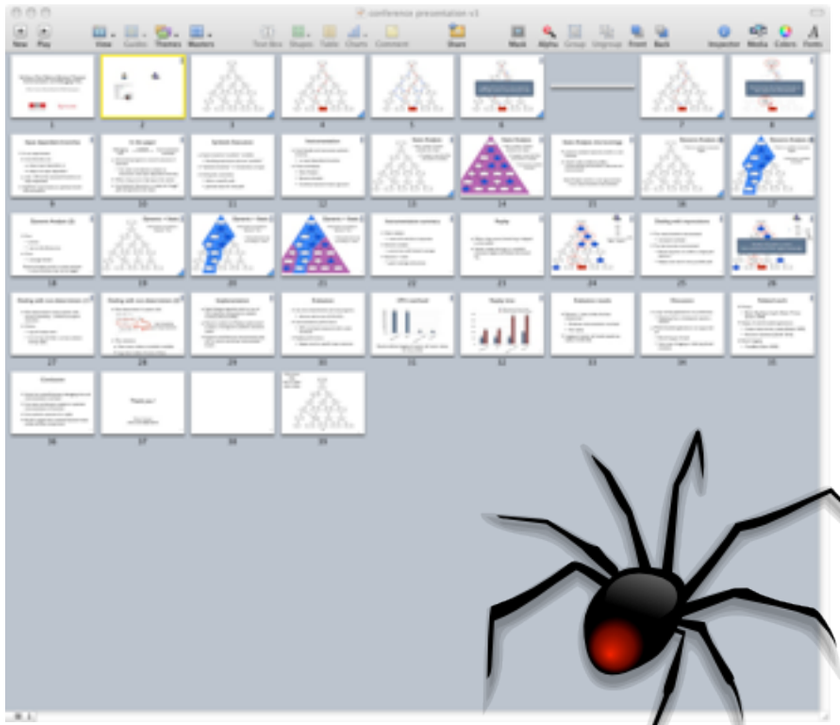
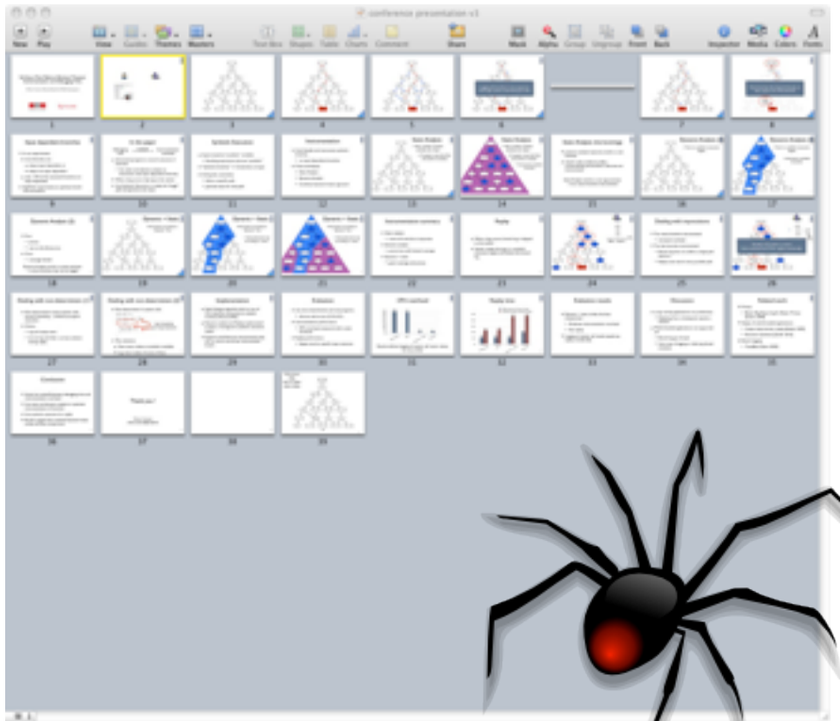
Striking a New Balance Between Program Instrumentation and Debugging Time

Olivier Crameri, Ricardo Bianchini, Willy Zwaenepoel

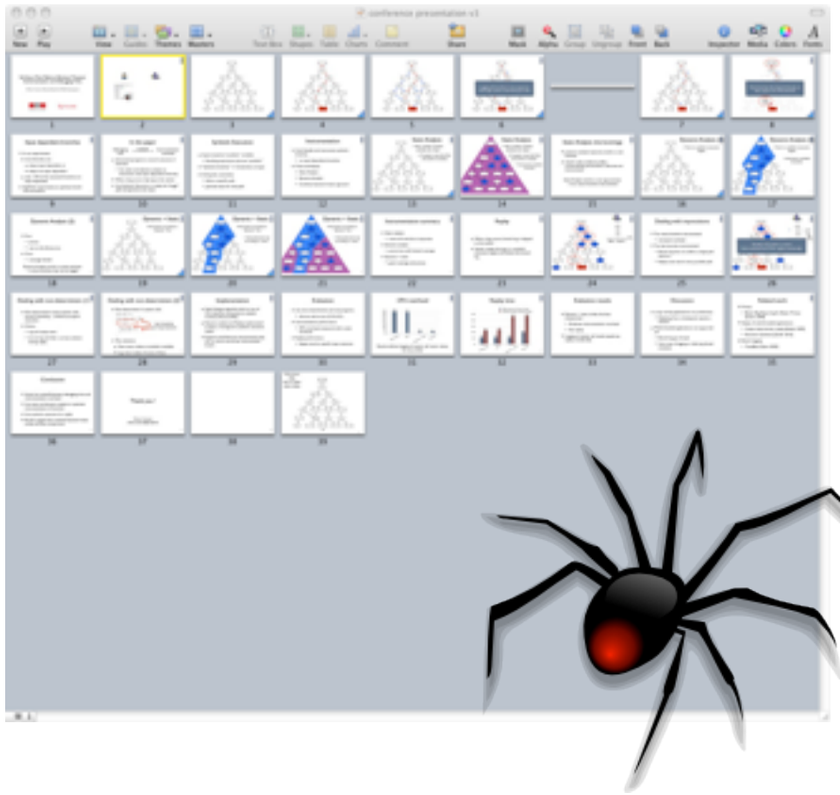
ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

RUTGERS



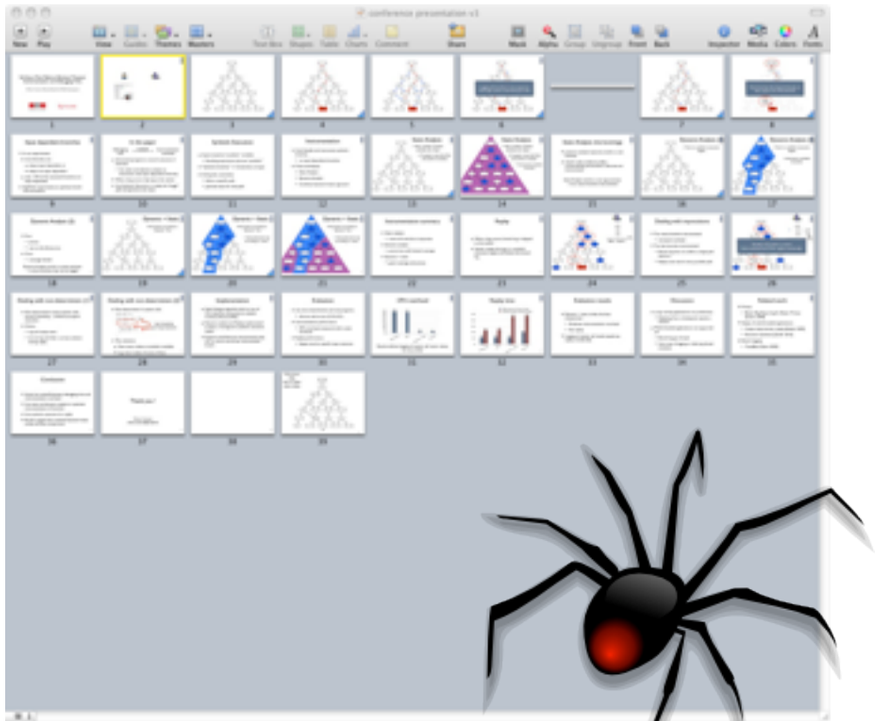


Bug report



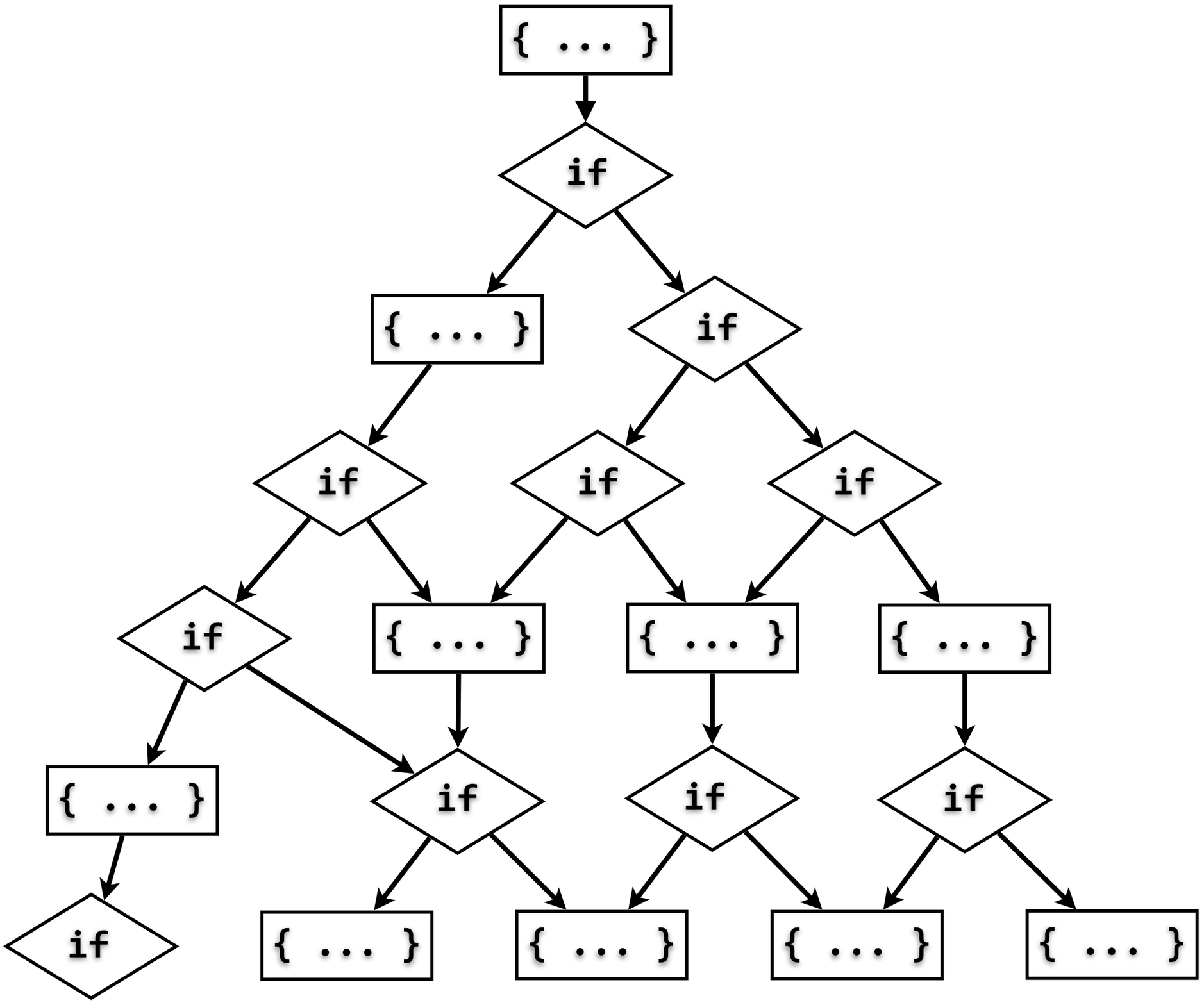
Bug report

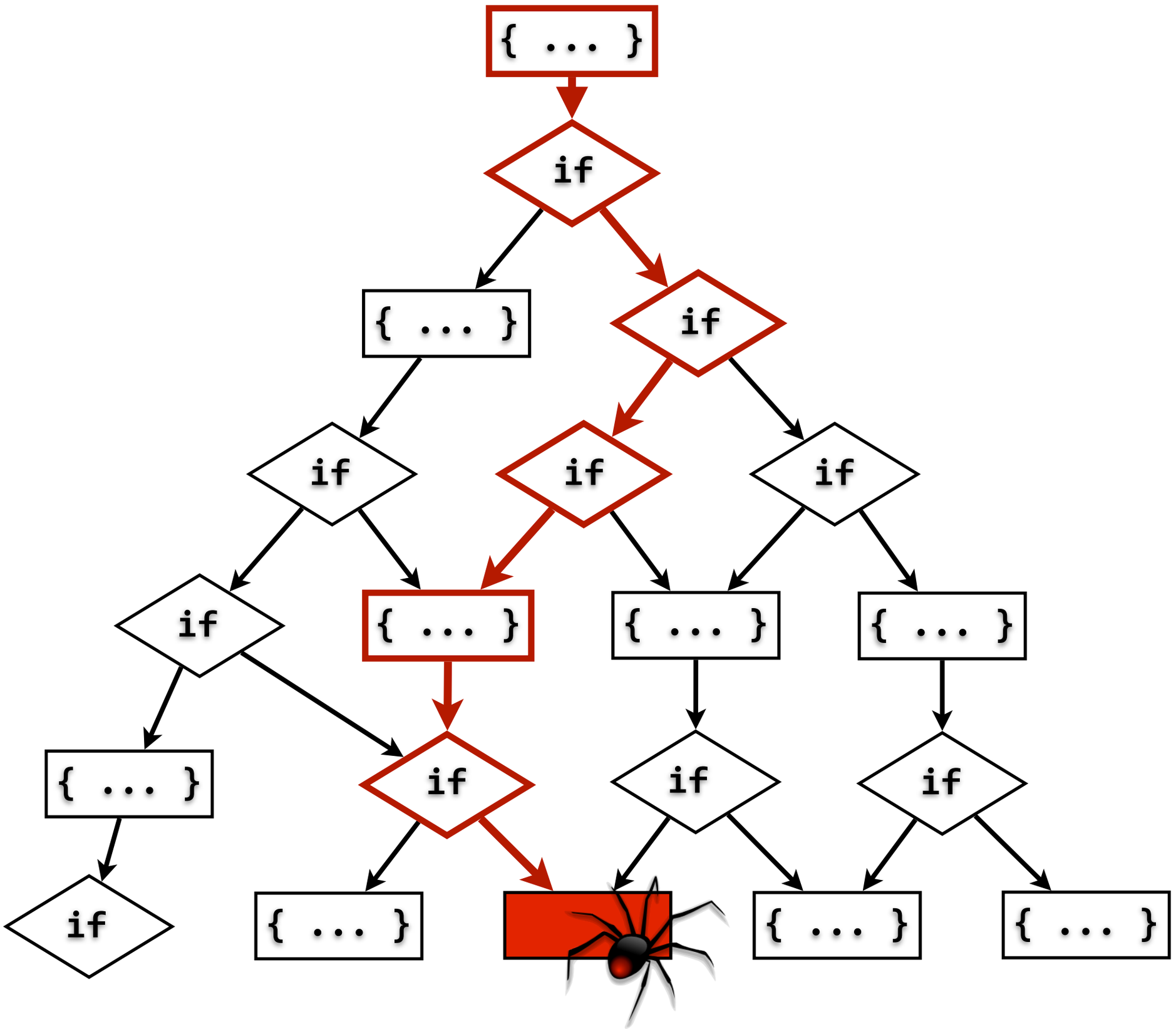
- Instrumentation ⇒ Better bug report ⇒ Easy debugging

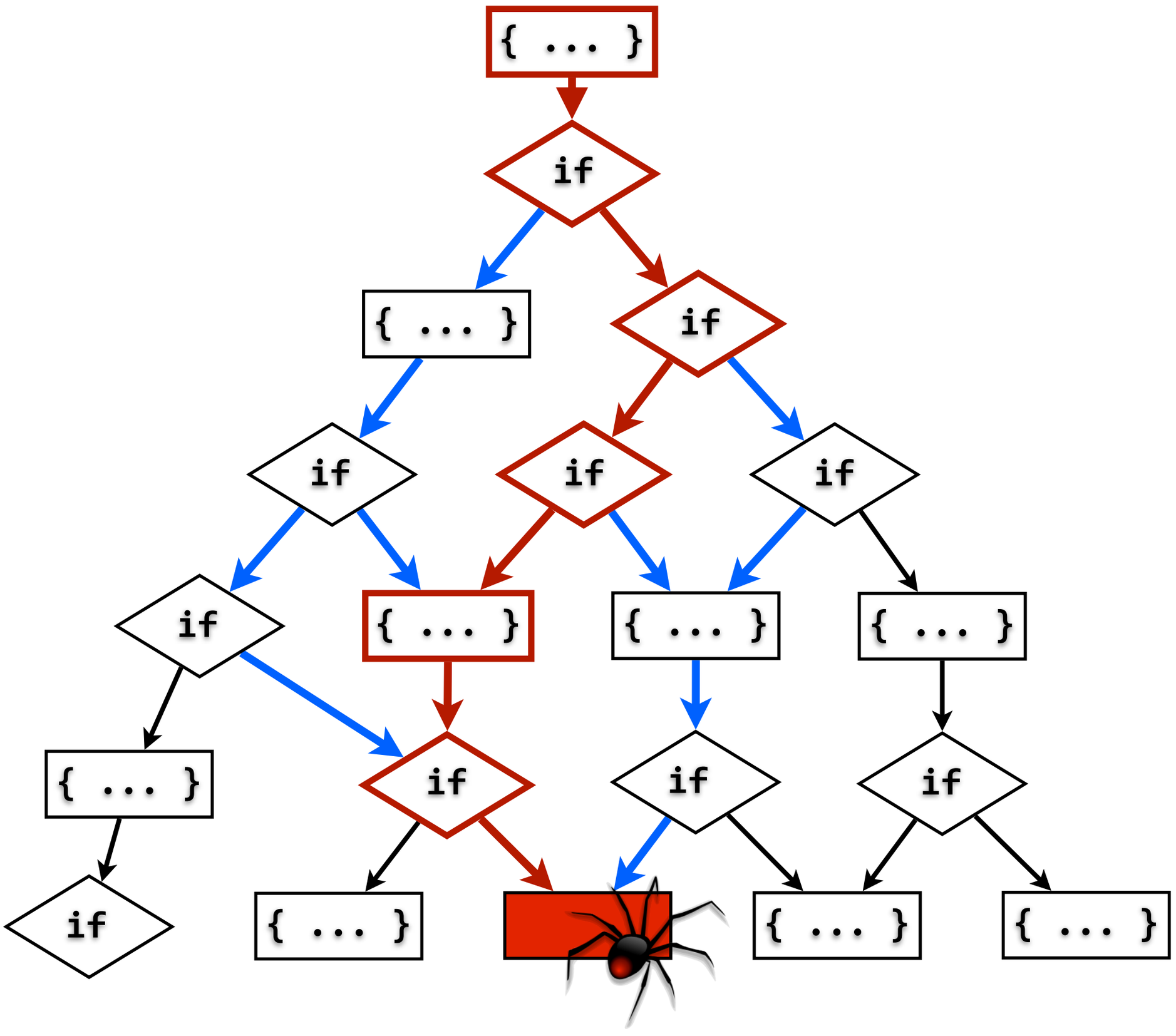


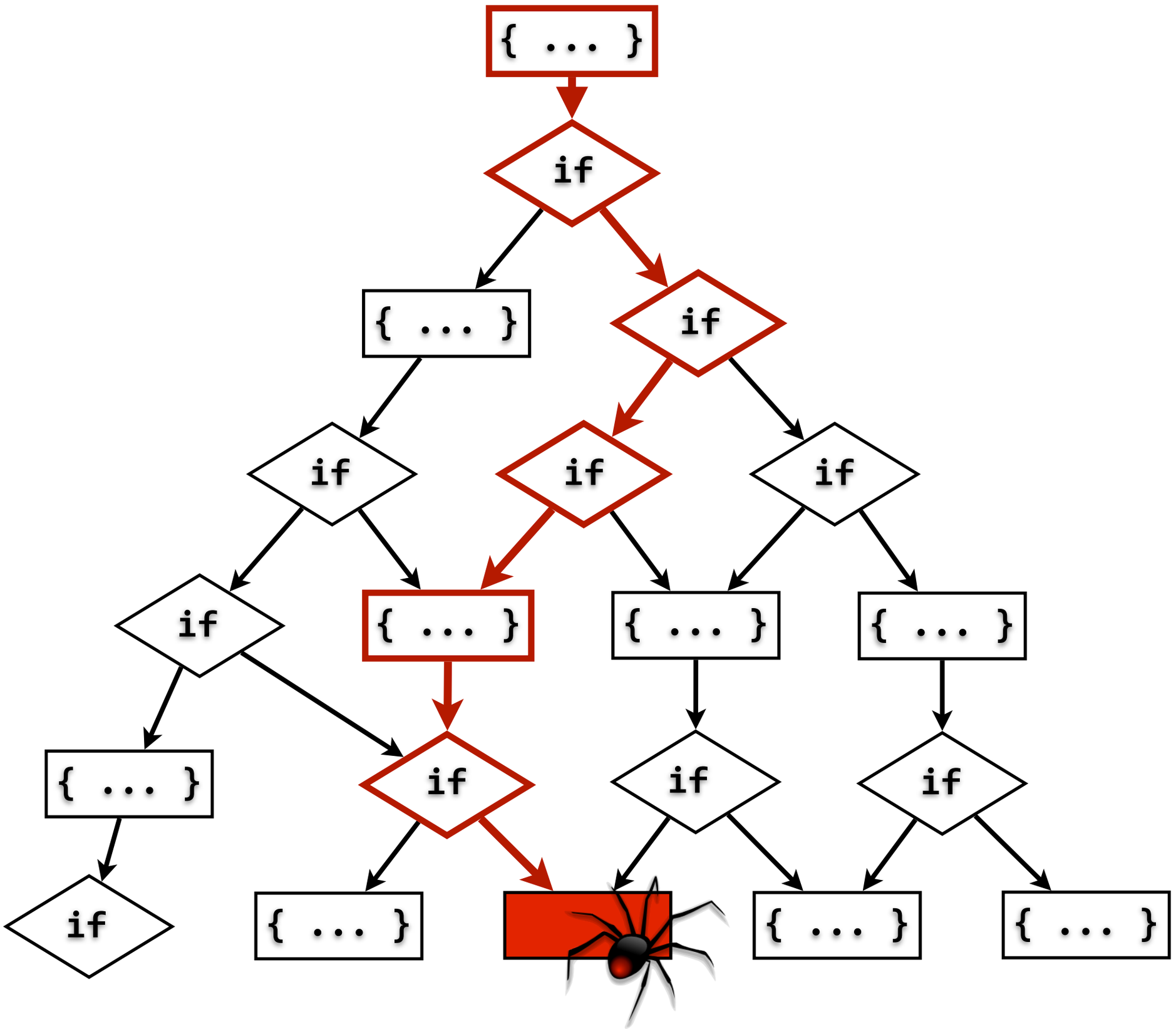
Bug report

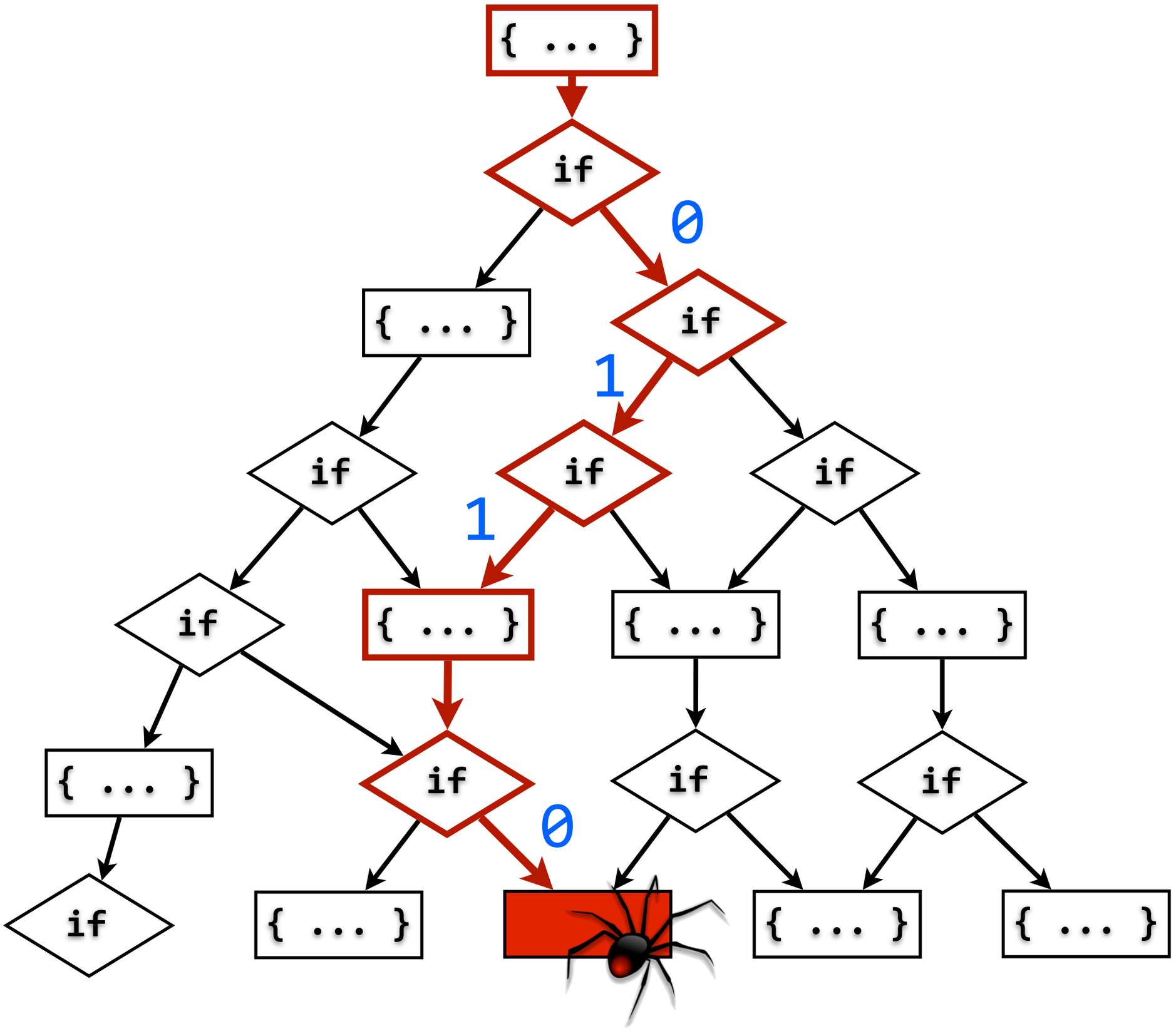
- Instrumentation ⇒ Better bug report ⇒ Easy debugging
- Instrumentation ⇒ Overhead for the user

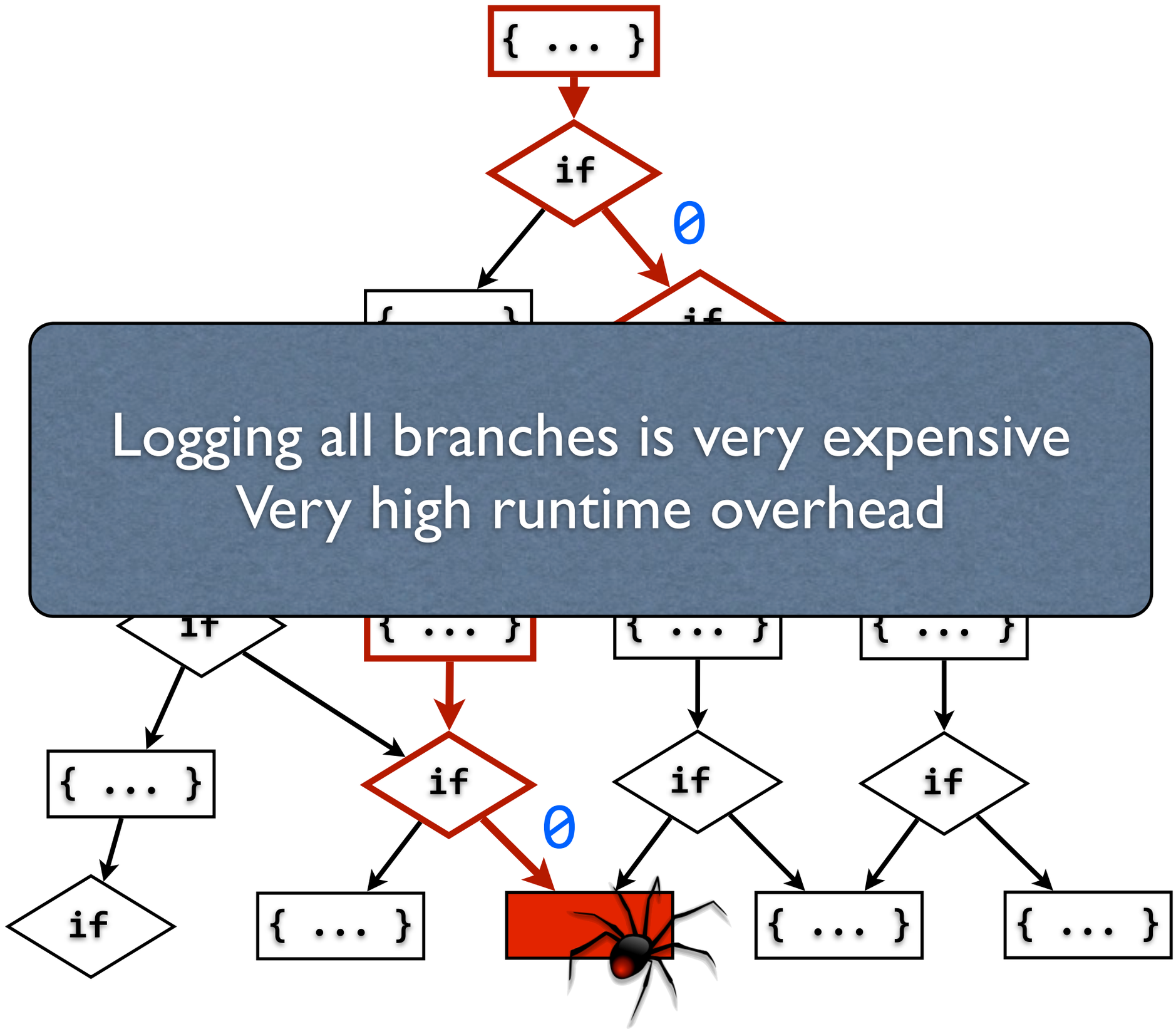

{ ... }
if
{ ... }
if
if
if
if
if
{ ... }
{ ... }
{ ... }
{ ... }
if
if
if
{ ... }
{ ... }
{ ... }
{ ... }
if
{ ... }
{ ... }
{ ... }
{ ... }


{ ... }
if
{ ... }
if
if
if
if
if
{ ... }
{ ... }
{ ... }
{ ... }
if
if
if
{ ... }
{ ... }
{ ... }
{ ... }
if


{ ... }
if
{ ... }
if
if
if
if
if
{ ... }
{ ... }
{ ... }
{ ... }
if
if
if
{ ... }
{ ... }
{ ... }
{ ... }


{ ... }
if
{ ... }
if
if
if
if
if
{ ... }
{ ... }
{ ... }
{ ... }
if
if
if
{ ... }
{ ... }
{ ... }
{ ... }
if


{ ... }
if
0
{ ... }
if
1
if
if
if
1
if
{ ... }
{ ... }
{ ... }
if
if
if
0
{ ... }
{ ... }
{ ... }
{ ... }



Logging all branches is very expensive
Very high runtime overhead

In this paper

Debugging time

Instrumentation overhead

In this paper

Debugging time ⟷ *tradeoff* ⟷ Instrumentation overhead

- Instrument program to record branches
  - ✦ Static and dynamic analysis to minimize instrumentation

In this paper

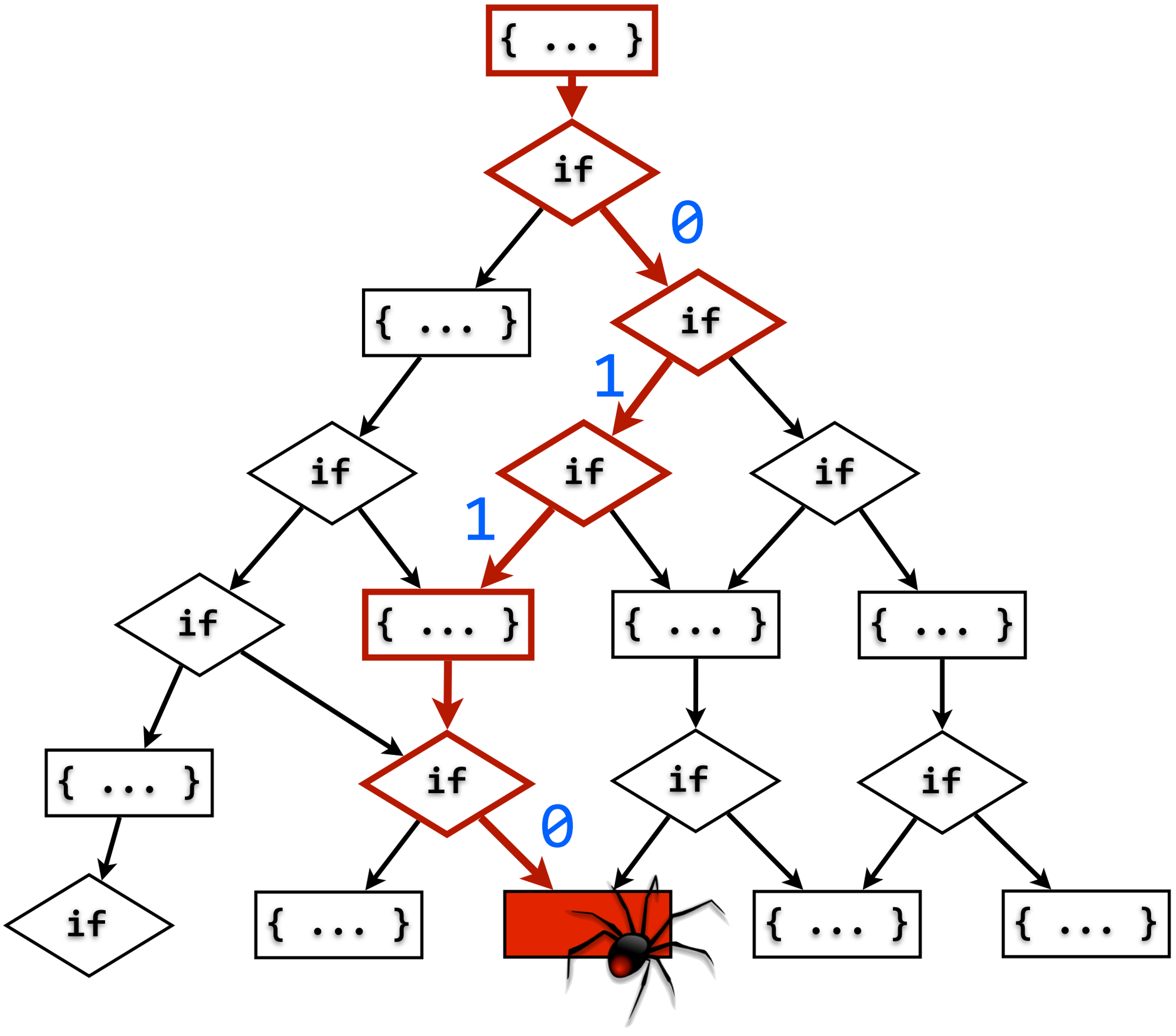
Debugging time ⟷ *tradeoff* ⟷ Instrumentation overhead

