Inferring Expected Runtimes for Probabilistic Integer Programs Using Expected Sizes

27th Conference on Tools and Algorithms for the Construction and Analysis of Systems

Fabian Meyer **Marcel Hark** Jürgen Giesl

Randomization in Programming

Randomization in Programming

- Recently: growing interest in randomization in programming.

Randomization in Programming

- Recently: growing interest in randomization in programming.
- Extension of classical programs by probability distributions.

Randomization in Programming

- Recently: growing interest in randomization in programming.
- Extension of classical programs by probability distributions.

  $\rightarrow$ efficiency of algorithms, cryptography, ...

Introduction

Randomization in Programming

- Recently: growing interest in randomization in programming.
- Extension of classical programs by probability distributions.
  $\rightarrow$ efficiency of algorithms, cryptography, ...

$$
\begin{array}{l}
\texttt{while}\,(\,x > 0\,)\,\{\\
\quad x \leftarrow x - 1\\
\}
\end{array}
$$

Randomization in Programming

- Recently: growing interest in randomization in programming.
- Extension of classical programs by probability distributions.
  $\rightarrow$ efficiency of algorithms, cryptography, ...

$$
\begin{array}{l}
\texttt{while}\,(\,x > 0\,)\,\{\\
\quad \{\,x \leftarrow x\,\}\ \left[\tfrac{1}{2}\right]\ \{\,x \leftarrow x - 1\,\}\\
\}
\end{array}
$$

Introduction

Randomization in Programming

- Recently: growing interest in randomization in programming.
- Extension of classical programs by probability distributions.
  $\rightarrow$ efficiency of algorithms, cryptography, ...
- Termination behavior diversifies.

$$
\begin{array}{l}
\texttt{while}\,(\,x > 0\,)\,\{ \\
\quad \{\, x \leftarrow x \,\}\ \left[\tfrac{1}{2}\right]\ \{\, x \leftarrow x - 1 \,\} \\
\}
\end{array}
$$

Randomization in Programming

- Recently: growing interest in randomization in programming.
- Extension of classical programs by probability distributions.
  $\to$ efficiency of algorithms, cryptography, ...
- Termination behavior diversifies.
  $\to$ Measure of efficiency: expected runtime.

$$\mathtt{while}\,(\,x > 0\,)\,\{$$
$$\quad \{\,x \leftarrow x\,\}\ \left[\tfrac{1}{2}\right]\ \{\,x \leftarrow x - 1\,\}$$
$$\}$$

Introduction

Randomization in Programming

- Recently: growing interest in randomization in programming.
- Extension of classical programs by probability distributions.

  $\rightarrow$ efficiency of algorithms, cryptography, ...
- Termination behavior diversifies.

  $\rightarrow$ Measure of efficiency: expected runtime.

  $\rightarrow$ How to infer upper bounds on expected runtime fully automatically?

$$
\begin{array}{l}
\texttt{while}\,(x > 0)\,\{\\
\quad \{x \leftarrow x\}\ [\tfrac{1}{2}]\ \{x \leftarrow x - 1\}\\
\}
\end{array}
$$

Linear Probabilistic Ranking Functions (LPRF)

Linear Probabilistic Ranking Functions (LPRF)

- Automatic complexity analysis of classical programs:

Linear Probabilistic Ranking Functions (LPRF)

- Automatic complexity analysis of classical programs:
  $\rightarrow$ Linear ranking functions.

Linear Probabilistic Ranking Functions (LPRF)

- Automatic complexity analysis of classical programs:
  $\rightarrow$ Linear ranking functions.
- In case of randomization: linear probabilistic ranking functions.

Linear Probabilistic Ranking Functions (LPRF)

- Automatic complexity analysis of classical programs:
  → Linear ranking functions.
- In case of randomization: linear probabilistic ranking functions.

$$\texttt{while}\,(x > 0)\,\{\\ \quad \{x \leftarrow x\}\ [\tfrac{1}{2}]\ \{x \leftarrow x - 1\}\\ \}$$

Linear Probabilistic Ranking Functions (LPRF)

- Automatic complexity analysis of classical programs:
  $\rightarrow$ Linear ranking functions.
- In case of randomization: linear probabilistic ranking functions.

  Consider $\mathfrak{r} = 2 \cdot x$.

$$
\begin{array}{l}
\texttt{while}\,(\,x > 0\,)\,\{\\
\quad \{\,x \leftarrow x\,\}\ [\tfrac{1}{2}]\ \{\,x \leftarrow x - 1\,\}\\
\}
\end{array}
$$

Linear Probabilistic Ranking Functions (LPRF)

- Automatic complexity analysis of classical programs:

  $\rightarrow$ Linear ranking functions.

- In case of randomization: linear probabilistic ranking functions.

  Consider $\mathfrak{r} = 2 \cdot x$.

  Whenever loop can be entered: $\mathfrak{r} > 0$.

$$\texttt{while}\,(x > 0)\,\{\ \{x \leftarrow x\}\ \left[\tfrac{1}{2}\right]\ \{x \leftarrow x - 1\}\ \}$$

Linear Probabilistic Ranking Functions (LPRF)

- Automatic complexity analysis of classical programs:
  $\rightarrow$ Linear ranking functions.
- In case of randomization: linear probabilistic ranking functions.

  Consider $\mathfrak{r} = 2 \cdot x$.

  Whenever loop can be entered: $\mathfrak{r} > 0$.

  In one loop iteration $\mathfrak{r}$ is expected to decrease by 1 in each iteration.

$$\texttt{while}\,(x > 0)\,\{ \quad \{x \leftarrow x\}\ \left[\tfrac{1}{2}\right]\ \{x \leftarrow x - 1\} \quad \}$$

Linear Probabilistic Ranking Functions (LPRF)

- Automatic complexity analysis of classical programs:

  $\to$ Linear ranking functions.

- In case of randomization: linear probabilistic ranking functions.

  Consider $\mathfrak{r} = 2 \cdot x$.

  Whenever loop can be entered: $\mathfrak{r} > 0$.

  In one loop iteration $\mathfrak{r}$ is expected to decrease by 1 in each iteration.

$$\tfrac{1}{2} \cdot \mathfrak{r}\,[x/x] + \tfrac{1}{2} \cdot \mathfrak{r}\,[x/x-1] = 2 \cdot x - 1 = \mathfrak{r} - 1$$

$$\begin{array}{l} \texttt{while}\,(x > 0)\,\{ \\ \quad \{x \leftarrow x\}\ [\tfrac{1}{2}]\ \{x \leftarrow x - 1\} \\ \} \end{array}$$

Linear Probabilistic Ranking Functions (LPRF)

- Automatic complexity analysis of classical programs:
  $\rightarrow$ Linear ranking functions.
- In case of randomization: linear probabilistic ranking functions.

  Consider $\mathfrak{r} = 2 \cdot x$.

  Whenever loop can be entered: $\mathfrak{r} > 0$.

  In one loop iteration $\mathfrak{r}$ is expected to decrease by 1 in each iteration.

$$\tfrac{1}{2} \cdot \mathfrak{r}\,[x/x] + \tfrac{1}{2} \cdot \mathfrak{r}\,[x/x-1] = 2 \cdot x - 1 = \mathfrak{r} - 1$$

  Expected runtime of loop: at most $\mathfrak{r} = 2 \cdot x$. (e.g., [Bournez & Garnier '05]).

```
while (x > 0) {
    {x ← x} [1/2] {x ← x − 1}
}
```

Linear Probabilistic Ranking Functions (LPRF)

$$\texttt{while}\,(\,x > 0\,)\,\{$$
$$\quad \{\,x \leftarrow x\,\}\ \left[\tfrac{1}{2}\right]\ \{\,x \leftarrow x - 1\,\}$$
$$\}$$

Linear Probabilistic Ranking Functions (LPRF)

- What about larger programs?

$$
\begin{array}{l}
\texttt{while}\,(x > 0)\,\{\\
\quad \{x \leftarrow x\}\;[\tfrac{1}{2}]\;\{x \leftarrow x - 1\}\\
\}
\end{array}
$$

Linear Probabilistic Ranking Functions (LPRF)

- What about larger programs?

$$
\begin{array}{l}
\texttt{while}\,(x > 0)\,\{\\
\quad \left\{ \begin{array}{l} y \leftarrow y \\ x \leftarrow x \end{array} \right\} \left[\tfrac{1}{2}\right] \left\{ \begin{array}{l} y \leftarrow y + x \\ x \leftarrow x - 1 \end{array} \right\}\\
\}\\
\texttt{while}\,(y > 0)\,\{\\
\quad y \leftarrow y - 1\\
\}
\end{array}
$$

Linear Probabilistic Ranking Functions (LPRF)

- What about larger programs?
- Value of y grows in first loop.

$$
\begin{array}{l}
\texttt{while}\,(x > 0)\,\{\\
\quad \left\{ \begin{array}{l} y \leftarrow y \\ x \leftarrow x \end{array} \right\} \left[\frac{1}{2}\right] \left\{ \begin{array}{l} y \leftarrow y + x \\ x \leftarrow x - 1 \end{array} \right\}\\
\}\\
\texttt{while}\,(y > 0)\,\{\\
\quad y \leftarrow y - 1\\
\}
\end{array}
$$

Linear Probabilistic Ranking Functions (LPRF)

- What about larger programs?
- Value of y grows in first loop.
  - $\rightarrow$ Cannot bound expected runtime with single LPRF.

$$
\begin{array}{l}
\texttt{while}\,(\,x > 0\,)\,\{ \\
\quad \left\{ \begin{array}{l} y \leftarrow y \\ x \leftarrow x \end{array} \right\} \left[\tfrac{1}{2}\right] \left\{ \begin{array}{l} y \leftarrow y + x \\ x \leftarrow x - 1 \end{array} \right\} \\
\} \\
\texttt{while}\,(\,y > 0\,)\,\{ \\
\quad y \leftarrow y - 1 \\
\}
\end{array}
$$

Linear Probabilistic Ranking Functions (LPRF)

- What about larger programs?
- Value of y grows in first loop.
  $\rightarrow$ Cannot bound expected runtime with single LPRF.
- Still, y is LPRF for the (standalone) second loop.

$$
\begin{array}{l}
\texttt{while}\,(x > 0)\,\{\\
\quad \left\{ \begin{array}{l} y \leftarrow y \\ x \leftarrow x \end{array} \right\} \left[\tfrac{1}{2}\right] \left\{ \begin{array}{l} y \leftarrow y + x \\ x \leftarrow x - 1 \end{array} \right\}\\
\}\\
\texttt{while}\,(y > 0)\,\{\\
\quad y \leftarrow y - 1\\
\}
\end{array}
$$

Linear Probabilistic Ranking Functions (LPRF)

- What about larger programs?
- Value of y grows in first loop.
  $\rightarrow$ Cannot bound expected runtime with single LPRF.
- Still, y is LPRF for the (standalone) second loop.
- Expected runtime of full program (intuitively):

$$
\begin{array}{l}
\texttt{while}\,(x > 0)\,\{ \\
\quad \left\{ \begin{array}{l} y \leftarrow y \\ x \leftarrow x \end{array} \right\} [\tfrac{1}{2}] \left\{ \begin{array}{l} y \leftarrow y + x \\ x \leftarrow x - 1 \end{array} \right\} \\
\} \\
\texttt{while}\,(y > 0)\,\{ \\
\quad y \leftarrow y - 1 \\
\}
\end{array}
$$

Linear Probabilistic Ranking Functions (LPRF)

- What about larger programs?
- Value of y grows in first loop.
  $\rightarrow$ Cannot bound expected runtime with single LPRF.
- Still, y is LPRF for the (standalone) second loop.
- Expected runtime of full program (intuitively):
  $\rightarrow 2 \cdot x +$ "expected size" (y).

$$\texttt{while}\,(x > 0)\,\{$$
$$\left\{\begin{array}{l} y \leftarrow y \\ x \leftarrow x \end{array}\right\} \left[\tfrac{1}{2}\right] \left\{\begin{array}{l} y \leftarrow y + x \\ x \leftarrow x - 1 \end{array}\right\}$$
$$\}$$
$$\texttt{while}\,(y > 0)\,\{$$
$$y \leftarrow y - 1$$
$$\}$$

Linear Probabilistic Ranking Functions (LPRF)

- What about larger programs?
- Value of y grows in first loop.
  $\rightarrow$ Cannot bound expected runtime with single LPRF.
- Still, y is LPRF for the (standalone) second loop.
- Expected runtime of full program (intuitively):
  $\rightarrow 2 \cdot x +$ "expected size" (y).
- Computation of runtimes via sizes:

$$
\begin{array}{l}
\texttt{while}\,(x > 0)\,\{ \\
\quad \left\{ \begin{array}{l} y \leftarrow y \\ x \leftarrow x \end{array} \right\} \left[\tfrac{1}{2}\right] \left\{ \begin{array}{l} y \leftarrow y + x \\ x \leftarrow x - 1 \end{array} \right\} \\
\} \\
\texttt{while}\,(y > 0)\,\{ \\
\quad y \leftarrow y - 1 \\
\}
\end{array}
$$

Linear Probabilistic Ranking Functions (LPRF)

- What about larger programs?
- Value of y grows in first loop.
  - $\rightarrow$ Cannot bound expected runtime with single LPRF.
- Still, y is LPRF for the (standalone) second loop.
- Expected runtime of full program (intuitively):
  - $\rightarrow 2 \cdot x +$ "expected size" (y).
- Computation of runtimes via sizes:
  - $\rightarrow$ very successful for classical programs [Giesl et al. '16].

$$
\begin{array}{l}
\texttt{while}\,(x > 0)\,\{\\
\quad \left\{ \begin{array}{l} y \leftarrow y \\ x \leftarrow x \end{array} \right\} \left[\tfrac{1}{2}\right] \left\{ \begin{array}{l} y \leftarrow y + x \\ x \leftarrow x - 1 \end{array} \right\}\\
\}\\
\texttt{while}\,(y > 0)\,\{\\
\quad y \leftarrow y - 1\\
\}
\end{array}
$$

Linear Probabilistic Ranking Functions (LPRF)

- What about larger programs?
- Value of y grows in first loop.
  $\rightarrow$ Cannot bound expected runtime with single LPRF.
- Still, y is LPRF for the (standalone) second loop.
- Expected runtime of full program (intuitively):
  $\rightarrow 2 \cdot x +$ "expected size" (y).
- Computation of runtimes via sizes:
  $\rightarrow$ very successful for classical programs [Giesl et al. '16].
- **Contribution**: Novel **modular** approach by combining expected runtimes and expected sizes.

$$
\begin{array}{l}
\texttt{while}\,(x > 0)\,\{ \\
\quad \left\{ \begin{array}{l} y \leftarrow y \\ x \leftarrow x \end{array} \right\} \left[\tfrac{1}{2}\right] \left\{ \begin{array}{l} y \leftarrow y + x \\ x \leftarrow x - 1 \end{array} \right\} \\
\} \\
\texttt{while}\,(y > 0)\,\{ \\
\quad y \leftarrow y - 1 \\
\}
\end{array}
$$

Probabilistic Integer Transition Systems

Probabilistic Integer Transition Systems

- We denote programs by graphs to capture control flow.

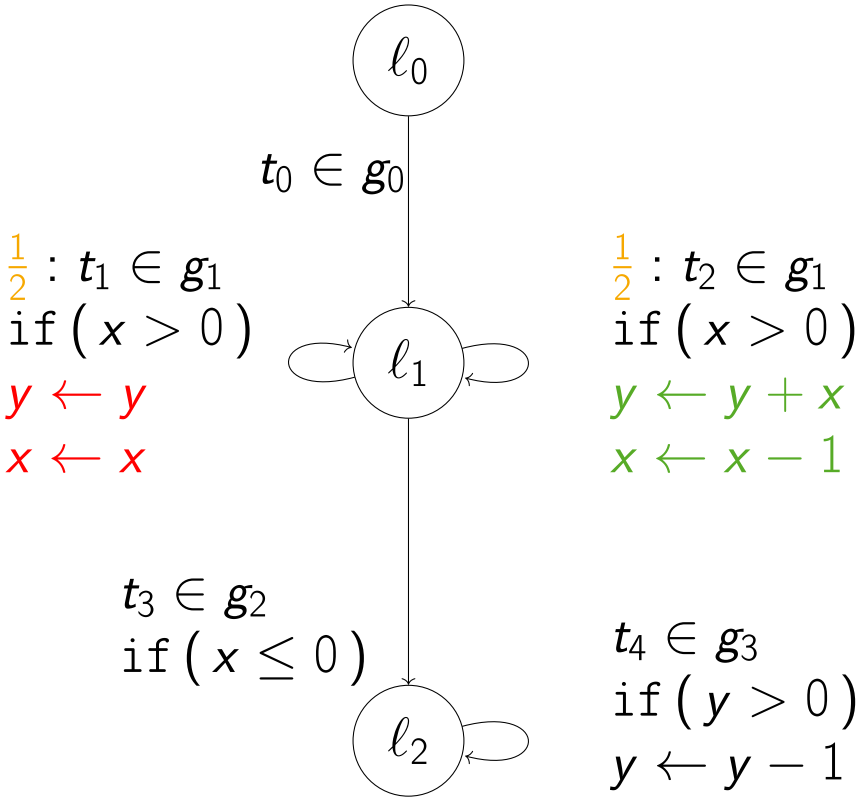
Probabilistic Integer Transition Systems

- We denote programs by graphs to capture control flow.

$$
\begin{array}{l}
\texttt{while}\,(x > 0)\,\{ \\
\quad \left\{ \begin{array}{c} y \leftarrow y \\ x \leftarrow x \end{array} \right\} \left[\tfrac{1}{2}\right] \left\{ \begin{array}{c} y \leftarrow y + x \\ x \leftarrow x - 1 \end{array} \right\} \\
\} \\
\texttt{while}\,(y > 0)\,\{ \\
\quad y \leftarrow y - 1 \\
\}
\end{array}
$$

Probabilistic Integer Transition Systems

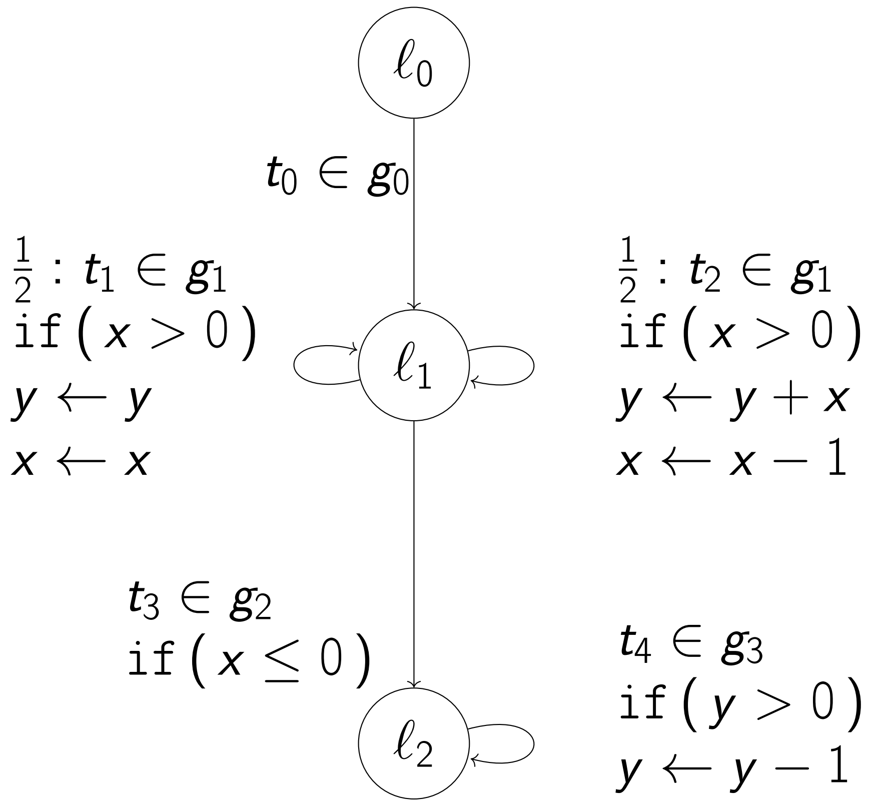
- We denote programs by graphs to capture control flow.



$$
\begin{array}{l}
\ell_1 : \ \texttt{while}\,(\,x > 0\,)\,\{ \\
\qquad \left\{ \begin{array}{l} y \leftarrow y \\ x \leftarrow x \end{array} \right\} \left[\tfrac{1}{2}\right] \left\{ \begin{array}{l} y \leftarrow y + x \\ x \leftarrow x - 1 \end{array} \right\} \\
\quad \} \\
\ell_2 : \ \texttt{while}\,(\,y > 0\,)\,\{ \\
\qquad y \leftarrow y - 1 \\
\quad \}
\end{array}
$$

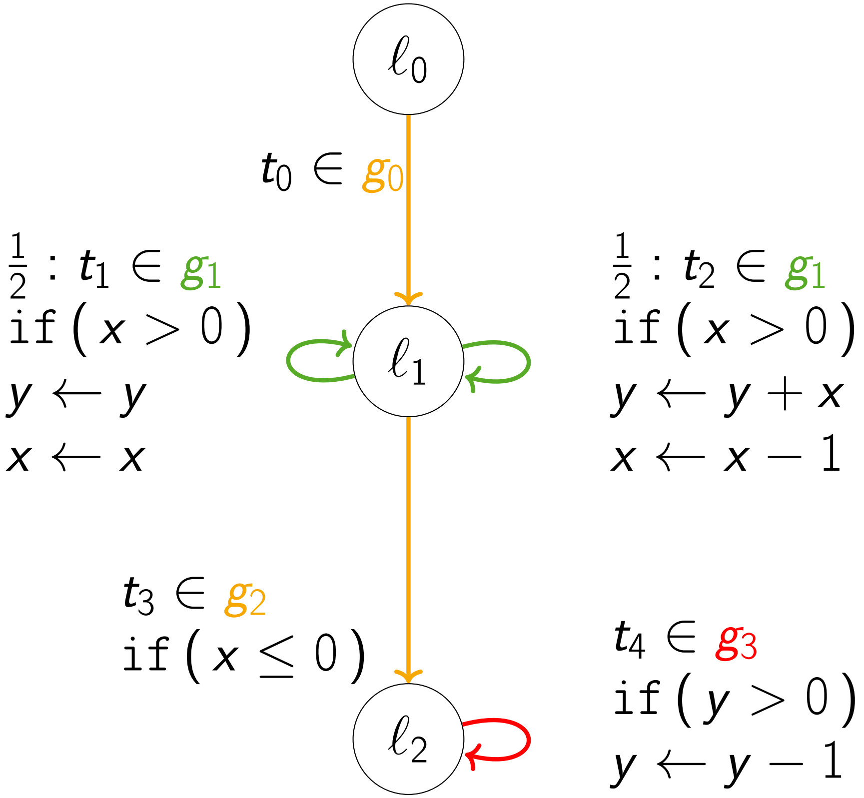
Outline

Overall Idea



Overall Idea

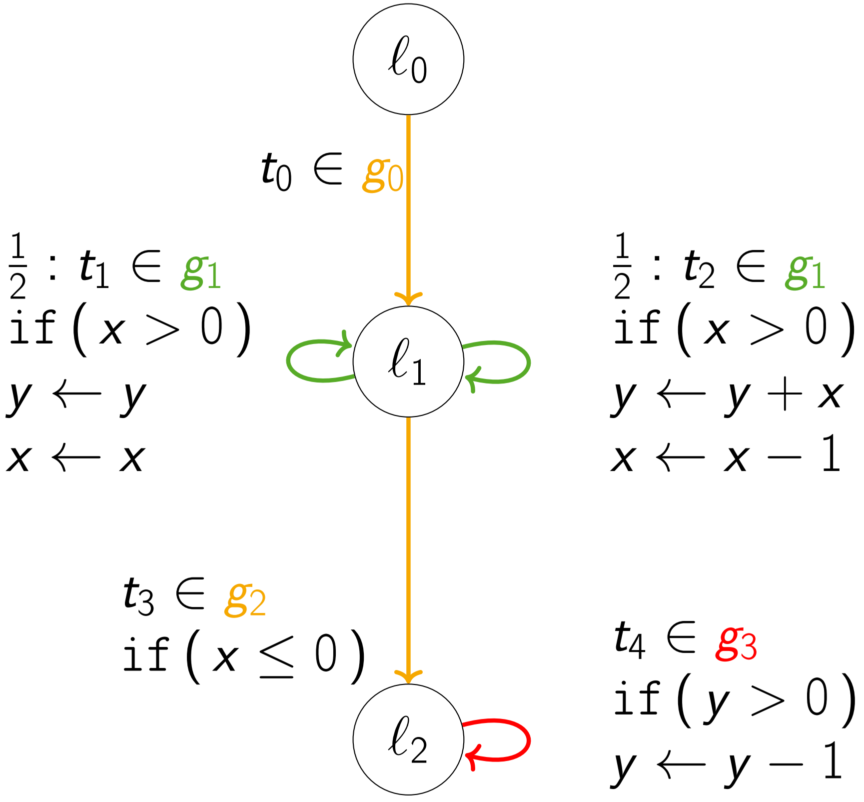
- For each general transition $g \in \mathcal{GT}$:



Computing Expected Runtime Bounds

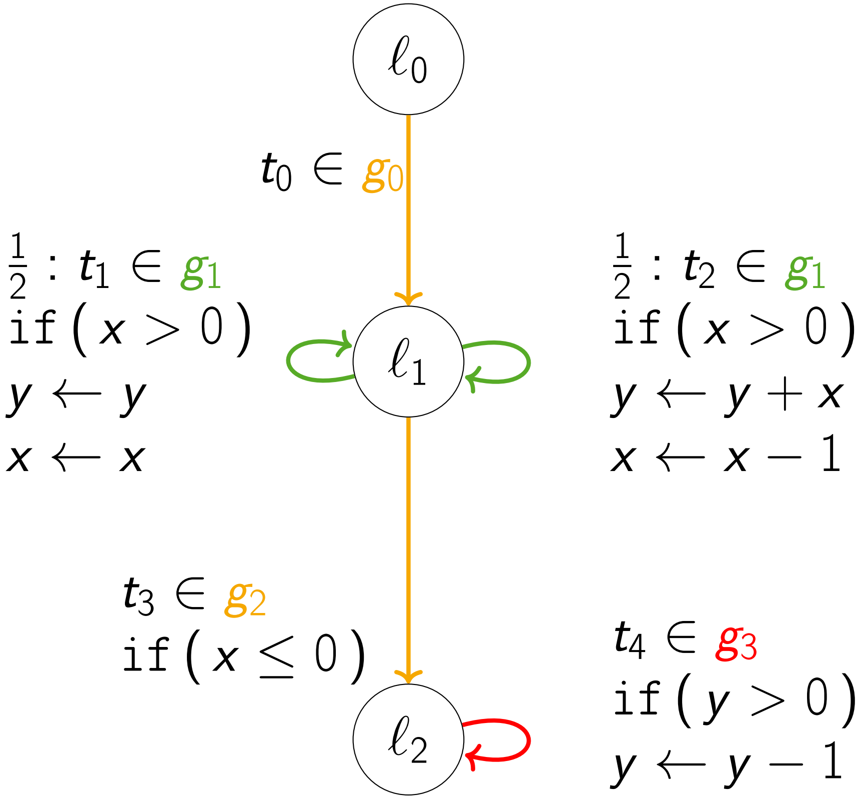
Overall Idea

- For each general transition $g \in \mathcal{GT}$:
  $\mathcal{R}(g) =$ number of executions of g in run of full program.



