Model-checking algorithms

The semantic definitions for LTL and CTL presented allow us to test whether the initial states of a given system satisfy an LTL or CTL formula. This is the basic model-checking question. In general, interesting transition systems will have a huge number of states and the formula we are interested in checking may be quite long. It is therefore well worth trying to find efficient algorithms. Although LTL is generally preferred by specifiers, as already noted, we start with CTL model checking because its algorithm is simpler.

1 The CTL model-checking algorithm

Humans may find it easier to do model checks on the unwinding of models into infinite trees, given a designated initial state, for then all possible paths are plainly visible. However, if we think of implementing a model checker on a computer, we certainly cannot unwind transition systems into infinite trees. We need to do checks on finite data structures. For this reason, we now have to develop new insights into the semantics of CTL. Such a deeper understanding will provide the basis for an efficient algorithm which, given M, s ∈ S and φ, computes whether M, s φ holds. In the case that φ is not satisfied, such an algorithm can be augmented to produce an actual path (= run) of the system demonstrating that M cannot satisfy φ. That way, we may debug a system by trying to fix what enables runs which refute φ. There are various ways in which one could consider

$$\mathcal{M}, s_0 \stackrel{?}{\models} \phi$$

2 CTL model checking with fairness

The verification of M, s0 φ might fail because the model M may contain behaviour which is unrealistic, or guaranteed not to occur in the actual system being analysed. For example, in the mutual exclusion case, we expressed that the process prc can stay in its critical section (st=c) as long as it needs. We modelled this by the non-deterministic assignment

```
next(st) :=
   case
      ...
      (st = c)    : {c,n};
      ...
   esac;
```

It is for that reason that SMV allows us to impose fairness constraints on top of the transition system it describes. These assumptions state that a given formula is true infinitely often along every computation path. We call such paths fair computation paths. The presence of fairness constraints means that, when evaluating the truth of CTL formulas in specifications, the connectives A and E range only over fair paths.

3 The LTL model-checking algorithm

The algorithm presented in the sections above for CTL model checking is quite intuitive: given a system and a CTL formula, it labels states of the system with the subformulas of the formula which are satisfied there. The state-labelling approach is appropriate because subformulas of the formula may be evaluated in states of the system. This is not the case for LTL: subformulas of the formula must be evaluated not in states but along paths of the system. Therefore, LTL model checking has to adopt a different strategy. There are several algorithms for LTL model checking described in the literature. Although they differ in detail, nearly all of them adopt the same basic strategy.