Preservation of Policy Adherence under Refinement

Bjørnar Solhaug[1,2] and Ketil Stølen[2,3]

[1]Dep. of Information Science and Media Studies, University of Bergen
[2]SINTEF ICT [3]Dep. of Informatics, University of Oslo

{bjornar.solhaug,ketil.stolen}@sintef.no

Abstract

Policy-based management is an approach to the management of systems with respect to issues such as security, access control and trust by the enforcement of policy rules. This paper addresses the problem of integrating the requirements imposed by a policy with the system development process. In order to take a policy specification into account in the development of a system specification, the notion of policy adherence is formalized as a relation between policy specifications and system specifications. Adherence of a system specification to a policy specification means that the former satisfies the latter. The integrated development process is supported by refinement, where both the policy specification and the system specification may be developed under any number of refinement steps. This paper characterizes the conditions under which adherence is preserved under refinement and identifies development rules that guarantee adherence preservation. By results of transitivity and compositionality the integrated development process and the analysis tasks can be conducted in a stepwise and modular way, thereby facilitating development.

Key words: Policy adherence, policy refinement, adherence preservation, UML sequence diagrams.

Contents

1 Introduction

Policy-based management of information systems is an approach to administer and control distributed systems with respect to issues such as security, access control, service level and trust by means of the enforcement of policy rules. A policy is commonly defined as a set of rules governing the choices in the behavior of a system [33], and a key feature of policies is that they "define choices in behavior in terms of the conditions under which predefined operations or actions can be invoked rather than changing the functionality of the actual operations themselves" [34]. By separating the policy from the system implementation, the behavior of the system can then be modified in order to meet new or changing requirements in a flexible and dynamic manner by modifying the policy only.

Several frameworks for the specification, development, analysis and enforcement of policies have been proposed [5, 34], but although recognized as an important research issue from the very initial research on policy-based management [23], policy refinement still remains poorly explored in the literature [2, 26]. Policy refinement is in [23] referred to as the process of transforming a high-level, abstract policy specification into a low-level, concrete one. At the initial, abstract level, policies may be derived from business goals, service level agreements, risk analyses, security requirements, etc., and policy refinement should ensure that the enforcement of the final, concrete policy implies the enforcement of the initial, abstract one.

In [35] we presented Deontic STAIRS, a policy specification language based on the STAIRS [12, 30] approach to formal system development with UML 2.1 sequence diagrams [24]. Deontic STAIRS facilitates abstraction, and is supported by a formal notion of policy refinement precisely capturing what it means that a concrete policy specification correctly represents an abstract policy specification. Abstraction involves the perspective or purpose of the viewer, and different purposes result in different abstractions [28]. During the initial activity of policy capturing, details about system entities, architecture and functionality that are irrelevant or unimportant from a given viewpoint can therefore be ignored. Abstraction is desirable also because detection and correction of errors, as well as policy analysis, are cheaper and easier at an abstract and high level [37].

In addition to refinement and abstraction, decomposition has been recognized as one of the most important features of system design and development [37]. Decomposition allows individual parts of a specification to be developed separately, thus breaking the analysis and development problems down into manageable pieces. In [35] we showed that the proposed policy refinement relation is transitive and compositional. Transitivity means that the result of any number of refinement steps is a valid refinement of the initial policy specification. Compositionality means that a policy specification can be refined by refining individual parts separately. We furthermore showed that the policy refinement relation ensures that the correct enforcement of the final, concrete policy specification implies the enforcement of all the previous, more abstract policy specifications.

For analysis of abstract specifications to be meaningful, the results of the proofs or verifications conducted at the abstract level must be preserved under refinement and by the eventual implementation. Otherwise the analysis must be conducted from scratch after every refinement step [21]. This paper addresses the problem of preservation of policy adherence under refinement. Adherence of a system to a policy specification means that the system satisfies the policy specification. In [35] policy adherence is formalized as a relation between policy specifications and system implementations. In [31] we addressed the relation between policy specifications and STAIRS system specifications, and formally captured what it means that a system specification adheres to a policy specification. In the latter paper we furthermore showed that for a fixed policy specification, policy adherence is preserved under refinement of system specifications. This means that in a process of system development using STAIRS, analysis of the system specification with respect to policy adherence can safely be conducted at abstract levels.

In this paper we address the problem of preservation of policy adherence in a setting in which both the policy specification and the system specification may undergo refinement. Since the requirements imposed by a policy specification are strengthened under refinement, adherence is not preserved in the general case. The challenge is therefore to identify the requirements for which adherence has been verified at the abstract level, and the conditions under which adherence to these requirements is preserved. We furthermore identify development rules the application of which ensures adherence preservation.

Preservation of properties under refinement is an important feature of system development as it allows these properties to be taken into account throughout the whole development process. Deontic STAIRS is a generic approach to policy-based management in the sense that it supports the specification of various types of policies, such as security, trust and access control. Preservation of policy adherence under refinement therefore means that the types of requirements that may be specified as policies using Deontic STAIRS can be an integrated part of system development.

In Section 2 we give a more detailed presentation of the challenge addressed in this paper and motivate the work by describing possible development cases in the setting of policy-based management. In Section 3 we present a formal semantics for policy specifications that characterizes the requirements imposed by a given policy specification. The semantics of Deontic STAIRS specifications presented in [31, 35] is given by describing the semantics of individual rules of the policy specification separately. The semantics presented in this paper combines the semantics of several rules into one representation, thus describing the requirements imposed by all the rules of a policy specification simultaneously. In Section 4 we define the notion of policy adherence as a relation between policy specifications and system specifications. The formalization of the adherence relation is based on the semantics of policy specifications proposed in Section 3, and we prove that this notion of adherence is equivalent to the notion of adherence formalized in [31]. The notion of policy refinement is in Section 5 formalized as a relation between policy specifications. In [35] refinement of policy specifications is formalized by defining refinement of individual policy rules. Based on the combined semantics of policy specifications proposed in this paper, the policy refinement relation defined in Section 5 formalizes what it means that the combined rules of one policy specification is a refinement of the combined rules of another policy specification. We prove that under a given assumption the proposed policy refinement relation is a generalization of the policy refinement relation in [35], and we motivate and explain the generalization. Section 6 addresses the problem of preservation of policy adherence under refinement. The section gives a general characterization of the conditions under which policy adherence is preserved. We furthermore present rules for refinement of system specifications that can be derived from steps of policy refinement, and show that the rules ensure preservation of adherence under the combined refinement of the policy specification and system specification. In Section 7 we present related work before we conclude in Section 8. Formal definitions and detailed proofs of the results are given in the appendix.

2 Problem Characterization

With support for specification of policies and systems at various levels of abstraction, where the abstraction levels are related by notions of refinement, we are interested in identifying conditions under which analysis results from an abstract level are preserved under refinement. In this paper we address the issue of preservation of adherence of system specifications S to policy specifications P under refinement of either S or P. When such conditions are fulfilled, adherence is guaranteed at the concrete level for the adherence results that were verified at the abstract level, and need not be checked again after refinement.

Policy refinement is not well addressed in the research on policy-based management, which means that the problem of preservation of policy adherence under refinement is also scarcely addressed. The problem of integrating policy requirements with the requirements to system design and functionality and the preservation of these under the development process is, however, recognized and well addressed. This is particularly the case for security requirements.

In [21] it is argued that security requirements should be taken into account during the system development process. The reason is that enforcing security only at the end of the development process "by preventing certain behaviors... may result in a so useless system that the complete development effort would be wasted" [21]. A further argument stated in [17] is that "it would be desirable to consider security aspects already in the design phase, before a system is actually implemented, since removing security flaws in the design phase saves cost and time".

The approach to security preservation under refinement addressed in e.g. [16, 21] assumes that the security properties to be enforced are given already at the beginning of the system development process. The challenge is therefore to ensure that these properties are preserved under refinement. The capturing and specification of the desired security requirements is in this case held separate from the system development in which these requirements are integrated. In the context of policy-based management this would correspond to a development case in which the policy is captured and developed before the system to which the policy applies is developed while ensuring that the system adheres to the policy.

The approach proposed in [4], on the other hand, is an integration of the security model with the system model in a combined development process. The system model specifies the system design and functionality, whereas the security model specifies the access control policy for the system. A similar approach is proposed in [17, 18] for providing support for various security aspects to be taken into account during the overall system development.

Fig. 1 illustrates the two development cases for a policy P that applies to a system S. In both cases the initial, most abstract specification of the policy is denoted by P_1, and the initial, most abstract specification of the system is correspondingly denoted by S_1. By $P_1 \rightsquigarrow P_2$ we denote that the policy specification P_2 is a refinement of the policy specification P_1, and similarly by $S_1 \rightsquigarrow S_2$ for system specifications. Since both the policy refinement relation and the system refinement relation are transitive, the final, most concrete specifications P_n and S_m are refinements of P_1 and S_1 respectively.

Case (a) in Fig. 1 illustrates a separate development of the policy P and the system S. The abstract policy specification P_1 may, for example, have been derived from a risk analysis and a set of security requirements and then further developed and refined into the low-level and detailed policy specification P_n that is to be enforced. Subsequently the system S to which the policy P applies is developed. In order to ensure that the security requirements are satisfied by S, the system development may be conducted in such a way that policy adherence is maintained during the development process. By $P_n \rightarrow_a S_1$ of case (a) in Fig. 1 we denote that the system specification S_1 adheres to the policy specification P_n. During the development of S the policy specification P_n is fixed, so in case adherence is preserved under refinement of system specifications adherence of the final system specification S_m to P_n is guaranteed. This property is denoted by the dashed arrow from

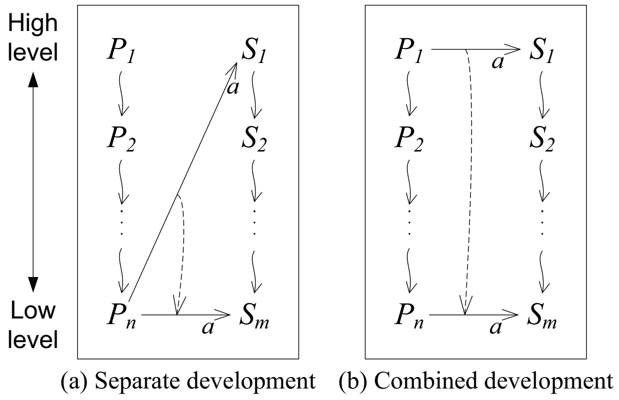

Figure 1: Development cases

$P_n \rightarrow_a S_1$ to $P_n \rightarrow_a S_m$. As mentioned in the introduction, this property of the STAIRS refinement relation was shown in [31] and expressed by the following theorem.

Theorem 1. $P \rightarrow_a S_1 \wedge S_1 \rightsquigarrow S_2 \Rightarrow P \rightarrow_a S_2$

Case (b) in Fig. 1 illustrates a combined development of the policy specification and the system specification. A policy specification P_i is taken into account in the system specification S_k in order to ensure adherence. As in the previous case, S_k may be further developed into a specification S_l, but the difference is that the development process should allow the policy specification to be strengthened into a refined policy specification P_j. Since both refinement relations are reflexive the case addresses situations of refinement of one or both of the policy specification and the system specification. The property of adherence preservation under such combined refinement is illustrated by the dashed arrow from $P_1 \rightarrow_a S_1$ to $P_n \rightarrow_a S_m$ in Fig. 1.

Generally, refinement of policy specifications involves a strengthening of the requirements imposed by the policy. This means that adherence of a system specification S is not preserved under refinement of a policy specification P in the general case. However, if a policy refinement step involves a strengthening by imposing additional policy rules, analysis with respect to adherence to these have not been conduced at the abstract level. Instead, these rules yield proof obligations that must be resolved at the refined level. The challenge is therefore to identify conditions under which the analysis results from the abstract level is preserved under refinement. For a condition C the desired property can be formulated as follows.

$$\frac{C}{P_1 \rightarrow_a S_1 \wedge P_1 \rightsquigarrow P_2 \wedge S_1 \rightsquigarrow S_2 \Rightarrow P_2 \rightarrow_a S_2} \tag{1}$$

For cases in which $P_1 \rightarrow_a S_1$ has been established, the strategy for preservation of policy adherence results under refinement of P_1 and S_1 proposed in this paper is to identify differences between the policy specification P_1 and the refined policy specification P_2. These differences represent the strengthening of the requirements imposed by the policy when moving from P_1 to P_2. By defining rules for refinement of the system specification S_1 based on the identified policy strengthening, we obtain a refined system specification S_2 in which the adherence results establishing $P_1 \rightarrow_a S_1$ are preserved.

In the proposed strategy we utilize properties of modularity and transitivity for the purpose of breaking the larger problem down into smaller sub-problems that can be addressed separately.

3 Semantics of Policy Specifications

A policy specification P is in Deontic STAIRS given as a set of policy rule specifications (dm, d_t, d_b), where $dm \in \{pe, ob, pr\}$ denotes the deontic modality of a permission, obligation or prohibition, d_t specifies the trigger and d_b specifies the rule body. The trigger d_t of a policy rule is a sequence diagram that specifies the conditions under which the policy rule applies, whereas the rule body d_b is a sequence diagram that specifies the behavior that is constrained by the rule. The semantics of the diagrams d_t and d_b is given by the function $[\![\,]\!]$ that yields the set of traces $[\![d_t]\!], [\![d_b]\!] \subseteq \mathcal{H}$ that is specified by the diagrams. The set $\mathcal{H}$ denotes the set of all traces and the set $\mathcal{R}$ denotes the set of all policy rule specifications. The reader is referred to [35] for a more detailed presentation.

Given a trace h describing a possible run of a system for which a policy rule (dm, d_t, d_b) applies, h triggers the rule if h fulfills the triggering scenario d_t. Since d_t is described by a set of traces, each describing a valid interpretation of the diagram, it suffices that h fulfills at least one trace $h' \in [\![d_t]\!]$ to trigger the rule. For h to fulfill h', h must be a super-trace of h' which we denote $h' \lhd h$. Equivalently, we say that h' is a sub-trace of h. We have, for example, that $\langle a, b, c\rangle \lhd \langle e, a, f, g, b, c, d\rangle$. For a trace set $H \subseteq \mathcal{H}$, $H \lhd h$ denotes $\exists h' \in H : h' \lhd h$. See Appendix A for the formal definition of the sub-trace relation $\lhd$ and the complementary relation $\not\lhd$.

If $[\![d_t]\!] \lhd h$ holds for a policy rule (dm, d_t, d_b), i.e. h triggers the rule, the rule imposes a constraint on the possible continuations of the execution of h after the rule has been triggered. A permission requires that the behavior described by the rule body d_b must be offered as a potential choice, i.e. there must exist a continuation of h that fulfills the rule body. An obligation requires that all possible continuations fulfill the rule body, whereas a prohibition requires that none of the possible continuations fulfills the rule body.

In this section we define the semantics of policy specifications as a function that takes a policy specification $P \in \mathbb{P}(\mathcal{R})$ and a trace $h \in \mathcal{H}$ and yields a tuple (a, u, C) that describes the requirements imposed by the rules in P given the execution of the trace h, where $a, u \subseteq \mathcal{H}$ and C is a set of trace sets $H \subseteq \mathcal{H}$. The interpretation of the tuple (a, u, C) is that the set a represents the acceptable traces as specified by the obligation rules of P that are triggered by h; the set u represents the unacceptable traces as specified by the prohibition rules of P that are triggered by h; each set $H \in C$ represents the traces that must be offered as potential choices as specified by a permission rule in P that is triggered by h.

Formally, the semantics of a policy specification is defined by the following function.

$$[\![\_]\!]_{\_} \in \mathbb{P}(\mathcal{R}) \times \mathcal{H} \rightarrow \mathbb{P}(\mathcal{H}) \times \mathbb{P}(\mathcal{H}) \times \mathbb{P}(\mathbb{P}(\mathcal{H}))$$

We first define the function for singleton sets of policy rules P and then define the composition of policy specifications P_1 and P_2.

The operator $\succsim$ takes two trace sets as operands and yields their sequential composition, capturing the formalization of the seq operator of sequence diagrams. For sequence diagrams d_1 and d_2, the semantics of d_1 seq d_2 is given by $[\![d_1]\!] \succsim [\![d_2]\!]$. For a trace set H, $H\lhd$ denotes the set of all supertraces of elements in H. The reader is referred to the appendix for the formal definitions.

Definition 1. Semantics of policy rules.

$$[\![\{(pe, d_t, d_b)\}]\!](h) \overset{\mathsf{def}}{=} \begin{cases} (\mathcal{H}, \emptyset, \{([\![d_t]\!] \succsim [\![d_b]\!])\lhd\}) & \text{if } [\![d_t]\!] \lhd h \\ (\mathcal{H}, \emptyset, \emptyset) & \text{if } [\![d_t]\!] \not\lhd h \end{cases}$$

$$[\![\{(ob, d_t, d_b)\}]\!](h) \overset{\mathsf{def}}{=} \begin{cases} (([\![d_t]\!] \succsim [\![d_b]\!])\lhd, \emptyset, \emptyset) & \text{if } [\![d_t]\!] \lhd h \\ (\mathcal{H}, \emptyset, \emptyset) & \text{if } [\![d_t]\!] \not\lhd h \end{cases}$$

$$[\![\{(pr, d_t, d_b)\}]\!](h) \overset{\mathsf{def}}{=} \begin{cases} (\mathcal{H}, ([\![d_t]\!] \succsim [\![d_b]\!])\lhd, \emptyset) & \text{if } [\![d_t]\!] \lhd h \\ (\mathcal{H}, \emptyset, \emptyset) & \text{if } [\![d_t]\!] \not\lhd h \end{cases}$$

We refer to $[\![P]\!](h) = (a, u, C)$ as "the denotation of the policy specification P with respect to the trace h", or "the policy denotation" as a brevity.

