How to Prove Loops to be Correct?

Thomas Baar / Mathias Krebs
EPFL, Lausanne, Switzerland

4th KeY-Workshop, June 8-10, 2005,
Lökeberg near Gothenburg, Sweden

Proving Loops in KeY

- Induction Rule generates
  - BaseCase
  - StepCase
  - UseCase
- User has to provide
  - induction formula
  - induction variable/term

Demo

Simple example:

```
il >= 0 -> {i:=il}<while(i>0){i--;}>i=0
```

Basic Steps

- Find appropriate induction term/variable
  - unwinding the loop body decreases ind-term by 1
- Find appropriate induction formula
  - normally, this is the same as proof goal
- Prove the POs of induction rule
  - base case and use case are normally trivial
  - step case can by tricky

Methodology: How can we assist the user in finding successful induction variables and formula?

Variants of DecrByOne

- Mismatch between BaseCase and Loop Termination

```
il>=5 -> {i:=il}<while(i>5){i--;}>i=5
```

  - BaseCase comes for free (0>=5 -> …)
  - StepCase has form

```
(il>= 5 -> {i:=il}…)
->
(succ(il)>=5) -> {i:=succ(il)…
```

Interesting case: `succ(il)=5`

Variants of DecrByOne

- Fml is valid but not ‘inductive’

```
il >= 5 -> {i:=il}<while(i>0){i--;}>i=0
```

If the original proof goal is not ‘inductive’ it must be made stronger.

Decrease Induction not only by One

```
il>=0 -> {i:=il}<while(i>0) {i--;i--;}>(i=0¦i=-1)
```

Step Case:

```
(il >= 0 -> {i:=il}<{while(…)}>)
   -> (succ(il)>=0 -> {i:=succ(il)}<{while(i>0){i--;i--;}} >
```

After unwinding:

```
(il >= 0 -> {i:=il}<{while(…)  }> )
   -> (succ(il)>=0 -> {i:=il-1}<{while(…)   }>
```

Induction term decreased by more than one:
-> use strong induction

Example: Russian Multiplication

```
(   geq(al, 0)
       -> {a:=al}
            {b:=bl}
             {z:=0}
               <{
                  while ( a!=0 ) {
                    if (a/2*2!=a) {
                      z=z+b;
                    }
                    a=a/2;
                    b=b*2;
                  }
                }> z = al * bl)


}
```

induction term is al

induction term is strictly decreased, possibly by more than one
-> strong induction

proof goal is not inductive
-> strengthening of ind-fml

Multiple Induction Terms

- Requires nested induction
  - exponential number of POs (2 Ind-terms-> 9 POs)

```
{i:=il}{j:=jl}
               <{
                  while ( i>0|j>0 ) {
                    if (i>j) {
                      i--; }
                    else {
                      j--;
                    }
                  }
                }> (i = 0 & j = 0))
```

Multiple Induction Terms

```
{i:=il} {j:=jl}
               <{
                  while ( i>0|j>0 ) {
                    if (j==0) {
                      i--; j=9;
                    }
                    else {
                      j--;
                    }
                  }
                }> (i = 0 & j = 0))
```

- Sometimes, more than one loop-unwind must be symbolically executed to make ind-terms smaller

Further Problems

- Induction Var is increased instead decreased
  - requires technical trick for induction formula
- Would be nice to have prestate-projection rule

$$\frac{POST1' \vdash POST2'}{<p>POST1 \vdash <p>POST2}$$ POST' is POST with fresh prog-var

- Accumulator variables can destroy update

Is There a Better Way?

