(Q,e,Ix,V))} \\
\quad\quad\quad (Q \cup St(Q,e,Ix,V),(V \cup Vi(Q,e,Ix,V)) \setminus Vi(St(Q,e,Ix,V))) \in \mathcal{R}'
\end{array}$$

Corollary 1 *Let SM be a well formed state machine such that $\langle\rangle \notin [\![SM]\!]$ and $ph2$ be side effect free for SM, then $ph2(SM)$ is an inversion of SM, i.e.,*

$$inv(SM, ph2(SM))$$

The transformation $ph2$ will correctly invert the state machine examples of Fig. 14, Fig. 15, Fig. 17, and Fig. 18, as well as all the examples of the first part of this report (the part before the appendices).

However, $ph2$ does not work for the state machine of Fig. 16, where the variable i is *shared* in the sense that it is used in a condition/guard of one execution and assigned to a value in another execution. We make this precise in the following definition.

Definition 27 (Shared variables) *Let $SM = (\mathcal{E},\mathcal{Q},\mathcal{R},q_I,\mathcal{F})$, then SM does not have any shared variables iff*

$$\begin{array}{l}
\forall s,t,t' \in \mathbf{Act}^* : \forall q,q' \in \mathcal{Q} : \\
\quad \forall (e_1,bx_1,as_1),(e_1',bx_1',as_1'),(e_2,bx_2,as_2),(e_2',bx_2',as_2') \in \mathbf{Act} : \\
\quad\quad \wedge\, q_I \xrightarrow{s \frown \langle (e_1,bx_1,as_1)\rangle \frown t \frown \langle (e_2,bx_2,as_2)\rangle} q \in \mathcal{R} \\
\quad\quad \wedge\, q_I \xrightarrow{s \frown \langle (e_1',bx_1',as_1')\rangle \frown t' \frown \langle (e_2',bx_2',as_2')\rangle} q' \in \mathcal{R} \\
\quad\quad \wedge\, (e_1,bx_1,as_1) \neq (e_1',bx_1',as_1') \\
\quad\quad \wedge\, \neg(\forall \sigma \in \widehat{\Sigma}_T : eval(\sigma(bx_1)) = eval(\sigma(bx_1')) \\
\quad\quad\quad \Longrightarrow ((var(bx_2) \cap avar(as_2')) \setminus \mathbf{PVar}) = \emptyset
\end{array}$$

where $avar \in (\mathbf{Var} \times \mathbf{Exp})^* \rightarrow \mathbb{P}(\mathbf{Var})$ yields all the variables that are assigned to a value in an assignment sequence, i.e., $avar(((x_1, ex_n), \ldots, (x_n, ex_n))) = \{x_1\} \cup \cdots \cup \{x_n\}$.

We conjecture that if a state machine SM does not have shared variables in the sense of (Def. 27), then $ph2$ is side effect free for SM (Def. 25). By Corollary 1, this means $ph2$ will yield the correct inversion of any state machine that does not have shared variables.

In practice, the condition that a state machine policy must not have shared variables, means that when we compose several policies, then these policies cannot have the same variable names. For instance, consider again the state machine W of Fig. 16. This state machine may be seen as the composition of the two policies: (1) more than 9 occurrences of a is not allowed, and (2) more than 9 occurrences of (2) b is not allowed. However, since both policies use the variable i to count the number of occurrences of a or b, the condition of no shared variables is violated.

Note that it is feasible to automatically check whether a state machine has no shared variables because the condition is formulated in terms of the transitions of a state machine as opposed to the transitions of the execution graph.

Together, the transformation of phase 1 and phase 2 is adherence preserving when the condition of phase 2 is satisfied.

Theorem 1 *Let d be a well formed single lifeline sequence diagram such that $ph2$ is side effect free for $ph1(d)$, then the transformation $d2p(d)$ is adherence preserving, i.e.,*

$$d \rightarrow_{da} \Phi \Leftrightarrow d2p(d) \rightarrow_{sa} \Phi \qquad \textit{for all systems } \Phi$$

D.2 From general sequence diagrams to state machines

In this section, we define the transformation that takes a (general) sequence diagram and yields a set of state machines.

Definition 28 (From sequence diagrams to sets of state machines) *The transformation $d2pc \in \mathbf{D} \rightarrow \mathbb{P}(\mathbf{SM})$ which takes a sequence diagram and yields a set of state machine, is defined by*

$$d2pc(d) \stackrel{\text{def}}{=} \bigcup_{l \in ll.d} \{d2p(\pi_l(d))\}$$

The transformation from sequence diagrams to state machine sets is adherence preserving when the condition of phase 2 is satisfied.

Theorem 2 *Let d be a well formed sequence diagram such that $ph2$ is side effect free for $ph1(\pi_l(d))$ for all lifelines l in d, then the transformation $d2pc(d)$ is adherence preserving, i.e.,*

$$d \rightarrow_{dag} \Phi \Leftrightarrow d2pc(d) \rightarrow_{sag} \Phi \qquad \textit{for all systems } \Phi$$

E Proofs

Lemma 1 Let d be a well formed sequence diagram, then the state machine $ph1(d)$ describes the negative traces of d, i.e.,

$$[\![ph1(d)]\!] = H_{neg} \qquad \text{for } [\![d]\!] = (H_{pos}, H_{neg})$$

Proof of Lemma 1 By Lemma 1.1 and Lemma 1.2, and definition of $ph1$ (Def. 21).

Lemma 1.1 Let d be a well formed sequence diagram, $SM = (eca.d \cap \mathbf{E}, \mathcal{Q}, \mathcal{R}, q_I, \mathcal{F})$ and $SM' = (\psi(\mathcal{E}), \mathcal{Q}, \psi(\mathcal{R}), q_I, \mathcal{F})$ for some variable renaming function $\psi \in \mathbf{PVar} \to \mathbf{PVar}$ satisfying constraint (26). Then the semantics of SM is equal to the semantics of SM', i.e.,

$$[\![SM]\!] = [\![SM']\!]$$

Proof of Lemma 1.1 By definition of the execution graph of state machines (Def. 11), all parameter variables are treated as local variables for each transition, thus a renaming of the parameter variables has no effect on the execution unless two distinct parameter variables of the same signal are renamed into the same parameter variable. However, this cannot occur since ψ is assumed to satisfy constraint (26). Note that it is always possible to find a renaming function that satisfies constraint (26) for the events of a given well formed sequence diagram because well formed diagrams must satisfy conditions **SD3** and **SD7**

Lemma 1.2 Let d be a well formed single lifeline sequence diagram such that $[\![d]\!] = (H_{pos}, H_{neg})$, let $ph1(d) = (\psi(\mathcal{E}), \mathcal{Q}, \psi(\mathcal{R}), q_I, \mathcal{F})$ for some parameter variable renaming function $\psi \in \mathbf{PVar} \to \mathbf{PVar}$, and let $SM = (\mathcal{E}, \mathcal{Q}, \mathcal{R}, q_I, \mathcal{F})$, then the semantics of SM is equal to H_{neg}, i.e.,

$$[\![SM]\!] = H_{neg}$$

Proof of Lemma 1.2

ASSUME: 1. $ph1(d) = (\psi(\mathcal{E}), \mathcal{Q}, \psi(\mathcal{R}), q_I, \mathcal{F})$ and $SM = (\mathcal{E}, \mathcal{Q}, \mathcal{R}, q_I, \mathcal{F})$ for some $d \in \mathbf{D}^l$, $l \in \mathbf{L}$, and $\psi \in \mathbf{PVar} \to \mathbf{PVar}$

2. $[\![d]\!] = (H_{pos}, H_{neg})$

3. d is well formed (i.e., d satisfies conditions **SD1** - **SD10**)

PROVE: $[\![SM]\!] = H_{neg}$

$\langle 1 \rangle 1.$ ASSUME: 1.1. $s \in H_{neg}$

PROVE: $s \in [\![SM]\!]$

$\langle 2 \rangle 1.$ $\exists s' \in (\widehat{\mathbf{E}} \cup \{\epsilon\})^*, \sigma_I, \sigma \in \widehat{\Sigma}_T, q \in \mathcal{F}:$

$\wedge\, [q_I, \sigma_I] \xrightarrow{s'} [q, \sigma] \in EG(SM)$

$\wedge\, s'|_{\mathbf{E}} = s$

$\langle 3 \rangle 1.$ Choose $\langle e_1, \ldots, e_n \rangle \in \widehat{\mathbf{E}}^*$ such that $\langle e_1, \ldots, e_n \rangle = s$

PROOF: By assumption 2, assumption 1.1, and definition of $[\![\_]\!]$ (Def. 7).

$\langle 3 \rangle 2.$ Choose

$t \in (\widehat{\mathbf{E}} \cup \{\mathtt{t}, \mathtt{f}, \perp, \tau_{alt}, \tau_{refuse}, \tau_{loop}\})^*$,

$\sigma_I, \sigma_n \in \widehat{\Sigma}_T$, and

$\beta_n \in \mathbf{B}$

such that

$[(\emptyset, ll.d), d, \sigma_I] \xrightarrow{t} [\beta_n, \mathsf{skip}, \sigma_n]$

$t|_{\{\tau_{refuse}, \mathtt{f}, \perp\}} \in \{\tau_{refuse}\}^+$

$t|_{\mathbf{E}} = s$

PROOF: By assumption 2, assumption 1.1, and definition of $[\![\_]\!]$ (Def. 7).

$\langle 3 \rangle 3.$ Choose
$tt_1, \ldots, tt_n \in \{\mathtt{t}\}^*$,
$tta_0, tta_1, \ldots, tta_n \in \{\tau_{assign}\}^*$, and
$E = (\mathbf{E} \cup \{\mathtt{t}, \tau_{assign}\}) \setminus \{\tau_{alt}, \tau_{loop}, \tau_{refuse}\}$ such that
$t|_E = tta_0 \frown tt_1 \frown \langle e_1 \rangle \frown tta_1 \frown \cdots \frown tt_n \frown \langle e_n \rangle \frown tta_n$
PROOF: By $\langle 3 \rangle 1$ and $\langle 3 \rangle 2$ we know that $t|_{\mathbf{E}} = \langle e_1, \ldots, e_n \rangle$. Furthermore, since d satisfies syntax constraints **SD4** and **SD8** (by assumption 3), we know that t must be of the form asserted by $\langle 3 \rangle 3$ by definition of the execution system for sequence diagrams (Def. 6).

$\langle 3 \rangle 4.$ $[(\emptyset, ll.d), d, \sigma_I] \xRightarrow{tta_0 \frown tt_1 \frown \langle e_1 \rangle \frown tta_1 \frown \cdots \frown tt_n \frown \langle e_n \rangle \frown tta_n} [\beta_n, \mathsf{skip}, \sigma_n]$
PROOF: By $\langle 3 \rangle 2$, $\langle 3 \rangle 3$ and definition of $\Rightarrow$ (Def. 19).

$\langle 3 \rangle 5.$ Choose
$\sigma_0, \ldots, \sigma_{n-1} \in \widehat{\Sigma}$,
$\beta_0, \beta_1, \ldots, \beta_{n-1} \in \mathbf{B}$,
$d_0, \ldots, d_{n-1} \in \mathbf{D}$
such that
$[(\emptyset, ll.d), d, \sigma_I] \xRightarrow{tta_0} [\beta_0, d_0, \sigma_0] \xRightarrow{tt_1 \frown \langle e_1 \rangle \frown tta_1} [\beta_1, d_1, \sigma_1] \cdots$
$\xRightarrow{tt_n \frown \langle e_n \rangle \frown tta_n} [\beta_n, \mathsf{skip}, \sigma_n]$
PROOF: By $\langle 3 \rangle 4$.

$\langle 3 \rangle 6.$ Choose
$tc_1, \ldots, tc_n \in \mathbf{C}^*$,
$ta_0, \ldots, ta_n \in \mathbf{A}^*$, and
$e'_1, \ldots, e'_n \in \mathbf{E}$
such that
$\Pi'(ll.d, d) \xRightarrow{ta_0} \Pi'(ll.d, d_0) \xRightarrow{tc_1 \frown \langle e'_1 \rangle \frown ta_1} \Pi'(ll.d, d_1) \cdots$
$\xRightarrow{tc_n \frown \langle e'_n \rangle \frown ta_n} \Pi'(ll.d, \mathsf{skip})$,
$eval(\sigma_i(e'_i)) = e_i$ for all $i \in \{1, \ldots, n\}$,
$eval(\sigma_i(c2g(tc_i))) = \mathtt{t}$ for all $i \in \{1, \ldots, n\}$,
$\sigma_0 = as2ds(\sigma_I, a2ef(ta_0))$, and
$\sigma_{i+1} = as2ds(\sigma_i, a2ef(ta_i))$ for all $i \in \{0, \ldots, n-1\}$
PROOF: By Def.6, a transition $\Pi'(ll.d, d') \xRightarrow{eca} \Pi'(ll.d, d'')$ of the revised projection system corresponds to the transition $[\beta', d', \sigma'] \xrightarrow{eval(\sigma(eca))} [\beta'', d'', \sigma']$ of the execution system if *eca* is an event or constraint, or the transition $[\beta', d', \sigma'] \xrightarrow{\tau_{assign}} [\beta'', d'', \sigma'']$ where $\sigma'' = \sigma'[x \mapsto eval(ex)] = asds(\sigma', \langle eca \rangle)$ if *eca* is an assignment of the form (x, ex). By $\langle 3 \rangle 5$, definition of the execution system for sequence diagrams (Def.6), definition of *c2g* (Eq. (25)), *a2ef* (Eq. (25)), and *as2ds* (Eq. (24)), we therefore have that $\langle 3 \rangle 6$ holds.

$\langle 3 \rangle 7.$ LET: Let pr be a function that sets the index of all parameter variables in a term to zero, e.g., $pr((vn, 3)) = (vn, 0)$

$\langle 3 \rangle 8.$ $\Pi(ll.d, d) \xRightarrow{ta_0} \Pi(ll.d, d_0) \xRightarrow{pr(tc_1) \frown \langle pr(e'_1) \rangle \frown pr(ta_1)} \Pi(ll.d, pr(d_1))$
$\cdots \xRightarrow{pr(tc_n) \frown \langle pr(e'_n) \rangle \frown pr(ta_n)} \Pi(ll.d, pr(d_n))$
PROOF: By $\langle 3 \rangle 6$, $\langle 3 \rangle 7$, and definition of the projection systems for sequence diagrams (Def.4 and Def. 5).

$\langle 3 \rangle 9.$ Choose
$(Q_0, mo_0), \ldots, (Q_n, mo_n) \in \mathcal{Q}$

such that
$(\{d\}, pos) \xrightarrow{(\epsilon, \epsilon, a2ef(ta_0))} (Q_0, mo_0) \xrightarrow{(pr(e_1'), pr(c2g(tc_1)), pr(a2ef(ta_1)))}$
$(Q_1, mo_1) \cdots \xrightarrow{(pr(e_n'), pr(c2g(tc_n)), pr(a2ef(ta_n)))} (Q_n, mo_n) \in \mathcal{R}$,
$pr(d_i) \in Q_i$ for all $i \in \{0, \ldots, n-1\}$, and
$\mathsf{skip} \in Q_n$

PROOF: By definition of $ph1$ (Def 21), a transition $\Pi(ll.d, d') \xRightarrow{tc \frown \langle e \rangle \frown ta}_{mo'} \Pi(ll.d, d'')$ of the projection system corresponds to a transition $(Q', mo) \xrightarrow{e, c2g(tc), a2ef(ta)} (Q'', mo')$ of the state machine (which is produced by $ph1$) where $d' \in Q'$ and $d'' \in Q''$. Therefore, by assumption 1 (since $ph1(d) = SM$), ⟨3⟩8, and definition of $ph1$ (Def 21), we have that ⟨3⟩9 holds.

⟨3⟩10. $(Q_n, mo_n) \in \mathcal{F}$

PROOF: By definition of $ph1$ (Def 21), (Q_n, mo_n) is in the set of final states $\mathcal{F}$ if $\mathsf{skip} \in Q_n$ and $mo_n = neg$. By ⟨3⟩9, we know that $\mathsf{skip} \in Q_n$. To see that $mo_n = neg$, note that by ⟨3⟩2, t must contain the silent event τ_{refuse}. This means that $[(\emptyset, ll.d), d, \sigma_I] \overset{t}{\Rightarrow}_{neg} [\beta_n, \mathsf{skip}, \sigma_n]$ holds by definition of $\Rightarrow_{mo}$ (Def. 20). By ⟨3⟩5 - ⟨3⟩9 and definition of $ph1$ (Def. 21), this implies that $mo_n = neg$.

⟨3⟩11. $\exists \sigma_0', \ldots, \sigma_n' \in \widehat{\Sigma}_T$:
$[(\{d\}, pos), \sigma_I] \xrightarrow{\epsilon} [(Q_0, mo_0), \sigma_0'] \xrightarrow{e_1} [(Q_1, mo_1), \sigma_1'] \cdots$
$\xrightarrow{e_n} [(Q_n, mo_n), \sigma_n'] \in EG(SM)$

⟨4⟩1. Choose
$\sigma_0', \ldots, \sigma_n' \in \widehat{\Sigma}_T$ and
$\sigma_1'', \ldots, \sigma_n'' \in \widehat{\Sigma}$
such that
$(A) \mathcal{D}om(\sigma_i'') = pvar(pr(e_i'))$ for all $i \in \{1, \ldots, n\}$,
$(B) eval(\sigma_i'[\sigma_i''](pr(e_i'))) = e_i$ for all $i \in \{1, \ldots, n\}$,
$(C) \sigma_0' = as2ds(\sigma_I, a2ef(ta_0))$, and
$(D) \sigma_{i+1}' = as2ds(\sigma_i'[\sigma_i''], pr(a2ef(ta_i)))$ for all $i \in \{0, \ldots, n-1\}$

PROOF: Data staes that satisfy (A), (C), and (D) of ⟨4⟩1 can always be chosen trivially. Furthermore, data states that satisfy (B) can be chosen because the message of each event $pr(e_i')$ contain parameter variables only since d is assumed to satisfy syntax constraint **SD3** by assumption 3. Therefore it is always possible to chose a data state σ_i'' such that $eval(\sigma_i'[\sigma_i''](pr(e_i'))) = eval(\sigma_i''(pr(e_i'))) = \sigma_i''(pr(e_i')) = e_i$.

