Falsifying safety properties

through games on over-approximating models

Nathaniel Charlton and Michael Huth
Imperial College London

Workshop on Reachability Problems 2008

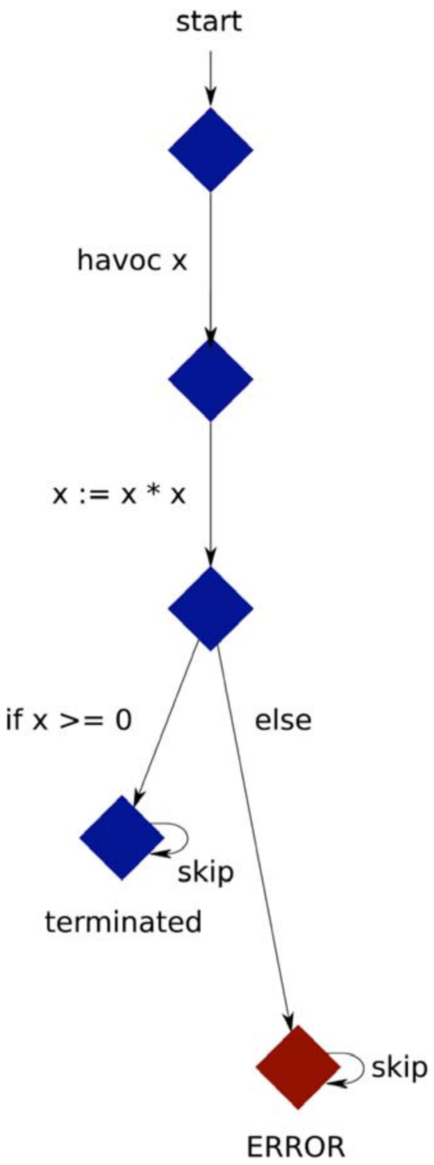
Background: verifying assertions

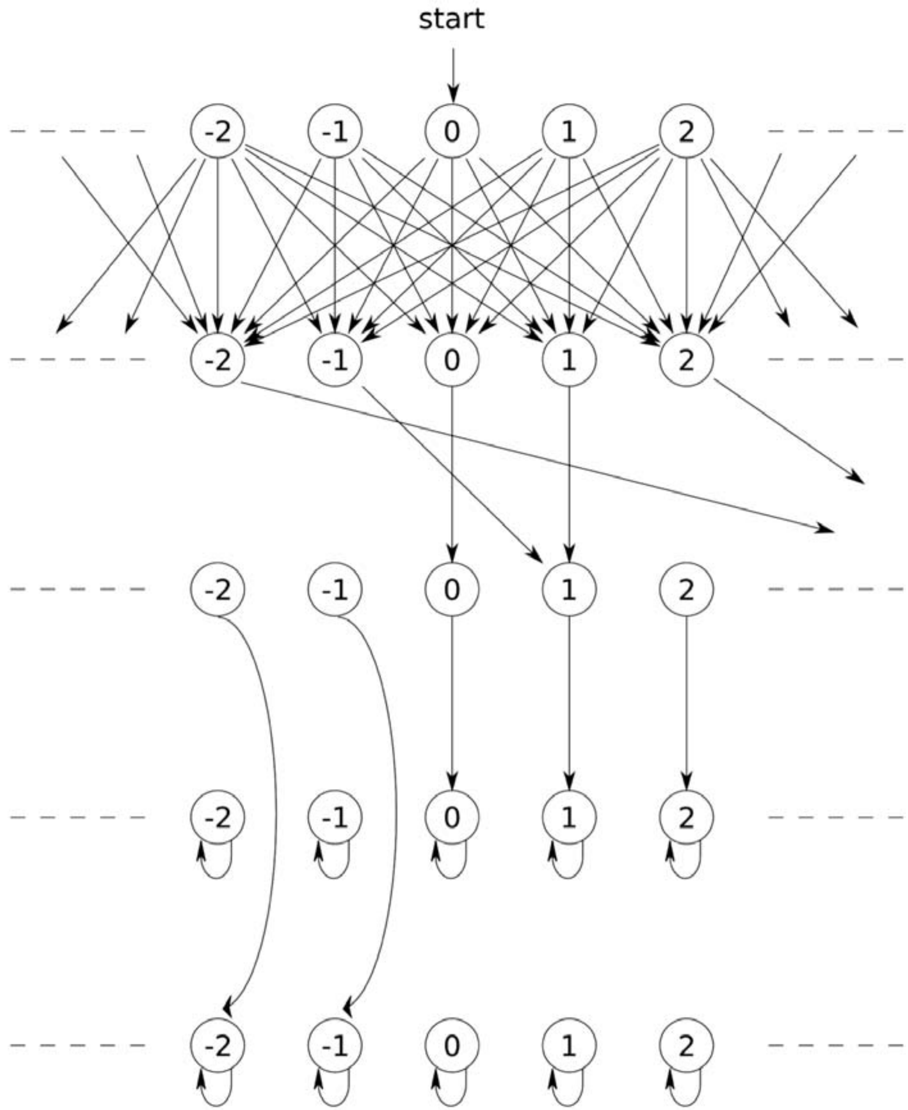
- Treat the program as (infinite-state) **transition system**
  - states are pairs **(N, s)**: N is program counter, s is memory

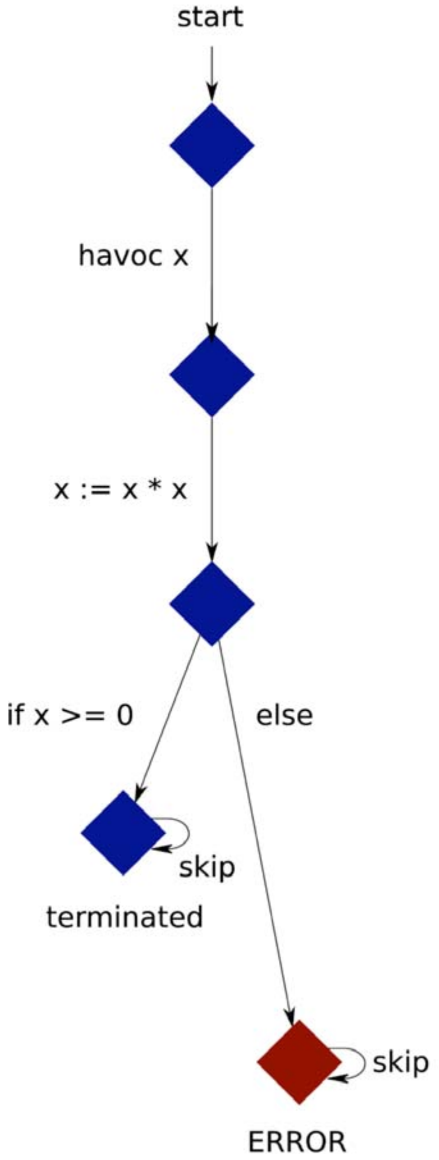
- Model assertion violations as transitions to special error location
  - **Error location unreachable IFF Program correct**

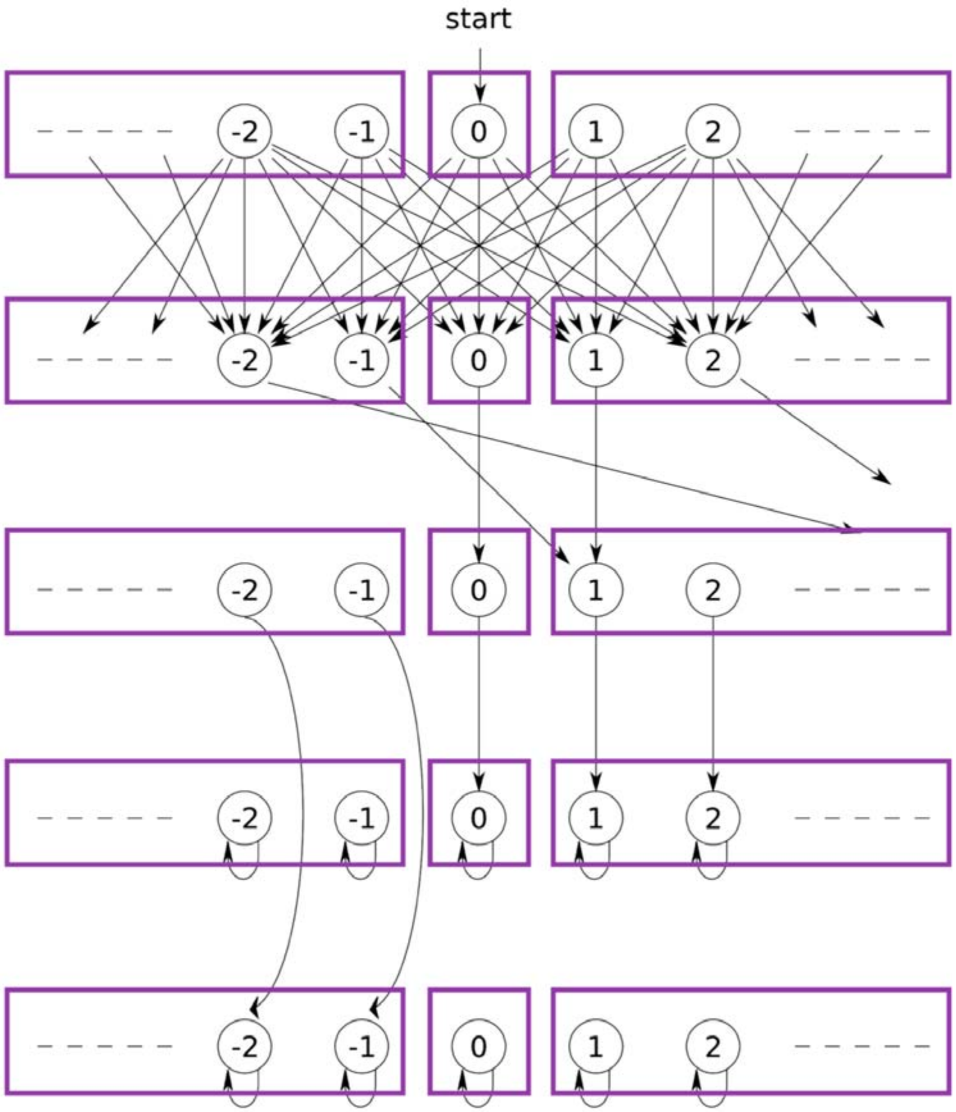
- To overcome infiniteness of state space, **use abstraction**
  - Build **finite** trans. sys. that over-approximates concrete one
  - states are **(N, a)**: N is prog. counter, **a abstracts memory**

