Refinement III

The general case

Ketil Stølen

Topics

- Sequential composition
- Negative behaviour
- Refinement in the general case

Sequential composition

Basic rules

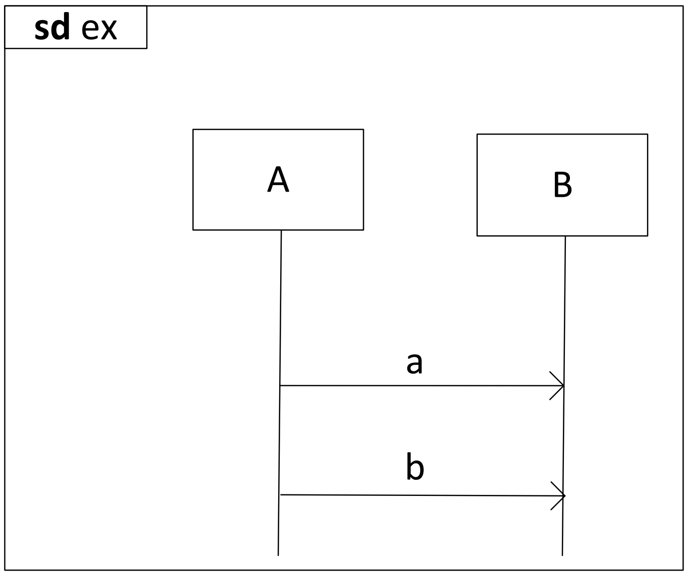
Causality

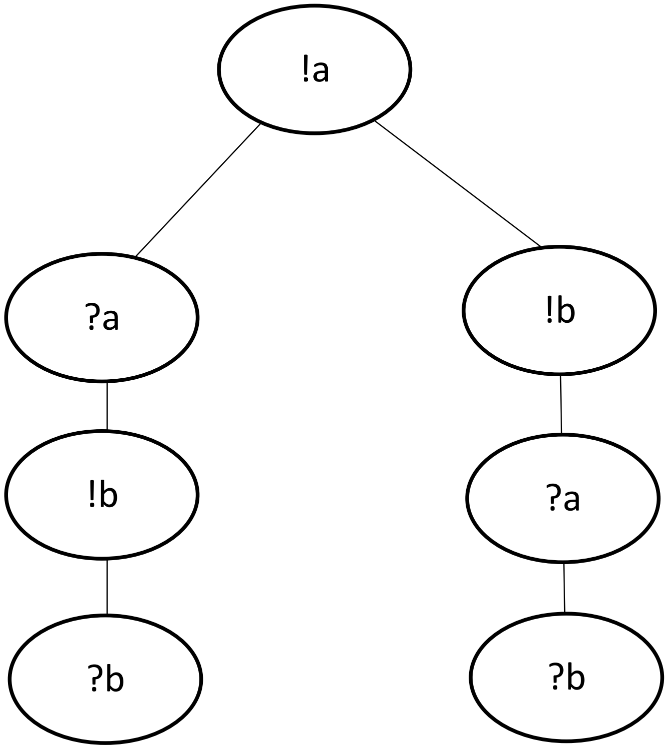
- a message can never be received before it has been transmitted
- the transmission event for a message is therefore always ordered before the reception event for the same message

Weak sequencing

- events from the same lifeline are ordered in the trace in the same order as on the lifeline (from top to bottom)

Example





Mathematically !a and ?a (etc.) are shorthands for !(a,A,B) and ?(a,A,B)
Hence, each event contains the names of its sending and receiving lifelines

Sequential composition of trace sets s_1 and s_2

$$s_1 \gtrsim s_2$$

=

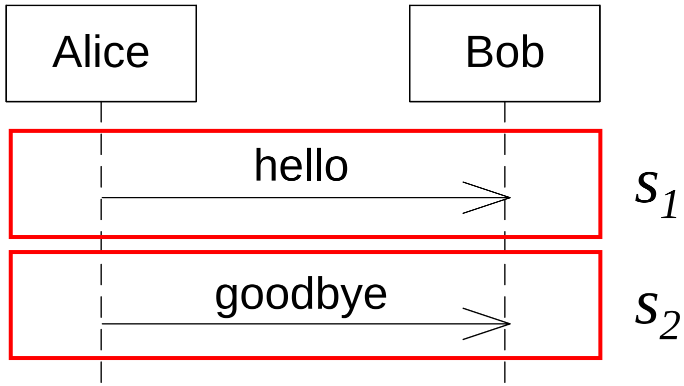
the set of all traces obtained by merging traces

