Refinements for LTS-based formalisms

Tomáš Poch, Ondřej Šerý
DISTRIBUTED SYSTEMS RESEARCH GROUP
http://dsrg.mff.cuni.cz

CHARLES UNIVERSITY PRAGUE
Faculty of Mathematics and Physics
Outline

• LTS based formalisms
• Refinement
  ▪ CSP,
  ▪ Interface automata,
  ▪ CCS
• Refinement for TBP
  ▪ Bad activity
  ▪ No activity
  ▪ Unlimited number of threads

• Conclusion
Formalisms with LTS behind

- Semantics given in form of LTS
  - Interface automata
  - Process algebras
  - Behavior Protocols

- Errors/Properties expressible are given by the formalism
  - Depends on what is on the edges
  - In this talk we consider finite LTSs labeled by method calls

- Correctness of architecture is a relative concept
  → makes sense in a **closed** system

- Error state in an open system can be avoided in particular environment
Refinement – SE

• Compare two open systems
  ▪ Role of specification vs. implementation /frame vs. architecture
  ▪ Do they behave similarly w.r.t. all environments?
    • e.g. preserve errors, different degree of non-determinism

• Goal: verify specification, use implementation
  ▪ Hierarchical component models
  ▪ Hi-level specification vs. real code
Refinement example

Controller → Player

DataServer

FileServer → Logger

SPlayer
Refinement example

(1) ErrFree

(2)
Refinement – SE

- Compare two open systems
  - Role of specification vs. implementation /frame vs. architecture
  - Do they behave similarly w.r.t. all environments?
    - e.g. preserve errors, different degree of non-determinism

- Goal: verify specification, use implementation
  - Hierarchical component models
  - Hi-level specification vs. real code

\[
A < B \Rightarrow \forall E : ErrFree(E \oplus B) \Rightarrow ErrFree(E \oplus A)
\]

- 3 steps
  - Check that implementation refines the specification
  - Check that the specification fits into the system
  - Use the implementation
Refinement

- Trivial refinement (i) – equality
  - Not much of use
  - Specification must be somewhat simpler, hide details

- Trivial refinement (ii) – everything
  - Not much of use
  - Do not provide any guarantee for the implementation

- Useful refinement relations
  - Reduce non-determinism (CSP)
  - Preserve errors w.r.t. all environments
    - aims at specific errors supported by the formalism
## Errors in formalisms

<table>
<thead>
<tr>
<th>Models</th>
<th>Bad Activity/IA Error</th>
<th>No activity</th>
<th>Stuck</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>!a !b</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>?a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>!b</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>?a</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>?b</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>?a</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>?a</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>?a</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>?a</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Bad Activity**
  - A method call is not accepted by counterpart

- **No Activity**
  - Waiting for a not coming method call in a non-final state

- **Stuck (CCS)**
  - An event in a stable state is not met by counterpart
  - Does not distinguish ! and ?
  - Does not consider final states
  - Resembles mixed BA and NA
CSP refinements (Roscoe)

- Process algebra
  - Does not distinguish ! and ?
  - Provides external choice and internal choice
    - \((a \rightarrow \text{STOP}) \square (b \rightarrow \text{STOP})\) is obliged to communicate a or b
    - \((a \rightarrow \text{STOP}) \sqcap (b \rightarrow \text{STOP})\) may reject either

- Refinement is defined as
  - \(R = R \sqcap P\) (\(R \subseteq P\))
    - \(P\) is more deterministic than \(R\) (\(P\) refines \(R\))
Refinement in Interface automata

- Considers Bad Activity
- Based on alternation simulation
  - Define $\varepsilon$-closure of a state
    - Set of states reachable by internal events
  - Define sets of observable events
    - $\text{obs!}(x) = \{!q: \exists s \in \varepsilon\text{-closure}(x) \text{ calling } !q\}$
    - $\text{obs?}(x) = \{?p: \forall s \in \varepsilon\text{-closure}(x) \text{ expects } ?p\}$
Refinement in Interface automata

- Preserves Bad Activity
- Based on alternation simulation
  - Define $\epsilon$-closure of a state
    - Set of states reachable by internal events
  - Define sets of observable events
    - $\text{obs!}(x) = \{!q: \text{Exists } s \in \epsilon\text{-closure}(x) \text{ calling } !q\} = \{!q, !s\}$
    - $\text{obs?}(x) = \{?p: \text{All } s \in \epsilon\text{-closure}(x) \text{ expects } ?p\} = \{?p\}$

- Having implementation and specification
  - implementation state $x'$ refines a specification state $x$ if
    - $\text{obs?}(x) \subseteq \text{obs?}(x')$ and $\text{obs!}(x) \supseteq \text{obs!}(x')$
    - Property propagated to successors
  - Must hold for initial states
obs?(x) ⊆ obs?(x') and obs!(x) ⊇ obs!(x')

