

# Constructive Computer Architecture: Multirule systems and Concurrent Execution of Rules

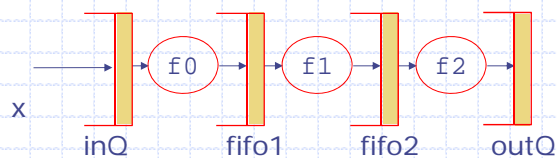
Arvind  
Computer Science & Artificial Intelligence Lab.  
Massachusetts Institute of Technology

September 26, 2016

<http://csg.csail.mit.edu/6.175>

L08-1

## Rewriting Elastic pipeline as a multirule system



```
rule stage1;  
  if(inQ.notEmpty && fifo1.notFull)  
    begin fifo1.enq(f0(inQ.first)); inQ.deq; end endrule  
rule stage2;  
  if(fifo1.notEmpty && fifo2.notFull)  
    begin fifo2.enq(f1(fifo1.first)); fifo1.deq; end endrule  
rule stage3;  
  if(fifo2.notEmpty && outQ.notFull)  
    begin outQ.enq(f2(fifo2.first)); fifo2.deq; end endrule
```

◆ How does such a system function?

September 26, 2016

<http://csg.csail.mit.edu/6.175>

L08-2

## Bluespec Execution Model

Repeatedly:

- ◆ Select a rule to execute
- ◆ Compute the state updates
- ◆ Make the state updates

Highly non-deterministic;  
User annotations  
can be used in  
rule selection

One-rule-at-a-time-semantics: Any legal behavior of a Bluespec program can be explained by observing the state updates obtained by applying only one rule at a time

However, for performance we need to execute multiple rules concurrently if possible

September 26, 2016

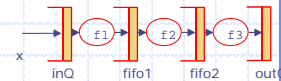
<http://csg.csail.mit.edu/6.175>

L08-3

## Multi-rule versus single rule elastic pipeline

```
rule elasticPipeline;
  if(inQ.notEmpty && fifo1.notFull)
    begin fifo1.enq(f1(inQ.first)); inQ.deq; end
  if(fifo1.notEmpty && fifo2.notFull)
    begin fifo2.enq(f2(fifo1.first)); fifo1.deq; end
  if(fifo2.notEmpty && outQ.notFull)
    begin outQ.enq(f3(fifo2.first)); fifo2.deq; end
endrule

rule stage1;
  if(inQ.notEmpty && fifo1.notFull)
    begin fifo1.enq(f1(inQ.first)); inQ.deq; end endrule
rule stage2;
  if(fifo1.notEmpty && fifo2.notFull)
    begin fifo2.enq(f2(fifo1.first)); fifo1.deq; end endrule
rule stage3;
  if(fifo2.notEmpty && outQ.notFull)
    begin outQ.enq(f3(fifo2.first)); fifo2.deq; end endrule
```



*How are these two systems the same (or different)?*

September 26, 2016

<http://csg.csail.mit.edu/6.175>

L08-4

# Elastic pipeline

- ◆ Do these systems see the same state changes?
  - The single rule system – fills up the pipeline and then processes a message at every pipeline stage for every rule firing – no more than one slot in any fifo would be filled unless the OutQ blocks
  - The multirule system has many more possible states. It can mimic the behavior of one-rule system but one can also execute rules in different orders, e.g., stage1; stage1; stage2; stage1; stage3; stage2; stage3; ... (assuming stage fifos have more than one slot)
- ◆ When can some or all the rules in a multirule system execute concurrently?

## Can these rules execute in parallel? (without violating the one-rule-at-a-time-semantic)

Example 1

```
rule ra;  
  if (z>10)  
    x <= x+1;  
endrule
```

```
rule rb;  
  if (z>20)  
    y <= y+2;  
endrule
```

Example 2

```
rule ra;  
  if (z>10)  
    x <= y+1;  
endrule
```

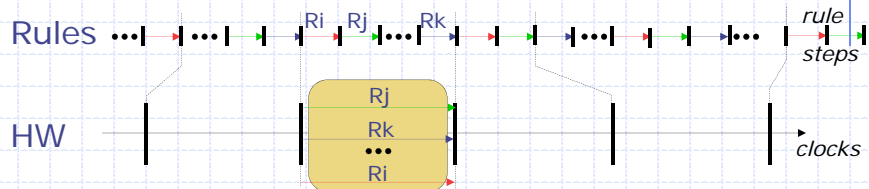
```
rule rb;  
  if (z>20)  
    y <= x+2;  
endrule
```

Example 3

```
rule ra;  
  if (z>10)  
    x <= y+1;  
endrule
```

```
rule rb;  
  if (z>20)  
    y <= y+2;  
endrule
```

