

UML 2 Interactions

Formal Semantics

(Part 1: Conventional Operators)

Dr. Harald Störrle

Ludwig-Maximilians-Universität München
stoerrle@informatik.uni-muenchen.de

UML 2 Interactions

Formal Semantics

(Part 2: Assert, Negate, Refine)

Dr. Harald Störrle

Ludwig-Maximilians-Universität München
stoerrie@informatik.uni-muenchen.de

ran out of traveling funds

Dr. Alexander Knapp

Ludwig-Maximilians-Universität München
knapp@informatik.uni-muenchen.de

Overview of Talk

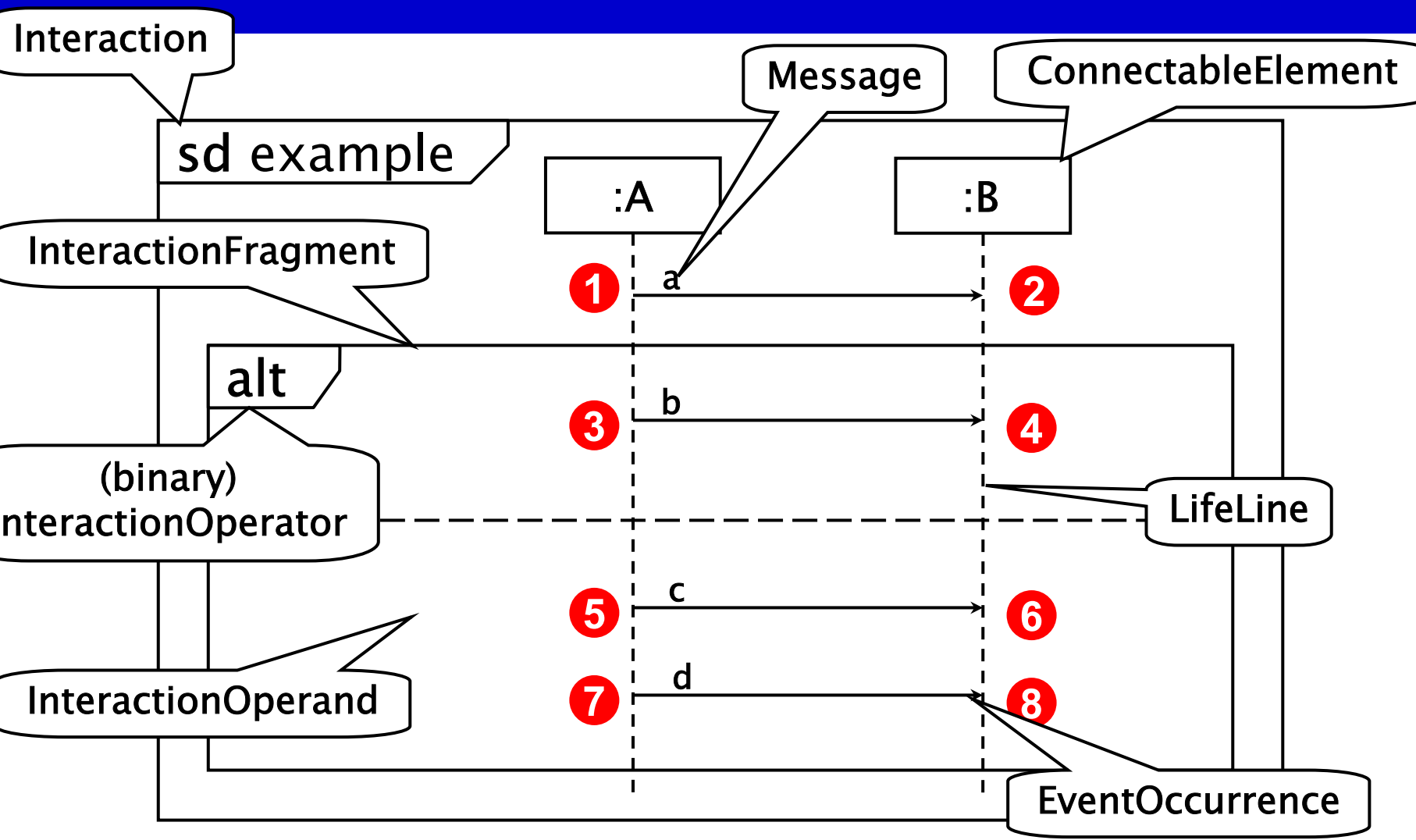
- Motivation
- Metamodel embedding of concrete syntax
- Semantic domains
- Simple operators
 - strict, alt, opt, par, break
- Advanced features
 - Weak sequencing
 - Reference
 - Negation, Assertion, Refinement
 - Time
- Summary
- Related / Further Work

