Proving invariants: how many times a program should be unfolded ?

Paul Caspi, Jan Mikáč

VERIMAG – Grenoble

Problem definition 1

A closed system

- state space S
- initial state I is a predicate on S
- transition relation T is a predicate on $S \times S$
- sequence of states $X_0, X_1, X_2, \ldots X_n, \ldots$
  such that $I(X_0)$ and $T(X_n, X_{n+1})$ hold

Properties

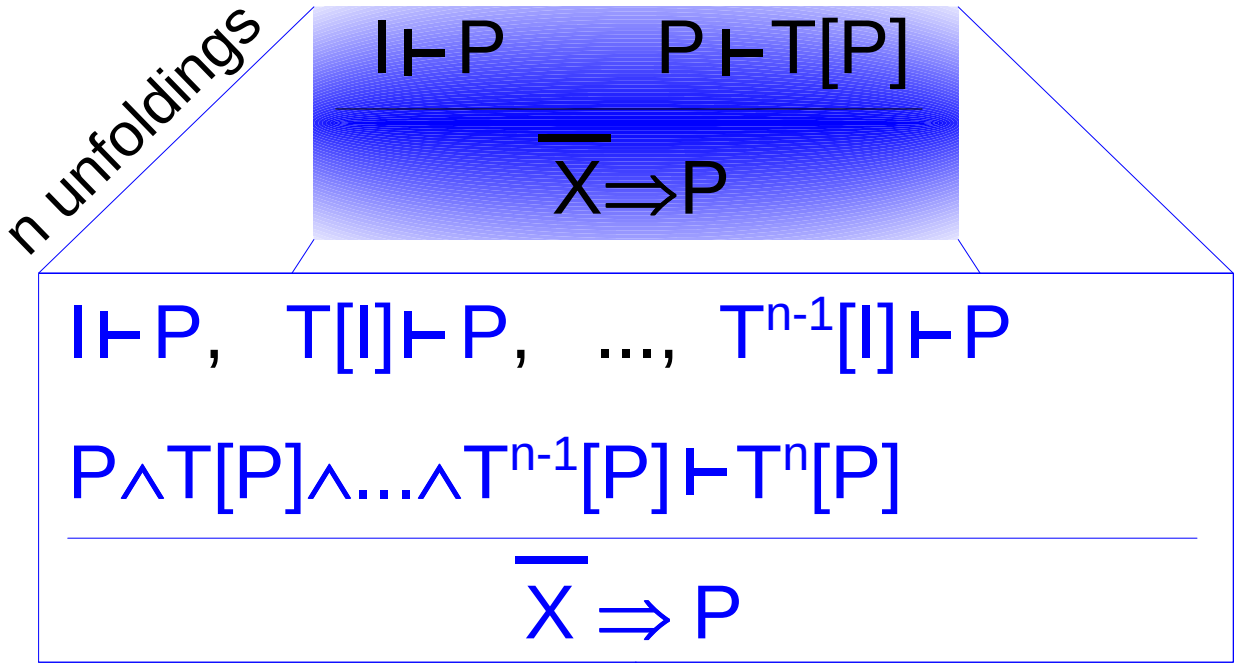
- strongest invariant of the system $\overline{X} = I \cup T[\overline{X}]$ (fix)
- an invariant property P: $\overline{X} \Rightarrow P$?