t_1 from s_1 and t_2 from s_2

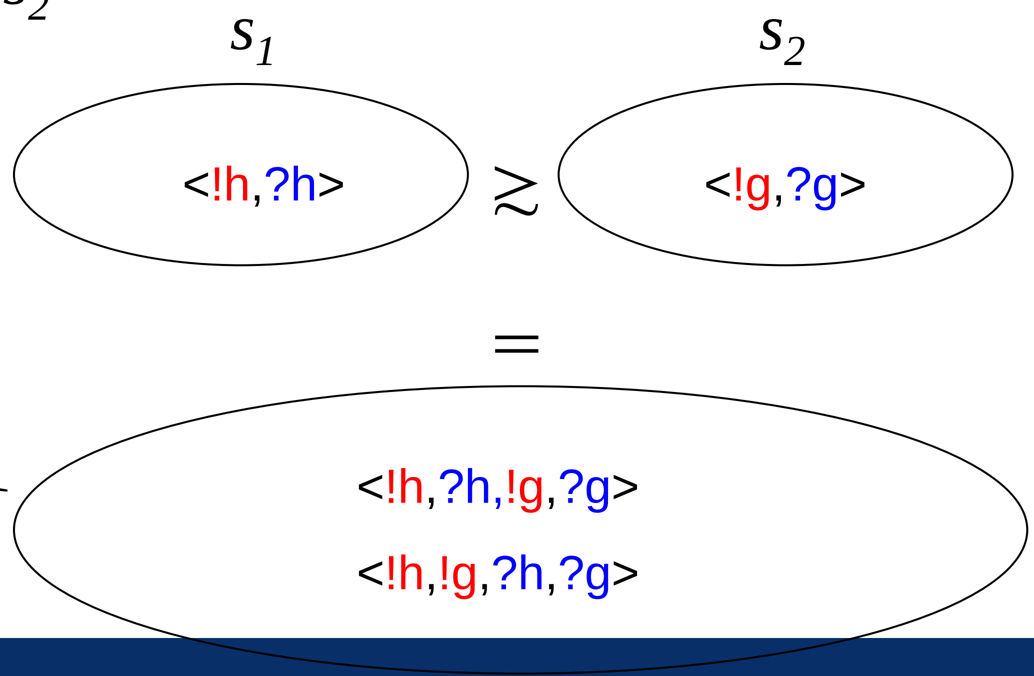
in such a way that for each lifeline,

the events from t_1 comes before the events from t_2

Sequential composition of trace sets



Red events occur on Alice, blue events on Bob



$s_1 \gtrsim s_2$ is the set of positive traces for the diagram

Note

- if s_1 or s_2 is empty then $s_1 \gtrsim s_2$ is also empty

Sequential composition of interaction obligations

- $(p_1,n_1) \succsim (p_2,n_2) \stackrel{\text{def}}{=} (p_1 \succsim p_2, (n_1 \succsim p_2) \cup (n_1 \succsim n_2) \cup (p_1 \succsim n_2))$

- Traces composed exclusively by positive traces become positive
- Traces composed with at least one negative trace become negative

Formal semantics of **seq**

- $[[d_1 \text{ seq } d_2]] \stackrel{\text{def}}{=} \{o_1 \gtrsim o_2 \mid o_1 \in [[d_1]] \wedge o_2 \in [[d_2]]\}$
- o_i is shorthand for (p_i, n_i)

Remember: By sequential composition

- positive followed by positive is positive
- positive followed by negative is negative
- negative followed by negative is negative
- negative followed by positive is negative

Negative behaviour

Specifying negative behaviour with **refuse**

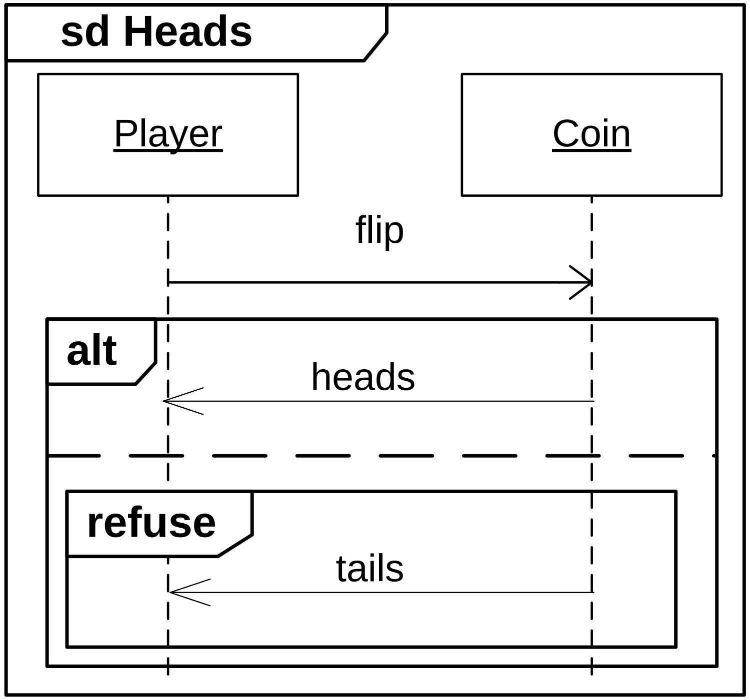
- $[[\text{refuse } d]] \overset{\text{def}}{=} \{(\{\,\}, p \cup n) \mid (p,n) \in [[d]]\}$

- All interaction obligations in $[[\text{refuse } d]]$ have empty positive sets

- Hence, all interaction obligations in $[[d_1 \text{ seq } (\text{refuse } d_2)]]$ have empty positive sets

- The same applies to $[[(\text{refuse } d_1) \text{ seq } d_2]]$

Example use of **refuse**



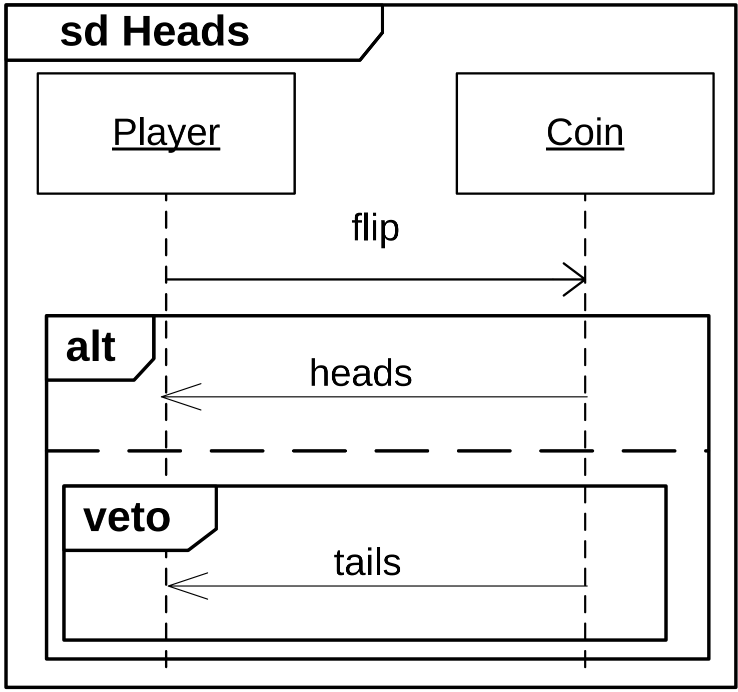
- [[Heads]] = {({<!f, ?f, !h, ?h>}, {<!f, ?f, !t, ?t>})}