Yes! Just use another tool ☹

BLAST

Berkeley **L**azy **A**bstraction **S**oftware **V**erification **T**ool

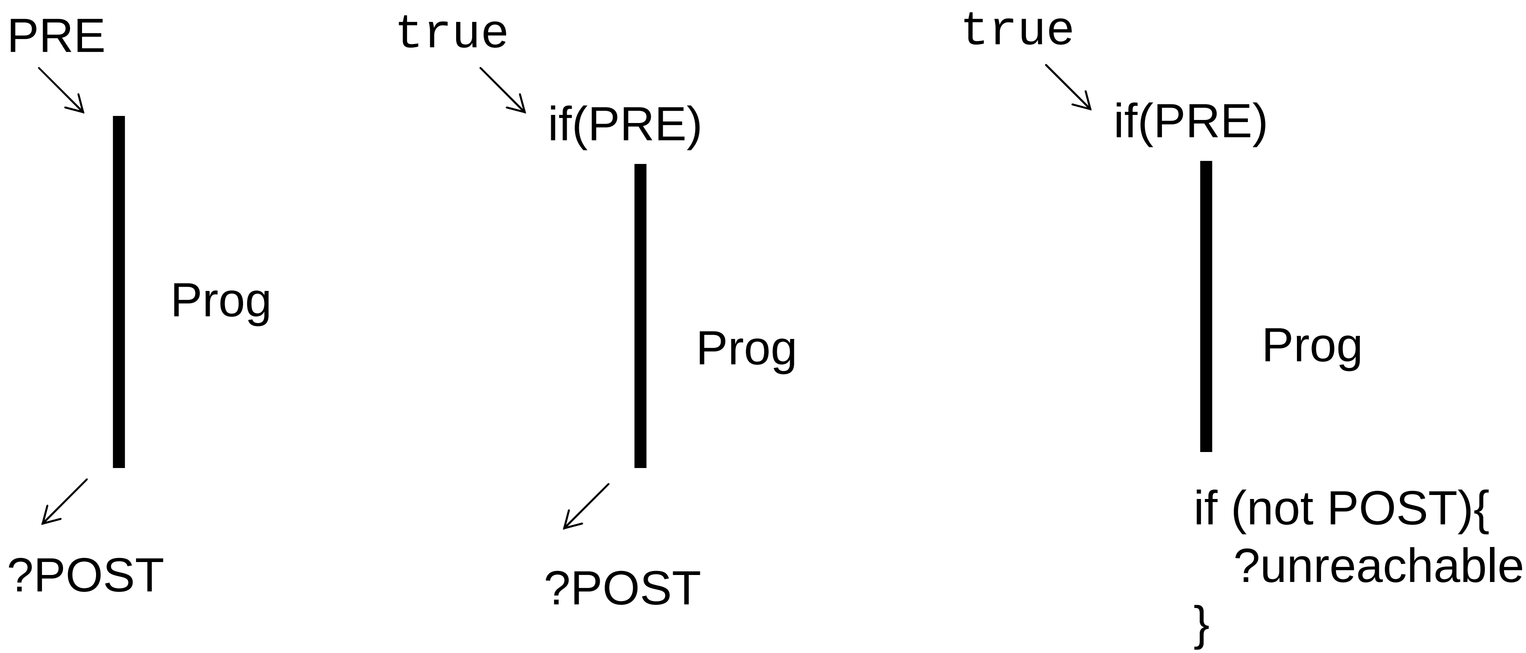
BLAST

- Verification tool for C programs
- Based on model checking
- Can only prove partial correctness (safety properties)

Many great ideas that can be applied in KeY as well!

Find more information on BLAST: Dirk Beyer, Thomas A. Henzinger, Renjit Jhala, and Rupak Majumdar: *Checking Memory Safety with Blast*. FASE 2005. LNCS 3442.

Ways to Express Safety



Every partial correctness property for a program (box modality) can be easily reformulated in terms of reachability of a certain statement.

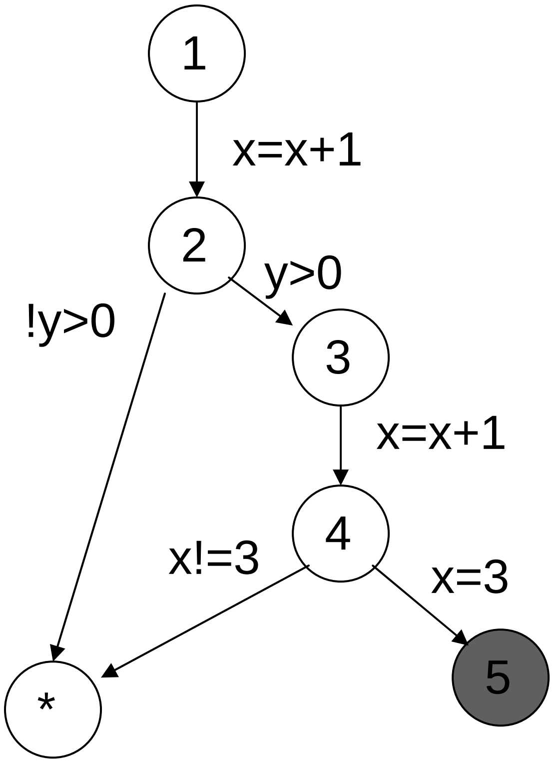
An Unsafe Example

Program

```
x=x+1,
if (y > 0){
  x=x+1;
  if (x=3){
    printf("error")
  }
}
```