⟨4⟩2. $eval(\sigma_i'[\sigma_i''](pr(c2g(tc_i)))) = \mathtt{t}$ for all $i \in \{1, \ldots, n\}$

PROOF: By ⟨3⟩6, ⟨3⟩7, ⟨3⟩8, and ⟨4⟩1.

⟨4⟩3. Q.E.D.

PROOF: By ⟨4⟩1, ⟨4⟩2, and definition of the execution graph for state machines (Def. 11).

⟨3⟩12. Q.E.D.

PROOF: By ⟨3⟩10 and ⟨3⟩11.

⟨2⟩2. Q.E.D.

PROOF: By ⟨2⟩1 and definition of $[\![\ ]\!]$ (Def.12).

⟨1⟩2. ASSUME: 1.1. $s \in [\![SM]\!]$
PROVE: $s \in H_{neg}$

⟨2⟩1. $\exists \sigma, \sigma' \in \widehat{\Sigma}_T, \beta \in \mathbf{B}, s' \in (\widehat{\mathbf{E}} \cup \{\mathtt{t}, \mathtt{f}, \bot, \tau_{alt}, \tau_{refuse}, \tau_{loop}\})^*$:

$\wedge\ [(\emptyset, ll.d), d, \sigma] \xrightarrow{s'} [\beta, \mathsf{skip}, \sigma']$
$\wedge\ s'|_{\mathbf{E}} = s$
$\wedge\ s'|_{\{\mathbf{f},\perp,\tau_{refuse}\}} = \{\tau_{refuse}\}^{+}$

⟨3⟩1. Choose
$t \in (\widehat{\mathbf{E}} \cup \{\epsilon\})^{*}$,
$\sigma_I, \sigma_n \in \widehat{\Sigma}_T$,
$(Q_n, mo_n) \in \mathcal{F}$
such that
$[q_I, \sigma_I] \xrightarrow{t} [(Q_n, neg), \sigma_n] \in EG(SM)$ and
$s = t|_{\mathbf{E}}$

PROOF: By assumption 1, assumption 1.1, definition of $[\![\ ]\!]$ (Def. 12), and definition of $ph1$ (Def. 21).

⟨3⟩2. $t = \langle\epsilon\rangle \frown s$

PROOF: By definition of $ph1$ (Def. 21), SM has exactly one transition from its inital state, and that transition is not labeled by an action containing an event. All other transitions of SM are labeled by actions containing events. By definition of $[\![\ ]\!]$ (Def. 12), this means that $t = \langle\epsilon\rangle \frown s$.

⟨3⟩3. Choose $\langle e_1, \ldots, e_n\rangle \in \mathbf{E}^{*}$ such that $s = \langle e_1, \ldots, e_n\rangle$

PROOF: By ⟨3⟩1 and definition of the execution graph of state machines (Def. 11).

⟨3⟩4. Choose
$(Q_0, mo_0), (Q_1, mo_1), \ldots, (Q_{n-1}, mo_{n-1}) \in \mathcal{Q}$ and
$\sigma_0, \sigma_1, \ldots, \sigma_{n-1} \in \widehat{\Sigma}_T$
such that
$[q_I, \sigma_I] \xrightarrow{\epsilon} [(Q_0, mo_0), \sigma_0] \xrightarrow{e_1} [(Q_1, mo_1), \sigma_1] \cdots$
$\xrightarrow{e_n} [(Q_n, neg), \sigma_n] \in EG(SM)$

PROOF: By ⟨3⟩1, ⟨3⟩2, ⟨3⟩3, and definition of the execution graph of state machines (Def. 11).

⟨3⟩5. Choose
$e'_1, \ldots, e'_n \in \mathbf{E}$,
$bx_1, \ldots, bx_n \in \mathbf{BExp}$,
$as_0, \ldots, as_n \in (\mathbf{Var} \times \mathbf{Exp})^{*}$, and
$\sigma'_1, \ldots, \sigma'_n \in \widehat{\Sigma}_T$
such that
$q_I \xrightarrow{(\epsilon,\epsilon,as_0)} (Q_0, mo_0) \xrightarrow{(e'_1,bx_1,as_1)} (Q_1, mo_1) \cdots$
$\xrightarrow{(e'_n,bx_n,as_n)} (Q_n, neg) \in \mathcal{R}$,
$\mathcal{D}om(\sigma'_i) = pvar(e'_i)$ for all $i \in \{1, \ldots, n\}$,
$\sigma_0 = as2ds(as_0)$,
$\sigma_{i+1} = as2ds(\sigma_i[\sigma'_i], as_i)$ for all $i \in \{0, \ldots, n-1\}$,
$eval(\sigma_i[\sigma'_i](e'_i)) = e_i$ for all $i \in \{0, \ldots, n\}$, and
$eval(\sigma_i[\sigma'_i](bx_i)) = \mathtt{t}$ for all $i \in \{0, \ldots, n\}$

PROOF: By assumption 1, $\mathcal{R}$ denotes the transitions of SM, therefore, ⟨3⟩5 holds by ⟨3⟩4 and definition of the execution graph for state machines (Def. 11).

⟨3⟩6. Choose
$d_0 \in Q_0, \ldots, d_{n-1} \in Q_{n-1}$,
$tc_1, \ldots, tc_n \in \mathbf{C}^{*}$, and
$ta_0, ta_1, \ldots, ta_n \in \mathbf{A}^{*}$

such that

$(A)\,\Pi(ll.d, d) \xRightarrow{ta_0} \Pi(ll.d, d_0) \xRightarrow{tc_1 \frown \langle e'_1 \rangle \frown ta_1} \Pi(ll.d, d_1) \cdots$
$\xRightarrow{tc_n \frown \langle e'_n \rangle \frown ta_n} \Pi(ll.d, \mathsf{skip})$,
$(B)\,bx_i = c2g(tc_i)$ for all $i \in \{1, \ldots, n\}$,
$(C)\,as_i = a2ef(ta_i)$ for all $i \in \{0, \ldots, n\}$, and (D)
$\Pi(ll.d, d_j) \xRightarrow{tc_j \frown \langle e'_j \rangle \frown as_j}_{neg} \Pi(ll.d, d_{j+1})$
for some $j \in \{0, \ldots, n-1\}$
or
$\Pi(ll.d, d) \xRightarrow{ta_0}_{neg} \Pi(ll.d, d_0)$

PROOF: By definition of $ph1$ (Def 21), a transition $\Pi(ll.d, d') \xRightarrow{tc \frown \langle e \rangle \frown ta}_{mo'} \Pi(ll.d, d'')$ of the projection system corresponds to a transition $(Q', mo) \xrightarrow{e, c2g(tc), a2ef(ta)} (Q'', mo')$ of the state machine (which is produced by $ph1$) where $d' \in Q'$ and $d'' \in Q''$. Therefore, by assumption 1 (since $ph1(d) = SM$), $\langle 3 \rangle 5$, and definition of $ph1$ (Def 21), we have that (A), (B), and (C) hold. (D) holds by definition of $ph1$ (Def 21) because the last state (Q_n, neg) $\langle 3 \rangle 5$ has mode neg.

$\langle 3 \rangle 7$. LET: Let pr be a function that sets the index of all parameter variables in a term to zero, e.g., $pr((vn, 3)) = (vn, 0)$

$\langle 3 \rangle 8$. Choose
$d'_1, \ldots, d'_n \in \mathbf{D}$,
$tc'_1, \ldots, tc'_n \in \mathbf{C}^*$,
$e''_1, \ldots, e''_n \in \mathbf{E}$, and
$ta'_1, \ldots, ta'_n \in \mathbf{A}^*$
such that
$pr(d'_i)) = d_i$ for all $i \in \{1, \ldots, n\}$,
$pr(tc'_i) = tc_i$ for all $i \in \{1, \ldots, n\}$,
$pr(e''_i) = e'_i$ for all $i \in \{1, \ldots, n\}$,
$pr(ta'_i) = ta_i$ for all $i \in \{1, \ldots, n\}$, and
$\Pi'(ll.d, d) \xRightarrow{ta_0} \Pi'(ll.d, d_0) \xRightarrow{tc'_1 \frown \langle e''_1 \rangle \frown ta'_1} \Pi'(ll.d, d'_1) \cdots$
$\xRightarrow{tc'_n \frown \langle e''_n \rangle \frown ta'_n} \Pi'(ll.d, d'_n)$

PROOF: By $\langle 3 \rangle 6$, definition of pr ($\langle 3 \rangle 7$) and definition of the revised projection system (Def. 5).

$\langle 3 \rangle 9$. $\exists tt_1, \ldots, tt_n \in \{\mathtt{t}\}^*$:
$\exists tta_0, \ldots, tta_n \in \{\tau_{assign}\}^*$:
$\exists \beta_0, \ldots, \beta_n \in \mathbf{B}$:
$\exists \sigma'_I, \sigma'_0, \sigma'_1, \ldots, \sigma'_n \in \widehat{\Sigma}$:
$[(\emptyset, ll.d), d, \sigma'_I] \xRightarrow{tta_0} [\beta_0, d_0, \sigma'_0] \xRightarrow{tt_1 \frown \langle e_1 \rangle \frown tta_1} [\beta_1, d'_1, \sigma'_1] \cdots$
$\xRightarrow{tt_n \frown \langle e_n \rangle \frown tta_n} [\beta_n, d'_n, \sigma'_n]$

$\langle 4 \rangle 1$. Choose
$\sigma'_I, \sigma'_0, \ldots, \sigma'_n \in \widehat{\Sigma}$
such that
$(A)\,\sigma'_I \setminus (\mathbf{PVar} \times \mathbf{Exp}) = \sigma_I \setminus (\mathbf{PVar} \times \mathbf{Exp})$,
$(B)\,\sigma'_0 = as2ds(\sigma'_I, a2ef(ta_0))$,
$(C)\,\sigma'_{i+1} = as2ds(\sigma'_i, a2ef(ta'_i))$ for all $i \in \{0, \ldots, n-1\}$, and
$(D)\,eval(\sigma'_i(e''_i)) = e_i$ for all $i \in \{1, \ldots, n\}$

PROOF: Conditions (A), (B), and (C) of $\langle 4\rangle 1$ hold trivially. Condition (D) holds because the signal of each event e''_i contain parameter variables only and no two events contain the same parameter variables (since d satisfies syntax constraints **SD3** and **SD7** by assumption 3). Therefore, we can always choose an initial state σ'_I that assigns variables to values such that (D) holds since parameter variables are never explicitly assigned to values by any assignment sequence ta'_i (since only a normal variable and not a parameter variable can be explicitly assigned to a value by the constructs in a sequence diagram).

$\langle 4\rangle 2$. $eval(\sigma_i(c2g(tc'_i))) = \mathtt{t}$ for all $i \in \{1, \ldots, n\}$

PROOF: By $\langle 3\rangle 4$ and $\langle 4\rangle 1$.

$\langle 4\rangle 3$. Q.E.D.

PROOF: By $\langle 4\rangle 1$ and $\langle 4\rangle 2$.

$\langle 3\rangle 10$. Q.E.D.

PROOF: By $\langle 3\rangle 9$, we know that $[(\emptyset, ll.d), d, \sigma'_I] \xrightarrow{t} [\beta_n, d'_n, \sigma'_n]$ for some trace t such that $t|_{\mathbf{E}} = s$. Furthermore, $d'_n = \mathsf{skip}$ by $\langle 3\rangle 6$ and $\langle 3\rangle 8$, and t must contain the silent event τ_{refuse} by $\langle 3\rangle 6$. Therefore $\langle 2\rangle 1$ must hold.

$\langle 2\rangle 2$. Q.E.D.

PROOF: By $\langle 2\rangle 1$ and definition of $[\![\ ]\!]$ (Def. 7).

$\langle 1\rangle 3$. Q.E.D.

PROOF: By $\langle 1\rangle 1$ and $\langle 1\rangle 2$.

Lemma 2 Let SM be a state machine whose transitions each contain exactly one event (whose signal has zero arguments) and no guards or assignments. Then $ph2'(SM)$ is an inversion of SM if $\langle\rangle \notin [\![SM]\!]$, i.e.,

$$\langle\rangle \notin [\![SM]\!] \implies inv(SM, ph2'(SM))$$

Proof of Lemma 2

ASSUME: 1. $SM = (\mathcal{E}, \mathcal{Q}, \mathcal{R}, q_I, \mathcal{F})$ and $\langle\rangle \notin [\![SM]\!]$

2. $q \xrightarrow{act} q' \in \mathcal{R} \implies act = (e, \epsilon, \epsilon)$ for some $e \in \mathbf{E}$ whose signal has zero arguments

3. $SM' = ph2'(SM) = (\mathcal{E}', \mathcal{Q}', \mathcal{R}', Q_I, \mathcal{F}')$

4. $s \in \{e \in \widehat{\mathbf{E}} \mid \exists e' \in \mathcal{E} : e = e'\}^*$

PROVE: $(\forall t \in [\![SM]\!] : \neg(t \lhd s)) \Leftrightarrow s \in [\![SM']\!]$

$\langle 1\rangle 1$. ASSUME: 1.1. $(\forall t \in [\![SM]\!] : \neg(t \lhd s))$

PROVE: $s \in [\![SM']\!]$

$\langle 2\rangle 1$. $\forall sp \in \widehat{\mathbf{E}}^* : sp \sqsubseteq s \implies sp \in [\![SM']\!]$

$\langle 3\rangle 1$. ASSUME: 2.1 $sp \sqsubseteq s$ for some $sp \in \widehat{\mathbf{E}}^*$

PROVE: $sp \in [\![SM']\!]$

$\langle 4\rangle 1$. CASE: 3.1. $sp = \langle\rangle$

PROOF: $Q_I \in \mathcal{F}'$ by definition of $ph2'$ (Def. 23) and assumption 3. This means that $\langle\rangle \in [\![SM']\!]$ by definition of $[\![\ ]\!]$ (Def.12).

$\langle 4\rangle 2$. CASE: 3.1. $sp = sp' \frown \langle e\rangle$ for some $sp' \in [\![SM']\!]$ and $e \in \widehat{\mathbf{E}}$

$\langle 5\rangle 1$. Choose $e_1, e_2, \ldots, e_n \in \mathbf{E}$ such that $sp' = \langle e_1, e_2, \ldots, e_n\rangle$

PROOF: By assumption 2.1 and assumption 3.1.

$\langle 5\rangle 2$. Choose $Q_1, Q_2, \ldots, Q_n \in \mathcal{Q}'$ such that $Q_I \xrightarrow{e_1} Q_1 \xrightarrow{e_2} Q_2 \cdots \xrightarrow{e_n} Q_n \in \mathcal{R}'$

PROOF: By assumption 2, case assumption 3.1, ⟨5⟩1, and definition of $[\![\ ]\!]$ (Def.12).

⟨5⟩3. Choose $Q_{n+1} \in \mathcal{F}$ such that $Q_n \xrightarrow{e} Q_{n+1} \in \mathcal{R}'$

⟨6⟩1. ASSUME: 4.1 $\neg(\exists Q_{n+1} \in \mathcal{F} : Q_n \xrightarrow{e} Q_{n+1} \in \mathcal{R}')$
PROVE: False

⟨7⟩1. Choose $q_n \in Q_n$ and $q_f \in \mathcal{F}$ such that $q_n \xrightarrow{e} q_f \in \mathcal{R}$
PROOF: Assumption 4.1 implies that $St(Q_n, e) \cap \mathcal{F} = \emptyset$ where St is the function defined in (Def. 23) (otherwise we would have $Q_n \xrightarrow{e} Q_n \cup St(Q_n, e)$ which contradicts assumption 4.1). By ⟨5⟩2, assumption 3, and definition of $ph2'$ (Def. 23), this means that ⟨7⟩1 holds.

⟨7⟩2. Choose $u \in \widehat{\mathbf{E}}^*$ such that $q_I \xrightarrow{u} q_n \in \mathcal{R}$ and $u \lhd sp'$
PROOF: By definition of $ph2'$, we know that for each transition $q' \xrightarrow{e} q''$ (where $q'' \notin \mathcal{F}$) of SM, there is a corresponding transition $Q' \xrightarrow{e} Q''$ (where $q' \in Q'$ and $q'' \in Q''$) in the inverted state machine SM'. Since we have $Q_I \xrightarrow{sp'} Q_n \in \mathcal{R}'$ by ⟨5⟩1 and ⟨5⟩2, it is easy to see that we can choose u such that $q_I \xrightarrow{u} q_n \in \mathcal{R}$ and $u \lhd sp'$.

⟨7⟩3. $u \frown \langle e \rangle \in [\![SM]\!]$
PROOF: By ⟨7⟩1, ⟨7⟩2, assumption 1, and definition of $[\![\ ]\!]$ (Def.12).

⟨7⟩4. $u \frown \langle e \rangle \lhd sp' \frown \langle e \rangle$
PROOF: By ⟨7⟩2 and definition of $\lhd$ (Eq. (20)).