Specifying negative behaviour with **veto**

[[veto *d*]] $\stackrel{\text{def}}{=}$ [[skip alt (refuse *d*)]]

This means:

[[veto *d*]] = $\{(\{<>\}, p \cup n) \mid (p,n) \in [[d]]\}$



[[Heads]] = {({<!f, ?f, !h, ?h>, <!f, ?f>} , {<!f, ?f, !t, ?t>})}

Specifying negative behaviour with **assert**

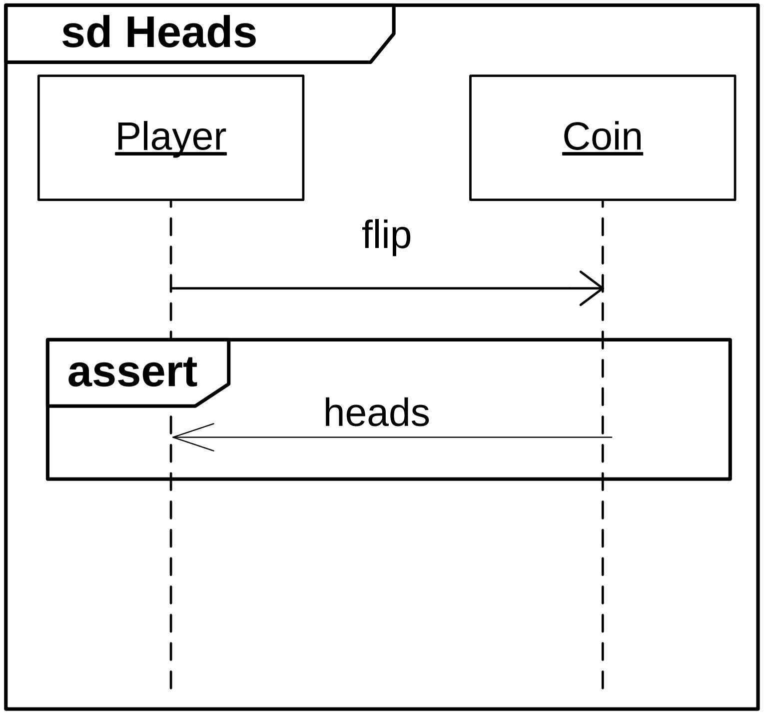
- By using **assert**, all inconclusive traces are redefined as negative
- This ensures that for each interaction obligation, at least one of its positive traces will be implemented in the final implementation

- $[[\text{assert } d]] \stackrel{\text{def}}{=} \{(p, n \cup (\mathcal{H} \setminus p)) \mid (p, n) \in [[d]]\}$

- $\mathcal{H}$ = all possible traces
- $\mathcal{H} \setminus p$ = all possible traces minus those in *p*

Example use of **assert**

- [[Heads]] = {({<!f, ?f, !h, ?h>}, n)}



- n = all traces where the first event on the lifeline of Player is !f and the first event on the lifeline of Coin is ?f except the trace <!f, ?f, !h, ?h>

