Software Verification

Grégoire Sutre

LaBRI, University of Bordeaux, CNRS, France

Summer School on Verification Technology, Systems & Applications
September 2008

Part 1
Part I

Introduction
Software Verification: Why?

Software Verification: How?
Outline — Introduction

1. Software Verification: Why?

2. Software Verification: How?
Ubiquity of Software in Modern Life

Once upon a time, lecturers used hand-written transparencies with an overhead projector.

- pens
- transparencies
- scissors
- sticky tape
- lamp
- lenses
- mirror
- screen

Nowadays softwares are used to design the slides and to project them.

Similar evolution in many, many areas
Once upon a time, lecturers used hand-written transparencies with an overhead projector.

- pens
- transparencies
- scissors
- sticky tape
- lamp
- lenses
- mirror
- screen

Nowadays softwares are used to design the slides and to project them.

Similar evolution in many, many areas
Why?

Some advantages of software over dedicated hardware components

- **Reduce time to market**
  - Less time to write the slides (really?)
  - Ability to re-organize the presentation

- **Reduce costs**
  - No pen, no transparencies
  - Re-usability of slides, ability to make minor modifications for free

- **Increase functionality**
  - Automatic generation of some slides (table of contents)
  - Nicer overlays (sticky tape is not required anymore!)
  - Ability to display videos

But software is not without risk...
Bugs are Frequent in Software
Bugs are Frequent in Software

Grégoire Sutre

Software Verification
Introduction
Bugs are Frequent in Software
Bugs are Frequent in Software
A problem has been detected and windows has been shut down to prevent damage to your computer.

The problem seems to be caused by the following file: SPCMDCON.SYS

PAGE_FAULT_IN_NONPAGED_AREA

If this is the first time you've seen this stop error screen, restart your computer. If this screen appears again, follow these steps:

Check to make sure any new hardware or software is properly installed. If this is a new installation, ask your hardware or software manufacturer for any Windows updates you might need.

If problems continue, disable or remove any newly installed hardware or software. Disable BIOS memory options such as caching or shadowing. If you need to use Safe Mode to remove or disable components, restart your computer, press F8 to select Advanced Startup Options, and then select Safe Mode.

Technical information:

*** STOP: 0x00000050 (0xFD3094C2,0x00000001,0xFBFE7617,0x00000000)

*** SPCMDCON.SYS - Address FBFE7617 base at FBFE5000, DateStamp 3d6dd67c
A Critical Software Bug: Ariane 5.01

« On 4 June 1996, the maiden flight of the Ariane 5 launcher ended in a failure. Only about 40 seconds after initiation of the flight sequence, at an altitude of about 3700 m, the launcher veered off its flight path, broke up and exploded. »

« The failure of the Ariane 5.01 was caused by the complete loss of guidance and attitude information 37 seconds after start of the main engine ignition sequence (30 seconds after lift-off). This loss of information was due to specification and design errors in the software of the inertial reference system. »
On 4 June 1996, the maiden flight of the Ariane 5 launcher ended in a failure. Only about 40 seconds after initiation of the flight sequence, at an altitude of about 3700 m, the launcher veered off its flight path, broke up and exploded.

The failure of the Ariane 5.01 was caused by the complete loss of guidance and attitude information 37 seconds after start of the main engine ignition sequence (30 seconds after lift-off). This loss of information was due to specification and design errors in the software of the inertial reference system.
Embedded systems in: cell phones, satellites, airplanes, cars, wireless routers, MP3 players, refrigerators, . . .

Examples of Critical Systems

- attitude and orbit control systems in satellites
- X-by-wire control systems in airplanes and in cars (soon)

Increasing importance of software in embedded systems

- custom hardware replaced by processor + custom software
- software is a dominant factor in design time and cost (70 %)

Critical embedded systems require “exhaustive” validation
As computational power grows . . .

Moore’s law: « the number of transistors on a chip doubles every two years »

. . . software complexity grows . . .

Wirth’s Law: « software gets slower faster than hardware gets faster »

. . . and so does the number of bugs!

Watts S. Humphrey: « 5 – 10 bugs per 1000 lines of code after product test »

Growing need for automatic validation techniques
As computational power grows . . .