⟨7⟩5. Q.E.D.
PROOF:By assumption 1.1, no trace in $[\![SM]\!]$ can be a sub trace of s. However, by ⟨7⟩3 $u \frown \langle e \rangle$ is in $[\![SM]\!]$. Furthermore, $u \frown \langle e \rangle$ is a subtrace of s because $u \frown \langle e \rangle$ is a subtrace of sp (by ⟨7⟩4 and assumption 3.1) which is a prefix of s (by assumption 2.1). Hence assumption 4.1 cannot hold.

⟨6⟩2. Q.E.D.
PROOF: By contradiction.

⟨5⟩4. Q.E.D.
PROOF: By case assumption 3.1, ⟨5⟩1, ⟨5⟩2, ⟨5⟩3, assumption 3, and definition of $[\![\ ]\!]$ (Def.12).

⟨4⟩3. Q.E.D.
PROOF: By ⟨4⟩1, ⟨4⟩2, and induction over the length of sp.

⟨3⟩2. Q.E.D.
PROOF: By $\forall$-rule.

⟨2⟩2. Q.E.D.
PROOF: By ⟨2⟩1.

⟨1⟩2. ASSUME: 1.1 $s \in [\![SM']\!]$
PROVE: $(\forall t \in [\![SM]\!] : \neg(t \lhd s))$

⟨2⟩1. ASSUME: 2.1 $t \lhd s$ for some $t \in [\![SM]\!]$
PROVE: False

⟨3⟩1. Choose $t' = \langle e_1, \ldots, e_n \rangle \in \widehat{\mathbf{E}}^*$ such that $t' \sqsubseteq t$, and choose $q_1, \ldots, q_{n-1} \in \mathcal{Q} \backslash \mathcal{F}$ and $q_n \in \mathcal{F}$ such that $q_I \xrightarrow{e_1} q_1 \xrightarrow{e_2} q_2 \cdots \xrightarrow{e_{n-1}} q_{n-1} \xrightarrow{e_n} q_n \in \mathcal{R}$
PROOF: By assumption 2, assumption 2.1, assumption 1 (since $\langle\rangle \notin [\![SM]\!]$), and definition of $[\![\ ]\!]$ (Def.12).

⟨3⟩2. Choose $u \in \widehat{\mathbf{E}}^*$ such that $u \frown \langle e_n \rangle \sqsubseteq s$ and $\langle e_1, e_2, \ldots, e_{n-1} \rangle \lhd u$ and

$\neg(t' \lhd u)$, and choose $Q, Q' \in \mathcal{Q}$ such that $Q_I \xrightarrow{u} Q \xrightarrow{e_n} Q' \in \mathcal{R}'$

PROOF: By assumption 1.1, ⟨3⟩1, assumption 2, assumption 3, and definition of $[\![\ ]\!]$ (Def.12).

⟨3⟩3. $q_{n-1} \in Q$

PROOF: By assumption 2.1, ⟨3⟩1, ⟨3⟩2, assumption 3 and definition of $ph2'$ (Def. 23).

⟨3⟩4. Q.E.D.

PROOF: By definition of $ph2'$ (Def. 23), $Q \xrightarrow{e_n} Q'$ cannot hold since $q_{n-1} \in Q$ and $q_{n-1} \xrightarrow{e_n} q_n$ for $q_n \in \mathcal{F}$. Therefore assumption 2.1 implies a contradiction.

⟨2⟩2. Q.E.D.

PROOF: By contradiction.

⟨1⟩3. Q.E.D.

PROOF: By ⟨1⟩1 and ⟨1⟩2.

Lemma 3 Let SM be a well formed state machine such that $\langle\rangle \notin [\![SM]\!]$ and $ph2''$ be side effect free for SM, then $ph2''(SM)$ is an inversion of SM, i.e.,

$$inv(SM, ph2''(SM))$$

Proof of Lemma 3

ASSUME: 1. $SM = (\mathcal{E}, \mathcal{Q}, \mathcal{R}, q_I, \mathcal{F})$ for $SM \in \mathbf{SM}$ satisfying **SM1 - SM4** and $\langle\rangle \notin [\![SM]\!]$

2. $ph2''$ is side effect free for SM

3. $SM' = ph2''(SM) = (\mathcal{E}', \mathcal{Q}', \mathcal{R}', q'_I, \mathcal{F}')$

4. $s \in \{e \in \widehat{\mathbf{E}} \mid \exists e' \in \mathcal{E} : e = e'\}^*$

PROVE: $(\forall t \in [\![SM]\!] : \neg(t \lhd s)) \Leftrightarrow s \in [\![SM']\!]$

⟨1⟩1. ASSUME: 1.1. $(\forall t \in [\![SM]\!] : \neg(t \lhd s))$

PROVE: $s \in [\![SM']\!]$

⟨2⟩1. $\forall sp \in \widehat{\mathbf{E}}^* : sp \sqsubseteq s \implies sp \in [\![SM']\!]$

⟨3⟩1. ASSUME: 2.1 $sp \sqsubseteq s$ for some $sp \in \widehat{\mathbf{E}}^*$

PROVE: $sp \in [\![SM']\!]$

⟨4⟩1. CASE: 3.1. $sp = \langle\rangle$

PROOF: $q'_I \in \mathcal{F}'$ by definition of $ph2''$ (Def. 24) and assumption 3. This means that $\langle\rangle \in [\![SM']\!]$ by definition of $[\![\ ]\!]$ (Def.12).

⟨4⟩2. CASE: 3.1. $sp = sp' \frown \langle e \rangle$ for some $sp' \in [\![SM']\!]$ (the induction hypothesis) and $e \in \widehat{\mathbf{E}}$

⟨5⟩1. Choose $e_1, e_2, \ldots, e_n \in \widehat{\mathbf{E}}$ such that $sp' = \langle e_1, e_2, \ldots, e_n \rangle$

PROOF: By assumption 2.1 and assumption 3.1.

⟨5⟩2. Choose $t \in (\mathbf{E} \cup \{\epsilon\})^*$, $Q_n \in \mathcal{Q}'$ and $\sigma_I, \sigma'_n \in \widehat{\Sigma}_T$ such that $[q'_I, \sigma_I] \xrightarrow{t} [Q_n, \sigma'_n] \in EG(SM')$ and $t|_{\mathbf{E}} = sp'$

PROOF: By case assumption 3.1 (since $sp' \in [\![SM']\!]$) and definition of $[\![\ ]\!]$ (Def. 12).

⟨5⟩3. $t = \langle \epsilon \rangle \frown sp'$

PROOF: By assumption 1, SM is assumed to satisfy syntax constraints **SM2** and **SM3** that ensure that all transitions of SM except the outgoing transition for the initial state of SM are labeled by actions containing events. By definition of $ph2''$ (Def. 24) this means that the same constraints hold for the inverted state machine SM'. Therefore we can assert that $t = \langle \epsilon \rangle \frown sp'$.

$\langle 5 \rangle 4$. Choose
$Q_0, Q_1, Q_2, \ldots, Q_{n-1} \in \mathcal{Q}'$ and
$\sigma'_0, \ldots, \sigma'_{n-1} \in \widehat{\Sigma}_T$
such that
$[q'_I, \sigma_I] \xrightarrow{\epsilon} [Q_0, \sigma'_0] \xrightarrow{e_1} [Q_1, \sigma'_1] \xrightarrow{e_2} [Q_2, \sigma'_2] \cdots$
$\xrightarrow{e_n} [Q_n, \sigma'_n] \in EG(SM')$

PROOF: By $\langle 5 \rangle 1$, $\langle 5 \rangle 2$, and $\langle 5 \rangle 3$.

$\langle 5 \rangle 5$. Choose $Q_{n+1} \in \mathcal{F}'$ and $\sigma'_{n+1} \in \widehat{\Sigma}_T$ such that
$[Q_n, \sigma'_n] \xrightarrow{e} [Q_{n+1}, \sigma'_{n+1}] \in EG(SM')$

$\langle 6 \rangle 1$. Choose
$q_1, \ldots, q_k \in Q_n$,
$p_1, \ldots, p_k \in \mathcal{Q}$,
$e' \in \mathcal{E}$,
$bx_1, \ldots, bx_k \in \mathbf{BExp}$, and
$as_1, \ldots, as_k \in (\mathbf{Var} \times \mathbf{Exp})^*$
such that
$((q_1, (e', bx_1, as_1), p_1), \ldots, (q_k, (e', bx_k, as_k), p_k)) =$
$list(\{(q, (e'', bx, as), q') \in \mathcal{R} \mid q \in Q_n \wedge e = e''\})$

PROOF: By assumption 1, definition of the alphabet of state machines (Def. 10), and definition of the *list* function (see App. D.1.2).

$\langle 6 \rangle 2$. Choose $\phi \in \widehat{\Sigma}$ such that $\mathcal{D}om(\phi) = pvar(e')$ and $\phi(e') = e$

PROOF: By $\langle 6 \rangle 1$ and definition of the alphabet of state machines (Def. 10).

$\langle 6 \rangle 3$. Choose $Ix \subseteq \{1, \ldots, n\}$ such that $i \in Ix$ iff $eval(\sigma'_n[\phi](bx_i)) =$ $\mathtt{t}$

PROOF: Trivial.

LET: $bxs \stackrel{\text{def}}{=} (bx_1, bx_2, \ldots, bx_k)$

LET: $bx'' \stackrel{\text{def}}{=} conj((conj(Ix \textcircled{T}\, bxs), neg(disj(\{1, \ldots, n\} \backslash Ix \textcircled{T}\, bxs))))$

$\langle 6 \rangle 4$. $eval(\sigma'_n[\phi](bx'')) = \mathtt{t}$

PROOF: By $\langle 6 \rangle 3$, definition of bx'', and definition of *conj*, *disj*, and *neg* (Sect. D.1.2).

$\langle 6 \rangle 5$. $set(Ix \textcircled{T}\, (p_1, \ldots, p_k)) \cap \mathcal{F} = \emptyset$

$\langle 7 \rangle 1$. ASSUME: 4.1 $set(Ix \textcircled{T}\, (p_1, \ldots, p_k)) \cap \mathcal{F} \neq \emptyset$
PROVE: False

$\langle 8 \rangle 1$. Choose some $j \in Ix$ such that $p_j \in \mathcal{F}$ and $eval(\sigma'_n[\phi](bx_j)) =$ $\mathtt{t}$

PROOF: By $\langle 6 \rangle 3$ and assumption 4.1.

$\langle 8 \rangle 2$. Choose
$q_0 \in Q_0$,
$\sigma_0, \sigma_1 \in \widehat{\Sigma}_T$, and
$u \in \widehat{\mathbf{E}}^*$
such that
$[q_I, \sigma_I] \xrightarrow{\epsilon} [q_0, \sigma_0] \xrightarrow{u} [q_j, \sigma_1] \in EG(SM)$ and
$comp([\![SM]\!], u, sp')$

PROOF: By assumption 1-3, $\langle 5 \rangle 4$, and Lemma 3.2.

$\langle 8 \rangle 3$. Choose $\sigma_2 \in \widehat{\Sigma}_T$ such that $[q_j, \sigma_1] \xrightarrow{e} [p_j, \sigma_2]$

⟨9⟩1. $eval(\sigma_1[\phi](e')) = e$
PROOF: By ⟨6⟩2.
⟨9⟩2. $eval(\sigma_1[\phi](bx_j)) = \mathtt{t}$
PROOF: By ⟨8⟩2, ⟨5⟩4, ⟨6⟩1, ⟨8⟩1 (since $q_j \xrightarrow{(e',bx_j,as_j)} p_j \in \mathcal{R}$), ⟨8⟩2, assumption 2, and definition of side effect freedom (Def. 25).
⟨9⟩3. Q.E.D.
PROOF: By ⟨9⟩1, ⟨9⟩2, and definition of the execution graph for state machines (Def. 11).
⟨8⟩4. $u \frown \langle e\rangle \in [\![SM]\!]$
PROOF: By ⟨8⟩1 (since $p_j \in \mathcal{F}$) and ⟨8⟩2 and ⟨8⟩3 (since $[q_I, \sigma_I] \xrightarrow{\langle\epsilon\rangle\frown u\frown\langle e\rangle} [p_j, \sigma_2]$), and definition of $[\![\ ]\!]$ (Def. 12).
⟨8⟩5. $u \frown \langle e\rangle \lhd s$
PROOF: We know that $u \lhd sp'$ (by ⟨8⟩2). By definition of $\lhd$, this means that $u \frown \langle e\rangle \lhd sp' \frown \langle e\rangle$. Since $sp' \frown \langle e\rangle \sqsubseteq s$ (by assumptions 2.1 and 3.1) we know that $u \frown \langle e\rangle \lhd s$ by definition of $\lhd$.
⟨8⟩6. Q.E.D.
PROOF: ⟨8⟩4 and ⟨8⟩5 contradict assumption 1.1, therefore assumption 4.1 does not hold.
⟨7⟩2. Q.E.D.
PROOF: By contradiction.
⟨6⟩6. $St(Q_n, e', Ix) \cap \mathcal{F} = \emptyset$ (where St is the function defined in (Def. 24)).
⟨7⟩1. $Vi(Q_n, e', Ix) =$
$$set(Ix \circledT ((q_1, (e', bx_1, as_1), p_1), \ldots, (q_n, (e', bx_n, as_n), p_n)))$$
PROOF:

$$\begin{array}{lll} & Vi(Q_n, e', Ix) & \\ = & set(Ix \circledT list(Vi(Q_n, e'))) & \text{By Def. 24} \\ = & set(Ix \circledT list(\{(q, (e'', bx, as), q') \in \mathcal{R} \mid & \\ & \quad q \in Q_n \wedge e = e''\})) & \text{By Def. 24} \\ = & set(Ix \circledT ((q_1, (e', bx_1, as_1), p_1), \ldots, & \\ & (q_k, (e', bx_k, as_k), p_k)))) & \text{By ⟨6⟩1} \end{array}$$

⟨7⟩2. $St(Q_n, e', Ix) = set(Ix \circledT (p_1, \ldots, p_k))$
PROOF:

$$\begin{array}{lll} & St(Q_n, e', Ix) & \\ = & \{q \in \mathcal{Q} \mid \exists q' \xrightarrow{(e',bx',as')} q'' & \\ & \in Vi(Q_n, e', Ix) : q = q''\} & \text{By Def. 24} \\ = & \{q \in \mathcal{Q} \mid \exists q' \xrightarrow{(e',bx',as')} q'' \in & \\ & set(Ix \circledT ((q_1, (e', bx_1, as_1), p_1), \ldots, & \\ & (q_k, (e', bx_k, as_k), p_k))) : q = q''\} & \text{By ⟨7⟩1} \\ = & set(Ix \circledT (p_1, \ldots, p_k)) & \text{By set abstraction} \end{array}$$

⟨7⟩3. Q.E.D.
PROOF: By ⟨6⟩5 and ⟨7⟩2.
⟨6⟩7. Q.E.D.

PROOF: By ⟨6⟩6 and definition of $ph2''$ (Def. 24)), we have that $Q_n \xrightarrow{(e',bx'',sa)} Q_n \cup St(Q_n, e', Ix) \in \mathcal{R}'$ for some assignment sequence sa (sa is not important in the current argument). This means that $[Q_n, \sigma'_n] \xrightarrow{e} [Q_{n+1}, \sigma'_{n+1}] \in EG(SM')$ (where $Q_{n+1} = Q_n \cup St(Q, e', Ix)$) by ⟨6⟩2 (since $\phi(e') = e$) and ⟨6⟩4 (since $eval(\sigma'_n[\phi](bx'')) = \mathtt{t}$) and definition of the execution graph of state machines (Def. 11).

⟨5⟩6. Q.E.D.

PROOF: By ⟨5⟩1 - ⟨5⟩5 and definition of $[\![\ ]\!]$ (Def. 12).

⟨4⟩3. Q.E.D.

PROOF: By ⟨4⟩1, ⟨4⟩2, and induction over the length of sp.

⟨3⟩2. Q.E.D.

PROOF: By ∀-rule.

⟨2⟩2. Q.E.D.

PROOF: By ⟨2⟩1.

⟨1⟩2. ASSUME: 1.1 $s \in [\![SM']\!]$

PROVE: $(\forall t \in [\![SM]\!] : \neg(t \lhd s))$

⟨2⟩1. ASSUME: 2.1 $t \lhd s$ for some $t \in [\![SM]\!]$

PROVE: False

⟨3⟩1. Choose

$t' = \langle e_1, \ldots, e_n \rangle \in \widehat{\mathbf{E}}^*$,
$q_0, q_1, \ldots, q_{n-1} \in \mathcal{Q} \setminus \mathcal{F}$,
$q_n \in \mathcal{F}$,
$\sigma_I, \sigma_0, \sigma_1, \ldots, \sigma_n \in \widehat{\Sigma}$, and

such that

$t' \lhd s$,
$\neg(\exists t'' \in [\![SM]\!] : t'' \sqsubset t' \wedge t'' \lhd s)$, and
$[q_I, \sigma_I] \xrightarrow{\epsilon} [q_0, \sigma_0] \xrightarrow{e_1} [q_1, \sigma_1] \xrightarrow{e_2} [q_2, \sigma_2] \cdots \xrightarrow{e_n} [q_n, \sigma_n] \in EG(SM)$

PROOF: By assumption 1, assumption 2.1 and definition of $[\![\ ]\!]$ (Def.12).

⟨3⟩2. Choose

$u \in \widehat{\mathbf{E}}^*$

such that

$(A)\, u \frown \langle e_n \rangle \sqsubseteq s$ and
$(B)\, comp([\![SM]\!], \langle e_1, e_2, \ldots, e_{n-1} \rangle, u)$

⟨5⟩1. $pr([\![SM]\!], \langle e_1, e_2, \ldots, e_{n-1} \rangle) \neq \emptyset$

PROOF: By ⟨3⟩1 and definition of pr (Def. 25).

