Refinement I

Ketil Stølen
Objectives for the lectures on refinement

- Motivate the role of refinement
- Introduce and related the following notions of refinement
  - supplementing
  - narrowing
  - detailing
- Illustrate the use of these notions of refinement
  - the interplay between specification and refinement
- Illustrate the translation of theory into practice
Why refinement is important

- Software systems are large and complex – abstraction is a necessary means to
  - explain what the systems do
  - explain how the systems are built
  - distinguish the essentials from the inessential
  - decompose large and complicated aspects into small more easily understandable entities
  - extract specialized system views

- Refinement
  - relates system descriptions at different levels of abstraction
  - connects and relates different system views
  - provides a foundation for verifications and validations
Documenting refinement

- Precision is just as important when we document refinements as when we write specifications.

- Refinements can be documented using standard specification languages.
  - in INF 5150 we will use UML for this purpose.

- Formal documentation of refinements facilitates:
  - analysis,
  - validation,
  - testing,
  - verification.
Three main concepts of language theory

● Syntax
  – The relationship between symbols or groups of symbols independent of content, usage and interpretation

● Semantics
  – The rules and conventions that are necessary to interpret and understand the content of language constructs

● Pragmatics
  – The study of the relationship between symbols or groups of symbols and their interpretation and usage
Syntactically correct expressions in the language to be explained.

What does it mean that a language is well-understood?

Semantic relation

Relates expressions that need interpretation to expressions that are well-understood.

Syntactically correct expressions in a language that is well-understood.
The need for a notion of observation

- A semantic relation will define an equivalence relation on the language that should be understood.

Of the same meaning

For a specification language these are defined with respect to a notion of observation.
Definition of a notion of observation

- May observe only external behavior
- May observe any potential behavior
- May observe time with respect to a global clock
- May observe safety properties
  - Always falsified by a partial execution
- May observe liveness
  - Falsified only by complete executions

May our notion of observation be implemented by a human being?
Assumption-guarantee paradigm

● Motivation:
  – The behavior of a system component depends on the context it is executed in
  – Not all contexts are equally interesting

● The assumption describes expected input
  – The input that can be produced by the relevant contexts

● The guarantee describes the output the specified component is obligated to produce as long as the context behaves in accordance with the assumption
NOTE: The A-G paradigm does not specify the context – only the context’s interaction with the system in question.
Pre-post specifications

Pre-post specifications are based on the assumption-guarantee paradigm

---

**Integer division**

- **var** dividend, divisor, quotient, rest : Nat

---

**pre** divisor ≠ 0  

---

**post** (dividend = (quotient' * divisor) + rest') &  
rest' < divisor

---

Assumption about the state at the moment the execution is initiated

Guarantee with respect to the state at the moment of termination
Semantics of pre-post specifications

- Pre false initially: no constraints on state at termination
- Pre true initially: post true at termination
- Legal, but arbitrary behavior
- Illegal system behavior

Legal system behavior

Illegal system behavior
Refinement in pre-post

- Weakening pre
- Strengthening post

<table>
<thead>
<tr>
<th>Pre false initially</th>
<th>Pre true initially</th>
</tr>
</thead>
<tbody>
<tr>
<td>post false at termination</td>
<td></td>
</tr>
<tr>
<td>post true at termination</td>
<td></td>
</tr>
</tbody>
</table>

- No constraint on state at termination
Weakening the pre-condition (assumption)

--- Integer division ---

```plaintext
var dividend, divisor, quotient, rest : Nat

pre   true

post

if divisor \neq 0 then
    ( dividend = (quotient’ * divisor) + rest’) & rest’ < divisor
else quotient’ = 0
```
Strengthening the post-condition (guarantee)

---

### Integer division

```plaintext
var dividend, divisor, quotient, rest : Nat

pre divisor \neq 0

post ( dividend = (quotient' * divisor) + rest' ) &
    rest' < divisor & dividend' = dividend &
    divisor' = divisor
```
The shortcomings of pre-post specifications

- Pre-condition describes only what the context may do before the operation is started up – not what the context may do during the execution of the operation

```plaintext
pre { divisor \neq 0 }
<quotient := 0>
while <dividend > divisor> do
  <dividend := dividend - divisor>
  <quotient := quotient + 1>
  od
<rest:=dividend>
post { ( dividend’ = (quotient * divisor’) + rest ) & rest < divisor’ }
```

- “<Statement>” denotes that “statement” is atomic (in the meaning that the context cannot interfere with its execution)
Traces

- Traces are used to represent system runs mathematically
- In the literature there are many different kinds of traces
- INF 5150 traces are sequences of events

\(<e_1, e_2, e_3, e_4, e_1, e_2, e_5, \ldots>\)

- Events are instantaneous
Trace lengths

- The number of events in a trace may be finite
  - may be caused by: termination, deadlock, infinite waiting, system crash

- The number of events in a trace may be infinite
  - May be cause by: nontermination, livelock, nontermination by purpose
    \(<e_1, e_2, e_3, e_4, e_4, e_1, e_2, e_5, \ldots\>\)
Traces with time ticks

- Traces are infinite sequences of events and time ticks

\[ \langle e_1, e_2, e_3, \text{tick, tick, } e_4, e_4, e_1, \text{tick, tick, tick, } e_2, \text{tick, } e_5, \text{tick, \ldots\ldots\ldots} \rangle \]

