Probabilistic program algebra

Annabelle McIver
Macquarie University, Sydney

Carroll Morgan
UNSW, Sydney

Probabilistic Distributed systems

Problems to be solved include

Shared resources;
Leadership election;
Distributed consensus...

Often interested in the probability that some property holds.

The features of the solutions include

Randomisation within processors' program execution;

Adversial scheduling in the selection of which processor can execute a program step.

These features interact in unanticipated ways making the entire system hard to understand!

Program algebra

To describe a distributed system using algebra we use the following language constructs and terms:

x, y, z represent programs executed by individual processors, eg X, Y, Z;

0 stands for a "miracle";

1 stands for "skip";

x y stands for "execute x first and then y";

x + y stands for "execute either x or y, arbitrarily"

x* stands for "execute x for some arbitrary number of times".

Used together we can express for example that
x and y run in parallel: $(x+y)^*$, or
first x runs for a while, and then y does: x^*y^*

Standard program algebra

$(x+y) + z = x + (y+z)$

$x + y = y + x$

$x + x = x$

$0 + x = x$

Demonic nondeterminism and refinement

$x\,(y+z) = x\,y + x\,z$

$(x+y)z = x\,z + y\,z$

$x \leq y \Leftrightarrow x + y = y$

$x^* = 1 + x\,x^*$

$x\,(y\,z) = (x\,y)\,z$

$0\,x = x\,0 = 0$

$1\,x = x\,1 = x$

Sequential composition, magic and skip

$x\,y \leq x \Rightarrow x\,y^* = x$

$x\,y \leq y \Rightarrow x^*\,y = y$

*- induction rules: a left and a right

In standard relational models of programming these rules are sound....

.... in probabilistic models these two rules are not!

Probabilistic program algebra

$x\,(y+z) = x\,y + x\,z$

Today's talk explains the problem with these equalities when x, y and z are interpreted as probabilistic programs.

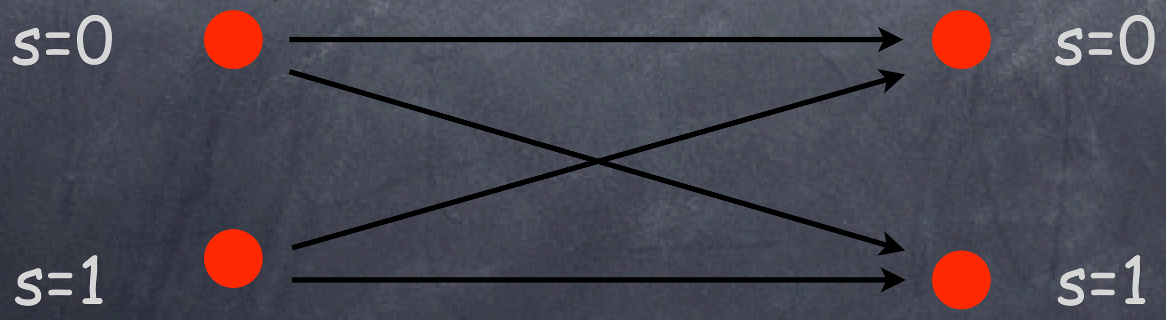
$x\,y \leq x \Rightarrow x\,y^* = x$

Some semantics

A probabilistic program such as "x" or "x+y" maps initial "states" to sets of probability distributions over final states:

$$S \longrightarrow \mathbb{P}\overline{S}$$

"s:=0 + s:=1" is the program that "outputs" 0 or 1 arbitrarily;

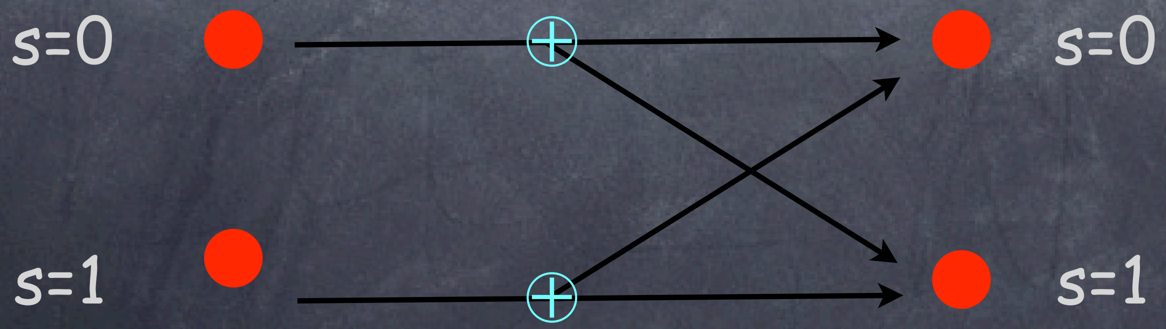


Some semantics

A probabilistic program such as "x" or "x+y" maps initial "states" to sets of probability distributions over final states:

$$S \longrightarrow \mathbb{P}\overline{S}$$

"s:=0 $\oplus_{½}$ s:=1" is the program that "outputs" 0 or 1 fairly;