⟨5⟩2. Choose $u \in \mathbf{E}^*$ such that $u \frown \langle e_n \rangle \sqsubseteq s$, $\langle e_1, e_2, \ldots, e_{n-1} \rangle \lhd u$, and $\neg(t' \lhd u)$.

PROOF: By assumption 2.1 and ⟨3⟩1, we know that $\langle e_1, \ldots, e_n \rangle \lhd s$. Choosing a prefix u of s such that ⟨5⟩2 is satisfied, is therefore possible.

⟨5⟩3. Q.E.D.

PROOF: By ⟨5⟩1, ⟨5⟩2, ⟨3⟩1, and definition of $comp(\_,\_,\_)$ (Def. 25).

⟨3⟩3. Choose

$Q_1, Q_2, Q_3 \in \mathcal{Q}'$, and
$\sigma'_1, \sigma'_2, \sigma'_3 \in \widehat{\Sigma}$

such that

$[q'_I, \sigma_I] \xrightarrow{\epsilon} [Q_1, \sigma'_1] \xrightarrow{u} [Q_2, \sigma'_2] \xrightarrow{e_n} [Q_3, \sigma'_3] \in EG(SM')$

⟨4⟩1. $u \frown \langle e_n \rangle \in [\![SM']\!]$

PROOF: By $\langle 3\rangle 2$, we have that $u \frown \langle e_n\rangle \sqsubseteq s$. Furhermore, by assumption 1.1, we know that $s \in [\![SM']\!]$. Since the semantic trace set of SM' is prefix-closed (because all states of SM' are final states), it is easy to see that $u \frown \langle e_n\rangle \in [\![SM']\!]$.

$\langle 4\rangle 2$. Choose
$u' \in (\mathbf{E} \cup \{\epsilon\})^*$,
$\sigma_3' \in \widehat{\Sigma}$, and $Q_3 \in \mathcal{Q}'$
such that
$[q_I', \sigma_I] \xrightarrow{u'} [Q_3, \sigma_3'] \in EG(SM')$, and
$u'|_{\mathbf{E}} = u \frown \langle e_n\rangle$

PROOF: By $\langle 4\rangle 1$ and definition of $[\![\_]\!]$ (Def. 12).

$\langle 4\rangle 3$. $u' = \langle \epsilon\rangle \frown u \frown \langle e_n\rangle$

PROOF: By assumption 1, SM is assumed to satisfy syntax constraints **SM2** and **SM3** that ensure that all transitions of SM except the outgoing transition for the initial state of SM are labeled by actions containing events. By definition of $ph2''$ (Def. 24) this means that the same constraints hold for the inverted state machine SM'. Therefore we can assert that $u' = \langle \epsilon\rangle \frown u \frown \langle e_n\rangle$ by $\langle 4\rangle 2$.

$\langle 4\rangle 4$. Q.E.D.

PROOF: By $\langle 4\rangle 2$ and $\langle 4\rangle 3$.

$\langle 3\rangle 4$. $q_{n-1} \in Q_2$

PROOF: By assumption 1-3, $\langle 3\rangle 1$, $\langle 3\rangle 3$, and Lemma 3.1.

$\langle 3\rangle 5$. Choose
$e' \in \mathcal{E}$ and $\phi \in \widehat{\Sigma}$
such that
$\mathcal{D}om(\phi) = pvar(e')$,
$\phi(e') = e_n$

PROOF: By $\langle 3\rangle 1$, and definition of the alphabet of state machines (Def. 10).

$\langle 3\rangle 6$. Choose
$bx \in \mathbf{BExp}$ and
$as \in (\mathbf{Var} \times \mathbf{Exp})^*$
such that
$q_{n-1} \xrightarrow{(e', bx, as)} q_n \in \mathcal{R}$ and
$eval(\sigma_{n-1}[\phi](bx)) = \mathtt{t}$

PROOF: By $\langle 3\rangle 1$, $\langle 3\rangle 5$, and definition of the execution graph of state machines (Def. 11).

$\langle 3\rangle 7$. $eval(\sigma_2'[\phi](bx)) = \mathtt{f}$

$\langle 4\rangle 1$. Choose $Q_2 \xrightarrow{(e', bx_2', as_2')} Q_3 \in \mathcal{R}'$ such that
$eval(\sigma_2'[\phi](bx_2')) = \mathtt{t}$

PROOF: By $\langle 3\rangle 3$, $\langle 3\rangle 5$, and definition of the execution graph of state machines (Def. 11).

$\langle 4\rangle 2$. $bx_2' = conj(b, neg(disj(b_1, \ldots, bx, \ldots, b_k)))$ for some $b, b_1, \ldots, b_k \in$ **BExp**

$\langle 5\rangle 1$. Choose $Ix \in \mathbb{P}(\mathbb{N})$ such that $Q_3 = Q_2 \cup St(Q_2, e', Ix, V_2)$

PROOF:By $\langle 4\rangle 1$ and definition of $ph2''$ (Def. 24).

$\langle 5\rangle 2$. Choose
$q_1', \ldots, q_m' \in Q_2$
$p_1', \ldots, p_m' \in \mathcal{Q}$
$bx_1, \ldots, bx_m \in \mathbf{BExp}$

$as_1, \ldots, as_m \in (\mathbf{Var} \times \mathbf{Exp})^*$
such that
$Vi(Q_2, e', Ix) =$
$set(Ix \circledR ((q'_1, (e', bx_1, as_1), p'_1), \ldots, (q'_m, (e', bx_m, as_m), p'_m))$
where Vi is the function defined in Def. 24.

PROOF: By ⟨4⟩1 and definition of Vi (Def. 24).

⟨5⟩3. Choose $j \in \mathbb{N}$ such that
$(q'_j, (e', bx_j, as_j), p_j) = (q_{n-1}, (e', bx, as), q_n)$

PROOF: By ⟨5⟩2, ⟨3⟩4 (since $q_{n-1} \in Q_2$), and ⟨3⟩6 (since $q_{n-1} \xrightarrow{(e',bx,as)} q_n \in \mathcal{R}$).

⟨5⟩4. $j \notin Ix$

⟨6⟩1. ASSUME: 3.1 $j \in Ix$
PROVE: False

⟨7⟩1. $St(Q_2, e', Ix) \cap \mathcal{F} \neq \emptyset$ where St is the function defined in Def. 24

PROOF: By ⟨5⟩2, ⟨5⟩3, assumption 3.1, and definition of St (Def. 24), we have that $p_j \in St(Q_2, e', Ix)$. By ⟨5⟩3 we know that $p_j = q_n$. Since $q_n \in \mathcal{F}$ (by ⟨3⟩1), this implies that $p_j \in \mathcal{F}$ which agian implies that ⟨7⟩1 holds.

⟨7⟩2. Q.E.D.

PROOF: ⟨7⟩1 contradicts ⟨4⟩1 by definition of $ph2''$ (Def. 24).

⟨6⟩2. Q.E.D.

By ⟨6⟩1, $j \in Ix$ leads to a contradiction, therefore $j \notin Ix$.

⟨5⟩5. $bx_j \in set(\mathbb{N} \setminus Ix \circledR (bx_1, \ldots, bx_m))$

PROOF: By ⟨5⟩2, ⟨5⟩4 and definition of $\circledR$ and *set* (see App. D.1.2).

⟨5⟩6. $bx'_2 = conj(Ix \circledR (bx_1, \ldots, bx_m), neg(disj(\mathbb{N} \setminus Ix \circledR (bx_1, \ldots, bx_m))))$

PROOF: By ⟨4⟩1, ⟨5⟩2 and definition of $ph2''$ (Def. 24).

⟨5⟩7. Q.E.D.

PROOF: By ⟨5⟩5 and ⟨5⟩6.

⟨4⟩3. Q.E.D.

PROOF: By ⟨4⟩1, ⟨4⟩2, and definition of *conj*, *disj*, and *neg*.

⟨3⟩8. $eval(\sigma_{n-1}[\phi](bx)) = eval(\sigma'_2[\phi](bx))$

⟨4⟩1. $\sigma_{n-1} \cap (var(bx) \times \mathbf{Exp}) = \sigma'_2 \cap (var(bx) \times \mathbf{Exp})$

⟨5⟩1. $[q_I, \sigma_I] \xrightarrow{\epsilon} [q_0, \sigma_0] \xrightarrow{e_1} [q_1, \sigma_1] \xrightarrow{e_2} [q_2, \sigma_2] \cdots \xrightarrow{e_{n-1}} [q_{n-1}, \sigma_{n-1}] \in EG(SM)$

PROOF: By ⟨3⟩1.

⟨5⟩2. $[q'_I, \sigma_I] \xrightarrow{\epsilon} [Q_1, \sigma'_1] \xrightarrow{u} [Q_2, \sigma'_2] \in EG(SM')$

PROOF: By ⟨3⟩3.

⟨5⟩3. $q_{n-1} \xrightarrow{(e',bx,as)} q_n \in \mathcal{R}$

PROOF: By ⟨3⟩6.

⟨5⟩4. $comp(\llbracket SM \rrbracket, \langle e_1, e_2, \ldots, e_{n-1} \rangle, u)$

PROOF: By ⟨3⟩2.

⟨5⟩5. Q.E.D.

PROOF: By ⟨5⟩1 - ⟨5⟩4, assumption 2, and definition of side-effect freedom (Def. 25).

⟨4⟩2. $\sigma_{n-1}[\phi] \cap (var(bx) \times \mathbf{Exp}) = \sigma'_2[\phi] \cap (var(bx) \times \mathbf{Exp})$

PROOF: By ⟨4⟩1 and definition of the data state overriding operator

⟨4⟩3. Q.E.D.

PROOF: By ⟨4⟩2 and definition of *eval* (see App. A.2.2).

⟨3⟩9. Q.E.D.

PROOF: We have that ⟨3⟩8 contradicts ⟨3⟩6 (which asserts $eval(\sigma_{n-1}[\phi](bx)) = \mathtt{t}$) and ⟨3⟩7 (which asserts $eval(\sigma'_2[\phi](bx)) = \mathtt{f}$).

⟨2⟩2. Q.E.D.

PROOF: By contradiction.

⟨1⟩3. Q.E.D.

PROOF: By ⟨1⟩1 and ⟨1⟩2.

Lemma 3.1 The proof is based on induction. We make use of the predicate $Ind \in \mathbf{SM} \times \mathbf{SM} \times \mathbb{N} \rightarrow \mathbb{B}$ which high-lights the induction:

$$
\begin{array}{l}
Ind((\mathcal{E}, \mathcal{Q}, \mathcal{R}, q_I, \mathcal{F}), (\mathcal{E}', \mathcal{Q}', \mathcal{R}', q'_I, \mathcal{F}'), m) \stackrel{\text{def}}{=} \\
\quad \forall n \in \mathbb{N} : \\
\quad\; \forall q_1 \in \mathcal{Q} \setminus \mathcal{F} : \cdots : \forall q_n \in \mathcal{Q} \setminus \mathcal{F} : \\
\quad\;\; \forall \sigma_1 \in \widehat{\Sigma}_T : \cdots : \forall \sigma_n \in \widehat{\Sigma}_T : \\
\quad\;\;\; \forall Q_1, Q_n \in \mathcal{Q}' : \\
\quad\;\;\;\; \forall \sigma'_1 \in \widehat{\Sigma}_T : \forall \sigma'_n \in \widehat{\Sigma}_T : \\
\quad\;\;\;\;\; \forall e_1 \in \mathbf{E} : \cdots : \forall e_n \in \mathbf{E} : \forall u \in \mathbf{E}^* \\
\quad\;\;\;\;\;\; \wedge\, n \leq m \\
\quad\;\;\;\;\;\; \wedge\, comp([\![SM]\!], \langle e_1, \ldots, e_{n-1} \rangle, u) \\
\quad\;\;\;\;\;\; \wedge\, [q_I, \sigma_I] \xrightarrow{\epsilon} [q_1, \sigma_1] \xrightarrow{e_1} [q_2, \sigma_2] \cdots \\
\quad\;\;\;\;\;\;\;\; \xrightarrow{e_n} [q_n, \sigma_n] \in EG((\mathcal{E}, \mathcal{Q}, \mathcal{R}, q_I, \mathcal{F})) \\
\quad\;\;\;\;\;\; \wedge\, [q_{I'}, \sigma_I] \xrightarrow{\epsilon} [Q_1, \sigma'_1] \xrightarrow{u} \\
\quad\;\;\;\;\;\;\;\; [Q_n, \sigma'_n] \in EG((\mathcal{E}', \mathcal{Q}', \mathcal{R}', q'_I, \mathcal{F}')) \\
\quad\;\;\;\;\;\;\;\; \Longrightarrow q_n \in Q_n
\end{array}
$$

ASSUME:
1. $SM = (\mathcal{E}, \mathcal{Q}, \mathcal{R}, q_I, \mathcal{F})$ for $SM \in \mathbf{SM}$ satisfying **SM1** - **SM4** and $\langle\rangle \notin [\![SM]\!]$
2. $ph2''$ is side effect free for SM
3. $SM' = ph2''(SM) = (\mathcal{E}', \mathcal{Q}', \mathcal{R}', q'_I, \mathcal{F}')$
4. $n \in \mathbb{N}$
5. $q_1, \in \mathcal{Q} \setminus \mathcal{F}, \ldots, q_n, \in \mathcal{Q} \setminus \mathcal{F}$
6. $\sigma_1, \ldots, \sigma_n \in \widehat{\Sigma}_T$
7. $Q_1, Q_n \in \mathcal{Q}'$
8. $\sigma'_1, \sigma'_n \in \widehat{\Sigma}_T$
9. $e_1, \ldots, e_n \in \mathbf{E}$
10. $comp([\![SM]\!], \langle e_1, \ldots, e_n \rangle, u)$ for $u \in \mathbf{E}^*$
11. $[q_I, \sigma_I] \xrightarrow{\epsilon} [q_1, \sigma_1] \xrightarrow{e_1} [q_2, \sigma_2] \cdots \xrightarrow{e_n} [q_n, \sigma_n] \in EG(SM)$
12. $[q'_I, \sigma_I] \xrightarrow{\epsilon} [Q_1, \sigma'_1] \xrightarrow{u} [Q_n, \sigma'_n] \in EG(SM')$

PROVE: $q_n \in Q_n$

⟨1⟩1. CASE: 1.1 $n = 0$

PROOF: Case assumption 1.1 leads to a contradiction. To see this, note that by case assumption 1.1, assumption 11, and definition of $[\![\_]\!]$ (Def. 12) have $\langle\rangle \in [\![SM]\!]$ which contradicts assumption 1.

⟨1⟩2. CASE: 1.1 $Ind(SM, SM', n-1)$

⟨2⟩1. Choose $u' \in \mathbf{E}^*$ such that $u' \frown \langle e_n \rangle \sqsubseteq u$ and $comp([\![SM]\!], \langle e_1, \ldots, e_{n-1} \rangle, u')$

⟨4⟩1. $pr([\![SM]\!], \langle e_1, \ldots, e_{n-1} \rangle) \neq \emptyset$

PROOF: By assumption 10 and definition of *pr* (Def. 25).

⟨4⟩2. Choose $u' \in \mathbf{E}^*$ such that $u' \frown \langle e_n \rangle \sqsubseteq u$ and $\langle e_1, \ldots, e_{n-1} \rangle \lhd u'$ and $\neg(\langle e_1, \ldots, e_n \rangle, u')$

PROOF: By assumption 10 and definition of $comp(\_,\_,\_)$ (Def. 25).

⟨4⟩3. $\neg \exists s' \in pr([\![SM]\!], \langle e_1, \ldots, e_{n-1} \rangle : \langle e_1, \ldots, e_{n-1} \rangle \sqsubset s' \wedge s' \lhd u'$

PROOF: By ⟨4⟩2.

⟨4⟩4. Q.E.D.

PROOF: By ⟨4⟩1 - ⟨4⟩3 and definition of $comp(\_,\_,\_)$ (Def. 25).

⟨2⟩2. Choose $Q_{n-2}, Q_{n-1} \in \mathcal{Q}'$ and $\sigma'_{n-2}, \sigma'_{n-1} \in \widehat{\Sigma}_T$ such that $[q'_I, \sigma_I] \xrightarrow{\epsilon}$ $[Q_1, \sigma'_1] \xrightarrow{u'} [Q_{n-2}, \sigma'_{n-2}] \xrightarrow{\langle e_n \rangle} [Q_{n-1}, \sigma'_{n-1}] \in EG(SM')$

PROOF: By assumption 12, ⟨2⟩1 and definition of the execution graph for state machines (Def. 11).

⟨2⟩3. $q_{n-1} \in Q_{n-2}$

PROOF: By ⟨2⟩1, ⟨2⟩2, assumptions 4-12, and induction hypothesis 1.1.

⟨2⟩4. Choose

$(e'_n, bx, as) \in \mathbf{Act}$ and

$\phi \in \widehat{\Sigma}$

such that

$q_{n-1} \xrightarrow{(e'_n, bx, as)} q_n \in \mathcal{R}$,

$\mathcal{D}om(\phi) = var(e'_n)$,

$\phi(e'_n) = e_n$, and

$eval(\sigma_{n-1}[\phi](bx)) = \mathtt{t}$

PROOF: By assumption 11 and definition of the execution graph for state machines (Def. 11).

⟨2⟩5. Choose

$(e'_n, bx', as') \in \mathbf{Act}$ and

such that

$Q_{n-2} \xrightarrow{(e'_n, bx', as')} Q_{n-1} \in \mathcal{R}'$,

$eval(\sigma'_{n-1}[\phi](bx')) = \mathtt{t}$

PROOF: By ⟨2⟩4 (since $\mathcal{D}om(\phi) = var(e'_n)$ and $\phi(e'_n) = e_n$) assumption 12, ⟨2⟩2, and definition of the execution graph for state machines (Def. 11).

