An Efficient Runtime Validation Framework based on the Theory of Refinement

Mitesh Jain, Synopsys
Pete Manolios, Northeastern University

Property-based Testing Methodology

Given an implementation

1. Define a set of properties
2. Design a test suite and define oracles
3. Execute tests and check for property violations
4. Analyze the violations and fix
   - Implementation
   - Properties

Property-based Testing Methodology

Given an implementation

1. Define a set of properties
2. Design a test suite and define oracles
3. Execute tests and check for property violations
4. Analyze the violations and fix
   - Implementation
   - Properties

Difficult to determine if the specification is complete

Property-based Testing Methodology

Given an implementation

1. Define a set of properties
2. Design a test suite and define oracles
3. Execute tests and check for property violations
4. Analyze the violations and fix
   - Implementation
   - Properties

Difficult to determine if the specification is complete

Defining oracles is expensive and error-prone

Property-based Testing Methodology

Given an implementation

1. Define a set of properties
2. Design a test suite and define oracles
3. Execute tests and check for property violations
4. Analyze the violations and fix
   - Implementation
   - Properties

Difficult to determine if the specification is complete

Defining oracles is expensive and error-prone

Changes in specification

Property-based Testing Methodology

Implementation

IF	ID	RF	EX	WB		
IF	ID	RF	EX	WB		
	IF	ID	RF	EX	WB	
	IF	ID	RF	EX	WB	
		IF	ID	RF	EX	WB
		IF	ID	RF	EX	WB

Set of Properties

1. Correct arithmetic operations
2. Detecting and stalling the pipeline on data hazards
3. Branch/Jump instructions
4. ...

Property-based Testing Methodology

Implementation

IF	ID	RF	EX	WB		
IF	ID	RF	EX	WB		
	IF	ID	RF	EX	WB	
	IF	ID	RF	EX	WB	
		IF	ID	RF	EX	WB
		IF	ID	RF	EX	WB

Set of Properties

1. Correct arithmetic operations
2. Detecting and stalling the pipeline on data hazards
3. Branch/Jump instructions
4. . . .

Trends in Functional Verification: A 2014 Industry Study[1]

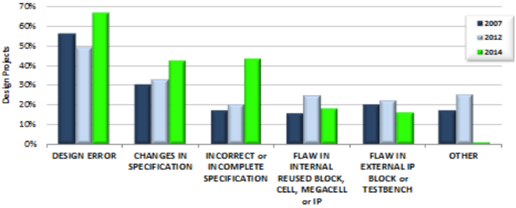


Figure 17. Root Cause of Functional Flaws

[1] Harry Foster, DAC 2015 (reproduced)

Trends in Functional Verification: A 2014 Industry Study[1]

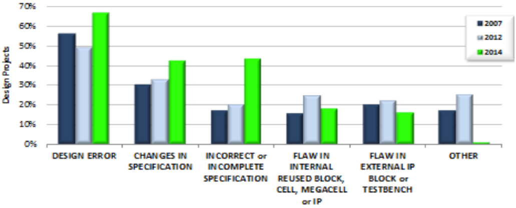


Figure 17. Root Cause of Functional Flaws

Refinement-based testing methodology

[1] Harry Foster, DAC 2015 (reproduced)

Trends in Functional Verification: A 2014 Industry Study[1]

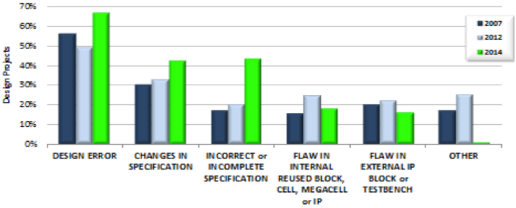


Figure 17. Root Cause of Functional Flaws

Refinement-based testing methodology

Compile the refinement conjecture to a runtime check

[1] Harry Foster, DAC 2015 (reproduced)

Refinement-based Testing Methodology

Implementation $\lesssim_r$ **Specification**

IF	ID	RF	EX	WB		
IF	ID	RF	EX	WB		
	IF	ID	RF	EX	WB	
	IF	ID	RF	EX	WB	
		IF	ID	RF	EX	WB
		IF	ID	RF	EX	WB

Instruction Set Architecture

- *add rd, ra, rb*
- *sub rd, ra, rb*
- *jnz imm*
- . . .

Concrete System **Abstract System**

Refinement-based Testing Methodology

Implementation $\lesssim_r$ **Specification**

IF	ID	RF	EX	WB		
IF	ID	RF	EX	WB		
	IF	ID	RF	EX	WB	
	IF	ID	RF	EX	WB	
		IF	ID	RF	EX	WB
		IF	ID	RF	EX	WB

Instruction Set Architecture

- *add rd, ra, rb*
- *sub rd, ra, rb*
- *jnz imm*
- ...

