UML 2.0 Interactions: Semantics and Refinement

María Victoria Cengarle[1] and Alexander Knapp[2]

[1] Technische Universität München
cengarle@in.tum.de
[2] Ludwig-Maximilians-Universität München
knapp@pst.ifi.lmu.de

Abstract. The UML 2.0 integrates a dialect of High-Level Message Sequence Charts (HMSCs) for interaction modelling. The most noteworthy addition of UML 2.0 interactions to HMSCs is the introduction of negated specifications which can be used to rule out behaviour from implementations. A trace-based semantics for UML 2.0 interactions is proposed which captures both the standard composition operators for UML 2.0 interactions from HMSCs, and the proprietary negation and assertion operators. The semantics lays the ground for discussing several alternatives for treating negation in interactions. In particular, the semantics decides whether a trace is positive or negative for a given interaction; all other traces are deemed to be inconclusive. Based on these verdicts, notions of implementation and refinement for interactions are defined.

1 Introduction

UML interactions describe possible message exchanges between system instances. In UML 2.0, a dialect of High-Level Messages Sequence Charts (HMSC [4]) replaced the quite inexpressive notion of UML 1.x interactions [5]. Besides integrating the standard HMSC primitives like sequential, parallel, and iterative composition of interactions, UML 2.0 provides means to specify negative behaviour, i.e., behaviour forbidden in system implementations. The ensuing increase in expressiveness makes UML 2.0 an acceptable choice for modelling safety-critical systems. However, in order to put UML 2.0 interactions on an equal footing with HMSCs or Live Sequence Charts (LSC [2]), a formal understanding of the semantics of its interaction language is indispensable. Moreover, the notion of implementation and refinement, based on the formal semantics, form a necessary prerequisite for using UML 2.0 interactions as a formal design language.

In fact, the UML 2.0 specification document [6] is rather vague on the innovative features of the interaction language, like negation. The semantics of what may be called the positive fragment of UML 2.0 interactions, i.e., the language part that does not contain negation, can be equipped straightforwardly with a formal semantics following the specification [9]. Not surprisingly, however, different interpretations of negative interactions have been proposed in the literature. According to the specification, a UML 2.0 interaction describes valid (or positive) and invalid (or negative) traces of event occurrences where invalid traces are induced by using the unary interaction operators $\mathsf{neg}(-)$ and $\mathsf{assert}(-)$. The set of positive and negative traces defined by an interaction need not cover all possible interactions, so the remaining traces may be called inconclusive

for the interaction. Störrle [8] discusses several alternatives for the negated interaction $\mathsf{neg}(S)$ ranging from "not the [valid] traces of S" over "anything but the [valid] traces of S" to exchanging the valid and the invalid traces of S; he finally adopts the last view in order ensure that double negation is the identity. Each of these interpretations shows a drawback: In general, the first and the last approach assign no positive traces to $\mathsf{neg}(S)$ and thus the combination of negation with non-negated interaction fragments leads to an empty set of positive traces. The second approach discards the possibility of inconclusive traces. In contrast, Haugen and Stølen [3] interpret the valid traces of $\mathsf{neg}(S)$ as consisting just of the empty trace; a formal definition of valid and invalid traces of an interaction, however, is not given.

We propose a trace-based, formal semantics for UML 2.0 interactions including part of the positive fragment but concentrating on the language constructs for specifying negative traces. For the definition of the semantics we employ Pratt's framework of partially ordered multisets or pomsets [7] for modelling concurrency. On the one hand, this framework simplifies the definition of the various composition operators for interactions; on the other hand, traces are subsumed by linear pomsets. The semantics decides if a trace is positive or if it is negative for an interaction. We only briefly summarise the semantics of the positive interaction fragment which coincides with Störrle's interpretation [9]. For negated interactions, we build on Haugen and Stølen's view [3] and define the negative traces of combined interaction fragments. We detail the consequences of this approach and contrast it with Störrle's interpretations. Moreover, we provide means for reducing the semantics to only calculating the positive traces of an interaction, albeit at the expense of a classical not-operator. The semantics is put to use by introducing a notion of an implementation of an interaction as a process that shows a least one positive trace of the interaction and no negative trace. In particular, our interaction semantics implies that a trace may be simultaneously positive and negative for the same interaction. We discern between such overspecified interactions and interactions that are contradictory in the sense that they do not admit an implementation. Based on interaction implementations, we introduce a model-theoretic notion of refinement of interactions.

The remainder of this paper is structured as follows: In Sect. 2 we briefly recall the notion of pomsets and traces. The fragment of the interaction language of UML 2.0 considered here is introduced in Sect. 3, together with its abstract syntax. In Sect. 4 the language of interactions is equipped with a trace-based formal semantics, which includes both valid and invalid traces. In Sect. 5 the reduction of the semantics of negation to the semantics of valid traces is studied. The semantics is used in Sect. 6 to define the concepts of implementation and refinement of interactions. In Sect. 7 we analyse implications of the introduced notions with respect to related work. We conclude in Sect. 8 with an outlook to future research.

2 Preliminaries

We briefly review the basic definitions on partially ordered, labelled multisets as introduced by Pratt [7] for modelling concurrency. In particular, we define sequential and parallel composition operators and the notion of traces and processes.

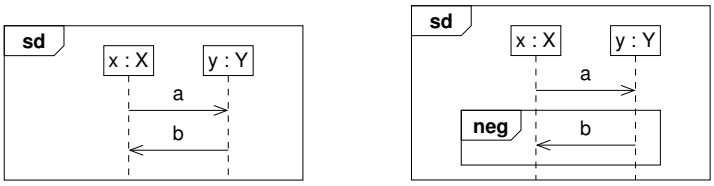
A partially ordered, labelled multiset, or *pomset*, is the isomorphism class $[(X, \leq_X, \lambda_X)]$ of a labelled partial order $(X, \leq_X, \lambda_X)$ w.r.t. monotone, label-preserving maps. A *trace* is a pomset whose ordering is total. We write $lin(p)$ for all possible linearisations of a pomset p, i.e., all traces that extend the ordering of p: $[(X', \leq_{X'}, \lambda_{X'})] \in lin([(X, \leq_X, \lambda_X)])$ if, and only if $X' = X$, $\lambda_{X'} = \lambda_X$, and $\leq_X \subseteq \leq_{X'}$ where $x_1 \leq_{X'} x_2$ or $x_2 \leq_{X'} x_1$ for all $x_1, x_2 \in X'$.

The *empty* pomset, represented by $(\emptyset, \emptyset, \emptyset)$, is denoted by ε. Let $p = [(X, \leq_X, \lambda_X)]$ and $q = [(Y, \leq_Y, \lambda_Y)]$ be pomsets such that $X \cap Y = \emptyset$. The *concurrence* of p and q, written as $p \| q$, is given by $[(X \cup Y, \leq_X \cup \leq_Y, \lambda_X \cup \lambda_Y)]$. The *concatenation* of p and q, written as $p ; q$, is given by $[(X \cup Y, (\leq_X \cup \leq_Y \cup (X \times Y))^*, \lambda_X \cup \lambda_Y)]$. Given a binary, symmetric relation $\asymp$ on labels, the $\asymp$-*concatenation* of p and q, written as $p ;_{\asymp} q$, is given by $[(X \cup Y, (\leq_X \cup \leq_Y \cup \{(x, y) \in X \times Y \mid \lambda_X(x) \asymp \lambda_Y(y)\})^*, \lambda_X \cup \lambda_Y)]$. Note that concatenation and $\asymp$-concatenation are associative, and concurrence is associative and commutative.