Motivation

- UML is the lingua franca of software engineering.
- The new version 2.0 of the standard (moving target, basis of talk „final adopted spec“ of July, 3rd 2003) is a substantial improvement of UML 1.4.
- Many of the improvements concern Interactions.
- Formal Semantics is instrumental for many purposes
 - CASE–tool construction (...MDA)
 - Critical Systems Development
 - clear and unequivocal specifications

=> model exchange, network effect of UML

Concrete and abstract syntax – an introductory example



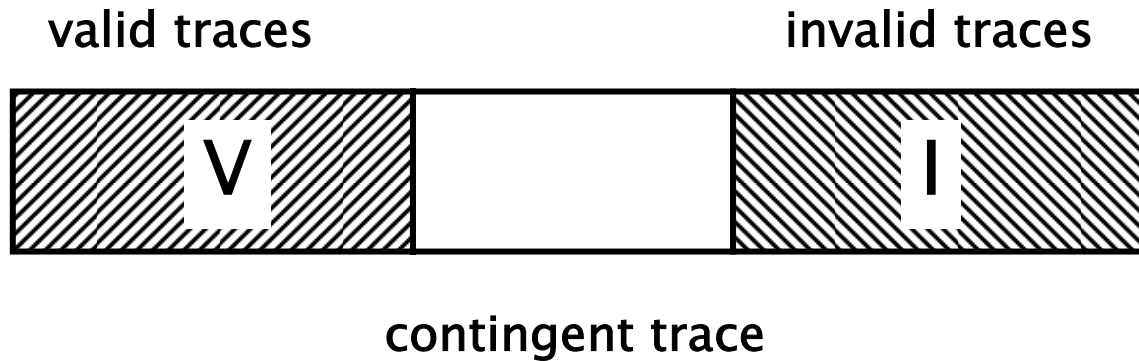
This sequence diagram represents the regular expression $a(b|cd)$.

Basic Semantic Notions – EventOccurrence and traces

- The standard says:
 - „*The semantics of an Interaction is given as a pair of sets of traces [of EventOccurrences].*“
- This clearly suggests an interleaving trace semantics, but the standard only says:
 - „*[...] trace–semantics [...] is a **preferred** way to describe the semantics of Interactions.*“
 - „*To **explain** Interactions we **apply** an Interleaving Semantics.*“
- This seems to imply, that there are other acceptable ways to explain (or define) the semantics of Interactions, in particular, partial order semantics like Petri–nets or Partial Words.

Basic Semantic Notions – valid and invalid traces

- The standard says:
 - „*The semantics of an Interaction is given by a pair $\langle V, I \rangle$ where V is the set of valid traces and I is the set of invalid traces [of EventOccurrences].*
 - *$V \cup I$ needs not be the whole universe of traces.“*



- For most operators, only valid traces are defined.

Basic operators: strict, alt, opt, par, break

- Specialties

- Gates are just syntactic abbreviation, may be resolved before semantic interpretation.
- Critical regions (atomic groups of EOs) are macro expansion, must be resolved after interpretation.
- Simple InteractionFragments are just partial order of EOs with additional constraints („GeneralOrdering“).

