SSA Translation Is an Abstract Interpretation

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

POPL 2023
SSA Translation : Source Program $\rightarrow$ Program in SSA Form

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

Static analysis based on Abstract Interpretation : Source Program → 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  :  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 : Source Program → Program in SSA Form

Static analysis based on Abstract Interpretation : Source Program → 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 builds 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]

Program 1

- **Transformation:** SSA translation

Program 2

- **Analysis:** Constant propagation
  - **Transformation:** Constant replacement

Program 3

- **Analysis:** Dead code
  - **Transformation:** Dead code elimination

Program 4

- SSA translation analysis
  - Constant Propagation analysis
  - Dead Code analysis

- 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
Abstract stores =
Program Variables → Symbolic Expressions

```
x := x + 1

y := 2*x

z := x*z
```

Impliciterm graph:

```
x = x0
y = y0
z = z0

x := x + 1
y := 2*(x + 1)
z := (x + 1)*(x + 1)
```
Symbolic execution (∼ single-path SSA translation)

Abstract stores =
Program Variables → Symbolic Expressions

\[
\begin{align*}
\ell_0 &: x := x + 1 \\
\ell_1 &: y := 2x \\
\ell_2 &: z := x*z \\
\ell_3 &:
\end{align*}
\]

Initial store:
\[
\begin{bmatrix}
\_ & \_ & \_ \\
x & x_0 & \_ \\
y & y_0 & \_ \\
z & z_0 & \_.
\end{bmatrix}
\]

Updated store:
\[
\begin{bmatrix}
\_ & \_ & \_ \\
x & x_0 + 1 & \_ \\
y & y_0 & \_ \\
z & z_0 & \_.
\end{bmatrix}
\]
Symbolic execution (≈ single-path SSA translation)

Abstract stores =
Program Variables $\rightarrow$ Symbolic Expressions

\[\ell_0\]
\[x := x + 1\]
\[\ell_1\]
\[y := 2 \times x\]
\[\ell_2\]
\[z := x \times z\]
\[\ell_3\]

\[
\begin{bmatrix}
x & \leftrightarrow & x_0 \\
y & \leftrightarrow & y_0 \\
z & \leftrightarrow & z_0 \\
\end{bmatrix}
\]
Symbolic execution ($\approx$ single-path SSA translation)

Abstract stores =
Program Variables $\rightarrow$ Symbolic Expressions

\[
\begin{align*}
\ell_0 & : x := x + 1 \\
& \begin{bmatrix} x \mapsto x_0 + 1 \\
y \mapsto y_0 \\
z \mapsto z_0 \end{bmatrix} \\
\ell_1 & : y := 2 \times x \\
& \begin{bmatrix} x \mapsto x_0 + 1 \\
y \mapsto 2 \times (x_0 + 1) \\
z \mapsto z_0 \end{bmatrix} \\
\ell_2 & : z := x \times z \\
& \begin{bmatrix} x \mapsto x_0 + 1 \\
y \mapsto 2 \times (x_0 + 1) \\
z \mapsto (x_0 + 1) \times z_0 \end{bmatrix}
\end{align*}
\]
Symbolic execution ($\approx$ single-path SSA translation)

Abstract stores =
Program Variables $\to$ Symbolic Expressions

$$\ell_0$$
$$x := x + 1$$
$$\begin{bmatrix}
x & \mapsto & x_0 \\
y & \mapsto & y_0 \\
z & \mapsto & z_0
\end{bmatrix}$$

$$\ell_1$$
$$y := 2 \times x$$
$$\begin{bmatrix}
x & \mapsto & x_0 + 1 \\
y & \mapsto & y_0 \\
z & \mapsto & z_0
\end{bmatrix}$$

$$\ell_2$$
$$z := x \times z$$
$$\begin{bmatrix}
x & \mapsto & x_0 + 1 \\
y & \mapsto & 2 \times (x_0 + 1) \\
z & \mapsto & z_0
\end{bmatrix}$$

$$\ell_3$$
$$\begin{bmatrix}
x & \mapsto & x_0 + 1 \\
y & \mapsto & 2 \times (x_0 + 1) \\
z & \mapsto & (x_0 + 1) \times z_0
\end{bmatrix}$$

