

Behavioral description with message-passing

Rajit Manohar



Communicating Hardware Processes

- Behavioral language
 - ❖ CHP = Communicating hardware processes
 - ❖ Based on Tony Hoare's CSP (Communicating Sequential Processes) language
- Simplified programming language
- Assignment-based language, but...
 - ❖ No memory allocation ("new", "malloc", etc.)
 - ❖ No memory references ("pointers", "references", etc.)
- Basic data types: Booleans and unsigned integers

```
bool x;  
  
int y;  
  
int<8> z;
```

Basic language constructs

- Simple statements

- ❖ **skip** statement that does nothing!

- ❖ **x := E** assignment statement

- ▶ Evaluate expression on the right-hand side, assign it to the variable on the left-hand side: just like a standard software programming language.

```
b := b & w | ~c
```

```
x := y*3 + 5
```

```
b+
```

```
w-
```

```
x := y{3..2} + 7
```

```
x := {p,q{3..2}}
```

```
x := (a > 0 ? b : myf(c,d))
```

- ❖ Sequencing: **S₁; S₂**

```
t := x; x := y; skip; y := t
```

<https://avlsi.csl.yale.edu/act/doku.php?id=language:expressions>

Arrays

- In hardware, an array results in an address-calculation mechanism

```
x[i] := x[i] + 1
```

- Array access is of two kinds
 - ❖ Standard array, where array index requires run-time information, or
 - ❖ Array index is a run-time constant

```
x[0] := x[0] + 1
```

- Only use standard arrays when absolutely necessary!

Example ACT CHP program

Process definition

Type definitions for
variables

Sublanguage body

```
defproc mytest()  
{  
  
    int<8> z;  
    int x[3]; // C++ style comments  
              // implicit: 32 bits  
  
    chp {  
        z := 0;  
        x[1] := z + 3;  
        z := x[1] * 2  
    }  
}
```

Note: in CHP, semi-colon is used as a separator (no trailing semi-colon)

Conditional execution via *selection* statements

- Selections : generalized if-statements

```
[  x > 10 -> y := 3
[ ] x < 10 -> y := 4
]
```

- ❖ If some condition (guard) is true, execute corresponding statement
- ❖ If all guards are false, then **wait**
- ❖ **If multiple guards are true, error!**

- To allow multiple true options, use non-deterministic selection

```
[ | x > 10 -> y := 3
[ ] x < 10 -> y := 4
[ ] x > 8  -> y := 7
| ]
```

The “chp-txt” sublanguage: text version of CHP

- Selections : generalized if-statements

```
[  x > 10 -> y := 3
[ ] x < 10 -> y := 4
]
```

- ❖ If some condition (guard) is true, execute corresponding statement
- ❖ If all guards are false, then **wait**
- ❖ **If multiple guards are true, error!**

- To allow multiple true options, use non-deterministic selection

```
[ | x > 10 -> y := 3
[ ] x < 10 -> y := 4
[ ] x > 8  -> y := 7
| ]
```

chp-txt equivalent

```
select {
  case x > 10: y := 3;
  case x < 10: y := 4
}
```

chp-txt equivalent

```
arb_select {
  case x > 10: y := 3;
  case x < 10: y := 4;
  case x > 8:  y := 7
}
```

Loops

- While loop

```
i := 0; j := 0;  
*[ i < 10 ->  
    j := j + i;  
    i := i + 1  
]
```

chp-txt equivalent

```
i := 0; j := 0;  
while (i < 10) {  
    j := j + i;  
    i := i + 1  
}
```

- Generalized deterministic loop

```
*[  x > 10 -> y := 3; x := x - 1  
   [ ] x < 10 -> y := 4; x := x + 1  
   ]
```

chp-txt equivalent

```
while {  
    case x > 10 : y := 3;  
                  x := x - 1;  
    case x < 10 : y := 4;  
                  x := x + 1  
}
```

- ❖ If some condition (guard) is true, execute corresponding block and then go back to the beginning of the loop
- ❖ All guards false: exit
- ❖ More than one true guard: error