Definition 2. Composition of policy denotations.

$$(a_1, u_1, C_1) \otimes (a_2, u_2, C_2) \overset{\mathsf{def}}{=} (a_1 \cap a_2, u_1 \cup u_2, C_1 \cup C_2)$$

Given the policy denotations (a_1, u_1, C_1) and (a_2, u_2, C_2), the sets a_1 and a_2 represent the obliged behavior. The composition of the policy denotations means that both a_1 and a_2 are obliged, which explains the composition of these sets using intersection. Since the sets u_1 and u_2 represent prohibited behavior, the composition using union ensures that both sets remain prohibited after composition. The sets C_1 and C_2 each represents the various behaviors that must be offered as potential alternatives as specified by permission rules. The union operator ensures that all the alternatives are still represented after the composition.

It follows immediately from the properties of associativity and commutativity of $\cap$ and $\cup$ that also the composition operator $\otimes$ is associative and commutative.

Composition of policy specifications is defined by the union operator since policy specifications are given by sets of policy rule specifications. The following defines the semantics of composed policy specifications.

Definition 3. Semantics of composed policy specifications.

$$[\![P_1 \cup P_2]\!](h) \overset{\mathsf{def}}{=} [\![P_1]\!](h) \otimes [\![P_2]\!](h)$$

For the special case in which the policy specification is empty, i.e. $P = \emptyset$, there are no requirements. The semantic representation of the empty policy specification is defined as follows.

Definition 4. Semantics of empty policy specifications.

$$\forall h \in \mathcal{H} : [\![\emptyset]\!](h) \overset{\mathsf{def}}{=} (\mathcal{H}, \emptyset, \emptyset)$$

In [35] the semantics of Deontic STAIRS specifications P were defined by defining the semantics of individual policy rule specifications such that $[\![P]\!] = \{[\![r]\!] \mid r \in P\}$. An advantage of the composed semantics of policy specifications proposed in this paper is that policy specifications that impose the same requirements on systems are also semantically equivalent. The semantics of the singleton rule set $\{(ob, d_t, d_{b1}\ \mathsf{par}\ d_{b2})\}$, for example, equals the semantics of the rule set $\{(ob, d_t, d_{b1}), (ob, d_t, d_{b2})\}$. Similarly, the semantics of $\{(pr, d_t, d_{b1}\ \mathsf{alt}\ d_{b2})\}$ equals the semantics of $\{(pr, d_t, d_{b1}), (pr, d_t, d_{b2})\}$.

4 Policy Adherence

In this section we define the adherence relation that precisely characterizes what it means that a system specification S satisfies a policy specification P, denoted $P \rightarrow_a S$.

The system is specified using the STAIRS approach to system development with UML sequence diagrams. In STAIRS, the semantics $[\![d]\!]$ of a sequence diagram d is a non-empty set of so-called interaction obligations (p, n), where $p, n \subseteq \mathcal{H}$. Each element $(p, n) \in [\![d]\!]$ represents an obligation in the sense that the behavior described by the trace sets must be offered as a potential system behavior. The set p is the set of positive traces, i.e.

the traces each of which represents a valid execution of the interaction obligation. The set n is the set of negative traces, i.e. the traces representing invalid executions of the interaction obligation. The reader is referred to [31] for a detailed presentation of the STAIRS syntax and semantics of UML sequence diagrams.

We now define adherence of a system specification S to a policy specification P with respect to the semantics $[\![P]\!](h)$ presented in the previous section.

Definition 5. Adherence of system specifications S to policy specifications P.

$$\begin{aligned}
P \rightarrow_a S \overset{\mathsf{def}}{=}\ & \forall (p,n) \in [\![S]\!] : \\
& \forall h \in (p \setminus n) : \\
& \quad h \in a \wedge \\
& \quad h \notin u \wedge \\
& \quad \forall H \in C : \exists (p',n') \in [\![S]\!] : \forall h' \in (p' \setminus n') : h' \in H \\
& \text{where } [\![P]\!](h) = (a,u,C)
\end{aligned}$$

The first and second conjuncts ensure that the trace h adheres to the obligation rules and prohibition rules, respectively, of P. The third conjunct ensures adherence to the permission rules of P by requiring that the behavior specified by the these rules are offered as potential choices by S.

Based on the semantics of policy specifications presented in [35], policy adherence of system specifications S to policy specifications P, denoted $P \rightarrow'_a S$,[1] is in [31] defined $\forall r \in P : r \rightarrow'_a S$.[2] The next theorem relates the definition of adherence of this paper to the adherence relation defined in [31].

Theorem 2. $P \rightarrow_a S \Leftrightarrow P \rightarrow'_a S$

Policy adherence is modular in the sense that a system adheres to a set of policy specifications if and only if the system adheres to the composition of the policy specifications. This is expressed by the following theorem, and implies that the problem of verification of adherence can be broken down into sub-problems.

Theorem 3. $P_1 \rightarrow_a S \wedge P_2 \rightarrow_a S \Leftrightarrow (P_1 \cup P_2) \rightarrow_a S$

The lowest level of granularity for a modular verification of adherence of a system specification S to a policy specification P is to address subsets of P that are singleton, i.e. to address individual policy rules. The following theorem follows immediately from Def. 1 and Def. 5 and expresses what it means that a system specification adheres to a policy rule.

[1]We denote the adherence relation of [31] by $\rightarrow'_a$ to distinguish it from the adherence relation $\rightarrow_a$ defined in this paper.

[2]See Def. 15 and Def. 16 in the appendix.

Theorem 4. Adherence to policy rules.

$$
\begin{aligned}
\{(pe, d_t, d_b)\} \rightarrow_a S \quad &\Leftrightarrow \quad \forall (p, n) \in [\![S]\!] : \forall h \in (p \setminus n) : (h \in [\![d_t]\!]\lhd \Rightarrow \\
&\qquad \exists (p', n') \in [\![S]\!] : \forall h' \in (p' \setminus n') : h' \in ([\![d_t]\!] \succsim [\![d_b]\!])\lhd) \\
\{(ob, d_t, d_b)\} \rightarrow_a S \quad &\Leftrightarrow \quad \forall (p, n) \in [\![S]\!] : \forall h \in (p \setminus n) : (h \in [\![d_t]\!]\lhd \Rightarrow \\
&\qquad h \in ([\![d_t]\!] \succsim [\![d_b]\!])\lhd) \\
\{(pr, d_t, d_b)\} \rightarrow_a S \quad &\Leftrightarrow \quad \forall (p, n) \in [\![S]\!] : \forall h \in (p \setminus n) : (h \in [\![d_t]\!]\lhd \Rightarrow \\
&\qquad h \notin ([\![d_t]\!] \succsim [\![d_b]\!])\lhd)
\end{aligned}
$$

This modularity property of policy adherence is utilized in Section 6 in the identification of refinement rules that guarantees preservation of adherence results.

5 Policy Refinement

In this section we formally define the notion of policy refinement as a relation between policy specifications, where $P_1 \rightsquigarrow P_2$ denotes that the policy specification P_2 is a refinement of the policy specification P_1. For policy denotations (a, u, C), refinement is defined as follows.

Definition 6. Refinement of policy denotations.

$$
\begin{aligned}
(a_1, u_1, C_1) \rightsquigarrow (a_2, u_2, C_2) \quad \overset{\mathsf{def}}{=} \quad & a_2 \subseteq a_1 \wedge \\
& u_1 \subseteq u_2 \wedge \\
& \forall H_1 \in C_1 : \exists H_2 \in C_2 : H_2 \subseteq H_1 \wedge \\
& \forall H_2 \in C_2 : \exists H_1 \in C_1 : H_2 \subseteq H_1
\end{aligned}
$$

Refinement of a policy specification P_1 to a policy specification P_2 then means that for all traces $h \in \mathcal{H}$, the denotation of P_2 with respect to h is a refinement of the denotation of P_1 with respect to h, formally defined as follows.

Definition 7. Refinement of policy specifications.

$$P_1 \rightsquigarrow P_2 \overset{\mathsf{def}}{=} \forall h \in \mathcal{H} : [\![P_1]\!](h) \rightsquigarrow [\![P_2]\!](h)$$

The first and second conjuncts of Def. 6 mean that the requirements imposed by the obligation rules and prohibition rules of the policy specifications are strengthened under policy refinement. The strengthening may stem from reduction of underspecification in existing rules, or from the addition of new policy rules. The third and fourth conjuncts of Def. 6 mean that all behaviors that are required to be offered potentially by permission rules of P_1 are also required by permission rules of P_2 and vice versa. The behavior described by the rule body of the permission rules may also be subject to reduction of underspecification.

Under this notion of policy refinement, the variation over potential choices of behavior that is required by the policy is fixed under refinement. The reduction of underspecification, however, means that traces that are admissible at the abstract level may be inadmissible at the refined level.

It follows immediately from Def. 6 and Def. 7 that the policy refinement relation is reflexive and transitive.

A modularity property of the policy refinement relation is that a policy specification P can be refined by refining subsets of P separately, as expressed by the following theorem.

Theorem 5. $P_1 \rightsquigarrow P_1' \wedge P_2 \rightsquigarrow P_2' \Rightarrow (P_1 \cup P_2) \rightsquigarrow (P_1' \cup P_2')$

Since refinement of policy specifications is a strengthening of the requirements imposed by the policy specification, the requirements from the abstract levels are preserved under refinement. This means that the enforcement of a concrete policy specification implies the enforcement of the previous, more abstract specifications. This property is expressed in the next theorem stating that adherence to a concrete policy specification guarantees adherence to an abstract policy specification.

Theorem 6. $P_1 \rightsquigarrow P_2 \wedge P_2 \rightarrow_a S \Rightarrow P_1 \rightarrow_a S$

The modularity properties expressed in Theorem 3 and Theorem 5 together with Theorem 6 mean that a policy specification can safely be split up into partial policy specifications that are further developed and eventually enforced separately. This is relevant for organizations in which an overall organizational policy is customized for and implemented separately by individual departments, and for policy-based management of distributed systems where policies are developed and enforced in a distributed manner.

The notion of policy refinement presented in [35] is based on the definition of refinement of individual policy rules; a policy specification P_2 is a refinement of a policy specification P_1 if every rule in P_2 is a refinement of a rule in P_1.[3] This means that policy rules may be added under refinement. The notion of policy refinement studied in this paper is more restrictive in the sense that it does not allow requirements to new potential choices of behavior to be introduced under refinement; this is allowed under the notion of policy refinement in [35], however, by the addition of permission rules.

Under the condition that requirements to new potential choices of behavior are not added by moving from the abstract specification P_1 to the refined specification P_2, we may compare the policy refinement relation of Def. 7 in this paper with the policy refinement relation in [35]. The condition is formalized with the relation $\gg$ between policy denotations as follows.

[3]See Def. 17 and Def. 18 in the appendix.

Definition 8.

$$(a_1, u_1, C_1) \gtrdot (a_2, u_2, C_2) \overset{\mathsf{def}}{=} \forall H_2 \in C_2 : \exists H_1 \in C_1 : H_2 \subseteq H_1$$

For policy specifications P_1 and P_2 the relation $\gtrdot$ is then defined as follows.

Definition 9. $P_1 \gtrdot P_2 \overset{\mathsf{def}}{=} \forall h \in \mathcal{H} : [\![P_1]\!](h) \gtrdot [\![P_2]\!](h)$

As expressed in the next theorem, under the condition that $P_1 \gtrdot P_2$ holds, the notion of policy refinement proposed in this paper is a generalization of the notion of policy refinement presented in [35].[4]

Theorem 7. $P_1 \rightsquigarrow' P_2 \wedge P_1 \gtrdot P_2 \Rightarrow P_1 \rightsquigarrow P_2$

A number of modularity properties for refinement of policy specifications were shown in [35], demonstrating that the sequence diagrams specifying policy rules can be refined by refining their individual parts separately. These modularity properties hold irrespective of whether rules are added under the refinement $P_1 \rightsquigarrow' P_2$. The next theorem states that under a refinement $P_1 \rightsquigarrow' P_2$ in which every rule at the refined level is a refinement of a rule at the abstract level, the relation $P_1 \gtrdot P_2$ holds.

Theorem 8. $P_1 \rightsquigarrow' P_2 \wedge \forall r_2 \in P_2 : \exists r_1 \in P_1 : r_1 \rightsquigarrow' r_2 \Rightarrow P_1 \gtrdot P_2$

By Theorem 7 and Theorem 8, the modularity properties of the policy refinement relation formalized in [35] carry over to the policy refinement relation proposed in this paper. This means, for example, that a sequence diagram d_b specifying a rule body can be refined by refining individual parts of d_b separately.

A clear advantage of the policy refinement relation proposed in this paper over the policy refinement relation in [35] is that it is based on the composed semantics of the policy rules, rather than the collection of the separate semantics of the individual rules. Theorem 6 reflecting the property that policy refinement is a strengthening of the requirements imposed by the policy specification also holds for the policy refinement relation presented in [35]. By Theorem 7, however, there exist policy specifications P_1 and P_2 such that $P_2 \rightarrow_a S \Rightarrow P_1 \rightarrow_a S$ for all system specifications S, while $P_1 \rightsquigarrow P_2$ and $\neg(P_1 \rightsquigarrow' P_2)$. Theorem 6 and Theorem 7 therefore demonstrate that the refinements that are valid under the relation $\rightsquigarrow$ presented in this paper while invalid under the relation $\rightsquigarrow'$ of [35] are sound in a policy development process in the sense that they involve a strengthening of the policy.

In Section 3 we explained that under the semantics of policy specifications presented in [35] there are policy specifications that are equivalent with

[4]We denote the policy refinement relation of [35] by $\rightsquigarrow'$ to distinguish it from the policy refinement relation $\rightsquigarrow$ defined in this paper.

respect to the requirements they impose, yet semantically non-equivalent. Since these are semantically equivalent under the semantics proposed in this paper, they also refine each other. For example, since the policy specification $\{(ob, d_t, d_{b1} \textsf{ par } d_{b2})\}$ is semantically equivalent to $\{(ob, d_t, d_{b1}), (ob, d_t, d_{b2})\}$, the two refine each other. Furthermore, all valid refinements of one policy specification are also valid refinements of semantically equivalent policy specifications.

The policy refinement relation of [35] allows refinement of permission rules and obligation rules by reducing the set of traces that are specified by the rule body, and refinement of prohibition rules by increasing this trace set. This is generalized by the policy refinement relation proposed in this paper by considering the set of super-traces of the specifications instead. This means, for example, that while $\{(ob, d_t, d_{b1})\} \rightsquigarrow \{(ob, d_t, d_{b1} \textsf{ par } d_{b2})\}$ is valid, it is not so for the policy refinement relation $\rightsquigarrow'$.

6 Adherence Preserving Refinement

In this section we first give a general characterization of conditions under which adherence is preserved in the case of the combined refinement of policy specifications and system specifications. Subsequently we identify refinement rules that fulfill these conditions, such that the application of the rules in the development process guarantees preservation of adherence.

6.1 A Characterization of Adherence Preserving Refinements

A policy specification characterizes certain behavior as admissible and certain behavior as inadmissible by the obligation rules and the prohibition rules. Additionally a policy specification requires certain behavior to be offered as potential choices by the permission rules. The expressiveness of STAIRS system specifications to capture inherent non-determinism by a set of interaction obligations and preserve this form of non-determinism under refinement are the properties that ensure preservation of adherence to permission rules under refinement. Underspecification is captured by the variation over positive traces in each interaction obligation. The set of positive traces represents the admissible behavior of the system under development. If these traces adhere to the obligation rules and prohibition rules at an abstract level, adherence is preserved under refinement of system specifications since refinement of interaction obligations is defined by reduction of underspecification. The property of preservation of adherence to a fixed policy specification under refinement of system specifications as expressed by Theorem 1 is therefore ensured by the fact that all the potential alternatives are preserved, while the admissible behavior within each of the alternatives may only be reduced.

Because policy refinement implies only a reduction of the admissible behavior as specified by the policy, adherence of a system specification is preserved under refinement by reducing the admissible behavior of the system specification accordingly. In other words, if $P_1 \rightarrow_a S_1$ has been established at the abstract level and the policy refinement $P_1 \rightsquigarrow P_2$ is conducted, a system specification S_2 can be derived from S_1 based on the reduction of admissible behavior when shifting from P_1 to P_2 such that $S_1 \rightsquigarrow S_2$ and $P_2 \rightarrow_a S_2$.

Policy adherence is obviously preserved if the requirements imposed by the refined policy specification is equivalent to the requirements imposed by the abstract specification. Under this condition, preservation of adherence under refinement can be formulated by the following instantiation of expression (1) in Section 2.

$$\frac{\forall h \in \mathcal{H} : [\![P_1]\!](h) = [\![P_2]\!](h)}{P_1 \rightarrow_a S_1 \wedge S_1 \rightsquigarrow S_2 \Rightarrow P_2 \rightarrow_a S_2} \tag{2}$$

Notice that we have omitted the conjunct $P_1 \rightsquigarrow P_2$ of expression (1) in Section 2 since it is implied by the condition in expression (2) and therefore redundant.

Under the condition in (2), there is no reduction of admissible behavior, and adherence is preserved by Theorem 1. The modularity properties of policy adherence and policy refinement captured by Theorem 3 and Theorem 5, respectively, can be utilized by identifying parts of the refined policy specification that is semantically equivalent to the abstract policy specification or a subset of the abstract policy specification. If, for example, $P_1 \cup P_2 \rightsquigarrow P_1' \cup P_2'$ and $\forall h \in \mathcal{H} : [\![P_1]\!](h) = [\![P_1']\!](h)$, policy adherence of a system specification S need only be checked with respect to P_2' if $P_1 \cup P_2 \rightarrow_a S$ has been verified.