Overall Idea

- For each general transition $g \in \mathcal{GT}$:
  $\mathcal{R}(g) =$ number of executions of g in run of full program.
- $\sum_{g \in \mathcal{GT}} \mathbb{E}(\mathcal{R}(g))$ is expected runtime of full program.



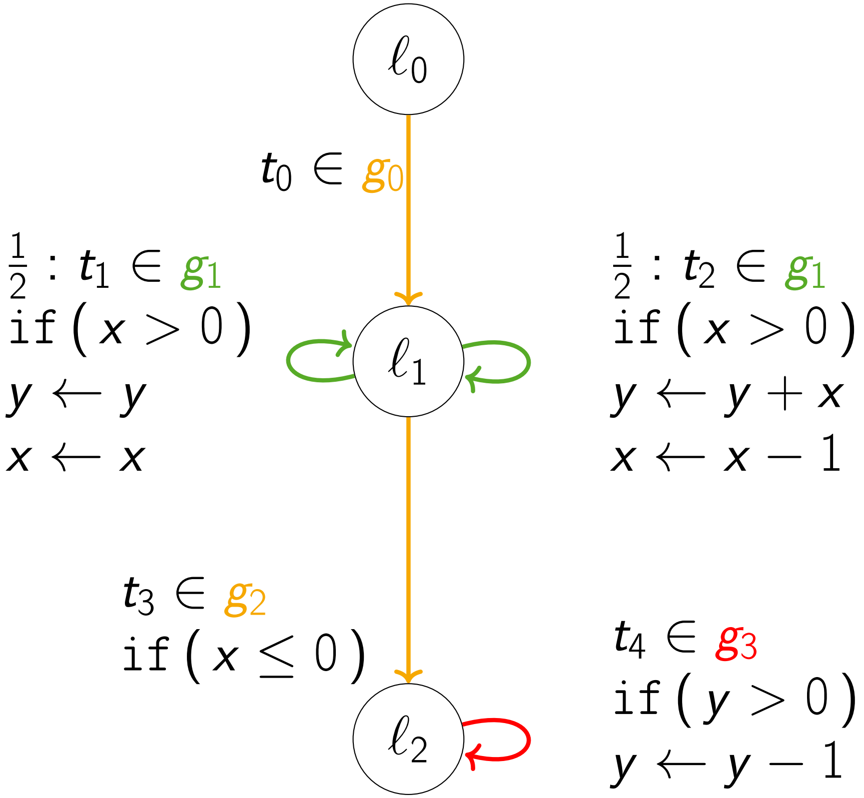
Computing Expected Runtime Bounds

Overall Idea

- For each general transition $g \in \mathcal{GT}$:

  $\mathcal{R}(g) =$ number of executions of g in run of full program.
- $\sum_{g \in \mathcal{GT}} \mathbb{E}(\mathcal{R}(g))$ is expected runtime of full program.

  $\rightarrow$ Over-approximate $\mathbb{E}(\mathcal{R}(g))$ for each $g \in \mathcal{GT}$.



Computing Expected Runtime Bounds

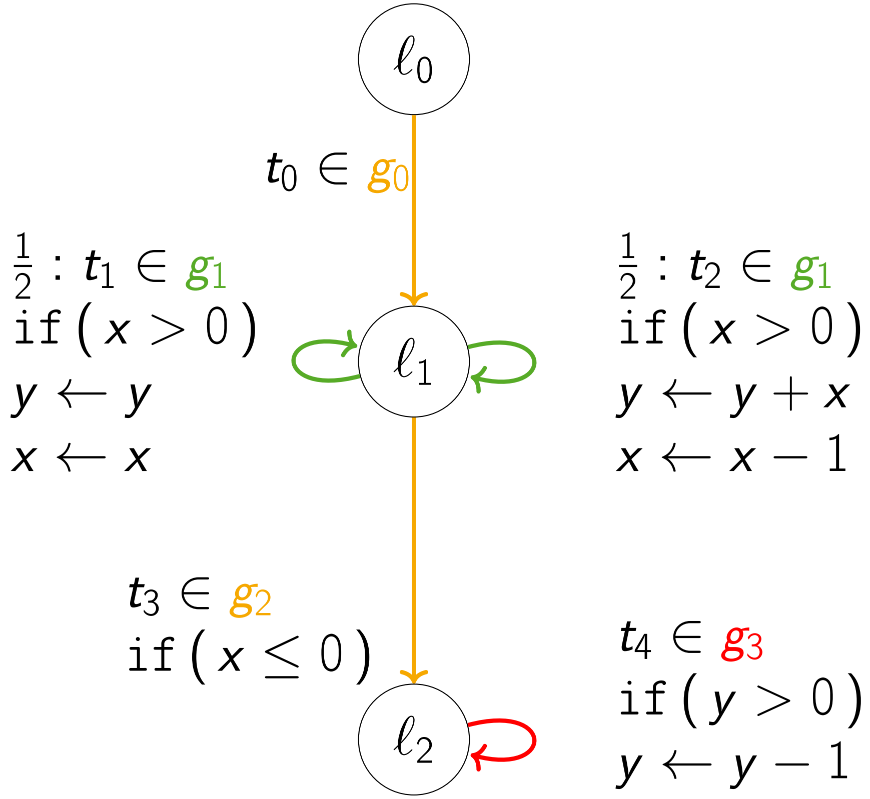
Overall Idea

- For each general transition $g \in \mathcal{GT}$:

  $\mathcal{R}(g) =$ number of executions of g in run of full program.
- $\sum_{g \in \mathcal{GT}} \mathbb{E}(\mathcal{R}(g))$ is expected runtime of full program.

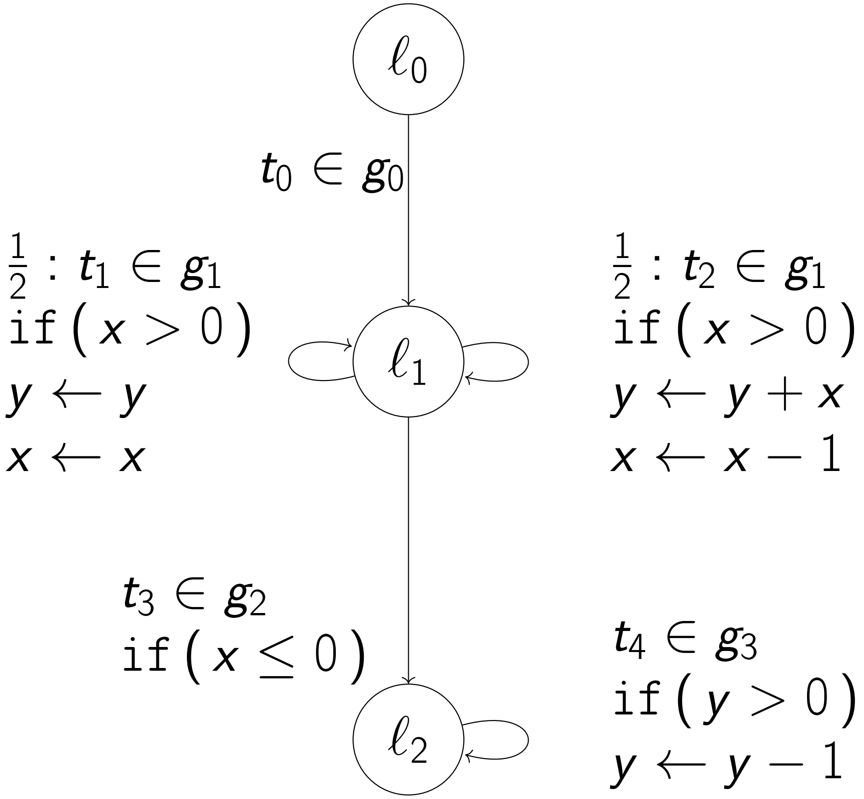
  $\rightarrow$ Over-approximate $\mathbb{E}(\mathcal{R}(g))$ for each $g \in \mathcal{GT}$.

  $\rightarrow$ Contribution: **Modular** inference of bound on $\mathbb{E}(\mathcal{R}(g))$.



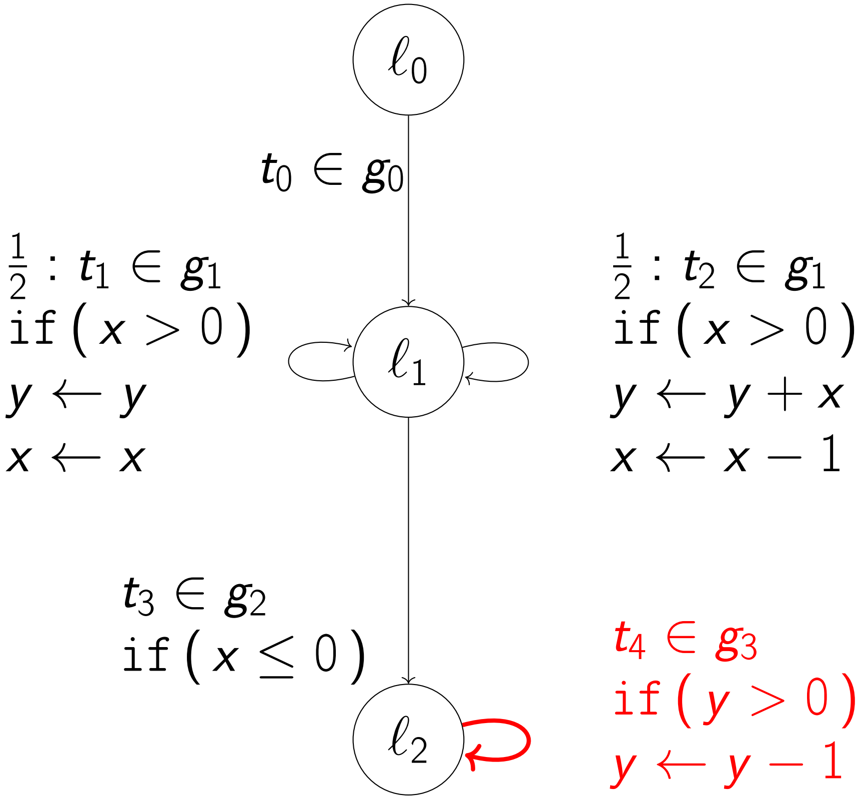
Computing Expected Runtime Bounds

Overall Idea



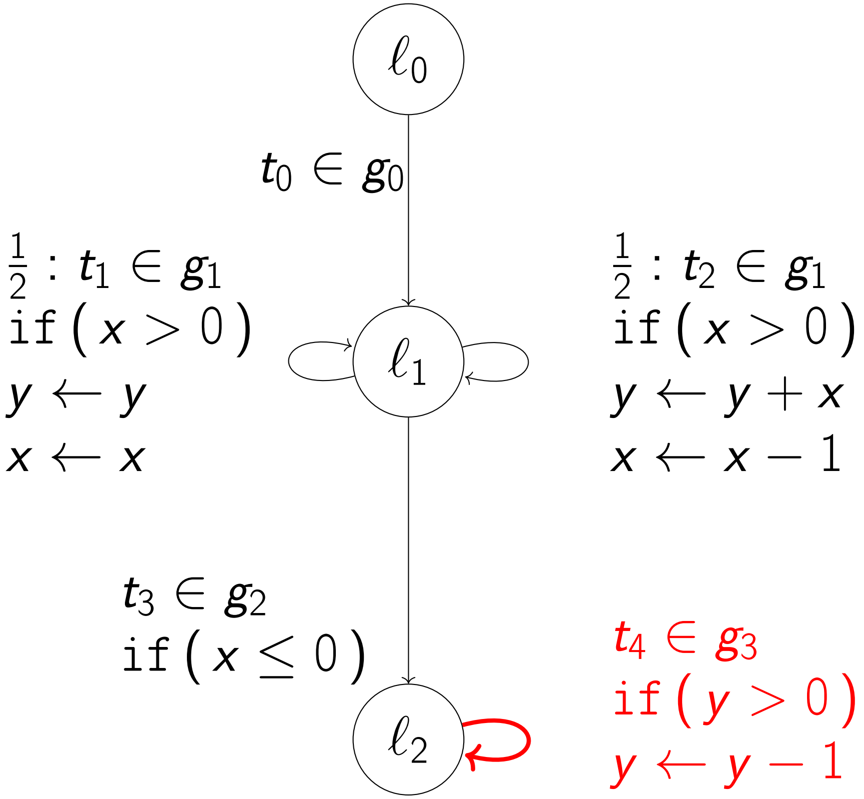
Overall Idea

- Given sub-program $\mathcal{GT}'$ of full program.



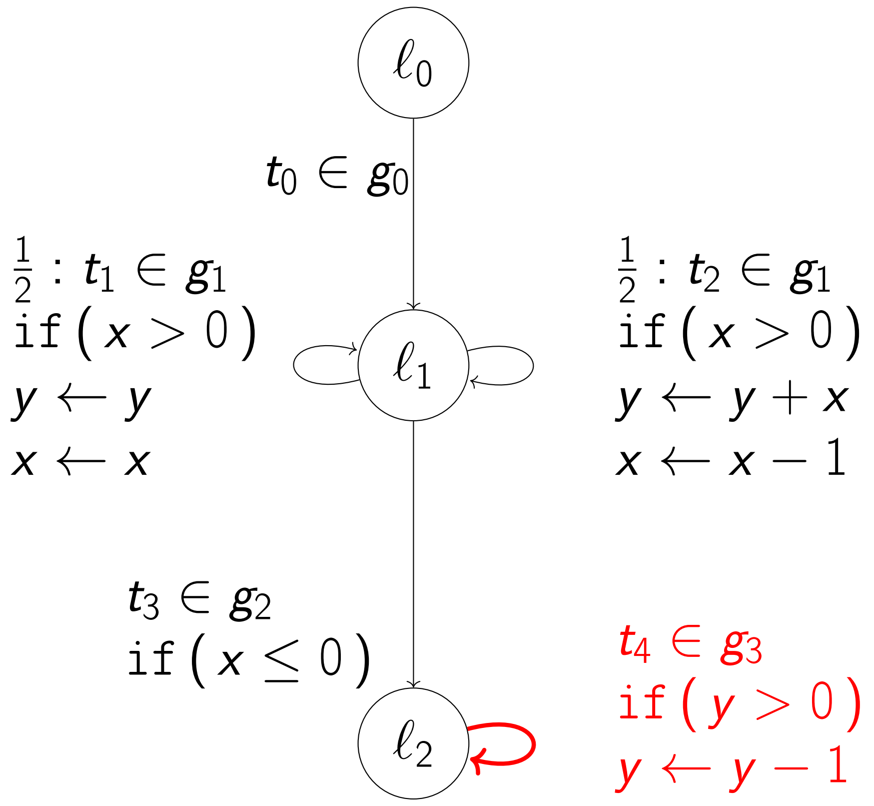
Overall Idea

- Given sub-program $\mathcal{GT}'$ of full program.
  Number of executions of $\mathcal{GT}'$ in run of full program?



Overall Idea

- Given sub-program $\mathcal{GT}'$ of full program.
  Number of executions of $\mathcal{GT}'$ in run of full program?
  $\leq (\#\ \text{enter}\ (\mathcal{GT}')) \cdot (\text{local-runtime}(\mathcal{GT}')\,[v/\text{``size''}_{\text{enter}\mathcal{GT}'}(v)])$



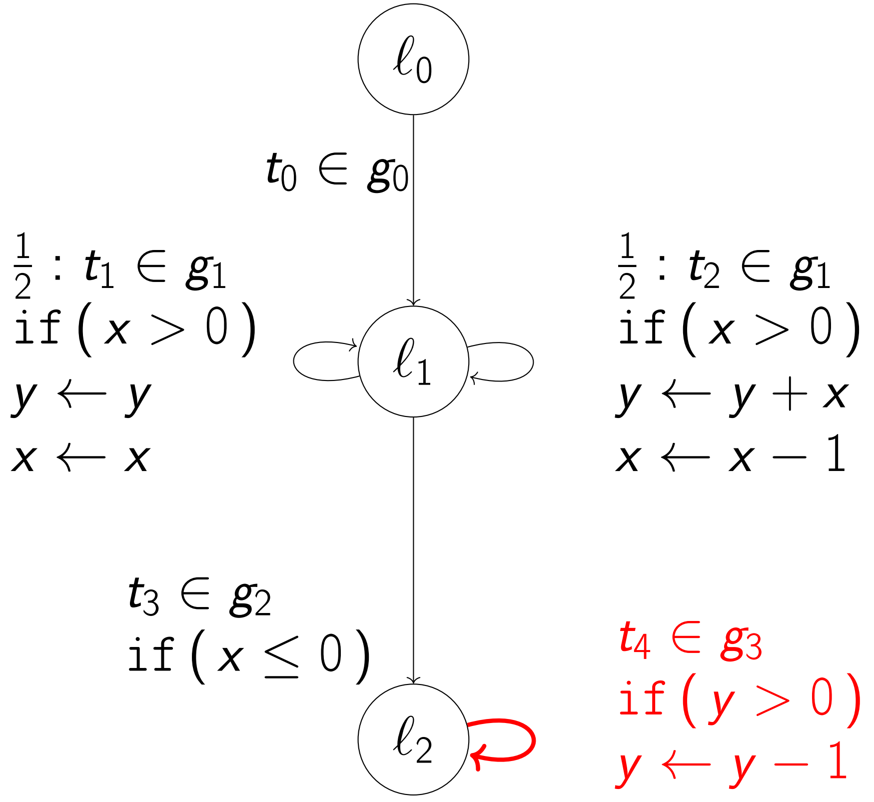
Overall Idea

- Given sub-program $\mathcal{GT}'$ of full program.

  Number of executions of $\mathcal{GT}'$ in run of full program?

  $\leq (\#\text{ enter }(\mathcal{GT}')) \cdot (\text{local-runtime}(\mathcal{GT}')\,[v/\text{``size''}_{\text{enter}\mathcal{GT}'}(v)])$

  Expected number of executions of $\mathcal{GT}'$?



Overall Idea

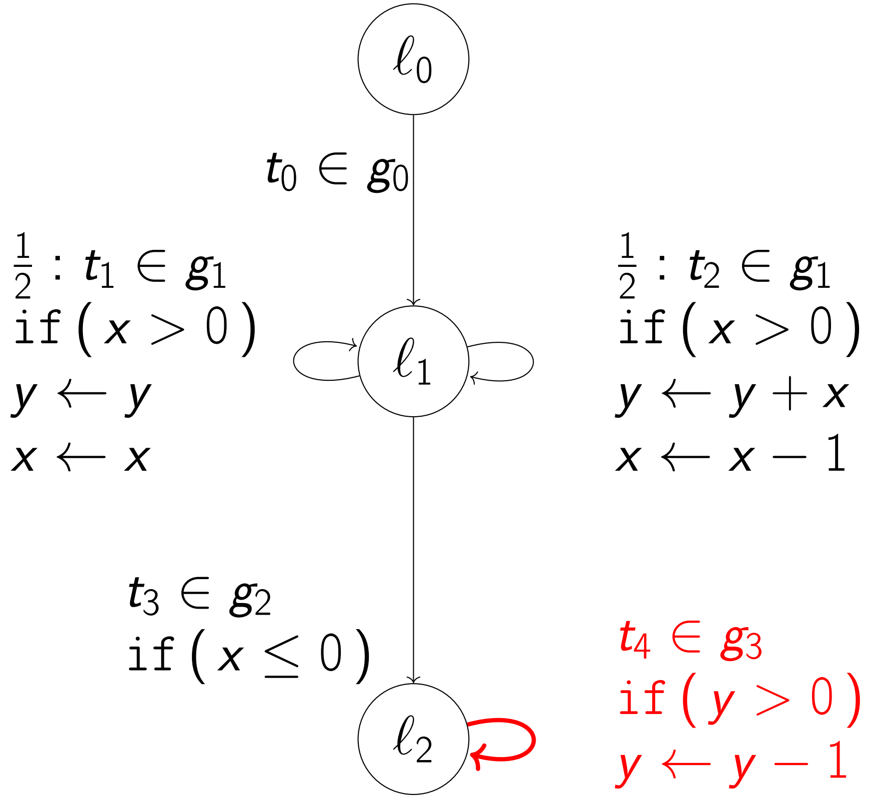
- Given sub-program $\mathcal{GT}'$ of full program.

  Number of executions of $\mathcal{GT}'$ in run of full program?

  $\leq (\#\ \text{enter}\ (\mathcal{GT}')) \cdot (\text{local-runtime}(\mathcal{GT}')\,[v/\text{``size''}_{\text{enter}\mathcal{GT}'}(v)])$

  Expected number of executions of $\mathcal{GT}'$?

