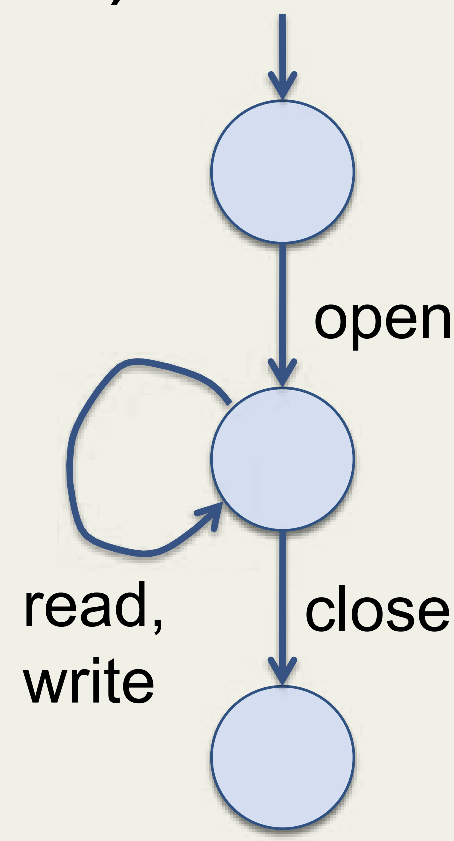
Inferring Likely Deterministic Specifications for Multithreaded Programs

Jacob Burnim
Koushik Sen
Parallel Computing Laboratory
University of California, Berkeley

Specification Inference

- Inferring temporal specs (protocols):
  - *Ammons, et al*, POPL 02
  - *Whaley, et al*, ISSTA 02
  - *Livshits, Zimmermann*, FSE 05
  - Perrecotta*: Yang, et al, ICSE 06*
  - JADET*: Wasylkowski, et al, FSE 07*
  - *Acharya, et al*, *FSE 07*
  - JAVERT: *Gabel, Su*, ICSE 08
  - many others …



Specification Inference

- Inferring likely *invariants:*

$$0 \le i < N \qquad\qquad sum = \sum_{j=0}^{i-1} A[j] \ \wedge \ i = N + 1$$

  - DAIKON*: Ernst, et al., ICSE 00*
  - DIDUCE*: Hangal, Lam, ICSE 02*
  - DySy*: Csallner, et al., ICSE 08*
  - *Gulwani, et al., VMCAI 09*
  - many others …

Parallel Programming is Hard

- **Key Culprit:** Nondeterminism.
  - Interleaving of parallel threads
- Determinism key to parallel correctness.
  - Same input ==> semantically same output.
  - Parallelism is wrong if some schedules give a correct answer while others don't.
- **Previously:** Help programmers make their parallel code deterministic.
  - Assertion framework to specify determinism.

Advantages of Deterministic Specs

Burnim, Sen. *Asserting and Checking Determinism for Multithreaded Programs*. FSE 2009.

- Lightweight spec of parallel correctness
  - Independent of functional specification
  - Decompose correctness efforts
- Useful for documentation
- Can effectively test deterministic specs
  - Combine with testing tools to distinguish harmful from benign data races, etc.

Advantages of Deterministic Specs

Burnim, Sen. *Asserting and Checking Determinism for Multithreaded Programs*. FSE 2009.

Goal: Automatically infer deterministic specifications by observing sample program runs.

- Useful for documentation
- Can effectively test deterministic specs
  - Combine with testing tools to distinguish harmful from benign data races, etc.

Advantages of Deterministic Specs

Burnim, Sen. *Asserting and Checking Determinism for Multithreaded Programs*. FSE 2009.

Goal: Automatically infer deterministic specifications by observing sample program runs.

- Useful for documentation

Result: Recover our previous manual specifications for most benchmarks.

Outline

- Motivation and Overview
- **Background: Deterministic Specs**
- Specification Inference Problem
- Inferring Deterministic Specifications
- Experimental Evaluation
- Related Work
- Conclusions

Background: Deterministic Specs

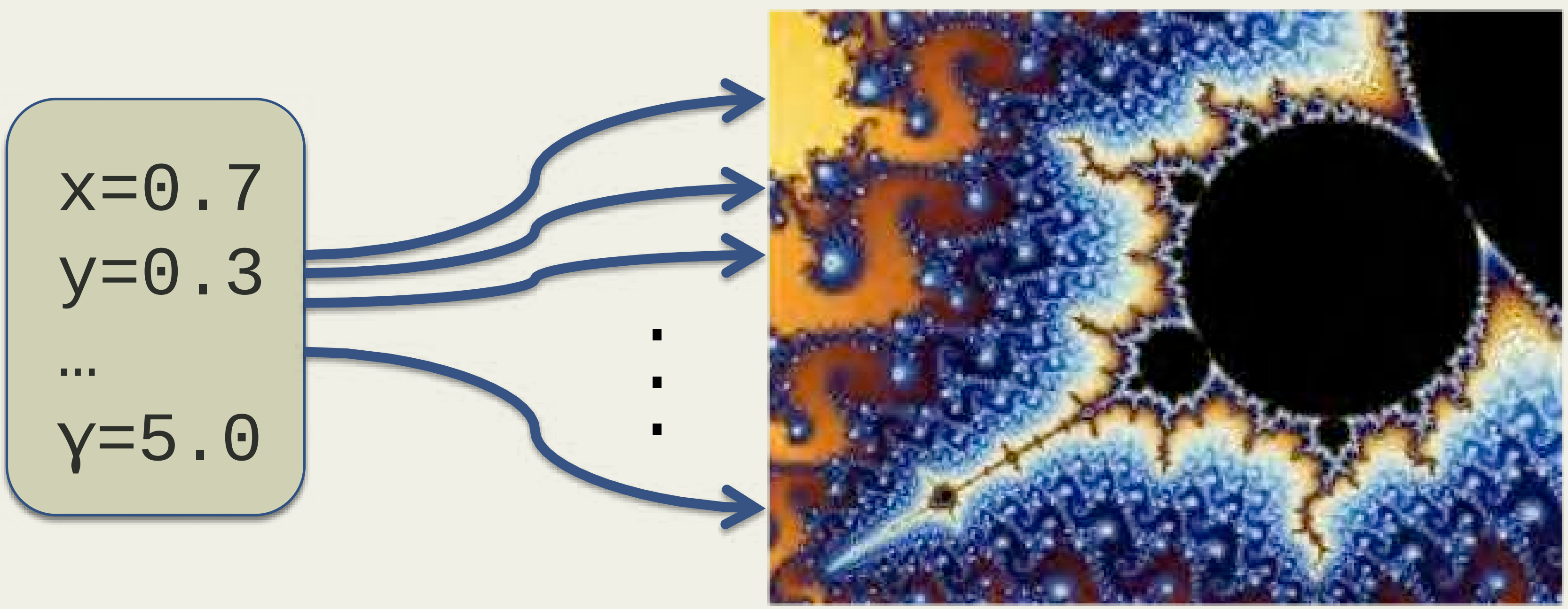
```
// Parallel fractal render.
mandelbrot(params, img);
```

- Want to assert **parallel correctness**.

Background: Deterministic Specs

```
// Parallel fractal render.
mandelbrot(params, img);
```

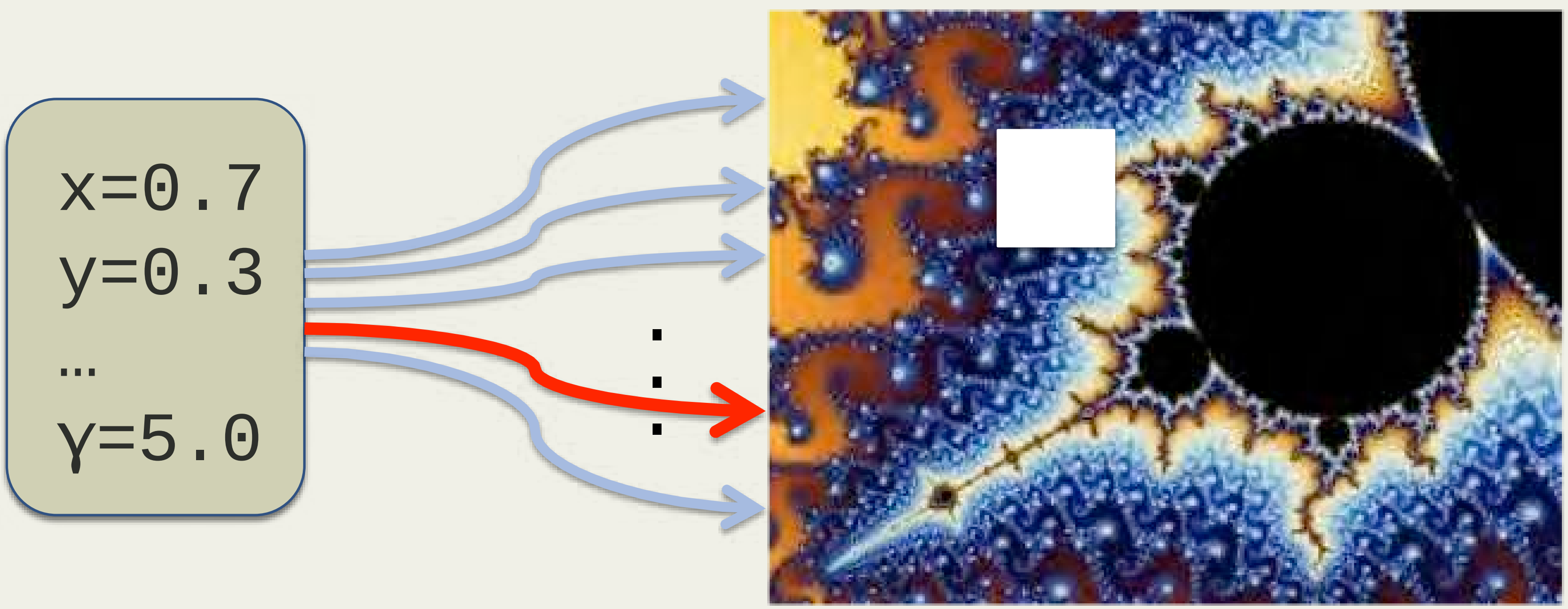
- Want to assert **parallel correctness**.



Background: Deterministic Specs

```
// Parallel fractal render.
mandelbrot(params, img);
```

- Want to assert **parallel correctness**.



Background: Deterministic Specs

```
deterministic
assume (params == params') {
  // Parallel fractal render.
  mandelbrot(params, img);
} assert (img == img');
```

- Specifies that any two runs on the same input parameters must yield the same output image.

Background: Deterministic Specs

```
deterministic
assume (params == params') {
  // Parallel fractal render.
  mandelbrot(params, img);
} assert (img == img');
```

```
s0:  … params   …

s0': … params'  …
```

Background: Deterministic Specs

```
deterministic
assume (params == params') {
  // Parallel fractal render.
  mandelbrot(params, img);
} assert (img == img');
```