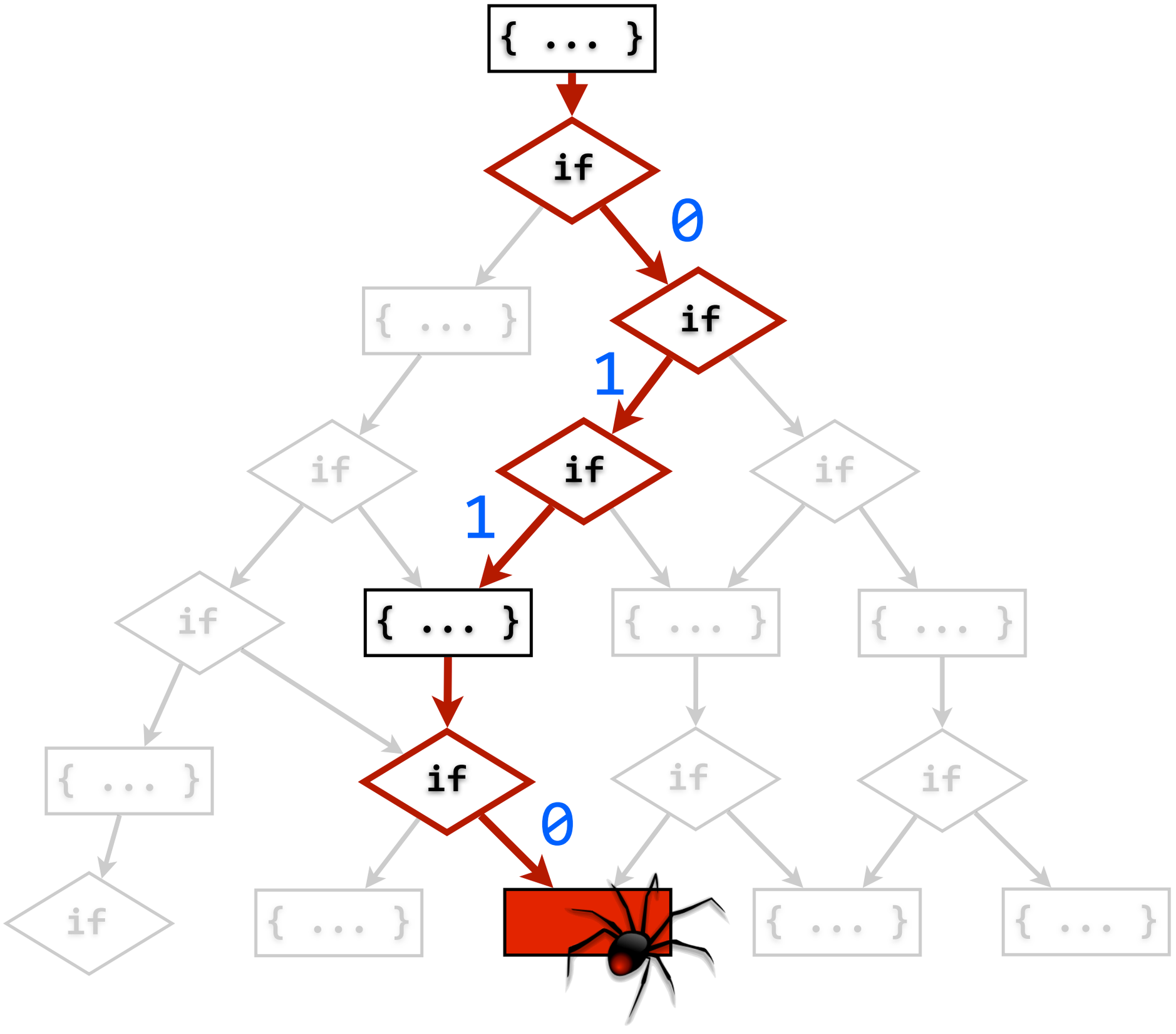
- Instrument program to record branches
  - ✦ Static and dynamic analysis to minimize instrumentation
- Bug ⇒ ship branch log to the vendor

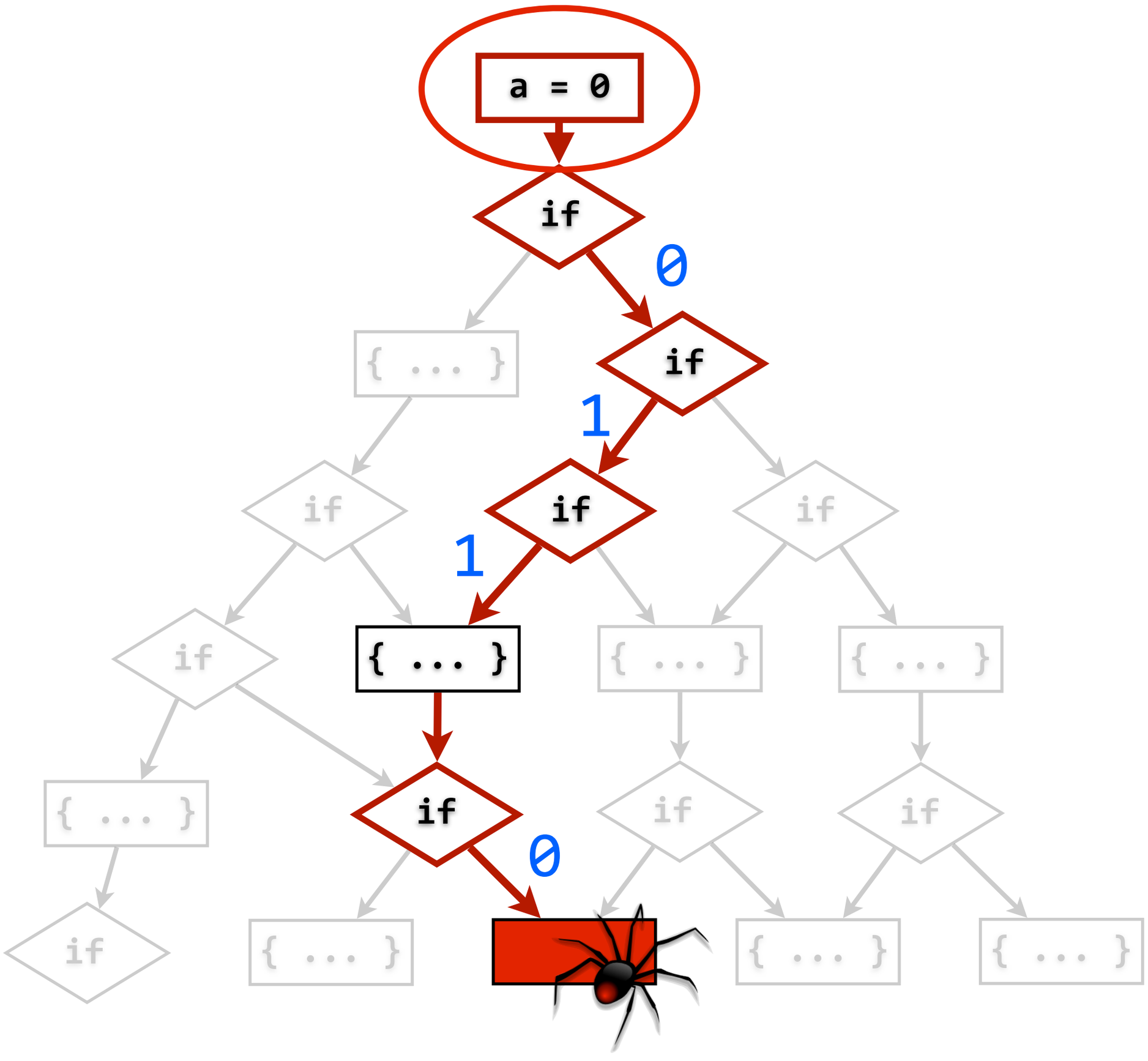
In this paper

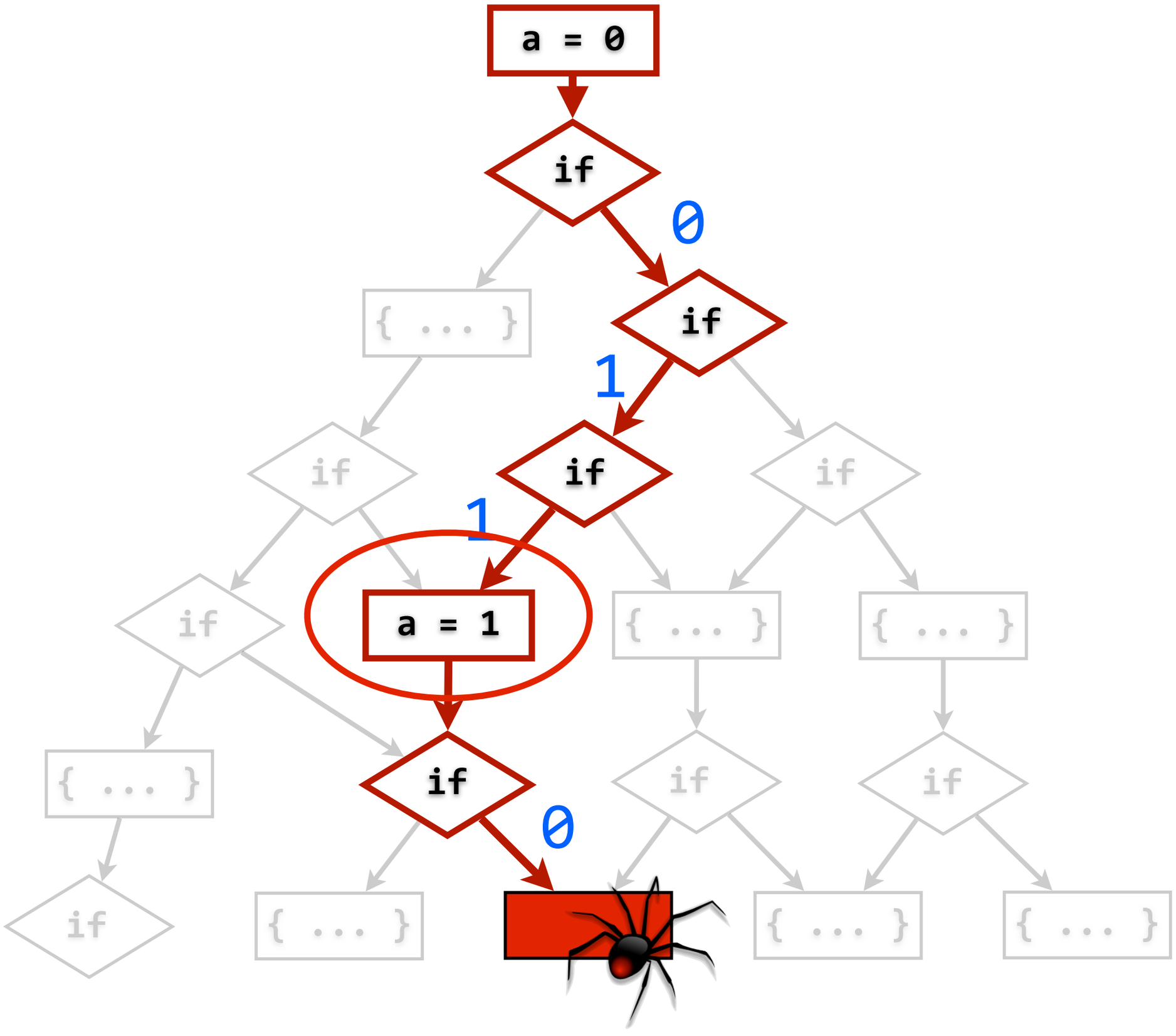
Debugging time ⟷ *tradeoff* ⟷ Instrumentation overhead

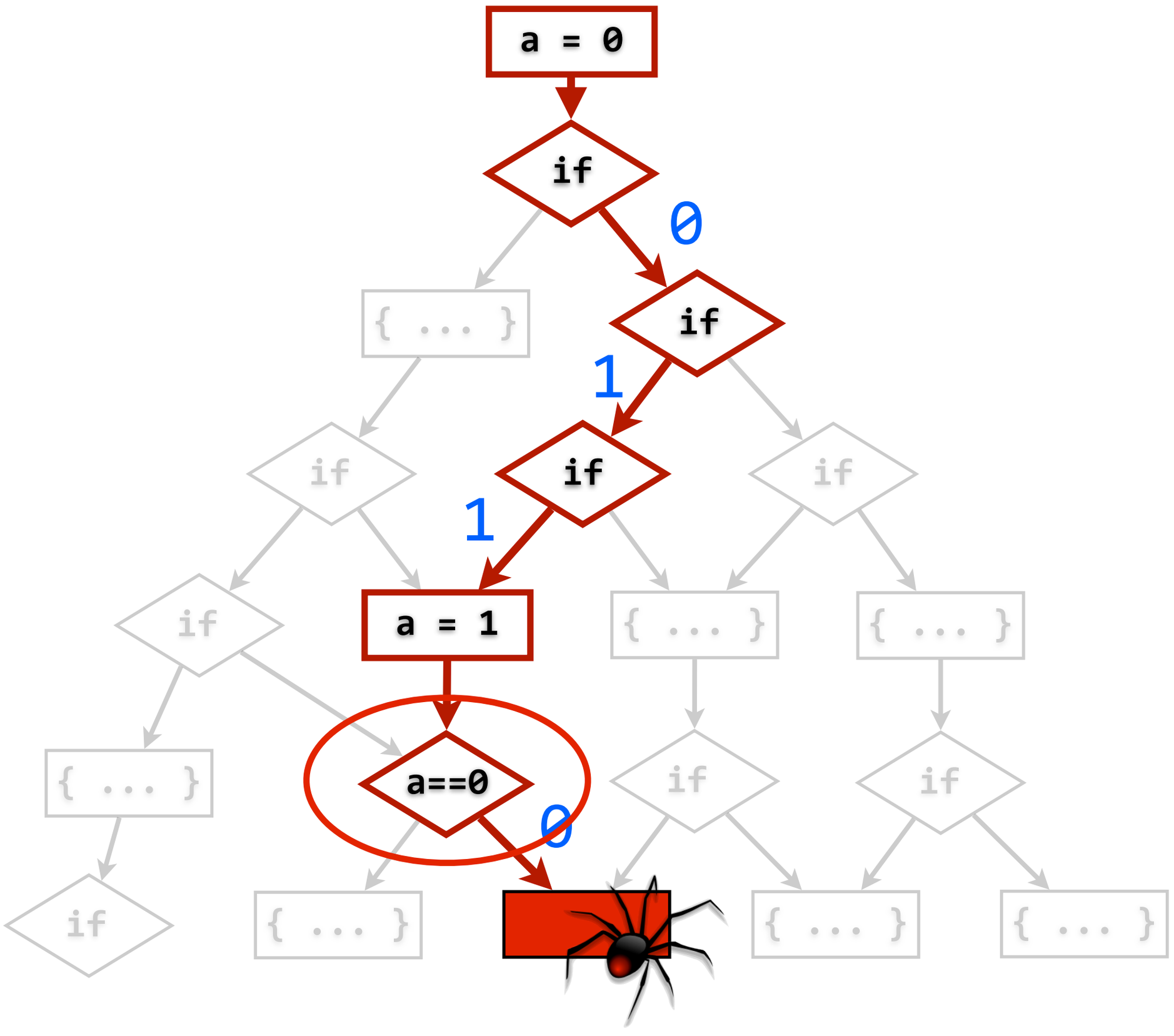
- Instrument program to record branches
  - ✦ Static and dynamic analysis to minimize instrumentation
- Bug ⇒ ship branch log to the vendor
- Symbolic Execution to replay the "buggy" path and generate new input

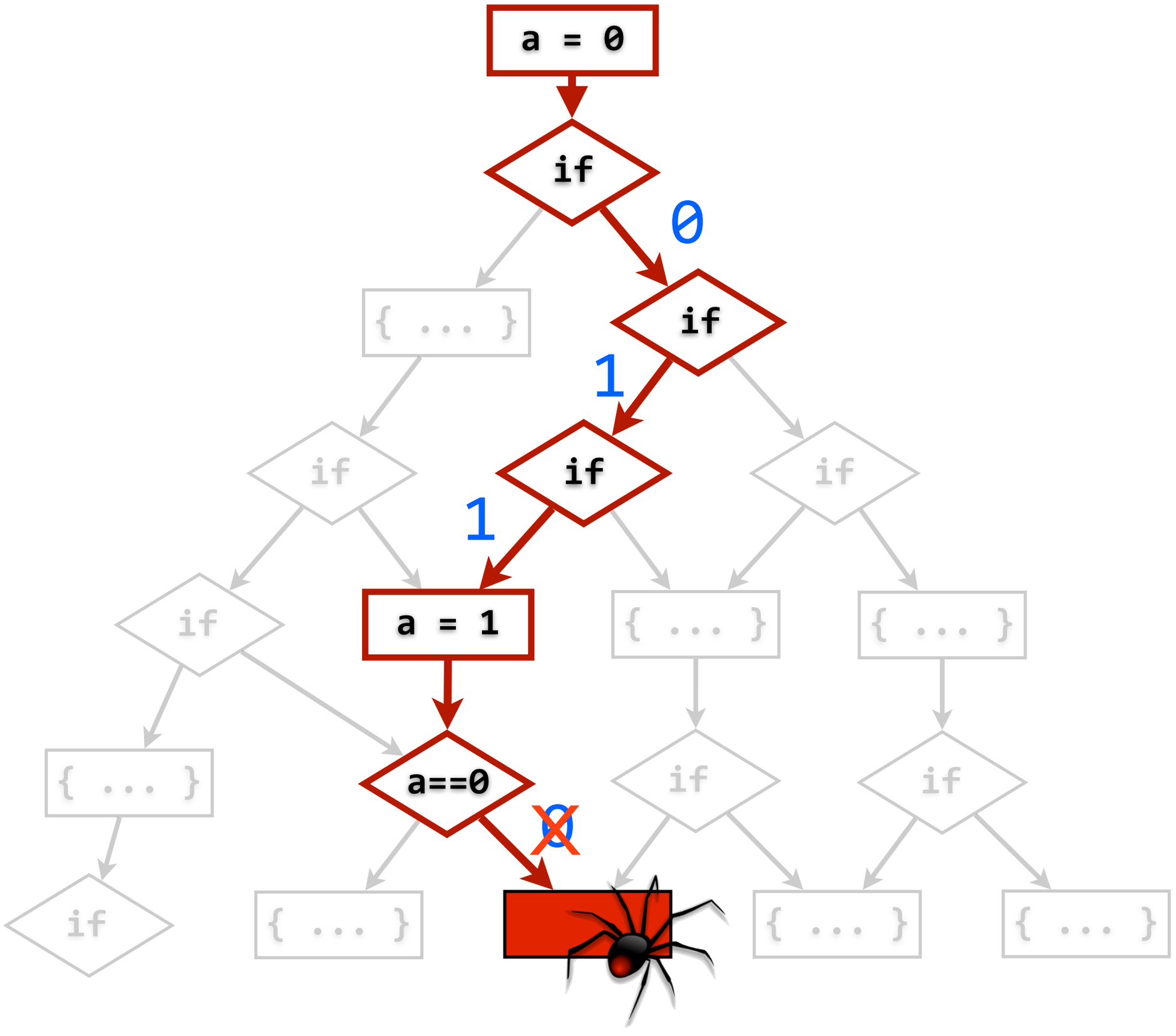

{ ... }
if
0
{ ... }
if
1
if
if
if
1
if
{ ... }
{ ... }
{ ... }
if
if
if
{ ... }
if
0
{ ... }
{ ... }
{ ... }

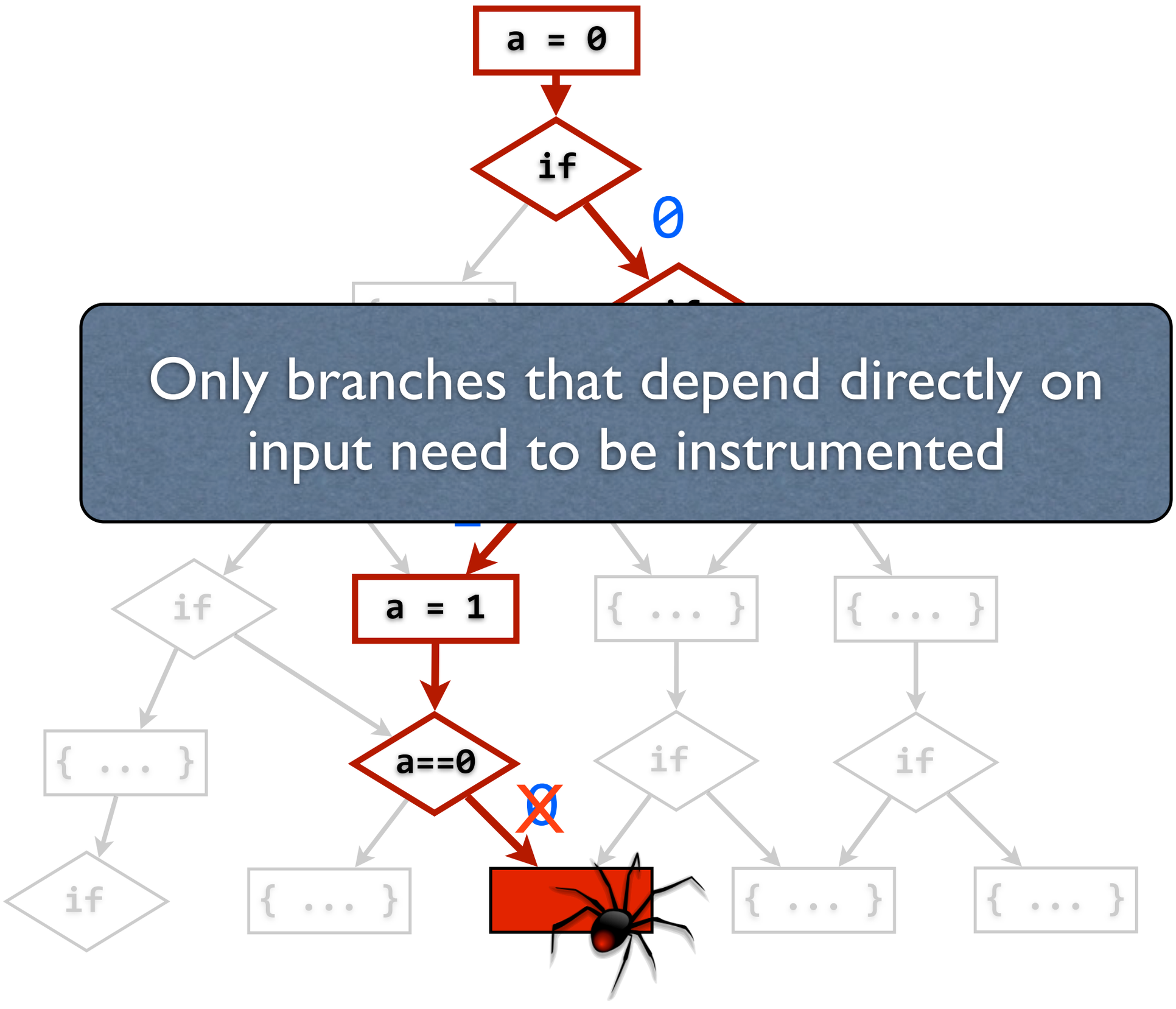

{ ... }
if
0
if
{ ... }
1
if
if
if
1
{ ... }
if
{ ... }
{ ... }
if
if
if
0
{ ... }
{ ... }
{ ... }
{ ... }

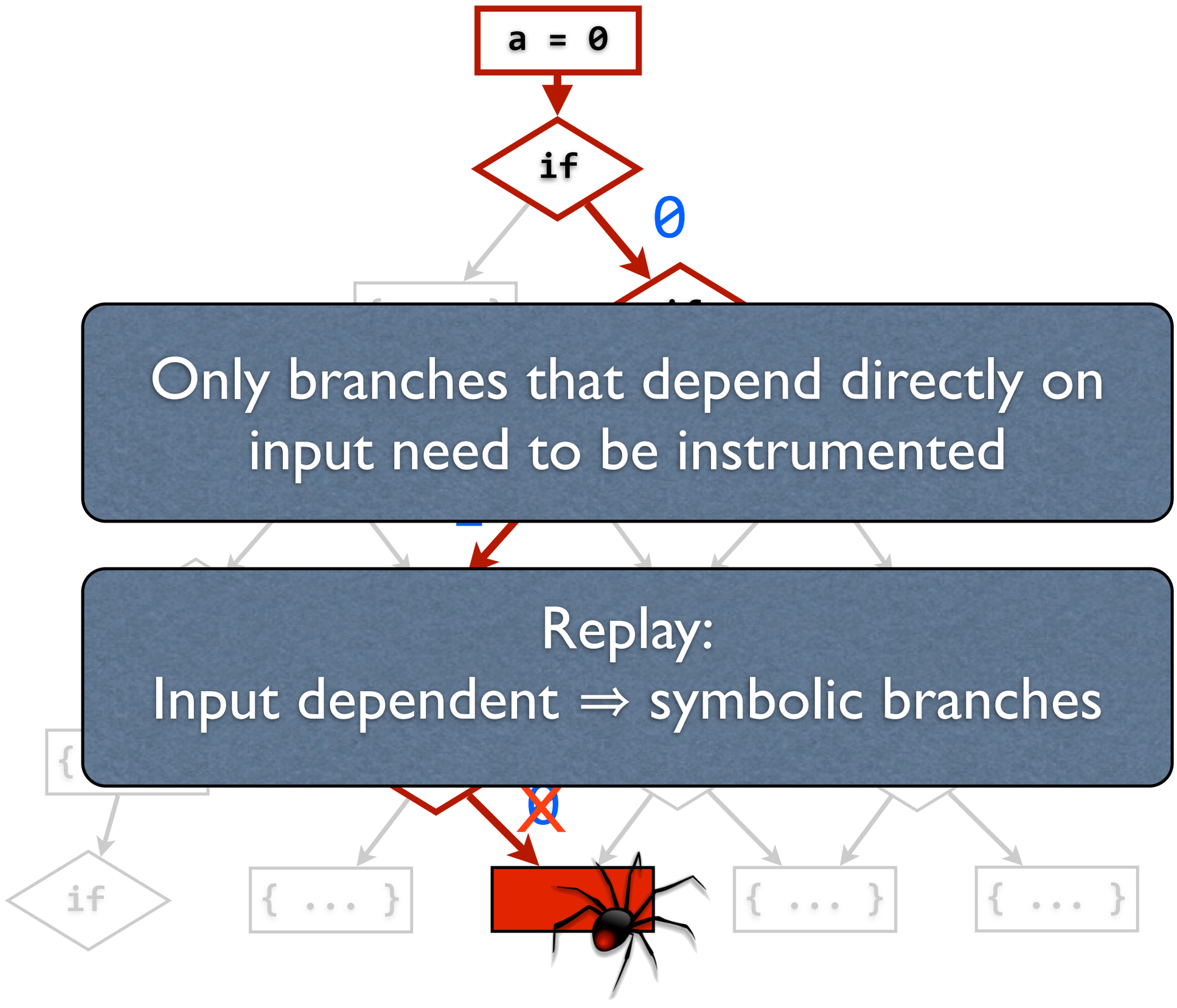

a = 0
if
0
{ ... }
if
1
if
if
if
1
if
{ ... }
{ ... }
{ ... }
if
if
if
{ ... }
if
0
{ ... }
{ ... }
{ ... }


a = 0
if
0
{ ... }
if
1
if
if
if
1
if
a = 1
{ ... }
{ ... }
if
{ ... }
if
if
0
{ ... }
{ ... }
{ ... }


a = 0
if
0
{ ... }
if
1
if
if
if
1
if
{ ... }
a = 1
{ ... }
{ ... }
{ ... }
a==0
if
if
0
if
{ ... }
{ ... }
{ ... }


a = 0
if
0
{ ... }
if
1
if
if
if
1
if
{ ... }
a = 1
{ ... }
{ ... }
{ ... }
a==0
if
if
0
{ ... }
{ ... }
{ ... }
{ ... }


a = 0
if
0
Only branches that depend directly on input need to be instrumented
if
a = 1
{ ... }
{ ... }
{ ... }
a==0
if
if
0
if
{ ... }
{ ... }
{ ... }
{ ... }


a = 0
if
0
Only branches that depend directly on input need to be instrumented
Replay:
Input dependent ⇒ symbolic branches
if
{ ... }
{ ... }
{ ... }

Symbolic branches

Symbolic branches

- In our experiments:

Symbolic branches

- In our experiments:
  - only ~10% of the branches

Symbolic branches

- In our experiments:
  - only ~10% of the branches
  - most branches are:
    - always symbolic
    - always concrete

Symbolic branches

- In our experiments:
  - Significant opportunity to optimize branch instrumentation
  - always concrete

Instrumentation

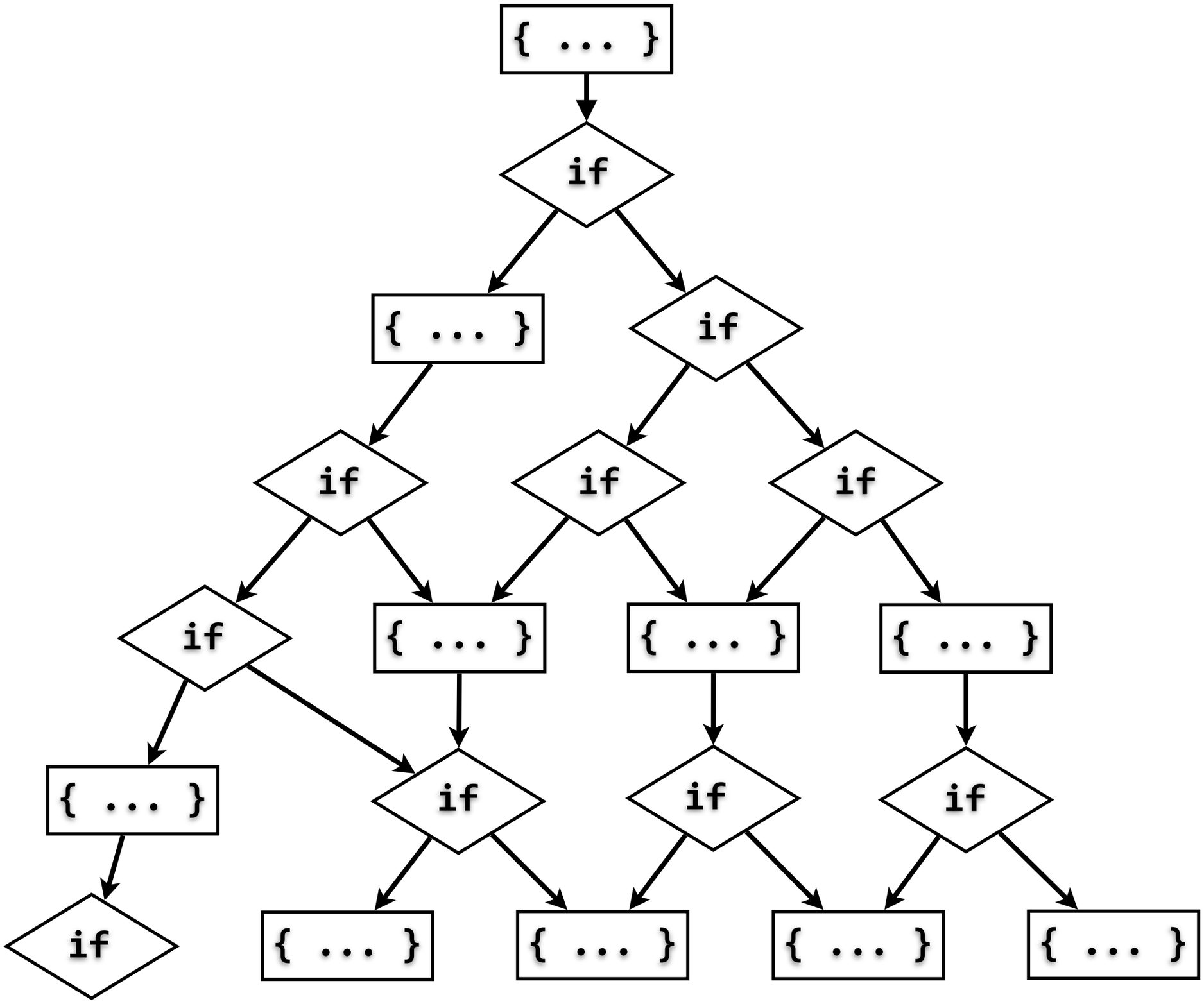
Instrumentation

- Goal:
  - ✦ Instrument symbolic branches

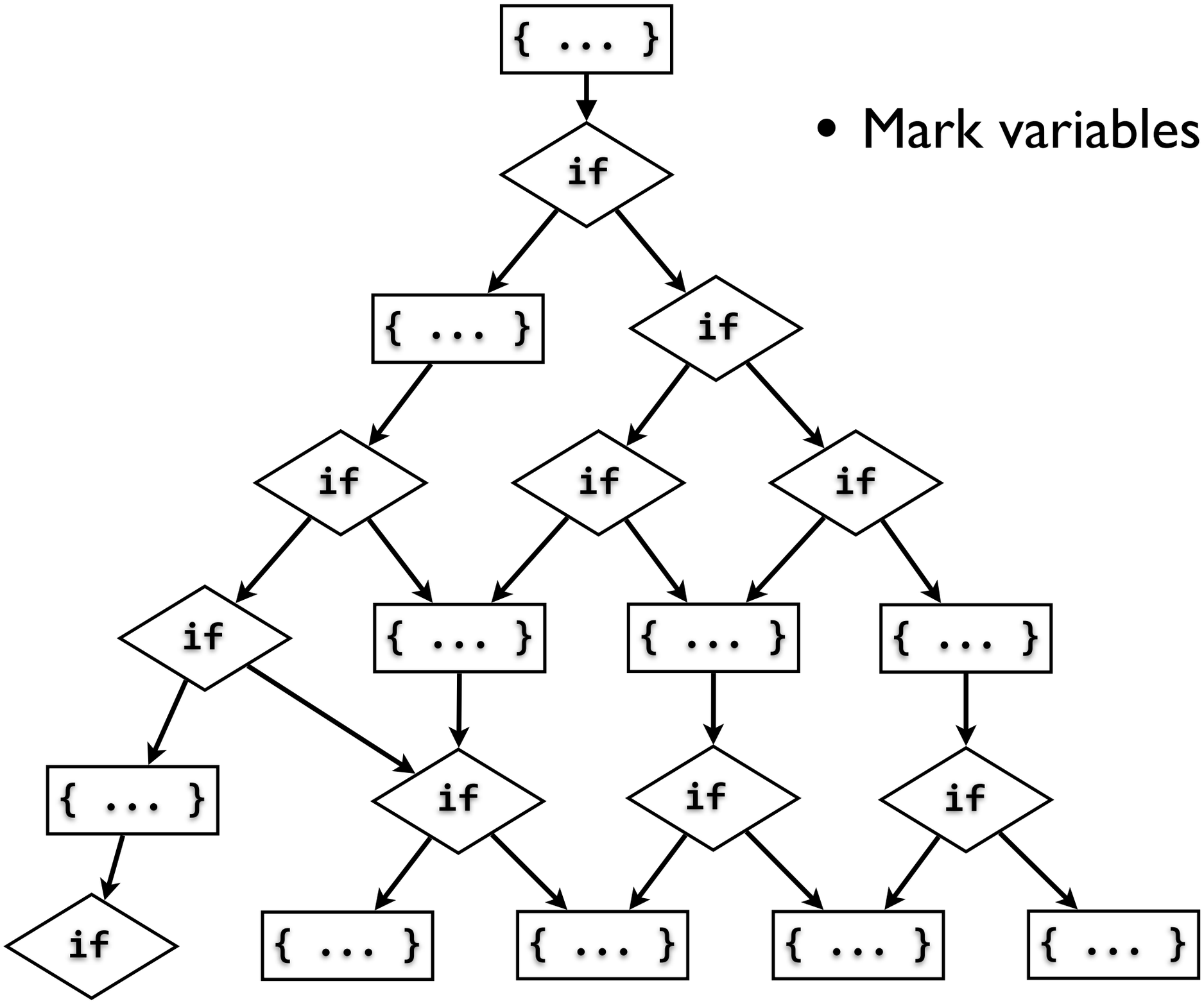
Instrumentation

- Goal:
  - ✦ Instrument symbolic branches
- Three techniques:
  - ✦ Static Analysis
  - ✦ Dynamic Analysis
  - ✦ Combined dynamic+static approach

