Decomposing Refinement Proofs using

Assume-Guarantee Reasoning

Tom Henzinger (UC Berkeley)
Shaz Qadeer (Compaq Research)
Sriram Rajamani (Microsoft Research)
Outline

1 Informal Introduction

2 Formal Definitions
   Reactive Systems
   Witnessed Refinement Proofs
   Slicing Reactive Systems
   Decomposing Refinement Proofs

3 Formal Example: Three-Stage Pipeline

4 Informal Example: Dataflow Processor Array
Outline

1 Informal Introduction

2 Formal Definitions
   - Reactive Systems
   - Witnessed Refinement Proofs
   - Slicing Reactive Systems
   - Decomposing Refinement Proofs

3 Formal Example: Three-Stage Pipeline

4 Informal Example: Dataflow Processor Array
Refinement Proof

Implementation
e.g. processor

<

Specification
e.g. ISA

"conforms with",
"refines",
"implements"

Every I/O behavior of impl. is an I/O behavior of spec.
State Explosion!

Approaches:

1. More efficient algorithms and data structures. But there are exponential lower bounds...

2. Decompose the refinement problem. Divide-and-conquer!!!
Decomposed Refinement Proof
Decomposed Refinement Proof
Decomposed Refinement Proof
Unsuccessful Proof Decomposition
Unsuccessful Proof Decomposition

Unconstrained

<
Unsuccessful Proof Decomposition

unconstrained

unconstrained
Unsuccessful Proof Decomposition

unconstrained

\[ x(0) = 0 \]
\[ y(0) = 0 \]
\[ x(t+1) = y(t) \]
\[ y(t+1) = x(t) \]

unconstrained

\[ x(t) = 0 \]
\[ y(t) = 0 \]
Circular Proof Decomposition
Circular Proof Decomposition
Circular Proof Decomposition

\[\text{Diagram showing the decomposition of circular proofs.} \]
Legal Circular Proof Decomposition

\[ x(0) = 0 \]
\[ x(t+1) = y(t) \]
\[ y(t+1) = x(t) \]

\[ S(x) \]
\[ S(y) \]

\[ S(z) : z(t) = 0 \]

"I will not launch."
Illegal Circular Proof Decomposition 1

\[ x(0) = 0 \]
\[ x(t+1) = y(t) \]
\[ y(0) = 0 \]
\[ y(t+1) = x(t) \]

\[ L(x) \]
\[ L(y) \]

L(z): exists k s.t. \( z(t) = \begin{cases} 0 & \text{if } t < k \\ 1 & \text{else} \end{cases} \)

“\( I \) will disarm.”
Illegal Circular Proof Decomposition 2

\[ x(t) = y(t) \]
\[ y(t) = 1 \]
\[ x(t) = 0 \]
\[ y(t) = \neg x(t) \]
Illegal Circular Proof Decomposition 2

\[
x(t) = y(t)
\]

\[
y(t) = 1
\]

\[
x(t) = 0 \quad y(t) = \neg x(t)
\]
Illegal Circular Proof Decomposition

\[ x(t) = \sim y(t) \]
\[ y(t) = z(t) \]
\[ z(t) = 0 \]

\[ x(t) = \sim y(t) \]
\[ y(t) = z(t) \]
\[ z(t) = 0 \]
Illegal Circular Proof Decomposition

\[
x(t) = \neg y(t)
\]
\[
y(t) = z(t)
\]
\[
z(t) = 0
\]
Illegal Circular Proof Decomposition

\[ x(t) = \neg y(t) \quad \neg y(t) = z(t) \quad z(t) = 0 \]

\[ x(t) = 0 \quad y(t) = 1 \quad z(t) = \neg x(t) \]
Illegal Circular Proof Decomposition

\[ x(t) = \sim y(t) \]
\[ y(t) = z(t) \]
\[ z(t) = 0 \]

\[ x(t) = 0 \]
\[ y(t) = 1 \]
\[ z(t) = \sim x(t) \]
Illegal Circular Proof Decomposition 3
Illegal Circular Proof Decomposition

\[ x(t) = \neg y(t) \quad y(t) = z(t) \quad z(t) = 0 \]

\[ x(t) = 0 \quad y(t) = 1 \quad z(t) = \neg x(t) \]
Illegal Circular Proof Decomposition

\[x(t) = \sim y(t) \quad y(t) = z(t) \quad z(t) = 0\]
Circular Assume-Guarantee Reasoning

For program verification:
  Chandy & Misra [1981]
  Abadi & Lamport [1993]

For model checking:
  McMillan [1997] → SMV

Large applications:
  Eiriksson [1998]: 1M-gate ASICs (SMV)
  Liu, Qadeer, Rajamani [1999]: 64-processor array (Mocha)
1  Informal Introduction

2  Formal Definitions
   Reactive Systems
   Witnessed Refinement Proofs
   Slicing Reactive Systems
   Decomposing Refinement Proofs