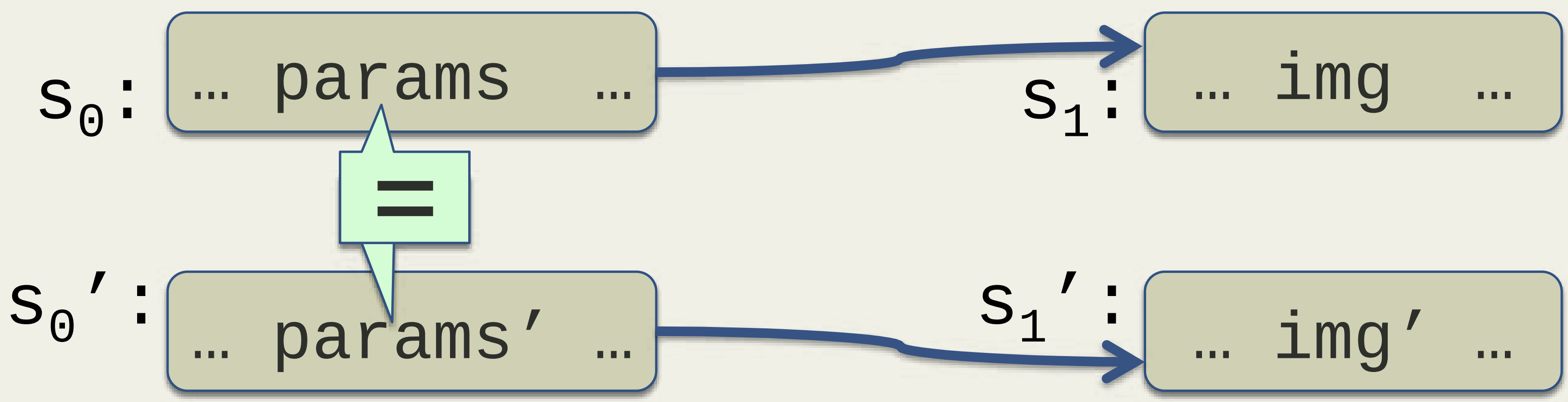
s_0: … params …

=

s_0': … params' …

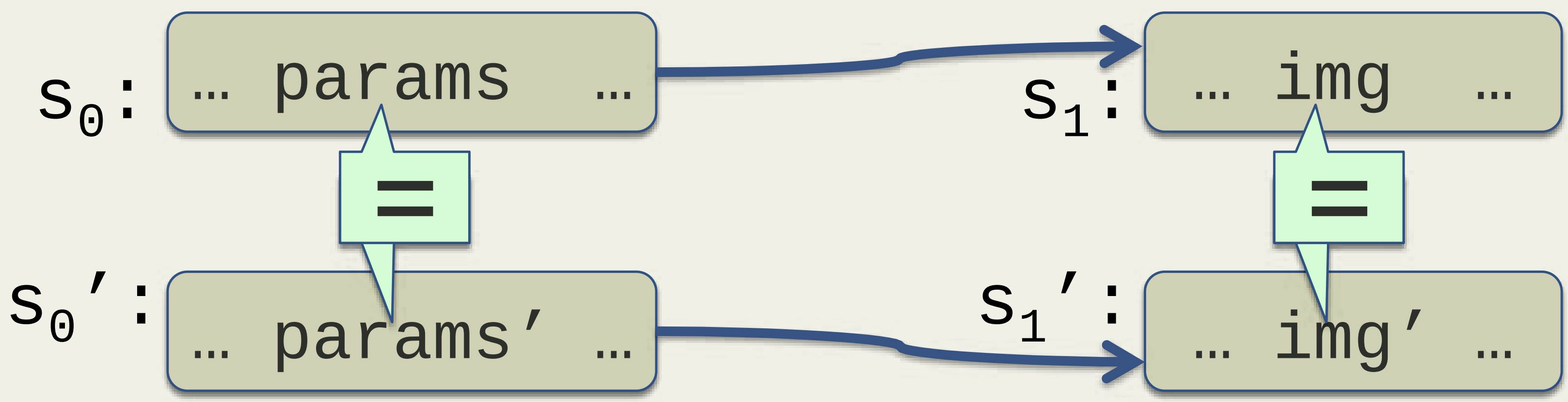
Background: Deterministic Specs

```
deterministic
assume (params == params’) {
  // Parallel fractal render.
  mandelbrot(params, img);
} assert (img == img’);
```



Background: Deterministic Specs

```
deterministic
assume (params == params') {
  // Parallel fractal render.
  mandelbrot(params, img);
} assert (img == img');
```



Background: Deterministic Specs

```
deterministic assume Pre(s₀,s₀′) {
    P
} assert Post(s₁,s₁′)
```

- **Formally, specifies**:

$$\forall\, s_0 \xrightarrow{P} s_1,\ s_0' \xrightarrow{P} s_1' :$$
$$\mathrm{Pre}(s_0,\ s_0') \implies \mathrm{Post}(s_1,\ s_1')$$

Background: Deterministic Specs

```
deterministic
assume (params == params') {
  // Parallel fractal render.
  mandelbrot(params, img);
} assert (img == img');
```

“Bridge” predicate

Background: Deterministic Specs

```
deterministic
assume (params == params’) {
  // Parallel fractal render.
  mandelbrot(params, img);
} assert (img == img’);
```

“Bridge” assertion

Background: Deterministic Specs

```
// Parallel fractal render.
mandelbrot(params, img);
```

- Much simpler than functional correctness:

$$
\begin{aligned}
&\forall_{0 \le x < width}.\ \forall_{0 \le y < height}. \\
&\quad \left(\left| f_{iter}^{\,maxiter}(0) \right| < 2 \ \wedge\ img[x][y] = 0 \right) \\
&\quad \vee\ \exists_{1 \le i < maxiter}.\ \left| f_{iter}^{\,i}(0) \right| \ge 2 \ \wedge\ \forall_{1 \le j < i}. \left| f_{iter}^{\,j}(0) \right| < 2 \\
&\qquad\qquad \wedge\ img[x][y] = \mathrm{HSB}\left((i/maxiter)^{\gamma}, 1, 1 \right)
\end{aligned}
$$

$$
\text{where}\quad f_{iter}(c) \ = \ c^2 + \left(xcenter + (xoff + x)/res \right) + i\left(ycenter + (yoff - y)/res \right)
$$

Background: Deterministic Specs

```
set t = new RedBlackTreeSet();
...
deterministic
assume (t.equals(t’)) {
   t.add(3)  ||  t.add(5);
} assert (t.equals(t’));
```

- Resulting sets are *semantically* equal.

Background: Deterministic Specs

```
double A[][], b[], x[];
...
deterministic
assume (A == A’ and b == b’) {
   // Solve A*x = b in parallel
   lufact_solve(A, b, x);
} assert (|x – x’| < ε);
```

Background: Deterministic Specs

```
deterministic
assume (data == data') {
  // Parallel branch-and-bound
  Tree t = min_phylo_tree(data);
} assert (t.cost == t'.cost());
```

- Determinism is user-specified.

Background: Deterministic Specs

- Can effectively test deterministic specs.
  - Added assertions to 13 benchmarks.
  - Ran CalFuzzer to test if concurrency issues (data races, atomicity violations, etc.) could lead to violations of deterministic spec.
- Specification inference would help automate the above testing.
- Also aid program understanding.

Outline

- Motivation and Overview
- Background: Deterministic Specs
- **Specification Inference Problem**
- Inferring Deterministic Specifications
- Experimental Evaluation
- Related Work
- Conclusions

Specification Inference Problem

```
// Parallel branch-and-bound
(tree, cost) =
     min_phylo_tree(N, data);
```

Specification Inference Problem

```
deterministic assume (???) {
  // Parallel branch-and-bound
  (tree, cost) =
       min_phylo_tree(N, data);
} assert (???)
```

Specification Inference Problem

```
deterministic assume (???) {
  // Parallel branch-and-bound
  (tree, cost) =
       min_phylo_tree(N, data);
} assert (???)
```

- **Observation:** Deterministic pre- and postcondition have simple structure.
  - Conjunction of equality bridge predicates:

$N = N'$ $\quad$ $tree.\mathrm{equals}(tree')$ $\quad$ $|cost - cost'| < \varepsilon$

Specification Inference Problem

```
deterministic assume (???) {
  // Parallel branch-and-bound
  (tree, cost) =
       min_phylo_tree(N, data);
} assert (???)
```

- Four possible deterministic preconditions:

$$true \qquad data = data'$$

$$N = N' \qquad data = data' \land N = N'$$

Specification Inference Problem

```
deterministic assume (???) {
  // Parallel branch-and-bound
  (tree, cost) =
       min_phylo_tree(N, data);
} assert (???)
```

- Six possible deterministic postconditions:

$true$ $\quad$ $|cost - cost'| < \varepsilon$ $\quad$ $cost = cost'$

$tree.\mathrm{equals}(tree')$ $\quad$ $tree.\mathrm{equals}(tree') \wedge |cost - cost'| < \varepsilon$ $\quad$ $tree.\mathrm{equals}(tree') \wedge cost = cost'$

Specification Inference Problem

```
deterministic assume (???) {
  // Parallel branch-and-bound
  (tree, cost) =
       min_phylo_tree(N, data);
} assert (???)
```

- Six … itions:

Which specification should we choose?

$true$ … st'

$tree.\mathrm{equals}(tree')$

… $\mathrm{equals}(tree')$ $\wedge \left| cost - cost' \right| < \varepsilon$

… $\mathrm{quals}(tree')$ $\wedge\, cost = cost'$

Specification Inference Problem

- Principles for specification inference:
  - **Must be consistent with observed runs.**

```
N=10, data=D1   →   cost=3.7, tree=T1
N=10, data=D1   →   cost=3.7, tree=T2
```

$$data = data' \cancel{\xrightarrow{\text{min\_phylo\_tree}}} tree.\text{equals}(tree')$$

Specification Inference Problem

- Principles for specification inference:
  - Must be consistent with observed runs.
  - **Postcondition as strong as possible.**

$$data = data' \xrightarrow{\text{min\_phylo\_tree}} \left| cost - cost' \right| < \varepsilon$$

$$\cancel{data = data' \xrightarrow{\text{min\_phylo\_tree}} true}$$

Specification Inference Problem

- Principles for specification inference:
  - Must be consistent with observed runs.
  - Postcondition as strong as possible.
  - **Precondition as weak as possible,** ***for post***

$$data = data' \xrightarrow{\text{min\_phylo\_tree}} \left| cost - cost' \right| < \varepsilon$$

$$\begin{array}{l} data = data' \\ \wedge\ N = N' \end{array} \xrightarrow{\text{min\_phylo\_tree}} \left| cost - cost' \right| < \varepsilon$$

Specification Inference Problem

- Principles for specification inference:
  - Must be consistent with observed runs.
  - Postcondition as strong as possible.
  - Precondition as weak as possible, *for post*.

How do we compute such a deterministic specification?

Outline

- Motivation and Overview
- Background: Deterministic Specs
- Specification Inference Problem
- **Inferring Deterministic Specifications**
- Experimental Evaluation
- Related Work
- Conclusions

Inferring Deterministic Specs

??? $\xrightarrow{\text{min\_phylo\_tree}}$???

- *M* variables => *Ω(2^{M})* specifications
  - Exhaustive search infeasible.
- Two-step algorithm:
  - Compute strongest consistent postcondition.
  - Compute weakest consistent precondition for the inferred postcondition.

