

# Dynamic Symbolic Execution of Distributed Concurrent Objects

Andreas Griesmayer

UNU-IIST Macau



United Nations  
University

**UNU-IIST**

International Institute for  
Software Technology

**Introduction**

Motivation

Outline

Dynamic Symbolic  
Execution

Application: Testing

# Introduction

# Motivation

Introduction

**Motivation**

Outline

Dynamic Symbolic  
Execution

Application: Testing

- Setting
  - ◆ Distributed (concurrent) systems
  - ◆ Executable models
  
- Goals
  - ◆ Examine models
  - ◆ Improve understanding of the code
  - ◆ Identify equivalence classes of inputs

# Outline

Introduction

Motivation

Outline

Dynamic Symbolic  
Execution

Application: Testing

- Introduction to (dynamic) symbolic execution
- DSE & concurrency
- Important properties of Creol
- Extension of DSE to Creol
- Application to testing

# Symbolic Execution

## Introduction

## Dynamic Symbolic Execution

### Symbolic Exec

DSE

Concurrency

Creol

A Run

RW Rule

Async. Calls

DSE RW Rules

## Application: Testing

- Values as symbolic expression instead of concrete data
- Succinct representation
- *Forking* on conditional statements
- Elimination of infeasible paths

# Symbolic Execution

## Introduction

## Dynamic Symbolic Execution

### Symbolic Exec

DSE

Concurrency

Creol

A Run

RW Rule

Async. Calls

DSE RW Rules

## Application: Testing

- Values as symbolic expression instead of concrete data
- Succinct representation
- *Forking* on conditional statements
- Elimination of infeasible paths

---

```
foo(int x)                                     1
  if (x > 0) then                               2
    x = x - 1;                                   3
    if (x < 0) then                               4
      x = 0;                                     5
    end                                          6
  end                                          7
end
```

# Dynamic Symbolic Execution

## Introduction

## Dynamic Symbolic Execution

### Symbolic Exec

### DSE

### Concurrency

### Creol

### A Run

### RW Rule

### Async. Calls

### DSE RW Rules

## Application: Testing

- Symbolic execution for a single run
- Run selected by “concrete execution”
- Symbolic execution in parallel to concrete execution
- Decisions along the path are recorded → *path-condition*
- Generalization of the run  
→ path condition represents values that perform the same run

# Concurrency

## Introduction

## Dynamic Symbolic Execution

Symbolic Exec  
DSE

## Concurrency

Creol  
A Run  
RW Rule  
Async. Calls  
DSE RW Rules

## Application: Testing

- Different tasks access the same data
- Task switch even during execution of a statement possible  
→ Race conditions
- State space explosion due to task switches

# Concurrency

## Introduction

## Dynamic Symbolic Execution

Symbolic Exec  
DSE

## Concurrency

Creol

A Run

RW Rule

Async. Calls

DSE RW Rules

## Application: Testing

- Different tasks access the same data
- Task switch even during execution of a statement possible
  - Race conditions
- State space explosion due to task switches
  - ⇒ use Creol as modeling language

# Creol

- Object oriented modeling language
- Decouples communication and synchronization
- Max. one active process per object
- Strict encapsulation of state
- Semantic formally defined
- Formal semantics directly gives an interpreter in Maude

## Introduction

## Dynamic Symbolic Execution

Symbolic Exec

DSE

Concurrency

**Creol**

A Run

RW Rule

Async. Calls

DSE RW Rules

## Application: Testing

# A Run in Creol

## Introduction

## Dynamic Symbolic Execution

Symbolic Exec

DSE

Concurrency

Creol

**A Run**

RW Rule

Async. Calls

DSE RW Rules

## Application: Testing

- Captures the parallel execution of different concurrent objects
- Concurrent executions are *independent*
  - ◆ may be perceived as one step, or in any sequential order
- Rename variables to unique name (according to scope)
- Conditions along the path are recorded
- Statements from the concrete execution are logged and executed symbolically
- Additional mapping for messages

# Example Rewrite Rules

## Introduction

## Dynamic Symbolic Execution

Symbolic Exec

DSE

Concurrency

Creol

A Run

**RW Rule**

Async. Calls

DSE RW Rules

## Application: Testing

---

---	await	1
---		2
crl		3
< O : C   Att: S, Pr: { L   await E ; SL }, PrQ: W, Lcnt: F > CN		4
=>		5
< O : C   Att: S, Pr: { L   SL }, PrQ: W, Lcnt: F > CN		6
<b>if</b> evalGuard(E, (S :: L), CN) asBool		7
[label await] .		8
		9

