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. $<\text{SC}> ::= <\text{PC}> | <\text{AC}> | <\text{RC}>$
- Null command: Never fails.
- Command list: Sequential commands.
- Parallel Commands: Disjoint, concurrent process execution. $<\text{PC}> ::= [<\text{process}>] | [||<\text{process}>]$  
- Assignment Command: insert(n):=has(n+1). Fail.
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

\[ [ G_1 S_1 [] G_2 S_2 [] ... [] G_n S_n ] \]

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

\[ * [ G_1 S_1 [] G_2 S_2 [] ... [] G_n S_n ] \]

repeatedly execute the alternative command until it terminates

Examples:

\[ [ x \geq y \rightarrow m := x [] y \geq x \rightarrow m := y ] \]
*[ c : character; west?c \rightarrow east!c ]
Coroutines

- Coroutines are fundamental program structures.
- Copy: \[ X::*[^c:character; \text{west?}c \rightarrow \text{east!}c] \]
- 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 coroutine acting as a subroutine=>executed concurrently with user process
- Function: Division with remainder.

\[
\text{Div}:: *[x,y:\text{integer}; X?(x,y) \rightarrow \text{quot}, \text{rem}: \text{integer}; \\
\text{quot}=0; \text{rem}=x; *[\text{rem} \geq y \rightarrow \text{rem} = \text{rem}-y; \$$
\text{quot}:= \text{quot}+1]; X!(\text{quot}, \text{rem})] \] || 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]