Better termination proving through cooperation

Marc Brockschmidt [1] Byron Cook [2,3] Carsten Fuhs [3]

[1]RWTH Aachen University

[2]Microsoft Research Cambridge

[3]University College London

May 2013

Automated Termination Analysis: Approaches

Automated Termination Analysis: Approaches

1. Monolithic approaches:
   - Linear/polyranking rank functions
   - Size change

Automated Termination Analysis: Approaches

1. Monolithic approaches:
   - Linear/polyranking rank functions
   - Size change
2. Iterative (by simplification) approaches:
   - Dependency Pair Framework
   - Monotonicity constraints

   Advantages:
   - Can re-use 1
   - Each proof step is progress (removes problem parts)

Automated Termination Analysis: Approaches

1. Monolithic approaches:
   - Linear/polyranking rank functions
   - Size change
2. Iterative (by simplification) approaches:
   - Dependency Pair Framework
   - Monotonicity constraints

   Advantages:
   - Can re-use 1
   - Each proof step is progress (removes problem parts)
3. Iterative construction of termination argument:
   - Transition Invariants
   - Lexicographic combinations

   Advantages:
   - Can re-use 1
   - Safety checker does reasoning about program semantics/invariants

Termination Analysis: Invariants and Rank Functions

Example (Termination depends on context)

```
y := 1;
while x > 0 do
    x := x − y;
    y := y + 1;
done
```

- Invariant $y > 0$ and rank function x prove termination
- How do we know that we need $y > 0$? ↷ x requires it

Termination Analysis: Invariants and Rank Functions

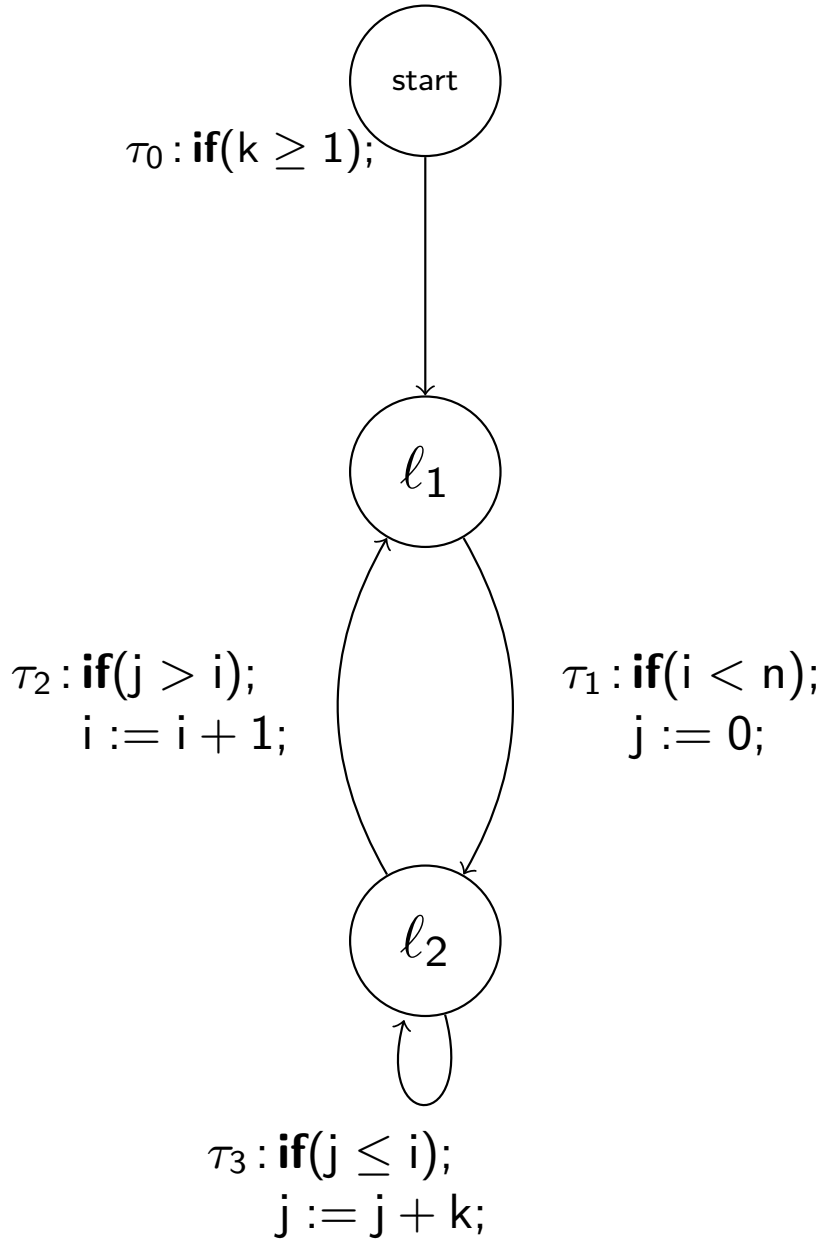
Example (Termination depends on context)

```
y := 1;
while x > 0 do
    x := x − y;
    y := y + 1;
done
```

- Invariant $y > 0$ and rank function x prove termination
- How do we know that we need $y > 0$? $\curvearrowright$ x requires it
- How do we know that x is a RF? $\curvearrowright$ $y > 0$ proves it

Overview

A nested example



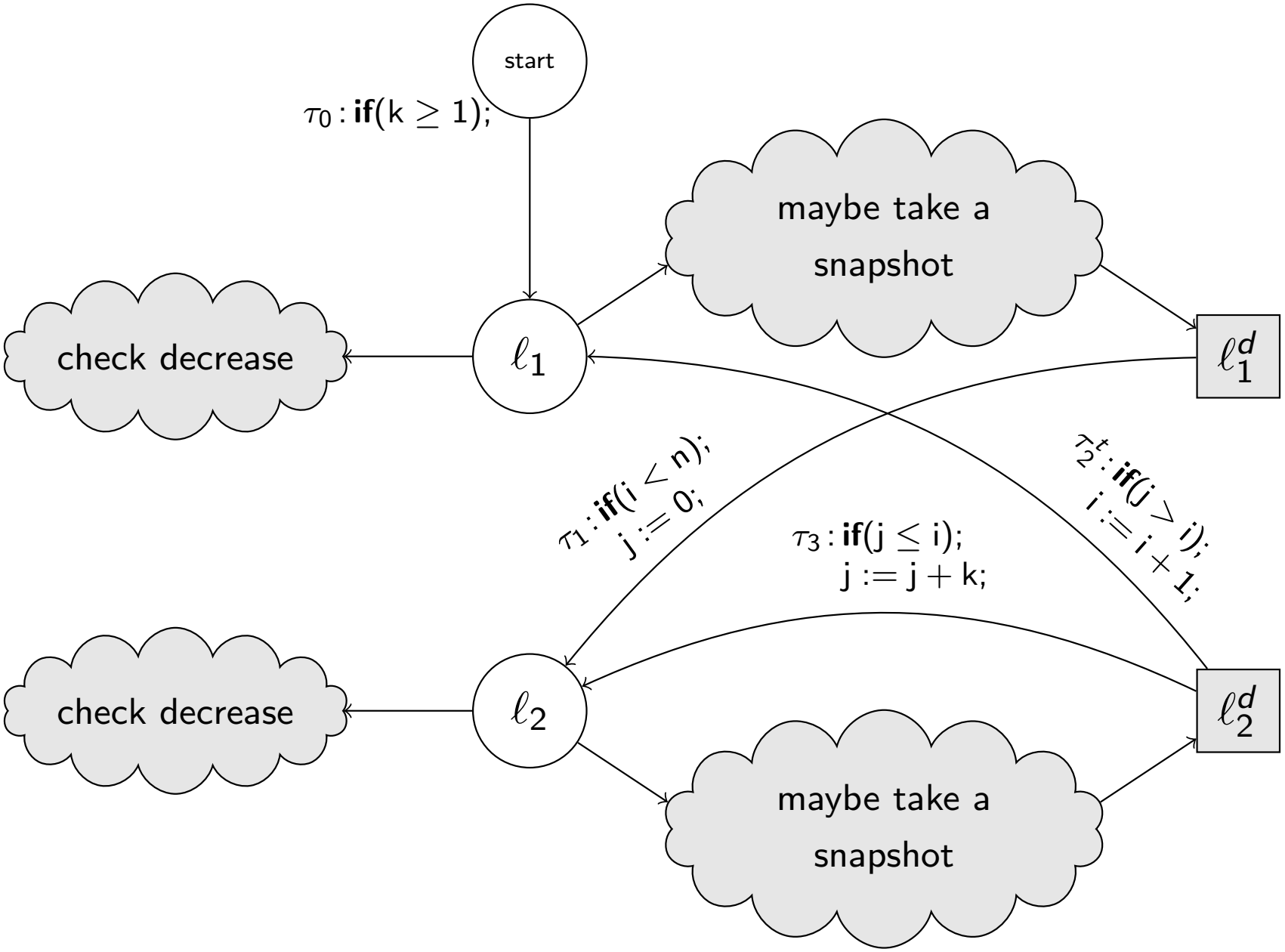
Example (Source)

if k $\geq$ 1 **then**
  i := 0;
ℓ_1 **while** i $<$ n **do**
    j := 0;

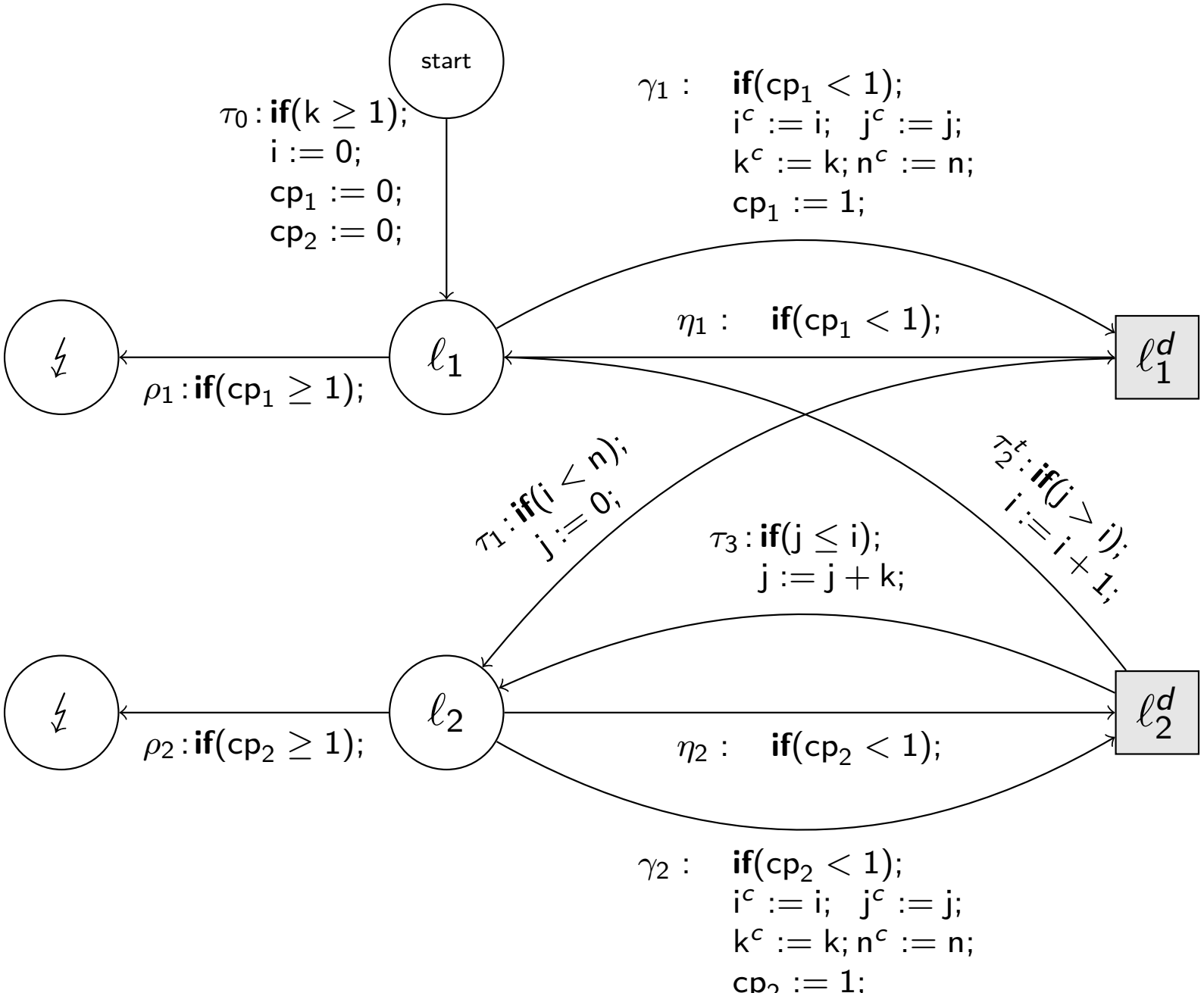
ℓ_2 **while** j $\leq$ i **do**
      j := j + k;
    done