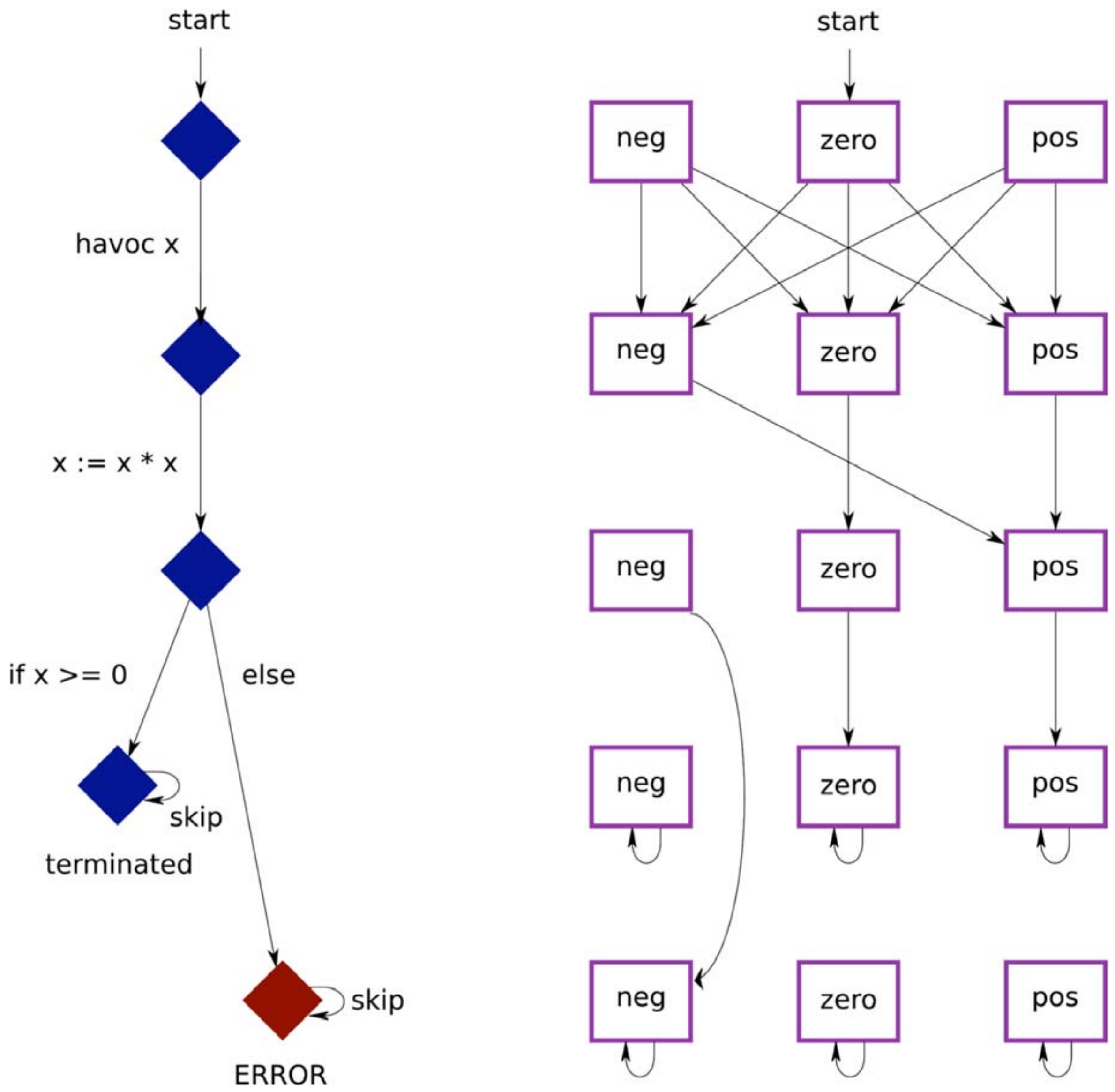
- **Abstract system has more paths**
  - Locations unreachable in abstract system are also unreachable in concrete program
  - **Error location unreachable IMPLIES Program correct**