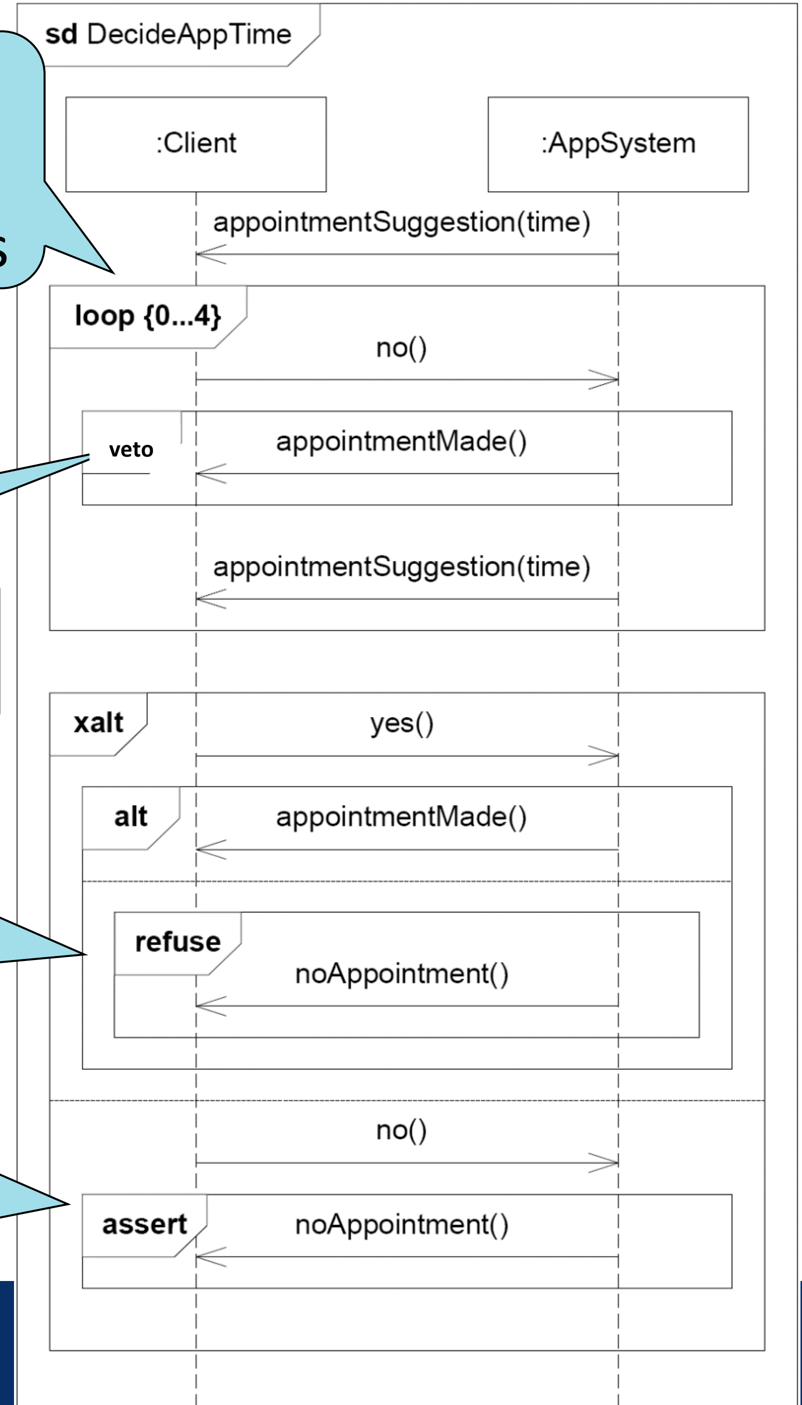
Negative behaviour

From
0 to 4
iterations

appointmentMade() may not occur here

noAppointment() may not occur instead of appointmentMade() here

noAppointment () is the only message that may occur here



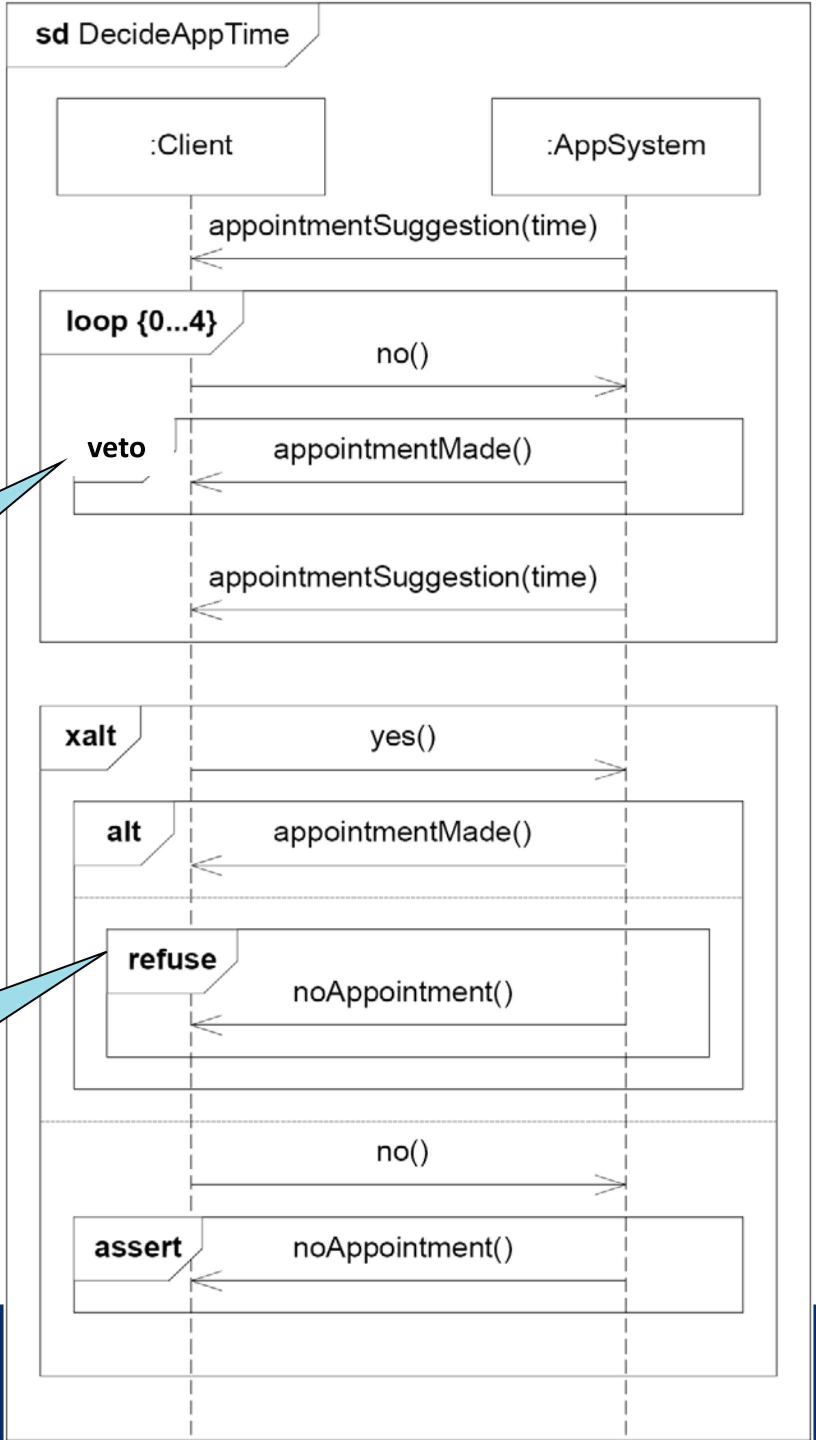
veto or refuse?

Should doing nothing be possible in the otherwise negative situation?

