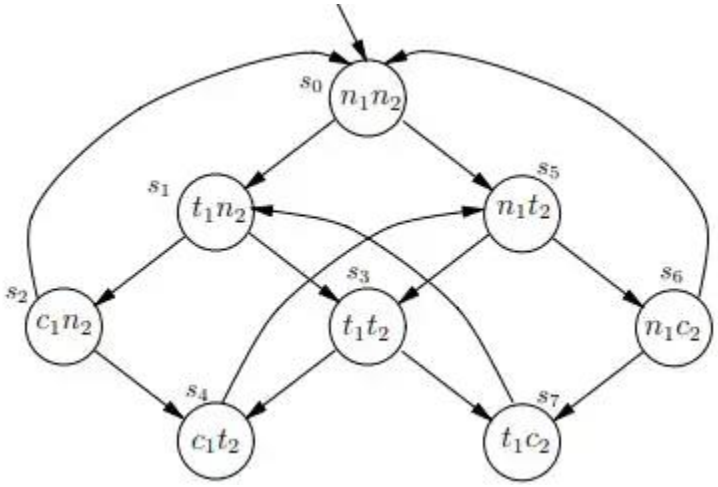
Mutual exclusion

Let us now look at a larger example of verification using LTL, having to do with mutual exclusion. When concurrent processes share a resource (such as a file on a disk or a database entry), it may be necessary to ensure that they do not have access to it at the same time. Several processes simultaneously editing the same file would not be desirable.

We therefore identify certain critical sections of each process' code and arrange that only one process can be in its critical section at a time. The critical section should include all the access to the shared resource (though it should be as small as possible so that no unnecessary exclusion takes place). The problem we are faced with is to find a protocol for determining which process is allowed to enter its critical section at which time. Once we have found one which we think works, we verify our solution by checking that it has some expected properties, such as the following ones:

Safety: Only one process is in its critical section at any time.



This safety property is not enough, since a protocol which permanently excluded every process from its critical section would be safe, but not very useful. Therefore, we should also require:

Liveness: Whenever any process requests to enter its critical section, it will eventually be permitted to do so.

Non-blocking: A process can always request to enter its critical section.

Some rather crude protocols might work on the basis that they cycle through the processes, making each one in turn enter its critical section. Since it might be naturally the case that some of them request access to the shared resource more often than others, we should make sure our protocol has the property:

No strict sequencing: Processes need not enter their critical section in strict sequence.

The NuSMV model checker

NuSMV stands for 'New Symbolic Model Verifier.' NuSMV is an Open Source product, is actively supported and has a substantial user community.

NuSMV (sometimes called simply SMV) provides a language for describing the models we have been drawing as diagrams and it directly checks the validity of LTL (and also CTL) formulas on those models. SMV takes as input a text consisting of a program describing a model and some specifications (temporal logic formulas). It produces as output either the word 'true' if the specifications hold, or a trace showing why the specification is false for the model represented by our program.

SMV programs consist of one or more modules. As in the programming language C, or Java, one of the modules must be called main. Modules can declare variables and assign to them. Assignments usually give the initial value of a variable and its next value as an expression in terms of the current values of variables. This expression can be non-deterministic (denoted by several expressions in braces, or no assignment at all). Non-determinism is used to model the environment and for abstraction.

The following input to SMV:

```
MODULE main
VAR
request : boolean;
status : {ready,busy};
ASSIGN
init(status) := ready;
next(status) := case
request : busy;
1 : {ready,busy};
esac;
LTLSPEC
G(request -> F status=busy)
```

consists of a program and a specification. The program has two variables, request of type boolean and status of enumeration type {ready, busy}: 0 denotes 'false' and 1 represents 'true.' The initial and subsequent values of variable request are not determined within this program; this conservatively models that these values are determined by an external environment. This under-

specification of request implies that the value of variable status is partially determined: initially, it is ready; and it becomes busy whenever request is true. If request is false, the next value of status is not determined.

Note that the case 1: signifies the default case, and that case statements are evaluated from the top down: if several expressions to the left of a ':' are true, then the command corresponding to the first, top-most true expression will be executed. there are four states, each one corresponding to a possible value of the two binary variables. Note that we wrote 'busy' as a shorthand for 'status=busy' and 'req' for 'request is true.'

It takes a while to get used to the syntax of SMV and its meaning. Since variable request functions as a genuine environment in this model, the program and the transition system are non-deterministic: i.e., the 'next state' is not uniquely defined. Any state transition based on the behaviour of status comes in a pair: to a successor state where request is false, or true, respectively. For example, the state '¬req, busy' has four states it can move to (itself and three others).

LTL specifications are introduced by the keyword LTLSPEC and are simply LTL formulas. Notice that SMV uses &, |, -> and ! for $\wedge$, $\vee$, $\rightarrow$ and $\neg$, respectively, since they are available on standard keyboards. We may easily verify that the specification of our module main holds of the mode.