More language constructs

- Internal parallelism: S_1, S_2

```
*[  x > 10 -> y := 3, x := x - 1
   [ ] x < 10 -> y := 4, x := x + 1
   ]
```

- Common short-hand

- ❖ Infinite loop

```
*[ true -> STMTS ]
```

```
*[ STMTS ]
```

chp-txt equivalent

```
forever {
    STMTS
}
```

- ❖ Wait for some condition

```
[ COND -> skip ]
```

```
[ COND ]
```

chp-txt equivalent

```
wait-for (COND)
```

Communication with other processes

- Hardware modules exchange information via **communication channels**
- Channel
 - ❖ single-sender, single-receiver
 - ❖ a matching send and receive behaves as a distributed assignment

$x!e$

*Evaluate “e” and
send it on output port X*

send(X, e) **chp-txt**

$y?x$

*Receive value from
input port Y and
assign it to variable “x”*

chp-txt **recv(Y, x)**

- ❖ If these two ports are connected, then this has the net effect of

$x := e$

- Channels are **blocking**: a send waits for matching receive, and a receive waits for a matching send.

Overall hardware description

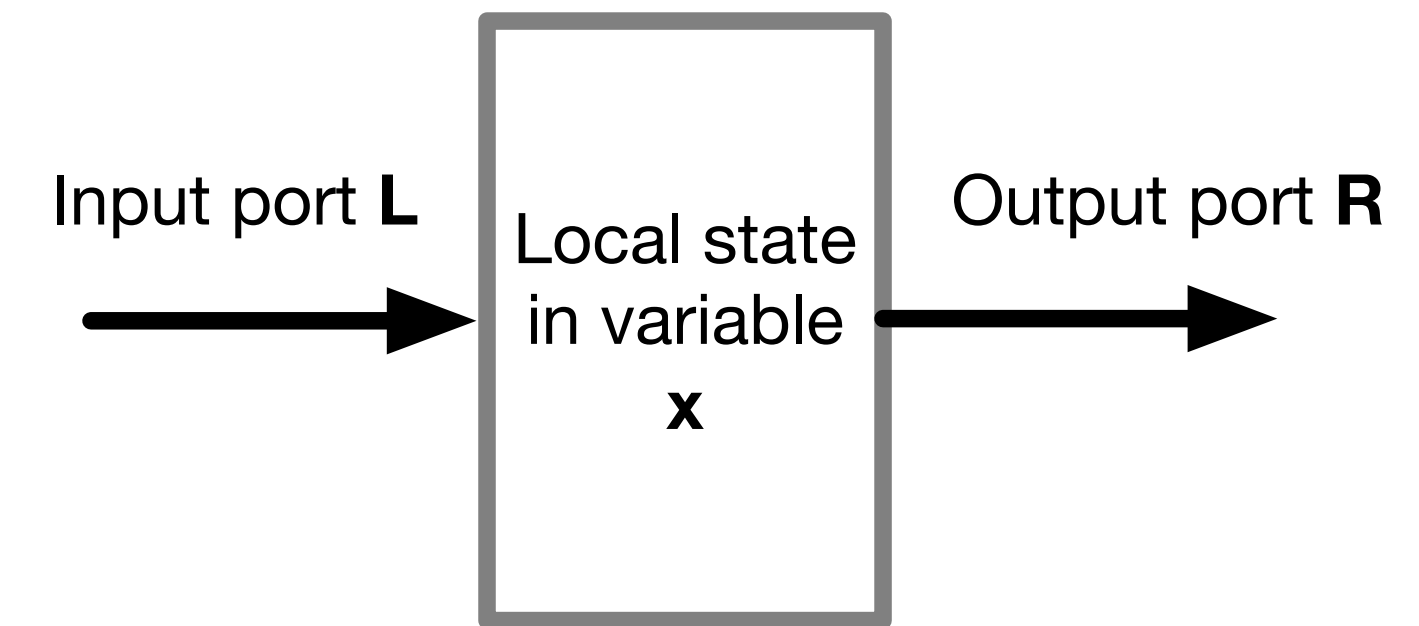
- A *parallel* collection of communicating hardware processes
 - ❖ By default, no shared state
- Connections between processes via channels to exchange information
 - ❖ (General shared variables possible; ignoring for this summer school!)
- For this summer school, syntax for connections, type declarations, etc. in the **ACT** language
 - ❖ There are other examples of CSP-like languages (e.g. Occam)