⟨2⟩6. $q_n \in Q_{n-1}$

⟨3⟩1. Choose $Ix \in \mathbb{P}(\mathbb{N})$ such that $Q_{n-1} = Q_{n-2} \cup St(Q_{n-2}, e'_n, Ix)$ where St is the function defined in Def. 24.

PROOF: By ⟨2⟩5 and definition of $ph2''$ (Def. 24).

⟨3⟩2. Choose

$q'_1, \ldots, q'_m \in Q_{n-2}$

$p'_1, \ldots, p'_m \in \mathcal{Q}$

$bx_1, \ldots, bx_m \in \mathbf{BExp}$

$as_1, \ldots, as_m \in (\mathbf{Var} \times \mathbf{Exp})^*$

such that

$Vi(Q_{n-2}, e'_n, Ix) =$

$set(Ix \circledT ((q'_1, (e'_n, bx_1, as_1), p'_1), \ldots, (q'_m, (e'_n, bx_m, as_m), p'_m))$

where Vi is the function defined in Def. 24.

PROOF: By definition of Vi (Def. 24).

⟨3⟩3. Choose $j \in \mathbb{N}$ such that

$(q'_j, bx_j, as_j) = (q_{n-1}, (e'_n, bx, as), q_n)$

PROOF: By ⟨2⟩3, ⟨2⟩4, ⟨3⟩2 and definition of Vi (Def. 24).

⟨3⟩4. $j \in Ix$

⟨4⟩1. ASSUME: 2.1 $j \notin Ix$
PROVE: False

⟨5⟩1. $eval(\sigma'_{n-2}[\phi](bx)) = \mathtt{f}$

⟨6⟩1. $bx' = conj(Ix \circledT (bx_1, \ldots, bx_m), neg(\mathbb{N} \backslash Ix \circledT (bx_1, \ldots, bx_m))))$
PROOF: By ⟨3⟩1, ⟨3⟩2, and definition of $ph2''$ (Def. 24).

⟨6⟩2. $bx_j \in set(\mathbb{N} \setminus Ix \circledT (bx_1, \ldots, bx_m)))$
PROOF: By assumption 2.1 and definition of $\circledT$ (see App. D.1.2).

⟨6⟩3. Q.E.D.
PROOF: By ⟨6⟩1, and ⟨6⟩2, definition of *conj*, *disj*, and *neg* (see App. D.1.2), bx must evaluate to false (i.e., $eval(\sigma'_{n-2}[\phi](bx)) = \mathtt{f}$) since bx' evaluates to true ($eval(\sigma_{n-1}[\phi](bx)) = \mathtt{t}$) by ⟨2⟩5.

⟨5⟩2. $eval(\sigma_{n-1}[\phi](bx)) = eval(\sigma'_{n-2}[\phi](bx))$

⟨6⟩1. $[q_I, \sigma_I] \xrightarrow{\epsilon} [q_1, \sigma_1] \xrightarrow{e_1} [q_2, \sigma_2] \cdots \xrightarrow{e_{n-1}} [q_{n-1}, \sigma_{n-1}] \in EG(SM)$
PROOF: By assumption 13.

⟨6⟩2. $[q_I, \sigma_I] \xrightarrow{\epsilon} [Q_1, \sigma'_1] \xrightarrow{u'} [Q_{n-2}, \sigma'_{n-2}] \in EG(SM')$
PROOF: By assumption 14.

⟨6⟩3. $q_{n-1} \xrightarrow{(e'_n, bx, as)} q_n \in \mathcal{R}$
PROOF: By ⟨2⟩4.

⟨6⟩4. $comp([\![SM]\!], \langle e_1, \ldots, e_{n-1} \rangle, u')$
PROOF: By ⟨2⟩1.

⟨6⟩5. Q.E.D.
PROOF: By ⟨6⟩1 - ⟨6⟩4, assumption 2, and definition of side effect freedom (Def. 25).

⟨5⟩3. Q.E.D.
PROOF: ⟨5⟩1 and ⟨5⟩2 contradict ⟨2⟩4 (since it asserts that $eval(\sigma_{n-1}[\phi](bx)) = \mathtt{t}$)

⟨4⟩2. Q.E.D.
PROOF: By contradiction.

⟨3⟩5. $q_n \in St(Q_{n-2}, e'_n, Ix)$ where St is the function defined in Def. 24.
PROOF: By ⟨3⟩2 - ⟨3⟩4, and definition of St (Def. 24).

⟨3⟩6. Q.E.D.
PROOF: By ⟨3⟩1 and ⟨3⟩5.

⟨2⟩7. Q.E.D.
PROOF: By ⟨2⟩6 since by definition of $ph2''$ (Def. 24), we know that $Q \xrightarrow{\epsilon} Q' \in EG(SM')$ implies $Q \subseteq Q'$.

⟨1⟩3. Q.E.D.
PROOF: By ⟨1⟩1, ⟨1⟩2, and induction over n.

Lemma 3.2 The proof is based on induction. We make use of the predicate $Ind \in \mathbf{SM} \times \mathbf{SM} \times \mathbf{E}^* \to \mathbb{B}$ which high-lights the induction. Let $SM =$

$\overset{\text{def}}{=} (\mathcal{E}, \mathcal{Q}, \mathcal{R}, q_I, \mathcal{F})$, and $SM' \overset{\text{def}}{=} (\mathcal{E}', \mathcal{Q}', \mathcal{R}', q'_I, \mathcal{F}')$, then Ind is defined by

$$\begin{array}{l}
Ind(SM, SM', s) \overset{\text{def}}{=} \\
\quad \forall s' \in \mathbf{E}^* : \\
\quad\quad \forall Q_n \in \mathcal{Q}' : \forall q_n \in \mathcal{Q} \cap Q_n : \\
\quad\quad\quad \forall \sigma_I \in \widehat{\Sigma}_T : \forall \sigma'_n \in \widehat{\Sigma}_T : \\
\quad\quad\quad\quad \wedge\, s' \sqsubseteq s \\
\quad\quad\quad\quad \wedge\, [q'_I, \sigma_I] \xrightarrow{\langle\epsilon\rangle \frown s'} [Q_n, \sigma'_n] \in EG(SM') \\
\quad\quad\quad\quad\quad \Longrightarrow \exists u \in \mathbf{E}^* : \\
\quad\quad\quad\quad\quad\quad\quad \exists \sigma_n \in \widehat{\Sigma}_T : \\
\quad\quad\quad\quad\quad\quad\quad\quad \wedge\, [q_I, \sigma_I] \xrightarrow{\langle\epsilon\rangle \frown u} [q_n, \sigma_n] \in EG(SM) \\
\quad\quad\quad\quad\quad\quad\quad\quad \wedge\, comp(\llbracket SM \rrbracket, u, s')
\end{array}$$

ASSUME: 1. $SM = (\mathcal{E}, \mathcal{Q}, \mathcal{R}, q_I, \mathcal{F})$ for $SM \in$ **SM** satisfying **SM1** - **SM4** and $\langle\rangle \notin \llbracket SM \rrbracket$
2. $ph2''$ is side effect free for SM
3. $SM' = ph2''(SM) = (\mathcal{E}', \mathcal{Q}', \mathcal{R}', q_I, \mathcal{F}')$
4. $s \in \mathbf{E}^*$
5. $Q_n \in \mathcal{Q}'$
6. $\sigma_I, \sigma'_n \in \widehat{\Sigma}_T$
7. $q_n \in Q_n \cap \mathcal{Q}$
8. $[q'_I, \sigma_I] \xrightarrow{\langle\epsilon\rangle \frown s} [Q_n, \sigma'_n] \in EG(SM')$

PROVE: $\exists u \in \mathbf{E}^* :$
$\exists \sigma_n \in \widehat{\Sigma}_T :$
$\wedge\, [q_I, \sigma_I] \xrightarrow{\langle\epsilon\rangle \frown u} [q_n, \sigma_n] \in EG(SM)$
$\wedge\, comp(\llbracket SM \rrbracket, u, s)$

$\langle 1\rangle 1$. CASE: 1.1 $s = \langle\rangle$

$\langle 2\rangle 1$. Choose $q_I \xrightarrow{(\epsilon, \epsilon, sa)} q' \in \mathcal{R}$
PROOF: By assumption 1 since SM satisfies syntax constraint **SM2**.

$\langle 2\rangle 2$. $q'_I \xrightarrow{(\epsilon, \epsilon, sa)} \{q'\} \in \mathcal{R}'$
PROOF: By $\langle 2\rangle 1$, assumption 3, and definition of $ph2''$ (Def. 24).

$\langle 2\rangle 3$. $[q'_I, \sigma_I] \xrightarrow{\epsilon} [Q_n, \sigma'_n] \in EG(SM')$
PROOF: By assumption 8, case assumption 1.1, and assumptions 1 (since SM is assumed to satisfy syntax constraint **SM3**) and 3 and definition of $ph2''$ (Def. 24).

$\langle 2\rangle 4$. $q_n = q'$
PROOF: By assumption 7, $\langle 2\rangle 2$, and $\langle 2\rangle 3$ and definition of $ph2''$ (Def. 24) (since $Q_n = \{q'\}$).

$\langle 2\rangle 5$. Choose $\sigma_n \in \widehat{\Sigma}_T$ such that $[q_I, \sigma'_I] \xrightarrow{\epsilon} [q_n, \sigma_n] \in EG(SM)$
PROOF: By $\langle 2\rangle 1$, $\langle 2\rangle 4$ and definition of the execution graph for state machines (Def. 11).

$\langle 2\rangle 6$. Q.E.D.
PROOF: By $\langle 2\rangle 5$.

$\langle 1\rangle 2$. CASE: 1.1 $s = s' \frown \langle e\rangle$
1.2 $Ind(SM, SM', s')$

$\langle 2\rangle 1$. Choose $Q_{n-1} \in \mathcal{Q}'$ and $\sigma'_{n-1} \in \widehat{\Sigma}_T$ such that $[q'_I, \sigma'_I] \xrightarrow{\langle\epsilon\rangle \frown s'} [Q_{n-1}, \sigma'_{n-1}] \xrightarrow{e} [Q_n, \sigma'_n] \in EG(SM')$

PROOF: By assumption 8, assumption 1.1, and definition of the execution graph for state machines (Def. 11).

⟨2⟩2. Choose
$(e', bx', as') \in \mathbf{Act}$ and
$\phi \in \widehat{\Sigma}$
such that
$Q_{n-1} \xrightarrow{(e',bx',as')} Q_n \in \mathcal{R}'$
$\mathcal{D}om(\phi) = var(e')$,
$\phi(e') = e$, and
$eval(\sigma'_{n-1}[\phi](bx')) = \mathtt{t}$

PROOF: By ⟨2⟩1 and definition of the execution graph for state machines (Def. 11).

⟨2⟩3. CASE: 2.1 $\exists Ix \in \mathbb{P}(\mathbb{N}) : q_n \in St(Q_{n-1}, e', Ix)$

⟨3⟩1. Choose $Ix \in \mathbb{P}(\mathbb{N})$ and $q_{n-1} \xrightarrow{(e',bx,sa)} q_n \in \mathcal{R}$ such that $q_{n-1} \xrightarrow{(e',bx,sa)} q_n \in Vi(Q_{n-1}, e', Ix)$

PROOF: By case assumption 2.1 and definition of Vi (Def. 24).

⟨3⟩2. Choose
$\sigma_{n-1} \in \widehat{\Sigma}_T$ and,
$u' \in \mathbf{E}^*$
such that
$(A)\, [q_I, \sigma_I] \xrightarrow{\langle\epsilon\rangle \frown u'} [q_{n-1}, \sigma_{n-1}] \in EG(SM)$,
$(B)\, comp([\![SM]\!], u', s')$

PROOF: By assumptions 1.1, 1.2, and ⟨3⟩1.

⟨3⟩3. Choose $\sigma_n \in \widehat{\Sigma}_T$ such that
$[q_{n-1}, \sigma_{n-1}] \xrightarrow{e} [q_n, \sigma_n] \in EG(SM)$

PROOF: By ⟨2⟩1, ⟨2⟩2, ⟨3⟩2, assumption 2, and assumption 3.

⟨3⟩4. $comp([\![SM]\!], u' \frown \langle e \rangle, s' \frown \langle e \rangle)$

⟨4⟩1. $pr([\![SM]\!], u' \frown \langle e \rangle) \neq \emptyset$

PROOF:By ⟨3⟩2 and definition of pr (Def. 25).

⟨4⟩2. $u' \frown \langle e \rangle \lhd s' \frown \langle e \rangle$

PROOF:By ⟨3⟩2 and definition of $comp(\_, \_, \_)$ (Def. 25).

⟨4⟩3. $\neg(\exists s'' \in pr([\![SM]\!], u' \frown \langle e \rangle) : u' \frown \langle e \rangle \sqsubset s'' \wedge s'' \lhd u' \frown \langle e \rangle)$

PROOF:If ⟨4⟩3 does not holds, then we can choose $e' \in \mathbf{E}$ such that $u' \frown \langle e, e' \rangle \lhd s' \frown \langle e \rangle$, but by definition of $\lhd$, this is impossible.

⟨4⟩4. Q.E.D.

PROOF:By ⟨4⟩1 - ⟨4⟩3 and definition of $comp(\_, \_, \_)$ (Def. 25).

⟨3⟩5. Q.E.D.

PROOF: By ⟨3⟩2 - ⟨3⟩4.

⟨2⟩4. CASE: 2.1 $\neg(\exists Ix \in \mathbb{P}(\mathbb{N}) : q_n \in St(Q_{n-1}, e', Ix))$

⟨3⟩1. Choose
$q_{n-1} \in Q_{n-1} \cap \mathcal{Q}$,
$\sigma_{n-1} \in \widehat{\Sigma}_T$ and,
$u' \in \mathbf{E}^*$
such that
$(A)\, [q_I, \sigma_I] \xrightarrow{\langle\epsilon\rangle \frown u'} [q_{n-1}, \sigma_{n-1}] \in EG(SM)$,
$(B)\, comp([\![SM]\!], u', s')$,
$(C)\, q_{n-1} = q_n$

PROOF: (A) and (B) hold by case assumption 1.2. (C) holds by case assumption 7, assumption 2.1, and definition of $ph2''$ (Def. 24) since we have $Q_n = Q_{n-1}$.

⟨3⟩2. $[q_{n-1}, \sigma_{n-1}] \xrightarrow{\langle\rangle} [q_n, \sigma_{n-1}] \in EG(SM)$
PROOF: By ⟨3⟩1 and definition of the execution graph for state machines (Def. 11).

⟨3⟩3. $comp(\llbracket SM \rrbracket, u', s' \frown \langle e \rangle)$

⟨4⟩1. $pr(\llbracket SM \rrbracket, u') \neq \emptyset$
PROOF:By ⟨3⟩1 and definition of pr (Def. 25).

⟨4⟩2. $u' \lhd s' \frown \langle e \rangle$
PROOF:By ⟨3⟩1 and definition of $comp(\_,\_,\_)$ (Def. 25).

⟨4⟩3. $\neg(\exists s'' \in pr(\llbracket SM \rrbracket, u') : u' \frown \langle e \rangle \sqsubset s'' \wedge s'' \lhd u' \frown \langle e \rangle)$
PROOF:If ⟨4⟩3 holds, then there must be trace $t \in \llbracket SM \rrbracket$ such that $u' \frown \langle e \rangle \sqsubseteq t$. However, this contradicts case assumption 2.1.

⟨4⟩4. Q.E.D.
PROOF:By ⟨4⟩1 - ⟨4⟩3 and definition of $comp(\_,\_,\_)$ (Def. 25).

⟨3⟩4. Q.E.D.
PROOF: By ⟨3⟩1 - ⟨3⟩3.

⟨2⟩5. Q.E.D.
PROOF: By ⟨2⟩1 - ⟨2⟩4.

⟨1⟩3. Q.E.D.
PROOF: By ⟨1⟩1 and ⟨1⟩2 and induction.

Corollary 1 Let SM be a well formed state machine such that $\langle\rangle \notin \llbracket SM \rrbracket$ and $ph2$ be side effect free for SM, then $ph2(SM)$ is an inversion of SM, i.e.,

$$inv(SM, ph2(SM))$$

Proof of Corollary 1

ASSUME: 1. $SM = (\mathcal{E}, \mathcal{Q}, \mathcal{R}, q_I, \mathcal{F})$ for $SM \in$ **SM** satisfying **SM1** - **SM4** and $\langle\rangle \notin \llbracket SM \rrbracket$
2. $ph2$ is side effect free for SM
3. $SM' = ph2(SM) = (\mathcal{E}', \mathcal{Q}', \mathcal{R}', q'_I, \mathcal{F}')$
4. $s \in \{e \in \widehat{\mathbf{E}} \mid \exists e' \in \mathcal{E} : e = e'\}^*$

PROVE: $(\forall t \in \llbracket SM \rrbracket : \neg(t \lhd s)) \Leftrightarrow s \in \llbracket SM' \rrbracket$

⟨1⟩1. ASSUME: 1.1. $(\forall t \in \llbracket SM \rrbracket : \neg(t \lhd s))$
PROVE: $s \in \llbracket SM' \rrbracket$

⟨2⟩1. $\forall sp \in \widehat{\mathbf{E}}^* : sp \sqsubseteq s \implies sp \in \llbracket SM' \rrbracket$

⟨3⟩1. ASSUME: 2.1 $sp \sqsubseteq s$ for some $sp \in \widehat{\mathbf{E}}^*$
PROVE: $sp \in \llbracket SM' \rrbracket$

⟨4⟩1. CASE: 3.1. $sp = \langle\rangle$
PROOF: $q'_I \in \mathcal{F}'$ by definition of $ph2$ (Def. 26) and assumption 3. This means that $\langle\rangle \in \llbracket SM' \rrbracket$ by definition of $\llbracket\ \rrbracket$ (Def.12).