For policy refinements $P_1 \rightsquigarrow P_2$ in which the semantics of the policy specifications are not equivalent, and $P_1 \rightarrow_a S$ has been verified, we need to identify the traces h of the system specification S that are admissible under P_1 and inadmissible under P_2. A refinement $S \rightsquigarrow S'$ of the system specification by eliminating these traces h from S then ensures adherence at the refined level, i.e. $P_2 \rightarrow_a S'$. We define the function

$$\Delta(\_,\_,\_) \in \mathbb{P}(\mathcal{R}) \times \mathbb{P}(\mathcal{R}) \times \mathcal{D} \rightarrow \mathbb{P}(\mathcal{H})$$

that takes the policy specifications P_1 and P_2 and the system specification S as operands and yields the set H of admissible traces from S that are admissible under P_1 and inadmissible under P_2. $\mathcal{D}$ denotes the set of all STAIRS sequence diagram specifications. Formally, the function Δ is defined as follows.

Definition 10. For policy specifications P_1 and P_2 and system specifications S such that $P_1 \rightsquigarrow P_2$ and $P_1 \rightarrow_a S$:

$$\begin{aligned}\Delta(P_1, P_2, S) \overset{\mathsf{def}}{=} \{h \in \mathcal{H} \mid\ & (\exists (p, n) \in [\![S]\!] : h \in (p \setminus n)) \wedge \\ & (h \in (a_1 \setminus a_2) \vee \\ & \ h \in (u_2 \setminus u_1) \vee \\ & \ \exists H_1 \in C_1 : \exists H_2 \in C_2 : H_2 \subseteq H_1 \wedge h \in (H_1 \setminus H_2))\} \\ \text{where } [\![P_1]\!](h) = (a_1, u_1, C_1) & \text{ and } [\![P_2]\!](h) = (a_2, u_2, C_2)\end{aligned}$$

Semantically, a sequence diagram is represented by a set O of interaction obligations (p, n). The singleton set $\{(\emptyset, \Delta(P_1, P_2, S))\}$ of interaction obligations is then the semantic representation of a sequence diagram that specifies as negative the behavior that should be inadmissible after the refinement of the system specification. We denote this representation by $[\![S]\!]^{\Delta(P_1,P_2,S)}$, abbreviated by $[\![S]\!]^{\Delta}$ when the given policy specifications are irrelevant or clear from the context.

For a system specification S such that $P_1 \rightarrow_a S$ and $P_1 \rightsquigarrow P_2$ hold, the following represents a refined system specification in which the inadmissible behavior $[\![S]\!]^{\Delta}$ has been specified as negative behavior: $[\![S]\!]^{\Delta} \uplus [\![S]\!]$.

The operator $\uplus$ takes two sets of interaction obligations as operands and yields their inner union, formally defined as follows.

$$O_1 \uplus O_2 \overset{\mathsf{def}}{=} \{(p_1 \cup p_2, n_1 \cup n_2) \mid (p_1, n_1) \in O_1 \wedge (p_2, n_2) \in O_2\}$$

Since $[\![S]\!]^{\Delta}$ is a singleton set of interaction obligations, the result of $[\![S]\!]^{\Delta} \uplus [\![S]\!]$ is equal to the result of adding the set $\Delta(P_1, P_2, S)$ to the negative traces of each interaction obligation of $[\![S]\!]$. More formally,

$$[\![S]\!]^{\Delta} \uplus [\![S]\!] = \{(p, n \cup \Delta(P_1, P_2, S)) \mid (p, n) \in [\![S]\!]\}$$

The result of this composition with inner union is furthermore a refinement of $[\![S]\!]$ as expressed by the following theorem which follows immediately from the definition of refinement of sets of interaction obligations [31].

Theorem 9. For all interaction obligations $(\emptyset, n)$ and all sets of interaction obligations O:

$$O \rightsquigarrow \{(\emptyset, n)\} \uplus O$$

The refinement $[\![S]\!]^{\Delta} \uplus [\![S]\!]$ of $[\![S]\!]$ then characterizes a system specification where the adherence $P_1 \rightarrow_a S$ is preserved when the policy refinement $P_1 \rightsquigarrow P_2$ has been conducted. The result is expressed by the following theorem.

Theorem 10. $P_1 \rightarrow_a S \wedge P_1 \rightsquigarrow P_2 \Rightarrow P_2 \rightarrow_a ([\![S]\!]^{\Delta} \uplus [\![S]\!])$

Notice that adherence as formalized in Def. 5 is a relation between a policy specification and a system specification, whereas $([\![S]\!]^{\Delta} \uplus [\![S]\!])$ in Theorem 10 denotes a set of interaction obligations. The formulation of Theorem 10 is for sake of brevity and readability.

The result of adherence preservation under the condition that the abstract system specification is S is refined by the composition with $[\![S]\!]^{\Delta}$ can then be formulated by the following instantiation of (1).

$$\frac{[\![S_2]\!] = [\![S_1]\!]^{\Delta} \uplus [\![S_1]\!]}{P_1 \rightarrow_a S_1 \wedge P_1 \rightsquigarrow P_2 \Rightarrow P_2 \rightarrow_a S_2} \tag{3}$$

Notice that since $[\![S_1]\!] \rightsquigarrow [\![S_1]\!]^{\Delta} \uplus [\![S_1]\!]$ holds by Theorem 9, the conjunct $S_1 \rightsquigarrow S_2$ of the antecedent of expression (1) in Section 2 is implied. Since the conjunct is redundant it is omitted in (3).

So far we have only given a general characterization of the refined system specification for which adherence is preserved by describing its semantics. In a practical setting the development process should, however, be supported by syntactical rules that ensure the desired refinements.

The trace set $\Delta(P_1, P_2, S)$ which is derived as expressed in Def. 10 characterizes precisely the traces in S that are inadmissible after the policy refinement $P_1 \rightsquigarrow P_2$. By identifying a sequence diagram d where exactly these inadmissible traces are negative, the diagram d can be syntactically composed with S to yield the desired result. More precisely, if $[\![d]\!] = \{(\emptyset, \Delta(P_1, P_2, S))\} = [\![S]\!]^{\Delta}$ we have the following.

$$[\![d \ \mathsf{alt}\ S]\!] = [\![S]\!]^{\Delta} \uplus [\![S]\!]$$

The semantics of the sequence diagram operator alt is defined by inner union $\uplus$ [31], so given the system specification S and the identified sequence diagram d, the specification d alt S denotes the desired refinement of the system specification.

Theorem 10 therefore means that in principle, the desired system refinement can be conducted. The challenge, however, is to identify the desired sequence diagram d. For certain cases of policy refinements $P_1 \rightsquigarrow P_2$ where $P_1 \rightarrow_a S$ holds, the sequence diagram may not exist since there exist valid semantic representations for which there are no matching representations in the syntax.

In the next subsection we describe strategies for identifying such diagrams d, and we identify specific rules for refinement of system specifications that guarantee adherence preservation.

6.2 Adherence Preserving Refinement Rules

Our strategy is to utilize the modularity properties of policy adherence and policy refinement as expressed by Theorem 3 and Theorem 5, respectively. These properties allow the problem of preservation of adherence under policy refinement to be addressed as a problem of preservation of adherence under policy rule refinement.

Given a refinement of a policy rule r to a policy rule r', the challenge is to identify a sequence diagram d that characterizes the strengthening of the policy rule under the refinement step. There are two requirements that should be fulfilled by this approach. Firstly, the strengthening of the system specification S should be applicable without any knowledge about S other than that it adheres to the abstract policy specification P. This ensures the generality of the approach. Solely by comparing the policy rule specification r and its refinement r', the sequence diagram d representing the strengthening should be derived. Subsequently, a composition of d and S using syntactic sequence diagram operators should yield the desired result in the form of a system specification S'. Secondly, the resulting system specification S' should be a refinement of the original system specification S. This ensures that S' is a valid representation of S.

Ideally, the identified sequence diagram d should capture exactly the set of traces that characterizes the strengthening of the policy rule. In a practical setting it may, however, be infeasible to identify the precise sequence diagram. In some cases, this sequence diagram may not even exist. The sequence diagram d is in the sequel therefore required to characterize at least the strengthening of the policy rule, i.e. $[\![d]\!]$ must be a superset of the desired trace set. Since the removal of traces from the system specification never introduces a policy breach, this requirement is sufficient for the results of adherence preservation.

The formula for strengthening a system specification S into a system specification S' with respect to an identified sequence diagram d is the following.

$$S' = \mathsf{refuse}(d \ \mathsf{par} \ \mathsf{any}) \ \mathsf{alt} \ S$$

The refuse construct is a STAIRS operator for specifying negative behavior [29]. For a sequence diagram d such that $[\![d]\!] = \{(p, n)\}$, the semantics is defined by $[\![\mathsf{refuse}\ d]\!] \overset{\mathsf{def}}{=} \{(\emptyset, p \cup n)\}$. The sequence diagram any denotes the maximal sequence diagram the semantics of which is defined by $[\![\mathsf{any}]\!] = \{(\mathcal{H}, \emptyset)\}$.[5] The reason for the parallel composition of the sequence diagram d and the maximal sequence diagram any in the formula is that it yields all the super-traces of traces in $[\![d]\!]$, i.e. the traces that fulfill the behavior specified by d. This is reflected by the following statement the proof of which is given in Lemma 2 in the appendix.

$$\forall H \subseteq \mathcal{H} : H\lhd = (H \parallel \mathcal{H})$$

The semantics of the par operator is defined by $\parallel$ which yields the interleavings of its operands [13]. The reader is referred to the appendix for the formal definition.

The fulfillment of the requirement that $\mathsf{refuse}(d \ \mathsf{par} \ \mathsf{any}) \ \mathsf{alt} \ S$ is a refinement of S is shown by the following theorem.

[5]The name any is adopted from [20] where it denotes the maximal MSC [15].

Theorem 11. For all sequence diagrams d and d' such that $[\![d']\!]$ is singleton:

$$d \rightsquigarrow (\mathsf{refuse}(d')\ \mathsf{alt}\ d)$$

The theorem follows immediately from the definition of the sequence diagram operators and the definition of refinement of sequence diagrams [31].

We now turn to the approach for adherence preserving refinement of each type of policy rule.

A permission rule (pe, d_t, d_b) is refined by a permission rule (pe, d_t, d'_b) if $[\![d'_b]\!] \subseteq [\![d_b]\!]$. By identifying sequence diagrams d_{b1} and d_{b2} such that $[\![d_b]\!] = [\![d_{b1}\ \mathsf{alt}\ d_{b2}]\!]$ and $d'_b = d_{b1}$, the sequence diagram d_{b2} represents the reduction of the admissible behavior after the refinement. By refining a system specification S for which adherence to (pe, d_t, d_b) has been verified by characterizing traces fulfilling $d_t\ \mathsf{seq}\ d_{b2}$ as negative, the adherence result that was established at the abstract level is preserved. This is expressed by the following theorem.

Theorem 12. Preservation of adherence under refinement of permission rules.

$$\begin{gathered}\{(pe, d_t, d_{b1}\ \mathsf{alt}\ d_{b2})\} \rightarrow_a S \Rightarrow \\ \{(pe, d_t, d_{b1})\} \rightarrow_a \mathsf{refuse}((d_t\ \mathsf{seq}\ d_{b2})\ \mathsf{par}\ \mathsf{any})\ \mathsf{alt}\ S\end{gathered}$$

Notice that for the sequence diagram d_{b2} to characterize the exact strengthening of the permission rule, the requirement $[\![d_{b1}]\!] \cap [\![d_{b2}]\!] = \emptyset$ must be fulfilled, where $[\![d_{b1}]\!] \cup [\![d_{b2}]\!]$ equals the semantics of the body d_b of the abstract permission rule and $[\![d_{b1}]\!]$ equals the semantics of the body d'_b of the refined permission rule. The theorem is still valid if this requirement is not fulfilled, but implies that the removal of traces under the refinement of the system specification is wider than required for adherence to be ensured.

With reference to expression (1) in Section 2 the policy specification P_1 is instantiated by $\{(pe, d_t, d_{b1}\ \mathsf{alt}\ d_{b2})\}$, whereas P_2 is instantiated by $\{(pe, d_t, d_{b1})\}$. The system specification S instantiates S_1 and the system specification S_2 is derived from P_1, P_2 and S_1. Since in this case we have $P_1 \rightsquigarrow P_2$ by definition, and $S_1 \rightsquigarrow S_2$ by Theorem 11, these conjuncts of the antecedent of (1) are redundant. The result of adherence preservation captured by Theorem 12 can then be formulated as follows.

$$\frac{S_2 = \mathsf{refuse}((d_t\ \mathsf{seq}\ d_{b2})\ \mathsf{par}\ \mathsf{any})\ \mathsf{alt}\ S_1}{\{(pe, d_t, d_{b1}\ \mathsf{alt}\ d_{b2})\} \rightarrow_a S_1 \Rightarrow \{(pe, d_t, d_{b1})\} \rightarrow_a S_2} \tag{4}$$

The same approach is applicable to obligation rules. An obligation rule (ob, d_t, d_b) is refined by an obligation rule (ob, d_t, d'_b) if $[\![d'_b]\!] \subseteq [\![d_b]\!]$. Again, we can identify sequence diagrams d_{b1} and d_{b2} such that $[\![d_b]\!] = [\![d_{b1}\ \mathsf{alt}\ d_{b2}]\!]$

and $d_b' = d_{b1}$, where the sequence diagram d_{b2} represents the reduction of the admissible behavior. In case $\{(ob, d_t, d_b)\} \rightharpoonup_a S$ has been verified, adherence is preserved by refining S with respect to the specification d_t seq d_{b2}. This is expressed by the following theorem.

Theorem 13. Preservation of adherence under refinement of obligation rules.

$$\{(ob, d_t, d_{b1} \text{ } \mathsf{alt} \text{ } d_{b2})\} \rightharpoonup_a S \Rightarrow$$
$$\{(ob, d_t, d_{b1})\} \rightharpoonup_a \mathsf{refuse}((d_t \text{ } \mathsf{seq} \text{ } d_{b2}) \text{ } \mathsf{par} \text{ } \mathsf{any}) \text{ } \mathsf{alt} \text{ } S$$

As for the result of preservation of adherence to permission rules, the identification of a sequence diagram characterizing the exact strengthening requires that $[\![d_{b1}]\!] \cap [\![d_{b2}]\!] = \emptyset$. The validity of Theorem 13 does, however, not depend on this requirement. The formulation of the result following the pattern of expression (1) with redundancy omitted is as follows.

$$\frac{S_2 = \mathsf{refuse}((d_t \text{ } \mathsf{seq} \text{ } d_{b2}) \text{ } \mathsf{par} \text{ } \mathsf{any}) \text{ } \mathsf{alt} \text{ } S_1}{\{(ob, d_t, d_{b1} \text{ } \mathsf{alt} \text{ } d_{b2})\} \rightharpoonup_a S_1 \Rightarrow \{(ob, d_t, d_{b1})\} \rightharpoonup_a S_2} \qquad (5)$$

A prohibition rule may be refined by increasing the set of traces that represents the prohibited behavior as specified by the rule body, i.e. a prohibition rule (pr, d_t, d_b) is refined by a prohibition rule (ob, d_t, d_b') if $[\![d_b']\!] \supseteq [\![d_b]\!]$. By identifying sequence diagrams d_{b1} and d_{b2} such that $d_b = d_{b1}$ and $[\![d_b']\!] = [\![d_{b1} \text{ } \mathsf{alt} \text{ } d_{b2}]\!]$, the sequence diagram d_{b2} represents the reduction of the admissible behavior after the refinement. By refining a system specification S for which adherence to (pr, d_t, d_b) has been verified by characterizing traces fulfilling d_t seq d_{b2} as negative, the adherence result that was established at the abstract level is preserved. This is expressed by the following theorem.