Concrete System

Abstract System

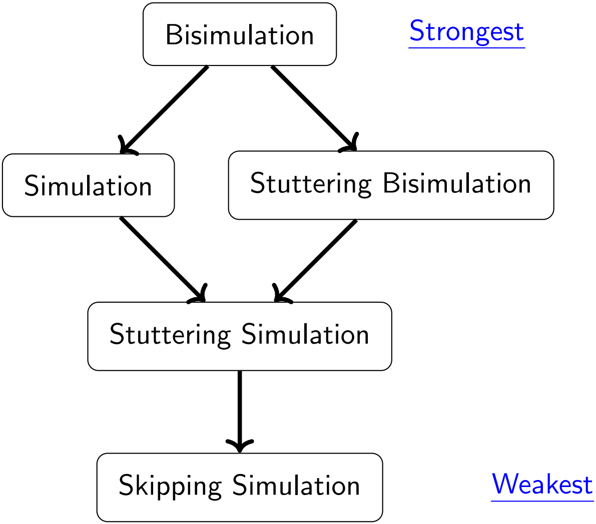
One Property

Challenges

- $\mathcal{A}$ and $\mathcal{C}$ may differ
  - Data representation
  - Number of state components
  - Atomicity of a computation step
  - ...

Relate behaviors of systems expressed at different levels of abstraction

Notions Of Refinement



Applications

- Microprocessor verification
- Compiler verification
- Distributed/Concurrent systems
- Microkernels
- ...

Applications

- Microprocessor verification
- Compiler verification
- Distributed/Concurrent systems
- Microkernels
- ...

Local characterizations for effective reasoning

Refinement Conjecture

A local characterization of Skipping Simulation

$$\begin{tikzcd} s \arrow[r, dash] \arrow[d] & w \arrow[d] \\ u \arrow[r, dash] & v \end{tikzcd}$$

one step

$\langle \exists v : w \rightarrow v : uBv \rangle$

$$\begin{tikzcd} s \arrow[r, dash] \arrow[d] & w \arrow[d, "\geq 2"] \\ u \arrow[r, dash] & v \end{tikzcd}$$

skipping on the right

$\langle \exists v : w \rightarrow^{\geq 2} v : uBv \rangle \rangle$

$$\begin{tikzcd} s \arrow[r, dash] \arrow[d] & w \arrow[dl, dash] \\ u & \end{tikzcd}$$

stuttering on left

$(uBw \wedge rankT(u, w) \prec rankT(s, w))$

$$\begin{tikzcd} s \arrow[r, dash] \arrow[d] \arrow[dr, dash] & w \arrow[d] \\ u & v \end{tikzcd}$$

stuttering on right

$\langle \exists v : w \rightarrow v : sBv \wedge$

$rankL(v, s, u) < rankL(w, s, u) \rangle$

Refinement Conjecture

A local characterization of Skipping Simulation

$$\begin{tikzcd} s \arrow[r, dash] \arrow[d] & w \arrow[d] \\ u \arrow[r, dash] & v \end{tikzcd}$$

one step

$\langle \exists v : w \rightarrow v : uBv \rangle$

Concrete system does not skip

$$\begin{tikzcd} s \arrow[r, dash] \arrow[d] & w \\ u \arrow[ur, dash] & \end{tikzcd}$$

stuttering on left

$(uBw \wedge rankT(u, w) \prec rankT(s, w))$

$$\begin{tikzcd} s \arrow[r, dash] \arrow[d] \arrow[dr, dash] & w \arrow[d] \\ u & v \end{tikzcd}$$

stuttering on right

$\langle \exists v : w \rightarrow v : sBv \wedge$
$rankL(v, s, u) < rankL(w, s, u) \rangle$

Refinement Conjecture

A local characterization of Skipping Simulation

$$
\begin{tikzcd}
s \arrow[r, dash] \arrow[d] & w \arrow[d] \\
u \arrow[r, dash] & v
\end{tikzcd}
$$

one step

$\langle \exists v : w \rightarrow v : uBv \rangle$

Concrete system does not skip

$$
\begin{tikzcd}
s \arrow[r, dash] \arrow[d] & w \\
u \arrow[ur, dash] &
\end{tikzcd}
$$

stuttering on left

$(uBw \wedge rankT(u, w) \prec rankT(s, w))$

Abstract system does not stutter

Refinement Conjecture

Runtime Checker

$$\begin{tikzcd} s \arrow[r, dash] \arrow[d] & w \arrow[d] \\ u \arrow[r, dash] & v \end{tikzcd}$$

one step

$\langle \exists v : w \rightarrow v : uBv \rangle$

$$\begin{tikzcd} s \arrow[r, dash] \arrow[d] & w \\ u \arrow[ur, dash] & \end{tikzcd}$$

stuttering on left

$(uBw \wedge rankT(u, w) \prec rankT(s, w))$

$$\xrightarrow{\textit{Compile}}$$

Algorithm 1: Refinement Check

Input : s: concrete system state
n: number of steps to run
r: refinement map
$rankT$: rank of concrete state