- Semantic Domains

- An alphabet \mathcal{EO} of EventOccurrences
- A Domain \mathcal{EO}^* of traces
- A semantic function from InteractionFragments to sets of traces, i.e. $[[_]] : \mathcal{IF} \rightarrow P(\mathcal{EO}^*)$

Basic operators: strict, alt, opt, par, break

- Simple Operators

- $[[\text{strict}(\mathcal{P}, \mathcal{Q})]]$ = $[[\mathcal{P}]] \cdot [[\mathcal{Q}]]$
- $[[\text{alt}(\mathcal{P}, \mathcal{Q})]]$ = $[[\mathcal{P}]] \cup [[\mathcal{Q}]]$
- $[[\text{opt}(\mathcal{P})]]$ = $[[\mathcal{P}]] \cup \{\varepsilon\}$
- $[[\text{par}(\mathcal{P}, \mathcal{Q})]]$ = $[[\mathcal{P}]] \sqcup [[\mathcal{Q}]]$
- $[[\text{break}(\mathcal{P})]]$ = $\text{prefixes}([[\mathcal{P}]])$

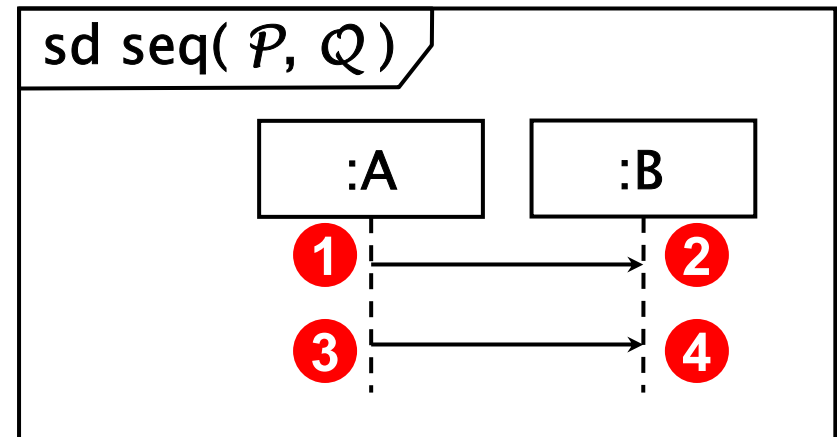
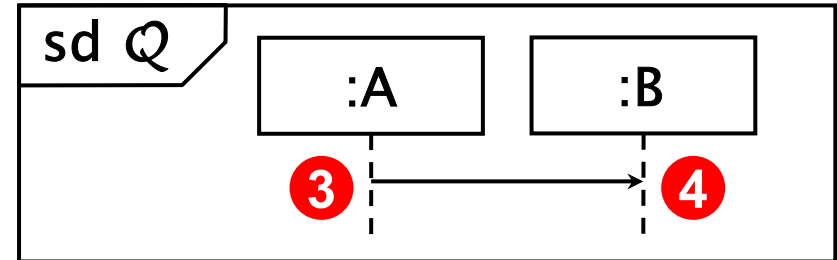
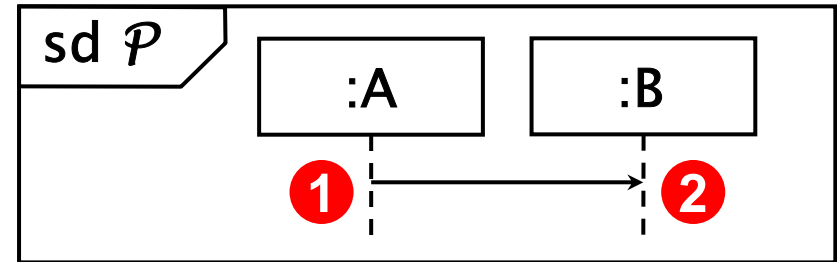
- Auxiliary function „shuffle“:

- $e \sqcup w$ = w
- $w \sqcup e$ = w
- $x.v \sqcup y.w$ = $\{x.(v \sqcup y.w),$
 $y.(x.v \sqcup w)\}$

Weak sequencing operator: seq

- Should really be called „partial order composition“ or so.
- Composes by lifelines, preserving order of EOs within lifelines but not across.
- Resulting traces may be „skewed“.
- Formal definition too technical to present (see paper).

$$\llbracket \text{seq}(\mathcal{P}, \mathcal{Q}) \rrbracket - \llbracket \text{strict}(\mathcal{P}, \mathcal{Q}) \rrbracket = \{ \langle \text{EO}_1 \cdot \text{EO}_3 \cdot \text{EO}_2 \cdot \text{EO}_4 \rangle \}$$



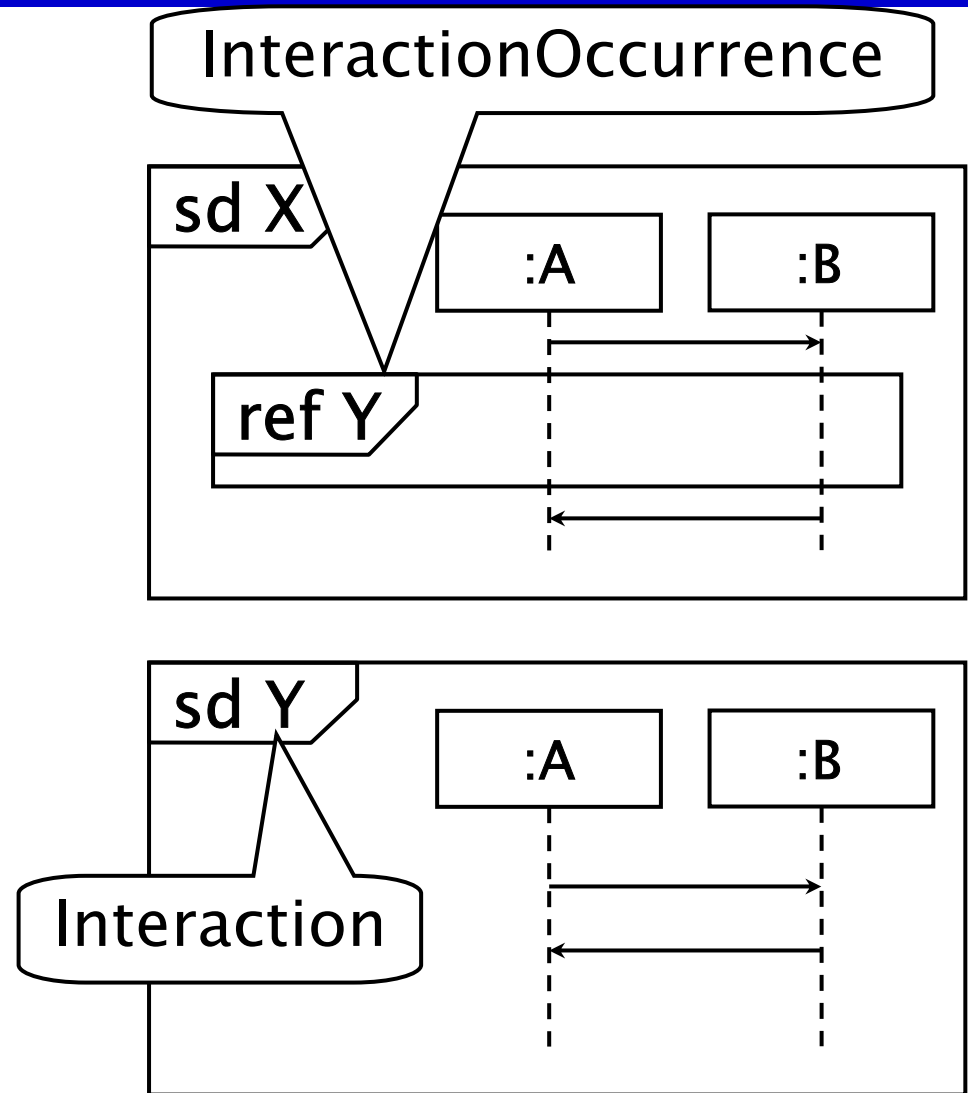
Iteration operator: loop

- Iteration expression with minimum and maximum durations, including infinity (denoted „*“).
- Syntax:
 - $\text{loop}(\mathcal{P}, \text{min}, \text{max})$
- Semantics:
 - $[[\text{loop}(\mathcal{P}, \text{min}, \text{max})]] = [[\text{loop}(\mathcal{P}, \text{min}, \text{max}, 0)]]$
 - where
 - $[[\text{loop}(\mathcal{P}, \text{min}, \text{max}, i)]] =$

$\{\varepsilon\}$	if $i \geq \text{max}$
$[[\text{opt}(\text{loop}(\mathcal{P}, \text{min}, \text{max}, i+1))]]$	if $\text{min} \leq i < \text{max}$
$[[\mathcal{P}]]^*$	if $\text{min} \leq i, \text{max} = *$
$[[\text{seq}(\mathcal{P}, \text{loop}(\mathcal{P}, \text{min}, \text{max}, i+1))]]$	if $i < \text{min}$

InteractionOccurrence: ref

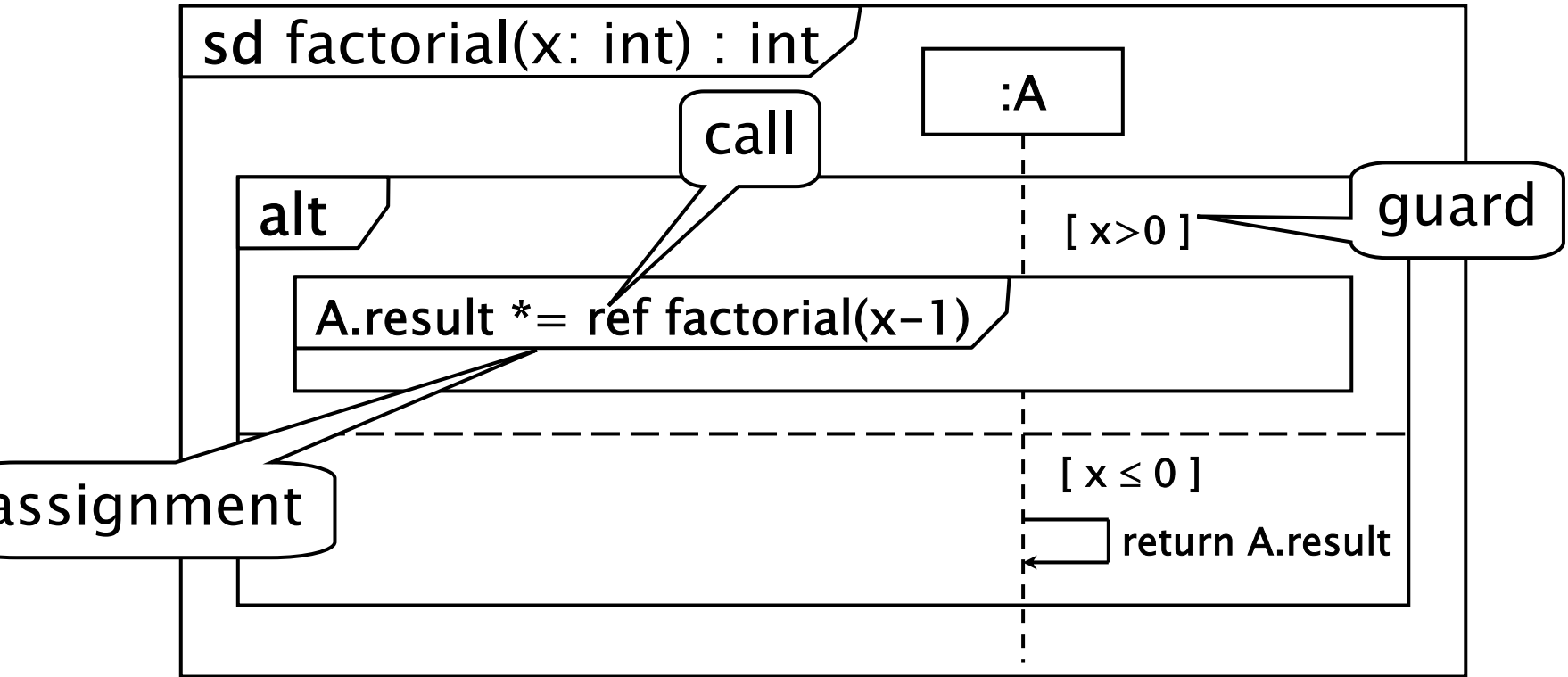
- Ref is not an operator – but what is it?
- Is it more like a macro expansion or more like a procedure call?
- Does it make a difference, semantically?
- Do we care?