Inferring Deterministic Specs

$$??? \xrightarrow{\text{min\_phylo\_tree}} ???$$

`N:`

$true$

$N = N'$

`data:`

$true$

$data = data'$

`tree:`

$true$

$tree.\text{equals}(tree')$

`cost:`

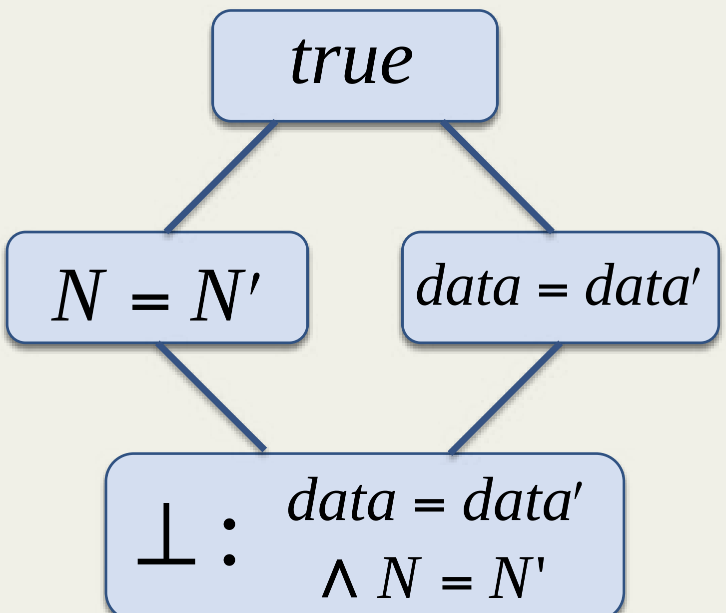
$true$

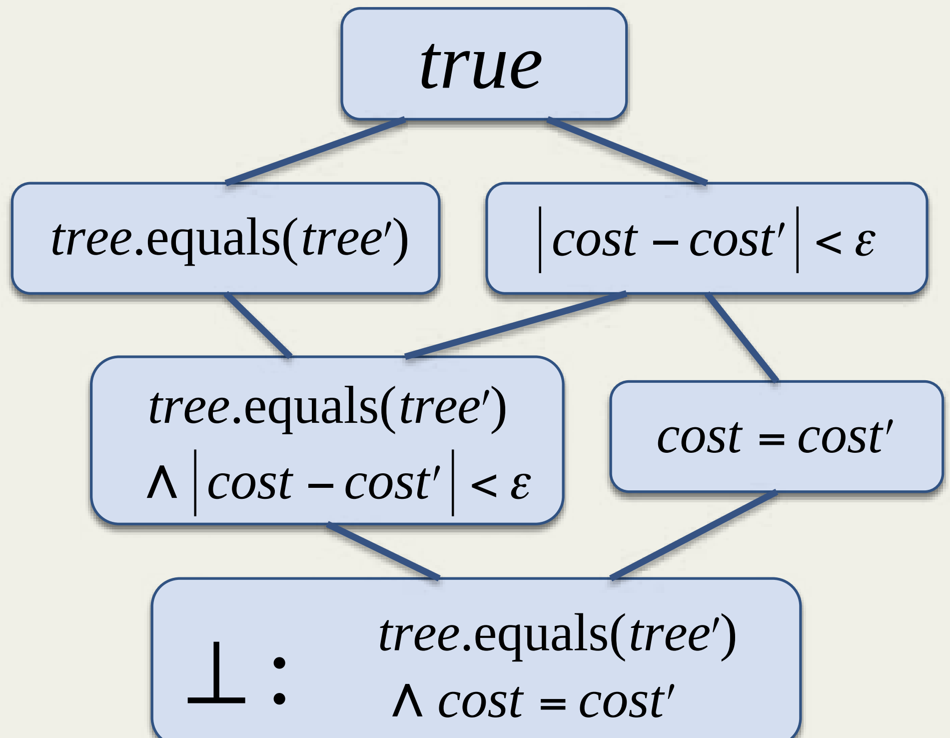
$|cost - cost'| < \varepsilon$

$cost = cost'$

Inferring Deterministic Specs

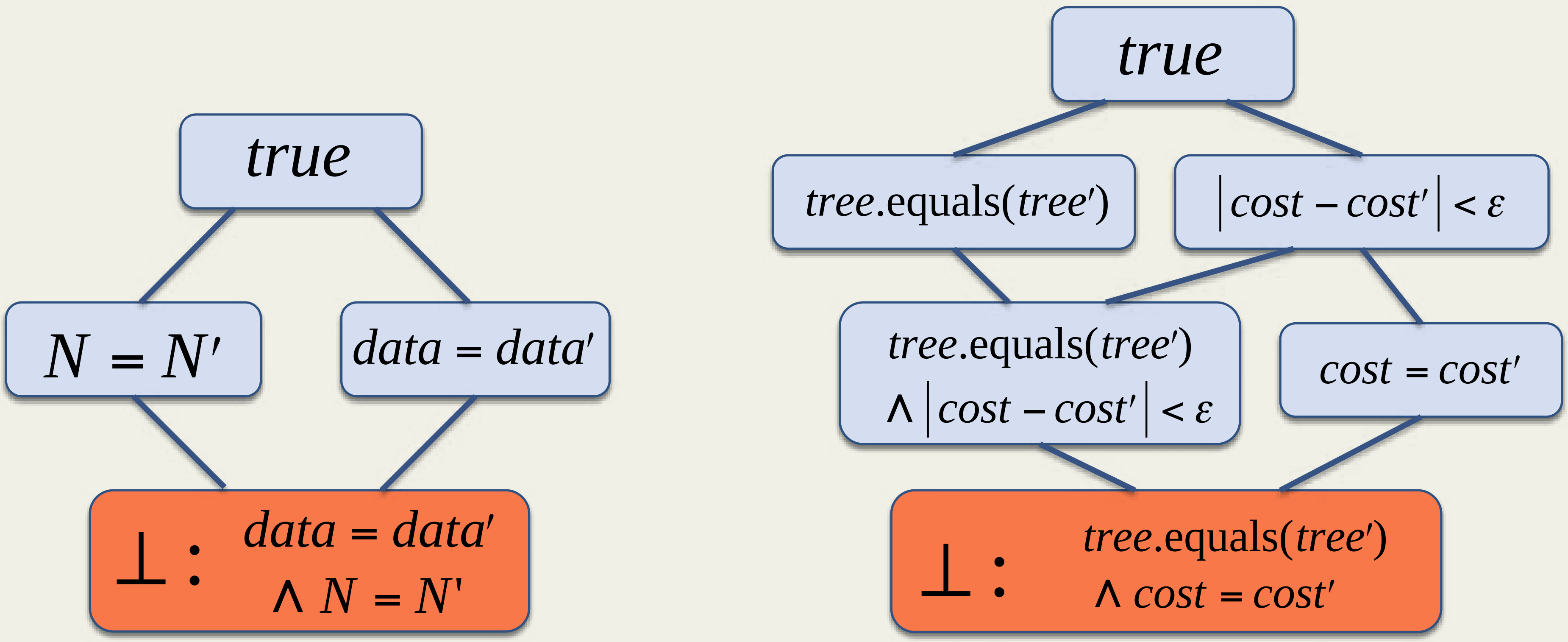
$$??? \xrightarrow{\text{min\_phylo\_tree}} ???$$





Inferring Strongest Post

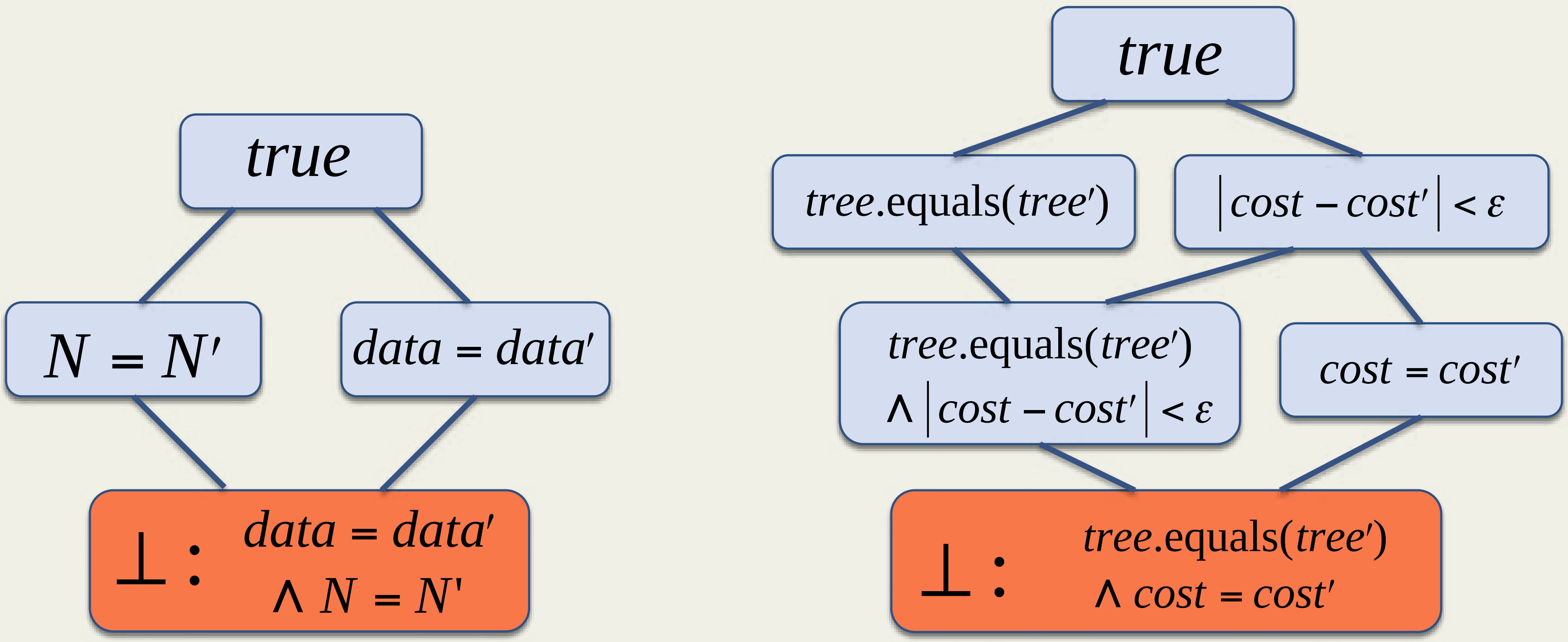
- For every pair of observed executions:
  - If they satisfy precondition ⊥, ensure that the postcondition holds.



