SSA Translation Is an Abstract Interpretation

Matthieu Lemerre
Université Paris-Saclay, CEA LIST
matthieu.lemerre@cea.fr

POPL 2023

SSA Translation VS Abstract Interpretation

SSA *Translation*	:	Source Program $\longrightarrow$ Program in SSA Form
Static analysis based on Abstract Interpretation	:	Source Program $\longrightarrow$ Join Semi-Lattice

SSA Translation VS Abstract Interpretation

SSA *Translation*	:	Source Program $\rightarrow$ Program in SSA Form
Static analysis based on Abstract Interpretation	:	Source Program $\rightarrow$ Join Semi-Lattice

SSA Translation and Analyses are complementary :

- Analyses can improve SSA translation (optimization);
- SSA translation can make analyses faster and/or more precise;

but distinct concepts.

SSA Translation VS Abstract Interpretation

SSA *Translation*	:	Source Program $\rightarrow$ Program in SSA Form
Static analysis based on Abstract Interpretation	:	Source Program $\rightarrow$ Join Semi-Lattice

SSA Translation and Analyses are complementary :

- Analyses can improve SSA translation (optimization);
- SSA translation can make analyses faster and/or more precise;

but distinct concepts... or are they?

SSA Translation ❤ Abstract Interpretation

SSA *Translation*	:	Source Program $\longrightarrow$ Program in SSA Form
Static analysis based on Abstract Interpretation	:	Source Program $\longrightarrow$ Join Semi-Lattice

SSA Translation and Analyses are complementary :

- Analyses can improve SSA translation (optimization);
- SSA translation can make analyses faster and/or more precise;

but distinct concepts… or are they?

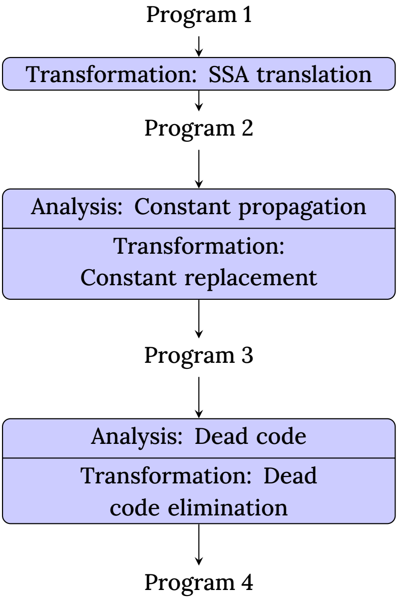
SSA Translation is An Abstract Interpretation

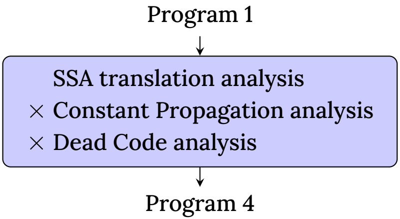
- SSA Translation can be done using a simple efficient dataflow analysis pass

Why is this important?

1. Theory : Better understanding of SSA and its transformation.
   - Simple syntax and semantics for SSA;
   - New algorithm for SSA translation (simple dataflow, no dominance);
   - SSA translation buids on Global Value Numbering (instead of the reverse);
   - Abstract interpretation technique that can be reused on other cyclic terms.
2. Practice :
   - Abstract domains allow modular combination [Cousot&Cousot1979],
   - It is more precise to combine analyses than do them in sequence [Cousot&Cousot1979,Click&Cooper1995]
   - Solves phase ordering problems

Example: Single-pass optimized translation to SSA [Artifact]

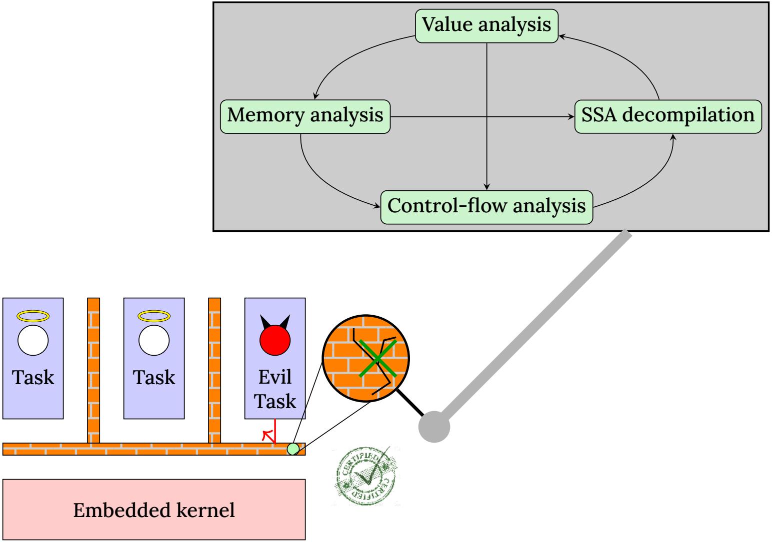




- It is more precise to combine analyses than do them in sequence [Cousot&Cousot1979,Click&Cooper1995]
- No need to write error-prone transformation passes

Example: Single-pass machine code decompilation to SSA

[Nicole, Lemerre, Bardin, Rival 2021]



SSA = Global value graph + control flow information

1. Symbolic expression analysis: computing the Global Value graph

2. SSA Translation: computing the SSA Graph

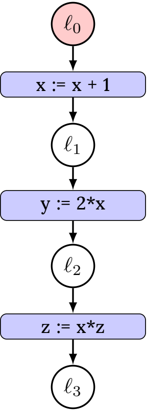
SSA = Global value graph + control flow information

1. Symbolic expression analysis: computing the Global Value graph
2. SSA Translation: computing the SSA Graph

Symbolic execution (≈ single-path SSA translation)

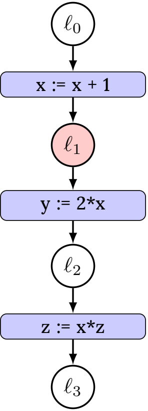
Abstract stores $=$
Program Variables $\rightarrow$ Symbolic Expressions

$$\begin{bmatrix} \mathtt{x} & \mapsto & x_0 \\ \mathtt{y} & \mapsto & y_0 \\ \mathtt{z} & \mapsto & z_0 \end{bmatrix}$$



Symbolic execution (≈ single-path SSA translation)

Abstract stores =
Program Variables → Symbolic Expressions

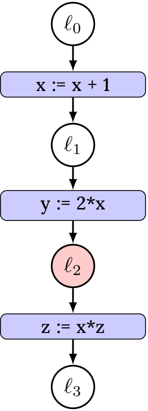


$$\begin{bmatrix} \mathrm{x} & \mapsto & x_0 \\ \mathrm{y} & \mapsto & y_0 \\ \mathrm{z} & \mapsto & z_0 \end{bmatrix}$$

$$\begin{bmatrix} \mathrm{x} & \mapsto & x_0 + 1 \\ \mathrm{y} & \mapsto & y_0 \\ \mathrm{z} & \mapsto & z_0 \end{bmatrix}$$

Symbolic execution (≈ single-path SSA translation)

Abstract stores =
Program Variables → Symbolic Expressions



$$\begin{bmatrix} \mathrm{x} & \mapsto & x_0 \\ \mathrm{y} & \mapsto & y_0 \\ \mathrm{z} & \mapsto & z_0 \end{bmatrix}$$

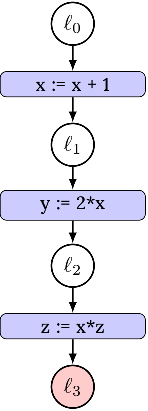
$$\begin{bmatrix} \mathrm{x} & \mapsto & x_0 + 1 \\ \mathrm{y} & \mapsto & y_0 \\ \mathrm{z} & \mapsto & z_0 \end{bmatrix}$$

$$\begin{bmatrix} \mathrm{x} & \mapsto & x_0 + 1 \\ \mathrm{y} & \mapsto & 2 * (x_0 + 1) \\ \mathrm{z} & \mapsto & z_0 \end{bmatrix}$$

Symbolic execution (≈ single-path SSA translation)

Abstract stores $=$

Program Variables $\to$ Symbolic Expressions



$$\begin{bmatrix} x \mapsto x_0 \\ y \mapsto y_0 \\ z \mapsto z_0 \end{bmatrix}$$