3  Formal Example: Three-Stage Pipeline

4  Informal Example: Dataflow Processor Array
System $S[P;Q]$

input signals

input ports $P=\{1,2,3\}$

output ports $Q=\{4,5\}$

output signals

x1: Nat -> Bool

1

x2: Nat -> Bool

2

x3: Nat -> Bool

3

S

4

5

y4: Nat -> Bool

y5: Nat -> Bool
System $S[P;Q]$}

**Input signals**
- $x_1: \text{Nat} \rightarrow \text{Bool}$
- $x_2: \text{Nat} \rightarrow \text{Bool}$
- $x_3: \text{Nat} \rightarrow \text{Bool}$

**Input ports**
- $P = \{1, 2, 3\}$

**Output signals**
- $y_4: \text{Nat} \rightarrow \text{Bool}$
- $y_5: \text{Nat} \rightarrow \text{Bool}$

**Output ports**
- $Q = \{4, 5\}$

**I/O function**
- $S(x) = y$

**Input behavior**
- $x = (x_1, x_2, x_3)$

**Output behavior**
- $y = (y_4, y_5)$
"Moore" System $S[P;Q;\text{init};\text{next}]$

Inductive definition:
Given input behavior $x$, unique output behavior $y$.

$y(0) = \text{init}$
$y(t+1) = \text{next} \ [\ y(t), \ x(t+1) \ ]$

$\text{init} \ \text{in} \ \mathsf{Vals}(Q)$
$\text{next}: \ \mathsf{Vals}(Q) \times \mathsf{Vals}(P) \rightarrow \mathsf{Vals}(Q)$
Reactive System $S[P;Q;\text{init};\text{next}]$

- $y(0) = \text{init}$
- $y_{4(t+1)} = \text{next}^4 \left[ y(t), \ x(t+1) \right]$ 
- $y_{5(t+1)} = \text{next}^5 \left[ y(t), y_{4(t+1)}, x(t+1) \right]$

4 >> 1, 2, 3
5 >> 1, 2, 3, 4

No combinational loop:
Given input behavior $x$, unique output behavior $y$. 
**Mealy System $S[P;Q;\text{init};\text{next}]$**

Combinational loop:
no output behavior $y$.

\[ y_4(t) = y_5(t) \]
\[ y_5(t) = \sim y_4(t) \]

4 \gg 5
5 \gg 4
Mealy System $S[P; Q; \text{init}; \text{next}]$

Combinational loop:
no output behavior $y$, or multiple output behaviors $y$. 

$y_4(t) = y_5(t)$
$y_5(t) = y_4(t)$

$x_1, x_2, x_3$ to $y_4, y_5$
Outline

1 Informal Introduction

2 Formal Definitions
   Reactive Systems
   Witnessed Refinement Proofs
   Slicing Reactive Systems
   Decomposing Refinement Proofs

3 Formal Example: Three-Stage Pipeline

4 Informal Example: Dataflow Processor Array
For every I/O behavior $(x_1, x_2, x_4, x_5, x_6)$ of $S$, there is an I/O behavior $(x_1, x_3, x_4, x_5, x_7)$ of $S'$. 
Refinement

\[ S[P;Q;\text{init};\text{next}] < S'[P';Q';\text{init'};\text{next'}] \] ?

Time complexity:

\[ 2^{O(|P| + |Q| + |P'| + 2|Q'|)} \]
Refinement

\[ S[P;Q;\text{init};\text{next}] < S'[P';Q';\text{init'};\text{next'}] \ ? \]

Time complexity:

- If \( P' \cup Q' \) subset of \( P \cup Q \):
  \[ O( |P| + |Q| ) \]

- Otherwise:
  \[ O( |P| + |Q| + |P'| + 2^{|Q'|} ) \]

“Witnessed” Refinement
Witnessed Refinement

Every specification port is an implementation port.

For every I/O behavior \((x_1, x_2, x_4, x_5, x_6)\) of \(S\), there is an I/O behavior \((x_1, x_3, x_4, x_5, x_7)\) of \(S'\).
Witnessed Refinement

Every specification port is an implementation port.

For every I/O behavior \((x_1,x_2,x_3,x_4,x_5,x_6)\) of \(S\), there is an I/O behavior \((x_1,x_3,x_4,x_5,x_7)\) of \(S'\).
Witnessed Refinement

Every specification port is an implementation port.

For every I/O behavior \((x_1, x_2, x_3, x_4, x_5, x_6)\) of \(S\), there is an I/O behavior \((x_1, x_3, x_4, x_5, x_7)\) of \(S'\).
Witnessed Refinement

Every specification port is an implementation port.

For every I/O behavior \((x_1, x_2, x_3, x_4, x_5, x_6, x_7)\) of \(S\), there is an I/O behavior \((x_1, x_3, x_4, x_5, x_7)\) of \(S'\).
Witnessed Refinement

Every specification port is an implementation port.