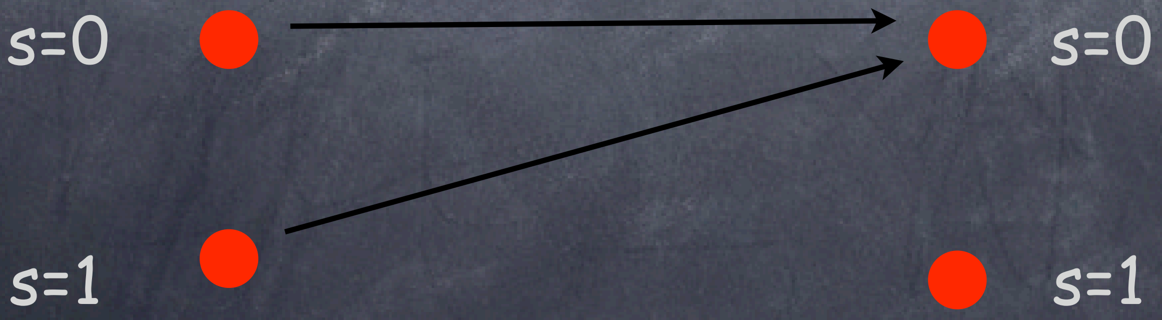


Some semantics

A probabilistic program such as "x" or "x+y" maps initial "states" to sets of probability distributions over final states:

$$S \longrightarrow \mathbb{P}\overline{S}$$

"s:=0" outputs 0 with "probability 1";

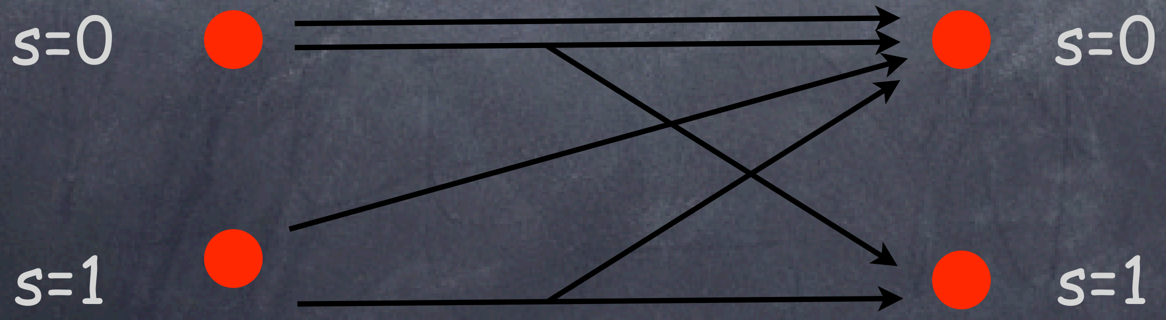


Some semantics

A probabilistic program such as "x" or "x+y" maps initial "states" to sets of probability distributions over final states:

$$S \longrightarrow \mathbb{P}\overline{S}$$

"s:=0" outputs 0 with "probability 1";
"s:=0 ⊕½ s:=1" is the program that "outputs" 0 or 1 fairly;

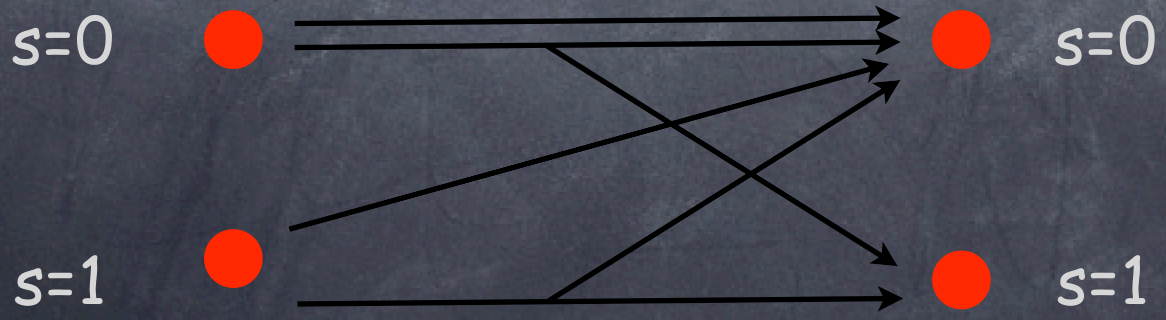


Some semantics

A probabilistic program such as “x” or “x+y” maps initial “states” to sets of probability distributions over final states:

$$S \longrightarrow \mathbb{P}\overline{S}$$

“s:=0 + (s:=0 ⊕½ s:=1)” outputs 0 with probability “at least 1/2”.