Semantics of InteractionOccurrences: procedure call or macro expansion?

- *„An InteractionOccurrence refers to an Interaction. The InteractionOccurrence is a shorthand for copying the contents of the referred Interaction where the InteractionOccurrence is.“*
- Sounds like macro-expansion, but neither clear definition nor discriminative example in the standard.
- Macro-expansion would raise many questions:
 - When are macros expanded? Recursion? Parameters?
- But so would procedure call-semantics:
 - call by value, call by name, or something else?

„Ref“ makes sequence diagrams recursive



What does this mean with a macro-expansion-semantics?
Looks more like VHDL than Ada.

Negation operator: neg

There are no examples in the standard for the neg-operator.

- 1) „*The InteractionOperator neg designates that the CombinedFragment represents traces that are defined to be invalid.*“

$$[[\text{neg}(\mathcal{P})]] = \langle \emptyset, [[\mathcal{P}]]^+ \rangle \quad (\text{loose})$$

- 2) „*All InteractionFragments that are different from negative are considered positive [...]*“

$$[[\text{neg}(\mathcal{P})]] = \langle \mathcal{E}\mathcal{O}^* / [[\mathcal{P}]]^+ , [[\mathcal{P}]]^+ \rangle \quad (\text{strict})$$

- 3) Intuitively, one would expect $[[\text{neg}(\text{neg}(\mathcal{P}))]] = [[\mathcal{P}]]$

$$[[\text{neg}(\mathcal{P})]] = \langle [[\mathcal{P}]]^- , [[\mathcal{P}]]^+ \rangle \quad (\text{flip})$$

Operator assert, first interpretation

- There is only one rather unhelpful example in the standard for the assert-operator.
 - „*The sequences of the operand are the only valid continuations. All other continuations result in invalid traces.*“
- This suggests the following:
$$[[\text{assert}(\mathcal{P})]] = \langle [[\mathcal{P}]]^+, \mathcal{E}\mathcal{O}^*/[[\mathcal{P}]]^+ \rangle$$
- This interpretation
 - ensures that $[[\text{assert}(\text{assert}(\mathcal{P}))]] = [[\text{assert}(\mathcal{P})]]$;
 - completely removes contingency.
- But how large is the pre-chart?

