Constructing Mid-points for Two-party Asynchronous Protocols

Petar Tsankov, Mohammad Torabi-Dashti, and David Basin

ETH Zürich, 8092 Zürich, Switzerland

Abstract. Communication protocols describe the steps that the communication end-points must take in order to achieve a common goal. In practice, networks often contain mid-points, which can relay, redirect, or filter messages exchanged by the end-points. A mid-point can enforce a communication protocol: it forwards the messages that conform to the protocol, and drops them otherwise. Protocol specifications typically define only the end-points' behavior. Implementing a mid-point that enforces a protocol is nontrivial: the mid-point's behavior depends on the end-point's behavior, and also on the behavior of the communication environment in which the protocol executes.
We present a process algebraic framework that takes as input the formal specifications of the protocol and the environment and outputs a specification for a mid-point that enforces the protocol. We prove that the mid-point specifications synthesized by our framework are correct: only messages that could have resulted from correctly executing end-points are forwarded. As an application, we construct a formal model for the mid-point that enforces the TCP three-way handshake protocol.

Key words: mid-point, specification, synthesis, formal methods, protocol enforcement

1 Introduction

Context. Communication protocols describe the steps that the communication end-points take in order to achieve a common goal, e.g. to exchange data reliably. In practice, the end-points often communicate over *mid-points*, which relay, redirect, or filter the communication. Firewalls are prominent examples of mid-points. They can not only observe the execution of a protocol between end-points, but also enforce that the protocol is correctly executed. Namely, the mid-point forwards the messages that conform to the protocol, and drops them otherwise. The messages that do not conform to the protocol may have been sent by a faulty end-point or by an adversary, may be the result of communication failures, etc. For example, a mid-point (or firewall) that enforces the TCP protocol should drop ack messages from B to A right after A has sent B a syn message. This is because, according to TCP's three-way handshake, B must reply to A's syn either with a syn&ack or with a rst message.

The behavior of a mid-point that enforces a communication protocol depends on the steps that the end-points must take, and also on the *communication environment* where the protocol should be executed. This is intuitively because the mid-point would observe

the actions of the end-points via a "lens", namely the communication channels that connect the end-points to the mid-point. In an asynchronous message-passing environment, for instance, it is possible that an end-point sends message a and then message b, but the mid-point observes the message b before the message a. The mid-point cannot simply dismiss the observed sequence of messages as a violation of the protocol because, depending on the channels' characteristics, the mid-point may observe different events, and events in different orders, compared to the end-points; cf. [2].

Contributions. We present a process algebraic framework[1] for automatically synthesizing formal models for mid-points. The input to the framework is the specification of the end-points of an asynchronous protocol and the characteristics of the channels that connect the end-points to the mid-point. The framework outputs a formal specification for a mid-point that enforces the protocol. Formal specifications for mid-points can in general be used for (model-based) testing, (model-driven) development of mid-points, and formal verification of mid-points. These are all practically important and nontrivial tasks: A case study on three commonly used firewalls (Checkpoint, netfilter/iptables, and ISA Server) shows that different firewall manufacturers implement the mid-point for (enforcing) the TCP protocol differently, and sometimes incorrectly with respect to the TCP specification given in [8] (see [3] for details). A formal specification for TCP mid-points can be used either to avoid or to pinpoint the causes of such discrepancies.

The inputs and the output of our framework are processes specified in the μCRL process algebraic language [13]. The resulting mid-point process can be expanded to a (finite) state machine, if desired. Choosing μCRL for automatically constructing mid-point specifications has two benefits:

1. (Theoretical) The problem of constructing mid-point specifications is reduced to computing parallel compositions in our framework, hence relating the problem to a well-studied body of research. This simplifies the correctness proof for the construction, and also enables us to use bisimulation reductions for minimizing the mid-point processes output by the framework.
2. (Practical) The μCRL process algebra comes with a mature tool support [5, 4, 6]. This allows us to put the proposed framework immediately into practice: the μCRL toolset has been used for the case study reported in this paper.

We have carried out a case study on constructing a formal model for the mid-point that enforces the TCP three-way handshake protocol [9].

Related work. The closest related work is [3], where the authors give an algorithm for constructing mid-points, assuming that the specifications of the end-points are given as finite-state machines. Our framework is more general and more modular than the algorithm of [3]: (1) end-points are defined as finite-state machines in [3] while in our framework μCRL processes with recursive data types allow for a larger class of end-point specifications, and (2) the algorithm of [3] is tailored for a fixed type of channels while any μCRL process can model the channels in our framework. Thus, our algorithm

[1] The framework can be downloaded at `www.infsec.ethz.ch/research/software`.

can be directly applied to settings where different channels have different characteristics.

Bhargavan et al. [2] consider a problem which is related to, but nonetheless different from, the mid-point construction problem. In [2], the end-points are assumed to be connected directly via communication channels, and the authors consider the problem of automatically constructing specifications for *monitors* that observe the communication between the end-points. There is a significant difference between monitors and mid-points as the following simple example shows. Suppose that A and B communicate over asynchronous channels. The mid-point, mediating the communication between A and B, knows that if it has not forwarded a message m from A to B, then B could not have received m. However, the monitor, passively observing the communication between A and B, cannot know this: it could be that m has reached B, but m has not reached the monitor due to the asynchronous nature of communication.

Related areas are firewall testing [15, 7], the extensive literature on test case generation from Mealy machines (e.g. see [19]), and testing TCP end-point automata [18]. In firewall testing a mid-point is tested. The previous works start with the firewall rules, while our focus is on the interactive nature of stateful firewalls. Test case generation from Mealy machines can be applied to the transition systems produced by our framework for testing mid-points. Testing TCP end-point automata is complementary to our work, as we consider constructing mid-point formal specifications that in turn can be used for testing TCP mid-points.

The remainder of this paper is organized as follows. In Section 2 we give a short introduction to the μCRL process algebra. In Section 3 we describe how we model communication protocols and their environments. In Section 4 we discuss the challenges in constructing mid-point specifications. In Section 5 we give formal definitions and in Section 6 we present our process algebraic framework. In Section 7 we present our case study on the TCP three-way handshake protocol and in Section 8 we draw conclusions. We prove the correctness of our method in Appendix A.

2 The μCRL process algebra

For specifying end-points, mid-points, and communication channels, we use the process algebra μCRL [13], which is an extension of the process algebra ACP [1] with abstract data types. Our results however do not depend on this choice in any crucial way, as μCRL is similar to other process calculi such as CSP. In what follows, we provide a brief introduction to μCRL. Its complete syntax and semantics are given in [13].