start
havoc x
x := x * x
if x >= 0
else
skip
terminated
skip
ERROR


start
-2
-1
0
1
2

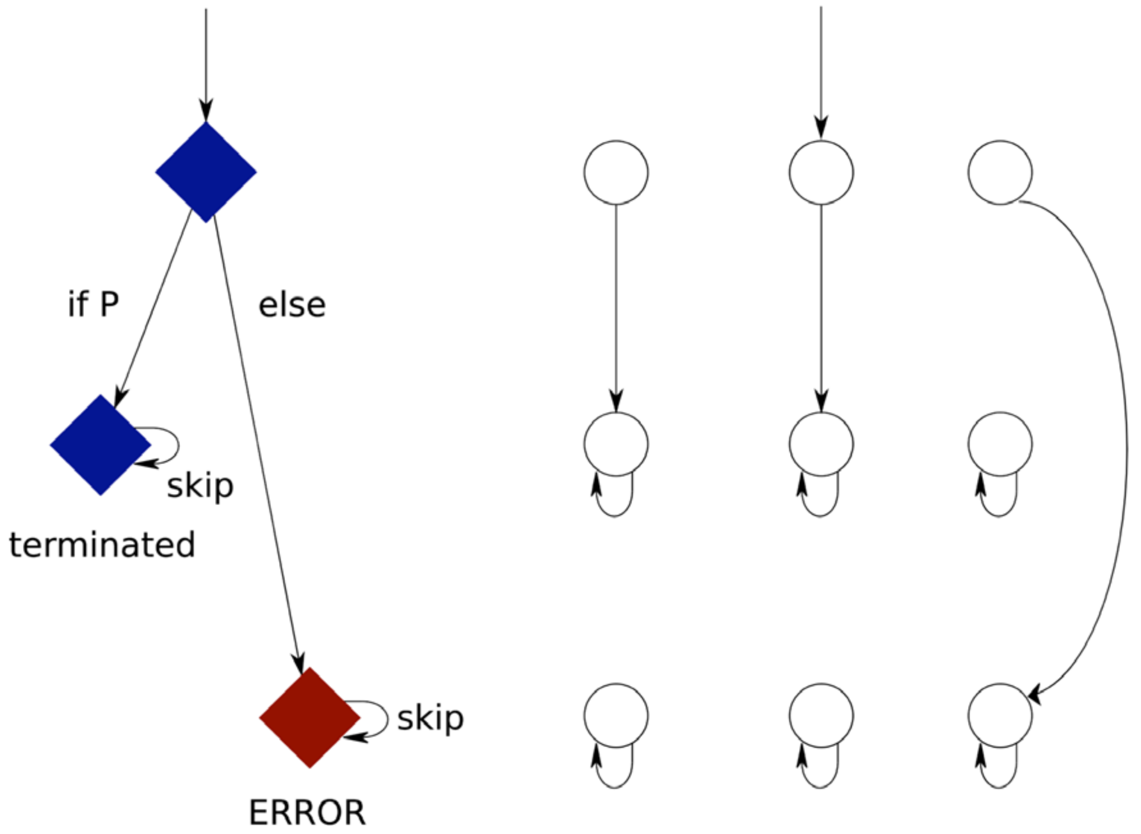

start
havoc x
x := x * x
if x >= 0
else
skip
terminated
skip
ERROR

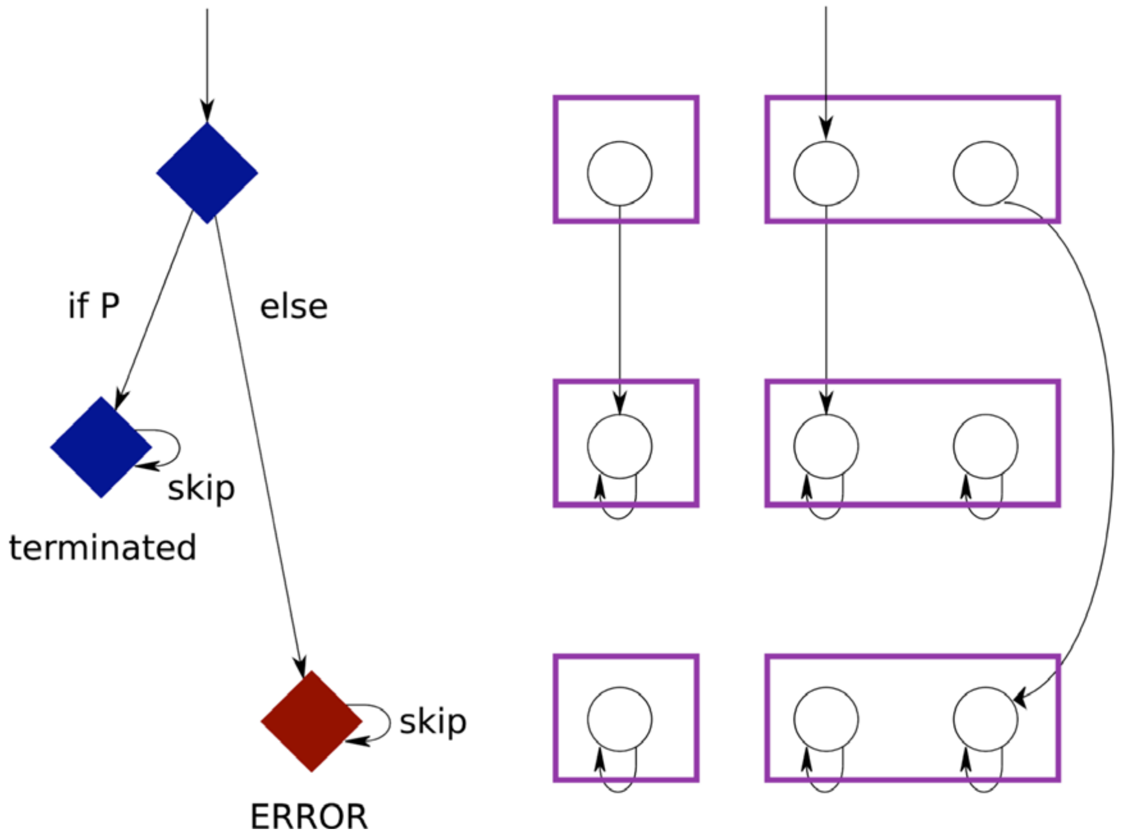

start
-2
-1
0
1
2

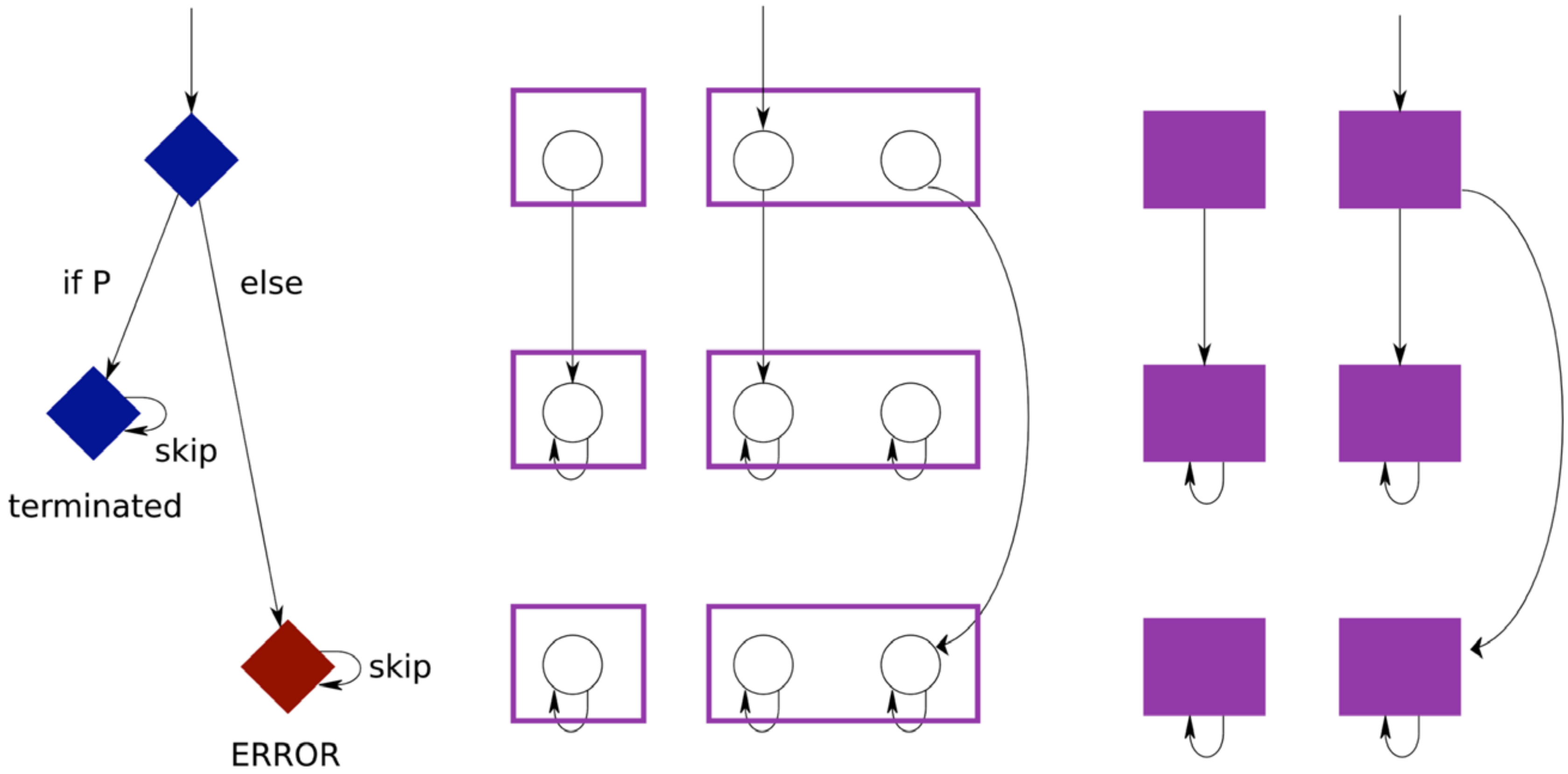

start
havoc x
x := x * x
if x >= 0
else
skip
terminated
skip
ERROR
start
neg
zero
pos
neg
zero
pos
neg
zero
pos
neg
zero
pos
neg
zero
pos

What about falsification?

- What if the **error location is reachable** in the abstract system?
  - **cannot** conclude that the program is incorrect
  - path to the error location may be **“artifact of abstraction”**


if P
else
skip
terminated
skip
ERROR


if P
else
skip
terminated
skip
ERROR


if P
else
skip
terminated
skip
ERROR

This talk: falsification via games

- We present a falsification method based on 2-player **GAMES**
  - reasonably simple, and independent of abstraction used
- Exploit two properties of programs:
  - concrete semantics are **serial**
  - most language constructs are **deterministic**
- Method of falsifying via games is essentially **FREE**:
  - no changes to abstract model construction process
  - falsification check runs in time linear in model size
- **Therefore, worth trying even if frequency of success is low**

Programs are serial

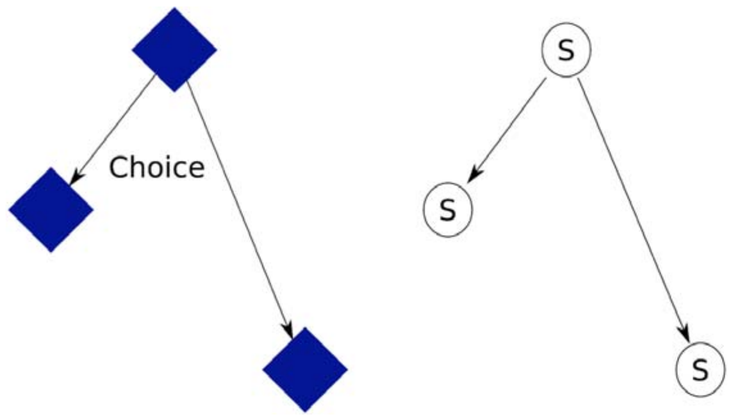
- Transition systems generated by programs are **serial**:
  - **every node has a successor**
  - in programs, execution doesn't just stop for no reason

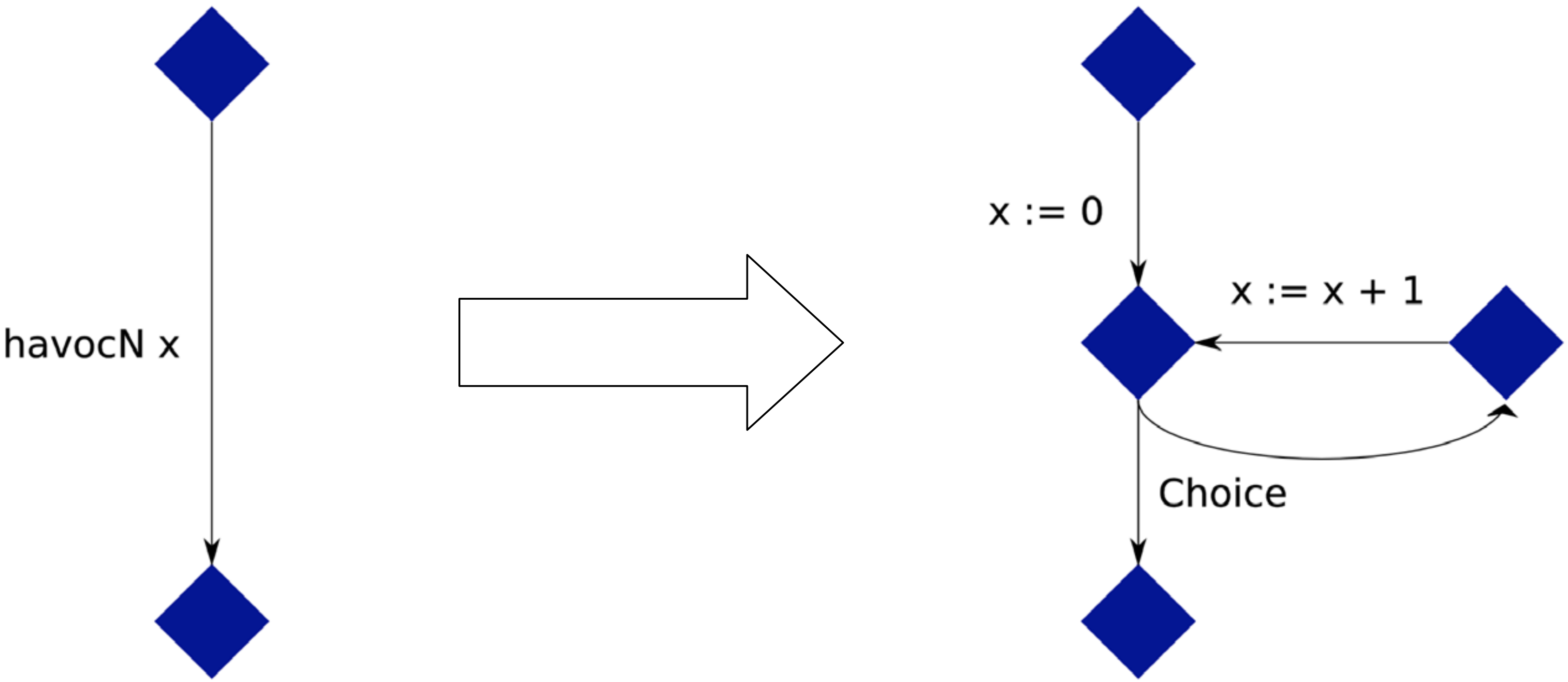
- **unlike** transition systems in general
  - such as transition systems specified using process algebra

