Methods and Formal Models / Nachum Dershowitz
Lecture #10, May 30th, 2000
Notes by: Nadav Rephaelli

Semantics of Concurrency

Introduction

In *interleaving* semantics, we consider the execution two instances of a parallel program $P \parallel R$ to be equivalent to the union of all possible interleavings of the execution sequences $P_0;P_1;P_2;...$ and $R_0;R_1;R_2;...$ of atomic statements of P and R.

For concurrent programs, it is convenient to view the program as a labeled state-transition relation, that is, as a possibly infinite graph, called the *process graph*, with nodes Q called *states*, and edges labeled by *actions* A. There's also a distinguished root node. The *one-step* transition relation τ is a subset of $Q \times A \times Q$.

Our motivation is to formalize a way for organizing requests for the resource put by the concurrent occurrences of P, $P_1 \| P_2 \| \ldots \| P_n$, so that it's not used by more times than its limit.

Example

Suppose we have a program P, and we would like to run it in several parallel occurrences. Suppose also, that P uses some kind of a common resource, which is limited in the system.

Let P be the following program:

0. Let y be 1; forever:
    1. Play
    2. Ask for permission to use slide when y=1, otherwise stay in this state
    3. Slide, decrement y
    4. Leave slide, increment y

And let B and G be two concurrent occurrences of P. A state of this system is, as specified above, an ordered quadrate: $Q=(b,y,g)$, where $b,g \in \{0,1,2,3,4\}$, and $y \in \mathbf{N}$, or $Q=\{0,1,2,3,4,5\} \times \mathbf{N} \times \{0,1,2,3,4,5\}$.

Lemma 1: $y \in \{0, 1\}$ for a single process.

Lemma 1.1: if in state $q=k$, in the same iteration, y will be either k or k-1.

Proof:
1. In state q=1, y=k.
2. In state q=2, y=k.
3. In state q=3, y=k-1.
4. In state q=4, y=k.

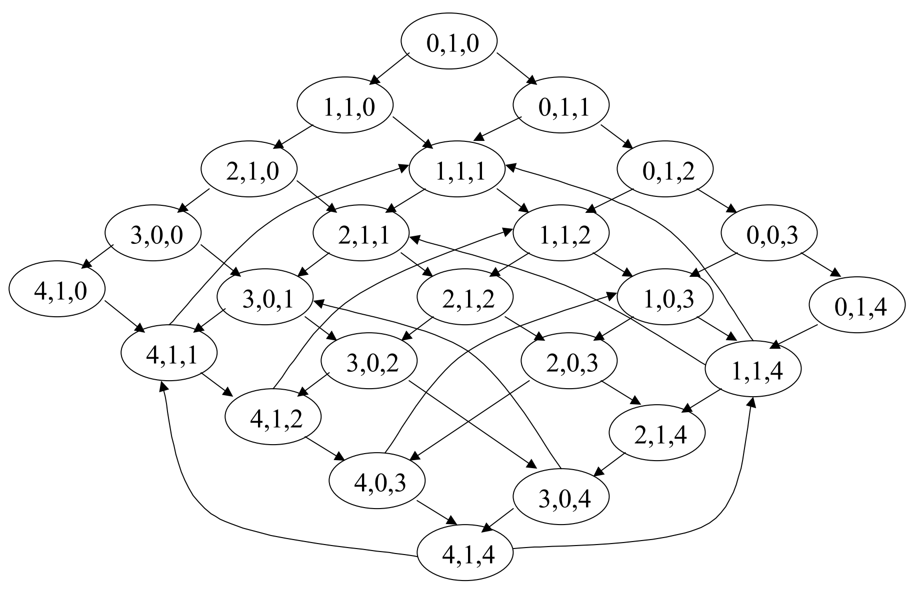
Lemma 1.2: in the state q=1, a will always be 1.
Proof: by induction on the number of iterations.
Base: on the first time we reach q=1, a is 1.
Step: suppose we ran k loop iterations, and the lemma is true. According to lemma 1.1, in the k+1-th iteration, a is 1.