A μCRL specification consists of data type declarations and process behavior definitions, where processes and actions can be parameterized by data. Data is typed in μCRL and types can be recursive. Each non-empty data type has constructors and possibly non-constructors associated with it. The semantics of non-constructors is given by equations. The presence of a type Bool of Booleans with constants T and F as constructors, and the usual connectives $\wedge$, $\vee$ and $\neg$ as non-constructors, is always assumed.

A process is specified as a guarded recursive equation that is constructed from a finite set of action labels, process algebraic operators and recursion variables; mutual recursion among processes is allowed. The set of action labels is denoted Act. All mem-

bers of Act, except for a designated action label τ for silent steps, may be parameterized with data to construct actions. The process algebraic operators $+$ and $\cdot$ denote nondeterministic choice and sequential composition, respectively: The process $p + q$ can behave either as process p or as process q, and the process $p \cdot q$ behaves as process p and when p terminates (if p ever does), it continues as process q. The constant δ denotes a deadlock process, i.e. one that cannot perform any actions. Recursion variables, which can be parameterized with data, are used in the natural way, e.g. $X = a \cdot X$, with $a \in Act$, describes a process that performs action a and then recurs, thereby performing an infinite number of a actions in sequence. A recursive equation is guarded if all its recursion variables are preceded by an action.

The parallel (asynchronous) composition $p \| q$ interleaves the actions of p and q. Moreover, actions from p and q may synchronize, when this is explicitly allowed by the predefined commutative and associative partial function $| : Act \times Act \rightarrow Act$. Two actions can synchronize only if their data parameters are semantically equal. This implies that synchronization can be used to represent data transfer between processes. Encapsulation $\partial_H(p)$, which renames all occurrences of actions from the set H in p to the deadlock action δ, can be used to force actions to communicate. For example, with $a, b, c \in Act$ and $a|b = c$, the process $(a.\delta) \| (b.\delta)$ behaves as $a.b.\delta + b.a.\delta + c.\delta$. Therefore, $\partial_{\{a,b\}}((a.\delta) \| (b.\delta)) = c.\delta$. The operator ρ is used for renaming: $\rho_{a \rightarrow b}(p)$ simultaneously renames all occurrences of action a to action b in process p.

The summation operator $\sum_{d:D} p(d)$, where d is a free variable in process $p(d)$, provides the possibly infinite choice over a data type D. The conditional construct $p \lhd b \rhd q$, with $b : \mathsf{Bool}$, behaves as p if $b = \mathsf{T}$ and as q if $b = \mathsf{F}$. In particular, the construct $\sum_{d:D} p(d) \lhd f(d) \rhd \delta$, with $f : D \rightarrow \mathsf{Bool}$, chooses values of $d \in D$ such that $f(d)$ is true. The operator $\cdot$ has the strongest precedence, the conditional construct binds stronger than $+$, and $+$ binds stronger than $\sum$.

A μCRL specification describes a labelled transition system (LTS) whose states represent process terms and edges are labelled with actions. The μCRL tool set [5, 4], together with LTSmin [6] and CADP [10] which act as μCRL's back-ends, features visualization, simulation, symbolic reduction, (distributed) state space generation and reduction, model checking, and theorem proving capabilities for μCRL specifications.

3 Communication protocols, environments, and mid-points

Below, we fix a data type Msg for messages. Let the two end-points be indexed by $j \in \{1, 2\}$. Given an end-point j, we refer to its partner (the other) end-point by $\bar{j} = 3 - j$.

Communication protocols. Communication protocols typically describe the steps that the communication end-points take to achieve a common goal, e.g. to exchange data reliably. We therefore define a communication protocol Π as a pair (E^1, E^2), where E^j specifies the protocol for end-point j. Note that we are concerned with two-party communication protocols, as opposed to multi-party protocols. The specifications E^j are subject to a number of restrictions defined below. We define two *communication actions* for each end-point:

$$\mathsf{snd} : \{1, 2\} \times Msg$$
$$\mathsf{rcv} : \{1, 2\} \times Msg$$

Intuitively, $\mathsf{snd}(j, m)$ denotes the event of message m being sent to E^j (via the communication environment, as defined below), and $\mathsf{rcv}(j, m)$ denotes the event of message m with destination E^j being received. We assume that all non-silent actions appearing in E^j are either of the form $\mathsf{snd}(\bar{j}, m)$ or $\mathsf{rcv}(j, m)$, for $j \in \{1, 2\}$ and some $m \in Msg$. All internal actions of E^j are therefore modeled by the silent action τ.

Communication environments. Communication protocols are executed in communication environments. A communication environment is a set of channels $\{C^1, \cdots, C^n\}$, with $n > 0$. A channel's behavior can be formally specified as a μCRL process. Therefore, a communication environment Env is defined as a tuple $(C^1, \cdots, C^n)$, where C^i is the specification of channel i for $1 \leq i \leq n$ (see § 4.1 for examples). The specifications C^i are subject to a number of restrictions defined below. We define two *channel actions* for each channel:

$$\begin{aligned} \mathsf{in} &: \{1, \cdots, n\} \times Msg \\ \mathsf{out} &: \{1, \cdots, n\} \times Msg \end{aligned}$$

Intuitively, $\mathsf{in}(i, m)$ with $1 \leq i \leq n$ and $m \in Msg$ denotes the event of message m being sent to channel i, and $\mathsf{out}(i, m)$ denotes the event of message m being received from channel i. We assume that all non-silent actions appearing in C^i are either of the form $\mathsf{in}(i, m)$ or $\mathsf{out}(i, m)$, for some $m \in Msg$. Any other action of channel i (e.g. dropping or duplicating messages) is therefore modeled as a silent step.

Mid-points. We assume that the mid-point is placed in the communication environment such that all the communication between the end-points passes through the mid-point. See Figure 1.

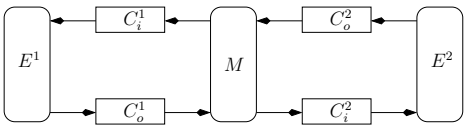

Fig. 1. The general setting: E^1 and E^2 are the end-points and M is the mid-point.