Induction

$$\frac{I \vdash P \qquad P \vdash T[P]}{\overline{X} \Rightarrow P}$$

Problem definition 2

Induction



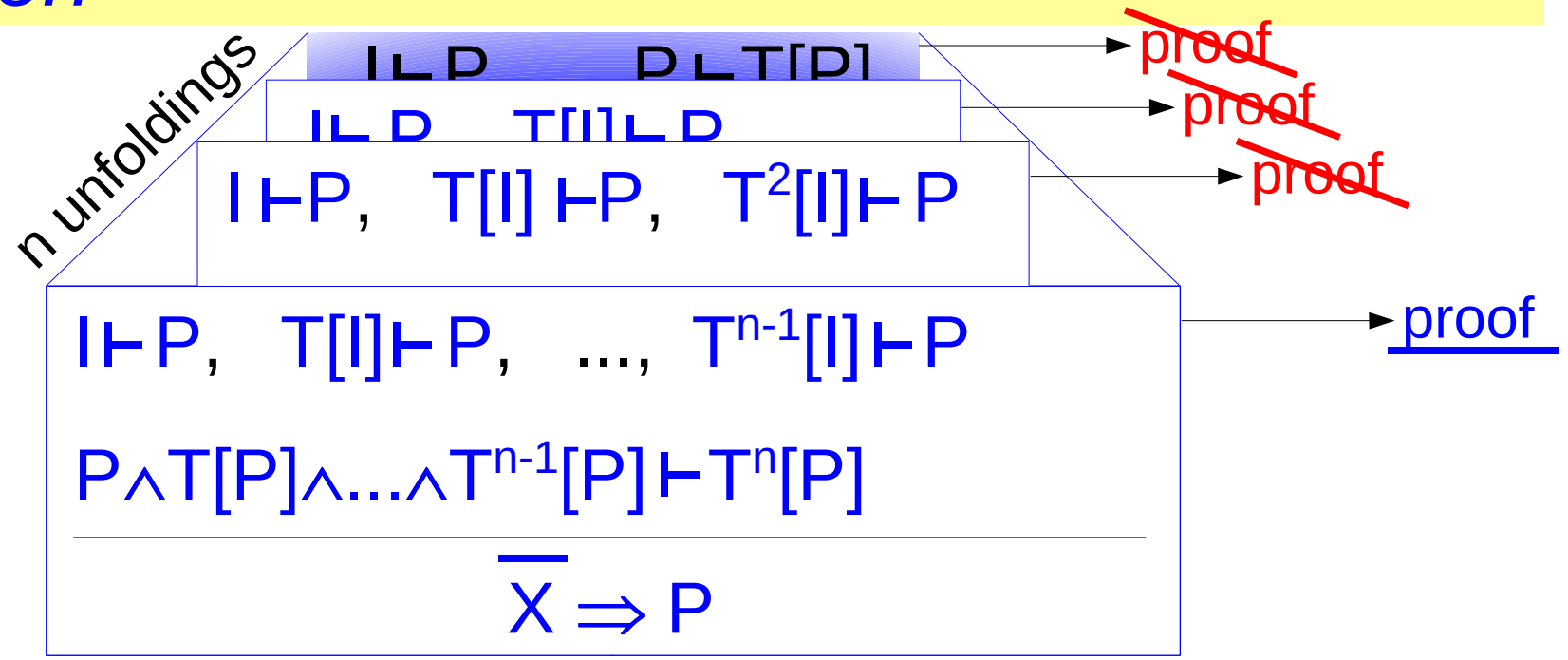
What is a ``good'' value for n ?

``Good'' means
- easy to find (by a machine)
- likely to work

→ number of variables of the system

Problem definition 2

Induction



What is a ``good'' value for n ?

``Good'' means
- easy to find (by a machine)
- likely to work

→ number of variables of the system

Small example 1

f = 1.(f + g)
g = 0.(f – g)
prove *All* f+g > 0

1 1 2 2 4 ...
0 1 0 2 0 ...

n = 1

f+g > 0

1.(f+g)+0.(f-g) > 0

impossible to prove

g = 1.(f + g) 1 1 0 -2 -4 ...
f = 0.(f – g) 0 -1 -2 -2 0 ...