Example: one-place buffer

- One-place buffer, initially empty
 - ❖ Empty state
 - ▶ Only operation that is valid: read next input
`L?x`
 - ▶ Final state: full
 - ❖ Full state
 - ▶ Only operation that is valid: send value on output
`R!x`
 - ▶ Final state: empty
 - ❖ Empty state to empty state:
`L?x; R!x`
- Buffer repeats this forever: `*[ L?x; R!x ]`

```
forever {  
  recv (L,x);  
  send (R,x)  
}
```

chp-txt

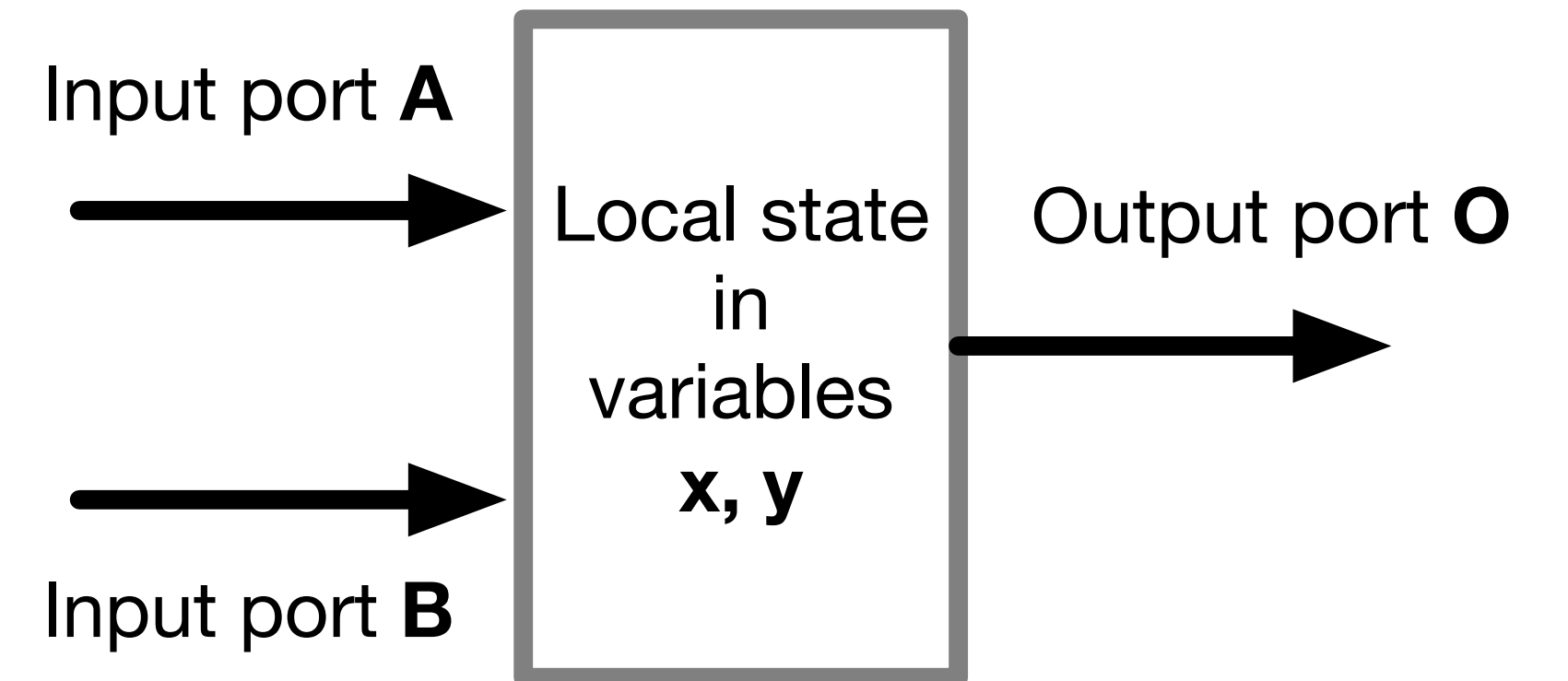


```
defproc buffer(chan?(int) L; chan!(int) R)  
{  
  int x; // local state  
  
  chp {  
    *[ L?x; R!x ]  
  }  
}
```