The communication protocol $\Pi = (E^1, E^2)$ is executed in environment Env by placing the channels C_i^j, C_o^j between E^j and M, as shown in Figure 1. We model the communication environment Env as a quadruple $(C_i^1, C_o^1, C_i^2, C_o^2)$. The subscript i denotes "input" and the subscript o denotes "output". We remark that each of the channels C_i^j, C_o^j may in reality consist of several channels linked together. In our model, say, C_o^1 is therefore the specification of a channel that simulates the behavior of all the channels that are used along the communication path that connects E^1 to M.

Note that the mid-point is assumed to be able to distinguish between messages arriving from different channels. In practice, the modeled environment is an IP network and the mid-point is placed such that it interconnects the networks of E^1 and E^2. The

mid-point must be the only entity connecting the two networks to ensure that it can observe all messages exchanged by the end-points. Each network is connected on a different port, hence our assumption is reasonable.

Protocol specifications are usually informal. We however assume that a formal specification for the end-points E^1 and E^2 is available. The characteristics of the communication channels C_i^j, C_o^j, with $j \in \{1, 2\}$, are also assumed to be formally specified. In § 4.1, we give formal specifications for a number of common channel types, such as lossy channels and reliable asynchronous channels. Our goal is to automatically construct a formal specification for the mid-point M that enforces the protocol, given formal specifications for E^j, C_i^j, C_o^j, with $j \in \{1, 2\}$. The notion of *enforcement* is formally defined in § 5.

4 Challenges

In this section, we describe the main aspects that should be considered when constructing formal models for mid-points: *channel fidelity* and *non-determinism*.

4.1 Channels fidelity

Channels fidelity refers to the fact that the sequence of events executed at the end-point and the sequence of events observed by the mid-point may differ depending on the characteristics of their communication environment. Depending on their properties, channels C_i^j, C_o^j distort the way the mid-point views the actions of E^j.

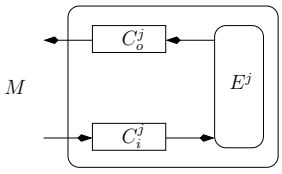

Fig. 2. Mid-point's view of end-point E^j

The mid-point views the actions of E^j via the "lenses" C_i^j and C_o^j; see Figure 2. We illustrate this with an example. Assume that the specification of the end-point E^1 is $E^1 = \mathsf{rcv}(1, x) \cdot \mathsf{snd}(2, y) \cdot \delta + \mathsf{rcv}(1, x) \cdot \delta$. That is, E^1 receives message x and then sends message y, or it receives message x and then stops. Furthermore, assume that the channel C_i^1 is reliable, while the channel C_o^1 is lossy (i.e. it can lose messages). Assume also that the mid-point M sends x to C_i^1. As long as M does not receive message y on C_o^1, it does not know whether E^1 executes $\mathsf{rcv}(1, x) \cdot \mathsf{snd}(2, y) \cdot \delta$ or $\mathsf{rcv}(1, x) \cdot \delta$. This is because message y can be lost by C_o^1 and therefore these two executions of E^1 are indistinguishable to the mid-point.

We remark that given formal specifications of the end-point E^j and the channels C_i^j and C_o^j, we can compute the behavior of the end-point as seen from the point of view of the mid-point; see § 6. As examples, the behavior of reliable, resilient, and lossy channels are formalized in μCRL below. We assume the data structures $Queue$ and Set are given with their usual operators, which we use to model how channels store the messages passed to them. [2]

Reliable channel. Messages are not lost, duplicated, or reordered in this model. The channel stores messages in a queue. When a message is received, modeled by action

[2] For a formal specification see `www.infsec.ethz.ch/research/software`.

$\mathsf{in}(i,m)$, the reliable channel i inserts the message in the queue. When the queue is not empty, the channel removes the first message from the queue and delivers it via action $\mathsf{out}(i,m)$. Below, we omit the name of the channel i from the action labels in and out.

$$C_{reliable}(Q : Queue) = \sum_{m:Msg} \mathsf{in}(m) \cdot C_{reliable}(enqueue(Q,m)) \\ + \\ \sum_{m:Msg} \mathsf{out}(m) \cdot C_{reliable}(dequeue(Q)) \lhd m = head(Q) \rhd \delta$$

Models of reliable channels are useful, e.g., when the mid-point is co-located at one of the end-points. For such a mid-point, the sequence of observed events matches the sequence of events executed by the end-point.

Resilient channel. Messages are not lost, but they may be duplicated or reordered in transmission. The channel stores received messages in a set. A message may be delivered multiple times after it is inserted in the channel.

$$C_{resilient}(S : Set) = \sum_{m:Msg} \mathsf{in}(m) \cdot C_{resilient}(S \cup \{m\}) \\ + \\ \sum_{m:Msg} \mathsf{out}(m) \cdot C_{resilient}(S) \lhd m \in S \rhd \delta$$

In practice, messages can be sent over different routes due to link failures, traffic load balancing, etc. This leads to messages arriving out of order, or multiple times, at the destination.

Lossy channel. Messages are lost and reordered, but are not duplicated. The channel stores messages in a multiset. When a message is in the multiset, it may be delivered or simply removed from the channel buffer.

$$C_{lossy}(S : Set) = \sum_{m:Msg} \mathsf{in}(m) \cdot C_{lossy}(S \cup \{m\}) \\ + \\ \sum_{m:Msg} \mathsf{out}(m) \cdot C_{lossy}(S \setminus \{m\}) \lhd m \in S \rhd \delta \\ + \\ \sum_{m:Msg} \tau \cdot C_{lossy}(S \setminus \{m\}) \lhd m \in S \rhd \delta$$

In practice, channel have finite buffers; when their buffer is full the channels lose messages. Messages are also dropped in case of link failures.

4.2 Non-determinism

Non-determinism in specifications is generally used to allow different alternative behaviors. The alternative behaviors can model, e.g., under-specification (that is, the implementations can follow one or several of the provided alternatives) and abstraction (for instance, probabilistic choices can be modeled as non-deterministic choices).

Since the specifications of the end-points are given in μCRL in our framework, non-determinism in end-points can be naturally expressed using the choice operator $+$. For instance, consider the end-point specification $E^1 = \mathsf{rcv}(1,x) \cdot (\mathsf{snd}(2,y) + \mathsf{snd}(2,z)) \cdot \delta$. That is, E^1 executes $\mathsf{rcv}(1,x)$ and then non-deterministically executes $\mathsf{snd}(2,y)$ or $\mathsf{snd}(2,z)$. The mid-point needs to consider both the executions $\mathsf{rcv}(1,x) \cdot \mathsf{snd}(2,y)$ and $\mathsf{rcv}(1,x) \cdot \mathsf{snd}(2,z)$ as valid, since they comply with the specification of E^1.