A *process* is a set of pomsets. An n-ary function f on pomsets is lifted to processes $P_1, \ldots, P_n$ by defining $f(P_1, \ldots, P_n) = \{f(p_1, \ldots, p_n) \mid p_1 \in P_1, \ldots, p_n \in P_n\}$.

3 UML 2.0 Interactions

UML 2.0 interactions describe message exchanges between instances. Consider the sample basic interaction in Fig. 1(a) which specifies two instances x and y which exchange the messages a and b. The dispatch of a message (depicted by the arrow tail) and the arrival of a message (arrow head) on the lifeline of an instance (dashed line) are called event occurrences. The pictorial representation of a basic interaction carries the intuitive meaning of a partial order of event occurrences: The dispatch of a message occurs before the arrival of the same message; and the event occurrences on the lifeline of an instance are ordered from top to bottom. Thus, the interaction in Fig. 1(a) defines a single valid trace in which the following event occurrences appear in this order: a is sent from x to y; a is received by y from x; b is sent from y to x; b is received by x from y. In particular, all other traces are inconclusive for this interaction. On the other hand, the interaction in Fig. 1(b) defines both negative and positive traces. The trace of first sending and receiving a and then sending and receiving b is negative, whereas the trace just consisting of sending and receiving a is positive. Again, all other traces are inconclusive, as the interaction provides no verdicts on these traces.



(a) Basic interaction diagram (b) Interaction diagram with negation

Fig. 1. Sample interactions

More generally, a UML 2.0 basic interaction consists of event occurrences and a general ordering relation which induces an arbitrary partial order on the set of event occurrences, subject to the following constraints: The dispatch of a message occurs before the arrival of the message; and all event occurrences for the same lifeline are totally ordered. Moreover, UML 2.0 puts a number of interaction-building operators at disposal. In sequential composition, the behaviour of the resulting interaction is the behaviour of the first given interaction followed by the behaviour of the second given interaction. There are two kinds of sequential composition which differ in the meaning of the word "followed". Strict composition requires the behaviour of the first interaction to be completely performed before starting with the behaviour of the second interaction. Weak composition only requires the behaviour specified for an instance in the first interaction to be completely performed before starting with the behaviour for that instance in the second interaction. Other operators are parallel composition, disjunction, loop, ignore, assert, and negation. Two parallel interactions are to be executed simultaneously. Disjunction means to execute any one of two given interactions. Loop repeatedly executes its interaction argument, as long as given by two additional natural numbers m and n passed as parameter: at least m and at most n times, where n can also be ∞ meaning an arbitrary number of times. Ignore allows additional messages to occur besides the ones specified in its interaction argument. Finally, assertion discards inconclusive traces, and negation prohibits the behaviour specified by its argument.

We define the abstract syntax of the fragment of the language of UML 2.0 interactions introduced above, by first characterising basic interactions as pomsets and then capturing the interaction operators by a context-free grammar. We assume two primitive domains for *instances* $\mathbb{I}$ and *messages* $\mathbb{M}$. An *event* e is either of the form $\mathsf{snd}(s, r, m)$ or of the form $\mathsf{rcv}(s, r, m)$, representing the dispatch and the arrival of message m from *sender* instance s to *receiver* instance r, respectively. The set of events is denoted by $\mathbb{E}$. We say that the instance s is *active* for $\mathsf{snd}(s, r, m)$ and, similarly, that the instance r is *active* for $\mathsf{rcv}(s, r, m)$. We define a binary, symmetric *conflict* relation $\asymp$ on events: If an instance is active for both events e and e' then $e \asymp e'$.

A *basic* interaction is given by an event-labelled pomset $[(E, \leq_E, \lambda_E)]$ such that conflicting events do not occur concurrently, i.e., if $e_1, e_2 \in E$ with $\lambda_E(e_1) \asymp \lambda_E(e_2)$, then $e_1 \leq_E e_2$ or $e_2 \leq_E e_1$.

$$
\begin{array}{rcl}
\mathit{Interaction} & ::= & \mathit{Basic} \\
& | & \mathit{CombinedFragment} \\
\mathit{CombinedFragment} & ::= & \mathsf{strict}(\mathit{Interaction}, \mathit{Interaction}) \\
& | & \mathsf{seq}(\mathit{Interaction}, \mathit{Interaction}) \\
& | & \mathsf{par}(\mathit{Interaction}, \mathit{Interaction}) \\
& | & \mathsf{loop}(\mathit{Nat}, (\mathit{Nat} \mid \infty), \mathit{Interaction}) \\
& | & \mathsf{ignore}(\mathit{Messages}, \mathit{Interaction}) \\
& | & \mathsf{alt}(\mathit{Interaction}, \mathit{Interaction}) \\
& | & \mathsf{neg}(\mathit{Interaction}) \\
& | & \mathsf{assert}(\mathit{Interaction})
\end{array}
$$

Table 1. Abstract syntax of interactions (fragment)

The abstract syntax of interactions is given by the grammar in Tab. 1. Therein, *Basic* ranges over the basic interactions, *Nat* ranges over the natural numbers, and *Messages* over the subsets of $\mathbb{M}$.

From the notion of basic interactions and the interaction operators in Tab. 1 a number of auxiliary interaction operators can be derived. We use the name skip for the empty (basic) interaction, which is given by the pomset $[(\emptyset, \emptyset, \emptyset)]$. The operator $\mathsf{opt}(-)$ is defined by $\mathsf{opt}(S) = \mathsf{alt}(\mathsf{skip}, S)$, the operator $\mathsf{consider}(-,-)$ by $\mathsf{consider}(M, S) = \mathsf{ignore}(\mathbb{M} \setminus M, S)$. In fact, the UML 2.0 specification defines several other interaction operators, in particular break and $\mathsf{critical}$; these operators, as well as message parameters and conditions, are not considered in this work.

4 Semantics

We define a classical satisfaction relation between traces and interactions that do not contain occurrences of the operator for negation. We afterwards extend this definition for negation, and complement it with a negative satisfaction relation. After presenting some notorious examples, we show some properties of the notions introduced so far.

4.1 Semantic Domains

The domain $\mathbb{P}$ comprises all pomsets $[(E, \leq_E, \lambda_E)]$ labelled with events from $\mathbb{E}$ such that if $e_1, e_2 \in E$ with $\lambda_E(e_1) \asymp \lambda_E(e_2)$, then $e_1 \leq_E e_2$ or $e_2 \leq_E e_1$. The subdomain $\mathbb{T}$ of $\mathbb{P}$ comprises all pomsets in $\mathbb{P}$ that are traces. In particular, the empty pomset ε is in $\mathbb{T}$. When representing a finite pomset in $\mathbb{P}$ we will also use a more concrete, set-based notation like writing $\{\mathsf{snd}(s,r,m) \leq \mathsf{rcv}(s,r,m)\}$ instead of $[(\{e_1, e_2\}, \{(e_1,e_1),(e_1,e_2),(e_2,e_2)\}, \{e_1 \mapsto \mathsf{snd}(s,r,m), e_2 \mapsto \mathsf{rcv}(s,r,m)\})]$. Similarly, for the representation of finite traces in $\mathbb{T}$, as in the example above, we also employ the more succinct notation $\mathsf{snd}(s,r,m) \cdot \mathsf{rcv}(s,r,m)$.

