Communicating Parallel Processes

Stephen Brookes

Carnegie Mellon University

Deconstructing CSP

CSP

- sequential *processes*
- *input* and *output* as primitives
- named *parallel composition*
- *synchronized* communication
- generalized *guarded commands*

Communicating Sequential Processes,
CACM, 1978

Parallel composition

$$[\pi_1 :: P_1 \| \cdots \| \pi_n :: P_n]$$

- disjoint processes P_i
- distinct process names π_i
- $\pi_j?x$ in P_i matches $\pi_i!e$ in P_j

Guarded commands

$$\textbf{if } (g_1 \to P_1) \Box \cdots \Box (g_k \to P_k) \textbf{ fi}$$
$$\textbf{do } (g_1 \to P_1) \Box \cdots \Box (g_k \to P_k) \textbf{ od}$$

- input guards: $b \wedge \pi?x$
- $b \wedge \pi_j?x$ in π_i is true when b holds and π_j is at a matching output
- input-guarded conditional waits for match

Contrasts

- shared-memory (Dijkstra 68)
  - "cooperating sequential processes"
  - no input-output
  - global state
  - conditional critical regions

- dataflow networks (Kahn/MacQueen 77)
  - asynchronous communication
  - deterministic, CSP-like syntax
  - fair execution

Despite common roots, these paradigms have grown apart from CSP

Design issues

- Communication vs. assignment
- Limited parallelism
- Naming and scope
- Synchronous vs. asynchronous
- Fairness
- Extending CSP
  - procedures
  - objects

Communication vs. assignment

- Regarded as "independent concepts"
  - communication affects *environment*
  - assignment affects *local state*

- Semantic models reflect this separation
  - traces, refusals, failures
  - local state change

Nevertheless it can be advantageous to blur the distinction...

Limited parallelism

- No nesting of parallel constructs
  - static network topology
  - need n-ary parallel operators

Naming and scope

- Explicit process names cause *library problem*
- Awkward scope rules:

  $$[\pi_1::[\sigma_1::P_{11} \parallel \sigma_2::P_{12}] \parallel \pi_2::P_2]$$

  How do π_1 and π_2 communicate?
  Hard to formulate *associativity* law

Solution

- Plotkin: decouple *name binding* 1983
- Hoare: *named channels*,
  hiding operator *TCSP, occam*

TCSP

- Process algebra based on CSP

  (*cf. Milner's CCS, 1982)*

- Channel-based events

  $h?v$ input

  $h!v$ output

- Processes, alphabets

  $P \,\Box\, Q$ external choice

  $P \sqcap Q$ internal choice

  $a \rightarrow P$ prefix

  $P \,\|\, Q$ parallel composition

  $P \backslash a$ hiding

- Algebraic laws

$$(P \sqcap Q) \| R = (P \| R) \sqcap (Q \| R)$$

A theory of communicating sequential processes,

HBR 81, J. ACM, 1984

Synchrony vs. asynchrony

- *Synchronous* communication
  - input or output waits until a match is ready

- *Asynchronous* communication
  - output always proceeds
  - input only waits if no output available

CSP assumed synchrony

"Equally reasonable to assume asynchrony"

Why synchrony?

- "easy to implement"... (?)
- "simple" models

Why not asynchrony?

- "less realistic to implement"
- "can readily be specified"

Reassessment

- asynchrony is *easier* to implement
- asynchrony is *more realistic*
- synchronization can readily be specified
- simplicity is misleading
  - ignores fairness
  - hard to generalize

Fairness

- Want to abstract away from scheduler
- Assume that processes are executed without unreasonable delay
- Pragmatically important
  - "every reasonable scheduler is fair"
  - vital for liveness properties
- Semantically awkward
  - hard to model with *powerdomains*
- A *plethora* of fairness notions
  - (strong, weak) process fairness
  - (strong, weak) channel fairness
  - unconditional Γ-extreme fairness...

Strong vs. weak

A scheduler is *strongly fair* if every process enabled *infinitely often* will run eventually

A scheduler is *weakly fair* if every process enabled *persistently* will run eventually

- For shared-variable programs *weak fairness* is reasonable (Park, 80)
- For synchronous processes *weak fairness* is reasonable but not very useful
  - enabledness is not local
- *Strong fairness* is not realistic
  - too much book-keeping (Older, 96)
- For asynchronous processes *weak fairness* is both reasonable and useful

Example

$$
\begin{array}{l}
a!0 \parallel\ n{:=}0;\ \ go{:=}\textbf{true};\\
\qquad\quad \textbf{do}\\
\qquad\qquad (go \wedge a?x \rightarrow go{:=}\textbf{false})\\
\qquad\quad \Box\ \ (go \qquad\ \ \rightarrow n{:=}n+1)\\
\qquad\quad \textbf{od}
\end{array}
$$

"It would be *unfair* to keep executing the second alternative. . . , since this would keep ignoring the potential for synchronized communication between the two processes, which could have been performed on an infinite number of occasions."

"An efficient implementation should try to be *reasonably fair*. . . and should ensure that an output command is not delayed unreasonably often after it first becomes executable."

What's fair?

- Such an execution is *not strongly fair*, but *weakly fair*.
- Assuming strong fairness the program terminates with $n \geq 0$.
- Assuming weak fairness the program may also diverge.

CSP isn't fair

"Should a programming language definition specify that an implementation must be fair?"

i.e. strongly fair

Hoare was "fairly sure that the answer is NO":

- unbounded non-determinism
- strong fairness isn't realistic

But we shouldn't dismiss weak fairness.