Confinement of non-determinism

- In many programming languages, **nearly all statements have deterministic semantics**
  - Non-determinism confined to a **small number of identifiable points**
- In fact, treat non-deterministic statements as **syntactic sugar**
  - all non-determinism comes from **Choice** hyperedge

All states have two successors at a Choice statement




havocN x
x := 0
x := x + 1
Choice

Falsification (1)

- Main idea: an abstract state is **hopeless** if, once execution reaches that state, it **inevitably flows to the error location**
  - program is incorrect if starting state is hopeless
- We give **deduction rules** for identifying hopeless states.
- Label (with function ρ) each abstract state as **either F or P**
  - Intuition is that F and P represent two players
  - **F** wants to **F**alsify the program, **P** wants to **P**revent this
- (In this talk) F nodes are those located at a Choice statement

Falsification (2)

Deduction rules for hopelessness:

h-already-there

$$\frac{l = error}{\mathbf{H}(l, a)}$$

h-P-move

$$\frac{\rho(n) = P \qquad \text{for all } n',\ \text{if } n \xrightarrow{abs} n' \text{ then } \ \mathbf{H}n'}{\mathbf{H}n}$$

h-F-move

$$\frac{\rho(n) = F \qquad \text{exists } n' \text{ such that } n \xrightarrow{abs} n' \text{ and } \ \mathbf{H}n'}{\mathbf{H}n}$$

Falsification (2)

Deduction rules for hopelessness:

h-already-there

$$\frac{l = error}{\mathbf{H}(l, a)}$$

h-P-move

Justified by seriality

$$\frac{\rho(n) = P \qquad \text{for all } n',\ \text{if } n \xrightarrow{abs} n' \text{ then } \mathbf{H}n'}{\mathbf{H}n}$$

h-F-move

$$\frac{\rho(n) = F \qquad \text{exists } n' \text{ such that } n \xrightarrow{abs} n' \text{ and } \mathbf{H}n'}{\mathbf{H}n}$$

Falsification (2)

Deduction rules for hopelessness:

h-already-there

$$\frac{l = error}{\mathbf{H}(l,a)}$$

h-P-move

Justified by seriality

$$\frac{\rho(n) = P \qquad \text{for all } n',\ \text{if } n \xrightarrow{abs} n' \text{ then } \mathbf{H}n'}{\mathbf{H}n}$$

h-F-move

Justified by semantics of Choice edges

$$\frac{\rho(n) = F \qquad \text{exists } n' \text{ such that } n \xrightarrow{abs} n' \text{ and } \mathbf{H}n'}{\mathbf{H}n}$$

Algorithm for Falsification

1. **Build abstract model as ususal, attempting to verify prog.**

2. **If model doesn't verify program, try falsification check:**

3. **Repeatedly apply deduction rules for hopelessness**
   - eventually no more applications will be possible
   - this computes a set of hopeless nodes
   - same as computing winning region in an attractor game between players F and P

4. **Check whether starting state is in computed set of nodes**

- Takes only **linear time O(n+e)**
  - where n and e are numbers of nodes and edges
  - can be programmed to visit each edge only once

Example of falsification

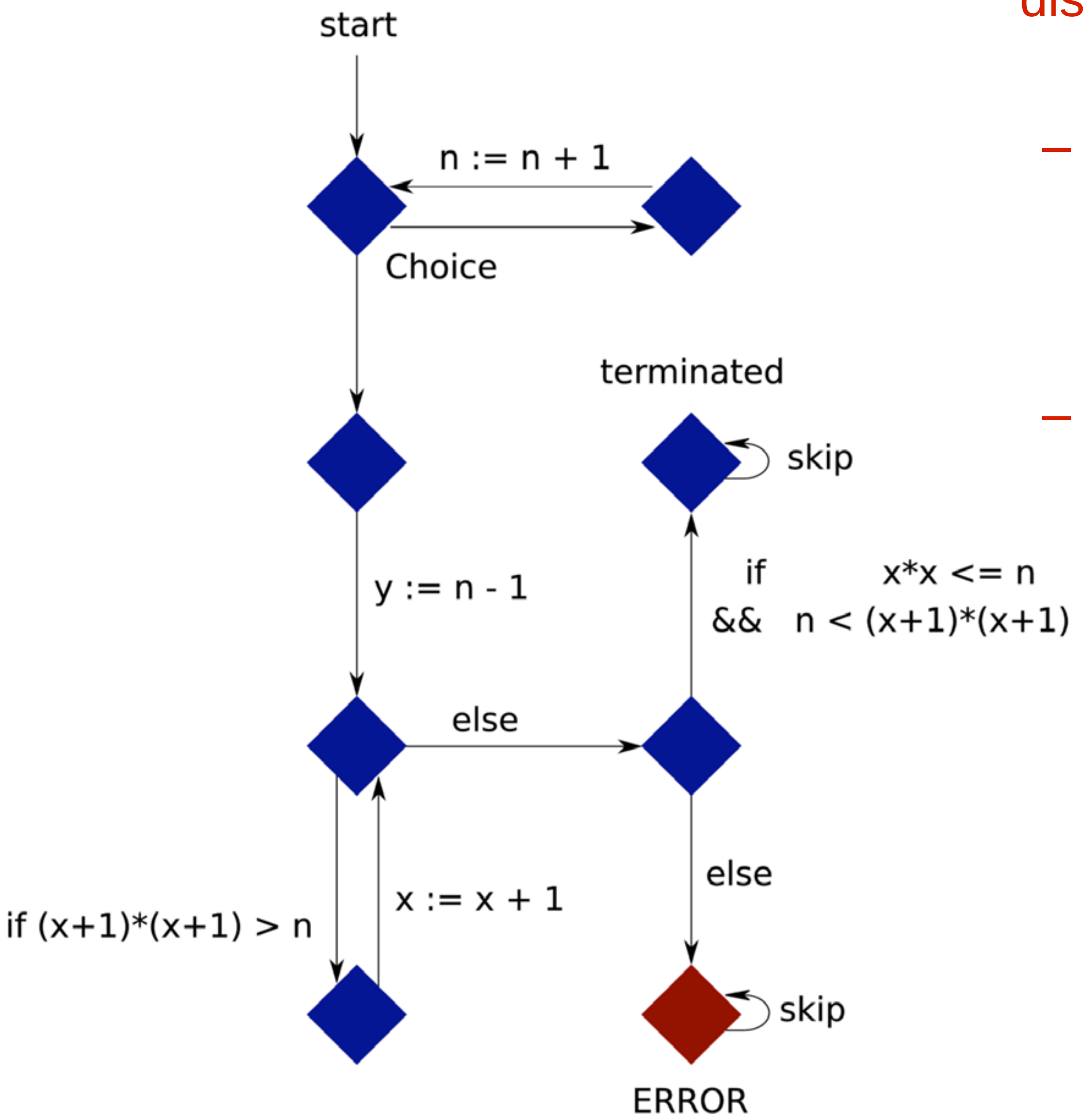
Consider this program:

```
havocN n;        // choose an arbitrary natural number

y := n – 1;

while ( (x+1)*(x+1) < n )     // Find integer square root of n
   { x := x + 1 }

assert ( x*x <= n  &&  (x+1)*(x+1) < n );
```