Theorem 14. Preservation of adherence under refinement of prohibition rules.

$$\{(pr, d_t, d_{b1})\} \rightharpoonup_a S \Rightarrow$$
$$\{(pr, d_t, d_{b1} \text{ } \mathsf{alt} \text{ } d_{b2})\} \rightharpoonup_a \mathsf{refuse}((d_t \text{ } \mathsf{seq} \text{ } d_{b2}) \text{ } \mathsf{par} \text{ } \mathsf{any}) \text{ } \mathsf{alt} \text{ } S$$

As for the rules for preservation of adherence under refinement of permissions and obligations, the identification of the sequence diagram that characterizes the exact strengthening requires that $[\![d_{b1}]\!] \cap [\![d_{b2}]\!] = \emptyset$ holds. However, since d_{b1} is already characterized as inadmissible by S since adherence to $\{(pr, d_t, d_{b1})\}$ has been verified, the strengthening of S by removing the traces that fulfill (d_t seq d_{b2}) yields precisely the desired refinement of S. The formulation of Theorem 14 following the pattern of expression (1) with redundancy omitted is as follows.

$$\frac{S_2 = \mathsf{refuse}((d_t \text{ } \mathsf{seq} \text{ } d_{b2}) \text{ } \mathsf{par} \text{ } \mathsf{any}) \text{ } \mathsf{alt} \text{ } S_1}{\{(pr, d_t, d_{b1})\} \rightharpoonup_a S_1 \Rightarrow \{(pr, d_t, d_{b1} \text{ } \mathsf{alt} \text{ } d_{b2})\} \rightharpoonup_a S_2} \qquad (6)$$

By properties of modularity the approaches for adherence preserving refinement of policy rules stated in expression (4), (5) and (6) can be combined into an approach for adherence preserving refinement of policies. Assume, for example, that $P_1 \rightarrow_a S_1$ has been verified at an abstract level and that $P_1 = \{r_1, \ldots, r_m\}$ is refined following the above described patterns. For each rule $r_i \in P_1$, let d_i denote the sequence diagram characterizing the strengthening of r_i. The desired strengthening of S_1 into S_2 such that $P_2 \rightarrow_a S_2$ can then be obtained by the following formula.

$$S_2 = \mathsf{refuse}(d_1 \;\mathsf{par}\; \mathsf{any}) \;\mathsf{alt}\; \ldots \;\mathsf{alt}\; \mathsf{refuse}(d_m \;\mathsf{par}\; \mathsf{any}) \;\mathsf{alt}\; S_1 \tag{7}$$

The modularity properties also allow the approach to be applied to a subset of the policy specification. For a policy specification $P_1 \cup P$ such that $P_1 \cup P \rightarrow_a S_1$, we have $P_1 \rightarrow_a S_1$ and $P \rightarrow_a S_1$ by Theorem 3. By refining the rules of P_1 into P_2 following the above pattern, the resulting policy specification $P_2 \cup P$ is a refinement of $P_1 \cup P$ by Theorem 5. Since $S_1 \rightsquigarrow S_2$ holds for the derived system specification S_2, and $P \rightarrow_a S_1$, we get $P \rightarrow_a S_2$ by Theorem 1. By Theorem 3 we finally have $P_2 \cup P \rightarrow_a S_2$.

As an alternative to utilizing the modularity properties as described above, the preservation of adherence under refinement of a set of policy rules can be broken down into a series of refinement steps. Since refinement of both policy specifications and system specifications are transitive, the result of any number of adherence preserving refinement steps of the specifications represents the desired result.

In Section 6.1 we gave with expression (3) a characterization of preservation of policy adherence under refinement by deriving the semantics $[\![S_2]\!]$ of the refined system specification from the policy specifications P_1 and P_2 and the system specification S_1. The formula (7) above describes how the properties of modularity can be utilized to obtain the refined system specification using syntactic sequence diagram operators. Since the instantiations of (7) depend only on the policy specifications P_1 and P_2, as well as the system specification S_1, we can introduce the construct op as syntactic sugar to define the following abbreviation.

$$\mathsf{op}(P_1, P_2, S_1) \overset{\mathsf{def}}{=} \mathsf{refuse}(d_1 \;\mathsf{par}\; \mathsf{any}) \;\mathsf{alt}\; \ldots \;\mathsf{alt}\; \mathsf{refuse}(d_m \;\mathsf{par}\; \mathsf{any})$$

$S_2 = \mathsf{op}(P_1, P_2, S_1) \;\mathsf{alt}\; S_1$ is then a syntactic characterization of the condition for adherence preserving refinement that were given by the semantic characterization $[\![S_2]\!] = [\![S_1]\!]^{\Delta(P_1,P_2,S_1)} \uplus [\![S_1]\!]$ in expression (3) of Section 6.1. The instantiation of expression (1) of Section 2 given the strategies described by expression (4), (5) and (6) above can then be given as follows.

$$\frac{S_2 = \mathsf{op}(P_1, P_2, S_1) \;\mathsf{alt}\; S_1}{P_1 \rightarrow_a S_1 \wedge P_1 \rightsquigarrow P_2 \Rightarrow P_2 \rightarrow_a S_2} \tag{8}$$

Since $S_1 \rightsquigarrow \mathsf{op}(P_1, P_2, S_1)$ alt S_1 holds by definition, the instantiation of $S_1 \rightsquigarrow S_2$ of expression (1) is redundant and omitted in (8).

The above strategies for preservation of adherence under refinement of policy rules assume that the policy triggers are fixed under refinement. This is crucial for adherence preservation since it means that the abstract system specification has been checked for adherence with respect to these triggers, and the result can be preserved under refinement. For policy rules (dm, d_t, d_b) that are added under policy refinement such that the trigger d_t did not occur in the abstract policy specification, adherence of a system specification S to (dm, d_t, d_b) is guaranteed by eliminating all the traces in S that fulfill traces in d_t. This is because a rule that is not triggered by a system specification is trivially adhered to, as captured by the following theorem.

Theorem 15. $\{(dm, d_t, d_b)\} \rightarrow_a \mathsf{refuse}(d_t\ \mathsf{par\ any})\ \mathsf{alt}\ S$

During the development of a system specification S, such a removal of behavior to which a rule (dm, d_t, d_b) applies may be adequate if it is decided that the particular behavior described by the diagram d_t is unimportant, irrelevant or dispensable for the desired functionality of the system under development. If the removal of the behavior is unacceptable, the rule (dm, d_t, d_b) yields a proof obligation that must be solved at the refined level since adherence to this rule has not been checked at the abstract level. In case a policy breach is identified, then either the system traces fulfilling d_t must be removed, or the system development must undergo backtracking and redesign in order to establish adherence.

7 Related Work

Model Driven Architecture (MDA) [9, 22] is an approach to system development based on specifying high-level, platform independent models of the systems using languages such as the UML that are automatically transformed into a resulting system architecture. In [4], the authors observe that model building is also applied in security modeling and policy specification, but claim that the integration of security models into the overall model-based system development process is problematic. This is partly, they claim, because security models and system models are typically disjoint, their integration is poorly understood, and their integration is inadequately supported by methods and tools. Partly this is also because security typically is integrated into systems in a post hoc manner. To remedy this, a specialization of MDA called Model Driven Security is presented in [4] in which security properties of the target system is specified as part of the system model, and later transformed into security mechanisms of the system. A UML-based language called SecureUML is presented that supports the specification of access control requirements in the setting of Role-Based Access Control [8]. These security models may then be combined with system design models in languages such as UML class diagrams and UML state charts where elements of the design models are identified as resources that should be protected according to the SecureUML specification. The paper addresses only access control in the form of permission rules that specify authorizations, however, and the specification of other security aspects is pointed out as a direction for future work. Refinement is also not addressed, but the authors state that by considering diagrams such as UML sequence diagrams and UML use case diagrams, the modeling of the system from different views and at different abstraction levels would be supported.

In [31] we explained that permission rules, and therefore policies, specify trace-set properties that must be satisfied by systems to which the policy specification applies. A trace-set property can only be falsified on sets of traces, unlike trace properties that can be falsified on single traces. Trace-set properties rely on non-determinism, which is captured by a set of system traces each of which represents a valid system execution. This form of non-determinism is often referred to as underspecification. The standard notion of refinement is defined as reduction of underspecification. Unlike trace properties such as safety and liveness which are preserved under refinement [1], trace-set properties are generally not. The latter was shown in [16] for certain security properties.

The results of preservation of policy adherence under refinement of system specifications using STAIRS that we showed in [31] rely on the expressiveness of STAIRS to distinguish between two types of non-determinism, namely underspecification and inherent non-determinism. The latter is a form of non-determinism that the system is required to possess, e.g. to satisfy permission rules, and is preserved under refinement in STAIRS. The development case addressed in this paper in which both the policy specification and the system specification may undergo refinement also rely on this distinction. In the following we discuss related work with particular focus on the expressiveness to capture trace-set properties and preserve these under refinement.

Message sequence charts (MSCs) [15] is an ITU recommendation which to a large extent has been adopted and extended by the UML sequence diagram notation. The distinction between underspecification and inherent non-determinism is, however, beyond the standard MSC language and its semantics [14]. MSCs are furthermore not supported by a well-defined notion of refinement, which means that there is no support for capturing trace-set properties and preserving these under the development process.

In [20], a variant of MSCs is given a formal semantics and provided a formal notion of refinement. Four different interpretations of MSCs are proposed, namely existential, universal, exact, and negative. The existential interpretation requires the fulfillment of the MSC in question by at least one system execution; the universal interpretation requires the fulfillment of the MSC in all executions; the exact interpretation is a strengthening of the universal interpretation by explicitly prohibiting behaviors other than the ones specified by the MSC in question; the negative interpretation requires that no execution is allowed to fulfill the MSC. The interesting interpretation in our context is the existential scenario as it may capture trace-set properties and permissions. As shown in [20], a system that fulfills an MSC specification under the universal, exact, or negative interpretation also fulfills specifications that are refinements of the initial specification where refinement is defined as reduction of underspecification. This is, however, not the case for the existential interpretation, which means that trace-set properties are not preserved under refinement.

The trace semantics for UML 2.0 interactions presented in [36] represents a sequence diagram by a pair of a set of positive traces and a set of negative traces. The approach provides no support for distinguishing between underspecification and inherent non-determinism. Rather, positive traces are interpreted as necessary, i.e. must be possible in an implementation, whereas negative traces are interpreted as forbidden, i.e. must not be possible in an implementation. A notion of refinement is introduced in which previously inconclusive traces may be redefined as positive or negative. With this approach, there is no guarantee that adherence to trace-set properties is preserved under refinement.

Live sequence charts (LSCs) [6, 11] extend MSCs and are particularly directed towards specifying liveness properties. LSCs support the specification of two types of diagrams, namely existential and universal. An existential diagram describes an example scenario that must be satisfied by at least one system run, whereas a universal diagram describes a scenario that must be satisfied by all system runs. Universal charts can furthermore be specified as conditional scenarios by the specification of a prechart that, if successfully executed by a system run, requires the fulfillment of the scenario described in the chart body. LSCs moreover have the expressiveness to specify forbidden scenarios. LSCs seem to have the expressiveness to capture trace-set properties by the use of existential diagrams; obviously, falsification of system satisfaction of requirements expressed by an existential diagram cannot be done on a single trace. However, a system development in LSCs is intended to undergo a shift from an existential view in the initial phases to a universal view in later stages as knowledge of the system evolves. Such a development process with LSCs will generally not preserve trace-set properties. Moving from an existential view to a universal view can be understood as a form of refinement, but LSCs are not supported by a well-defined notion of refinement.

Modal sequence diagrams (MSDs) [10] are defined as a UML 2.0 profile. The notation is an extension of UML sequence diagrams, and is based on the universal/existential distinction of LSCs. The semantics for MSDs is basically the same as for LSCs, and so is the problem of capturing trace-set properties and preserving these under refinement.

Triggered message sequence charts (TMSCs) [32] allow the specification of conditional scenarios and are supported by a formal notion of refinement. In addition to composition operators such as sequential composition, parallel composition and recursion, TMSCs are supported by two different choice operators, namely delayed choice and internal choice. Internal choice is similar to underspecification in the sense that an implementation that offers only at least one of the choices correctly fulfills the specification. For delayed choice, however, the implementation must offer all choices as potential behavior. It is observed in [32] that the trace semantics of MSCs does not have the expressiveness to distinguish between such optional and required behavior, which means that a heterogeneous mix of these in the specification is not supported. The semantics of TMSCs makes this distinction, and it is also preserved under refinement. The paper stresses the importance of preserving properties such as safety and liveness under refinement, but the problem of capturing trace-set properties and preserving these under refinement is not discussed. A further shortcoming is that support for the specification of negative or prohibited behavior is not provided.

As mentioned in the introduction, the problem of policy refinement is poorly explored in the research on policy-based management. The investigation of policy refinement has gained interest only recently [5], and the literature on the issue is still scarce.

One aspect of policy refinement as proposed in [23] is that of goal refinement, where the set of low-level goals derived from a high-level goal intends to fulfill the latter. Goal refinement has been further elaborated within the area of requirements engineering and has served as basis for more recent approaches to policy refinement. The KAOS method [7] is based on identifying refinement patterns, where each pattern represents a possible decomposition of a high-level goal into a set of low level-goals. The fulfillment of the low-level goals of a pattern ensures the fulfillment of the high-level goal.

Goal refinement and the KAOS method have been adopted by several approaches to policy refinement [2, 3, 25, 26, 27]. The approach presented in [2, 3] identifies the policy refinement problem as composed of two parts. The first part is the refinement of high-level goals into operations supported by the concrete objects/devices, such that when performed will achieve the high-level goal. The proposed solution to this problem is to combine the KAOS goal elaboration method with techniques for deriving mechanisms by which a given system can achieve a particular goal. The second part is the refinement of abstract entities into concrete objects/devices. The proposed solution to this problem is a formal representation of object relationships based on domain hierarchies with rules for deducing concrete objects/devices for abstract ones. The formalism of the approach is implemented in event calculus [19] which is held as suitable as it allows formal reasoning and fits the event-driven nature of the systems that are addressed by the approach.

The policy refinement framework presented in [27] also uses the KAOS method for goal elaboration to derive low-level goals from high-level ones. The strategy for identifying the system execution traces that fulfill the low-level goals is, however, based on linear temporal logic and model checking. The work in [26] extends [27] by introducing a mechanism for abstracting policies from system trace executions in a systematic manner.

These approaches to policy refinement based on goal refinement focus on the problem of deriving low-level policies the enforcement of which ensures the fulfillment of the initial high-level goals. Policy adherence therefore guarantees adherence to the abstract specifications, but the problem of preservation of adherence under refinement is not discussed. The problem of integrating requirements from policy specifications with system specifications, or understanding the relation between policy specifications and system specifications where both may be represented at various abstraction levels, is also not addressed.

8 Conclusion

In this paper we have addressed the problem of preservation of properties under refinement in a setting of policy-based management. In particular we have addressed the problem of adherence preservation under refinement for development cases in which both a policy specification and a system specification to which the policy applies may undergo refinement.

The policies are specified using Deontic STAIRS, and the systems are specified using STAIRS. Both approaches allow specifications at various levels of abstraction and are supported by refinement relations that are transitive and modular. Abstraction and refinement are desirable development features since they allow details that are irrelevant or unimportant at a certain development stage and from a certain viewpoint to be ignored. Abstraction is desirable also because analysis of specifications generally is easier and cheaper at abstract levels, as are detection and correction of errors. For analysis to be meaningful at abstract levels, however, the analysis results must be preserved under refinement.

The work presented in [31] demonstrates that adherence of STAIRS system specifications is preserved under refinement of system specifications when the policy specification is fixed. Adherence is generally not preserved under refinement of policy specifications, but in this paper we have presented strategies for identifying system refinements on the basis of policy refinements such that adherence is guaranteed in the result.

The strategies utilize properties of modularity of both policy adherence and policy refinement, as well as the property of transitivity of refinement of policy specifications and system specifications. This means that the problem of preservation of analysis results under refinement can be broken down into manageable sub-problems, and that the strategies are applicable also if only a subset of the policy specification is refined.

In development case (a) depicted to the left in Fig. 1 of Section 2, the property of preservation of adherence under refinement of system specifications is ensured by the fact that underspecification is reduced while the required potential choices of behavior are preserved under refinement. This means that behavior that represents a policy breach cannot be introduced to the system specification under refinement. In this development case, the requirements imposed by the policy specification are taken into account from the very beginning of and throughout the system development process. If the policy were taken into account only at later development phases or during implementation, this could in the worst case result in a system that do not have the desired functionality, thus wasting the development efforts [21].

The potential pitfall of having to backtrack and repeat parts of the system development process is, however, present in the combined development process depicted as case (b) in Fig. 1. The results of preservation of adherence are based on removing traces from the system specification that have become inadmissible after a step of policy refinement. Since some the requirements imposed by the refined policy specification were not taken into account by the abstract system specification, it may be that the refinement of the system specification requires removal of traces that are critical for the desired system functionality. The results of this paper are nevertheless a step towards establishing methods for an integrated process of policy and system development that are supported by modeling languages, precise notions of refinement, and results of adherence preservation. A mature method should be supported by pragmatic development guidelines and tools to assist developers. Strategies for how to avoid backtracking and redesign should also be developed. The development of such guidelines, strategies and tools are directions for future work.

A further topic for future work is to investigate our approach to adherence preserving refinement with respect to completeness. A policy specification can be refined in various ways, and in this paper we have focused on refinement of individual policy rules by reduction of underspecification. Completeness of a method for adherence preserving refinement means that the method offers development rules that ensure preservation of adherence for all valid refinements.

Understanding and formalizing the relation between policy specifications and system specifications are important and beneficial as it allows the requirements imposed by the policy to be taken into account from the very initial phases of the development process. Since Deontic STAIRS is a generic approach to policy-based management, requirements with respect to various issues such as security, trust, service level and access control can be integrated with the system development process.

Acknowledgments. The research on which this paper reports has partly been funded by the Research Council of Norway through the ENFORCE project (164382/V30).

References

[1] B. Alpern and F. B. Schneider. Defining liveness. *Information Processing Letters*, 21(4):181–185, 1985.

[2] A. K. Bandara, E. C. Lupu, J. Moffet, and A. Russo. A goal-based approach to policy refinement. In *Proceedings of the 5th International Workshop on Policies for Distributed Systems and Networks (POLICY'04)*, pages 229–239. IEEE Computer Society, 2004.

[3] A. K. Bandara, E. C. Lupu, A. Russo, N. Dulay, M. Sloman, P. Flegkas, M. Charalambides, and G. Pavlou. Policy refinement for DiffServ quality of service management. In *Proceedings of the 9th IFIP/IEEE International Symposium on Integrated Network Management (IM'05)*, pages 469–482, 2005.

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

[5] R. Boutaba and I. Aib. Policy-based management: A historical perspective. *Journal of Network and Systems Management*, 15(4):447–480, 2007.

[6] W. Damm and D. Harel. LSCs: Breathing life into message sequence charts. *Formal Methods in System Design*, 19(1):45–80, 2001.