Deadlock

$$P = \textbf{if } (\textbf{true} \to a?x) \Box (\textbf{true} \to b?x) \textbf{ fi}$$

$$Q = \textbf{if } (a?x \to \textbf{skip}) \Box (b?x \to \textbf{skip}) \textbf{ fi}$$

- $P \backslash a$ may deadlock, $Q \backslash a$ cannot

Divergence

$$R = [\ \textbf{do } (a?x \to \textbf{skip}) \textbf{ od} \\ \| \ \textbf{do } (\textbf{true} \to a!0) \textbf{ od}] \backslash a$$

$$R' = [\ \textbf{do } (a?x \to \textbf{skip}) \Box (in?y \to out!y) \textbf{ od} \\ \| \ \textbf{do } (\textbf{true} \to a!0) \textbf{ od}] \backslash a$$

- R *must* diverge, R' *may* diverge

Models of TCSP

- **traces** (H, 1980)
  - communication sequences
  - $P = Q$
- **failures** (HBR, 1981)
  - traces, refusals
  - $P \supseteq Q$
- **failures/divergences** (BR, 1982)
  - traces, refusals, divergences
  - "divergence is catastrophic"
  - $R = R', P \supseteq Q$
- **stable failures** (Jategaonkar/Meyer/Roscoe/Valmari)
  - traces, stable refusals
  - "ignore divergence"
  - $R \subseteq R', P \supseteq Q$

Features

- Close ties to operational intuition
  - describe *partial* behaviors
  - natural closure conditions
  - full abstraction results
  - ordering based on non-determinism
- Algebraic laws
  - basis for model checking
- Reasoning principles
  - unique fixed-point
  - Milner-style expansion

Limitations

- Fairness hard to incorporate
- Extreme view of divergence

Idealized CSP

- *parallel* imperative processes
- *asynchronous* communication
- *weakly fair* parallel composition
- *local variable* declarations
- *recursive* processes

Unification

- asynchronous CSP
- shared-variable
- Kahn-style dataflow

Generalization

- CSP + procedures
- Idealized Algol + processes
- non-deterministic Kahn networks

Semantic model

Transition traces

$$\langle s_0, s_0' \rangle \langle s_1, s_1' \rangle \ldots \langle s_n, s_n' \rangle \ldots$$

- fair interactive computation
- process changes *vs.* environment changes

Communication

- Channel = queue-valued "variable"
- Input and output = state change
- Waiting = stuttering

Process

- set of traces
- closed under *stuttering* and *mumbling*
- trace sets ordered by inclusion

Semantics

- $P \supseteq Q, R \neq R'$
- Sequential composition = concatenation
- Parallel composition = fair merge
- Recursion = greatest fixed-point
- Environment never changes local variables

Advantages

- Mathematically simple
  - traces suffice...
  - models deadlock and divergence
  - fairness built-in
- Unification of paradigms
  - one model fits all

Features

- Close ties to operational intuition
  - describes *complete* behaviors
  - safety and liveness properties
  - full abstraction at ground types
- Algebraic laws
  - locality properties
  - fairness
- Reasoning principles
  - fixed-point
  - *fair expansion*

    $\Sigma A_i; P_i \parallel \Sigma B_j; Q_j = \Sigma(A_i \| B_j); (P_i \| Q_j)$

No unique fixed-point property, but...

Fixed-point theorems

Parallel recursion

- If $p_i = \sum_j A_{ij}; p_j$ are fair expansions then

$$p_{ij} = \sum_{i',j'} (A_{ii'} \| A_{jj'}); p_{i'j'}$$

are valid and the fixed-points satisfy

$$p_{ij} \;=\; p_i \| p_j$$

Local recursion

- If $p_i = \sum_j A_{ij}; p_j$ are fair expansions and

$$\vdash \{x = v_i\} A_{ij} \{x = v_j\}$$
$$A'_{ij} =_{\text{def}} \textbf{local } x = v_i \textbf{ in } A_{ij}$$

then $q_i = \sum_j A'_{ij}; q_j$ are valid and the fixed-points satisfy

$$q_i \;=\; \textbf{local } x = v_i \textbf{ in } p_i$$

Laws

Local input

$$\begin{aligned}&\textbf{local } h = v\rho \textbf{ in } P \| (h?x; Q)\\ &\quad = \textbf{ local } h = \rho \textbf{ in } P \| (x{:=}v; Q)\\ &\qquad\qquad \text{if } h? \text{ not free in } P\end{aligned}$$

Local output

$$\begin{aligned}&\textbf{local } h = \rho \textbf{ in } P \| (h!v; Q)\\ &\quad = \textbf{ local } h = \rho v \textbf{ in } P \| Q\\ &\qquad\qquad \text{if } h! \text{ not free in } P\end{aligned}$$

Fair promotion

$$\begin{aligned}&\textbf{local } h = \epsilon \textbf{ in } (h?x; P) \| (Q_1; Q_2)\\ &\quad = Q_1; \textbf{local } h = \epsilon \textbf{ in } (h?x; P) \| Q_2\\ &\qquad\qquad \text{if } h \text{ not free in } Q_1\end{aligned}$$

Desiderata

- develop practical tools
  - model checking
  - theorem-proving
- exploit unification
  - reasoning across paradigms
- explore further
  - probabilistic processes
  - real-time
  - concurrent objects
- stay faithful to ideals
  - simplicity
  - elegance

References

Full abstraction for a shared-variable parallel language

LICS'93

The essence of Parallel Algol

LICS'96

Idealized CSP: combining procedures with communicating processes

MFPS'97

On the Kahn Principle and fair networks

MFPS'98

Reasoning about recursive processes: Expansion isn't always fair

MFPS'99