System Specification, Verification and Synthesis (SSVS) – CS 4830/7485, Fall 2019

8: Formal System Modeling: Fairness

Stavros Tripakis

Northeastern University
Khoury College of Computer Sciences

Notes on Homework 01

- Moore vs Mealy distinction still a bit unclear.
- Both machines in homework are Moore, assuming we don't encode the input in the state.
- Some encoded the input in the state and said it's Mealy. This is OK.
- But some did not encode the input in the state, and still said it's Mealy. This is not OK.
- Also some got the transition function wrong (self-loops).

- From now on, please submit code as separate files.

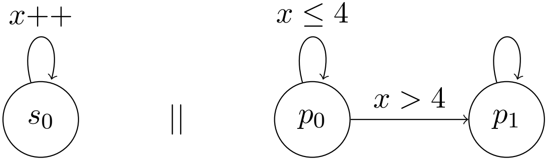
Notes on Presentations and Projects

- Preliminary lists posted on piazza.
- Send me your own suggestions (both for papers and projects).
- Goal: finalize things by end of next week (Oct 4).

FAIRNESS

Fairness: motivation

Consider the following asynchronous (interleaving) composition of two processes with shared variable x:



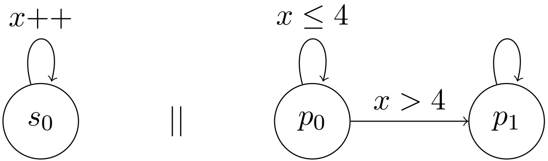
Will the rightmost process eventually move to p_1?

Is there any behavior where this will not happen?

Let's see what the transition system says:

Fairness: motivation

Consider the following asynchronous (interleaving) composition of two processes with shared variable x:



The transition system contains a behavior where the leftmost process keeps taking transitions forever, while the rightmost process never moves.

This is not realistic: no matter how slow the rightmost process is, it **will** move at some point (e.g., in a multi-threaded program, or a distributed system).

$\Rightarrow$ We need to exclude such unrealistic behaviors. But how?

Fairness: motivation

Fairness is a mechanism to exclude such unrealistic (**unfair**) behaviors.

Indispensable for proving properties of systems, e.g.:

- A message will eventually reach its destination: need to assume that the communication channel will not keep losing the message forever. This is a fairness assumption.
- In a distributed protocol, say, leader election, a leader will eventually be elected: need to assume that nodes will not keep failing.
  Again, a fairness assumption.
- Every bank transaction eventually completes: need to assume that a given transaction will not constantly be overlooked due to other transactions (no **starvation**).
  Again, a fairness assumption.
- ...

Defining fairness

We need to be precise: what exactly constitutes a "fair" behavior?

Two basic types [Manna and Pnueli, 1991]:

- **Weak fairness** (sometimes called "justice"): a process cannot be enabled forever after some point on, without getting to move.
- **Strong fairness** (sometimes called "compassion"): a process cannot be enabled infinitely often without getting to move.

where some process i is **enabled** means that the overall system (consisting of process i and potentially other processes) is at a state where process i **can** move.

Defining fairness

We need to be precise: what exactly constitutes a "fair" behavior?

Two basic types [Manna and Pnueli, 1991]:

- **Weak fairness** (sometimes called "justice"): a process cannot be enabled forever after some point on, without getting to move.
- **Strong fairness** (sometimes called "compassion"): a process cannot be enabled infinitely often without getting to move.

where some process i is **enabled** means that the overall system (consisting of process i and potentially other processes) is at a state where process i **can** move.

We can define fairness in different ways. E.g., we can make it part of the system, or we can make it part of the specification. E.g., instead of verifying that ϕ holds, we verify that $\phi_{\mathsf{fair}} \Rightarrow \phi$ holds. We will return to this later.

Weak fairness

- Let TS be a transition system formed by the asynchronous composition of n processes, $P_1, P_2, ..., P_n$.
- Let $s \rightarrow s'$ be a transition of TS. We write $s \xrightarrow{i} s'$, if process P_i makes a move in this transition. (Note that in asynchronous interleaving, a unique process makes a move at each step.)
- We say that P_i is **enabled** at some state s, if there exists a transition $s \xrightarrow{i} s'$.

Then we can define weak fairness:

If P_i is always enabled after some point on, it will eventually get to move.

Weak fairness

- Let TS be a transition system formed by the asynchronous composition of n processes, $P_1, P_2, ..., P_n$.
- Let $s \rightarrow s'$ be a transition of TS. We write $s \xrightarrow{i} s'$, if process P_i makes a move in this transition. (Note that in asynchronous interleaving, a unique process makes a move at each step.)
- We say that P_i is **enabled** at some state s, if there exists a transition $s \xrightarrow{i} s'$.