⟨4⟩2. CASE: 3.1. $sp = sp' \frown \langle e \rangle$ for some $sp' \in \llbracket SM' \rrbracket$ (the induction hypothesis) and $e \in \widehat{\mathbf{E}}$

⟨5⟩1. Choose $e_1, e_2, \ldots, e_n \in \widehat{\mathbf{E}}$ such that $sp' = \langle e_1, e_2, \ldots, e_n \rangle$
PROOF: By assumption 2.1 and assumption 3.1.

⟨5⟩2. Choose $t \in (\mathbf{E} \cup \{\epsilon\})^*$, $(Q_n, V_n) \in \mathcal{Q}'$ and $\sigma_I, \sigma'_n \in \widehat{\Sigma}_T$ such that $[q'_I, \sigma_I] \xrightarrow{t} [(Q_n, V_n), \sigma'_n] \in EG(SM')$ and $t|_{\mathbf{E}} = sp'$

PROOF: By case assumption 3.1 (since $sp' \in [\![SM']\!]$) and definition of $[\![\ ]\!]$ (Def. 12).

⟨5⟩3. $t = \langle\epsilon\rangle \frown sp'$

PROOF: By assumption 1, SM is assumed to satisfy syntax constraints **SM2** and **SM3** that ensure that all transitions of SM except the outgoing transition for the initial state of SM are labeled by actions containing events. By definition of $ph2$ (Def. 26) this means that the same constraints hold for the inverted state machine SM'. Therefore we can assert that $t = \langle\epsilon\rangle \frown sp'$.

⟨5⟩4. Choose

$(Q_0, V_0), (Q_1, V_1), (Q_2, V_2), \ldots, (Q_{n-1}, V_{n-1}) \in \mathcal{Q}'$ and

$\sigma'_0, \ldots, \sigma'_{n-1} \in \widehat{\Sigma}_T$

such that

$[q'_I, \sigma_I] \xrightarrow{\epsilon} [(Q_0, V_0), \sigma'_0] \xrightarrow{e_1} [(Q_1, V_1), \sigma'_1] \xrightarrow{e_2} [(Q_2, V_2), \sigma'_2] \cdots$
$\xrightarrow{e_n} [(Q_n, V_n), \sigma'_n] \in EG(SM')$

PROOF: By ⟨5⟩1, ⟨5⟩2, and ⟨5⟩3.

⟨5⟩5. Choose $(Q_{n+1}, V_{n+1}) \in \mathcal{F}'$ and $\sigma'_{n+1} \in \widehat{\Sigma}_T$ such that

$[(Q_n, V_n), \sigma'_n] \xrightarrow{e} [(Q_{n+1}, V_{n+1}), \sigma'_{n+1}] \in EG(SM')$

⟨6⟩1. Choose

$q_1, \ldots, q_k \in Q_n$,

$p_1, \ldots, p_k \in \mathcal{Q}$,

$e' \in \mathcal{E}$,

$bx_1, \ldots, bx_k \in$ **BExp**, and

$as_1, \ldots, as_k \in (\mathbf{Var} \times \mathbf{Exp})^*$

such that

$((q_1, (e', bx_1, as_1), p_1), \ldots, (q_k, (e', bx_k, as_k), p_k)) =$
$list(\{(q, (e'', bx, as), q') \in \mathcal{R} \mid q \in Q_n \wedge e = e''\} \setminus V_n)$

PROOF: By assumption 1, definition of the alphabet of state machines (Def. 10), and definition of the $list$ function (see App. D.1.2).

⟨6⟩2. Choose $\phi \in \widehat{\Sigma}$ such that $\mathcal{D}om(\phi) = pvar(e')$ and $\phi(e') = e$

PROOF: By ⟨6⟩1 and definition of the alphabet of state machines (Def. 10).

⟨6⟩3. Choose $Ix \subseteq \{1, \ldots, n\}$ such that $i \in Ix$ iff $eval(\sigma'_n[\phi](bx_i)) =$ t

PROOF: Trivial.

LET: $bxs \overset{\text{def}}{=} (bx_1, bx_2, \ldots, bx_k)$

LET: $bx'' \overset{\text{def}}{=} conj((conj(Ix \circledS bxs), neg(disj(\{1, \ldots, n\} \backslash Ix \circledS bxs))))$

⟨6⟩4. $eval(\sigma'_n[\phi](bx'')) = \mathtt{t}$

PROOF: By ⟨6⟩3, definition of bx'', and definition of $conj$, $disj$, and neg (Sect. D.1.2).

⟨6⟩5. $set(Ix \circledS (p_1, \ldots, p_k)) \cap \mathcal{F} = \emptyset$

⟨7⟩1. ASSUME: 4.1 $set(Ix \circledS (p_1, \ldots, p_k)) \cap \mathcal{F} \neq \emptyset$

PROVE: False

⟨8⟩1. Choose some $j \in Ix$ such that $p_j \in \mathcal{F}$ and $eval(\sigma'_n[\phi](bx_j)) =$ t

PROOF: By ⟨6⟩3 and assumption 4.1.

⟨8⟩2. Choose

$q_0 \in Q_0$,
$\sigma_0, \sigma_1 \in \widehat{\Sigma}_T$, and
$u \in \widehat{\mathbf{E}}^*$
such that
$[q_I, \sigma_I] \xrightarrow{\epsilon} [q_0, \sigma_0] \xrightarrow{u} [q_j, \sigma_1] \in EG(SM)$ and
$comp([\![SM]\!], u, sp')$
PROOF: By assumption 1-3, ⟨5⟩4, and Corollary 3.2.

⟨8⟩3. Choose $\sigma_2 \in \widehat{\Sigma}_T$ such that $[q_j, \sigma_1] \xrightarrow{e} [p_j, \sigma_2]$

⟨9⟩1. $eval(\sigma_1[\phi](e')) = e$
PROOF: By ⟨6⟩2.

⟨9⟩2. $eval(\sigma_1[\phi](bx_j)) = \mathtt{t}$
PROOF: By ⟨8⟩2, ⟨5⟩4, ⟨6⟩1, ⟨8⟩1 (since $q_j \xrightarrow{(e', bx_j, as_j)} p_j \in \mathcal{R}$), ⟨8⟩2, assumption 2, and definition of side effect freedom (Def. 25).

⟨9⟩3. Q.E.D.
PROOF: By ⟨9⟩1, ⟨9⟩2, and definition of the execution graph for state machines (Def. 11).

⟨8⟩4. $u \frown \langle e \rangle \in [\![SM]\!]$
PROOF: By ⟨8⟩1 (since $p_j \in \mathcal{F}$) and ⟨8⟩2 and ⟨8⟩3 (since $[q_I, \sigma_I] \xrightarrow{\langle\epsilon\rangle \frown u \frown \langle e \rangle} [p_j, \sigma_2]$), and definition of $[\![\ ]\!]$ (Def. 12).

⟨8⟩5. $u \frown \langle e \rangle \lhd s$
PROOF: We know that $u \lhd sp'$ (by ⟨8⟩2). By definition of $\lhd$, this means that $u \frown \langle e \rangle \lhd sp' \frown \langle e \rangle$. Since $sp' \frown \langle e \rangle \sqsubseteq s$ (by assumptions 2.1 and 3.1) we know that $u \frown \langle e \rangle \lhd s$ by definition of $\lhd$.

⟨8⟩6. Q.E.D.
PROOF: ⟨8⟩4 and ⟨8⟩5 contradict assumption 1.1, therefore assumption 4.1 does not hold.

⟨7⟩2. Q.E.D.
PROOF: By contradiction.

⟨6⟩6. $St(Q_n, e', Ix, V_n) \cap \mathcal{F} = \emptyset$ (where St is the function defined in (Def. 26)).

⟨7⟩1. $Vi(Q_n, e', Ix, V_n) =$
$set(Ix \oplus ((q_1, (e', bx_1, as_1), p_1), \ldots, (q_n, (e', bx_n, as_n), p_n)))$
PROOF:

$$\begin{array}{lll}
 & Vi(Q_n, e', Ix, V_n) & \\
= & set(Ix \oplus list(Vi(Q_n, e', V_n))) & \text{By Def. 26} \\
= & set(Ix \oplus list(\{(q, (e'', bx, as), q') \in \mathcal{R} \mid & \\
 & \quad q \in Q_n \wedge e = e''\} \setminus V_n)) & \text{By Def. 26} \\
= & set(Ix \oplus ((q_1, (e', bx_1, as_1), p_1), \ldots, & \\
 & \quad (q_k, (e', bx_k, as_k), p_k)))) & \text{By } \langle 6 \rangle 1
\end{array}$$

⟨7⟩2. $St(Q_n, e', Ix, V_n) = set(Ix \oplus (p_1, \ldots, p_k))$

PROOF:

$$\begin{array}{lll} & St(Q_n, e', Ix, V_n) & \\ = & \{q \in \mathcal{Q} \mid \exists q' \xrightarrow{(e', bx', as')} q'' & \\ & \in Vi(Q_n, e', Ix, V_n) : q = q''\} & \text{By Def. 26} \\ = & \{q \in \mathcal{Q} \mid \exists q' \xrightarrow{(e', bx', as')} q'' \in & \\ & \quad set(Ix \circledT ((q_1, (e', bx_1, as_1), p_1), \ldots, & \\ & \quad (q_k, (e', bx_k, as_k), p_k))) : q = q''\} & \text{By } \langle 7\rangle 1 \\ = & set(Ix \circledT (p_1, \ldots, p_k)) & \text{By set abstraction} \end{array}$$

⟨7⟩3. Q.E.D.
PROOF: By ⟨6⟩5 and ⟨7⟩2.

⟨6⟩7. Q.E.D.
PROOF: By ⟨6⟩6, definition of $ph2$ (Def. 26), ⟨6⟩2, and definition of the execution graph of state machines (Def. 11).

⟨5⟩6. Q.E.D.
PROOF: By ⟨5⟩1 - ⟨5⟩5 and definition of $[\![\ ]\!]$ (Def. 12).

⟨4⟩3. Q.E.D.
PROOF: By ⟨4⟩1, ⟨4⟩2, and induction over the length of sp.

⟨3⟩2. Q.E.D.
PROOF: By ∀-rule.

⟨2⟩2. Q.E.D.
PROOF: By ⟨2⟩1.

⟨1⟩2. ASSUME: 1.1 $s \in [\![SM']\!]$
PROVE: $(\forall t \in [\![SM]\!] : \neg(t \lhd s))$

⟨2⟩1. ASSUME: 2.1 $t \lhd s$ for some $t \in [\![SM]\!]$
PROVE: False

⟨3⟩1. Choose
$t' = \langle e_1, \ldots, e_n \rangle \in \widehat{\mathbf{E}}^*$,
$q_0, q_1, \ldots, q_{n-1} \in \mathcal{Q} \setminus \mathcal{F}$,
$q_n \in \mathcal{F}$,
$\sigma_I, \sigma_0, \sigma_1, \ldots, \sigma_n \in \widehat{\Sigma}$, and
such that
$t' \lhd s$,
$\neg(\exists t'' \in [\![SM]\!] : t'' \sqsubset t' \wedge t'' \lhd s)$, and
$[q_I, \sigma_I] \xrightarrow{\epsilon} [q_0, \sigma_0] \xrightarrow{e_1} [q_1, \sigma_1] \xrightarrow{e_2} [q_2, \sigma_2] \cdots \xrightarrow{e_n} [q_n, \sigma_n] \in EG(SM)$
PROOF: By assumption 1, assumption 2.1 and definition of $[\![\ ]\!]$ (Def.12).

⟨3⟩2. Choose
$u \in \widehat{\mathbf{E}}^*$
such that
$(A)\, u \frown \langle e_n \rangle \sqsubseteq s$ and
$(B)\, comp([\![SM]\!], \langle e_1, e_2, \ldots, e_{n-1} \rangle, u)$

⟨5⟩1. $pr([\![SM]\!], \langle e_1, e_2, \ldots, e_{n-1} \rangle) \neq \emptyset$
PROOF: By ⟨3⟩1 and definition of pr (Def. 25).

⟨5⟩2. Choose $u \in \mathbf{E}^*$ such that $u \frown \langle e_n \rangle \sqsubseteq s$, $\langle e_1, e_2, \ldots, e_{n-1} \rangle \lhd u$, and $\neg(t' \lhd u)$.
PROOF: By assumption 2.1 and ⟨3⟩1, we know that $\langle e_1, \ldots, e_n \rangle \lhd s$. Choosing a prefix u of s such that ⟨5⟩2 is satisfied, is therefore possible.

⟨5⟩3. Q.E.D.
PROOF: By ⟨5⟩1, ⟨5⟩2, ⟨3⟩1, and definition of $comp(\_,\_,\_)$ (Def. 25).

⟨3⟩3. Choose
$(Q_1, V_1), (Q_2, V_2), (Q_3, V_3) \in \mathcal{Q}'$, and
$\sigma'_1, \sigma'_2, \sigma'_3 \in \widehat{\Sigma}$
such that
$[q'_I, \sigma_I] \xrightarrow{\epsilon} [(Q_1, V_1), \sigma'_1] \xrightarrow{u} [(Q_2, V_2), \sigma'_2] \xrightarrow{e_n} [(Q_3, V_3), \sigma'_3] \in EG(SM')$

⟨4⟩1. $u \frown \langle e_n \rangle \in [\![SM']\!]$
PROOF: By ⟨3⟩2, we have that $u \frown \langle e_n \rangle \sqsubseteq s$. Furhermore, by assumption 1.1, we know that $s \in [\![SM']\!]$. Since the semantic trace set of SM' is prefix-closed (because all states of SM' are final states), it is easy to see that $u \frown \langle e_n \rangle \in [\![SM']\!]$.

⟨4⟩2. Choose
$u' \in (\mathbf{E} \cup \{\epsilon\})^*$,
$\sigma'_3 \in \widehat{\Sigma}$, and $(Q_3, V_3) \in \mathcal{Q}'$
such that
$[q'_I, \sigma_I] \xrightarrow{u'} [(Q_3, V_3), \sigma'_3] \in EG(SM')$, and
$u'|_{\mathbf{E}} = u \frown \langle e_n \rangle$
PROOF: By ⟨4⟩1 and definition of $[\![\_]\!]$ (Def. 12).

⟨4⟩3. $u' = \langle \epsilon \rangle \frown u \frown \langle e_n \rangle$
PROOF: By assumption 1, SM is assumed to satisfy syntax constraints **SM2** and **SM3** that ensure that all transitions of SM except the outgoing transition for the initial state of SM are labeled by actions containing events. By definition of $ph2$ (Def. 26) this means that the same constraints hold for the inverted state machine SM'. Therefore we can assert that $u' = \langle \epsilon \rangle \frown u \frown \langle e_n \rangle$ by ⟨4⟩2.

⟨4⟩4. Q.E.D.
PROOF: By ⟨4⟩2 and ⟨4⟩3.

⟨3⟩4. $q_{n-1} \in Q_2$
PROOF: By assumption 1-3, ⟨3⟩1, ⟨3⟩3, and Corollary 3.1.

⟨3⟩5. Choose
$e' \in \mathcal{E}$ and $\phi \in \widehat{\Sigma}$
such that
$\mathcal{D}om(\phi) = pvar(e')$,
$\phi(e') = e_n$
PROOF: By ⟨3⟩1, and definition of the alphabet of state machines (Def. 10).

⟨3⟩6. Choose
$bx \in \mathbf{BExp}$ and
$as \in (\mathbf{Var} \times \mathbf{Exp})^*$
such that
$q_{n-1} \xrightarrow{(e', bx, as)} q_n \in \mathcal{R}$ and
$eval(\sigma_{n-1}[\phi](bx)) = \mathtt{t}$
PROOF: By ⟨3⟩1, ⟨3⟩5, and definition of the execution graph of state machines (Def. 11).

⟨3⟩7. $eval(\sigma'_2[\phi](bx)) = \mathtt{f}$

⟨4⟩1. Choose $(Q_2, V_2) \xrightarrow{(e', bx'_2, as'_2)} (Q_3, V_3) \in \mathcal{R}'$ such that
$eval(\sigma'_2[\phi](bx'_2)) = \mathtt{t}$

PROOF: By ⟨3⟩3, ⟨3⟩5, and definition of the execution graph of state machines (Def. 11).

⟨4⟩2. $bx'_2 = conj(b, neg(disj(b_1, \ldots, bx, \ldots, b_k)))$ for some $b, b_1, \ldots, b_k \in$ **BExp**

⟨5⟩1. Choose $Ix \in \mathbb{P}(\mathbb{N})$ such that $Q_3 = Q_2 \cup St(Q_2, e', Ix)$

PROOF:By ⟨4⟩1 and definition of $ph2$ (Def. 26).

⟨5⟩2. Choose

$q'_1, \ldots, q'_m \in Q_2$

$p'_1, \ldots, p'_m \in Q$

$bx_1, \ldots, bx_m \in \mathbf{BExp}$

$as_1, \ldots, as_m \in (\mathbf{Var} \times \mathbf{Exp})^*$

such that

$Vi(Q_2, e', Ix, V_2) =$

$set(Ix \circledT ((q'_1, (e', bx_1, as_1), p'_1), \ldots, (q'_m, (e', bx_m, as_m), p'_m))$

where Vi is the function defined in Def. 26.

PROOF: By ⟨4⟩1 and definition of Vi (Def. 26).

⟨5⟩3. Choose $j \in \mathbb{N}$ such that

$(q'_j, (e', bx_j, as_j), p_j) = (q_{n-1}, (e', bx, as), q_n)$

PROOF: By ⟨5⟩2, ⟨3⟩4 (since $q_{n-1} \in Q_2$), and ⟨3⟩6 (since $q_{n-1} \xrightarrow{(e', bx, as)} q_n \in \mathcal{R}$).