Implicit term graph:
the global value graph

1 $\xrightarrow{} x_0$

$\begin{bmatrix} x_0 + 1 \end{bmatrix}$

$2 \times (x_0 + 1) \xrightarrow{} (x_0 + 1) \times z_0$
\( \square \approx \phi + \text{name} \)

```
\begin{align*}
& \ell_0 \\
& \quad \begin{cases}
& x := 11 \\
& x := 22
\end{cases} \\
& \quad \ell_1 \\
& \quad \ell_2 \\
& \quad \ell_3 \\
& \quad y := x \\
& \quad \ell_4 \\
& \quad z := 11 \\
& \quad z := 22 \\
& \quad \ell_5
\end{align*}
```

\( [ \begin{array}{c}
& x \mapsto x_0 \\
& y \mapsto y_0 \\
& z \mapsto z_0
\end{array} ] \)
$$\Box \approx \phi + \text{name}$$

Diagram:

- $\ell_0$
  - $x := 11$
  - $x := 22$
- $\ell_1$
- $\ell_2$
- $\ell_3$
- $y := x$
- $\ell_4$
- $z := 11$
- $z := 22$
- $\ell_5$

Mappings:

- $x \mapsto x_0$
- $y \mapsto y_0$
- $z \mapsto z_0$
\[ \square \approx \phi + \text{name} \]

\[
\begin{align*}
\ell_0 & \quad \left[ \begin{array}{c}
\ell_1 \quad \begin{array}{l}
x := 11 \\
\end{array} \\
\ell_2 \quad \begin{array}{l}
x := 22 \\
\end{array} \\
\ell_3 \\
\ell_4 \quad \begin{array}{l}
z := 11 \\
\end{array} \\
\ell_5 \quad \begin{array}{l}
z := 22 \\
\end{array} \\
\end{array} \right] \\
\ell_1 & \quad \left[ \begin{array}{c}
x := 11 \\
y := y_0 \\
z := z_0 \\
\end{array} \right] \\
\ell_2 & \quad \left[ \begin{array}{c}
x := 22 \\
y := y_0 \\
z := z_0 \\
\end{array} \right] \\
\ell_3 & \\
\ell_4 & \\
\ell_5 & \\
\end{align*}
\]

\[ \begin{array}{c}
x \mapsto x_0 \\
y \mapsto y_0 \\
z \mapsto z_0 \\
\end{array} \]
\[ \emptyset \approx \phi + \text{name} \]

\[ \begin{bmatrix} x \mapsto 11 \\ y \mapsto y_0 \\ z \mapsto z_0 \end{bmatrix} \rightarrow \ell_0 \]

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

\[ x := 11 \]

\[ \ell_1 \]

\[ x := 22 \]

\[ \ell_2 \]

\[ x \mapsto 22 \]

\[ x \mapsto \phi \ (11, 22) \]

\[ y := x \]

\[ \ell_3 \]

\[ \begin{bmatrix} x \mapsto \phi \ (11, 22) \\ y \mapsto y_0 \\ z \mapsto z_0 \end{bmatrix} \]

\[ z := 11 \]

\[ \ell_4 \]

\[ z := 22 \]

\[ \ell_5 \]
\[ \square \approx \phi + \text{name} \]
$$\square \approx \phi + \text{name}$$

\[
\begin{array}{ccc}
\ell_0 & \xrightarrow{x := 11} & \ell_1 \\
& & \xrightarrow{y := y_0} \ell_3 \\
& & \xrightarrow{z := z_0} \ell_5 \\
\ell_2 & \xrightarrow{x := 22} & \ell_4 \\
& & \xrightarrow{y := \phi (11, 22)} \ell_5 \\
\end{array}
\]