- If yes, use veto
- If no, use refuse

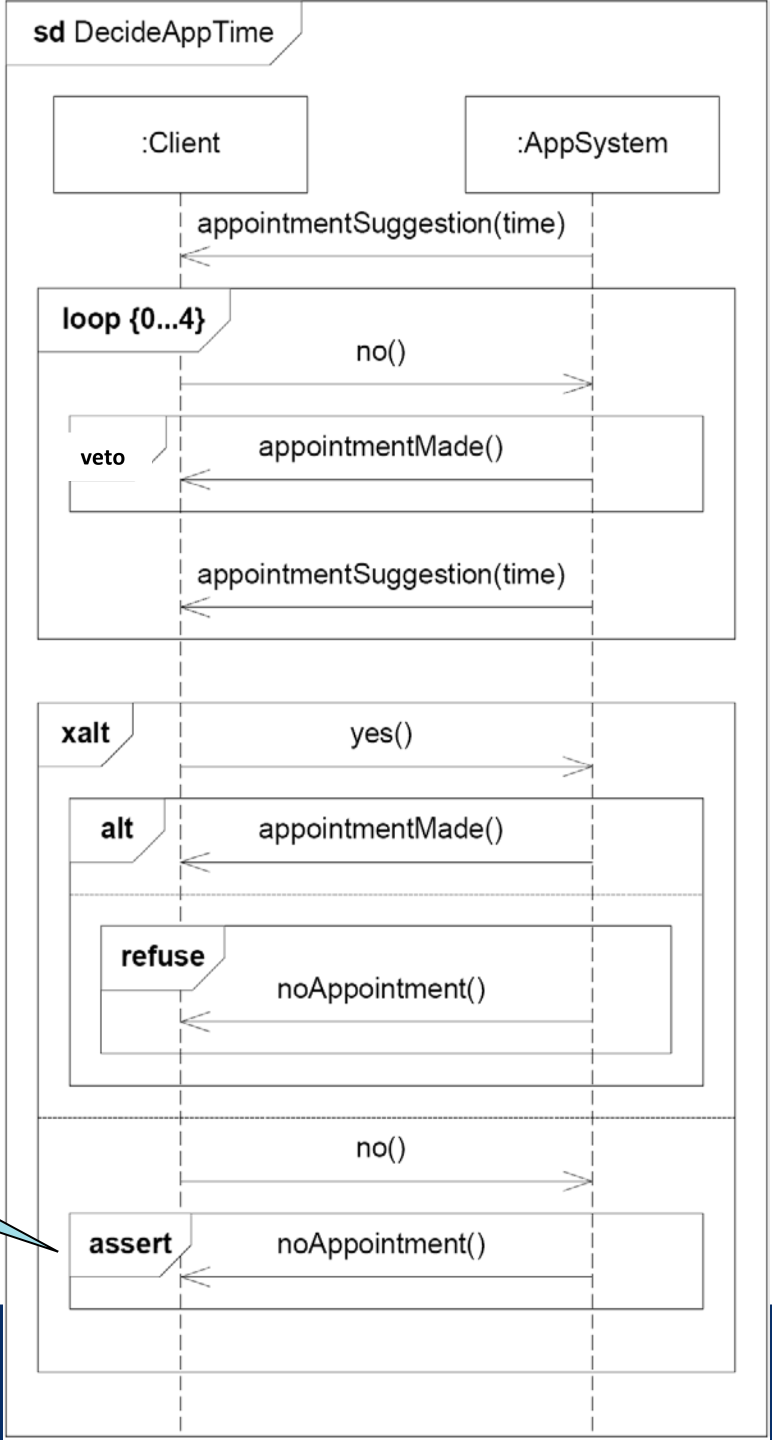
ok to do nothing between no() and appointment-Suggestion(time)

not ok to do nothing after yes()



When to use assert?

Sending noAppointment() is the only acceptable response to the no() message at this point



The pragmatics of negative behaviour

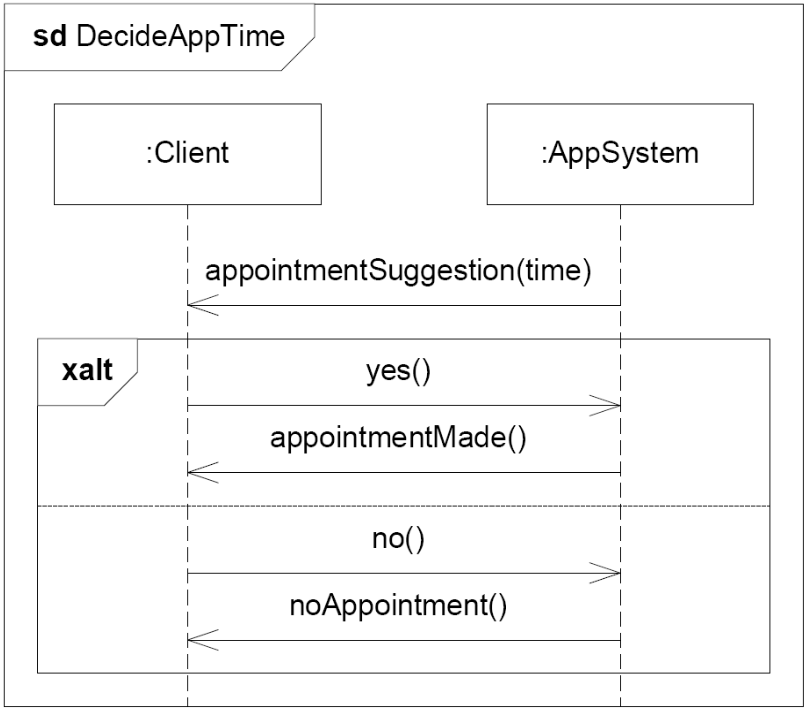
- To effectively constrain the implementation, the specification should include a reasonable set of negative traces
- Use **refuse** when specifying that one of the alternatives in an **alt** represents negative traces
- Use **veto** when the empty trace (i.e. doing nothing) should be positive, as when specifying a negative message in an otherwise positive scenario
- Use **assert** on an interaction fragment when all positive traces for that fragment have been described

