Retracing the semantics of CSP

Stephen Brookes
Carnegie Mellon University

Outline

- The Essence of CSP
  - Original CSP
  - Theoretical CSP
- Semantic models
  - communication traces, failures, divergences
- Unification
  - action traces

Original CSP

Hoare 1978

- sequential guarded commands
- input and output
- parallel composition
  - synchronized communication between named processes

Design features

original CSP

- Imperative processes
  - disjoint local states
  - no concurrent writes to the same variable
- Static network structure
  - no nested parallelism
  - pairwise use of channels

Theoretical CSP

Hoare, Brookes, Roscoe 1984

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

Design features

Theoretical CSP

- State-free
- Event-based model
- Nested parallelism
- Dynamic network structure

Semantic models

- Communication traces
- Traces + refusals = failures
- Failures + divergence traces

designed to support
compositional reasoning

Communication traces

Hoare 1980

- Trace = sequence of inputs and outputs
- Process = prefix-closed set of traces
- Trace sets ordered by set inclusion

suitable for
safety properties

Failures

Hoare, Brookes, Roscoe 1984

- Failure = communication trace + refusal
- Refusal = set of inputs and outputs
- Process = closed set of failures
- Ordered by reverse set inclusion

suitable for
safety properties

+

deadlock analysis

Failures/divergences

Brookes, Roscoe 1985

- Process = failures + divergence traces
- Divergence is catastrophic

safety properties

deadlock analysis

absence of 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 communication traces

different failures

Examples

if (a?x→c!x) fi

if (a?x→c!x)□(true→stop) fi

same communication traces

different failures

Examples

chan a in

do (true→a?x) od || do (true→a!0) od

infinite internal chatter

no finite failures

divergence at empty trace

Summary so far

- Communication traces
  - cannot model deadlock or divergence
- Failures
  - cannot model divergence
- Failures/divergences
  - allows compositional reasoning
  - basis for FDR model checker

Issues

- Lack of fairness
  - relevant for liveness analysis
- Hard to extend
  - traces + refusals + divergences + ???
- Catastrophic view of divergence
  - not the only choice
- Models are specialized
  - not applicable to other paradigms

Unification

- Shared-memory parallel programs
- Communicating processes
  - asynchronous i/o
  - synchronous i/o

need a common semantic framework

traces suffice ....

Traces, revisited

action traces

- Actions include input, output, waiting, ...
  - have e*ffect* on state
- Action trace = sequence of actions
- Process = set of action traces
- Ordered by set inclusion

Features

- Complete traces
  - not prefix-closed
- Fairness
- State-free presentation
- State-dependent interpretation
- Suitable unifying framework

CPP

Communicating Parallel Processes

- Imperative processes
  - disjoint local state
  - shared state, including channels
- Synchronization
  - conditional critical regions
  - input and output

Actions

- Communication
  - h?v, h!v, δ_D
- Reading and writing
  - x=v, x:=v
- Resource management
  - try(r), acq(r), rel(r)
- Runtime error
  - abort

Semantics

- Process denotes set of traces
  - $[\![h!0]\!] = \delta_{\{h!\}}^{\infty}\ h!0$
  - $[\![h?x]\!] = \delta_{\{h?\}}^{\infty}\ \{h?v\ x:=v \mid v \in V_{int}\}$
  - $[\![c_1 \mid\mid c_2]\!] = [\![c_1]\!]\ \varnothing||\varnothing\ [\![c_2]\!]$
  - $[\![\text{with r do c}]\!] = \text{wait}^{\infty}\ \text{enter}$

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

    $\text{enter} = \text{acq(r)}\ [\![c]\!]\ \text{rel(r)}$

Parallel composition

- Resource-sensitive
  - mutual exclusion for each resource
- Race-detecting
  - concurrent write => catastrophe
- Weakly fair
  - unfair to ignore a persistent match

Rationale

- Process behavior depends on resources
- Race causes unpredictable behavior
  - results sensitive to granularity
- Fairness crucial for liveness analysis

Define

$$\alpha_1 \; A_1 \| A_2 \; \alpha_2$$

when A_1 and A_2 are disjoint sets of resources

Parallel composition

$$(\lambda_1\,\alpha_1)\;{}_{A_1}\|_{A_2}\,(\lambda_2\,\alpha_2) \;=\; \{abort\}$$

if $\lambda_1 \bowtie \lambda_2$

$$\begin{aligned}
&(\lambda_1\alpha_1)\;{}_{A_1}\|_{A_2}\,(\lambda_2\alpha_2) \;= \\
&\quad \{\lambda_1\gamma \mid (A_1, A_2) \xrightarrow{\lambda_1} (A_1', A_2)\;\&\;\gamma \in \alpha_1\;{}_{A_1'}\|_{A_2}\,(\lambda_2\alpha_2)\} \\
&\cup\, \{\lambda_2\gamma \mid (A_2, A_1) \xrightarrow{\lambda_2} (A_2', A_1)\;\&\;\gamma \in (\lambda_1\alpha_1)\;{}_{A_1}\|_{A_2'}\,\alpha_2\} \\
&\cup\, \{\delta\gamma \mid match(\lambda_1, \lambda_2)\;\&\;\gamma \in \alpha_1\;{}_{A_1}\|_{A_2}\,\alpha_2\}
\end{aligned}$$