  $\leq \mathbb{E}\Big((\#\ \text{enter}\ (\mathcal{GT}')) \cdot (\text{local-runtime}(\mathcal{GT}')\,[v/\text{``size''}_{\text{enter}\mathcal{GT}'}(v)])\Big)$



Overall Idea

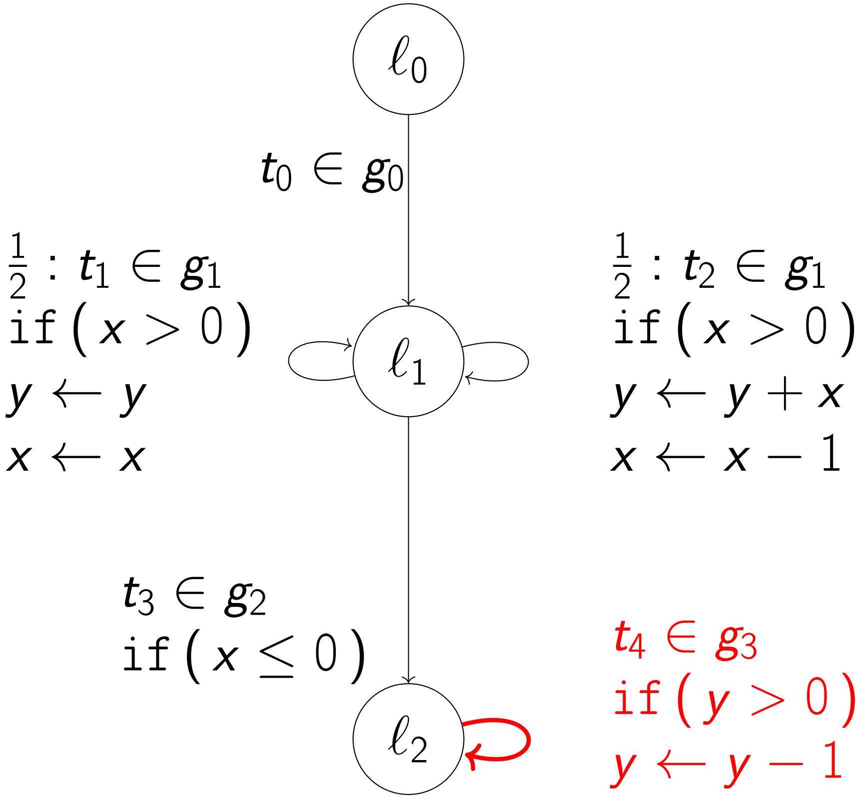
- Given sub-program $\mathcal{GT}'$ of full program.

  Number of executions of $\mathcal{GT}'$ in run of full program?

  $\leq (\#\text{ enter }(\mathcal{GT}')) \cdot (\text{local-runtime}(\mathcal{GT}')\,[v/\text{``size''}_{\text{enter}_{\mathcal{GT}'}}(v)])$

  Expected number of executions of $\mathcal{GT}'$?

  $\leq \mathbb{E}\Big((\#\text{ enter }(\mathcal{GT}')) \cdot (\text{local-runtime}(\mathcal{GT}')\,[v/\text{``size''}_{\text{enter}_{\mathcal{GT}'}}(v)])\Big)$
- How to over-approximate this expected value?



UnRAVeL | RWTH AACHEN UNIVERSITY

Computing Expected Runtime Bounds

Overall Idea

- Given sub-program $\mathcal{GT}'$ of full program.

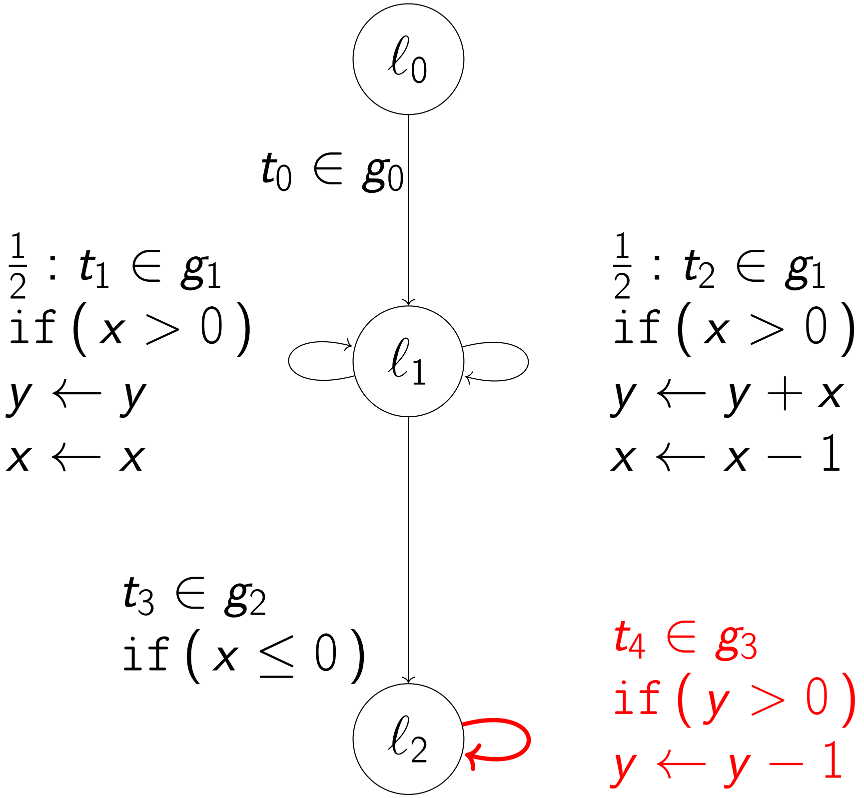
  Number of executions of $\mathcal{GT}'$ in run of full program?

  $\leq (\#\text{ enter }(\mathcal{GT}')) \cdot (\text{local-runtime}(\mathcal{GT}')\,[v/\text{“size”}_{\text{enter}\mathcal{GT}'}(v)])$

  Expected number of executions of $\mathcal{GT}'$?

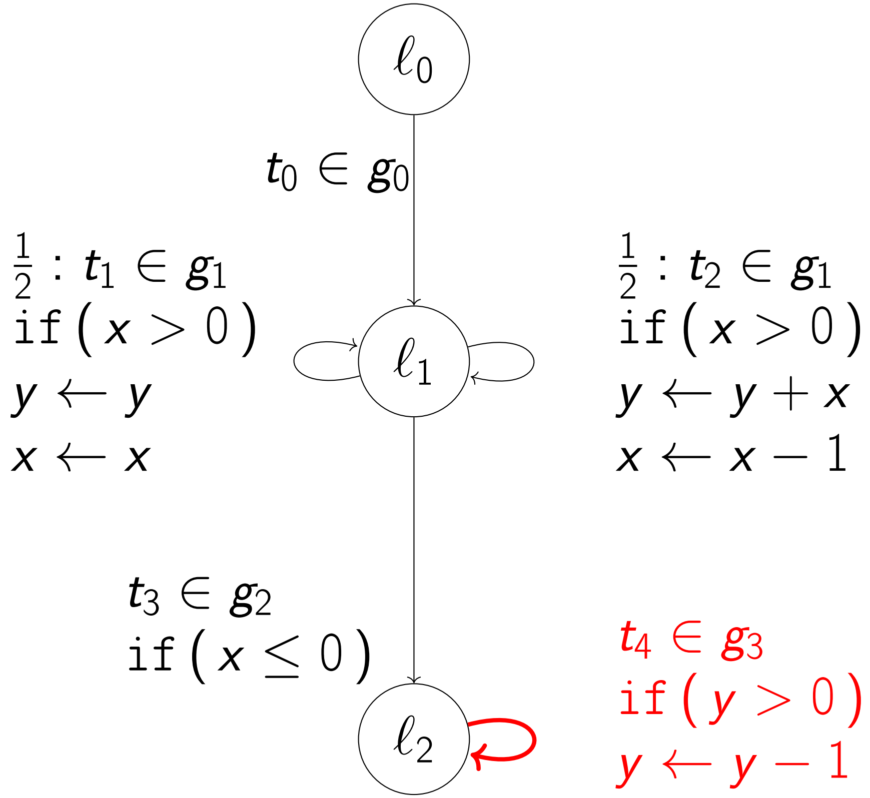
  $\leq \mathbb{E}\Big((\#\text{ enter }(\mathcal{GT}')) \cdot (\text{local-runtime}(\mathcal{GT}')\,[v/\text{“size”}_{\text{enter}\mathcal{GT}'}(v)])\Big)$
- How to over-approximate this expected value?

  $\rightarrow$ Expected value is not multiplicative.



Computation

$$\mathbb{E}\Big(\big(\#\ \text{enter}\ (\mathcal{GT}')\big)\cdot\big(\text{local-runtime}(\mathcal{GT}')\,[v/\text{``size''}_{\text{enter}_{\mathcal{GT}'}}(v)]\big)\Big)$$

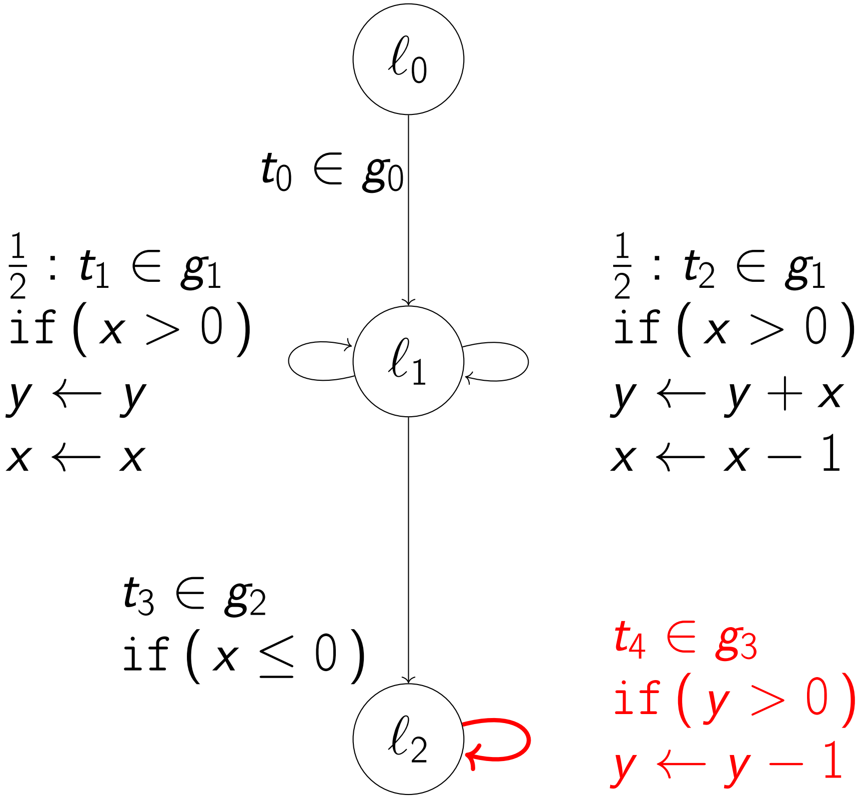


Computing Expected Runtime Bounds

Computation

$$\mathbb{E}\Big((\#\ \text{enter}\ (\mathcal{GT}')) \cdot (\text{local-runtime}(\mathcal{GT}')\,[v/\text{``size''}_{\text{enter}_{\mathcal{GT}'}}(v)]) \Big)$$

- How to over-approximate this expected value?



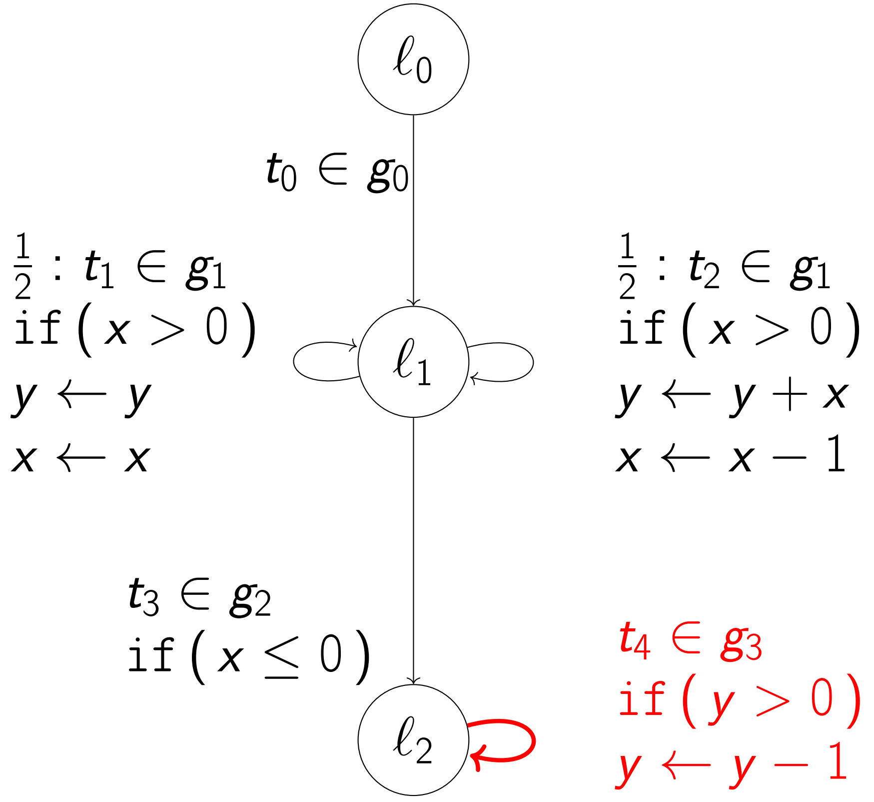
Computing Expected Runtime Bounds

Computation

$$\mathbb{E}\Big(\big(\#\ \text{enter}\ (\mathcal{GT}')\big)\cdot\big(\text{local-runtime}(\mathcal{GT}')\,[v/\text{``size''}_{\text{enter}\mathcal{GT}'}(v)]\big)\Big)$$

- How to over-approximate this expected value?

  $\#$ enter $(\mathcal{GT}')$:



Computing Expected Runtime Bounds

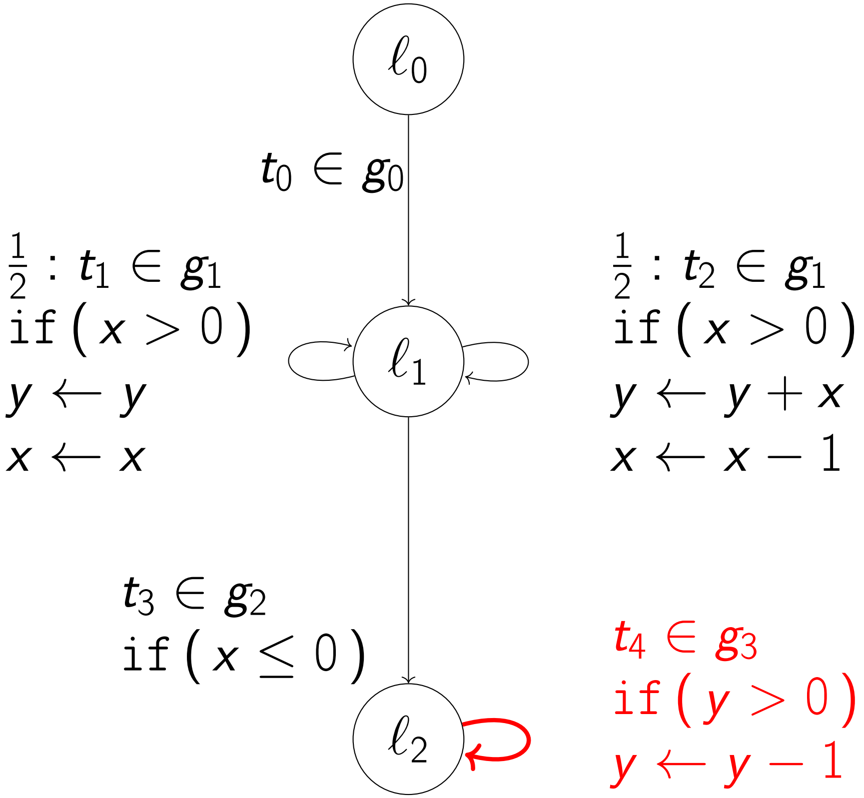
Computation

$$\mathbb{E}\Big(\big(\#\ \text{enter}\ (\mathcal{GT}')\big)\cdot\big(\text{local-runtime}(\mathcal{GT}')\,[v/\text{``size''}_{\text{enter}\,\mathcal{GT}'}(v)]\big)\Big)$$

- How to over-approximate this expected value?

  $\#$ enter $(\mathcal{GT}')$:

  $\rightarrow$ Use (classical) worst-case bound from [Giesl et al. '16].



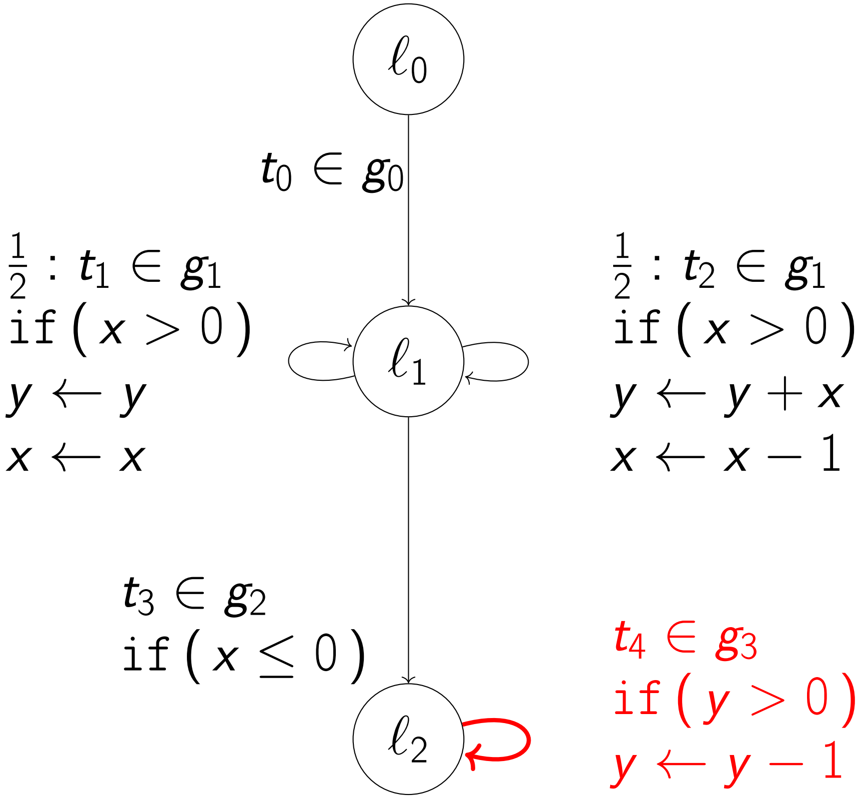
Computation

$$(\#\,\text{enter}\,(\mathcal{GT}'))\cdot\mathbb{E}\Big((\text{local-runtime}(\mathcal{GT}')\,[v/\text{``size''}_{\text{enter}\,\mathcal{GT}'}(v)])\Big)$$

- How to over-approximate this expected value?

  $\#$ enter $(\mathcal{GT}')$:

  $\rightarrow$ Use (classical) worst-case bound from [Giesl et al. '16].



Computing Expected Runtime Bounds

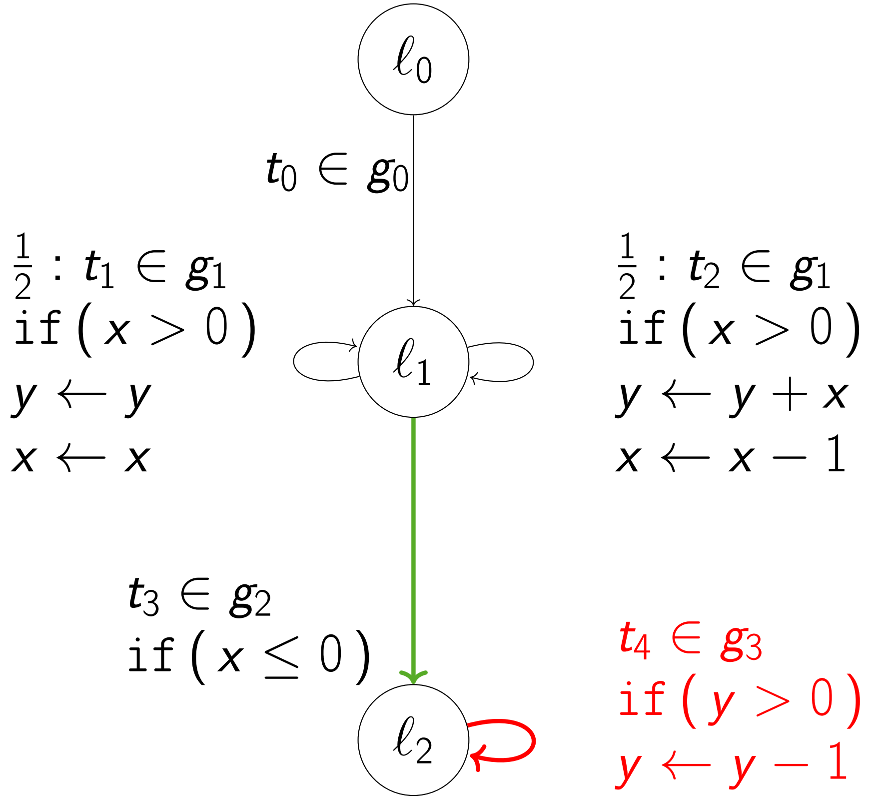
Computation

$$(\#\,\text{enter}\,(\mathcal{GT}'))\cdot\mathbb{E}\Big(\big(\text{local-runtime}(\mathcal{GT}')\,[v/\text{``size''}_{\text{enter}\mathcal{GT}'}(v)]\big)\Big)$$

- How to over-approximate this expected value?

  $\#$ enter $(\mathcal{GT}')$:

  $\rightarrow$ Use (classical) worst-case bound from [Giesl et al. '16].



Computation

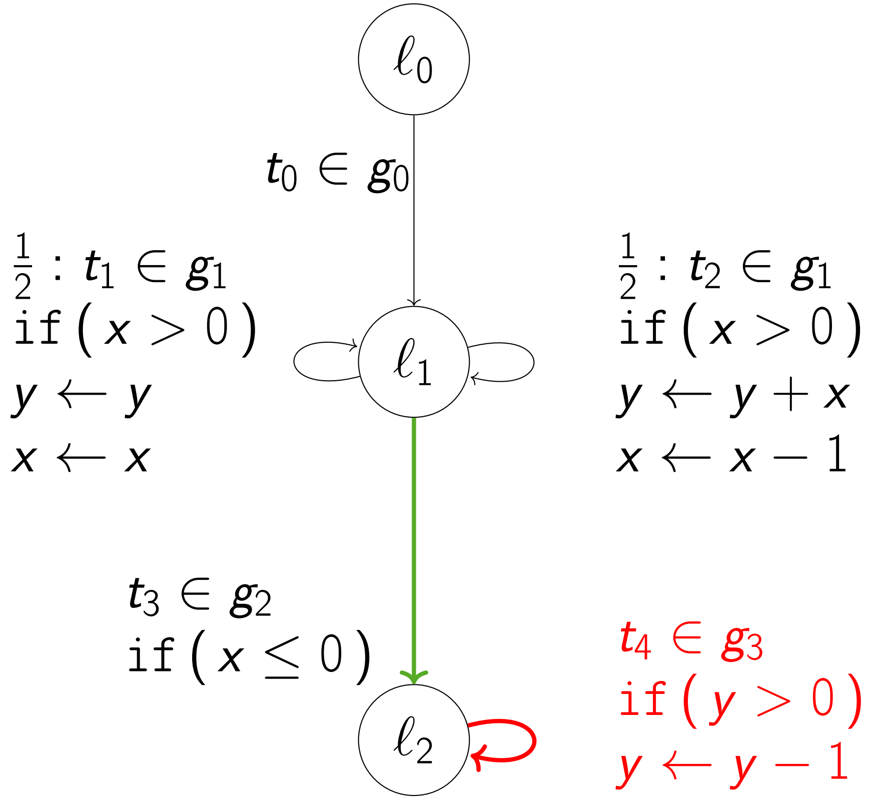
$$(\#\ \text{enter}\ (\mathcal{GT}')) \cdot \mathbb{E}\Big((\text{local-runtime}(\mathcal{GT}')\,[v/\text{“size”}_{\text{enter}\,\mathcal{GT}'}(v)])\Big)$$

- How to over-approximate this expected value?

  # enter $(\mathcal{GT}')$:

  → Use (classical) worst-case bound from [Giesl et al. '16].

  → # enter $(\mathcal{GT}') = 1$.