On pomsets in $\mathbb{P}$, the filtering relation $\mathit{filter}(M) : \mathbb{P} \to \wp\mathbb{P}$ removes some elements of p whose labels show a message in M. More precisely, we first define $\mathit{filter}(M)$ on event-labelled sets: Let E be a set and $\lambda : E \to \mathbb{E}$ a labelling function; then $E' \in \mathit{filter}(M)(E, \lambda)$ if $E' \subseteq E$ and, if $e \in E \setminus E'$, then $(\lambda(e) = \mathsf{snd}(s,r,m) \vee \lambda(e) = \mathsf{rcv}(s,r,m)) \wedge m \in M$. For an event-labelled partial order $(E, \leq_E, \lambda_E)$ we set $(E', \leq_E \cap (E' \times E'), \lambda_E{\restriction}E') \in \mathit{filter}(M)(E, \leq_E, \lambda_E)$ if $E' \in \mathit{filter}(M)(E, \lambda_E)$. Finally, we extend these definitions to event-labelled pomsets by setting $[(E', \leq_{E'}, \lambda_{E'})] \in \mathit{filter}(M)([(E, \leq_E, \lambda_E)])$ if $(E', \leq_{E'}, \lambda_{E'}) \in \mathit{filter}(M)(E, \leq_E, \lambda_E)$, which is obviously well-defined. Note that $\mathit{filter}(M)$ restricted to traces delivers traces, i.e., $\mathit{filter}(M)$ is also a relation $\mathit{filter}(M) : \mathbb{T} \to \wp\mathbb{T}$.

4.2 The Positive Fragment

Let us begin considering interactions with no occurrence of negation or assertion, which we call the *positive fragment* of the language. The positive satisfaction relation between traces and interactions, denoted by $t \models_{\mathrm{p}} S$ and read t *positively satisfies* S, where t is a trace and S an interaction of the positive fragment, is inductively defined on the

$t \models_{\mathrm{p}} B \quad \text{if } t \in \mathit{lin}(B)$

$t \models_{\mathrm{p}} \mathsf{strict}(S_1, S_2) \quad \text{if } \exists t_1, t_2 \,.\, t = t_1 \,;\, t_2 \wedge t_1 \models_{\mathrm{p}} S_1 \wedge t_2 \models_{\mathrm{p}} S_2$

$t \models_{\mathrm{p}} \mathsf{seq}(S_1, S_2) \quad \text{if } \exists t_1, t_2 \,.\, t \in \mathit{lin}(t_1 \fatsemi t_2) \wedge t_1 \models_{\mathrm{p}} S_1 \wedge t_2 \models_{\mathrm{p}} S_2$

$t \models_{\mathrm{p}} \mathsf{par}(S_1, S_2) \quad \text{if } \exists t_1, t_2 \,.\, t \in \mathit{lin}(t_1 \parallel t_2) \wedge t_1 \models_{\mathrm{p}} S_1 \wedge t_2 \models_{\mathrm{p}} S_2$

$t \models_{\mathrm{p}} \mathsf{loop}(0, 0, S) \quad \text{if } t = \varepsilon$

$t \models_{\mathrm{p}} \mathsf{loop}(0, n+1, S) \quad \text{if } t = \varepsilon \vee t \models_{\mathrm{p}} \mathsf{seq}(S, \mathsf{loop}(0, n, S))$

$t \models_{\mathrm{p}} \mathsf{loop}(m+1, n+1, S) \quad \text{if } t \models_{\mathrm{p}} \mathsf{seq}(S, \mathsf{loop}(m, n, S))$

$t \models_{\mathrm{p}} \mathsf{loop}(m, \infty, S) \quad \text{if } \exists n \geq m \,.\, t \models_{\mathrm{p}} \mathsf{loop}(m, n, S)$

$t \models_{\mathrm{p}} \mathsf{ignore}(M, S) \quad \text{if } \exists t_1 \,.\, t_1 \in \mathit{filter}(M)(t) \wedge t_1 \models_{\mathrm{p}} S$

$t \models_{\mathrm{p}} \mathsf{alt}(S_1, S_2) \quad \text{if } t \models_{\mathrm{p}} S_1 \vee t \models_{\mathrm{p}} S_2$

(a) Semantics of the positive fragment

$t \models_{\mathrm{p}} \mathsf{neg}(S) \quad \text{if } t = \varepsilon$

$t \models_{\mathrm{p}} \mathsf{assert}(S) \quad \text{if } t \models_{\mathrm{p}} S$

$t \models_{\mathrm{n}} \mathsf{strict}(S_1, S_2) \quad \text{if } \exists t_1, t_2 \,.\, t = t_1 \,;\, t_2 \wedge (t_1 \models_{\mathrm{n}} S_1 \vee (t_1 \models_{\mathrm{p}} S_1 \wedge t_2 \models_{\mathrm{n}} S_2))$

$t \models_{\mathrm{n}} \mathsf{seq}(S_1, S_2) \quad \text{if } \exists t_1, t_2 \,.\, t \in \mathit{lin}(t_1 \fatsemi t_2) \wedge (t_1 \models_{\mathrm{n}} S_1 \vee (t_1 \models_{\mathrm{p}} S_1 \wedge t_2 \models_{\mathrm{n}} S_2))$

$t \models_{\mathrm{n}} \mathsf{par}(S_1, S_2) \quad \text{if } \exists t_1, t_2 \,.\, t \in \mathit{lin}(t_1 \parallel t_2) \wedge ((t_1 \models_{\mathrm{n}} S_1 \wedge t_2 \models_{\mathrm{n}} S_2) \vee (t_1 \models_{\mathrm{n}} S_1 \wedge t_2 \models_{\mathrm{p}} S_2) \vee (t_1 \models_{\mathrm{p}} S_1 \wedge t_2 \models_{\mathrm{n}} S_2))$

$t \models_{\mathrm{n}} \mathsf{loop}(0, n+1, S) \quad \text{if } t \models_{\mathrm{n}} \mathsf{seq}(S, \mathsf{loop}(0, n, S))$

$t \models_{\mathrm{n}} \mathsf{loop}(m+1, n+1, S) \quad \text{if } t \models_{\mathrm{n}} \mathsf{seq}(S, \mathsf{loop}(m, n, S))$

$t \models_{\mathrm{n}} \mathsf{loop}(m, \infty, S) \quad \text{if } \exists n \geq m \,.\, t \models_{\mathrm{n}} \mathsf{loop}(m, n, S)$

$t \models_{\mathrm{n}} \mathsf{ignore}(M, S) \quad \text{if } \exists t_1 \,.\, t_1 \in \mathit{filter}(M)(t) \wedge t_1 \models_{\mathrm{n}} S$

$t \models_{\mathrm{n}} \mathsf{alt}(S_1, S_2) \quad \text{if } t \models_{\mathrm{n}} S_1 \wedge t \models_{\mathrm{n}} S_2$

$t \models_{\mathrm{n}} \mathsf{neg}(S) \quad \text{if } t \neq \varepsilon \wedge t \models_{\mathrm{p}} S$

$t \models_{\mathrm{n}} \mathsf{assert}(S) \quad \text{if } t \not\models_{\mathrm{p}} S$

(b) Extended semantics for negation

Table 2. Semantics of interactions

structure of S as shown in Tab. 2(a). Therein, B ranges over basic interactions. This semantics is a reformulation of Störrle's definition [9] using pomsets.

