HOW TO BE FAIR

Stephen Brookes

Carnegie Mellon University
School of Computer Science

FOCUS

- **parallel programming**
  - shared-variable programs
  - communicating processes
- **reasoning about programs**
  - safety and liveness
  - fairness assumptions

THEME

Dispelling myths about fairness

SHARED-VARIABLE PROGRAMS

- processes share a global state
- also have private local state
- communicate by reading and writing to shared variables
- synchronize with conditional atomic action await B then A
- busy-wait interpretation

COMMUNICATING PROCESSES

- processes have disjoint local states
- communicate by synchronized input and output along channels
- local actions are autonomous

PROGRAM PROPERTIES

Want to reason about:

- **safety**
  "something bad never happens"
  - mutual exclusion
  - absence of deadlock
- **liveness**
  "something good eventually happens"
  - critical code will get executed
  - no starvation

SEMANTIC CRITERIA

- Need to model the interaction or interference between parallel processes
- Properties of sequences of states, not state transformers

WHAT IS FAIRNESS?

- **an assumption**
  - ∗ no process is ignored forever
- **an abstraction**
  - ∗ every reasonable scheduler is fair

WHY FAIRNESS?

- abstracts away from unknown or unknowable scheduling details
- robustness of program analysis
- computational analogue of
  - – justice
  - – impartiality
  - – political correctness

MUTUAL EXCLUSION

```
local s:=true in
cobegin
     while true do
       (n1; await s then s:=false;
        c1; s:=true)
   || while true do
       (n2; await s then s:=false;
        c2; s:=true)
coend
```

PROPERTIES

- s is a *binary semaphore*
- c_1 and c_2 never concurrent
- fairness does not prevent starvation

Fairness is not a panacea

A GCD PROGRAM

$$P_x \| P_y \| P_z$$

where

$$
\begin{array}{ll}
P_x :: \texttt{while}\ \ x \neq y \vee x \neq z\ \texttt{do} & \\
\qquad\qquad \texttt{do} & \\
\qquad\qquad\quad (x > y \ \rightarrow\ x{:=}x - y) & \\
\qquad\qquad \square\ (x > z \ \rightarrow\ x{:=}x - z) & \\
\qquad\qquad \texttt{od} &
\end{array}
$$

P_y and P_z similar

PROPERTIES

- x, y, z are shared variables
- Only P_x changes x

A BAD GCD PROGRAM

$$Q_x \| Q_y \| Q_z$$

where

$$\begin{array}{l} Q_x :: \texttt{while } x \neq y \vee x \neq z \texttt{ do} \\ \qquad\qquad \texttt{do} \\ \qquad\qquad\quad (x > y \ \rightarrow \ x{:=}x - y) \\ \qquad\qquad \Box \ (y > x \ \rightarrow \ y{:=}y - x) \\ \qquad\qquad \texttt{od} \end{array}$$

Q_y and Q_z similar

PROPERTIES

- x, y, z are shared variables
- Q_x and Q_z change x

PROPERTIES

Assuming that initially

$$x = a > 0 \ \wedge \ y = b > 0 \ \wedge x = c > 0$$

the program $P_x \| P_y \| P_z$

- preserves $x > 0 \wedge y > 0 \wedge z > 0$
- preserves $\gcd(x, y, z) = \gcd(a, b, c)$
- always terminates with $x = y = z$

provided the scheduler is fair.

The program has unfair executions in which P_z never makes a step

- irrelevant, unrealistic

Fairness is a reasonable abstraction

PROPERTIES

Assuming that initially

$$x = a > 0 \ \wedge \ y = b > 0 \ \wedge x = c > 0$$

the program $Q_x \| Q_y \| Q_z$

- may violate positivity of x, y, or z
- may fail to preserve $\gcd(x, y, z)$
- may loop forever

even if the scheduler is fair.

REASON

If $x = y + z$ then Q_x and Q_z might each decide to change x, leaving $x = 0$.

It's hard to write correct programs, let alone deal with fairness!

A GCD NETWORK

$$\texttt{channels}\ h_{12}, \ldots, h_{31}$$
$$\texttt{in}\ R_x \| R_y \| R_z$$

where

$$
\begin{array}{ll}
R_x :: & \texttt{local}\ \ y, z\ \texttt{in} \\
& h_{12}!x \| h_{13}!x \| h_{21}?y \| h_{31}?z; \\
& \texttt{while}\ x \neq y \vee x \neq z\ \texttt{do} \\
& \quad (\texttt{do} \\
& \qquad\quad (x > y \ \rightarrow\ x{:=}x - y) \\
& \qquad \Box\ \ (x > z \ \rightarrow\ x{:=}x - z) \\
& \qquad \texttt{od}; \\
& \qquad h_{12}!x \| h_{13}!x \| h_{21}?y \| h_{31}?z)
\end{array}
$$

R_y and R_z similar

Distributed snapshot

PROPERTIES

Assuming that initially

$x = a > 0 \;\wedge\; y = b > 0 \;\wedge x = c > 0$

the program $R_x \| R_y \| R_z$

- preserves $x > 0 \wedge y > 0 \wedge z > 0$
- preserves $\gcd(x, y, z)$=$\gcd(a, b, c)$
- always terminates with $x = y = z$
- is free of deadlock

provided the scheduler is fair.