For every I/O behavior \((x_1,x_2,x_3,x_4,x_5,x_6,x_7)\) of \(S\), there is an I/O behavior \((x_1,x_3,x_4,x_5,x_7)\) of \(S'\).
Witnessed Refinement

For every I/O behavior \((x_1, x_2, x_3, x_4, x_5, x_6, x_7)\) of \(S\), the projection \((x_1, x_3, x_4, x_5, x_7)\) is an I/O behavior of \(S'\).

Every specification port is an implementation port.
Checking Witnessed Refinement:

\[ S[P;Q;\text{init};\text{next}] < S'[P';Q';\text{init'};\text{next'}] \]

1. Find the reachable states of the implementation \( S \).
2. For each reachable state \( s \), check that

\[ \text{next}(s[P;Q]) \mid_{Q'} = \text{next'}(s[P';Q']) \]

This requires only a small extension of any reachability engine.

See, for example, [www.eecs.berkeley.edu/~mocha](http://www.eecs.berkeley.edu/~mocha).
 Outline

1 Informal Introduction

2 Formal Definitions 
   Reactive Systems 
   Witnessed Refinement Proofs 
   Slicing Reactive Systems 
   Decomposing Refinement Proofs 

3 Formal Example: Three-Stage Pipeline 

4 Informal Example: Dataflow Processor Array
Composition $S_1[P_1;Q_1] \parallel S_2[P_2;Q_2]$

1. $Q_1$ and $Q_2$ disjoint
2. $\ll_1 \cup \ll_2$ acyclic
Composition  $S1[P1;Q1] \parallel S2[P2;Q2]$
\[(S_1 \parallel S_2) \ [ (P_1 \cup P_2) \setminus (Q_1 \cup Q_2); \ Q_1 \cup Q_2]\]
$Q'$-Slice of $S_{[P;Q]}$

$Q'$ subset of $Q$
$Q'$-Slice of $S_{[P;Q]}$

$Q'$ subset of $Q$

$Q' = \{3, 4\}$
Q'-Slice of $S[P;Q]$
Q'-Slice of $S[P;Q]$
Q'-Slice of S[P;Q]
\[(S|_{Q'}) [P u (Q'\setminus Q'); Q']\]
1 Informal Introduction

2 Formal Definitions
   Reactive Systems
   Witnessed Refinement Proofs
   Slicing Reactive Systems
   Decomposing Refinement Proofs

3 Formal Example: Three-Stage Pipeline

4 Informal Example: Dataflow Processor Array
Weak Decomposition Rule
Weak Decomposition Rule

One subproof for each specification output port
Weak Decomposition Rule

One subproof for each specification output port
Weak Decomposition Rule

One subproof for each specification output port
Weak Decomposition Rule

One subproof for each specification output port
Weak Decomposition Rule

One subproof for each specification output port
Weak Decomposition Rule

Any slice of impl.

3-slice of spec.

One subproof for each specification output port
Weak Decomposition Rule

One subproof for each specification output port.
Strong (Assume-Guarantee) Decomposition Rule

One subproof for each specification output port
Strong (Assume-Guarantee) Decomposition Rule

One subproof for each specification output port
Strong (Assume-Guarantee) Decomposition Rule

Assumption: Any ~3-slice of spec.

Any slice of impl.

3-slice of spec.

One subproof for each specification output port.
Example

\[ x(0) = 0 \]
\[ x(t+1) = y(t) \]
\[ y(0) = 0 \]
\[ y(t+1) = x(t) \]
Example

$x(0)=0$
$x(t+1)=y(t)$
$y(0)=0$
$y(t+1)=x(t)$
Example

\[
\begin{align*}
  x(0) &= 0 \\
  x(t+1) &= y(t) \\
  y(0) &= 0 \\
  y(t+1) &= x(t)
\end{align*}
\]
Example
Example

\[ x(0) = 0 \]
\[ x(t+1) = y(t) \]
\[ y(t) = 0 \]

\[ y(0) = 0 \]
\[ y(t+1) = x(t) \]
Example

\[ x(t+1) = y(t) \]
\[ y(t+1) = x(t) \]

\[ x(t) = 0 \]
\[ y(t) = 0 \]
The Need for Abstractions

One subproof for each specification output port.
The Need for Abstractions

One subproof for each specification output port

One subproof for each specification output port

One subproof for each specification output port

One subproof for each specification output port
The Need for Abstractions

One subproof for each specification output port

Construct abstraction for 6
The Need for Abstractions

One subproof for each specification output port, including abstractions.
The Need for Abstractions

One subproof for each specification output port, including abstractions.
The Need for Abstractions

Abstraction for 6 involves new 7
The Need for Abstractions

Copy witness for 7

Abstraction for 6 involves new 7
The Need for Abstractions

One subproof for each specification output port, including abstractions.
The Need for Abstractions

One subproof for each specification output port, including abstractions.
The Need for Abstractions

One subproof for each specification output port, including abstractions.