In particular, the only trace positively satisfying the emtpy interaction skip is the empty pomset. A basic interaction $B = \{\mathsf{snd}(s, r, m) \leq \mathsf{rcv}(s, r, m)\}$ is positively satisfied solely by the trace $t_B = \mathsf{snd}(s, r, m) \cdot \mathsf{rcv}(s, r, m)$.

4.3 Negation

The semantics of a negated interaction $\mathsf{neg}(S)$ is classically defined by making positive for $\mathsf{neg}(S)$ all those traces that are not positive for S and making negative for $\mathsf{neg}(S)$ all those traces that are positive for S. Such a definition, however, rules out inconclusive

traces. In general, thus, we need to distinguish *positive*, *negative*, and *inconclusive* runs for an interaction. We write $t \models_{\mathrm{n}} S$ if t *negatively* satisfies S. The inductive definition of $\models_{\mathrm{p}}$ is extended and the relation $\models_{\mathrm{n}}$ is inductively defined on the structure of S as shown in Tab. 2(b).

In particular, we define $\models_{\mathrm{n}}$ for all combined interaction fragments and, in accordance with Haugen and Stølen [3], we regard the empty trace as being positive for $\mathsf{neg}(S)$. For the combined fragments $\mathsf{strict}(-,-)$ and $\mathsf{seq}(-,-)$ we adopt the view that only those traces are negative that either run through the first operand negatively or fulfil the first operand positively but the second operand negatively. A similar stance is taken towards $\mathsf{par}(-,-)$ where either both operands have to be run through negatively or one of the operands negatively the other one positively in order to make a run negative. In $\mathsf{alt}(-,-)$ both operands have to be run through negatively. Our semantics for assertion is the "assertion as affirmation" interpretation of Störrle [8].

Störrle [8] considers three different interpretations of $\mathsf{neg}(S)$. All of them coincide in declaring negative for $\mathsf{neg}(S)$ all those traces that are positive for S. For the positive traces of $\mathsf{neg}(S)$, interpretation (1), called "not the [valid] traces of S", assigns no positive traces to $\mathsf{neg}(S)$; interpretation (2), called "anything but the [valid] traces of S", makes all traces that are not positive for S the positive traces of $\mathsf{neg}(S)$; interpretation (3) declares the negative traces of S to be the positive traces for $\mathsf{neg}(S)$. Employing the interpretations (1) or (3), the usage of negation inside combined fragments leads to the undesirable consequence that the overall interaction shows no positive traces at all. Interpretation (2) excludes the possibility of inconclusive traces for $\mathsf{neg}(S)$.

4.4 Examples

Let B_i be the basic interactions $\{\mathsf{snd}(s_i, r_i, m_i) \leq \mathsf{rcv}(s_i, r_i, m_i)\}$ $(i = 1, 2, 3)$, where all m_i are different, and let t_i be the traces $\mathsf{snd}(s_i, r_i, m_i) \cdot \mathsf{rcv}(s_i, r_i, m_i)$ $(i = 1, 2, 3)$. We then have that

- $t_1 \models_{\mathrm{p}} \mathsf{strict}(B_1, \mathsf{neg}(B_2))$
- $t_1; t_2 \models_{\mathrm{n}} \mathsf{strict}(B_1, \mathsf{neg}(B_2))$
- $t_1; t_2 \models_{\mathrm{n}} \mathsf{strict}(B_1, \mathsf{strict}(\mathsf{neg}(B_2), B_3))$
- $t_1; t_3 \models_{\mathrm{p}} \mathsf{strict}(B_1, \mathsf{strict}(\mathsf{neg}(B_2), B_3))$
- $t_1; t_2; t_3 \models_{\mathrm{n}} \mathsf{strict}(B_1, \mathsf{strict}(\mathsf{neg}(B_2), B_3))$
- $t_2 \models_{\mathrm{p}} \mathsf{par}(\mathsf{neg}(B_2), B_2)$ and
- $t_2 \not\models_{\mathrm{n}} \mathsf{par}(\mathsf{neg}(B_2), B_2)$.

A more interesting case is given by the following two facts:

- $t_2 \models_{\mathrm{p}} \mathsf{strict}(\mathsf{neg}(B_2), B_2)$ and
- $t_2 \models_{\mathrm{n}} \mathsf{strict}(\mathsf{neg}(B_2), B_2)$.

Thus, t_2 is simultaneously positive and negative for $\mathsf{strict}(\mathsf{neg}(B_2), B_2)$. We therefore call $\mathsf{strict}(\mathsf{neg}(B_2), B_2)$ an overspecified interaction.

Definition 1. *An interaction S is* overspecified *if there exists a trace t with $t \models_{\mathrm{p}} S$ and $t \models_{\mathrm{n}} S$.*

For the same B_2, a further overspecified interaction is $\mathsf{par}(\mathsf{assert}(B_2), \mathsf{neg}(B_2))$. The trace t_2 satisfies this interaction both positively and negatively.

4.5 Properties

It is easy to check that both forms of sequential composition are associative, and that parallel and alternative composition are associative and commutative.

Lemma 1. *Let S_1, S_2, and S_3 be interactions, and t be a trace.*

1. $t \models_{\mathrm{p}} \mathsf{strict}(S_1, \mathsf{strict}(S_2, S_3))$ *iff* $t \models_{\mathrm{p}} \mathsf{strict}(\mathsf{strict}(S_1, S_2), S_3)$
2. $t \models_{\mathrm{p}} \mathsf{seq}(S_1, \mathsf{seq}(S_2, S_3))$ *iff* $t \models_{\mathrm{p}} \mathsf{seq}(\mathsf{seq}(S_1, S_2), S_3)$
3. $t \models_{\mathrm{p}} \mathsf{par}(S_1, \mathsf{par}(S_2, S_3))$ *iff* $t \models_{\mathrm{p}} \mathsf{par}(\mathsf{par}(S_1, S_2), S_3)$
4. $t \models_{\mathrm{p}} \mathsf{par}(S_1, S_2)$ *iff* $t \models_{\mathrm{p}} \mathsf{par}(S_2, S_1)$
5. $t \models_{\mathrm{p}} \mathsf{alt}(S_1, \mathsf{alt}(S_2, S_3))$ *iff* $t \models_{\mathrm{p}} \mathsf{alt}(\mathsf{alt}(S_1, S_2), S_3)$
6. $t \models_{\mathrm{p}} \mathsf{alt}(S_1, S_2)$ *iff* $t \models_{\mathrm{p}} \mathsf{alt}(S_2, S_1)$

Furthermore, all these propositions also hold when replacing $\models_{\mathrm{p}}$ by $\models_{\mathrm{n}}$.

By abuse of notation we thus abbreviate e.g. $\mathsf{strict}(S_1, \mathsf{strict}(S_2, \mathsf{strict}(\ldots, S_n)))$ to $\mathsf{strict}(S_1, S_2, \ldots, S_n)$ and $\mathsf{alt}(S_1, \mathsf{alt}(S_2, \mathsf{alt}(\ldots, S_n)))$ to $\mathsf{alt}(S_1, S_2, \ldots, S_n)$.

