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
LTS based formalisms

- Semantics given by LTS
  - Composition errors
    - Bad Activity
    - No Activity
    - Stack-freedom
- Process Algebras
- Interface automata
- Behavior Protocols

Refinement

- Broad concept
  - Intuitive meaning (used on yesterday seminar 😊)
  - Degree of non-determinism
  - Preorder relation
- Important property is preservation of correctness:
  \[ \text{Impl} \prec \text{Spec} \Rightarrow \forall E : \text{ErrFree}(E \oplus \text{Spec}) \Rightarrow \text{ErrFree}(E \oplus \text{Impl}) \]

Refinement examples

- CSP
  - Internal/External choice
- CCS
  - Preserving stuck-freedom
- Interface Automata
  - Based on alternation simulation
  - Preserving correctness w.r.t BA
- What about TBP
  - Preserving BA, NA

Parametrized alternation simulation

- 3 steps
  - Observation projection
  - Identify pairs of states
  - Check property P
  - P available for
    - Bad activity
    - No activity

Previously on DSRG seminar …
... and now conclusion
Threads

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

DataServer: ?a^ !b !c !a$
Threads

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

DataServer:

- Specification is (automatically) unrolled to accept more requests in parallel.
**Threads**

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

- Specification is (automatically) unrolled to accept more requests in parallel.
Threads

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

• Specification is (automatically) unrolled to accept more requests in parallel.
  - Fixed number of threads known in advance
  - Jiri’s “Unbounded parallelism” paper
Refinement for limited number of threads

- For open systems, the assumption is even more limiting
- Refinement should work for all environments
  - Assumption: environment won’t use more than k threads

\[ \forall E \text{ with limited parallelism}: \text{ErrFree}(E \oplus \text{Spec}) \Rightarrow \text{ErrFree}(E \oplus \text{Impl}) \]

Naïve approach
- Unroll both, Impl and Spec to accept k requests in parallel
- Use refinement for LTS-based formalism
- Apparently:
Improvement – describe reentrancy

- Do not unroll the states as the first step
  - Make the unrolling information to be a property of state
- Add the information about reentrancy \(\text{reent}(s) = s'\)
  - Semantics: "at this point I can in parallel behave like another state"
- Add the information about critical sections \(\text{crit}(s) = \text{set of states}\)
  - Semantics: "do not create a state involving more than one state from area"

\[ \text{A: } ?a^\wedge \text{ !b !c !a} \]

- Use the unrolling information in the refinement property
Improvement – describe reentrancy

• Do not unroll the states as the first step
  ▪ Make the unrolling information 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 from area”

A: \( ?a^\land b \rightarrow c \rightarrow !a \)  
Unroll(A,2): \( \begin{align*} 
?a^\land !b \rightarrow !c \rightarrow !a \end{align*} \)

• Use the unrolling information in the refinement property
Property

• Come up with a property such that:

\[ \text{Impl} \prec \text{Spec} \Rightarrow \forall k \ \text{Unroll}(\text{Impl},k) \prec \text{Unroll}(\text{Spec},k) \]
Property

• Come up with a property such that:

\[ \text{Impl} < \text{Spec} \Rightarrow \forall k \ \text{Unroll}(\text{Impl},k) < \text{Unroll}(\text{Spec},k) \]

Consequence of Impl<Spec

No need to check
No \( k \) is needed

Standard LTS refinement

Refinement for LTS with reentrancy info
Candidate on a property (just for BA)

• Goal:
  - Impl<Spec => ∀k Unroll(Impl,k)<Unroll(Spec,k)

• Assuming:
  - reent(s) is followed only by ?

• a_{Impl} <a_{Spec} =>
  - Reent(s_{Impl}) = a_{Impl} => ∃a_{spec}: a_{Impl} <a_{spec} ∧ Reent(s_{spec}) = a_{spec}
  - Crit(s_{Impl}) = C_{Impl} => ∃C_{spec}: C_{Impl} <C_{spec} ∧ Crit(s_{spec}) = C_{spec}
Candidate on a property

![Diagram showing a property candidate with nodes and edges labeled with logical symbols: ?a^, !b, !c, !a$.]
Candidate on a property

- Environment is able to accept ![b] in parallel.
- Environment does not require multiple ![a] simultaneously.

**Critical area**

Tomáš Poch, František Plášil

CBSE 09
Future work (Threads)

• Formalize
  • Only 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
Conclusion

• Refinement for TBP done
  - For limited number of threads
  - Preserves correctness w.r.t Bad activity and No activity

• Unlimited threads
  - Lot of work to be done
Questions ...