ErrFree(e ⊕ x) ⇒ ErrFree(e ⊕ x')
CCS Refinement w.r.t. Stuck-freedom

- The evil to avoid is a stuck CCS process
  - An event in a stable state is not met by counterpart
  - Does not distinguish $!$ and $?$
  - Does not consider final states
  - Resembles mixed BA and NA
CCS Refinement w.r.t. Stuck-freedom

• Refinement relation (stuck-free conformance) is again property of pairs of states

• Ready refusal
  • Process P can refuse
    • \{?p,!p,?q,!q,?r,!s\} while ready on \{?r\}
    • \{?p,!p,?q,!q,?r,!s\} while ready on \{!s\}
    • \{!p,?q,!q,?r,!r,?s,!s\} while ready on \{?p\}

• Stuck-free conformance (P<Q)
  • P can refuse X while ready on Y => Q can refuse X while ready on Y
  • Property propagated to successors
CCS Refinement w.r.t. Stuck-freedom

Ready refusals(Q) = {
    <{?p,!p,?q,!]q,[!r,?s}, {?r}>,
    <{?p,!p,?q,!]q,[!r,?s}, {!s}>,
    <{!p,?q,!]q,[!r,?r,?s,!s}, {?p}>
}

UI

Ready refusals(P) = {
    <{?p,!p,?q,!]q,[!r,?s}, {?r}>,
    <{?p,!p,?q,!]q,[!r,?s}, {!s}>,
    <{?p,!p,?q,!]q,[!r,?s}, {!s}>
}
Generalized framework (Specific for BP)

- An error to be preserved is a parameter

- 3 steps (Impl<Spec)
  1. Observation Projection (Impl\textsubscript{OP}, Spec\textsubscript{OP})
     - Hide the internal communication, non-determinism,
  2. Relation R(s\textsubscript{Impl},s\textsubscript{Spec})
     - Identify pairs of states from Impl\textsubscript{OP}, Spec\textsubscript{OP}
  3. Property
     - To hold by pairs identified in 3
Preserving Bad Activity

- Observation Projection ($\text{Impl}_{\text{OP}}, \text{Spec}_{\text{OP}}$)
  - $\varepsilon$-closure + determinization
  - Subset construction, pessimistic approach
    - $!$ is required iff at least one of original states requires it
    - $?\) is provided iff all of original states require it
    - follows “obs?” and “obs!” sets from interface automata (+ non-determinism)
  - Error states (bad activities caused by composition) $E$

- Alternation simulation $R(s_{\text{Impl}}, s_{\text{Spec}})$
  - $\text{leaving}(s_{\text{impl}}) \subseteq \text{leaving}(s_{\text{spec}})$ and $\text{leaving}(s_{\text{impl}}) \supseteq \text{leaving}(s_{\text{spec}})$
  - Similar to the IA property

- Property
  - Consider error states from $E$
    - $s_{\text{impl}} \in E_{\text{impl}} \Rightarrow s_{\text{spec}} \in E_{\text{spec}}$
    - $E_{\text{spec}}$ is typically empty
Preserving Bad Activity

\[ \text{obs}(x) \subseteq \text{obs}(x') \text{ and } \text{obs!(x)} \supseteq \text{obs!(x')} \]

\[ \text{ErrFree}(e \oplus x) \Rightarrow \text{ErrFree}(e \oplus x') \]
Preserving Bad Activity

\[ \text{obs}\, ?(x) \subseteq \text{obs}\, ?(x') \text{ and } \text{obs}\, !(x) \supseteq \text{obs}\, !(x') \]

\[ \text{ErrFree}(e \oplus x) \Rightarrow \text{ErrFree}(e \oplus x') \]

\[ \{ ?p \} \quad \{ \} \]
\[ \{ ?p, ?c \} \quad \{ \} \]

Error
Preserving No activity

• Observation Projection \((\text{Impl}_{\text{OP}}, \text{Spec}_{\text{OP}})\)
  - Add information about final states
    • \(F\) – final state – process can terminate
      - \((?a)^*\)
    • \(!F\) – active final state – process can decide to terminate (even if someone waits)
      - \((!a)^*\)
    • \(!F \subseteq F\)
  - Error states (bad and no activities caused by composition) \(E\)

• Alternation simulation \(R(s_{\text{Impl}}, s_{\text{Spec}})\)
  - Remains the same