Inferring Strongest Post I

`N=10, data=D1` → `cost=3.7, tree=T1`

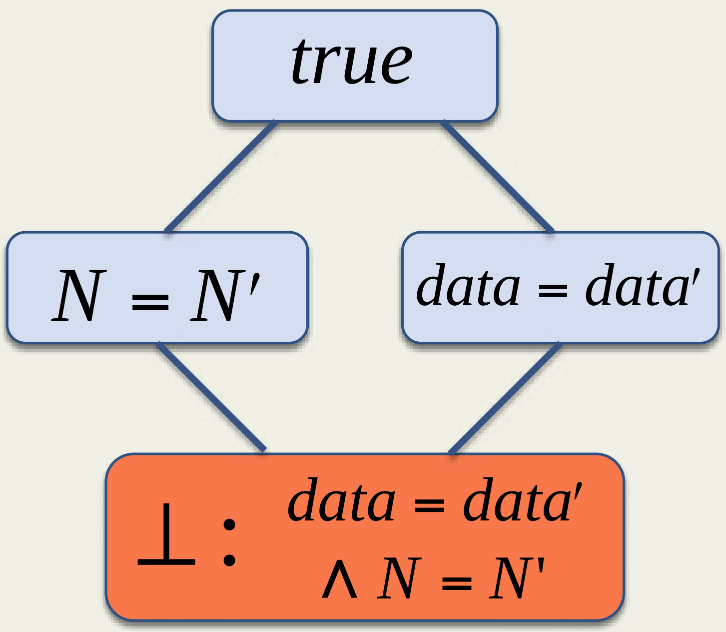
`N=10, data=D1` → `cost=3.7, tree=T2`

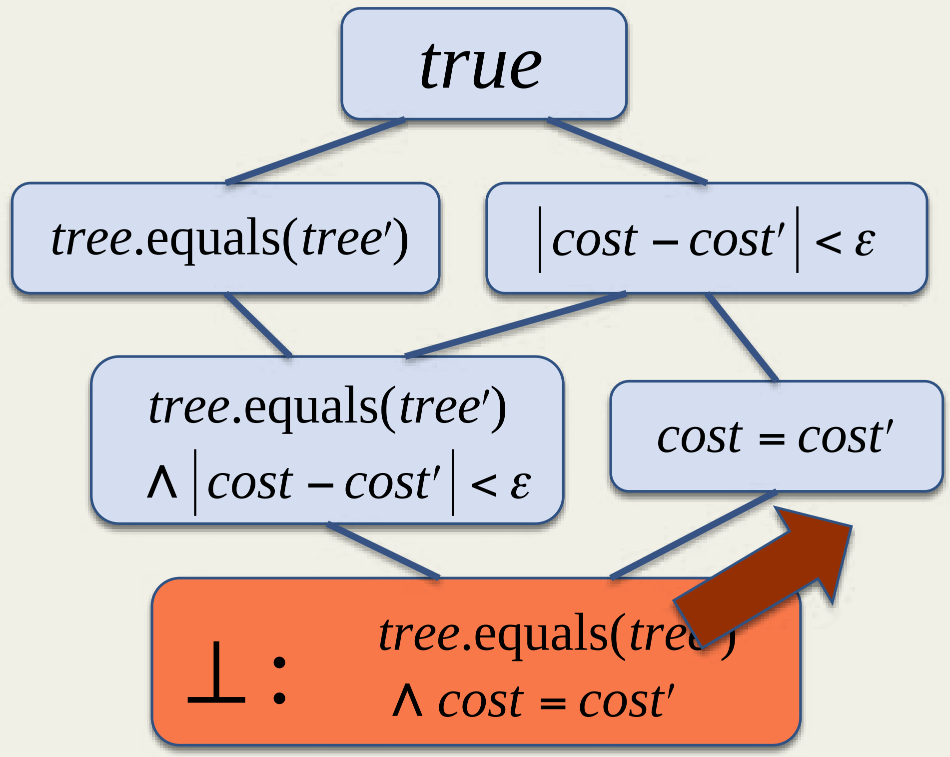


Inferring Strongest Post I

`N=10, data=D1` → `cost=3.7, tree=T1`

`N=10, data=D1` → `cost=3.7, tree=T2`

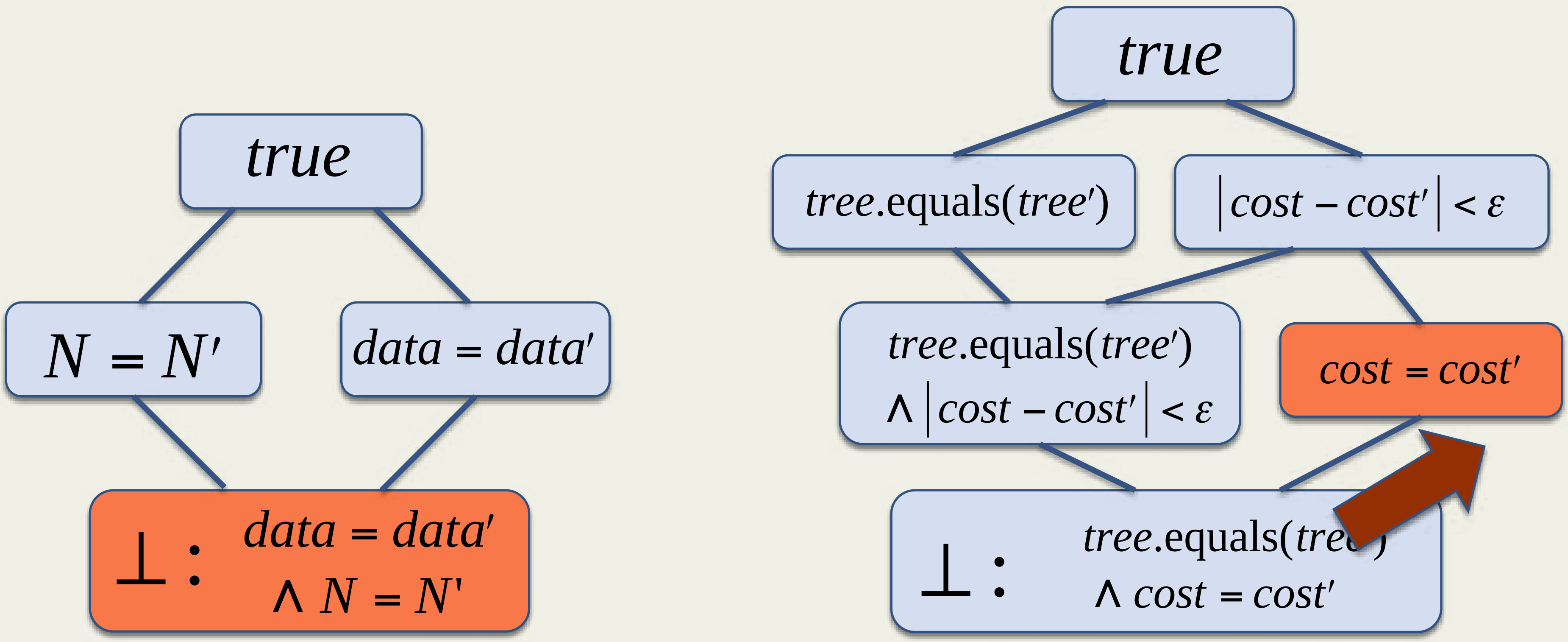




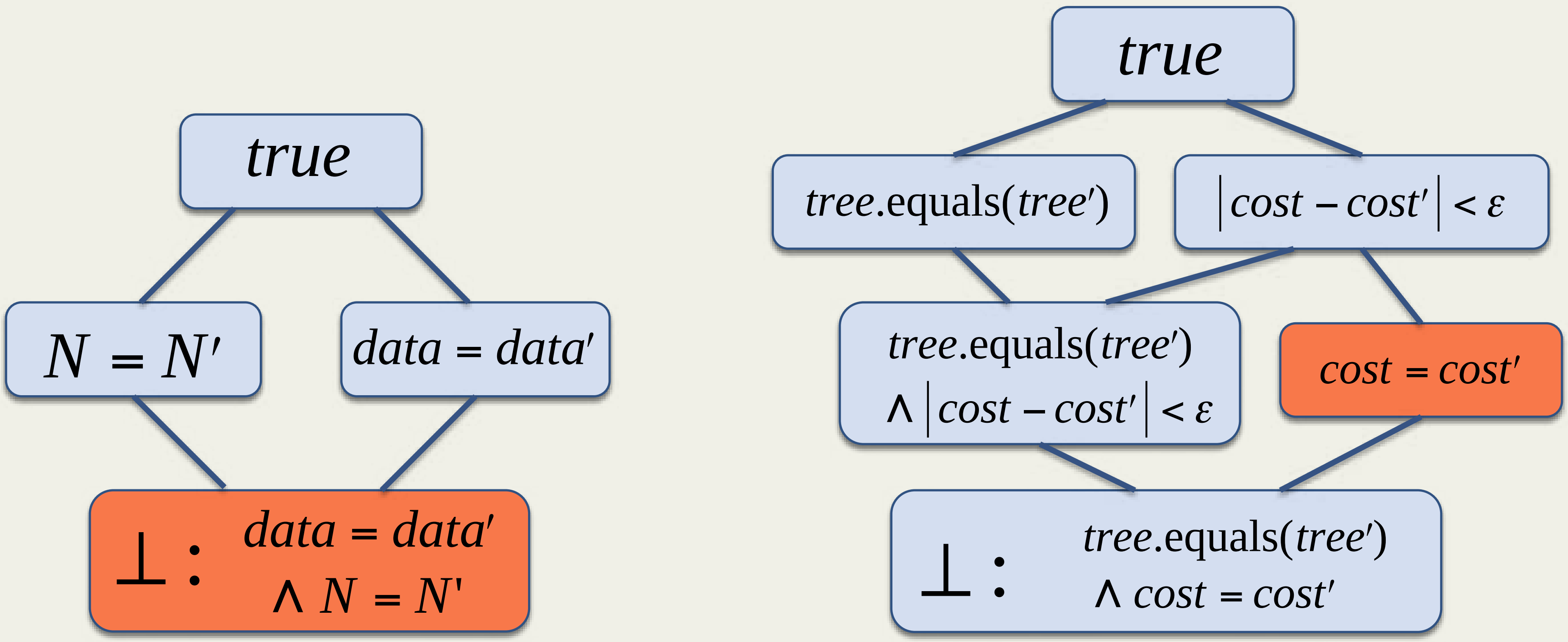
Inferring Strongest Post I

`N=10, data=D1` → `cost=3.7, tree=T1`

`N=10, data=D1` → `cost=3.7, tree=T2`



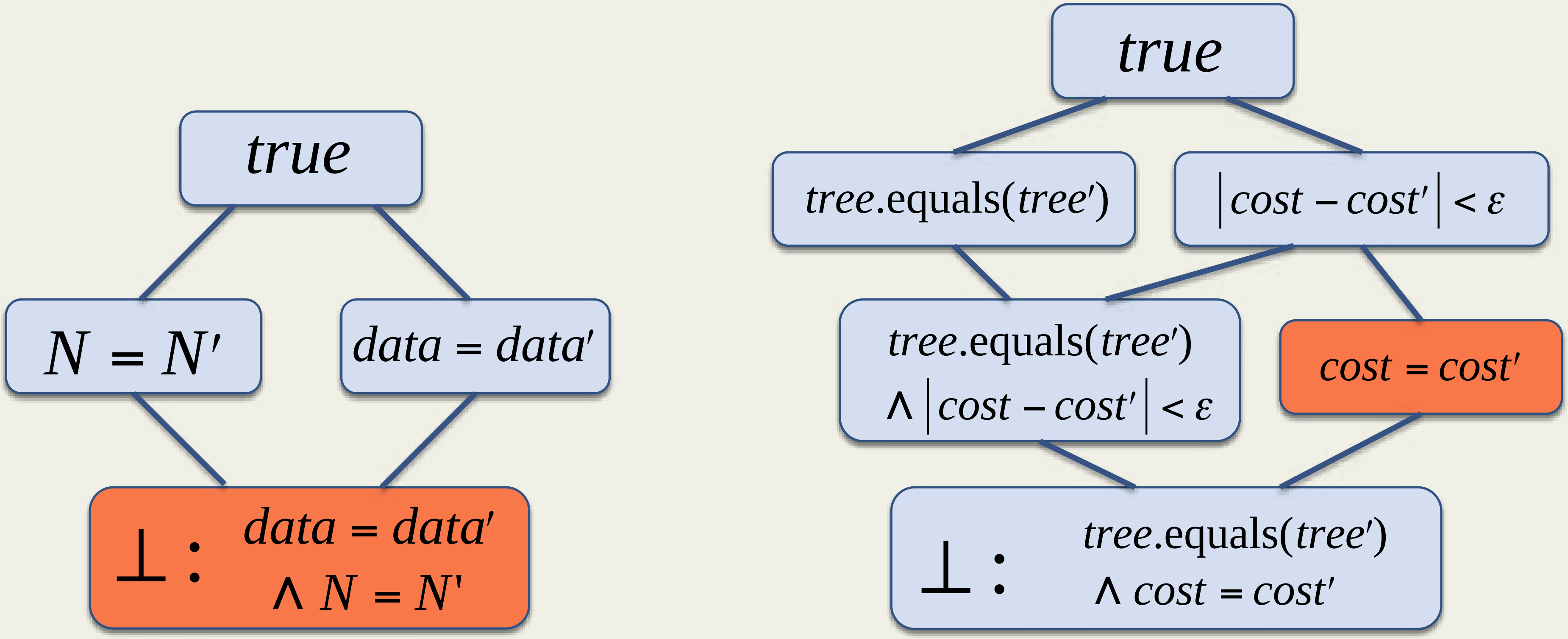
Inferring Strongest Post I



Inferring Strongest Post II

`N=10, data=D1` → `cost=3.7, tree=T1`

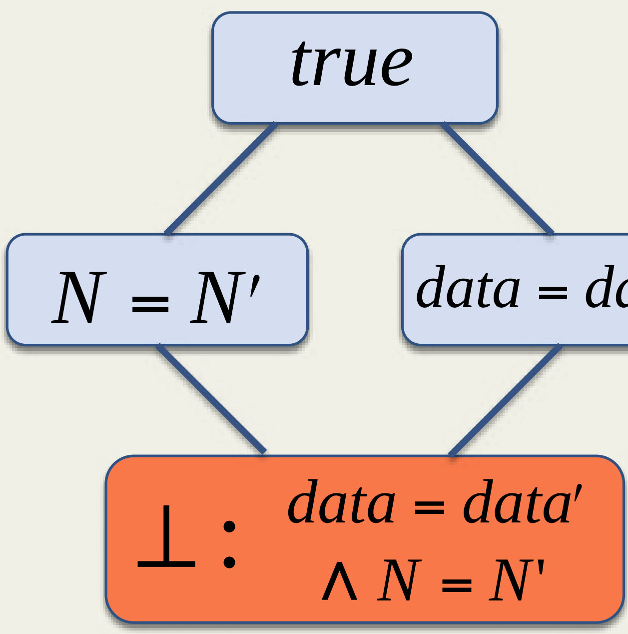
`N=10, data=D2` → `cost=2.4, tree=T3`

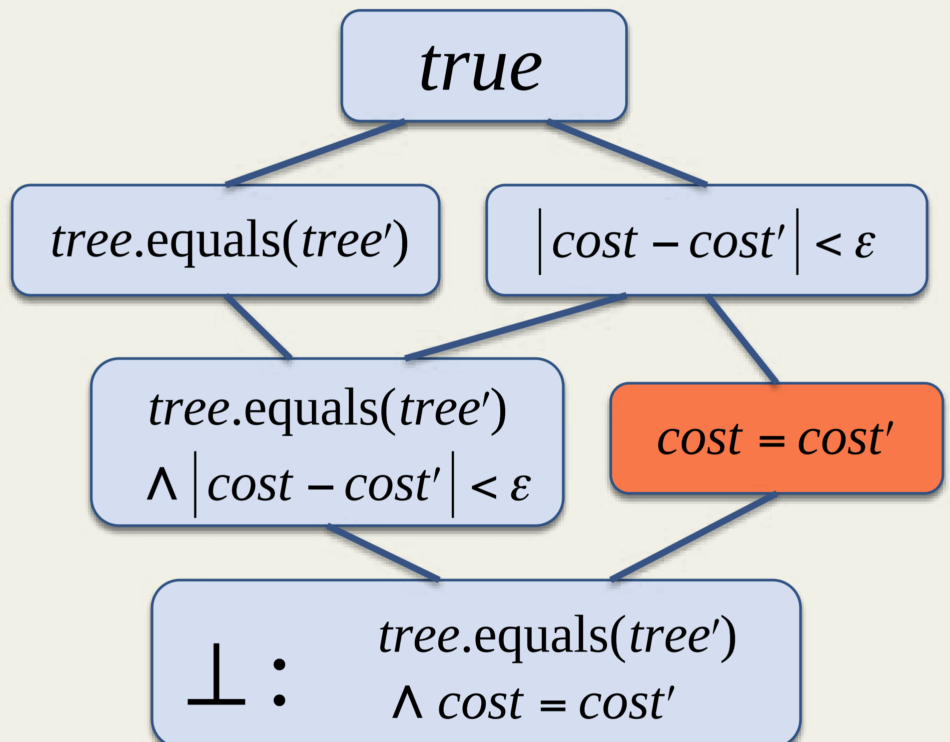


Inferring Strongest Post II

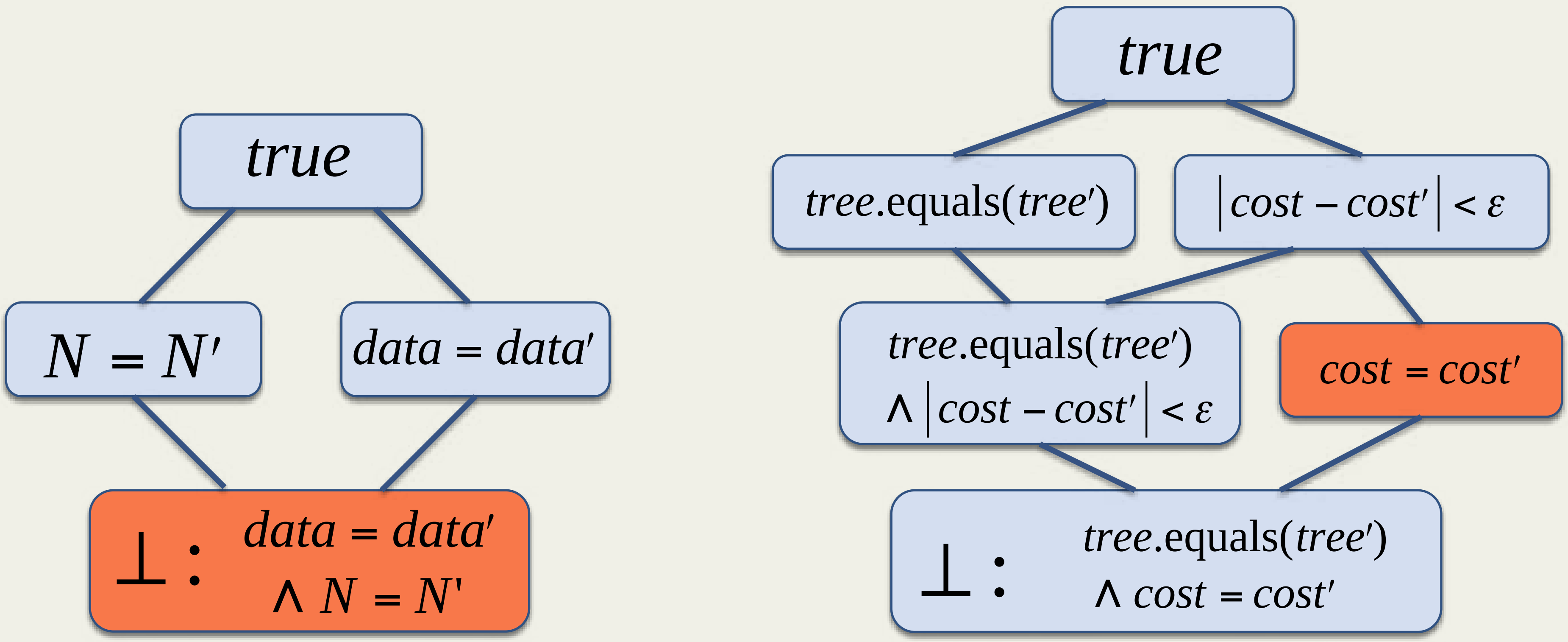
`N=10, data=D1` → `cost=3.7, tree=T1`

`N=10, data=D2` → `cost=2.4, tree=T3`





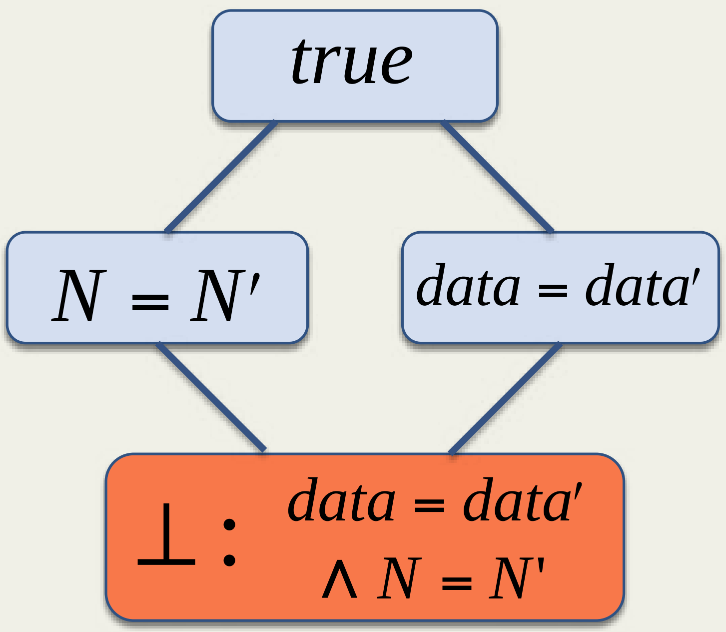
Inferring Strongest Post II

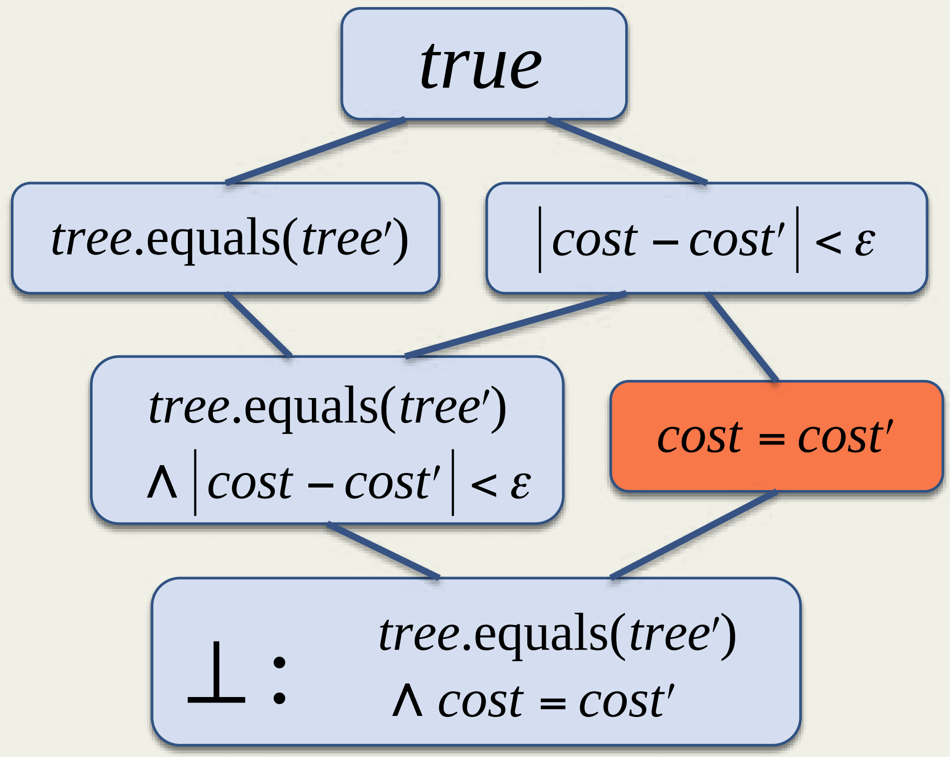