Output: Partition, Error Status

$w \leftarrow r(s)$; $error \leftarrow false$; $partition \leftarrow \langle\rangle$; $i \leftarrow 0$; $j \leftarrow 0$;

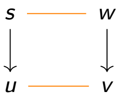
do
- $u \leftarrow Select\text{-}concrete\text{-}next\text{-}state(s)$;
- $\boxed{\langle match, v\rangle \leftarrow Match\text{-}abstract\text{-}next\text{-}state(w, u)}$;
- **if** $match$
  - $partition \leftarrow partition :: \langle i, j\rangle$;
  - $i \leftarrow i + 1$; $j \leftarrow j + 1$;
  - $s \leftarrow u$; $w \leftarrow v$;
- **else if** $\boxed{r(u) = w \ \wedge \ rankT(u, w) \prec rankT(s, w)}$
  - $i \leftarrow i + 1$;
  - $s \leftarrow u$;
- **else** $error \leftarrow true$;
- $n \leftarrow n - 1$;

while $n > 0 \wedge \neg error$;

return $\langle partition, error\rangle$;

Refinement Conjecture

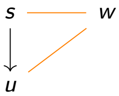
Runtime Checker



one step

$\langle \exists v : w \rightarrow v : uBv \rangle$

Compile $\longrightarrow$



stuttering on left

$(uBw \wedge rankT(u, w) \prec rankT(s, w))$

Algorithm 1: Refinement Check

Input : s: concrete system state
n: number of steps to run
r: refinement map
$rankT$: rank of concrete state

Output: Partition, Error Status

$w \leftarrow r(s)$; $error \leftarrow false$; $partition \leftarrow \langle\rangle$; $i \leftarrow 0$;
$j \leftarrow 0$;

do

  $u \leftarrow Select\text{-}concrete\text{-}next\text{-}state(s)$;

  $\boxed{\langle match, v\rangle \leftarrow Match\text{-}abstract\text{-}next\text{-}state(w, u)}$;

  if $match$

    $partition \leftarrow partition :: \langle i, j\rangle$;

    $i \leftarrow i + 1$; $j \leftarrow j + 1$;

    $s \leftarrow u$; $w \leftarrow v$;

  else if

  $\boxed{r(u) = w \ \wedge \ rankT(u, w) \prec rankT(s, w)}$

    $i \leftarrow i + 1$;

    $s \leftarrow u$;

  else $error \leftarrow true$;

  $n \leftarrow n - 1$;

while $n > 0 \wedge \neg error$;

return $\langle partition, error\rangle$;

Local proof method $\rightsquigarrow$ efficient runtime refinement checker

Evaluation: RISC-V Sodor

- 5-stage pipeline Sodor Processor
  - Single issue in-order pipeline processor
  - Supports full bypassing between functions units
- Spike, executable RISC-V ISA simulator
  - High-level specification
  - Serves as the oracle

Evaluation

- Effectiveness in detecting bugs
- Overhead cost of the refinement checker

Evaluation

- Effectiveness in detecting bugs

Errors Injected	Detect
Instruction classification	✓
Control: Stall Mechanism	✓
Control: Pipeline hazard detection	✓
Arithmetic-logic unit	✓
Load store unit	✓

- Overhead cost of the refinement checkers

Evaluation

- Effectiveness in detecting bugs

Errors Injected	Detect
Instruction classification	✓
Control: Stall Mechanism	✓
Control: Pipeline hazard detection	✓
Arithmetic-logic unit	✓
Load store unit	✓

- Overhead cost of the refinement checkers

Evaluation

- Effectiveness in detecting bugs
- Overhead cost of the refinement checker

  1. Match concrete and abstract states
  2. Computing the refinement map
  3. Computing the rank of a concrete state

Algorithm 1: Refinement Check

...
do
  $u \leftarrow \textit{Select-concrete-next-state}(s)$;
  $\boxed{\langle \textit{match}, v\rangle \leftarrow \textit{Match-abstract-next-state}(w, u)}$;

  if *match* **then**
    $\textit{partition} \leftarrow \textit{partition} :: \langle i, j\rangle$;
    $i \leftarrow i + 1$; $j \leftarrow j + 1$;
    $s \leftarrow u$; $w \leftarrow v$;
  end
  else if $\boxed{r(u) = w \ \wedge \ \textit{rankT}(u) < \textit{rankT}(s)}$
   then
    $i \leftarrow i + 1$;
    $s \leftarrow u$;
  end
  else $\textit{error} \leftarrow \textit{true}$;
  $n \leftarrow n - 1$;
while $n > 0 \wedge \neg \textit{error}$;
return $\langle \textit{partition}, \textit{error}\rangle$;