Control Flow Automaton (CFA)

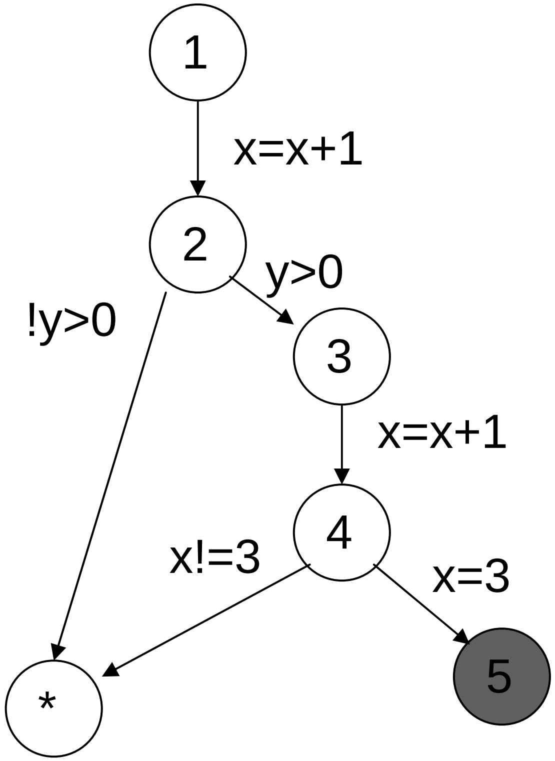
- nodes = control points
- edges = decisions/statements



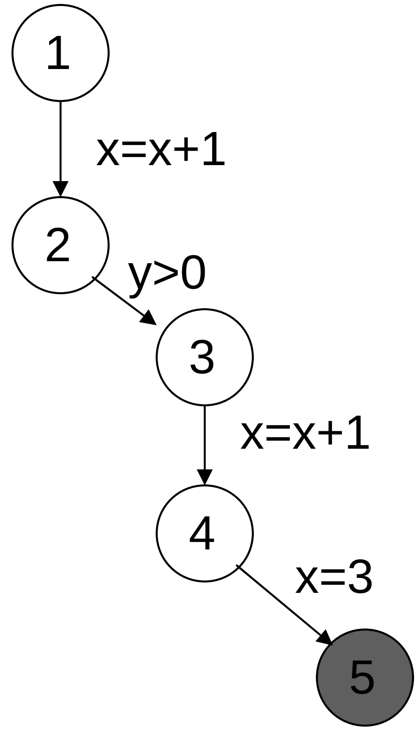
An Unsafe Example

Is Trace-Formula satisfiable?
YES -- genuine counterexample
NO -- spurious counterexample

CFA



ART (Abstract Reachability Tree)