5 Formal definitions

We assume that the protocol specification $\Pi = (E^1, E^2)$, and the communication environment specification $Env = (C_i^1, C_o^1, C_i^2, C_o^2)$ are given in μCRL, and they conform to the restrictions specified in § 3. Our goal here is to define when a mid-point M enforces the protocol described by $\Pi = (E^1, E^2)$, executing in the communication environment described by $Env = (C_i^1, C_o^1, C_i^2, C_o^2)$. We first define how the protocol Π executes in the communication environment Env. We define the set of actions $Act = \{a : \{1,2\} \times Msg \mid a \in \{\mathsf{snd}, \mathsf{rcv}, \mathsf{in}, \mathsf{out}, \alpha, \beta, \mathsf{com}\}\}$ and the synchronization rules $\mathsf{snd}|\mathsf{in} = \mathsf{com}, \mathsf{out}|\mathsf{rcv} = \mathsf{com}, \alpha|\beta = \mathsf{f}$. We define two processes P and Q that describe how E^1 and E^2 execute in the communication environment:

$$P = \tau_{\{\mathsf{com}\}}\partial_{\{Act\setminus\{\alpha,\beta,\mathsf{com}\}\}}(E^1 \| \rho_{\{\mathsf{out}\to\alpha\}} C_o^1 \| \rho_{\{\mathsf{in}\to\beta\}} C_i^1)$$
$$Q = \tau_{\{\mathsf{com}\}}\partial_{\{Act\setminus\{\alpha,\beta,\mathsf{com}\}\}}(E^2 \| \rho_{\{\mathsf{out}\to\alpha\}} C_o^2 \| \rho_{\{\mathsf{in}\to\beta\}} C_i^2)$$

Note that we rename the actions out and in in C_o^j and C_i^j to α and β, respectively, and force communication between α and β actions in order to link each input channel C_i^j to the output channel $C_o^{\bar{j}}$, for $j \in \{1,2\}$; see Figure 3. Finally, we define our reference model R:

$$R = \partial_{\{\alpha,\beta\}}(P \| Q)$$

Intuitively, R describes how the mid-point observes the execution of E^1 and E^2 in the communication environment defined by Env.

We now define how arbitrary end-points, constrained by a mid-point M, execute in the communication environment. We assume the extreme case when the end-points arbitrarily execute snd and rcv actions over the set of messages Msg; we model this as $\perp^j = \sum_{m:Msg}(snd(\bar{j}, m) + \mathsf{rcv}(j, m)) \cdot \perp^j$. Let M be the mid-point process such that M executes only $\mathsf{f}(j, m)$ actions, for $j \in \{1,2\}$ and $m \in Msg$. Action $\mathsf{f}(j, m)$ denotes that the mid-point forwards message m to end-point j. We define processes P' and Q' that describe how the arbitrary end-points execute in the communication environment:

$$P' = \tau_{\{\mathsf{com}\}}\partial_{\{Act\setminus\{\alpha,\beta,\mathsf{com}\}\}}(\perp^1 \| \rho_{\{\mathsf{out}\to\alpha\}} C_o^1 \| \rho_{\{\mathsf{in}\to\beta\}} C_i^1)$$
$$Q' = \tau_{\{\mathsf{com}\}}\partial_{\{Act\setminus\{\alpha,\beta,\mathsf{com}\}\}}(\perp^2 \| \rho_{\{\mathsf{out}\to\alpha\}} C_o^2 \| \rho_{\{\mathsf{in}\to\beta\}} C_i^2)$$

We set the synchronization rules to $\alpha|\mathsf{f} = c_1, \mathsf{f}|\beta = c_2, c_1|\beta = \lambda, \alpha|c_2 = \lambda$, and define our implementation model I:

$$I = \partial_{\{\alpha,\beta,\mathsf{f},c_1,c_2\}}(P' \| M \| Q')$$

Given the synchronization rules, a message delivered by an output channel (action α) is received by an input channel (action β) only after synchronizing with the mid-point (action f).

A symmetric binary relation B over processes is a *bisimulation relation* [17, 16] iff $(P, P') \in B$ implies that for any action a and any message m, $P \overset{a(m)}{\to} P_1 \implies P' \overset{a(m)}{\to} P_1'$ with $(P_1, P_1') \in B$. Two processes P and P' are bisimilar, denoted $P \equiv P'$, iff there is bisimulation relation B such that $(P, P') \in B$. The Bisimilarity of two processes intuitively indicates that the two processes are indistinguishable from an observer's point of view. This is the core of our definition of enforcement.

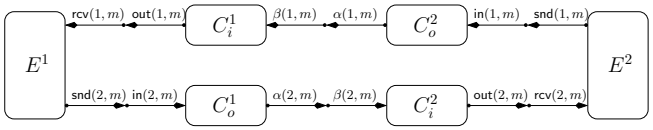

Fig. 3. μCRL action synchronization

Definition 1 (Enforcement). *Mid-point* M enforces *the communication protocol described by* $\Pi = (E^1, E^2)$ *in the communication environment described by* $Env = (C_i^1, C_o^1, C_i^2, C_o^2)$ *iff* $I \equiv \rho_{\mathsf{f}\to\lambda} R$.

Note that we rename action f to λ so that we can compare the implementation and the reference models. The intuition behind this definition is that if the reference and the implementation processes have executed the same protocol steps until some point in time and the reference process can continue the protocol execution with some step s, then the implementation process can also execute s. Conversely, if the implementation process can continue by taking a step s', then the reference process can also take s'.

6 The framework

In this section we present our framework which computes a formal specification of the mid-point. The framework takes as an input the protocol specification $\Pi = (E^1, E^2)$, and the communication environment specification $Env = (C_i^1, C_o^1, C_i^2, C_o^2)$. Π and Env are both given in μCRL and must conform to the restrictions specified in § 3. In § 4.1 we provided several common channel specifications that can be used as an input to our framework. We remark that our framework is modular and each of the four channels can have a different specification. The mid-point specification computed by our framework enforces the communication protocol in the environment defined by Env. A message from E^j to $E^{\bar{j}}$ is allowed, i.e. forwarded to $E^{\bar{j}}$, if it could have been sent by E^j, and rejected otherwise. An incorrect message could result from a faulty end-point or due to communication channel noise.

