SATURN and Calysto

Andrew Becker
21.10.2011

Why “Rethinking” SymbEx?

- Previous SymbEx techniques not scalable
  - Exponential path explosion
  - Size and number of SAT queries grow exponentially

- Try to tackle the explosion
  - SATURN: function summaries compress constraint expressions
  - Calysto: use MSGs and Structural Refinement for 'lazy abstraction'

What hasn't changed?

- (Un)soundness
  - Multiple sources: pointers, loops...
- Basic strategy
  - Delve through execution paths combining constraints
  - Verify some properties of the program using the constraints

Design Choices

- Both are bit-precise
  - Model machine arithmetic exactly
  - Bounded range on integers
    - Need type knowledge
- Both are *intra*procedurally path-sensitive
  - Down to bit level
  - Use knowledge of dependencies computed in SymbEx to automatically slice SAT queries for properties

Design Choices (2)

- SATURN is partially *inter*procedurally path-sensitive
  - Function summaries → interprocedural path-sensitivity not guaranteed
- Calysto is fully interprocedurally path-sensitive
  - Context-sensitive

Unsoundess (1)

```
int cnt = 0;
bool c2 = false;
while(c1){
   if(c2){
      cnt++;
   }
   c2 = true;
}
if(cnt == 0){ exit(1); }
//More code...
```

- Unroll the loop once and miss a lot of code
- This is what Calysto does

Unsoundess (2)

```
int cnt = 0;
bool c2 = false;
while(c1){
   if(c2){
      cnt++;
   }
   c2 = true;
}
if(cnt == 0){ exit(1); }
//More code...
```

- Unroll the loop fixed number of times
- If termination condition unsatisfiable, HAVOC
- More code reachable; analysis still unsound

Unsoundess (3)

```
int bar(int x){
   x = (x <= 10)? (1) : (bar(x - 2) + 1);
   return x;
}
void foo(){
   int x = bar(1000);
   if(x > 500) exit(1);
   //More code...
}
```

- Same problem as above
- Unsoundness unless you unroll indefinitely
  - Infinite loops?

Unsoundess (4)

- Even worse: mutually recursive functions
  - Real example: Postfix `xmalloc` calls `fail` on failure, `fail` calls `xmalloc` to build buffer...
- SATURN picks arbitrary ordering, cuts recursion, generates summaries
  - Summaries can be very imprecise...
- Calysto just cuts at recursive call
  - And also introduces unsoundness

SATURN Function Summaries

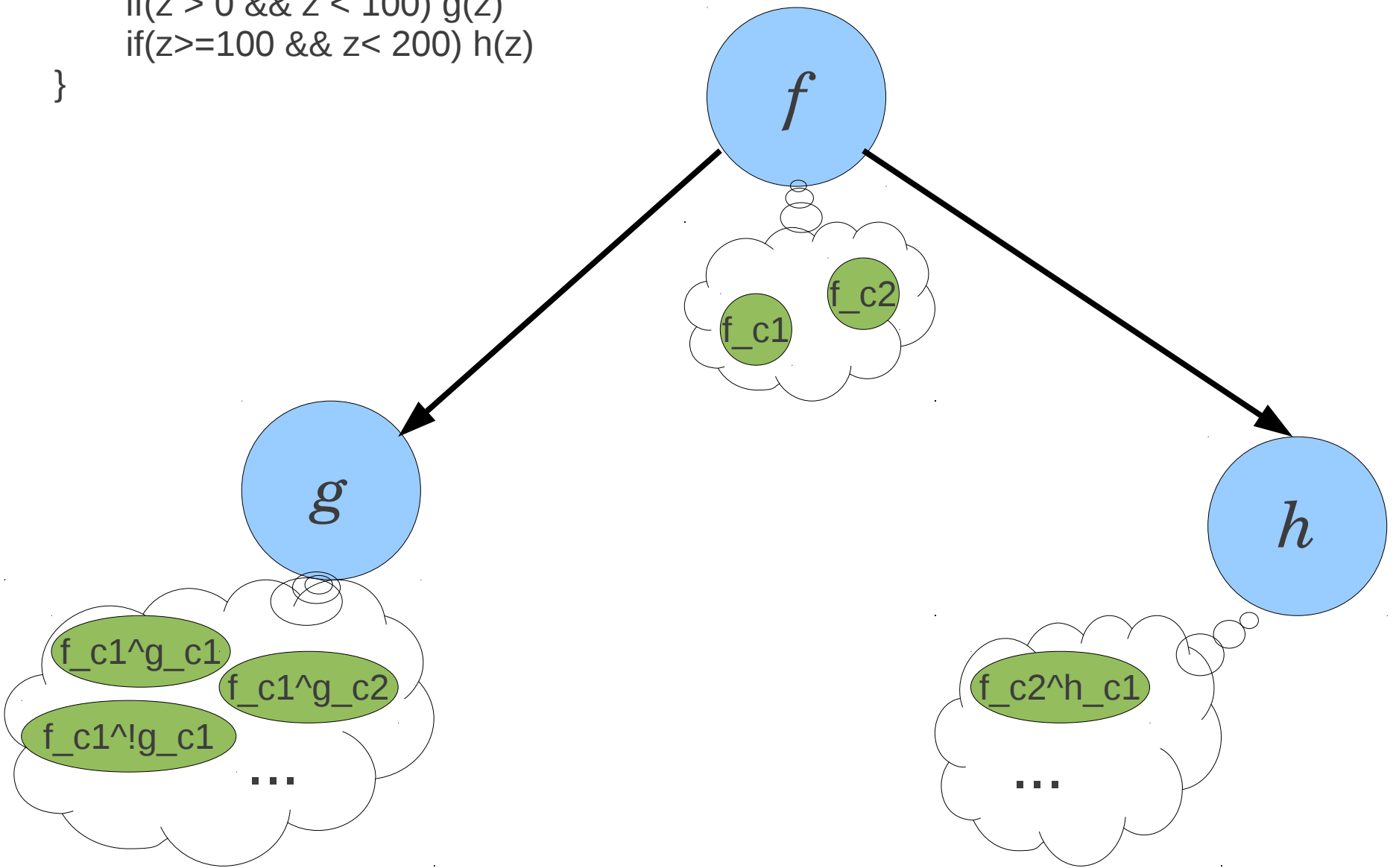
- A compact description of all effects relevant to some verification property
  - Leak detection
    - Whether the function returns malloc'ed memory
    - Set of locations which “escape”
  - Lock checker
    - Pre- and Post-condition for valid property
    - Set of relevant locations which may change (e.g. the lock)
    - Set of transitions mapping (lock modified X input condition X beginning state) to (output cond. X ending state)

SATURN Function Summaries (2)

- Verifying properties much faster
  - Less clutter
  - Only uses what is relevant to the property
- Abstract away irrelevant details