x1=x+1

y>0

x2=x1+1

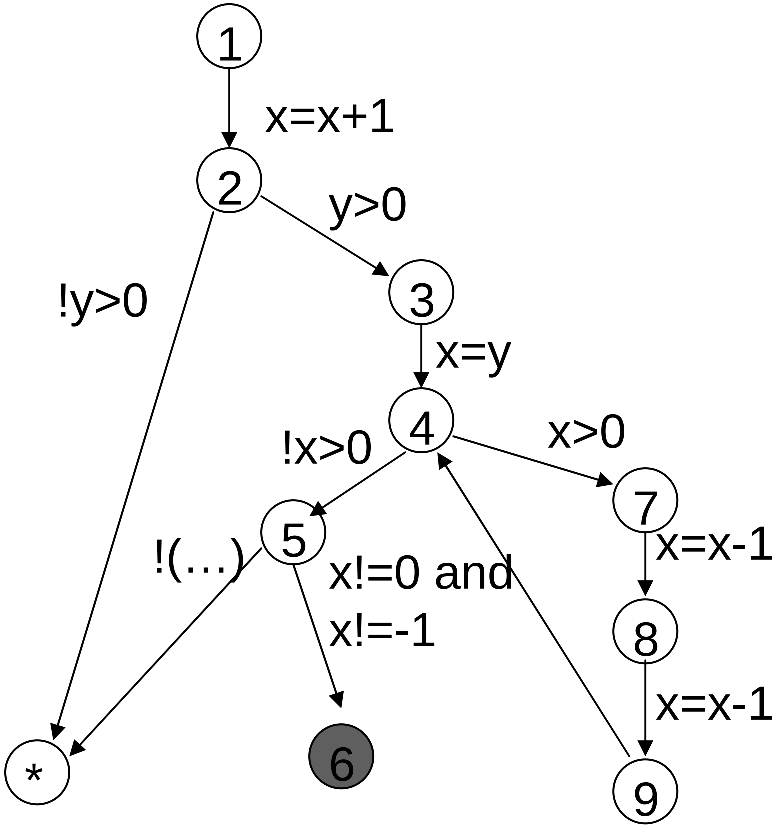
x2=3

A Safe Example

Program

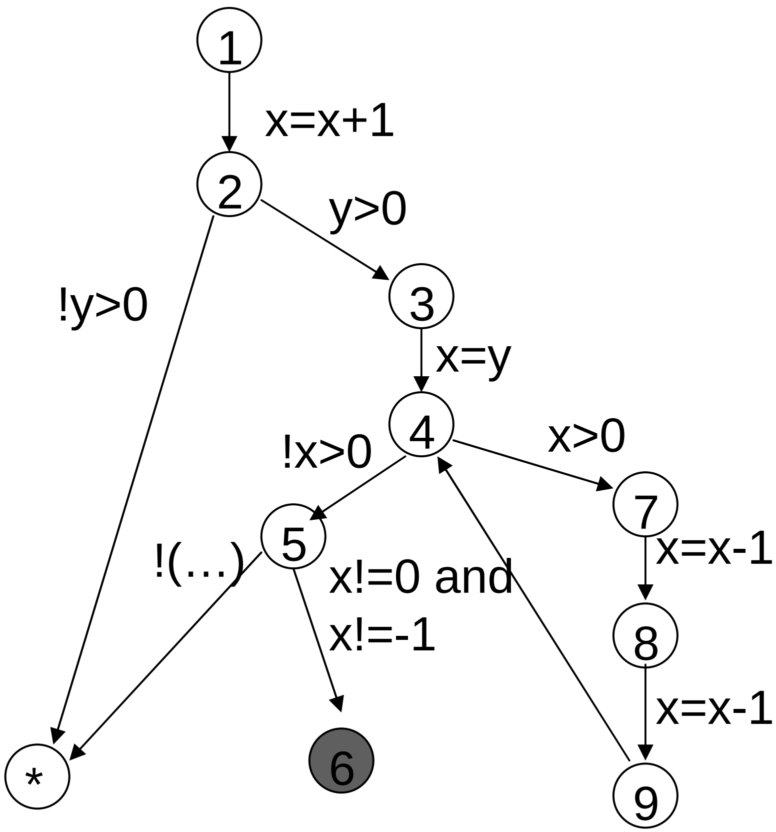
```
x=x+1;
if (y>0){
  x=y;
  while (x > 0){
     x=x-1;
     x=x-1;
  }
  if (x!=0 and x!=-1){
    printf("error");
  }
}
```