We distinguish three steps performed in our framework. The first step (*construction*) takes as inputs the specifications of Π and Env given in μCRL and outputs a specification of M in μCRL. Step two (*minimization*) minimizes the state space of M using a branching bisimilarity algorithm. Optionally, the specification of M can be expanded to a finite state machine using a standard ϵ-removal algorithm in the third step. All three steps are automated using the μCRL toolset.

6.1 Mid-point construction

The mid-point construction computes a process that enforces the protocol executed by the two end-points. We define one *enforcement action* for the mid-point:

$$\mathsf{f} : \{1, 2\} \times Msg$$

Intuitively, the action $\mathsf{f}(j, m)$ denotes the event of message m being forwarded to end-point E^j for some message $m \in Msg$ and $j \in \{1, 2\}$. By forwarding a message m to E^j we mean that the mid-point receives a message on $C_o^{\bar{j}}$ and inserts it in channel C_i^j. To determine what messages should be forwarded by the mid-point, we compute the parallel composition of the input μCRL processes E^j, C_i^j, and C_o^j for $j \in \{1, 2\}$. We link channel C_o^1 to C_i^2 and channel C_o^2 to C_i^1, as illustrated in Figure 3. The channels are linked by renaming the action out in channels C_o^1 and C_o^2 to α, renaming the action in in channels C_i^1 and C_i^2 to β, and forcing communication between α and β actions. Given the synchronization between α and β actions, every message delivered by an output channel is inserted into the corresponding input channel.

We synchronize actions that must happen together. Figure 3 illustrates the actions performed by the end-points and the channel processes. We declare the following synchronization rules:

$$\mathsf{snd} \mid \mathsf{in} = \mathsf{com}$$
$$\mathsf{out} \mid \mathsf{rcv} = \mathsf{com}$$
$$\alpha \mid \beta = \mathsf{f}$$

$\mathsf{snd} \mid \mathsf{in}$ enforces that output channel C_o^j receives a message from E^j only when E^j triggers a send message event (action snd); we synchronize these two actions to action com which denotes communication between an end-point and a channel. $\mathsf{out} \mid \mathsf{rcv}$ enforces that end-point E^j receives a message from input channel C_i^j only when C_i^j triggers a deliver message event (action out). We also force communication between α and β actions to enforce that an input channel C_i^j gets a message from $C_o^{\bar{j}}$ only when $C_o^{\bar{j}}$ triggers a deliver message event (action α).

The mid-point process is synthesized by computing the parallel composition of the processes $E^1, E^2, C_i^1, C_o^1, C_i^2, C_o^2$ and then hiding all actions that are unobservable to the mid-point. Intuitively, the parallel composition of the input processes gives us a process that describes all possible protocol executions in the given environment. Hiding all actions unobservable by the mid-point gives us the mid-point's point of view of the protocol executions. The mid-point receives messages from C_o^1 and C_o^2, and sends messages to C_i^1 and C_i^2. Therefore, M observes the α and β events that are synchronized to action f and hence we do not hide action f. As an example, action $\mathsf{f}(1, m)$ indicates that upon receiving message m, the mid-point should forward it to end-point E^1. The mid-point cannot observe communication between an end-point and a channel and hence we hide the action com. We compute the mid-point process as follows:

$$\begin{aligned} M = \partial_{\{\alpha,\beta\}}(\; & \tau_{\{\mathsf{com}\}}\partial_{\{Act\setminus\{\alpha,\beta,\mathsf{com}\}\}}(E^1 \| \rho_{\{\mathsf{out}\to\alpha\}}C_o^1(\emptyset) \| \rho_{\{\mathsf{in}\to\beta\}}C_i^1(\emptyset)) \| \\ & \tau_{\{\mathsf{com}\}}\partial_{\{Act\setminus\{\alpha,\beta,\mathsf{com}\}\}}(E^2 \| \rho_{\{\mathsf{out}\to\alpha\}}C_o^2(\emptyset) \| \rho_{\{\mathsf{in}\to\beta\}}C_i^2(\emptyset))) \end{aligned}$$

Theorem 1. M enforces *the communication protocol Π in the communication environment Env.*

Proof. We show that $I \equiv \rho_{\mathsf{f}\to\lambda}R$, where I is the implementation model and R is the reference model, both defined in § 5. According to Theorem 2 (given in Appendix A) $I \equiv \rho_{\mathsf{f}\to\lambda}M$ if $P \preceq P'$ and $Q \preceq Q'$, where P and Q are the processes that describe how E^1 and E^2 execute in Env, and P' and Q' describe how arbitrary end-points execute in

Env; these processes are all defined in § 5. By construction, the mid-point process M is equivalent to the reference model R, therefore, $I \equiv \rho_{\mathsf{f}\to\lambda} R$ holds if $P \preceq P'$ and $Q \preceq Q'$. Recall that $P = \tau_G \partial_H(E^1 \| \rho_K C_o^1 \| \rho_L C_i^1)$ and $P' = \tau_G \partial_H(\bot^1 \| \rho_K C_o^1 \| \rho_L C_i^1)$ for $G = \{\mathsf{com}\}, H = \{Act \backslash \{\alpha, \beta, \mathsf{com}\}\}, K = \{\mathsf{out} \to \alpha\}$, and $L = \{\mathsf{in} \to \beta\}$. Using the fact that $(P \| X) \preceq (P' \| X)$ if $P \preceq P'$ (proved in Lemma 1 in Appendix A) and that $E^1 \preceq \bot^1$, we have $P \preceq P'$; analogously $Q \preceq Q'$. □

We remark that the computed mid-point is *permissive*: it forwards messages that could have resulted from correctly executing end-points. If the mid-point M receives a message sent by an intruder, and the mid-point cannot distinguish between the intruder's message and the end-point's message, M will forward the message. Constructing a permissive mid-point is the best we can do as we do not want to block legitimate messages and interfere with the protocol execution.

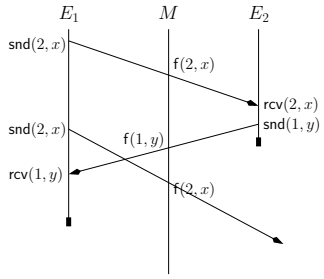

Fig. 4. A permissive mid-point

Note that when the mid-point forwards a message to E^j, there is no guarantee that E^j can receive the message. Using the end-point specifications we can compute a mid-point that blocks messages that cannot be received by the receiving end-point. We illustrate this observation using a simple example. Consider two end-points with specifications $E^1 = \mathsf{snd}(2,x) \cdot (\mathsf{rcv}(1,y) \cdot \delta + E^1)$ and $E^2 = \mathsf{rcv}(2,x) \cdot \mathsf{snd}(1,y) \cdot \delta$. E^1 repeatedly sends x to E^2 until it receives y from E^2, then terminates. E^2 receives x from E^1, sends y, and terminates. An acceptable execution is illustrated in Figure 4. The second x message from E^1 is forwarded to E^2, although E^2 has already terminated after sending message y to E^1.