⟨5⟩4. $j \notin Ix$

⟨6⟩1. ASSUME: 3.1 $j \in Ix$

PROVE: False

⟨7⟩1. $St(Q_2, e', Ix, V_2) \cap \mathcal{F} \neq \emptyset$ where St is the function defined in Def. 26

PROOF: By ⟨5⟩2, ⟨5⟩3, assumption 3.1, and definition of St (Def. 26), we have that $p_j \in St(Q_2, e', Ix, V_2)$. By ⟨5⟩3 we know that $p_j = q_n$. Since $q_n \in \mathcal{F}$ (by ⟨3⟩1), this implies that $p_j \in \mathcal{F}$ which agian implies that ⟨7⟩1 holds.

⟨7⟩2. Q.E.D.

PROOF: ⟨7⟩1 contradicts ⟨4⟩1 by definition of $ph2$ (Def. 26).

⟨6⟩2. Q.E.D.

By ⟨6⟩1, $j \in Ix$ leads to a contradiction, therefore $j \notin Ix$.

⟨5⟩5. $bx_j \in set(\mathbb{N} \setminus Ix \circledT (bx_1, \ldots, bx_m))$

PROOF: By ⟨5⟩2, ⟨5⟩4 and definition of $\circledT$ and set (see App. D.1.2).

⟨5⟩6. $bx'_2 = conj(Ix \circledT (bx_1, \ldots, bx_m), neg(disj(\mathbb{N} \setminus Ix \circledT (bx_1, \ldots, bx_m))))$

PROOF: By ⟨4⟩1, ⟨5⟩2 and definition of $ph2$ (Def. 26).

⟨5⟩7. Q.E.D.

PROOF: By ⟨5⟩5 and ⟨5⟩6.

⟨4⟩3. Q.E.D.

PROOF: By ⟨4⟩1, ⟨4⟩2, and definition of $conj$, $disj$, and neg.

⟨3⟩8. $eval(\sigma_{n-1}[\phi](bx)) = eval(\sigma'_2[\phi](bx))$

⟨4⟩1. $\sigma_{n-1} \cap (var(bx) \times \mathbf{Exp}) = \sigma'_2 \cap (var(bx) \times \mathbf{Exp})$

⟨5⟩1. $[q_I, \sigma_I] \xrightarrow{\epsilon} [q_0, \sigma_0] \xrightarrow{e_1} [q_1, \sigma_1] \xrightarrow{e_2} [q_2, \sigma_2] \cdots \xrightarrow{e_{n-1}} [q_{n-1}, \sigma_{n-1}] \in EG(SM)$

PROOF: By ⟨3⟩1.

⟨5⟩2. $[q'_I, \sigma_I] \xrightarrow{\epsilon} [(Q_1, V_1), \sigma'_1] \xrightarrow{u} [(Q_2, V_2), \sigma'_2] \in EG(SM')$

PROOF: By ⟨3⟩3.

⟨5⟩3. $q_{n-1} \xrightarrow{(e',bx,as)} q_n \in \mathcal{R}$
PROOF: By ⟨3⟩6.
⟨5⟩4. $comp([\![SM]\!], \langle e_1, e_2, \ldots, e_{n-1}\rangle, u)$
PROOF: By ⟨3⟩2.
⟨5⟩5. Q.E.D.
PROOF: By ⟨5⟩1 - ⟨5⟩4, assumption 2, and definition of side-effect freedom (Def. 25).
⟨4⟩2. $\sigma_{n-1}[\phi] \cap (var(bx) \times \mathbf{Exp}) = \sigma'_2[\phi] \cap (var(bx) \times \mathbf{Exp})$
PROOF: By ⟨4⟩1 and definition of the data state overriding operator
⟨4⟩3. Q.E.D.
PROOF: By ⟨4⟩2 and definition of *eval* (see App. A.2.2).
⟨3⟩9. Q.E.D.
PROOF:We have that ⟨3⟩8 contradicts ⟨3⟩6 (which asserts $eval(\sigma_{n-1}[\phi](bx)) =$ t) and ⟨3⟩7 (which asserts $eval(\sigma'_2[\phi](bx)) = \mathtt{f}$).
⟨2⟩2. Q.E.D.
PROOF: By contradiction.
⟨1⟩3. Q.E.D.
PROOF: By ⟨1⟩1 and ⟨1⟩2.

Corollary 1.1 The proof is based on induction. We make use of the predicate $Ind \in \mathbf{SM} \times \mathbf{SM} \times \mathbb{N} \to \mathbb{B}$ which high-lights the induction:

$$
\begin{array}{l}
Ind((\mathcal{E}, \mathcal{Q}, \mathcal{R}, q_I, \mathcal{F}), (\mathcal{E}', \mathcal{Q}', \mathcal{R}', q'_I, \mathcal{F}'), m) \stackrel{\text{def}}{=} \\
\forall n \in \mathbb{N} : \\
\quad \forall q_1 \in \mathcal{Q} \setminus \mathcal{F} : \cdots : \forall q_n \in \mathcal{Q} \setminus \mathcal{F} : \\
\quad\quad \forall \sigma_1 \in \widehat{\Sigma}_T : \cdots : \forall \sigma_n \in \widehat{\Sigma}_T : \\
\quad\quad\quad \forall (Q_1, V_1), (Q_n, V_n) \in \mathcal{Q}' : \\
\quad\quad\quad\quad \forall \sigma'_1 \in \widehat{\Sigma}_T : \forall \sigma'_n \in \widehat{\Sigma}_T : \\
\quad\quad\quad\quad\quad \forall e_1 \in \mathbf{E} : \cdots : \forall e_n \in \mathbf{E} : \forall u \in \mathbf{E}^* \\
\quad\quad\quad\quad\quad\quad \wedge\, n \leq m \\
\quad\quad\quad\quad\quad\quad \wedge\, comp([\![SM]\!], \langle e_1, \ldots, e_{n-1}\rangle, u) \\
\quad\quad\quad\quad\quad\quad \wedge\, [q_I, \sigma_I] \xrightarrow{\epsilon} [q_1, \sigma_1] \xrightarrow{e_1} [q_2, \sigma_2] \cdots \\
\quad\quad\quad\quad\quad\quad\quad \xrightarrow{e_n} [q_n, \sigma_n] \in EG((\mathcal{E}, \mathcal{Q}, \mathcal{R}, q_I, \mathcal{F})) \\
\quad\quad\quad\quad\quad\quad \wedge\, [q_{I'}, \sigma_I] \xrightarrow{\epsilon} [(Q_1, V_1), \sigma'_1] \xrightarrow{u} \\
\quad\quad\quad\quad\quad\quad\quad [(Q_n, V_n), \sigma'_n] \in EG((\mathcal{E}', \mathcal{Q}', \mathcal{R}', q'_I, \mathcal{F}')) \\
\quad\quad\quad\quad\quad\quad\quad \Longrightarrow q_n \in Q_n
\end{array}
$$

ASSUME: 1. $SM = (\mathcal{E}, \mathcal{Q}, \mathcal{R}, q_I, \mathcal{F})$ for $SM \in \mathbf{SM}$ satisfying **SM1 - SM4** and $\langle\rangle \notin [\![SM]\!]$
2. *ph2* is side effect free for *SM*
3. $SM' = ph2(SM) = (\mathcal{E}', \mathcal{Q}', \mathcal{R}', q'_I, \mathcal{F}')$
4. $n \in \mathbb{N}$
5. $q_1, \in \mathcal{Q} \setminus \mathcal{F}, \ldots, q_n, \in \mathcal{Q} \setminus \mathcal{F}$
6. $\sigma_1, \ldots, \sigma_n \in \widehat{\Sigma}_T$
7. $(Q_1, V_1), (Q_n, V_n) \in \mathcal{Q}'$
8. $\sigma'_1, \sigma'_n \in \widehat{\Sigma}_T$
9. $e_1, \ldots, e_n \in \mathbf{E}$
10. $comp([\![SM]\!], \langle e_1, \ldots, e_n\rangle, u)$ for $u \in \mathbf{E}^*$
11. $[q_I, \sigma_I] \xrightarrow{\epsilon} [q_1, \sigma_1] \xrightarrow{e_1} [q_2, \sigma_2] \cdots \xrightarrow{e_n} [q_n, \sigma_n] \in EG(SM)$

12. $[q'_I, \sigma_I] \xrightarrow{\epsilon} [(Q_1, V_n), \sigma'_1] \xrightarrow{u} [(Q_n, V_n), \sigma'_n] \in EG(SM')$

PROVE: $q_n \in Q_n$

$\langle 1\rangle 1$. CASE: 1.1 $n = 0$

PROOF: Case assumption 1.1 leads to a contradiction. To see this, note that by case assumption 1.1, assumption 11, and definition of $[\![\_]\!]$ (Def. 12) have $\langle\rangle \in [\![SM]\!]$ which contradicts assumption 1.

$\langle 1\rangle 2$. CASE: 1.1 $Ind(SM, SM', n-1)$

$\langle 2\rangle 1$. Choose $u' \in \mathbf{E}^*$ such that $u' \frown \langle e_n\rangle \sqsubseteq u$ and $comp([\![SM]\!], \langle e_1, \ldots, e_{n-1}\rangle, u')$

$\langle 4\rangle 1$. $pr([\![SM]\!], \langle e_1, \ldots, e_{n-1}\rangle) \neq \emptyset$

PROOF: By assumption 10 and definition of pr (Def. 25).

$\langle 4\rangle 2$. Choose $u' \in \mathbf{E}^*$ such that $u' \frown \langle e_n\rangle \sqsubseteq u$ and $\langle e_1, \ldots, e_{n-1}\rangle \lhd u'$ and $\neg(\langle e_1, \ldots, e_n\rangle, u')$

PROOF: By assumption 10 and definition of $comp(\_,\_,\_)$ (Def. 25).

$\langle 4\rangle 3$. $\neg\exists s' \in pr([\![SM]\!], \langle e_1, \ldots, e_{n-1}\rangle) : \langle e_1, \ldots, e_{n-1}\rangle \sqsubset s' \wedge s' \lhd u'$

PROOF: By $\langle 4\rangle 2$.

$\langle 4\rangle 4$. Q.E.D.

PROOF: By $\langle 4\rangle 1$ - $\langle 4\rangle 3$ and definition of $comp(\_,\_,\_)$ (Def. 25).

$\langle 2\rangle 2$. Choose $(Q_{n-2}, V_{n-2}), (Q_{n-1}, V_{n-1}) \in Q'$ and $\sigma'_{n-2}, \sigma'_{n-1} \in \widehat{\Sigma}_{T'}$ such that $[q'_I, \sigma_I] \xrightarrow{\epsilon} [(Q_1, V_1), \sigma'_1] \xrightarrow{u'} [(Q_{n-2}, V_{n-2}), \sigma'_{n-2}] \xrightarrow{\langle e_n\rangle} [(Q_{n-1}, V_{n-1}), \sigma'_{n-1}] \in EG(SM')$

PROOF: By assumption 12, $\langle 2\rangle 1$ and definition of the execution graph for state machines (Def. 11).

$\langle 2\rangle 3$. $q_{n-1} \in Q_{n-2}$

PROOF: By $\langle 2\rangle 1$, $\langle 2\rangle 2$, assumptions 4-12, and induction hypothesis 1.1.

$\langle 2\rangle 4$. Choose

$(e'_n, bx, as) \in \mathbf{Act}$ and

$\phi \in \widehat{\Sigma}$

such that

$q_{n-1} \xrightarrow{(e'_n, bx, as)} q_n \in \mathcal{R}$,

$\mathcal{D}om(\phi) = var(e'_n)$,

$\phi(e'_n) = e_n$, and

$eval(\sigma_{n-1}[\phi](bx)) = \mathtt{t}$

PROOF: By assumption 11 and definition of the execution graph for state machines (Def. 11).

$\langle 2\rangle 5$. Choose

$(e'_n, bx', as') \in \mathbf{Act}$ and

such that

$(Q_{n-2}, V_{n-2}) \xrightarrow{(e'_n, bx', as')} (Q_{n-1}, V_{n-1}) \in \mathcal{R}'$,

$eval(\sigma'_{n-1}[\phi](bx')) = \mathtt{t}$

PROOF: By $\langle 2\rangle 4$ (since $\mathcal{D}om(\phi) = var(e'_n)$ and $\phi(e'_n) = e_n$) assumption 12, $\langle 2\rangle 2$, and definition of the execution graph for state machines (Def. 11).

$\langle 2\rangle 6$. $q_n \in Q_{n-1}$

$\langle 3\rangle 1$. Choose $Ix \in \mathbb{P}(\mathbb{N})$ such that

$(Q_{n-1}, V_{n-1}) = (Q_{n-2} \cup St(Q_{n-2}, e'_n, Ix, V_{n-2}), (V_{n-2} \cup Vi(Q_{n-2}, e'_n, Ix, V_{n-2})) \setminus Vi(St(Q_{n-2}, e'_n, Ix, V_{n-2})))$

where St is the function defined in Def. 26.

PROOF: By $\langle 2\rangle 5$ and definition of $ph2$ (Def. 26).

$\langle 3\rangle 2$. Choose

$q'_1, \ldots, q'_m \in Q_{n-2}$
$p'_1, \ldots, p'_m \in \mathcal{Q}$
$bx_1, \ldots, bx_m \in \mathbf{BExp}$
$as_1, \ldots, as_m \in (\mathbf{Var} \times \mathbf{Exp})^*$
such that
$Vi(Q_{n-2}, e'_n, Ix, V_{n-2}) =$
$set(Ix \circledT ((q'_1, (e'_n, bx_1, as_1), p'_1), \ldots, (q'_m, (e'_n, bx_m, as_m), p'_m))$
where Vi is the function defined in Def. 26.
PROOF: By definition of Vi (Def. 26).

⟨3⟩3. Choose $j \in \mathbb{N}$ such that
$(q'_j, bx_j, as_j) = (q_{n-1}, (e'_n, bx, as), q_n)$
PROOF: By ⟨2⟩3, ⟨2⟩4, ⟨3⟩2 and definition of Vi (Def. 26).

⟨3⟩4. $j \in Ix$
⟨4⟩1. ASSUME: 2.1 $j \notin Ix$
PROVE: False
⟨5⟩1. $eval(\sigma'_{n-2}[\phi](bx)) = \mathtt{f}$
⟨6⟩1. $bx' = conj(Ix \circledT (bx_1, \ldots, bx_m), neg(\mathbb{N} \backslash Ix \circledT (bx_1, \ldots, bx_m))))$
PROOF: By ⟨3⟩1, ⟨3⟩2, and definition of $ph2$ (Def. 26).
⟨6⟩2. $bx_j \in set(\mathbb{N} \setminus Ix \circledT (bx_1, \ldots, bx_m)))$
PROOF: By assumption 2.1 and definition of $\circledT$ (see App. D.1.2).
⟨6⟩3. Q.E.D.
PROOF: By ⟨6⟩1, and ⟨6⟩2, definition of $conj$, $disj$, and neg (see App. D.1.2), bx must evaluate to false (i.e., $eval(\sigma'_{n-2}[\phi](bx)) = \mathtt{f}$) since bx' evaluates to true ($eval(\sigma_{n-1}[\phi](bx)) = \mathtt{t}$) by ⟨2⟩5.
⟨5⟩2. $eval(\sigma_{n-1}[\phi](bx)) = eval(\sigma'_{n-2}[\phi](bx))$
⟨6⟩1. $[q_I, \sigma_I] \xrightarrow{\epsilon} [q_1, \sigma_1] \xrightarrow{e_1} [q_2, \sigma_2] \cdots \xrightarrow{e_{n-1}} [q_{n-1}, \sigma_{n-1}] \in EG(SM)$
PROOF: By assumption 13.
⟨6⟩2. $[q_I, \sigma_I] \xrightarrow{\epsilon} [(Q_1, V_1), \sigma'_1] \xrightarrow{u'} [(Q_{n-2}, V_{n-2}), \sigma'_{n-2}] \in EG(SM')$
PROOF: By assumption 14.
⟨6⟩3. $q_{n-1} \xrightarrow{(e'_n, bx, as)} q_n \in \mathcal{R}$
PROOF: By ⟨2⟩4.
⟨6⟩4. $comp([\![SM]\!], \langle e_1, \ldots, e_{n-1} \rangle, u')$
PROOF: By ⟨2⟩1.
⟨6⟩5. Q.E.D.
PROOF: By ⟨6⟩1 - ⟨6⟩4, assumption 2, and definition of side effect freedom (Def. 25).
⟨5⟩3. Q.E.D.
PROOF: ⟨5⟩1 and ⟨5⟩2 contradict ⟨2⟩4 (since it asserts that $eval(\sigma_{n-1}[\phi](bx)) = \mathtt{t}$)
⟨4⟩2. Q.E.D.
PROOF: By contradiction.

⟨3⟩5. $q_n \in St(Q_{n-2}, e'_n, Ix, V_{n-2})$ where St is the function defined in Def. 26.
PROOF: By ⟨3⟩2 - ⟨3⟩4, and definition of St (Def. 26).

⟨3⟩6. Q.E.D.
PROOF: By ⟨3⟩1 and ⟨3⟩5.

⟨2⟩7. Q.E.D.
PROOF: By ⟨2⟩6 since by definition of $ph2$ (Def. 26), we know that $(Q, V) \xrightarrow{e} (Q', V') \in EG(SM')$ implies $Q \subseteq Q'$.

$\langle 1\rangle 3$. Q.E.D.
PROOF: By $\langle 1\rangle 1$, $\langle 1\rangle 2$, and induction over n.