Some semantics

A probabilistic program such as "x" or "x+y" maps initial "states" to sets of probability distributions over final states:

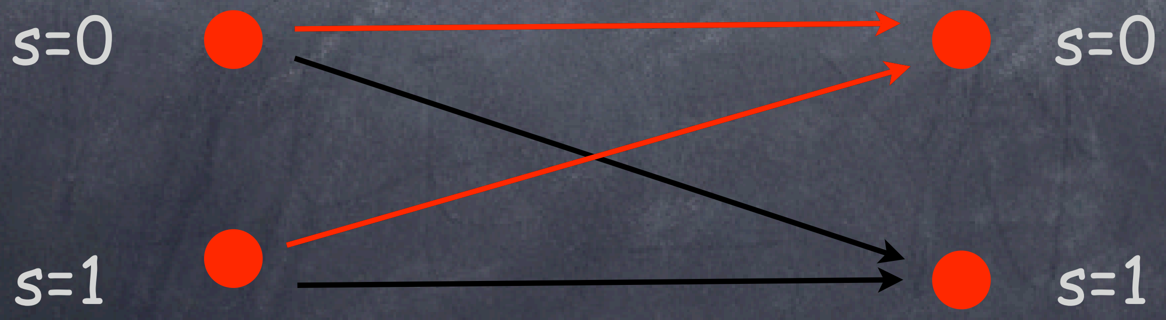
$$S \longrightarrow \mathbb{P}\overline{S}$$

Refinement reduces the range of nondeterminism:

$s{:=}0 \leq s{:=}0 + s{:=}1$

if P then $s{:=}0$ else $s{:=}1 \leq s{:=}0 + s{:=}1$

$s{:=}0 \oplus_{½}\, s{:=}1 \leq s{:=}0 + s{:=}1$



Some semantics

A probabilistic program such as "x" or "x+y" maps initial "states" to sets of probability distributions over final states:

$$S \longrightarrow \mathbb{P}\overline{S}$$

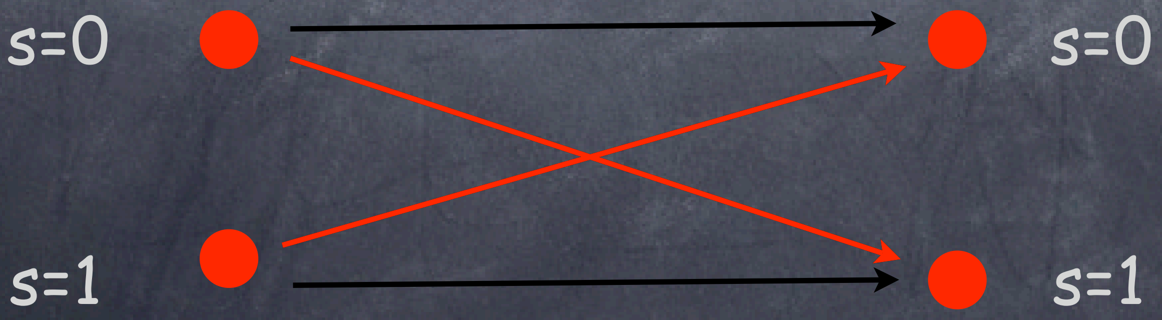
Refinement reduces the range of nondeterminism:

s:=0 ≤ s:=0 + s:=1

if P then s:=0 else s:=1 ≤ s:=0 + s:=1

s:=0 $\oplus_{½}$ s:=1 ≤ s:=0 + s:=1

Take P to be "s=1"



What's wrong with this equality?

$$x\,(y+z) = x\,y + x\,z$$

Let x be the program "$s_0 := 0 \oplus_{½} s_0 := 1$"

let y be the program "$s_1 := 0$"

let z be the program "$s_1 := 1$"

recall that "if $(s_0=0)$ then $s_1 := 0$ else $s_1 := 1 \leq s_1 := 0 + s_1 := 1$"

$$s_0 := 0;\ s_1 := 0 \quad \oplus_{½} \quad s_0 := 1;\ s_1 := 1$$

$$=$$

$$(s_0 := 0 \oplus_{½} s_0 := 1)\ ;$$

$$\text{if } (s_0=0) \text{ then } s_1 := 0 \text{ else } s_1 := 1$$

$$\leq$$

$$x\,(y+z)$$

The scheduler has a strategy to force s_0 and s_1 to take the same value.