Then we can define weak fairness:

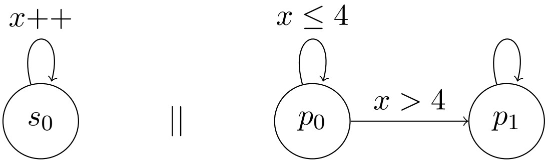
> *If P_i is always enabled after some point on, it will eventually get to move.*

or better:

> *A run $s_0 \xrightarrow{i_0} s_1 \xrightarrow{i_1} s_2 \xrightarrow{i_2} \cdots$ is weakly unfair to process P_i if there exists some integer K, such that P_i is enabled at all states s_j with $j \geq K$, but $\forall j \geq K : i_j \neq i$.*

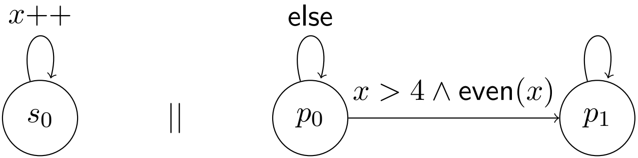
Weak fairness: example

Consider our earlier example. Weak fairness solves this problem:



The run where the transition from p_0 to p_1 never happens is weakly unfair to the rightmost process.

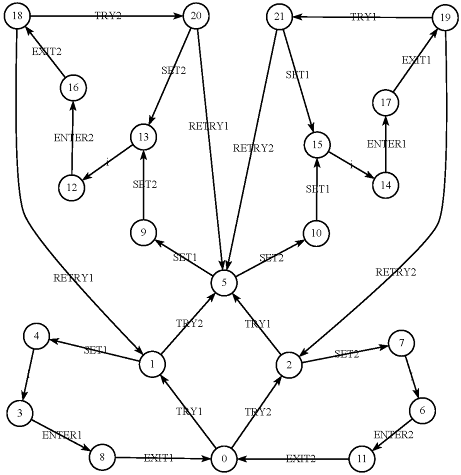
But weak fairness is sometimes too weak



Here, the run where the transition from p_0 to p_1 never happens is **not** weakly unfair, because the transition is **not always** enabled after some point on.

Weak fairness is sometimes too weak

A more realistic example:



How to ensure that both processes eventually enter their critical section?

Strong fairness

We define strong fairness:

If P_i is infinitely-often enabled, it eventually gets to move.

Strong fairness

We define strong fairness:

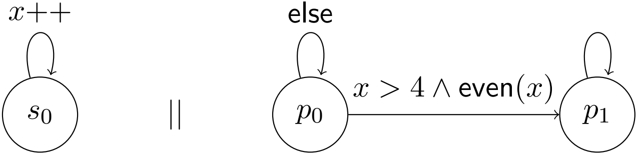
If P_i is infinitely-often enabled, it eventually gets to move.

or better:

A run $s_0 \xrightarrow{i_0} s_1 \xrightarrow{i_1} s_2 \xrightarrow{i_2} \cdots$ is strongly unfair to process P_i if P_i is enabled at state s_j for infinitely many j's, but there exists some integer K, such that $\forall j \geq K : i_j \neq i$.

Strong fairness: example

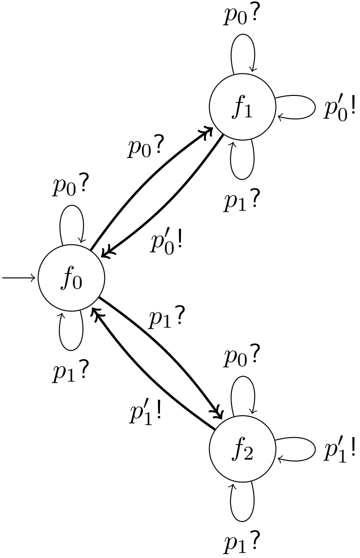
Strong fairness fixes our last example:



Here, the run where the transition from p_0 to p_1 never happens is unfair w.r.t. strong fairness, because the transition is infinitely-often enabled (more precisely: enabled every other step).

More fine-grained notions of fairness

Recall the Forward Channel process of the ABP example:



Transitions in bold lines and double arrows are strongly fair, meaning they cannot be enabled infinitely often without being taken.

Example of fairness in nuXmv

```
MODULE main
VAR
  state : { new, broken, fixed };
INIT
  state = new;
TRANS
  (state = new & (next(state) = new | next(state) = broken))
  |
  (state = broken & (next(state) = broken | next(state) = fixed))
  |
  (state = fixed & (next(state) = fixed | next(state) = broken));

JUSTICE state = fixed;

SPEC
  AG (state = broken -> AF (state = fixed));
```

Specification is violated without the JUSTICE assumption.

Fairness: poor man's probability

We could view fairness as an **abstraction** of probabilities.

- Example: consider a communication channel, which loses a message with probability $p = 10^{-6}$ and transmits it correctly with probability $1 - p$.
- In this system, a behavior where the message keeps getting forever lost has **zero** probability. So, in principle, probabilistic systems do not need fairness, since unfair behaviors have zero probability of occurring.
- Fairness allows us to avoid specifying probabilities. Even if we don't know what p is, we can still claim that a certain behavior is unfair.
- Also, probabilistic systems are (other things being equal) harder to verify than nondeterministic systems (because in addition to state-space exploration, we have to deal with the numbers).

Bibliography

Manna, Z. and Pnueli, A. (1991).
The Temporal Logic of Reactive and Concurrent Systems: Specification.
Springer-Verlag, New York.

Piterman, N. and Pnueli, A. (2018).
Temporal logic and fair discrete systems.
In Clarke, E. M., Henzinger, T. A., Veith, H., and Bloem, R., editors, *Handbook of Model Checking.*, pages 27–73. Springer.