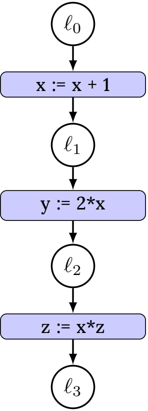
$$\begin{bmatrix} x \mapsto x_0 + 1 \\ y \mapsto y_0 \\ z \mapsto z_0 \end{bmatrix}$$

$$\begin{bmatrix} x \mapsto x_0 + 1 \\ y \mapsto 2 * (x_0 + 1) \\ z \mapsto z_0 \end{bmatrix}$$

$$\begin{bmatrix} x \mapsto x_0 + 1 \\ y \mapsto 2 * (x_0 + 1) \\ z \mapsto (x_0 + 1) * z_0 \end{bmatrix}$$

Symbolic execution (≈ single-path SSA translation)

Abstract stores = Program Variables → Symbolic Expressions



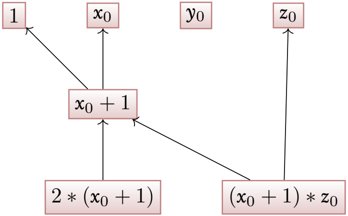
$$\begin{bmatrix} x \mapsto x_0 \\ y \mapsto y_0 \\ z \mapsto z_0 \end{bmatrix}$$

$$\begin{bmatrix} x \mapsto x_0 + 1 \\ y \mapsto y_0 \\ z \mapsto z_0 \end{bmatrix}$$

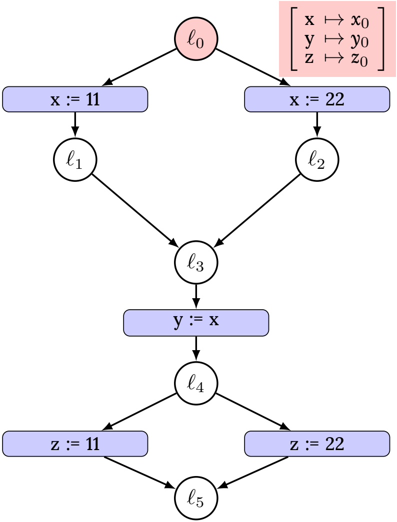
$$\begin{bmatrix} x \mapsto x_0 + 1 \\ y \mapsto 2 * (x_0 + 1) \\ z \mapsto z_0 \end{bmatrix}$$

$$\begin{bmatrix} x \mapsto x_0 + 1 \\ y \mapsto 2 * (x_0 + 1) \\ z \mapsto (x_0 + 1) * z_0 \end{bmatrix}$$

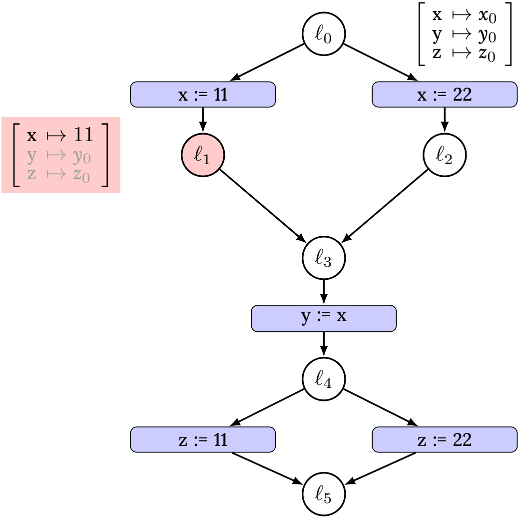
Implicit term graph: the global value graph



