

# Communicating Sequential Processes (CSP)

Ali Saoud

## Introduction

- Traditional stored program digital computer has been designed primarily for deterministic execution of a single sequential program.
- Desire for greater speed has led to the introduction of parallelism.
- Parallel computing => communication, synchronization, reliability, expense.

## Introduction

### Solution:

- Guarded command: sequential, non-determinism control.
- Parallel commands start simultaneously with consistent sequential commands.
- Simple input and output commands.
- Sender and receiver name each other.
- Input commands may appear in guards.
- Repetitive commands may have input commands.

## Characteristics

- Single thread of control
- Autonomous
- Static
- Synchronous
- Reliable
- Point –to Point
- Unidirectional

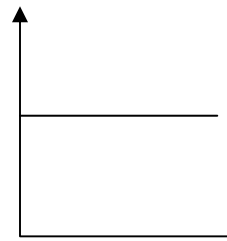
## Commands

- **Structured Command:** Succeeds if all constituent commands succeed.  $\langle SC \rangle ::= \langle PC \rangle | \langle AC \rangle | \langle RC \rangle$
- **Null command:** Never fails.
- **Command list:** Sequential commands.
- **Parallel Commands:** Disjoint, concurrent process execution.  $\langle PC \rangle ::= [ \langle process \rangle \{ || \langle process \rangle \} ]$
- **Assignment Command:**  $insert(n) := has(n+1)$ . Fail.
- **Input/Output command:** Send:  $B!x$ . Receive:  $A?y$ . Blocks if not ready. Variables must match.

# Guarded Commands

## Guarded Commands

**<guard> → <command list>**



**boolean expression**

**at most one ? , must be at end of  
guard, considered true iff  
message pending**

## Examples

**$n < 10 \rightarrow A!index(n); n := n + 1;$**

**$n < 10; A?index(n) \rightarrow next = MyArray(n);$**

**$(i:l..n)G \rightarrow CL$  stands for**

**$G1 \rightarrow CL1[] G2 \rightarrow CL2[[]...[] Gn \rightarrow CLn$**

## Alternative/Repetitive Commands

### *Alternative Command*

$$[ G_1 \xrightarrow{\quad} S_1 [] G_2 \xrightarrow{\quad} S_2 [] \dots [] G_n \xrightarrow{\quad} S_n ]$$

1. evaluate all guards
2. if more than one guard is true, nondeterministically select one.
3. if no guard is true, terminate.

**Note:** if all true guards end with an input command for which there is no pending message, then delay the evaluation until a message arrives. If all senders have terminated, then the alternative command terminates.

### *Repetitive Command*

$$* [ G_1 \xrightarrow{\quad} S_1 [] G_2 \xrightarrow{\quad} S_2 [] \dots [] G_n \xrightarrow{\quad} S_n ]$$

repeatedly execute the alternative command until it terminates

### *Examples:*

$$[ x \geq y \text{ --> } m := x [] y \geq x \text{ --> } m := y ]$$

$$*[ c: \text{character}; \text{west?}c \text{ --> } \text{east!}c ]$$

## Coroutines

- Coroutines are fundamental program structures.
- Copy:  $X::*[c:\text{character}; \text{west?}c \rightarrow \text{east!}c]$
- Squash: Substitute Character in a message
- Disassemble: from card file to  $X \Rightarrow$  extra space at the end of card must be added.
- Assemble: To print from  $X$  125 char/line and complete with spaces.
- Reformat: Assemble and disassemble



## Subroutines

- A corountine acting as a subroutine=>executed concurrently with user process

- Function: Division with remainder.

```
[Div::*[x,y:integer; X?(x,y)→quot, rem: integer;
  quote=0;rem:=x; *[ rem>= y → rem= rem-y;
  quot:= quot+1 ];X!(quot,rem)]|| X::USER
```

- Recursion: Factorial
- Data Representation: small set of integers

## Monitors and scheduling

### ■ Dining philosophers:

```
Phil=*[...during ith lifetime... → Think;  
      room!enter;  
      fork(i)!pickup();fork((i+1)mod 5)!pickup();  
      EAT;  
      fork(i)!pickup();fork((i+1)mod 5)!pickup();  
      Room!exit()]
```

Parallel components:

```
[room::ROOM||fork(i:0..4)::FORK||phil(i:0..4)::  
PHIL]
```