Basic interactions are not negatively satisfiable.

Lemma 2. *$t \not\models_{\mathrm{n}} B$ for any basic interaction B and any trace t.*

The satisfaction relations $\models_{\mathrm{p}}$ and $\models_{\mathrm{n}}$ as defined in Sects. 4.2 and 4.3 are not conclusive, that is, there exist inconclusive traces.

Lemma 3. *There exist a trace t and an interaction S with $t \not\models_{\mathrm{p}} S$ and $t \not\models_{\mathrm{n}} S$.*

Proof. Take for S the basic interaction $B = \{\mathsf{snd}(s, r, m) \leq \mathsf{rcv}(s, r, m)\}$ and for t the trace $\mathsf{rcv}(s, r, m) \cdot \mathsf{snd}(s, r, m)$.

The operator $\mathsf{assert}(-)$ discards inconclusive traces of its operand, that is, it establishes the link between the semantics of interactions and classical two-valued logic.

Lemma 4. *Let S be an interaction and t be a trace. Then $t \models_{\mathrm{p}} \mathsf{assert}(S)$ or $t \models_{\mathrm{n}} \mathsf{assert}(S)$.*

On the syntactic structure of interactions we define a well-founded ordering, which can be used to demonstrate further properties of interactions by induction.

Definition 2. *We define a partial order on interaction terms as the reflexive and transitive closure of the following binary relation:*

$$
\begin{aligned}
\mathsf{skip} &\leq S & S &\leq \mathsf{neg}(S) \\
S_1 &\leq \mathsf{strict}(S_1, S_2) & S_2 &\leq \mathsf{strict}(S_1, S_2) \\
S_1 &\leq \mathsf{seq}(S_1, S_2) & S_2 &\leq \mathsf{seq}(S_1, S_2) \\
S_1 &\leq \mathsf{par}(S_1, S_2) & S_2 &\leq \mathsf{par}(S_1, S_2) \\
S_1 &\leq \mathsf{alt}(S_1, S_2) & S_2 &\leq \mathsf{alt}(S_1, S_2) \\
S &\leq \mathsf{ignore}(M, S) & S &\leq \mathsf{assert}(S) \\
\mathsf{seq}(S, \mathsf{loop}(m, n, S)) &\leq \mathsf{loop}(m, n+1, S) & \mathsf{loop}(m, n, S) &\leq \mathsf{loop}(m, \infty, S)
\end{aligned}
$$

where S, S_1, and S_2 are arbitrary interactions, and m and n natural numbers.

Lemma 5. *The above defined ordering $\leq$ on interactions is well founded, i.e., there exists no infinite descending chain $S_1 \geq S_2 \geq \cdots \geq S_n \geq \cdots$.*

5 Negation Revisited

The non-classical interpretation of negation is difficult to deal with. Above all, the need for two satisfaction relations, one positive and one negative, makes it hard to decide what kind of trace is a given one for an interaction, i.e., if the trace is positive, inconclusive, or negative for the interaction. When reconsidering the semantics introduced in the previous section, we firstly discover that negation is unnecessary for testing positive satisfaction. Secondly, when handling negative satisfaction, negation is of course needed, but it can be replaced by a classical version. We introduce therefore the language of *interactions in the classical sense* as opposed to the language of UML 2.0 interactions considered so far.

We begin by reinvestigating the interplay between negation and positive satisfaction. Observe that a negative interaction can only be positively satisfied by the empty process, the same as skip. It therefore seems natural, when it comes to check positive satisfaction, to replace any negative subinteraction by skip.

Definition 3. *The function σ from interactions to interactions of the positive fragment is given by induction on the syntactic structure of its argument as follows:*

$$\begin{aligned}
\sigma(B) &= B \\
\sigma(\mathsf{strict}(S_1, S_2)) &= \mathsf{strict}(\sigma(S_1), \sigma(S_2)) \\
\sigma(\mathsf{seq}(S_1, S_2)) &= \mathsf{seq}(\sigma(S_1), \sigma(S_2)) \\
\sigma(\mathsf{par}(S_1, S_2)) &= \mathsf{par}(\sigma(S_1), \sigma(S_2)) \\
\sigma(\mathsf{loop}(m, \overline{n}, S)) &= \mathsf{loop}(m, \overline{n}, \sigma(S)) \\
\sigma(\mathsf{ignore}(M, S)) &= \mathsf{ignore}(M, \sigma(S)) \\
\sigma(\mathsf{alt}(S_1, S_2)) &= \mathsf{alt}(\sigma(S_1), \sigma(S_2)) \\
\sigma(\mathsf{neg}(S)) &= \mathsf{skip} \\
\sigma(\mathsf{assert}(S)) &= \sigma(S)
\end{aligned}$$

where B ranges over basic interactions, S, S_1, and S_2 over interactions, M over sets of messages, m over the natural numbers, and $\overline{n}$ over the natural numbers or ∞.

Lemma 6. *Let S be an interaction and t be a trace. Then, $t \models_{\mathrm{p}} S$ iff $t \models_{\mathrm{p}} \sigma(S)$.*

This means that the positive fragment of the language and the positive satisfaction relation defined for it as given in Sect. 4.2 and in Tab. 2(a) are sufficient for testing positive satisfaction of arbitrary interactions.

Now we turn our attention to negative satisfaction. The question is if something similar cannot be done for it as well. More precisely, it would be advantageous to get rid of the negative satisfaction relation by defining it in terms of the positive one. This is obviously true for a sublanguage, namely for sequences of interactions involving a negated subinteraction.

Lemma 7. *Let $S = \mathsf{strict}(S_1, \mathsf{neg}(S'), S_2)$ be an interaction with S_1, S', and S_2 from the positive fragment and t be a trace. Then, $t \models_{\mathrm{n}} S$ iff there exists a prefix t' of t such that $t' \models_{\mathrm{p}} \mathsf{strict}(S_1, S')$.*

This result, however, cannot be generalised to the full language of interactions: a binary logic without negation is not enough. A binary logic with classical negation, on the contrary, does suffice. We add an operator $\mathsf{not}(-)$ to the positive fragment of UML 2.0 interactions, which gives rise to the so-called interactions in classical sense. This new unary operator is provided with the classical semantics of negation. We define a transformation from UML 2.0 interactions to interactions in the classical sense, and show that the positive satisfaction of the resulting interaction is equivalent to negative satisfaction of the given one.