[7] R. Darimont and A. van Lamsweerde. Formal refinement patterns for goal-driven requirements elaboration. In *Proceedings of the 4th ACM Symposium on the Foundations of Software Engineering (FSE4)*, pages 179–190, 1996.

[8] D. Ferraiolo and R. Kuhn. Role-based access controls. In *Proceedings of the 15th NIST-NCSC National Computer Security Conference*, pages 554–563, 1992.

[9] D. S. Frankel. *Model Driven Architecture: Applying MDA to Enterprise Computing.* John Wiley & Sons, 2003.

[10] D. Harel and S. Maoz. Assert and negate revisited: Modal semantics for UML sequence diagrams. *Software and Systems Modeling*, 7(2):237–252, 2007.

[11] D. Harel and R. Marelly. *Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine.* Springer, 2003.

[12] Ø. Haugen, K. E. Husa, R. K. Runde, and K. Stølen. STAIRS towards formal design with sequence diagrams. *Software and Systems Modeling*, 4(4):355–367, 2005.

[13] Ø. Haugen, K. E. Husa, R. K. Runde, and K. Stølen. Why timed sequence diagrams require three-event semantics. Technical Report 309, Department of Informatics, University of Oslo, 2006.

[14] International Telecommunication Union. *Recommendation Z.120 Annex B – Semantics of Message Sequence Chart (MSC)*, 1998.

[15] International Telecommunication Union. *Recommendation Z.120 – Message Sequence Chart (MSC)*, 2004.

[16] J. Jacob. On the derivation of secure components. In *Proceedings of the IEEE Symposium on Security and Privacy (SP'89)*, pages 242–247. IEEE Computer Society, 1989.

[17] J. Jürjens. UMLsec: Extending UML for secure systems development. In *Proceedings of the 5th International Conference on the Unified Modeling Language (UML'02)*, volume 2460 of *LNCS*, pages 412–425. Springer, 2002.

[18] J. Jürjens. *Secure Systems Development with UML.* Springer, 2005.

[19] R. A. Kowalski and M. J. Sergot. A logic-based calculus of events. *New Generation Computing*, 4(1):67–95, 1986.

[20] I. H. Krüger. *Distributed System Design with Message Sequence Charts.* PhD thesis, Institut für Informatik, Ludwig-Maximilians-Universität München, July 2000.

[21] H. Mantel. Preserving information flow properties under refinement. In *Proceedings of the IEEE Symposium on Security and Privacy (SP'01)*, pages 78–91. IEEE Computer Society, 2001.

[22] J. Miller and J. Mukerji. MDA guide. Object Management Group, document: omg/2003-06-01, 2003.

[23] J. D. Moffett and M. S. Sloman. Policy hierarchies for distributed systems management. *IEEE Journal on Selected Areas in Communications*, 11:1404–1414, 1993.

[24] Object Management Group. *Unified Modeling Language: Superstructure, version 2.1.1*, 2007.

[25] J. Rubio-Loyola. *A Methodological Approach to Policy Refinement in Policy-based Management Systems.* PhD thesis, Departament de Teoria del Senyal i Comunicacions, Universitat Politècnica de Catalunya, 2007.

[26] J. Rubio-Loyola, J. Serrat, M. Charalambides, P. Flegkas, and G. Pavlou. A functional solution for goal-oriented policy refinement. In *Proceedings of the 7th International Workshop on Policies for Distributed Systems and Networks (POLICY'06)*, pages 133–144. IEEE Computer Society, 2006.

[27] J. Rubio-Loyola, J. Serrat, M. Charalambides, P. Flegkas, G. Pavlou, and A. L. Lafuente. Using linear temporal model checking for goal-oriented policy refinement frameworks. In *Proceedings of the 6th International Workshop on Policies for Distributed Systems and Networks (POLICY'05)*, pages 181–190. IEEE Computer Society, 2005.

[28] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual.* Addison Wesley, second edition, 2005.

[29] R. K. Runde, Ø. Haugen, and K. Stølen. How to transform UML neg into a useful construct. In *Proceedings of Norsk Informatikkonferanse (NIK'05)*, pages 55–66. Tapir, 2005.

[30] R. K. Runde, Ø. Haugen, and K. Stølen. Refining UML interactions with underspecification and nondeterminism. *Nordic Journal of Computing*, 12(2):157–188, 2005.

[31] F. Seehusen, B. Solhaug, and K. Stølen. Adherence preserving refinement of trace-set properties in STAIRS: Exemplified for information flow properties and policies. *Software and Systems Modeling*, 8(1):46–65, 2009.

[32] B. Sengupta and R. Cleaveland. Triggered message sequence charts. *IEEE Transactions on Software Engineering*, 32(8):587–607, 2006.

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

[34] M. Sloman and E. Lupu. Security and management policy specification. *Network, IEEE*, 16(2):10–19, 2002.

[35] B. Solhaug and K. Stølen. Compositional refinement of policies in UML – Exemplified for access control. In *Proceedings of the 13th European Symposium on Research in Computer Security (ESORICS'08)*, volume 5283 of *LNCS*, pages 300–316. Springer, 2008.

[36] H. Störrle. Trace semantics of interactions in UML 2.0. Technical Report TR 0403, University of Munich, 2004.

[37] J. M. Wing. A specifier's introduction to formal methods. *IEEE Computer*, 23(9):8,10–22,24, 1990.

A Formal Definitions

The functions $\frown$, Ⓢ and Ⓣ are for concatenation of sequences, filtering of sequences and filtering of pairs of sequences, respectively. Concatenation is to glue sequences together, so $h_1 \frown h_2$ is the sequence that equals h_1 if h_1 is infinite. Otherwise it denotes the sequence that has h_1 as prefix and h_2 as suffix, where the length equals the sum of the length of h_1 and h_2. By EⓈh we denote the sequence obtained from the sequence h by removing all elements from h that are not in the set of elements E. For example, $\{1,3\}$ Ⓢ $\langle 1,1,2,1,3,2\rangle = \langle 1,1,1,3\rangle$. The filtering function Ⓣ is described as follows. For any set of pairs of elements F and pair of sequences t, by FⓉt we denote the pair of sequences obtained from t by truncating the longest sequence in t at the length of the shortest sequence in t if the two sequences are of unequal length; for each $j \in \{1,\ldots,k\}$, where k is the length of the shortest sequence in t, selecting or deleting the two elements at index j in the two sequences, depending on whether the pair of these elements is in the set F. For example, we have that $\{(1,f),(1,g)\}$ Ⓣ $(\langle 1,1,2,1,2\rangle, \langle f,f,f,g,g\rangle) = (\langle 1,1,1\rangle, \langle f,f,g\rangle)$. See [31] for the formal definitions.

Definition 11. Parallel composition.

$$H_1 \parallel H_2 \overset{\mathsf{def}}{=} \{h \in \mathcal{H} \mid \exists s \in \{1,2\}^\infty : \pi_2((\{1\} \times \mathcal{E}) \text{ Ⓣ } (s,h)) \in H_1 \wedge \pi_2((\{2\} \times \mathcal{E}) \text{ Ⓣ } (s,h)) \in H_2\}$$

Definition 12. Sequential composition.

$$H_1 \succsim H_2 \overset{\mathsf{def}}{=} \{h \in \mathcal{H} \mid \exists h_1 \in H_1, h_2 \in H_2 : \forall l \in \mathcal{L} : e.l\text{Ⓢ}h = e.l\text{Ⓢ}h_1 \frown e.l\text{Ⓢ}h_2\}$$

$\{1,2\}^\infty$ is the set of all infinite sequences over the set $\{1,2\}$, and π_2 is a projection operator returning the second element of a pair. The infinite sequence s in the definition can be understood as an oracle that determines which of the events in h that are filtered away. $\mathcal{E}$ denotes the set of all events, and the expression $e.l$ denotes the set of events that may take place on the lifeline l, where $\mathcal{L}$ denotes the set of all lifelines. The formal definition is as follows.

$$e.l \overset{\mathsf{def}}{=} \{e \in \mathcal{E} \mid (k.e = \,! \wedge tr.e = l) \vee (k.e = \,? \wedge re.e = l)\}$$

An event e is a pair (k,m) of a kind $k \in \{!,?\}$ and a message m, where ! denotes a send event and ? denotes a receive event. A message m is a tuple (s, tr, re) of a signal s, a transmitter tr and a receiver re. Both tr and re are lifelines. The functions

$$k.\_ \in \mathcal{E} \to \{!,?\} \qquad tr.\_ \in \mathcal{E} \to \mathcal{L} \qquad re.\_ \in \mathcal{E} \to \mathcal{L}$$

yield the kind, transmitter and receiver, respectively, of an event.

Definition 13. Sub-trace relation.

$$h_1 \lhd h_2 \overset{\mathsf{def}}{=} \exists s \in \{1,2\}^\infty : \pi_2((\{1\} \times \mathcal{E}) \circledT (s, h_2)) = h_1$$

Definition 14. For a trace set H and traces h and h' we define the following.

$$\begin{aligned} H \lhd h &\overset{\mathsf{def}}{=} \exists h' \in H : h' \lhd h \\ h' \ntriangleleft h &\overset{\mathsf{def}}{=} \neg(h' \lhd h) \\ H \ntriangleleft h &\overset{\mathsf{def}}{=} \neg\exists h' \in H : h' \lhd h \\ H\lhd &\overset{\mathsf{def}}{=} \{h \in \mathcal{H} \mid \exists h' \in H : h' \lhd h\} \end{aligned}$$

Definition 15. Adherence of system specifications S to policy rules r as defined in [31].

$$\begin{aligned} (pe, d_t, d_b) \rightarrow_a' S \overset{\mathsf{def}}{=} {} & \forall (p,n) \in [\![S]\!] : \forall h \in (p \setminus n) : ([\![d_t]\!] \lhd h \Rightarrow \\ & \exists (p', n') \in [\![S]\!] : \forall h' \in (p' \setminus n') : ([\![d_t]\!] \succsim [\![d_b]\!]) \lhd h') \\ (ob, d_t, d_b) \rightarrow_a' S \overset{\mathsf{def}}{=} {} & \forall (p,n) \in [\![S]\!] : \forall h \in (p \setminus n) : [\![d_t]\!] \lhd h \Rightarrow ([\![d_t]\!] \succsim [\![d_b]\!]) \lhd h \\ (pr, d_t, d_b) \rightarrow_a' S \overset{\mathsf{def}}{=} {} & \forall (p,n) \in [\![S]\!] : \forall h \in (p \setminus n) : [\![d_t]\!] \lhd h \Rightarrow ([\![d_t]\!] \succsim [\![d_b]\!]) \ntriangleleft h \end{aligned}$$

Definition 16. Adherence of system specification S to policy specification P as defined in [31].

$$P \rightarrow_a' S \overset{\mathsf{def}}{=} \forall r \in P : r \rightarrow_a' S$$

Definition 17. Refinement of policy rules.

$$\begin{aligned} (pe, d_t, d_b) \rightsquigarrow' (pe, d_t, d_b') &\overset{\mathsf{def}}{=} [\![d_b']\!] \subseteq [\![d_b]\!] \\ (ob, d_t, d_b) \rightsquigarrow' (ob, d_t, d_b') &\overset{\mathsf{def}}{=} [\![d_b']\!] \subseteq [\![d_b]\!] \\ (pr, d_t, d_b) \rightsquigarrow' (pr, d_t, d_b') &\overset{\mathsf{def}}{=} [\![d_b']\!] \supseteq [\![d_b]\!] \end{aligned}$$

Definition 18. Refinement of policy specifications.

$$P_1 \rightsquigarrow' P_2 \overset{\mathsf{def}}{=} \forall r_1 \in P_1 : \exists r_2 \in P_2 : r_1 \rightsquigarrow' r_2$$

Notice that by Def. 18, policy rules are allowed to be added under policy refinement. Refinement of a policy rule (dm, d_t, d_b) by weakening the trigger can then be achieved by adding a rule (dm, d_t', d_b).

B Proofs

Theorem 2. $P \rightarrow_a S \Leftrightarrow P \rightarrow'_a S$

Proof. Each direction of the biconditional is proved separately below.

Assume: $P \rightarrow_a S$
Prove: $P \rightarrow'_a S$
$\langle 1\rangle 1.$ Case: $P = \emptyset$
  Proof: Immediately from Def. 16
$\langle 1\rangle 2.$ Case: $P \neq \emptyset$
  $\langle 2\rangle 1.$ Choose arbitrary $(dm, d_t, d_b) \in P$, $dm \in \{pe, ob, pr\}$
    Proof: The rule exists by case assumption
  $\langle 2\rangle 2.$ $(dm, d_t, d_b) \rightarrow'_a S$
    $\langle 3\rangle 1.$ Case: $dm = pe$
      $\langle 4\rangle 1.$ Case: $\forall (p,n) \in [\![S]\!] : \forall h \in (p \setminus n) : [\![d_t]\!] \not\lhd h$
        Proof: Immediate from Def. 15
      $\langle 4\rangle 2.$ Case: $\exists (p,n) \in [\![S]\!] : \exists h \in (p \setminus n) : [\![d_t]\!] \lhd h$
        $\langle 5\rangle 1.$ Choose arbitrary $(p,n) \in [\![S]\!]$ and $h \in (p \setminus n)$ such that $[\![d_t]\!] \lhd h$
          Proof: The interaction obligation and trace exist by case assumption
        $\langle 5\rangle 2.$ Let: $[\![P]\!](h) = (a, u, C)$
        $\langle 5\rangle 3.$ $\exists (p', n') \in [\![S]\!] : \forall h' \in (p' \setminus n') : ([\![d_t]\!] \succsim [\![d_b]\!]) \lhd h'$
          $\langle 6\rangle 1.$ $([\![d_t]\!] \succsim [\![d_b]\!])\lhd \in C$
            Proof: $\langle 2\rangle 1$, $\langle 3\rangle 1$, $\langle 5\rangle 1$, Def. 1 and Def. 3
          $\langle 6\rangle 2.$ $\exists (p', n') \in [\![S]\!] : \forall h' \in (p' \setminus n') : h' \in ([\![d_t]\!] \succsim [\![d_b]\!])\lhd$
            Proof: $\langle 6\rangle 1$, proof assumption and Def. 5
          $\langle 6\rangle 3.$ Q.E.D.
            Proof: $\langle 6\rangle 2$ and Def. 14
        $\langle 5\rangle 4.$ Q.E.D.
          Proof: $\langle 5\rangle 1$, $\langle 5\rangle 3$ and Def. 15
      $\langle 4\rangle 3.$ Q.E.D.
        Proof: The cases are exhaustive
    $\langle 3\rangle 2.$ Case: $dm = ob$
      $\langle 4\rangle 1.$ Case: $\forall (p,n) \in [\![S]\!] : \forall h \in (p \setminus n) : [\![d_t]\!] \not\lhd h$
        Proof: Immediate from Def. 15
      $\langle 4\rangle 2.$ Case: $\exists (p,n) \in [\![S]\!] : \exists h \in (p \setminus n) : [\![d_t]\!] \lhd h$
        $\langle 5\rangle 1.$ Choose arbitrary $(p,n) \in [\![S]\!]$ and $h \in (p \setminus n)$ such that $[\![d_t]\!] \lhd h$
          Proof: The interaction obligation and trace exist by case assumption
        $\langle 5\rangle 2.$ Let: $[\![P]\!](h) = (a, u, C)$
        $\langle 5\rangle 3.$ $([\![d_t]\!] \succsim [\![d_b]\!]) \lhd h$
          $\langle 6\rangle 1.$ $a \subseteq ([\![d_t]\!] \succsim [\![d_b]\!])\lhd$
            Proof: $\langle 2\rangle 1$, $\langle 3\rangle 2$, $\langle 5\rangle 1$, Def. 1 and Def. 3
          $\langle 6\rangle 2.$ $h \in a$

Proof: ⟨6⟩1, proof assumption and Def. 5

⟨6⟩3. Q.E.D.

Proof: ⟨6⟩1, ⟨6⟩2 and Def. 14

⟨5⟩4. Q.E.D.

Proof: ⟨5⟩1, ⟨5⟩3 and Def. 15

⟨4⟩3. Q.E.D.

Proof: The cases are exhaustive

⟨3⟩3. Case: $dm = pr$

⟨4⟩1. Case: $\forall (p, n) \in [\![S]\!] : \forall h \in (p \setminus n) : [\![d_t]\!] \not\lhd h$

Proof: Immediate from Def. 15

⟨4⟩2. Case: $\exists (p, n) \in [\![S]\!] : \exists h \in (p \setminus n) : [\![d_t]\!] \lhd h$

⟨5⟩1. Choose arbitrary $(p, n) \in [\![S]\!]$ and $h \in (p \setminus n)$ such that $[\![d_t]\!] \lhd h$

Proof: The interaction obligation and trace exist by case assumption

⟨5⟩2. Let: $[\![P]\!](h) = (a, u, C)$

⟨5⟩3. $([\![d_t]\!] \lhd [\![d_b]\!]) \not\lhd h$

⟨6⟩1. $([\![d_t]\!] \succsim [\![d_b]\!])_{\lhd} \subseteq u$

Proof: ⟨2⟩1, ⟨3⟩3, ⟨5⟩1, Def. 1 and Def. 3

⟨6⟩2. $h \notin u$

Proof: ⟨6⟩1, proof assumption and Def. 5

⟨6⟩3. Q.E.D.

Proof: ⟨6⟩1, ⟨6⟩2 and Def. 14

⟨5⟩4. Q.E.D.

Proof: ⟨5⟩1, ⟨5⟩3 and Def. 15

⟨4⟩3. Q.E.D.

Proof: The cases are exhaustive

⟨2⟩3. Q.E.D.