• Property
  - Consider error states from \(E\)
    • \(s_{\text{Impl}} \in E_{\text{Impl}} \Rightarrow s_{\text{Spec}} \in E_{\text{Spec}}\)
  - \(\text{running}(s) \Leftrightarrow q \notin !F \& \text{there is an } ! \text{ transition leaving } s\)
    • The state \(q\) will emit an \(!\) event for sure
  - Preserve No activity
    • \((s_{\text{Spec}} \in F \Rightarrow (s_{\text{Impl}} \in F \lor \text{running}(s_{\text{Impl}})) \land (\text{running}(s_{\text{Spec}}) \Rightarrow \text{running}(s_{\text{Impl}})))\)
Preserving No activity

From the property

\[(s_{\text{Spec}} \in F \Rightarrow (s_{\text{Impl}} \in F \lor \text{running}(s_{\text{Impl}})) \land (\text{running}(s_{\text{Spec}}) \Rightarrow \text{running}(s_{\text{Impl}})))\]

we want to prove

\[\text{ErrFree}(E \oplus \text{Spec}) \Rightarrow \text{ErrFree}(E \oplus \text{Impl})\]

\[(\text{NoActivityFree}(E \oplus \text{Spec}) \Rightarrow \text{NoActivityFree}(E \oplus \text{Impl}))\]

\[\text{NoActivityFree}(E \oplus \text{Spec}) \Rightarrow\]

For each pair of states \(<s_{\text{Spec}}, e>\) put together by composition:

\[(s_{\text{Spec}} \in F \land e \in F) \lor \text{running}(s_{\text{Spec}}) \lor \text{running}(e)\]

The property implies

\[(s_{\text{Impl}} \in F \land e \in F) \lor \text{running}(s_{\text{Impl}}) \lor \text{running}(e)\]

Thus, \[\text{NoActivityFree}(E \oplus \text{Impl})\]
... and now for something completely different
Threads

- In a closed system one can make an assumption on fixed number of threads.
Threads

- In a closed system one can make an assumption on fixed number of threads

DataServer:

Player → Player

DataServer:
Threads

- In a closed system one can make an assumption on fixed number of threads

DataServer:

```
Player

DataServer
```

Critical area

Diagram showing the flow of data and interactions between Player and DataServer.
Threads

- In a closed system one can make an assumption on fixed number of threads

- The assumption is even more limiting for open systems (refinement)

DataServer:

```
?a^ !b !c !a$
```

Critical area
Refinement for limited number of threads

• Refinement should work for all environments
  - Assumption: environment won’t use more than n threads
Improvement – describe reentrancy

- Do not unroll the states in advance
  - Make unrolling to be a property of state
- Add the information about reentrancy (reent(s) = s’)
  - Semantics: ”at this point I can in parallel behave like another state”
- Add the information about critical sections (crit(s) = set of states)
  - Semantics: ”do not create a state involving more than one state”
- Use the information in the refinement property

A: ?a^ !b !c !a$ reent(s) crit(s)
Improvement – describe reentrancy

• Do not unroll the states in advance
  ▪ Make unrolling to be a property of state
• Add the information about reentrancy
  ▪ Semantics: “at this point I can in parallel behave like another state”
• Add the information about critical sections
  ▪ Semantics: “do not create a state involving more than two of states”
• Use the information in the refinement property

A: !a$ !c !b ?a^ Unroll(A,2): ?a^ !b !c !a$
Candidate on a property

- **Goal:**
  - $\text{Impl} \prec \text{Spec} \Rightarrow \forall k \text{ Unroll(Impl,k)} \prec \text{Unroll(Spec,k)}$

- **Assuming:**
  - reent(s) is followed only by ?
  - crit(s) are connected only by !

- $\text{a}_{\text{Impl}} < \text{a}_{\text{Spec}} \Rightarrow$
  - $\text{Reent(s}_{\text{Impl}}) = \text{a}_{\text{Impl}} \Rightarrow \exists \text{a}_{\text{Spec}}: \text{a}_{\text{Impl}} < \text{a}_{\text{Spec}} \land \text{Reent(s}_{\text{Spec}}) = \text{a}_{\text{Spec}}$
  - $\text{Crit(s}_{\text{Impl}}) = \text{C}_{\text{Impl}} \Rightarrow \exists \text{C}_{\text{Spec}}: \text{C}_{\text{Impl}} < \text{C}_{\text{Spec}} \land \text{Crit(s}_{\text{Spec}}) = \text{C}_{\text{Spec}}$
Candidate on a property
Candidate on a property
Future work (Threads)

• Formalize
  - So far, Bad activity considered

• Proof
  - Other assumptions may appear

• How to get the information
  - Means in TBP
    • Reentrancy operator (\(|^*\))
    • Mutexes

• Issues
  - Reentrancy and parallelism
  - Multiple critical sections
Questions ...

Thank you