Corollary 1.2 The proof is based on induction. We make use of the predicate $Ind \in \mathbf{SM} \times \mathbf{SM} \times \mathbf{E}^* \to \mathbb{B}$ which high-lights the induction. Let $SM \stackrel{\text{def}}{=} (\mathcal{E}, \mathcal{Q}, \mathcal{R}, q_I, \mathcal{F})$, and $SM' \stackrel{\text{def}}{=} (\mathcal{E}', \mathcal{Q}', \mathcal{R}', q'_I, \mathcal{F}')$, then Ind is defined by

$$
\begin{array}{l}
Ind(SM, SM', s) \stackrel{\text{def}}{=} \\
\quad \forall s' \in \mathbf{E}^* : \\
\quad\quad \forall (Q_n, V_n) \in \mathcal{Q}' : \forall q_n \in \mathcal{Q} \cap Q_n : \\
\quad\quad\quad \forall \sigma_I \in \widehat{\Sigma}_T : \forall \sigma'_n \in \widehat{\Sigma}_T : \\
\quad\quad\quad\quad \wedge\, s' \sqsubseteq s \\
\quad\quad\quad\quad \wedge\, [q'_I, \sigma_I] \xrightarrow{\langle\epsilon\rangle \frown s'} [(Q_n, V_n), \sigma'_n] \in EG(SM') \\
\quad\quad\quad\quad\quad \Longrightarrow \exists u \in \mathbf{E}^* : \\
\quad\quad\quad\quad\quad\quad\quad \exists \sigma_n \in \widehat{\Sigma}_T : \\
\quad\quad\quad\quad\quad\quad\quad\quad \wedge\, [q_I, \sigma_I] \xrightarrow{\langle\epsilon\rangle \frown u} [q_n, \sigma_n] \in EG(SM) \\
\quad\quad\quad\quad\quad\quad\quad\quad \wedge\, comp(\llbracket SM \rrbracket, u, s')
\end{array}
$$

ASSUME: 1. $SM = (\mathcal{E}, \mathcal{Q}, \mathcal{R}, q_I, \mathcal{F})$ for $SM \in \mathbf{SM}$ satisfying **SM1** - **SM4** and $\langle\rangle \notin \llbracket SM \rrbracket$
2. $ph2$ is side effect free for SM
3. $SM' = ph2(SM) = (\mathcal{E}', \mathcal{Q}', \mathcal{R}', q_I, \mathcal{F}')$
4. $s \in \mathbf{E}^*$
5. $(Q_n, V_n) \in \mathcal{Q}'$
6. $\sigma_I, \sigma'_n \in \widehat{\Sigma}_T$
7. $q_n \in Q_n \cap \mathcal{Q}$
8. $[q'_I, \sigma_I] \xrightarrow{\langle\epsilon\rangle \frown s} [(Q_n, V_n), \sigma'_n] \in EG(SM')$

PROVE: $\exists u \in \mathbf{E}^* :$
$\exists \sigma_n \in \widehat{\Sigma}_T :$
$\wedge\, [q_I, \sigma_I] \xrightarrow{\langle\epsilon\rangle \frown u} [q_n, \sigma_n] \in EG(SM)$
$\wedge\, comp(\llbracket SM \rrbracket, u, s)$

$\langle 1\rangle 1$. CASE: 1.1 $s = \langle\rangle$

$\langle 2\rangle 1$. Choose $q_I \xrightarrow{(\epsilon,\epsilon,sa)} q' \in \mathcal{R}$
PROOF: By assumption 1 since SM satisfies syntax constraint **SM2**.

$\langle 2\rangle 2$. $q'_I \xrightarrow{(\epsilon,\epsilon,sa)} \{q'\} \in \mathcal{R}'$
PROOF: By $\langle 2\rangle 1$, assumption 3, and definition of $ph2$ (Def. 26).

$\langle 2\rangle 3$. $[q'_I, \sigma_I] \xrightarrow{\epsilon} [(Q_n, V_n), \sigma'_n] \in EG(SM')$
PROOF: By assumption 8, case assumption 1.1, and assumptions 1 (since SM is assumed to satisfy syntax constraint **SM3**) and 3 and definition of $ph2$ (Def. 26).

$\langle 2\rangle 4$. $q_n = q'$
PROOF: By assumption 7, $\langle 2\rangle 2$, and $\langle 2\rangle 3$ and definition of $ph2$ (Def. 26) (since $Q_n = \{q'\}$).

$\langle 2\rangle 5$. Choose $\sigma_n \in \widehat{\Sigma}_T$ such that $[q_I, \sigma'_I] \xrightarrow{\epsilon} [q_n, \sigma_n] \in EG(SM)$
PROOF: By $\langle 2\rangle 1$, $\langle 2\rangle 4$ and definition of the execution graph for state machines (Def. 11).

$\langle 2\rangle 6$. Q.E.D.

PROOF: By $\langle 2\rangle 5$.

$\langle 1\rangle 2$. CASE: 1.1 $s = s' \frown \langle e\rangle$
1.2 $Ind(SM, SM', s')$

$\langle 2\rangle 1$. Choose $(Q_{n-1}, V_{n-1}) \in \mathcal{Q}'$ and $\sigma'_{n-1} \in \widehat{\Sigma}_T$ such that $[q'_I, \sigma'_I] \xrightarrow{\langle \epsilon\rangle \frown s'} [(Q_{n-1}, V_{n-1}), \sigma'_{n-1}] \xrightarrow{e} [(Q_n, V_n), \sigma'_n] \in EG(SM')$
PROOF: By assumption 8, assumption 1.1, and definition of the execution graph for state machines (Def. 11).

$\langle 2\rangle 2$. Choose
$(e', bx', as') \in \mathbf{Act}$ and
$\phi \in \widehat{\Sigma}$
such that
$(Q_{n-1}, V_{n-1}) \xrightarrow{(e', bx', as')} (Q_n, V_n) \in \mathcal{R}'$
$\mathcal{D}om(\phi) = var(e')$,
$\phi(e') = e$, and
$eval(\sigma'_{n-1}[\phi](bx')) = \mathrm{t}$
PROOF: By $\langle 2\rangle 1$ and definition of the execution graph for state machines (Def. 11).

$\langle 2\rangle 3$. CASE: 2.1 $\exists Ix \in \mathbb{P}(\mathbb{N}) : q_n \in St(Q_{n-1}, e', Ix, V_{n-1})$

$\langle 3\rangle 1$. Choose $Ix \in \mathbb{P}(\mathbb{N})$ and $q_{n-1} \xrightarrow{(e', bx, sa)} q_n \in \mathcal{R}$ such that $q_{n-1} \xrightarrow{(e', bx, sa)} q_n \in Vi(Q_{n-1}, e', Ix, V_{n-1})$
PROOF: By case assumption 2.1 and definition of Vi (Def. 26).

$\langle 3\rangle 2$. Choose
$\sigma_{n-1} \in \widehat{\Sigma}_T$ and,
$u' \in \mathbf{E}^*$
such that
$(A)\, [q_I, \sigma_I] \xrightarrow{\langle \epsilon\rangle \frown u'} [q_{n-1}, \sigma_{n-1}] \in EG(SM)$,
$(B)\, comp([\![\, SM \,]\!], u', s')$
PROOF: By assumptions 1.1, 1.2, and $\langle 3\rangle 1$.

$\langle 3\rangle 3$. Choose $\sigma_n \in \widehat{\Sigma}_T$ such that
$[q_{n-1}, \sigma_{n-1}] \xrightarrow{e} [q_n, \sigma_n] \in EG(SM)$
PROOF: By $\langle 2\rangle 1$, $\langle 2\rangle 2$, $\langle 3\rangle 2$, assumption 2, and assumption 3.

$\langle 3\rangle 4$. $comp([\![\, SM \,]\!], u' \frown \langle e\rangle, s' \frown \langle e\rangle)$

$\langle 4\rangle 1$. $pr([\![\, SM \,]\!], u' \frown \langle e\rangle) \neq \emptyset$
PROOF:By $\langle 3\rangle 2$ and definition of pr (Def. 25).

$\langle 4\rangle 2$. $u' \frown \langle e\rangle \lhd s' \frown \langle e\rangle$
PROOF:By $\langle 3\rangle 2$ and definition of $comp(\_, \_, \_)$ (Def. 25).

$\langle 4\rangle 3$. $\neg(\exists s'' \in pr([\![\, SM \,]\!], u' \frown \langle e\rangle) : u' \frown \langle e\rangle \sqsubset s'' \wedge s'' \lhd u' \frown \langle e\rangle)$
PROOF:If $\langle 4\rangle 3$ does not holds, then we can choose $e' \in \mathbf{E}$ such that $u' \frown \langle e, e'\rangle \lhd s' \frown \langle e\rangle$, but by definition of $\lhd$, this is impossible.

$\langle 4\rangle 4$. Q.E.D.
PROOF:By $\langle 4\rangle 1$ - $\langle 4\rangle 3$ and definition of $comp(\_, \_, \_)$ (Def. 25).

$\langle 3\rangle 5$. Q.E.D.
PROOF: By $\langle 3\rangle 2$ - $\langle 3\rangle 4$.

$\langle 2\rangle 4$. CASE: 2.1 $\neg(\exists Ix \in \mathbb{P}(\mathbb{N}) : q_n \in St(Q_{n-1}, e', Ix, V_{n-1}))$

$\langle 3\rangle 1$. Choose
$q_{n-1} \in Q_{n-1} \cap \mathcal{Q}$,
$\sigma_{n-1} \in \widehat{\Sigma}_T$ and,
$u' \in \mathbf{E}^*$

such than
$(A)\,[q_I, \sigma_I] \xrightarrow{\langle \epsilon \rangle \frown u'} [q_{n-1}, \sigma_{n-1}] \in EG(SM)$,
$(B)\,comp(\llbracket SM \rrbracket, u', s')$,
$(C)\,q_{n-1} = q_n$
PROOF: (A) and (B) hold by case assumption 1.2. (C) holds by case assumption 7, assumption 2.1, and definition of $ph2$ (Def. 26) since we have $Q_n = Q_{n-1}$.

$\langle 3 \rangle 2$. $[q_{n-1}, \sigma_{n-1}] \xrightarrow{\langle\rangle} [q_n, \sigma_{n-1}] \in EG(SM)$
PROOF: By $\langle 3 \rangle 1$ and definition of the execution graph for state machines (Def. 11).

$\langle 3 \rangle 3$. $comp(\llbracket SM \rrbracket, u', s' \frown \langle e \rangle)$
$\langle 4 \rangle 1$. $pr(\llbracket SM \rrbracket, u') \neq \emptyset$
PROOF:By $\langle 3 \rangle 1$ and definition of pr (Def. 25).
$\langle 4 \rangle 2$. $u' \lhd s' \frown \langle e \rangle$
PROOF:By $\langle 3 \rangle 1$ and definition of $comp(\_,\_,\_)$ (Def. 25).
$\langle 4 \rangle 3$. $\neg(\exists s'' \in pr(\llbracket SM \rrbracket, u') : u' \frown \langle e \rangle \sqsubset s'' \land s'' \lhd u' \frown \langle e \rangle)$
PROOF:If $\langle 4 \rangle 3$ holds, then there must be trace $t \in \llbracket SM \rrbracket$ such that $u' \frown \langle e \rangle \sqsubseteq t$. However, this contradicts case assumption 2.1.
$\langle 4 \rangle 4$. Q.E.D.
PROOF:By $\langle 4 \rangle 1$ - $\langle 4 \rangle 3$ and definition of $comp(\_,\_,\_)$ (Def. 25).
$\langle 3 \rangle 4$. Q.E.D.
PROOF: By $\langle 3 \rangle 1$ - $\langle 3 \rangle 3$.
$\langle 2 \rangle 5$. Q.E.D.
PROOF: By $\langle 2 \rangle 1$ - $\langle 2 \rangle 4$.
$\langle 1 \rangle 3$. Q.E.D.
PROOF: By $\langle 1 \rangle 1$ and $\langle 1 \rangle 2$ and induction.

Theorem 1 Let d be a well formed single lifeline sequence diagram such that $ph2$ is side effect free for $ph1(d)$, then the transformation $d2p(d)$ is adherence preserving, i.e.,

$$d \rightharpoondown_{da} \Phi \Leftrightarrow d2p(d) \rightharpoondown_{sa} \Phi \qquad \text{for all systems } \Phi$$

Proof of Theorem 1
ASSUME: 1. $d \in \mathbf{D}^l$ for some $l \in \mathbf{L}$
2. d satisfies conditions **SD1** - **SD10**
3. $ph2$ is side effect free for $ph1(d)$
PROVE: $d \rightharpoondown_{da} \Phi \Leftrightarrow d2p(d) \rightharpoondown_{sa} \Phi$

$\langle 1 \rangle 1$. ASSUME: 1.1 $d \rightharpoondown_{da} \Phi$
PROVE: $d2p(d) \rightharpoondown_{sa} \Phi$
$\langle 2 \rangle 1$. $\Phi|_{\mathbf{E}^l} \subseteq \llbracket d2p(d) \rrbracket$
$\langle 3 \rangle 1$. ASSUME: 2.1 $t \in \Phi|_{\mathbf{E}^l}$
PROVE: $t \in \llbracket d2p(d) \rrbracket$
$\langle 4 \rangle 1$. $\forall s \in \llbracket ph1(d) \rrbracket : \neg(s \lhd t)$
PROOF: By assumptions 1, 2, 1.1, and 2.1, Lemma 1, and definition of $\rightharpoondown_{da}$ (Def 8).
$\langle 4 \rangle 2$. $t \in \llbracket ph2(ph1(d)) \rrbracket$
PROOF: By assumption 2.1, $\langle 4 \rangle 1$, assumptions 1-3, and Corollary 1.
$\langle 4 \rangle 3$. Q.E.D.

PROOF: By ⟨4⟩2 and definition of $d2p$ (Def. 18).

⟨3⟩2. Q.E.D.

PROOF: By ⟨3⟩1 and definition of $\subseteq$.

⟨2⟩2. Q.E.D.

PROOF: By ⟨2⟩1 and definition of $\rightharpoonup_{sa}$ (Def. 13).

⟨1⟩2. ASSUME: 1.1 $d2p(d) \rightharpoonup_{sa} \Phi$

PROVE: $d \rightharpoonup_{da} \Phi$

⟨2⟩1. ASSUME: 2.1 $s \in H_{neg}$ for $[\![d]\!] = (H_{pos}, H_{neg})$

2.2 $t \in \Phi|_{\mathbf{E}^l}$

PROVE: $\neg(s \lhd t)$

⟨3⟩1. $t \in [\![ph2(ph1(d))]\!]$

PROOF: By assumption 1.1, assumption 2.2, and definition of $\rightharpoonup_{sa}$ (Def. 13).

⟨3⟩2. $s \in [\![ph1(d)]\!]$

PROOF: Assumptions 1 and 2, assumption 2.1, and Lemma 1.

⟨3⟩3. Q.E.D.

PROOF: By ⟨3⟩1, ⟨3⟩2, assumptions 1- 3, and Corollary 1.

⟨2⟩2. Q.E.D.

PROOF: By ⟨2⟩1 and definition of $\rightharpoonup_{da}$ (Def. 8).

⟨1⟩3. Q.E.D.

PROOF: By ⟨1⟩1 and ⟨1⟩2.

Theorem 2 Let d be a well formed sequence diagram such that $ph2$ is side effect free for $ph1(\pi_l(d))$ for all lifelines l in d, then the transformation $d2pc(d)$ is adherence preserving, i.e.,

$$d \rightharpoonup_{dag} \Phi \Leftrightarrow d2pc(d) \rightharpoonup_{sag} \Phi \qquad \text{for all systems } \Phi$$

Proof of Theorem 2

ASSUME: 1. $d \in \mathbf{D}$ and d satisfies conditions **SD1 - SD10**

2. $ph2$ is side effect free for $ph1(\pi_l(d))$ for all $l \in ll.d$

PROVE: $d \rightharpoonup_{dag} \Phi \Leftrightarrow d2pc(d) \rightharpoonup_{sag} \Phi$

⟨1⟩1. ASSUME: 1.1 $d \rightharpoonup_{dag} \Phi$

PROVE: $d2pc(d) \rightharpoonup_{sag} \Phi$

⟨2⟩1. $d2p(\pi_l(d)) \rightharpoonup_{sa} \Phi$ for all $l \in ll.d$

⟨3⟩1. $\pi_l(d) \rightharpoonup_{da} \Phi$ for all $l \in ll.d$

PROOF: By assumption 1.1 and definition of $\rightharpoonup_{dag}$ (Def. 9)).

⟨3⟩2. Q.E.D.

PROOF: By ⟨3⟩1, assumptions 1 and 2, and Theorem 1.

⟨2⟩2. Q.E.D.

PROOF: By ⟨2⟩1, definition of $\rightharpoonup_{sag}$ (Def. 14)), and definition of $d2pc$ (Def. 28).

⟨1⟩2. ASSUME: 1.1 $d2pc(d) \rightharpoonup_{sag} \Phi$

PROVE: $d \rightharpoonup_{dag} \Phi$

⟨2⟩1. $\pi_l(d) \rightharpoonup_{da} \Phi$ for all $l \in ll.d$

⟨3⟩1. $d2p(\pi_l(d)) \rightharpoonup_{sa} \Phi$ for all $l \in ll.d$

PROOF: By assumption 1.1 and definition of $\rightharpoonup_{sag}$ (Def. 14)), and definition of $d2pc$ (Def. 28).

⟨3⟩2. Q.E.D.

PROOF: By ⟨3⟩1, assumptions 1 and 2, and Theorem 1.

⟨2⟩2. Q.E.D.

PROOF: By ⟨2⟩1 and definition of $\rightarrow_{dag}$ (Def. 9).

⟨1⟩3. Q.E.D.

PROOF: By ⟨1⟩1 and ⟨1⟩2.