- Events and time ticks are instantaneous
- Each trace contains infinitely time ticks
  - this reflects that time never halts
- The number of events in a trace may be finite
Traces with time stamps

- Each element of the trace is a pair of an event and a time stamp

\[ <e_1:t_1, e_2:t_2, e_3:t_3, e_4:t_4, e_4:t_5, e_1:t_6, e_2:t_7, e_5:t_8, \ldots \ldots \ldots > \]

- The elements are ordered according to their time stamps
  - \((t_1 \leq t_2 \leq t_3 \ldots)\)
- Events are instantaneous
Zenon's paradox

- That which is in locomotion must arrive at the half-way stage before it arrives at the goal
  - as recounted by Aristotle, *Physics* VI:9
A trace is either

- finite, or

- there is for every point in time $k$ an element $n:t$ with time stamp $t$ such that $t > k$
– STAIRS –

Steps to Analyze Sequence Diagrams with Refinement Semantics
Motivation

- Exploit classical theory of refinement in a practical UML setting
  - From theory to practice, and not the other way around

- Sequence diagrams can be used to capture the meaning of other UML description techniques for behavior

- By defining refinement for sequence diagrams we therefore implicitly define refinement for UML
Traces for sequence diagrams summarized

- Traces for sequence diagrams are sequences of events
  \(<e_1, e_2, e_3, e_4, e_4, e_1, e_2, e_5, \ldots>\)

- An event represents either the transmission or reception of messages
  - ?m - reception of message m
  - !m - transmission of message m

- Events are instantaneous

- A trace may be finite
  - termination, deadlock, infinite waiting, crash

- A trace may also be infinite
  - infinite loop, intended non-termination
Causality and weak sequencing

- **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
Weak sequencing

.sd W

L1

x

y

L2

<!x,?x,!y,?y>
<!x,?x,?x,?y>
These two diagrams are semantically the same
Alternative composition
Parallel composition

\[
\begin{array}{c}
\text{sd P} \\
\text{L1} \quad \text{L2}
\end{array}
\]

\[
\text{par} \quad x \quad y
\]
Interaction overview diagram

S seq (IO par W) seq (IO alt W)
Dinner

- a Salad as a starter
- then a main course consisting of an Entree and SideOrder in parallel

**Entree choices**
- Vegetarian
- Beef
- Pork

**SideOrder choices**
- Baked Potato
- Rice
- Frites

A Salad as a starter, then a main course consisting of an Entree and SideOrder in parallel
Some potential positive traces of Beef

1. Cook
   - main dish please
   - turn on heat
   - heat is adequate
   - put on grill (sirloin)
   - fetch_meat():sirloin

2. Stove
   - fetch_meat():sirloin

3. Refrigerator
   - main dish:sirloin
   - fetch_meat():sirloin
Potential negative Beef experiences

- Negative traces:
  - Smell of burned meat
  - Veto

- Positive traces:
  - Beef with French fries
  - Turkey entree

- Inconclusive traces:
  - Forgotten Sirloin

- Burned Sirloin

- Process:
  1. Fetch meat (sirloin)
  2. Turn on heat
  3. Heat is adequate
  4. Put on grill (sirloin)
  5. Main dish: sirloin
STAIRS semantics: simple case

- Each positive execution is represented by a trace
- Each negative execution is represented by a trace
- The semantics of a sequence diagram is a pair of sets of traces (Positive, Negative)

All other traces over the actual alphabet of events are inconclusive
Semantics of pre-post specifications

- **pre false initially**
  - no constraints on state at termination
  - Legal, but arbitrary behavior

- **pre true initially**
  - post true at termination
  - Illegal system behavior
  - post false at termination
  - Legal system behavior
Comparing STAIRS with pre-post

<table>
<thead>
<tr>
<th>assumption</th>
<th>post=true positive</th>
</tr>
</thead>
<tbody>
<tr>
<td>post=false negative</td>
<td></td>
</tr>
<tr>
<td>inconclusive</td>
<td></td>
</tr>
</tbody>
</table>

pre=false

pre=true
Refinement in pre-post

- Weakening pre
- Strengthening post
STAIRS: supplementing

- Supplementing involves reducing the set of inconclusive traces by redefining inconclusive traces as either positive or negative.
- Positive trace remains positive.
- Negative trace remains negative.
Supplementing in pre-post

- pre=false
- post=true positive
- post=false negative
- inconclusive
- weakening the assumption

assumption
guarantee
STAIRS: narrowing

- Narrowing involves reducing the set of positive traces by redefining them as negative
- Inconclusive traces remain inconclusive
- Negative traces remain negative

![Diagram showing narrowing process between sets of traces and restaurant options]
Narrowing in pre-post

- **pre=false**
  - post=true: positive
  - post=false: negative
  - inconclusive

- **pre=true**
  - assumption
  - strengthening the guarantee
  - guarantee
Indirect definition: Refinement in STAIRS

A sequence diagram B is a general refinement of a sequence diagram A if

- A and B are semantically identical
- B can be obtained from A by supplementing
- B can be obtained from A by narrowing
- B can be obtained from A by a finite number of steps
  \[ A \rightarrow C_1 \rightarrow C_2 \rightarrow \ldots \rightarrow C_n \rightarrow B \]
  each of which is either a supplementing or a narrowing
Is B a refinement of A?
Is B a refinement of A?
Is B a refinement of A?

sd A

S

e

b

c

T

sd B

S

alt

e

b

c

dl

k

f

T
Is B a refinement A?
Is B a refinement of A?