- $\ell_0$: $x := 11$
- $\ell_1$: $y := y_0$
- $\ell_2$: $z := z_0$
- $\ell_3$: $x := \phi (11, 22)$
- $\ell_4$: $y := \phi (11, 22)$
- $\ell_5$: $z := \phi (11, 22)$
We want:
\[ \models x = y \]
\[ \not\models x = z \]
Either name the $\phi$
Either name the $\phi$

Or name the terms:

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

We want:

$$\models x = y
\quad \not\models x = z$$
Loops

$\ell_0$

$x := 0$

$\ell_1$

$\ell_2$

$x := x + 1$

$\ell_3$

$[x \mapsto x_0]$

{ }

$g$

Using fresh variables leads to non-termination.

The global value graph is an acyclic 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 an acyclic term graph.
Loops

Using fresh variables leads to non-termination.

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.
Loops

\[ \ell_0 \]  
\[ x := 0 \]  
\[ [x \mapsto x_0] \]  
\{ 
\}

\[ \ell_1 \]  
\[ [x \mapsto 0] \]  
\[ \ell_2 \]  
\[ [x \mapsto 0] \]  
\[ x := x + 1 \]  
\[ \ell_3 \]  

Using fresh variables leads to non-termination.

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 graph.
Loops

Loops

Using fresh variables leads to non-termination.

The global value graph is a cyclic term graph [Ariola & Klop 1996]

Symbolic (recursion) variables used for both non-determinism and termination

Wenames symbolic variables using the names of control locations

Control locations = recursion variables in the CFG viewed as a cyclic term graph
Loops

\[ \begin{align*}
\ell_0 & \quad [x \mapsto x_0] \\
\ell_1 & \quad [x \mapsto 0] \\
\ell_2 & \quad [x \mapsto x_1] \\
\ell_3 & \quad [x \mapsto 0 + 1] \\
\end{align*} \]

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

Using fresh variables lead to non-termination.

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.
Using fresh variables leads to non-termination. The global value graph is a cyclic term graph [Ariola & Klop 1996].

Symbolic (recursion) variables used for both nondeterminism and termination.

We name symbolic variables using the names of control locations.

\[
\begin{align*}
\ell_0 &\quad \text{[ } \ x \mapsto x_0 \text{ ]} \\
\ell_1 &\quad \text{[ } \ x \mapsto 0 \text{ ]} \\
\ell_2 &\quad \text{[ } \ x \mapsto x_1 \text{ ]} \\
x := x + 1 &\quad \text{[ } \ x \mapsto x_1 + 1 \text{ ]} \\
{x_1} &\quad \phi(0, 0 + 1),
\end{align*}
\]
Using fresh variables leads to non-termination.
Loops

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

Matthieu Lemerre

SSA Translation Is an Abstract Interpretation

POPL 2023
Loops

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 graph.

SSA Translation Is an Abstract Interpretation
Loops

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 graph.
Loops

The global value graph is a cyclic term graph \([Ariola\&Klop1996]\).

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.

Matthieu Lemerre

SSA Translation Is an Abstract Interpretation

POPL 2023
The global value graph is a cyclic term graph \([\text{Ariola & Klop} 1996]\). Symbolic (recursion) variables are used for both non-determinism and termination. We name symbolic variables using the names of control locations. Control locations are recursion variables in the CFG viewed as a cyclic term graph.
Loops

\[ x := 0 \]

\[ x := x + 1 \]

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 graph.

