Retracing CSP

Stephen Brookes
Carnegie Mellon University
Outline

- Original CSP
- Theoretical CSP
- Traditional models
- Limitations and omissions
- Unification and generalization
Original CSP

... a programming language

- guarded commands
- input and output
- parallel composition
- synchronized communication between named processes
- no shared variables

Hoare 1978
influenced by Dijkstra
Theoretical CSP

... a process algebra

- internal and external choice
- input and output
- parallel composition
- synchronized communication on named channels

Hoare, Brookes, Roscoe 1984
influenced by Milner
Traditional models

- communication traces
  - Hoare 1980
- failures
  - Hoare, Brookes, Roscoe 1984
- failures/divergences
  - Brookes, Roscoe 1985

... all denotational
trace = input/output sequence
process = set of traces
prefix-closed, ordered by inclusion

good for safety properties
Failures

- failure = trace + refusal
- refusal = input/output set
- process = set of failures
  ordered by reverse inclusion

Good for
- safety properties
  + deadlock
Failures/divergences

divergence = trace
viewed as catastrophic
process = failures + divergences
ordered by reverse inclusion

good for
safety, deadlock, divergence
Examples

if (true → a?x; c!x) □ (true → b?x; c!x) fi

if (a?x → c!x) □ (b?x → c!x) fi

same traces
different failures
Examples

if (a?x → c!x) fi

if (a?x → c!x) ☐ (true → stop) fi

same traces

different failures
Example

infinite internal chatter

chan a in
  do (true → a?x) od || do (true → a!0) od

no finite failures
divergence
communication traces

cannot model deadlock or divergence failures

cannot model divergence failures/divergences

allows compositional reasoning

basis for FDR model checker
Limitations

- Lack of fairness
- Less suitable for liveness analysis
- Hard to extend
- Traces + refusals + divergences + ???
- Catastrophic divergence
- Not the only choice
- Models are specialized
- Not applicable to other paradigms
We need a common semantic framework:

- Shared-memory
- Synchronized i/o
- Asynchronous i/o

Traditional models are incompatible...
Action traces

... a unifying theme

Trace = sequence of actions
Actions have effect
input, output, waiting, ...
read, write, ...
Process = set of action traces
ordered by inclusion
Design features

- Sets of complete traces
  - finite and infinite
  - not prefix-closed
- Fairness
  - only include fair traces
- Robustness
  - race condition = catastrophe

*cf. Reynolds*
CPP

Communicating Parallel Processes

- Imperative programs
  - local state
  - shared state, including channels
- Synchronization
  - conditional critical regions, semaphores
  - input and output

... a natural successor of CSP
Actions

Communication
- $h?v$, $h!v$, $h.v$, $\delta_D$ (D is a set of directions)

Reading and writing
- $x=v$, $x:=v$

Resource management
- $\text{try}(r)$, $\text{acq}(r)$, $\text{rel}(r)$

Runtime error
- $\text{abort}$
Semantics

Process denotes a set of action traces

\[[h!0]\] = \(\delta_{\{h!\}}^{\infty} \{h!0\}\)

\[[h?x]\] = \(\delta_{\{h?\}}^{\infty} \{h?v \ x:=v \mid v \in V_{int}\}\)

\[[c_1 || c_2]\] = \([[c_1]\] \emptyset || \emptyset [[c_2]\]

\[[\text{with } r \text{ do } c]\] = \text{wait}^{\infty} \text{ enter}

\(\text{wait} = \{\text{try}(r)\}\)

\(\text{enter} = \text{acq}(r) [[c]\] \text{ rel}(r)\)
Parallel composition

- Resource-sensitive
  - mutual exclusion for each resource
- Race-detecting
  - concurrent write $\Rightarrow$ catastrophe

- Fair
  - unfair to ignore persistent synchronization

Reynolds
Example

\[
\text{if } (\text{true} \rightarrow a?x;c!x) \quad \square \quad (\text{true} \rightarrow b?x;c!x) \quad \text{fi}
\]

\[
\delta_{\{a?\}}^\infty \{a?v \ x:=v \ c!v \mid v \in \mathbb{V}_{\text{int}} \} \\
\cup \\
\delta_{\{b?\}}^\infty \{b?v \ x:=v \ c!v \mid v \in \mathbb{V}_{\text{int}} \}
\]

denotes
Example

\[
\text{if } (a?x \rightarrow c!x) \Box (b?x \rightarrow c!x) \text{ fi}
\]

denotes

\[
\delta_{\{a?,b?\}} \infty \{ a?v \ x:=v \ c!v \mid v \in V_{int} \} \\
\cup \\
\delta_{\{a?,b?\}} \infty \{ b?v \ x:=v \ c!v \mid v \in V_{int} \}
\]
Example

\[
\text{if } (a?x \rightarrow c!x), (\text{true} \rightarrow \text{stop}) \text{ fi}
\]

\[
\text{denotes}
\]

\[
\delta_{\{a?\}} \{ a?v \ x:=v \ c!v \mid v \in V_{\text{int}} \}
\]

\[
\cup
\]

\[
\delta_{\{a?\}} \{ \delta\omega \}
\]
Example

\[ \text{chan } a \text{ in} \]
\[ \text{do } (\text{true} \rightarrow a?x) \text{ od } || \text{ do } (\text{true} \rightarrow a!0) \text{ od} \]

\[ \{ \delta^\omega \} \]

denotes
Connections

Original CSP
- no shared variables
- restricted use of channels

Theoretical CSP
- no imperative constructs
- hiding vs. local channel declaration
Generality

Action trace semantics for:

- shared memory parallel programs
  Brookes (MFPS’05)

- asynchronous communication
  Brookes (CONCUR’02)

- Concurrent Separation Logic
  Brookes, O’Hearn (CONCUR’04)
Conclusion

- Traces suffice
  - compositional, fair
  - deadlock, safety, liveness
- Unification of paradigms
  - shared memory
  - message-passing

CSP continues to thrive....
Related Work

CCS
- branching vs linear time
- bisimulation vs trace equivalence

Traces
- for shared memory (Park)
- for concurrent constraint programs (Palamidessi, Rutten, deBoer, ...)
- many other variations on this theme ...
Lessons

One man’s trace is another man’s failure

Traces suffice, after all...