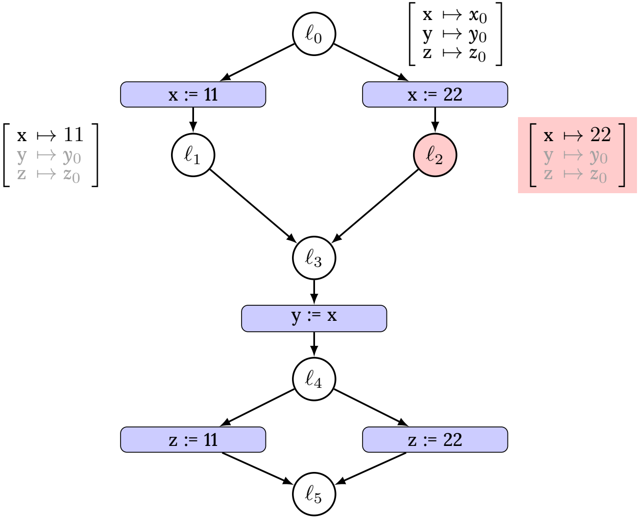
$\sqcup \approx \phi$ + name



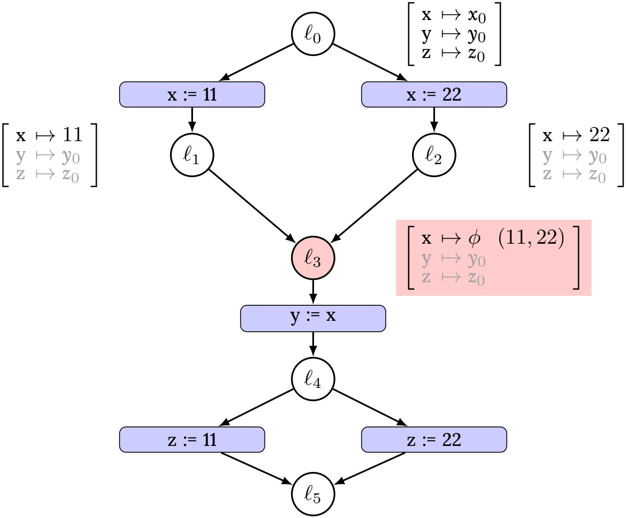
$\sqcup \approx \phi$ + name



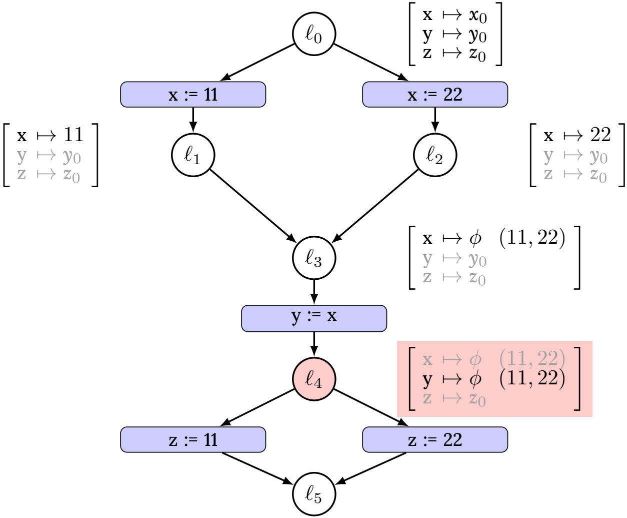
$\sqcup \approx \phi$ + name



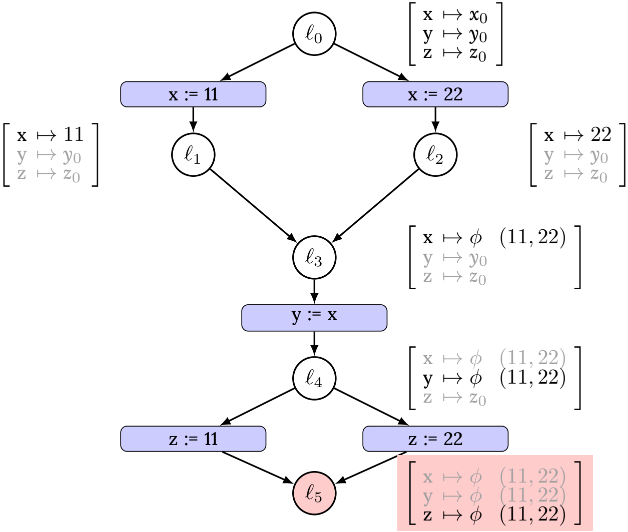
$\sqcup \approx \phi$ + name



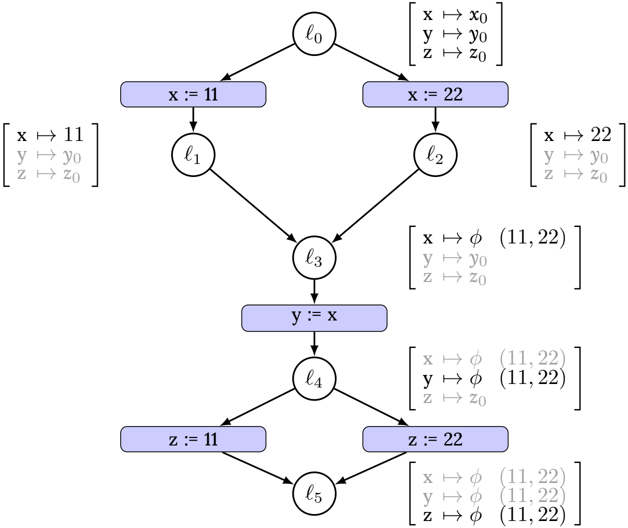
$\sqcup \approx \phi$ + name



$\sqcup \approx \phi$ + name



$\sqcup \approx \phi$ + name

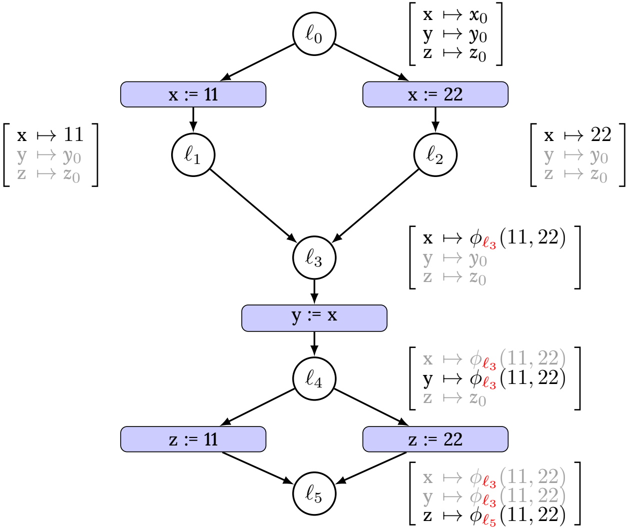


We want:

$\models x = y$

$\not\models x = z$

$\sqcup \approx \phi$ + name



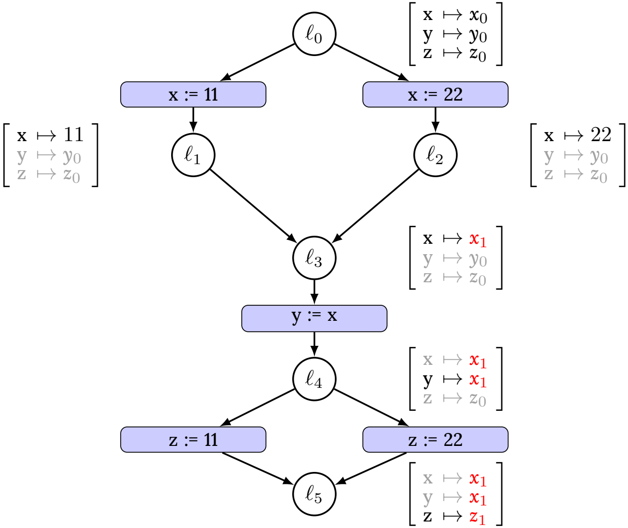
- Either name the ϕ

We want:

$\models x = y$

$\not\models x = z$

$\sqcup \approx \phi$ + name



- Either name the ϕ
- Or name the terms :

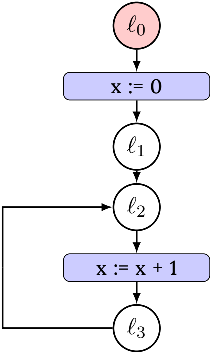
$$\{\, x_1 = \phi(11, 22),\; z_1 = \phi(11, 22) \,\}$$

We want:

$\models x = y$

$\not\models x = z$

Loops

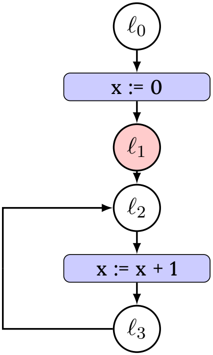


$[\ \mathtt{x} \mapsto x_0\ ]$

$\{$

$\}$

Loops



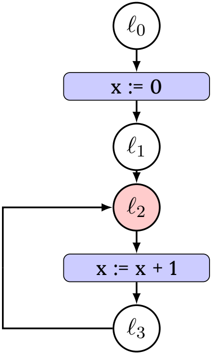
$\left[\, \mathtt{x} \mapsto x_0 \,\right]$

$\left[\, \mathtt{x} \mapsto 0 \,\right]$

$\{$

$\}$

Loops



$[\ \mathrm{x} \mapsto x_0\ ]$

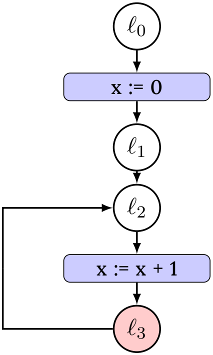
$[\ \mathrm{x} \mapsto 0\ ]$

$[\ \mathrm{x} \mapsto 0\ ]$

$\{$

$\}$

Loops



ℓ_0: $[\ \mathrm{x} \mapsto x_0\ ]$

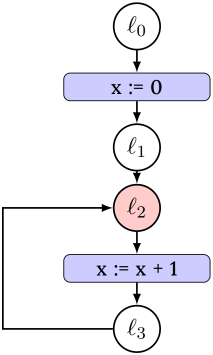
ℓ_1: $[\ \mathrm{x} \mapsto 0\ ]$

ℓ_2: $[\ \mathrm{x} \mapsto 0\ ]$

ℓ_3: $[\ \mathrm{x} \mapsto 0 + 1\ ]$

$\{ \qquad \}$

Loops



$[\ \mathtt{x} \mapsto x_0\ ]$

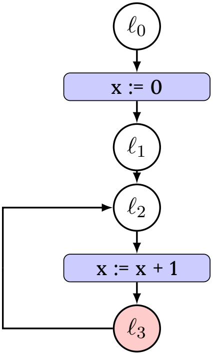
$[\ \mathtt{x} \mapsto 0\ ]$

$[\ \mathtt{x} \mapsto x_1\ ]$

$[\ \mathtt{x} \mapsto 0 + 1\ ]$

$$\{\ x_1 = \phi(0, 0+1),\ \}$$

Loops



$[\, x \mapsto x_0 \,]$

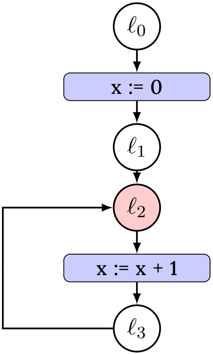
$[\, x \mapsto 0 \,]$

$[\, x \mapsto x_1 \,]$

$[\, x \mapsto x_1 + 1 \,]$

$$\{\, x_1 = \phi(0, 0+1), \quad \}$$

Loops



$[\, x \mapsto x_0 \,]$

$[\, x \mapsto 0 \,]$

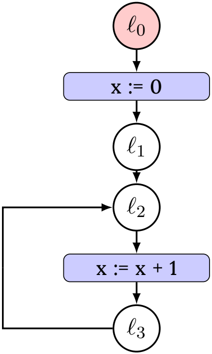
$[\, x \mapsto x_2 \,]$

$[\, x \mapsto x_1 + 1 \,]$

$$\left\{ \begin{array}{l} x_1 = \phi(0, 0+1), \\ x_2 = \phi(0, x_1 + 1), \\ \vdots \qquad \vdots \end{array} \right\}$$

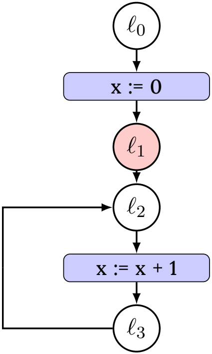
Using fresh variables leads to non-termination.