What's wrong with this equality?

$x\ (y+z) = x\ y + x\ z$

Let x be the program "$s_0:=0 \oplus_{½} s_0:=1$"
let y be the program "$s_1:=0$"
let z be the program "$s_1:=1$"

$$\begin{array}{lcl} (s_0:=0 \oplus_{½} s_0:=1)\ ;\ s_1:=0 & & \\ + & = & x\ y + x\ z \\ (s_0:=0 \oplus_{½} s_0:=1)\ ;\ s_1:=1 & & \\ s_0:=0;\ s_1:=0 \quad \oplus_{½} \quad s_0:=1;\ s_1:=1 & \leq & x\ (y+z) \end{array}$$

The scheduler in x (y + z) has a strategy to set s_0 and s_1 to the same value with probability 1....

.... but no such strategy exists for the scheduler in x y + x z, as the probabilistic choice occurs after the scheduler has decided.

What's wrong with this *-induction rule?

$$x\,y \leq x \Rightarrow x\,y^* = x$$

Recall that $y^* = 1 + y\,y^*$, thus there is a scheduler implicitly acting within a *-iteration.

Again, this rule doesn't take the scheduler into into account....

Let y be the program "$s_0{:=}0 \oplus_{½} s_0{:=}1$".

Easily we have
$s_0{:=}0 \oplus_{½} s_0{:=}1 \,;\, s_0{:=}0 \oplus_{½} s_0{:=}1$ is the same as $s_0{:=}0 \oplus_{½} s_0{:=}1$....

.... but $(s_0{:=}0 \oplus_{½} s_0{:=}1)(s_0{:=}0 \oplus_{½} s_0{:=}1)^*$ is not the same as $s_0{:=}0 \oplus_{½} s_0{:=}1$.

The adversary scheduler could choose the strategy "stop when s_0 is 0" so that it can force the state to be set to 0 with probability 1.

Probabilistic program algebra

$(x+y) + z = x + (y+z)$

$x + y = y + x$

$x + x = x$

$0 + x = x$

$x\ (y\ z) = (x\ y)\ z$

$0\ x = x\ 0 = 0$

$1\ x = x\ 1 = x$

$x\ (y+z) \geq x\ y + x\ z$

$(x+y)\ z = x\ z + y\ z$

$x \leq y \Leftrightarrow x + y = y$

$x^* = 1 + x\ x^*$

$x(y+1) \leq x \Rightarrow x\ y^* = x$

$x\ y \leq y \Rightarrow x^*\ y = y$

These rules are now sound when interpreted over probabilistic systems,
and in fact there has only been one real change...

Some easy theorems

Transforming the body of an iteration:

$x\,(1+y) \leq z\,x \Rightarrow x\,y^* \leq z^* x$

generalised to:

$x\,(1+y_i\,) \leq z^*\,x$, for $i \in I$, then

$+_{i \in I}\,(x\,y_i{}^*) \leq z^* x$

This allows a simple specification of probabilistic properties of distributed systems.

Probabilistic separation:

$y\,(x+1) \leq (x+1)\,y^* \Rightarrow (x+y)^* = x^* y^*$

This allows actions of a protocol to appear to be executed atomically.

Bounded mutual exclusion: A tricky problem

A number of processors need to access a shared resource. Processors wishing to gain access to the resource at any time must compete with any others, in such a way that each process has probability at least k/m of "winning".

Here k is a constant value, independent of the size of the system, and m is the number of processes participating in the competition.

Processors are scheduled by an adversarial scheduler and "communicate" via shared variables if necessary. The scheduler usually has to satisfy some strong fairness criterion.

Let us suppose that the names of the processors are drawn from the index set I.

Let x_i represent the program of the i'th processor

The distributed system is thus represented as $(+_{i \in I}\ x_i)^*$

Let select be a program that nondeterministically selects a subset of I:

$$\text{select} := \text{names} :\subseteq I$$

Let vote be a program that chooses uniformly at random between a subset of names:

$$\text{vote} := w := \text{uniform}(M)$$

Problem: find definitions of the x_i such that

$$(+_{i \in I}\ x_i)^* \leq \text{select vote}$$

Something that doesn't work, but looks like it oughter....