**Moore’s law:** « the number of transistors on a chip doubles every two years »

. . . software complexity grows . . .

**Wirth’s Law:** « software gets slower faster than hardware gets faster »

. . . and so does the number of bugs!

**Watts S. Humphrey:** « 5 – 10 bugs per 1000 lines of code after product test »

Growing need for automatic validation techniques
As computational power grows . . .

Moore’s law: « the number of transistors on a chip doubles every two years »

. . . software complexity grows . . .

Wirth’s Law: « software gets slower faster than hardware gets faster »

. . . and so does the number of bugs!

Watts S. Humphrey: « 5 – 10 bugs per 1000 lines of code after product test »
As computational power grows . . .

Moore’s law: « the number of transistors on a chip doubles every two years »

. . . software complexity grows . . .

Wirth’s Law: « software gets slower faster than hardware gets faster »

. . . and so does the number of bugs!

Watts S. Humphrey: « 5 – 10 bugs per 1000 lines of code after product test »

Growing need for automatic validation techniques
Outline

Introduction

1. Software Verification: Why?

2. Software Verification: How?
Software Testing

Running the executable (obtained by compilation)
- on multiple inputs
- usually on the target platform

Testing is a widespread validation approach in the software industry

- can be (partially) automated
- can detect a lot of bugs

But

Costly and time-consuming
Not exhaustive
Software Testing

Running the executable (obtained by compilation)
- on multiple inputs
- usually on the target platform

Testing is a widespread validation approach in the software industry

- can be (partially) automated
- can detect a lot of bugs

But

Costly and time-consuming

Not exhaustive
x = 1;
if (y <= 10) {
    y = 10;
} else {
    while (x < y) {
        x = 2 * x;
        y = y - 1;
    }
} x = y + 1;
Rice’s Theorem

Any non-trivial semantic property of programs is undecidable.

Classical Example: Termination

There exists no algorithm which can solve the halting problem:
- given a description of a program as input,
- decide whether the program terminates or loops forever.
Implicit in Rice’s Theorem is an idealized program model, where programs have access to *unbounded memory*. In reality programs are run on a computer with *bounded memory*. Model-checking becomes decidable for finite-state systems. But even with bounded memory, *complexity* in practice is *too high* for finite-state model-checking:

- 1 megabyte (1,000,000 bytes) of memory $\approx 10^{2,400,000}$ states
- 1000 variables $\times 64$ bits $\approx 10^{19,200}$ states
- optimistic limit for finite-state model checkers: $10^{100}$ states
More Realistic Objectives for Software Verification

Incomplete Methods

Approximate Algorithms

- Always terminate
- Indefinite answer (yes / no / ?)

Exact Semi-Algorithms

- Definite answer (yes / no)
- May not terminate

Topics of the lecture

Static Analysis

Abstraction Refinement
More Realistic Objectives for Software Verification

Incomplete Methods

Approximate Algorithms
- Always terminate
- Indefinite answer (yes / no / ?)

Exact Semi-Algorithms
- Definite answer (yes / no)
- May not terminate

Topics of the lecture

Static Analysis

Abstraction Refinement
Static Analysis

Tentative Definition

Compile-time techniques to gather run-time information about programs without actually running them

Example

Detection of variables that are used before initialization

- Always terminates
- Applies to large programs
- Simple analyses (original goal was compilation)
- Indefinite answer (yes / no / ?)

In the Lecture

Data Flow Analysis  Abstract Interpretation
Static Analysis

Tentative Definition
Compile-time techniques to gather run-time information about programs without actually running them

Example
Detection of variables that are used before initialization

- Always terminates
- Applies to large programs
- Simple analyses (original goal was compilation)
- Indefinite answer (yes / no / ?)