Inferring Strongest Post III

`N=10, data=D2` → `cost=2.4, tree=T3`

`N=10, data=D2` → `cost=2.39, tree=T4`

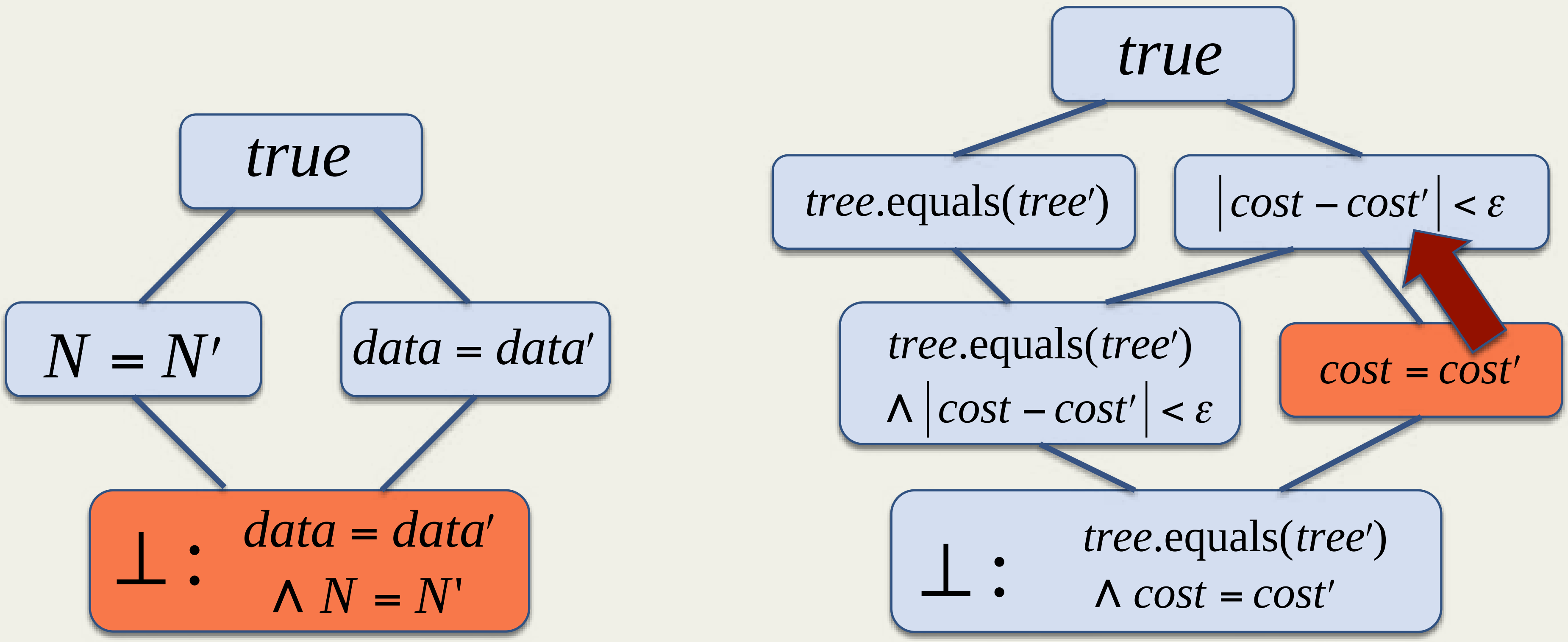




Inferring Strongest Post III

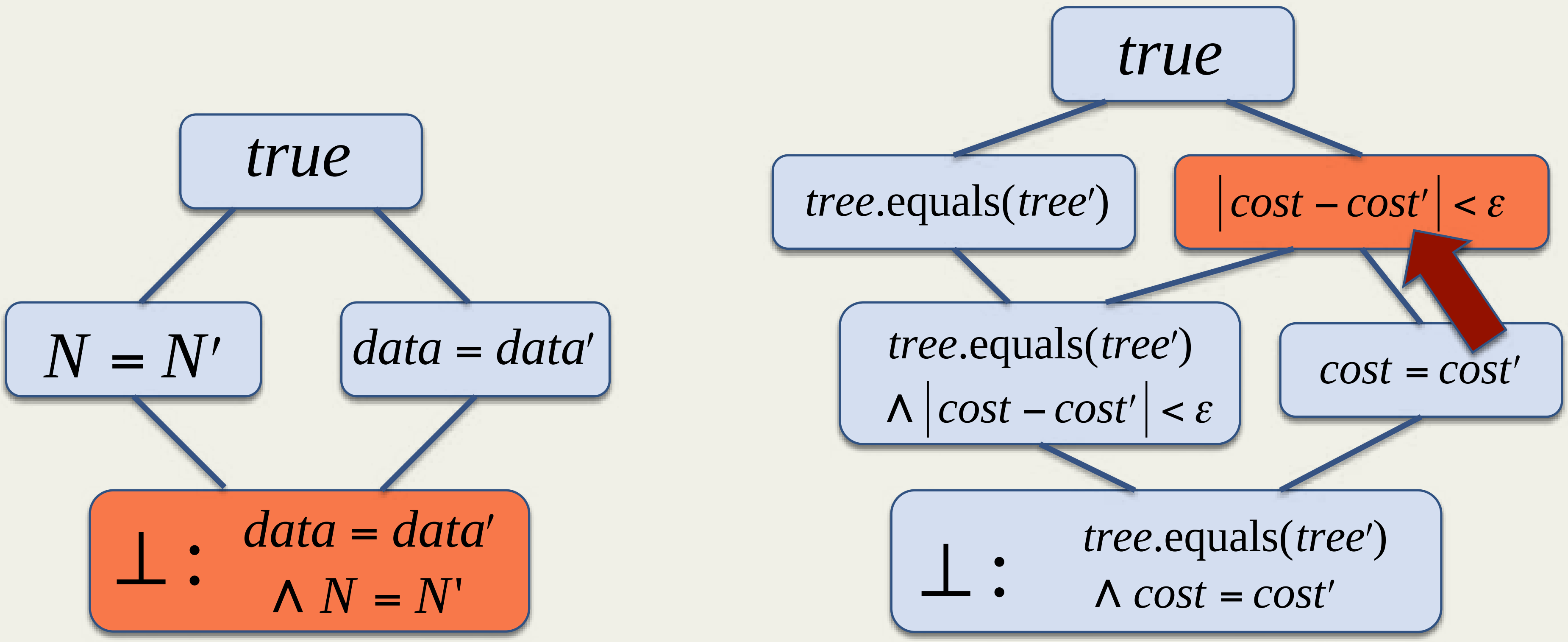
`N=10, data=D2` → `cost=2.4, tree=T3`

`N=10, data=D2` → `cost=2.39, tree=T4`

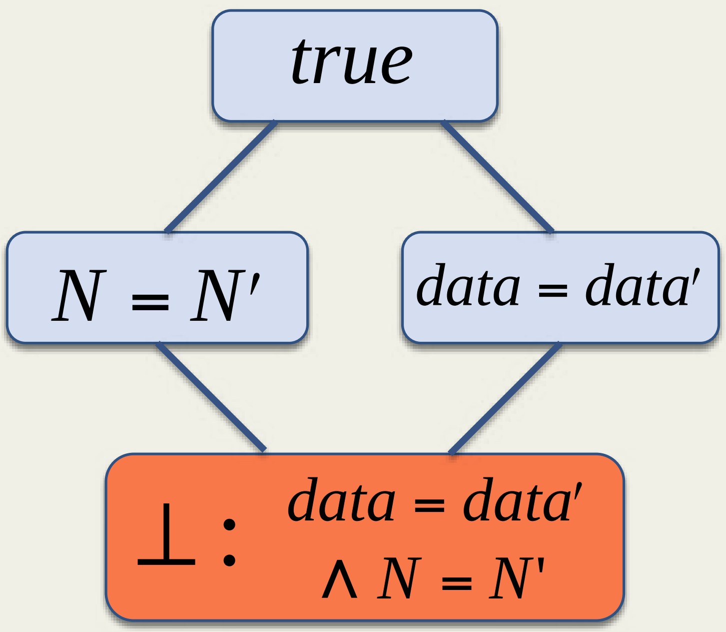


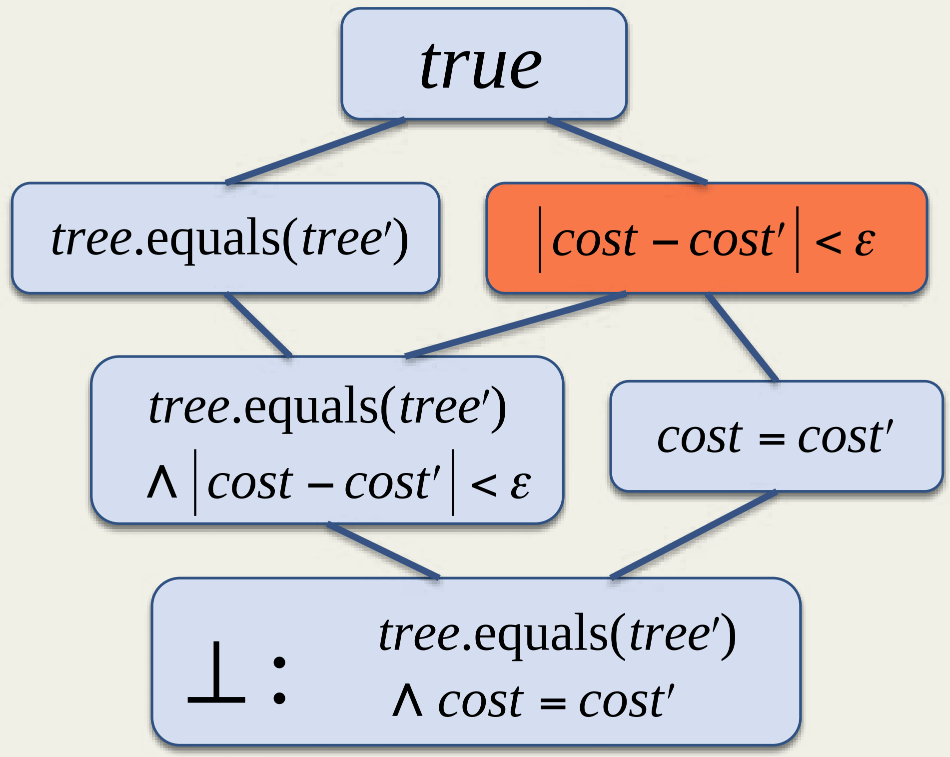
Inferring Strongest Post III

```
N=10, data=D2  →  cost=2.4, tree=T3
N=10, data=D2  →  cost=2.39, tree=T4
```



Inferred Strongest Post

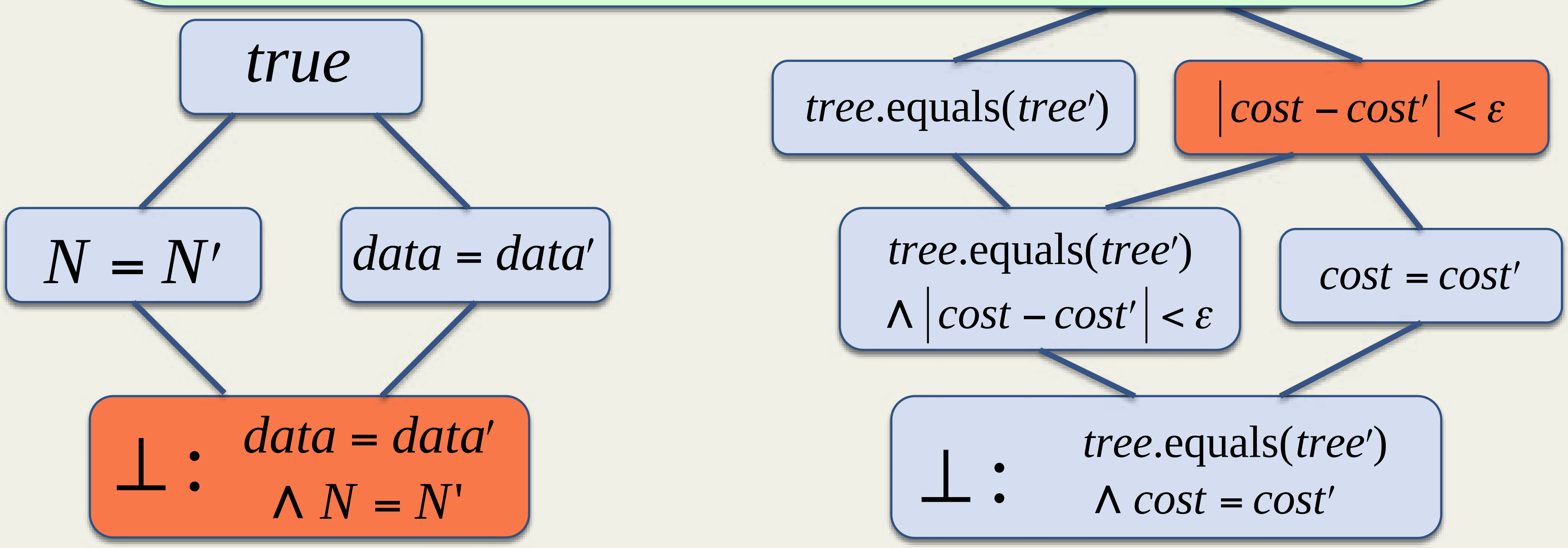




Inferred Strongest Post

Proposition 1. The inferred postcondition $Post_R$ is the unique strongest postcondition given the observed runs.

I.e., if $\varphi_{pre} \xrightarrow{P,R} \varphi_{post}$ *then* $Post_R \Rightarrow \varphi_{post}$.



Inferred Strongest Post

Proposition 1. The inferred postcondition $Post_R$ is the unique strongest postcondition given the observed runs.