# Example Rewrite Rules

## Introduction

## Dynamic Symbolic Execution

Symbolic Exec

DSE

Concurrency

Creol

A Run

**RW Rule**

Async. Calls

DSE RW Rules

## Application: Testing

---

---	await	1
---		2
cr1		3
	$\langle O : C \mid \text{Att} : S, \text{Pr} : \{ L \mid \text{await } E ; SL \}, \text{PrQ} : W, \text{Lcnt} : F \rangle \text{CN}$	4
	$\Rightarrow$	5
	$\langle \text{log Type} : \text{"await"} \text{ Data} : \{ \text{await } E \mid \dots \mid \text{"eq"} \} \rangle \text{renExpr}(S, L, E) \rangle$	6
	$\langle O : C \mid \text{Att} : S, \text{Pr} : \{ L \mid SL \}, \text{PrQ} : W, \text{Lcnt} : F \rangle \text{CN}$	7
	<b>if</b> evalGuard(E, (S :: L), CN) asBool	8
	[label await] .	9

# Asynchronous Method Call

## Introduction

## Dynamic Symbolic Execution

Symbolic Exec

DSE

Concurrency

Creol

A Run

RW Rule

Async. Calls

DSE RW Rules

## Application: Testing

- Does not wait for a method to return
- Caller and callee can proceed computation concurrently
- Runtime-decision if the return value is received
- Four events mark a method call:
  - ◆  $o_1 \xrightarrow{l} o_2.m(\bar{e})$
  - ◆  $o_1 \xrightarrow{l} o_2.m(\bar{v})$
  - ◆  $o_1 \xleftarrow{l} o_2.m(\bar{e})$
  - ◆  $o_1 \xleftarrow{l} o_2.m(\bar{v})$

# DSE Rewrite Rules

## Introduction

## Dynamic Symbolic Execution

Symbolic Exec

DSE

Concurrency

Creol

A Run

RW Rule

Async. Calls

**DSE RW Rules**

## Application: Testing

$\Theta$  ... Messages

$\sigma$  ... Variable assignments

$\mathcal{C}$  ... Path condition

$$\bar{s}[\Theta, \sigma, \mathcal{C}] \Longrightarrow \bar{s}'[\Theta', \sigma', \mathcal{C}']$$

$$\bar{v} := \bar{e}; \bar{s}[\Theta, \sigma, \mathcal{C}] \Longrightarrow \bar{s}[\Theta, \sigma \uplus \langle \bar{v} \triangleright (\bar{e}\sigma) \rangle, \mathcal{C}] \quad (\text{ASSIGN})$$

$$o_1 \xrightarrow{l} o_2.m(\bar{e}); \bar{s}[\Theta, \sigma, \mathcal{C}] \Longrightarrow \bar{s}[\Theta \uplus \langle l \triangleright \bar{e}\sigma \rangle, \sigma, \mathcal{C}] \quad (\text{CALL})$$

$$o_1 \xrightarrow{l} o_2.m(\bar{v}); \bar{s}[\Theta, \sigma, \mathcal{C}] \Longrightarrow \bar{s}[\Theta, \sigma \uplus \langle \bar{v} \triangleright l\Theta \rangle, \mathcal{C}] \quad (\text{BIND})$$

$$\langle g \rangle; \bar{s}[\Theta, \sigma, \mathcal{C}] \Longrightarrow \bar{s}[\Theta, \sigma, \mathcal{C} \wedge \langle g\sigma \rangle] \quad (\text{COND})$$