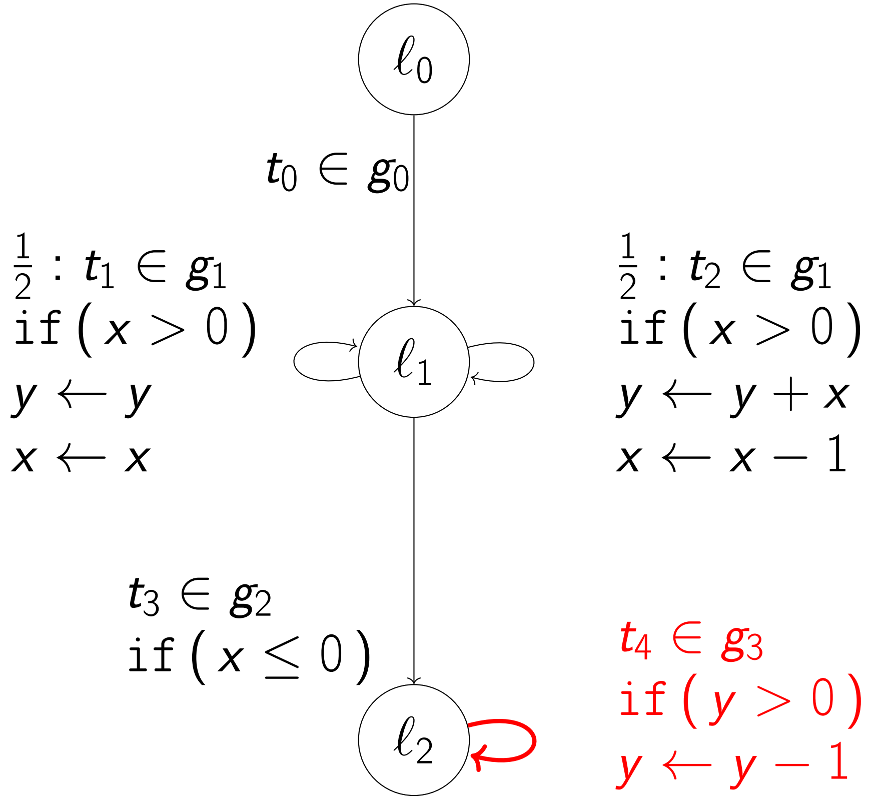


Computing Expected Runtime Bounds

Computation

$$\mathbb{E}\Big(\big(\text{local-runtime}(\mathcal{GT}')\,[v/\text{``size''}_{\text{enter}_{\mathcal{GT}'}}(v)]\big)\Big)$$

- How to over-approximate this expected value?

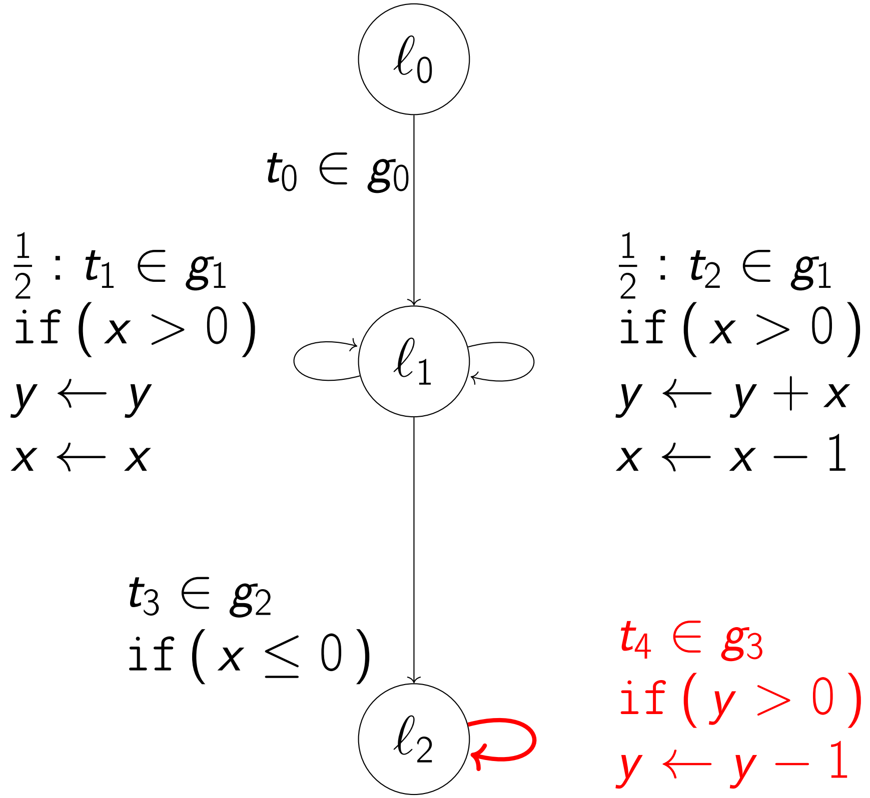


Computing Expected Runtime Bounds

Computation

$$\mathbb{E}\Big(\big(\text{local-runtime}(\mathcal{GT}')\,[v/\text{``size''}_{\text{enter}_{\mathcal{GT}'}}(v)]\big)\Big)$$

- How to over-approximate this expected value?
  - $\rightarrow$ Use linear (probabilistic) ranking function $\mathfrak{r}$ for $\mathcal{GT}'$.

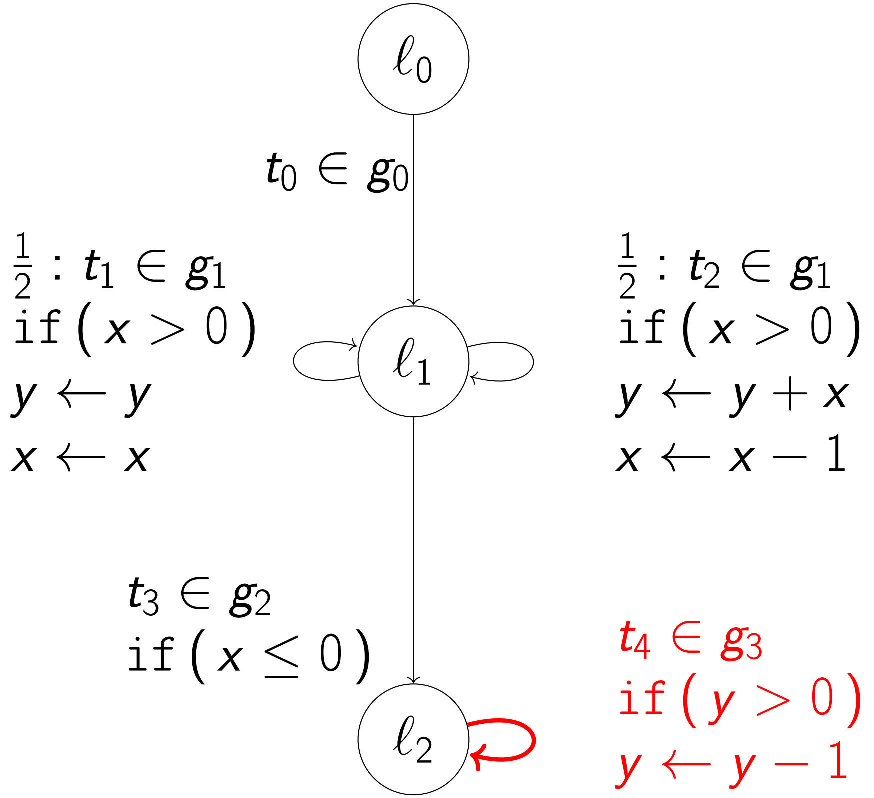


Computing Expected Runtime Bounds

Computation

$$\mathfrak{r}\left[v/\mathbb{E}(\text{“size”}_{\text{enter}_{\mathcal{GT}'}}(v))\right]$$

- How to over-approximate this expected value?
  - $\rightarrow$ Use linear (probabilistic) ranking function $\mathfrak{r}$ for $\mathcal{GT}'$.

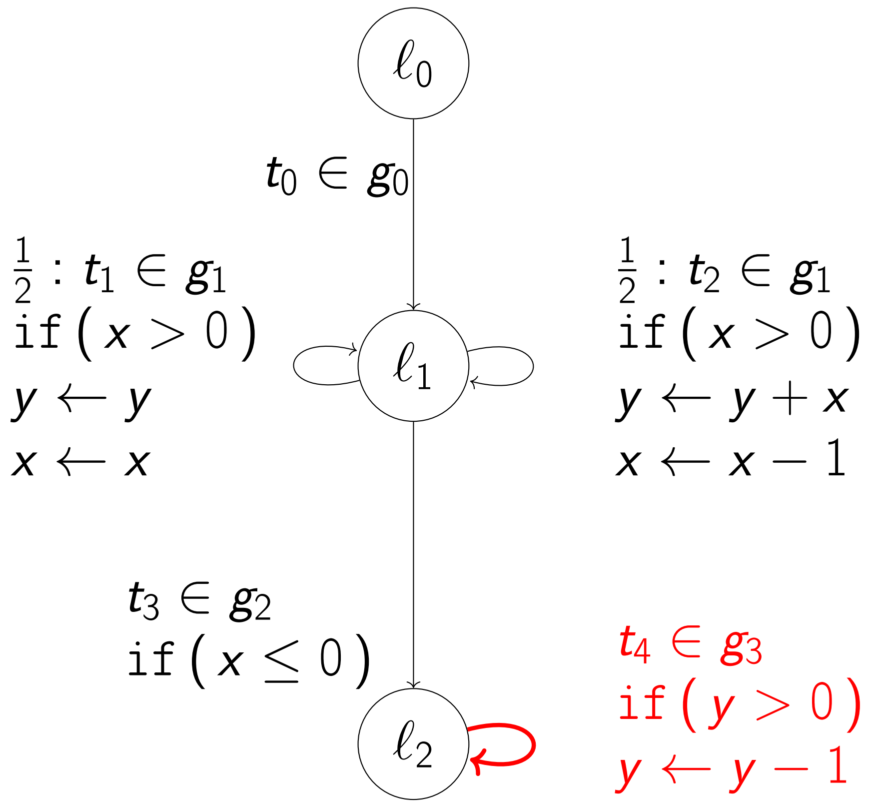


Computing Expected Runtime Bounds

Computation

$$\mathfrak{r}\left[v/\mathbb{E}(\text{``size''}_{\text{enter}\mathcal{GT}'}(v))\right]$$

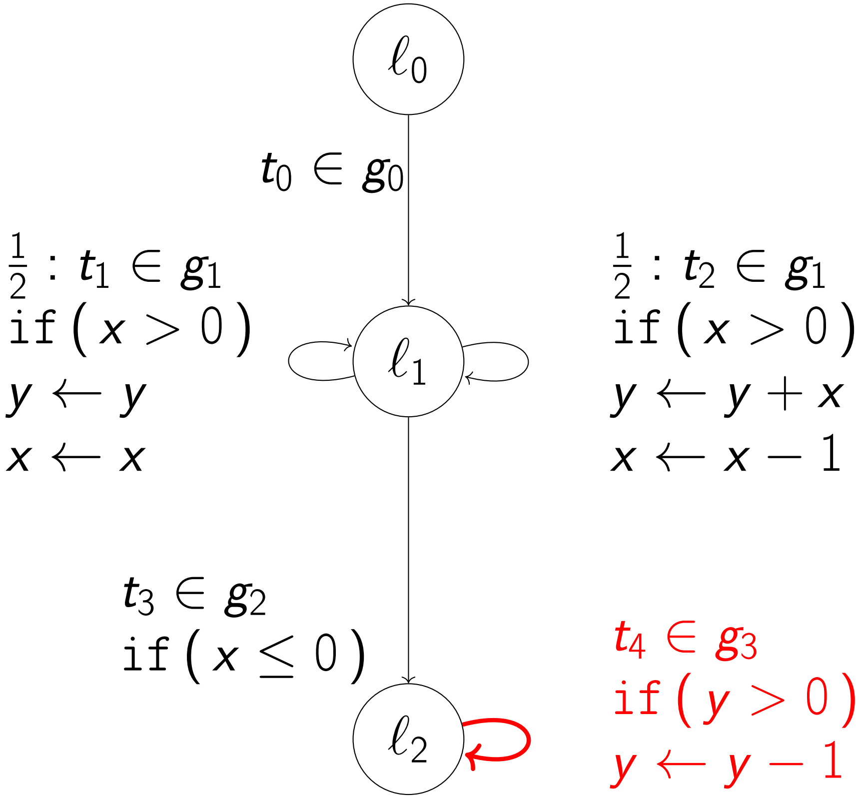
- How to over-approximate this expected value?
  - $\rightarrow$ Use linear (probabilistic) ranking function $\mathfrak{r}$ for $\mathcal{GT}'$.
  - $\rightarrow$ Already seen: y is a ranking function for $\mathcal{GT}'$.



Computation

$$y\,\left[y/\mathbb{E}(\text{``size''}_{\text{enter}\mathcal{GT}'}(y))\right]$$

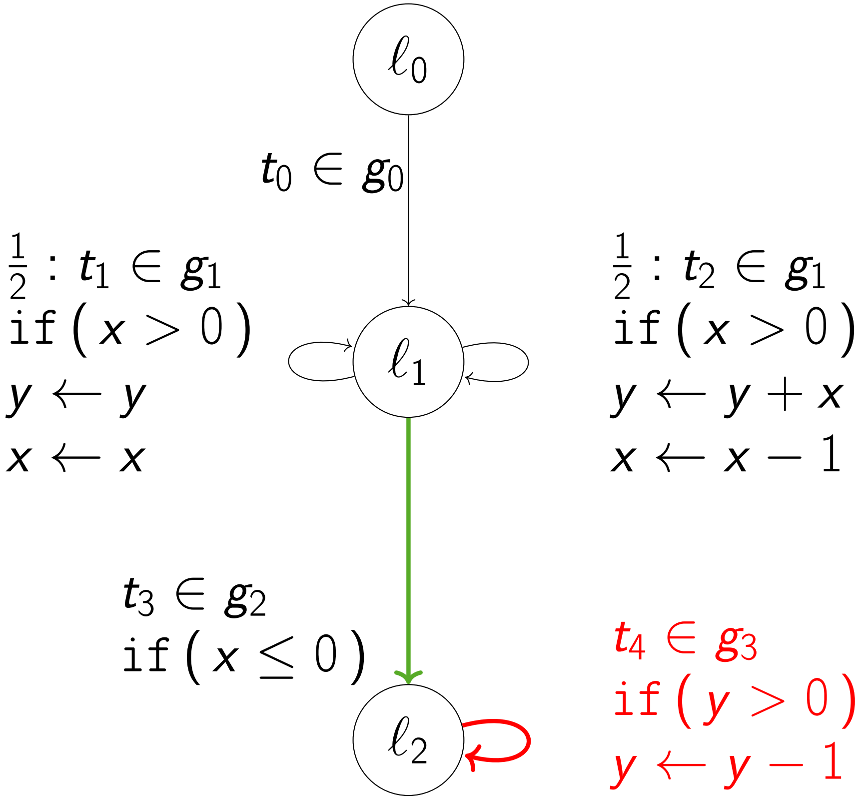
- How to over-approximate this expected value?
  - $\rightarrow$ Use linear (probabilistic) ranking function $\mathfrak{r}$ for $\mathcal{GT}'$.
  - $\rightarrow$ Already seen: y is a ranking function for $\mathcal{GT}'$.



Computation

$$y\left[y/\mathbb{E}(\text{"size"}_{\text{enter}\mathcal{GT}'}(y))\right]$$

- How to over-approximate this expected value?
  - $\to$ Use linear (probabilistic) ranking function $\mathfrak{r}$ for $\mathcal{GT}'$.
  - $\to$ Already seen: y is a ranking function for $\mathcal{GT}'$.
  - $\to$ Later: expected size of y after g_2: $x_0^2 + y_0$.

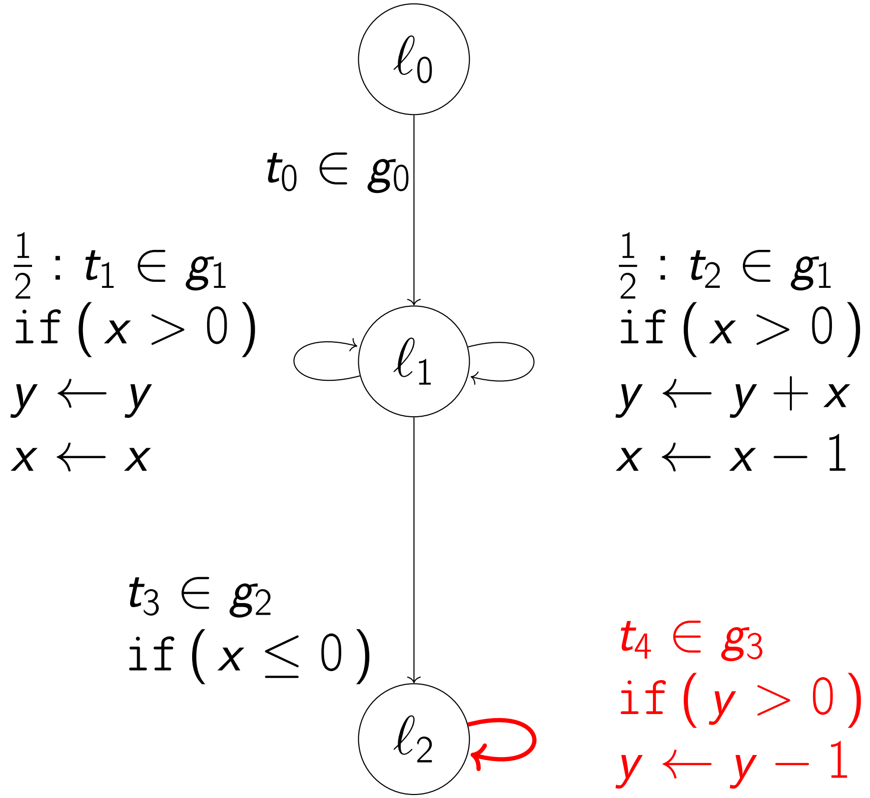


Computing Expected Runtime Bounds

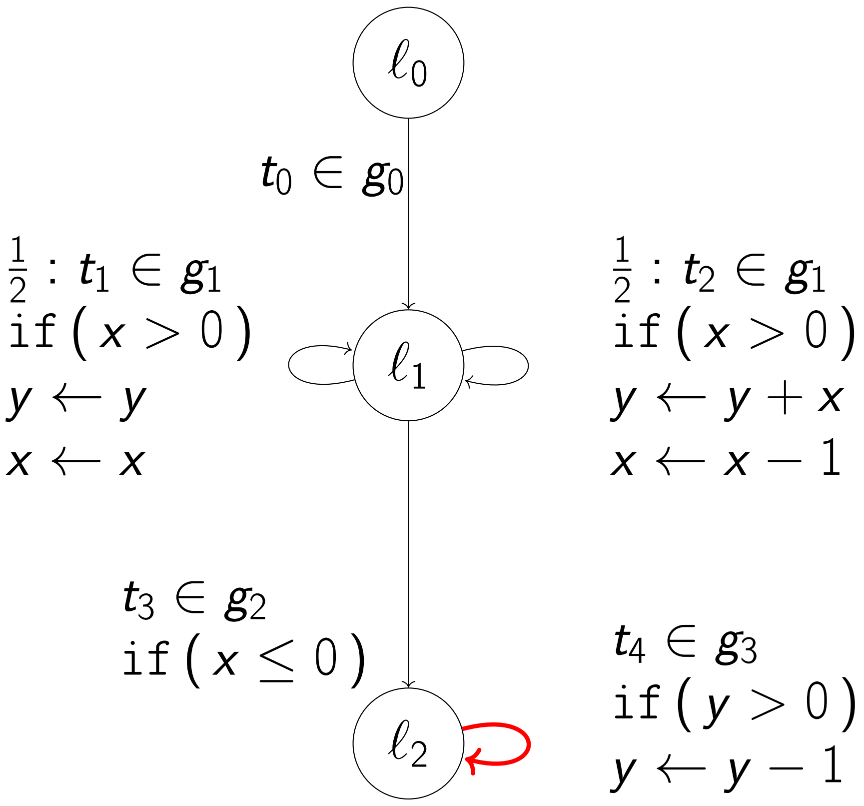
Computation

$x_0^2 + y_0$

- How to over-approximate this expected value?
  - $\rightarrow$ Use linear (probabilistic) ranking function $\mathfrak{r}$ for $\mathcal{GT}'$.
  - $\rightarrow$ Already seen: y is a ranking function for $\mathcal{GT}'$.
  - $\rightarrow$ Later: expected size of y after g_2: $x_0^2 + y_0$.



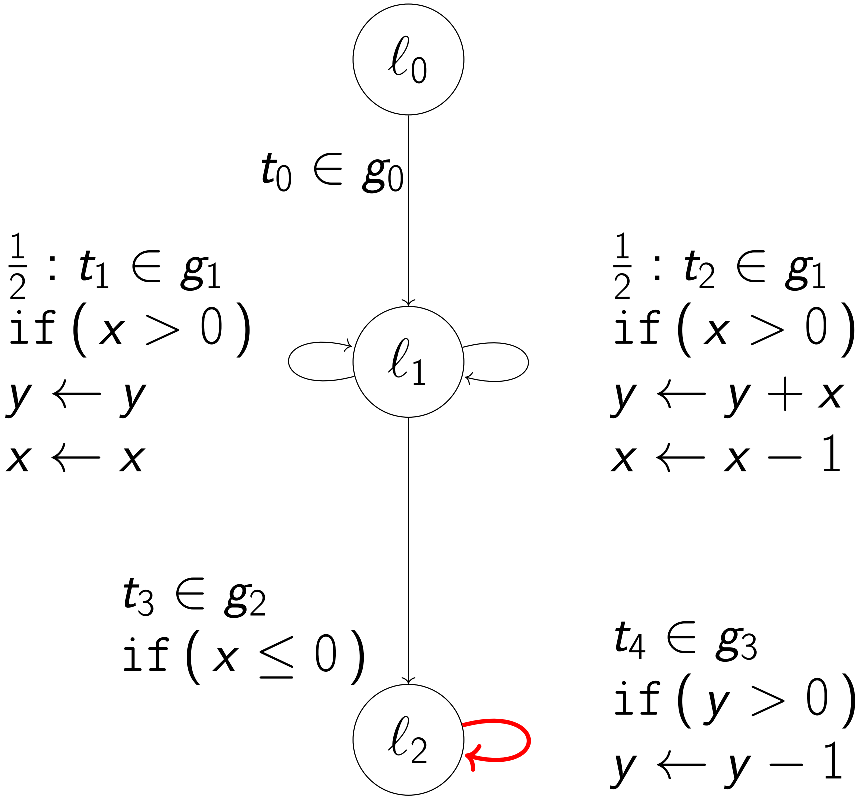
Summary


ℓ_0
$t_0 \in g_0$
$\frac{1}{2} : t_1 \in g_1$
if ($x > 0$)
$y \leftarrow y$
$x \leftarrow x$
ℓ_1
$\frac{1}{2} : t_2 \in g_1$
if ($x > 0$)
$y \leftarrow y + x$
$x \leftarrow x - 1$
$t_3 \in g_2$
if ($x \leq 0$)
ℓ_2
$t_4 \in g_3$
if ($y > 0$)
$y \leftarrow y - 1$

Computing Expected Runtime Bounds

Summary

Overall expected runtime of program:

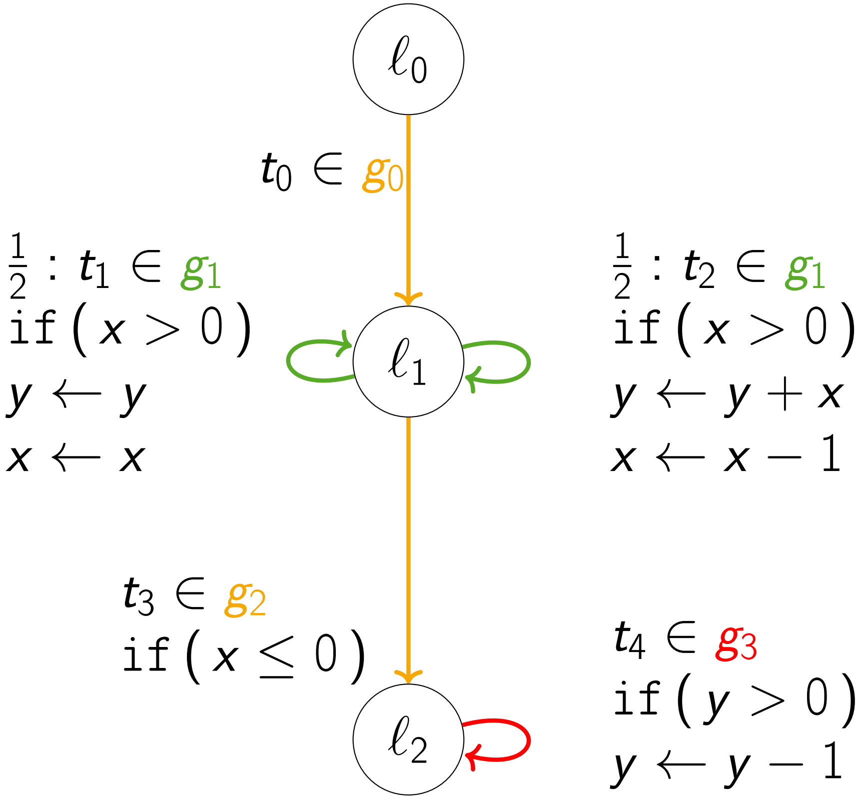


Computing Expected Runtime Bounds

Summary

Overall expected runtime of program:

$$1 + 2 \cdot x_0 + 1 + x_0^2 + y_0$$

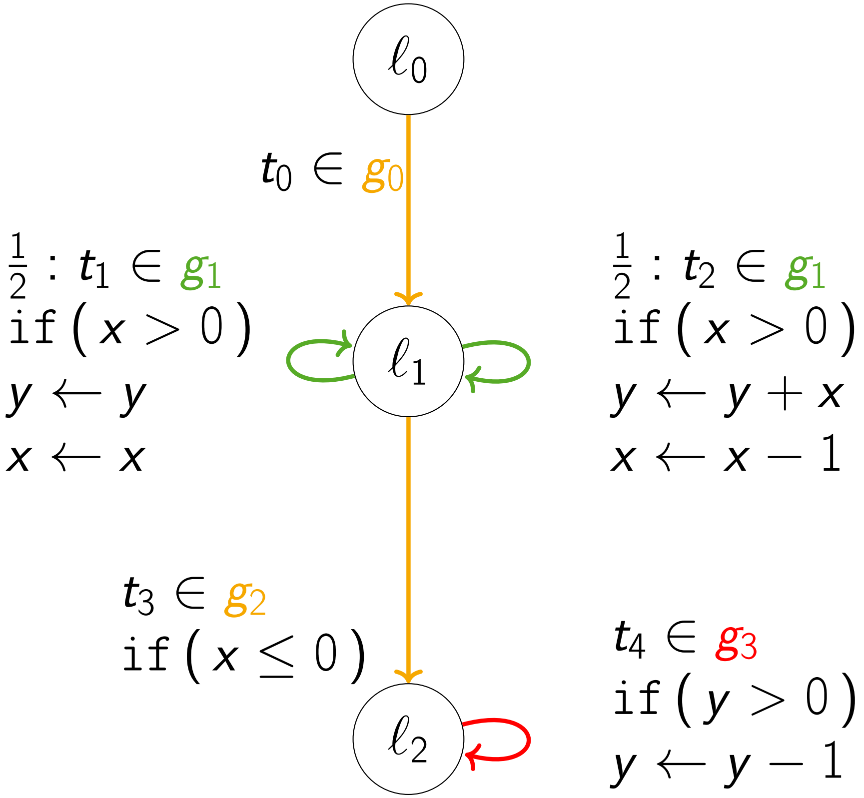


Summary

Overall expected runtime of program:

$1 + 2 \cdot x_0 + 1 + x_0^2 + y_0$

- Implementation: heuristic chooses sub-programs.