Evaluation

- Effectiveness in detecting bugs
- Overhead cost of the refinement checker

1. Match concrete and abstract states
2. Computing the refinement map
3. Computing the rank of a concrete state

Conclusion

An alternate testing methodology based on the theory of refinement.

Refinement implies functional correctness

Algorithm 1: Refinement Check

Input : s: concrete system state
n: number of steps to run
r: refinement map
$rankT$: rank of concrete state

Output: Partition, Error Status

$w \leftarrow r(s)$; $error \leftarrow false$; $partition \leftarrow \langle\rangle$; $i \leftarrow 0$;
$j \leftarrow 0$;

do

$u \leftarrow Select\text{-}concrete\text{-}next\text{-}state(s)$;

$\boxed{\langle match, v\rangle \leftarrow Match\text{-}abstract\text{-}next\text{-}state(w, u)}$;

if $match$

$partition \leftarrow partition :: \langle i, j\rangle$;

$i \leftarrow i + 1$; $j \leftarrow j + 1$;

$s \leftarrow u$; $w \leftarrow v$;

else if

$\boxed{r(u) = w \ \wedge \ rankT(u, w) \prec rankT(s, w)}$

$i \leftarrow i + 1$;

$s \leftarrow u$;

else $error \leftarrow true$;

$n \leftarrow n - 1$;

while $n > 0 \wedge \neg error$;

return $\langle partition, error\rangle$;

Conclusion

An alternate testing methodology based on the theory of refinement.

Refinement implies functional correctness

Abstract system serves as the oracle

Algorithm 1: Refinement Check

Input : s: concrete system state
n: number of steps to run
r: refinement map
$rankT$: rank of concrete state

Output: Partition, Error Status

$w \leftarrow r(s)$; $error \leftarrow false$; $partition \leftarrow \langle\rangle$; $i \leftarrow 0$;
$j \leftarrow 0$;

do

$u \leftarrow$ *Select-concrete-next-state*(s);

$\boxed{\langle match, v\rangle \leftarrow \textit{Match-abstract-next-state}(w, u)}$;

if $match$

$partition \leftarrow partition :: \langle i, j\rangle$;

$i \leftarrow i + 1$; $j \leftarrow j + 1$;

$s \leftarrow u$; $w \leftarrow v$;

else if

$\boxed{r(u) = w \ \wedge \ rankT(u, w) \prec rankT(s, w)}$

$i \leftarrow i + 1$;

$s \leftarrow u$;

else $error \leftarrow true$;

$n \leftarrow n - 1$;

while $n > 0 \wedge \neg error$;

return $\langle partition, error\rangle$;

Conclusion

An alternate testing methodology based on the theory of refinement.

Refinement implies functional correctness

Abstract system serves as the oracle

Robust to changes in the implementation

Algorithm 1: Refinement Check

Input : s: concrete system state
n: number of steps to run
r: refinement map
$rankT$: rank of concrete state

Output: Partition, Error Status

$w \leftarrow r(s)$; $error \leftarrow false$; $partition \leftarrow \langle\rangle$; $i \leftarrow 0$; $j \leftarrow 0$;

do

$u \leftarrow Select\text{-}concrete\text{-}next\text{-}state(s)$;

$\boxed{\langle match, v\rangle \leftarrow Match\text{-}abstract\text{-}next\text{-}state(w, u)}$;

if $match$

$partition \leftarrow partition :: \langle i, j\rangle$;

$i \leftarrow i + 1$; $j \leftarrow j + 1$;

$s \leftarrow u$; $w \leftarrow v$;

else if

$\boxed{r(u) = w \ \wedge \ rankT(u, w) \prec rankT(s, w)}$

$i \leftarrow i + 1$;

$s \leftarrow u$;

else $error \leftarrow true$;

$n \leftarrow n - 1$;

while $n > 0 \wedge \neg error$;

return $\langle partition, error\rangle$;

Thank You

References

- An efficient refinement-based testing methodology

Skipping refinement
Mitesh Jain and Pete Manolios
CAV, 2015, 2017

Proving skipping refinement using ACL2s
Mitesh Jain and Pete Manolios
ACL2, 2015