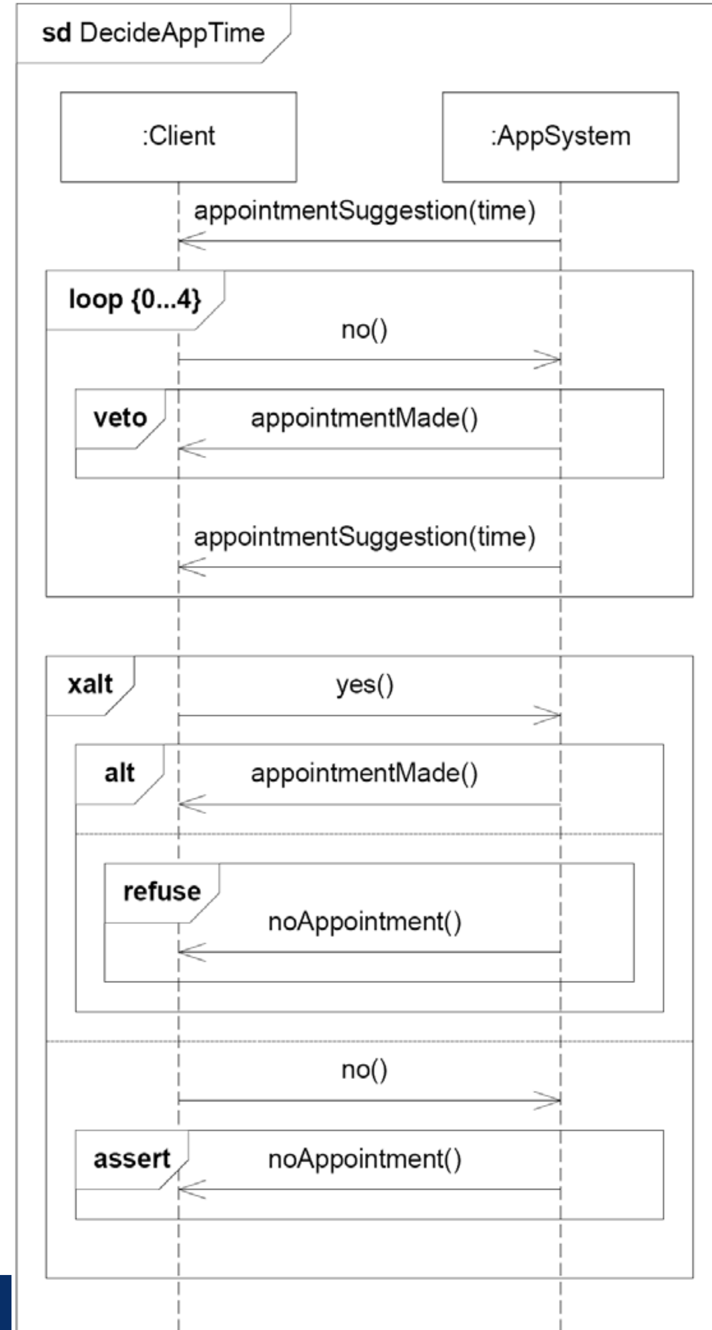
Refinement in the general case

Supplementing

- Inconclusive trace are recategorized as either positive or negative (for an interaction obligation)
- New situations are considered
  - adding fault tolerance
  - new user requirements
  - ...
- Typically used in early phases