6.2 State space minimization.

The mid-point process M has a state space associated to it. The computation of M involves hiding all events that the mid-point cannot observe, which appear as τ events in M. Due to the τ events, the mid-point's state space can be large. We reduce the state space by applying branching bisimulation reduction on the mid-point process. The choice of branching bisimulation reduction is motivated by the fact that the notion of enforcement in our framework is based on bisimulation, and branching bisimulation reduction preserves the branching structure of processes while removing the action τ [12]. Our framework computes a process M', which is branching bisimilar to M. The state space of M' is potentially smaller than the state space of M, and M' is branching bisimilar to M (hence Theorem 1 holds for M' as well).

6.3 The mid-point as a state machine.

The μCRL specification of the mid-point is in the form of a linear process equation which can be automatically expanded to a state machine. The state space can be explored by a depth-first search. The generated state space contains τ transitions for all actions unobservable by the mid-point. To eliminate all τ transitions, we apply a standard

ϵ-removal algorithm. The output is a state machine that can also be used as a mid-point specification. For example, Figure 5 (in Section 7) illustrates the mid-point state machine for enforcing the TCP three-way handshake protocol, output by our framework. Clearly, this step cannot be completed if the state space of the mid-point is infinite.

7 TCP case study

An evaluation on three popular firewalls (Checkpoint, netfilter/iptables, and ISA Server) shows that different firewall manufacturers implement mid-points for the TCP protocol differently and incorrectly [3], i.e. they forward messages that should not be sent by the end-points if they implement the protocol correctly. We performed a case study on the TCP protocol to demonstrate how our framework constructs a specification for a mid-point that enforces the protocol. The mid-point specification synthesized by our framework eliminates any ambiguities concerning which packets should be forwarded by the mid-point.

A formal mid-point specification has several applications in practice. It can be used for model-based testing in order to test an implementation for inconsistencies. The tester can use the mid-point specification to generate test cases and run them against the implementation. Additionally, when the mid-point specification is relatively simple, which is the case of the TCP mid-point, a software engineer can use the formal specification to perform code inspection, i.e. systematically examine the source code of the mid-point using the formal specification as a reference. Another application of our framework is model-driven development for mid-points, e.g., using the formal specification to automatically generate the implementation of a stateful TCP firewall.

Firewalls typically distinguish between internal and external networks. The policy for handling TCP connections initiated from the external network are usually handled differently from TCP connections initiated from the internal network. To reflect this, we take the TCP protocol specification [9] and construct two end-point specifications: one that models the *initiator* role and another that models the *responder* role. Below we give the specification of the two roles in μCRL, where we assume that E^1 represents the initiator role and E^2 the responder role.

Initiator end-point. It is the end-point that initiates a TCP connection. Below we give the μCRL specification for the initiator role:

$$
\begin{aligned}
E^1 = {} & \mathsf{snd}(2,\mathsf{syn}) \cdot \mathsf{rcv}(1,\mathsf{synack}) \cdot \mathsf{snd}(2,\mathsf{ack}) \cdot \\
& (\ \mathsf{rcv}(1,\mathsf{fin}) \cdot \mathsf{snd}(2,\mathsf{ack}) \cdot \mathsf{snd}(2,\mathsf{fin}) \cdot \mathsf{rcv}(1,\mathsf{ack}) \\
& \quad + \\
& \quad \mathsf{snd}(2,\mathsf{fin}) \cdot (\ \mathsf{rcv}(1,\mathsf{ack}) \cdot \mathsf{rcv}(1,\mathsf{fin}) \cdot \mathsf{snd}(2,\mathsf{ack}) \\
& \qquad\qquad + \\
& \qquad\qquad \mathsf{rcv}(1,\mathsf{fin}) \cdot \mathsf{snd}(2,\mathsf{ack}) \cdot \mathsf{rcv}(1,\mathsf{ack}))) \cdot \delta
\end{aligned}
$$

Responder end-point. The responder end-point waits for an initiator end-point to open a TCP connection. The actions performed by the responder are symmetric to the initiator actions. We assume that the responder role can initiate a tear-down after it has sent a synack to E^1, i.e. before receiving an ack from E^1.

$$
\begin{aligned}
E^2 &= \mathsf{rcv}(2,\mathsf{syn}) \cdot \mathsf{snd}(1,\mathsf{synack}) \cdot (\mathsf{rcv}(2,\mathsf{ack}) \cdot E^2_T + E^2_T) \\
E^2_T &= \mathsf{rcv}(2,\mathsf{fin}) \cdot \mathsf{snd}(1,\mathsf{ack}) \cdot \mathsf{snd}(1,\mathsf{fin}) \cdot \mathsf{rcv}(2,\mathsf{ack}) \cdot \delta \\
&\quad + \\
&\quad \mathsf{snd}(1,\mathsf{fin}) \cdot (\ \mathsf{rcv}(2,\mathsf{ack}) \cdot \mathsf{rcv}(2,\mathsf{fin}) \cdot \mathsf{snd}(1,\mathsf{ack}) \\
&\qquad\qquad + \\
&\qquad\qquad \mathsf{rcv}(2,\mathsf{fin}) \cdot \mathsf{snd}(1,\mathsf{ack}) \cdot \mathsf{rcv}(2,\mathsf{ack})) \cdot \delta
\end{aligned}
$$

In our case study we assume that the environment can lose and reorder packets, but cannot duplicate messages. For the channel specification we use the μCRL specification of a lossy channel as defined in § 4.1. We compute M using our framework and perform the optional step 3 to expand the state space of M to a state machine, given in Figure 5. The input alphabet to the mid-point automaton is $\mathsf{f}(j,m)$, $j \in \{1,2\}, m \in \{\mathsf{ack},\mathsf{synack},\mathsf{syn},\mathsf{fin}\}$. Action $\mathsf{f}(j,m)$ denotes that M receives a message m from end-point $E^{\bar{j}}$ and forwards it to end-point E^j.