Static Analysis

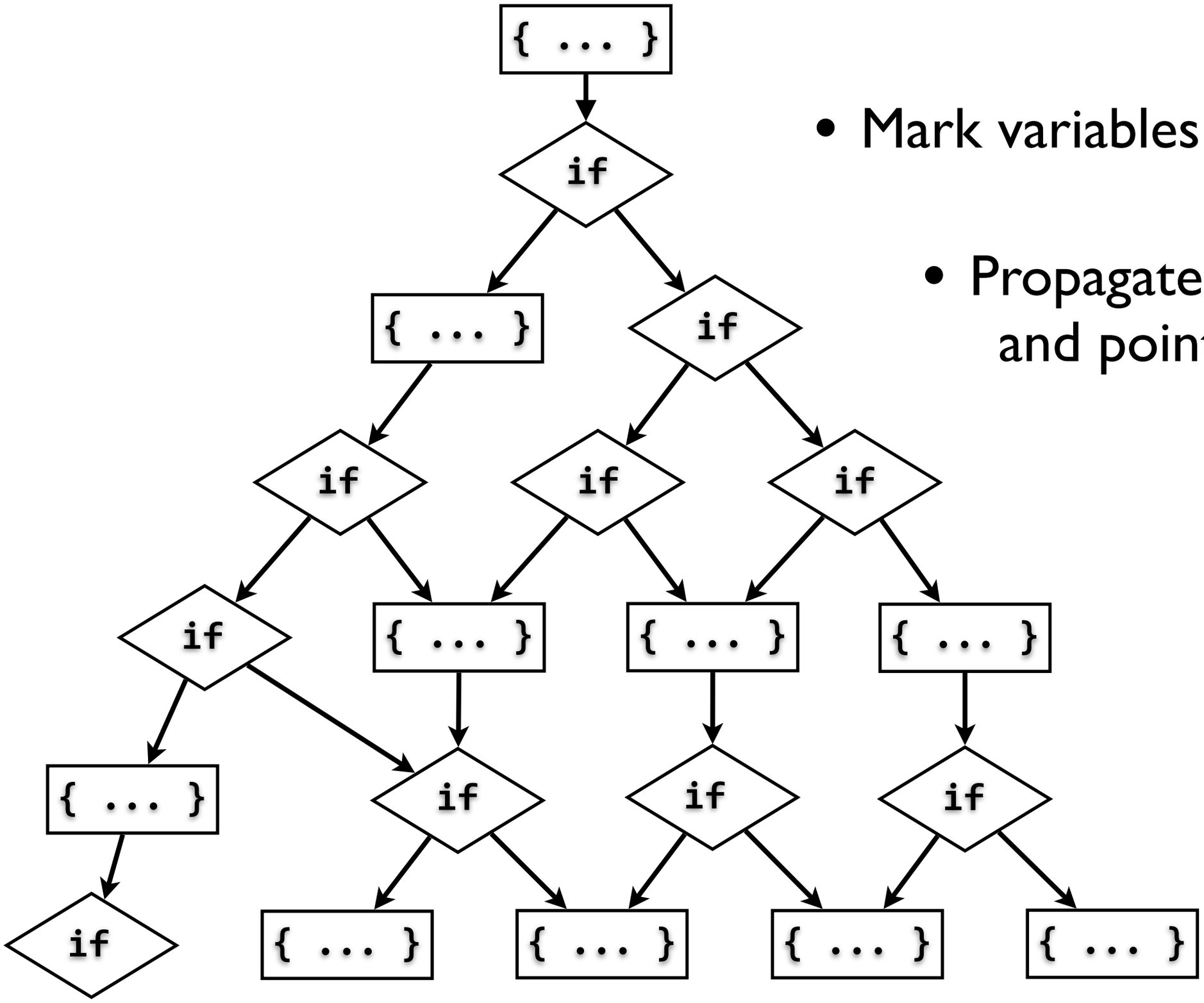


Static Analysis

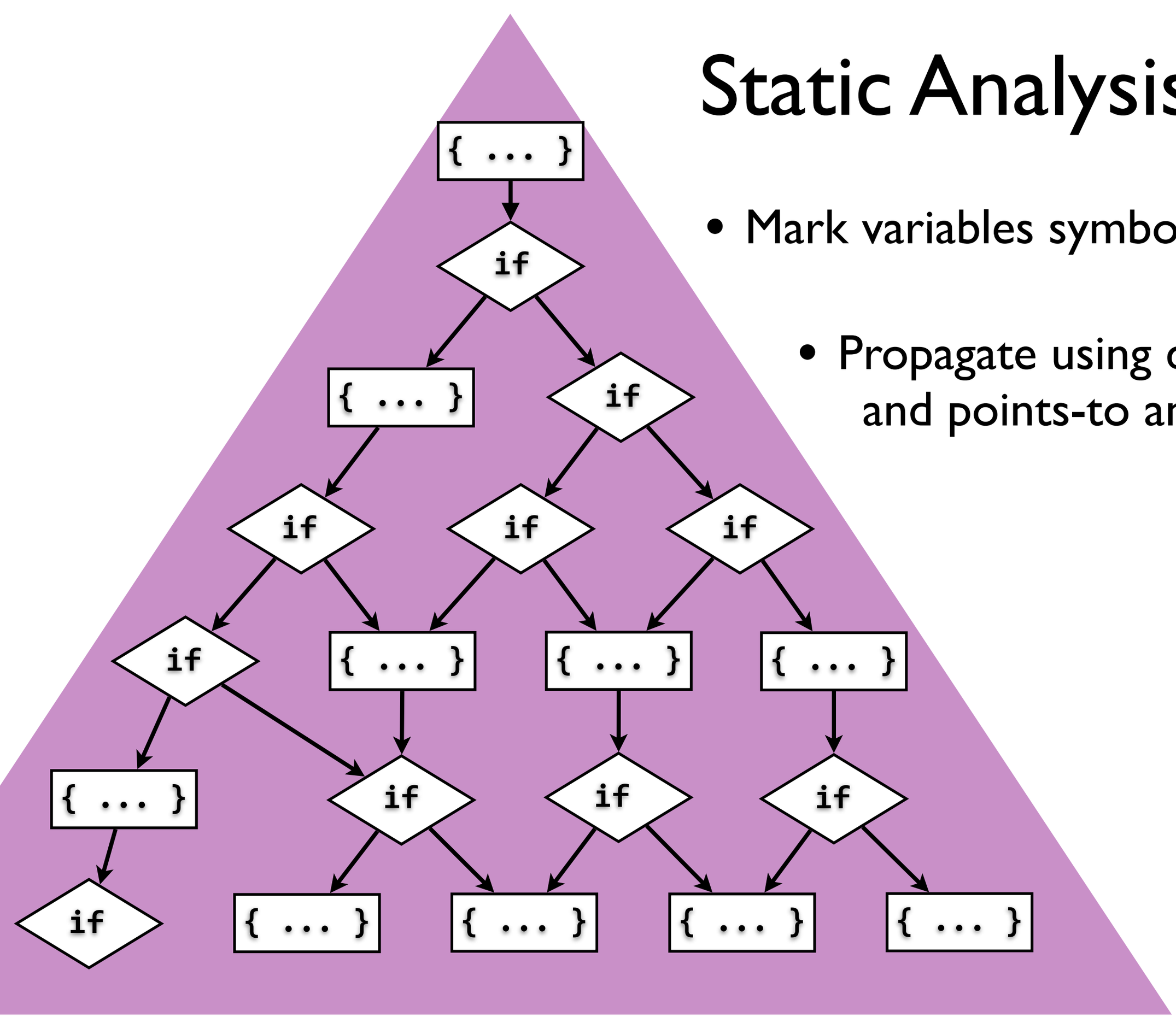


- Mark variables symbolic

Static Analysis

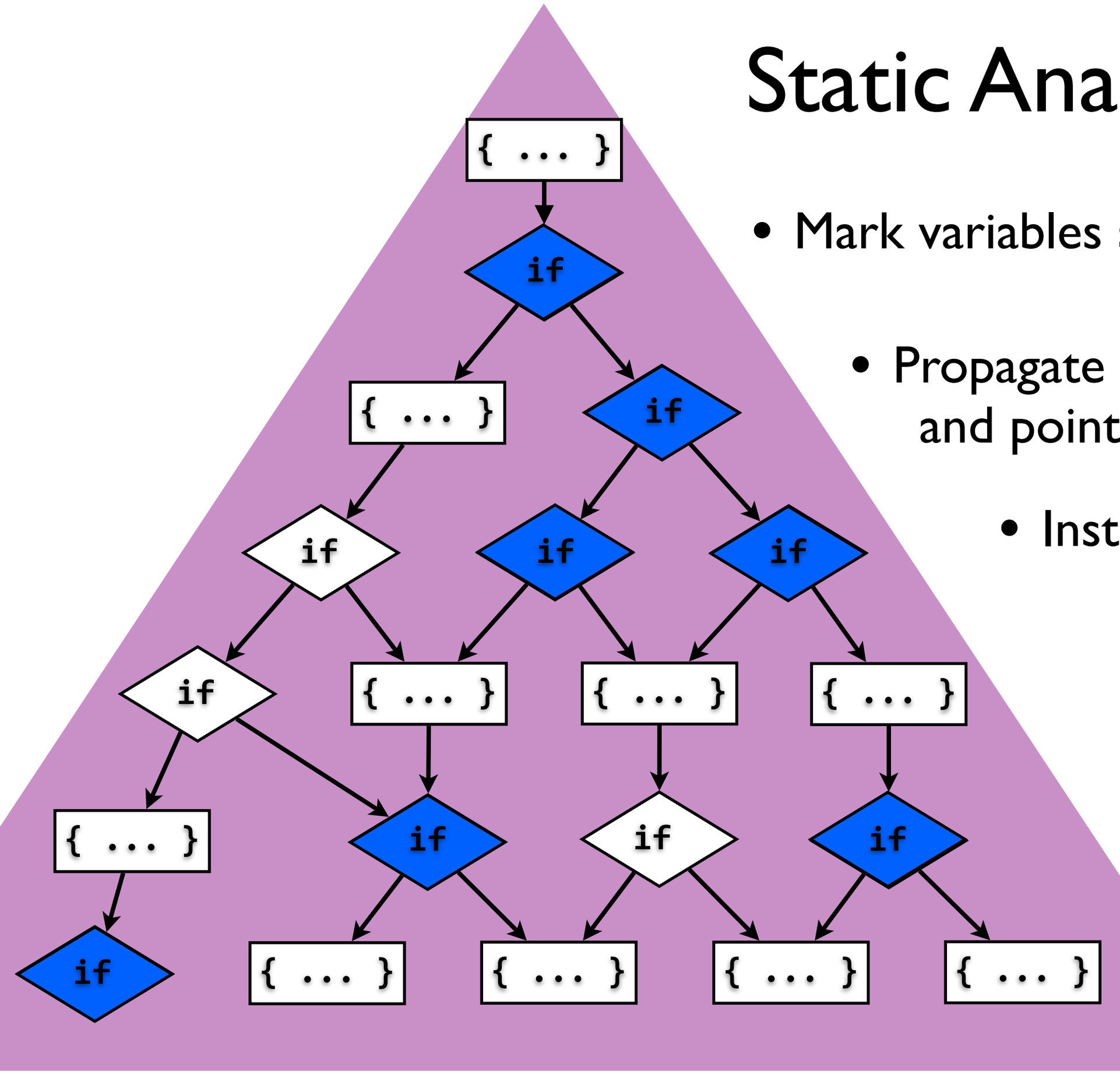


- Mark variables symbolic
  - Propagate using data-flow and points-to analysis



Static Analysis

- Mark variables symbolic
- Propagate using data-flow and points-to analysis



Static Analysis

- Mark variables symbolic
- Propagate using data-flow and points-to analysis
- Instrument symbolic branches

Static Analysis shortcomings

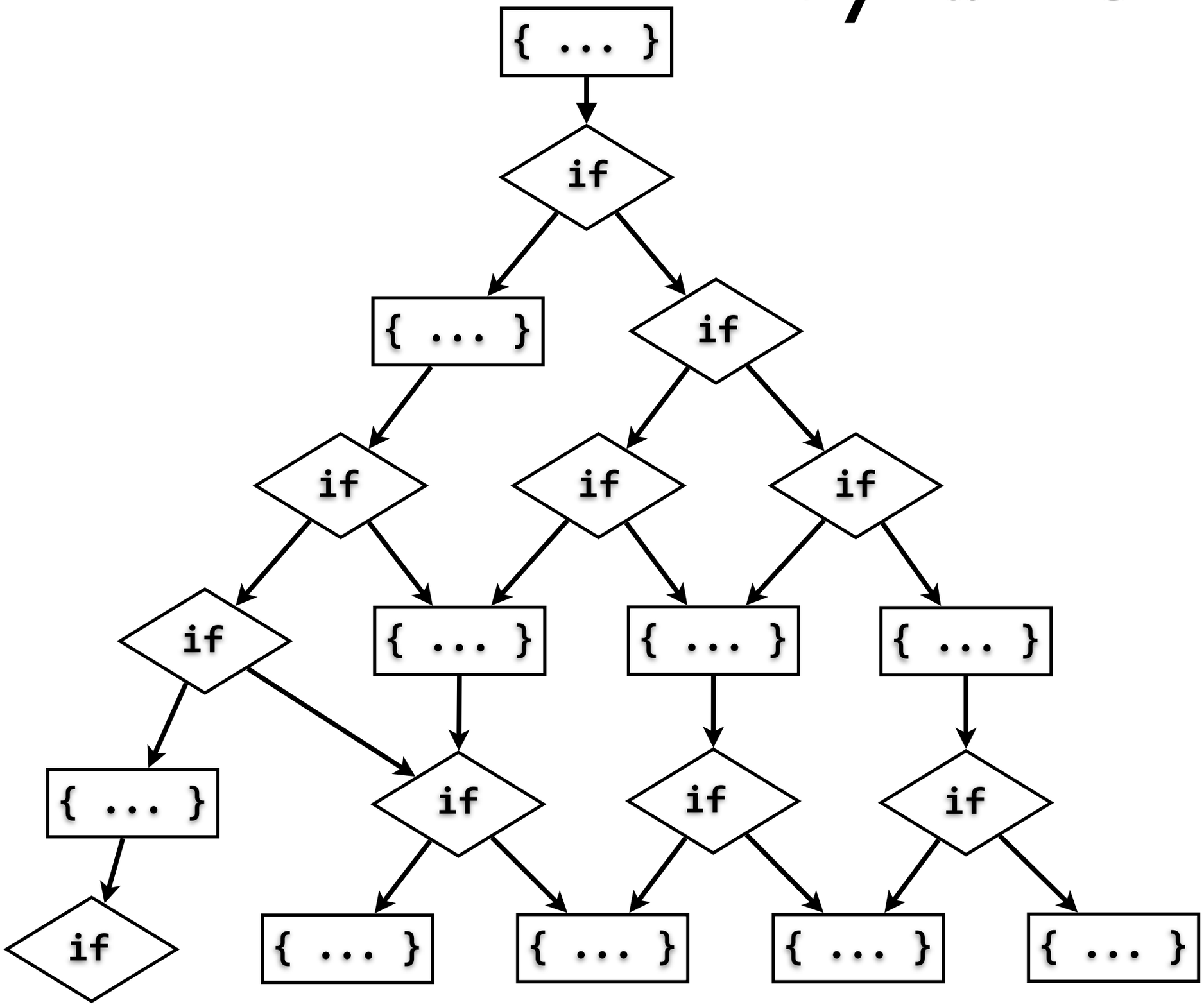
- points-to analysis imprecise, tends to over-estimate
- doesn't scale to libraries (libc):
  - ✦ conservatively, instrument all branches

Static Analysis shortcomings

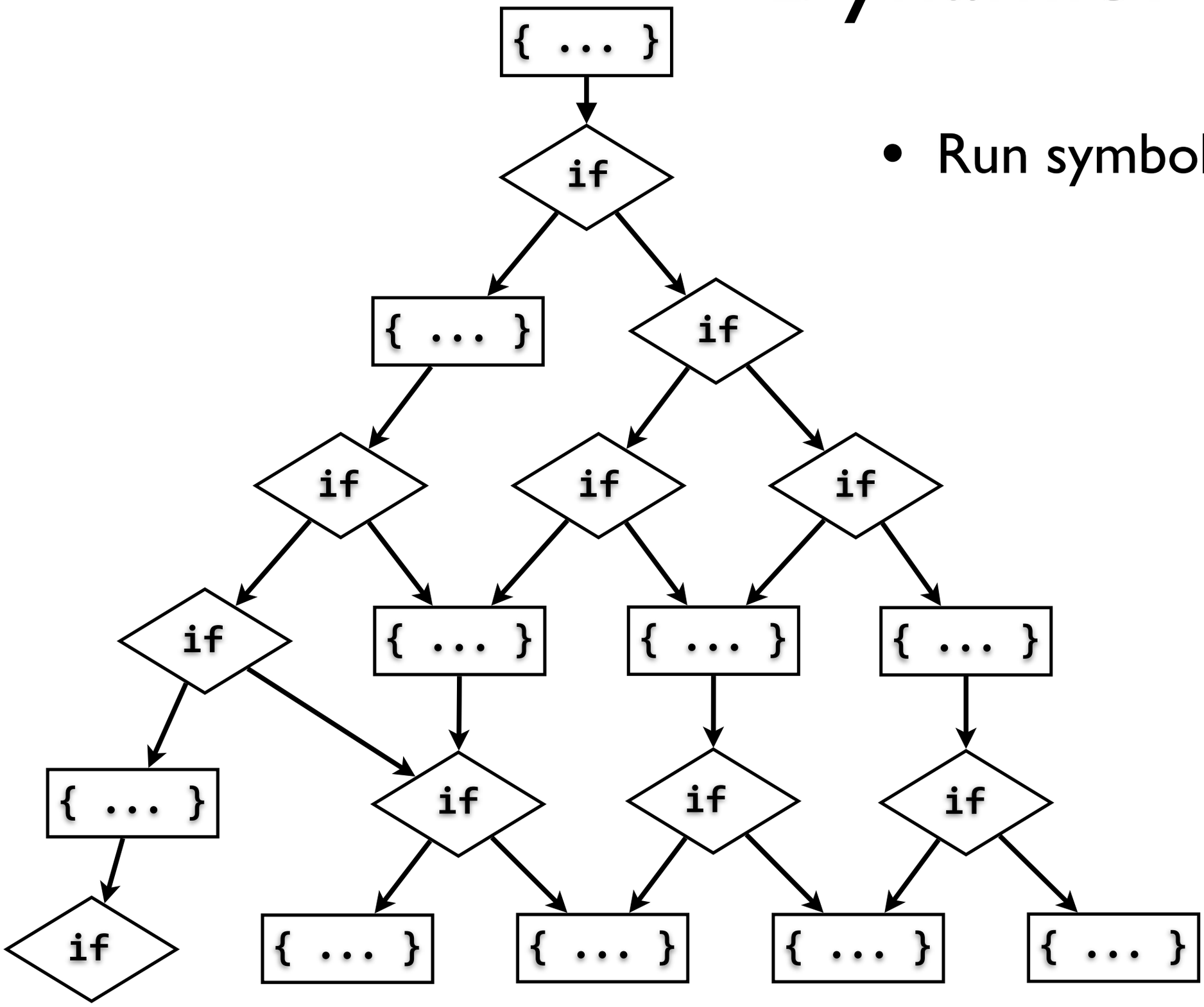
- points-to analysis imprecise, tends to over-estimate
- doesn't scale to libraries (libc):
  - ✦ conservatively, instrument all branches

Static Analysis tends to over-approximate:
⇒ too many branches instrumented

Dynamic Analysis (1)

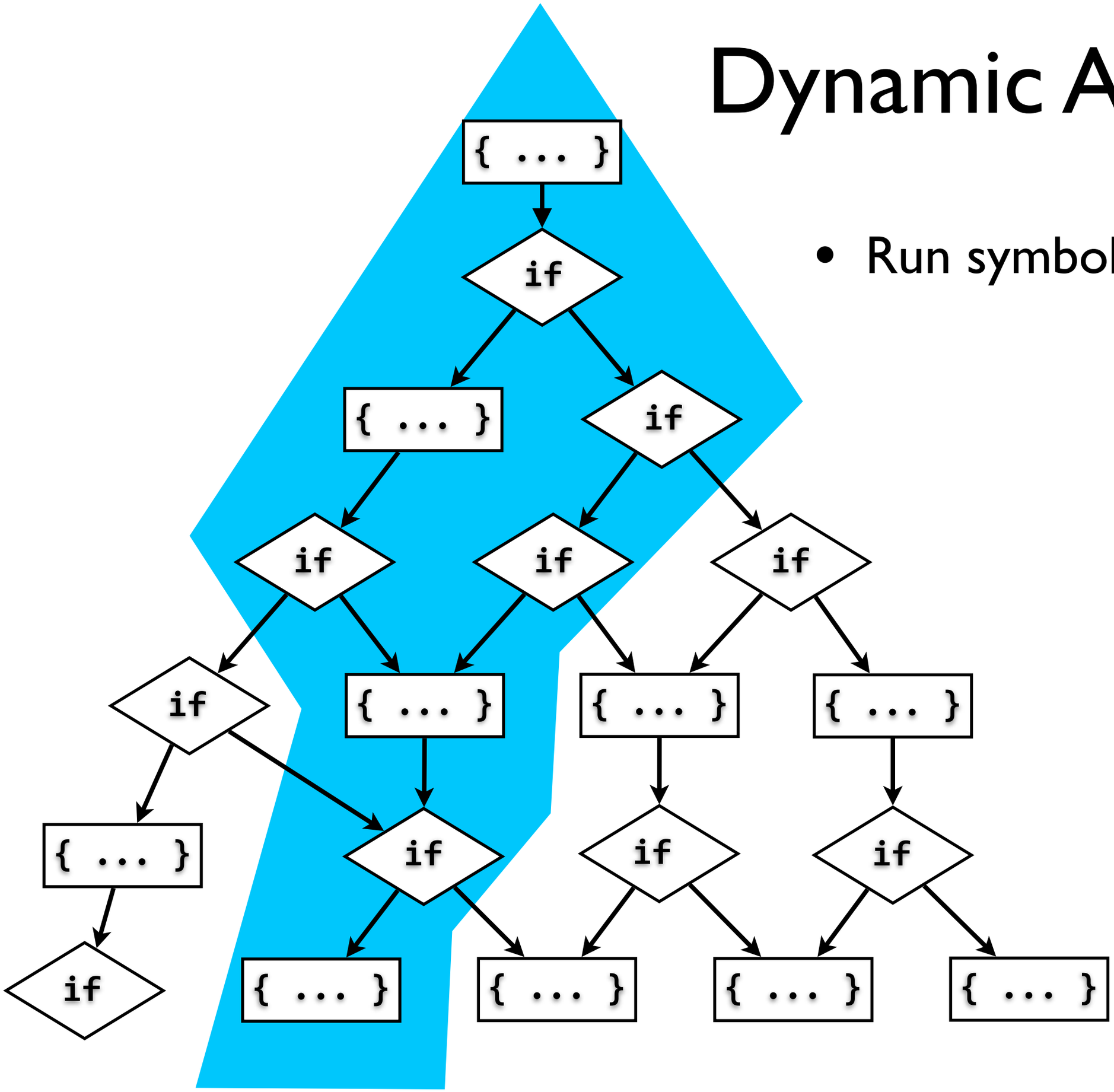


Dynamic Analysis (1)



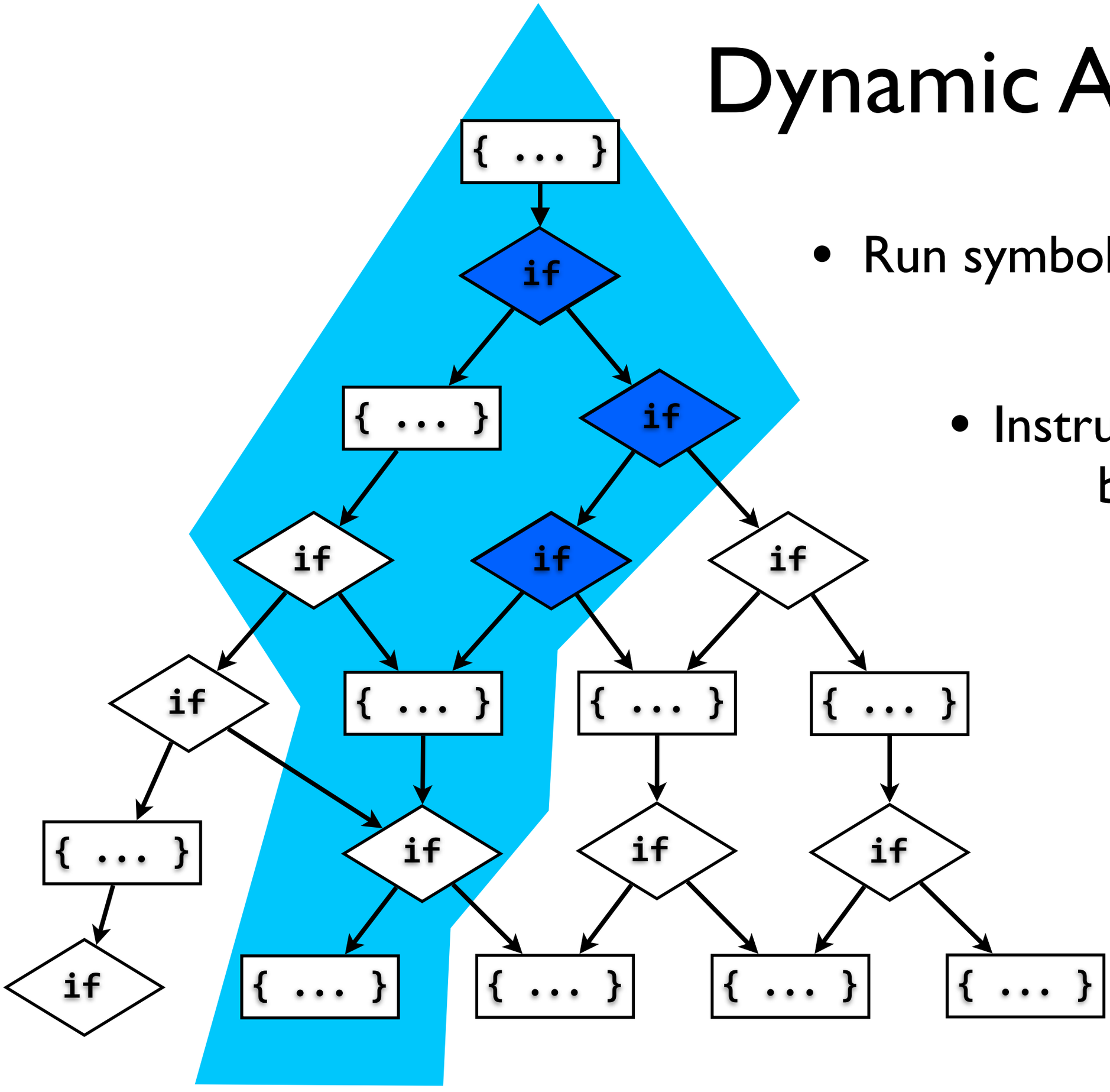
- Run symbolic execution

Dynamic Analysis (1)



- Run symbolic execution

Dynamic Analysis (1)



- Run symbolic execution
- Instrument symbolic branches

Dynamic Analysis (2)

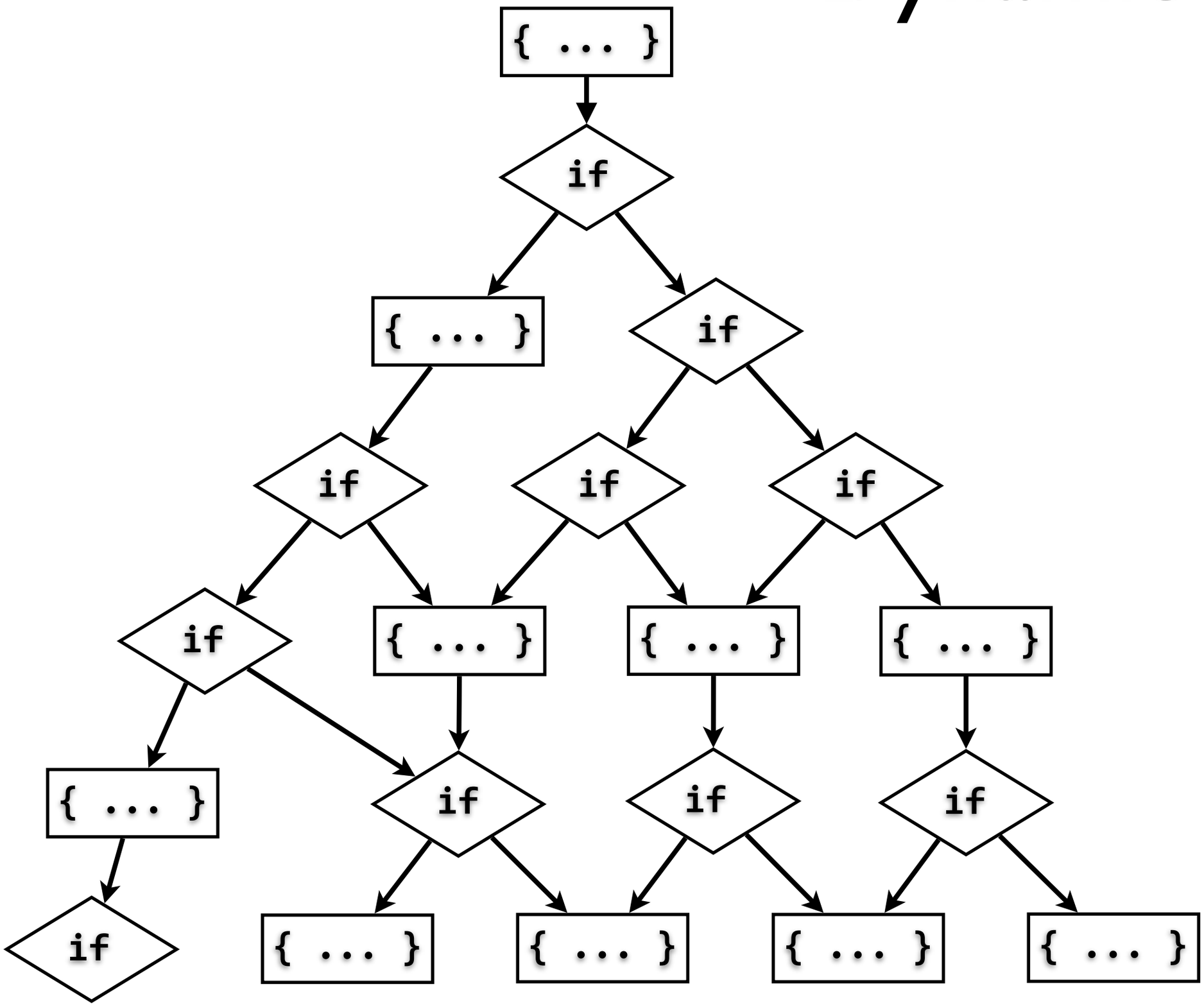
- Pros:
  - ✦ precise
  - ✦ runs on the library too
- Cons:
  - ✦ coverage limited

Dynamic Analysis (2)