Proof of lemma 1: by induction on the number of iterations.
Base: for state q=0, the lemma is true.
Step: suppose we ran k loop iterations in the loop, and we're inside the k+1-th iteration. If q=1, the lemma is true according to lemma 1.2. Otherwise, according to lemma 1.1, in any other state, a=0 or a=1.

Lemma 2: $y\in\{0, 1\}$ for two concurrent processes.
Proof: the only places where y's value changes, are states 3 and 4. Only a single process can reach state 3 at any given time. The value of y is then restored when reaching state 4, and only then can another process reach state 3.

So actually, our Q is $\{0,1,2,3,4,5\}\times\{0,1\}\times\{0,1,2,3,4,5\}$, yet states (3,_,3) are unreachable.

The *process graph* in this case would be:



In *branching time* semantics, we interpret processes as their computation trees. In *linear time* semantics, we interpret processes as the set of all their computations.

Formal Definition of CTL*[1]

There are two types of formulas in CTL*: *state formulas* (which are true in a specific state), and *path formulas* (which are true along a specific path). Let AP be the set of atomic proposition names. A *state formula* is either:

- A, if $A \in AP$.
- If f and g are state formulas, then $\neg f$ and $f \vee g$ are state formulas.
- If f is a *path formula*, then E(f) is a state formula.

A path formula is either:

- A state formula
- If f and g are path formulas, then $\neg f$, $f \vee g$, Xf, and fUg are path formulas.

Semantics of CTL*

CTL* semantics is interpreted over *Kripke Models*. A Kripke model is an ordered quadrate $M=(S,R,L)$, where:

- S is a finite set of states
- $R \subseteq S \times S$ is a transition relation
- $L : S \rightarrow 2^{AP}$ is a function, that assigns each state the set of atomic formulas that are true in it

A *path* $\pi = s_0 s_1 \ldots$ for a Kripke model M is an infinite series of states, for which forall i, $(s_i, s_{i+1}) \in R$.

Formal Definition of LTL (Linear Temporal Logic)

Linear temporal logic (LTL) consists of formulas that have the form Af, where f is a path formula, in which the only state subformulas permitted are atomic propositions.

LTL is a private case of CTL*.

Temporal Operators and Quantifiers

Theorem (not proved here): a system containing only the E quantifier and the Xf and fUg operators, is complete (meaning, has the same expression power as a system containing the A quantifier and the G, F temporal operators).

Additional Requirements from Concurrent Systems

Safety

We prohibit certain conditions from happening. In our example, the state $(3,y,3)$ is forbidden for any possible y, or: AG$\neg$ ($b=g=3$).

---

[1] E.M.Clarke, O.Grumberg: "Research on automatic verification of finite-state concurrent systems", Annual Reviews of Computer Science, Vol. 2, 269-290, 1987 (J.F. Traub, editor)

Response

We demand that every request we have is fulfilled within finite time frame, or:
$AG[b=2 \rightarrow AFb=3] \wedge AG[g=2 \rightarrow AFg=3]$

Fairness

Let $H=\{h_1, h_2, ...\} \subseteq 2^Q$ (where Q is the set of states and for every i, h_i is a subset of Q) be a set of *fairness requirements*. A computation path $\pi=q_0q_1...$ is considered to be *fair* regarding to H, if for all i, π has an infinite number of states that fulfill h_i.

In our example, h_0 is a set containing the state on which B plays, and h_1 is a set containing the state on which G plays.

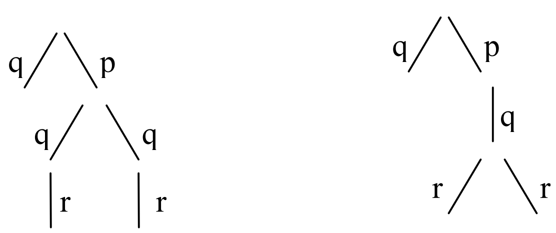
Model Cheching

Model checking for CTL* is an NPC problem.

Bisimulation

Two processes P, R are *bisimilar*, if for every action $a \in A$, the subtrees of P following a are each equivalent to some subtree of R following a, and vice-versa. The smallest bisimilarity exists, and is an equivalence.

Example: These are samples for bisimilar processes.



Example: This is a sample for non-bisimilar processes.