    i := i + 1;
  done
fi

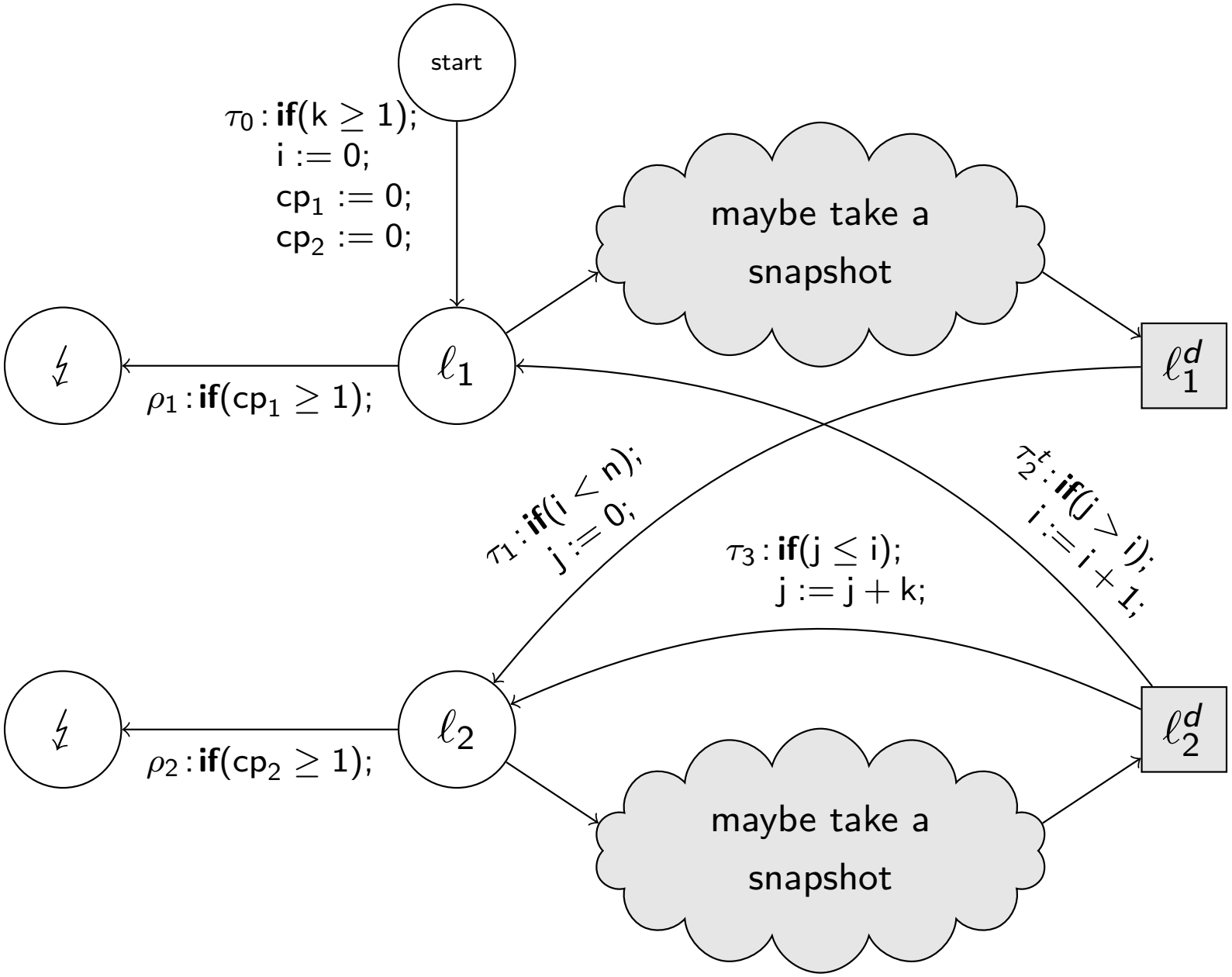
A nested example: Instrumented



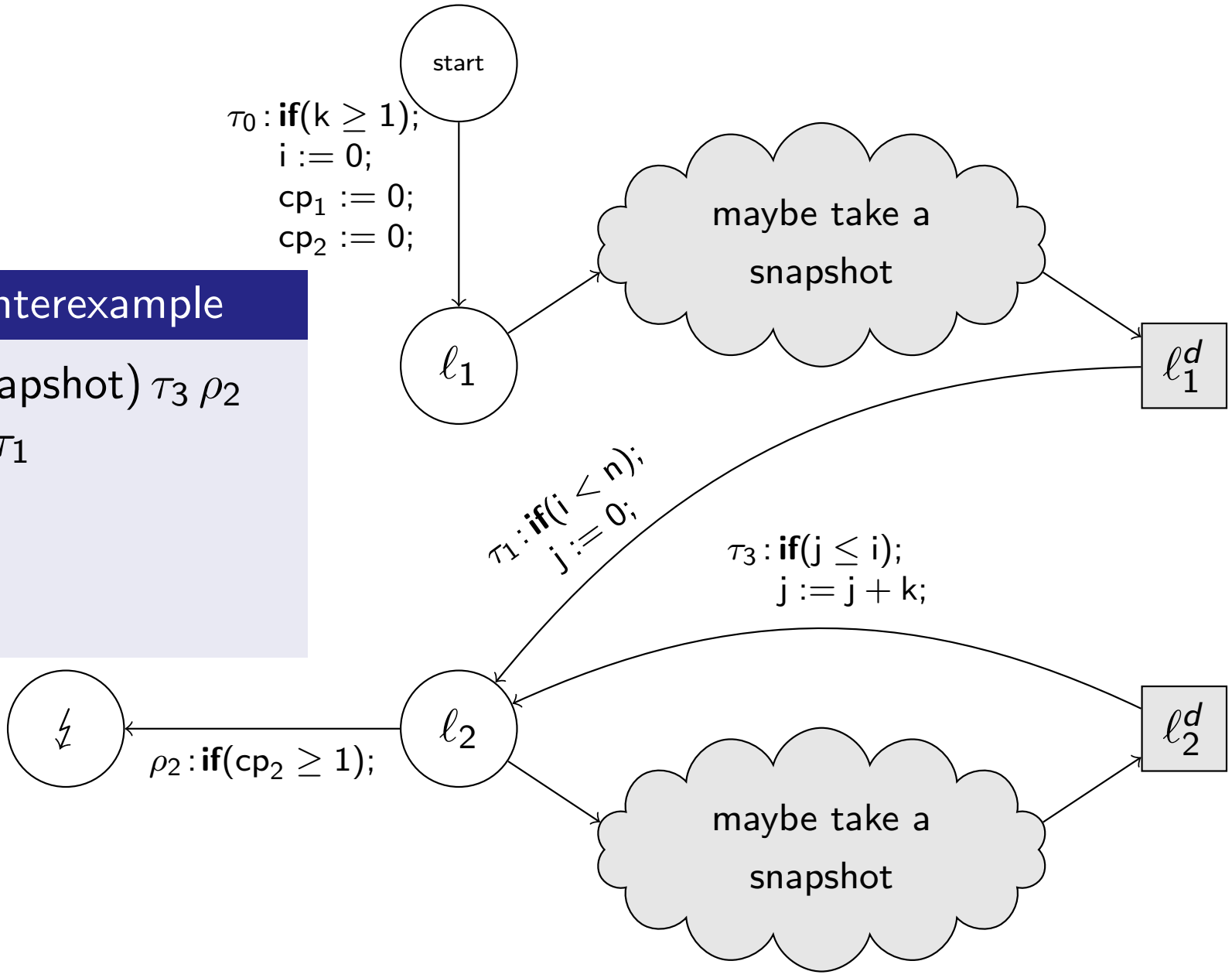
A nested example: Instrumented



A nested example: Instrumented



A nested example: First counterexample



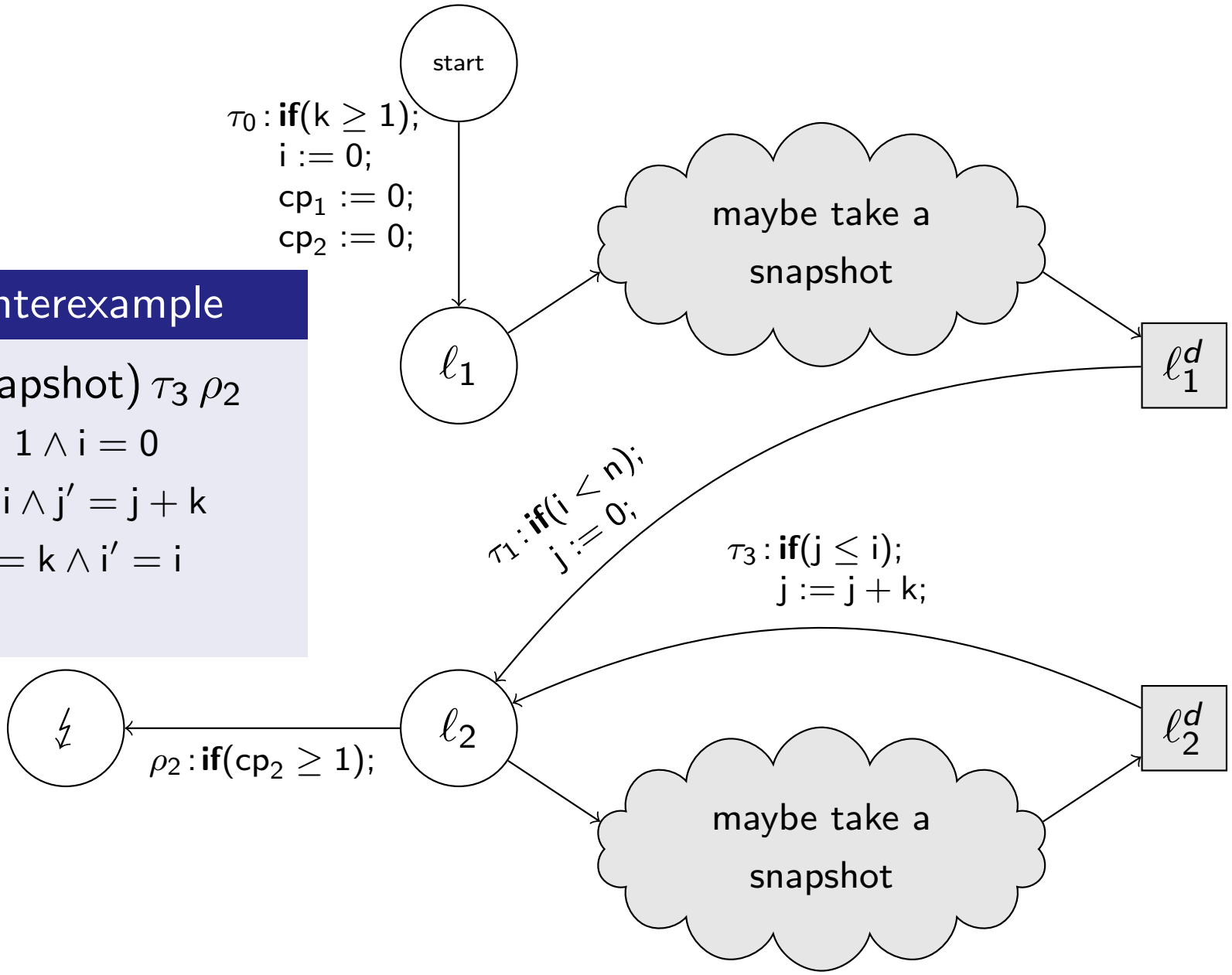
First Counterexample

$\tau_0\ \tau_1$ (snapshot) $\tau_3\ \rho_2$

Stem: $\tau_0\ \tau_1$

Loop: τ_3

A nested example: First counterexample



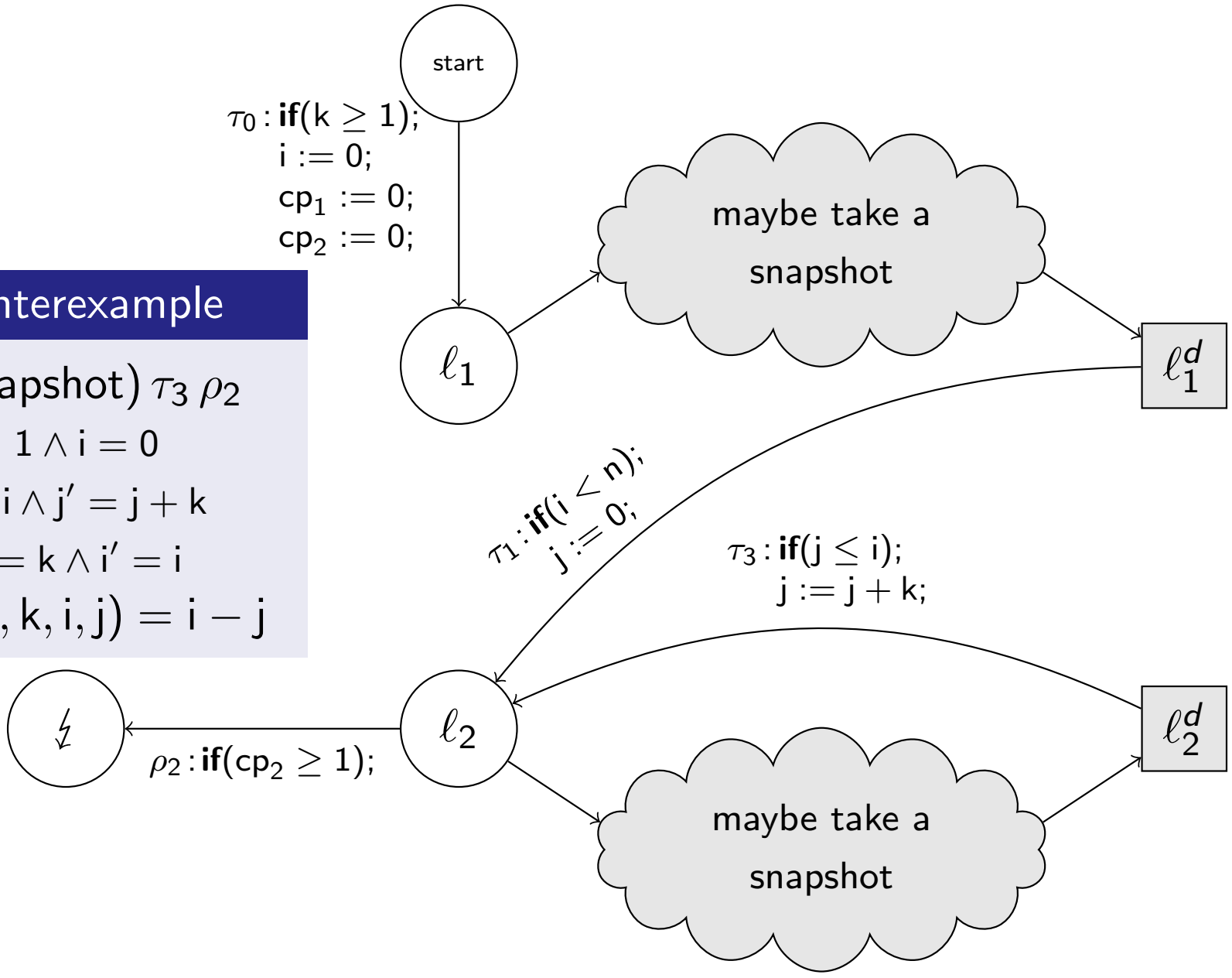
First Counterexample

$\tau_0\ \tau_1\ (\text{snapshot})\ \tau_3\ \rho_2$

Stem: $k \geq 1 \wedge i = 0$

Loop: $j \leq i \wedge j' = j + k \wedge k' = k \wedge i' = i$

A nested example: First counterexample



First Counterexample

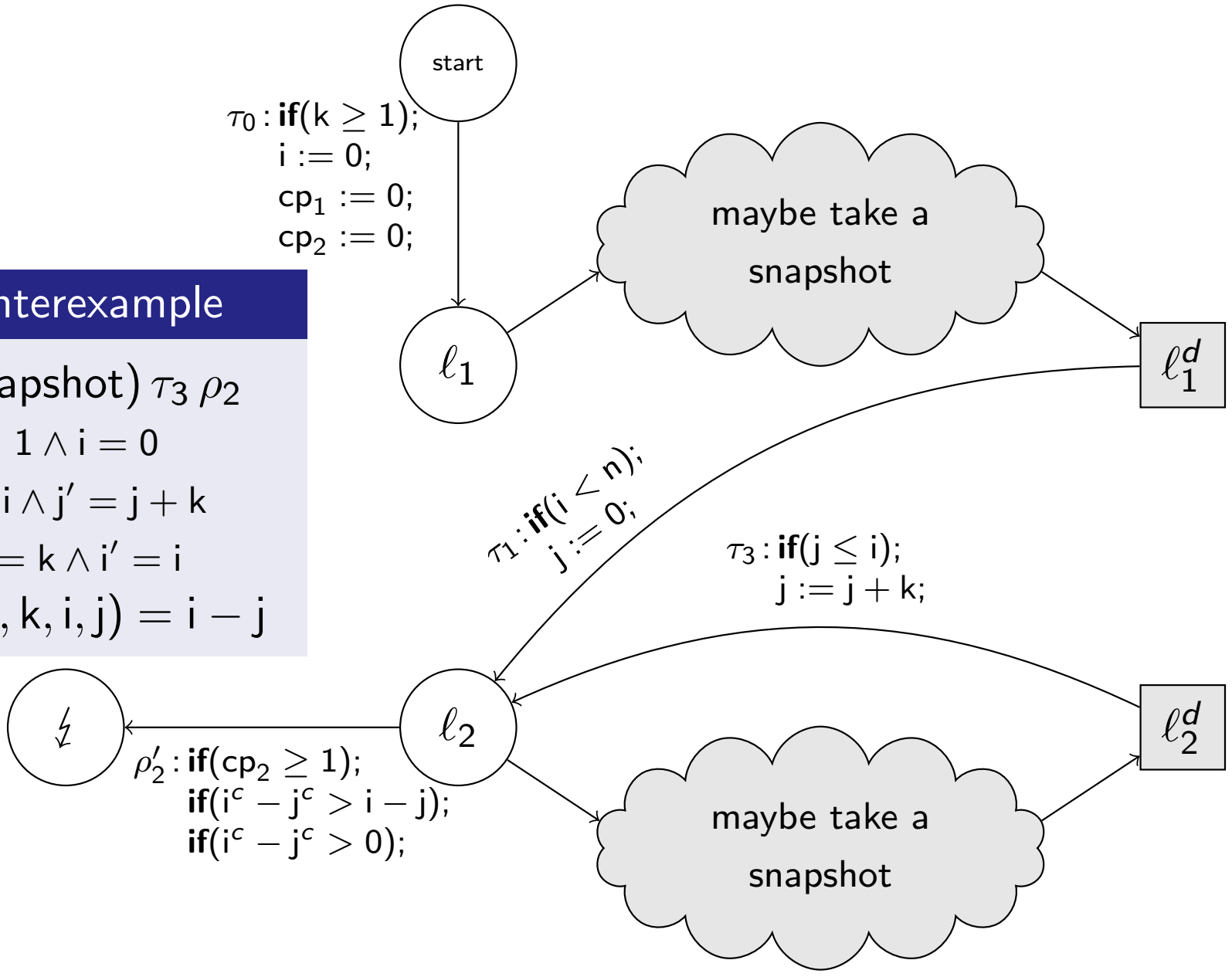
$\tau_0\ \tau_1\ (\text{snapshot})\ \tau_3\ \rho_2$

Stem: $k \geq 1 \wedge i = 0$

Loop: $j \leq i \wedge j' = j + k \wedge k' = k \wedge i' = i$

RF: $f_{\ell_2}(n, k, i, j) = i - j$

A nested example: First counterexample



First Counterexample

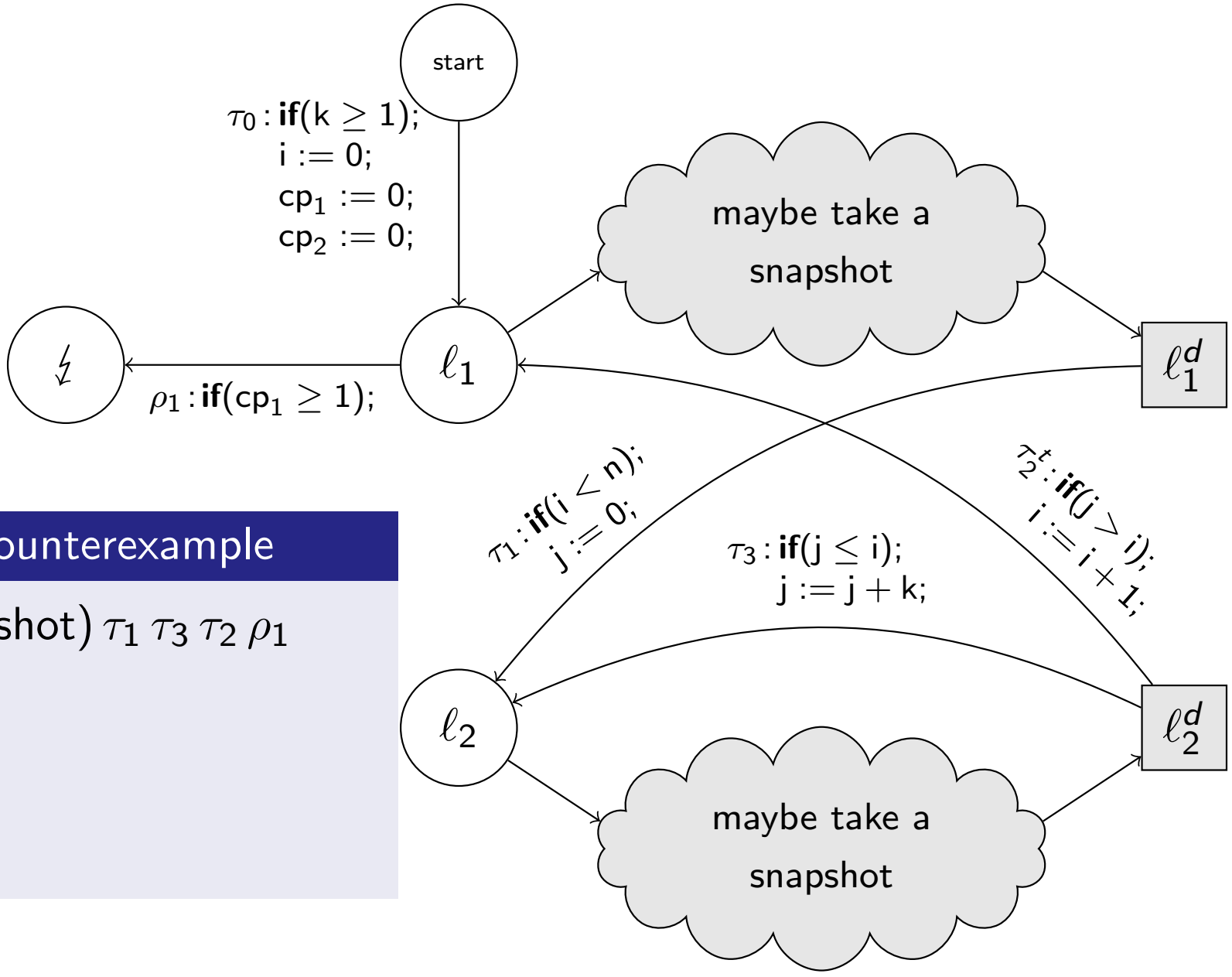
$\tau_0\ \tau_1\ (\text{snapshot})\ \tau_3\ \rho_2$