otherwise

Interference

$$x{=}v \bowtie x{:=}v'$$
$$x{:=}v \bowtie x{=}v'$$
$$x{:=}v \bowtie x{:=}v'$$

$$h!v \bowtie h!v'$$
$$h?v \bowtie h?v'$$

concurrent write to same variable

concurrent output on same channel

concurrent input on same channel

Resource enabling

$$(A_1, A_2) \xrightarrow{try(r)} (A_1, A_2)$$

$$(A_1, A_2) \xrightarrow{acq(r)} (A_1 \cup \{r\}, A_2) \quad if\ r \notin A_1 \cup A_2$$

$$(A_1, A_2) \xrightarrow{rel(r)} (A_1 - \{r\}, A_2) \quad if\ r \in A_1$$

$$(A_1, A_2) \xrightarrow{\lambda} (A_1, A_2) \quad otherwise$$

Match

$$match(\lambda_1,\ \lambda_2)$$

if

$$\exists h, v.\ \{\lambda_1,\ \lambda_2\} = \{h?v,\ h!v\}$$

Fairness

- Unfair to ignore a persistent match

$$\delta_X{}^{\omega}\ {}_{A_1}\|_{A_2}\ \delta_Y{}^{\omega} \;=\; \{\}$$

if

$$\exists h.\ (h? \in X\ \&\ h! \in Y) \vee (h! \in X\ \&\ h? \in Y)$$

Example

if (true→a?x;c!x)□(true→b?x;c!x) fi

denotes

$$\delta_{\{a?\}}^{\infty}\{a?v\ x:=v\ c!v \mid v \in V\}$$

$$\cup$$

$$\delta_{\{b?\}}^{\infty}\{b?v\ x:=v\ c!v \mid v \in V\}$$

Example

$$\text{if } (a?x \rightarrow c!x) \,\square\, (b?x \rightarrow c!x) \text{ fi}$$

denotes

$$\delta_{\{a?,b?\}}^{\infty} \{a?v\ x:=v\ c!v \mid v \in V\}$$

$$\cup$$

$$\delta_{\{a?,b?\}}^{\infty} \{b?v\ x:=v\ c!v \mid v \in V\}$$

Example

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

denotes

$$\delta_{\{a?\}}^{\infty} \{a?v\ x:=v\ c!v \mid v \in V\}$$

$$\cup$$

$$\delta_{\{a?\}}^{\infty} \{ \delta^{\omega} \}$$

Example

chan a in

do (true→a?x) od || do (true→a!0) od

denotes

$\{ \delta^{\omega} \}$

Connections

- Original CSP
  - no shared variables
  - restricted use of channels
- Theoretical CSP
  - no imperative constructs
  - hiding vs. local channel declaration

Granularity

- Our model is “high-level”
  - reads, writes, input/output are atomic
- Can also give a “low-level” model
  - integer = sequence of words
  - high-level action = sequence of low-level actions
- High-level is consistent with low-level
  - because of race-detection

Generality

- Action trace semantics also works for:
  - shared memory parallel programs
  - asynchronous communication

Asynchrony

- Process denotes set of traces
  - $[\![h!0]\!] = \{ h!0 \}$
  - $[\![h?x]\!] = \delta_{\{h?\}}^{\infty} \{h?v\ x:=v \mid v \in V_{int}\}$
  - $[\![c_1 \mid\mid c_2]\!] = [\![c_1]\!] \varnothing||\varnothing [\![c_2]\!]$
  - $[\![\text{with r do c}]\!] = \text{wait}^{\infty}\ \text{enter}$

Parallel composition

asynchronous

$$(\lambda_1\,\alpha_1)\;{}_{A_1}\|_{A_2}\,(\lambda_2\,\alpha_2) \;=\; \{abort\}$$

if $\lambda_1 \bowtie \lambda_2$

$$(\lambda_1\alpha_1)\;{}_{A_1}\|_{A_2}\,(\lambda_2\alpha_2) \;=$$
$$\{\lambda_1\gamma \mid (A_1, A_2) \xrightarrow{\lambda_1} (A_1', A_2)\;\&\;\gamma \in \alpha_1\;{}_{A_1'}\|_{A_2}\,(\lambda_2\alpha_2)\}$$
$$\cup\,\{\lambda_2\gamma \mid (A_2, A_1) \xrightarrow{\lambda_2} (A_2', A_1)\;\&\;\gamma \in (\lambda_1\alpha_1)\;{}_{A_1}\|_{A_2'}\,\alpha_2\}$$

otherwise

Conclusion

- Traces are enough
  - compositional, fair
  - deadlock, safety, liveness
- Smooth blend of paradigms
  - shared memory
  - message-passing
- CSP continues to thrive....