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. <SC>::=<PC>|<AC>|<RC>
- Null command: Never fails.
- Command list: Sequential commands.
- Parallel Commands: Disjoint, concurrent process execution.<PC>::=[<process>{||<process>}]
- 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 → A!index(n); n := n + 1;
n < 10; A?index(n) → next = MyArray(n);
(i:l..n)G → CL stands for
G1 →CLI[]G2 → CL2[]...[]Gn → CLn

Alternative/Repetitive Commands

Alternative Command

[$G_1 \rightarrow S_1$ [] $G_2 \rightarrow S_2$ [] … [] $G_n \rightarrow S_n$]

1. evaluate <u>all</u> guards
2. if more than on guard is true, <u>nondeterministically</u> select one.
3. if no guard is true, <u>terminate</u>.

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

* [$G_1 \rightarrow S_1$ [] $G_2 \rightarrow S_2$ [] … [] $G_n \rightarrow S_n$]

repeatedly execute the alternative command until it terminates

Examples:

```
[x >= y --> m := x [] y >= x --> m := y ]
*[ c: character; west?c --> east!c ]
```

Coroutines

- Coroutines are fundamental program structures.
- Copy: X::*[c:character; west?c→east!c]
- Squash: Substitute Character in a messege
- Disassemble: from card file to X=> 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:= quote+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]