Modeling and Verification of an Air Traffic Concept of Operations

Gilles Dowek

Joint work with César Muñoz (NIA) and Víctor Carreño (NASA)

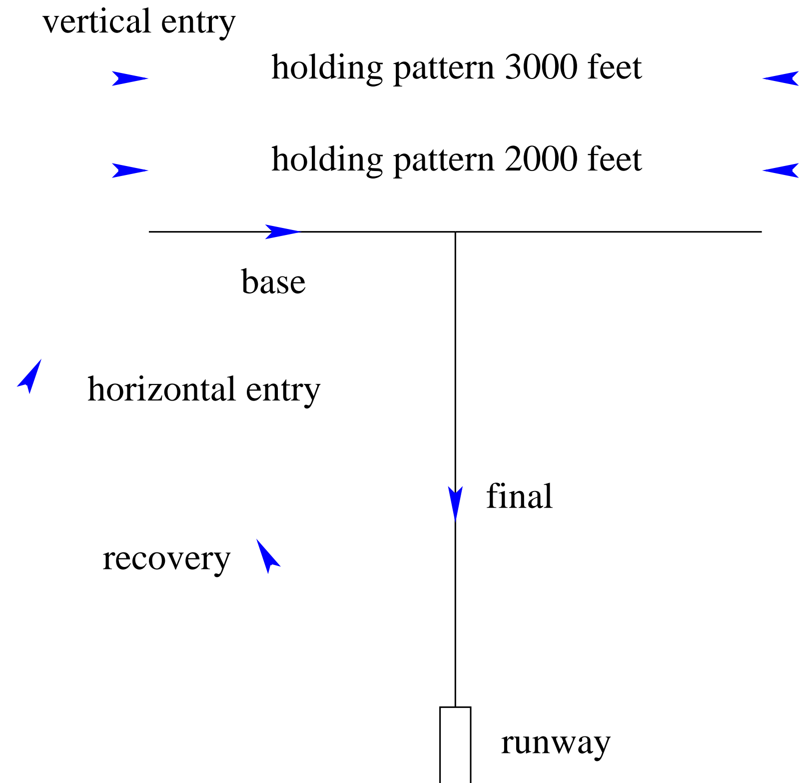
Paper presented at ISSTA'04

Small airports

An example what we can do with a proof-checker (Coq, PVS, ...)
for establishing safety (and security) of systems

An experimental concept of operations for small airports

An experimental concept of operations for small airports



Lead aircraft and fixes

First come, first served: lead aircraft

Missed approach fix: Right or Left (opposite to its lead aircraft)

Examples of rules

An aircraft can enter horizontally on the right if no aircraft in a Right zone or in a Left or Central zone with a Right missed approach fix

Examples of rules

An aircraft can enter horizontally on the right if no aircraft in a Right zone or in a Left or Central zone with a Right missed approach fix

An aicraft can enter vertically on the right if

- no aircraft is in the Right holding pattern at 3000 feet
- no aircraft is in the Right missed approach zone
- no aircraft is currently in the Right horizontal approach zone
- at most one aircraft in a Right zone or in a Left or Central zone with a Right missed approach fix

Questions

- How many aircraft can be on the airspace of the airport at the same time ?
- How many aicraft can be in the same zone at the same time ?
- Does an aircraft in the missed approach zone always have a place to go ?
- Can all aircraft land ? (no deadlocks)

How can we solve these problems ?

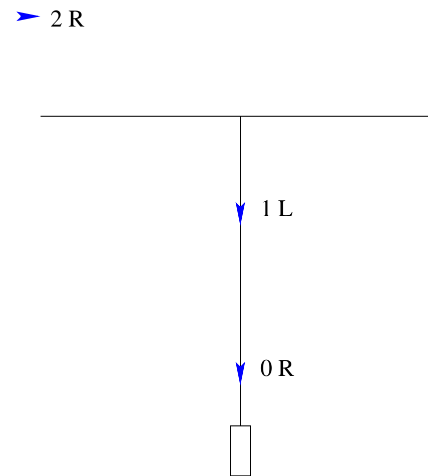
Try and observe (but many crashes if we do so...)

Modelize by a program, simulate and observe (but no certainty)

Modelize and prove the property on the model

A modelization

A state describes a possible configuration of the airspace




```
{hp3r = [(2,R)], ...  , tr = [], ..., final =
[(0,R);(1,L)]}
```