WHAT'S FAIR?

- **weak fairness**
  - ∗ every continuously enabled process is eventually scheduled
- **strong fairness**
  - ∗ every continually enabled process is eventually scheduled

PROPERTIES

- A strongly fair scheduler is also weakly fair.
- Easy to build weakly fair schedulers using roundrobin strategy.
- No implied bound on service time.

REALITY CHECK

- **shared-variable programs**
  - ∗ enabledness is locally checkable
  - ∗ real schedulers are weakly fair
  - ∗ busy wait implies weak=strong
- **communicating processes**
  - ∗ enabledness not local
  - ∗ real schedulers are strongly fair
  - ∗ weakly fair schedulers less useful

WAIVER

Other forms of fairness may also be considered, e.g.

- channel
- communication
- unconditional-Γ-extreme

SEMANTIC STYLES

- **denotational**
  - ∗ semantic domains
  - ∗ semantic functions defined by structural induction
  - ∗ abstract
  - ∗ compositional
- **operational**
  - ∗ abstract machine
  - ∗ transition relation defined by inference rules
  - ∗ detailed
  - ∗ not compositional

MYTHS

- Denotational semantics cannot incorporate fairness
  * inherently non-continuous
  * unbounded non-determinism
  * problems with powerdomains
- Operational semantics can handle fairness easily
  * Francez-style treatment

SPIN

- Operational treatments are awkward
  * too sensitive to nuances of presentation
  * don’t handle nested parallelism
- Denotational semantics *can* incorporate fairness
  * monotonicity is enough
  * don’t need powerdomains

TRADITION

- **operational semantics**

$$\frac{\langle c_0, s\rangle \to \langle c_0', s'\rangle}{\langle c_0 \| c_1, s\rangle \to \langle c_0' \| c_1, s'\rangle}$$

$$\frac{\langle c_1, s\rangle \to \langle c_1', s'\rangle}{\langle c_0 \| c_1, s\rangle \to \langle c_0 \| c_1', s'\rangle}$$

  $\ast$ based on single steps

  $\ast$ unfair sequences must be removed

  $\ast$ no nested parallelism

- **resumption semantics**

$$R = S \to \wp(S + (R \times S))$$

  – based on single steps

  – recursive domain equation

  – powerdomain $\wp$

  – cannot extract fair sequences

TRACE SEMANTICS

- Programs denote trace sets

  semantic domain is $\wp(\Sigma^\infty)$,
  where $\Sigma = S \times S$, $\wp$ is powerset

- A trace $(s_0, s'_0)(s_1, s'_1)\ldots(s_n, s'_n)\ldots$ represents a fair interactive computation

- "Interference-free" traces represent fair computations

- Semantic function defined structurally

  – traces of $c_0; c_1$ by concatenation

  – traces of $c_0 \| c_1$ by fair interleaving

  – traces of a loop by iteration

- All operations on trace sets are monotone w.r.t. inclusion

SEMANTIC PROPERTIES

- Trace sets are *closed under stutters*

  $$\alpha\beta \in c \;\&\; s \in S \;\Rightarrow\; \alpha(s,s)\beta \in c$$

  and *closed under mumbles*

  $$\alpha(s,s')(s',s'')\beta \in c \;\Rightarrow\; \alpha(s,s'')\beta \in c$$

- Steps (s_i, s_i') represent finite sequences of atomic actions
- *Only* includes fair traces
- Fully abstract

  Semantics only distinguishes terms if they exhibit different safety or liveness behavior in some context

FAIRMERGE

Let $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$.

$(\alpha, \beta, \gamma) \in \text{fairmerge} \Leftrightarrow \gamma$ merges α and β

Characteristic properties:

- For all $\alpha \in \Sigma^\infty$,

$$(\alpha, \epsilon, \alpha), (\epsilon, \alpha, \alpha) \in \text{fairmerge};$$

- For all $\alpha, \beta \in \Sigma^+, (\alpha', \beta', \gamma') \in \text{fairmerge}$,

  $(\alpha\alpha', \beta\beta', \alpha\beta\gamma') \in \text{fairmerge}$, and
  $(\alpha\alpha', \beta\beta', \beta\alpha\gamma') \in \text{fairmerge}$.

FIXED POINT PROPERTY

fairmerge is the greatest fixed point of the above definition

MORALS

- Infinite behaviors and fair merges come from greatest fixed points

- Fairness is easy, denotationally
  - handles nested parallelism
  - adapts to communicating processes

- Powerdomains are a red herring
  - seem to preclude fairness
  - wrong computational intuition

- It pays to re-examine "tradition"
  - "folk theorems" may be myths

It's not hard to be fair...