---

f =-1.(f + g) -1 1 -2 2 -4 ...
g = 2.(f – g) 2 -3 4 -6 8 ...

n = 2

f+g > 0
1.(2f) > 0

1.2.(2f + 2g)>0

provable

Small examples 2

fibo = 1.(fibo + g)
g = 0.fibo
prove *All* fibo > 0

1 1 2 3 5 8 ...
0 1 1 2 3 5 ...

n = 2

fibo > 0
1.(fibo + g) > 0

1.(1.(fibo+g) + 0.fibo)>0

provable

fibo = 1.(fibo + 0.fibo)
prove *All* fibo > 0

n = 1

fibo > 0

1.(fibo + 0.fibo) > 0

provable

Small example 3

f = 1.(f + g)
g = 0.(f + h)
h = 1.g
prove *All* f > 0

1 1 3 4 9 14 ...
0 2 1 5 5 14 ...
1 0 2 1 5 5 ...

n = ? f > 0
1.(f + g) > 0
1.1.(2f+g+h) > 0
1.1.3.(3f+3g+h) > 0
1.1.3.4.(6f+4g+3h) > 0
1.1.3.4.9.(10f+9g+4h) > 0
... ...

Example conclusion

What is a ``good'' value for n ?

``Good'' is
- easy to find (by a machine)
- likely to work
- number of variables of the system

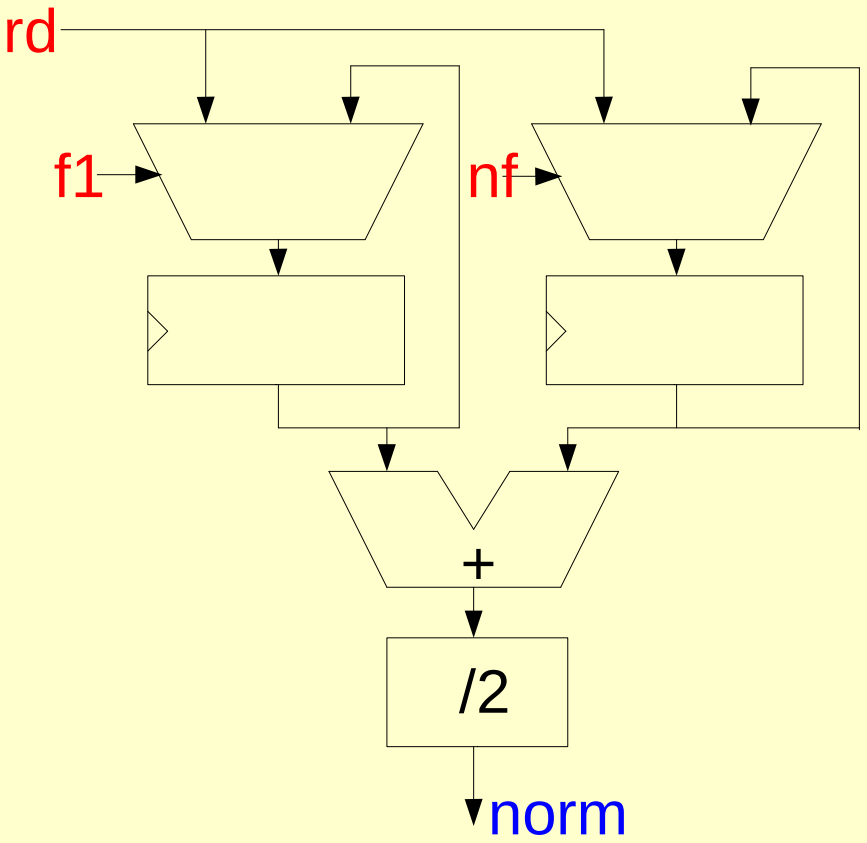
- In fact we want : dimension of the system.
- Number of variables mostly ≥ dimension
- Number of variables is easy to find

- Unfold < n times : can work (property holds on a whole class)
- Unfold n times : works because all information is used
- Unfold n times : if it doesn't work, we have no other limit