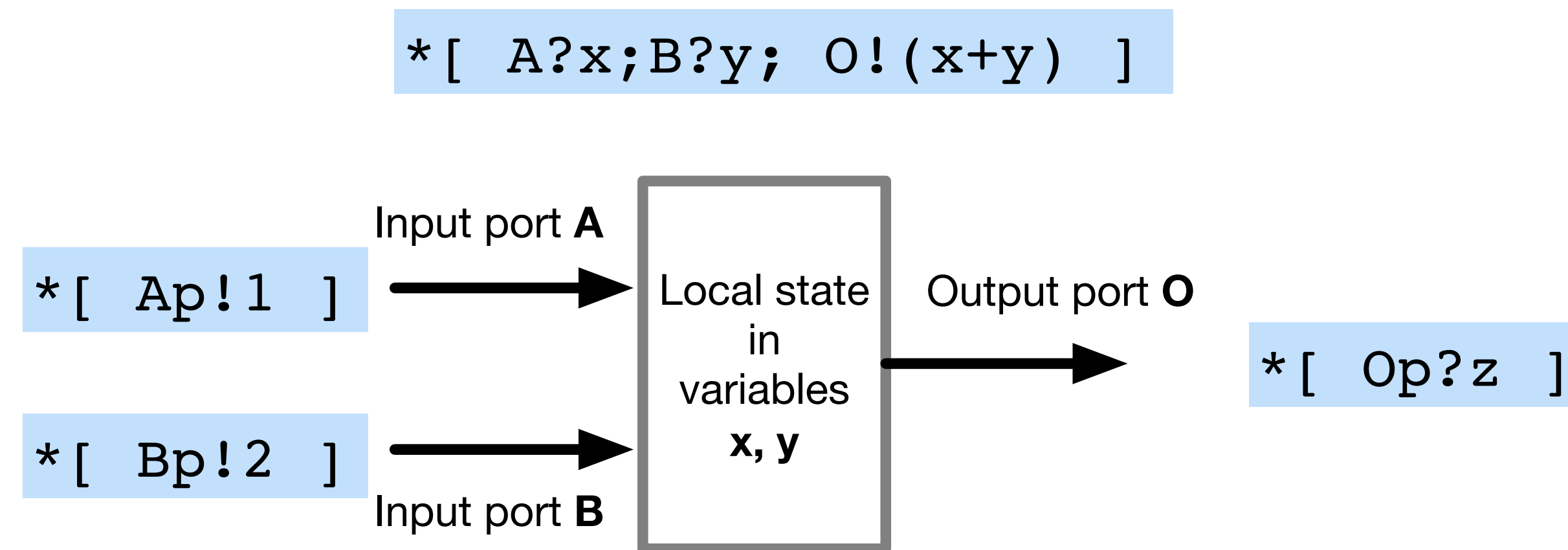
Example: two input, one output process

```
defproc adder(chan?(int) A,B; chan!(int) O)
{
  int x,y;  // local state

  chp {
    *[ A?x, B?y;
      O!(x+y)
    ]
  }
}
```



Synchronization operations are part of message-passing



- Communication actions **synchronize** different parallel processes
 - ❖ Knowing where one process is in its local program can give you information about what other processes in the system are doing.

Non-determinism induced by the environment

- Problem: two input ports **A** and **B** and one output **Z**
 - ❖ Receive the “next input” from either **A** or **B**
 - ❖ Send this value on the output **Z**
- We need some new syntax!
 - ❖ Probe: “is there a communication pending on this port?”

```
*[ [ | #A -> A?x  
   [ ] #B -> B?x  
   | ];  
   Z!x  
]
```

- Use with care, and only when absolutely necessary