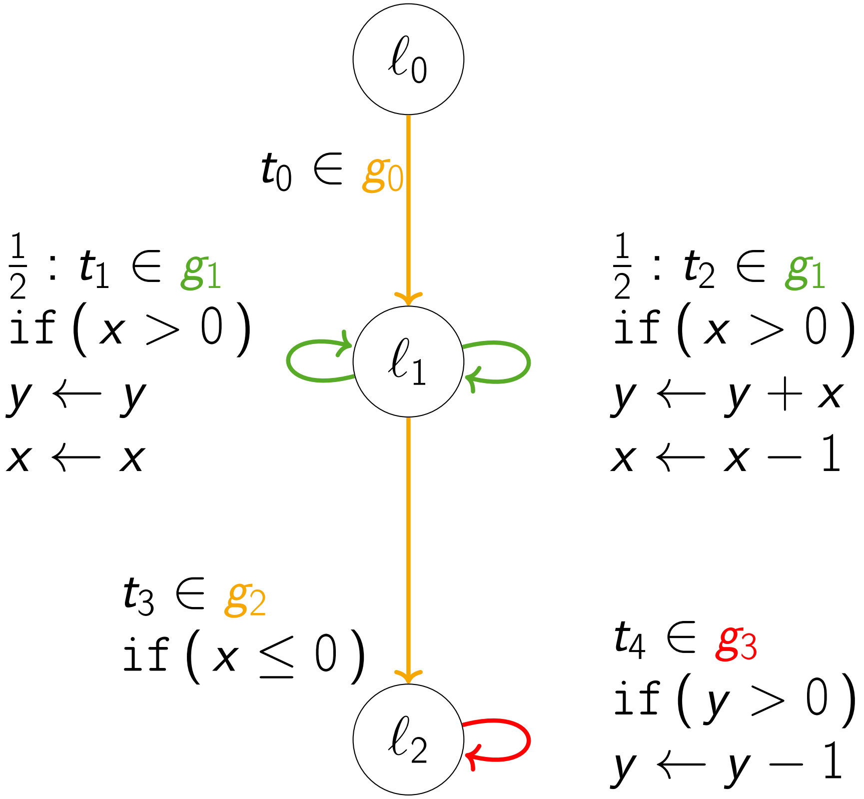
Computing Expected Runtime Bounds

Summary

Overall expected runtime of program:

$1 + 2 \cdot x_0 + 1 + x_0^2 + y_0$

- Implementation: heuristic chooses sub-programs.
- **Modular** approach for expected runtimes:



Computing Expected Runtime Bounds

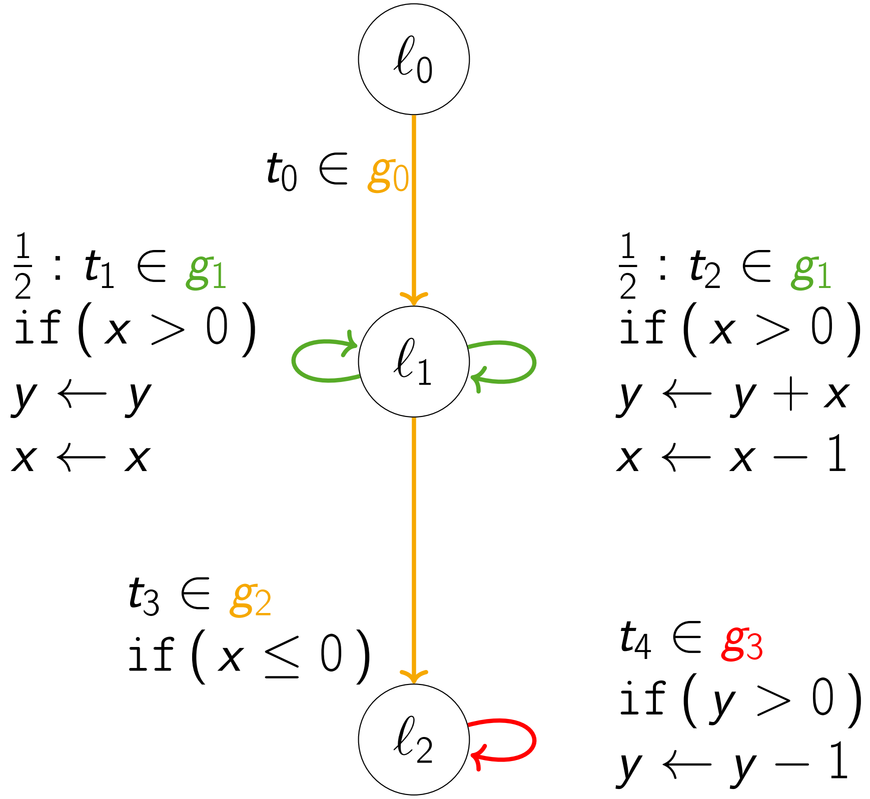
Summary

Overall expected runtime of program:

$$1 + 2 \cdot x_0 + 1 + x_0^2 + y_0$$

- Implementation: heuristic chooses sub-programs.
- **Modular** approach for expected runtimes:

  $\rightarrow$ Use of (classical) worst-case runtime bounds.



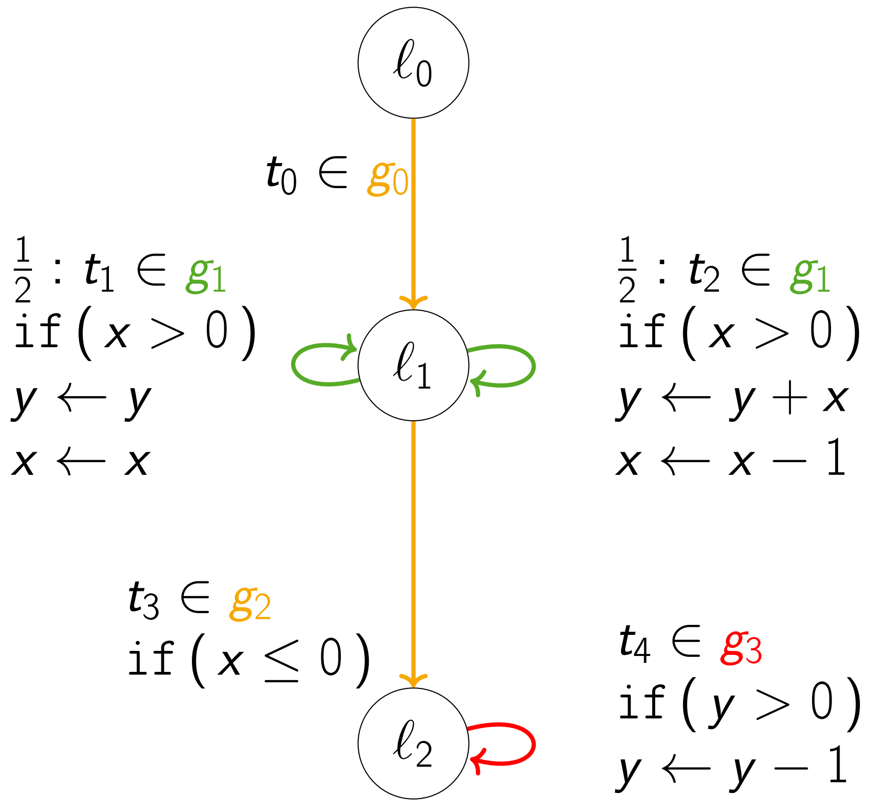
Computing Expected Runtime Bounds

Summary

Overall expected runtime of program:

$$1 + 2 \cdot x_0 + 1 + x_0^2 + y_0$$

- Implementation: heuristic chooses sub-programs.
- **Modular** approach for expected runtimes:
  - $\rightarrow$ Use of (classical) worst-case runtime bounds.
  - $\rightarrow$ Adaption of probabilistic ranking functions.

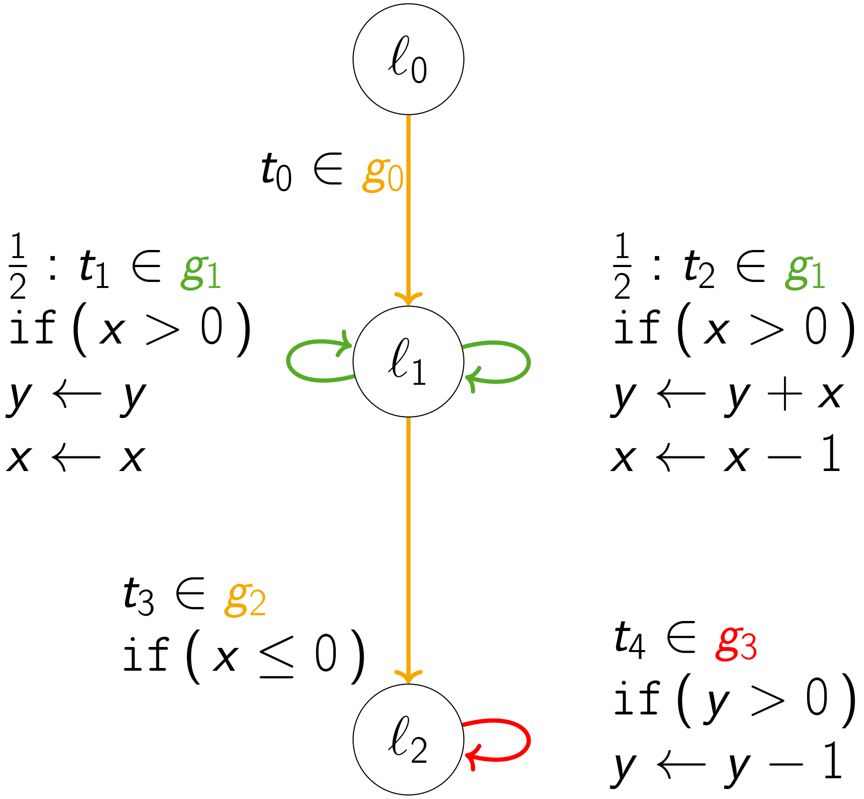


Summary

Overall expected runtime of program:

$$1 + 2 \cdot x_0 + 1 + x_0^2 + y_0$$

- Implementation: heuristic chooses sub-programs.
- **Modular** approach for expected runtimes:
  - → Use of (classical) worst-case runtime bounds.
  - → Adaption of probabilistic ranking functions.
  - → Use of expected sizes.

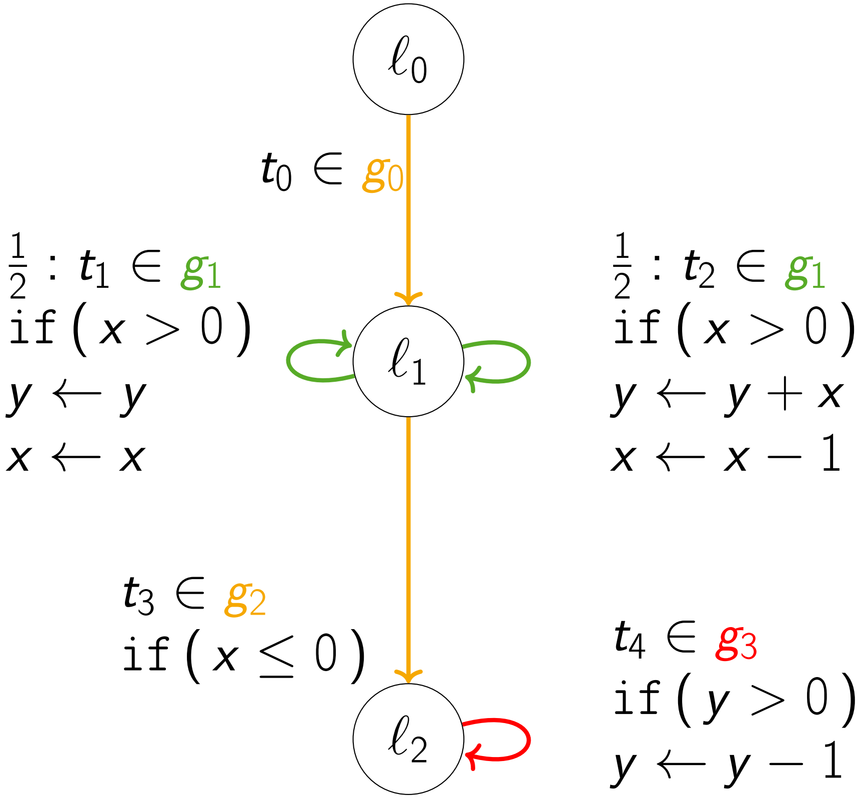


Summary

Overall expected runtime of program:

$1 + 2 \cdot x_0 + 1 + x_0^2 + y_0$

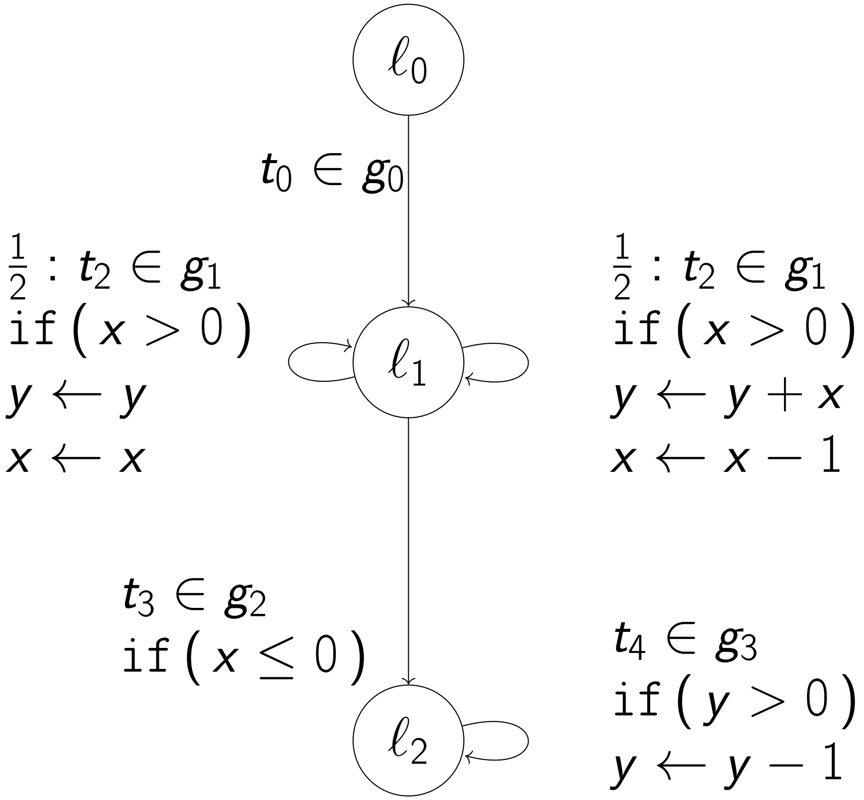
- Implementation: heuristic chooses sub-programs.
- **Modular** approach for expected runtimes:
  - → Use of (classical) worst-case runtime bounds.
  - → Adaption of probabilistic ranking functions.
  - → Use of expected sizes.
  - → Alternating computation of runtime and size bounds.



Computing Expected Size Bounds

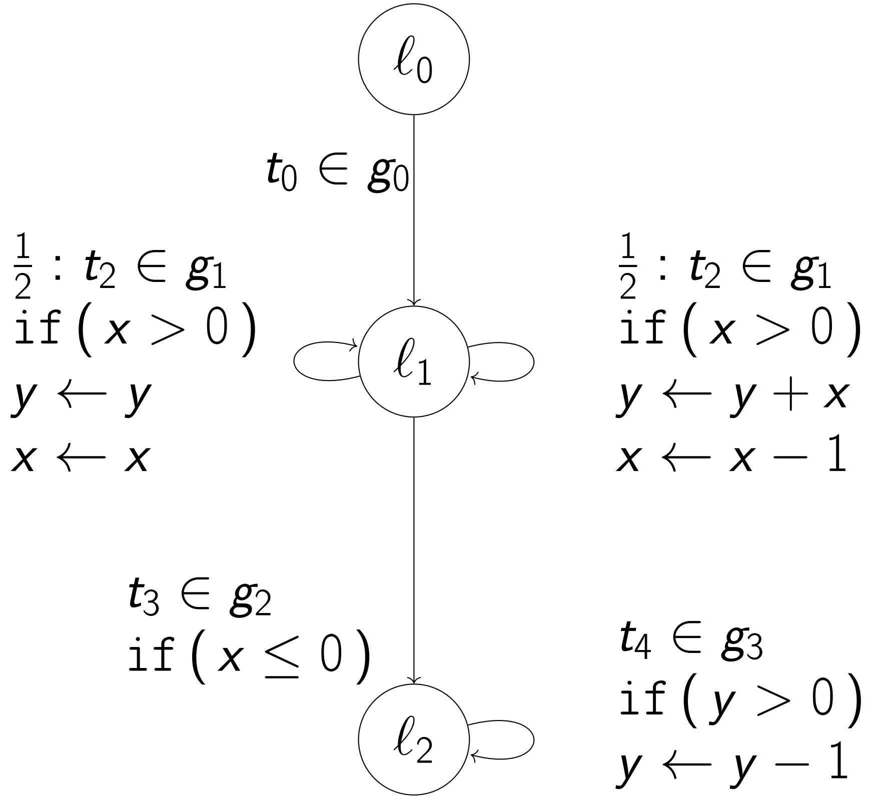
Overall Idea

Overall Idea



Overall Idea

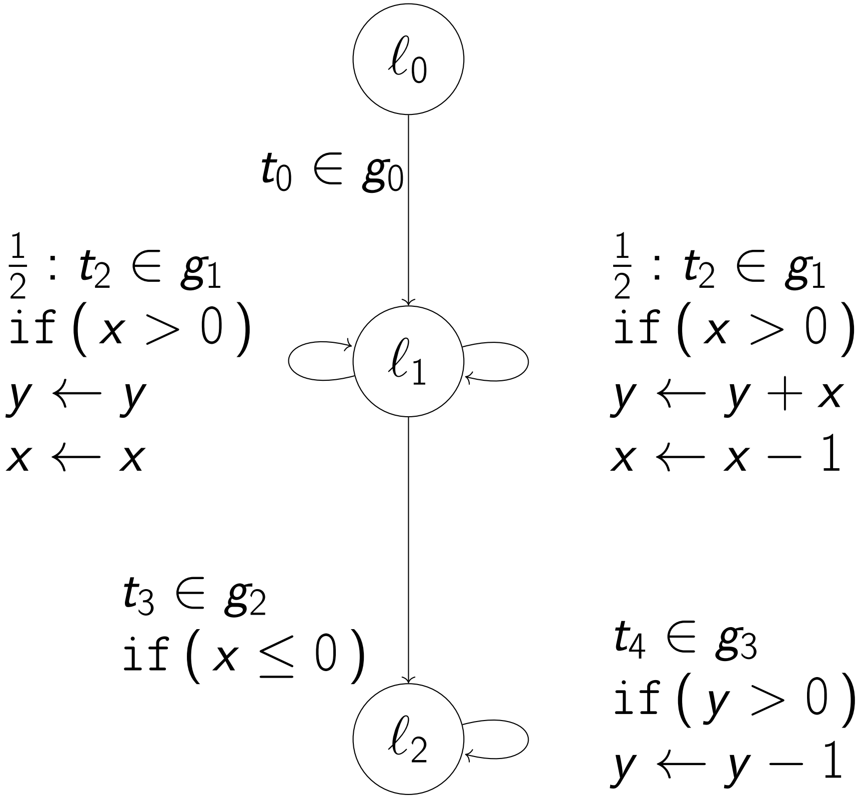
- What means "size"?



Computing Expected Size Bounds

Overall Idea

- What means "size"?
  → $\mathcal{S}(g, v)$: largest value v takes after execution of g

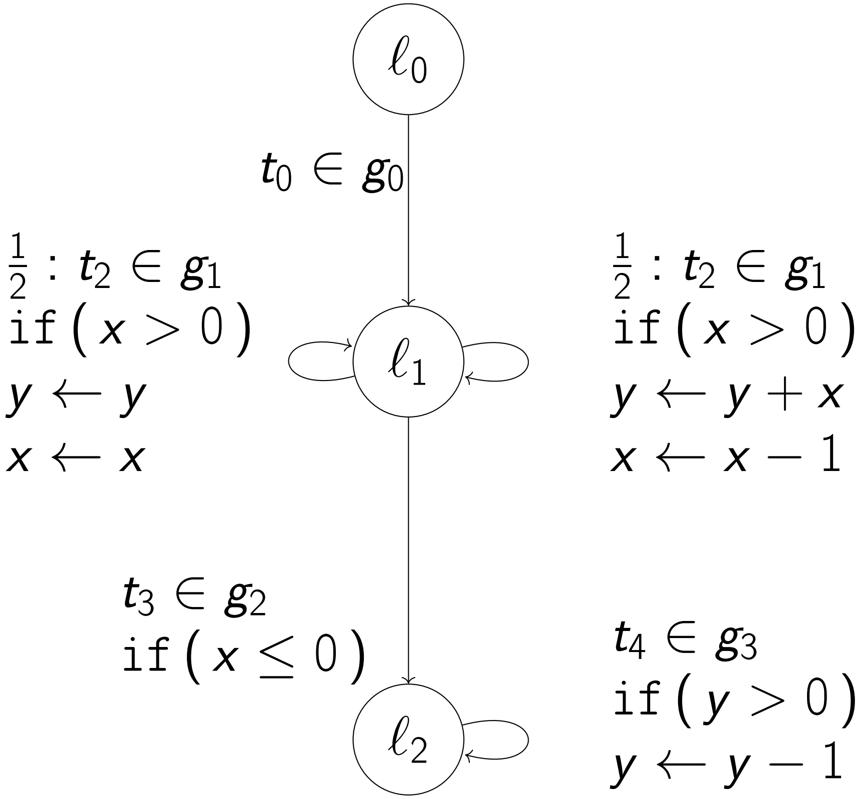


Overall Idea

- What means "size"?