I.e., if $\varphi_{pre} \xrightarrow{P,R} \varphi_{post}$ *then* $Post_R \Rightarrow \varphi_{post}$.

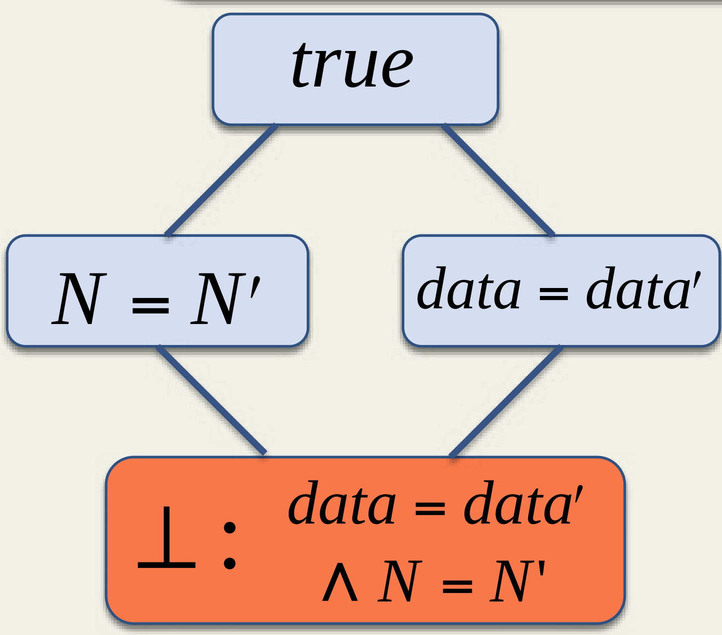
Corollary 2. Let the inferred specification be Pre_R ➔ $Post_R$. Then, $Post_R$ is the strongest postcondition of Pre_R .

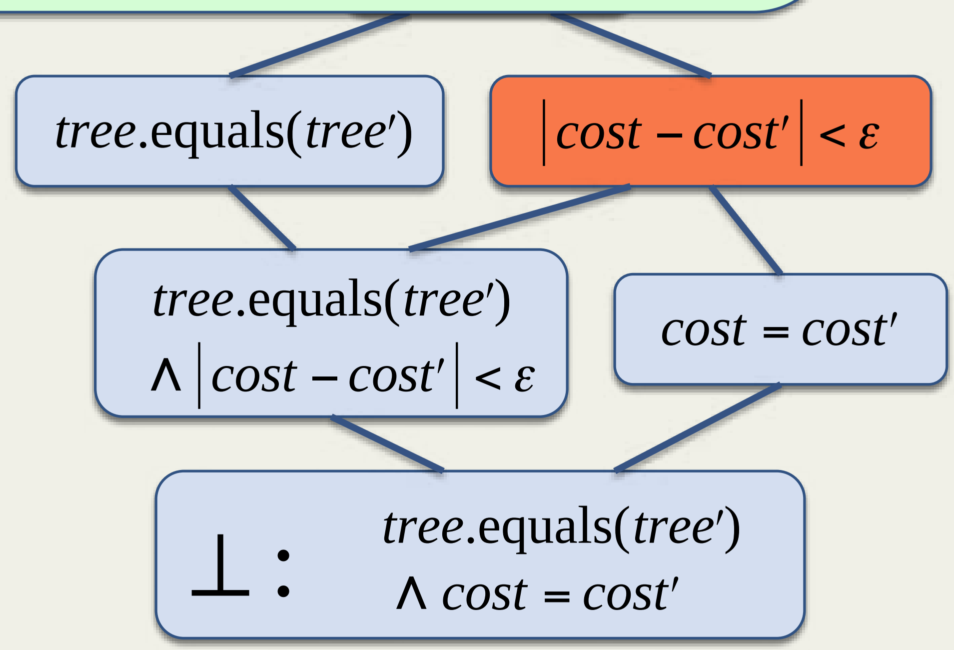
I.e., if $Pre_R \xrightarrow{P,R} \varphi_{post}$ *then* $Post_R \Rightarrow \varphi_{post}$.

Inferred Strongest Post

Proposition 3. The inferred postcondition $Post_R$ is at least as strong as the true unique strongest postcondition $SP_P(\bot)$.

I.e., if $\varphi_{pre} \xrightarrow{P} \varphi_{post}$ *then* $Post_R \Rightarrow \varphi_{post}$.





Inferred Strongest Post

Proposition 3. The inferred postcondition $Post_R$ is at least as strong as the true unique strongest postcondition $SP_P(\bot)$.

I.e., if $\varphi_{pre} \xrightarrow{P} \varphi_{post}$ *then* $Post_R \Rightarrow \varphi_{post}$.

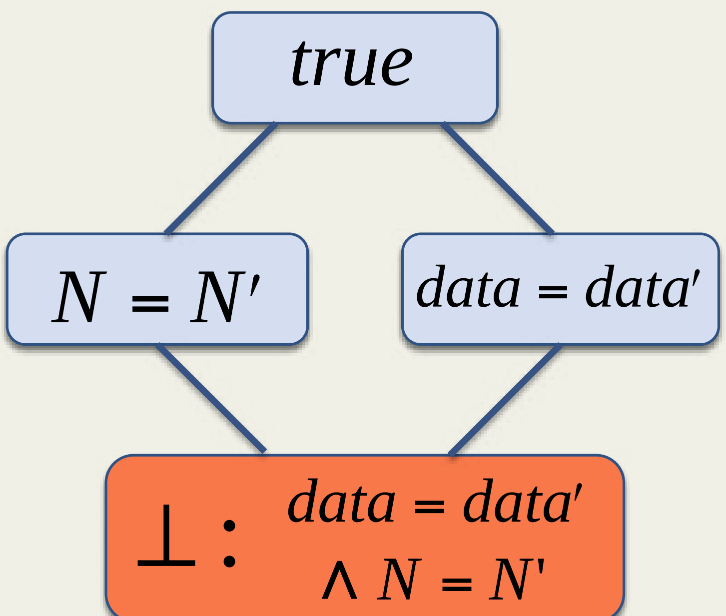
Proposition 4. More observed executions lead to a weaker inferred postcondition.