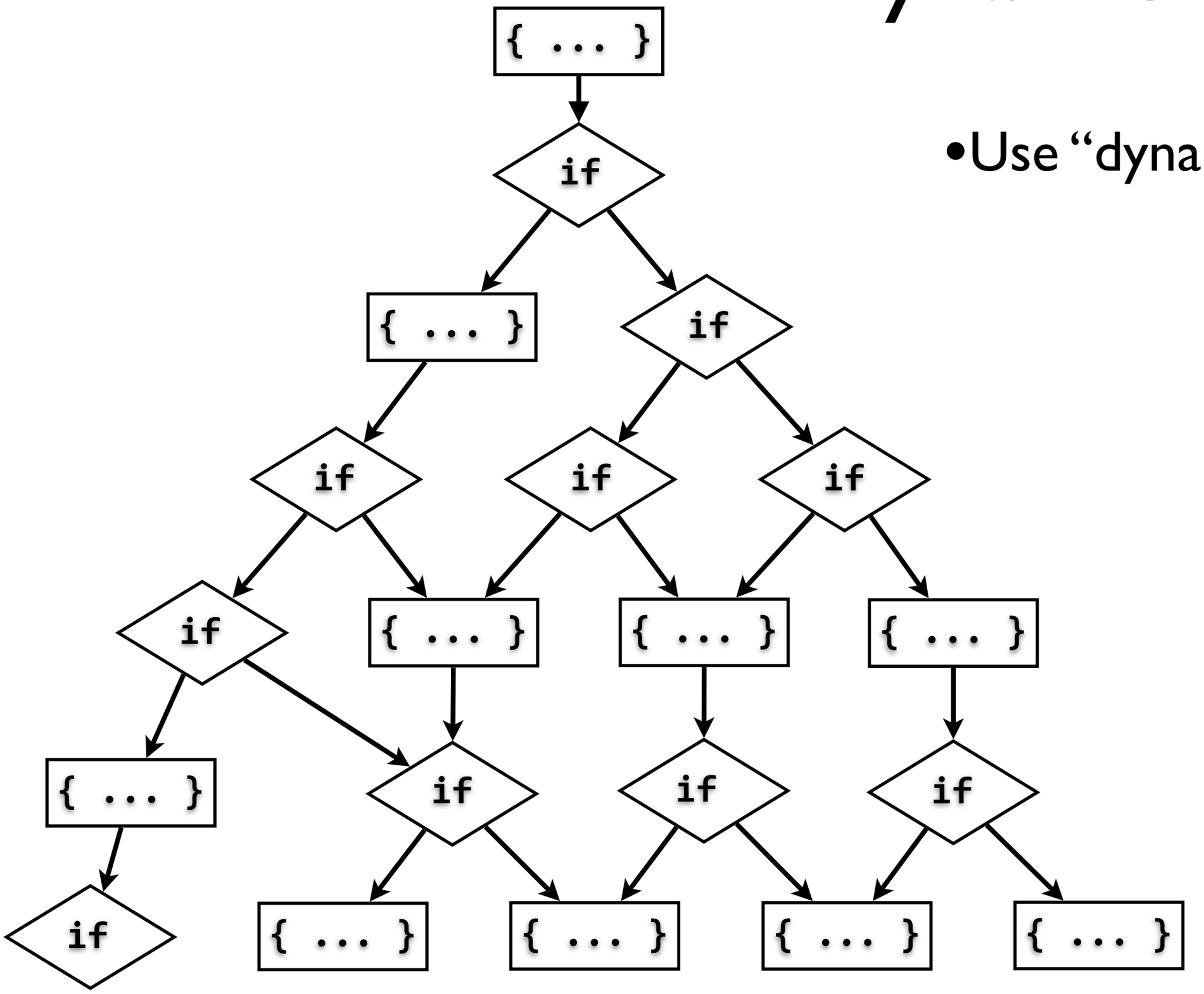
- Pros:
  - ✦ precise
  - ✦ runs on the library too
- Cons:
  - ✦ coverage limited

Dynamic Analysis tends to under-estimate:
⇒ some symbolic branches may not be logged

Dynamic + Static

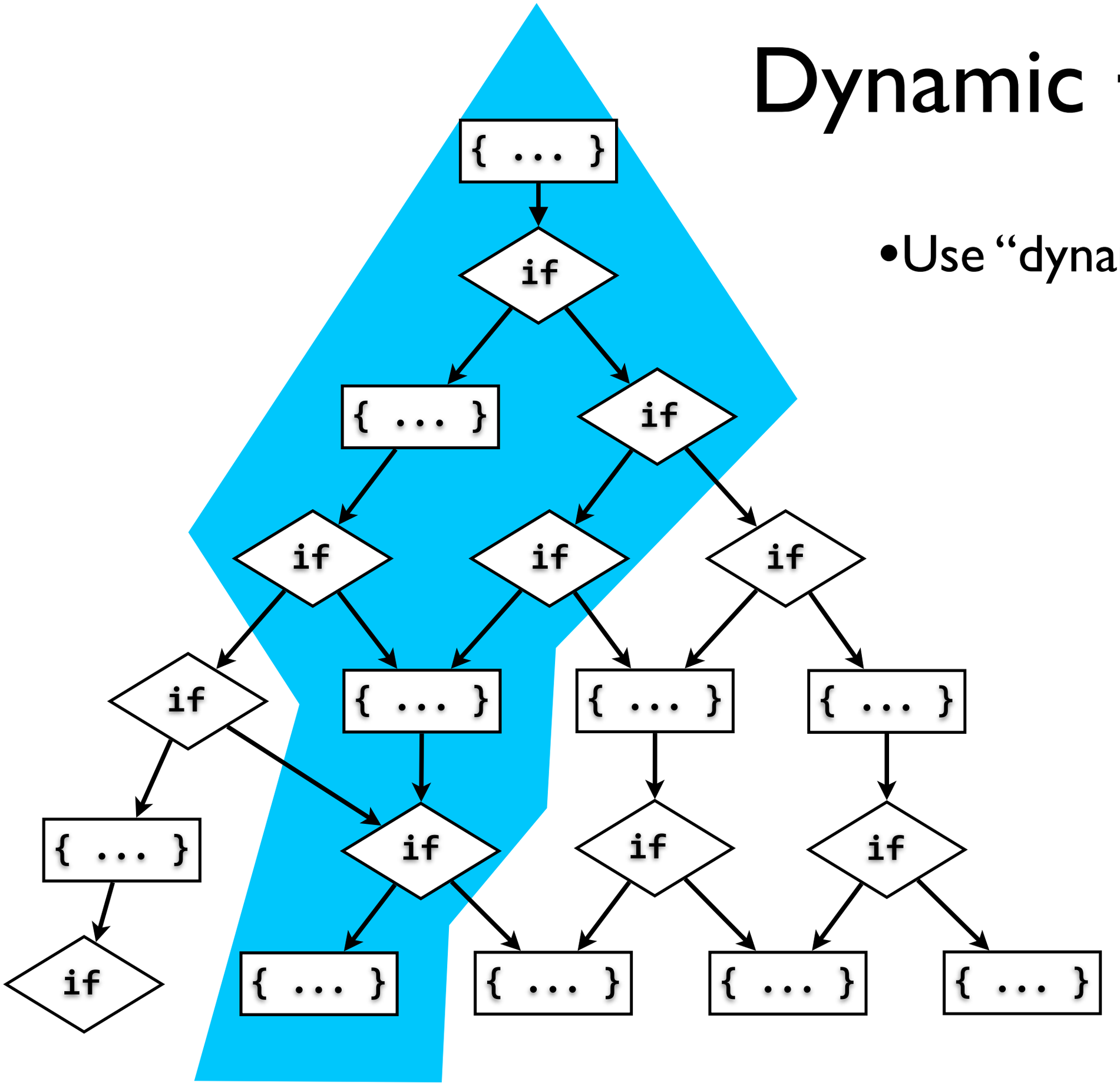


Dynamic + Static



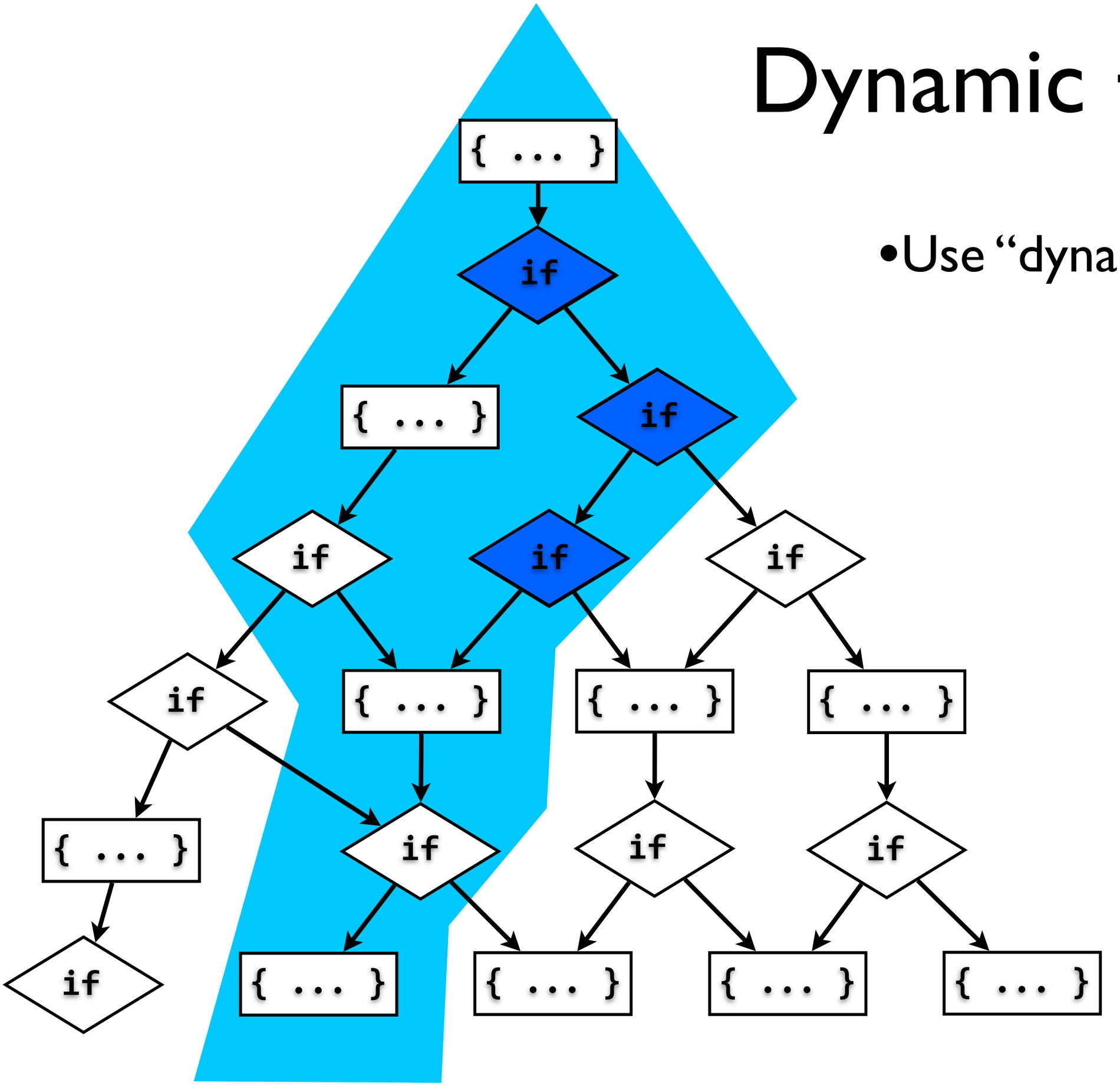
- Use "dynamic" first

Dynamic + Static



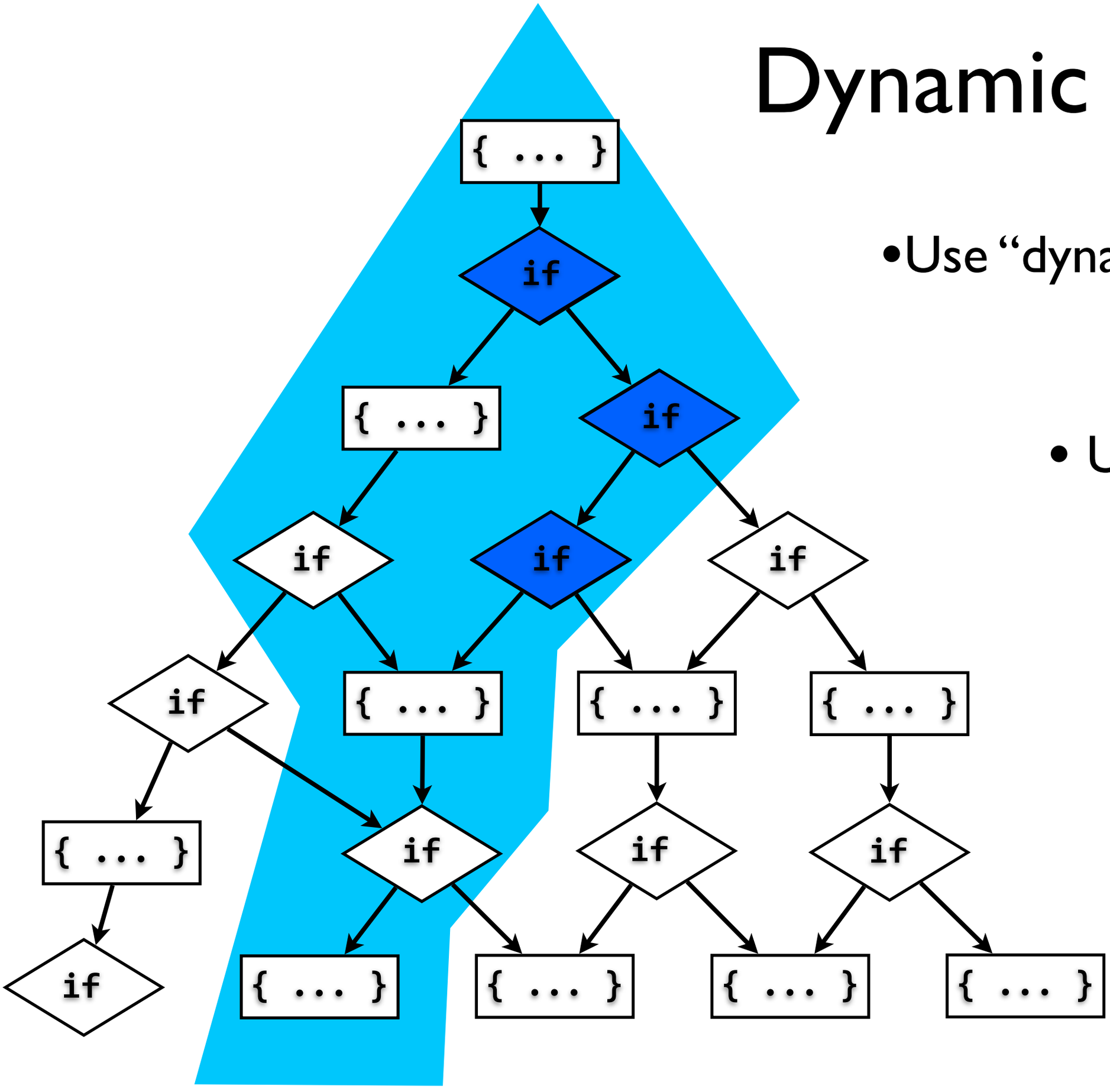
- Use “dynamic” first

Dynamic + Static



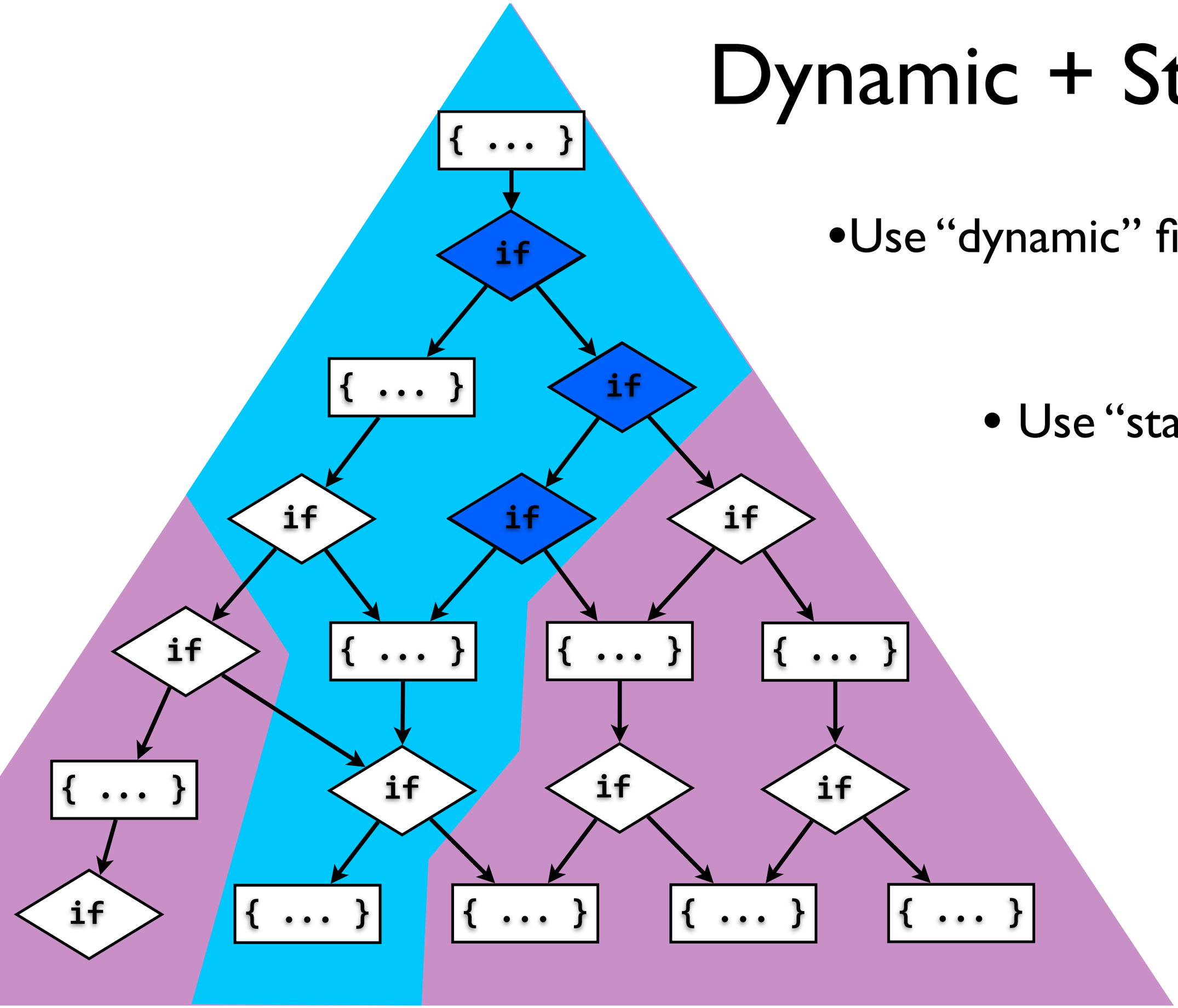
- Use “dynamic” first

Dynamic + Static



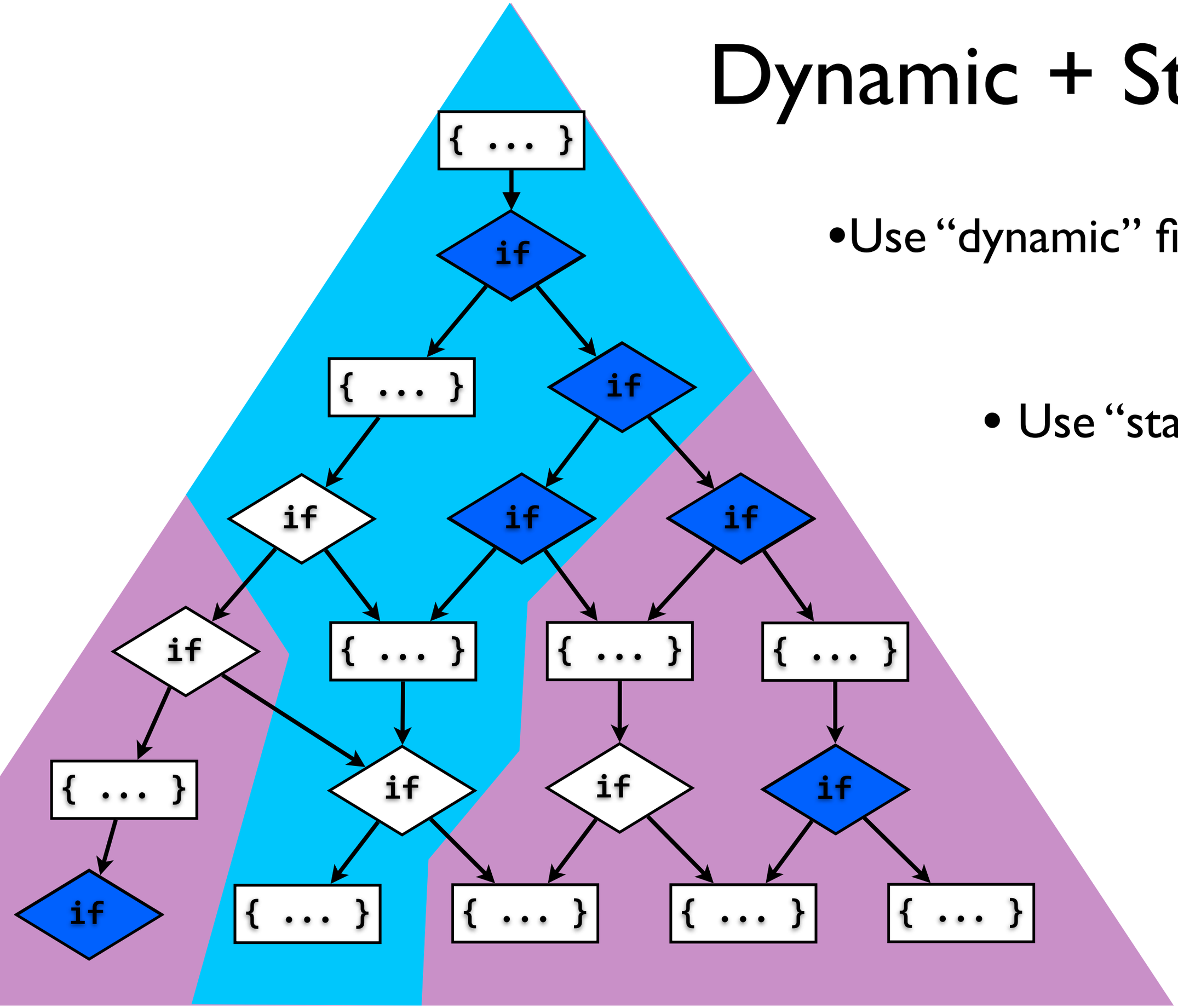
- Use "dynamic" first
- Use "static"

Dynamic + Static



- Use “dynamic” first
- Use “static”

Dynamic + Static



- Use "dynamic" first
- Use "static"

Instrumentation summary

Instrumentation summary

- Static analysis:
  - ✦ covers all code but is imprecise

Instrumentation summary

- Static analysis:
  - ✦ covers all code but is imprecise
- Dynamic analysis
  - ✦ precise but with limited coverage

Instrumentation summary

- Static analysis:
  - ✦ covers all code but is imprecise
- Dynamic analysis
  - ✦ precise but with limited coverage
- Dynamic + static
  - ✦ good coverage and precise

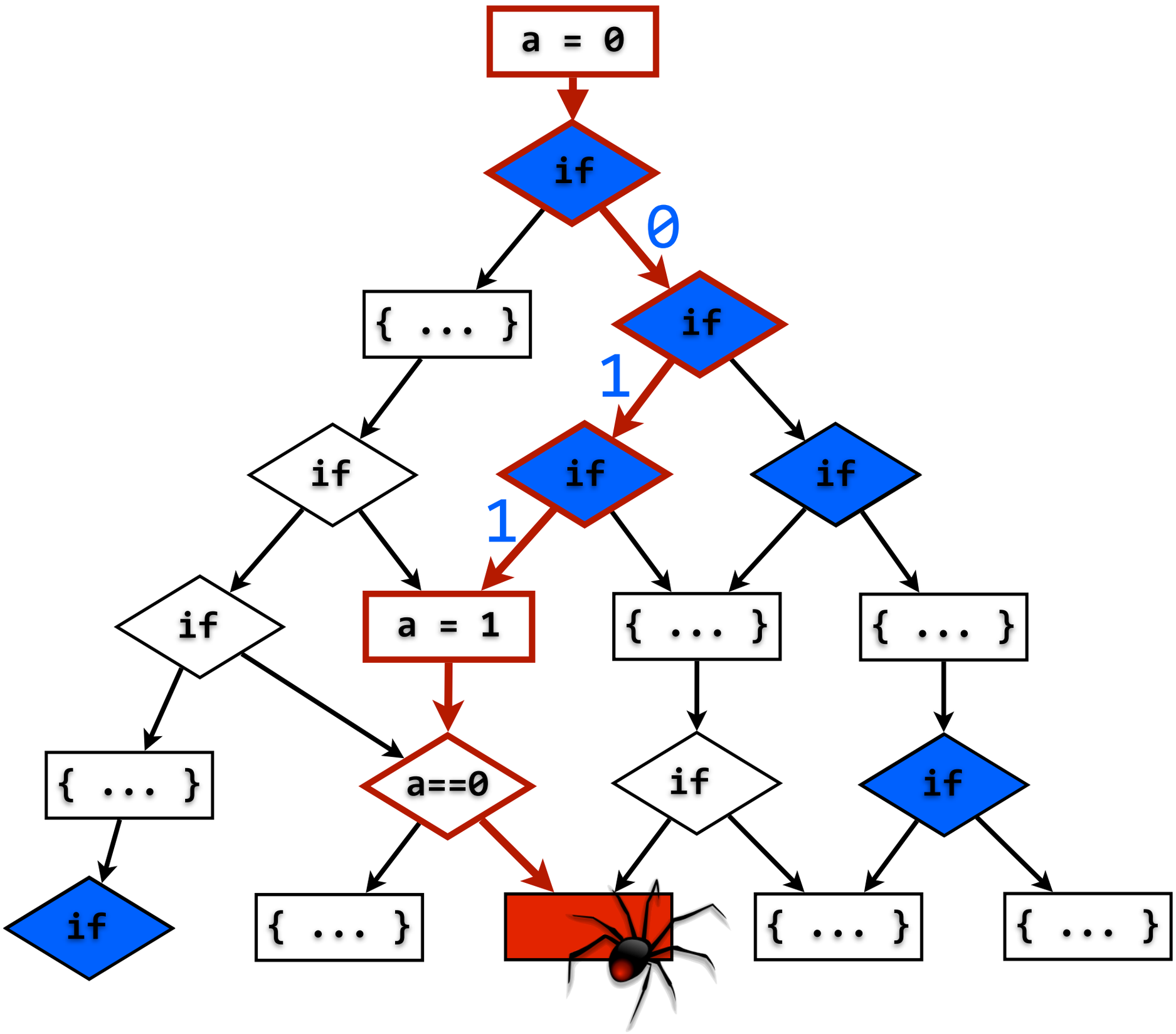
Replay

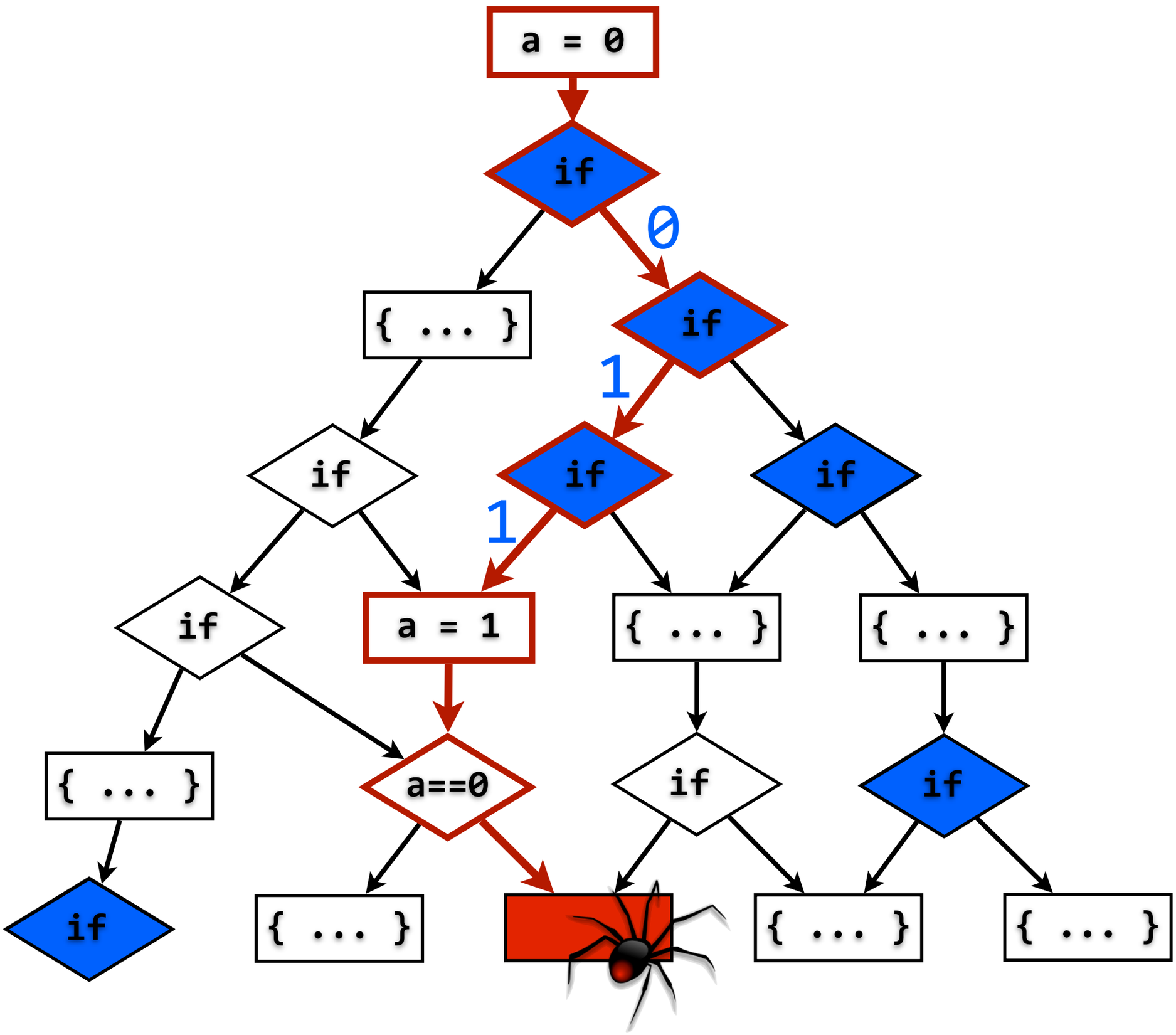
Replay

- Bug ⇒ branch log is shipped to the vendor

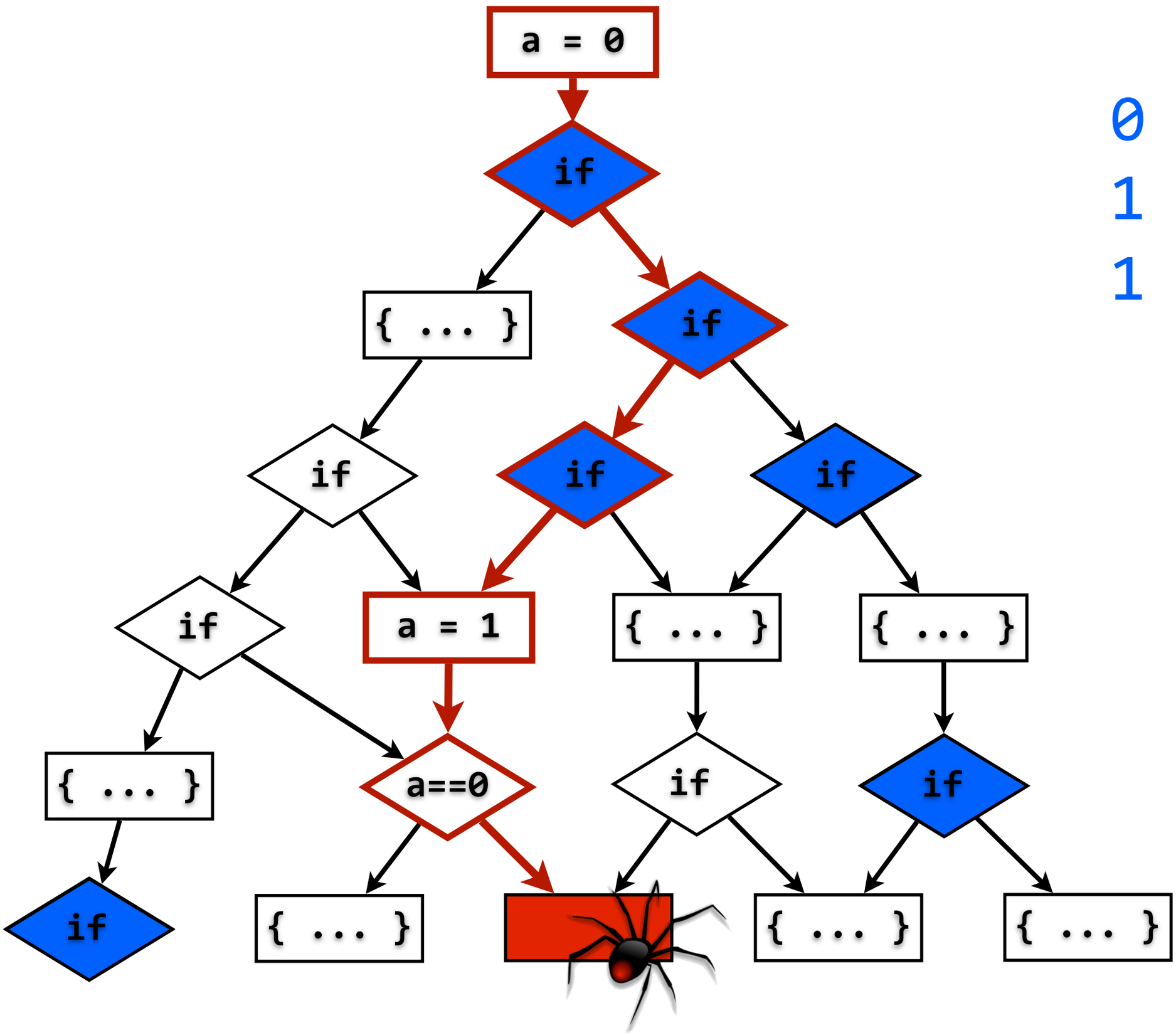
Replay

- Bug ⇒ branch log is shipped to the vendor
- Vendor replays the application:
  - ✦ uses symbolic execution
  - ✦ follows the branch log