Definition 4. *The syntax of* interactions in the classical sense *is given by the syntax in Tab. 1 where* $\mathsf{neg}(-)$ *and* $\mathsf{assert}(-)$ *are removed and* $\mathsf{not}(-)$ *is added to CombinedFragment.*

The positive semantics of interactions in the classical sense is given by the semantics for the positive fragment of UML 2.0 interactions in Tab. 2(a) and

$$t \models_{\mathrm{p}} \mathsf{not}(S) \quad \textit{if } t \not\models_{\mathrm{p}} S$$

We furthermore use the following abbreviations:

$$\begin{aligned} \mathsf{Any} &= \mathsf{ignore}(\mathbb{M}, \mathsf{skip}) \\ \mathsf{None} &= \mathsf{not}(\mathsf{Any}) \\ \mathsf{and}(S_1, S_2) &= \mathsf{not}(\mathsf{alt}(\mathsf{not}(S_1), \mathsf{not}(S_2))) \end{aligned}$$

Definition 5. *The function ν from UML 2.0 interactions to interactions in the classical sense is given by induction on the syntactic structure of its argument as follows:*

$$\begin{aligned} \nu(B) &= \mathsf{None} \\ \nu(\mathsf{strict}(S_1, S_2)) &= \mathsf{alt}(\mathsf{strict}(\nu(S_1), \mathsf{Any}), \mathsf{strict}(\sigma(S_1), \nu(S_2))) \\ \nu(\mathsf{seq}(S_1, S_2)) &= \mathsf{alt}(\mathsf{seq}(\nu(S_1), \mathsf{Any}), \mathsf{seq}(\sigma(S_1), \nu(S_2))) \\ \nu(\mathsf{par}(S_1, S_2)) &= \mathsf{alt}(\mathsf{par}(\nu(S_1), \nu(S_2)), \mathsf{par}(\nu(S_1), \sigma(S_2)), \mathsf{par}(\sigma(S_1), \nu(S_2))) \\ \nu(\mathsf{loop}(m, \overline{n}, S)) &= \mathsf{and}(\mathsf{loop}(m, \overline{n}, \nu(S)), \mathsf{not}(\mathsf{skip})) \\ \nu(\mathsf{ignore}(M, S)) &= \mathsf{ignore}(M, \nu(S)) \\ \nu(\mathsf{alt}(S_1, S_2)) &= \mathsf{and}(\nu(S_1), \nu(S_2)) \\ \nu(\mathsf{neg}(S)) &= \mathsf{and}(\sigma(S), \mathsf{not}(\mathsf{skip})) \\ \nu(\mathsf{assert}(S)) &= \mathsf{not}(\sigma(S)) \end{aligned}$$

where B ranges over basic interactions, S, S_1, and S_2 over interactions, M over sets of messages, m over the natural numbers, and $\overline{n}$ over the natural numbers or ∞.

Lemma 8. *Let S be a UML 2.0 interaction and t be trace. Then $t \models_{\mathrm{n}} S$ iff $t \models_{\mathrm{p}} \nu(S)$.*

Proof. By induction on the partial ordering $\leq$ on UML 2.0 interactions.

Summarising, a closer look at negation leads to the following two results:

$$\begin{aligned} t \models_{\mathrm{p}} S \quad & \text{if } t \models_{\mathrm{p}} \sigma(S) \\ t \models_{\mathrm{n}} S \quad & \text{if } t \models_{\mathrm{p}} \nu(S) \end{aligned}$$

where S is an arbitrary UML 2.0 interaction, $\sigma(S)$ is an interaction of the positive fragment of the language of UML 2.0 interactions obtained in terms of S, and $\nu(S)$ is an interaction in the classical sense in terms of S. Notice that the positive fragment of the language of UML 2.0 interactions is also the positive fragment of the language of interactions in classical sense. More importantly, by means of these two transformations, σ and ν, we do not need to test negative satisfaction. This observation may be useful for checking overspecification, but we defer a closer investigation to future work.

6 Implementation and Refinement

Having a formal semantics for interactions, further concepts can be defined in terms of it. We introduce the notions of implementation of an interaction by a process, of equivalence of interactions, and of refinement of an interaction by another one. These notions show a number of useful properties, and are intended for formal verification.

Definition 6. *A process I is an* implementation *of an interaction S, written $I \models S$, if*

1. *there exists $t \in lin(I)$ with $t \models_{\mathrm{p}} S$, and*
2. *$t \not\models_{\mathrm{n}} S$ for every $t \in lin(I)$.*

An interaction S is implementable *if there is a process I such that $I \models S$; it is* contradictory *if it is not implementable.*

The following lemma ensures that any interaction admits positive traces and thus that the first condition of the implementation relation is always satisfiable.

Lemma 9. *For every interaction S there exists a trace t with $t \models_{\mathrm{p}} S$.*

Proof. By induction on the partial ordering $\leq$ and the fact that $\varepsilon \models_{\mathrm{p}} \mathsf{neg}(S)$.

This lemma, however, does not imply that any interaction is implementable. Indeed, having a positive trace is not enough, since this very trace may also be negative for the same interaction. Take for instance the overspecified interaction $\mathsf{strict}(\mathsf{neg}(B_2), B_2)$ of Sect. 4.4: its only positive trace t_2 is at the same time negative. Nonetheless, an overspecified interaction may be implementable, that is, overspecified interactions are not necessarily contradictory. Take for instance the interaction $S = \mathsf{alt}(\mathsf{seq}(\mathsf{neg}(B_2), B_1), \mathsf{seq}(\mathsf{neg}(B_2), B_2))$ with B_1 and B_2 as in Sect. 4.4. The trace t_2 is both positive and negative for S, i.e., both $t_2 \models_{\mathrm{p}} S$ and $t_2 \models_{\mathrm{n}} S$, whereas the trace t_1 is only positive for S, i.e., $t_1 \models_{\mathrm{p}} S$ and $t_1 \not\models_{\mathrm{n}} S$. Thus $\{t_1\} \models S$.

Moreover, note that a combination of interactions, each equipped with its own implementation, not necessarily is implemented by the same combination of the corresponding implementations. Take for instance $S_1 = \mathsf{neg}(B_1)$, $S_2 = \mathsf{neg}(B_2)$, $I_1 = \{t_2, \varepsilon\}$, and $I_2 = \{t_1, \varepsilon\}$, with B_i and t_i as defined in Sect. 4.4 ($i = 1, 2$). It is easy to check that, while $I_i \models S_i$ ($i = 1, 2$), it is not true that $I_1 \parallel I_2 \models \mathsf{par}(S_1, S_2)$.

A notion of implementation allows the definition of an equivalence relation.

Definition 7. *Two interactions S_1 and S_2 are* equivalent*, denoted by $S_1 \equiv S_2$, whenever $I \models S_1$ iff $I \models S_2$ for any process I.*

Furthermore, the implementation relation gives rise to a model-theoretic notion of refinement.

Definition 8. *An interaction S'* refines *an interaction S, written $S \rightsquigarrow S'$, if any implementation of S' is also an implementation of S, i.e., if $I \models S'$ implies $I \models S$ for any implementation I.*

Lemma 10. *Refinement is a partial order w.r.t. the equivalence on interactions, i.e., refinement is reflexive, transitive, and antisymmetric w.r.t. $\equiv$.*

An example of an interaction refinement is provided by the removal of disjunctions.

Lemma 11. $\mathsf{alt}(S_1, S_2) \rightsquigarrow S_i$ *for* $i = 1, 2$.

Proof. Let $I \models S_1$. On the one hand, there exists $t \in lin(I)$ with $t \models_{\mathrm{p}} S_1$ and thus $t \models_{\mathrm{p}} \mathsf{alt}(S_1, S_2)$. On the other hand, $t \not\models_{\mathrm{n}} S_1$ and hence $t \not\models_{\mathrm{n}} \mathsf{alt}(S_1, S_2)$ for all $t \in lin(I)$. The case $I \models S_2$ is treated analogously.