Stem: $k \geq 1 \wedge i = 0$

Loop: $j \leq i \wedge j' = j + k$
$\wedge\ k' = k \wedge i' = i$

RF: $f_{\ell_2}(n, k, i, j) = i - j$

A nested example: Second counterexample



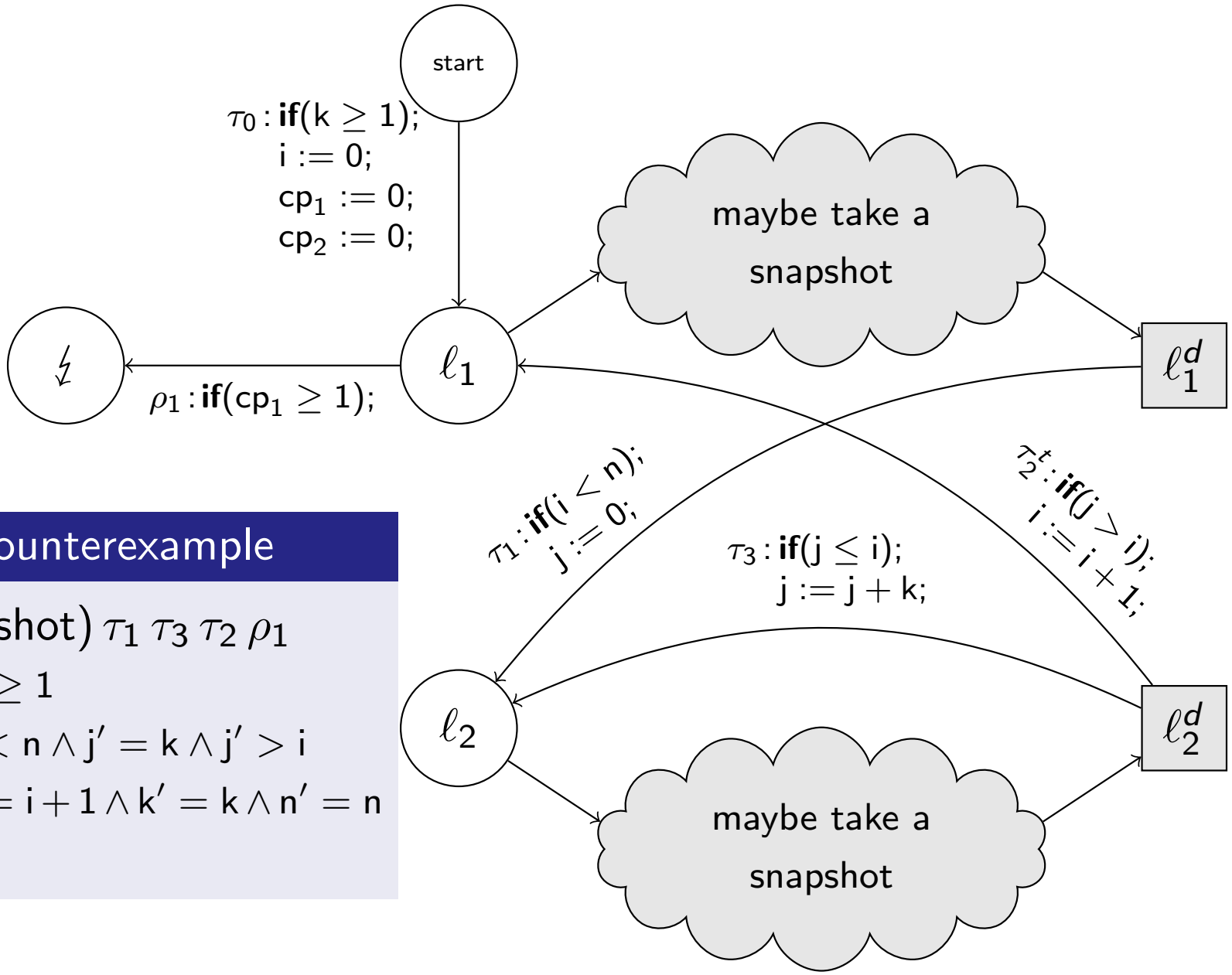
Second Counterexample

τ_0 (snapshot) τ_1 τ_3 τ_2 ρ_1

Stem: τ_0

Loop: τ_3

A nested example: Second counterexample



Second Counterexample

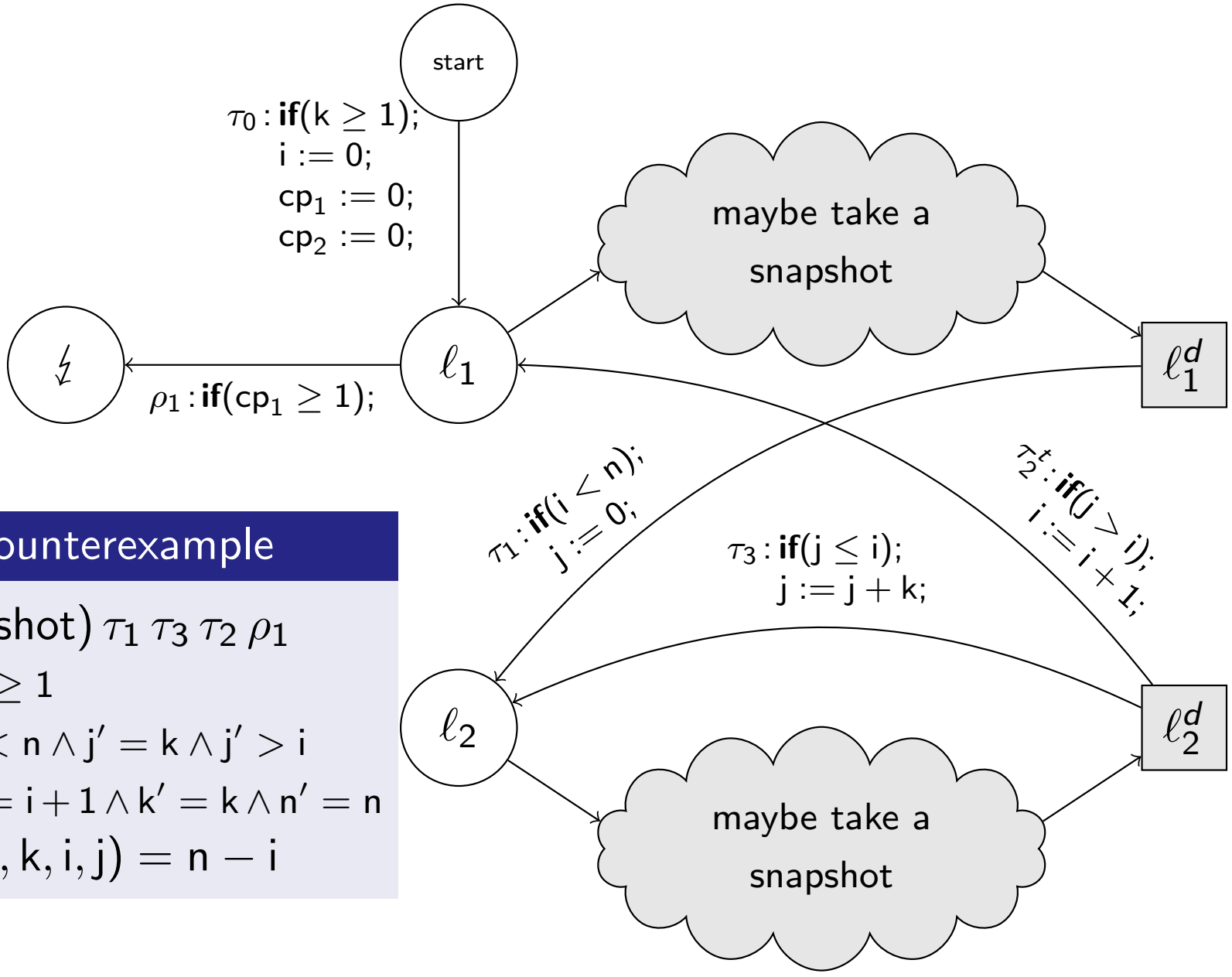
τ_0 (snapshot) τ_1 τ_3 τ_2 ρ_1

Stem: $k \geq 1$

Loop: $i < n \wedge j' = k \wedge j' > i$

$\wedge\ i' = i + 1 \wedge k' = k \wedge n' = n$

A nested example: Second counterexample



Second Counterexample

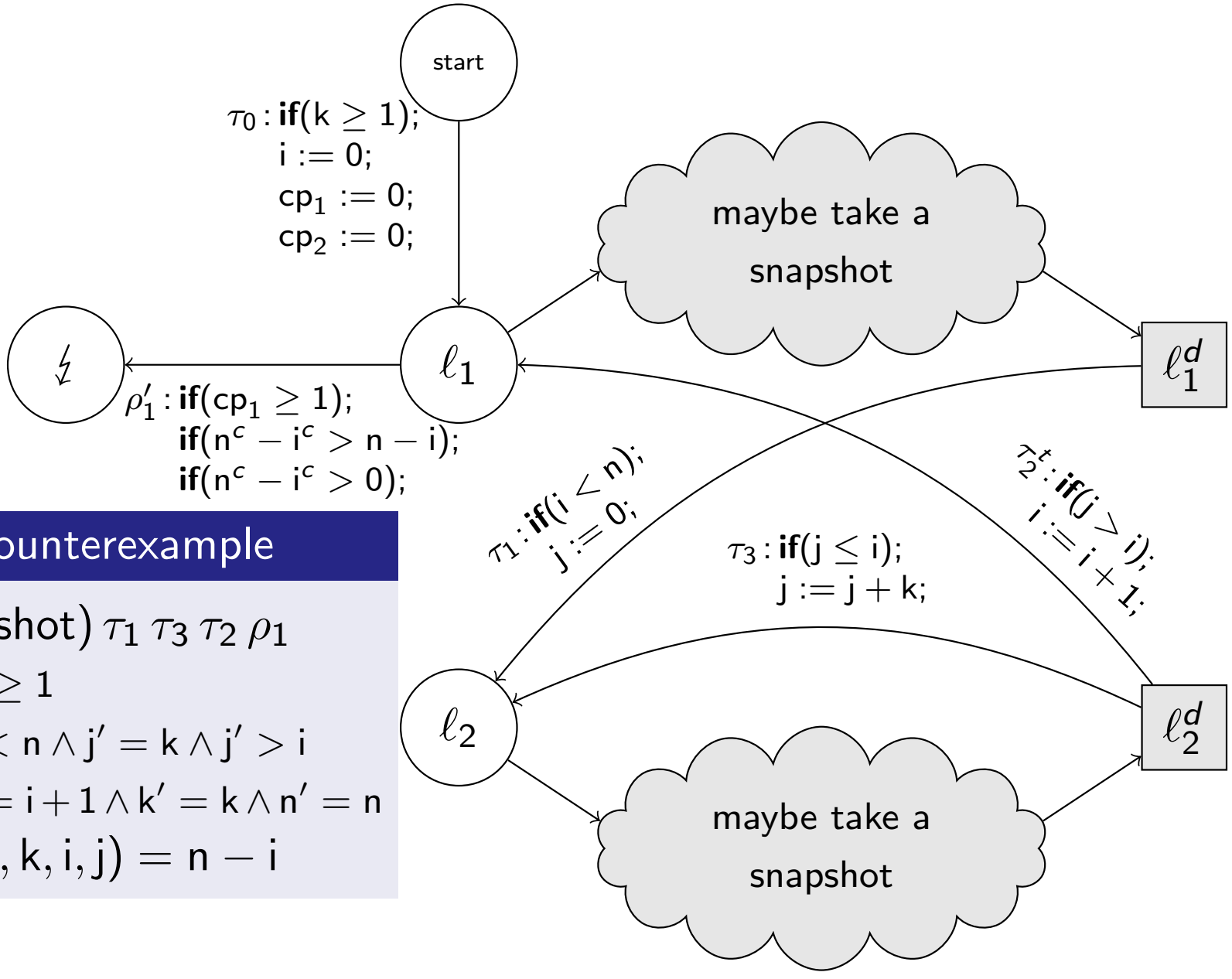
τ_0 (snapshot) τ_1 τ_3 τ_2 ρ_1

Stem: $k \geq 1$

Loop: $i < n \wedge j' = k \wedge j' > i$
$\wedge\ i' = i + 1 \wedge k' = k \wedge n' = n$

RF: $f_{\ell_1}(n, k, i, j) = n - i$

A nested example



Second Counterexample

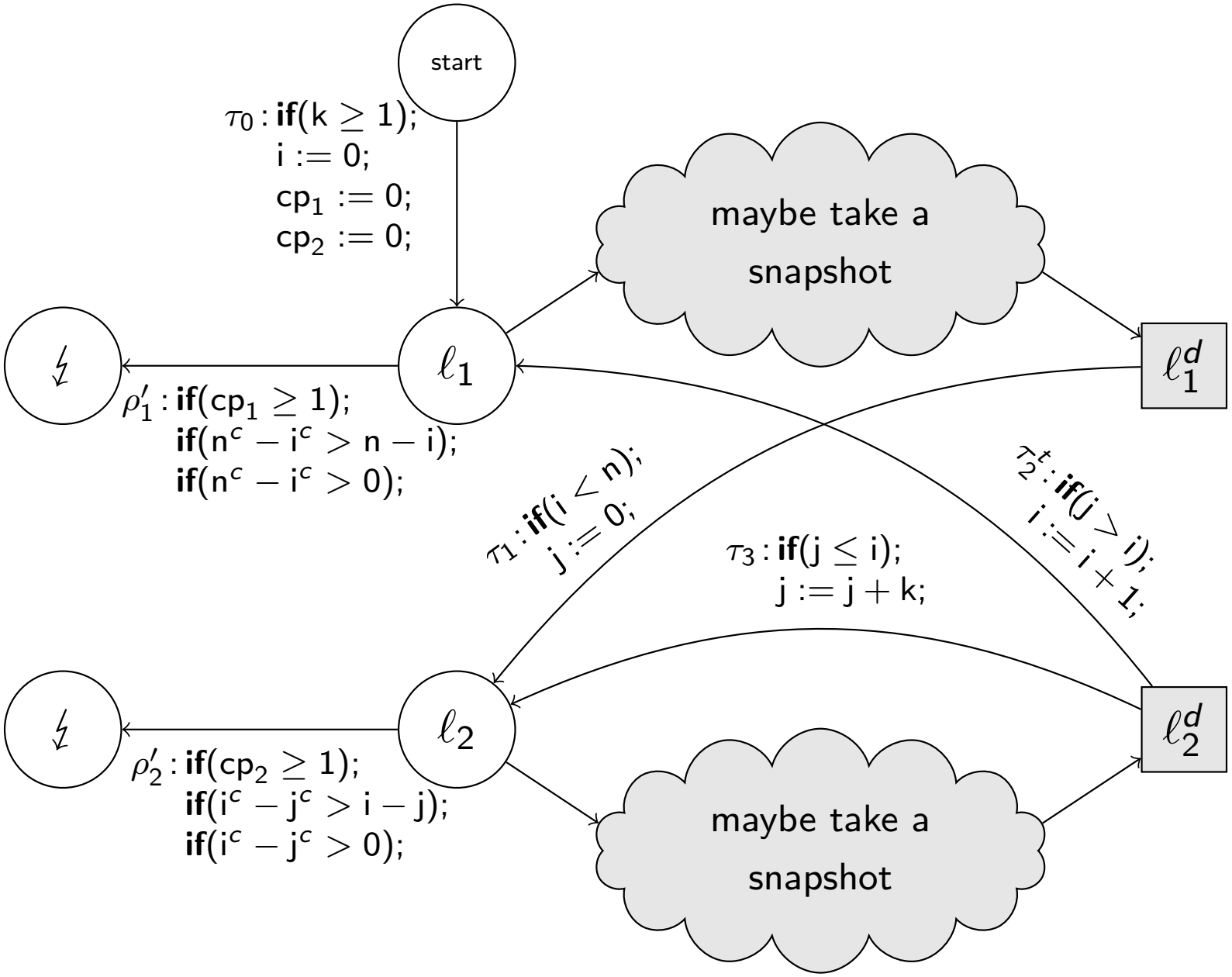
τ_0 (snapshot) τ_1 τ_3 τ_2 ρ_1

Stem: $k \geq 1$

Loop: $i < n \wedge j' = k \wedge j' > i$
$\wedge\ i' = i + 1 \wedge k' = k \wedge n' = n$

RF: $f_{\ell_1}(n, k, i, j) = n - i$

A nested example: Solution



The TERMINATOR approach

Advantages:

- Program semantics & invariants handled by external tool
- Only small program slices considered

The TERMINATOR approach

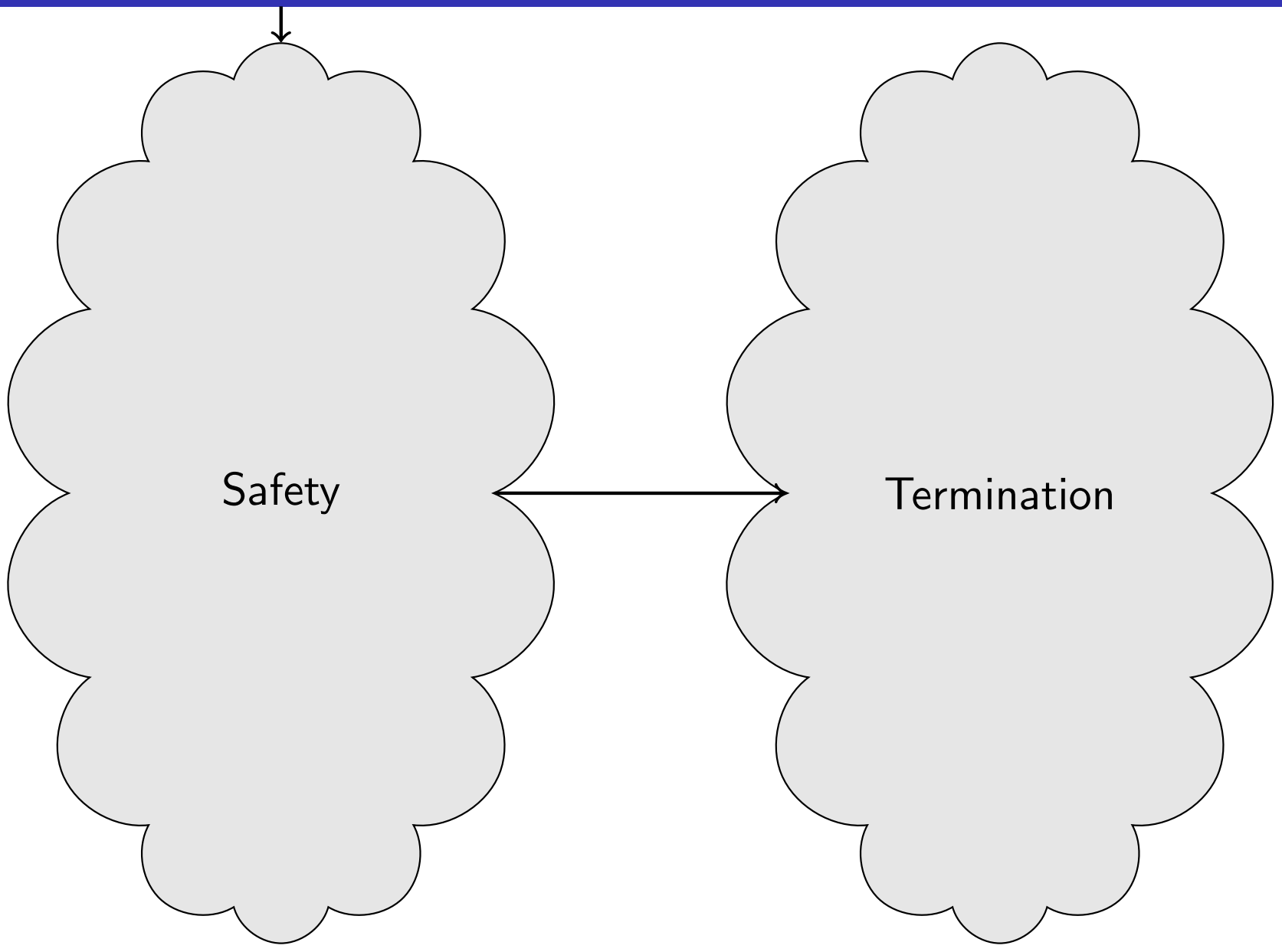
Advantages:

- Program semantics & invariants handled by external tool
- Only small program slices considered

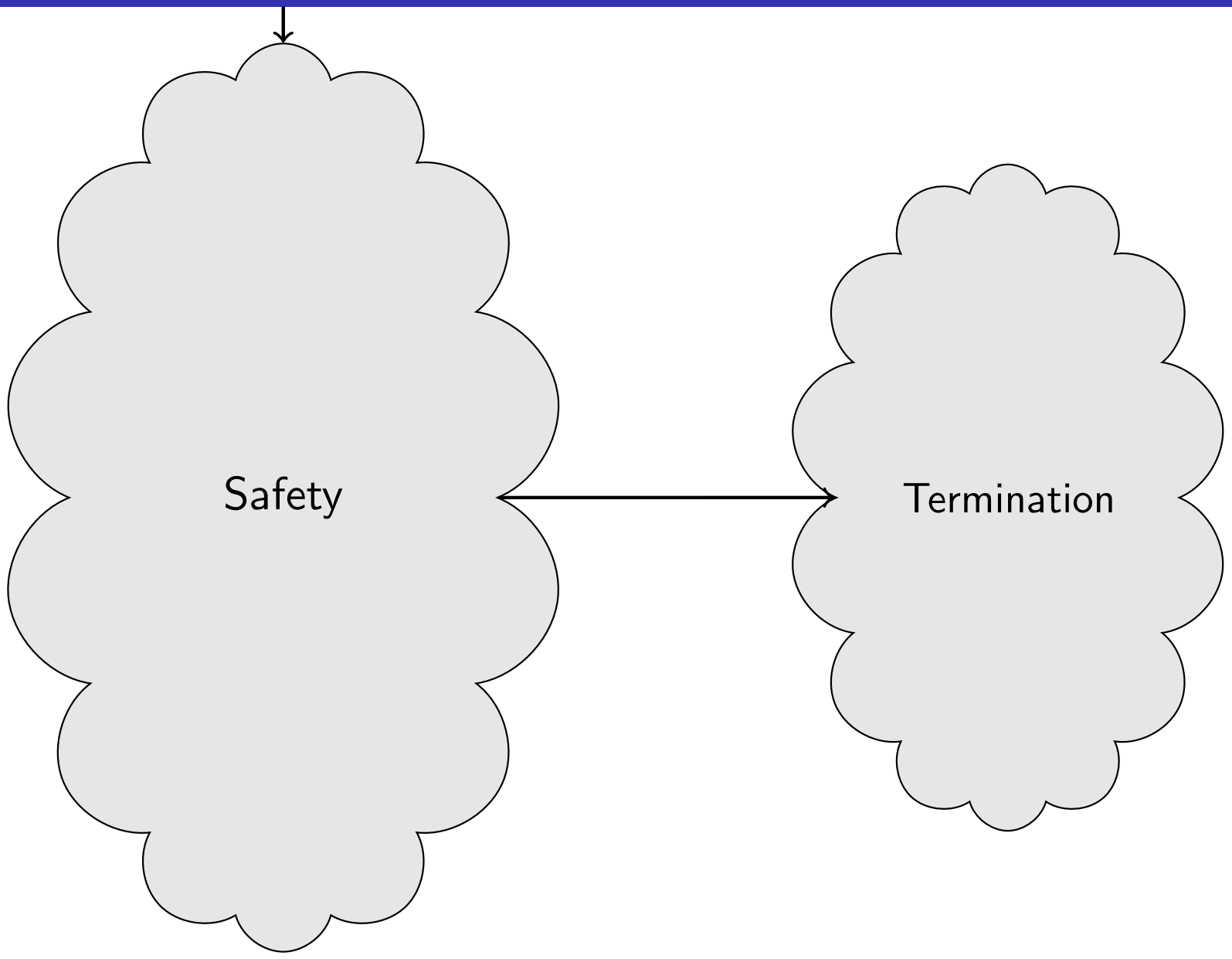
Problems:

- Proof progress not visible
- Order of counterexamples important
- Program slices sometimes too small for informed decisions

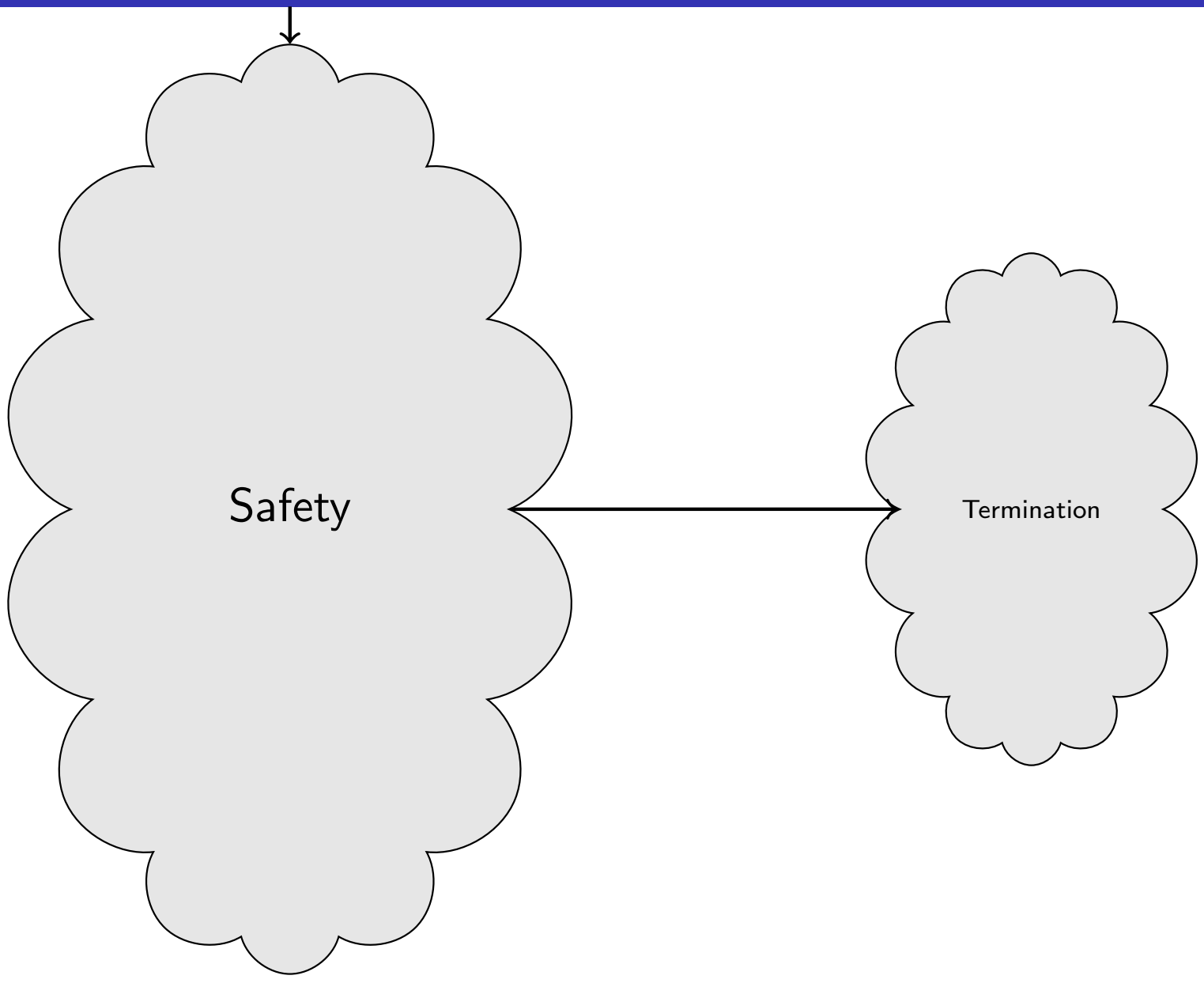
Cooperation: High-level view



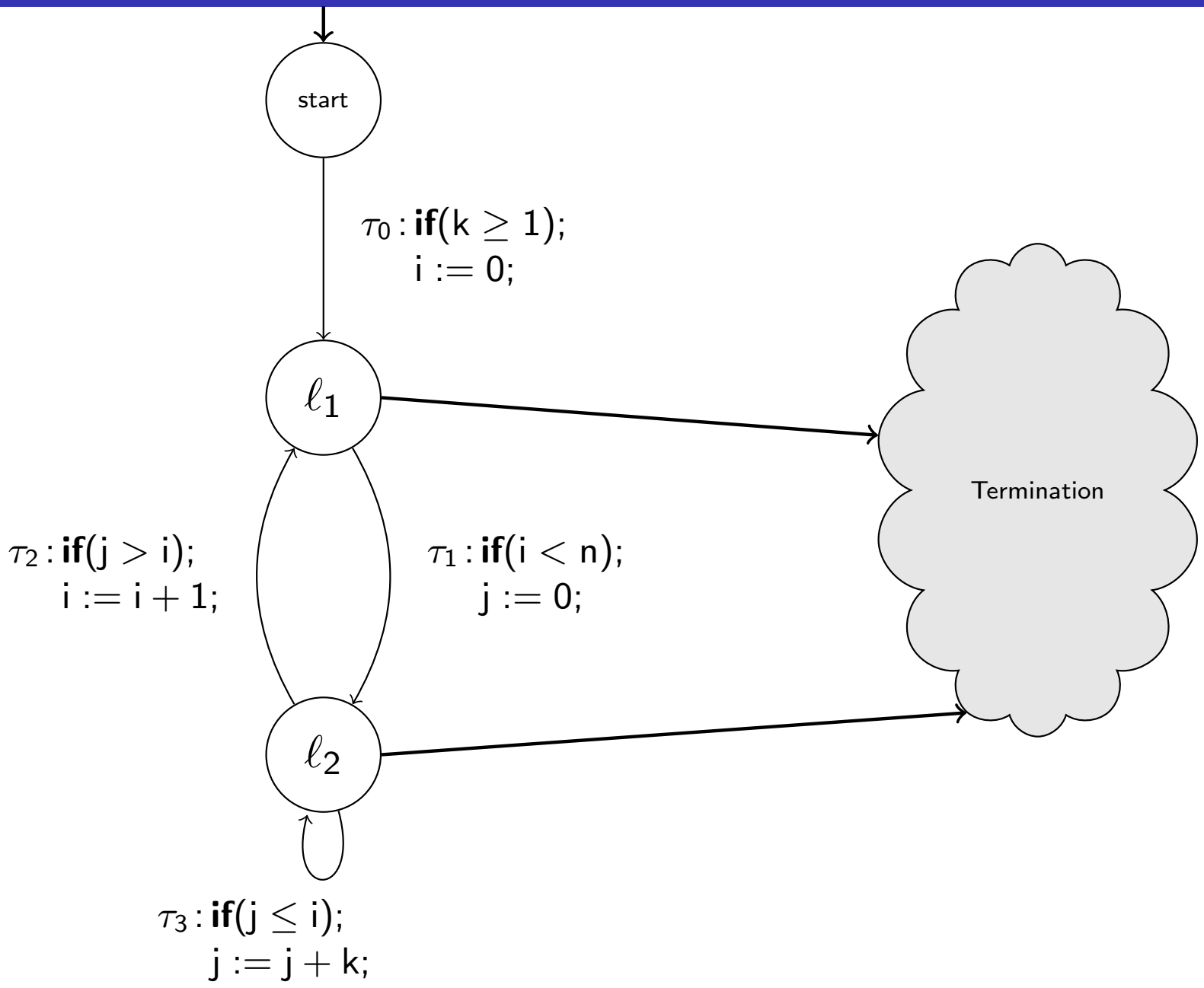
Cooperation: High-level view



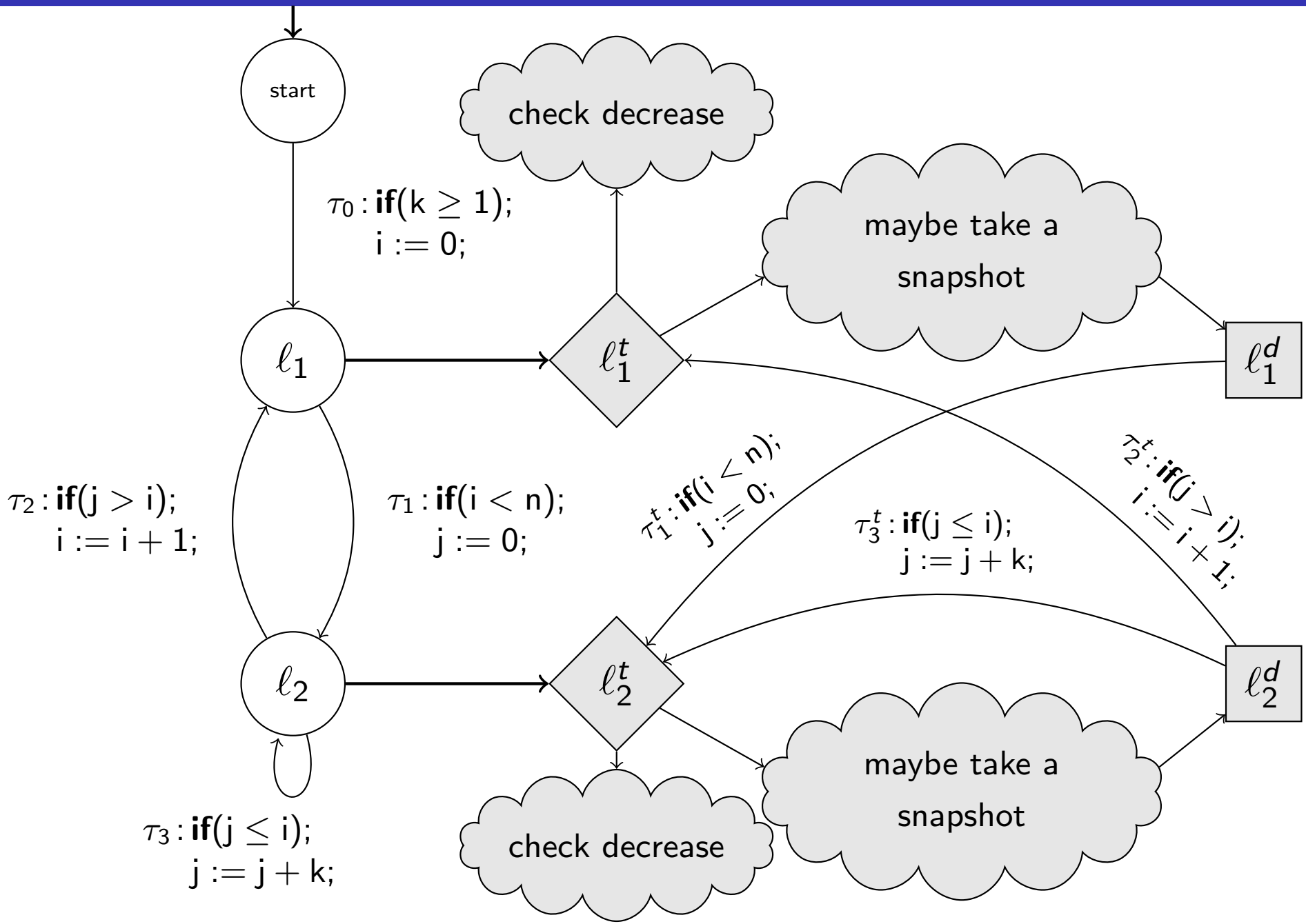
Cooperation: High-level view



Cooperation: High-level view



Cooperation: High-level view



Cooperation

Intuition:

- Safety subgraph: original program
- Termination subgraph: instrumented copy

Cooperation

Intuition:

- Safety subgraph: original program
- Termination subgraph: instrumented copy
- Termination: Simplify problem, “point out hard bits”

Cooperation

Intuition:

- Safety subgraph: original program
- Termination subgraph: instrumented copy
- Termination: Simplify problem, “point out hard bits”
- Safety: Analyze whole program, “point out invariants”

Cooperation

Intuition:

- Safety subgraph: original program
- Termination subgraph: instrumented copy
- Termination: Simplify problem, “point out hard bits”
- Safety: Analyze whole program, “point out invariants”

Approach:

- Analyze whole SCC, not counterexample slice

Cooperation

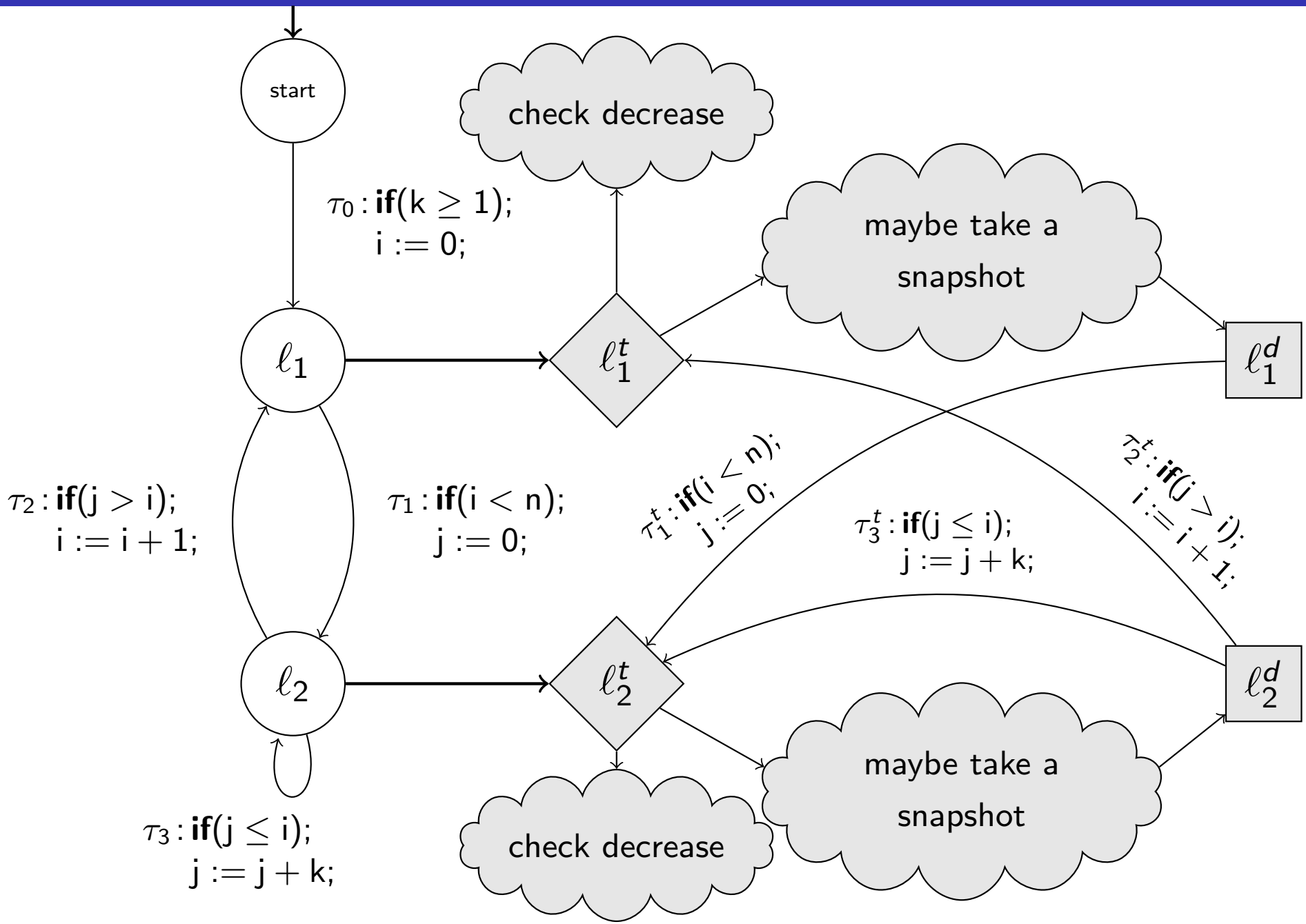
Intuition:

- Safety subgraph: original program
- Termination subgraph: instrumented copy
- Termination: Simplify problem, “point out hard bits”
- Safety: Analyze whole program, “point out invariants”

Approach:

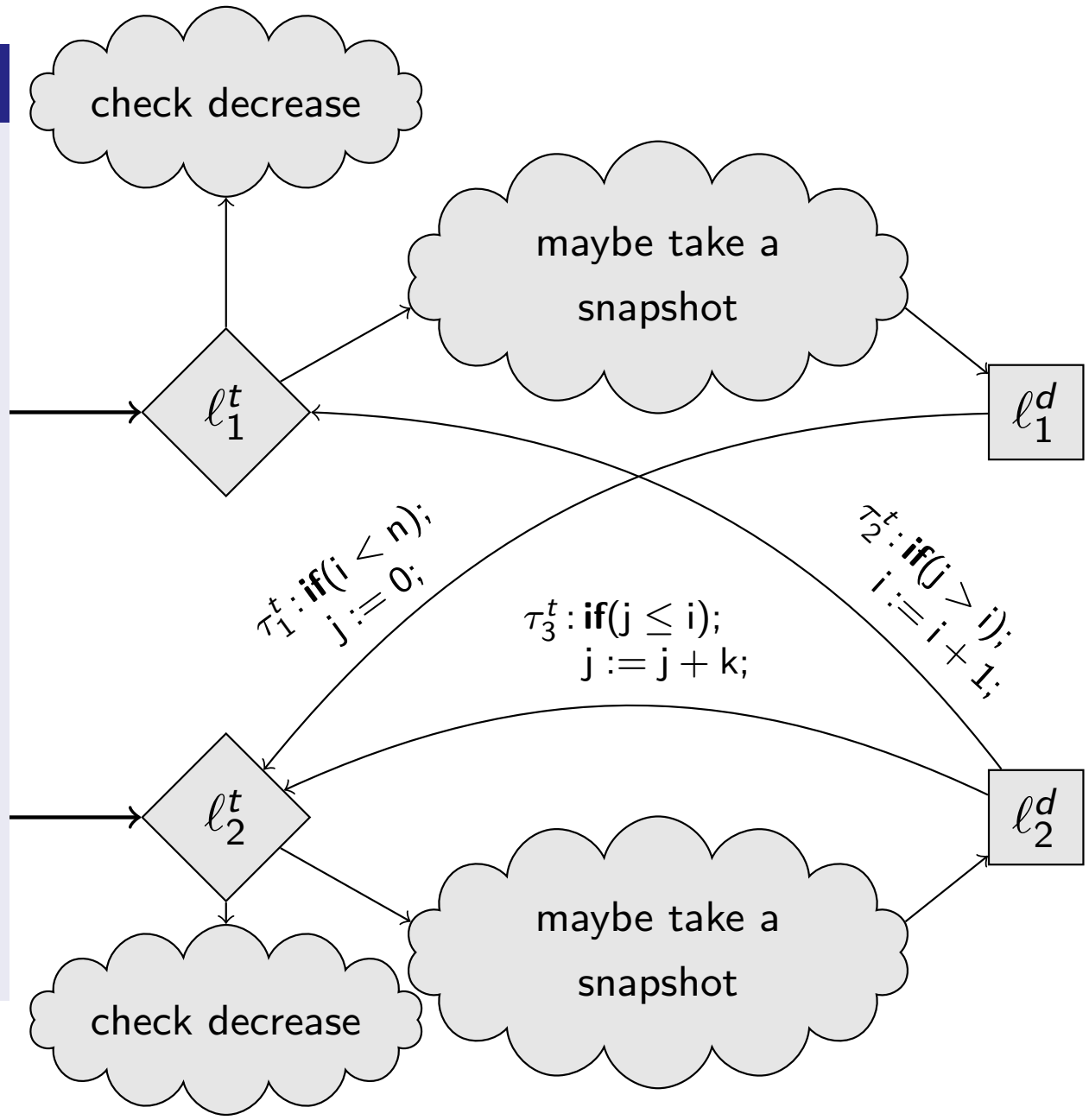
- Analyze whole SCC, not counterexample slice
- Remove transitions after proof

Cooperation: Simplification



Cooperation: Simplification

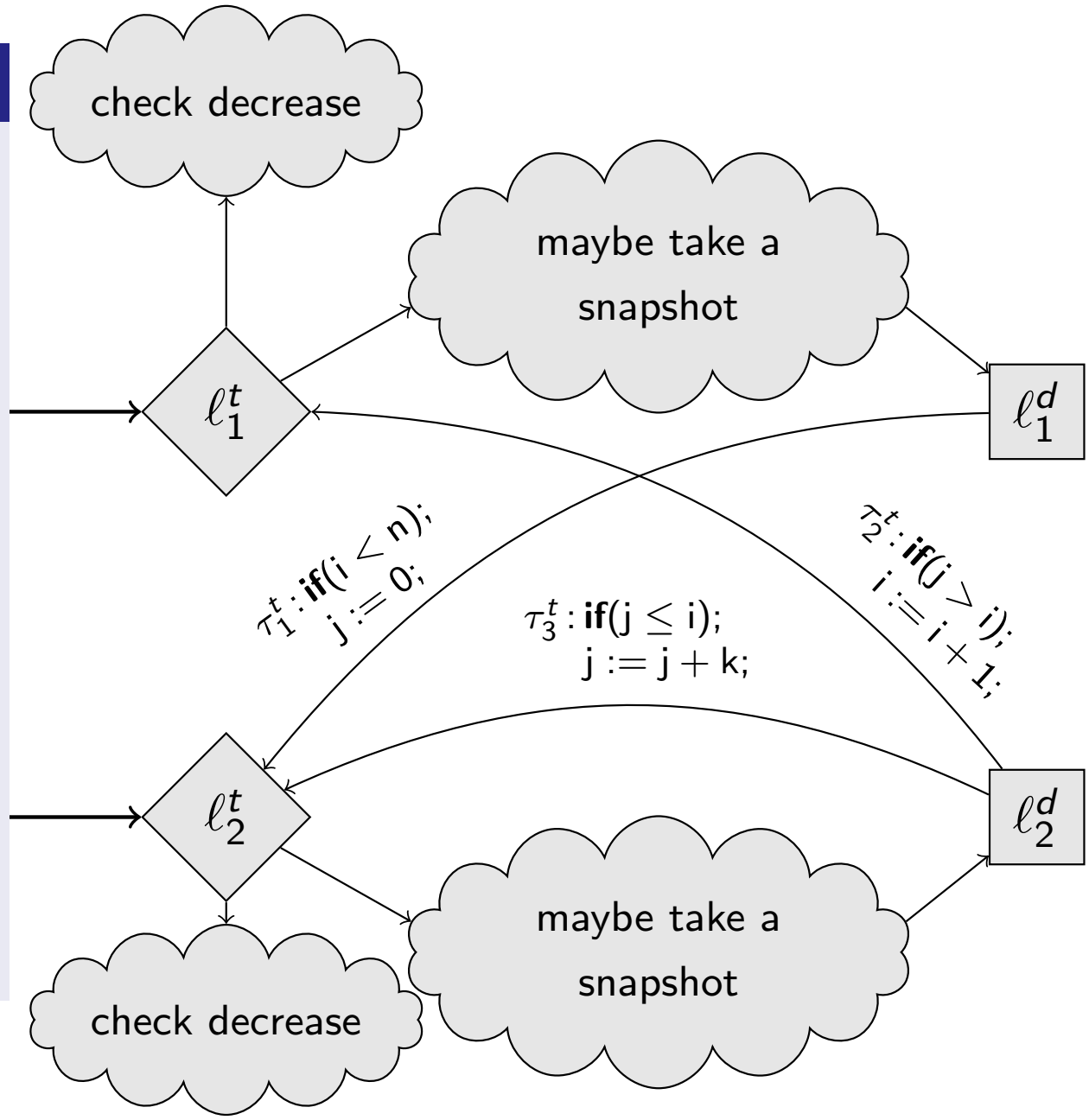
Simplification



Cooperation: Simplification

Simplification

1. Find SCC $\mathcal{S}$ in termination graph:
   $\ell_1^t, \ell_1^d, \ell_2^t, \ell_2^d$



Cooperation: Simplification

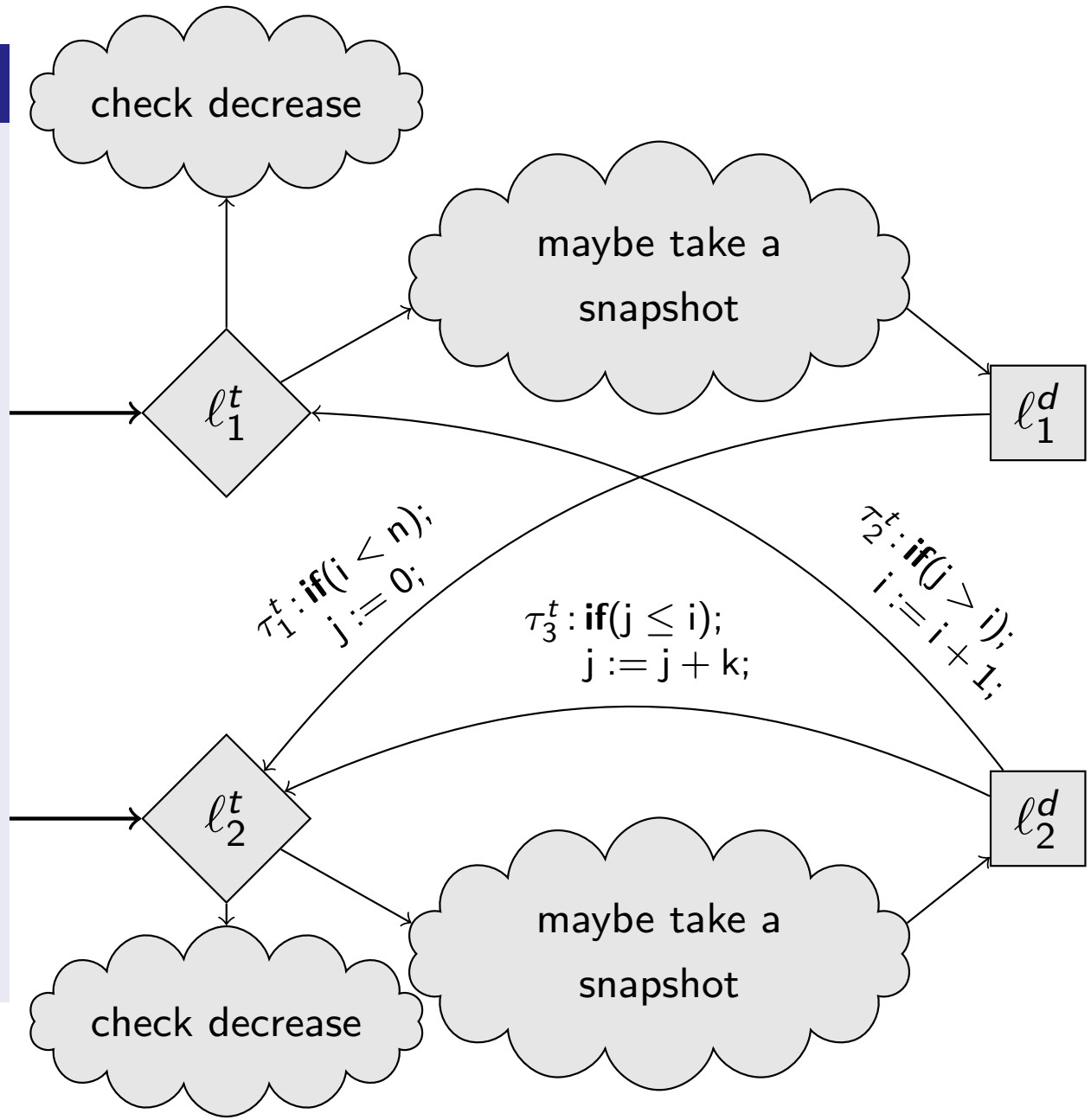
Simplification

1. Find SCC $\mathcal{S}$ in termination graph:

   $$\ell_1^t, \ell_1^d, \ell_2^t, \ell_2^d$$

2. Find $\mathcal{S}$-orienting RF:

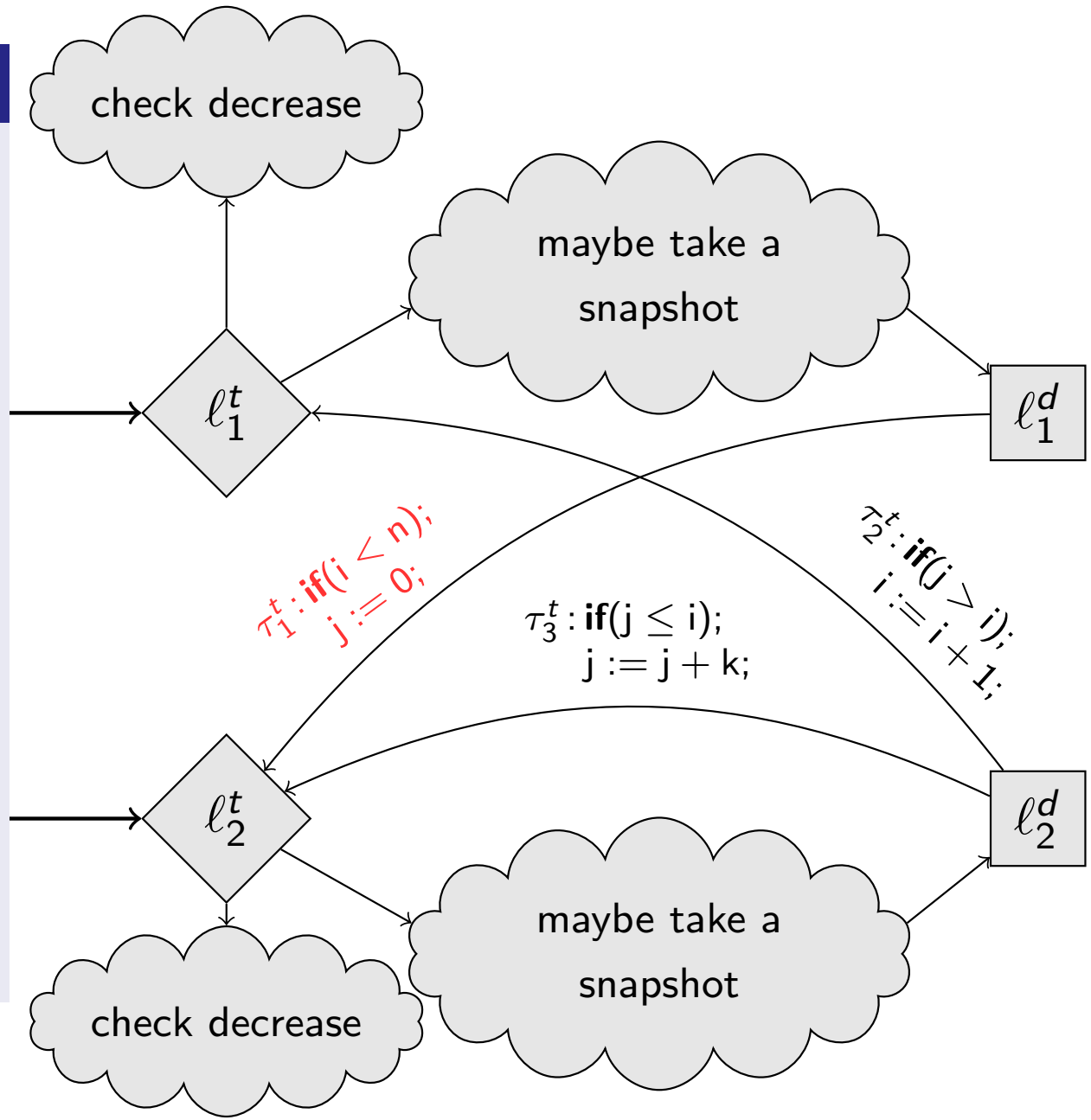
   $$f^1_{\ell_1^t}(\mathsf{i},\mathsf{j},\mathsf{k},\mathsf{n}) = \mathsf{n} - \mathsf{i} + 1$$
   $$f^1_{\ell_1^d}(\mathsf{i},\mathsf{j},\mathsf{k},\mathsf{n}) = \mathsf{n} - \mathsf{i} + 1$$
   $$f^1_{\ell_2^t}(\mathsf{i},\mathsf{j},\mathsf{k},\mathsf{n}) = \mathsf{n} - \mathsf{i}$$
   $$f^1_{\ell_2^d}(\mathsf{i},\mathsf{j},\mathsf{k},\mathsf{n}) = \mathsf{n} - \mathsf{i}$$