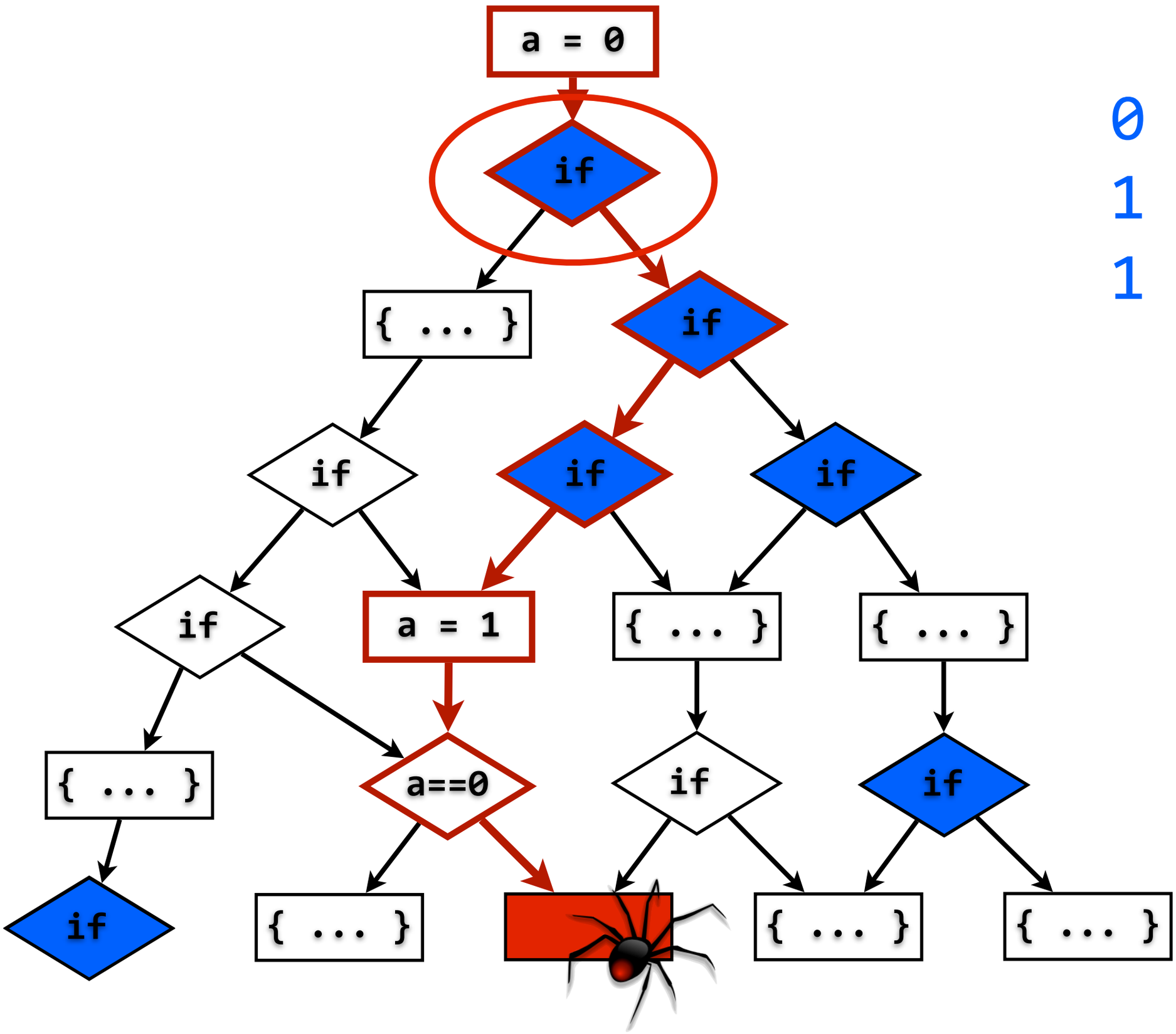

a = 0
if
0
{ ... }
if
1
if
if
if
1
if
a = 1
{ ... }
{ ... }
{ ... }
a==0
if
if
if
{ ... }
{ ... }
{ ... }
{ ... }


a = 0
if
0
{ ... }
if
1
if
if
if
1
a = 1
{ ... }
{ ... }
if
a==0
if
if
{ ... }
{ ... }
{ ... }
{ ... }

```
0
1
1
```