Matthieu Lemerre
SSA Translation Is an Abstract Interpretation
POPL 2023
Loops

\[ \ell_0 \]

\[ x := 0 \]

\[ \ell_1 \]

\[ x := x + 1 \]

\[ \ell_2 \]

\[ x := x + 1 \]

\[ \ell_3 \]

\[ x := x \]

\[ \ell_0 \]

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

\[ [x \mapsto 0] \]

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

\[ [x \mapsto x_{\ell_2} + 1] \]
Implicit cyclic term graph:

\[ \{ x_{\ell_2} = \psi(0, x_{\ell_2} + 1) \} \]
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
\[\gamma : \text{Abstract store} \rightarrow (\text{Valuation of symbolic variables} \rightarrow \text{Concrete store})\]
\[\gamma(\sigma^\#) = [\Gamma \in \text{Valuation} \mapsto \text{evaluate the symbolic expressions in } \sigma^\# \text{ using } \Gamma]\]

**Example**

\[
\gamma \left( \begin{bmatrix}
    x & \mapsto & 2 \times x_{\ell_0} \\
    y & \mapsto & 2 \times x_{\ell_0} + 1
\end{bmatrix} \right) = \left\{ \begin{bmatrix} x_{\ell_0} & \mapsto & 0 \\
    y & \mapsto & 1 \end{bmatrix}, \begin{bmatrix} x_{\ell_0} & \mapsto & 1 \\
    y & \mapsto & 3 \end{bmatrix}, \cdots \right\}
\]
Meaning, soundness and completeness

\[ \gamma : \text{Abstract store } \rightarrow (\text{Valuation of symbolic variables } \rightarrow \text{Concrete store}) \]
\[ \gamma(\sigma^\#) = \left[ \Gamma \in \text{Valuation} \mapsto \text{evaluate the symbolic expressions in } \sigma^\# \text{ using } \Gamma \right] \]

**Example**

\[ \gamma \left( \left[ \begin{array}{c} x \mapsto 2 \times x_0 \\ y \mapsto 2 \times x_0 + 1 \end{array} \right] \right) = \left\{ \left[ x_0 \mapsto 0 \right] \mapsto \left[ \begin{array}{c} x \mapsto 0 \\ y \mapsto 1 \end{array} \right], \left[ x_0 \mapsto 1 \right] \mapsto \left[ \begin{array}{c} x \mapsto 2 \\ y \mapsto 3 \end{array} \right], \ldots \right\} \]

Abstract stores can represent state properties and abstract transformations:

\[ \left[ \begin{array}{c} x \mapsto 2 \times x_0 \\ y \mapsto 2 \times x_0 + 1 \end{array} \right] \models x \text{ is even} \land y \text{ is odd} \land y - x = 1 \land x = 2 \times \text{old}(x) \]
Abstract stores can represent state properties and abstract transformations:

\[
\begin{bmatrix}
  x & \mapsto & 2 \times x_{\ell_0} \\
y & \mapsto & 2 \times x_{\ell_0} + 1
\end{bmatrix} \models x \text{ is even} \land y \text{ is odd} \land y - x = 1 \land x = 2 \times \text{old}(x)
\]
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 lose precision? Absence of control flow

Origin of symbolic variables is lost.

- Value graph recovers some information, but variables are independent.
Where do we loose precision? Absence of control flow

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.
Optimistically assume that unvisited edges are dead, revise later.
Translating guard expressions into symbolic expressions.

\[ x \geq 23 \]

\[ x := 0 \]

\[ x := x + 1 \]

\[ x < 23 \]
Translating guard expressions into symbolic expressions.
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.
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.
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.
Fixpoint reached, our SSA graph is sound and complete.
Operational semantics of the SSA graph

\[
\begin{align*}
(\ell_0, [x_{\ell_0} \mapsto 2]) & \mapsto (\ell_1, [x_{\ell_0} \mapsto 2, x_{\ell_1} \mapsto 2]) & \mapsto (\ell_3, [x_{\ell_0} \mapsto 2, x_{\ell_1} \mapsto 2]) & \mapsto (\ell_4, [x_{\ell_0} \mapsto 2, x_{\ell_1} \mapsto 2, x_{\ell_4} \mapsto 3]) & \mapsto (\ell_1, [x_{\ell_0} \mapsto 2, x_{\ell_1} \mapsto 3])
\end{align*}
\]
Operational semantics of the SSA graph