Loops



$\left[\ \text{x} \mapsto \text{x}_{\ell_0}\ \right]$

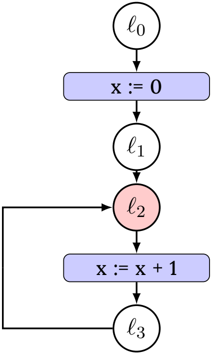
Loops



$[\, x \mapsto x_{\ell_0} \,]$

$[\, x \mapsto 0 \,]$

Loops

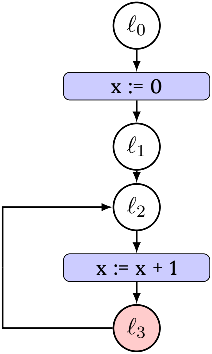


$[\,x \mapsto x_{\ell_0}\,]$

$[\,x \mapsto 0\,]$

$[\,x \mapsto 0\,]$

Loops



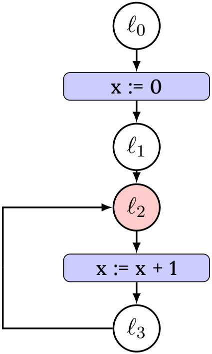
$[\, x \mapsto x_{\ell_0} \,]$

$[\, x \mapsto 0 \,]$

$[\, x \mapsto 0 \,]$

$[\, x \mapsto 0 + 1 \,]$

Loops



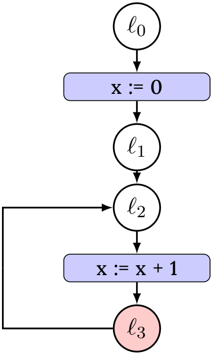
$[\, x \mapsto x_{\ell_0} \,]$

$[\, x \mapsto 0 \,]$

$[\, x \mapsto x_{\ell_2} \,]$

$[\, x \mapsto 0 + 1 \,]$

Loops



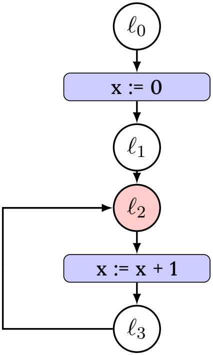
$[\ \mathrm{x} \mapsto \mathtt{x}_{\ell_0}\ ]$

$[\ \mathrm{x} \mapsto 0\ ]$

$[\ \mathrm{x} \mapsto \mathtt{x}_{\ell_2}\ ]$

$[\ \mathrm{x} \mapsto \mathtt{x}_{\ell_2} + 1\ ]$

Loops



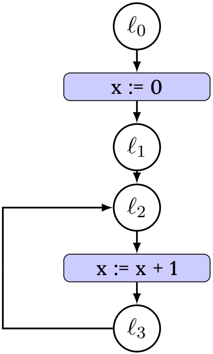
$[\ \mathtt{x} \mapsto x_{\ell_0}\ ]$

$[\ \mathtt{x} \mapsto 0\ ]$

$[\ \mathtt{x} \mapsto x_{\ell_2}\ ]$ (fixpoint reached)

$[\ \mathtt{x} \mapsto x_{\ell_2} + 1\ ]$

Loops



$$[\ x \mapsto x_{\ell_0}\ ]$$

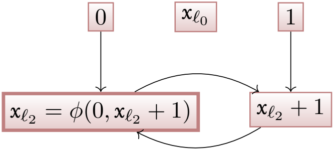
$$[\ x \mapsto 0\ ]$$

$$[\ x \mapsto x_{\ell_2}\ ]$$

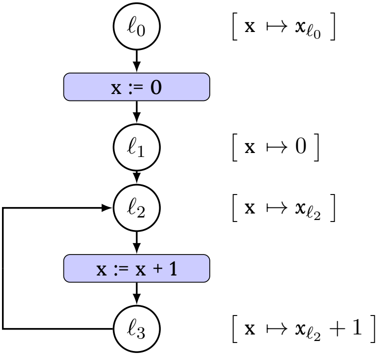
$$[\ x \mapsto x_{\ell_2} + 1\ ]$$

Implicit *cyclic* term graph:

$$\{\ x_{\ell_2} = \phi(0, x_{\ell_2} + 1)\}$$

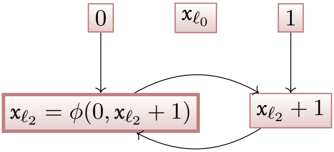


Loops



Implicit *cyclic* term graph:

$$\{\ x_{\ell_2} = \phi(0, x_{\ell_2} + 1)\}$$



- The global value graph is a cyclic term graph [Ariola&Klop 1996]
  - Symbolic (recursion) variables used for both non-determinism and termination
- We name symbolic variables using the names of control locations
  - Control locations = recursion variables in the CFG viewed as a cyclic term

Meaning, soundness and completeness

$$\gamma : \text{Abstract store} \rightarrow (\text{Valuation of symbolic variables} \rightarrow \text{Concrete store})$$

$$\gamma(\sigma^\sharp) = [\Gamma \in \text{Valuation} \mapsto \text{evaluate the symbolic expressions in } \sigma^\sharp \text{ using } \Gamma]$$

Example

$$\gamma\left(\begin{bmatrix} x \mapsto 2 * x_{\ell_0} \\ y \mapsto 2 * x_{\ell_0} + 1 \end{bmatrix}\right) = \left\{[x_{\ell_0} \mapsto 0] \mapsto \begin{bmatrix} x \mapsto 0 \\ y \mapsto 1 \end{bmatrix}, [x_{\ell_0} \mapsto 1] \mapsto \begin{bmatrix} x \mapsto 2 \\ y \mapsto 3 \end{bmatrix}, \ldots\right\}$$

Meaning, soundness and completeness

$$\gamma : \text{Abstract store} \rightarrow (\text{Valuation of symbolic variables} \rightarrow \text{Concrete store})$$

$$\gamma(\sigma^\sharp) = [\Gamma \in \text{Valuation} \mapsto \text{evaluate the symbolic expressions in } \sigma^\sharp \text{ using } \Gamma]$$

Example

$$\gamma\left(\begin{bmatrix} x \mapsto 2 * x_{\ell_0} \\ y \mapsto 2 * x_{\ell_0} + 1 \end{bmatrix}\right) = \left\{[x_{\ell_0} \mapsto 0] \mapsto \begin{bmatrix} x \mapsto 0 \\ y \mapsto 1 \end{bmatrix}, [x_{\ell_0} \mapsto 1] \mapsto \begin{bmatrix} x \mapsto 2 \\ y \mapsto 3 \end{bmatrix}, \ldots\right\}$$

Abstract stores can represent state properties and abstract transformations:

$$\begin{bmatrix} \mathsf{x} \mapsto 2 * x_{\ell_0} \\ \mathsf{y} \mapsto 2 * x_{\ell_0} + 1 \end{bmatrix} \models \mathsf{x} \text{ is even} \land \mathsf{y} \text{ is odd} \land \mathsf{y} - \mathsf{x} = 1 \land \mathsf{x} = 2 * \mathsf{old}(\mathsf{x})$$

Meaning, soundness and completeness

$$\gamma : \text{Abstract store} \to (\text{Valuation of symbolic variables} \to \text{Concrete store})$$

$$\gamma(\sigma^\sharp) = [\Gamma \in \text{Valuation} \mapsto \text{evaluate the symbolic expressions in } \sigma^\sharp \text{ using } \Gamma]$$

Example

$$\gamma\left(\begin{bmatrix} x \mapsto 2 * x_{\ell_0} \\ y \mapsto 2 * x_{\ell_0} + 1 \end{bmatrix}\right) = \left\{ [x_{\ell_0} \mapsto 0] \mapsto \begin{bmatrix} x \mapsto 0 \\ y \mapsto 1 \end{bmatrix}, [x_{\ell_0} \mapsto 1] \mapsto \begin{bmatrix} x \mapsto 2 \\ y \mapsto 3 \end{bmatrix}, \ldots \right\}$$

Abstract stores can represent state properties and abstract transformations:

$$\begin{bmatrix} \mathsf{x} \mapsto 2 * x_{\ell_0} \\ \mathsf{y} \mapsto 2 * x_{\ell_0} + 1 \end{bmatrix} \models \mathsf{x} \text{ is even} \land \mathsf{y} \text{ is odd} \land \mathsf{y} - \mathsf{x} = 1 \land \mathsf{x} = 2 * \mathsf{old}(\mathsf{x})$$

Theorem (The symbolic expression analysis is sound)

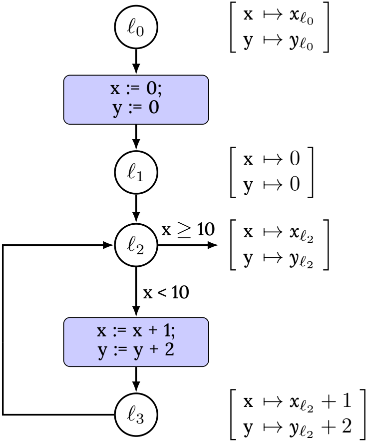
Assignments are sound and complete, guards incomplete, join is incomplete and unsound.