## some insight into Concurrent rule execution



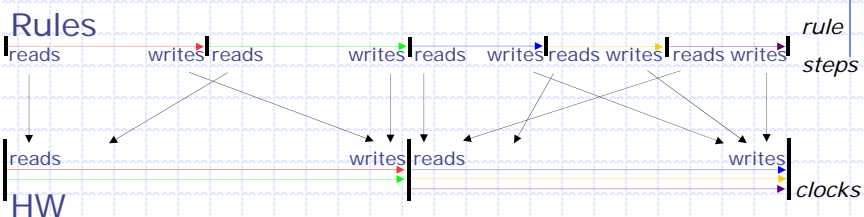
- ◆ There are more intermediate states in the rule semantics (a state after each rule step)
- ◆ In the HW, states change only at clock edges

September 26, 2016

<http://csg.csail.mit.edu/6.175>

L08-7

## Parallel execution reorders reads and writes



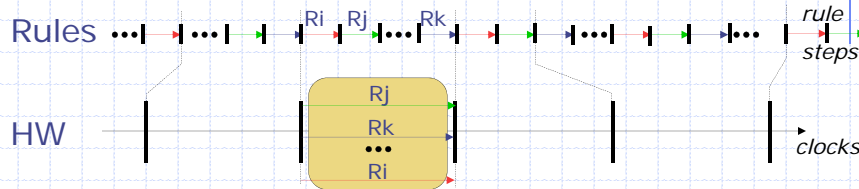
- ◆ In the rule semantics, each rule sees (reads) the effects (writes) of previous rules
- ◆ In the HW, rules only see the effects from previous clocks, and only affect subsequent clocks

September 26, 2016

<http://csg.csail.mit.edu/6.175>

L08-8

## Correctness



- ◆ The compiler will schedule rules concurrently only if the net state change is equivalent to sequential rule execution (which is what our theorem ensures)

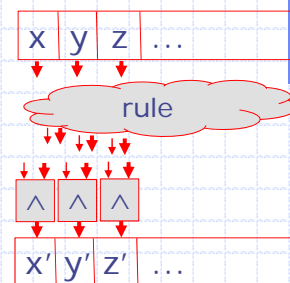
September 26, 2016

<http://csg.csail.mit.edu/6.175>

L08-9

## Evaluating or applying a rule

- ◆ The state of the system  $s$  is defined as the value of all its registers
- ◆ An *expression* is evaluated by computing its value on the current state
- ◆ An *action* defines the next value of some of the state elements based on the current value of the state
- ◆ A *rule* is evaluated by evaluating the corresponding action and simultaneously updating all the affected state elements



Given action  $a$  and state  $S$ , let  $a(S)$  represent the state after the application of action  $a$

September 26, 2016

<http://csg.csail.mit.edu/6.175>

L08-10

## One-rule-at-a-time semantics

- ◆ Given a program with a set of rules  $\{\text{rule } r_i \ a_i\}$  and an initial state  $S_0$ ,  $S$  is a legal state if and only if there exists a sequence of rules  $r_{j_1}, \dots, r_{j_n}$  such that  $S = a_{j_n}(\dots(a_{j_1}(S_0))\dots)$

## Concurrent execution of two rules

- ◆ Concurrent execution of two rules, rule  $r_1 \ a_1$  and rule  $r_2 \ a_2$ , means executing a rule whose body looks like  $(a_1; a_2)$ , that is a rule which is a parallel composition of the actions of the two rules with the following restrictions to preserve the one-rule-at-a-time semantics:
  - Either  $\forall S. (a_1; a_2)(S) = a_2(a_1(S))$   
or  $\forall S. (a_1; a_2)(S) = a_1(a_2(S))$

## Concurrent scheduling of rules

- ◆ rule  $r_1$   $a_1$  to rule  $r_n$   $a_n$  can be scheduled concurrently, preserving one-rule-at-a-time semantics, if and only if there exists a permutation  $(p_1, \dots, p_n)$  of  $(1, \dots, n)$  such that
  - $\forall S. (a_1; \dots; a_n)(S) = a_{p_n}(\dots(a_{p_1}(S)))$

September 26, 2016

<http://csg.csail.mit.edu/6.175>

L08-13

A compiler can determine if two rules can be executed in parallel without violating the one-rule-at-a-time semantics

James Hoe, Ph.D., 2000

Construct a conflict matrix (CM) for rules

September 26, 2016

<http://csg.csail.mit.edu/6.175>

L08-14

## Extending CM to rules

◆ CM between two rules is computed exactly the same way as CM for the methods of a module