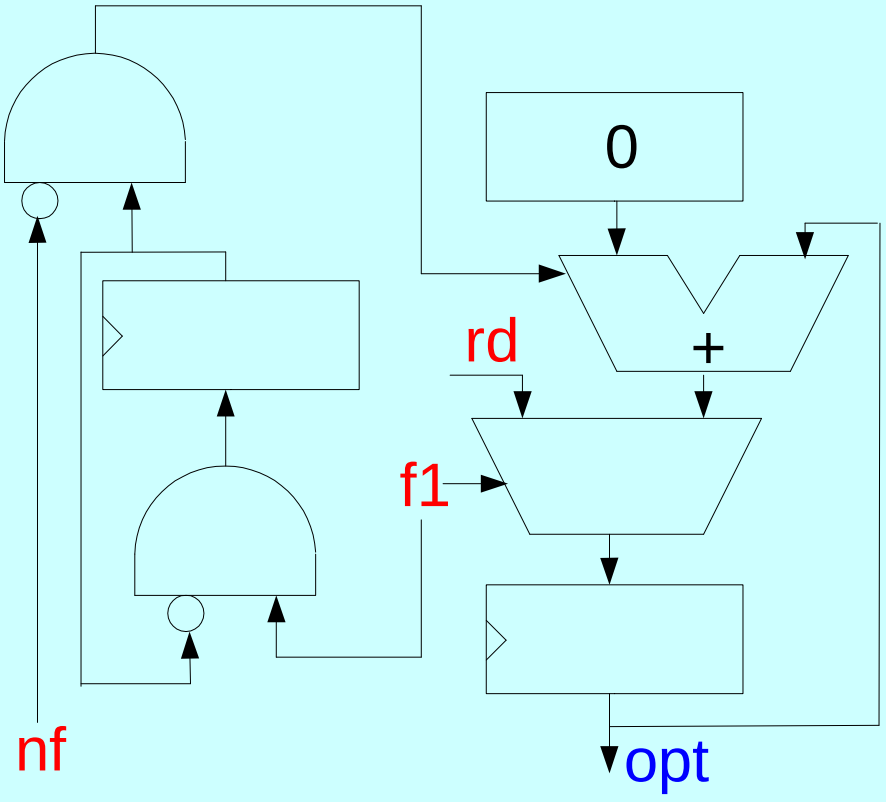
Example of synchronizer

Paul Miner & Steven Johnson
Verification of an Optimized Fault-Tolerant Clock Synchronization Circuit

Normal circuit



Optimized circuit



Example of synchronizer

Paul Miner & Steven Johnson
Verification of an Optimized Fault-Tolerant Clock Synchronization Circuit

```
 rd=0 -> if R then 0 else pre rd+1 ;
nor=(t1 + tn) div 2 ;
 t1=0->if R then 0 else (if f1 then pre t1 else rd);
 tn=0->if R then 0 else (if nf then pre tn else rd);
 h =false -> (not R) and f1 and (not pre h) ;
cin=false -> (pre h) and not nf ;
opt=0 -> if R then 0 else (if f1 then
          (if cin then 1 else 0) + pre opt else rd);
```

```
assert R -> true ;
assert nf => f1 ;
assert not f1 -> true ;
assert true -> R or (pre nf => nf) ;
assert R => (not f1) ;
```