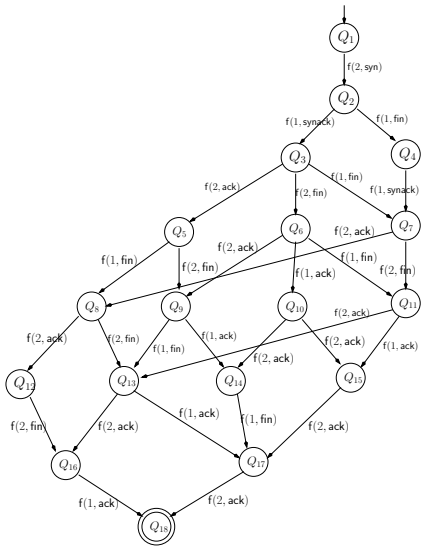

Fig. 5. Mid-point automaton for TCP

Although the end-points have a small number of non-deterministic choices in their specifications, the mid-point process can receive different types of messages in most states, as depicted in Figure 5. This is explained by the effect of the environment, which can reorder and drop messages. For instance, assume M is in state Q_2, i.e. it has forwarded the initial syn message to E^2. E^2 replies to the syn with a synack message and afterwards it can send a fin. The network may reorder the two messages. Therefore, M would forward the fin message if it is received before the synack message.

The TCP specification computed by our framework is equivalent to the TCP automaton presented in [3]. The environment models in both case studies exercise the same properties, hence, the mid-point specification is identical, as expected. In contrast to [3] which fixes the behavior of the environment, we can easily modify the channel specifications and compute a mid-point specification for a different environment. For instance, suppose that the mid-point is co-located at one of the end-points, say E^1. To handle this scenario, we set the specifications of C^1_i and C^1_o to reliable channels and re-run our framework on the new inputs.

8 Conclusions and Future Work

We give a process algebraic approach to automatically synthesizing a formal specification for a mid-point that enforces a communication protocol. Formal mid-point specifications can be used for model-based testing, for model-driven development, and for

formal verification of mid-points. In this paper we have systematically explored the aspects that must be considered when constructing formal models for mid-points. Our approach to handling these challenges can be applied to other related problems; for instance, our framework can be extended to synthesize specifications for passive monitors. Passive monitors are entities that observe messages exchanged over a channel and can be used to check security properties or to guard against network intrusion.

An interesting direction for future work is synthesizing more restrictive mid-points. As we mentioned in § 6, our current framework implementation computes mid-point specifications that are in some cases too permissive. For instance, forwarding a message to an end-point does not guarantee that the receiving end-point can actually receive the message. This may happen, e.g., when an end-point repeatedly re-transmits a message until receiving an acknowledgment from the other end-point or when a channel can duplicate messages. We can modify our framework to compute a mid-point process that forwards a message only if it could have been sent by the source end-point and it can be received by the destination end-point. We remark that such a mid-point achieves more than enforcing the protocol and can be seen as an additional optimization, e.g. to reduce network traffic.

Acknowledgments The work has been supported by the EU FP7 projects SPACIOS (no. 257876).

References

1. J. Bergstra and J. Klop. Algebra of communicating processes with abstraction. *Theor. Comput. Sci.*, 37:77–121, 1985.
2. K. Bhargavan, S. Chandra, P. McCann, and C. Gunter. What packets may come: Automata for network monitoring. In *POPL*, pages 206–219. ACM, 2001.
3. D. Bidder-Senn, D. Basin, and G. Caronni. Midpoints versus endpoints: From protocols to firewalls. In *ACNS*, volume 4521 of *LNCS*, pages 46–64. Springer, 2007.
4. S. Blom, J. Calamé, B. Lisser, S. Orzan, J. Pang, J. van de Pol, M. Torabi Dashti, and A. Wijs. Distributed analysis with μCRL: A compendium of case studies. In *TACAS '07*, volume 4424 of *LNCS*, pages 683–689. Springer, 2007.
5. S. Blom, W. Fokkink, J. Groote, I. van Langevelde, B. Lisser, and J. van de Pol. μCRL: A toolset for analysing algebraic specifications. In *CAV '01*, volume 2102 of *LNCS*, pages 250–254, 2001.
6. S. Blom, J. van de Pol, and M. Weber. Ltsmin: Distributed and symbolic reachability. In T. Touili, B. Cook, and P. Jackson, editors, *CAV*, volume 6174 of *Lecture Notes in Computer Science*, pages 354–359. Springer, 2010.
7. A. Brucker, L. Brügger, P. Kearney, and B. Wolff. Verified firewall policy transformations for test case generation. In *ICST*, pages 345–354. IEEE Computer Society, 2010.
8. Achim D. Brucker, Lukas Brgger, and Burkhart Wolff. Model-based firewall conformance testing. In *In 8th International Workshop on Formal Approaches to Testing of Software, Tokyo,Japan*, pages 103–118, 2008.
9. J. Postel (editor). Transmission control protocol, 1981.
10. J. Fernandez, H. Garavel, A. Kerbrat, L. Mounier, R. Mateescu, and M. Sighireanu. CADP - a protocol validation and verification toolbox. In *CAV '96*, volume 1102 of *LNCS*, pages 437 – 440. Springer, 1996.

11. W. Fokkink. *Modelling Distributed Systems*. Texts in Theoretical Computer Science. Springer, 2007.
12. R. van Glabbeek. The linear time - branching time spectrum II. In *CONCUR '93*, volume 715 of *LNCS*, pages 66–81. Springer, 1993.
13. J. Groote and A. Ponse. The syntax and semantics of μCRL. In *Algebra of Communicating Processes '94*, Workshops in Computing Series, pages 26–62. Springer, 1995. Also as technical report CS-R9076, CWI, Amsterdam, The Netherlands, Dec. 1990.
14. J. Groote and M. Reniers. Algebraic process verification. In J. Bergstra, A. Ponse, and S. Smolka, editors, *Handbook of Process Algebra*, pages 1151–1208. Elsevier, 2001.
15. A. Mayer, A. Wool, and E. Ziskind. Offline firewall analysis. *Int. J. Inf. Sec.*, 5(3):125–144, 2006.
16. Robin Milner. *Communication and concurrency*. PHI Series in computer science. Prentice Hall, 1989.
17. David Park. Concurrency and automata on infinite sequences. In Peter Deussen, editor, *Theoretical Computer Science*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer, 1981.
18. V. Paxson. Automated packet trace analysis of TCP implementations. In *SIGCOMM*, pages 167–179, 1997.
19. M. Utting and B. Legeard. *Practical Model-Based Testing: A Tools Approach*. Morgan-Kaufmann, 2007.