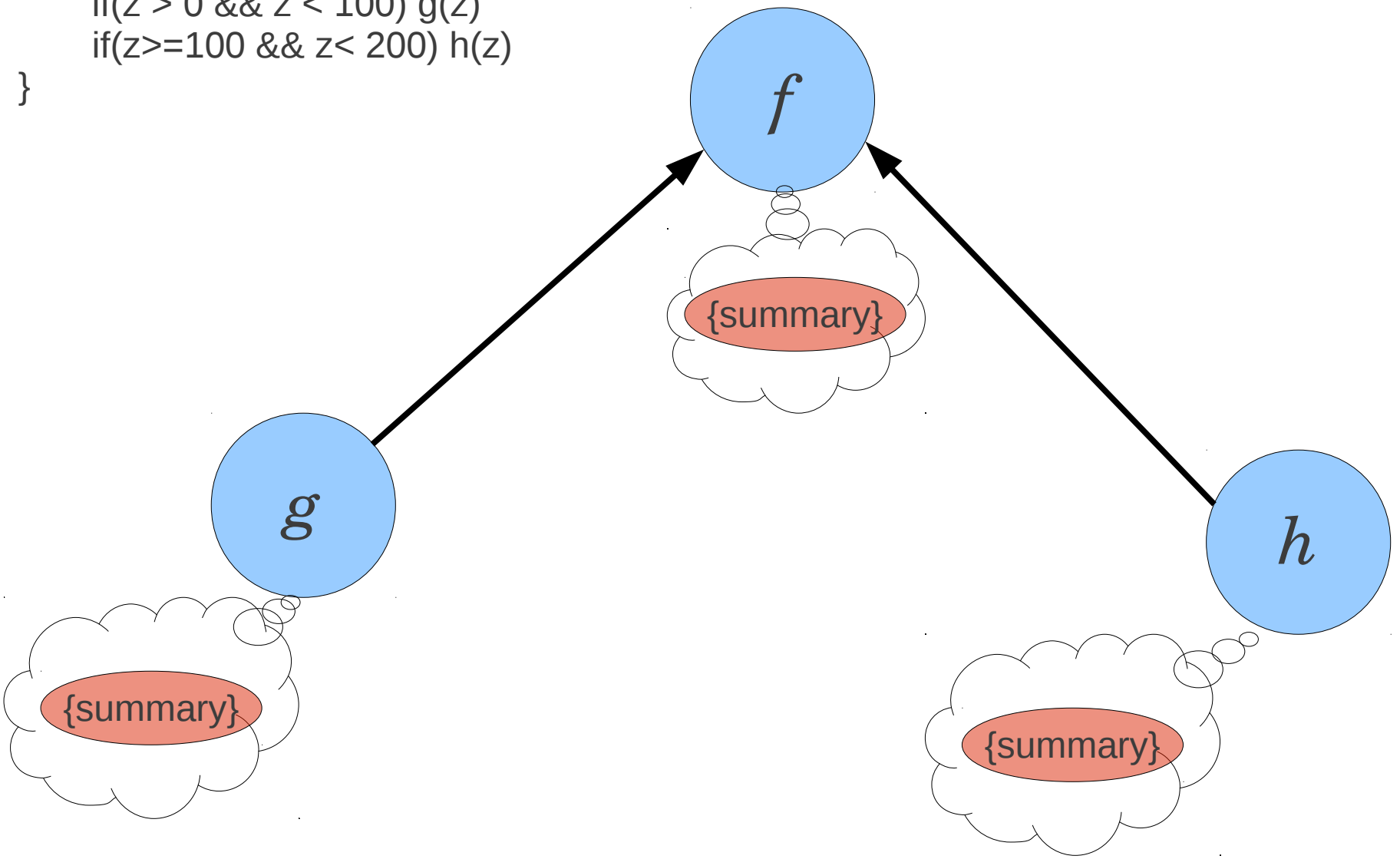
SATURN Function Summaries (3)

```
void f(int z){
        if(z > 0 && z < 100) g(z)
        if(z>=100 && z< 200) h(z)
}
```



SATURN Function Summaries (3)

```
void f(int z){
        if(z > 0 && z < 100) g(z)
        if(z>=100 && z< 200) h(z)
}
```

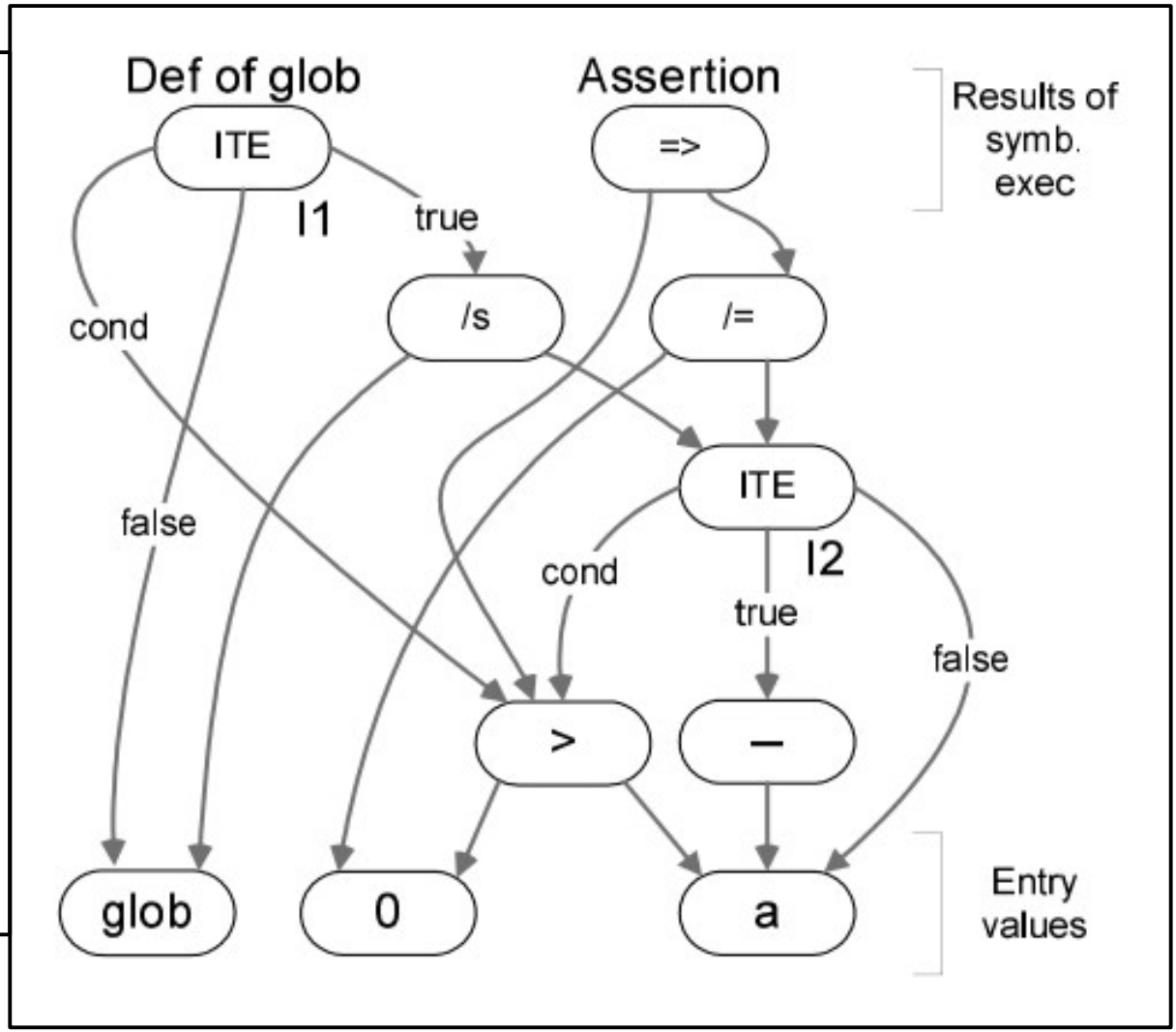


Calysto MSGs

- SymbEx of every function generates a MSG
- Similar to a function's AST with CSE
- Symbolic values of visible variables and properties are nodes with pointers into MSG