  $\to \mathcal{S}(g, v)$: largest value v takes after execution of g

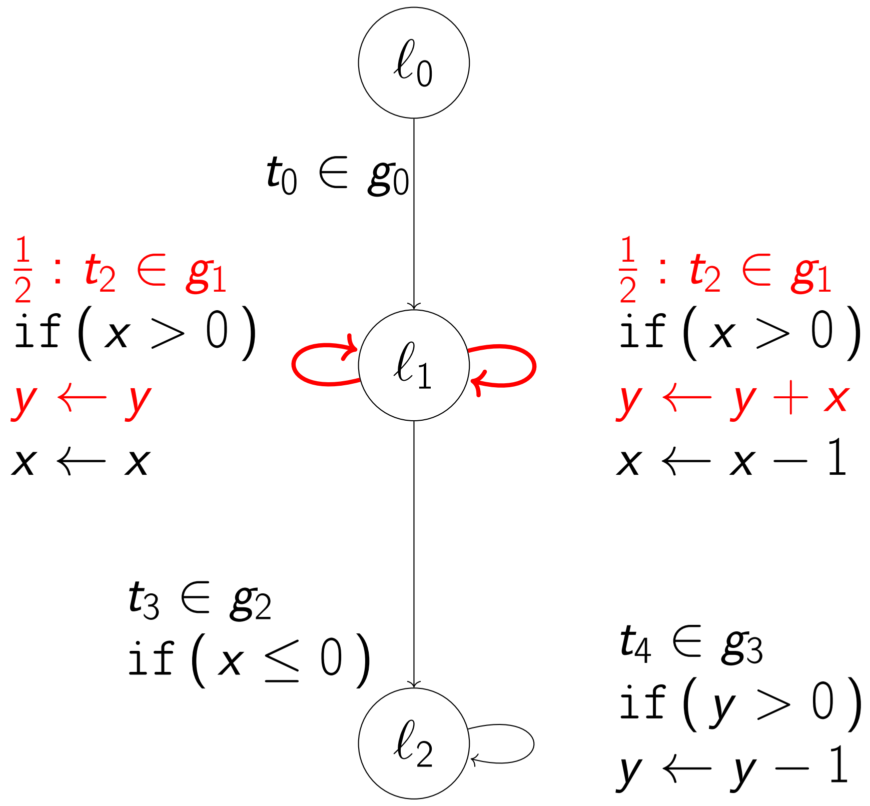
  $\leq$ incoming-size$(v) + \mathcal{R}(g) \cdot$ worst-case-change(g, v)



Computing Expected Size Bounds

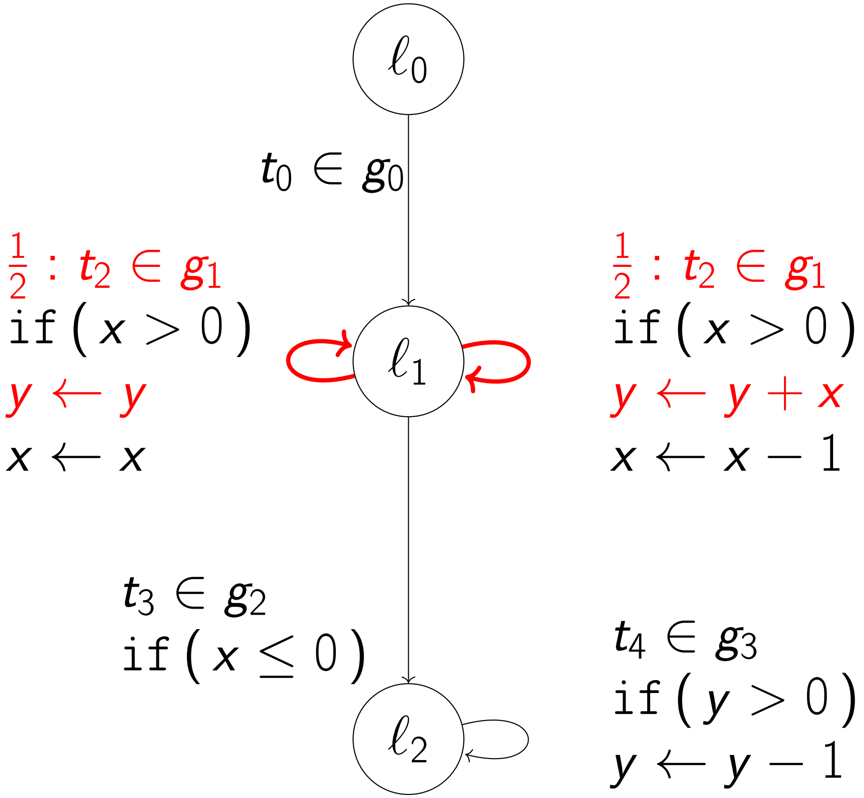
Overall Idea

- What means "size"?
  - $\to \mathcal{S}(g_1, y)$ largest value y takes after execution of g_1
  - $\leq$ incoming-size$(y) + \mathcal{R}(g_1) \cdot$ worst-case-change(g_1, y)



Overall Idea

Expected size $\mathbb{E}(\mathcal{S}(g_1, y))$:

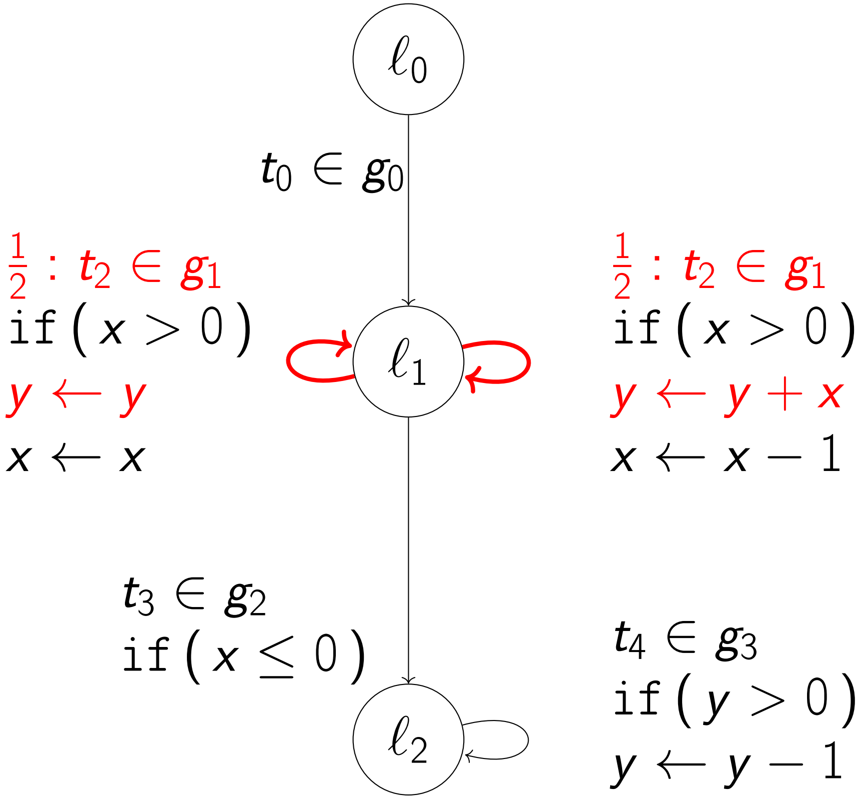


Computing Expected Size Bounds

Overall Idea

Expected size $\mathbb{E}(\mathcal{S}(g_1, y))$:

$$\leq \mathbb{E}(\text{incoming-size}(y) + \mathcal{R}(g_1) \cdot \text{worst-case-change}(g_1, y))$$



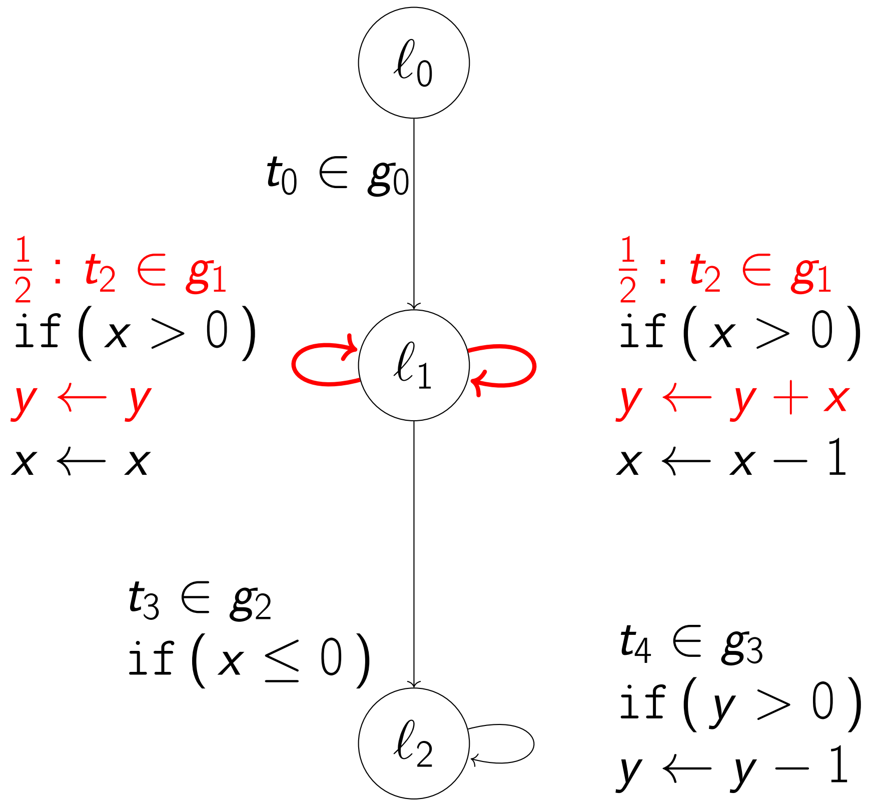
Computing Expected Size Bounds

Overall Idea

Expected size $\mathbb{E}(\mathcal{S}(g_1, y))$:

$$\leq \mathbb{E}\big(\text{incoming-size}(y) + \mathcal{R}(g_1) \cdot \text{worst-case-change}(g_1, y)\big)$$

$$= \mathbb{E}\big(\text{incoming-size}(y)\big) + \mathbb{E}\big(\mathcal{R}(g_1) \cdot \text{worst-case-change}(g_1, y)\big)$$



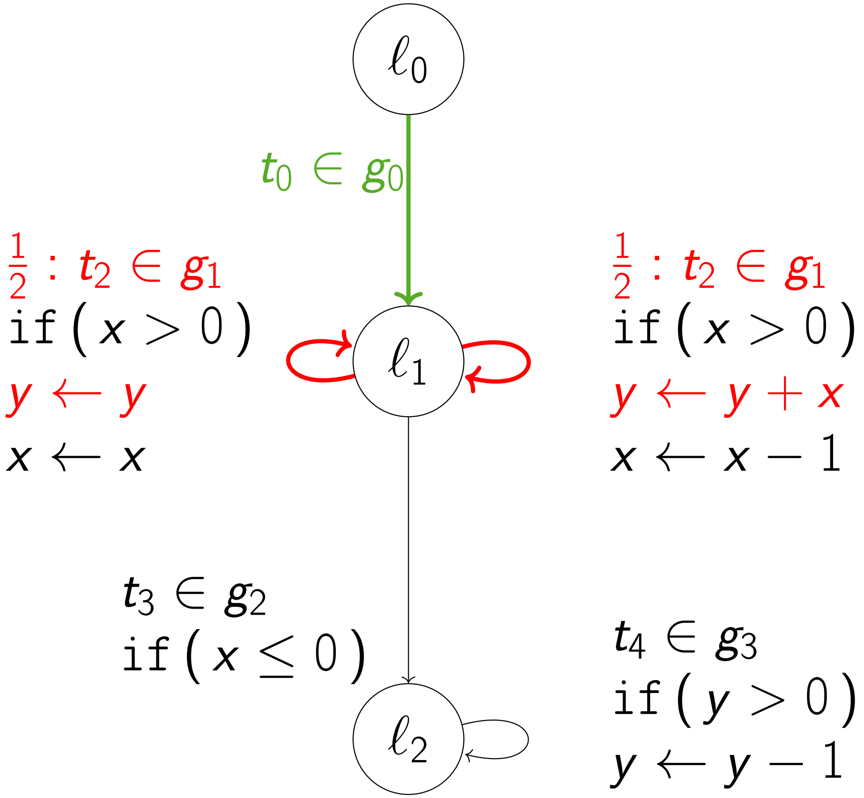
Computing Expected Size Bounds

Overall Idea

Expected size $\mathbb{E}(\mathcal{S}(g_1, y))$:

$$\leq \mathbb{E}(\text{incoming-size}(y) + \mathcal{R}(g_1) \cdot \text{worst-case-change}(g_1, y))$$

$$= \mathbb{E}(\text{incoming-size}(y)) + \mathbb{E}(\mathcal{R}(g_1) \cdot \text{worst-case-change}(g_1, y))$$

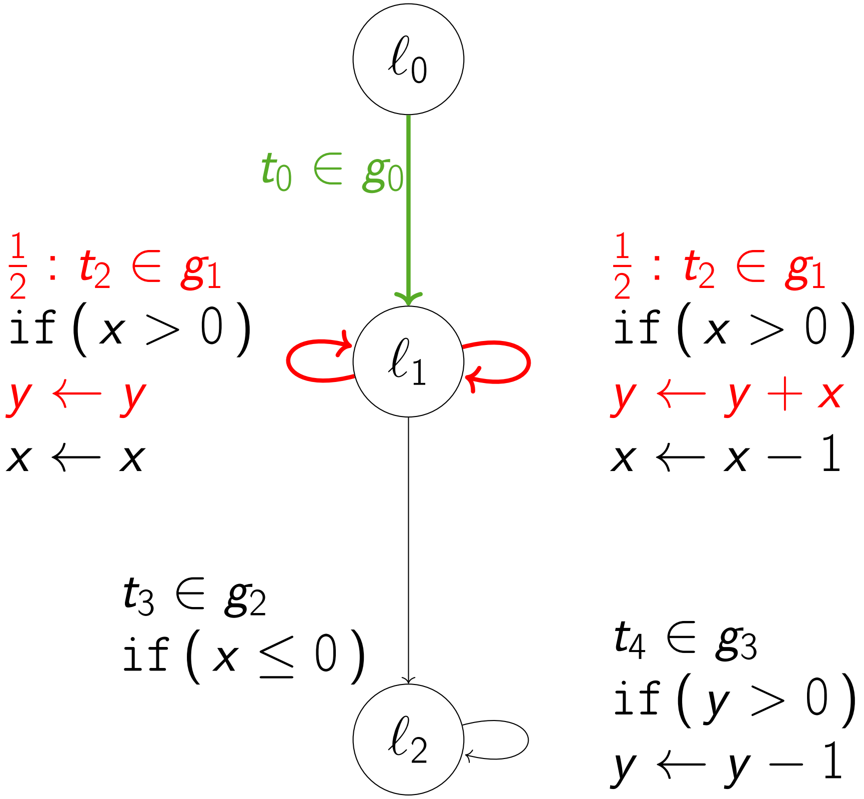


Computing Expected Size Bounds

Overall Idea

Expected size $\mathbb{E}(\mathcal{S}(g_1, y))$:

$$\begin{aligned} &\leq \mathbb{E}\big(\text{incoming-size}(y) + \mathcal{R}(g_1) \cdot \text{worst-case-change}(g_1, y)\big) \\ &= y_0 + \mathbb{E}\big(\mathcal{R}(g_1) \cdot \text{worst-case-change}(g_1, y)\big) \end{aligned}$$

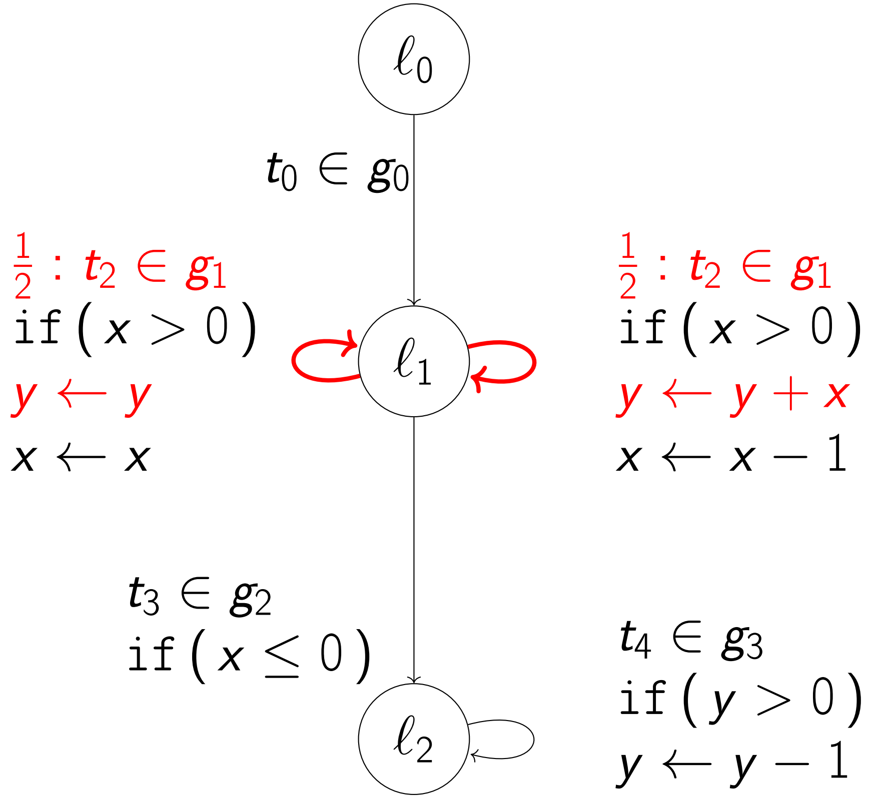


Computing Expected Size Bounds

Computation

Expected size $\mathbb{E}(\mathcal{S}(g_1, y))$:

$$\leq \; y_0 + \mathbb{E}(\mathcal{R}(g_1) \cdot \text{worst-case-change}(g_1, y))$$



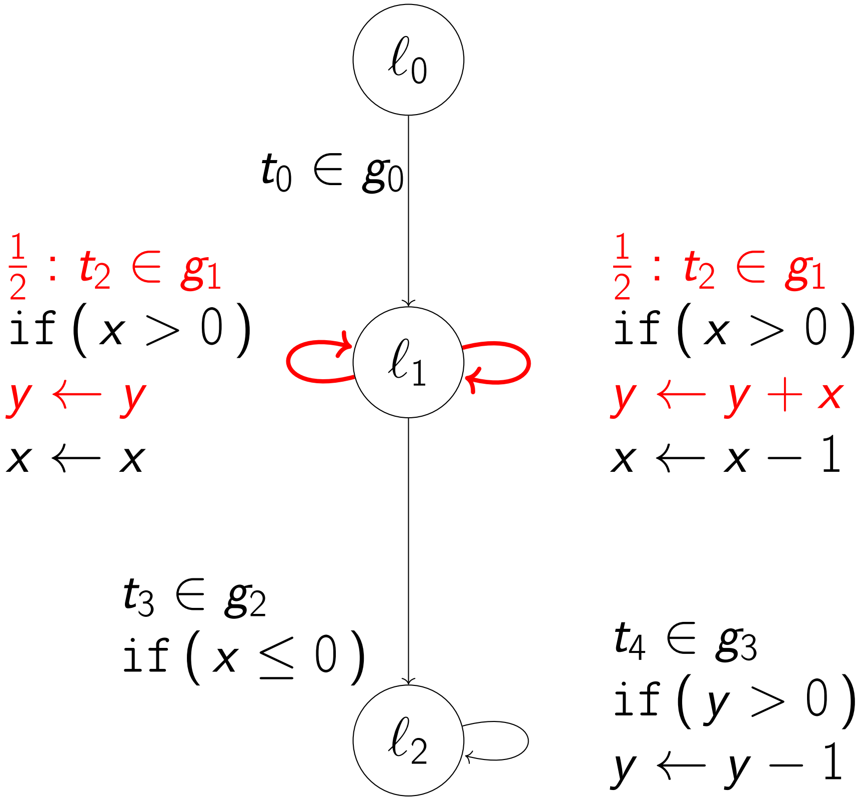
Computing Expected Size Bounds

Computation

Expected size $\mathbb{E}(\mathcal{S}(g_1, y))$:

$\leq \; y_0 + \mathbb{E}(\mathcal{R}(g_1) \cdot \text{worst-case-change}(g_1, y))$

$\rightarrow$ worst-case-change is independent of runtime.

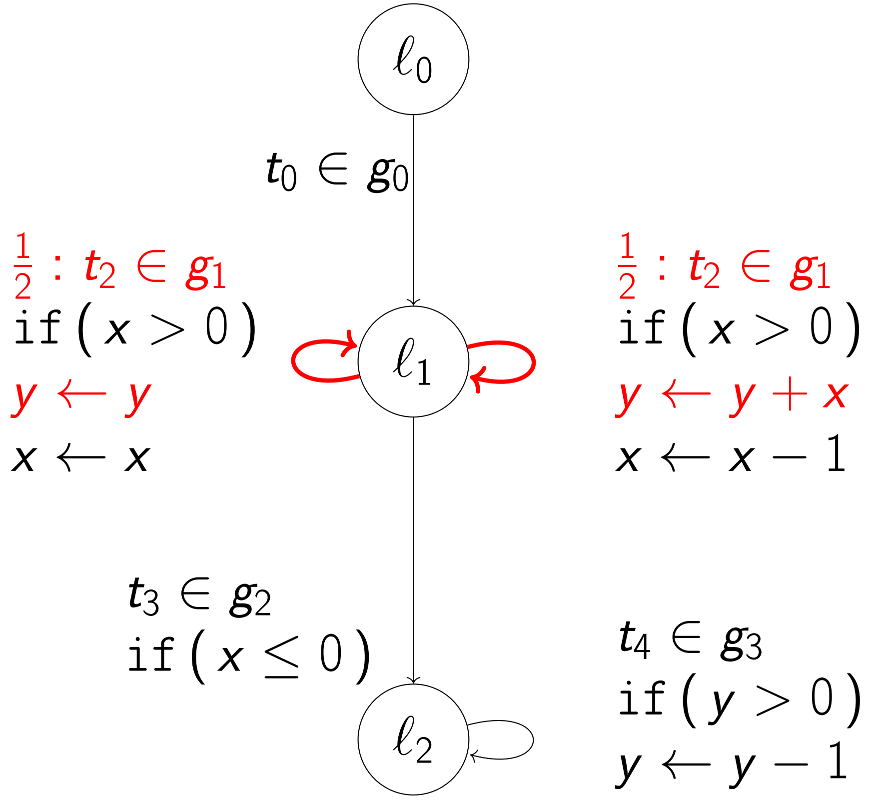


Computing Expected Size Bounds

Computation

Expected size $\mathbb{E}(\mathcal{S}(g_1, y))$:

$$\leq \; y_0 + \mathbb{E}(\mathcal{R}(g_1) \cdot \text{worst-case-change}(g_1, y))$$

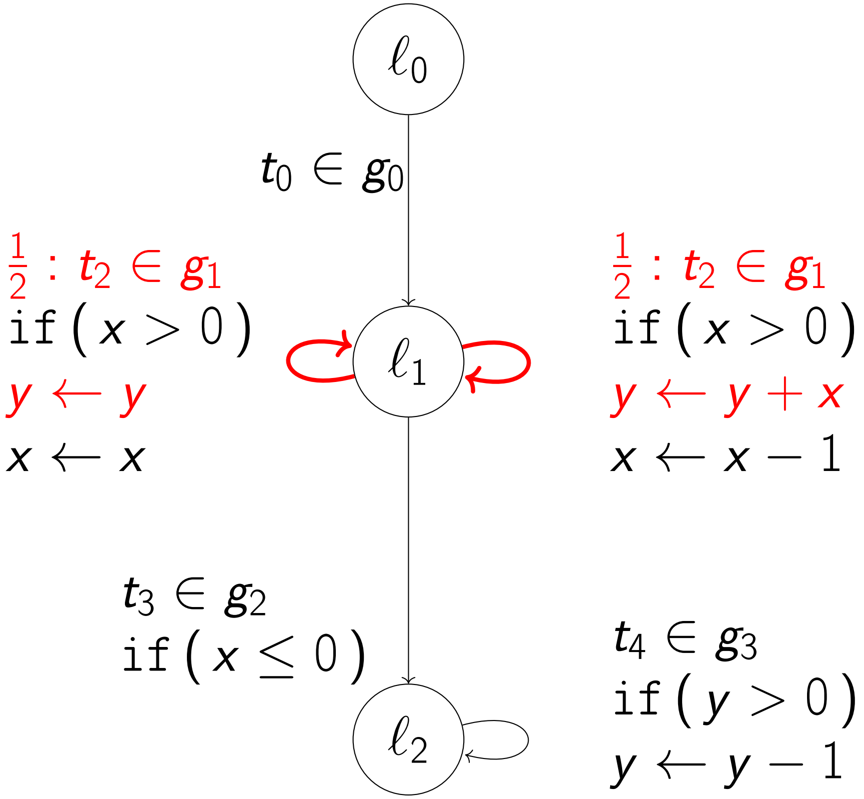


Computing Expected Size Bounds

Computation

Expected size $\mathbb{E}(\mathcal{S}(g_1, y))$:

$$\begin{aligned}
&\le\ y_0 + \mathbb{E}(\mathcal{R}(g_1) \cdot \text{worst-case-change}(g_1, y)) \\
&=\ y_0 + \mathbb{E}(\mathcal{R}(g_1)) \cdot \mathbb{E}(\text{worst-case-change}(g_1, y))
\end{aligned}$$