(\ell_0, [x_{\ell_0} \mapsto 2]) \leadsto (\ell_1, [x_{\ell_0} \mapsto 2, x_{\ell_1} \mapsto 2]) \leadsto (\ell_3, [x_{\ell_0} \mapsto 2, x_{\ell_1} \mapsto 2]) \leadsto (\ell_4, [x_{\ell_0} \mapsto 2, x_{\ell_1} \mapsto 2, x_{\ell_4} \mapsto 3]) \leadsto (\ell_1, [x_{\ell_0} \mapsto 2, x_{\ell_1} \mapsto 3])

\ell_0 \rightarrow [x_{\ell_1} \leftarrow x_{\ell_0}]

\ell_1 \rightarrow [x_{\ell_4} \leftarrow 0, x_{\ell_4} \leftarrow x_{\ell_1} + 1]

\ell_2 \rightarrow [x_{\ell_1} \geq 23, []]

\ell_3 \rightarrow [x_{\ell_1} < 23, []]

\ell_4 \rightarrow [x_{\ell_1} \leftarrow x_{\ell_4}]

Matthieu Lemerre
SSA Translation Is an Abstract Interpretation
POPL 2023 15/17
Operational semantics of the SSA graph

\[(\ell_0, [x_{\ell_0} \mapsto 2]) \leadsto (\ell_1, [x_{\ell_1} \mapsto 2]) \leadsto (\ell_3, [x_{\ell_0} \mapsto 2]) \leadsto (\ell_4, [x_{\ell_0} \mapsto 2, x_{\ell_1} \mapsto 2, x_{\ell_4} \mapsto 3]) \leadsto (\ell_1, [x_{\ell_0} \mapsto 2, x_{\ell_1} \mapsto 3])\]
Operational semantics of the SSA graph

\[(\ell_0, [x_{\ell_0} \mapsto 2]) \leadsto (\ell_1, [x_{\ell_0} \mapsto 2, x_{\ell_1} \mapsto 2]) \leadsto (\ell_3, [x_{\ell_0} \mapsto 2, x_{\ell_1} \mapsto 2]) \leadsto (\ell_4, [x_{\ell_0} \mapsto 2, x_{\ell_1} \mapsto 2, x_{\ell_4} \mapsto 3]) \leadsto (\ell_1, [x_{\ell_0} \mapsto 2, x_{\ell_1} \mapsto 3])\]
Operational semantics of the SSA graph

\( (\ell_0, [x_{\ell_0} \mapsto 2]) \leadsto (\ell_1, [x_{\ell_0} \mapsto 2, x_{\ell_1} \mapsto 2]) \leadsto (\ell_3, [x_{\ell_0} \mapsto 2]) \leadsto (\ell_4, [x_{\ell_0} \mapsto 2, x_{\ell_1} \mapsto 2, x_{\ell_4} \mapsto 3]) \leadsto (\ell_1, [x_{\ell_0} \mapsto 2, x_{\ell_1} \mapsto 3]) \)
Bisimulation & homomorphism between transition systems

\begin{align*}
& \ell_0 \xrightarrow{[x \mapsto x \ell_0]} \\
& \ell_1 \xrightarrow{[x \mapsto x \ell_1]} \xrightarrow{x \geq 23} \\
& \ell_2 \xrightarrow{[x \mapsto x \ell_1 + 1]} \xrightarrow{x := 0} \\
& \ell_3 \xrightarrow{[x \mapsto x \ell_1]} \xrightarrow{x := x + 1} \\
& \ell_4 \xrightarrow{[x \mapsto x \ell_4]} \\
& \ell_1 \xrightarrow{[x \mapsto x \ell_1 + 1]} \xrightarrow{x \ell_1 \geq 23} \\
& \ell_2 \xrightarrow{[x \mapsto x \ell_4]} \xrightarrow{x \ell_1 < 23} \\
& \ell_3 \xrightarrow{[x \mapsto x \ell_4 \ell_1]} \\
& \ell_4 \xrightarrow{[x \mapsto x \ell_4 \ell_1]} \\
\end{align*}
Bisimulation & homomorphism between transition systems