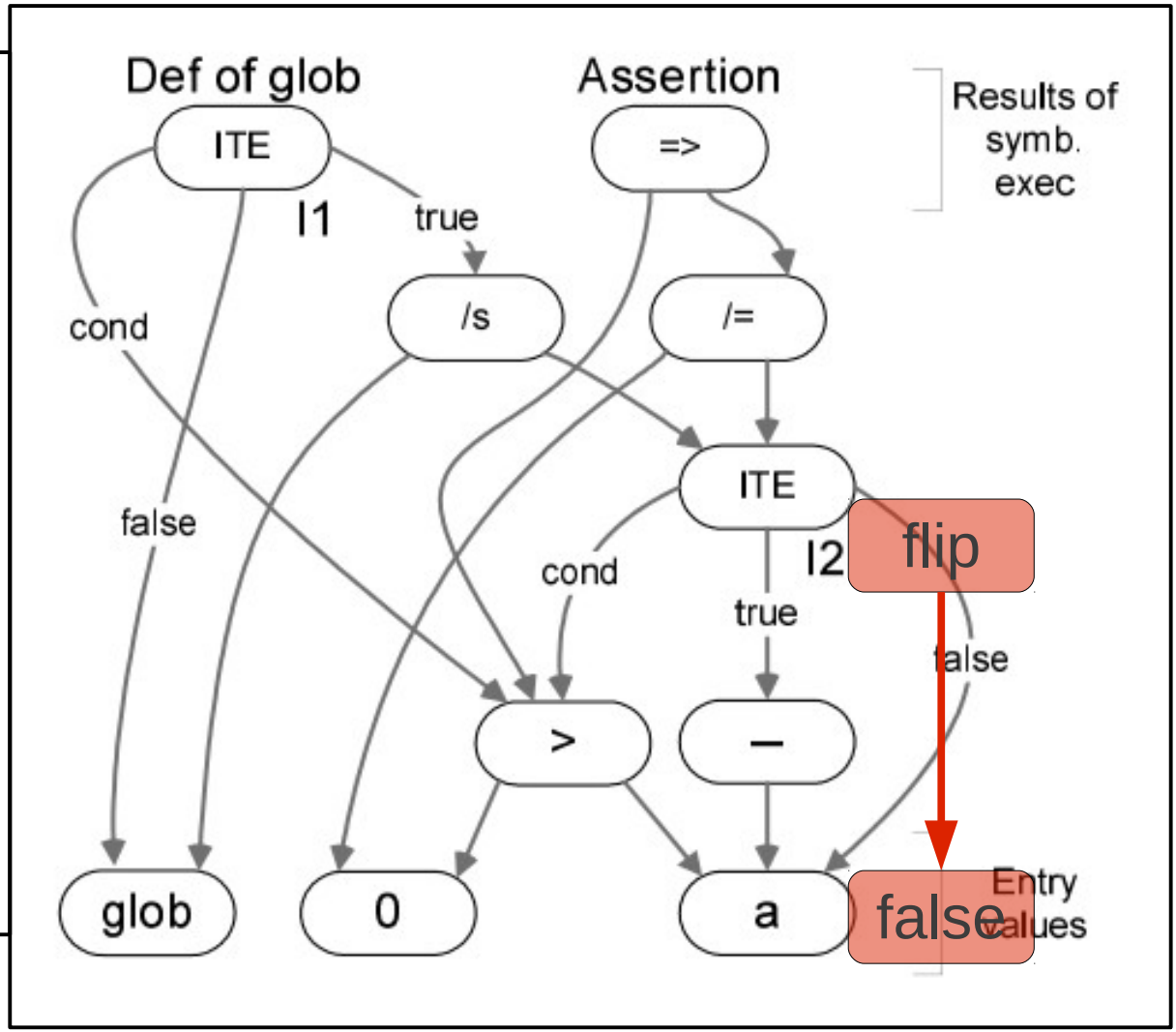
MSG Example

```
int glob; // Global
...
void f(int a) {
    bool flip = false;
    if (a < 0) {
        a = -a;
        flip = true;
    }

    if (flip) {
        assert(a != 0); // A1
        glob /= a;
    }
    return;
}
```



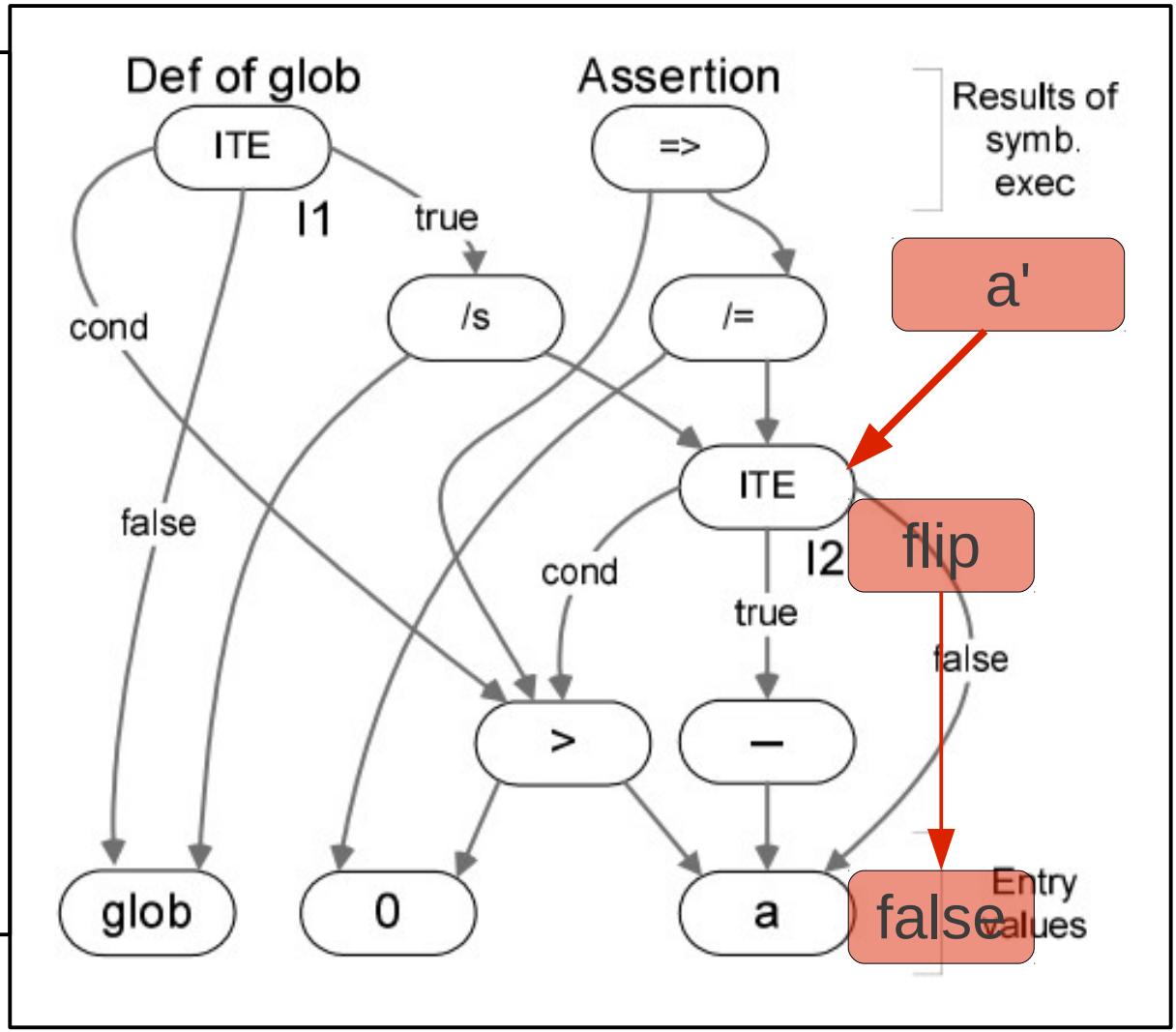
MSG Example

```
int glob; // Global
...
void f(int a) {
    bool flip = false;
    if (a < 0) {
        a = -a;
        flip = true;
    }

    if (flip) {
        assert(a != 0); // A1
        glob /= a;
    }
    return;
}
```