Cooperation: Simplification

Simplification

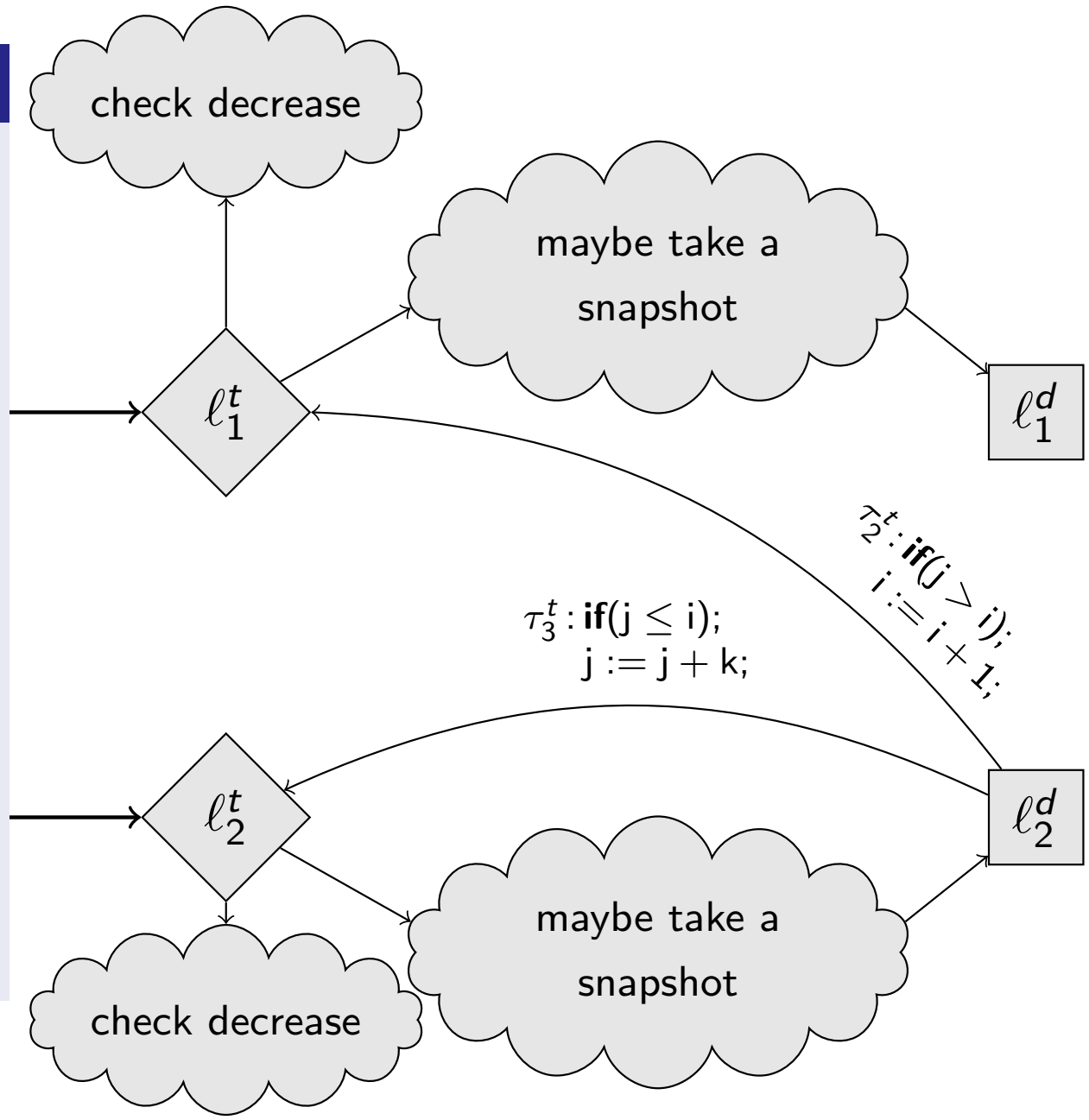
1. Find SCC $\mathcal{S}$ in termination graph:
   $$\ell_1^t, \ell_1^d, \ell_2^t, \ell_2^d$$
2. Find $\mathcal{S}$-orienting RF:
   $$f^1_{\ell_1^t}(\mathsf{i},\mathsf{j},\mathsf{k},\mathsf{n}) = \mathsf{n} - \mathsf{i} + 1$$
   $$f^1_{\ell_1^d}(\mathsf{i},\mathsf{j},\mathsf{k},\mathsf{n}) = \mathsf{n} - \mathsf{i} + 1$$
   $$f^1_{\ell_2^t}(\mathsf{i},\mathsf{j},\mathsf{k},\mathsf{n}) = \mathsf{n} - \mathsf{i}$$
   $$f^1_{\ell_2^d}(\mathsf{i},\mathsf{j},\mathsf{k},\mathsf{n}) = \mathsf{n} - \mathsf{i}$$
3. Delete decr./bounded



Cooperation: Simplification

Simplification

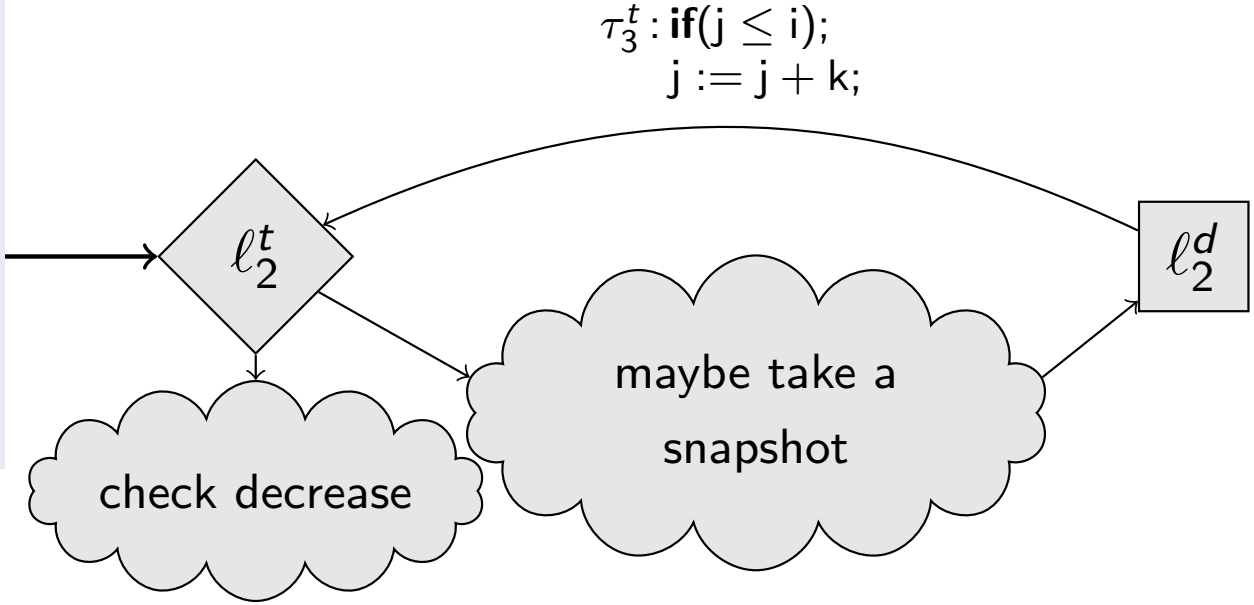
1. Find SCC $\mathcal{S}$ in termination graph:
   $$\ell_1^t, \ell_1^d, \ell_2^t, \ell_2^d$$
2. Find $\mathcal{S}$-orienting RF:
   $$f^1_{\ell_1^t}(\mathsf{i},\mathsf{j},\mathsf{k},\mathsf{n}) = \mathsf{n} - \mathsf{i} + 1$$
   $$f^1_{\ell_1^d}(\mathsf{i},\mathsf{j},\mathsf{k},\mathsf{n}) = \mathsf{n} - \mathsf{i} + 1$$
   $$f^1_{\ell_2^t}(\mathsf{i},\mathsf{j},\mathsf{k},\mathsf{n}) = \mathsf{n} - \mathsf{i}$$
   $$f^1_{\ell_2^d}(\mathsf{i},\mathsf{j},\mathsf{k},\mathsf{n}) = \mathsf{n} - \mathsf{i}$$
3. Delete decr./bounded



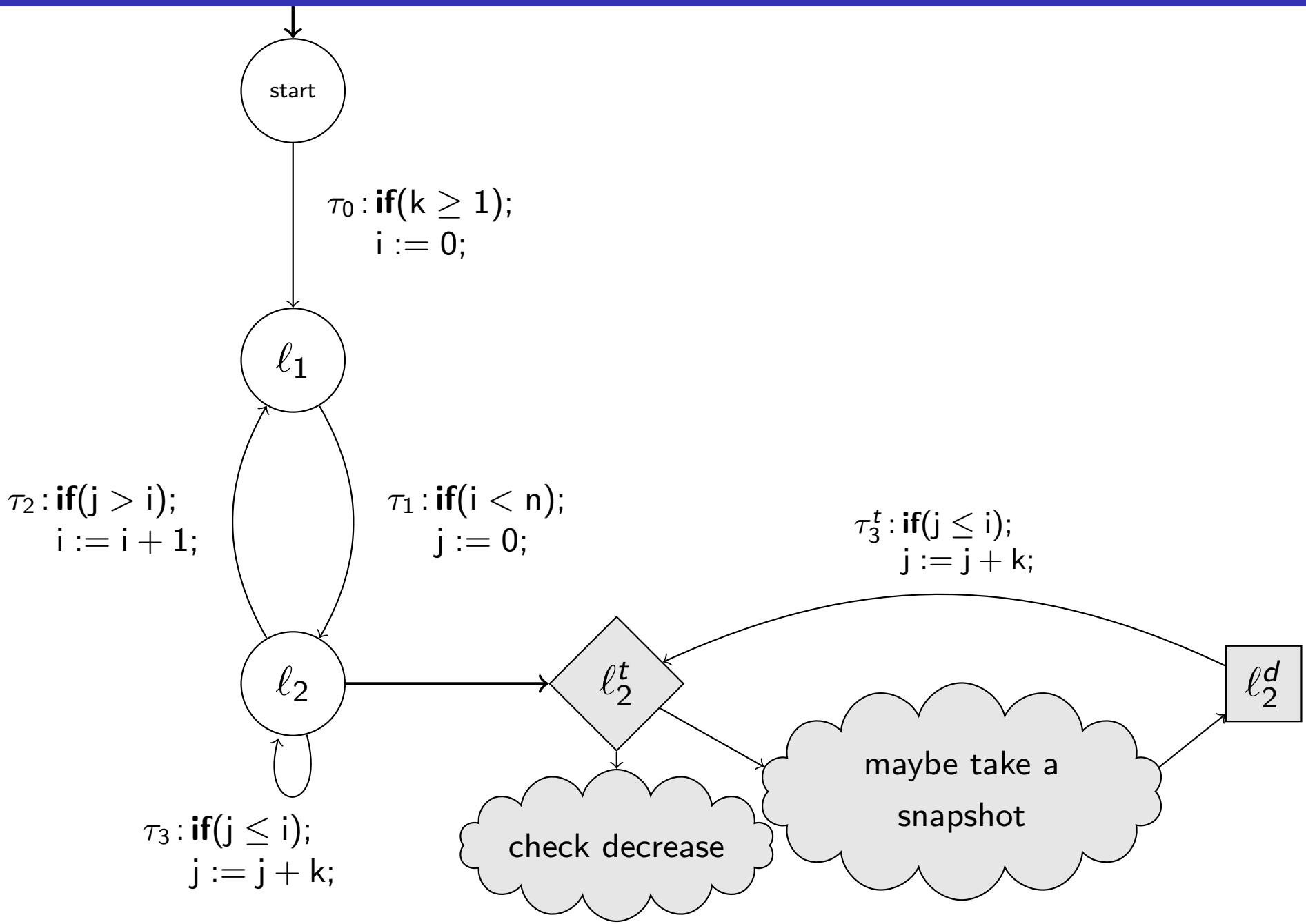
Cooperation: Simplification

Simplification

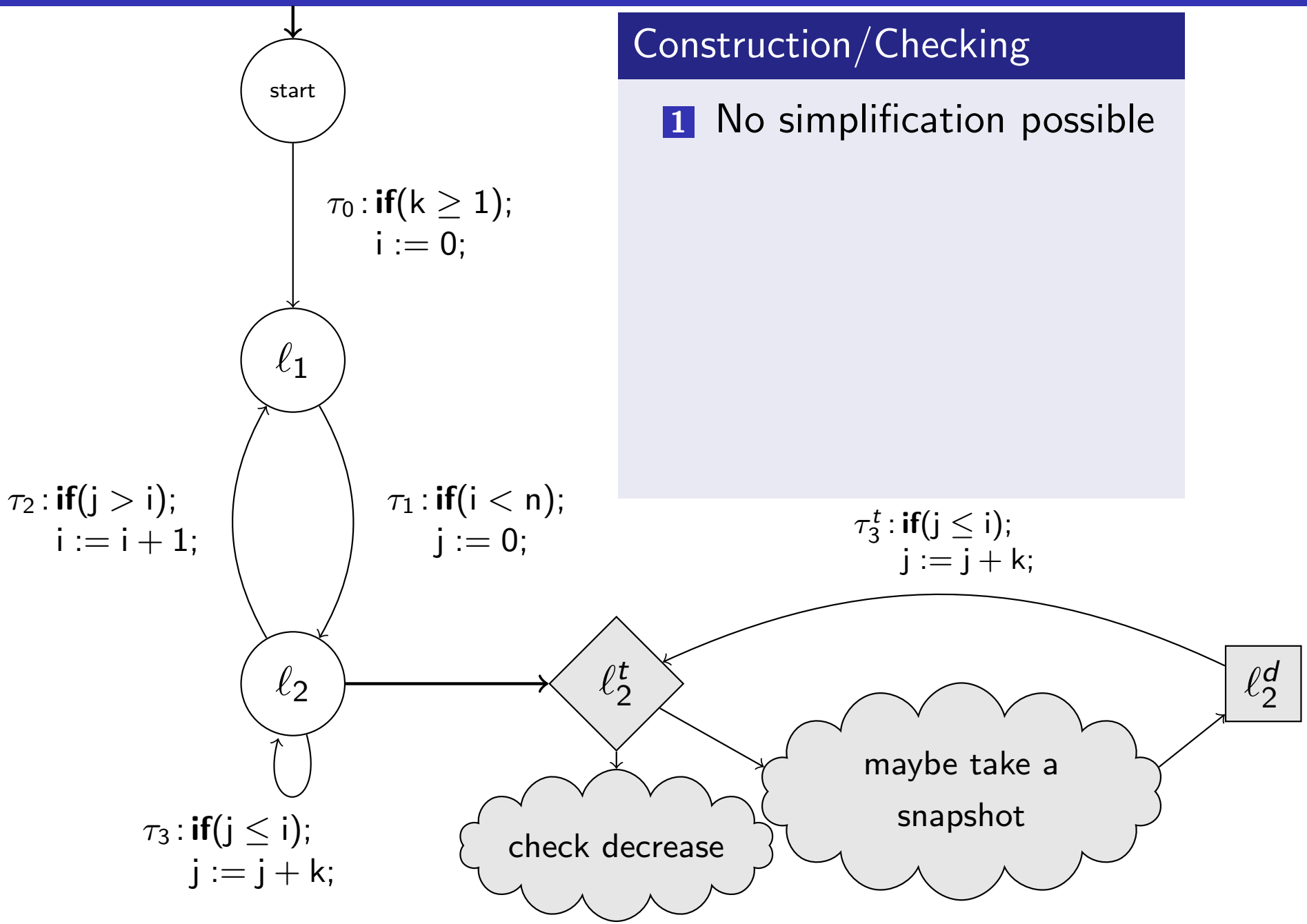
1. Find SCC $\mathcal{S}$ in termination graph:
   $$\ell_1^t, \ell_1^d, \ell_2^t, \ell_2^d$$
2. Find $\mathcal{S}$-orienting RF:
   $$f^1_{\ell_1^t}(\mathsf{i},\mathsf{j},\mathsf{k},\mathsf{n}) = \mathsf{n} - \mathsf{i} + 1$$
   $$f^1_{\ell_1^d}(\mathsf{i},\mathsf{j},\mathsf{k},\mathsf{n}) = \mathsf{n} - \mathsf{i} + 1$$
   $$f^1_{\ell_2^t}(\mathsf{i},\mathsf{j},\mathsf{k},\mathsf{n}) = \mathsf{n} - \mathsf{i}$$
   $$f^1_{\ell_2^d}(\mathsf{i},\mathsf{j},\mathsf{k},\mathsf{n}) = \mathsf{n} - \mathsf{i}$$
3. Delete decr./bounded
4. Clean up