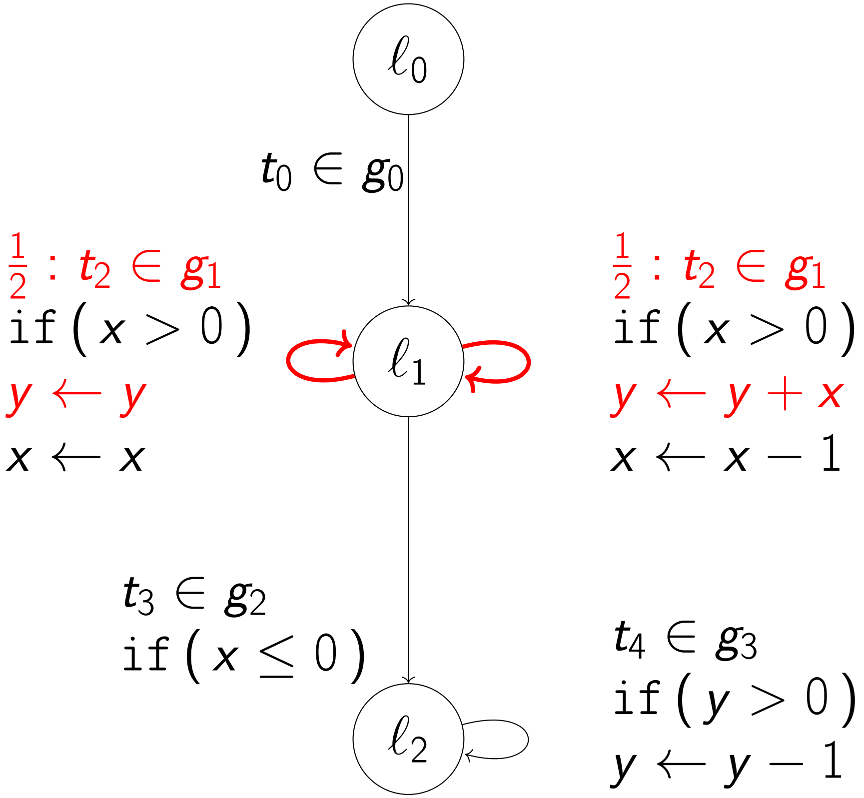


Computing Expected Size Bounds

Computation

Expected size $\mathbb{E}(\mathcal{S}(g_1, y))$:

$$\begin{aligned} &\leq \; y_0 + \mathbb{E}(\mathcal{R}(g_1) \cdot \text{worst-case-change}(g_1, y)) \\ &= \; y_0 + 2 \cdot x_0 \cdot \mathbb{E}(\text{worst-case-change}(g_1, y)) \end{aligned}$$

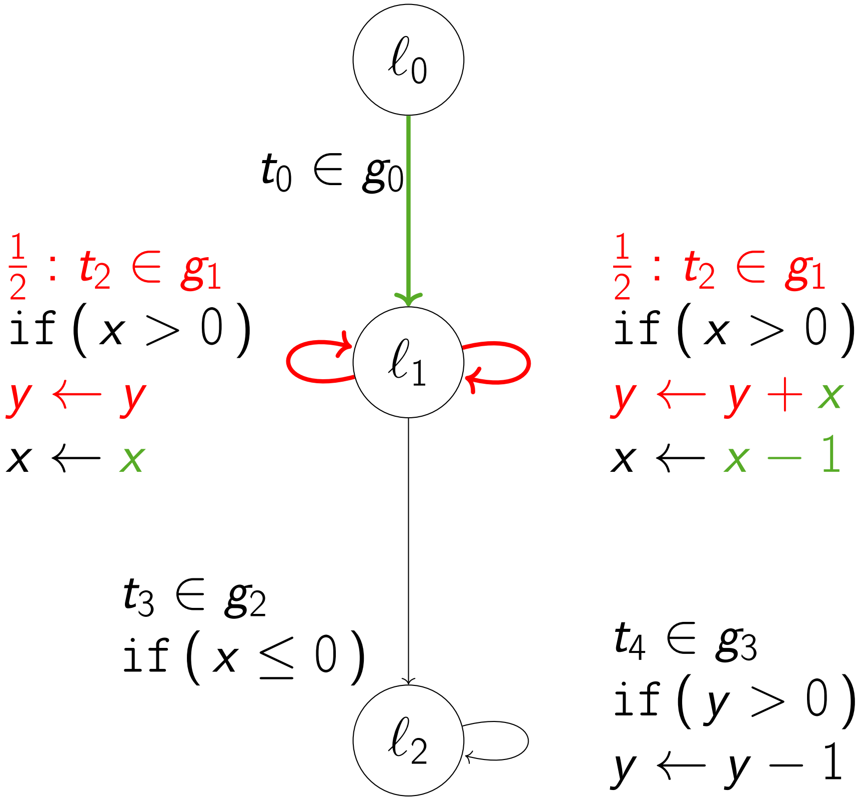


Computing Expected Size Bounds

Computation

Expected size $\mathbb{E}(\mathcal{S}(g_1, y))$:

$$\begin{aligned} &\leq y_0 + \mathbb{E}(\mathcal{R}(g_1) \cdot \text{worst-case-change}(g_1, y)) \\ &= y_0 + 2 \cdot x_0 \cdot \mathbb{E}(\text{worst-case-change}(g_1, y)) \end{aligned}$$

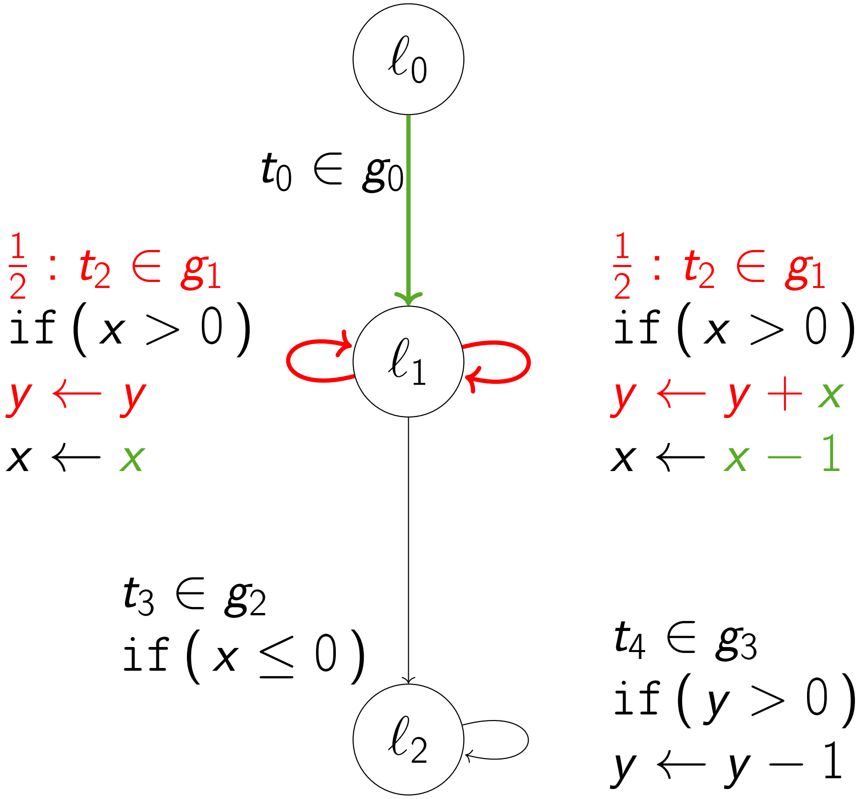


Computing Expected Size Bounds

Computation

Expected size $\mathbb{E}(\mathcal{S}(g_1, y))$:

$$\begin{aligned}&\leq\ y_0 + \mathbb{E}(\mathcal{R}(g_1) \cdot \text{worst-case-change}(g_1, y))\\ &=\ y_0 + 2 \cdot x_0 \cdot \tfrac{x_0}{2}\end{aligned}$$

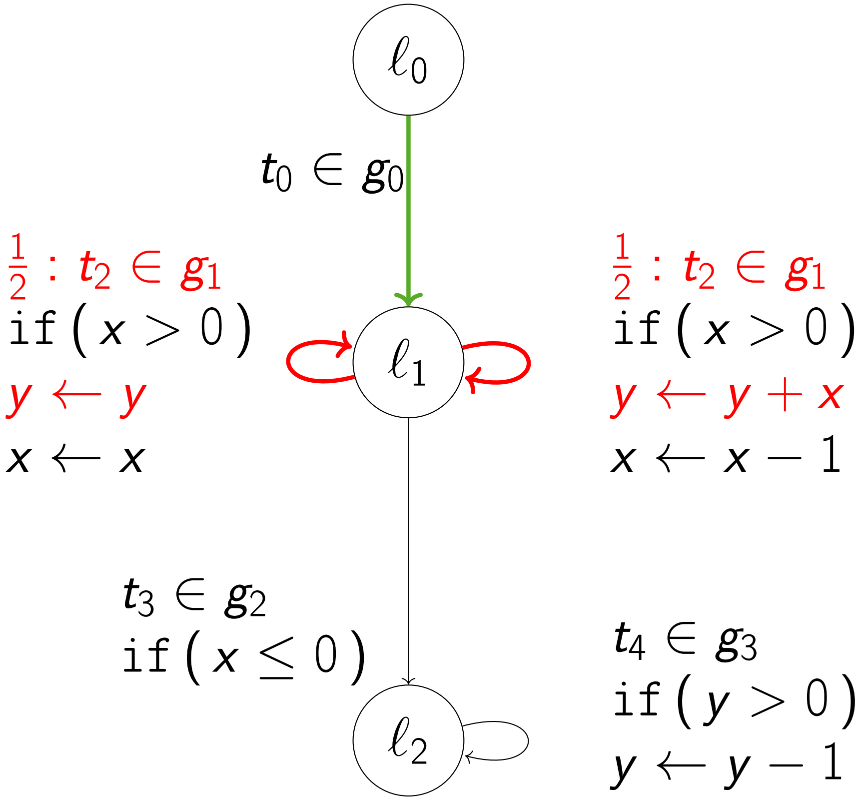


Computing Expected Size Bounds

Computation

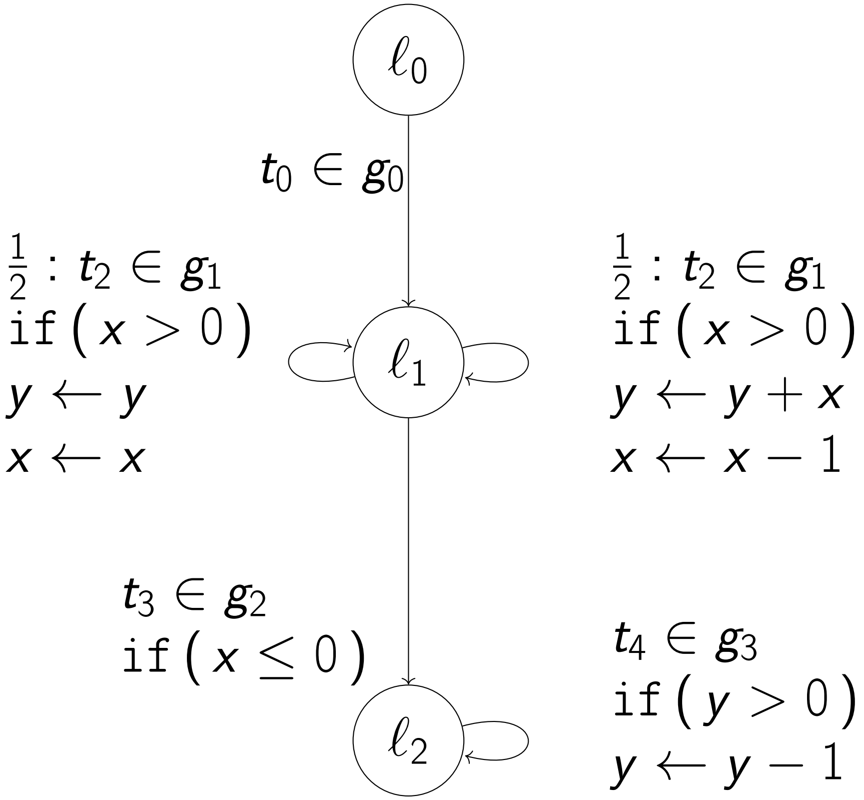
Expected size $\mathbb{E}(\mathcal{S}(g_1, y))$:

$$\begin{aligned} &\leq \; y_0 + \mathbb{E}(\mathcal{R}(g_1) \cdot \text{worst-case-change}(g_1, y)) \\ &= \; y_0 + x_0^2 \end{aligned}$$



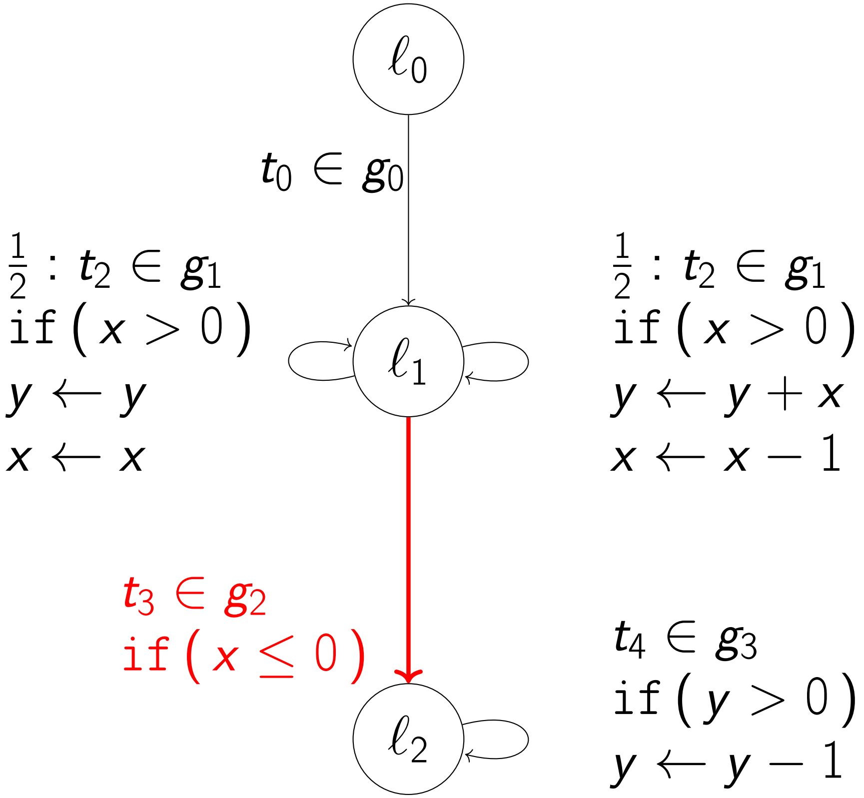
Computing Expected Size Bounds

Computation



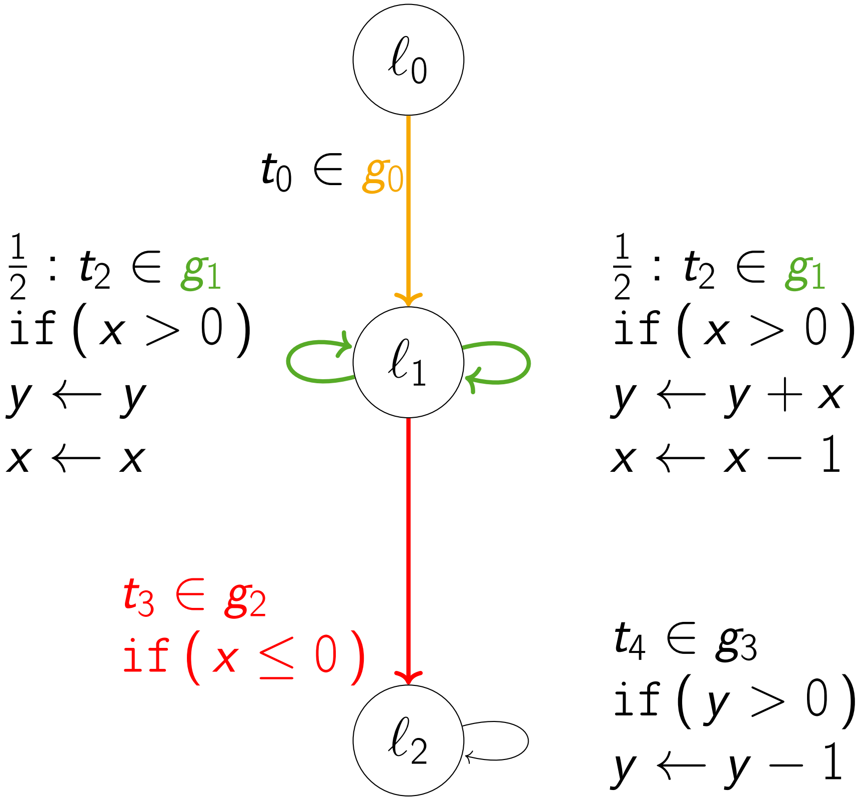
Computing Expected Size Bounds

Computation



Computation

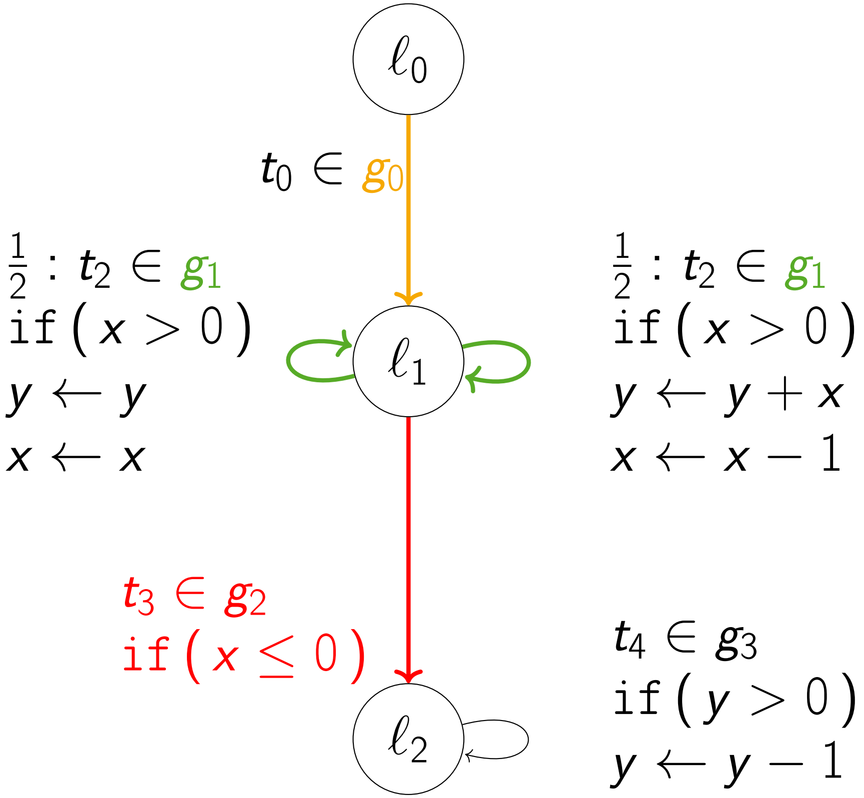
Expected size of v after g_2:



Computation

Expected size of v after g_2:

$\leq$ Maximal expected size of v after g_0, g_1.



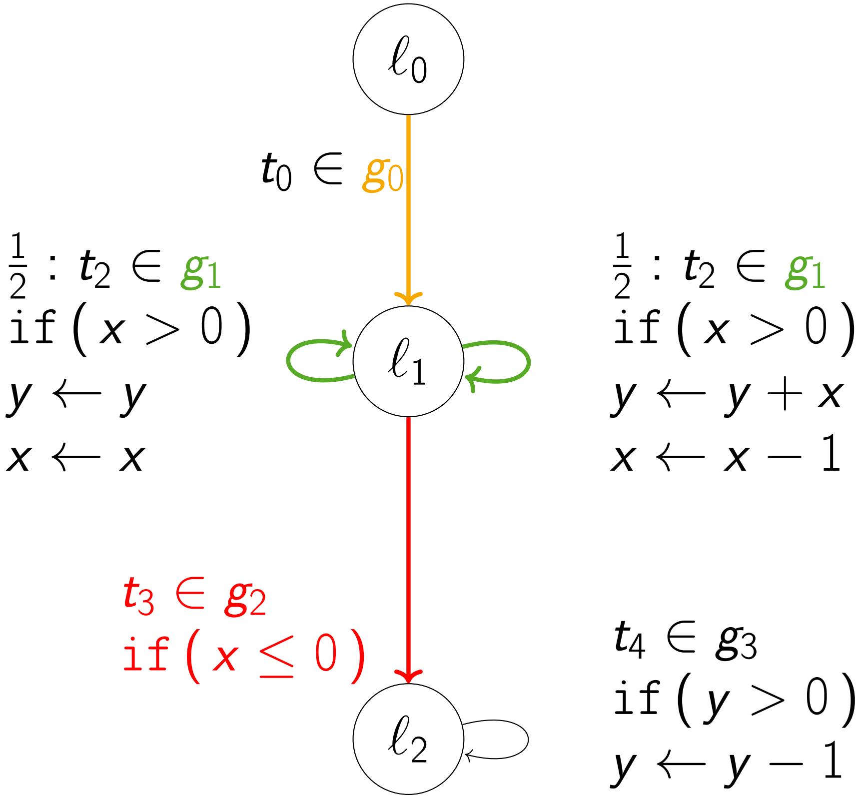
Computing Expected Size Bounds

Computation

Expected size of v after g_2:

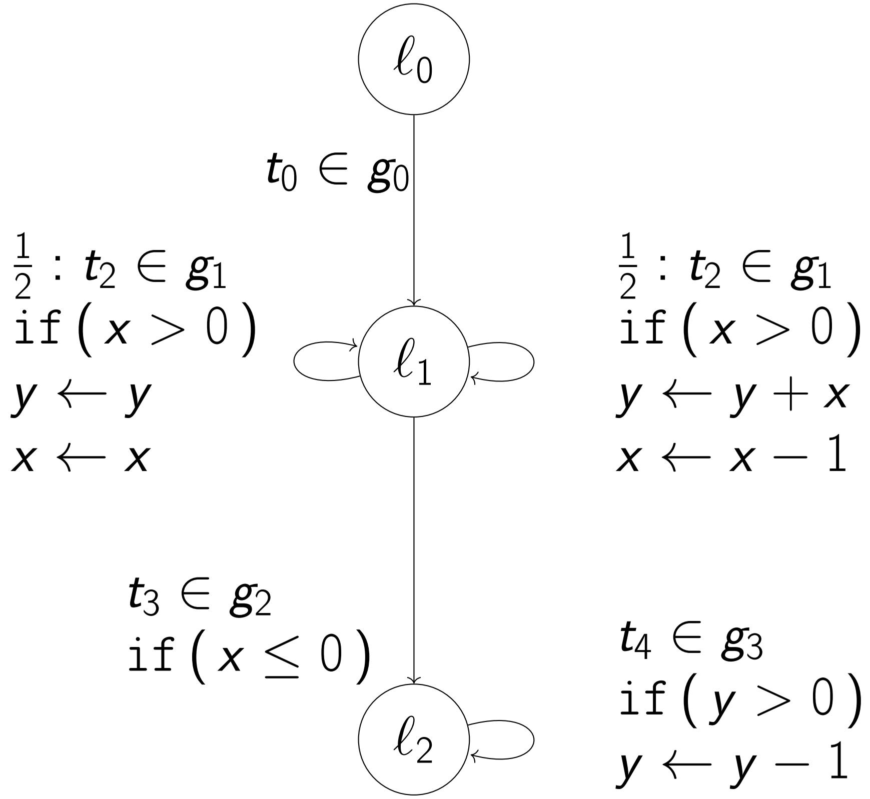
$\leq$ Maximal expected size of v after g_0, g_1.

$\to$ Expected size of y after g_2: $y_0 + x_0^2$.