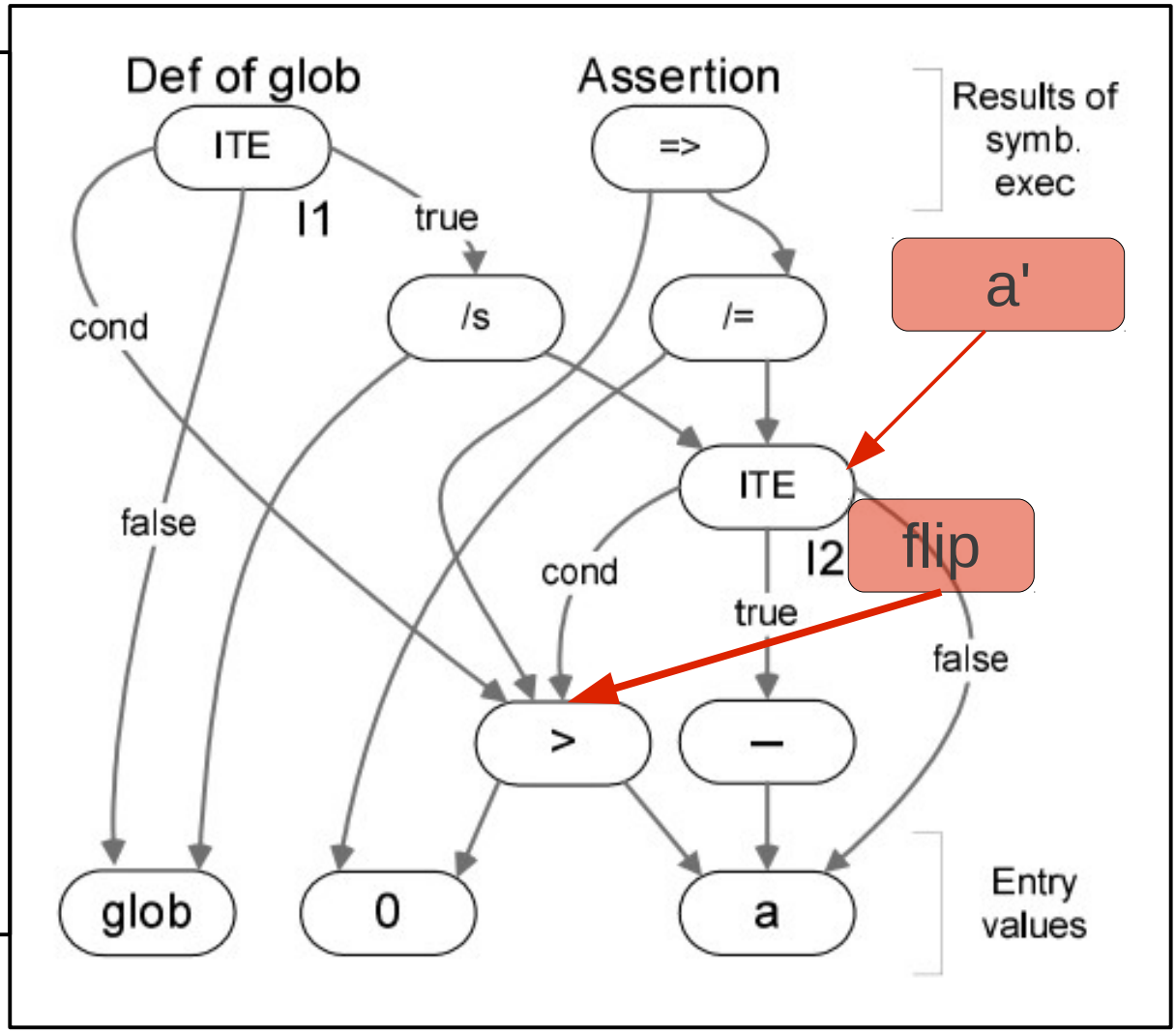
MSG Example

```
int glob; // Global
...
void f(int a) {
    bool flip = false;
    if (a < 0) {
        a = -a;
        flip = true;
    }

    if (flip) {
        assert(a != 0); // A1
        glob /= a;
    }
    return;
}
```