In the Lecture
Data Flow Analysis  Abstract Interpretation
Abstraction Refinement

Tentative Definition

Analysis-time techniques to verify programs by model-checking and refinement of finite-state approximate models

Example

Verification of safety and fairness of a mutual exclusion algorithm

😊 Complex analyses (properties expressed in temporal logics)
😊 Definite answer (yes / no)
😊 May not terminate
😊 Modeling of the program into a finite-state transition system

In the Lecture

Abstract Model Refinement for Safety Properties
Abstraction Refinement

**Tentative Definition**

Analysis-time techniques to verify programs by model-checking and refinement of finite-state approximate models

**Example**

Verification of safety and fairness of a mutual exclusion algorithm

- Complex analyses (properties expressed in temporal logics)
- Definite answer (yes / no)
- May not terminate
- Modeling of the program into a finite-state transition system

**In the Lecture**

Abstract Model Refinement for Safety Properties
Common Ingredient: Property-Preserving Abstraction

Abstraction Process
Interpret programs according to a simplified, “abstract” semantics.

Property-Preserving Abstraction
Formally relate the “abstract” semantics with the “standard” semantics, so as to preserve relevant properties.

Preservation of Properties
Program interpretation with this abstract semantics therefore gives “correct” information about properties of real runs.
Objective of Sign Analysis

Discover for each program point the sign of possible run-time values that numerical variables can have at that point.

The abstract semantics “tracks” the following information, for each variable $x$:

- $x < 0$
- $x \leq 0$
- $x = 0$
- $x \geq 0$
- $x > 0$
Abstract Interpretation Example: Sign Analysis

1. \( x = 1; \)

2. \( \textbf{if} \ (y \leq 10) \ { \}

3. \( \quad y = 10; \)

4. \} \)

5. \( \textbf{else} \ { \}

6. \( \quad \textbf{while} \ (x < y) \ { \}

7. \( \quad \quad x = 2 \times x; \)

8. \( \quad \quad y = y - 1; \)

9. \} \)

10. \} \)

11. \( x = y + 1; \)

12. \( \textbf{assert} (x > 0); \)
1 \ x = 1; \quad x > 0
2 \ if \ (y \leq 10) \ { \\
3 \quad y = 10; \\
4 \ }
5 \ else \ { \\
6 \quad while \ (x < y) \ { \\
7 \quad \quad x = 2 \times x; \\
8 \quad \quad y = y - 1; \\
9 \quad } \\
10 \ } \\
11 \ x = y + 1; \\
12 \ assert(x > 0);
Abstract Interpretation Example: Sign Analysis

1  \( x = 1; \quad x > 0 \)
2  \textbf{if} (y \leq 10) \{ \quad x > 0 \\
3    y = 10; \}
4  \}
5  \textbf{else} \{ \\
6    \textbf{while} (x < y) \{ \\
7      x = 2 \times x; \\
8      y = y - 1; \\
9    \}
10  \}
11  x = y + 1; \\
12  \textbf{assert}(x > 0);
Abstract Interpretation Example: Sign Analysis

1. \( x = 1; \)
2. \( \textbf{if} \ (y \leq 10) \ { \)
   \( x > 0 \)
   \( y = 10; \)
   \( x > 0 \land y > 0 \)
3. \( } \)
4. \( \textbf{else} \ { \)
5. \( \textbf{while} \ (x < y) \ { \)
6. \( x = 2 \times x; \)
7. \( y = y - 1; \)
8. \( } \)
9. \( } \)
10. \( x = y + 1; \)
11. \( \textbf{assert}(x > 0); \)
Abstract Interpretation Example: Sign Analysis

1. \( x = 1; \) 
2. \( \textbf{if} \ (y \leq 10) \{ \) 
3. \( y = 10; \) 
4. \( \} \) 
5. \( \textbf{else} \ { \) 
6. \( \textbf{while} \ (x < y) \{ \) 
7. \( x = 2 \times x; \) 
8. \( y = y - 1; \) 
9. \( \} \) 
10. \( \} \) 
11. \( x = y + 1; \) 
12. \( \textbf{assert} (x > 0); \)
Abstract Interpretation Example: Sign Analysis

1. \( x = 1; \)  
2. \( \textbf{if} \ (y \leq 10) \ {\ } \{ \)  
3. \( y = 10; \)  
4. \( \} \)
5. \( \textbf{else} \ {\ } \{ \)  
6. \( \textbf{while} \ (x < y) \ {\ } \{ \)  
7. \( x = 2 \times x; \)
8. \( y = y - 1; \)
9. \( \} \)
10. \} \)
11. \( x = y + 1; \)
12. \( \textbf{assert}(x > 0); \)
Abstract Interpretation Example: Sign Analysis