A modelization

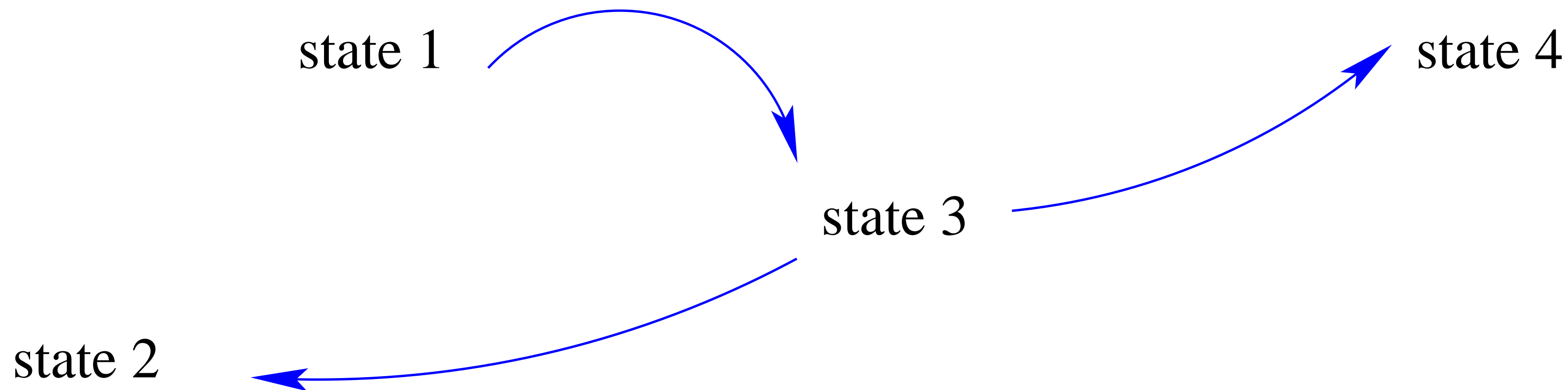
\ rule is an algorithm mapping a state to a list of states

```
.et rule6r s = match s#tr with
  [] -> []
  h::r -> let x = order h
          in if x = 0 || mem (x-1) s#final
             then [{<tr = r; final = final@[h]>}]
             else []
```

\ dozen of rules like this one (two pages) ... more precise than a
50 page long document

A graph

Nodes are states edges are transitions



A priori, the number of reachable states may be infinite

Although it happens to be finite

Two ways to prove a property

n all reachable states there is at most one aircraft in the 3000 feet
holding pattern

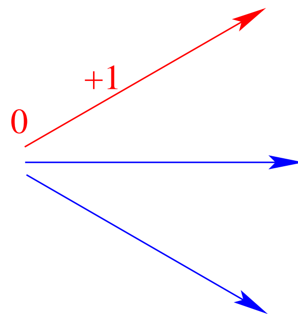
General proof

Consider a state reachable state from the empty state

$$s_0 \longrightarrow s_1 \longrightarrow \ldots \longrightarrow s_n$$

prove by induction on n that s_n has at most one aircraft in hp

Assume it for s_n and prove it for s_{n+1}



General proof

Not a difficult proof

But many opportunities for a mistake

The transition rules always evolve (moving target)

Which properties are kept, which ones are discarded ?

Such a proof must be checked with a proof-checker (PVS, Coq, ...)

Taking advantage of finiteness

To prove

$$\forall x \ (x \in \{2, 3, 7\} \Rightarrow prime(x))$$

prove $prime(x)$ using the hypothesis $x \in \{2, 3, 7\}$

Prove

$$prime(2) \wedge prime(3) \wedge prime(7)$$

Finite universal quantifiers are conjunctions

Taking advantage of finiteness

n all reachable states there is at most one aircraft in the 3000 feet
holding pattern

Check them all

What tool can we use to check all the states ?

An explicit model checker (SPIN, ...)

A symbolic model checker (SMV, SAL, ...)

A programming language (Caml, Java, ...)

A proof checker (PVS, Coq, ...)

Some particularities of our example

- States have complex descriptions (record of lists of pairs formed with an integer and a member of an enumerated type)
- Infinite datatype for states (although the number of reachable states is finite)
- Rules are "complex" algorithms (for condition, count number of aircraft having such and such property, ...)
- The properties we want to check are complex (count number of aircraft, predicates inductively defined on lists, ...)
- Symetries to be exploited (Right/Left)

Explicit model checker

Poor data structures for states: int, bool, lists of integers, lists of booleans (lists of pairs must be flattened as pairs of lists, ...)

Rule language is a subset of C without procedures and functions (no parametric variables, no way to exploit symmetry)

The language to express predicates is temporal logic: difficult to express predicates inductively defined on lists

Seems possible but cumbersome

Symbolic model checker

'he number of potential states must be finite (limited to a couple
of hundreds boolean variables)

Even if we restrict the numbers to be smaller than 4 or 5, we are
t the limit

Programming language

Rich datatypes, rich language for algorithms and for predicates

Weak points:

- need to program our own enumeration algorithm (*e.g.* Tarjan's dfs algorithm)
- possibility of bugs in the program

Proof checkers

Rich datatypes, rich language for functions (a programming
anguage), rich langage for predicates

Program Tarjan's algorithm in the proof system

Prove that it is correct

$$Tarjan(T,P) \Rightarrow (\forall s\ (Reachable(s_0,T,s) \Rightarrow P(s)))$$

f $Tarjan(T,P)$ computes to $True$, we can deduce

$$\forall s\ (Reachable(s_0,T,s) \Rightarrow P(s))$$

Conclusion

Ten — accepted — recommandations (including one bug fix)

Computational power of programming langage

Expressiveness of a logical langage

The complexity of the pb is not only in the number of states but also in the form of rules and properties