\[ x \geq 23 \]

\[ x \leftarrow 0 \]

\[ x \leftarrow x + 1 \]

\[ (l_0, [x \leftarrow 2]) \xrightarrow{x \leftarrow x_0} (l_1, [x \leftarrow 2]) \]

\[ (l_0, [x \leftarrow x_0]) \xrightarrow{x \leftarrow x_1} (l_1, [x \leftarrow x_1]) \]

\[ x \leftarrow x + 1 \]

\[ x \leftarrow x + 1 \]

\[ x \leftarrow x_1 \]

\[ [x_1 \leftarrow x_4] \]

\[ x_{\ell_1} \geq 23, [] \]

\[ x_{\ell_1} < 23, [] \]

\[ [x_4 \leftarrow 0] \]

\[ [x_4 \leftarrow x_{\ell_1} + 1] \]

\[ (l_1, [x \leftarrow x_1]) \]
Bisimulation & homomorphism between transition systems

\[ x \geq 23 \]
\[ x := 0 \]
\[ x := x + 1 \]
\[ x \leq 23 \]

\[ (\ell_0, [x \mapsto x_0]) \leadsto (\ell_1, [x \mapsto x_1]) \leadsto (\ell_3, [x \mapsto x_1, x \mapsto x_1 + 1]) \leadsto (\ell_4, [x \mapsto x_4]) \]

\[ \ell_0 \]
\[ \ell_1 \]
\[ \ell_2 \]
\[ \ell_3 \]
\[ \ell_4 \]

\[ x \geq 23 \]
\[ x := 0 \]
\[ x := x + 1 \]
\[ x \leq 23 \]

\[ (\ell_0, [x \mapsto x_0]) \leadsto (\ell_1, [x \mapsto x_1]) \leadsto (\ell_3, [x \mapsto x_1]) \leadsto (\ell_4, [x \mapsto x_4]) \]

\[ \ell_0 \]
\[ \ell_1 \]
\[ \ell_2 \]
\[ \ell_3 \]
\[ \ell_4 \]

\[ x \leq 23 \]
\[ x := x + 1 \]
\[ x \leq 23 \]

\[ (\ell_0, [x \mapsto x_0]) \leadsto (\ell_1, [x \mapsto x_1]) \leadsto (\ell_3, [x \mapsto x_3]) \leadsto (\ell_4, [x \mapsto x_4]) \]

\[ \ell_0 \]
\[ \ell_1 \]
\[ \ell_2 \]
\[ \ell_3 \]
\[ \ell_4 \]
Bisimulation & homomorphism between transition systems

\[
\begin{align*}
\ell_0 & \xRightarrow{\text{\[ x \mapsto x_{\ell_0} \]}} \ell_1 \\
& \xRightarrow{\text{\[ x \mapsto x_{\ell_1} \]}} \ell_2 \\
& \xRightarrow{\text{\[ x \mapsto x_{\ell_1} + 1 \]}} \ell_3 \\
& \xRightarrow{\text{\[ x \mapsto x_{\ell_4} \]}} \ell_4
\end{align*}
\]

\[
\begin{align*}
\ell_0 & \xRightarrow{\text{\[ x_{\ell_1} \mapsto x_{\ell_0} \]}} \ell_1 \\
& \xRightarrow{\text{\[ x_{\ell_1} \mapsto x_{\ell_4} \]}} \ell_2 \\
& \xRightarrow{\text{\[ x_{\ell_1} \mapsto x_{\ell_1} + 1 \]}} \ell_3 \\
& \xRightarrow{\text{\[ x_{\ell_1} \mapsto x_{\ell_4} \]}} \ell_4
\end{align*}
\]