Proof: ⟨2⟩1, ⟨2⟩2 and Def. 16

⟨1⟩3. Q.E.D.

Proof: The cases are exhaustive

ASSUME: $P \rightarrow'_a S$
PROVE: $P \rightarrow_a S$
$\langle 1 \rangle$1. CASE: $P = \emptyset$
  PROOF: Immediately from Def. 5 since $\forall h \in \mathcal{H} : [\![P]\!](h) = (\mathcal{H}, \emptyset, \emptyset)$
$\langle 1 \rangle$2. CASE: $P \neq \emptyset$
  $\langle 2 \rangle$1. Choose arbitrary $(p, n) \in [\![S]\!]$
    PROOF: The semantics of any sequence diagram is non-empty
  $\langle 2 \rangle$2. $\forall h \in (p \setminus n)$:
      $h \in a \wedge h \notin u \wedge \forall H \in C : \exists (p', n') \in [\![S]\!] : \forall h' \in (p' \setminus n') : h' \in H$
      where $[\![P]\!](h) = (a, u, C)$
    $\langle 3 \rangle$1. CASE: $(p \setminus n) = \emptyset$
      PROOF: Trivial
    $\langle 3 \rangle$2. CASE: $(p \setminus n) \neq \emptyset$
      $\langle 4 \rangle$1. Choose arbitrary $h \in (p \setminus n)$
        PROOF: The trace exists by case assumption
      $\langle 4 \rangle$2. LET: $[\![P]\!](h) = (a, u, C)$
      $\langle 4 \rangle$3. $h \in a \wedge h \notin u \wedge \forall H \in C : \exists (p', n') \in [\![S]\!] : \forall h' \in (p' \setminus n') : h' \in H$
        $\langle 5 \rangle$1. $h \in a$
          $\langle 6 \rangle$1. CASE: $a = \mathcal{H}$
            PROOF: Trivial
          $\langle 6 \rangle$2. CASE: $a \neq \mathcal{H}$
            $\langle 7 \rangle$1. $a = \bigcap \{([\![d_t]\!] \succsim [\![d_b]\!])\lhd \mid (ob, d_t, d_b) \in P \wedge [\![d_t]\!] \lhd h\}$
              PROOF: Case assumption, Def. 1 and Def. 3
            $\langle 7 \rangle$2. $\forall (ob, d_t, d_b) \in P : [\![d_t]\!] \lhd h \Rightarrow h \in ([\![d_t]\!] \succsim [\![d_b]\!])\lhd$
              PROOF: Proof assumption, and Def. 16, Def. 15 and Def, 14
            $\langle 7 \rangle$3. Q.E.D.
              PROOF: $\langle 7 \rangle$1 and $\langle 7 \rangle$2
          $\langle 6 \rangle$3. Q.E.D.
            PROOF: The cases are exhaustive
        $\langle 5 \rangle$2. $h \notin u$
          $\langle 6 \rangle$1. CASE: $u = \emptyset$
            PROOF: Trivial
          $\langle 6 \rangle$2. CASE: $u \neq \emptyset$
            $\langle 7 \rangle$1. $u = \bigcup \{([\![d_t]\!] \succsim [\![d_b]\!])\lhd \mid (pr, d_t, d_b) \in P \wedge [\![d_t]\!] \lhd h\}$
              PROOF: Case assumption, Def. 1 and Def. 3
            $\langle 7 \rangle$2. $\forall (pr, d_t, d_b) \in P : [\![d_t]\!] \lhd h \Rightarrow h \notin ([\![d_t]\!] \succsim [\![d_b]\!])\lhd$
              PROOF: Proof assumption, and Def. 16, Def. 15 and Def. 14
            $\langle 7 \rangle$3. Q.E.D.
              PROOF: $\langle 7 \rangle$1 and $\langle 7 \rangle$2
          $\langle 6 \rangle$3. Q.E.D.
            PROOF: The cases are exhaustive
        $\langle 5 \rangle$3. $\forall H \in C : \exists (p', n') \in [\![S]\!] : \forall h' \in (p' \setminus n') : h' \in H$
          $\langle 6 \rangle$1. CASE: $C = \emptyset$
            PROOF: Trivial

$\langle 6 \rangle 2.$ Case: $C \neq \emptyset$

$\langle 7 \rangle 1.$ Choose arbitrary $H \in C$

Proof: The set exists by case assumption

$\langle 7 \rangle 2.$ $\exists (p', n') \in [\![S]\!] : \forall h' \in (p' \setminus n') : h' \in H$

$\langle 8 \rangle 1.$ $\exists (pe, d_t, d_b) \in P$ such that $[\![d_t]\!] \lhd h$ and $H = ([\![d_t]\!] \precsim [\![d_b]\!])\lhd$

Proof: Case assumption, Def. 1 and Def. 3

$\langle 8 \rangle 2.$ Q.E.D.

Proof: $\langle 8 \rangle 1$, proof assumption, and Def. 16 and Def. 15

$\langle 7 \rangle 3.$ Q.E.D.

Proof: $\langle 7 \rangle 1$ and $\langle 7 \rangle 2$

$\langle 6 \rangle 3.$ Q.E.D.

Proof: The cases are exhaustive

$\langle 5 \rangle 4.$ Q.E.D.

Proof: $\langle 5 \rangle 1$, $\langle 5 \rangle 2$ and $\langle 5 \rangle 3$

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

Proof: $\langle 4 \rangle 1$ and $\langle 4 \rangle 3$

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

Proof: The cases are exhaustive

$\langle 2 \rangle 3.$ Q.E.D.

Proof: $\langle 2 \rangle 1$, $\langle 2 \rangle 2$ and Def. 5

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

Proof: The cases are exhaustive

□

Theorem 3. $P_1 \rightarrow_a S \wedge P_2 \rightarrow_a S \Leftrightarrow (P_1 \cup P_2) \rightarrow_a S$

Proof.

Prove: $P_1 \rightarrow_a S \wedge P_2 \rightarrow_a S \Leftrightarrow (P_1 \cup P_2) \rightarrow_a S$

$\langle 1 \rangle 1.$ $P_1 \rightarrow'_a S \wedge P_2 \rightarrow'_a S \Leftrightarrow (P_1 \cup P_2) \rightarrow'_a S$

Proof: Immediate from Def. 16

$\langle 1 \rangle 2.$ Q.E.D.

Proof: $\langle 1 \rangle 1$ and Theorem 2

□

Theorem 5. $P_1 \rightsquigarrow P_1' \wedge P_2 \rightsquigarrow P_2' \Rightarrow (P_1 \cup P_2) \rightsquigarrow (P_1' \cup P_2')$

Proof.

Assume: 1. $P_1 \rightsquigarrow P_1'$
2. $P_2 \rightsquigarrow P_2'$

Prove: $(P_1 \cup P_2) \rightsquigarrow (P_1' \cup P_2')$

$\langle 1\rangle 1.$ Choose arbitrary $h \in \mathcal{H}$
Proof: $\mathcal{H}$ is non-empty

$\langle 1\rangle 2.$ Let: $[\![P_1]\!](h) = (a_1, u_1, C_1)$
$[\![P_1']\!](h) = (a_1', u_1', C_1')$
$[\![P_2]\!](h) = (a_2, u_2, C_2)$
$[\![P_2']\!](h) = (a_2', u_2', C_2')$
$[\![P_1 \cup P_2]\!](h) = (a, u, C)$
$[\![P_1' \cup P_2']\!](h) = (a', u', C')$

$\langle 1\rangle 3.$ $a' \subseteq a \wedge$
$u \subseteq u' \wedge$
$\forall H \in C : \exists H' \in C' : H' \subseteq H \wedge$
$\forall H' \in C' : \exists H \in C : H' \subseteq H$

$\langle 2\rangle 1.$ $a' \subseteq a$
$\langle 3\rangle 1.$ $a_1' \subseteq a_1$
Proof: Assumption 1 and Def. 7
$\langle 3\rangle 2.$ $a_2' \subseteq a_2$
Proof: Assumption 2 and Def. 7
$\langle 3\rangle 3.$ $a = a_1 \cap a_2$
Proof: Def. 3
$\langle 3\rangle 4.$ $a' = a_1' \cap a_2'$
Proof: Def. 3
$\langle 3\rangle 5.$ Q.E.D.
Proof: $\langle 3\rangle 1$, $\langle 3\rangle 2$, $\langle 3\rangle 3$ and $\langle 3\rangle 4$

$\langle 2\rangle 2.$ $u \subseteq u'$
$\langle 3\rangle 1.$ $u_1 \subseteq u_1'$
Proof: Assumption 1 and Def. 7
$\langle 3\rangle 2.$ $u_2 \subseteq u_2'$
Proof: Assumption 2 and Def. 7
$\langle 3\rangle 3.$ $u = u_1 \cup u_2$
Proof: Def. 3
$\langle 3\rangle 4.$ $u' = u_1' \cup u_2'$
Proof: Def. 3
$\langle 3\rangle 5.$ Q.E.D.
Proof: $\langle 3\rangle 1$, $\langle 3\rangle 2$, $\langle 3\rangle 3$ and $\langle 3\rangle 4$

$\langle 2\rangle 3.$ $\forall H \in C : \exists H' \in C' : H' \subseteq H$
$\langle 3\rangle 1.$ Case: $C = \emptyset$
Proof: Trivial
$\langle 3\rangle 2.$ Case: $C \neq \emptyset$

```
            ⟨4⟩1. Choose arbitrary $H \in C$
              PROOF: The trace set exists by case assumption
            ⟨4⟩2. $\exists H' \in C' : H' \subseteq H$
              ⟨5⟩1. $H \in C_1 \cup C_2$
                PROOF: Def. 3
              ⟨5⟩2. CASE: $H \in C_1$
                ⟨6⟩1. $\exists H' \in C'_1 : H' \subseteq H$
                  PROOF: Assumption 1 and Def. 7
                ⟨6⟩2. $C'_1 \subseteq C'$
                  PROOF: Def. 3
                ⟨6⟩3. Q.E.D.
                  PROOF: ⟨6⟩1 and ⟨6⟩2
              ⟨5⟩3. CASE: $H \in C_2$
                PROOF: Symmetric to ⟨5⟩2
              ⟨5⟩4. Q.E.D.
                PROOF: ⟨5⟩1, ⟨5⟩2 and ⟨5⟩3
            ⟨4⟩3. Q.E.D.
              PROOF: ⟨4⟩1 and ⟨4⟩2
          ⟨3⟩3. Q.E.D.
            PROOF: The cases are exhaustive
        ⟨2⟩4. $\forall H' \in C' : \exists H \in C : H' \subseteq H$
          ⟨3⟩1. CASE: $C' = \emptyset$
            PROOF: Trivial
          ⟨3⟩2. CASE: $C' \neq \emptyset$
            ⟨4⟩1. Choose arbitrary $H' \in C'$
              PROOF: The trace set exists by case assumption
            ⟨4⟩2. $\exists H \in C : H' \subseteq H$
              ⟨5⟩1. $H' \in C'_1 \cup C'_2$
                PROOF: Def. 3
              ⟨5⟩2. CASE: $H' \in C'_1$
                ⟨6⟩1. $\exists H \in C_1 : H' \subseteq H$
                  PROOF: Assumption 1 and Def. 7
                ⟨6⟩2. $C_1 \subseteq C$
                  PROOF: Def. 3
                ⟨6⟩3. Q.E.D.
                  PROOF: ⟨6⟩1 and ⟨6⟩2
              ⟨5⟩3. CASE: $H' \in C'_2$
                PROOF: Symmetric to ⟨5⟩2
              ⟨5⟩4. Q.E.D.
                PROOF: ⟨5⟩1, ⟨5⟩2 and ⟨5⟩3
            ⟨4⟩3. Q.E.D.
              PROOF: ⟨4⟩1 and ⟨4⟩2
          ⟨3⟩3. Q.E.D.
            PROOF: The cases are exhaustive
```

$\langle 2\rangle 5$. Q.E.D.

PROOF: $\langle 2\rangle 1$, $\langle 2\rangle 2$, $\langle 2\rangle 3$ and $\langle 2\rangle 4$

$\langle 1\rangle 4$. Q.E.D.

PROOF: $\langle 1\rangle 1$, $\langle 1\rangle 2$, $\langle 1\rangle 3$ and Def. 7

$\square$

Theorem 6. $P_1 \rightsquigarrow P_2 \wedge P_2 \rightarrow_a S \Rightarrow P_1 \rightarrow_a S$

Proof.

ASSUME: 1. $P_1 \rightsquigarrow P_2$
2. $P_2 \rightarrow_a S$

PROVE: $P_1 \rightarrow_a S$

$\langle 1\rangle 1.$ CASE: $P_1 = \emptyset$
PROOF: Immediately from Def. 5 since $\forall h \in \mathcal{H} : [\![P_1]\!](h) = (\mathcal{H}, \emptyset, \emptyset)$

$\langle 1\rangle 2.$ CASE: $P_1 \neq \emptyset$
$\langle 2\rangle 1.$ Choose arbitrary $(p, n) \in [\![S]\!]$
PROOF: The semantics of any sequence diagram is non-empty
$\langle 2\rangle 2.$ $\forall h \in (p \setminus n) :$
$h \in a_1 \wedge h \notin u_1 \wedge \forall H_1 \in C_1 : \exists (p', n') \in [\![S]\!] : \forall h' \in (p' \setminus n') : h' \in H_1$
where $[\![P_1]\!](h) = (a_1, u_1, C_1)$
$\langle 3\rangle 1.$ CASE: $(p \setminus n) = \emptyset$
PROOF: Trivial
$\langle 3\rangle 2.$ CASE: $(p \setminus n) \neq \emptyset$
$\langle 4\rangle 1.$ Choose arbitrary $h \in (p \setminus n)$
PROOF: The trace exists by case assumption
$\langle 4\rangle 2.$ LET: $[\![P_1]\!](h) = (a_1, u_1, C_1)$
$[\![P_2]\!](h) = (a_2, u_2, C_2)$
$\langle 4\rangle 3.$ $h \in a_1 \wedge$
$h \notin u_1 \wedge$
$\forall H_1 \in C_1 : \exists (p', n') \in [\![S]\!] : \forall h' \in (p' \setminus n') : h' \in H_1$
$\langle 5\rangle 1.$ $h \in a_1$
$\langle 6\rangle 1.$ $a_2 \subseteq a_1$
Assumption 1 and Def. 7
$\langle 6\rangle 2.$ $h \in a_2$
PROOF: Assumption 2 and Def. 5
$\langle 6\rangle 3.$ Q.E.D.
PROOF: $\langle 6\rangle 1$ and $\langle 6\rangle 2$
$\langle 5\rangle 2.$ $h \notin u_1$
$\langle 6\rangle 1.$ $u_1 \subseteq u_2$
Assumption 1 and Def. 7
$\langle 6\rangle 2.$ $h \notin u_2$
PROOF: Assumption 2 and Def. 5
$\langle 6\rangle 3.$ Q.E.D.
PROOF: $\langle 6\rangle 1$ and $\langle 6\rangle 2$
$\langle 5\rangle 3.$ $\forall H_1 \in C_1 : \exists (p', n') \in [\![S]\!] : \forall h' \in (p' \setminus n') : h' \in H_1$
$\langle 6\rangle 1.$ CASE: $C_1 = \emptyset$
PROOF: Trivial
$\langle 6\rangle 2.$ CASE: $C_1 \neq \emptyset$
$\langle 7\rangle 1.$ Choose arbitrary $H_1 \in C_1$
PROOF: The trace set exists by case assumption

$\langle 7\rangle 2.$ $\exists (p', n') \in [\![S]\!] : \forall h' \in (p' \setminus n') : h' \in H_1$

$\langle 8\rangle 1.$ Choose $H_2 \in C_2$ such that $H_2 \subseteq H_1$

PROOF: Assumption 1 and Def. 7

$\langle 8\rangle 2.$ $\exists (p', n') \in [\![S]\!] : \forall h' \in (p' \setminus n') : h' \in H_2$

PROOF: $\langle 8\rangle 1$ and assumption 2

$\langle 8\rangle 3.$ Q.E.D.

PROOF: $\langle 8\rangle 1$ and $\langle 8\rangle 2$

$\langle 7\rangle 3.$ Q.E.D.

PROOF: $\langle 7\rangle 1$ and $\langle 7\rangle 2$

$\langle 6\rangle 3.$ Q.E.D.

PROOF: The cases are exhaustive

$\langle 5\rangle 4.$ Q.E.D.

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

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

PROOF: $\langle 4\rangle 1$, $\langle 4\rangle 2$ and $\langle 4\rangle 3$

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

PROOF: The cases are exhaustive

$\langle 2\rangle 3.$ Q.E.D.

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

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

PROOF: The cases are exhaustive

$\square$

Theorem 7. $P_1 \rightsquigarrow' P_2 \wedge P_1 \geqslant P_2 \Rightarrow P_1 \rightsquigarrow P_2$

Proof.