◆ Given rule  $r_1$   $a_1$  and rule  $r_2$   $a_2$  such that

$$\text{mcalls}(a_1) = \{g_{11}, g_{12}, \dots, g_{1n}\}$$

$$\text{mcalls}(a_2) = \{g_{21}, g_{22}, \dots, g_{2m}\}$$

◆ Compute

- $\text{Conflict}(x, y)$  = if  $x$  and  $y$  are methods of the same module then  $\text{CM}[x, y]$  else CF
- $\text{CM}[r_1, r_2] = \text{conflict}(g_{11}, g_{21}) \cap \text{conflict}(g_{11}, g_{22}) \cap \dots$   
 $\cap \text{conflict}(g_{12}, g_{21}) \cap \text{conflict}(g_{12}, g_{22}) \cap \dots$   
 $\dots$   
 $\cap \text{conflict}(g_{1n}, g_{21}) \cap \text{conflict}(g_{1n}, g_{22}) \cap \dots$

◆ **Conflict relation is not transitive**

- $r_1 < r_2, r_2 < r_3$  does not imply  $r_1 < r_3$

September 26, 2016

<http://csg.csail.mit.edu/6.175>

L08-15

## Using CMs for concurrent scheduling of rules

Two rules that are conflict free can be scheduled together without violating the one-rule-at-a-time semantics. In general, we use the following theorem

*Theorem:* Given a set of rules  $\{\text{rule } r_i \ a_i\}$ , if there exists a permutation  $\{p_1, p_2, \dots, p_n\}$  of  $\{1..n\}$  such that

$$\forall i < j. \text{CM}(a_{p_i}, a_{p_j}) \text{ is CF or } <$$

then the rules  $r_1, r_2, \dots, r_n$  can be scheduled concurrently with the effect  $\forall i, j. r_{p_i} < r_{p_j}$

September 26, 2016

<http://csg.csail.mit.edu/6.175>

L08-16



## Example 2: Compiler Analysis

```
rule ra;  
  if (z>10)  
    x <= y+1;  
endrule
```

```
rule rb;  
  if (z>20)  
    y <= x+2;  
endrule
```

$mcalls(ra) = \{z.r, x.w, y.r\}$

$mcalls(rb) = \{z.r, y.w, x.r\}$

$CM(ra, rb) =$

$\text{conflict}(z.r, z.r) \cap \text{conflict}(z.r, y.w)$   
 $\cap \text{conflict}(z.r, x.r) \cap \text{conflict}(x.w, z.r)$   
 $\cap \text{conflict}(x.w, y.w) \cap \text{conflict}(x.w, x.r)$   
 $\cap \text{conflict}(y.r, z.r) \cap \text{conflict}(y.r, y.w)$   
 $\cap \text{Conflict}(y.r, x.r)$

## Example 3: Compiler Analysis

```
rule ra;  
  if (z>10)  
    x <= y+1;  
endrule
```

```
rule rb;  
  if (z>20)  
    y <= y+2;  
endrule
```

$mcalls(ra) = \{z.r, x.w, y.r\}$

$mcalls(rb) = \{z.r, y.w, y.r\}$

$CM(ra, rb) =$

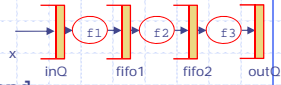
$\text{conflict}(z.r, z.r) \cap \text{conflict}(z.r, y.w)$   
 $\cap \text{conflict}(z.r, y.r) \cap \text{conflict}(x.w, z.r)$   
 $\cap \text{conflict}(x.w, y.w) \cap \text{conflict}(x.w, y.r)$   
 $\cap \text{conflict}(y.r, z.r) \cap \text{conflict}(y.r, y.w)$   
 $\cap \text{Conflict}(y.r, y.r)$

# Multi-rule versus single rule elastic pipeline

```

rule elasticPipeline;
  if(inQ.notEmpty && fifo1.notFull)
    begin fifo1.enq(f1(inQ.first)); inQ.deq; end
  if(fifo1.notEmpty && fifo2.notFull)
    begin fifo2.enq(f2(fifo1.first)); fifo1.deq; end
  if(fifo2.notEmpty && outQ.notFull)
    begin outQ.enq(f3(fifo2.first)); fifo2.deq; end
endrule

```



```

rule stage1;
  if(inQ.notEmpty && fifo1.notFull)
    begin fifo1.enq(f1(inQ.first)); inQ.deq; end endrule
rule stage2;
  if(fifo1.notEmpty && fifo2.notFull)
    begin fifo2.enq(f2(fifo1.first)); fifo1.deq; end endrule
rule stage3;
  if(fifo2.notEmpty && outQ.notFull)
    begin outQ.enq(f3(fifo2.first)); fifo2.deq; end endrule

```

If we do concurrent scheduling in the multirule system then the multi-rule system behaves like the single rule system

# Concurrency when the FIFOs do not permit concurrent enq and deq