SSA = Global value graph + control flow information

1. Symbolic expression analysis: computing the Global Value graph

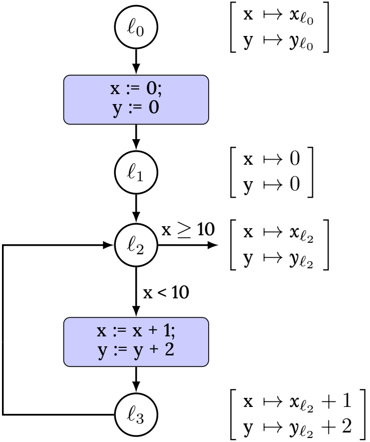
2. SSA Translation: computing the SSA Graph

Where do we loose precision? Absence of control flow

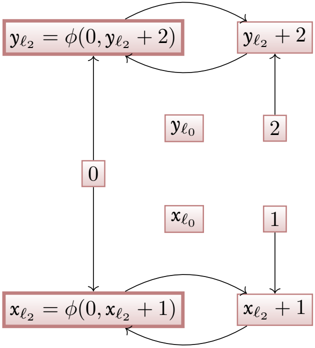


- Origin of symbolic variables is lost.

Where do we loose precision? Absence of control flow

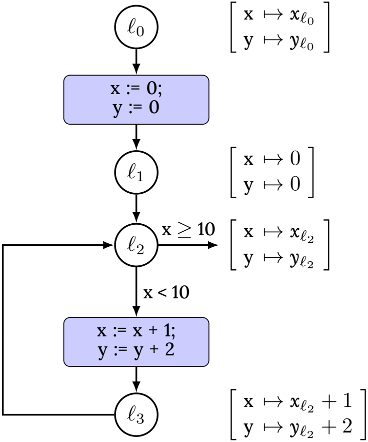


$$\{\, x_{\ell_2} = \phi(0, x_{\ell_2} + 1),\; y_{\ell_2} = \phi(0, y_{\ell_2} + 2) \,\}$$

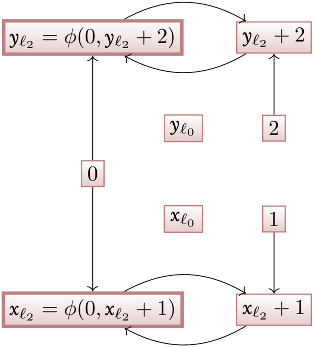


- Origin of symbolic variables is lost.
  - Value graph recovers some information, but variables are independent.

Where do we loose precision? Absence of control flow

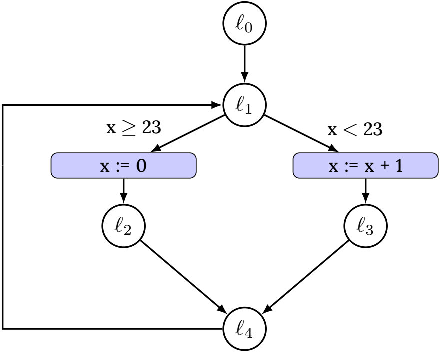


$$\{\, x_{\ell_2} = \phi(0, x_{\ell_2} + 1), \\ y_{\ell_2} = \phi(0, y_{\ell_2} + 2) \}$$



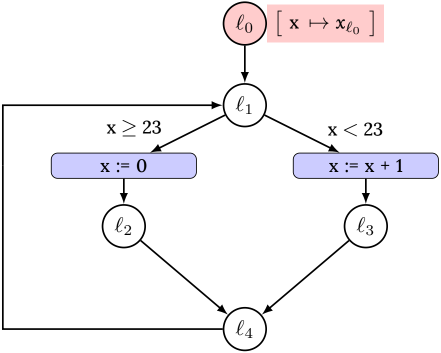
- Origin of symbolic variables is lost.
  - Value graph recovers some information, but variables are independent.
- Guards are completely ignored.

Computing the SSA graph : SSA translation by dataflow analysis



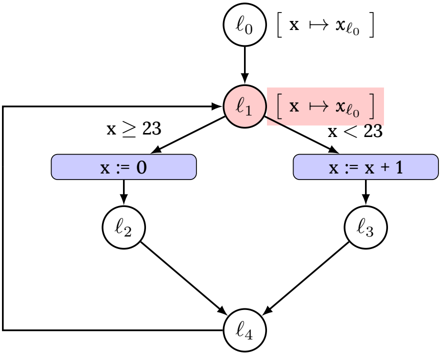
- Create the SSA graph together with the symbolic expression analysis.

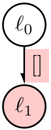
Computing the SSA graph : SSA translation by dataflow analysis



- Create the SSA graph together with the symbolic expression analysis.

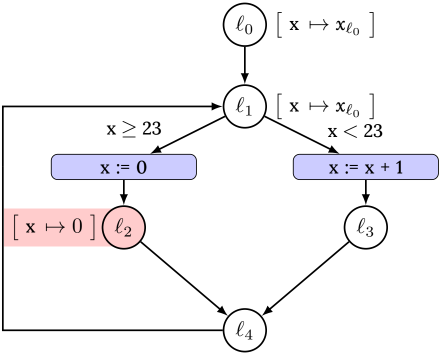
Computing the SSA graph : SSA translation by dataflow analysis

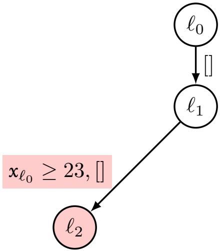




- Optimistically assume that unvisited edges are dead, revise later.

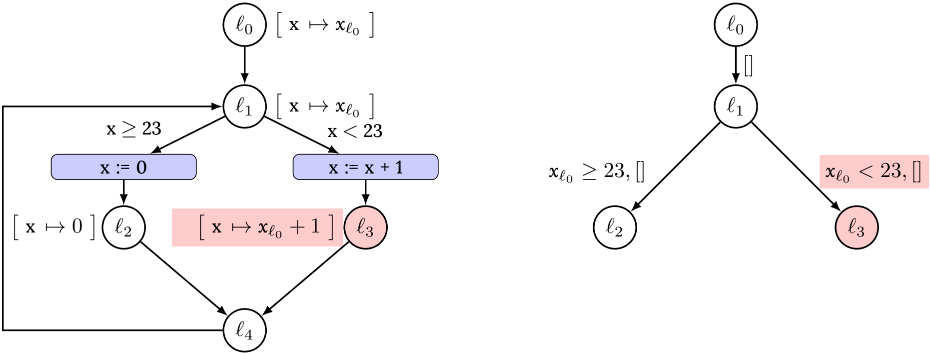
Computing the SSA graph : SSA translation by dataflow analysis





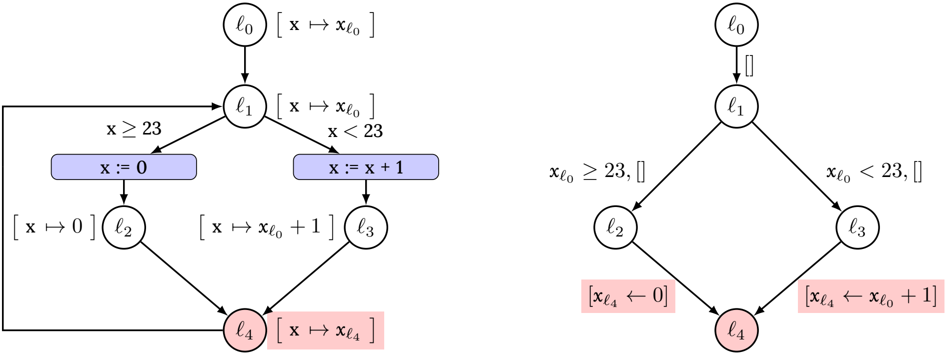
- Translate guard expressions into symbolic expressions.

Computing the SSA graph : SSA translation by dataflow analysis



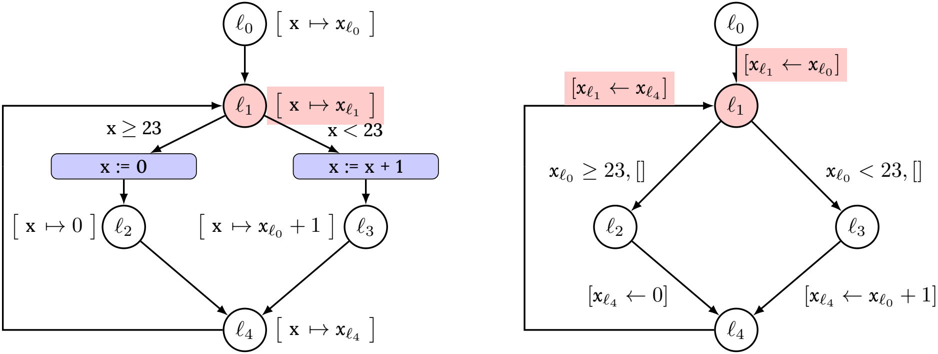
- Translate guard expressions into symbolic expressions.