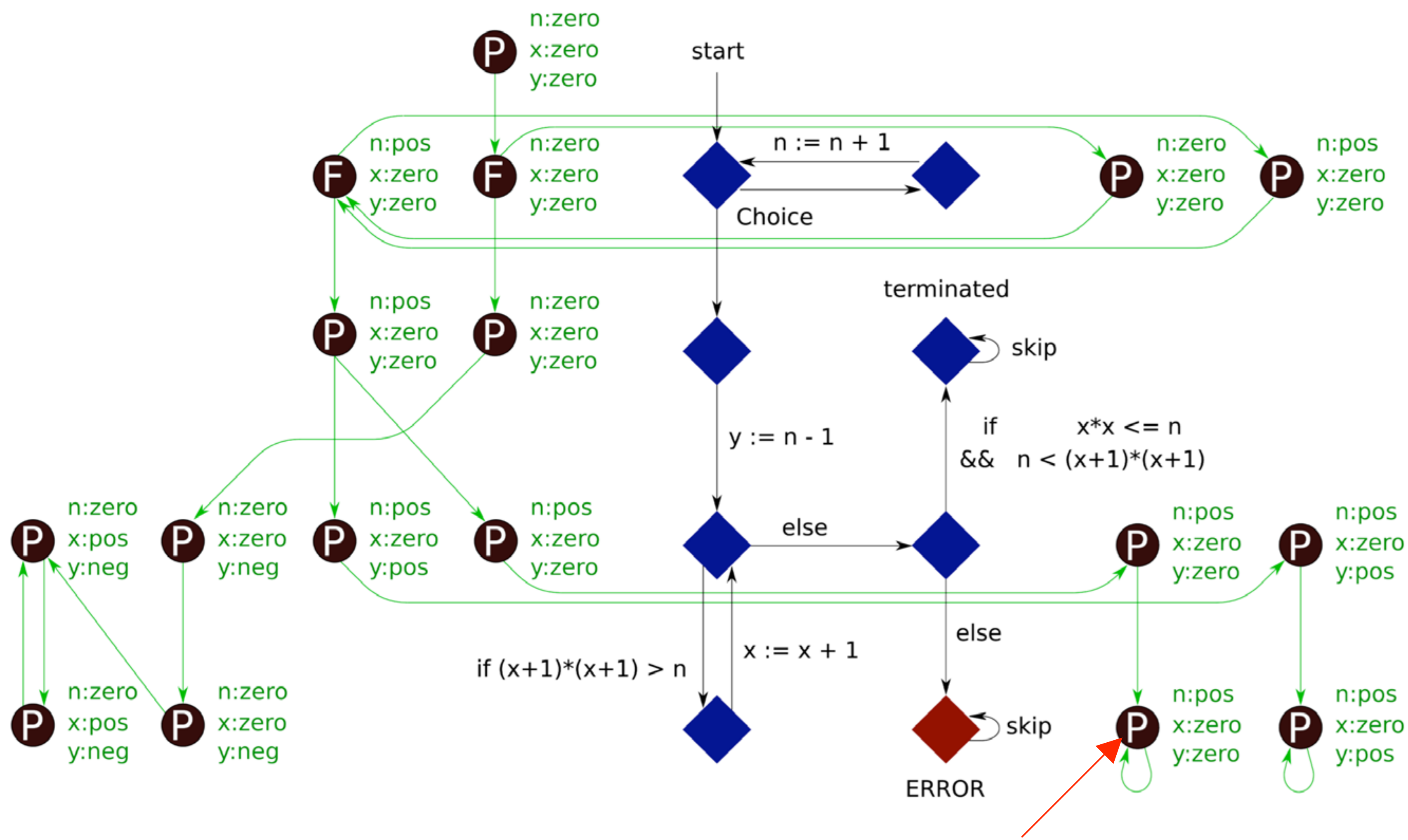
Example of falsification

- Now introduce a **“mistake”:** reverse inequality in loop guard.
  - What happens when we try to verify the broken program?

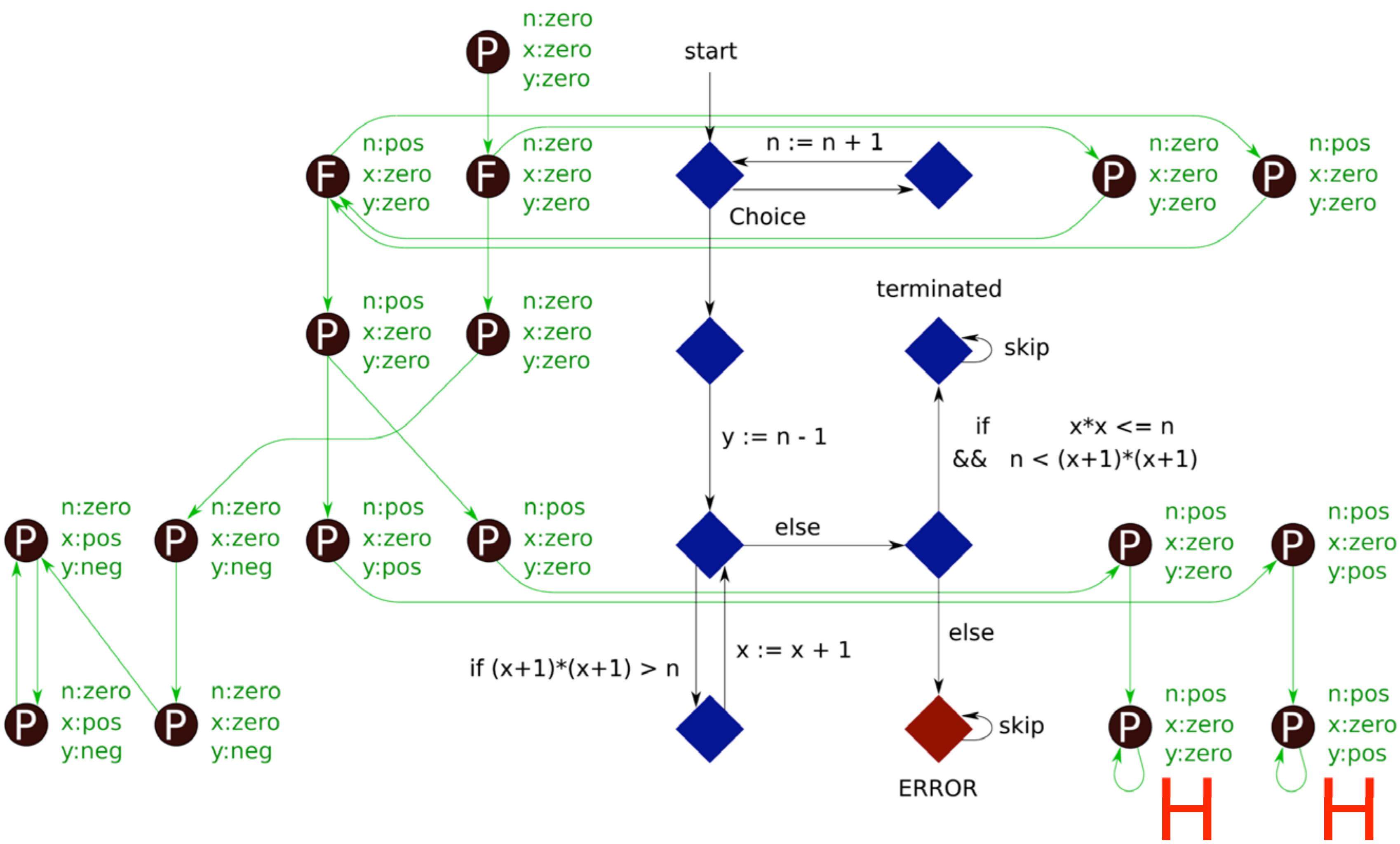
```
havocN n;        // choose an arbitrary natural number

y := n – 1;

while ( (x+1)*(x+1) > n )     // Find integer square root of n
   { x := x + 1 }

assert ( x*x <= n  &&  (x+1)*(x+1) < n );
```



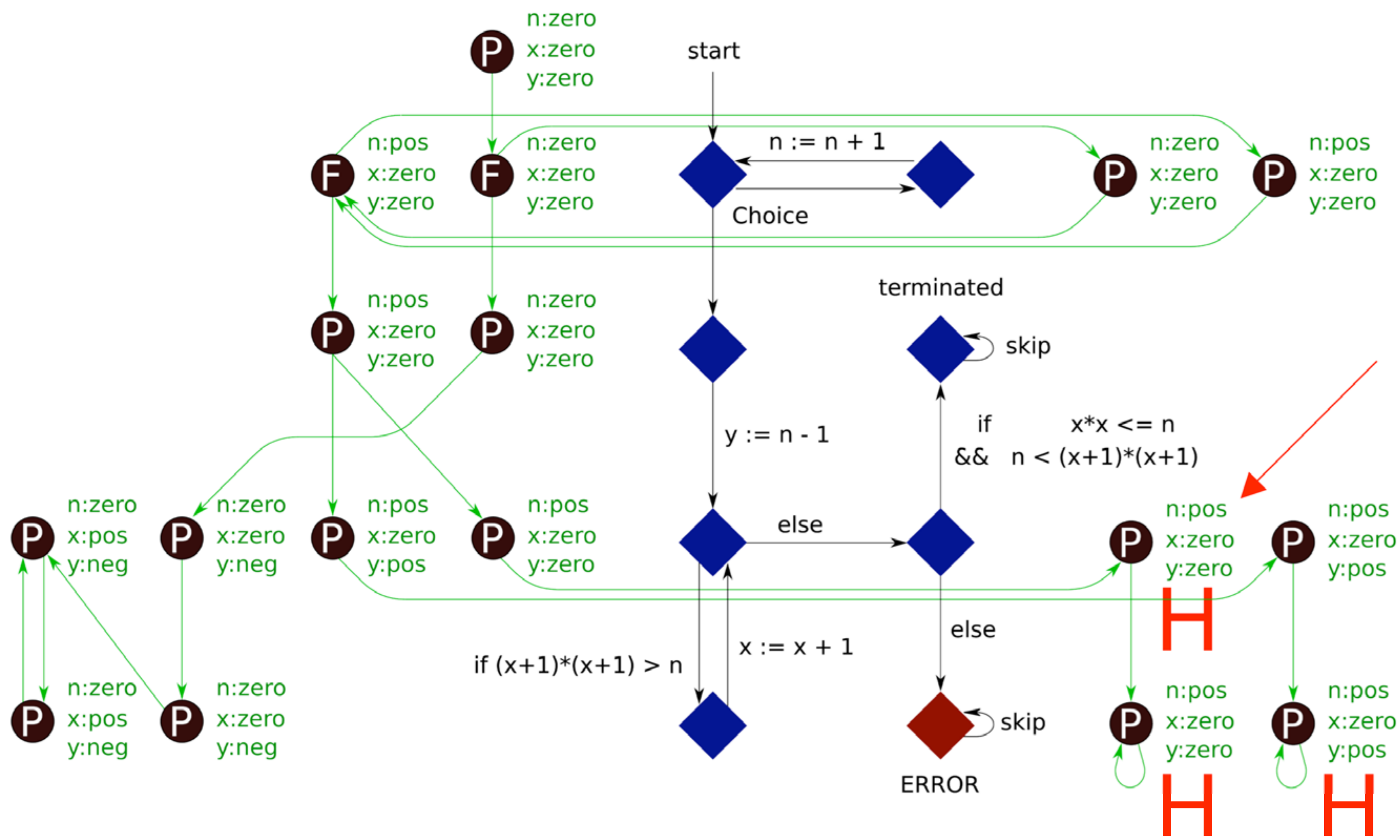
- **Abstract-check-refine** discipline says:
  - start with a simple abstraction, then refine if necessary
  - so we'll start with just sign analysis