Cooperation



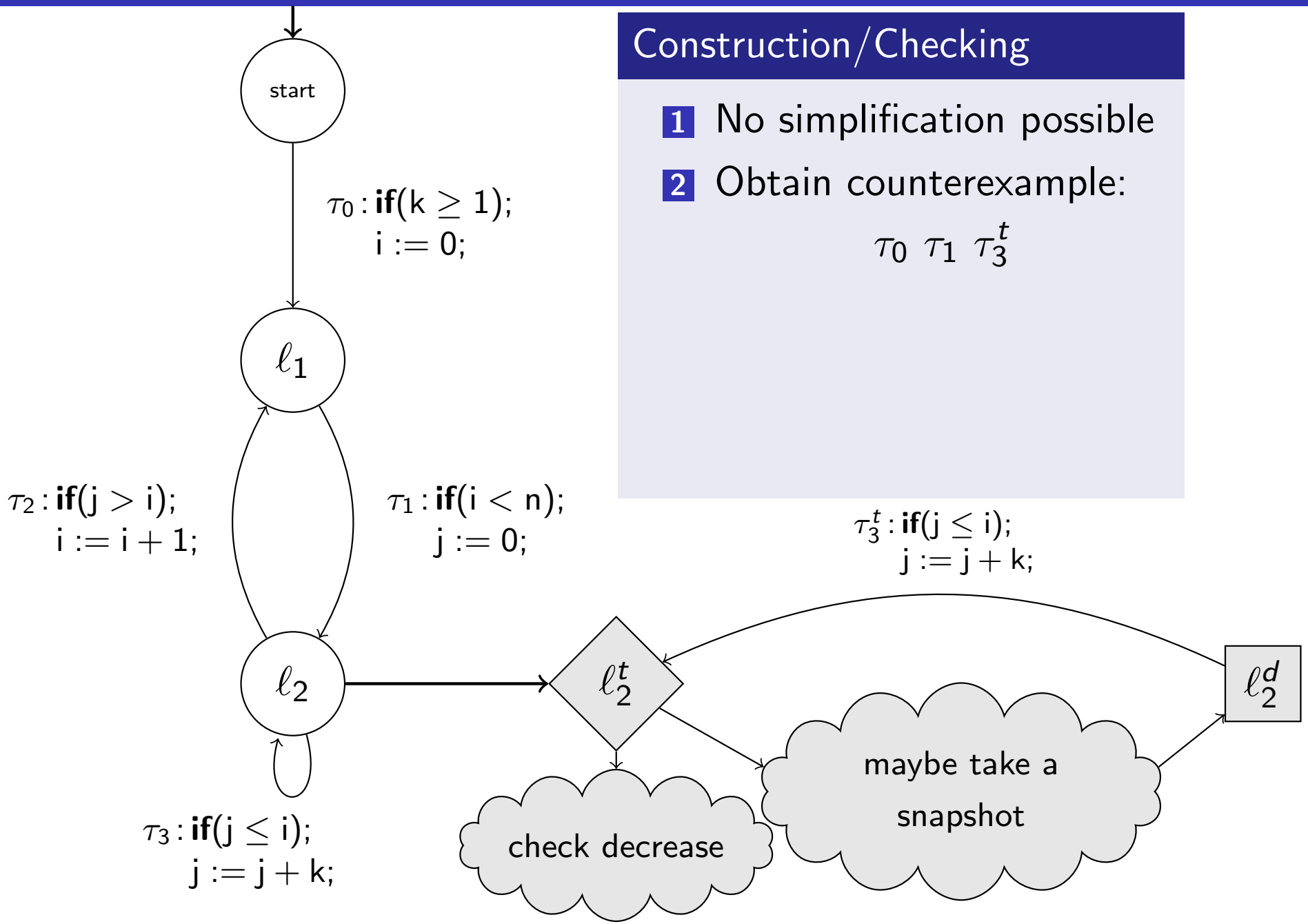
Cooperation: Invariants



Construction/Checking

1. No simplification possible

Cooperation: Invariants

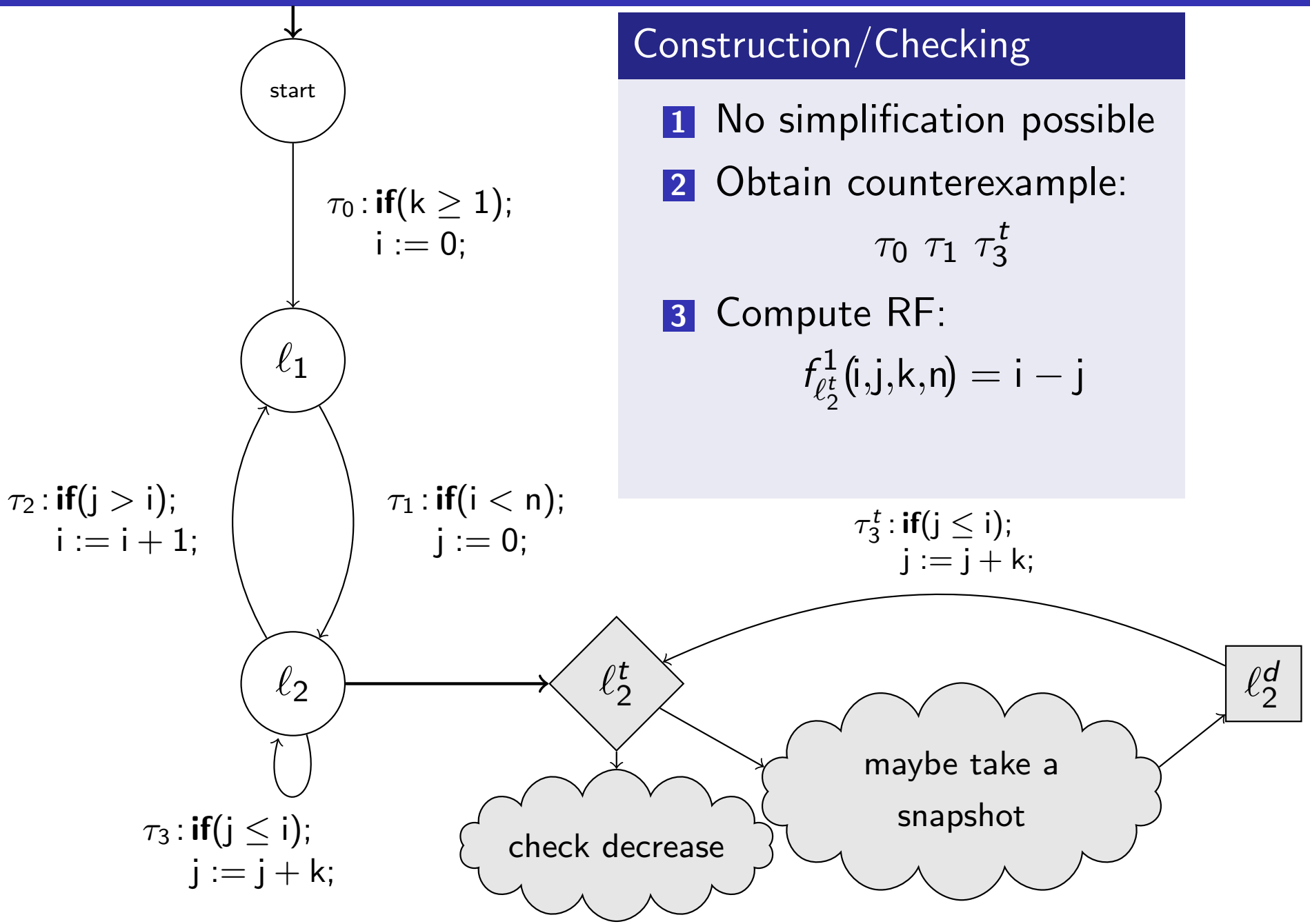


Construction/Checking

1. No simplification possible
2. Obtain counterexample:

$$\tau_0 \ \tau_1 \ \tau_3^t$$

Cooperation: Invariants



Construction/Checking

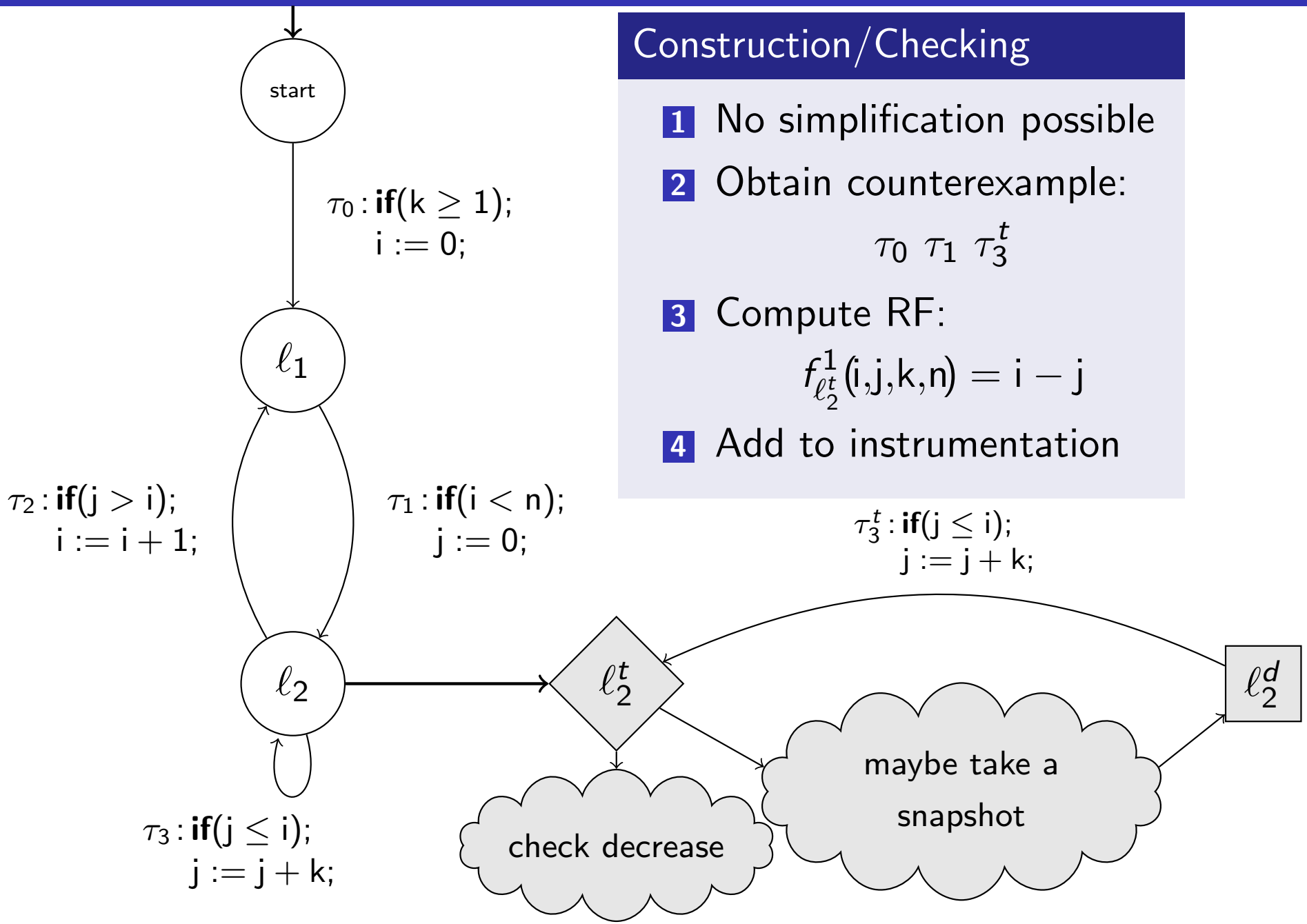
1. No simplification possible
2. Obtain counterexample:

$$\tau_0 \ \tau_1 \ \tau_3^t$$

3. Compute RF:

$$f^1_{\ell_2^t}(\mathrm{i},\mathrm{j},\mathrm{k},\mathrm{n}) = \mathrm{i} - \mathrm{j}$$

Cooperation: Invariants



Construction/Checking

1. No simplification possible
2. Obtain counterexample:

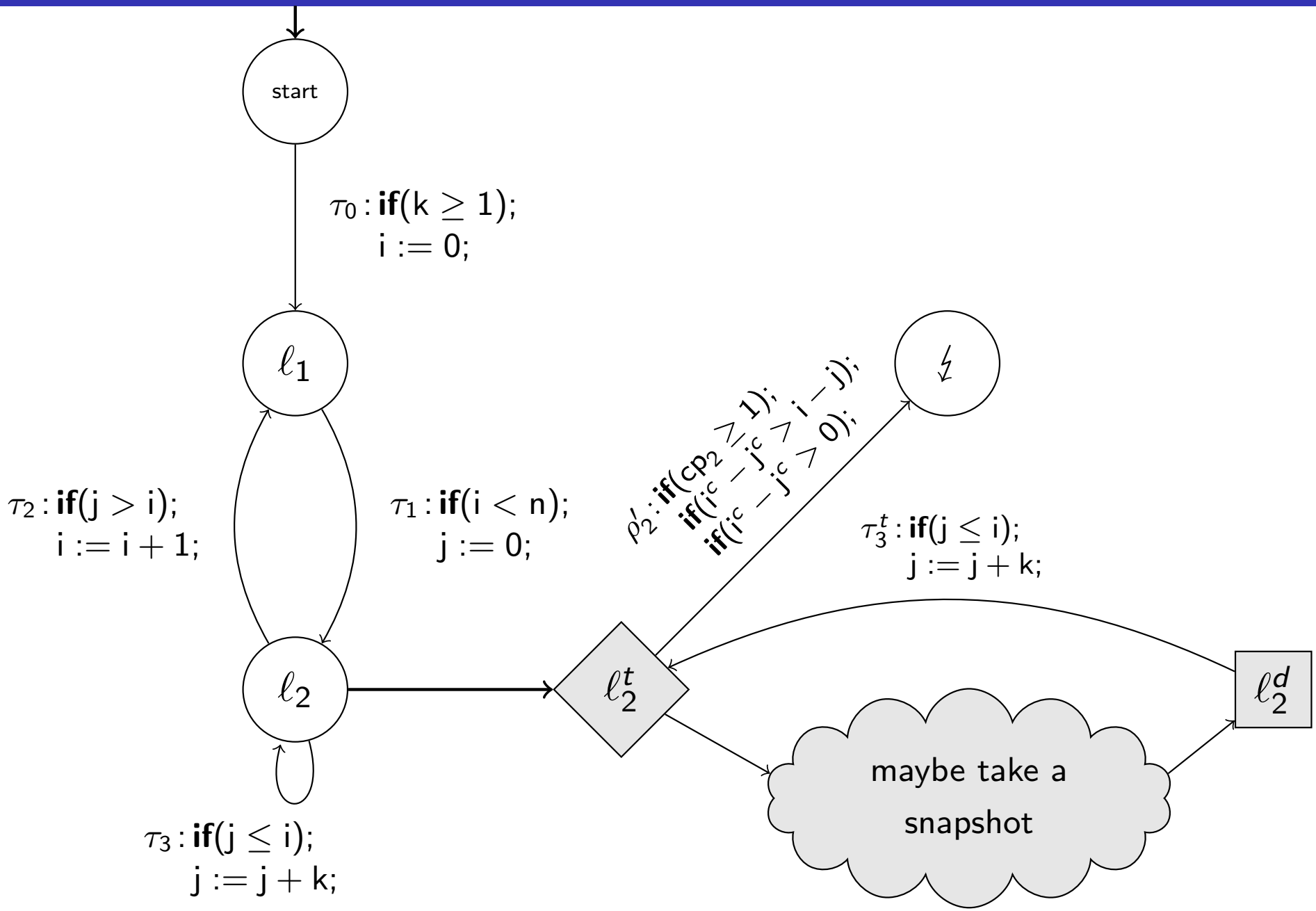
$$\tau_0 \ \tau_1 \ \tau_3^t$$

3. Compute RF:

$$f^1_{\ell_2^t}(\mathrm{i},\mathrm{j},\mathrm{k},\mathrm{n}) = \mathrm{i} - \mathrm{j}$$

4. Add to instrumentation

Cooperation: Invariants



Algorithm Refinement

$\mathcal{C} := \textsc{Instrument}(\mathcal{P})$

Algorithm Refinement

$\mathcal{C} := \textsc{Instrument}(\mathcal{P})$
for all $\mathcal{S}$ in $\mathrm{SCCs}(\textsc{Termination}(\mathcal{C}))$ **do**
  while $\exists$ $\mathcal{S}$-orienting rank function f **do**
    $\mathcal{C} := \mathcal{C} \setminus \textsc{Decreasing}(\mathcal{S}, f)$
    $\mathcal{S} := \mathcal{S} \setminus \textsc{Decreasing}(\mathcal{S}, f)$
  end while
end for

Algorithm Refinement

$\mathcal{C} := \textsc{Instrument}(\mathcal{P})$
for all $\mathcal{S}$ in $\mathrm{SCCs}(\textsc{Termination}(\mathcal{C}))$ **do**
  while $\exists$ $\mathcal{S}$-orienting rank function f **do**
    $\mathcal{C} := \mathcal{C} \setminus \textsc{Decreasing}(\mathcal{S}, f)$
    $\mathcal{S} := \mathcal{S} \setminus \textsc{Decreasing}(\mathcal{S}, f)$
  end while
end for
while $\exists$ counterexample $(\mathit{stem}, \mathit{cycle})$ in $\mathcal{C}$ **do**
  $\mathcal{S} := \mathrm{SCC\_Context}(\textsc{Termination}(\mathcal{C}), \mathit{cycle})$
  if $\exists$ $\mathcal{S}$-orienting rank function f **then**
    $\mathcal{C} := \mathcal{C} \setminus \textsc{Decreasing}(\mathcal{S}, f)$
    $\mathcal{C} := \textsc{Strengthen}(\mathcal{C}, \mathit{cycle}, f)$
  else if $\exists$ any rank function f for cycle **then**
    $\mathcal{C} := \textsc{Strengthen}(\mathcal{C}, \mathit{cycle}, f)$
  else
    return "Unknown"
  end if
end while
return "Terminating"

Cooperation: Evaluation

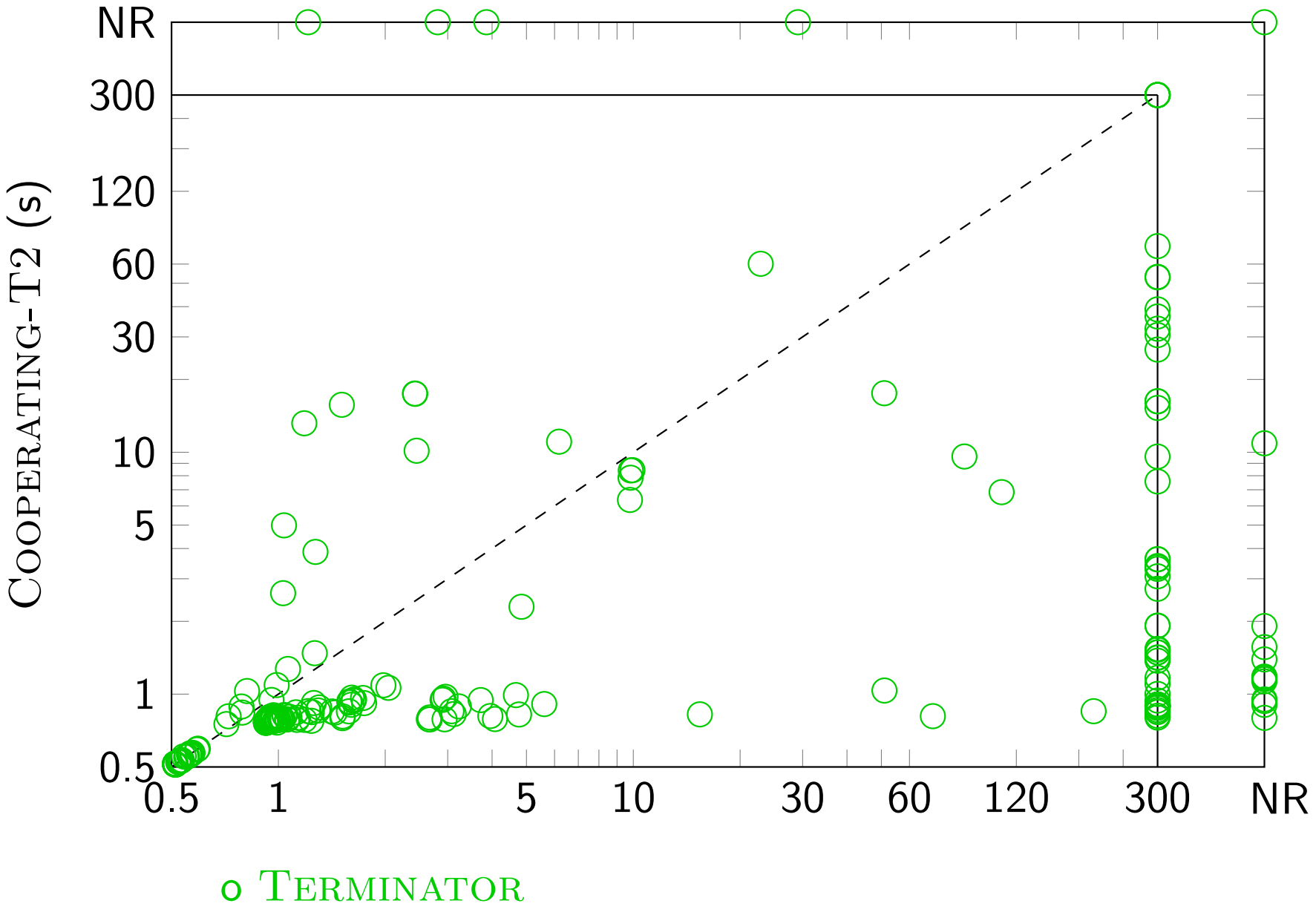
Evaluated on 449 termination proving benchmarks
260 known terminating, 181 known non-terminating, 8 unknown
Sources: Windows drivers, APACHE, POSTGRESQL, ...