ASSUME: 1. $P_1 \rightsquigarrow' P_2$
  2. $P_1 \geqslant P_2$

PROVE: $P_1 \rightsquigarrow P_2$

$\langle 1 \rangle 1.$ Choose arbitrary $h \in \mathcal{H}$
  PROOF: $\mathcal{H}$ is non-empty

$\langle 1 \rangle 2.$ LET: $[\![P_1]\!](h) = (a_1, u_1, C_1)$
  $[\![P_2]\!](h) = (a_2, u_2, C_2)$

$\langle 1 \rangle 3.$ $a_2 \subseteq a_1 \wedge$
  $u_1 \subseteq u_2 \wedge$
  $\forall H_1 \in C_1 : \exists H_2 \in C_2 : H_2 \subseteq H_1 \wedge$
  $\forall H_2 \in C_2 : \exists H_1 \in C_1 : H_2 \subseteq H_1$

  $\langle 2 \rangle 1.$ $a_2 \subseteq a_1$
    $\langle 3 \rangle 1.$ CASE: $a_1 = \mathcal{H}$
      PROOF: Trivial
    $\langle 3 \rangle 2.$ CASE: $a_1 \neq \mathcal{H}$
      $\langle 4 \rangle 1.$ $a_1 = \bigcap\{([\![d_t]\!] \precsim [\![d_b]\!])\lhd \mid (ob, d_t, d_b) \in P_1 \wedge [\![d_t]\!] \lhd h\}$
        PROOF: Case assumption, Def. 1 and Def. 3
      $\langle 4 \rangle 2.$ $a_2 = \bigcap\{([\![d_t]\!] \precsim [\![d_b]\!])\lhd \mid (ob, d_t, d_b) \in P_2 \wedge [\![d_t]\!] \lhd h\}$
        $\langle 5 \rangle 1.$ $\exists (ob, d_t, d_b) \in P_2 : [\![d_t]\!] \lhd h$
          PROOF: Case assumption, assumption 1, Def. 18 and Def. 17
        $\langle 5 \rangle 2.$ Q.E.D.
          PROOF: $\langle 5 \rangle 1$, Def. 1 and Def. 3
      $\langle 4 \rangle 3.$ $\forall (ob, d_t, d_{b1}) \in P_1 : \exists (ob, d_t, d_{b2}) \in P_2 : [\![d_{b2}]\!] \subseteq [\![d_{b1}]\!]$
        PROOF: Assumption 1, Def. 18 and Def. 17
      $\langle 4 \rangle 4.$ $\forall (ob, d_t, d_{b1}) \in P_1 : \exists (ob, d_t, d_{b2}) \in P_2 :$
        $([\![d_t]\!] \precsim [\![d_{b2}]\!]) \subseteq ([\![d_t]\!] \precsim [\![d_{b1}]\!])$
        PROOF: $\langle 4 \rangle 3$ and Lemma 27 in [13] of monotonicity of $\subseteq$ with respect to $\precsim$
      $\langle 4 \rangle 5.$ Q.E.D.
        PROOF: $\langle 4 \rangle 1$, $\langle 4 \rangle 2$, $\langle 4 \rangle 4$ and Lemma 1
    $\langle 3 \rangle 3.$ Q.E.D.
      PROOF: The cases are exhaustive

  $\langle 2 \rangle 2.$ $u_1 \subseteq u_2$
    $\langle 3 \rangle 1.$ CASE: $u_1 = \emptyset$
      PROOF: Trivial
    $\langle 3 \rangle 2.$ CASE: $u_1 \neq \emptyset$
      $\langle 4 \rangle 1.$ $u_1 = \bigcup\{([\![d_t]\!] \precsim [\![d_b]\!])\lhd \mid (pr, d_t, d_b) \in P_1 \wedge [\![d_t]\!] \lhd h\}$
        PROOF: Case assumption, Def. 1 and Def. 3
      $\langle 4 \rangle 2.$ $u_2 = \bigcup\{([\![d_t]\!] \precsim [\![d_b]\!])\lhd \mid (pr, d_t, d_b) \in P_2 \wedge [\![d_t]\!] \lhd h\}$
        $\langle 5 \rangle 1.$ $\exists (pr, d_t, d_b) \in P_2 : [\![d_t]\!] \lhd h$
          PROOF: Case assumption, assumption 1, Def. 18 and Def. 17

⟨5⟩2. Q.E.D.
PROOF: ⟨5⟩1, Def. 1 and Def. 3

⟨4⟩3. $\forall (pr, d_t, d_{b1}) \in P_1 : \exists (pr, d_t, d_{b2}) \in P_2 : [\![d_{b1}]\!] \subseteq [\![d_{b2}]\!]$
PROOF: Assumption 1, Def. 18 and Def. 17

⟨4⟩4. $\forall (pr, d_t, d_{b1}) \in P_1 : \exists (pr, d_t, d_{b2}) \in P_2 :$
$([\![d_t]\!] \succsim [\![d_{b1}]\!]) \subseteq ([\![d_t]\!] \succsim [\![d_{b2}]\!])$
PROOF: ⟨4⟩3 and Lemma 27 in [13] of monotonicity of $\subseteq$ with respect to $\succsim$

⟨4⟩5. Q.E.D.
PROOF: ⟨4⟩1, ⟨4⟩2, ⟨4⟩4 and Lemma 1

⟨3⟩3. Q.E.D.
PROOF: The cases are exhaustive

⟨2⟩3. $\forall H_1 \in C_1 : \exists H_2 \in C_2 : H_2 \subseteq H_1$

⟨3⟩1. CASE: $C_1 = \emptyset$
PROOF: Trivial

⟨3⟩2. CASE: $C_1 \neq \emptyset$

⟨4⟩1. Choose arbitrary $H_1 \in C_1$
PROOF: The trace set exists by case assumption

⟨4⟩2. $\exists H_2 \in C_2 : H_2 \subseteq H_1$

⟨5⟩1. Choose $(pe, d_t, d_{b1}) \in P_1$ such that $H_1 = ([\![d_t]\!] \succsim [\![d_{b1}]\!])\lhd$
PROOF: The permission rule exists by case assumption, Def. 1 and Def. 3

⟨5⟩2. Choose $(pe, d_t, d_{b2}) \in P_2$ such that $[\![d_{b2}]\!] \subseteq [\![d_{b1}]\!]$
PROOF: The permission rule exists by ⟨5⟩1, assumption 1, Def. 18 and Def. 17

⟨5⟩3. $([\![d_t]\!] \succsim [\![d_{b2}]\!])\lhd \in C_2$
PROOF: ⟨5⟩2, Def. 1 and Def. 3

⟨5⟩4. $([\![d_t]\!] \succsim [\![d_{b2}]\!]) \subseteq ([\![d_t]\!] \succsim [\![d_{b1}]\!])$
PROOF: ⟨5⟩2 and Lemma 27 in [13] of monotonicity of $\subseteq$ with respect to $\succsim$

⟨5⟩5. Q.E.D.
PROOF: ⟨5⟩3, ⟨5⟩4 and Lemma 1

⟨4⟩3. Q.E.D.
PROOF: ⟨4⟩1 and ⟨4⟩2

⟨3⟩3. Q.E.D.
PROOF: The cases are exhaustive

⟨2⟩4. $\forall H_2 \in C_2 : \exists H_1 \in C_1 : H_2 \subseteq H_1$
PROOF: Assumption 2 and Def. 9

⟨2⟩5. Q.E.D.
PROOF: ⟨2⟩1, ⟨2⟩2, ⟨2⟩3 and ⟨2⟩4

⟨1⟩4. Q.E.D.
PROOF: ⟨1⟩1, ⟨1⟩2, ⟨1⟩3 and Def. 7

□

Lemma 1. $H_1 \subseteq H_2 \Rightarrow H_1\lhd \subseteq H_2\lhd$

Proof.

ASSUME: $H_1 \subseteq H_2$
PROVE: $H_1\lhd \subseteq H_2\lhd$
$\langle 1\rangle$1. CASE: $H_1 = \emptyset$
  PROOF: Trivial since $H_1\lhd = \emptyset$ by case assumption and Def 14
$\langle 1\rangle$2. CASE: $H_1 \neq \emptyset$
  $\langle 2\rangle$1. Choose arbitrary $h \in H_1\lhd$
    PROOF: The trace exists by case assumption and Def. 14
  $\langle 2\rangle$2. $h \in H_2\lhd$
    $\langle 3\rangle$1. Choose $h' \in H_1$ such that $h' \lhd h$
      PROOF: The trace exists by $\langle 2\rangle$1 and Def. 14
    $\langle 3\rangle$2. $h' \in H_2$
      PROOF: $\langle 3\rangle$1 and assumption
    $\langle 3\rangle$3. Q.E.D.
      PROOF: $\langle 3\rangle$1, $\langle 3\rangle$2 and Def. 14
  $\langle 2\rangle$3. Q.E.D.
    PROOF: $\langle 2\rangle$1 and $\langle 2\rangle$2
$\langle 1\rangle$3. Q.E.D.
  PROOF: The cases are exhaustive

□

Theorem 8. $P_1 \rightsquigarrow' P_2 \wedge \forall r_2 \in P_2 : \exists r_1 \in P_1 : r_1 \rightsquigarrow' r_2 \Rightarrow P_1 \gtrdot P_2$

Proof.

ASSUME: 1. $P_1 \rightsquigarrow' P_2$
2. $\forall r_2 \in P_2 : \exists r_1 \in P_1 : r_1 \rightsquigarrow' r_2$

PROVE: $P_1 \gtrdot P_2$

$\langle 1 \rangle 1$. Choose arbitrary $h \in \mathcal{H}$
PROOF: $\mathcal{H}$ is non-empty

$\langle 1 \rangle 2$. LET: $[\![P_1]\!](h) = (a_1, u_1, C_1)$
$[\![P_2]\!](h) = (a_2, u_2, C_2)$

$\langle 1 \rangle 3$. $\forall H_2 \in C_2 : \exists H_1 \in C_1 : H_2 \subseteq H_1$

$\langle 2 \rangle 1$. CASE: $C_2 = \emptyset$
PROOF: Trivial

$\langle 2 \rangle 2$. CASE: $C_2 \neq \emptyset$

$\langle 3 \rangle 1$. Choose arbitrary $H_2 \in C_2$
PROOF: The trace set exists by case assumption

$\langle 3 \rangle 2$. $\exists H_1 \in C_1 : H_2 \subseteq H_1$

$\langle 4 \rangle 1$. Choose arbitrary $(pe, d_t, d_{b2}) \in P_2$ such that $H_2 = ([\![d_t]\!] \succsim [\![d_{b2}]\!])\lhd$
PROOF: The permission rule exists by case assumption, Def. 1 and Def. 3

$\langle 4 \rangle 2$. Choose $(pe, d_t, d_{b1}) \in P_1$ such that $[\![d_{b2}]\!] \subseteq [\![d_{b1}]\!]$
PROOF: The permission rule exists by assumption 2 and Def. 17

$\langle 4 \rangle 3$. $([\![d_t]\!] \succsim [\![d_{b1}]\!])\lhd \in C_1$
PROOF: $\langle 4 \rangle 2$, Def. 1 and Def. 3

$\langle 4 \rangle 4$. $([\![d_t]\!] \succsim [\![d_{b2}]\!]) \subseteq ([\![d_t]\!] \succsim [\![d_{b1}]\!])$
PROOF: $\langle 4 \rangle 2$ and Lemma 27 in [13] of monotonicity of $\subseteq$ with respect to $\succsim$

$\langle 4 \rangle 5$. Q.E.D.
PROOF: $\langle 4 \rangle 3$, $\langle 4 \rangle 4$ and Lemma 1

$\langle 3 \rangle 3$. Q.E.D.
PROOF: $\langle 3 \rangle 1$ and $\langle 3 \rangle 2$

$\langle 2 \rangle 3$. Q.E.D.
PROOF: The cases are exhaustive

$\langle 1 \rangle 4$. Q.E.D.
PROOF: $\langle 1 \rangle 1$, $\langle 1 \rangle 2$, $\langle 1 \rangle 3$ and Def. 9

□

Theorem 10. $P_1 \rightharpoonup_a S \wedge P_1 \rightsquigarrow P_2 \Rightarrow P_2 \rightharpoonup_a ([\![S]\!]^\Delta \uplus [\![S]\!])$

Proof.

ASSUME: 1. $P_1 \rightharpoonup_a S$
2. $P_1 \rightsquigarrow P_2$

PROVE: $P_2 \rightharpoonup_a ([\![S]\!]^\Delta \uplus [\![S]\!])$

$\langle 1 \rangle 1$. CASE: $P_2 = \emptyset$
PROOF: Immediately from Def. 5 since $\forall h \in \mathcal{H} : [\![P_2]\!](h) = (\mathcal{H}, \emptyset, \emptyset)$

$\langle 1 \rangle 2$. CASE: $P_2 \neq \emptyset$

$\langle 2 \rangle 1$. Choose arbitrary $(p, n) \in ([\![S]\!]^\Delta \uplus [\![S]\!])$
PROOF: The semantics of any sequence diagram is non-empty

$\langle 2 \rangle 2$. $\forall h \in (p \setminus n)$:
$h \in a_2 \wedge$
$h \notin u_2 \wedge$
$\forall H_2 \in C_2 : \exists (p', n') \in ([\![S]\!]^\Delta \uplus [\![S]\!]) : \forall h' \in (p' \setminus n') : h' \in H_2$
where $[\![P_2]\!](h) = (a_2, u_2, C_2)$

$\langle 3 \rangle 1$. CASE: $(p \setminus n) = \emptyset$
PROOF: Trivial

$\langle 3 \rangle 2$. CASE: $(p \setminus n) \neq \emptyset$

$\langle 4 \rangle 1$. Choose arbitrary $h \in (p \setminus n)$
PROOF: The trace exists by case assumption

$\langle 4 \rangle 2$. LET: $[\![P_1]\!](h) = (a_1, u_1, C_1)$
$[\![P_2]\!](h) = (a_2, u_2, C_2)$

$\langle 4 \rangle 3$. $h \in a_2 \wedge$
$h \notin u_2 \wedge$
$\forall H_2 \in C_2 : \exists (p', n') \in ([\![S]\!]^\Delta \uplus [\![S]\!]) : \forall h' \in (p' \setminus n') : h' \in H_2$

$\langle 5 \rangle 1$. $h \in a_2$

$\langle 6 \rangle 1$. $h \in a_1$

$\langle 7 \rangle 1$. $\exists (p^\dagger, n^\dagger) \in [\![S]\!] : (p \setminus n) \subseteq (p^\dagger \setminus n^\dagger)$
PROOF: Theorem 9 and Def. 3 in [31] of refinement of sets of interaction obligations

$\langle 7 \rangle 2$. $\exists (p^\dagger, n^\dagger) \in [\![S]\!] : h \in (p^\dagger \setminus n^\dagger)$
PROOF: $\langle 4 \rangle 1$ and $\langle 7 \rangle 1$

$\langle 7 \rangle 3$. Q.E.D.
PROOF: $\langle 7 \rangle 2$, assumption 1 and Def. 5

$\langle 6 \rangle 2$. $a_2 \subseteq a_1$
PROOF: Assumption 2 and Def. 7

$\langle 6 \rangle 3$. $(a_1 \setminus a_2) \cap (p \setminus n) = \emptyset$
PROOF: Def. 10, construction of $[\![S]\!]^\Delta$ and definition of $\uplus$

$\langle 6 \rangle 4$. Q.E.D.
PROOF: $\langle 4 \rangle 1$, $\langle 6 \rangle 1$, $\langle 6 \rangle 2$ and $\langle 6 \rangle 3$

$\langle 5 \rangle 2$. $h \notin u_2$

$\langle 6 \rangle 1$. $h \notin u_1$

$\langle 7 \rangle 1$. $\exists (p^\dagger, n^\dagger) \in [\![S]\!] : (p \setminus n) \subseteq (p^\dagger \setminus n^\dagger)$

Proof: Theorem 9 and Def. 3 in [31] of refinement of sets of interaction obligations

$\langle 7\rangle 2.$ $\exists(p^\dagger, n^\dagger) \in [\![S]\!] : h \in (p^\dagger \setminus n^\dagger)$

Proof: $\langle 4\rangle 1$ and $\langle 7\rangle 1$

$\langle 7\rangle 3.$ Q.E.D.

Proof: $\langle 7\rangle 2$, assumption 1 and Def. 5

$\langle 6\rangle 2.$ $u_1 \subseteq u_2$

Proof: Assumption 2 and Def. 7

$\langle 6\rangle 3.$ $(u_2 \setminus u_1) \cap (p \setminus n) = \emptyset$

Proof: Def. 10, construction of $[\![S]\!]^\Delta$ and definition of $\uplus$

$\langle 6\rangle 4.$ Q.E.D.

Proof: $\langle 4\rangle 1$, $\langle 6\rangle 1$, $\langle 6\rangle 2$ and $\langle 6\rangle 3$

$\langle 5\rangle 3.$ $\forall H_2 \in C_2 : \exists(p', n') \in ([\![S]\!]^\Delta \uplus [\![S]\!]) : \forall h' \in (p' \setminus n') : h' \in H_2$

$\langle 6\rangle 1.$ Case: $C_2 = \emptyset$

Proof: Trivial

$\langle 6\rangle 2.$ Case: $C_2 \neq \emptyset$

$\langle 7\rangle 1.$ Choose arbitrary $H_2 \in C_2$

Proof: The trace set exists by case assumption

$\langle 7\rangle 2.$ $\exists(p', n') \in ([\![S]\!]^\Delta \uplus [\![S]\!]) : \forall h' \in (p' \setminus n') : h' \in H_2$

$\langle 8\rangle 1.$ Choose $H_1 \in C_1$ such that $H_2 \subseteq H_1$

Proof: The trace set exists by $\langle 7\rangle 1$, assumption 2 and Def. 7

$\langle 8\rangle 2.$ Choose $(p^\dagger, n^\dagger) \in [\![S]\!]$ such that $(p \setminus n) \subseteq (p^\dagger \setminus n^\dagger)$

Proof: The interaction obligation exists by Theorem 9 and Def. 3 in [31] of refinement of sets of interaction obligations

$\langle 8\rangle 3.$ $h \in (p^\dagger \setminus n^\dagger)$

Proof: $\langle 4\rangle 1$ and $\langle 8\rangle 2$

$\langle 8\rangle 4.$ Choose $(p^\star, n^\star) \in [\![S]\!]$ such that
$\forall h' \in (p^\star \setminus n^\star) : h' \in H_1$

Proof: $\langle 8\rangle 1$, $\langle 8\rangle 2$, $\langle 8\rangle 3$, assumption 1 and Def. 5

$\langle 8\rangle 5.$ Choose $(p', n') \in ([\![S]\!]^\Delta \uplus [\![S]\!])$ such that
$(p' \setminus n') \subseteq (p^\star \setminus n^\star)$

Proof: The interaction obligation exists by Theorem 9 and Def. 3 in [31] of refinement of sets of interaction obligations

$\langle 8\rangle 6.$ $\forall h' \in (p' \setminus n') : h' \in H_1$

Proof: $\langle 8\rangle 4$ and $\langle 8\rangle 5$

$\langle 8\rangle 7.$ $(H_1 \setminus H_2) \cap (p' \setminus n') = \emptyset$

Proof: $\langle 8\rangle 1$, Def. 10, construction of $[\![S]\!]^\Delta$ and definition of $\uplus$

$\langle 8\rangle 8.$ $\forall h' \in (p' \setminus n') : h' \in H_2$

Proof: $\langle 8\rangle 1$, $\langle 8\rangle 6$ and $\langle 8\rangle 7$

$\langle 8\rangle 9.$ Q.E.D.