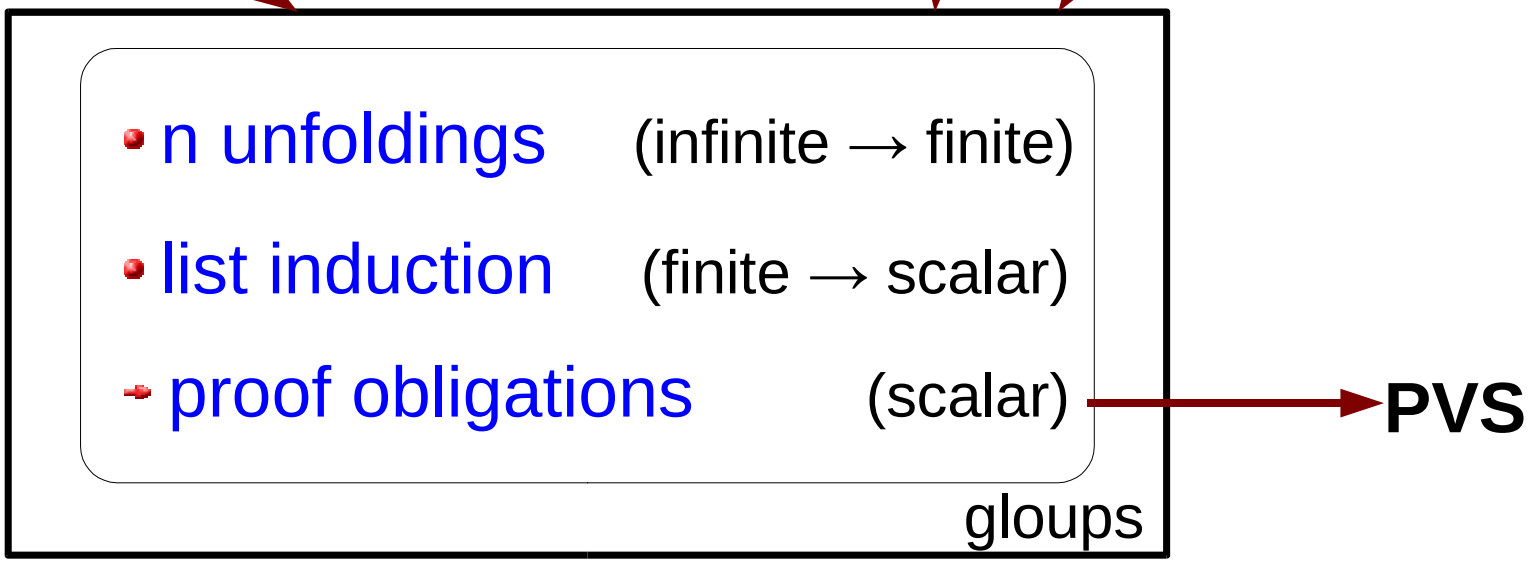
Example of synchronizer

Paul Miner & Steven Johnson
Verification of an Optimized Fault-Tolerant Clock Synchronization Circuit

```
  rd=0 -> if R then 0 else pre rd+1 ;
nor=(t1 + tn) div 2 ;
 t1=0->if R then 0 else (if f1 then pre t1 else rd);
 tn=0->if R then 0 else (if nf then pre tn else rd);
 h =false -> (not R) and f1 and (not pre h) ;
cin=false -> (pre h) and not nf ;
opt=0 -> if R then 0 else (if f1 then
          (if cin then 1 else 0) + pre opt else rd);
```

```
assert R -> true ;
assert nf => f1 ;
assert not f1 -> true ;
assert true -> R or (pre nf => nf) ;
assert R => (not f1) ;
```

nor = opt



Conclusion

Proving invariants:
how many times a program should be unfolded ?

$$\frac{I \vdash P,\quad T[I] \vdash P,\quad \ldots,\quad T^{n-1}[I] \vdash P \qquad P \wedge T[P] \wedge \ldots \wedge T^{n-1}[P] \vdash T^{n}[P]}{\overline{X} \Rightarrow P}$$

What is a ``good'' value for n ?

``Good'' means
- easy to find (by a machine)
- likely to work

→ number of variables of the system

Perspectives

- Prove Time-Trigered Protocole Membership Algorithm (Next-TTA)
  - Difficult problem
  - Space requirements
- Modular proofs

- Explore link refinement-unfolding
  - Proving refinements (vertical modularity)
  - Evolution of the number of unfoldings during the refinement

Questions ?