A Proof of correctness

We start with a definition: A binary relation S over processes is a *simulation* relation iff $(P, P') \in S$ implies that $P \stackrel{a(m)}{\rightarrow} P_1 \implies P' \stackrel{a(m)}{\rightarrow} P'_1$, with $(P_1, P'_1) \in S$, for all actions a and messages m. Process P simulates process P', denoted $P' \preceq P$, iff there is a simulation relation S such that $(P, P') \in S$.

Below, we fix

- The reference model: $M = \partial_{\{\alpha,\beta\}}(P \| Q)$ and $\alpha|\beta = f$.
- The implementation model: $I = \partial_{\{\alpha,\beta,\mathsf{f},c_1,c_2\}}(P' \| M \| Q')$ and the synchronization rules $\alpha|f = c_1, f|\beta = c_2, c_1|\beta = \lambda, \alpha|c_2 = \lambda$.

Theorem 2. *$I \equiv \rho_{f\rightarrow\lambda} M$ if $P \preceq P'$ and $Q \preceq Q'$.*

Proof. We define the relation B as $(S, S') \in B$ iff

$$S = \rho_{f\rightarrow\lambda}\partial_{\{\alpha,\beta\}}(P \| Q)$$

and $S' = \partial_{\{\alpha,\beta,\mathsf{f},c_1,c_2\}}(P' \| M \| Q')$ for all processes P, P', Q, Q' with $P \preceq P'$ and $Q \preceq Q'$. Below, we show that B is indeed a bisimulation relation. In the following we refer to the assumption $P \preceq P'$ and $Q \preceq Q'$ as the *simulation assumption*. We split the proof into two parts:

- Assume $S \stackrel{\lambda}{\rightarrow} S_1$. We claim $S' \stackrel{\lambda}{\rightarrow} S'_1$ and $(S_1, S'_1) \in B$. Notice that in order for S to perform λ, $\partial_{\{\alpha,\beta\}}(P \| Q)$ must execute f, and in turn the processes P and Q must execute α and β respectively (the symmetric case is trivial; hence omitted

here). Let $P \xrightarrow{\alpha} P_1$ and $Q \xrightarrow{\beta} Q_1$. Due to the simulation assumption, $P' \xrightarrow{\alpha} P'_1$ and $Q' \xrightarrow{\beta} Q'_1$ and $P_1 \preceq P'_1$ with $Q_1 \preceq Q'_1$. That is,

$$S' = \partial_{\{\alpha,\beta,\mathsf{f},c_1,c_2\}}(\alpha \cdot P'_1 \| f \cdot \delta_{\alpha,\beta}(P_1 \| Q_1) \| \beta \cdot Q'_1)$$

Given the aforementioned synchronization rules, we have $S' \xrightarrow{\lambda} S'_1$ where $S'_1 = \partial_{\{\alpha,\beta,\mathsf{f},c_1,c_2\}}(P'_1 \| \delta_{\alpha,\beta}(P_1 \| Q_1) \| Q'_1)$. It is immediate that $(S_1, S'_1) \in B$.

– Assume $S' \xrightarrow{\lambda} S'_1$, with $S'_1 = \partial_{\{\alpha,\beta,\mathsf{f},c_1,c_2\}}(P'_1 \| \delta_{\alpha,\beta}(P_1 \| Q_1) \| Q'_1)$ for some P_1, Q_1, P'_1 and Q'_1. We claim $S \xrightarrow{\lambda} S_1$ and $(S_1, S'_1) \in B$. Notice that in order for S' to perform λ, the following two conditions must be satisfied:
  - The process $\partial_{\{\alpha,\beta\}}(P \| Q)$ must execute f. This implies that $P \xrightarrow{\alpha} P_1$ and $Q \xrightarrow{\beta} Q_1$ (the symmetric case is omitted here). Then it is immediate that $S \xrightarrow{\lambda} \partial_{\{\alpha,\beta\}}(P_1 \| Q_1)$.
  - Moreover, $P' \xrightarrow{\alpha} P'_1$ and $Q' \xrightarrow{\beta} Q'_1$ (the symmetric case is omitted). Due to the simulation assumption, $P_1 \preceq P'_1$ and $Q_1 \preceq Q'_1$. Now it is immediate that $(\partial_{\{\alpha,\beta\}}(P_1 \| Q_1), S'_1) \in B$.

  These two points prove our claim.

This completes the proof. □

Lemma 1. *$(P \| X) \preceq (P' \| X)$ for all X, if $P \preceq P'$.*

Proof. Let $P \preceq P'$ for some processes P and P'. We define the binary relation S over processes as: $((Q \| Z), (Q' \| Z)) \in S$ for all $Q \preceq Q'$ and any Z. Obviously we have $((P \| X), (P' \| X)) \in S$. Below, we show that S is indeed a simulation relation.

Suppose that $(P \| X) \xrightarrow{a} Y$. It must be that either P or X executed a, or that P and X executed some b and c, respectively, and $b|c = a$. We look at these three cases:

– $(P \| X) \xrightarrow{a} (P_1 \| X)$. This implies that $P \xrightarrow{a} P_1$. Given that $P \preceq P'$ it follows that $P' \xrightarrow{a} P'_1$ and $P_1 \preceq P'_1$. It is immediate that $(P' \| X) \xrightarrow{a} (P'_1 \| X)$. Clearly, $(P_1 \| X, P'_1 \| X) \in S$.
– $(P \| X) \xrightarrow{a} (P \| X_1)$. This implies that $X \xrightarrow{a} X_1$. Then, $(P' \| X) \xrightarrow{a} (P' \| X_1)$, and hence $(P \| X_1, P' \| X_1) \in S$.
– $(P \| X) \xrightarrow{a} (P_1 \| X_1)$. This implies that $P \xrightarrow{b} P_1$ and $X \xrightarrow{c} X_1$, for some $b, c \in Act$ and $b|c = a$. Given that $P \preceq P'$, it follows that $P' \xrightarrow{b} P'_1$ and $P_1 \preceq P'_1$. It is immediate that $(P' \| X) \xrightarrow{a} (P'_1 \| X_1)$. Hence $(P_1 \| X_1, P'_1 \| X_1) \in S$.

This completes the proof. □