Traces: a unifying semantic framework for parallel programming languages

Stephen Brookes

Department of Computer Science
Carnegie Mellon University

PARADIGMS

- Deterministic sequential
  - while-loops, assignment
- Non-deterministic sequential
  - guarded commands
- Shared-memory parallel
  - parallel composition
  - conditional atomic actions
- Communicating parallel
  - parallel composition
  - message-passing
    - synchronous
    - asynchronous

SEMANTIC MODELS

- **Deterministic sequential**

  partial functions

  $\mathbf{S} \rightarrow \mathbf{S}_\perp$

- **Non-deterministic sequential**

  relations

  $\mathcal{P}(\mathbf{S} \times \mathbf{S}_\perp)$

- **Shared memory parallel**

  transition traces

  $\mathcal{P}((\mathbf{S} \times \mathbf{S})^\infty)$

- **Asynchronous parallel**

  transition traces

  $\mathcal{P}((\mathbf{S} \times \mathbf{S})^\infty)$

- **Synchronous parallel**

  failures

  $\mathcal{P}(\Sigma^* \times \mathcal{P}(\Sigma))$

PROGRAM BEHAVIOR

- **Partial correctness**

$$\{pre\}\ P\ \{post\}$$

- **Total correctness**

$$[pre]\ P\ [post]$$

- **Safety properties**

$$pre \;\Rightarrow\; \Box \neg bad$$

- **Liveness properties**

$$pre \;\Rightarrow\; \Diamond\, good$$

Fairness is crucial
for liveness analysis

FAIRNESS

For shared-memory or asynchrony

- **Enabling is local**

  $$P \| Q \xrightarrow{\lambda} \quad \text{if} \quad P \xrightarrow{\lambda} \text{ or } Q \xrightarrow{\lambda}$$

- **Reasonable assumption:**

  no process is ignored forever

Weak (process) fairness

Satisfied by round-robin scheduler

Can model with *transition traces*

FAIRNESS

For synchronous processes…

- **Enabling is not local**

  $P \| Q \xrightarrow{\delta}$ if $P \xrightarrow{h!v}$ & $Q \xrightarrow{h?v}$

- **Reasonable assumptions:**

  no process is ignored forever

  no potential synchronization is ignored forever

Satisfied by variant of round-robin

Not modelled by failures

THIS TALK

- **A fair semantics for CSP**
  - avoids complex book-keeping
  - *state* handled implicitly
- **Generalization of failures**
  - handles deadlock, divergence
- **Full abstraction**
  - safety and liveness
- **A unifying framework**
  - shared-memory
  - asynchronous
  - synchronous
    - blocking or non-blocking guards

SYNTAX

- **Processes**

$$
\begin{aligned}
P \; ::= \; & \textbf{skip} \mid x{:=}e \mid P_1 ; P_2 \mid \\
& h?x \mid h!e \mid \\
& P_1 \| P_2 \mid \\
& \textbf{if } G \textbf{ fi} \mid \textbf{do } G \textbf{ od} \mid \\
& \textbf{local } x, h \textbf{ in } P
\end{aligned}
$$

- **Guarded commands**

$$G \; ::= \; (g \rightarrow P) \mid G_1 \Box G_2$$

- **Guards**

$$g \; ::= \; b \mid b \wedge h?x \mid b \wedge h!e$$

ACTIONS

$$
\begin{array}{rcll}
\lambda & ::= & x{=}v & \text{read} \\
& | & x{:=}v & \text{write} \\
& | & h?v & \text{input} \\
& | & h!v & \text{output} \\
& | & \delta_X & \text{wait}
\end{array}
$$