Computing Expected Size Bounds

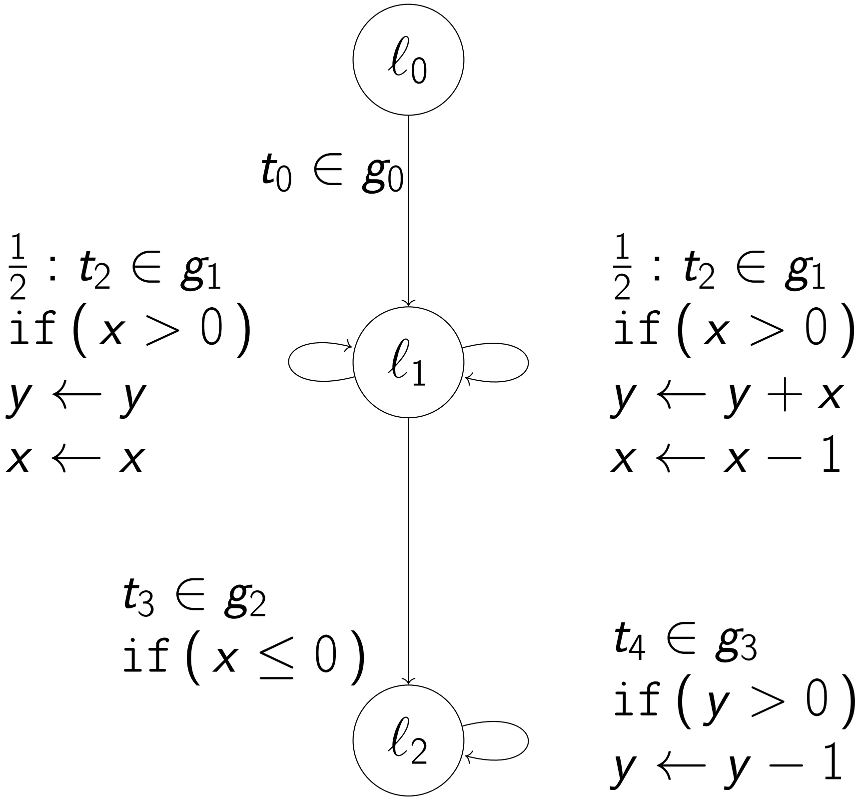
Summary



Computing Expected Size Bounds

Summary

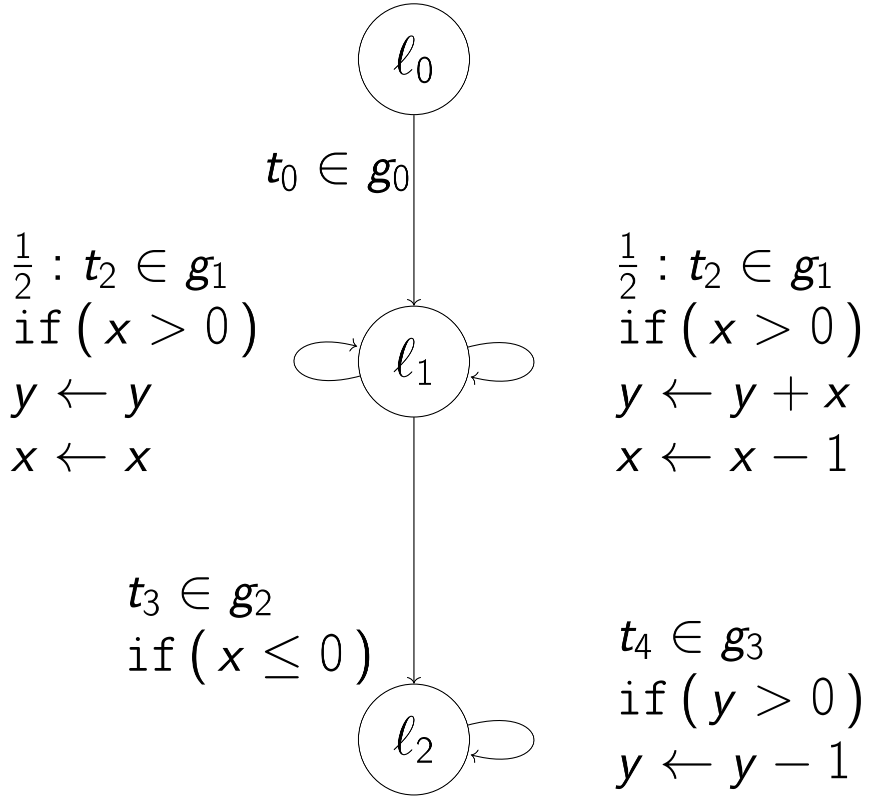
- Automatic inference of expected sizes:



Computing Expected Size Bounds

Summary

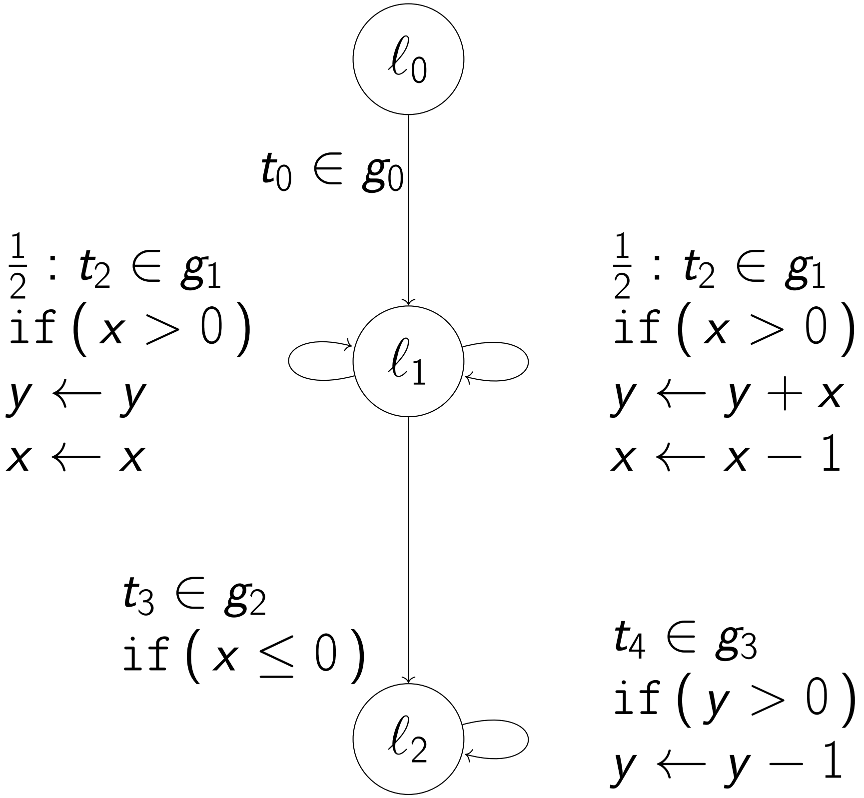
- Automatic inference of expected sizes:
  - $\rightarrow$ Incoming expected sizes.



Computing Expected Size Bounds

Summary

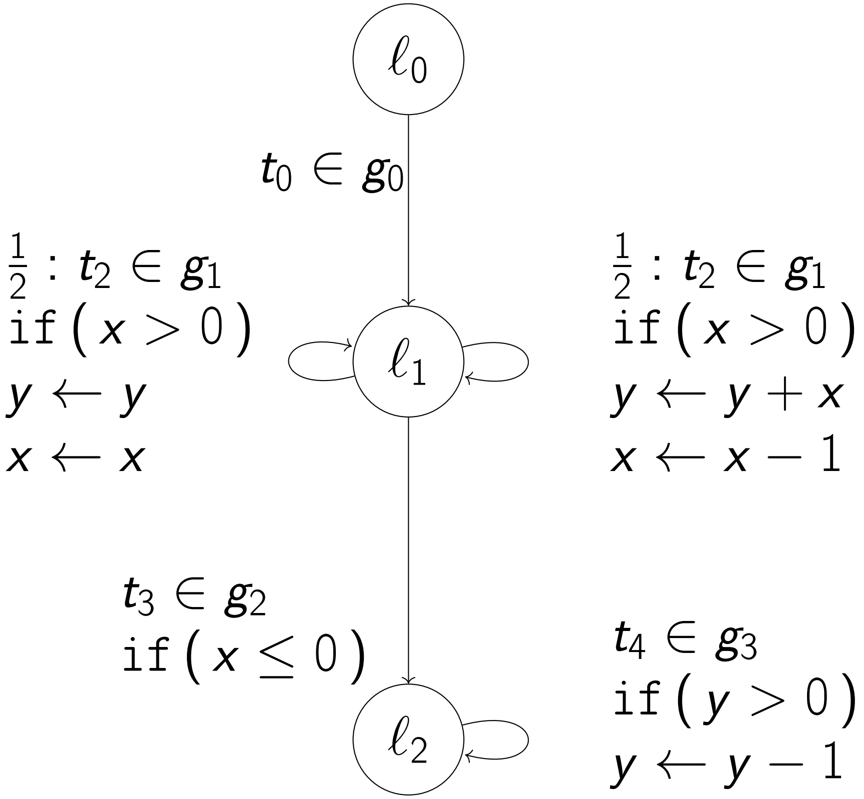
- Automatic inference of expected sizes:
  - $\rightarrow$ Incoming expected sizes.
  - $\rightarrow$ Worst-case expected change.



Computing Expected Size Bounds

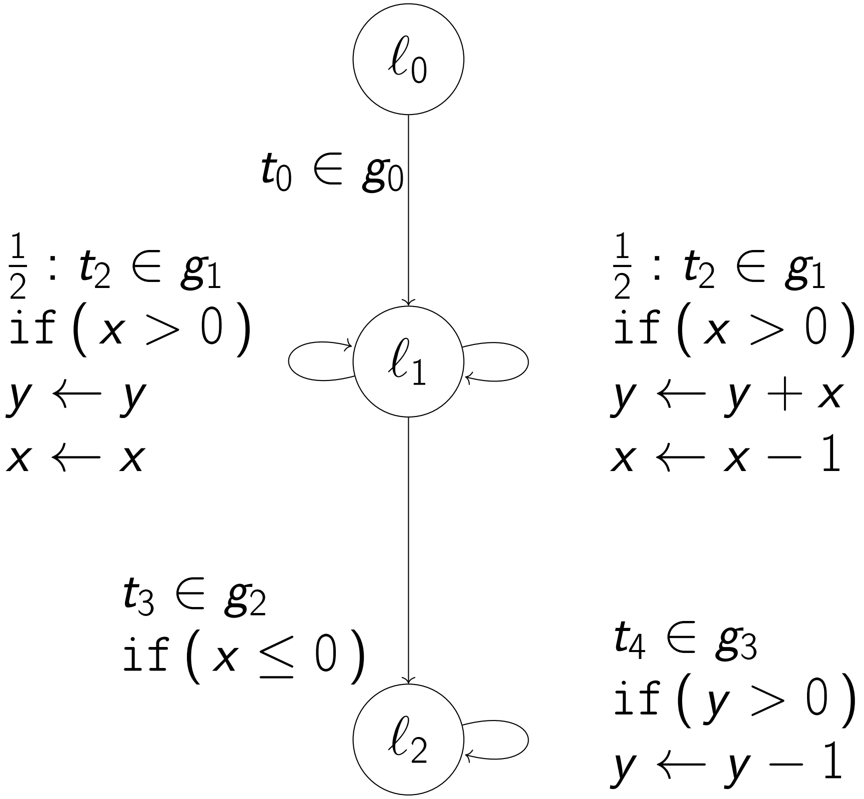
Summary

- Automatic inference of expected sizes:
  - → Incoming expected sizes.
  - → Worst-case expected change.
  - → Expected runtime.



Summary

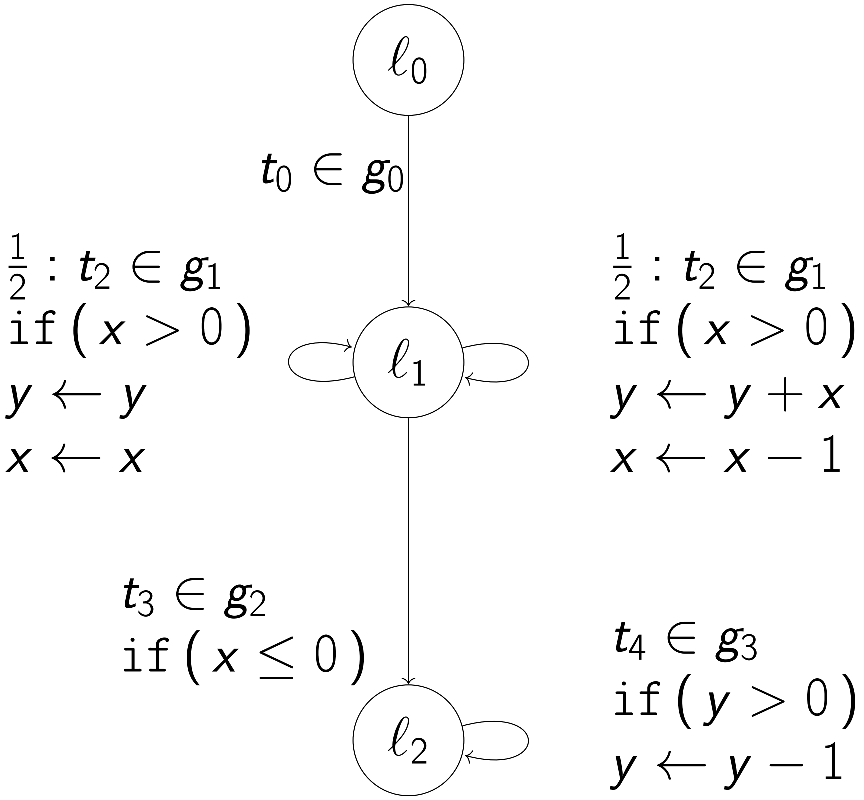
- Automatic inference of expected sizes:
  - → Incoming expected sizes.
  - → Worst-case expected change.
  - → Expected runtime.
- Implementation:



Computing Expected Size Bounds

Summary

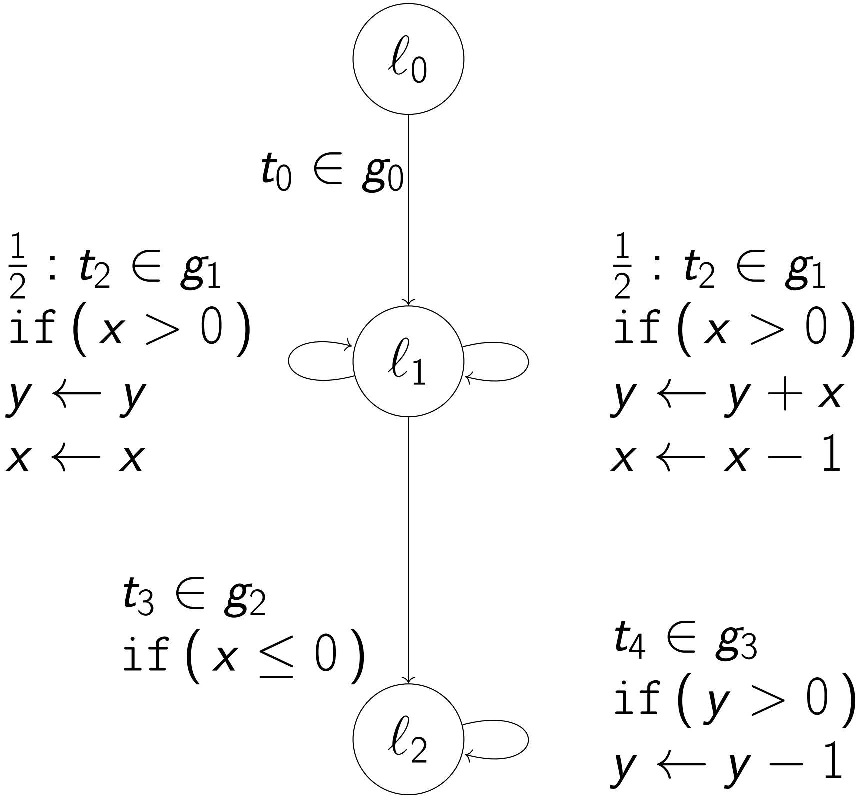
- Automatic inference of expected sizes:
  - → Incoming expected sizes.
  - → Worst-case expected change.
  - → Expected runtime.
- Implementation:
  - → Handles transitions in topological order.



Computing Expected Size Bounds

Summary

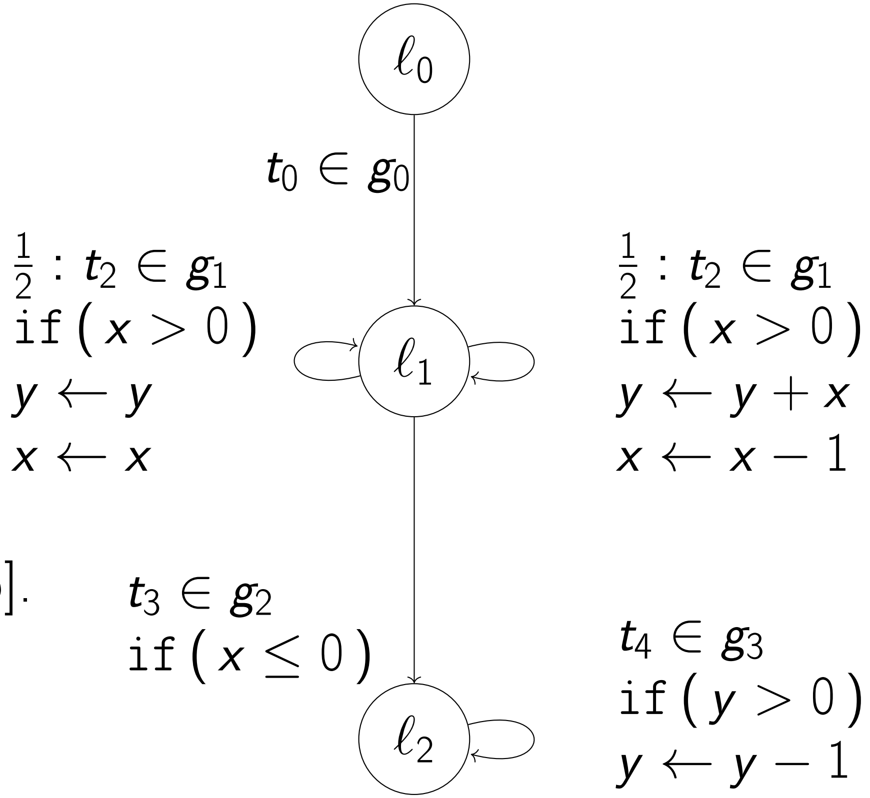
- Automatic inference of expected sizes:
  - → Incoming expected sizes.
  - → Worst-case expected change.
  - → Expected runtime.
- Implementation:
  - → Handles transitions in topological order.
  - → Graph to detect variables influencing expected change.



Computing Expected Size Bounds

Summary

- Automatic inference of expected sizes:
  - → Incoming expected sizes.
  - → Worst-case expected change.
  - → Expected runtime.
- Implementation:
  - → Handles transitions in topological order.
  - → Graph to detect variables influencing expected change.
  - → Uses classical worst-case sizes for these variables [Giesl et al. '16].



Implementation

Experiments

Implementation

- Approach is implemented in KoAT [Giesl et al. '16].

Implementation

- Approach is implemented in KoAT [Giesl et al. '16].
- Uses SMT-solver Z3 [de Moura & Bjørner '08] and abstract domain library Apron [Jeannet & Mine '09].

Experiments

Implementation

- Approach is implemented in KoAT [Giesl et al. '16].
- Uses SMT-solver Z3 [de Moura & Bjørner '08] and abstract domain library Apron [Jeannet & Mine '09].
- Code is open-source, available via Github.

Implementation

- Approach is implemented in KoAT [Giesl et al. '16].
- Uses SMT-solver Z3 [de Moura & Bjørner '08] and abstract domain library Apron [Jeannet & Mine '09].
- Code is open-source, available via Github.
- Provide web interface, Docker image, static binary.

Experiments

Evaluation

Evaluation

- Comparison with existing tools Absynth [Ngo et al. '18] and eco-imp [Avanzini et al. '20].

Evaluation

- Comparison with existing tools Absynth [Ngo et al. '18] and eco-imp [Avanzini et al. '20].
- All 46 benchmarks from [Ngo et al. '18].

Evaluation

- Comparison with existing tools Absynth [Ngo et al. '18] and eco-imp [Avanzini et al. '20].
- All 46 benchmarks from [Ngo et al. '18].
- 29 new examples containing 10 large examples from TPDB enriched with randomization.

Evaluation

- Comparison with existing tools Absynth [Ngo et al. '18] and eco-imp [Avanzini et al. '20].
- All 46 benchmarks from [Ngo et al. '18].
- 29 new examples containing 10 large examples from TPDB enriched with randomization.
- Applied timeout of 5 minutes.

Evaluation

Evaluation

Bound	KoAT	Absynth	eco-imp
$\mathcal{O}(1)$	8	7	8
$\mathcal{O}(n)$	42	35	35
$\mathcal{O}(n^2)$	15	9	15
$\mathcal{O}(n^{>2})$	2	0	0
EXP	1	0	0
∞	7	15	14
TO	0	9	3

Experiments

Evaluation

Bound	KoAT	Absynth	eco-imp
$\mathcal{O}(1)$	8	7	8
$\mathcal{O}(n)$	42	35	35
$\mathcal{O}(n^2)$	15	9	15
$\mathcal{O}(n^{>2})$	2	0	0
EXP	1	0	0
∞	7	15	14
TO	0	9	3

- Successful runs: 91% KoAT, 68% Absynth, 77% eco-imp.

Evaluation

Bound	KoAT	Absynth	eco-imp
$\mathcal{O}(1)$	8	7	8
$\mathcal{O}(n)$	42	35	35
$\mathcal{O}(n^2)$	15	9	15
$\mathcal{O}(n^{>2})$	2	0	0
EXP	1	0	0
∞	7	15	14
TO	0	9	3

- Successful runs: 91% KoAT, 68% Absynth, 77% eco-imp.
- KoAT especially strong on large examples with many loops but only few randomization.

Conclusion

Summary

Conclusion

Summary

- Presented novel modular approach for inferring upper bounds on expected runtimes.

Conclusion

Summary

- Presented novel modular approach for inferring upper bounds on expected runtimes.
- Core idea: alternate computation of bounds on expected runtimes and expected sizes.

Conclusion

Summary

- Presented novel modular approach for inferring upper bounds on expected runtimes.
- Core idea: alternate computation of bounds on expected runtimes and expected sizes.
- Approach showed very good results in experimental evaluation.

Conclusion

Summary

- Presented novel modular approach for inferring upper bounds on expected runtimes.
- Core idea: alternate computation of bounds on expected runtimes and expected sizes.
- Approach showed very good results in experimental evaluation.

Conclusion

Summary

- Presented novel modular approach for inferring upper bounds on expected runtimes.
- Core idea: alternate computation of bounds on expected runtimes and expected sizes.
- Approach showed very good results in experimental evaluation.

Future Work

Conclusion

Summary

- Presented novel modular approach for inferring upper bounds on expected runtimes.
- Core idea: alternate computation of bounds on expected runtimes and expected sizes.
- Approach showed very good results in experimental evaluation.

Future Work

- Switch order of analysis (top-down $\rightarrow$ bottom-up).

Conclusion

Summary

- Presented novel modular approach for inferring upper bounds on expected runtimes.
- Core idea: alternate computation of bounds on expected runtimes and expected sizes.
- Approach showed very good results in experimental evaluation.

Future Work

- Switch order of analysis (top-down $\rightarrow$ bottom-up).
- Generalize approach to cost-bounds.

Conclusion

Summary

- Presented novel modular approach for inferring upper bounds on expected runtimes.
- Core idea: alternate computation of bounds on expected runtimes and expected sizes.
- Approach showed very good results in experimental evaluation.

Future Work

- Switch order of analysis (top-down $\rightarrow$ bottom-up).
- Generalize approach to cost-bounds.
- Combination with underlying approach of Absynth resp. eco-imp.

Conclusion

Summary

- Presented novel modular approach for inferring upper bounds on expected runtimes.
- Core idea: alternate computation of bounds on expected runtimes and expected sizes.
- Approach showed very good results in experimental evaluation.

Future Work

- Switch order of analysis (top-down $\rightarrow$ bottom-up).
- Generalize approach to cost-bounds.
- Combination with underlying approach of Absynth resp. eco-imp.

Conclusion

Summary

- Presented novel modular approach for inferring upper bounds on expected runtimes.
- Core idea: alternate computation of bounds on expected runtimes and expected sizes.
- Approach showed very good results in experimental evaluation.

Future Work

- Switch order of analysis (top-down $\rightarrow$ bottom-up).
- Generalize approach to cost-bounds.
- Combination with underlying approach of Absynth resp. eco-imp.

Thank you