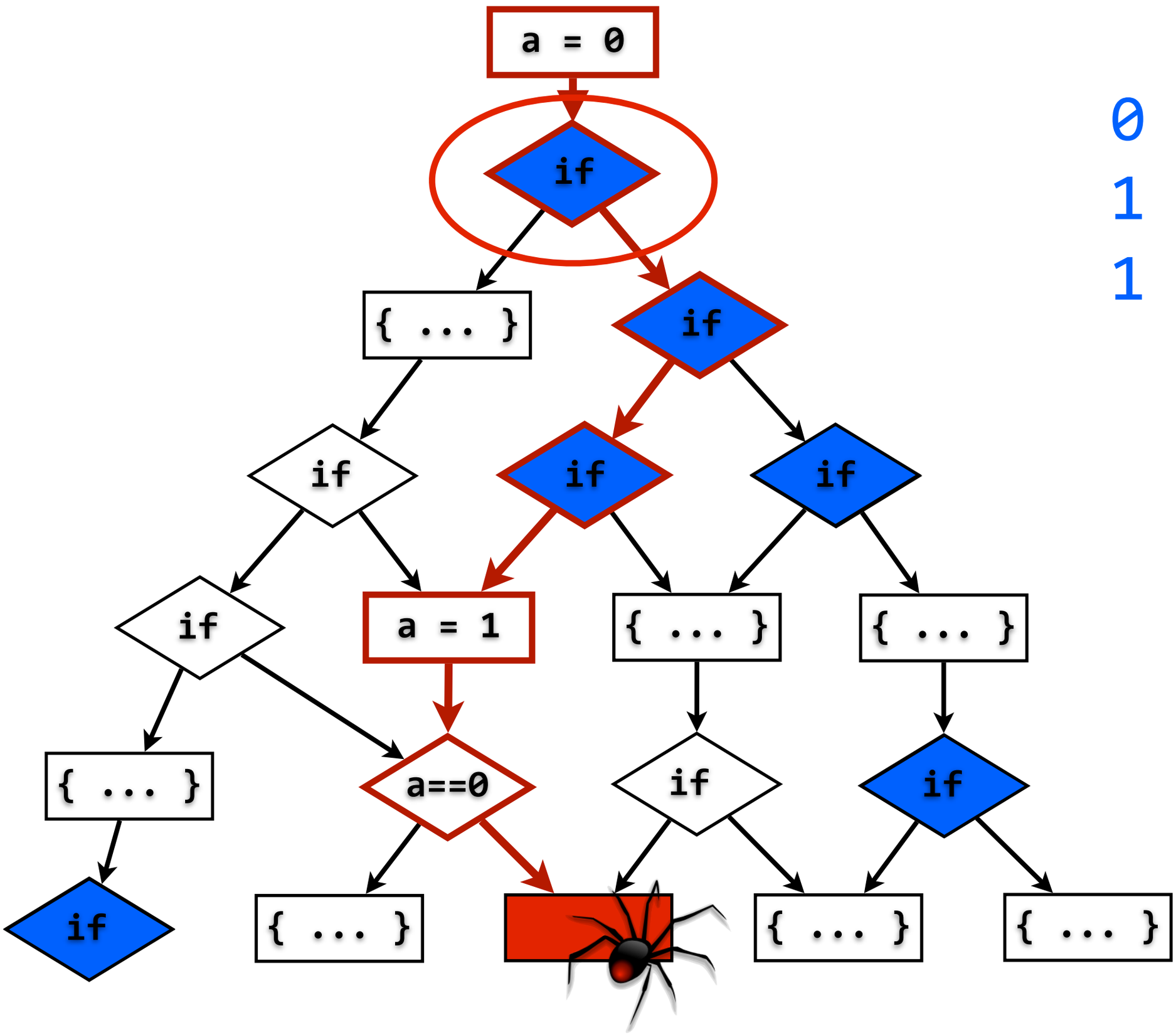
```
0
1
1
```

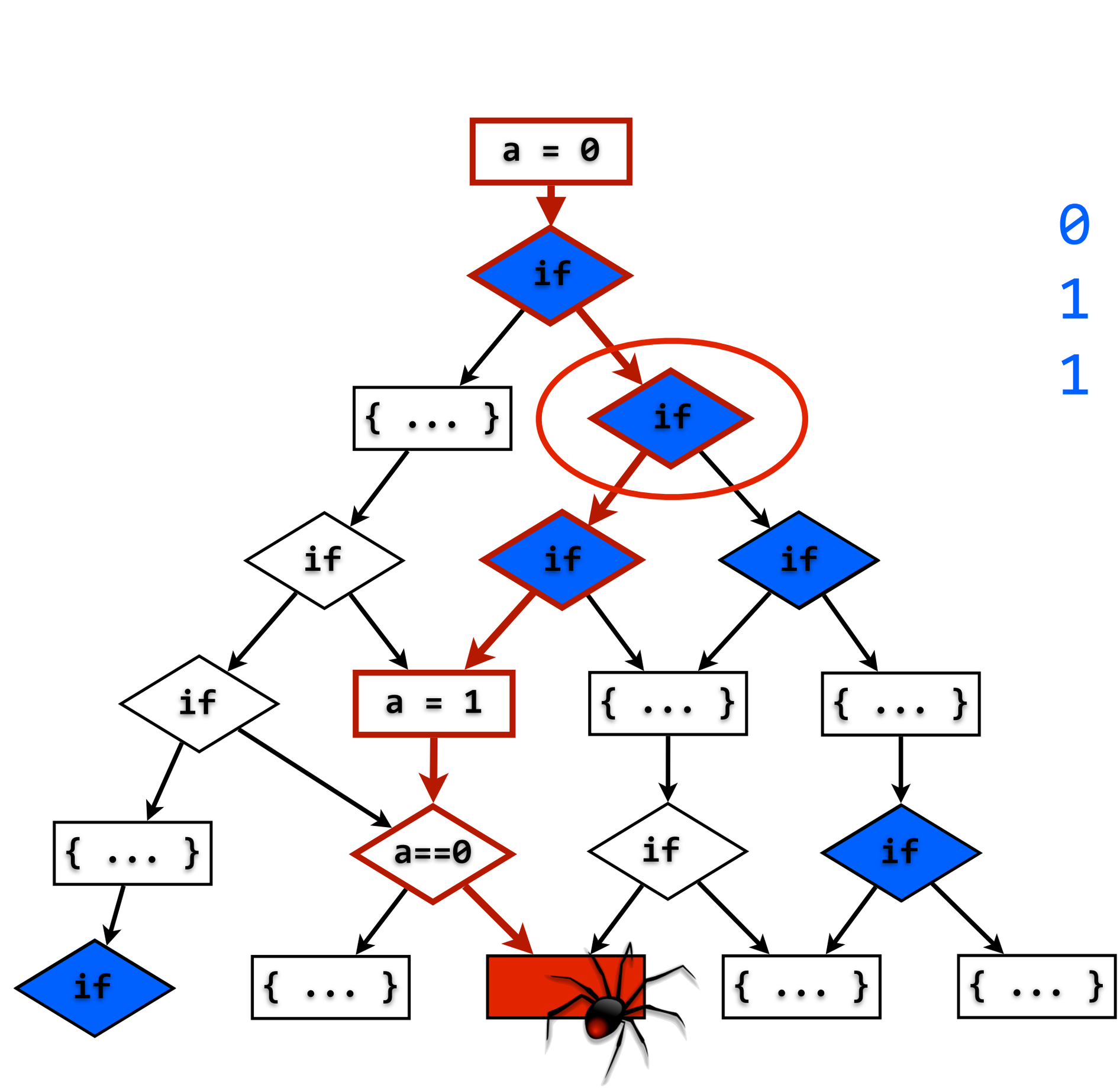


0

1

1





0

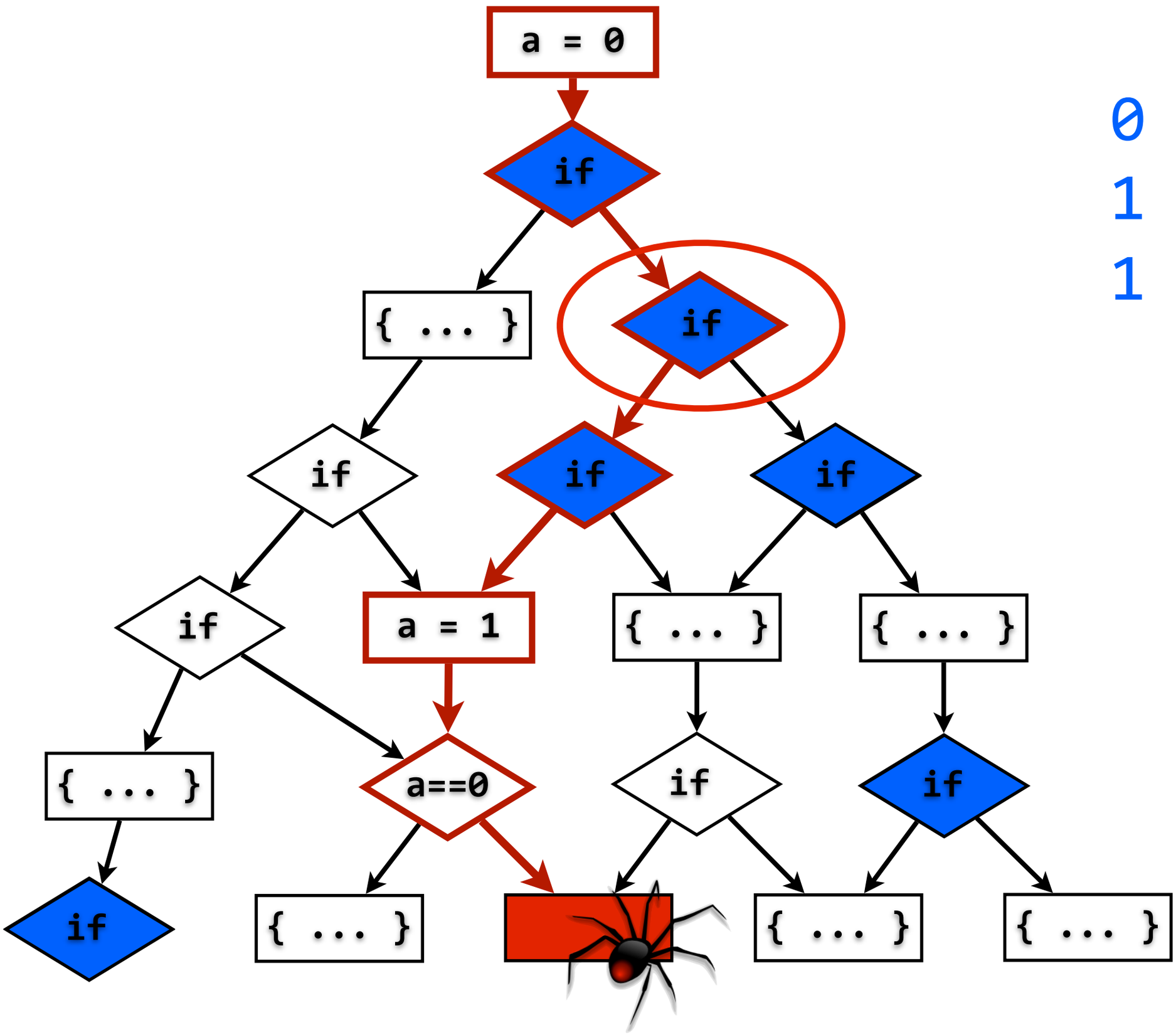
1

1

0

1

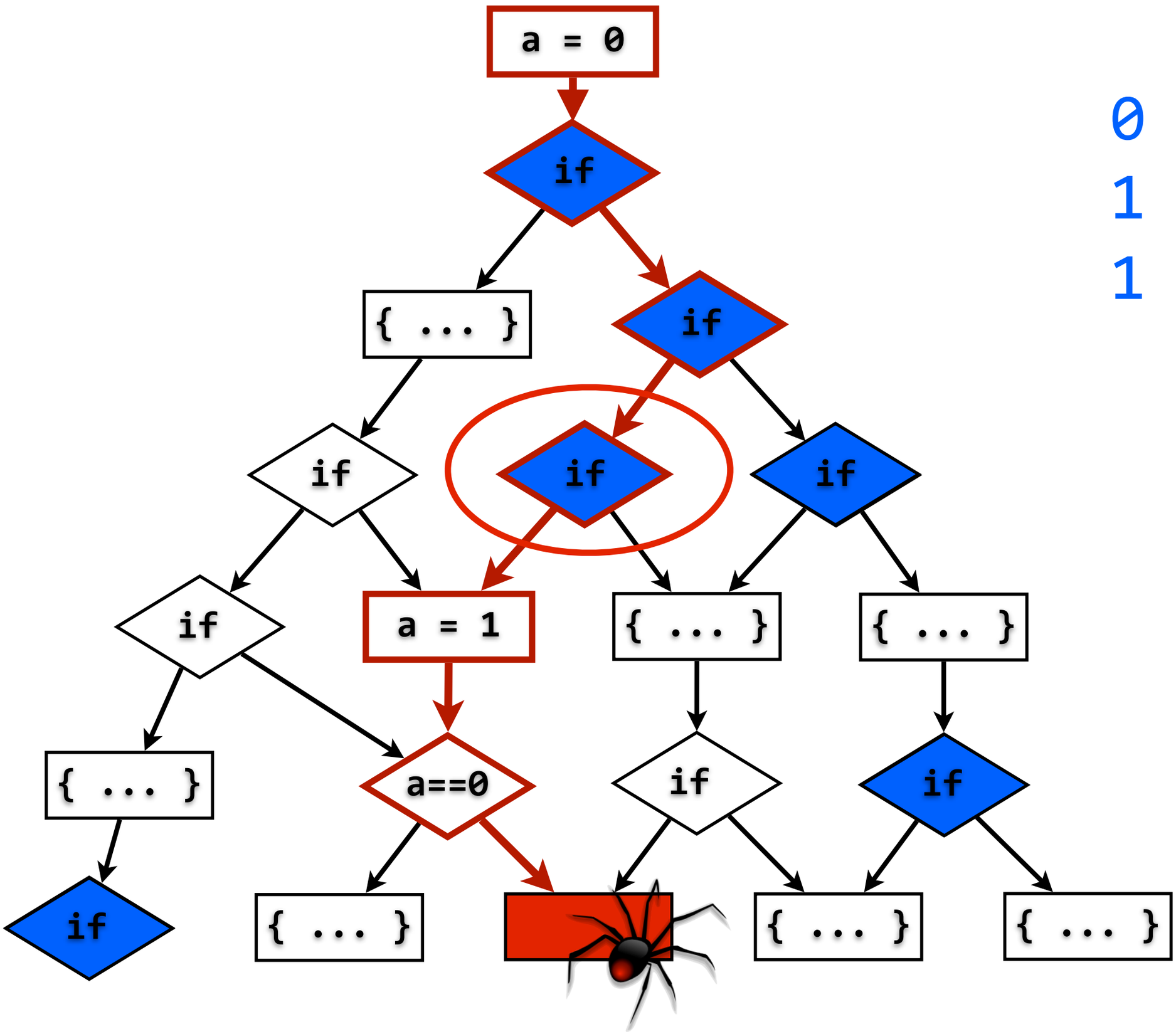
1



0

1

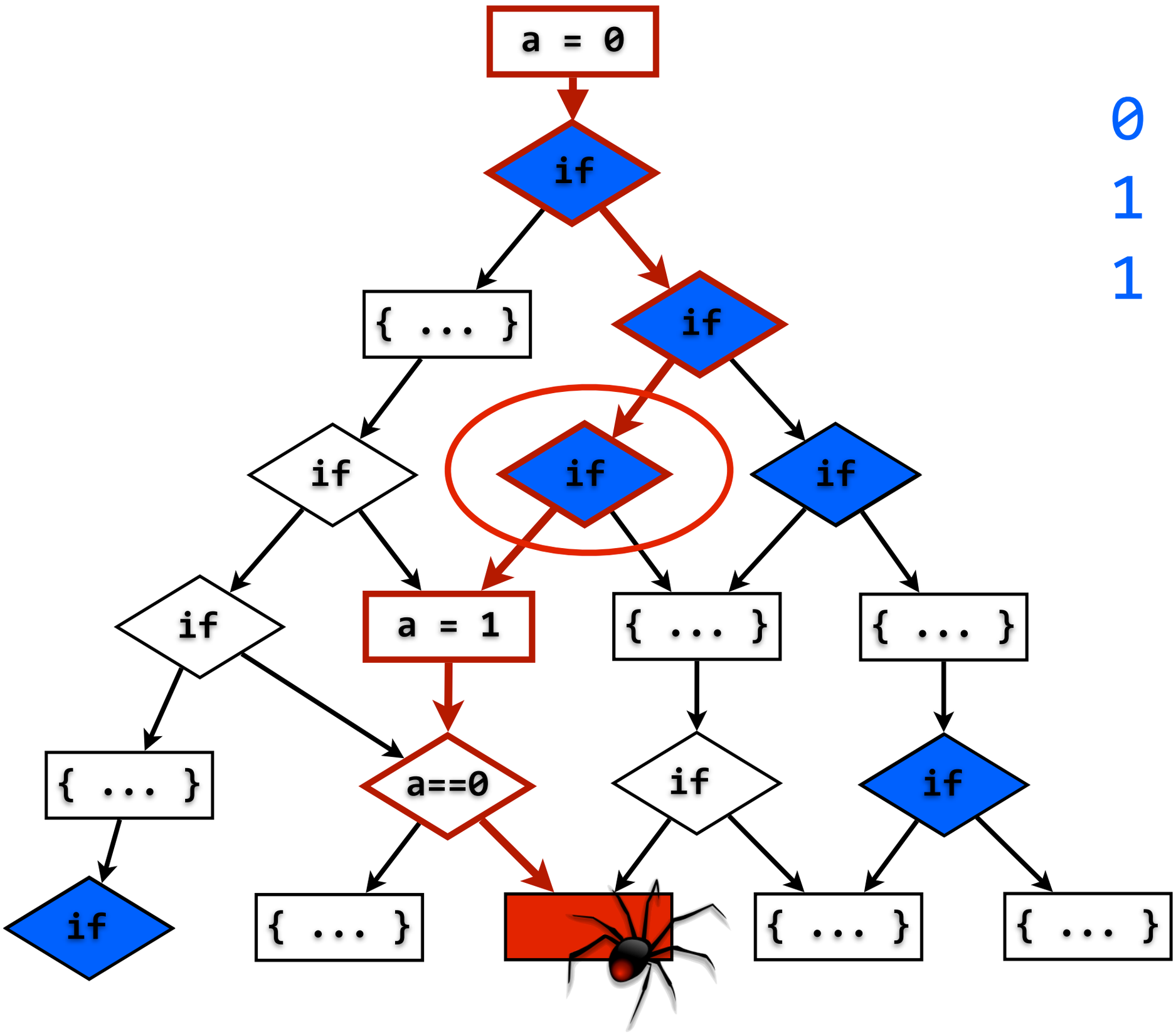
1



0 ↘

1 ↙

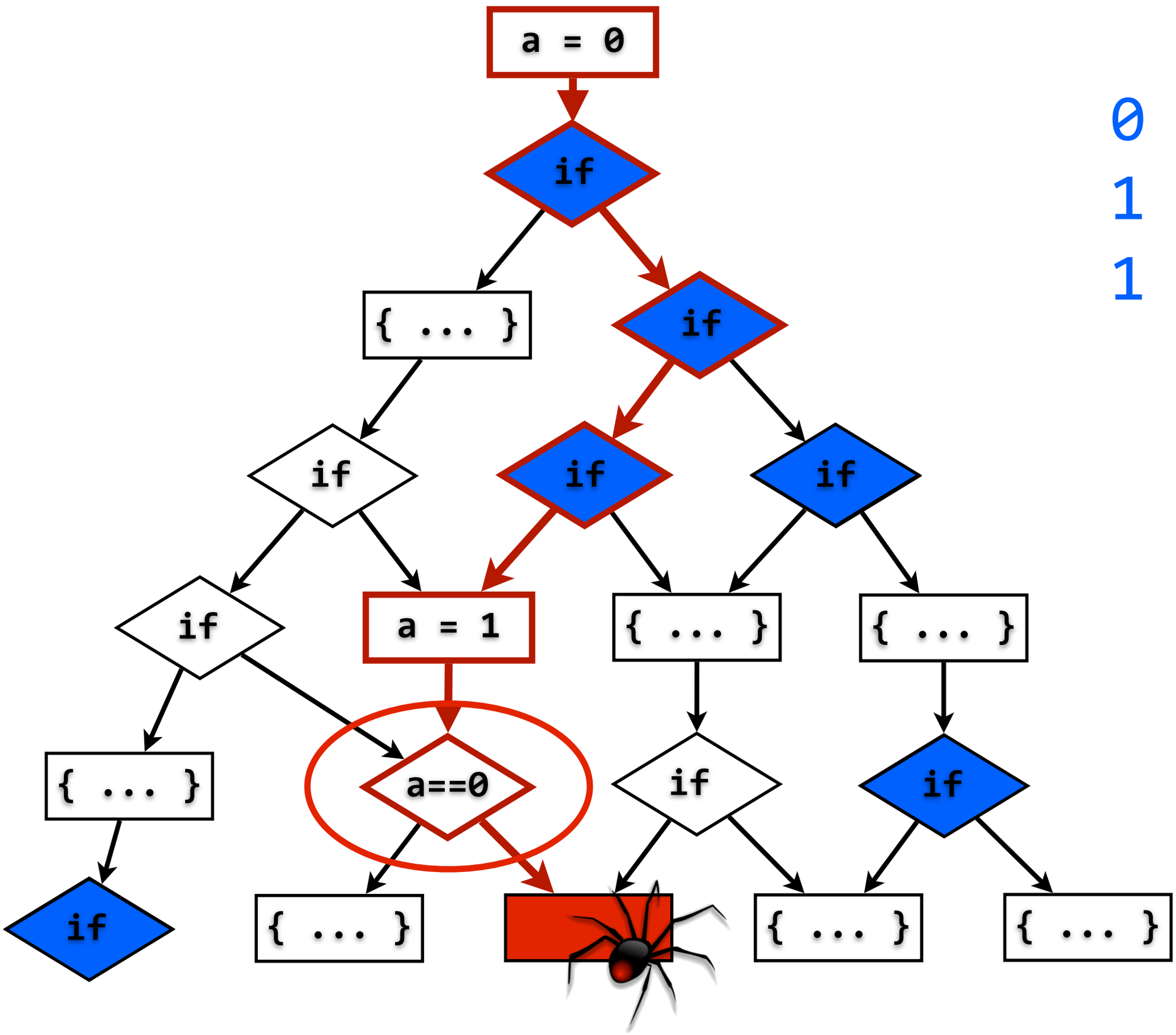
1 ↙

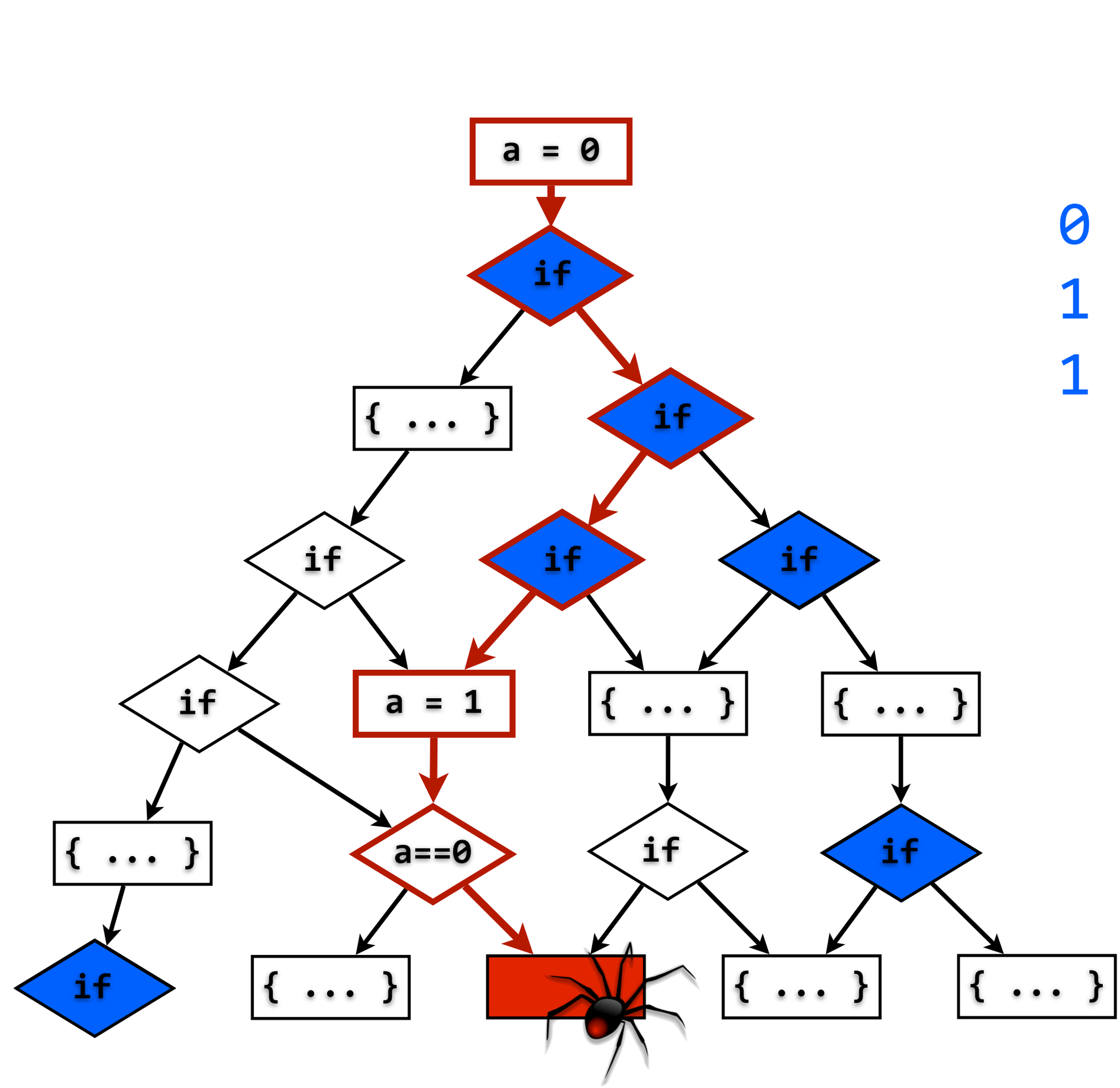


0 ↘

1 ↙

1 ↙





0 ↘

1 ↘

1 ↘

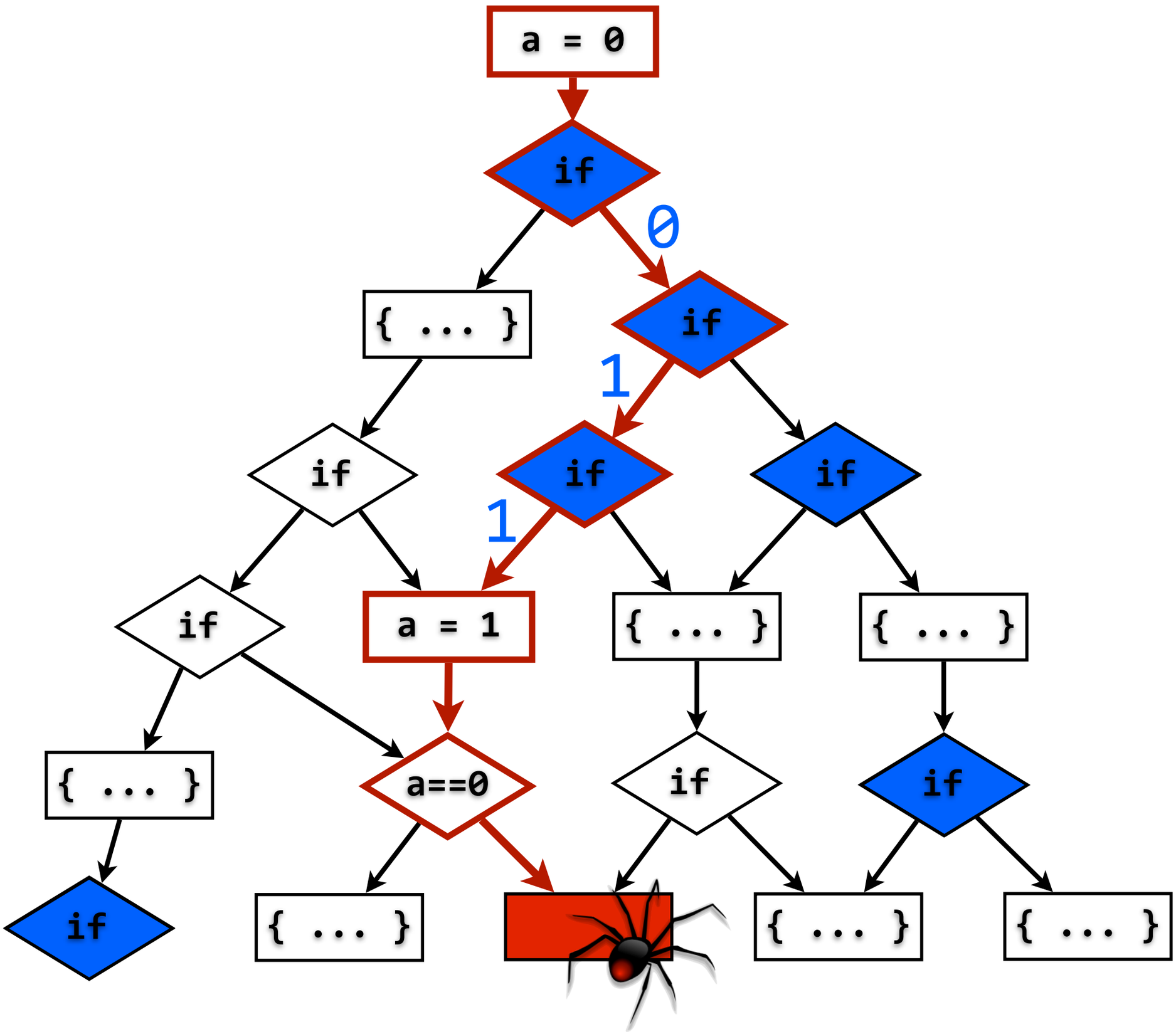
Dealing with approximations

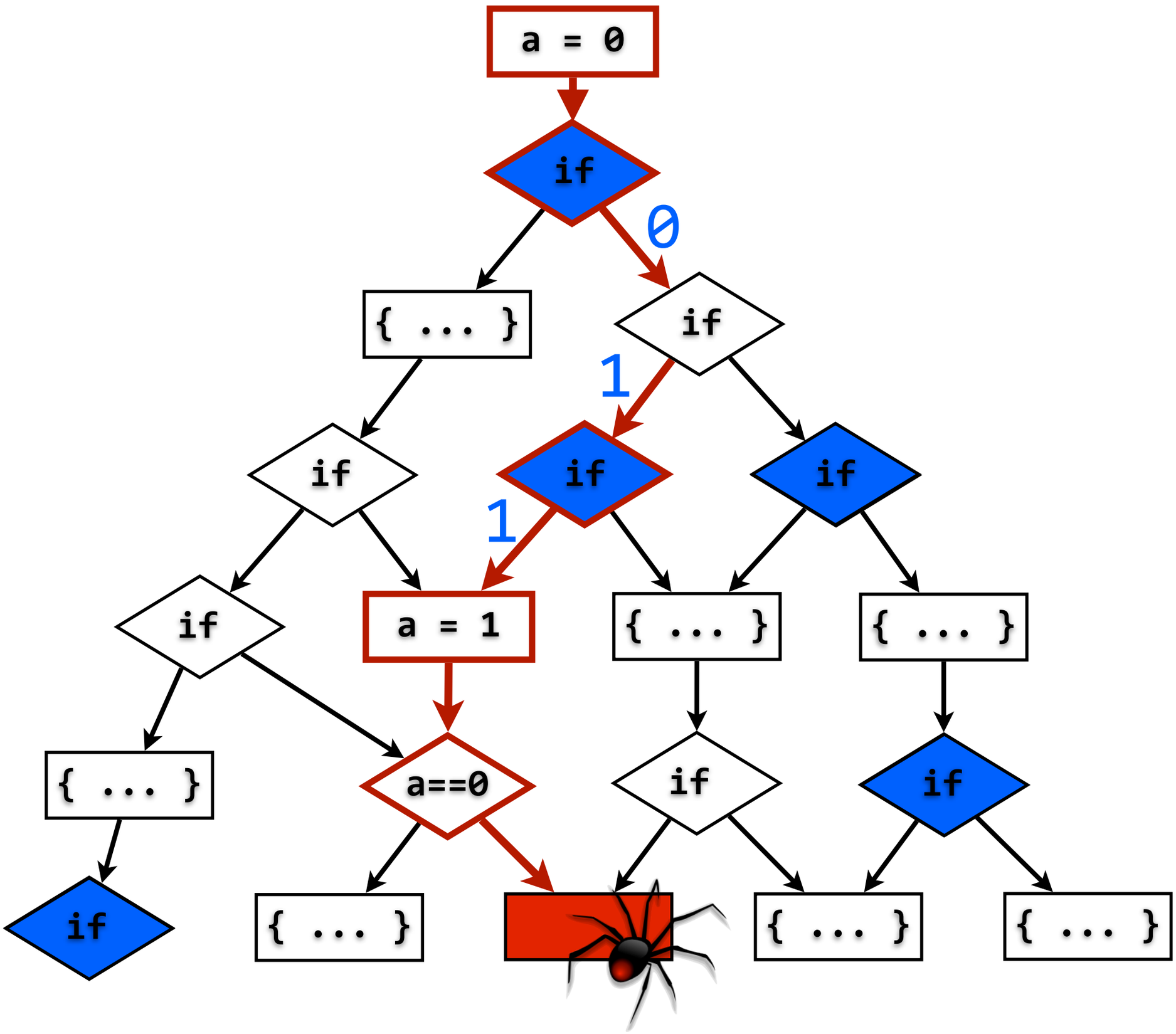
Dealing with approximations

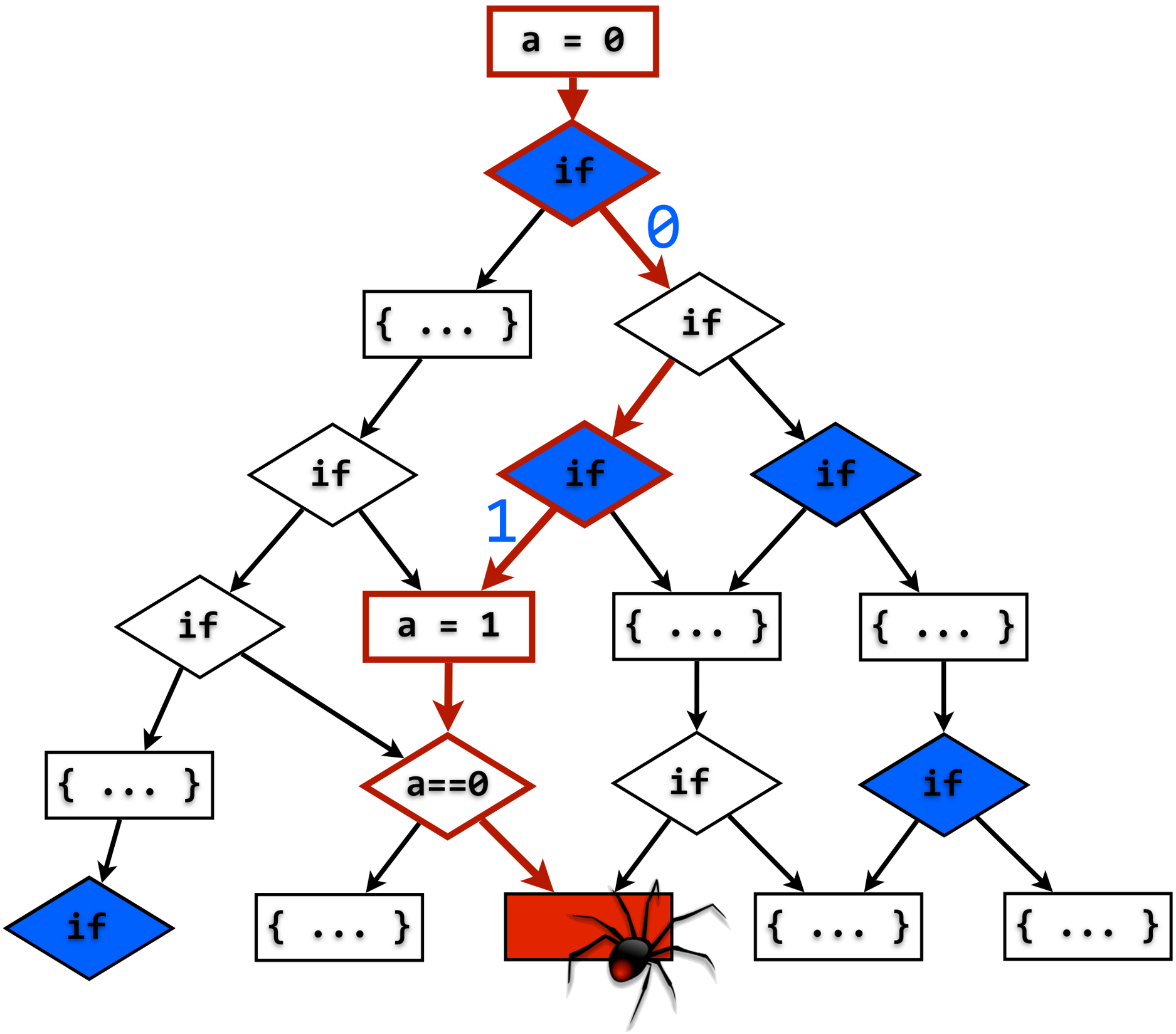
- Too many branches instrumented:
  - ✦ increased overhead

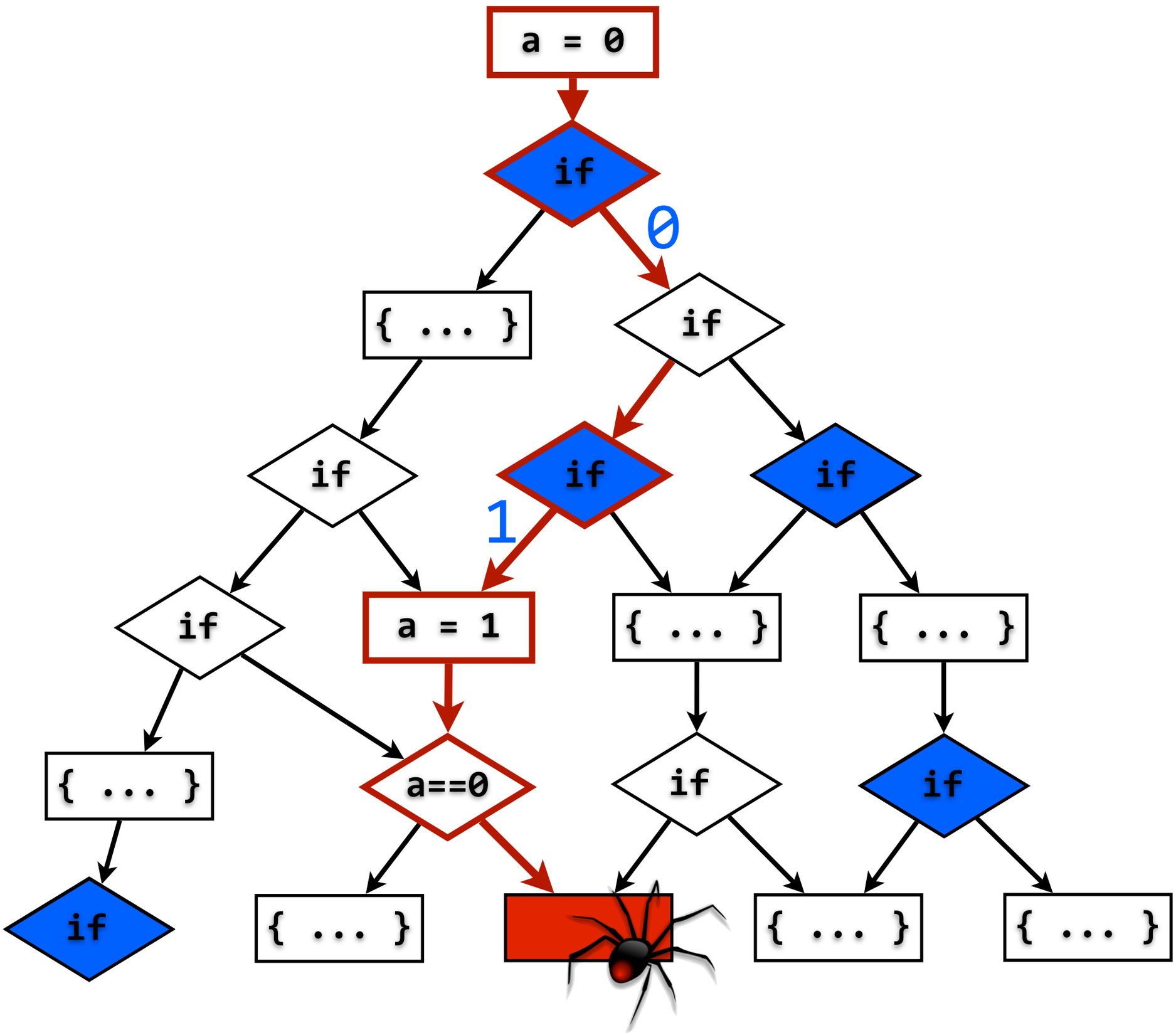
Dealing with approximations

- Too many branches instrumented:
  - ✦ increased overhead
- Too few branches instrumented
  - ✦ Branch log does not define a single path anymore
  - ✦ Replay must search every possible path


a = 0
if
0
{ ... }
if
1
if
if
if
1
if
a = 1
{ ... }
{ ... }
{ ... }
a==0
if
if
if
{ ... }
{ ... }
{ ... }
{ ... }

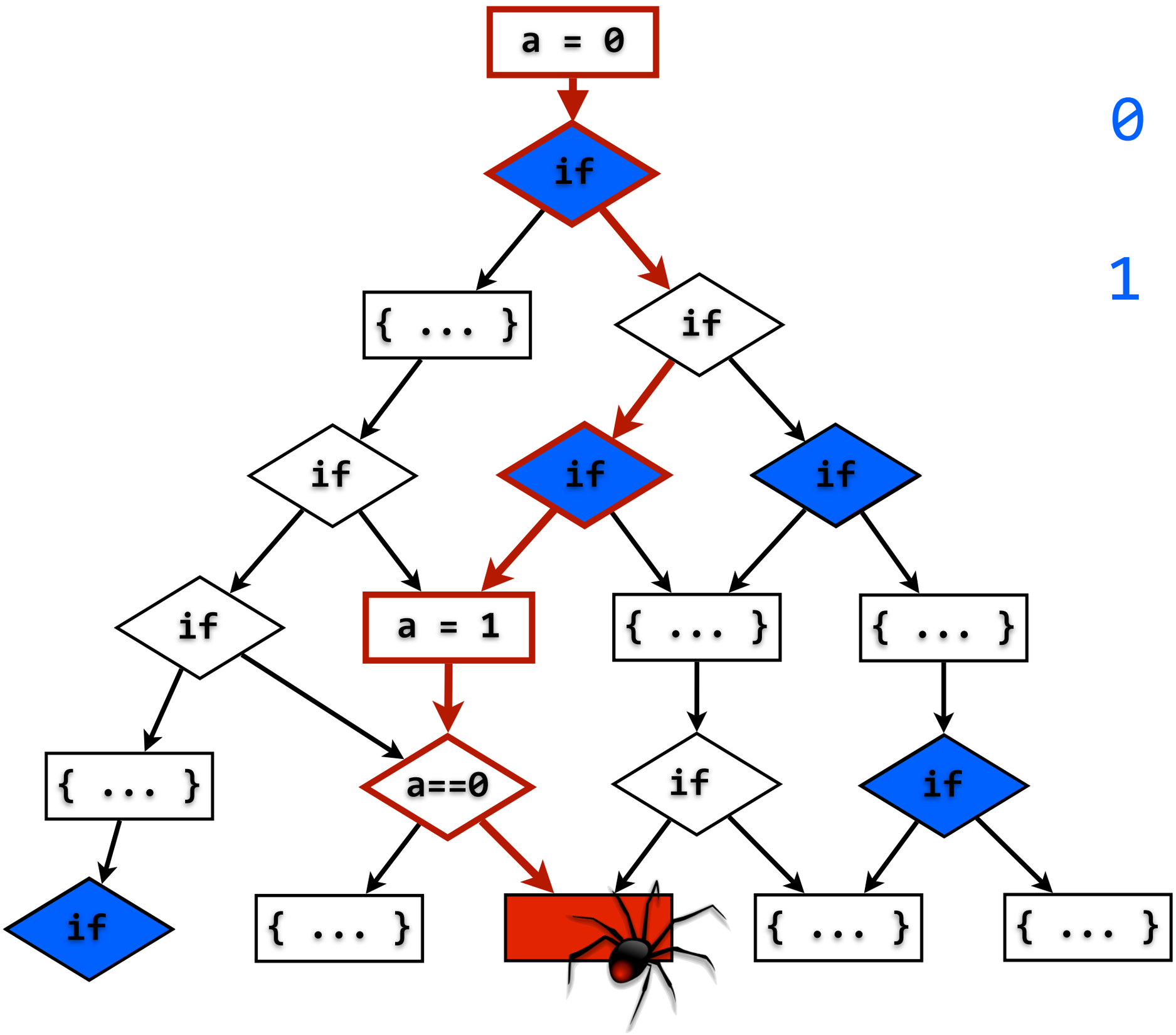

a = 0
if
0
{ ... }
if
1
if
if
if
1
if
a = 1
{ ... }
{ ... }
{ ... }
a==0
if
if
if
{ ... }
{ ... }
{ ... }
{ ... }


a = 0
if
0
{ ... }
if
if
if
if
1
a = 1
{ ... }
{ ... }
if
{ ... }
a==0
if
if
{ ... }
{ ... }
{ ... }
{ ... }


a = 0
if
0
{ ... }
if
if
if
if
1
a = 1
{ ... }
{ ... }
if
a==0
if
if
{ ... }
{ ... }
{ ... }
{ ... }