Observe: a uniform random selection of one out of N items can be implemented iteratively....

```
Let $x_i$ :=   if ($i \notin$ names) then names := names $\cup$ $\{i\}$;
                                     w:= i $_{1/n}\oplus$ skip
               else skip
```

If the i'th processor has not voted....
.... win with probability relative to the number of processors that precede this one in the round.
(n= |names|)

vote x_i= $x_i$vote

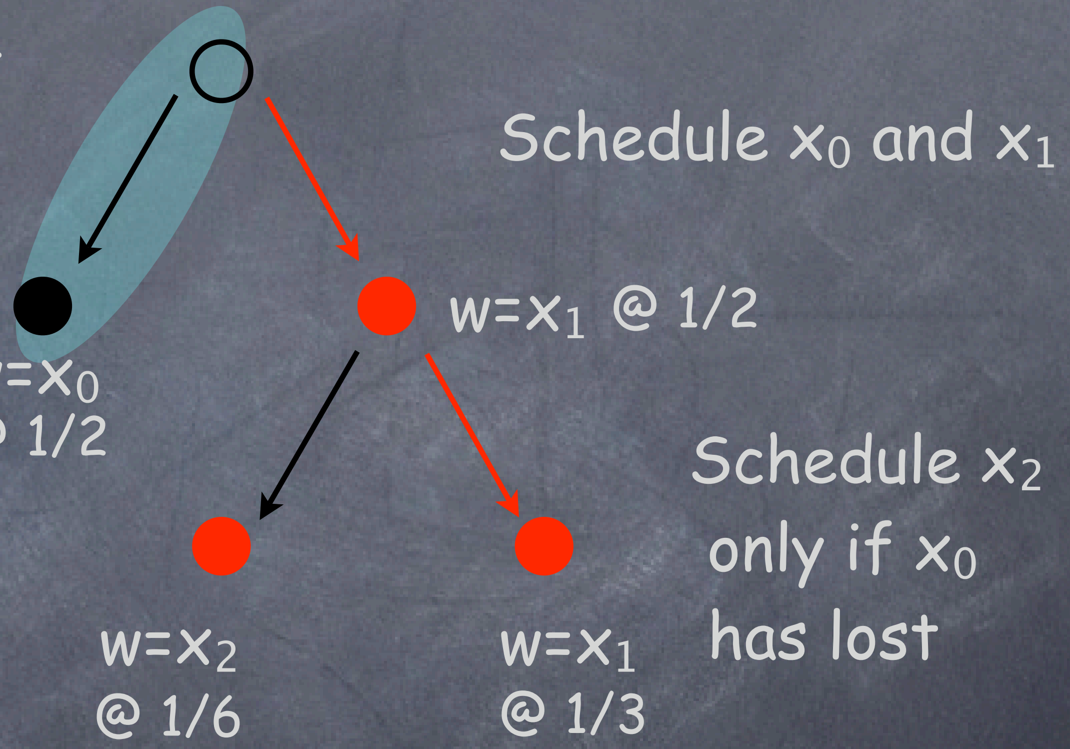
Does this solve the problem?

The adversary can favour a particular processor by terminating the round if its favourite is currently winning, selecting a new processor only if its favourite has already lost.

```
Let $x_i$ :=   if ($i \notin$ names) then names := names $\cup$ $\{i\}$;
                                w:= i $_{1/n}\oplus$ skip
               else skip
```

Now consider $(x_0 + x_1 + x_2)^*$

Observe that in all rounds in which exactly two processors are selected to vote x_0 wins every time....



Algebraically, this behaviour is captured by the failure of the inequality

vote $(x_i + 1) \not\leq (x_i + 1)$ vote

Something better....

.... split the protocol into two phases:

(a) processors register to vote;
(b) registered processors vote.

```
Let $x'_i$ :=   if ($i \notin names \wedge r=0$) then names := names $\cup$ {i};
                                                   $r := 0 + r := 1$
                else if ( $i \in names \wedge r=1$) then  $w := i \;{}_{1/n}\!\oplus$ skip
                else skip
```

The adversary cannot favour a processor since once the registration has ended, the length of the round is decided and the adversary is forced to allow all of the registered processors to vote.

If p= (r=0), then we can show that

$$(+_i x_i)^* \;\leq\; (+ \{p\}\, x_i\,)^* \; (+_i \{\neg p\}\, x_i\,)^* \leq \text{select; uniform}$$

Our contribution

Extension of (Kleene) algebra to a model of probabilistic programs

Further challenges

Infinite iterations (termination);
Study the impact of "tests" and probability.

http://web.comlab.ox.ac.uk/oucl/research/areas/probs/