$$\text{where } X \subseteq \{h?, h! \mid h \in \mathbf{Chan}\}$$

TRACES

Finite or infinite sequences of actions

$$\alpha \in \Lambda^{\infty} = \Lambda^{+} \cup \Lambda^{\omega}$$

$$\delta\lambda = \lambda\delta = \lambda$$

STATES

Characterized implicitly by enabling relation

$$s \xrightarrow{\lambda} s'$$

OPERATIONAL SEMANTICS

- **Transitions**

$$P \xrightarrow{\lambda} P'$$

$$G \xrightarrow{\lambda} G'$$

- **Termination**

$$P \ \mathbf{term}$$

- **Fair execution**

$$P \xrightarrow{\alpha}$$

TRANSITIONS FOR GUARDED COMMANDS

$$\frac{}{(h?x \rightarrow P) \xrightarrow{h?v} x{:=}v; P}$$

$$\frac{}{(h?x \rightarrow P) \xrightarrow{\delta_{h?}} (h?x \rightarrow P)}$$

$$\frac{G_1 \xrightarrow{\lambda} P_1}{G_1 \Box G_2 \xrightarrow{\lambda} P_1} \quad \lambda \notin \Delta$$

$$\frac{G_2 \xrightarrow{\lambda} P_2}{G_1 \Box G_2 \xrightarrow{\lambda} P_2} \quad \lambda \notin \Delta$$

$$\frac{G_1 \xrightarrow{\delta_X} G_1 \quad G_2 \xrightarrow{\delta_Y} G_2}{G_1 \Box G_2 \xrightarrow{\delta_{X \cup Y}} G_1 \Box G_2}$$

TRANSITIONS FOR PROCESSES

$$\frac{P_1 \xrightarrow{\lambda} P_1'}{P_1 \| P_2 \xrightarrow{\lambda} P_1' \| P_2} \qquad \frac{P_2 \xrightarrow{\lambda} P_2'}{P_1 \| P_2 \xrightarrow{\lambda} P_1 \| P_2'}$$

$$\frac{P_1 \xrightarrow{\lambda_1} P_1' \quad P_2 \xrightarrow{\lambda_2} P_2'}{P_1 \| P_2 \xrightarrow{\delta} P_1' \| P_2'}$$
$$\text{if } match(\lambda_1, \lambda_2)$$

TERMINATION

$$\frac{P_1 \ \mathbf{term} \quad P_2 \ \mathbf{term}}{P_1 \| P_2 \ \mathbf{term}}$$

FAIR EXECUTIONS

Parallel composition

$$P \| Q \xrightarrow{\gamma} \quad \text{iff}$$

$$P \xrightarrow{\alpha},\ Q \xrightarrow{\beta},$$
$$\gamma \in \mathit{merges}(\alpha, \beta),$$
$$\neg \mathit{match}(\mathit{blocks}(\alpha), \mathit{blocks}(\beta))$$

- $\mathit{merges}(\alpha, \beta)$ allows synchronization
- $\mathit{blocks}(\alpha)$ is set of directions occurring infinitely often in δ_X steps

Local channels

$$\textbf{local } h \textbf{ in } P \xrightarrow{\alpha} \quad \text{iff} \quad P \xrightarrow{\alpha},\ h \notin \mathit{chans}(\alpha)$$

- forces synchronization on h

DENOTATIONAL SEMANTICS

- Define trace sets

  $$\mathcal{T}(P) \subseteq \Lambda^\infty$$

  with

  $$\begin{array}{l} \mathcal{T}(e) \subseteq \Lambda^* \times V \\ \mathcal{T}(g) \subseteq \Lambda^* \times \{\mathbf{true}, \mathbf{false}\} \\ \mathcal{T}(G) \subseteq \Lambda^\infty \end{array}$$

  by structural induction

- Designed to match operational semantics

- $\mathcal{T}(P)$ only includes fair traces

SEMANTIC DEFINITIONS

$$\mathcal{T}(\textbf{skip}) = \{\delta\}$$

$$\mathcal{T}(h?x) = {\delta_{h?}}^{*}\{h?v\, x{:=}v \mid v \in V\} \cup {\delta_{h?}}^{\omega}$$

$$\mathcal{T}(h!e) = \{\alpha\, {\delta_{h!}}^{*}\, h!v,\ \alpha{\delta_{h!}}^{\omega} \mid (\alpha, v) \in \mathcal{T}(e)\}$$