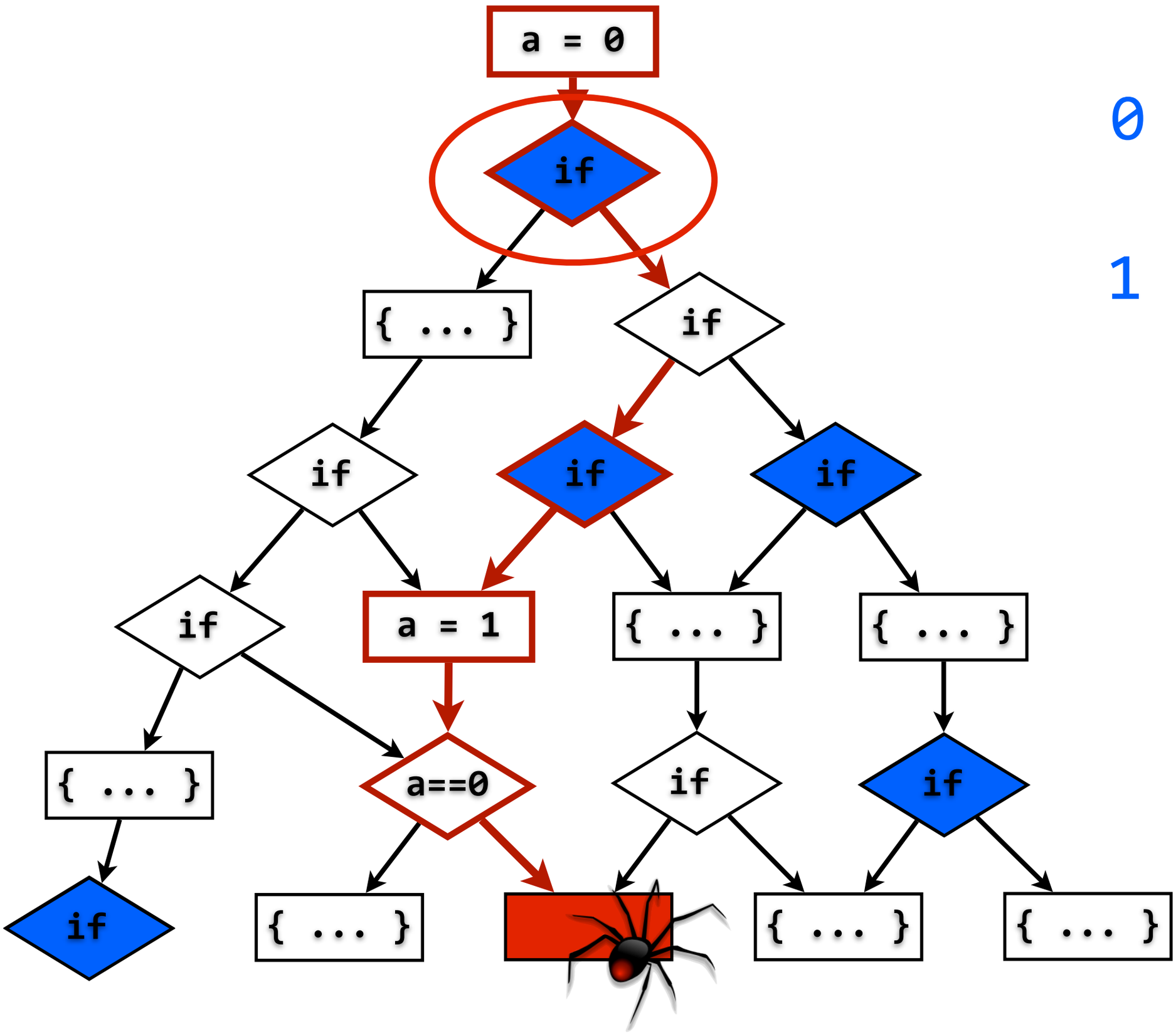
0

1



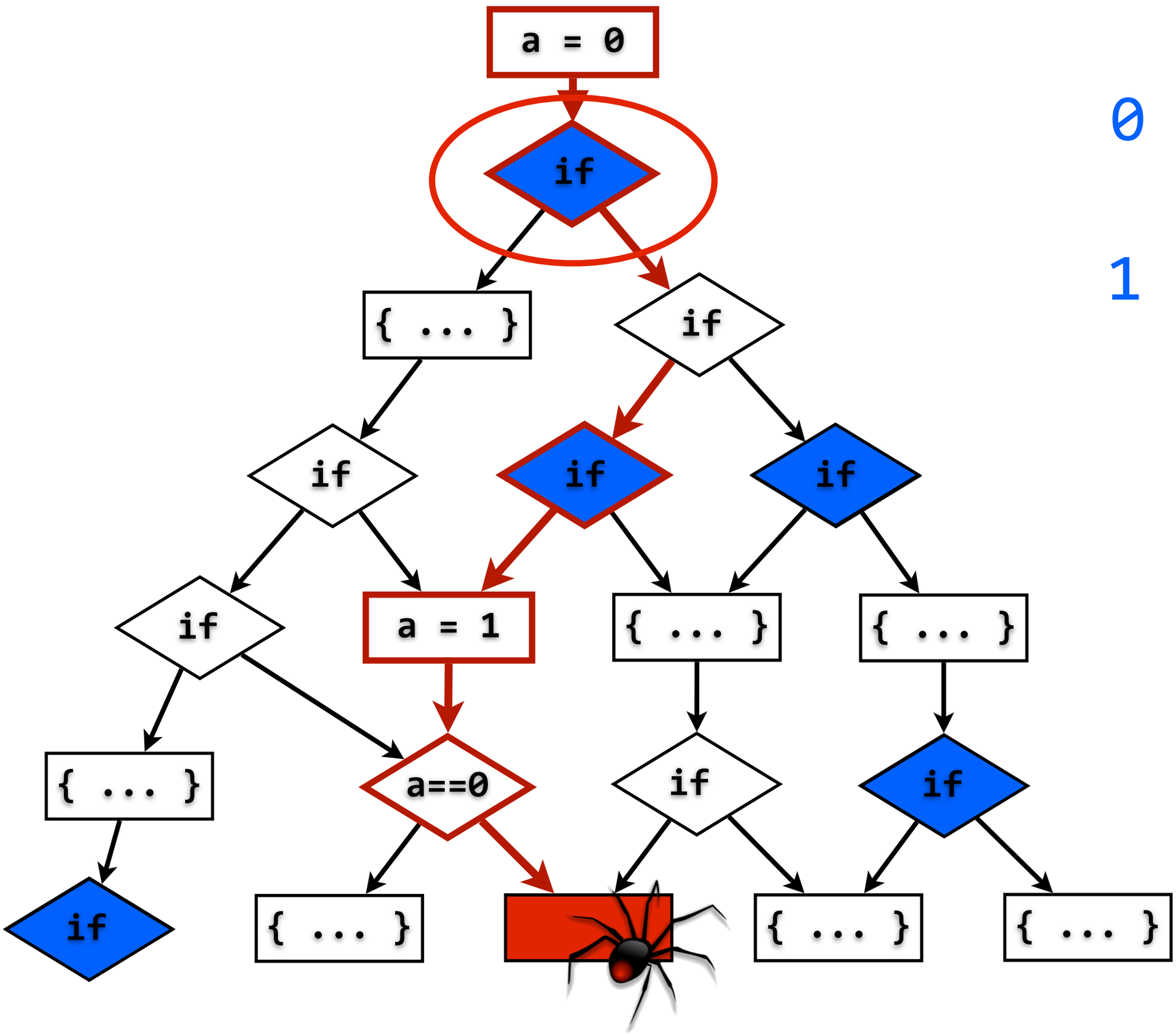
0

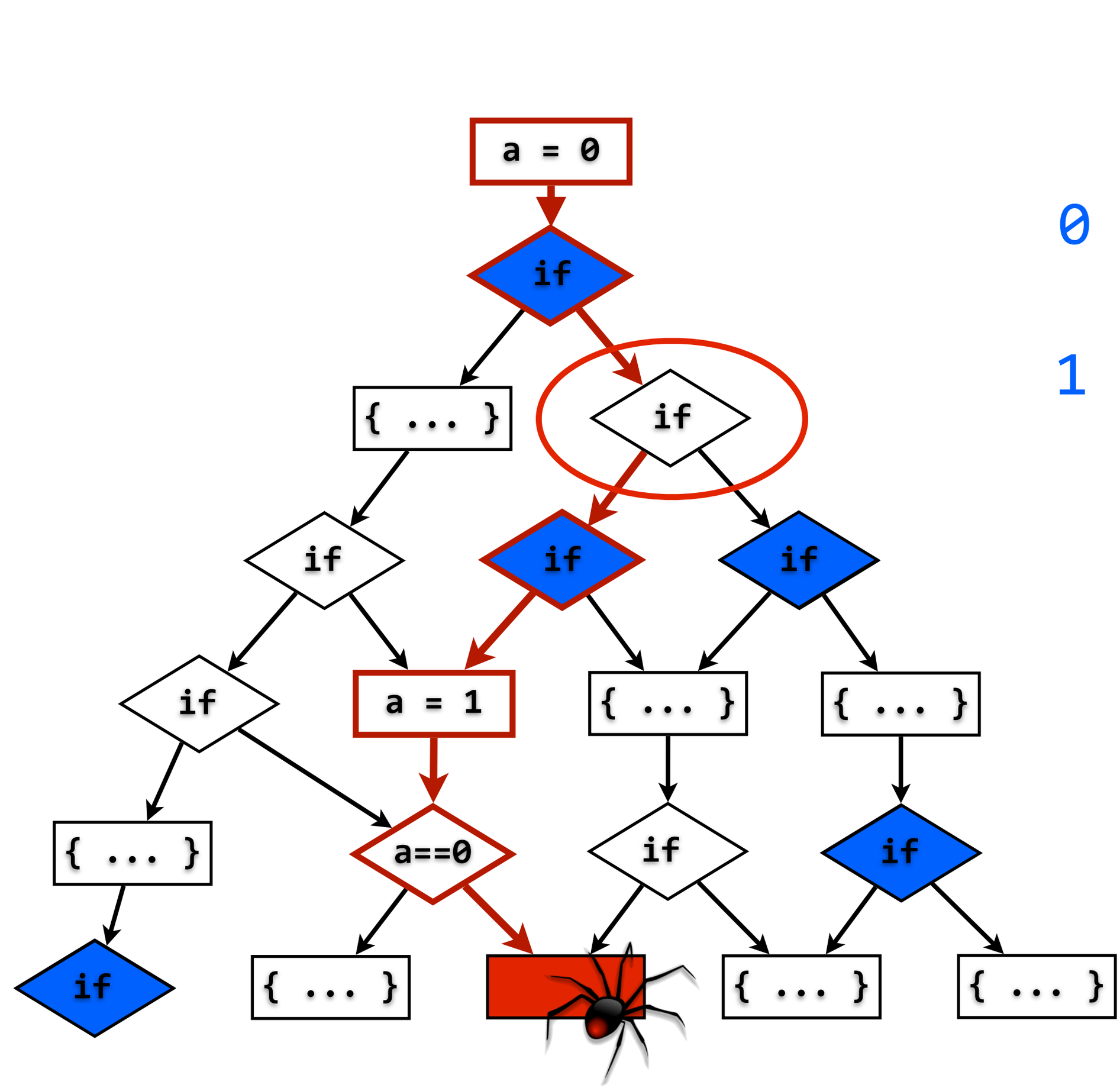
1



0

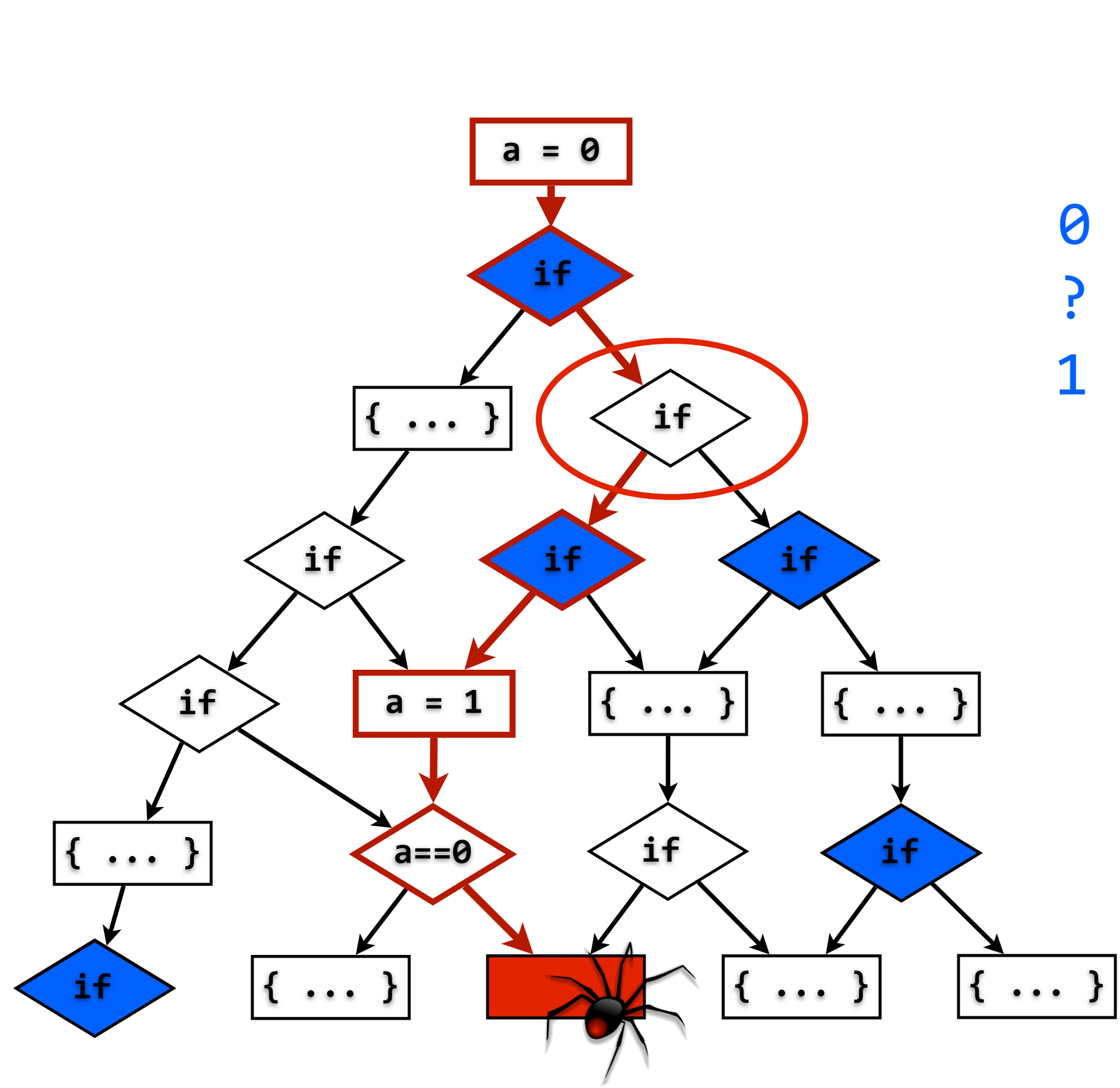
1





0

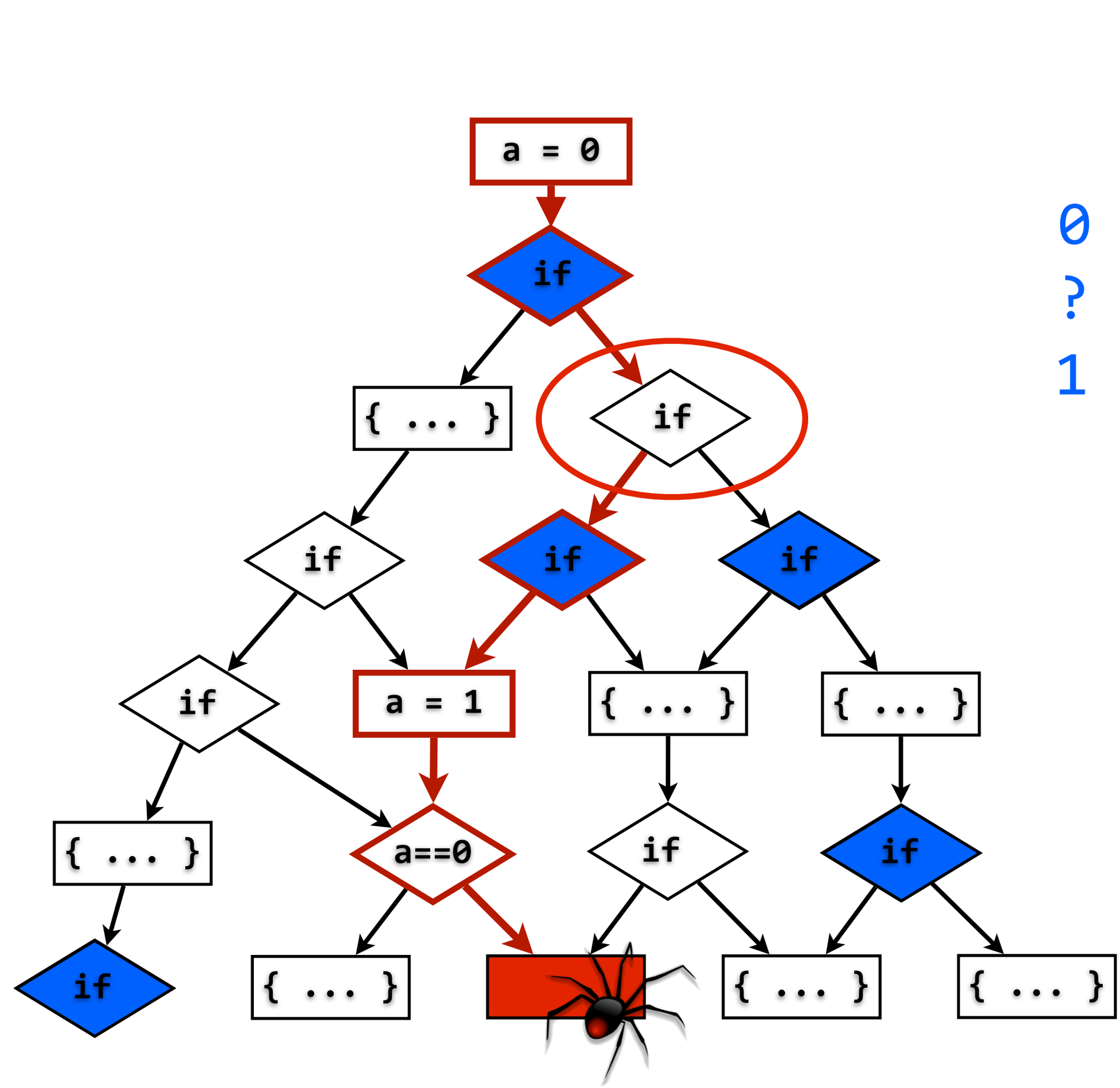
1

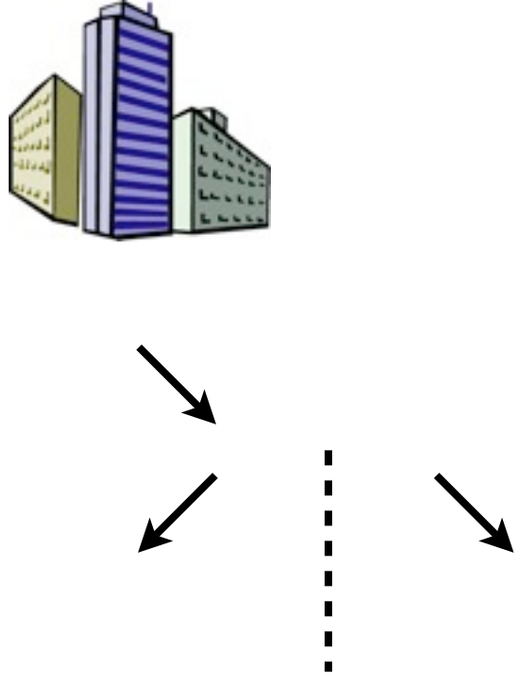


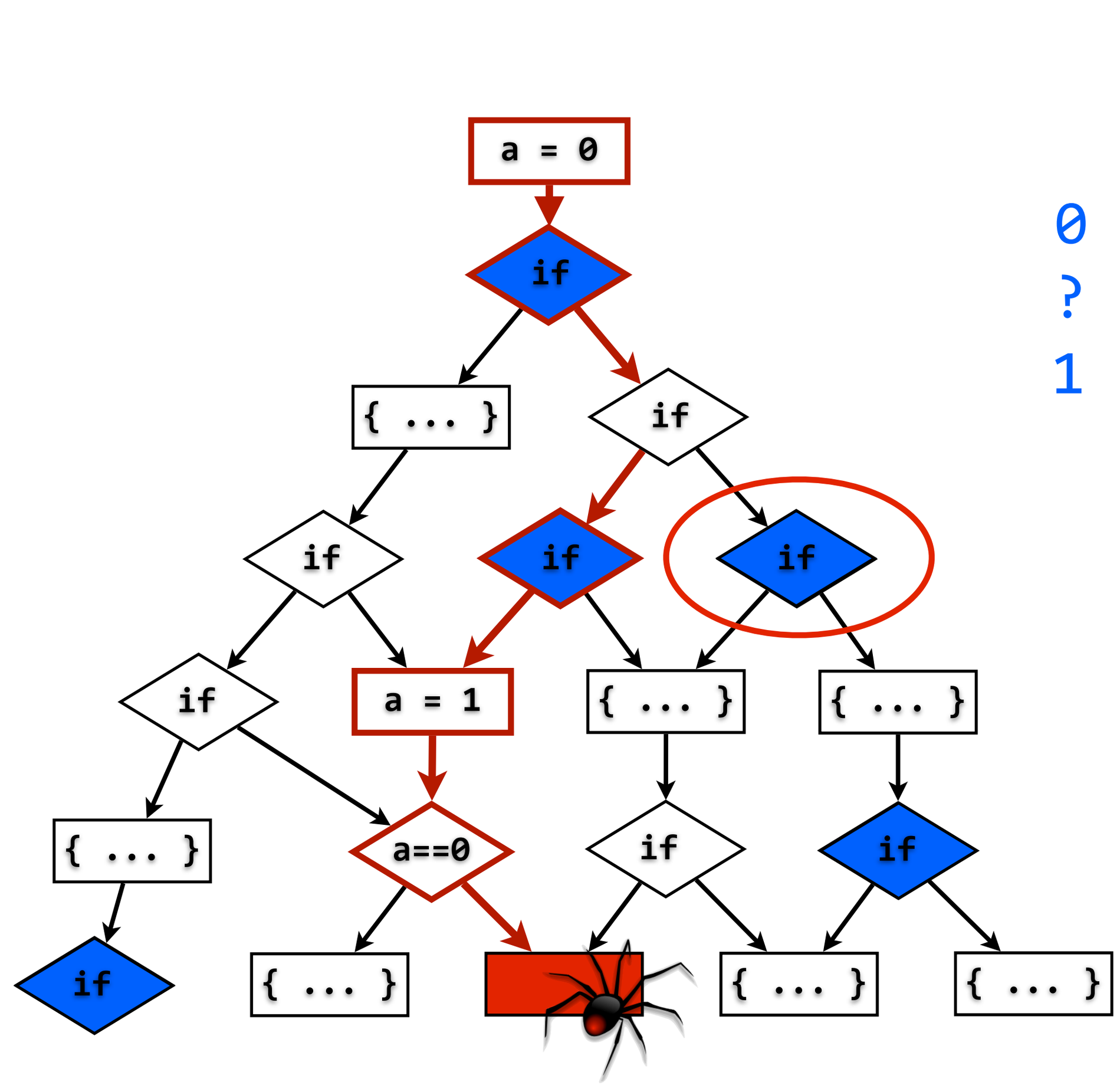
0

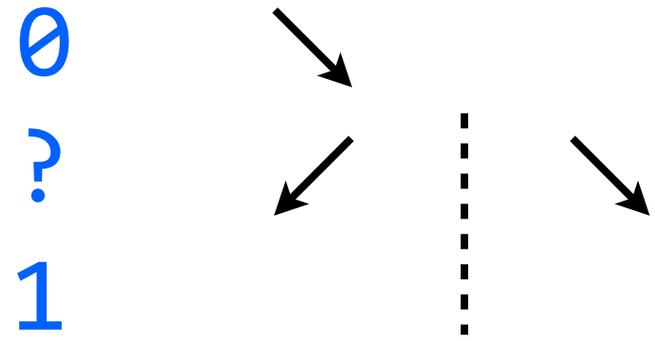
?

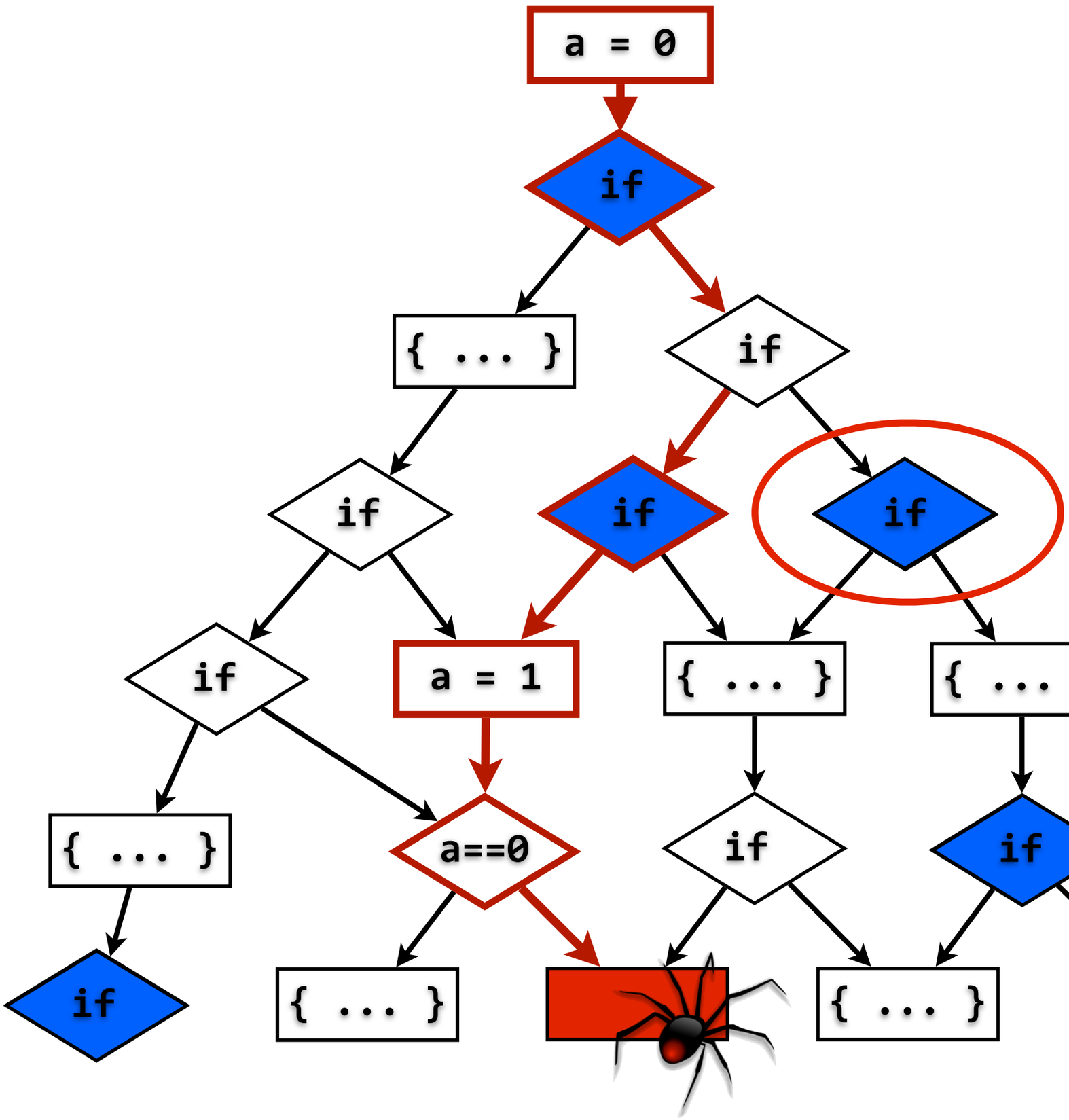
1

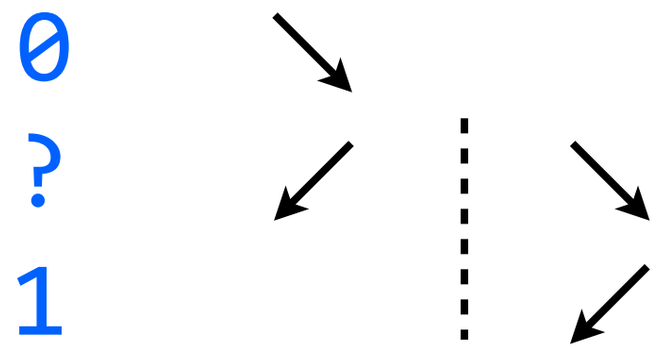

a = 0
if
{ ... }
if
if
if
if
if
a = 1
{ ... }
{ ... }
{ ... }
a==0
if
if
if
{ ... }
{ ... }
{ ... }
{ ... }

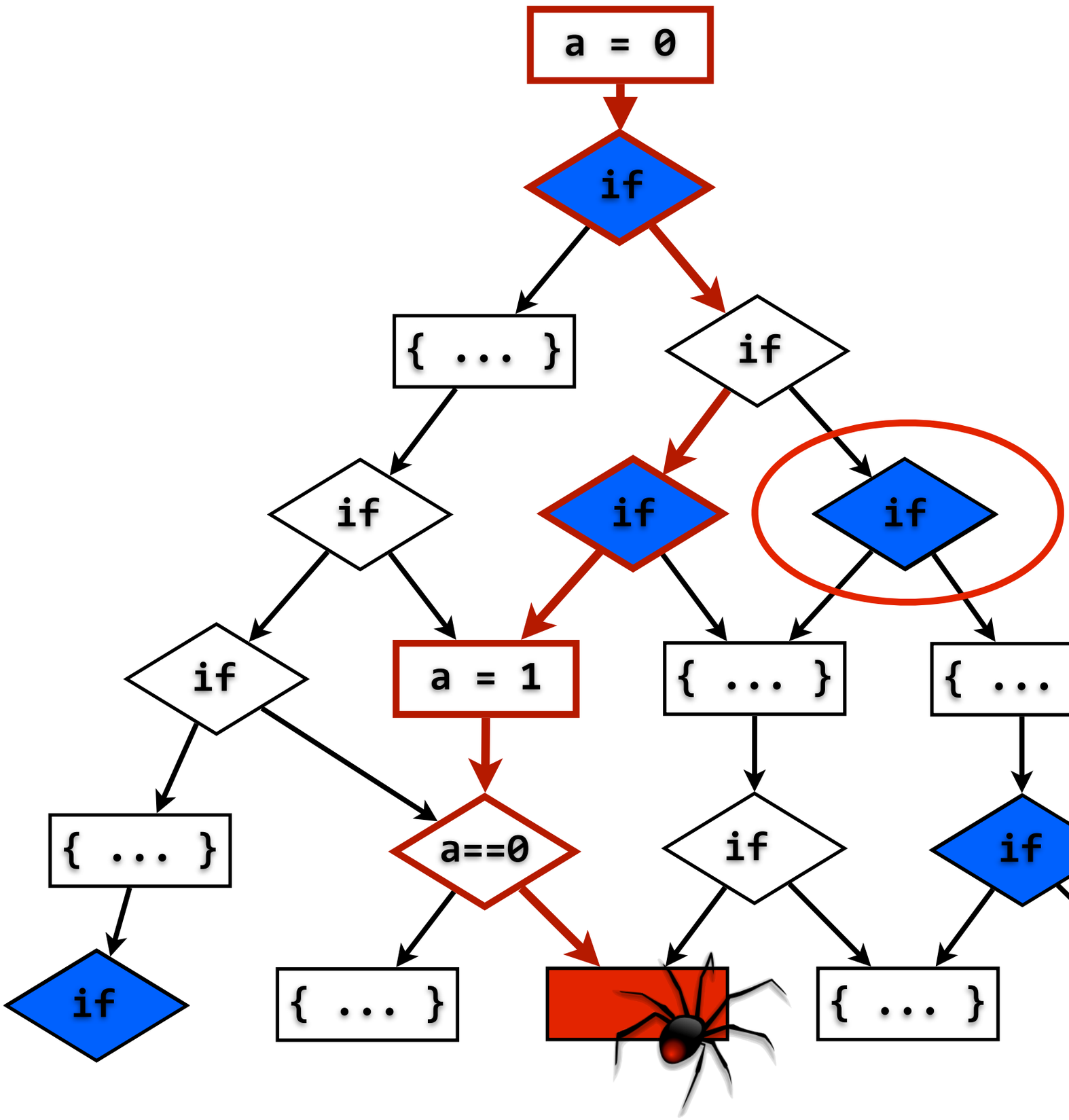

0
?
1

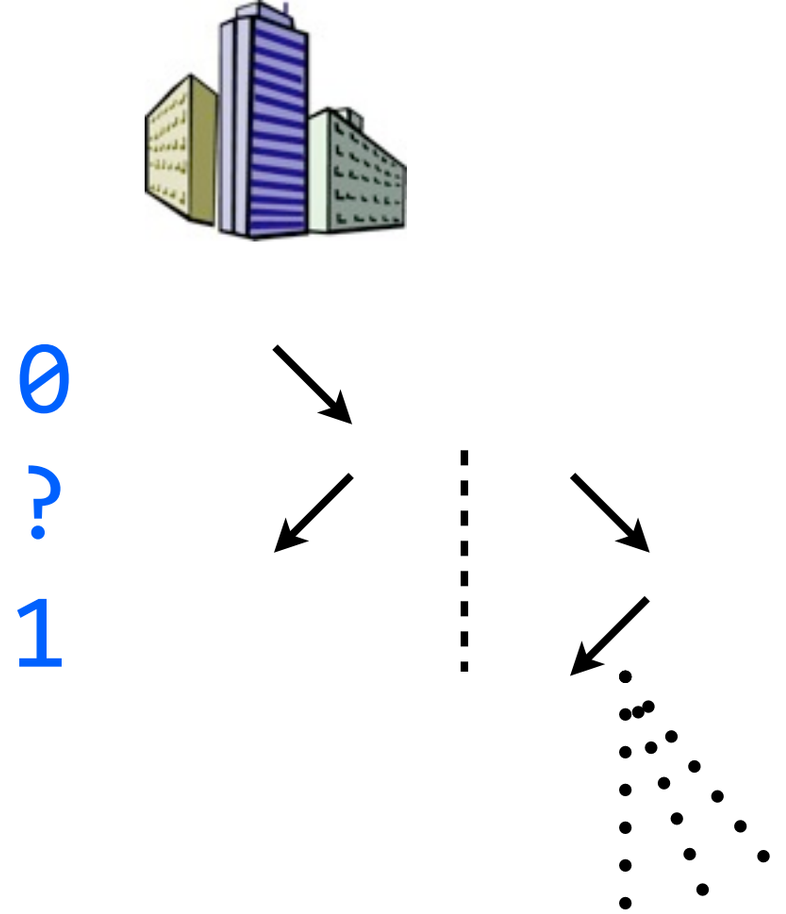

a = 0
if
{ ... }
if
if
if
if
if
a = 1
{ ... }
{ ... }
{ ... }
a==0
if
if
if
{ ... }
{ ... }
{ ... }
{ ... }

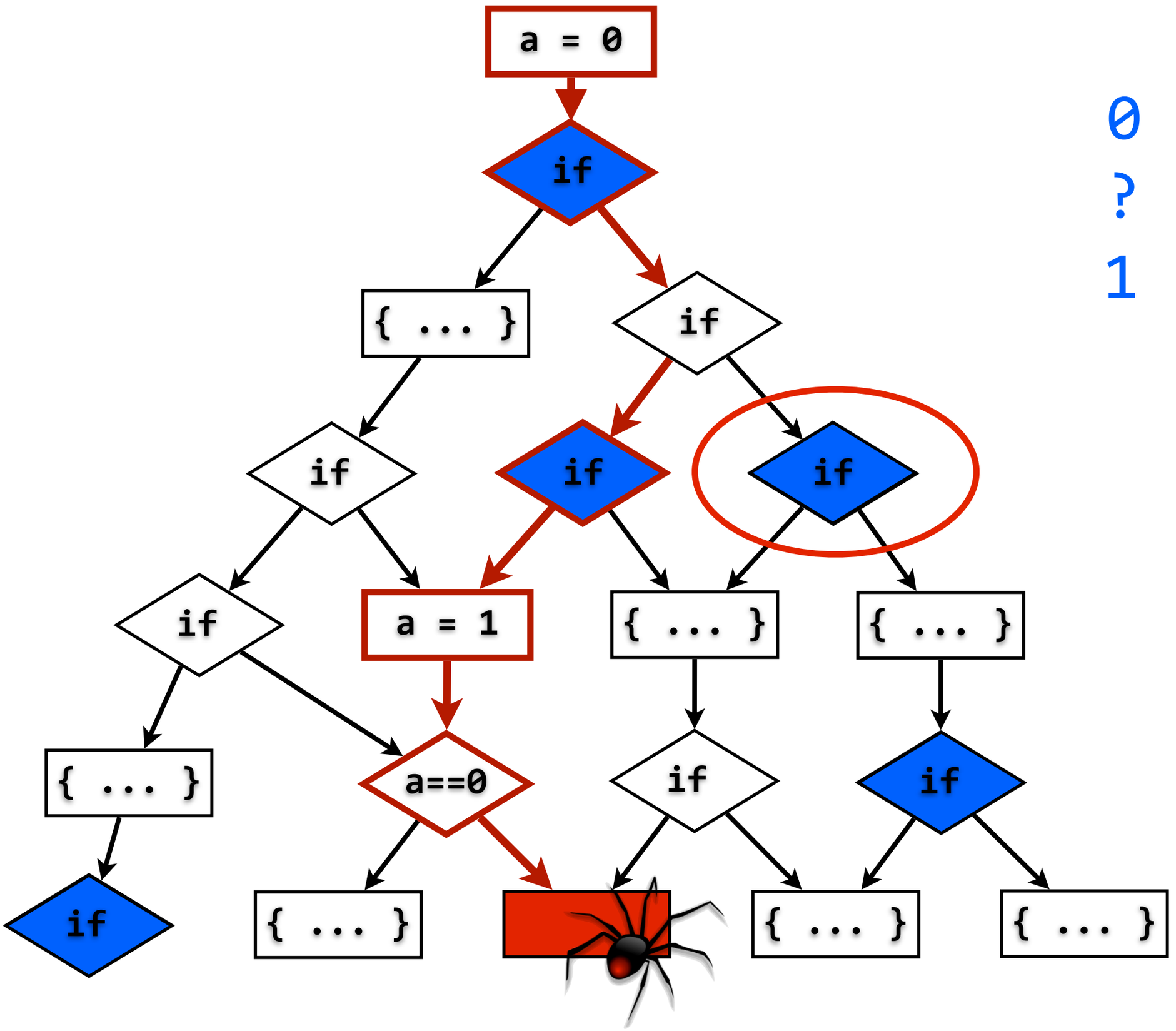

0
?
1

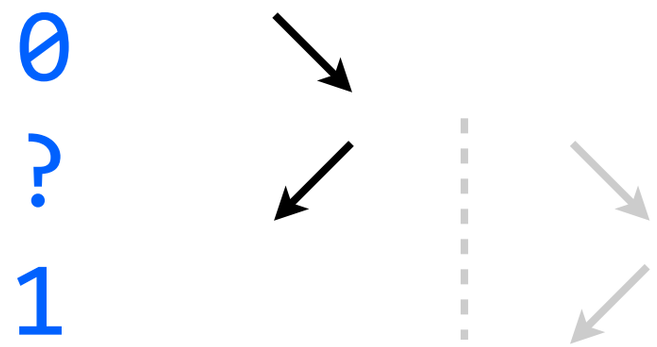

a = 0
if
{ ... }
if
if
if
if
if
a = 1
{ ... }
{ ... }
{ ... }
a==0
if
if
if
{ ... }
{ ... }
{ ... }
{ ... }

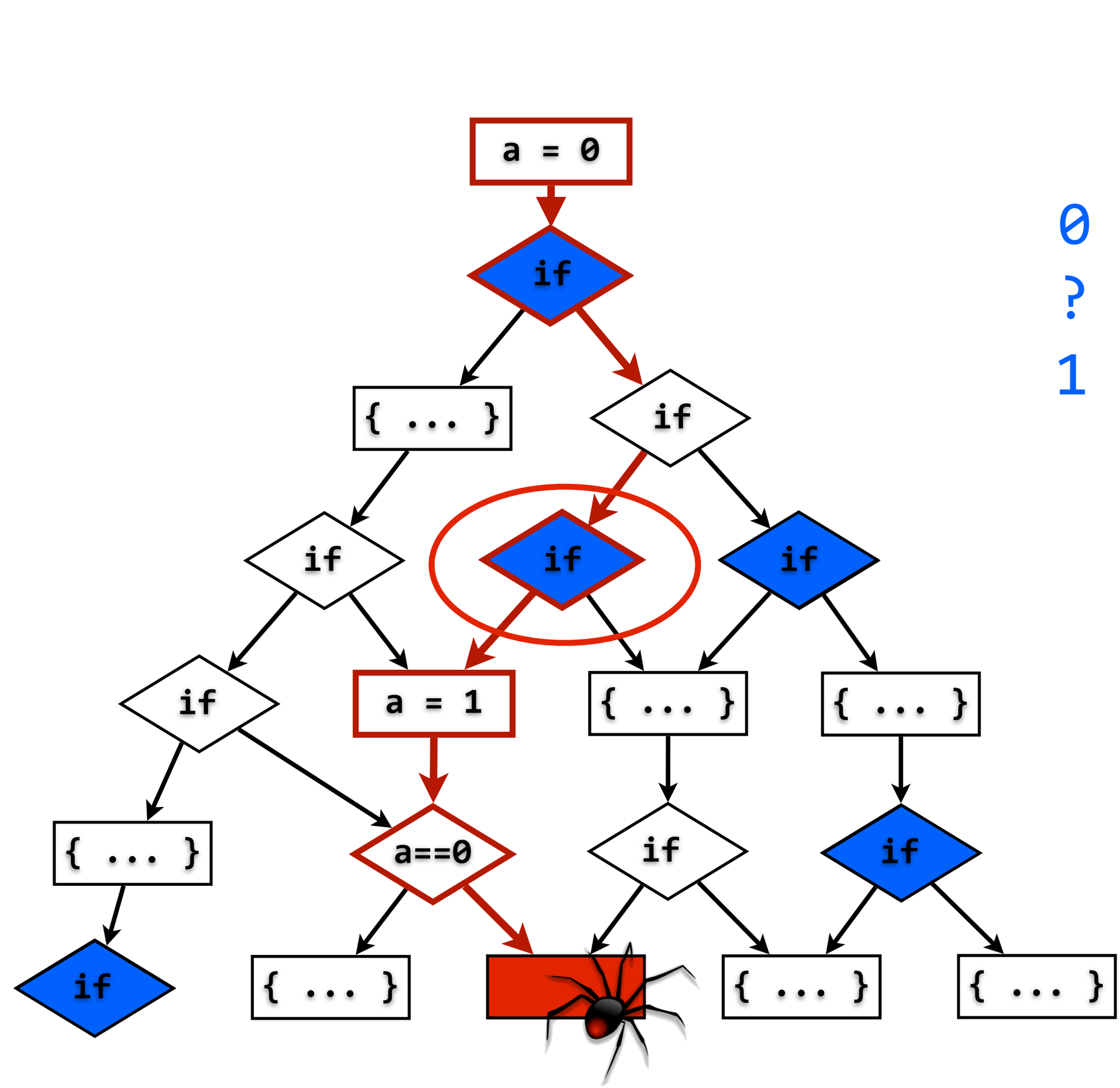

0
?
1

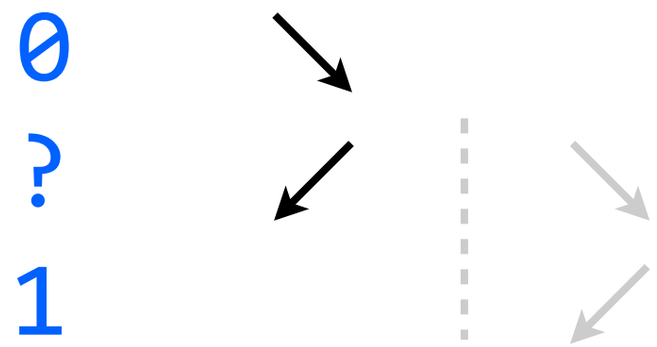

a = 0
if
{ ... }
if
if
if
if
if
a = 1
{ ... }
{ ... }
{ ... }
a==0
if
if
if
{ ... }
{ ... }
{ ... }
{ ... }

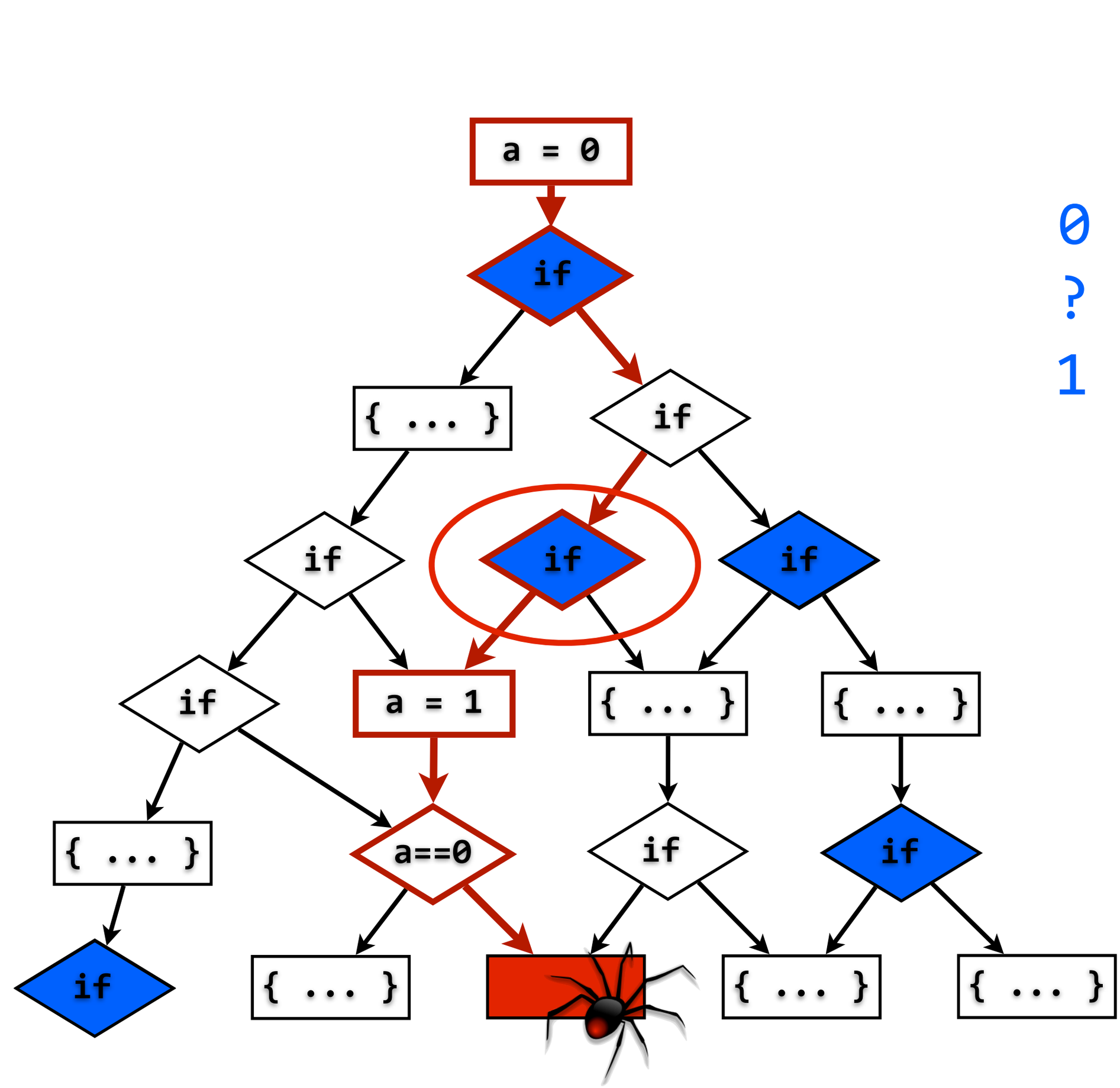

0
?
1

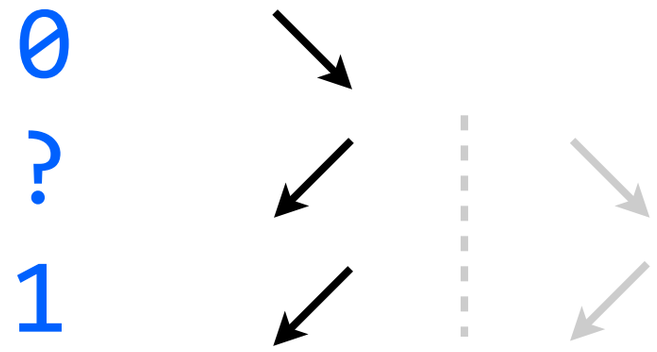

a = 0
if
{ ... }
if
if
if
if
if
a = 1
{ ... }
{ ... }
{ ... }
a==0
if
if
if
{ ... }
{ ... }
{ ... }
{ ... }

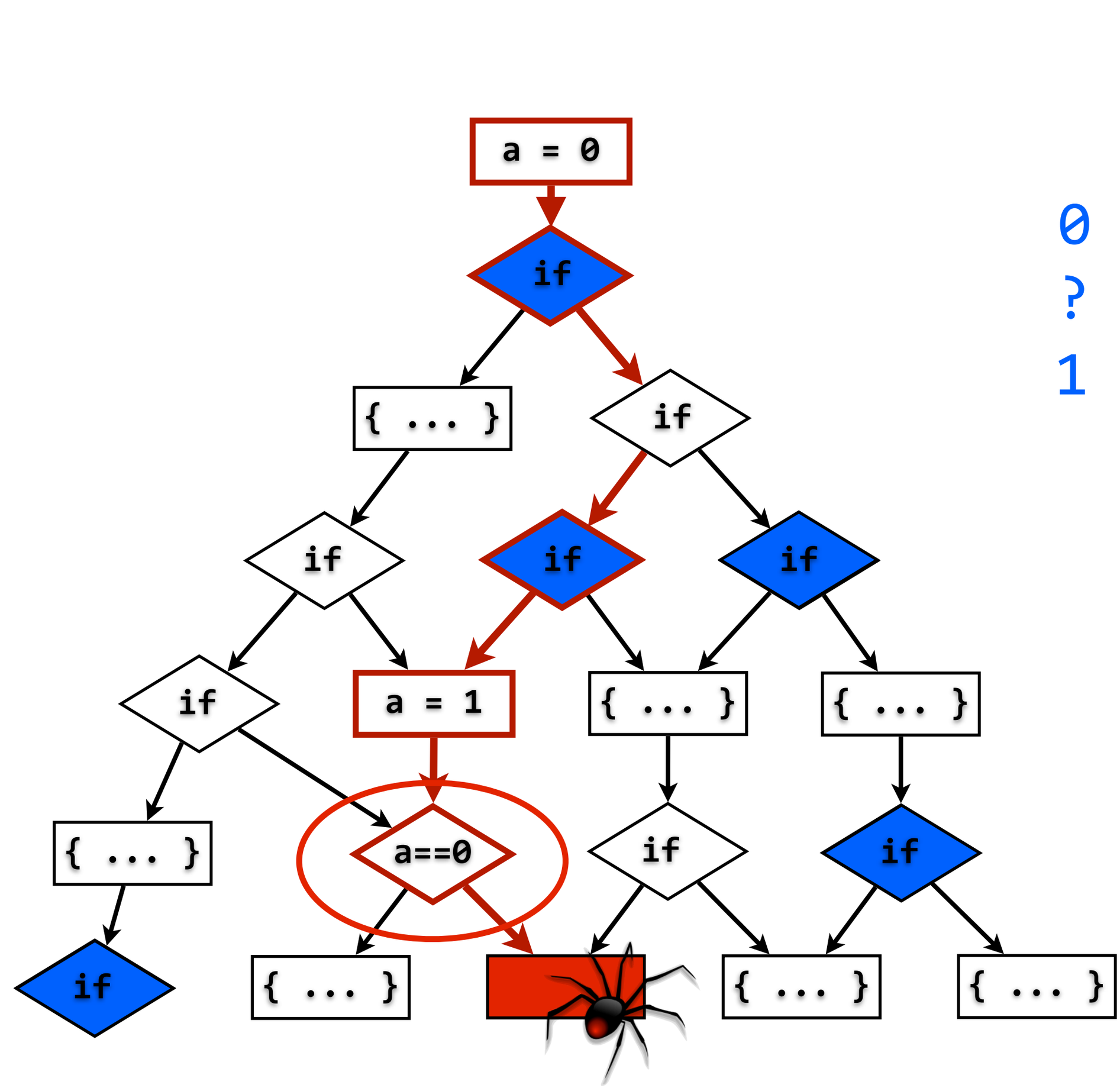

0
?
1

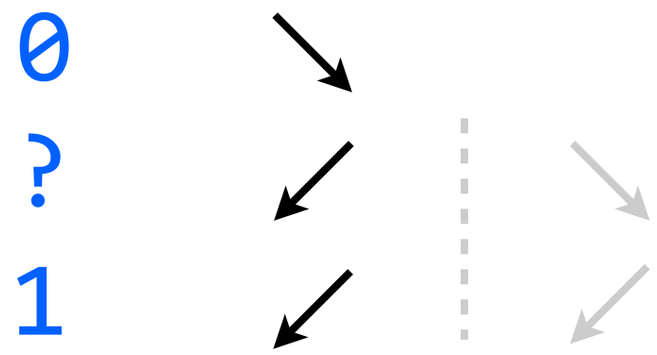

a = 0
if
{ ... }
if
if
if
if
if
a = 1
{ ... }
{ ... }
{ ... }
a==0
if
if
if
{ ... }
{ ... }
{ ... }
{ ... }