Computing the SSA graph : SSA translation by dataflow analysis



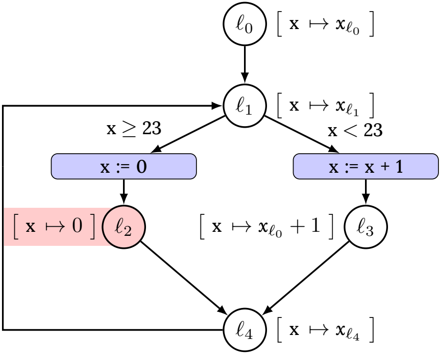
- Creating a variable in symbolic expression is compensated by adding bindings in the SSA graph.

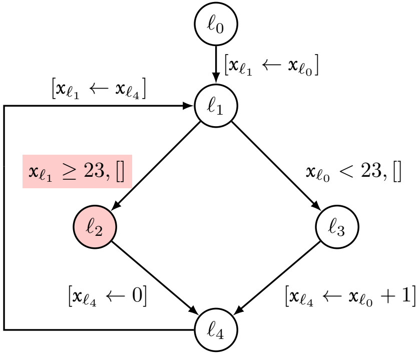
Computing the SSA graph : SSA translation by dataflow analysis



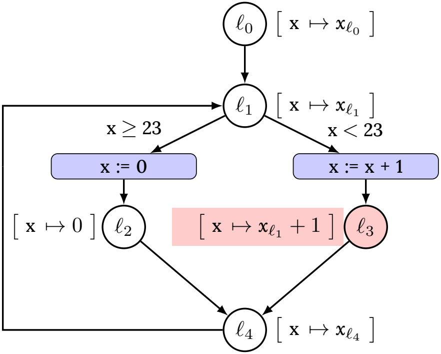
- Adding new edges and revision of the SSA graph.
  Here the SSA graph is wrong (neither sound nor complete). In the paper: soundness proof + technique to have completeness at every step.

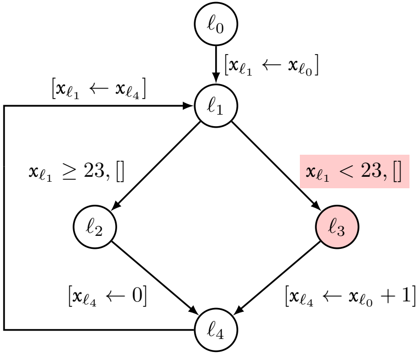
Computing the SSA graph : SSA translation by dataflow analysis





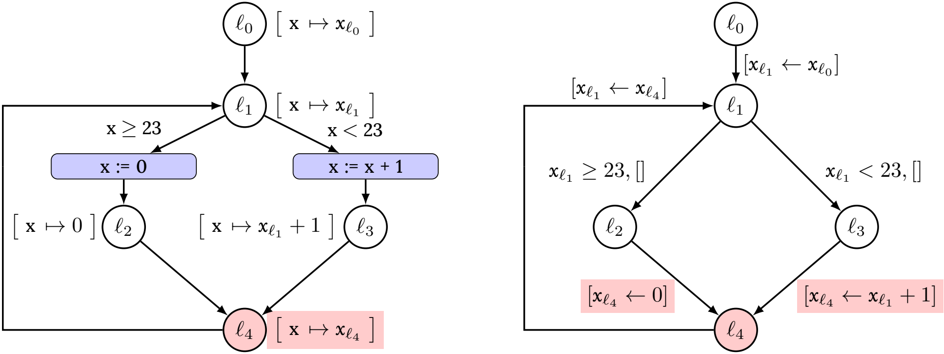
- Adding new edges and revision of the SSA graph.
  Here the SSA graph is wrong (neither sound nor complete). In the paper: soundness proof + technique to have completeness at every step.





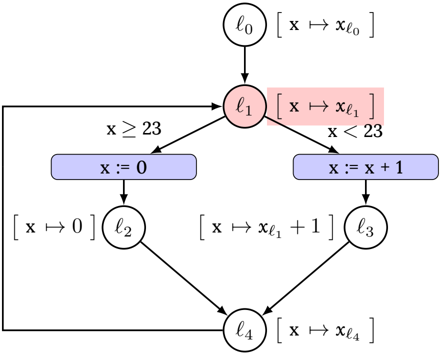
- Adding new edges and revision of the SSA graph.
  Here the SSA graph is wrong (neither sound nor complete). In the paper: soundness proof + technique to have completeness at every step.

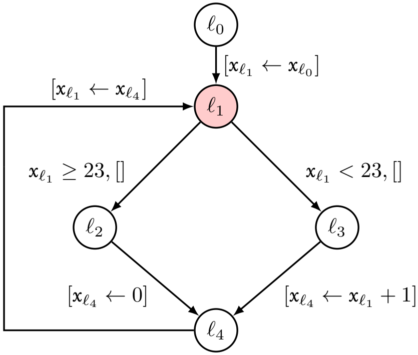
Computing the SSA graph : SSA translation by dataflow analysis



- Adding new edges and revision of the SSA graph.
  Here the SSA graph is wrong (neither sound nor complete). In the paper: soundness proof + technique to have completeness at every step.

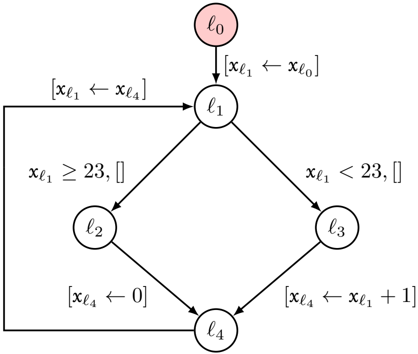
Computing the SSA graph : SSA translation by dataflow analysis





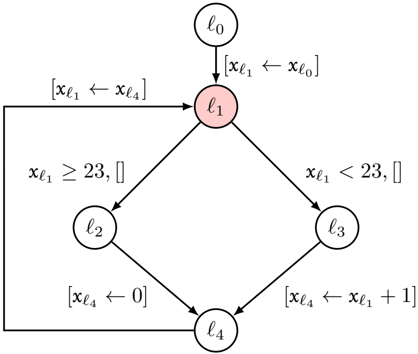
- Fixpoint reached, our SSA graph is sound and complete.

Operational semantics of the SSA graph



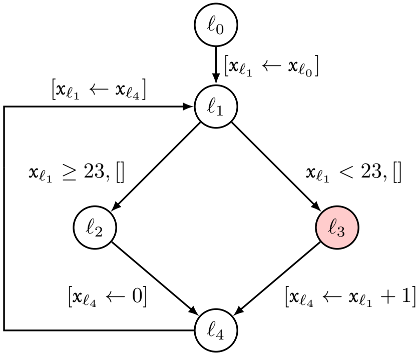
$$(\ell_0, [\, x_{\ell_0} \mapsto 2 \,]) \rightsquigarrow \left(\ell_1, \begin{bmatrix} x_{\ell_0} \mapsto 2 \\ x_{\ell_1} \mapsto 2 \end{bmatrix}\right) \rightsquigarrow \left(\ell_3, \begin{bmatrix} x_{\ell_0} \mapsto 2 \\ x_{\ell_1} \mapsto 2 \end{bmatrix}\right) \rightsquigarrow \left(\ell_4, \begin{bmatrix} x_{\ell_0} \mapsto 2 \\ x_{\ell_1} \mapsto 2 \\ x_{\ell_4} \mapsto 3 \end{bmatrix}\right) \rightsquigarrow \left(\ell_1, \begin{bmatrix} x_{\ell_0} \mapsto 2 \\ x_{\ell_1} \mapsto 3 \end{bmatrix}\right)$$