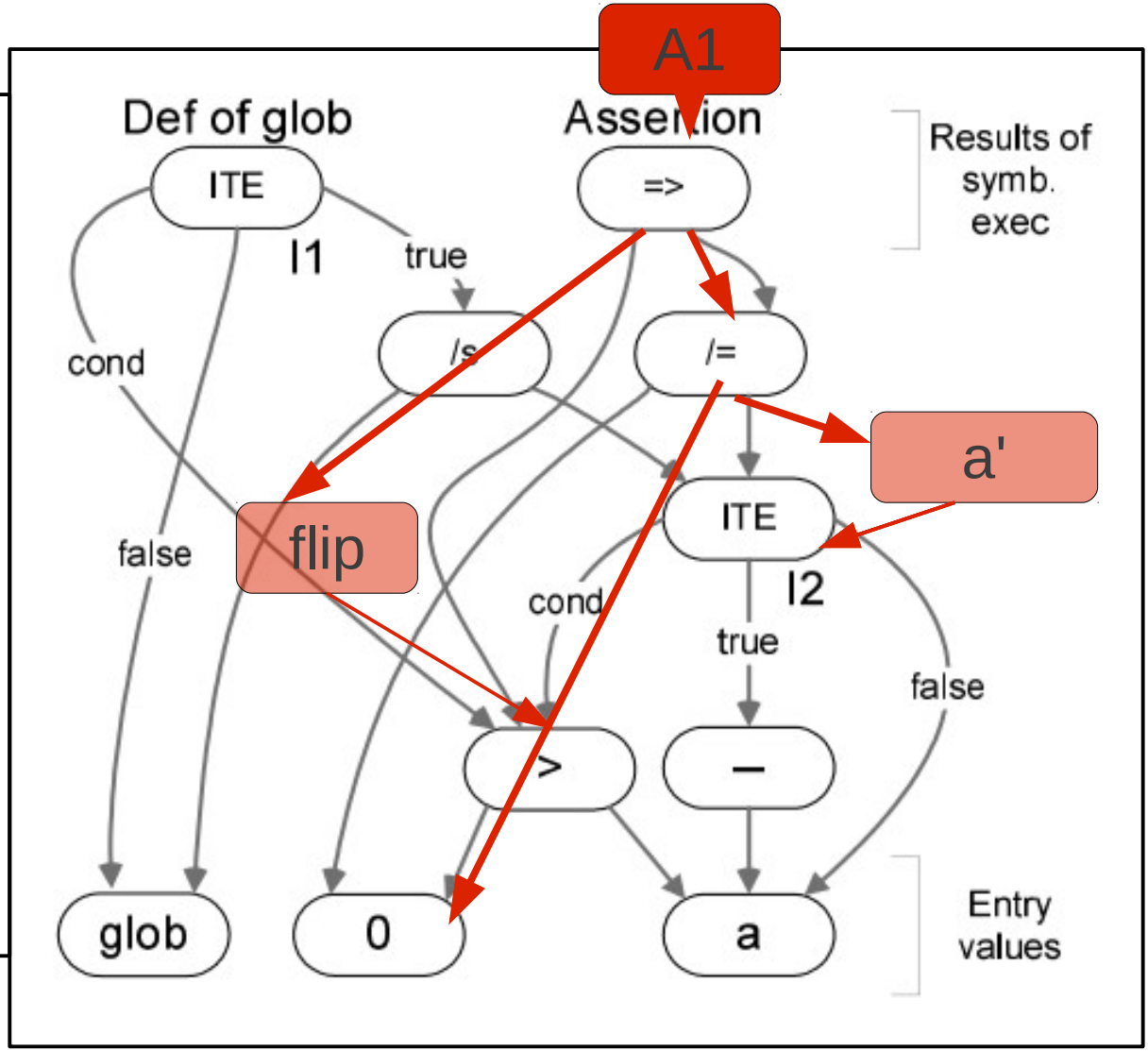
MSG Example

```
int glob; // Global
...
void f(int a) {
    bool flip = false;
    if (a < 0) {
        a = -a;
        flip = true;
    }

    if (flip) {
        assert(a != 0); // A1
        glob /= a;
    }
    return;
}
```



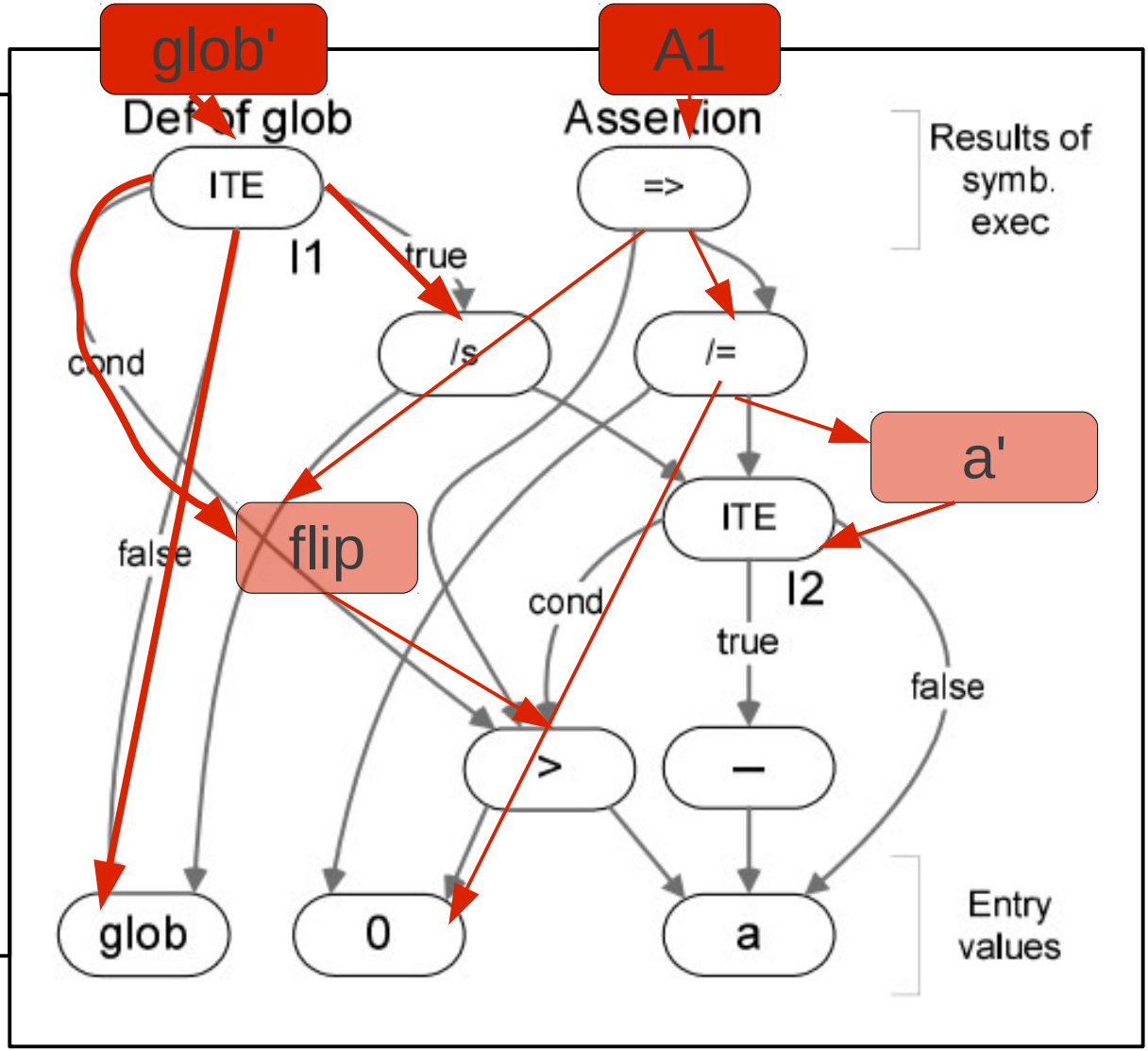
MSG Example

```
int glob; // Global
...
void f(int a) {
    bool flip = false;
    if (a < 0) {
        a = -a;
        flip = true;
    }

    if (flip) {
        assert(a != 0); // A1
        glob /= a;
    }
    return;
}
```