Operator assert, second interpretation

- The assert-Operator might be something like an implication, assuming that a (valid) trace can be understood as a (valid) interpretation of a formula.
- This would relate alt to logical disjunction and thus assert to implication, and we thus would expect

$$[[\text{assert}(\mathcal{P}, \mathcal{Q})]] = [[\text{alt}(\text{neg}(\mathcal{P}), \mathcal{Q})]].$$

- But we do not have $[[\text{assert}(\mathcal{P}, \mathcal{P})]] = [[\mathcal{P}]]$

Operator assert, third interpretation

- The assert-Operator might also be something like a „next“ operator in temporal logics.

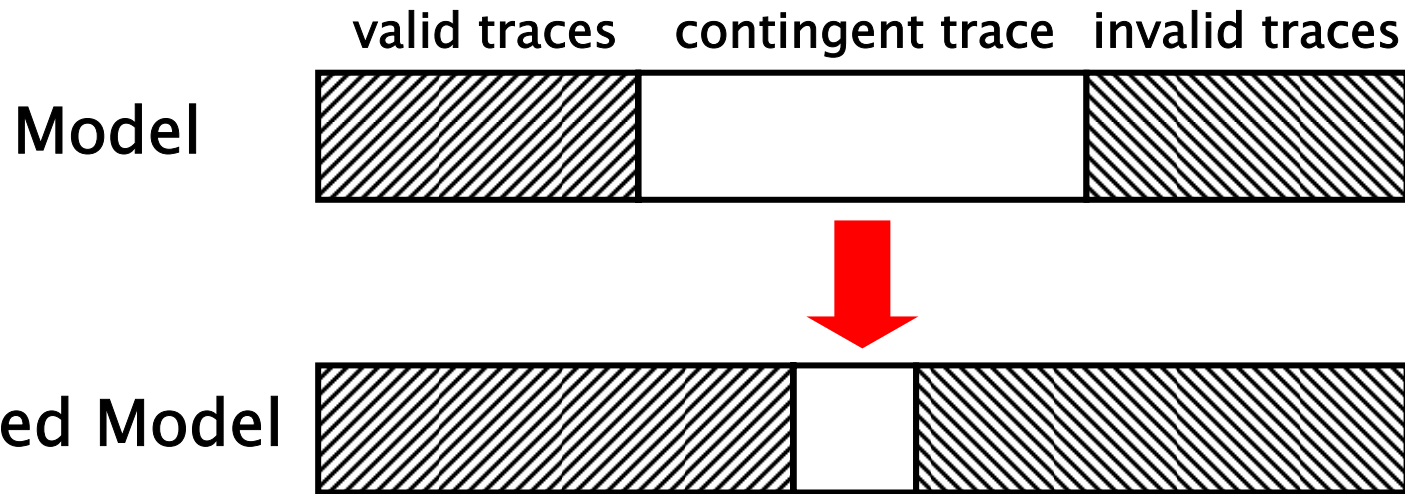
$$\text{a) } [[\text{assert}(\mathcal{P}, \mathcal{Q})]]_1 = \langle \emptyset, [[\mathcal{P}]]^+ \cdot \mathcal{EO}^* / [[\mathcal{Q}]]^+ \rangle$$

$$\text{b) } [[\text{assert}(\mathcal{P}, \mathcal{Q})]]_1 = \langle [[\mathcal{P}]]^+ \cdot [[\mathcal{Q}]]^+, [[\mathcal{P}]]^+ \cdot (\mathcal{EO}^* / [[\mathcal{Q}]]^+) \rangle$$

$$\text{c) } [[\text{assert}(\mathcal{P}, \mathcal{Q})]]_1 = \langle [[\mathcal{P}]]^+ \cdot [[\mathcal{Q}]]^+, [[\mathcal{P}]]^- \cup [[\mathcal{Q}]]^- \cup [[\mathcal{P}]]^+ \cdot \mathcal{EO}^* / [[\mathcal{Q}]]^+ \rangle$$