Example of supplementing





Positive

Negative

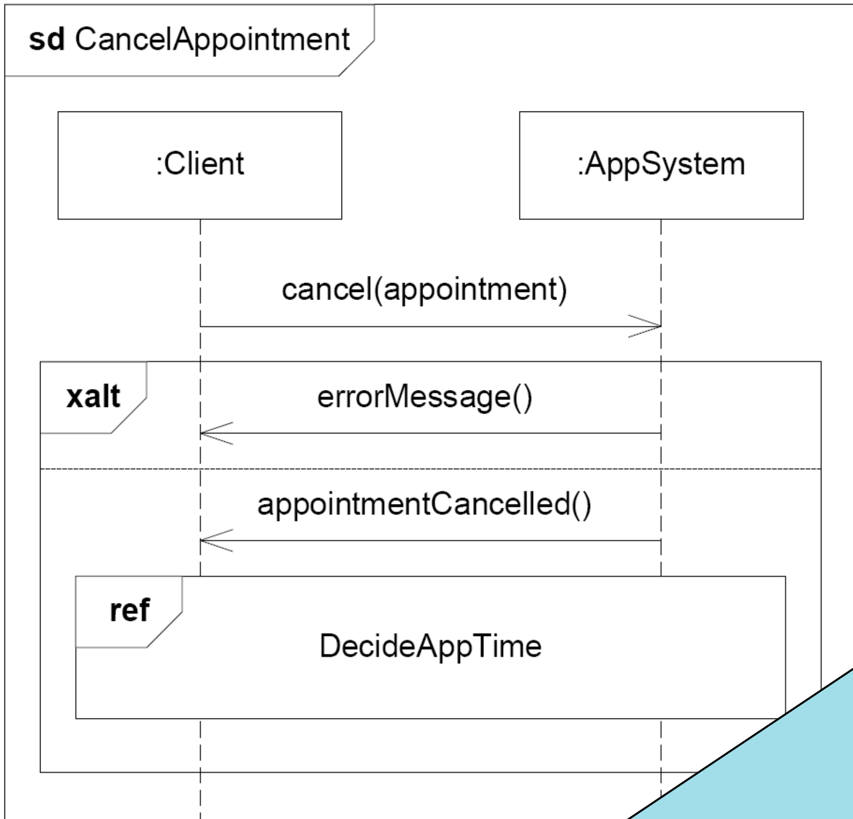
The pragmatics of supplementing

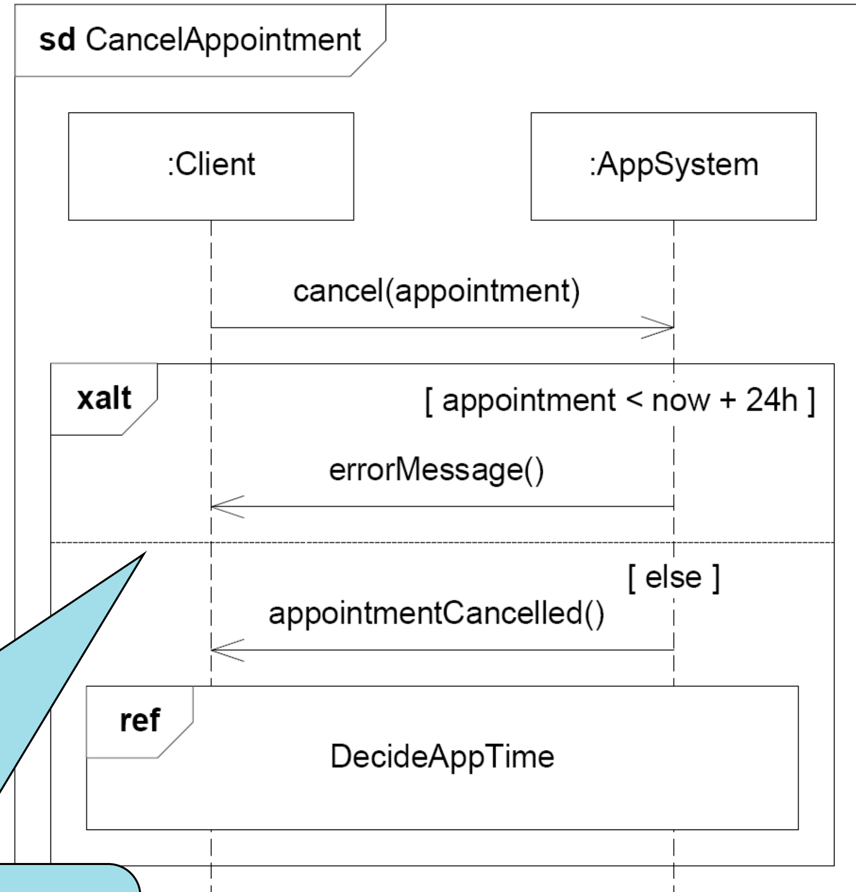
- Use supplementing to add positive or negative traces to the specification
- When supplementing, all of the original positive traces must remain positive, and all of the original negative traces must remain negative
- Do not use supplementing on the operand of an assert
  - no traces are inconclusive in the operand

Narrowing

- Reduce underspecification by redefining positive traces as negative
- For example adding guards, or replacing a guard with a stronger one
  - traces where the guard is false become negative

Example of narrowing





For each operand, traces where the guard is false become negative

The pragmatics of narrowing

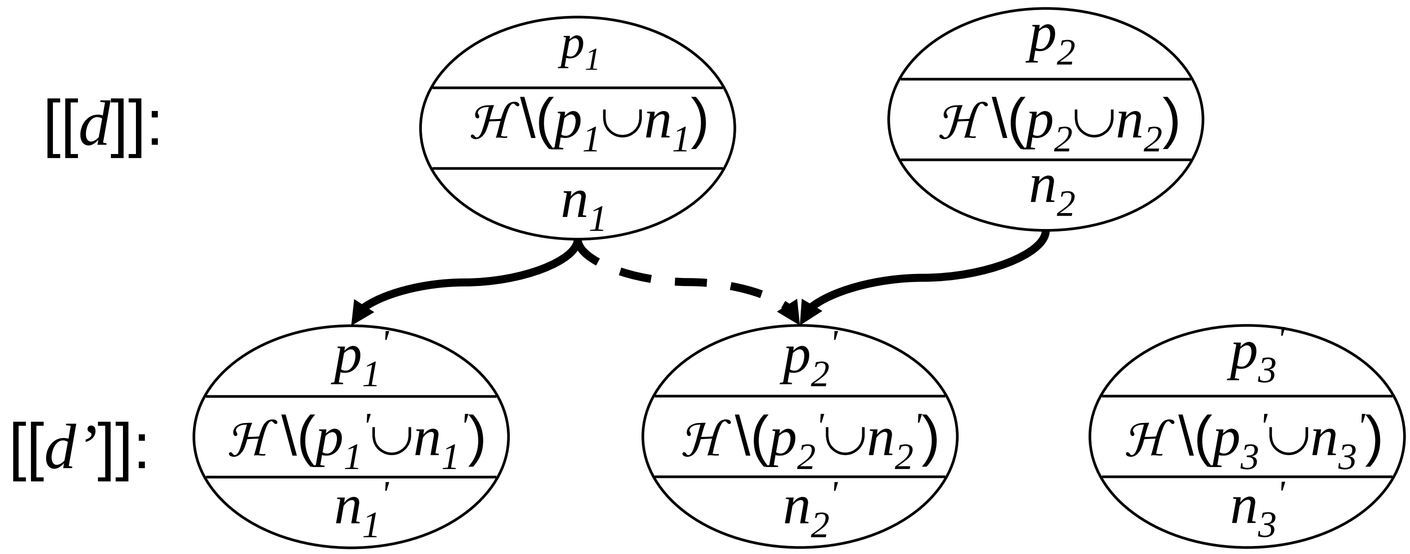
- Use narrowing to remove underspecification by redefining positive traces as negative
- In cases of narrowing, all of the original negative traces must remain negative
- Guards may be added to an **alt** as a legal narrowing step
- Guards may be added to an **xalt** as a legal narrowing step
- Guards may be narrowed, i.e. the refined condition must imply the original one