Operational semantics of the SSA graph



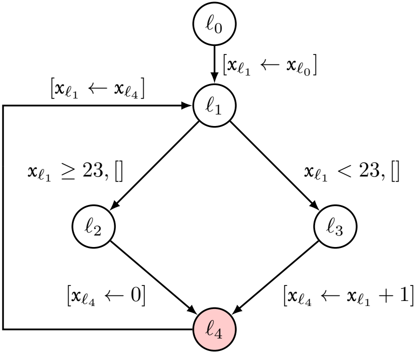
$$(\ell_0, [\, \mathsf{x}_{\ell_0} \mapsto 2 \,]) \leadsto \left(\ell_1, \begin{bmatrix} \mathsf{x}_{\ell_0} \mapsto 2 \\ \mathsf{x}_{\ell_1} \mapsto 2 \end{bmatrix}\right) \leadsto \left(\ell_3, \begin{bmatrix} \mathsf{x}_{\ell_0} \mapsto 2 \\ \mathsf{x}_{\ell_1} \mapsto 2 \end{bmatrix}\right) \leadsto \left(\ell_4, \begin{bmatrix} \mathsf{x}_{\ell_0} \mapsto 2 \\ \mathsf{x}_{\ell_1} \mapsto 2 \\ \mathsf{x}_{\ell_4} \mapsto 3 \end{bmatrix}\right) \leadsto \left(\ell_1, \begin{bmatrix} \mathsf{x}_{\ell_0} \mapsto 2 \\ \mathsf{x}_{\ell_1} \mapsto 3 \end{bmatrix}\right)$$

Operational semantics of the SSA graph



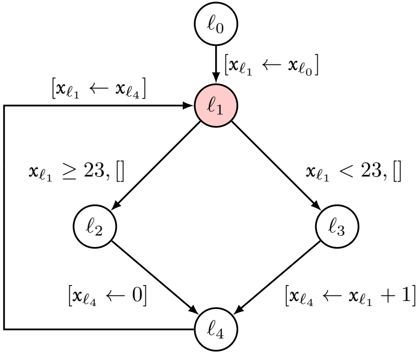
$$(\ell_0, [\, x_{\ell_0} \mapsto 2 \,]) \rightsquigarrow \left(\ell_1, \begin{bmatrix} x_{\ell_0} \mapsto 2 \\ x_{\ell_1} \mapsto 2 \end{bmatrix}\right) \rightsquigarrow \left(\ell_3, \begin{bmatrix} x_{\ell_0} \mapsto 2 \\ x_{\ell_1} \mapsto 2 \end{bmatrix}\right) \rightsquigarrow \left(\ell_4, \begin{bmatrix} x_{\ell_0} \mapsto 2 \\ x_{\ell_1} \mapsto 2 \\ x_{\ell_4} \mapsto 3 \end{bmatrix}\right) \rightsquigarrow \left(\ell_1, \begin{bmatrix} x_{\ell_0} \mapsto 2 \\ x_{\ell_1} \mapsto 3 \end{bmatrix}\right)$$

Operational semantics of the SSA graph



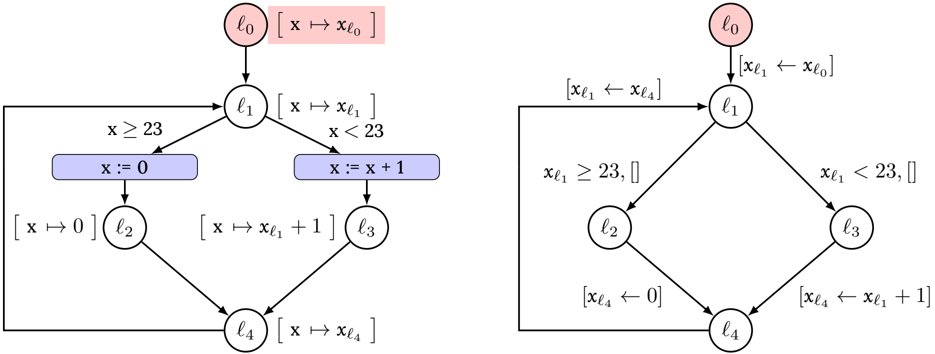
$$(\ell_0, [\, x_{\ell_0} \mapsto 2 \,]) \rightsquigarrow \left(\ell_1, \begin{bmatrix} x_{\ell_0} \mapsto 2 \\ x_{\ell_1} \mapsto 2 \end{bmatrix}\right) \rightsquigarrow \left(\ell_3, \begin{bmatrix} x_{\ell_0} \mapsto 2 \\ x_{\ell_1} \mapsto 2 \end{bmatrix}\right) \rightsquigarrow \left(\ell_4, \begin{bmatrix} x_{\ell_0} \mapsto 2 \\ x_{\ell_1} \mapsto 2 \\ x_{\ell_4} \mapsto 3 \end{bmatrix}\right) \rightsquigarrow \left(\ell_1, \begin{bmatrix} x_{\ell_0} \mapsto 2 \\ x_{\ell_1} \mapsto 3 \end{bmatrix}\right)$$

Operational semantics of the SSA graph



$$(\ell_0, [\, x_{\ell_0} \mapsto 2 \,]) \rightsquigarrow \left(\ell_1, \begin{bmatrix} x_{\ell_0} \mapsto 2 \\ x_{\ell_1} \mapsto 2 \end{bmatrix}\right) \rightsquigarrow \left(\ell_3, \begin{bmatrix} x_{\ell_0} \mapsto 2 \\ x_{\ell_1} \mapsto 2 \end{bmatrix}\right) \rightsquigarrow \left(\ell_4, \begin{bmatrix} x_{\ell_0} \mapsto 2 \\ x_{\ell_1} \mapsto 2 \\ x_{\ell_4} \mapsto 3 \end{bmatrix}\right) \rightsquigarrow \left(\ell_1, \begin{bmatrix} x_{\ell_0} \mapsto 2 \\ x_{\ell_1} \mapsto 3 \end{bmatrix}\right)$$

Bisimulation & homomorphism between transition systems

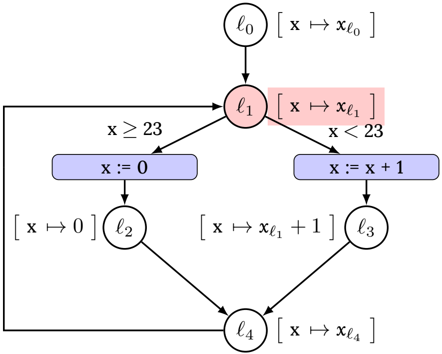


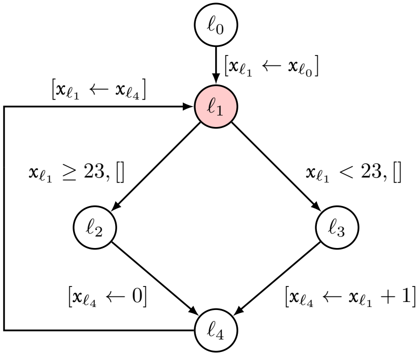
$$(\ell_0, [\, x_{\ell_0} \mapsto 2 \,]) \rightsquigarrow \left(\ell_1, \begin{bmatrix} x_{\ell_0} \mapsto 2 \\ x_{\ell_1} \mapsto 2 \end{bmatrix}\right) \rightsquigarrow \left(\ell_3, \begin{bmatrix} x_{\ell_0} \mapsto 2 \\ x_{\ell_1} \mapsto 2 \end{bmatrix}\right) \rightsquigarrow \left(\ell_4, \begin{bmatrix} x_{\ell_0} \mapsto 2 \\ x_{\ell_1} \mapsto 2 \\ x_{\ell_4} \mapsto 3 \end{bmatrix}\right) \rightsquigarrow \left(\ell_1, \begin{bmatrix} x_{\ell_0} \mapsto 2 \\ x_{\ell_1} \mapsto 3 \end{bmatrix}\right)$$

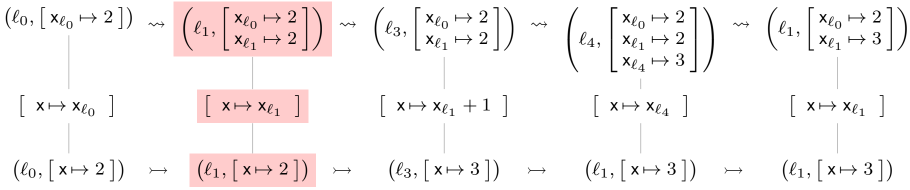
$$[\, x \mapsto x_{\ell_0} \,] \qquad [\, x \mapsto x_{\ell_1} \,] \qquad [\, x \mapsto x_{\ell_1} + 1 \,] \qquad [\, x \mapsto x_{\ell_4} \,] \qquad [\, x \mapsto x_{\ell_1} \,]$$

$$(\ell_0, [\, x \mapsto 2 \,]) \rightarrowtail (\ell_1, [\, x \mapsto 2 \,]) \rightarrowtail (\ell_3, [\, x \mapsto 3 \,]) \rightarrowtail (\ell_1, [\, x \mapsto 3 \,]) \rightarrowtail (\ell_1, [\, x \mapsto 3 \,])$$