PROOF: $\langle 8\rangle 5$ and $\langle 8\rangle 8$

$\langle 7\rangle 3$. Q.E.D.

PROOF: $\langle 7\rangle 1$ and $\langle 7\rangle 2$

$\langle 6\rangle 3$. Q.E.D.

PROOF: The cases are exhaustive

$\langle 5\rangle 4$. Q.E.D.

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

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

PROOF: $\langle 4\rangle 1$, $\langle 4\rangle 2$ and $\langle 4\rangle 3$

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

PROOF: The cases are exhaustive

$\langle 2\rangle 3$. Q.E.D.

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

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

PROOF: The cases are exhaustive

$\square$

Theorem 12.

$$\{(pe, d_t, d_{b1} \text{ alt } d_{b2})\} \rightarrow_a S \Rightarrow$$
$$\{(pe, d_t, d_{b1})\} \rightarrow_a \mathsf{refuse}((d_t \text{ seq } d_{b2}) \text{ par any}) \text{ alt } S$$

Proof.

ASSUME: $\{(pe, d_t, d_{b1} \text{ alt } d_{b2})\} \rightarrow_a S$
PROVE: $\{(pe, d_t, d_{b1})\} \rightarrow_a \mathsf{refuse}((d_t \text{ seq } d_{b2}) \text{ par any}) \text{ alt } S$
LET: $S' = \mathsf{refuse}((d_t \text{ seq } d_{b2}) \text{ par any}) \text{ alt } S$

⟨1⟩1. CASE: $\forall (p'_i, n'_i) \in [\![S']\!] : \forall h \in (p'_i \setminus n'_i) : h \notin [\![d_t]\!] \lhd$
  PROOF: Immediately from Theorem 4

⟨1⟩2. CASE: $\exists (p'_i, n'_i) \in [\![S']\!] : \exists h \in (p'_i \setminus n'_i) : h \in [\![d_t]\!] \lhd$
  ⟨2⟩1. Choose arbitrary $(p'_i, n'_i) \in [\![S']\!]$ and $h \in (p'_i \setminus n'_i)$ such that $h \in [\![d_t]\!] \lhd$
    PROOF: The interaction obligation and trace exist by case assumption
  ⟨2⟩2. $\exists (p'_j, n'_j) \in [\![S']\!] : \forall h' \in (p'_j \setminus n'_j) : h' \in ([\![d_t]\!] \succsim [\![d_{b1}]\!]) \lhd$
    ⟨3⟩1. $S \rightsquigarrow S'$
      PROOF: By Theorem 11 since the set $[\![(d_t \text{ seq } d_{b2}) \text{ par any}]\!]$ is singleton
    ⟨3⟩2. Choose $(p_i, n_i) \in [\![S]\!]$ such that $h \in (p_i \setminus n_i)$
      PROOF: The interaction obligation exists by ⟨3⟩1 and Def. 3 in [31] of refinement of sequence diagrams
    ⟨3⟩3. Choose $(p_j, n_j) \in [\![S]\!]$ such that
      $\forall h' \in (p_j \setminus n_j) : h' \in ([\![d_t]\!] \succsim ([\![d_{b1}]\!] \cup [\![d_{b2}]\!])) \lhd$
      PROOF: The interaction obligation exists by case assumption, proof assumption and Theorem 4
    ⟨3⟩4. Choose $(p'_j, n'_j) \in [\![S']\!]$ such that $(p'_j \setminus n'_j) \subseteq (p_j \setminus n_j)$
      PROOF: The interaction obligation exists by ⟨3⟩1 and Def. 3 in [31] of refinement of sequence diagrams
    ⟨3⟩5. $\forall h' \in (p'_j \setminus n'_j) : h' \in ([\![d_t]\!] \succsim ([\![d_{b1}]\!] \cup [\![d_{b2}]\!])) \lhd$
      PROOF: ⟨3⟩3 and ⟨3⟩4
    ⟨3⟩6. $\forall h' \in (p'_j \setminus n'_j) : h' \in (([\![d_t]\!] \succsim [\![d_{b1}]\!]) \lhd \cup ([\![d_t]\!] \succsim [\![d_{b2}]\!]) \lhd)$
      PROOF: ⟨3⟩5, Def. 14 and Lemma 14 in [13] of left distributivity of $\succsim$ over $\cup$
    ⟨3⟩7. $\forall h' \in (p'_j \setminus n'_j) : h' \notin (([\![d_t]\!] \succsim [\![d_{b2}]\!]) \parallel \mathcal{H})$
      PROOF: By construction of S'
    ⟨3⟩8. $([\![d_t]\!] \succsim [\![d_{b2}]\!]) \lhd = (([\![d_t]\!] \succsim [\![d_{b2}]\!]) \parallel \mathcal{H})$
      PROOF: Lemma 2
    ⟨3⟩9. Q.E.D.
      PROOF: ⟨3⟩4, ⟨3⟩6, ⟨3⟩7 and ⟨3⟩8
  ⟨2⟩3. Q.E.D.
    PROOF: ⟨2⟩1, ⟨2⟩2 and Theorem 4

⟨1⟩3. Q.E.D.
  PROOF: The cases are exhaustive

□

Theorem 13.

$$\{(ob, d_t, d_{b1} \text{ alt } d_{b2})\} \rightarrow_a S \Rightarrow$$
$$\{(ob, d_t, d_{b1})\} \rightarrow_a \text{refuse}((d_t \text{ seq } d_{b2}) \text{ par any}) \text{ alt } S$$

Proof.

ASSUME: $\{(ob, d_t, d_{b1} \text{ alt } d_{b2})\} \rightarrow_a S$
PROVE: $\{(ob, d_t, d_{b1})\} \rightarrow_a \text{refuse}((d_t \text{ seq } d_{b2}) \text{ par any}) \text{ alt } S$
LET: $S' = \text{refuse}((d_t \text{ seq } d_{b2}) \text{ par any}) \text{ alt } S$

⟨1⟩1. CASE: $\forall (p', n') \in [\![S']\!] : \forall h \in (p' \setminus n') : h \notin [\![d_t]\!]\lhd$
  PROOF: Immediately from Theorem 4

⟨1⟩2. CASE: $\exists (p', n') \in [\![S']\!] : \exists h \in (p' \setminus n') : h \in [\![d_t]\!]\lhd$
  ⟨2⟩1. Choose arbitrary $(p', n') \in [\![S']\!]$ and $h \in (p' \setminus n')$ such that $h \in [\![d_t]\!]\lhd$
    PROOF: The interaction obligation and trace exist by case assumption
  ⟨2⟩2. $h \in ([\![d_t]\!] \succsim [\![d_{b1}]\!])\lhd$
    ⟨3⟩1. $S \rightsquigarrow S'$
      PROOF: By Theorem 11 since the set $[\![(d_t \text{ seq } d_{b2}) \text{ par any}]\!]$ is singleton
    ⟨3⟩2. Choose $(p, n) \in [\![S]\!]$ such that $h \in (p \setminus n)$
      PROOF: The interaction obligation exists by ⟨3⟩1 and Def. 3 in [31] of refinement of sequence diagrams
    ⟨3⟩3. $h \in ([\![d_t]\!] \succsim ([\![d_{b1}]\!] \cup [\![d_{b2}]\!]))\lhd$
      PROOF: Case assumption, proof assumption and Theorem 4
    ⟨3⟩4. $h \in ([\![d_t]\!] \succsim [\![d_{b1}]\!])\lhd$ or $h \in ([\![d_t]\!] \succsim [\![d_{b2}]\!])\lhd$
      PROOF: ⟨3⟩3, Def. 14 and Lemma 14 in [13] of left distributivity of $\succsim$ over $\cup$
    ⟨3⟩5. $h \notin (([\![d_t]\!] \succsim [\![d_{b2}]\!]) \parallel \mathcal{H})$
      PROOF: By construction of S'
    ⟨3⟩6. $([\![d_t]\!] \succsim [\![d_{b2}]\!])\lhd = (([\![d_t]\!] \succsim [\![d_{b2}]\!]) \parallel \mathcal{H})$
      PROOF: Lemma 2
    ⟨3⟩7. Q.E.D.
      PROOF: ⟨3⟩4, ⟨3⟩5 and ⟨3⟩6
  ⟨2⟩3. Q.E.D.
    PROOF: ⟨2⟩1, ⟨2⟩2 and Theorem 4

⟨1⟩3. Q.E.D.
  PROOF: The cases are exhaustive

□

Theorem 14.

$$\{(pr, d_t, d_{b1})\} \rightarrow_a S \Rightarrow$$
$$\{(pr, d_t, d_{b1} \ \mathsf{alt} \ d_{b2})\} \rightarrow_a \mathsf{refuse}((d_t \ \mathsf{seq} \ d_{b2}) \ \mathsf{par} \ \mathsf{any}) \ \mathsf{alt} \ S$$

Proof.

ASSUME: $\{(pr, d_t, d_{b1})\} \rightarrow_a S$
PROVE: $\{(pr, d_t, d_{b1} \ \mathsf{alt} \ d_{b2})\} \rightarrow_a \mathsf{refuse}((d_t \ \mathsf{seq} \ d_{b2}) \ \mathsf{par} \ \mathsf{any}) \ \mathsf{alt} \ S$
LET: $S' = \mathsf{refuse}((d_t \ \mathsf{seq} \ d_{b2}) \ \mathsf{par} \ \mathsf{any}) \ \mathsf{alt} \ S$
⟨1⟩1. CASE: $\forall (p', n') \in [\![S']\!] : \forall h \in (p' \setminus n') : h \notin [\![d_t]\!] \lhd$
  PROOF: Immediately from Theorem 4
⟨1⟩2. CASE: $\exists (p', n') \in [\![S']\!] : \exists h \in (p' \setminus n') : h \in [\![d_t]\!] \lhd$
  ⟨2⟩1. Choose arbitrary $(p', n') \in [\![S']\!]$ and $h \in (p' \setminus n')$ such that $h \in [\![d_t]\!] \lhd$
    PROOF: The interaction obligation and trace exist by case assumption
  ⟨2⟩2. $h \notin ([\![d_t]\!] \succsim ([\![d_{b1}]\!] \cup [\![d_{b2}]\!])) \lhd$
    ⟨3⟩1. $S \rightsquigarrow S'$
      PROOF: By Theorem 11 since the set $[\![(d_t \ \mathsf{seq} \ d_{b2}) \ \mathsf{par} \ \mathsf{any}]\!]$ is singleton
    ⟨3⟩2. Choose $(p, n) \in [\![S]\!]$ such that $h \in (p \setminus n)$
      PROOF: The interaction obligation exists by ⟨3⟩1 and Def. 3 in [31] of refinement of sequence diagrams
    ⟨3⟩3. $h \notin ([\![d_t]\!] \succsim [\![d_{b1}]\!]) \lhd$
      PROOF: Case assumption, proof assumption and Theorem 4
    ⟨3⟩4. $h \notin (([\![d_t]\!] \succsim [\![d_{b2}]\!]) \parallel \mathcal{H})$
      PROOF: By construction of S'
    ⟨3⟩5. $([\![d_t]\!] \succsim [\![d_{b2}]\!]) \lhd = (([\![d_t]\!] \succsim [\![d_{b2}]\!]) \parallel \mathcal{H})$
      PROOF: Lemma 2
    ⟨3⟩6. $h \notin (([\![d_t]\!] \succsim [\![d_{b1}]\!]) \lhd \cup ([\![d_t]\!] \succsim [\![d_{b2}]\!]) \lhd)$
      PROOF: ⟨3⟩3, ⟨3⟩4 and ⟨3⟩5
    ⟨3⟩7. Q.E.D.
      PROOF: ⟨3⟩6, Def. 14 and Lemma 14 in [13] of left distributivity of $\succsim$ over $\cup$
  ⟨2⟩3. Q.E.D.
    PROOF: ⟨2⟩1, ⟨2⟩2 and Theorem 4
⟨1⟩3. Q.E.D.
  PROOF: The cases are exhaustive

□

Theorem 15. $\{(dm, d_t, d_b)\} \rightarrow_a \mathsf{refuse}(d_t \ \mathsf{par}\ \mathsf{any})\ \mathsf{alt}\ S$

Proof.

PROVE: $\{(dm, d_t, d_b)\} \rightarrow_a \mathsf{refuse}(d_t \ \mathsf{par}\ \mathsf{any})\ \mathsf{alt}\ S$
LET: $S' = \mathsf{refuse}(d_t \ \mathsf{par}\ \mathsf{any})\ \mathsf{alt}\ S$

⟨1⟩1. $\forall (p, n) \in [\![S']\!] : \forall h \in (p \setminus n) : h \notin [\![d_t]\!]\lhd$

  ⟨2⟩1. Choose arbitrary $(p, n) \in S'$

    PROOF: The semantics of any sequence diagram is non-empty

  ⟨2⟩2. $\forall h \in (p \setminus n) : h \notin [\![d_t]\!]\lhd$

    ⟨3⟩1. $([\![d_t]\!] \parallel \mathcal{H}) \subseteq n$

      PROOF: By construction of S'

    ⟨3⟩2. $[\![d_t]\!]\lhd = ([\![d_t]\!] \parallel \mathcal{H})$

      PROOF: Lemma 2

    ⟨3⟩3. Q.E.D.

      PROOF: ⟨3⟩1 and ⟨3⟩2

  ⟨2⟩3. Q.E.D.

    PROOF: ⟨2⟩1 and ⟨2⟩2

⟨1⟩2. Q.E.D.

  PROOF: ⟨1⟩1 and Theorem 4

□

Lemma 2. $\forall H \subseteq \mathcal{H} : H\lhd = (H \parallel \mathcal{H})$

Proof.

PROVE: $H\lhd = (H \parallel \mathcal{H})$
⟨1⟩1. CASE: $H = \emptyset$
  ⟨2⟩1. $H\lhd = \emptyset$
    PROOF: Def. 14
  ⟨2⟩2. $(H \parallel \mathcal{H}) = \emptyset$
    PROOF: Def. 11
  ⟨2⟩3. Q.E.D.
    PROOF: ⟨2⟩1 and ⟨2⟩2
⟨1⟩2. CASE: $H \neq \emptyset$
  ⟨2⟩1. $H\lhd \subseteq (H \parallel \mathcal{H})$
    ⟨3⟩1. Choose arbitrary $h \in H\lhd$
      PROOF: The trace exists by case assumption and Def. 14
    ⟨3⟩2. $h \in (H \parallel \mathcal{H})$
      ⟨4⟩1. Choose $s \in \{1,2\}^\infty$ such that $\pi_2((\{1\} \times \mathcal{E}) \circledT (s,h)) \in H$
        PROOF: ⟨3⟩1, Def. 14 and Def. 13
      ⟨4⟩2. $\pi_2((\{2\} \times \mathcal{E}) \circledT (s,h)) \in \mathcal{H}$
        PROOF: Trivial
      ⟨4⟩3. Q.E.D.
        PROOF: ⟨4⟩1, ⟨4⟩2 and Def. 11
    ⟨3⟩3. Q.E.D.
      PROOF: ⟨3⟩1, ⟨3⟩2 and definition of $\subseteq$
  ⟨2⟩2. $(H \parallel \mathcal{H}) \subseteq H\lhd$
    ⟨3⟩1. Choose arbitrary $h \in (H \parallel \mathcal{H})$
      PROOF: The trace exists by case assumption and Def. 11
    ⟨3⟩2. $h \in H\lhd$
      ⟨4⟩1. Choose $s \in \{1,2\}^\infty$ and $h' \in H$ such that
          $h' = \pi_2((\{1\} \times \mathcal{E}) \circledT (s,h))$
        PROOF: The sequence s and trace h' exist by Def. 11 and ⟨3⟩1
      ⟨4⟩2. $h' \lhd h$
        PROOF: ⟨4⟩1 and Def. 13
      ⟨4⟩3. Q.E.D.
        PROOF: ⟨4⟩1, ⟨4⟩2 and Def. 14
    ⟨3⟩3. Q.E.D.
      PROOF: ⟨3⟩1, ⟨3⟩2 and definition of $\subseteq$
  ⟨2⟩3. Q.E.D.
    PROOF: ⟨2⟩1 and ⟨2⟩2
⟨1⟩3. Q.E.D.
  PROOF: The cases are exhaustive

□