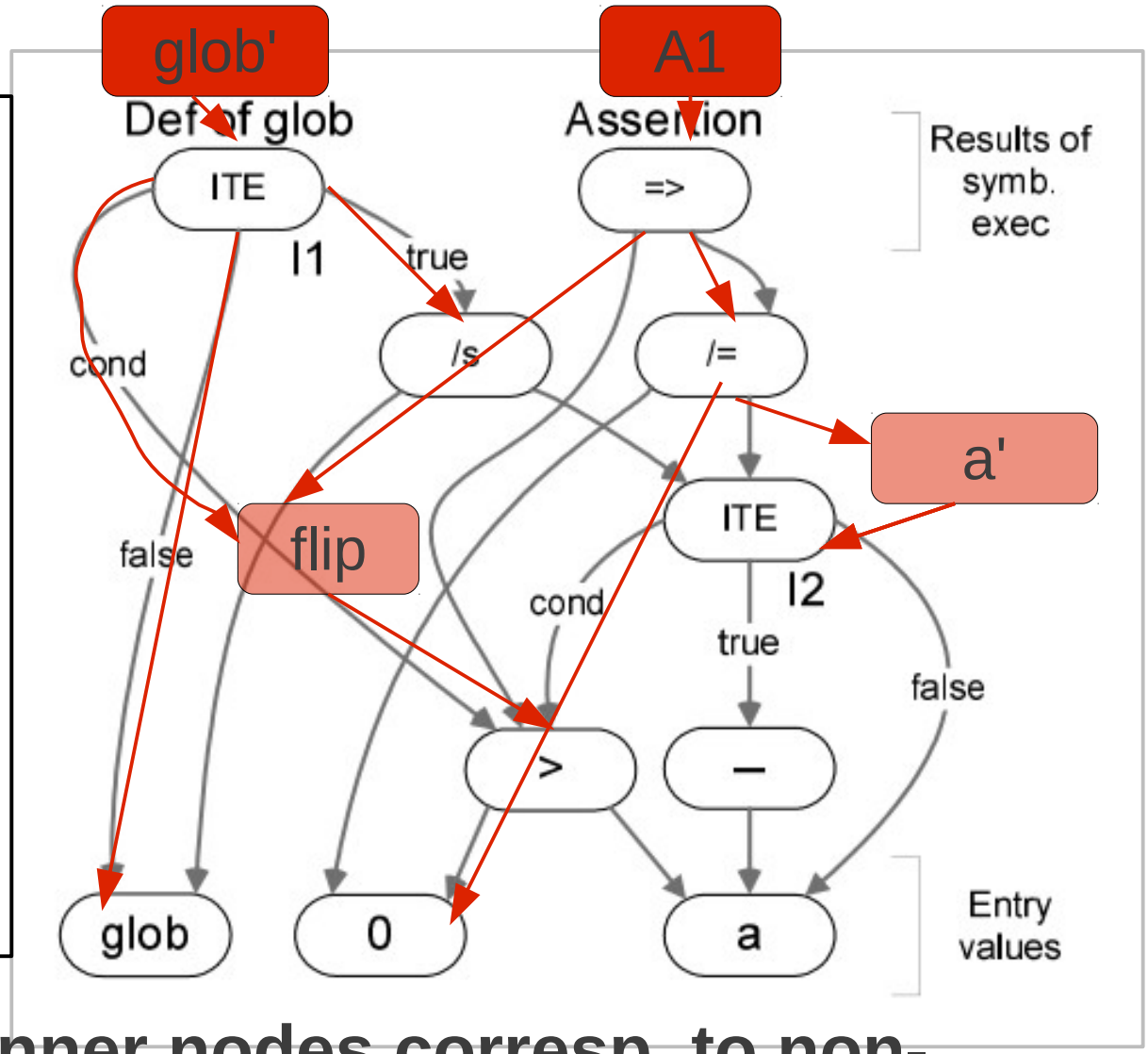
MSG Example

```
int glob; // Global
...
void f(int a) {
    bool flip = false;
    if (a < 0) {
        a = -a;
        flip = true;
    }

    if (flip) {
        assert(a != 0); // A1
        glob /= a;
    }
    return;
}
```



MSG Example

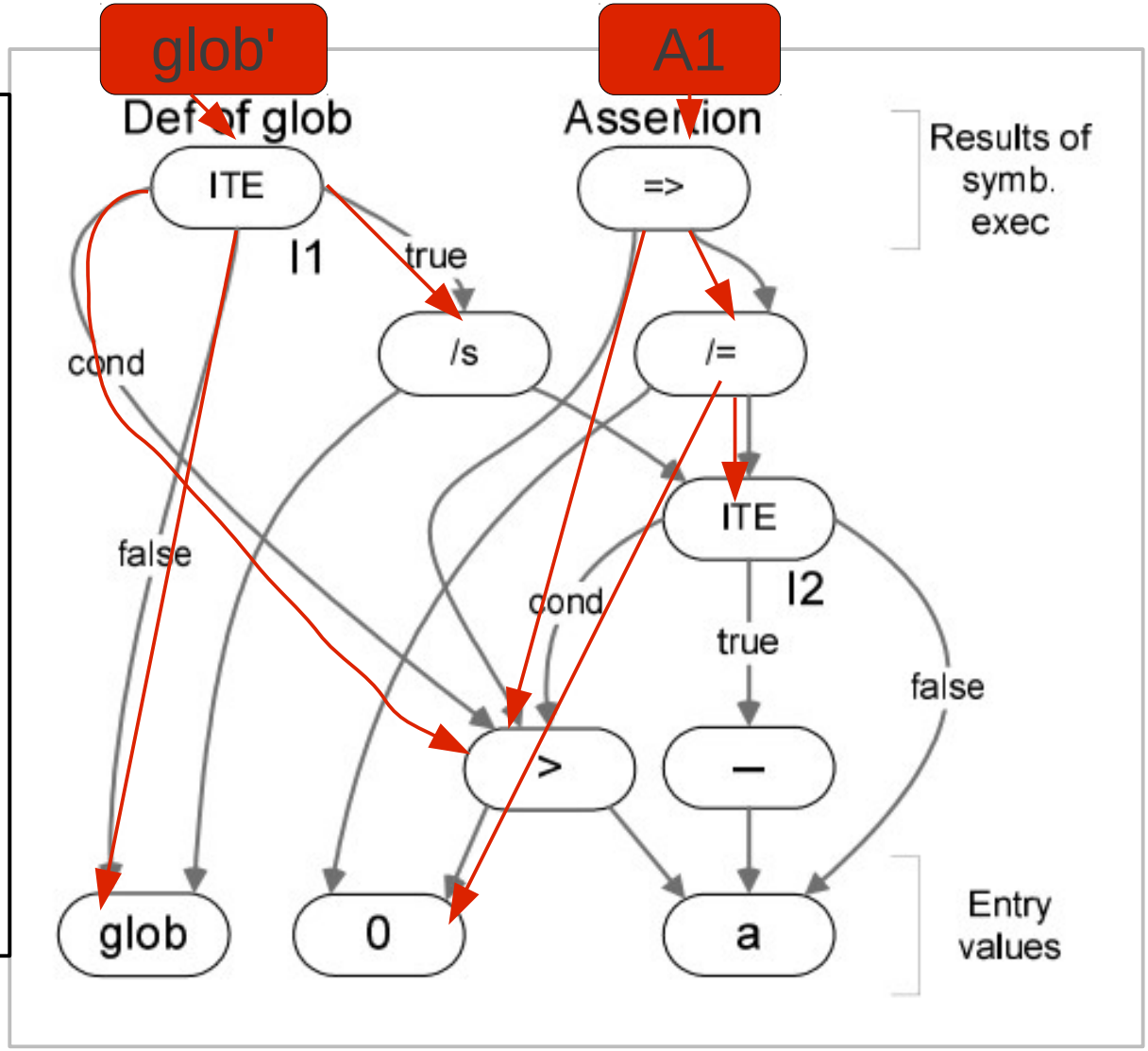
```
int glob; // Global
...
void f(int a) {
    bool flip = false;
    if (a < 0) {
        a = -a;
        flip = true;
    }

    if (flip) {
        assert(a != 0); // A1
        glob /= a;
    }
    return;
}
```