\[
\begin{align*}
(\ell_0, \text{\[ x_{\ell_0} \mapsto 2 \]}) & \xRightarrow{\text{\[ x \mapsto x_{\ell_0} \]}} (\ell_1, \text{\[ x_{\ell_0} \mapsto 2 \]}) \\
& \xRightarrow{\text{\[ x \mapsto x_{\ell_1} \]}} (\ell_3, \text{\[ x_{\ell_0} \mapsto 2 \]}) \\
& \xRightarrow{\text{\[ x \mapsto x_{\ell_1} \]}} (\ell_4, \text{\[ x_{\ell_0} \mapsto 2 \]}) \\
& \xRightarrow{\text{\[ x \mapsto x_{\ell_1} \]}} (\ell_1, \text{\[ x_{\ell_0} \mapsto 2 \]})
\end{align*}
\]

\[
\begin{align*}
(\ell_0, \text{\[ x \mapsto 2 \]}) & \xRightarrow{\text{\[ x \mapsto x_{\ell_1} \]}} (\ell_1, \text{\[ x \mapsto 2 \]}) \\
& \xRightarrow{\text{\[ x \mapsto x_{\ell_1} \]}} (\ell_3, \text{\[ x \mapsto 3 \]}) \\
& \xRightarrow{\text{\[ x \mapsto x_{\ell_1} \]}} (\ell_1, \text{\[ x \mapsto 3 \]})
\end{align*}
\]
Bisimulation \& homomorphism between transition systems

\[
\begin{align*}
\ell_0 & \xrightarrow{[x \mapsto x_{\ell_0}]} \\
\ell_1 & \xrightarrow{[x \mapsto x_{\ell_1}]} \\
\ell_4 & \xrightarrow{[x \mapsto x_{\ell_4}]} \\
\ell_3 & \xrightarrow{[x \mapsto x_{\ell_3} + 1]} \\
\ell_2 & \xrightarrow{[x \mapsto x_{\ell_2} + 1]} \\
\ell_0 & \xrightarrow{[x \mapsto x_{\ell_0} + 1]} \\
\ell_1 & \xrightarrow{[x \mapsto x_{\ell_1} + 1]} \\
\ell_3 & \xrightarrow{[x \mapsto x_{\ell_3} + 1]} \\
\ell_4 & \xrightarrow{[x \mapsto x_{\ell_4} + 1]} \\
\end{align*}
\]

\[
\begin{align*}
(l_0, [x_{\ell_0} \mapsto 2]) & \rightsquigarrow (l_1, [x_{\ell_0} \mapsto 2, x_{\ell_1} \mapsto 2]) \rightsquigarrow (l_3, [x_{\ell_0} \mapsto 2, x_{\ell_1} \mapsto 2]) \rightsquigarrow (l_4, [x_{\ell_0} \mapsto 2, x_{\ell_1} \mapsto 2, x_{\ell_4} \mapsto 3]) \rightsquigarrow (l_1, [x_{\ell_0} \mapsto 2]) \\
([x \mapsto x_{\ell_0}]) & \rightsquigarrow ([x \mapsto x_{\ell_1}]) \rightsquigarrow ([x \mapsto x_{\ell_1} + 1]) \rightsquigarrow ([x \mapsto x_{\ell_4}]) \rightsquigarrow ([x \mapsto x_{\ell_1}]) \\
(l_0, [x \mapsto 2]) & \rightsquigarrow (l_1, [x \mapsto 2]) \rightsquigarrow (l_3, [x \mapsto 3]) \rightsquigarrow (l_1, [x \mapsto 3]) \rightsquigarrow (l_1, [x \mapsto 3])
\end{align*}
\]
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 \( \sqsubset \).
- Combining SSA translation with dead code elimination/constant propagation improves analysis time.

Open question: Can our approach outperform traditional approaches?
Conclusions

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.

The essence of SSA = Global Value Graph + control flow information.
- SSA graphs provide a simple syntax and semantics for SSA.

We can use abstract interpretation to produce cyclic term graphs.

Thank you!

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