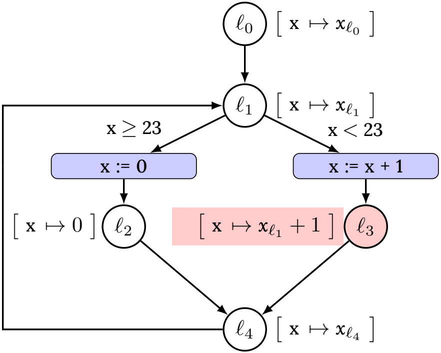
Bisimulation & homomorphism between transition systems

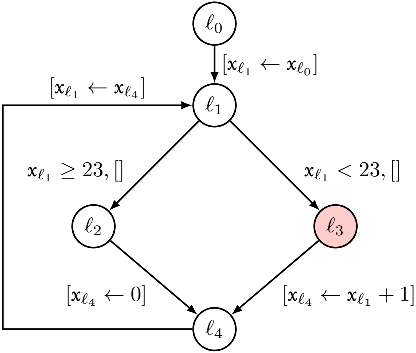






Bisimulation & homomorphism between transition systems



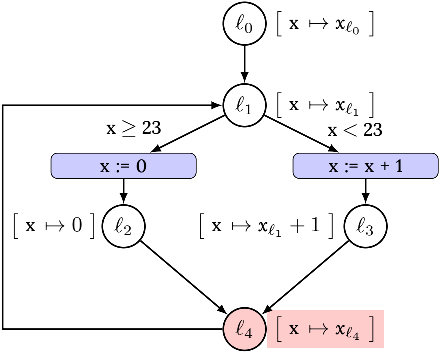


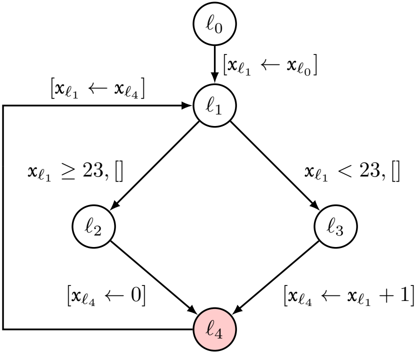
$$(\ell_0, [\, x_{\ell_0} \mapsto 2 \,]) \leadsto \left(\ell_1, \begin{bmatrix} x_{\ell_0} \mapsto 2 \\ x_{\ell_1} \mapsto 2 \end{bmatrix}\right) \leadsto \left(\ell_3, \begin{bmatrix} x_{\ell_0} \mapsto 2 \\ x_{\ell_1} \mapsto 2 \end{bmatrix}\right) \leadsto \left(\ell_4, \begin{bmatrix} x_{\ell_0} \mapsto 2 \\ x_{\ell_1} \mapsto 2 \\ x_{\ell_4} \mapsto 3 \end{bmatrix}\right) \leadsto \left(\ell_1, \begin{bmatrix} x_{\ell_0} \mapsto 2 \\ x_{\ell_1} \mapsto 3 \end{bmatrix}\right)$$

$$[\, x \mapsto x_{\ell_0} \,] \qquad [\, x \mapsto x_{\ell_1} \,] \qquad [\, x \mapsto x_{\ell_1} + 1 \,] \qquad [\, x \mapsto x_{\ell_4} \,] \qquad [\, x \mapsto x_{\ell_1} \,]$$

$$(\ell_0, [\, x \mapsto 2 \,]) \rightarrowtail (\ell_1, [\, x \mapsto 2 \,]) \rightarrowtail (\ell_3, [\, x \mapsto 3 \,]) \rightarrowtail (\ell_1, [\, x \mapsto 3 \,]) \rightarrowtail (\ell_1, [\, x \mapsto 3 \,])$$

Bisimulation & homomorphism between transition systems



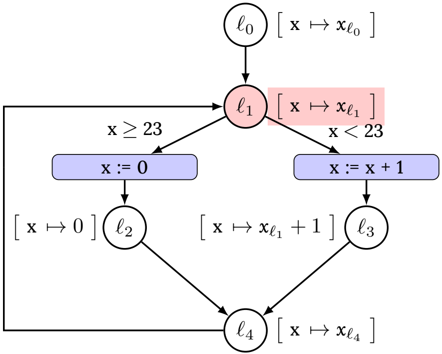


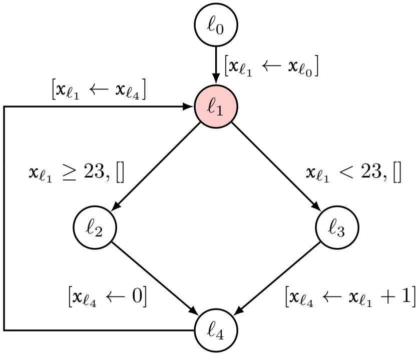
$$(\ell_0, [\, x_{\ell_0} \mapsto 2 \,]) \rightsquigarrow \left(\ell_1, \begin{bmatrix} x_{\ell_0} \mapsto 2 \\ x_{\ell_1} \mapsto 2 \end{bmatrix}\right) \rightsquigarrow \left(\ell_3, \begin{bmatrix} x_{\ell_0} \mapsto 2 \\ x_{\ell_1} \mapsto 2 \end{bmatrix}\right) \rightsquigarrow \left(\ell_4, \begin{bmatrix} x_{\ell_0} \mapsto 2 \\ x_{\ell_1} \mapsto 2 \\ x_{\ell_4} \mapsto 3 \end{bmatrix}\right) \rightsquigarrow \left(\ell_1, \begin{bmatrix} x_{\ell_0} \mapsto 2 \\ x_{\ell_1} \mapsto 3 \end{bmatrix}\right)$$

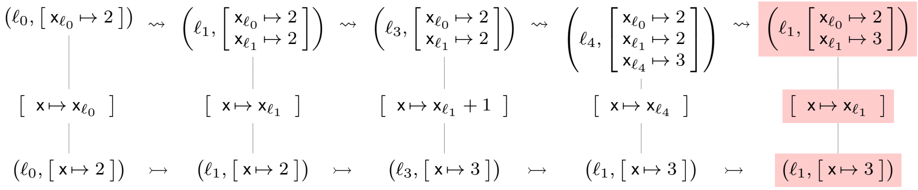
$$[\ x \mapsto x_{\ell_0}\ ] \qquad [\ x \mapsto x_{\ell_1}\ ] \qquad [\ x \mapsto x_{\ell_1} + 1\ ] \qquad [\ x \mapsto x_{\ell_4}\ ] \qquad [\ x \mapsto x_{\ell_1}\ ]$$

$$(\ell_0, [\, x \mapsto 2 \,]) \rightarrowtail (\ell_1, [\, x \mapsto 2 \,]) \rightarrowtail (\ell_3, [\, x \mapsto 3 \,]) \rightarrowtail (\ell_1, [\, x \mapsto 3 \,]) \rightarrowtail (\ell_1, [\, x \mapsto 3 \,])$$

Bisimulation & homomorphism between transition systems







Evaluation & Artifact

Research question

Can SSA translation based on Abstract Interpretation can be used in practice?

- Evaluation Settings
  - Use our algorithm to transform to SSA graph.
  - Translate our SSA graph to the LLVM format of SSA.
  - 970 lines of OCaml code (+ support functions + Frama-C parser)
  - Use Csmith to generate huge unstructured C functions.
- Results
  - Execution of the binary returns the same value than GCC&LLVM.
  - Few analysis iterations are needed to converge.
  - Analysis time on huge instances compatible with realistic usage
    - (max observed slowdown compared to GCC: $\approx \times 6$)
  - Fast Mergeable Integer Maps [Okasaki&Gill1998] important for fast $\sqcup$.
  - Combining SSA translation with dead code elimination/constant propagation improves analysis time.
- Open question: Can our approach outperform traditional approaches?

Conclusions

1. SSA translation can be described as a sound and complete abstract interpretation.
   - and performed using simple yet quite efficient dataflow analysis.
   - this allows combination of SSA translation with other abstract domains.
2. The essence of SSA = Global Value Graph + control flow information.
   - SSA graphs provide a simple syntax and semantics for SSA.
3. We can use abstract interpretation to produce cyclic term graphs.

Thank you!

Contact: Matthieu.Lemerre [at] cea.fr (We have open positions).