General refinement

- d' is a general refinement of d if
  - for every interaction obligation o in $[[d]]$ there is at least one interaction obligation o' in $[[d']]$ such that o' is a refinement of o

- Interaction obligations that do not refine any obligation at the abstract level may be added

General refinement illustrated



The pragmatics of general refinement

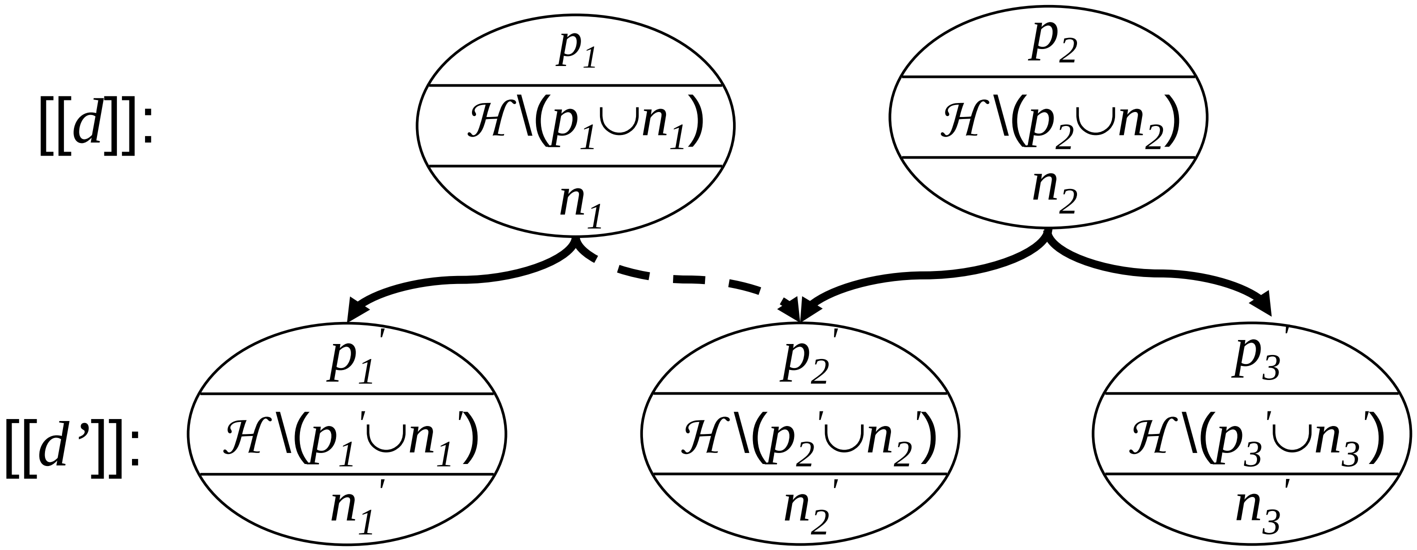
- General refinement is required for specifications with **xalt**
- It corresponds to the pointwise application of refinement for each single interaction obligation
- General refinement supports the introduction of additional inherent nondeterminism

Limited refinement

- d' is a limited refinement of d if
  - d' is a general refinement of d, and
  - every interaction obligation in $[[d']]$ is a refinement of at least one interaction obligation in $[[d]]$

- Limits the possibility of adding new interaction obligations
- Typically used at a later stage

Limited refinement illustrated



SINTEF

Technology for a better society

The pragmatics of limited refinement

- Limited refinement is a special case of general refinement
- Limited refinement disallows the introduction of additional inherent nondeterminism
- Limited refinement is normally used in the later stages of a system development

Compositionality

A refinement operator $\rightsquigarrow$ is compositional if it is

reflexive: $d \rightsquigarrow d$

transitive: $d \rightsquigarrow d' \wedge d' \rightsquigarrow d'' \Rightarrow d \rightsquigarrow d''$

the operators refuse, veto, alt, xalt and seq are monotonic w.r.t. $\rightsquigarrow$:

$d \rightsquigarrow d' \Rightarrow$ refuse $d \rightsquigarrow$ refuse d'

$d \rightsquigarrow d' \Rightarrow$ veto $d \rightsquigarrow$ veto d'

$d_1 \rightsquigarrow d_1' \wedge d_2 \rightsquigarrow d_2' \Rightarrow d_1$ alt $d_2 \rightsquigarrow d_1'$ alt d_2'

$d_1 \rightsquigarrow d_1' \wedge d_2 \rightsquigarrow d_2' \Rightarrow d_1$ xalt $d_2 \rightsquigarrow d_1'$ xalt d_2'

$d_1 \rightsquigarrow d_1' \wedge d_2 \rightsquigarrow d_2' \Rightarrow d_1$ seq $d_2 \rightsquigarrow d_1'$ seq d_2'

- Transitivity allows stepwise development
- Monotonicity allow different parts of the specification to be refined separately

Supplementing, narrowing, general refinement and limited refinement are all compositional ☺

The mathematical foundation

- Haugen, Husa, Runde, Stølen: *STAIRS towards formal design with sequence diagrams*, 2005. SoSyM, Springer.
  - http://heim.ifi.uio.no/~ketils/kst/Articles/2005.SoSyM-onlinefirst.pdf

- Runde, Haugen, Stølen: *The Pragmatics of STAIRS*, 2006. Springer-Verlag. LNCS 4111.
  - http://heim.ifi.uio.no/~ketils/kst/Articles/2006.FMCO-LNCS4111.pdf