# Testing

Introduction

Dynamic Symbolic  
Execution

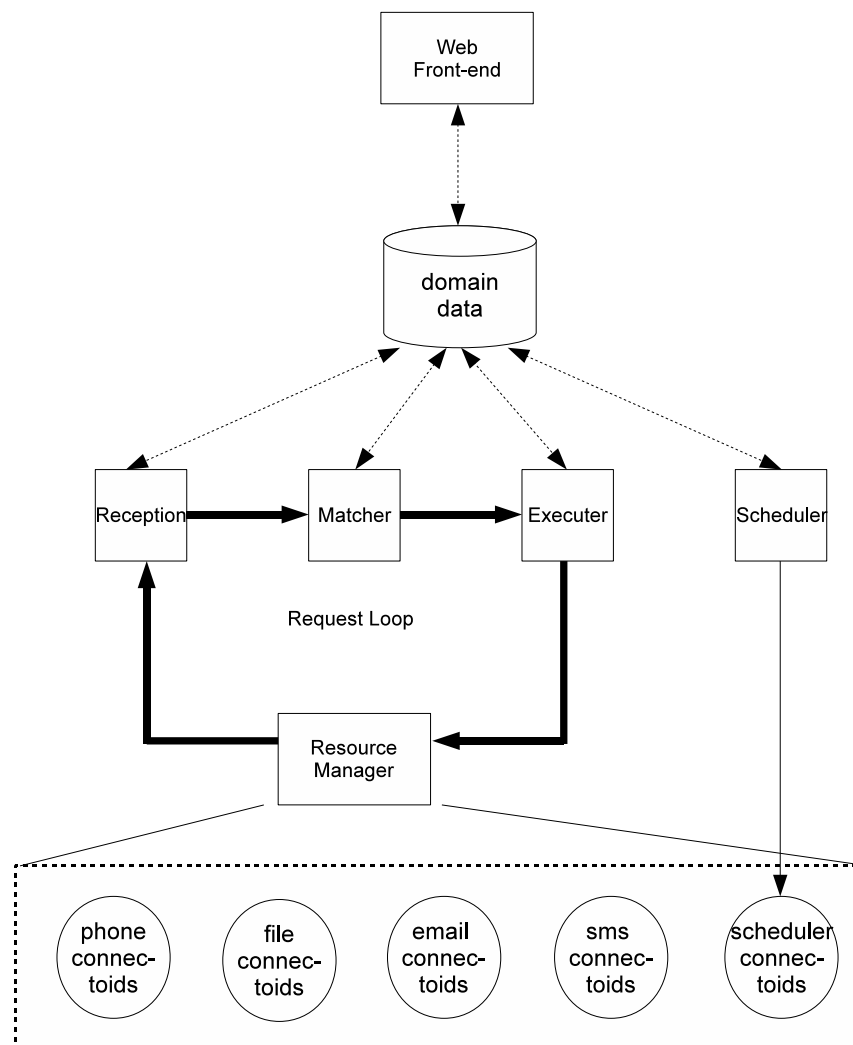
Application: Testing

Testing

Example

- Initiate a test run
- Compute the path condition for the run
  - ◆ path condition represents an equivalence class of input values for that run
- Select new input values for the run
  - Systematic of selection determines the kind of coverage achieved
- Compute new equivalence classes until the full state space is covered

# Example



# Example

Introduction

Dynamic Symbolic  
Execution

Application: Testing

Testing

Example

---

```
op createThreads == 1
  var amountToCreate: Int ; 2
  var idlethreads : Int := threads - busythreads; 3
  await (( threads < maxthreads) 4
        && (( idlethreads - tasks) < (threads / 2))); 5
  amountToCreate := tasks - idlethreads + (threads / 2); 6
  if (amountToCreate > (maxthreads - threads)) then 7
    amountToCreate := maxthreads - threads; 8
  end; 9
  if (amountToCreate > 0) then 10
    await threadpool . createThreads (amountToCreate); 11
  end; 12
  createThreads (); 13
```

# Example

Introduction

Dynamic Symbolic  
Execution

Application: Testing

Testing

Example

```
[maxthreadsS = 0]  
  { "disabled_await": not( 1 < (maxthreadsS + 1) & true) }
```

# Example

Introduction

Dynamic Symbolic  
Execution

Application: Testing

Testing

Example

```
[maxthreadsS = 0]  
  { "disabled_await": not( 1 < (maxthreadsS + 1) & true) }
```

```
[maxthreadsS = 15]  
  { "enabled_await": (1 < (maxthreadsS + 1) & true) }  
  { "ifthenelse": not(10 > maxthreadsS) }
```

# Example

Introduction

Dynamic Symbolic  
Execution

Application: Testing

Testing

Example

```
[maxthreadsS = 0]  
  {"disabled_await": not( 1 < (maxthreadsS + 1) & true) }
```

```
[maxthreadsS = 15]  
  {"enabled_await": (1 < (maxthreadsS + 1) & true) }  
  {"ifthenelse": not(10 > maxthreadsS) }
```

```
[maxthreadsS = 10]  
  {"disabled_await": (1 < (maxthreadsS + 1) & true) }  
  {"ifthenelse": 10 > maxthreadsS }  
  {"ifthenelse": maxthreadsS > 0 }
```

# Thanks for your attention!

Introduction

Dynamic Symbolic  
Execution

Application: Testing

Testing

Example

## Questions ?

Introduction

Dynamic Symbolic  
Execution

Application: Testing

Testing

Example