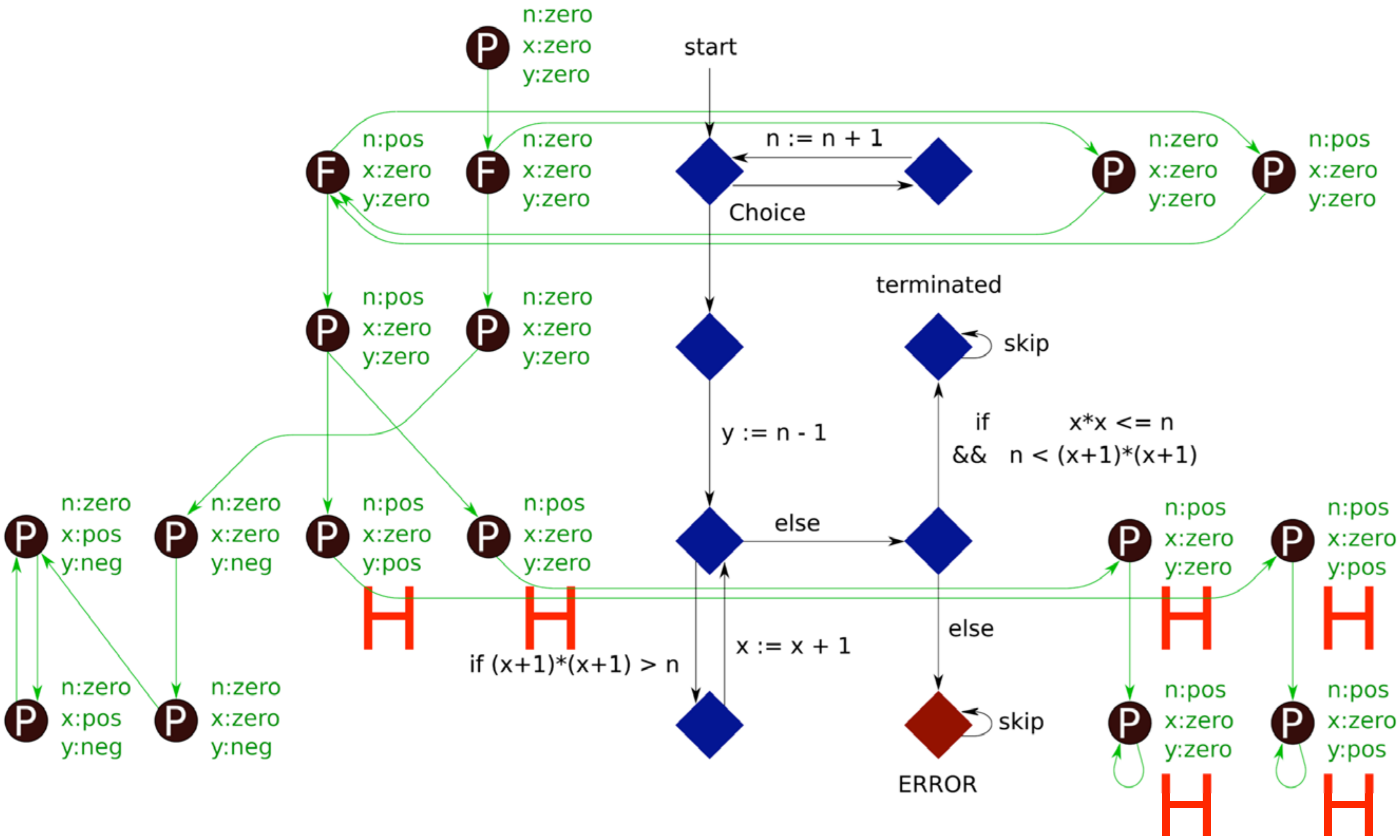
- Program isn't verified because there's a path to the error location
  - But is it a real error?
  - Let's run our falsification check…

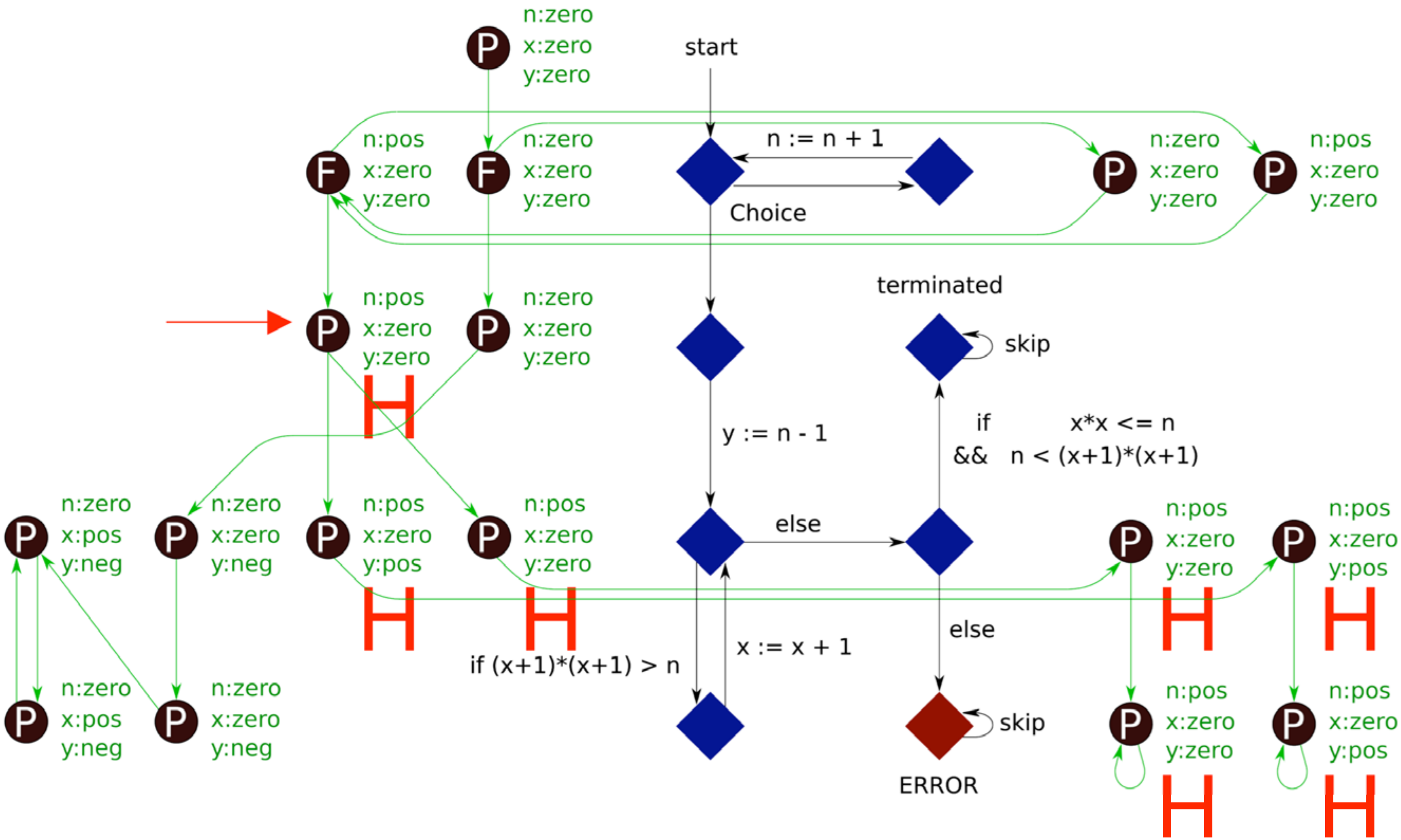


- Use the **h-already-there** rule
  - mark states at the error location as **Hopeless**