CFA

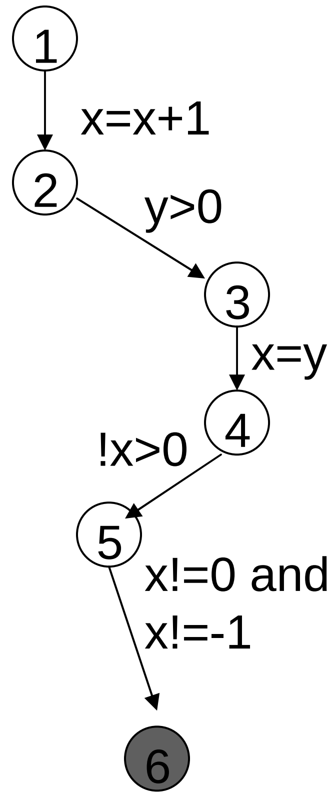


Craig-Interpolation

CFA



ART



Trace-Fml

x1=x+1

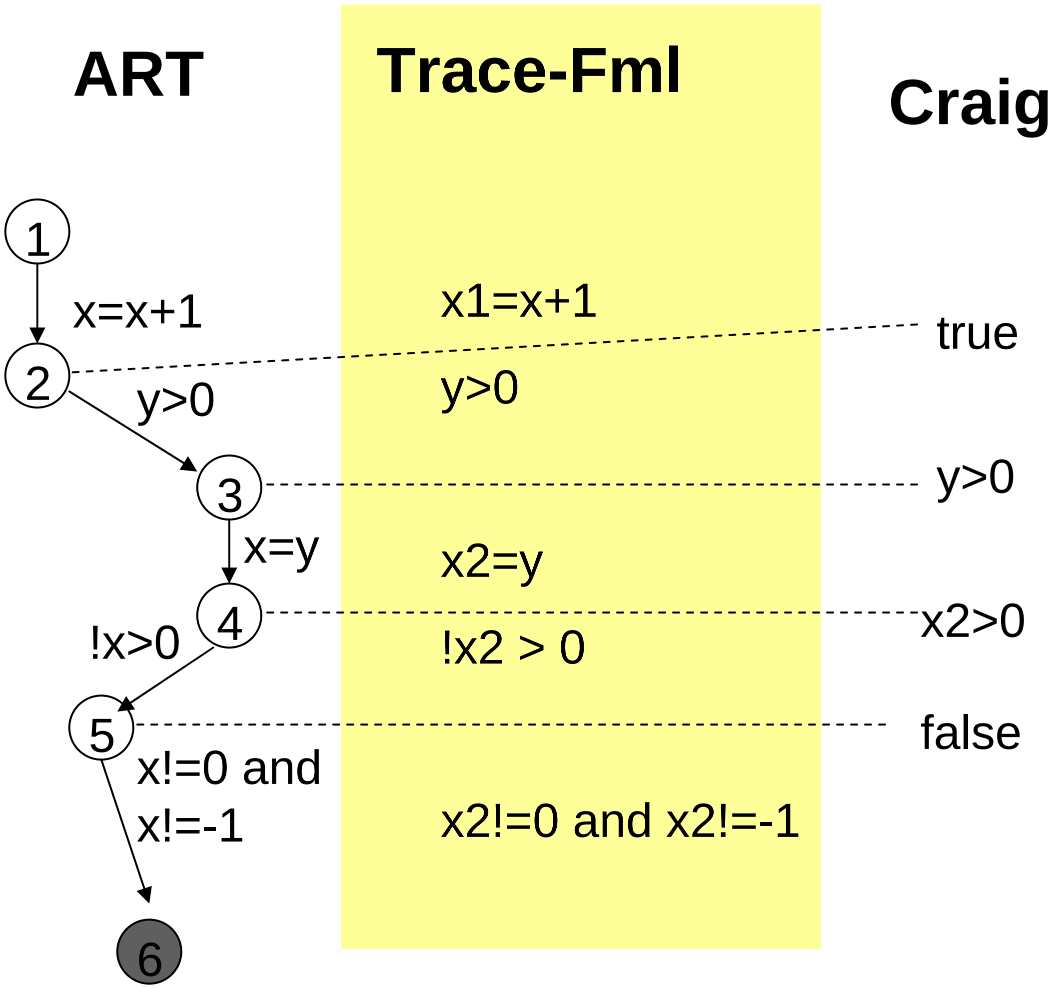
y>0

x2=y

!x2 > 0

x2!=0 and x2!=-1

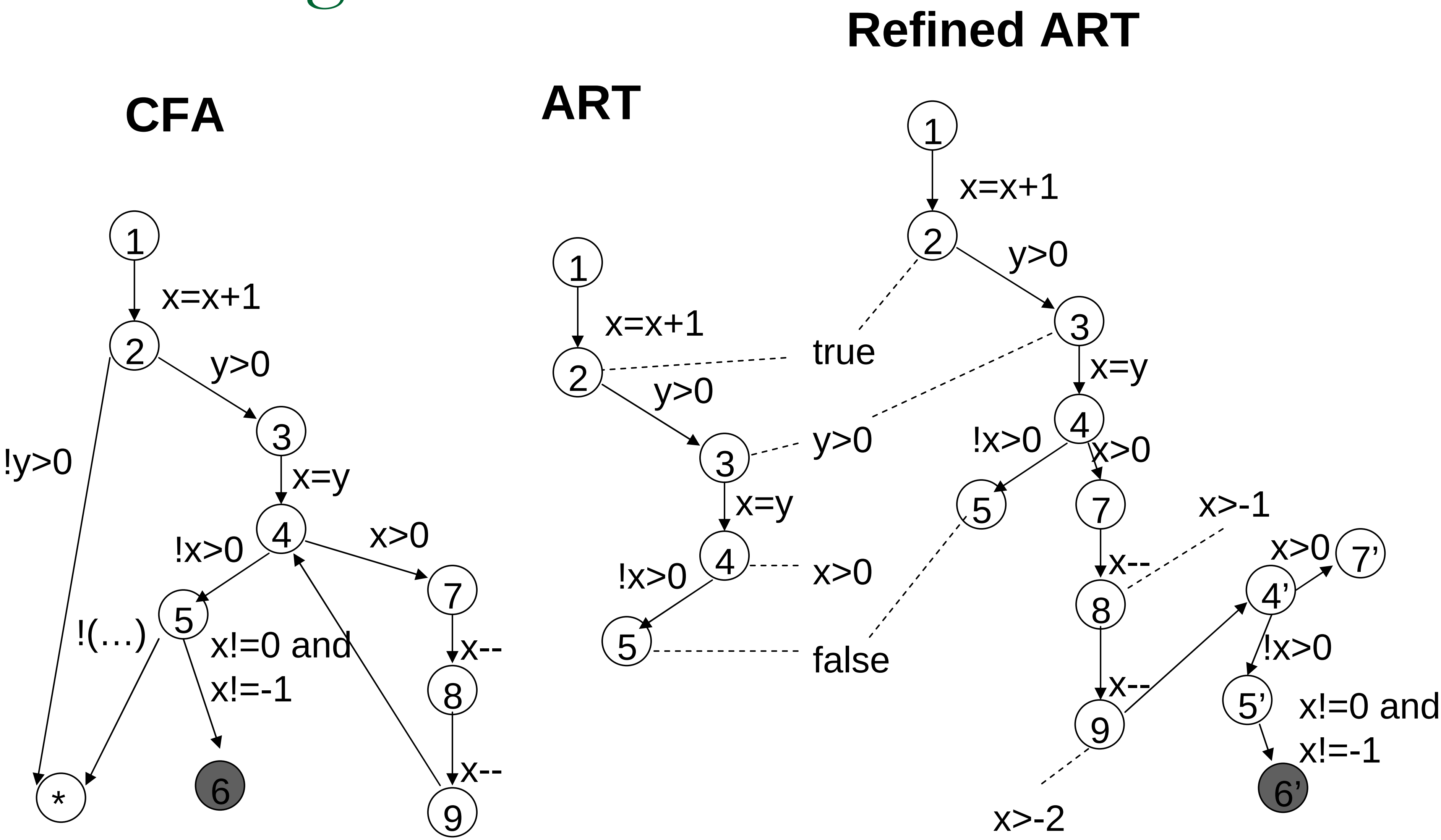
Craig-Interpolation



If trace fml is not satisfiable we 'refine the abstraction'.
For each control point:
- split trace fml into before/after
- find cp-fml such that
    - BEF implies CP
    - CP and AFT implies false
    - vocabulary is intersection

Craig formulas are attached to corresponding trace node:
-they 'overapproximate' properties of trace state
-nodes with 'false' are never reachable

Refining the ART



Closing the ART

- An ART is closed iff
  - all possible alternatives are explored
  - each leaf node is
    - exit node
    - annotated with `false`
    - there is another node with same label and weaker annotation

Summary SW-Model Checking

- Fully automatically
- Closed ART is formal proof for safety property
- Proof is found by analyzing (spurious) counterexamples
- Scalable approach (Craig-Interpolation)
- Concrete counterexample for incorrect props

- Open Problem: TERMINATION
  - Room for combining BLAST/KEY
  - Key-Proof: Look out for Ind-Term which is made strictly smaller in loop body

KeY vs. BLAST

- KeY
  - requires interaction
    - user can give hints
  - no support yet for easy bug detection
  - total correctness
- BLAST
  - push-button
  - easily finds bugs
  - does not prove termination
  - does not support multiplication

Other Activities

- OCL workshop at MODELS’05
  - conference in Montego Bay, Jamaica ☺
  - focus on tool support for OCL
- Paper Thomas Baar: *Non-deterministic Constructs in OCL – What does any() Mean*, SDL’05, Grimstad, Norway.