Let us now investigate the properties of the refinement relation. As the following lemma shows, in refinement the set of genuine positive traces cannot be enlarged, negative traces remain negative, and at least one positive trace is kept.

Lemma 12. *Let S and S' be interaction with $S \rightsquigarrow S'$.*

1. *For all traces t, if $t \not\models_{\mathrm{p}} S$ or $t \models_{\mathrm{n}} S$, then $t \not\models_{\mathrm{p}} S'$ or $t \models_{\mathrm{n}} S'$.*
2. *If S' is implementable, then for all traces t, $t \models_{\mathrm{n}} S$ implies $t \models_{\mathrm{n}} S'$.*
3. *If S' is implementable, then there is a trace t such that $t \models_{\mathrm{p}} S$ and $t \models_{\mathrm{p}} S'$.*

Proof. For claim (1), suppose $t \models_{\mathrm{p}} S'$ and $t \not\models_{\mathrm{n}} S'$. Then $\{t\} \models S'$, and also $\{t\} \models S$ since $S \rightsquigarrow S'$. Thus $t \models_{\mathrm{p}} S$ and $t \not\models_{\mathrm{n}} S$ which contradicts $t \not\models_{\mathrm{p}} S$ or $t \models_{\mathrm{n}} S$.

For claim (2), suppose $t \not\models_{\mathrm{n}} S'$ and let I be any process such that $I \models S'$. Then also $I \cup \{t\} \models S'$, and thus $I \cup \{t\} \models S$ because $S \rightsquigarrow S'$, which contradicts $t \models_{\mathrm{n}} S$.

For claim (3), assume that $t \not\models_{\mathrm{p}} S$ for all $t \models_{\mathrm{p}} S'$. Since S' is implementable, there is a trace such that $t \models_{\mathrm{p}} S'$ but $t \not\models_{\mathrm{n}} S'$. Then $\{t\} \models S'$, but $\{t\} \not\models S$.

An inconclusive trace can indeed become negative. Recall for instance the interaction B_2 and the trace t_2 from Sect. 4.4: trace t_2 is inconclusive for skip and negative for $\mathsf{neg}(B_2)$, where $\mathsf{skip} \rightsquigarrow \mathsf{neg}(B_2)$. On the other hand, a positive trace may become inconclusive, as witnessed by Lemma 11.

A desirable property of refinement is that the operators be monotonic with respect to it. For instance, for a proof of monotonicity of disjunction w.r.t. refinement, we need to show that a process implementing $\mathsf{alt}(S_1', S_2)$ also implements $\mathsf{alt}(S_1, S_2)$ if $S_1 \rightsquigarrow S_1'$. Unfortunately this is not true. Consider the following constellation:

$$S_1 = B_1 \qquad S_1' = \mathsf{alt}(\mathsf{seq}(\mathsf{neg}(B_2), B_1), \mathsf{seq}(\mathsf{neg}(B_2), B_2))$$
$$S_2 = B_3 \qquad t = t_2$$

where B_1, B_2, B_3, and t_2 are the interactions resp. trace of Sect. 4.4; in particular, $S_1 \rightsquigarrow S_1'$. We have then the following facts:

$$t \models_{\mathrm{p}} S_1' \text{ and } t \models_{\mathrm{n}} S_1' \qquad t \not\models_{\mathrm{p}} S_1$$
$$t \not\models_{\mathrm{p}} S_2 \text{ and } t \not\models_{\mathrm{n}} S_2$$

$$\frac{S_1 \rightsquigarrow_{\mathrm{p}} S_1'}{\mathsf{strict}(S_1, S_2) \rightsquigarrow_{\mathrm{p}} \mathsf{strict}(S_1', S_2)} \qquad \frac{S_2 \rightsquigarrow S_2'}{\mathsf{strict}(S_1, S_2) \rightsquigarrow \mathsf{strict}(S_1, S_2')}$$

$$\frac{S_1 \rightsquigarrow_{\mathrm{p}} S_1'}{\mathsf{seq}(S_1, S_2) \rightsquigarrow_{\mathrm{p}} \mathsf{seq}(S_1', S_2)} \qquad \frac{S_2 \rightsquigarrow S_2'}{\mathsf{seq}(S_1, S_2) \rightsquigarrow \mathsf{seq}(S_1, S_2')}$$

$$\frac{S_1 \rightsquigarrow_{\mathrm{p}} S_1'}{\mathsf{par}(S_1, S_2) \rightsquigarrow_{\mathrm{p}} \mathsf{par}(S_1', S_2)} \qquad \frac{S_1 \rightsquigarrow S_1'}{\mathsf{alt}(S_1, S_2) \rightsquigarrow \mathsf{alt}(S_1', S_2)}$$

$$\frac{S \rightsquigarrow S'}{\mathsf{neg}(S') \rightsquigarrow \mathsf{neg}(S)} \qquad \frac{S \rightsquigarrow S'}{\mathsf{assert}(S) \rightsquigarrow \mathsf{assert}(S')}$$

Table 3. Compositional refinements of interactions

that is, $\{t\} \models \mathsf{alt}(S_1', S_2)$ and $\{t\} \not\models \mathsf{alt}(S_1, S_2)$, i.e., $\mathsf{alt}(S_1, S_2) \not\rightsquigarrow \mathsf{alt}(S_1', S_2)$.

When restricting ourselves to refinements by non-overspecified interactions, disjunction indeed is monotonic w.r.t. refinement.

Lemma 13. *Let S_1, S_1', and S_2 be interactions and let S_1' be implementable and not overspecified. If $S_1 \rightsquigarrow S_1'$, then $\mathsf{alt}(S_1, S_2) \rightsquigarrow \mathsf{alt}(S_1', S_2)$.*

Proof. Let I be a process such that $I \models \mathsf{alt}(S_1', S_2)$. Let $t \in lin(I)$ be a trace of I. Then $t \not\models_{\mathrm{n}} \mathsf{alt}(S_1', S_2)$. In particular, $t \not\models_{\mathrm{n}} S_1'$ or $t \not\models_{\mathrm{n}} S_2$ and thus, by Lemma 12(2), $t \not\models_{\mathrm{n}} S_1$ or $t \not\models_{\mathrm{n}} S_2$, that is, $t \not\models_{\mathrm{n}} \mathsf{alt}(S_1, S_2)$.

Moreover, there is a $t \in lin(I)$ with $t \models_{\mathrm{p}} \mathsf{alt}(S_1', S_2)$, i.e., $t \models_{\mathrm{p}} S_1'$ or $t \models_{\mathrm{p}} S_2$. If $t \models_{\mathrm{p}} S_2$ then $t \models_{\mathrm{p}} \mathsf{alt}(S_1, S_2)$. If $t \models_{\mathrm{p}} S_1'$ then $t \not\models_{\mathrm{n}} S_1'$, as S_1' is not overspecified, and thus $t \models_{\mathrm{p}} S_1$ by Lemma 12(1); hence again $t \models_{\mathrm{p}} \mathsf{alt}(S_1, S_2)$.