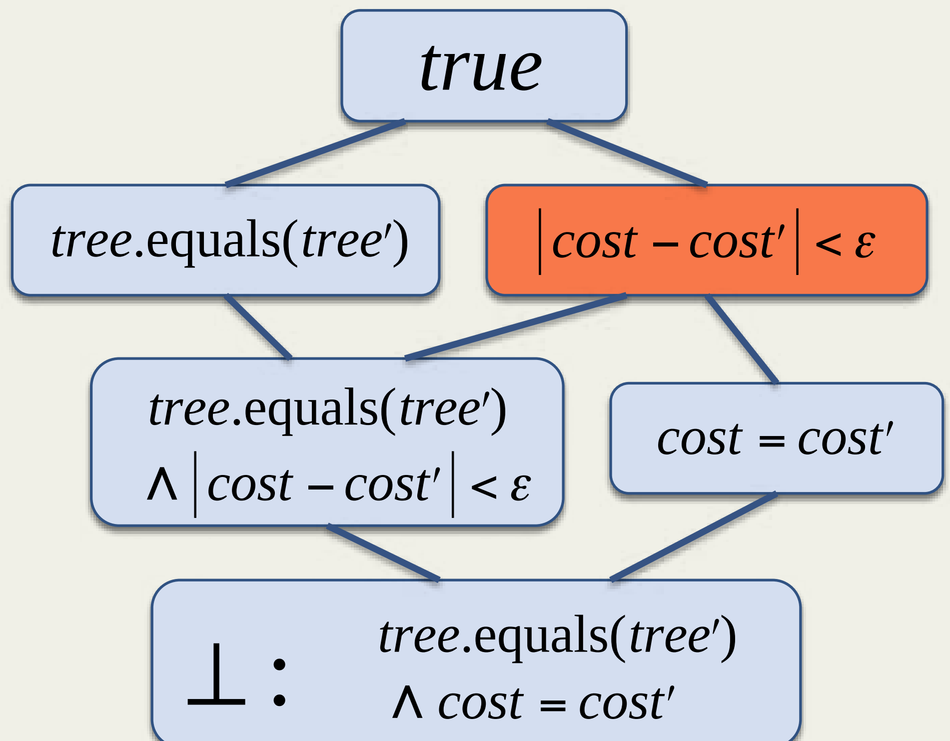
That is, if $R_1 \subseteq R_2 \subseteq R_3 \subseteq \cdots$ *then*

$$Post_{R_1} \Rightarrow Post_{R_2} \Rightarrow \cdots \Rightarrow SP_P(\bot).$$

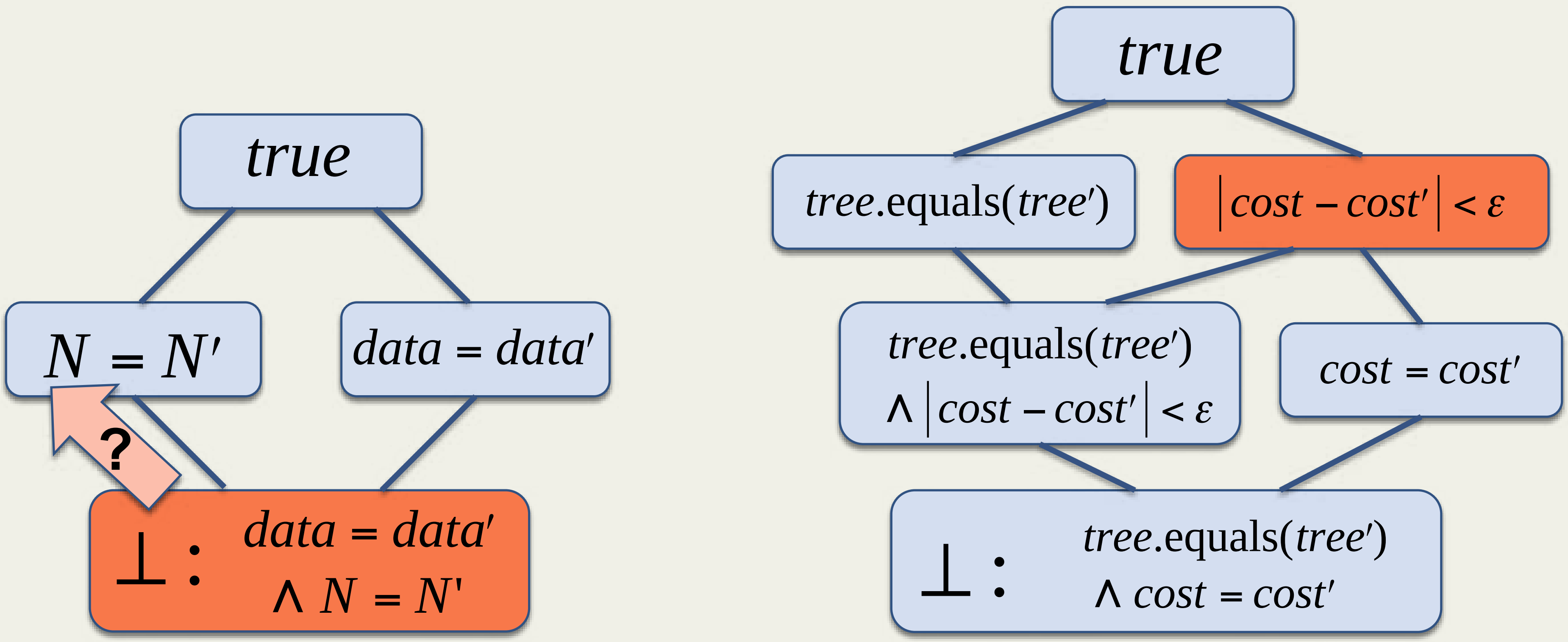
Inferring Weakest Precondition

- Repeatedly weaken the precondition
  - As long as it still ensures the postcondition on every pair of observed executions.