$$\begin{aligned}\mathcal{T}(P_1 \| P_2) = \{\alpha \in \mathit{merges}(\alpha_1, \alpha_2) \mid \\ \alpha_1 \in \mathcal{T}(P_1),\ \alpha_2 \in \mathcal{T}(P_2), \\ \neg \mathit{match}(\mathit{blocks}(\alpha_1), \mathit{blocks}(\alpha_2))\}\end{aligned}$$

$$\begin{aligned}&\mathcal{T}(\textbf{local}\ h\ \textbf{in}\ P) = \\ &\qquad \{\alpha \backslash h \mid \alpha \in \mathcal{T}(P)\ \&\ h \notin \mathit{chans}(\alpha)\}\end{aligned}$$

$$\begin{aligned}&\mathcal{T}(G_1 \Box G_2) = \\ &\quad \{\alpha \in \mathcal{T}(G_1) \cup \mathcal{T}(G_2) \mid \alpha \notin \Delta^{\omega}\} \cup \\ &\quad \{{\delta_{X \cup Y}}^{\omega} \mid {\delta_X}^{\omega} \in \mathcal{T}(G_1), {\delta_Y}^{\omega} \in \mathcal{T}(G_2)\}\end{aligned}$$

RESULTS

- **Denotational matches operational**

$$\mathcal{T}(P) = \{\alpha \mid P \xrightarrow{\alpha}\}$$

- **Traces are sensitive to deadlock**

$$\textbf{if } (a?x \rightarrow P)\Box(b?y \rightarrow Q) \textbf{ fi}$$

has $\delta_{\{a?,b?\}}{}^{\omega}$

$$\textbf{if } (\textbf{true} \rightarrow a?x; P)\Box(\textbf{true} \rightarrow b?y; Q) \textbf{ fi}$$

has $\delta_{a?}{}^{\omega}$ and $\delta_{b?}{}^{\omega}$

- **Full abstraction**

$$\mathcal{T}(P) = \mathcal{T}(Q) \iff \forall C.\mathcal{B}(C[P]) = \mathcal{B}(C[Q])$$

where $\mathcal{B}$ observes sequence of states

SEMANTIC LAWS

Fairness properties

$$\begin{aligned}\textbf{local } h \textbf{ in } & (h?x; P)\|(h!v; Q)\|R \\ = \ & \textbf{local } h \textbf{ in } (x{:=}v; (P\|Q))\|R \\ & \quad \text{if } h \notin \texttt{chans}(R)\end{aligned}$$

$$\begin{aligned}\textbf{local } h \textbf{ in } & (h?x; P)\|(Q_1; Q_2) \\ = \ & Q_1; \textbf{local } h \textbf{ in } (h?x; P)\|Q_2 \\ & \quad \text{if } h \notin \texttt{chans}(Q_1)\end{aligned}$$

$$\begin{aligned}\textbf{local } h \textbf{ in } & (h!v; P)\|(Q_1; Q_2) \\ = \ & Q_1; \textbf{local } h \textbf{ in } (h!v; P)\|Q_2 \\ & \quad \text{if } h \notin \texttt{chans}(Q_1)\end{aligned}$$

Not valid in unfair semantics

RELATED WORK

- **Traditional CSP models**
  - used finite, prefix-closed traces
  - cannot model fairness
  - treat divergence as catastrophic

- **Traces subsume (stable) failures**

  $(\alpha, R) \in \mathcal{F}(P) \;\Leftrightarrow\; \alpha(\delta_X)^\omega \in \mathcal{T}(P)$

  for some X such that $\neg match(X, R)$

- **Older's models**
  - different fairness notions
  - introduced *fairness mod X*
  - α is fair *mod* X if $blocks(\alpha) \subseteq X$

ADAPTABILITY

Can handle other parallel paradigms by making minor changes

- Choose appropriate set of actions Λ
- Adjust relevant semantic definitions
  - parallel composition
  - input/output
  - local channels

In each case:

- Processes denote trace sets
- Full abstraction for safety and liveness

ASYNCHRONOUS COMMUNICATION

$$\lambda ::= x{=}v \mid x{:=}v \mid h?v \mid h!v \mid \delta_X$$
$$\text{where } X \subseteq \{h? \mid h \in \mathbf{Chan}\}$$

$$\mathcal{T}(h!e) = \{\alpha\, h!v \mid (\alpha, v) \in \mathcal{T}(e)\}$$

$$\mathcal{T}(P_1 \| P_2) = \{\alpha \in \mathit{merges}(\alpha_1, \alpha_2) \mid \alpha_1 \in \mathcal{T}(P_1),\ \alpha_2 \in \mathcal{T}(P_2)\}$$

$$\mathcal{T}(\textbf{local } h \textbf{ in } P) = \{\alpha \backslash h \mid \alpha \in \mathcal{T}(P) \ \&\ \alpha\lceil h \text{ is FIFO}\}$$

- $\mathit{merges}(\alpha, \beta)$ without synchronization
- $\alpha\lceil h$ is FIFO if every input is *justified* by earlier output

SEMANTIC LAWS

asynchronous

Fairness properties

$$\begin{aligned}&\textbf{local } h \textbf{ in } (h?x;P)\|(h!v;Q)\|R\\ &\quad = \textbf{local } h \textbf{ in } (x{:=}v;P)\|Q\|R\\ &\qquad\qquad \text{if } h \notin \texttt{chans}(R)\end{aligned}$$

$$\begin{aligned}&\textbf{local } h \textbf{ in } (h?x;P)\|(Q_1;Q_2)\\ &\quad = Q_1; \textbf{local } h \textbf{ in } (h?x;P)\|Q_2\\ &\qquad\qquad \text{if } h \notin \texttt{chans}(Q_1)\end{aligned}$$

Not valid in unfair semantics

SHARED MEMORY

$$\lambda ::= x{=}v \mid x{:=}v \;\mid \langle\alpha\rangle \quad (\alpha \text{ finite})$$

$$\mathcal{T}(P_1 \| P_2) = \{\alpha \in \mathit{merges}(\alpha_1, \alpha_2) \mid \alpha_1 \in \mathcal{T}(P_1),\ \alpha_2 \in \mathcal{T}(P_2)\}$$

$$\mathcal{T}(\textbf{local } x \textbf{ in } P) = \{\alpha\backslash x \mid \alpha \in \mathcal{T}(P)\ \&\ \alpha\lceil x \text{ sequential}\}$$

$$\mathcal{T}(\textbf{await } b \textbf{ then } a) = \mathit{wait}^*\mathit{go} \;\cup\; \mathit{wait}^\omega$$

$$\mathit{wait} = \{\langle\alpha\rangle \mid (\alpha, \textbf{false}) \in \mathcal{A}(b)\}$$

$$\mathit{go} = \{\langle\alpha\beta\rangle \mid (\alpha, \textbf{true}) \in \mathcal{A}(b), \beta \in \mathcal{A}(a)\}$$

- $\alpha\lceil x$ sequential if each read of x is *justified* by previous write

COMMON THEME

- Programs denote sets of traces
  - built from action set Λ
- Fully abstract for safety and liveness
- Can extract traditional semantics
- Trace sets form complete lattice
- Program constructs denote monotone functions on trace sets

$$T_1 \subseteq T_2 \;\Rightarrow\; F(T_1) \subseteq F(T_2)$$

- Recursive constructs denote fixed points
  - least = finite traces
  - greatest = finite + infinite traces

FUTURE RESEARCH

- **Other fairness notions**
  - strong, weak / process, channel
- **Partial order semantics**
  - "truly fair" concurrency
- **Low-level traces**
  - pointers, stores, heaps
- **Procedures**
  - possible worlds, parametricity
- **Intensional traces**
  - abstract runtime
- **Probabilistic traces**
  - "fairly true" correctness

REFERENCES

- *Full abstraction for a shared-variable parallel language*, S. Brookes, LICS'93
- *On the Kahn Principle and Fair Networks*, S. Brookes, MFPS 14 (1998)
- *Communicating Sequential Processes*, C. A. R. Hoare, CACM (1978)
- *A Framework for Fair Communicating Processes*, S. Older, MFPS 13 (1997)
- *On the semantics of fair parallelism*, D. Park, Springer LNCS 86 (1979)
- *The Theory and Practice of Concurrency*, A. W. Roscoe, Prentice-Hall (1998)