However, for proving the monotonicity of the sequential operators in the first argument w.r.t. refinement, the restriction to refinements by non-overspecified interactions is not enough. In fact, in demonstrating that $S_1 \rightsquigarrow S_1'$ implies $\mathsf{strict}(S_1, S_2) \rightsquigarrow \mathsf{strict}(S_1', S_2)$, we have to assume that all positive traces of S_1 are still positive in S_1': If a positive trace of S_1 becomes inconclusive in S_1', there may be more negative traces in $\mathsf{strict}(S_1, S_2)$ than in $\mathsf{strict}(S_1', S_2)$. We therefore introduce a restricted refinement relation $\rightsquigarrow_{\mathrm{p}}$ that keeps all positive traces.

Definition 9. *An interaction S'* positively refines *an interaction S, written $S \rightsquigarrow_{\mathrm{p}} S'$, if S' refines S and for all traces t it holds: if $t \models_{\mathrm{p}} S$ then $t \models_{\mathrm{p}} S'$.*

Some results on the monotonicity of interaction operators w.r.t. the refinement relations $\rightsquigarrow$ and $\rightsquigarrow_{\mathrm{p}}$ are summarised in Tab. 3, where S, S_1, S_2, S', S_1', and S_2' are interactions and S', S_1' and S_2' are implementable and not overspecified. A more complete calculus for interaction refinement is subject of future study.

7 Discussion

Lemma 7 concludes that, for a given interaction, any trace is negative if it completely traverses a negative region, independently of the steps performed afterwards, if any. The

proposal of Haugen and Stølen [3] states that "[. . .] any trace that [completely traverses a negative region] is a negative scenario. Anything may happen [afterwards], it will never make it positive." It is not explicitly said that further steps cannot make the trace inconclusive. If in particular their proposal allows the trace to become inconclusive, then the semantics of Sect. 4 above is more restrictive.

Indeed, this is not a merely speculation of ours. The example used there is that of a restaurant, where a customer orders and is served a beef, including an inbetween negative subinteraction that forbids to burn the meat. Intuitively, hence, if in fact the meat burns in the oven, the obvious thing to do is to take it from the oven and not to bring it to the customer's table. This means that a trace is only negative if, after traversing the negative region, the next positive region is exhaustively traversed as well. It therefore seems that a trace is negative if it traverses all positive regions plus at least one negative region. A big disadvantage of this interpretation is that a semantic definition for it cannot be compositional. Compositionality is not just a comfortable mathematical property, it allows for instance an on-the-fly recognition of a negative trace (or to warn a running system from generating a negative trace), since decisions are taken locally, i.e., independently of what happened before or what will happen henceforth.

The semantics of Sect. 4, plainly worded, states that "the trace is *bad* as soon as it leaves a negative region, it is *good* if both it is exhaustive (i.e., the interaction does not specify any event beyond the trace's last event) and it only traverses positive regions, and it is *inconclusive* otherwise." The key point here is that a trace, which has completely traversed a negative region, is definitively negative. We do think that this is a better choice, and hence put it at the community's disposal for discussion.

A further deviation from the proposal of Haugen and Stølen [3] is the existence of overspecified interactions. The cited work states that "the same trace cannot be both positive and negative." We dispute the convenience of this requirement. Consider any of the overspecified interactions shown above, and a trace that is both positive and negative for the interaction. It is by far not obvious how to rule out one of both possibilities (i.e., deciding if the trace is positive or negative) in a non-arbitrary manner, and making this trace inconclusive is capricious.

Let us finally consider the concepts of supplementing, narrowing and detailing by Haugen and Stølen [3]. Supplementing means reducing the set of inconclusive traces by making some of them either positive or negative; in doing so, positive (negative) traces remain positive (negative). Narrowing means reducing the set of positive traces by making some of them negative; inconclusive (negative) traces remain inconclusive (negative). Detailing consists in providing a translation from a more detailed (concrete) interaction to a given (abstract) interaction; it leaves the sets of positive, negative, and inconclusive traces unchanged. These notions are colloquially defined using the three types of traces associated with an interaction; as with our refinement relation, there is no clue on how to define those in syntactical terms. Our refinement relation is somehow supplementing and narrowing at the same time; supplementing cannot be defined in terms of refinement, since supplementing may make positive an inconclusive trace. The spirit behind all these concepts, however, makes them difficult to compare, since supplementing and narrowing address design evolution, whereas refinement is a tool for formal verification.

8 Conclusions and Outlook

The contribution of the present article is twofold. On the one hand, it defines a semantics for UML 2.0 interactions that is both formal and consistent with the standard [6]. This proposal is compared with earlier ones. On the other hand, a formal semantics allows a mathematically precise definition of implementation and of refinement, such that these relations can be formally proved. These notions show some desirable and some questionable properties, so that they may be subject to further adjustments. They nevertheless set the ground for lifting UML 2.0 to a formal design technique, a sine qua non for its use in the development of critical systems.

Some UML 2.0 operators for interactions were disregarded, namely break and critical, and also message parameters, conditions, and time. We plan to extend the semantics above to include these other features of UML 2.0. The semantics for OCL/RT of [1] can be a good starting point for traces which include time and on which conditions are checkable. A calculus for formal verification is the utmost challenge. This matter can be addressed once implementation and refinement have reached a stable, i.e., broadly accepted, definition.

Acknowledgements. We thank Øystein Haugen and Harald Störrle for fruitful discussions.

References

1. María Victoria Cengarle and Alexander Knapp. Towards OCL/RT. In Lars-Henrik Eriksson and Peter Alexander Lindsay, editors, *Proc. 11th Int. Symp. Formal Methods Europe (FME'02)*, volume 2391 of *Lect. Notes Comp. Sci.*, pages 390–409. Springer, Berlin, 2002.
2. Werner Damm and David Harel. LSCs: Breathing Life into Message Sequence Charts. *Formal Methods in System Design*, 19(1):45–80, 2001.
3. Øystein Haugen and Ketil Stølen. STAIRS — Steps to Analyze Interactions with Refinement Semantics. In Perdita Stevens, Jon Whittle, and Grady Booch, editors, *Proc. 6th Int. Conf. Unified Modeling Language (UML'03)*, volume 2863 of *Lect. Notes Comp. Sci.*, pages 388–402. Springer, Berlin, 2003.
4. International Telecommunication Union. Message Sequence Chart (MSC). Recommendation Z.120, ITU-T, Genève, 1999.
5. Alexander Knapp. A Formal Semantics for UML Interactions. In Robert France and Bernhard Rumpe, editors, *Proc. 2nd Int. Conf. Unified Modeling Language (UML'99)*, volume 1723 of *Lect. Notes Comp. Sci.*, pages 116–130. Springer, Berlin, 1999.
6. Object Management Group. Unified Modeling Language Specification, Version 2.0 (Superstructure). Adopted draft, OMG, 2003. `http://www.omg.org/cgi-bin/doc?ptc/03-08-02`.
7. Vaughan Pratt. Modeling Concurrency with Partial Orders. *Int. J. Parallel Program.*, 15(1):33–71, 1986.
8. Harald Störrle. Assert, Negate and Refinement in UML-2 Interactions. In Jan Jürjens, Bernhard Rumpe, Robert France, and Eduardo B. Fernandez, editors, *Proc. Wsh. Critical Systems Development with UML (CSDUML'03)*, San Francisco, 2003. Technische Universität München, Technical report TUM-I0317.
9. Harald Störrle. Semantics of Interactions in UML 2.0. In *Proc. IEEE Symp. Visual Languages and Formal Methods (VLFM'03)*, Auckland, 2003.