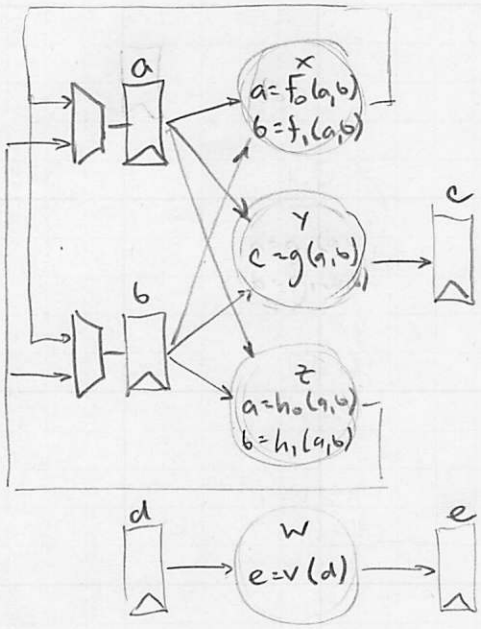
Bluespec Notes



X, Y, Z, W are conceptually different "operations" we want to perform on the state elements

Whether or not we want to actually "do" X, Y, Z, W depends on various other conditions

All four operations might be enabled at the same time

Assume Y, W <u>always</u> enabled

RTL Modeling

```
assign a = (x') ? f0(a,b)
         : (z') ? h0(a,b);

assign b = (x') ? f1(a,b)
           (z') ? h1(a,b);

assign c = g(a,b)

assign e = v(d)
```

STATE CENTRIC

Must group logic by the state it updates instead of by operation

Execution semantics: explicitly clocked, must explicitly manage moving between different operations

Guarded Atomic Action Modeling

```
rule rX (x'):
   a = f0(a,b);
   b = f1(a,b);
endrule

rule rZ (z'):
   a = h0(a,b);
   b = h1(a,b);
endrule

rule rY:
   c = g(a,b);
endrule

rule rW:
   e = v(d);
endrule
```

guard

Fine to write same state from different rules

Operation Centric

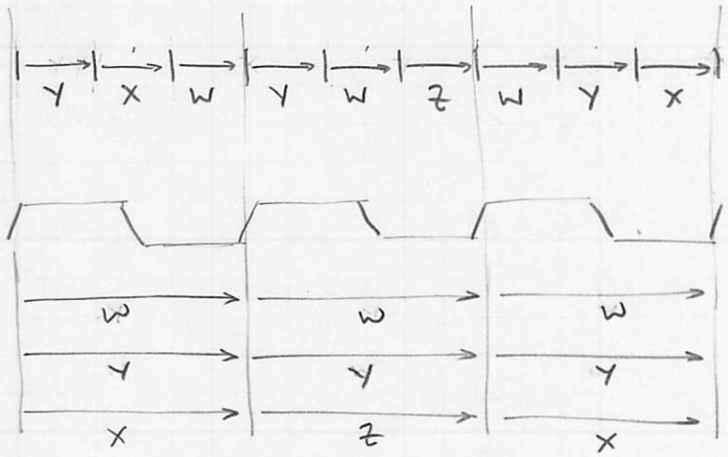
group logic by operation, moving handles by compiler

No guard, always enabled

Execution semantics: untimed, pick one enabled rule, fire the rule, pick again

Rules fire atomically and in isolation

Semantics:



Y can always fire W since it is conflict free

only X or Z can fire every cycle because they are mutually exclusive

Y can always fire in parallel, but it always appears to happen before X or Z in the serial execution semantics

WHAT ARE POSSIBLE VALUES FOR X, Y AFTER AN EXECUTION?

```
rule rA;
  x <= y;
endrule

rule rB;
  y <= x;
endrule
```

ASSUME INITIALLY $x = 1$, $y = 2$

---

rA < rB	:	$x = 2$, $y = 2$
rB < rA	:	$x = 1$, $y = 1$
rA	:	$x = 2$, $y = 2$
rB	:	$x = 1$, $y = 1$
ϕ	:	$x = 1$, $y = 2$
rA < rA < rB	:	$x = 2$, $y = 2$

For each set of two rules explain if these rules can be enabled at the same time, and if so can they be executed concurrently?

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

rule rb (z > 20);
  y <= y + 1;
endrule
```

enables? yes
concurrently? yes
ra < rb
rb < ra

Conflict Free

2)
```
rule rA (z > 10);
  x <= y+1;
endrule

rule rb (z > 20);
  y <= x + 1;
endrule
```

enables? yes
concurrently? NO

Conflict

3)
```
rule rA (z > 10);
  x <= y + z;
endrule

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

enables? yes
concurrently? yes
ra < rb

SEQUENTIALLY COMPOSABLE

4)
```
rule rA (z > 10);
  x <= y+1
endrule

rule rB (z > 20);
  x <= y + 2
```

enables? YES
concurrently? YES

Write-After-Write

Depending on update logic it will appear as if rA < rB or rB < rA