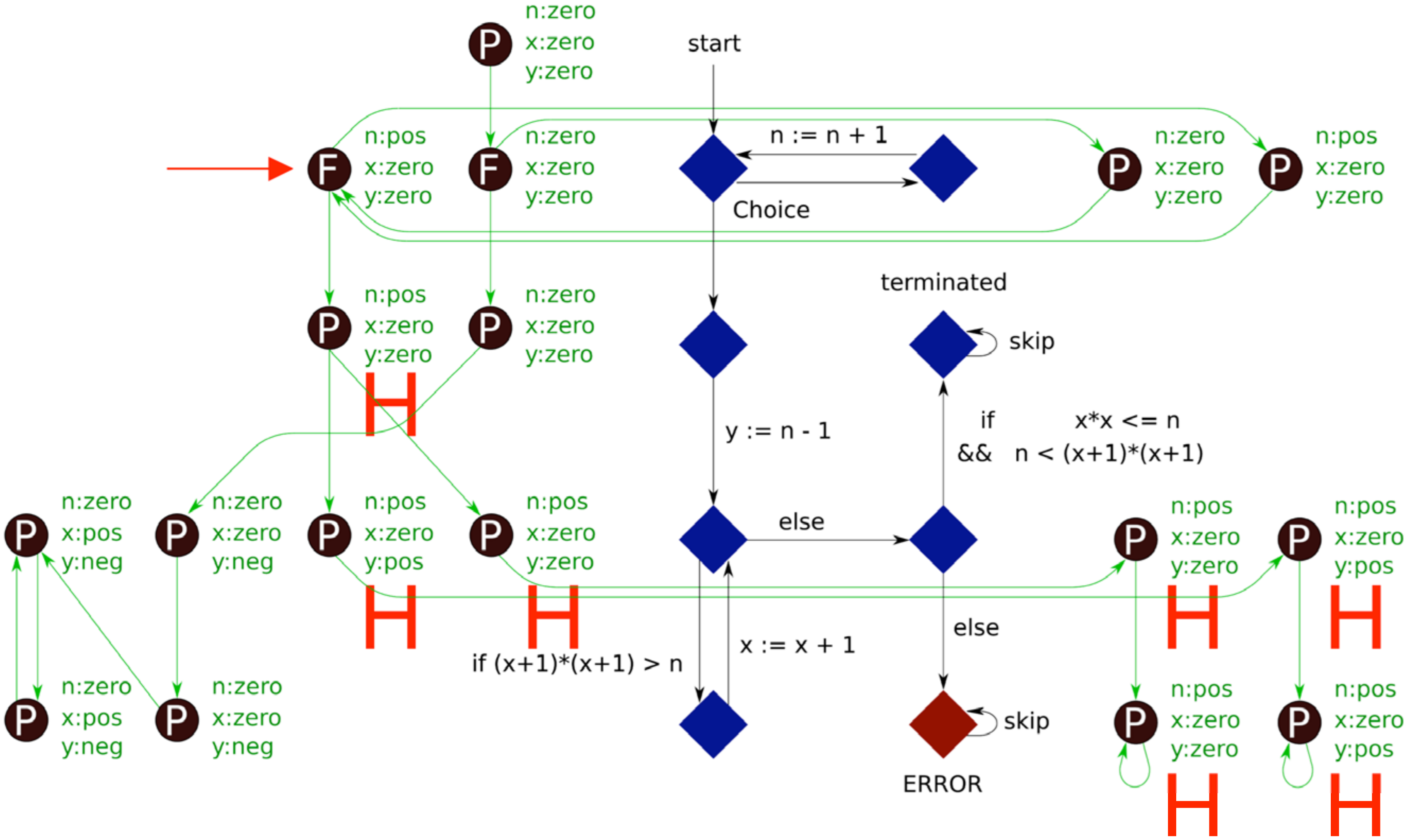


- Use the **h-P-move** rule
  - when there's only one possible successor, execution must flow there (by seriality)

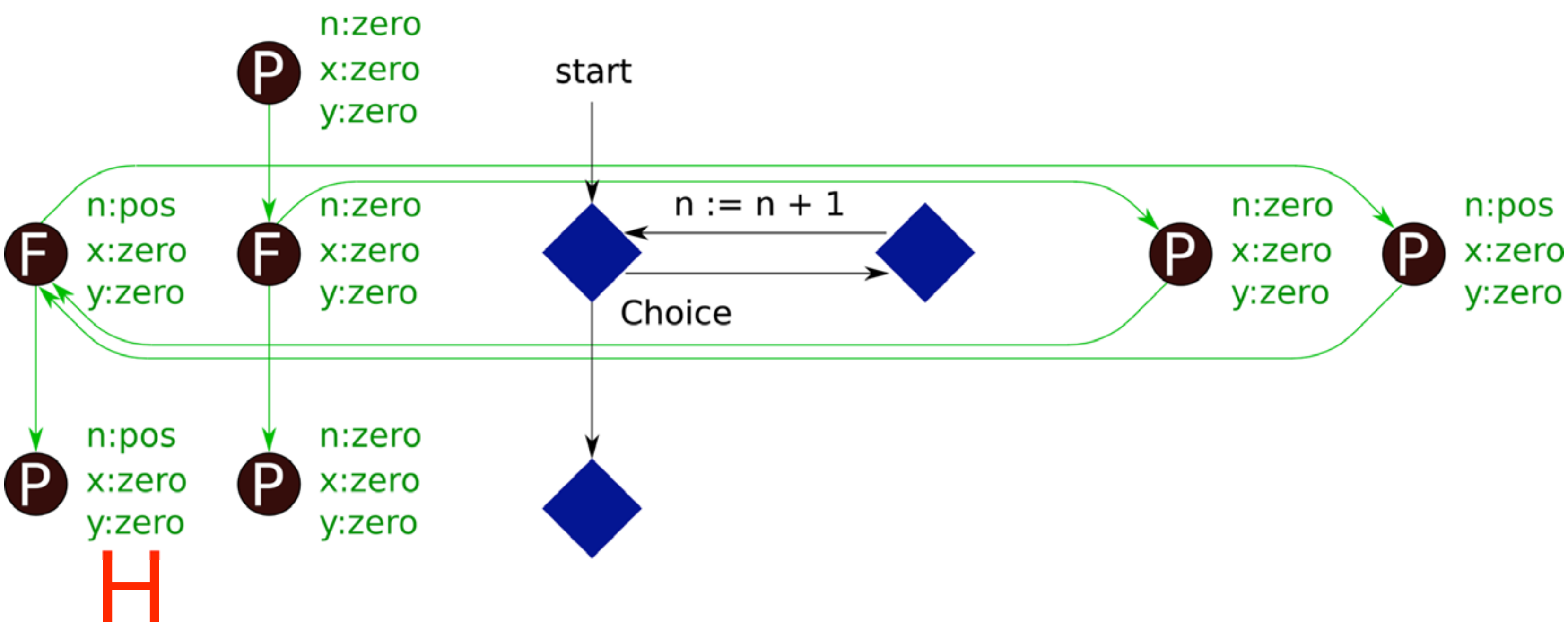

n:zero
x:zero
y:zero
start
n:pos
x:zero
y:zero
n:zero
x:zero
y:zero
n := n + 1
Choice
n:zero
x:zero
y:zero
n:pos
x:zero
y:zero
terminated
n:pos
x:zero
y:zero
n:zero
x:zero
y:zero
skip
y := n - 1
if x*x <= n
&& n < (x+1)*(x+1)
n:zero
x:pos
y:neg
n:zero
x:zero
y:neg
n:pos
x:zero
y:pos
n:pos
x:zero
y:zero
else
n:pos
x:zero
y:zero
n:pos
x:zero
y:pos
H
H
H
H
if (x+1)*(x+1) > n
x := x + 1
else
n:zero
x:pos
y:neg
n:zero
x:zero
y:neg
skip
ERROR
n:pos
x:zero
y:zero
n:pos
x:zero
y:pos
H
H



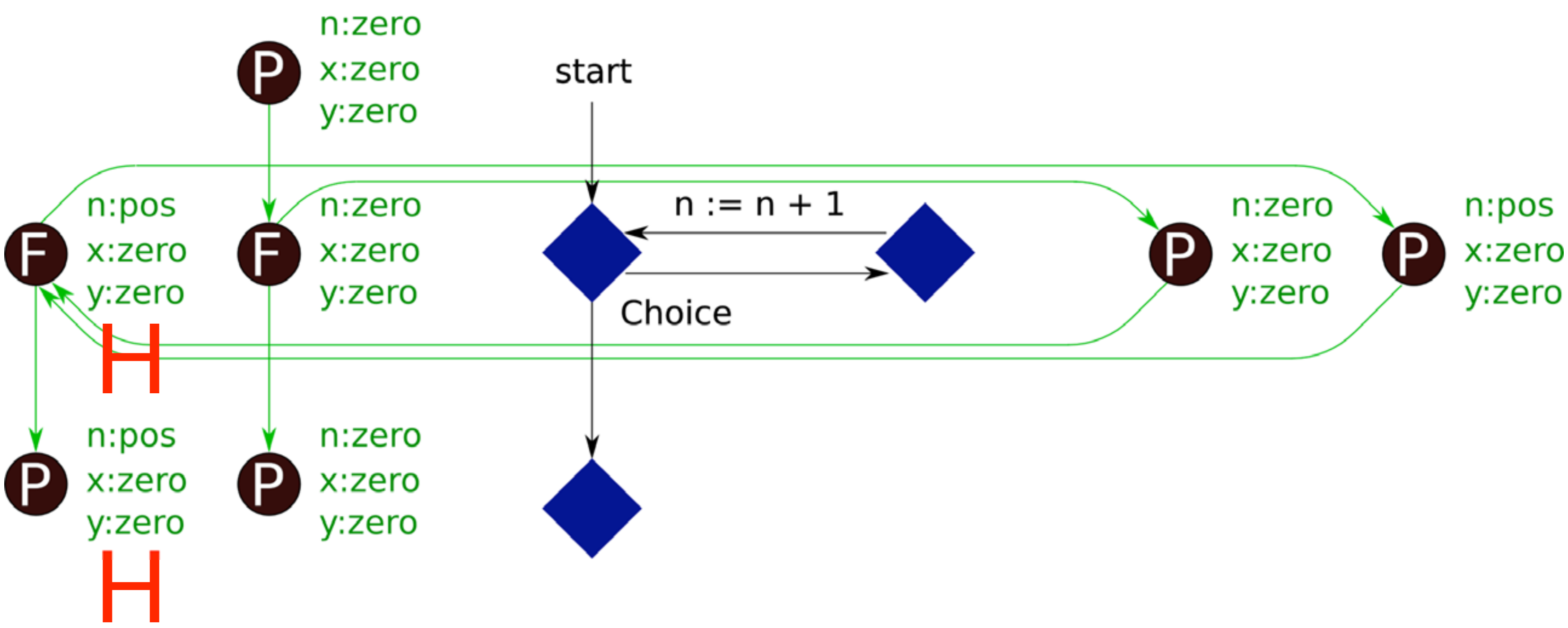
- Here there are two susccessors
  - **abstraction used wasn't precise enough to determine which way execution goes**
  - **but it doesn't matter**, as both are hopeless