Now remove all inner nodes corresp. to non-visible variables

MSG Example

```
int glob; // Global
...
void f(int a) {
    bool flip = false;
    if (a < 0) {
        a = -a;
        flip = true;
    }

    if (flip) {
        assert(a != 0); // A1
        glob /= a;
    }
    return;
}
```



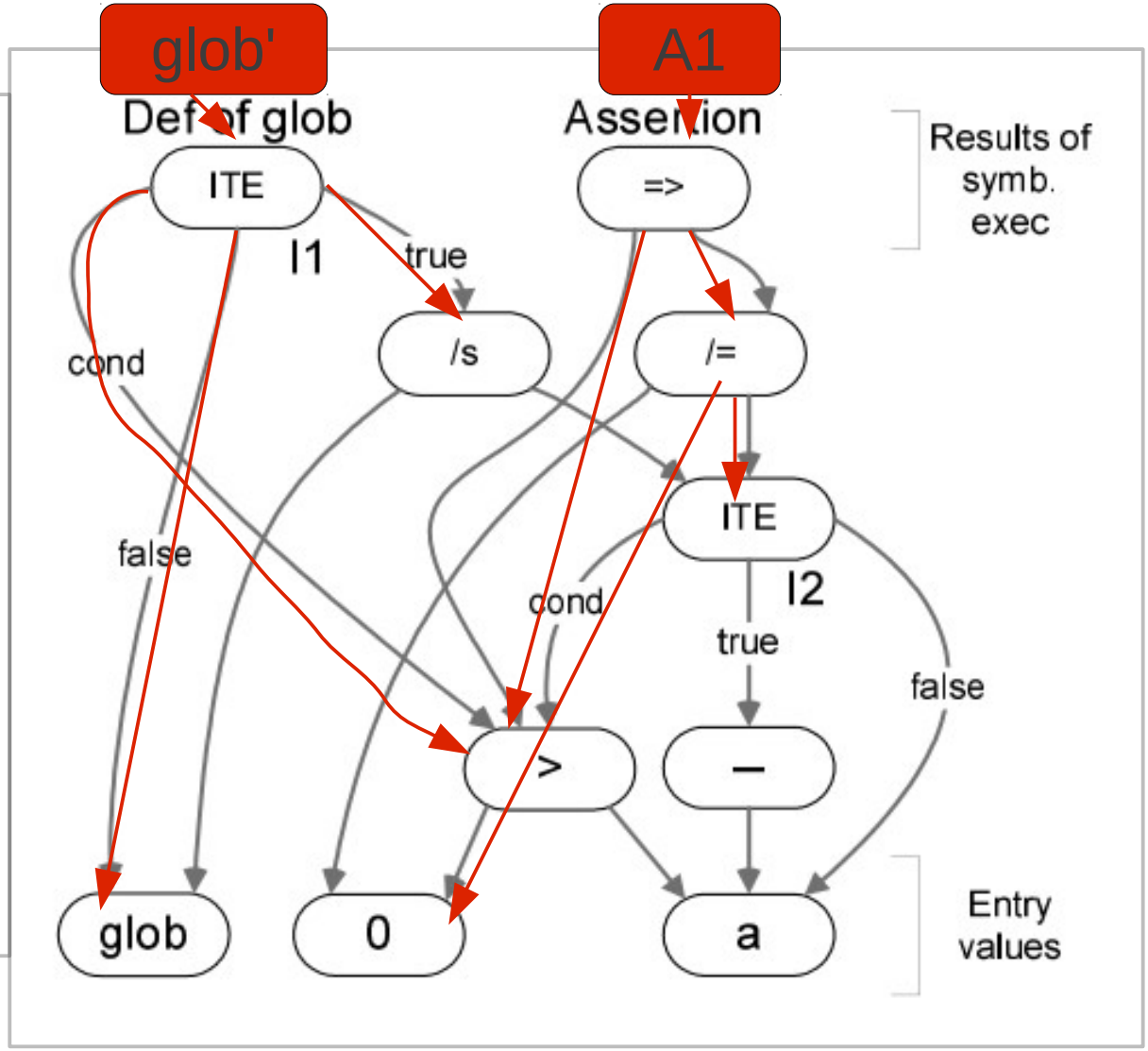
Structural Abstraction

- Inline small function calls into MSG
  - Heuristic is inline bar() if |MSG(bar)| < 50
- For others, substitute a placeholder node
- What if we called f(h())?

Structural Abstraction

We call f(h())