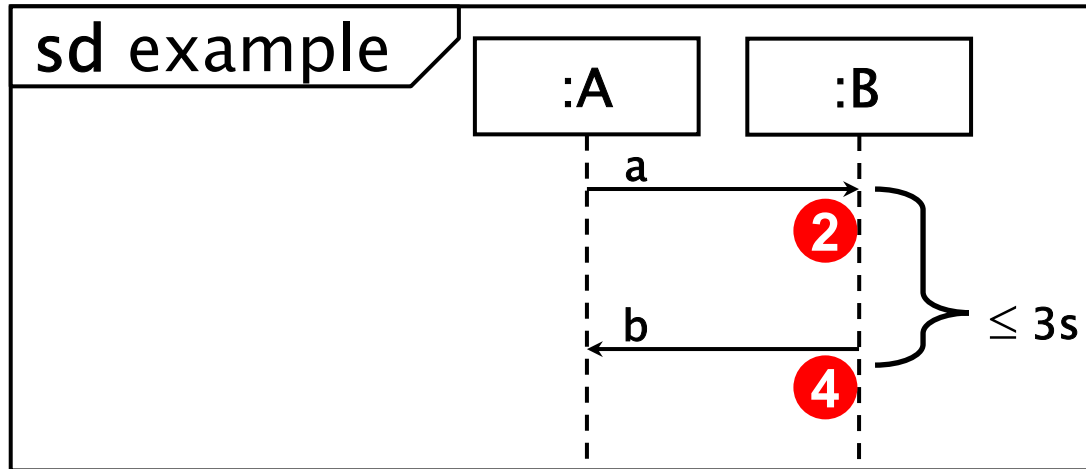
Model-Refinement: Does it make sense for Interactions?

- What does refinement mean for UML2-Interactions?



- Models are sets of Interactions, adding more Interactions reduces contingency.
- Using Assert does the same, but for just a single Interaction.
- **So, refinements has nothing to do with Interactions.**

Time constraints



time constraint

- two EventOccurrences
- a time expression

$\nu \in \mathcal{EO}^*$

a given trace

\mathcal{EO}_ν

the \mathcal{EO} s in ν

$\alpha, \beta \in \mathcal{EO}_\nu$

two \mathcal{EO} s from ν

\mathbb{R}

a domain of time points

$t: \mathcal{EO}_\nu \rightarrow \mathbb{R}$

a timing interpretation for ν

$\hat{\nu} \in \mathbb{R} \times \mathbb{R}$

a time interval of min/max durations

„ ν satisfies $\langle \alpha, \beta, \hat{\nu} \rangle$ under t_ν “ $\Leftrightarrow t(\beta) - t(\alpha) \in \hat{\nu}$

Summary

- **Status of current version („finally adopted spec“)** is:
 - much better metamodel–embedding than in 1.x;
 - much more consistency within new UML;
 - much clearer semantics for „conventional“ operators.
- **But still:**
 - a lot of open questions;
 - some left–overs from the „standardization wars“;
 - even some grave omissions.
- **Bottom line**
 - Practical value of negate, assert, ref etc. as yet unclear.
 - UML is becoming a very high level programming language.

Related/Further Work

- Companion paper on the part of the semantics not covered here.
- Semantics
 - Partial-order semantics as benchmark
 - Semantics for refinements, neg and assert.
 - Experiments with XMI-Prolog-Tool
- Syntax/Pragmatics
 - Explore the power of Timing Diagrams
 - Powerful editor for all kinds of interactions (plugin to ArgoUML)
 - Connection to RT-Modelchecker Uppaal

Ignore/consider

- Preliminaries

- Msg a domain of messages
- $\Gamma \subseteq Msg$ Messages to be ignored
- $\mu: EO \rightarrow Msg$ mapping EventOccurrences to Messages

- $[[ignore(P, \Gamma, \mu)]] = \langle EO^* \sqcup [[P]]^+, EO^* \sqcup [[Q]]^+ \rangle$