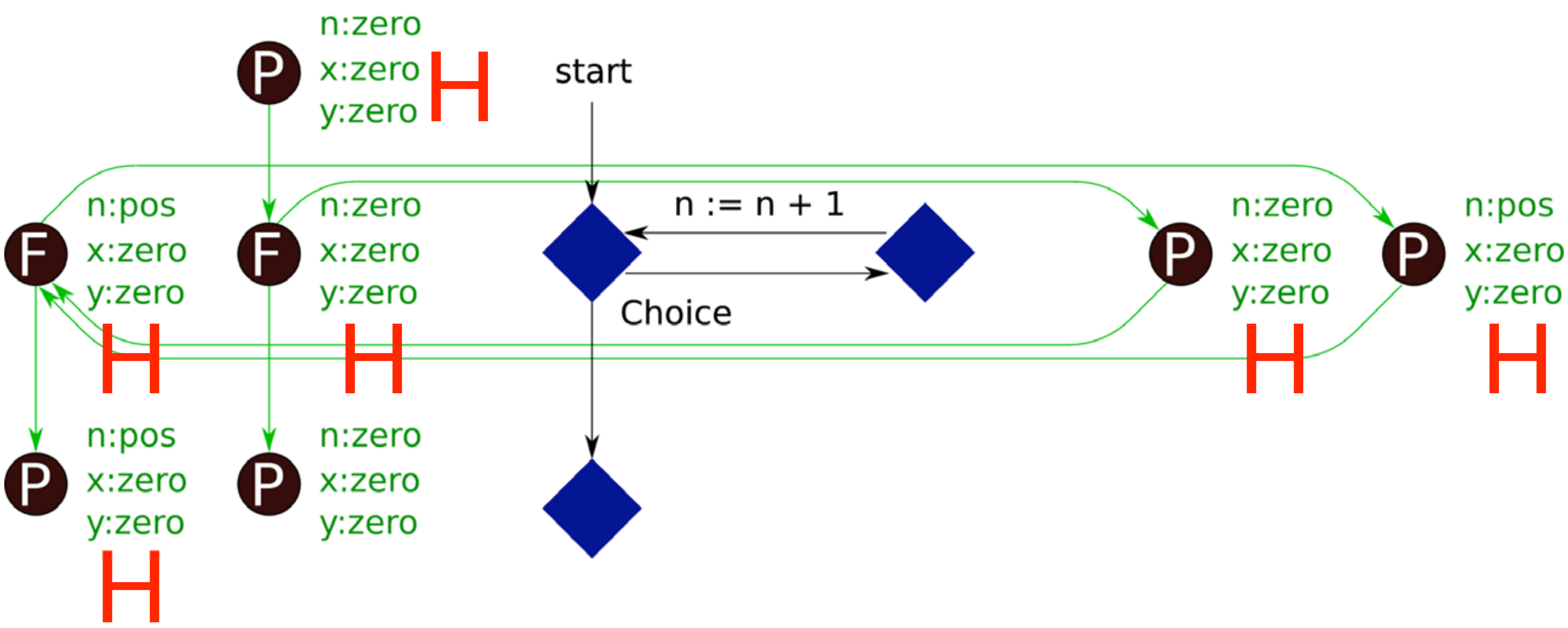
- To finish the falsification, need the **h-F-move** rule
  - to deal with the Choice (hyper)edge



- To finish the falsification, need the h-F-move rule
  - requires **just one** (instead of all) successors to be hopeless



- To finish the falsification, need the h-F-move rule
  – requires **just one** (instead of all) successors to be hopeless

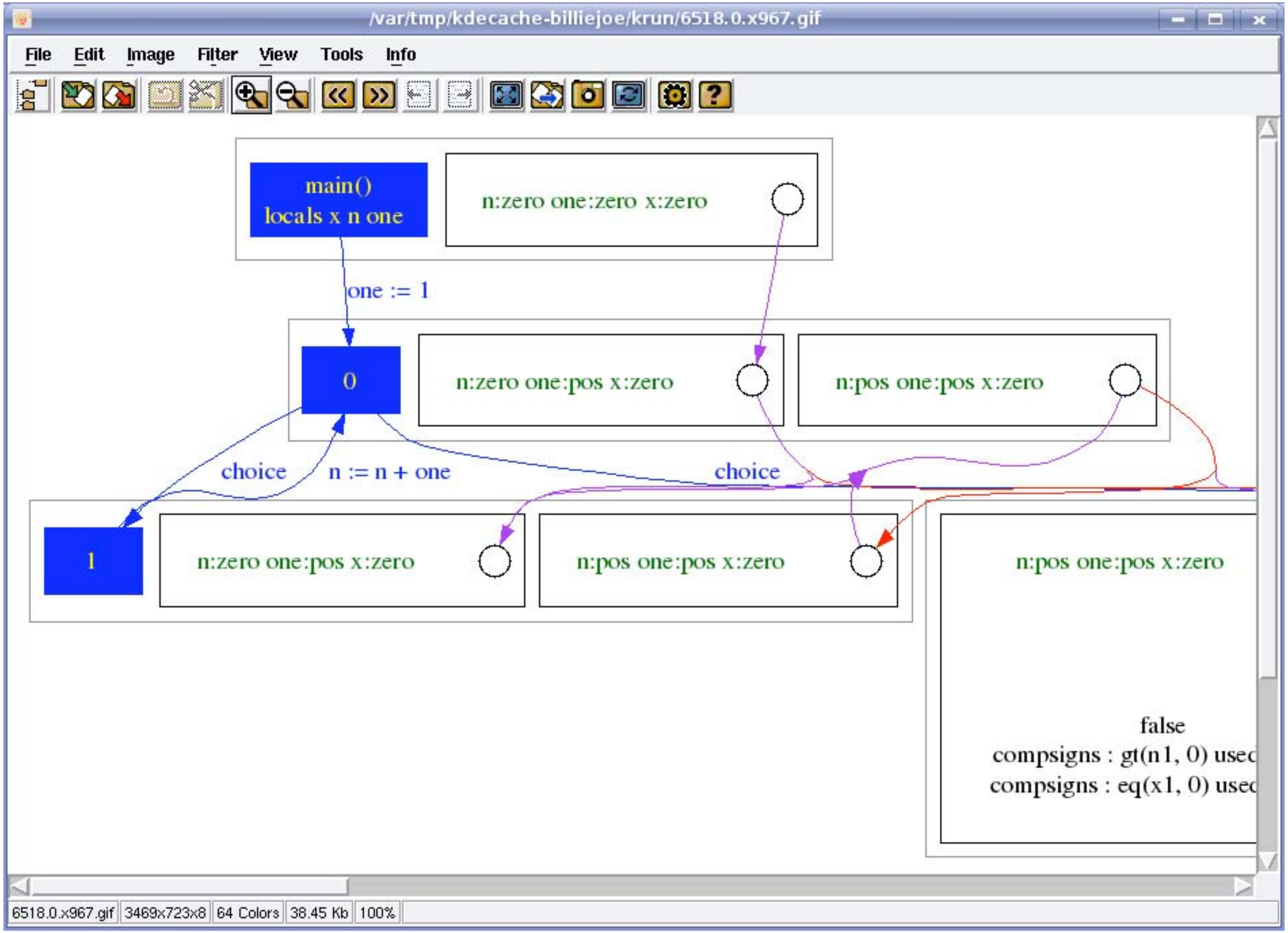


- Intuitively player F can play to **force generation of positive n**
  - which breaks the program

- **Won't work** if Choice treated in same manner as other statements

Implementation

- This check is **implemented in HECTOR**, our software model checker
  - **HECTOR attempts falsification whenever a verification has failed**
- **HECTOR** builds abstract models of imperative programs using “pluggable” abstractions:
  - E.g. predicate abstraction, three-valued shape analysis, sign analysis, type-system-based abstraction
  - and **“good” products of any of the above**
- Extends to recursive procedures (via summarisation)


/var/tmp/kdecache-billiejoe/krun/6518.0.x967.gif
File Edit Image Filter View Tools Info
main()
locals x n one
n:zero one:zero x:zero
one := 1
0
n:zero one:pos x:zero
n:pos one:pos x:zero
choice
n := n + one
choice
1
n:zero one:pos x:zero
n:pos one:pos x:zero
n:pos one:pos x:zero
false
compsigns : gt(n1, 0) usec
compsigns : eq(x1, 0) usec
6518.0.x967.gif 3469x723x8 64 Colors 38.45 Kb 100%

Existing falsification approaches

1. **Search for a concrete counterexample**
   - programs' transition systems typically infinite state
   - and may be infinitely branching
2. **Test whether abstract counterexample path is feasible**
   - not possible in general (underlying SAT problem is undec.)
3. **Introduce under-approximating "must transitions"**
   - extra work: need to construct and manage two transition relations
4. **Use "must hypertransitions"**
   - more precise than 3., but still need two transition relations

More about this in the paper.

Remarks

- **Our method obtains falsifications that 1. - 3. do not**
  - relative to a particular abstraction
  - extreme example: **if { Goldbach conjecture holds }**
then ERROR else ERROR
  - approaches 1. - 3. must discover which branch is taken!
- Must hypertransitions cope with this
  - but still require separate construction of a must relation
  - and extra “transfer functions” for under-approximation
  - whereas we don’t (because we exploit confinement of non-determinism)
- **Playing a safety game seems conceptually simpler**

Summary

- We presented a method for falsifying simple safety properties (i.e. establishing rechability) based on **GAMES**
  - exploits seriality and confinement of non-determinism
- Attributes of method:
  - conceptually simple
  - works with any abstraction domain
  - gives falsifications unobtainable by main current methods
  - no changes to abstract model construction process
  - falsification check runs in time linear in model size
  - therefore, essentially **FREE**.
- Therefore, worth trying even if frequency of success is low


the end