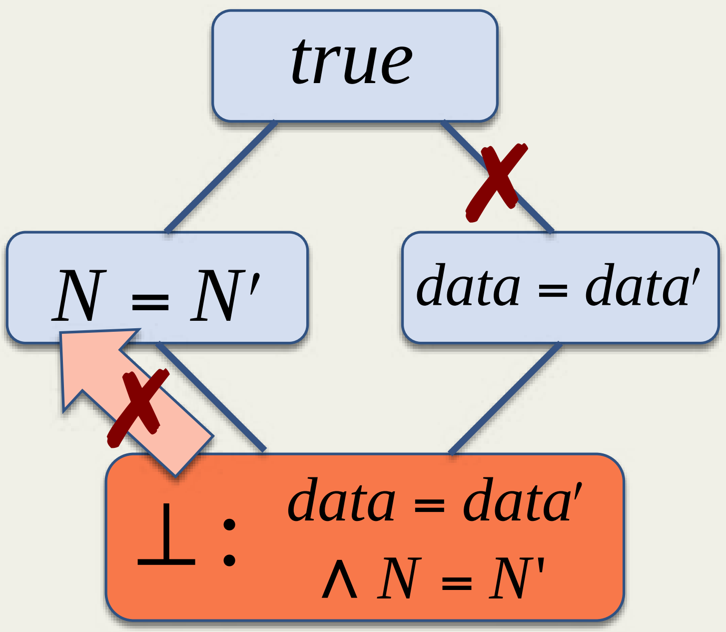
Inferring Weakest Pre I

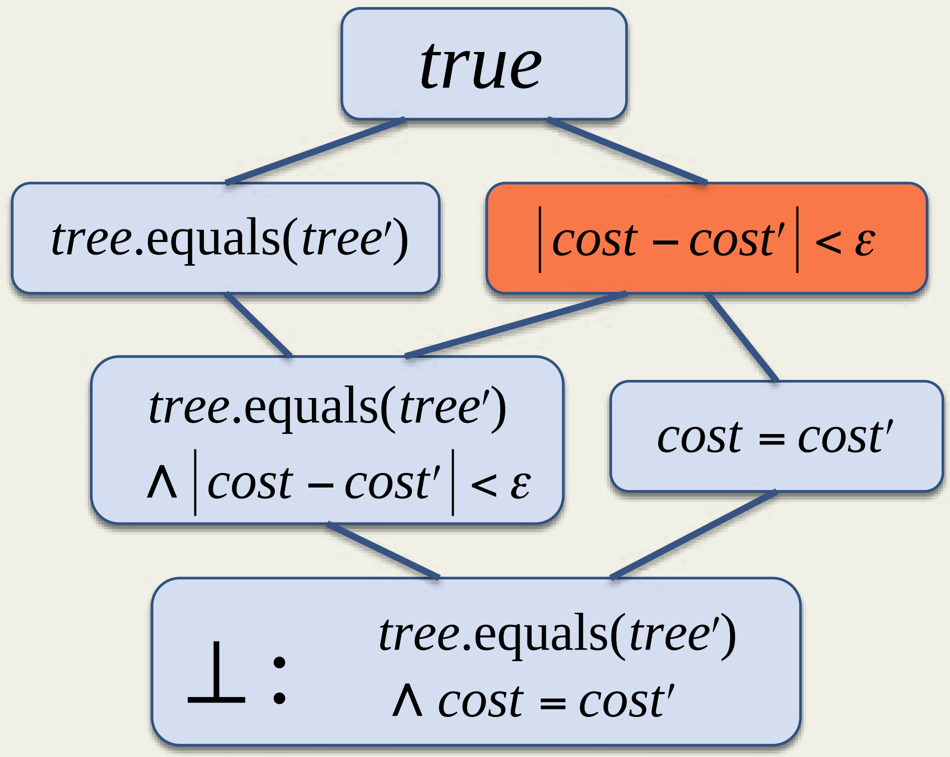


Inferring Weakest Pre I

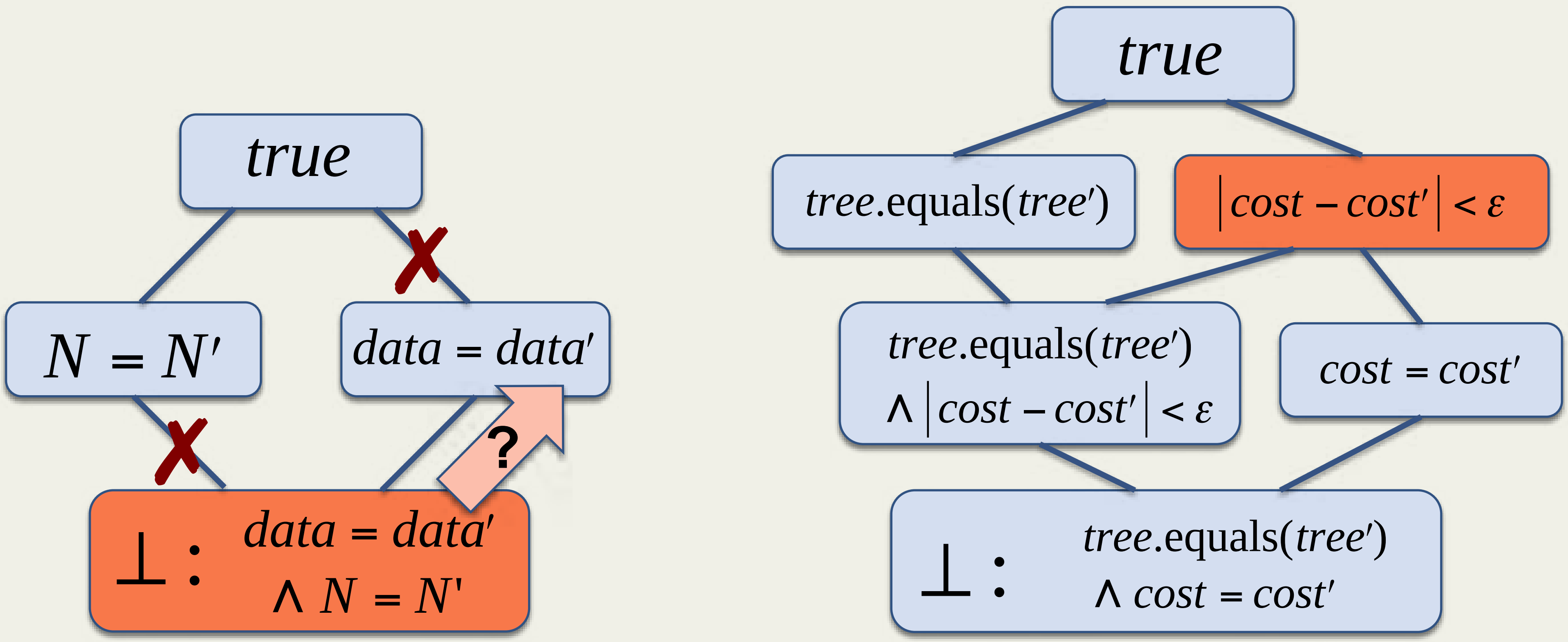
`N=10, data=D1` → `cost=3.7, tree=T1`

`N=10, data=D2` → `cost=2.4, tree=T3`





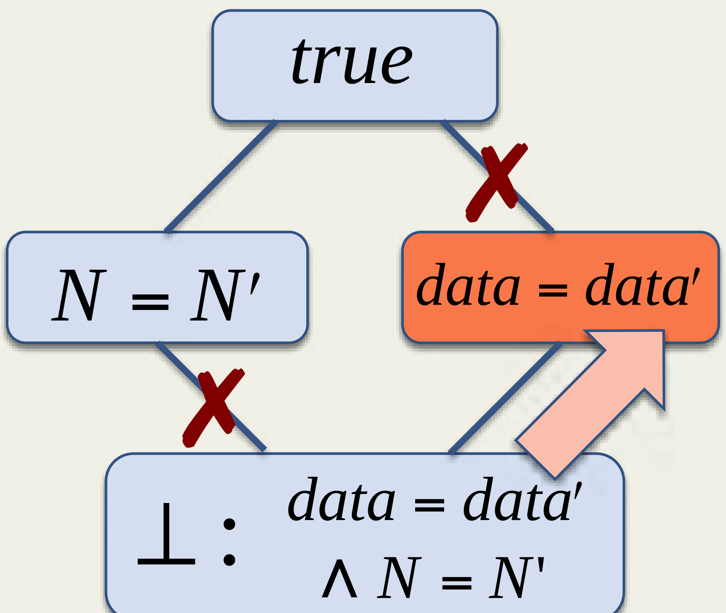
Inferring Weakest Pre II

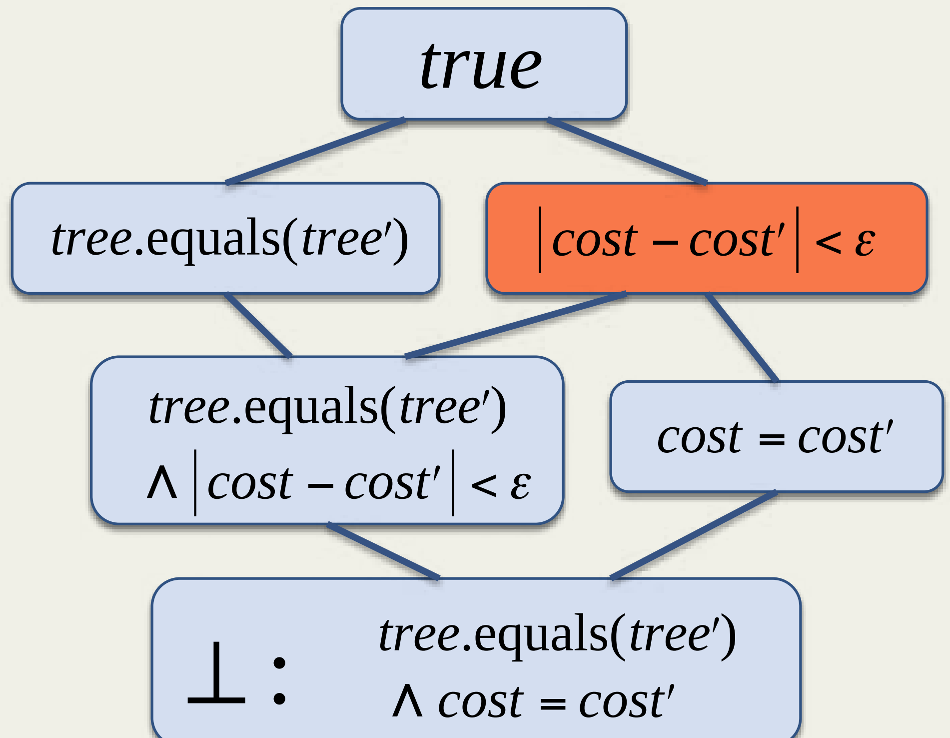


Inferring Weakest Pre II

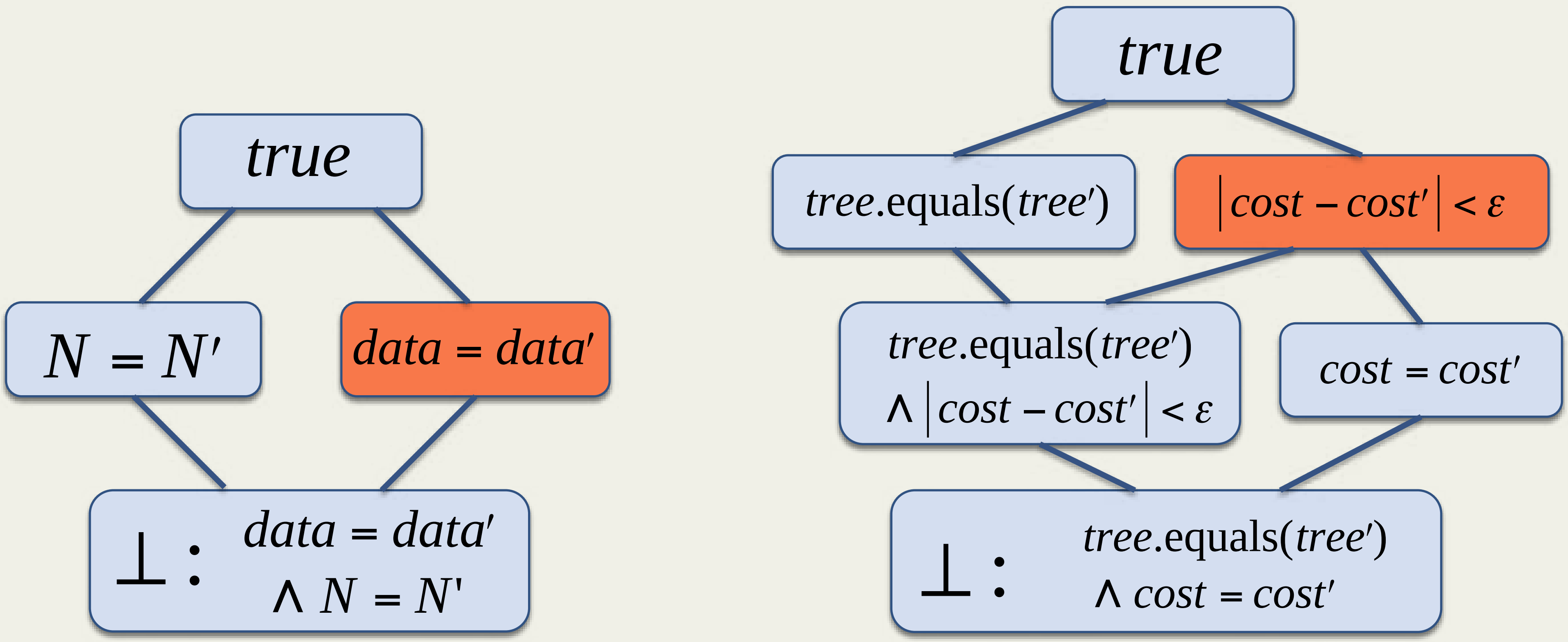
For all observed pairs of runs:

$$data = data' \xrightarrow{\text{min\_phylo\_tree}} \left| cost - cost' \right| < \varepsilon$$





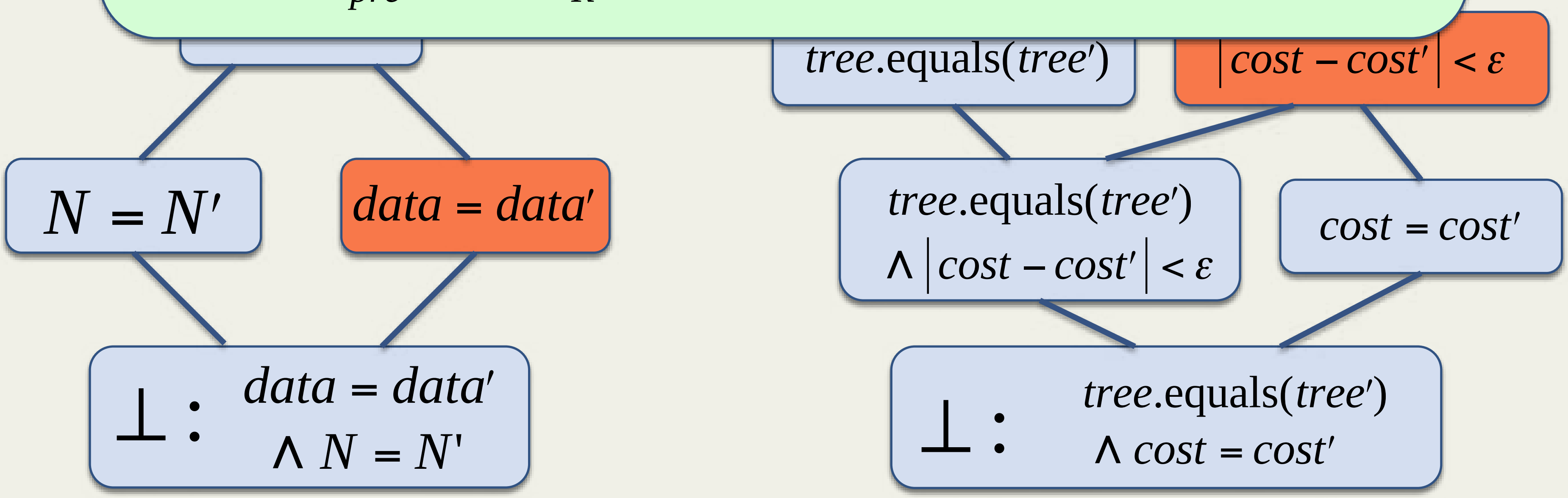
Inferred Weakest Precondition



Inferred Weakest Precondition

Proposition 5. Let the inferred specification be Pre_R ➔ $Post_R$. Then, Pre_R is a weakest precondition of $Post_R$ for the observed runs.

I.e., if $\varphi_{pre} \xrightarrow{P,R} Post_R$ and $\varphi_{pre} \Rightarrow Pre_R$, then $\varphi_{pre} = Pre_R$.



Outline

- Motivation and Overview
- Background: Deterministic Specs
- Specification Inference Problem
- Inferring Deterministic Specifications
- **Experimental Evaluation**
- Related Work
- Conclusions

Determinism Inference Experiments

- For previous benchmarks, infer specs for manually-identified deterministic blocks.
  - Benchmarks: 8 from Java Grande Forum (JGF), 4 from Parallel Java (PJ) Library.

Implementation (for Java)

- Record memory graph at open and close of deterministic block.
- Three equality predicates considered:
  - equals(), approximate equality, set equality
  - Compare any chain of fields (up to length 8): e.g., `this.tree.cost`, `Harness.matrix`
- Heuristics to reduce specification size.
  - By removing "uninteresting" conjuncts.

Implementation (for Java)

- Heuristics needed to shrink specifications:
  - **Remove inputs from postcondition:**
    If no run changes *v*, don't include *v=v'*.

```
data=data' => cost=cost' ∧ data=data'
```

Implementation (for Java)

- Heuristics needed to shrink specifications:
  - Remove inputs from postcondition.
  - **Remove constants from pre- and post-:**
    If *v* is equal in every run, don't include *v*=*v'*.

```
data=data' ∧ MAX_MEM=MAX_MEM'
     =>  cost=cost' ∧ done=done'
```

Implementation (for Java)

- Heuristics needed to shrink specifications:
  - Remove inputs from postcondition.
  - Remove constants from pre- and post-.
  - **Remove redundant conditions:**

```
params=params'
        =>  point.equals(point')
              ∧ point.x=point.x'
              ∧ point.y=point.y'
```

Determinism Inference Experiments

- For previous benchmarks, infer specs for manually-identified deterministic blocks.
  - Benchmarks: 8 from Java Grande Forum (JGF), 4 from Parallel Java (PJ) Library.
- Compare to manual specifications.
  - Are inferred specs correct?
  - Capture intended deterministic behavior?
  - Small enough to be useful?

Experimental Results

- Inferred specification vs. manual one:
  - Equivalent for 7/13 benchmarks.

Manual:

`params=params′` $\longrightarrow$ `matrix=matrix′`

Inferred:

`params=params′` $\longrightarrow$ `matrix=matrix′`
′ $\wedge$ *`img.equals(img′)`*

Experimental Results

- Inferred specification vs. manual one:
  - Equivalent for 7/13 benchmarks.
  - Inferred correct while manual wrong for 2/13!

Experimental Results

- Inferred specification vs. manual one:
  - Equivalent for 7/13 benchmarks.
  - Inferred correct while manual wrong for 2/13!
  - 1/13 is correct but stronger than desired.

Manual:

`params=params'` $\longrightarrow$ `|ek[0]-ek[0]'|` $< \varepsilon$

Inferred:

`params=params'` $\wedge$ *`nthreads=nthreads'`*

$\longrightarrow$ `|ek-ek'|` $< \varepsilon$

Experimental Results: JGF

Bench	LoC	Size of Precondition		Size of Postcondition	
		Manual	Inferred	Manual	Inferred
series	800	1	3	1	1
crypt	1.1k	1	3	2	2
moldyn	1.3k	2	14	3	7
raytracer	1.9k	2	3	1	1
monte	3.6k	1	2	1	1

Experimental Results: PJ and tsp

Bench	LoC	Size of Precondition		Size of Postcondition	
		Manual	Inferred	Manual	Inferred
pi3	150	2	3	1	1
keysearch3	200	3	5	1	3
mandelbrot	250	7	11	1	5
phylogeny	4.4k	3	5	2	11
tsp*	700	1	3	1	2

Experimental Results

- Limitations:
  - For 3/13 benchmarks, inferred spec is incorrect because of insufficient test suite.

Manual:

```
graph=graph' ⟶ tour.cost=tour.cost'
```

Inferred:

```
graph=graph' ⟶ tour.equals(tour')
```

Outline

- Motivation and Overview
- Background: Deterministic Specs
- Specification Inference Problem
- Inferring Deterministic Specs
- Experimental Evaluation
- **Related Work**
- Conclusions

Related Work: Determinism

- Deterministic languages and runtimes.
  - Deterministic Parallel Java (UIUC)
  - Kendo (Olszewski, et al, ASPLOS 09)
  - DMP (Devietti, et al, ASPLOS 09)
- Determinism Checking.
  - SingleTrack (Sadowski, et al, ESOP 09)
  - *Race detection* can be thought of as determinism checking.

Outline

- Motivation and Overview
- Background: Deterministic Specs
- Specification Inference Problem
- Inferring Deterministic Specs
- Experimental Evaluation
- Related Work
- **Conclusions**

Conclusions

- Deterministic specifications – lightweight spec of parallel correctness.
  - Much simpler structure than functional correctness specifications.
- Can infer high-quality deterministic specs.
  - From small number of observed runs.
  - Mostly recovered previous manual specs.
  - Found two errors in previous manual specs.

Any Questions?