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 $\text{await } B \text{ then } A$
• 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

declares a local boolean variable `s` which is initialized to `true`.

The `cobegin` construct allows multiple processes to execute in parallel.

Within each process, a `while` loop is used to ensure mutual exclusion.

The `await` keyword is used to synchronize with other processes.

Properties:

• `s` is a binary semaphore
• `c_1` and `c_2` never concurrently execute
• Fairness does not prevent starvation

`Fairness is not a panacea`
A GCD PROGRAM

\[ P_x \parallel P_y \parallel P_z \]

where

\[ P_x :: \text{while } x \neq y \lor x \neq z \text{ do} \]
\[ \text{do} \]
\[ (x > y \rightarrow x := x - y) \]
\[ \text{□} (x > z \rightarrow x := x - z) \]
\[ \text{od} \]

\[ P_y \text{ and } P_z \text{ similar} \]

PROPERTIES

• \( x, y, z \) are shared variables
• Only \( P_x \) changes \( x \)
A BAD GCD PROGRAM

\[ Q_x \parallel Q_y \parallel Q_z \]

where

\[ Q_x :: \text{while } x \neq y \lor x \neq z \text{ do} \]

\[ \text{do} \]

\[ (x > y \rightarrow x := x - y) \]

\[ \Box \ (y > x \rightarrow y := y - x) \]

\[ \text{od} \]

\[ Q_y \text{ and } Q_z \text{ similar} \]

PROPERTIES

• \( x, y, z \) are shared variables

• \( Q_x \text{ and } Q_z \) change \( x \)
PROPERTIES

Assuming that initially
\[ x = a > 0 \land y = b > 0 \land x = c > 0 \]
the program \( P_x \| P_y \| P_z \)

- preserves \( x > 0 \land y > 0 \land z > 0 \)
- 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

\[ x = a > 0 \land y = b > 0 \land x = c > 0 \]

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

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

where

$R_x :: \text{local } y, z \text{ in}$
$h_{12}!x \parallel h_{13}!x \parallel h_{21}?y \parallel h_{31}?z;$

while $x \neq y \lor x \neq z$ do

(do

($x > y \rightarrow x := x - y$)

$\Box (x > z \rightarrow x := x - z)$

od;

$h_{12}!x \parallel h_{13}!x \parallel h_{21}?y \parallel h_{31}?z$)

$R_y$ and $R_z$ similar

Distributed snapshot
PROPERTIES

Assuming that initially

\[ x = a > 0 \land y = b > 0 \land x = c > 0 \]

the program \( R_x \| R_y \| R_z \)

- preserves \( x > 0 \land y > 0 \land z > 0 \)
- 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**

\[
\langle c_0, s \rangle \rightarrow \langle c'_0, s' \rangle \\
\langle c_0 \parallel c_1, s \rangle \rightarrow \langle c'_0 \parallel c_1, s' \rangle \\
\langle c_1, s \rangle \rightarrow \langle c'_1, s' \rangle \\
\langle c_0 \parallel c_1, s \rangle \rightarrow \langle c_0 \parallel c'_1, s' \rangle
\]

* based on single steps
* unfair sequences must be removed
* no nested parallelism

**resumption semantics**

\[ R = S \rightarrow \wp(S + (R \times S)) \]

– 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 $c_0 \parallel c_1$ 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
  \[ \alpha \beta \in c \land s \in \mathcal{S} \Rightarrow \alpha(s, s)\beta \in c \]
  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 $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$.

$(\alpha, \beta, \gamma) \in \text{fairmerge} \iff \gamma \text{ merges } \alpha \text{ and } \beta$

Characteristic properties:

• For all $\alpha \in \Sigma^\infty$,
  $$(\alpha, \epsilon, \alpha), (\epsilon, \alpha, \alpha) \in \text{fairmerge};$$

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

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...