Cooperation: Evaluation

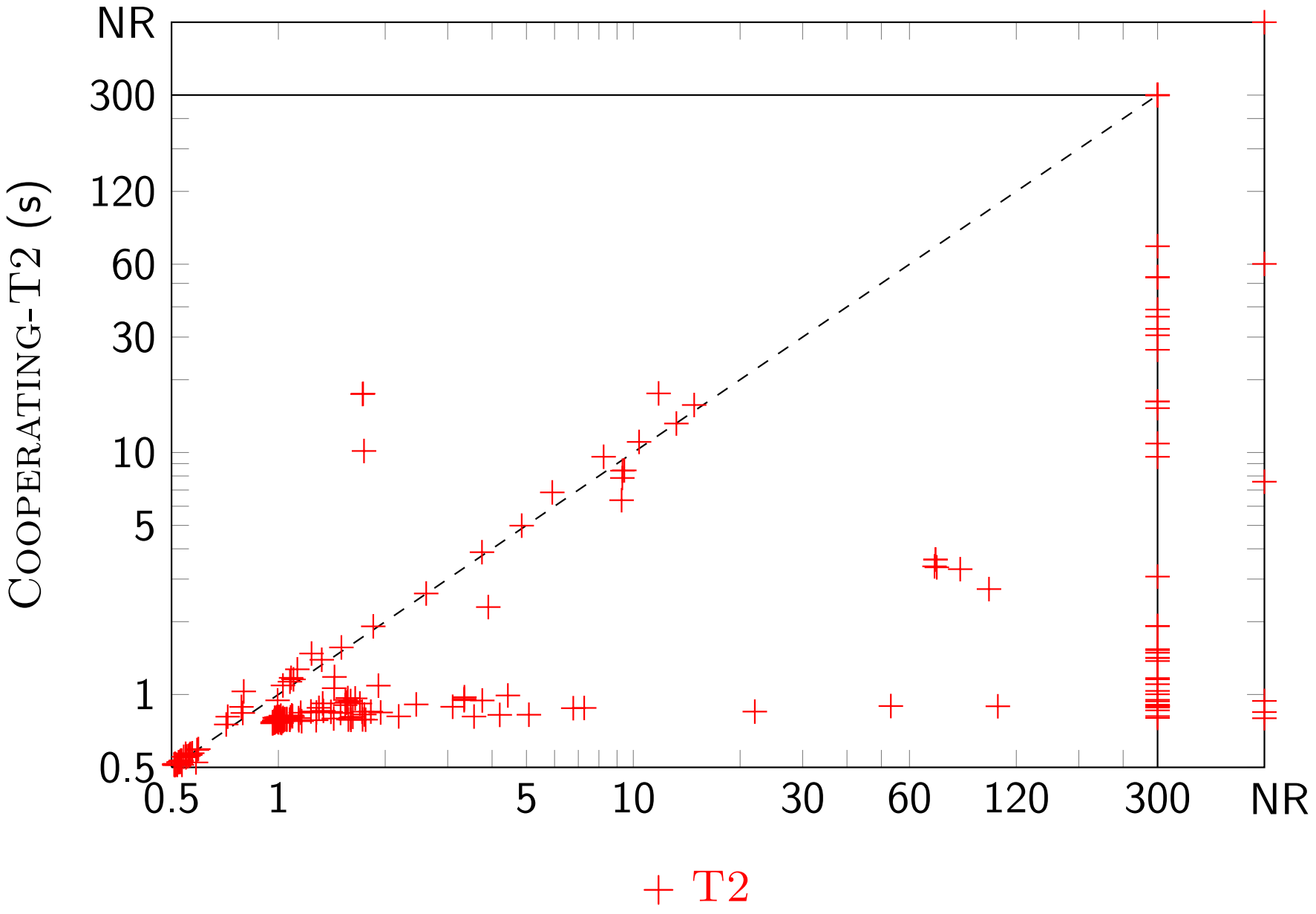
Evaluated on 449 termination proving benchmarks
260 known terminating, 181 known non-terminating, 8 unknown
Sources: Windows drivers, Apache, PostgreSQL, ...

	Term (#)	Term (avg. s)
Cooperating-T2	245	3.42
AProVE	197	2.21
KITTeL	196	4.65
T2	189	5.15
AProVE+Interproc	185	1.53
Terminator	177	4.99
Size-Change/MCNP	156	17.50
ARMC	138	16.16

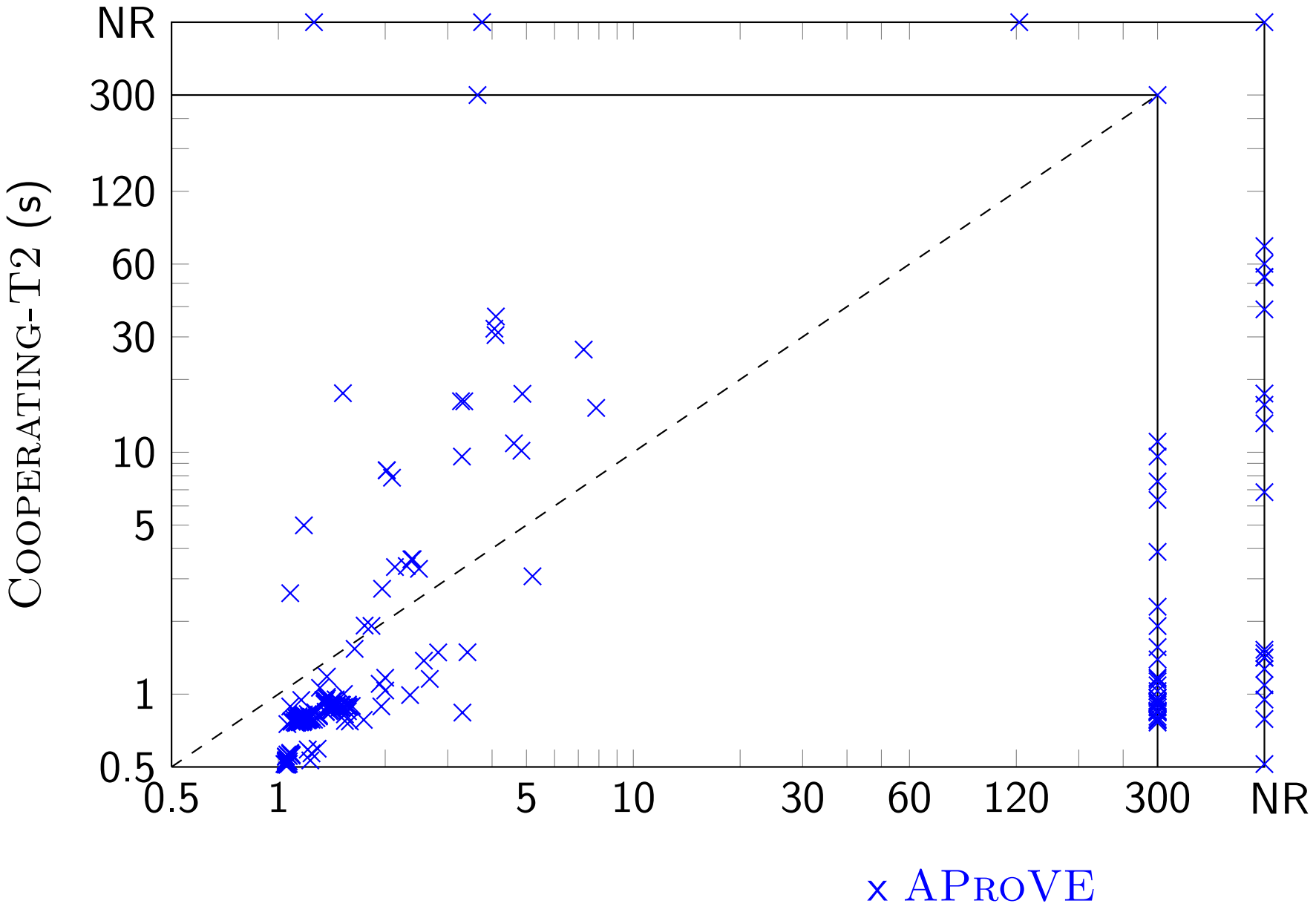
Cooperation: Evaluation



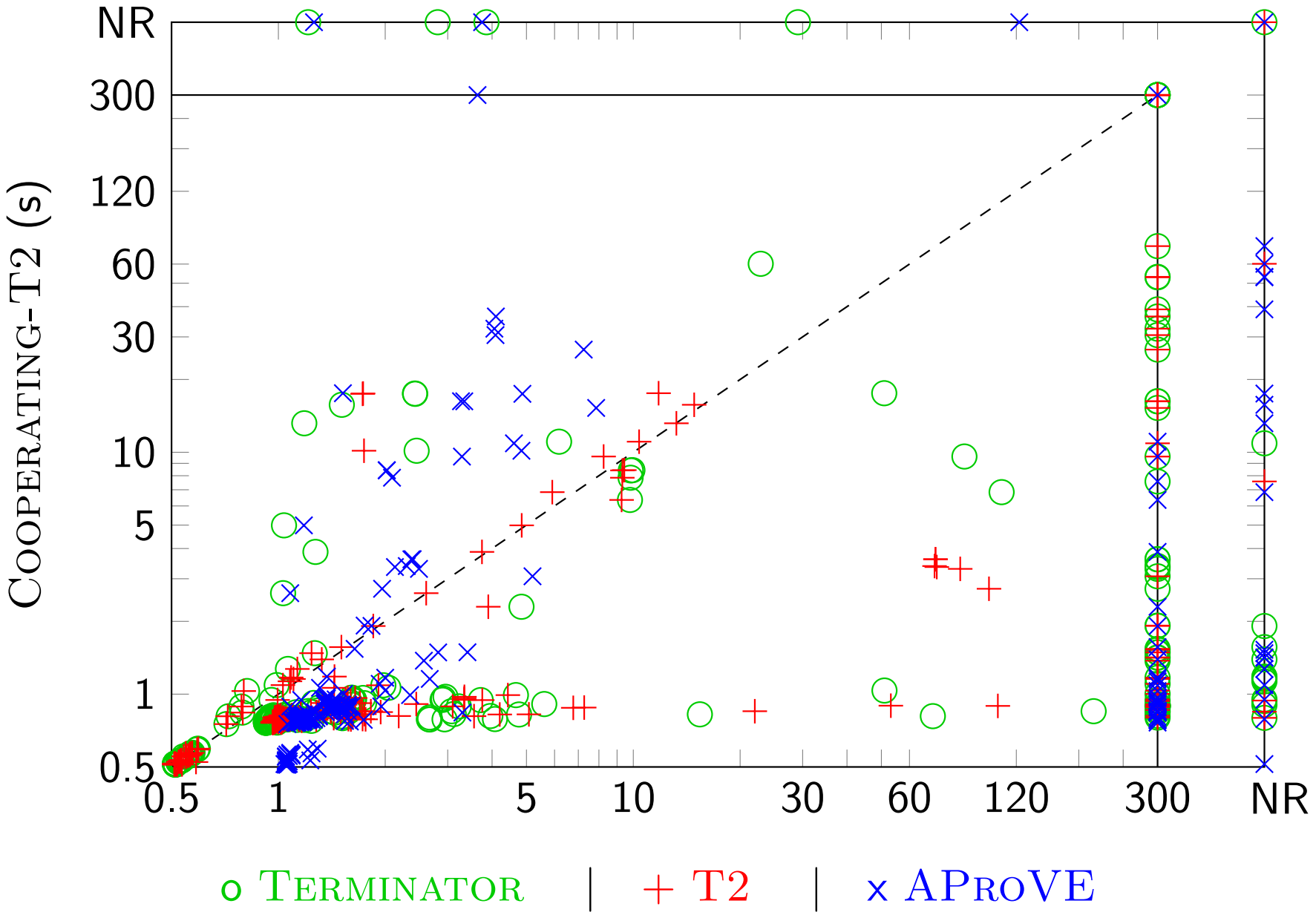
Cooperation: Evaluation



Cooperation: Evaluation



Cooperation: Evaluation



Cooperation: Evaluation

Evaluated on 449 termination proving benchmarks
260 known terminating, 181 known non-terminating, 8 unknown
Sources: Windows drivers, Apache, PostgreSQL, ...

	Term (#)	Term (avg. s)
Cooperating-T2	245	3.42
AProVE	197	2.21
KITTeL	196	4.65
T2	189	5.15
AProVE+Interproc	185	1.53
Terminator	177	4.99
Size-Change/MCNP	156	17.50
ARMC	138	16.16

Sources available: `http://research.microsoft.com/en-us/projects/t2/`

Beyond termination: Non-termination

Example (Non-Termination and reachability)

assume $(x \geq 0)$
while $x \neq 0$ **do**
    $x := x + k;$
done

- Loop non-terminating for $x \neq 0 \wedge k = 0$: Not reachable

Beyond termination: Non-termination

Example (Non-Termination and reachability)

assume $(x \geq 0)$
while $x \neq 0$ **do**
    $x := x + k;$
done

- Loop non-terminating for $x \neq 0 \wedge k = 0$: Not reachable
- Loop non-terminating for $x < 0 \wedge k < 0$: Not reachable

Beyond termination: Non-termination

Example (Non-Termination and reachability)

assume $(x \geq 0)$
while $x \neq 0$ **do**
    $x := x + k;$
done

- Loop non-terminating for $x \neq 0 \wedge k = 0$: Not reachable
- Loop non-terminating for $x < 0 \wedge k < 0$: Not reachable
- Loop non-terminating for $x > 0 \wedge k > 0$: Reachable

Non-termination: Basic definitions

Definition (Programs)

Program $P = (S, R, I)$

- S program states
- $R \subseteq S \times S$ program relation
- I initial states

Non-termination: Basic definitions

Definition (Programs)

Program $P = (S, R, I)$

- S program states
- $R \subseteq S \times S$ program relation
- I initial states

Definition (Recurrent set)

$\emptyset \neq G \subseteq S$ *recurrent*: $\forall s \in G. \exists s' \in G. (s, s') \in R^+$

Non-termination: Basic definitions

Definition (Programs)

Program $P = (S, R, I)$

- S program states
- $R \subseteq S \times S$ program relation
- I initial states

Definition (Recurrent set)

$\emptyset \neq G \subseteq S$ *recurrent*: $\forall s \in G. \exists s' \in G. (s, s') \in R^+$

Lemma (Non-termination of programs)

P non-terminating if and only if

- *Recurrence: G recurrent set for P*
- *Reachability: $(i, s) \in R^+$, $i \in I$, $s \in G$*

Termination & Non-termination are related

Example (Termination and invariants)

assume $(k < 0)$
while $x > 0$ **do**
    $x := x + k;$
done

Termination & Non-termination are related

Example (Termination and invariants)

assume $(k < 0)$
while $x > 0$ **do**
    $x := x + k;$
done

- RF x requires invariant $k < 0$

Termination & Non-termination are related

Example (Termination and invariants)

assume $(k < 0)$
while $x > 0$ **do**
    $x := x + k;$
done

- RF x requires invariant $k < 0$
- Recurrent set $x > 0 \wedge k \geq 0$

Termination & Non-termination are related

Example (Termination and invariants)

assume $(k < 0)$
while $x > 0$ **do**
    $x := x + k;$
done

- RF x requires invariant $k < 0$
- Recurrent set $x > 0 \wedge k \geq 0$ *not reachable*

Termination & Non-termination are related

Example (Termination and invariants)

assume $(k < 0)$
while $x > 0$ **do**
    $x := x + k;$
done

- RF x requires invariant $k < 0$
- Recurrent set $x > 0 \wedge k \geq 0$ *not reachable*
  Reason: $k < 0$ is invariant

Termination & Non-termination are related

Example (Termination and invariants)

assume $(k < 0)$
while $x > 0 \wedge k < 0$ **do**
    $x := x + k;$
done

- RF x requires invariant $k < 0$
- Recurrent set $x > 0 \wedge k \geq 0$ *not reachable*
  Reason: $k < 0$ is invariant

Alternating Termination & Non-termination

Algorithm Refinement

$G = \textsc{RecurrentSet}(R)$
if $\textsc{Reachable}(G \text{ in } P)$ **then**
  return "Non-terminating"
else
  $P = \textsc{Strengthen}(P \text{ with invariant})$
end if

Alternating Termination & Non-termination

Algorithm Refinement

$G = \textsc{RecurrentSet}(R)$
if $\textsc{Reachable}(G \text{ in } P)$ **then**
  return "Non-terminating"
else
  $P = \textsc{Strengthen}(P \text{ with invariant})$
end if
if $\textsc{Termination}(P)$ **then**
  return "Terminating"
else
  $P = \textsc{Simplify}(P)$
end if

Alternating Termination & Non-termination

Algorithm Refinement

while *true* **do**
  $G = \textsc{RecurrentSet}(R)$
  if $\textsc{Reachable}(G \text{ in } P)$ **then**
    return "Non-terminating"
  else
    $P = \textsc{Strengthen}(P \text{ with invariant})$
  end if
  if $\textsc{Termination}(P)$ **then**
    return "Terminating"
  else
    $P = \textsc{Simplify}(P)$
  end if
end while