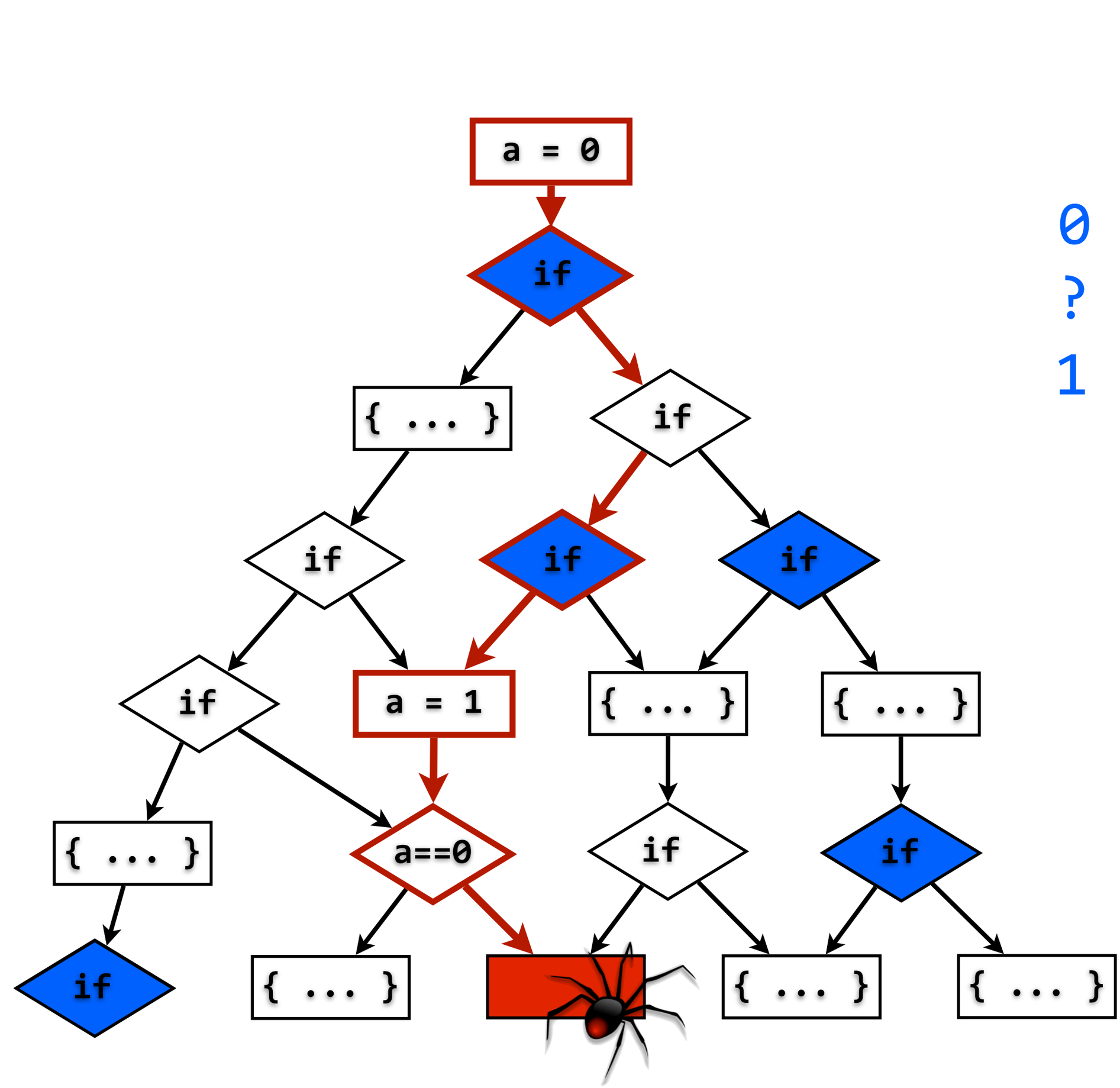

0
?
1

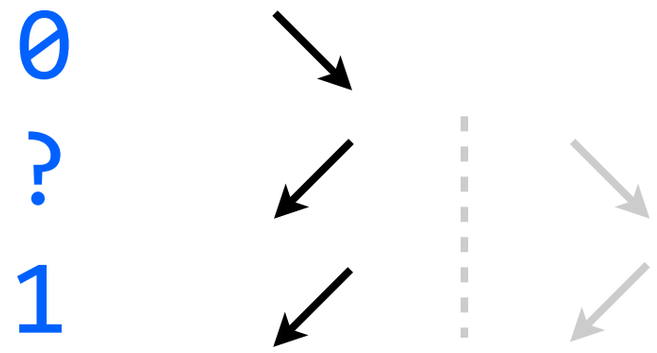

a = 0
if
{ ... }
if
if
if
if
if
a = 1
{ ... }
{ ... }
{ ... }
a==0
if
if
if
{ ... }
{ ... }
{ ... }
{ ... }


0
?
1


a = 0
if
{ ... }
if
if
if
if
if
a = 1
{ ... }
{ ... }
{ ... }
a==0
if
if
if
{ ... }
{ ... }
{ ... }
{ ... }


0
?
1


a = 0
if
{ ... }
if
if
if
if
if
a = 1
{ ... }
{ ... }
{ ... }
a==0
if
if
if
{ ... }
{ ... }
{ ... }
{ ... }


0
?
1

0
?
1

a = 0
if
{ ... }
if
{ ... }
if
a==0
{ ... }
if
{ ... }
if
{ ... }

Possibly many paths to search
Back-tracking slows down replay considerably

Dealing with non-determinism (1)

Dealing with non-determinism (1)

- Non-deterministic events:
  - system calls, thread scheduling, ...
  - influence program execution

Dealing with non-determinism (1)

- Non-deterministic events:
  - system calls, thread scheduling, ...
  - influence program execution
- Choice:
  - log and replay event
  - do not log and infer solution during replay

Dealing with non-determinism (2)

Dealing with non-determinism (2)

- Non-determinism in system calls:

Dealing with non-determinism (2)

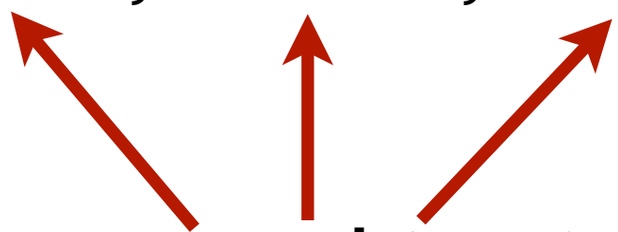
- Non-determinism in system calls:

```
select(n, readfds, writefds, exceptfds, timeout) ;
```

Dealing with non-determinism (2)

- Non-determinism in system calls:

```
select(n, readfds, writefds, exceptfds, timeout) ;
```



Returns any combination of ready file descriptors

Implementation

Implementation

- Static Analysis algorithm:
  - ✦ uses CIL [Necula 2002]

Implementation

- Static Analysis algorithm:
  - ✦ uses CIL [Necula 2002]
- Dynamic analysis and Replay:
  - ✦ Oasis symbolic execution engine

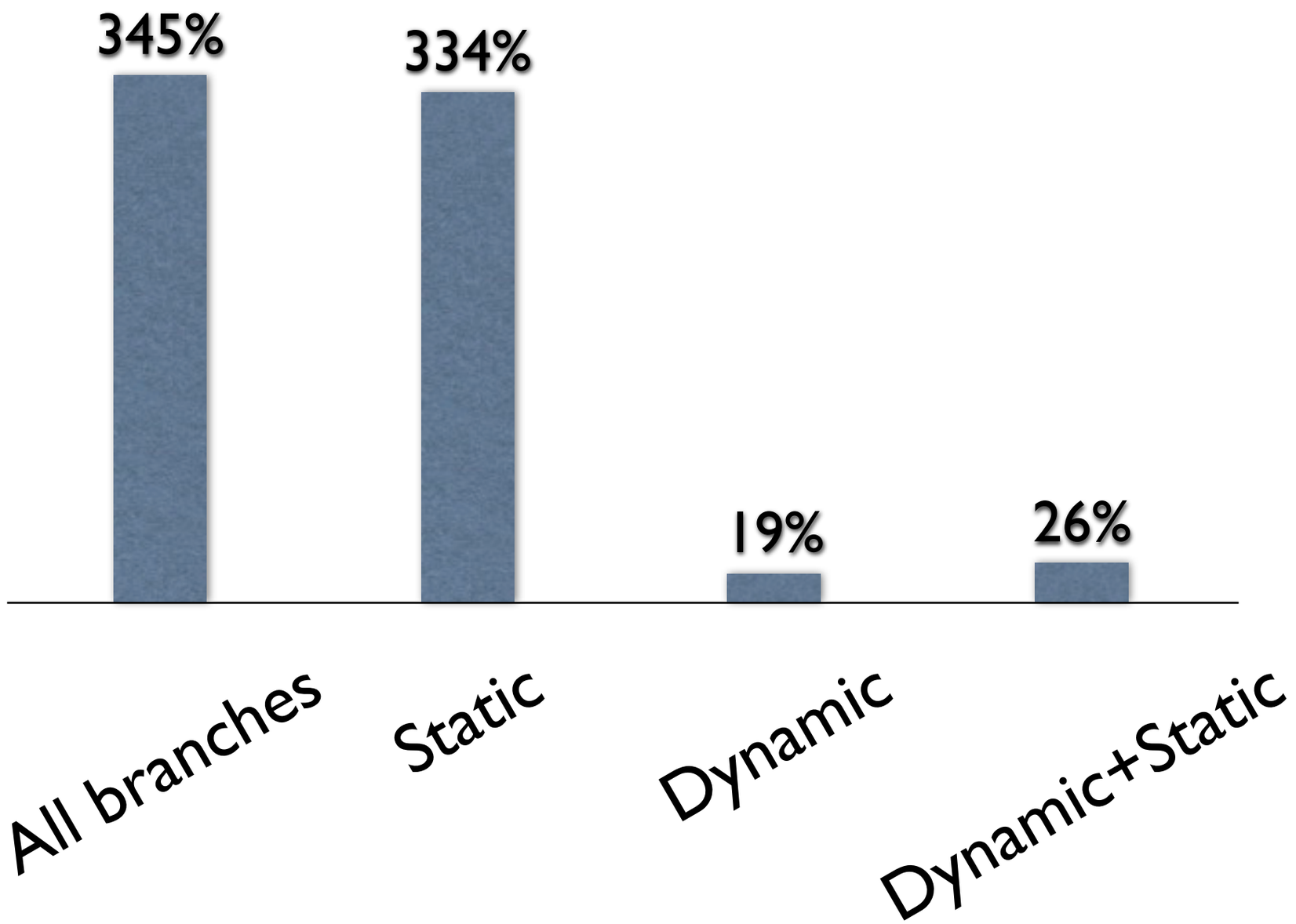
Implementation

- Static Analysis algorithm:
  - ✦ uses CIL [Necula 2002]
- Dynamic analysis and Replay:
  - ✦ Oasis symbolic execution engine
- Program (and libraries) instrumentation:
  - ✦ one bit per branch

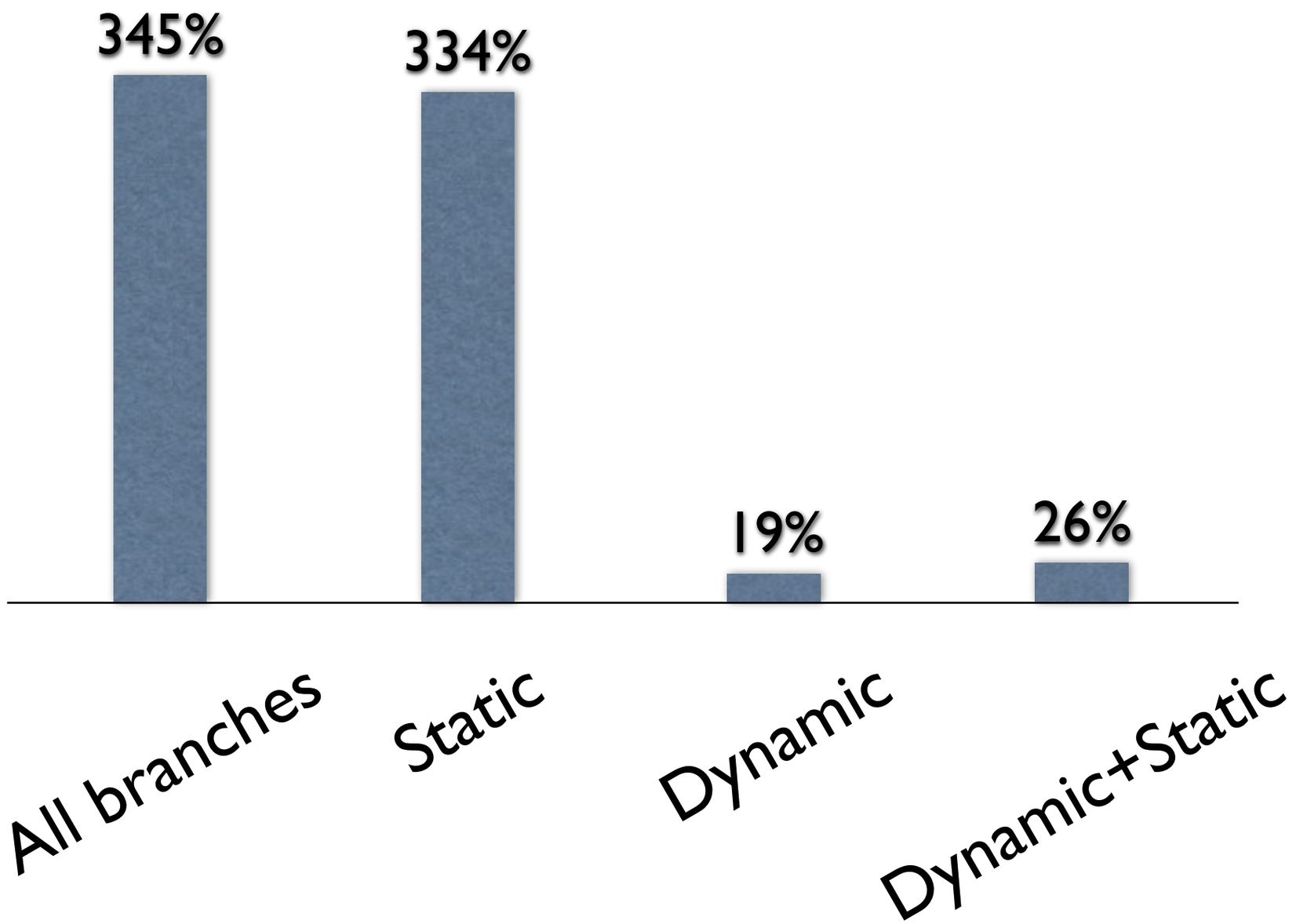
Evaluation

- Use micro-benchmarks and real programs:
  - ✦ uServer, web server (32 KLOC)
- Instrumentation performance:
  - ✦ CPU overhead
- Replay performance:
  - ✦ Replay time for specific input scenarios

CPU overhead



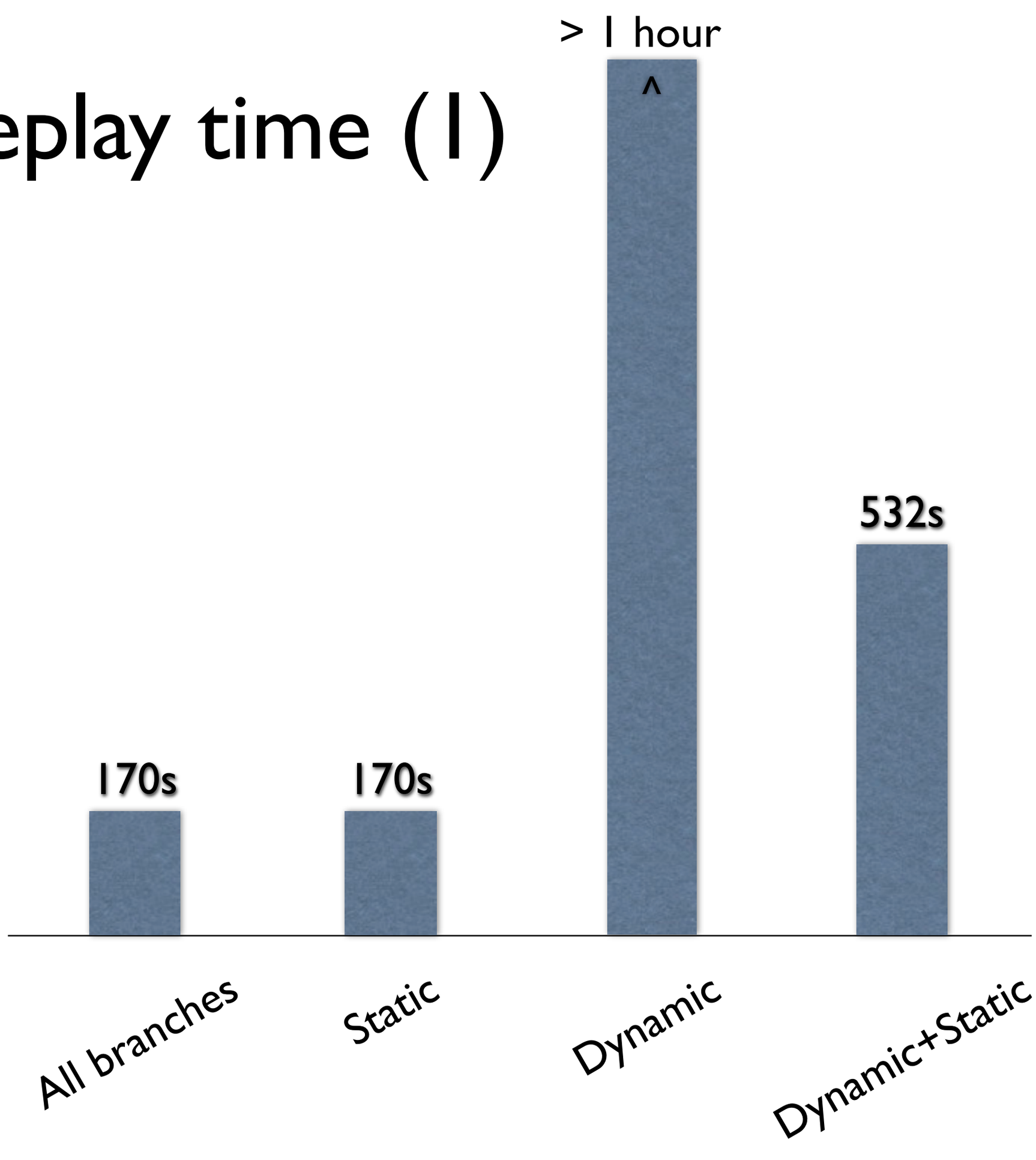
CPU overhead



Results without system call return values are similar

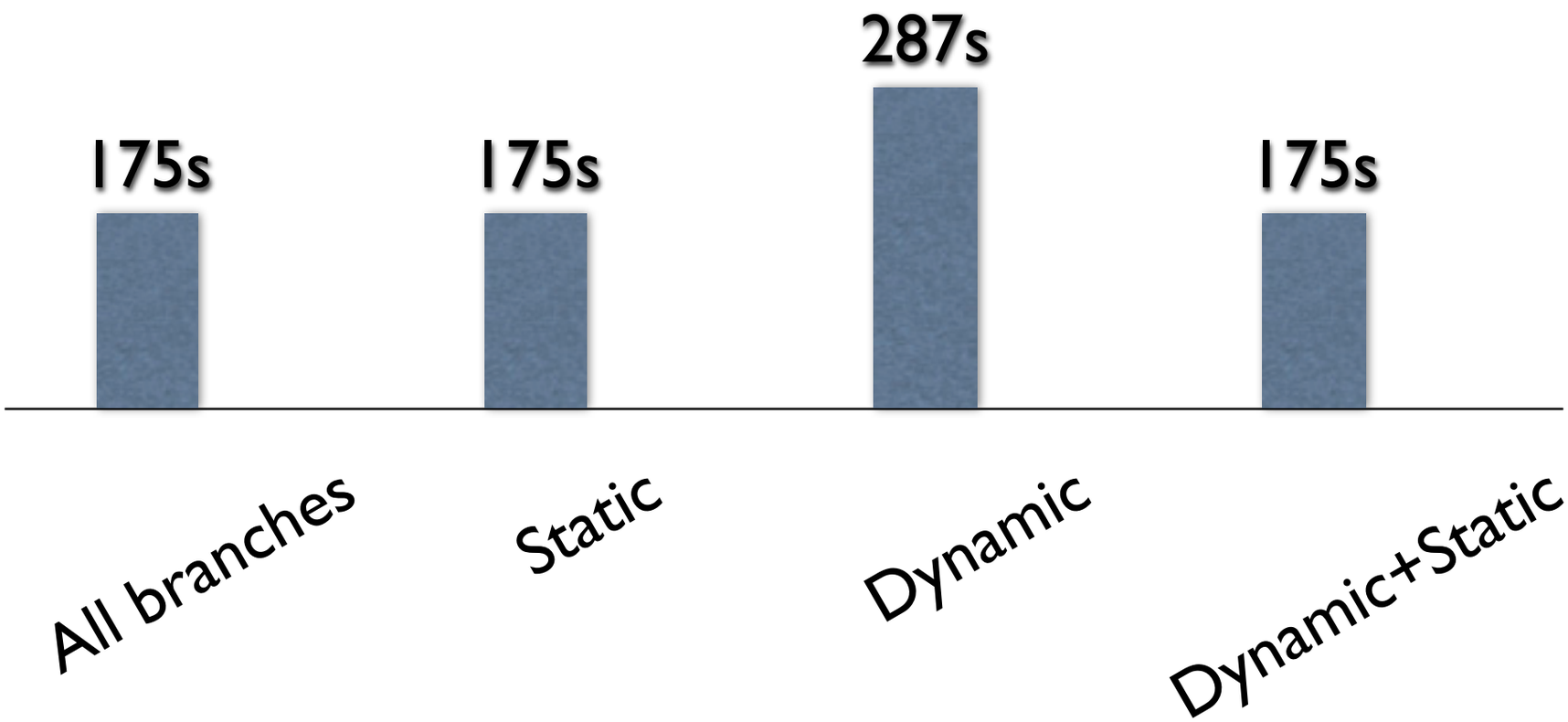
Replay time (1)

Replay time (1)

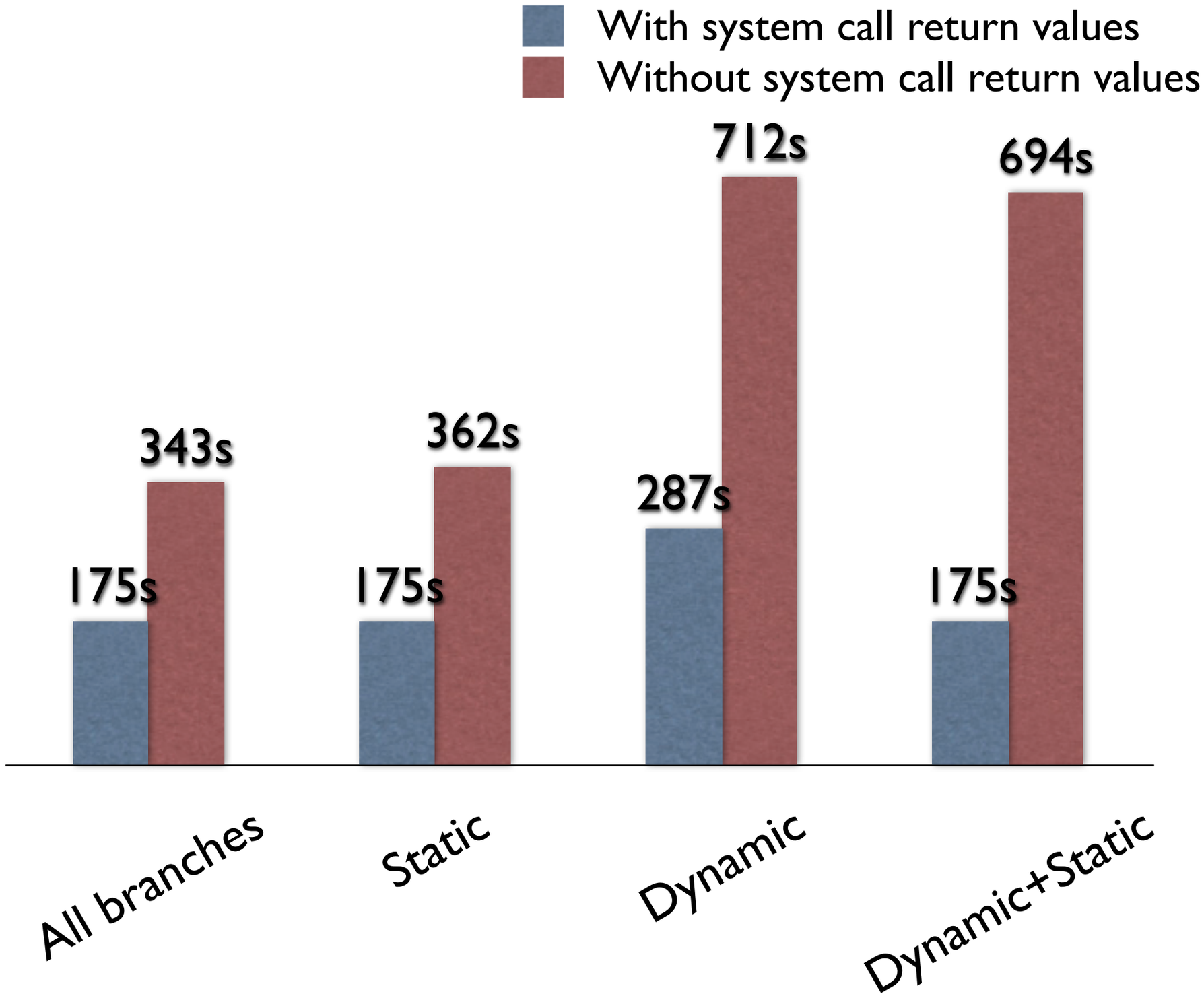


Replay time (2)

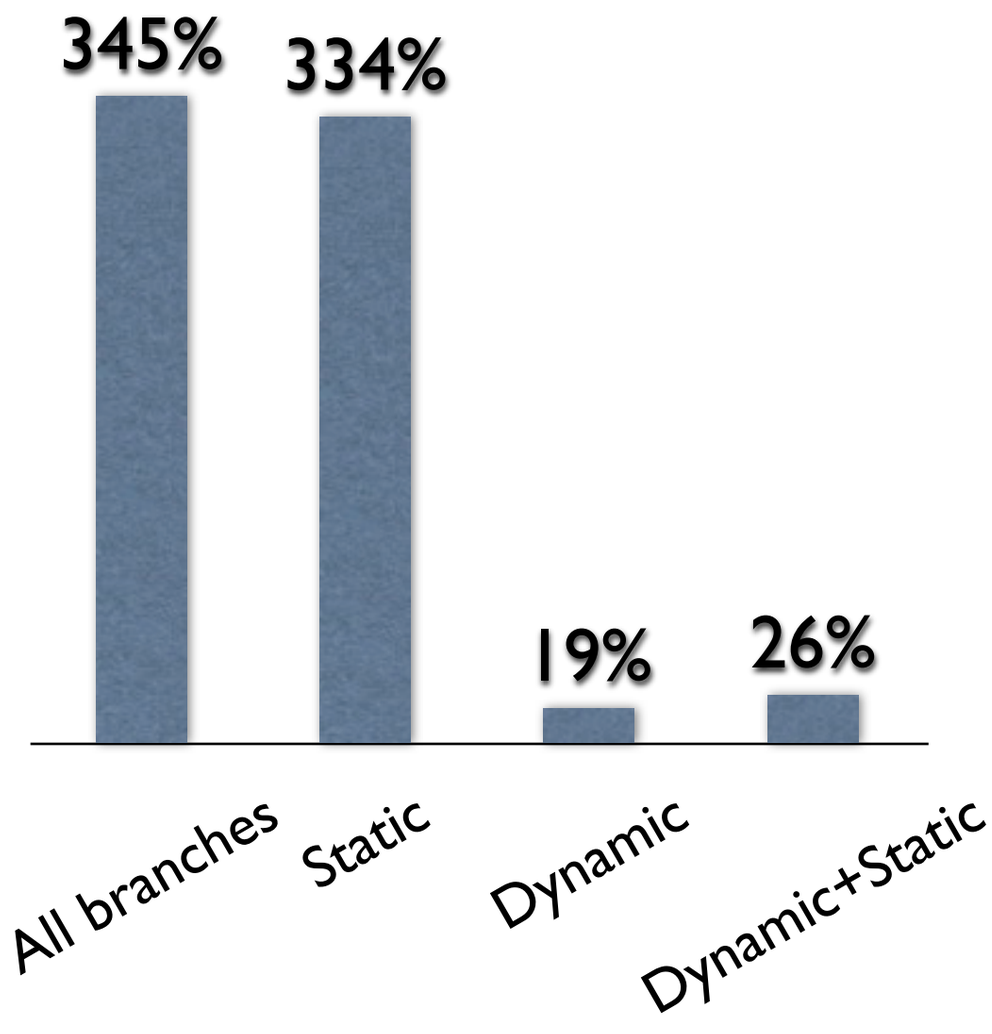
Replay time (2)



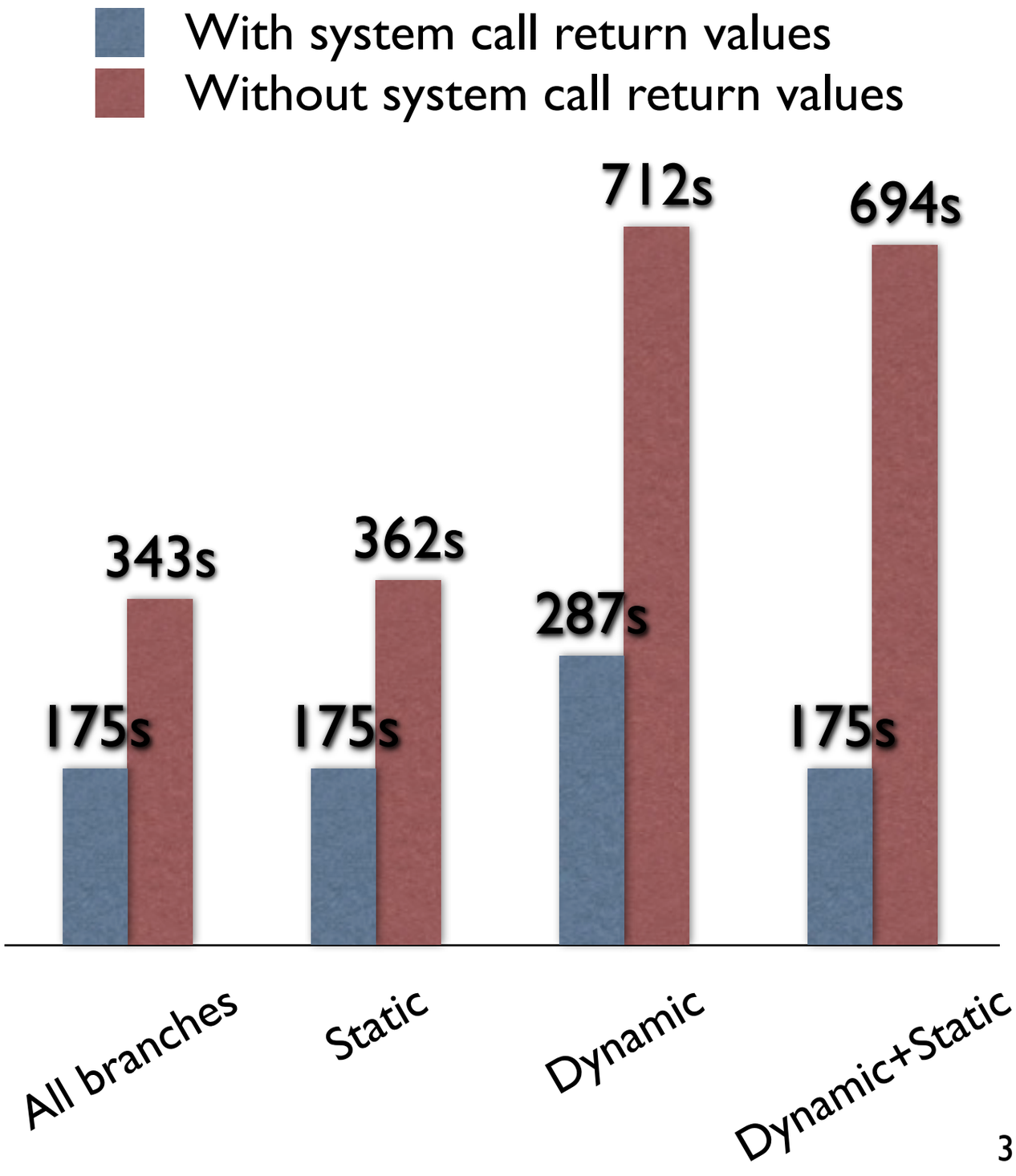
Replay time (2)



CPU overhead



Replay time



Discussion

- Long-running applications:
  - ✦ Replaying from a checkpoint
- Multi-threaded applications:
  - ✦ Branch-log per thread
  - ✦ Logging or inferring thread schedules

Related work

Related work

- Multi-threaded applications:
  - ✦ Output deterministic replay [Altekar 2009]
  - ✦ Execution Synthesis [Zamfir 2010]

Related work

- Multi-threaded applications:
  - ✦ Output deterministic replay [Altekar 2009]
  - ✦ Execution Synthesis [Zamfir 2010]
- Privacy:
  - ✦ Better Bug Reporting for Better Privacy [Castro 2008]

Related work

- Multi-threaded applications:
  - ✦ Output deterministic replay [Altekar 2009]
  - ✦ Execution Synthesis [Zamfir 2010]
- Privacy:
  - ✦ Better Bug Reporting for Better Privacy [Castro 2008]
- Branch logging:
  - ✦ TraceBack [Ayers 2005]

Conclusion

- Study tradeoff debugging time v.s. instrumentation overhead
- Static and dynamic analysis to optimize instrumentation of branches
- Symbolic execution for replay
- Combined dynamic+static strikes the best compromise

Thank you !

Olivier Crameri
olivier.crameri@a3.epfl.ch