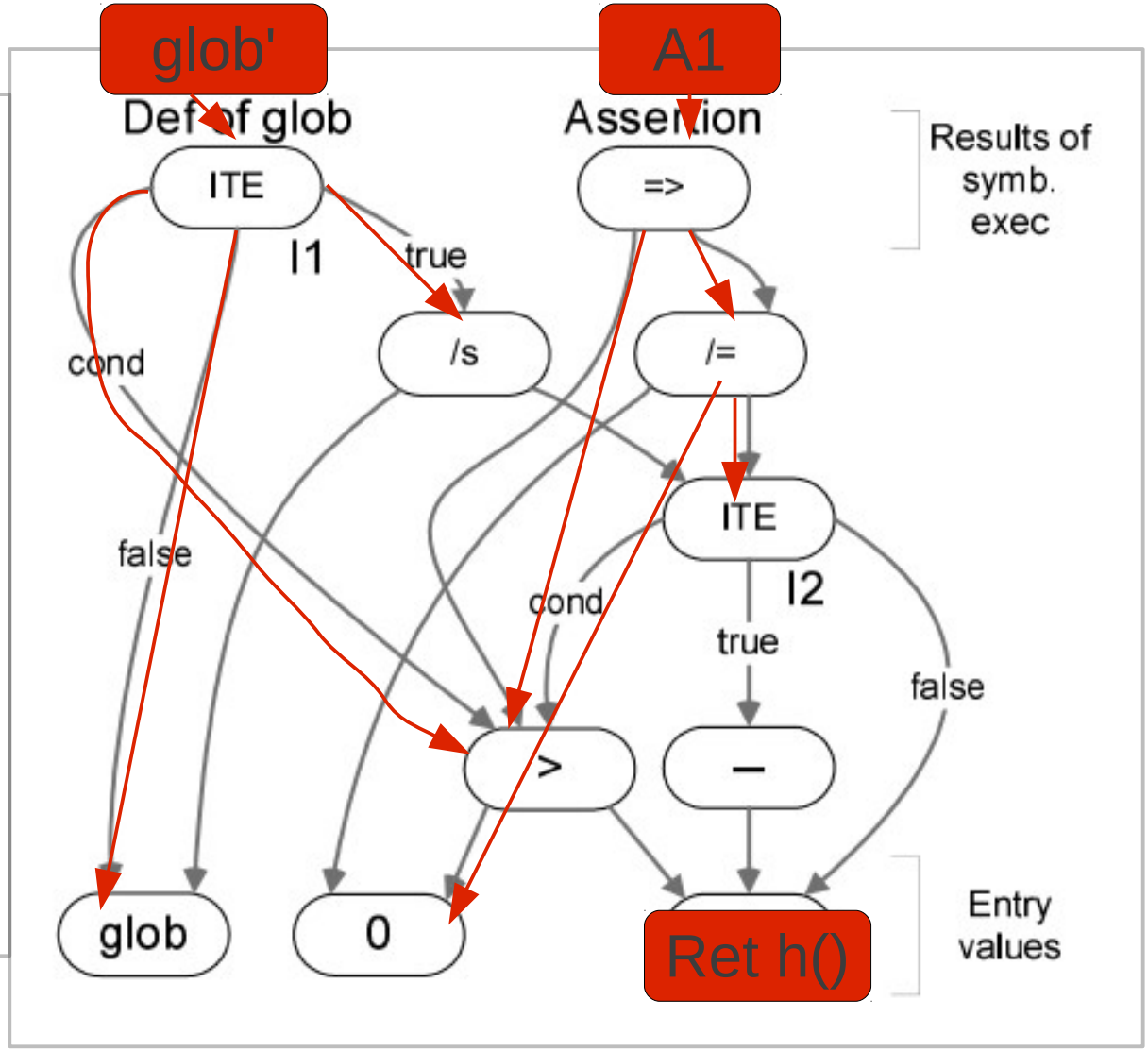
```
int glob; // Global
...
void f(int a) {
    bool flip = false;
    if (a < 0) {
        a = -a;
        flip = true;
    }

    if (flip) {
        assert(a != 0); // A1
        glob /= a;
    }
    return;
}
```



Structural Abstraction

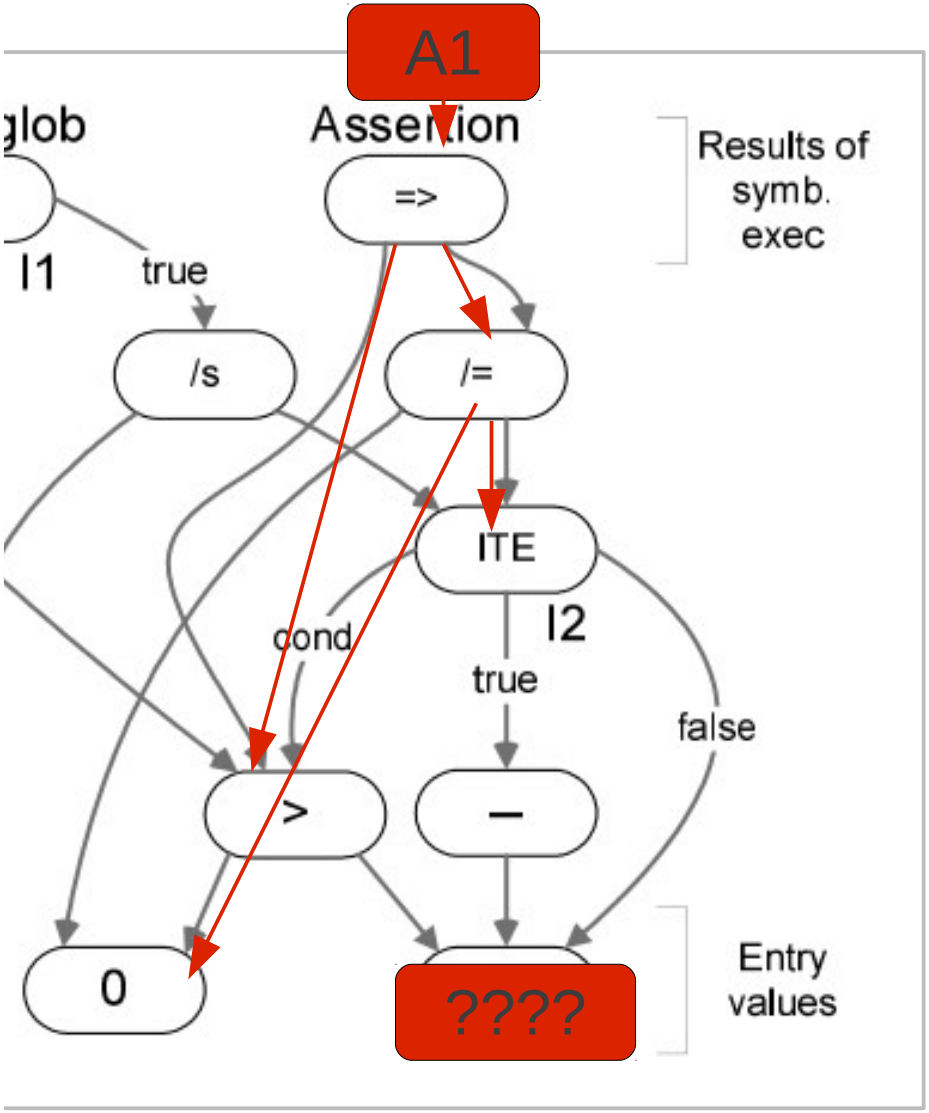
We call f(h())

```
int glob; // Global
...
void f(int a) {
    bool flip = false;
    if (a < 0) {
        a = -a;
        flip = true;
    }

    if (flip) {
        assert(a != 0); // A1
        glob /= a;
    }
    return;
}
```



Structural Abstraction

- Substitute unconstrained value for placeholder
- Try to prove A1's VC
  - We succeed
    - Didn't even explore MSG for h()!
- If we fail, inline placeholder with subgraph of h()'s MSG
  - Rinse, repeat
- If we fail and no more placeholders, we found a bug!



One More Thing...

- Nonlinear constraints modeled as circuits
- Use a linear-time (I think?) transform
- SAT solvers work best with area-optimal encoding
  - In general, finding area-optimal circuit is NP-hard
  - But many nonlinear operators generate acyclic graphs easy to decompose into forest of trees
    - Area minimization of tree is polynomial-time
  - Even for general graphs, good heuristics exist
    - Could apply technology mapping for LUT-based FPGAs
    - LUT sizes dynamically tuned to fit entire LUT in cache
    - “Heuristics for Area Minimization in LUT-based FPGA Technology Mapping” V. Manohararajah et al.

27