1 \( x = 1; \)  
2 \( \textbf{if} \ (y \leq 10) \{ \)  
3 \( \quad y = 10; \)  
4 \( \} \)  
5 \( \textbf{else} \{ \)  
6 \( \quad \textbf{while} \ (x < y) \{ \)  
7 \( \quad \quad x = 2 \times x; \)  
8 \( \quad \quad y = y - 1; \)  
9 \( \quad \} \)  
10 \} \)  
11 \( x = y + 1; \)  
12 \( \textbf{assert} (x > 0); \)
Abstract Interpretation Example: Sign Analysis

```plaintext
1 x = 1;
2 if (y ≤ 10) {
3     y = 10;
4 }
5 else {
6     while (x < y) {
7         x = 2 * x;
8         y = y - 1;
9     }
10 }
11 x = y + 1;
12 assert (x > 0);
```
Abstract Interpretation Example: Sign Analysis

1  \( x = 1; \)
2  \( \textbf{if} \ (y \leq 10) \ {\}
3     \( y = 10; \)
4  \} \]
5  \( \textbf{else} \ {\}
6     \( \textbf{while} \ (x < y) \ {\}
7         \( x = 2 \times x; \)
8         \( y = y - 1; \)
9     \} \]
10 \} \]
11 \( x = y + 1; \)
12 \textbf{assert} (x > 0);
Abstract Interpretation Example: Sign Analysis

```plaintext
1  x = 1;
2  if (y ≤ 10) {
3      y = 10;
4  }
5  else {
6      while (x < y) {
7          x = 2 * x;
8          y = y - 1;
9      }
10  }
11  x = y + 1;
12  assert (x > 0);
```

Abstract Interpretation Example: Sign Analysis

1  \( x = 1; \)
2  \( \text{if} (y \leq 10) \) {
3    \( y = 10; \)
4  }
5  \else \{
6    \textbf{while} (x < y) {
7      \( x = 2 \ast x; \)
8      \( y = y - 1; \)
9    }
10  }
11  \( x = y + 1; \)
12  \textbf{assert} (x > 0);
Abstract Interpretation Example: Sign Analysis

1. \( x = 1; \)
2. \( \textbf{if} \ (y \leq 10) \{ \)
3. \( \quad y = 10; \)
4. \( \} \)
5. \( \textbf{else} \{ \)
6. \( \quad \textbf{while} \ (x < y) \{ \)
7. \( \quad \quad x = 2 \times x; \)
8. \( \quad \quad y = y - 1; \)
9. \( \quad \} \)
10. \( \} \)
11. \( x = y + 1; \)
12. \( \textbf{assert}(x > 0); \)
Abstract Interpretation Example: Sign Analysis

1 \( x = 1; \)

2 \( \textbf{if} \ (y \leq 10) \{ \)

3 \( \quad y = 10; \)

4 \( \}\)

5 \( \textbf{else} \{ \)

6 \( \quad \textbf{while} \ (x < y) \{ \)

7 \( \quad x = 2 \times x; \)

8 \( \quad y = y - 1; \)

9 \( \}\)

10 \( \}\)

11 \( x = y + 1; \)

12 \( \textbf{assert} \ (x > 0); \)

\( x \geq 0 \land y \geq 0 \land x < y \)
Credits: Pioneers (1970’s)

Iterative Data Flow Analysis
Gary Kildall
John Kam & Jeffrey Ullman
Michael Karr
...

Abstract Interpretation
Patrick Cousot & Radhia Cousot
Nicolas Halbwachs
...

And many, many more...
Outline of the Lecture

- Control Flow Automata
- Data Flow Analysis
- Abstract Interpretation
- Abstract Model Refinement
Outline of the Lecture

- Control Flow Automata
- Data Flow Analysis
- Abstract Interpretation
- Abstract Model Refinement

Static Analysis
Outline of the Lecture

- Control Flow Automata
- Data Flow Analysis
- Abstract Interpretation
- Abstract Model Refinement

Static Analysis
Abstraction Refinement
Part II

Control Flow Automata
Outline — Control Flow Automata

3 Syntax and Semantics

4 Verification of Control Flow Automata
Outline — Control Flow Automata

3 Syntax and Semantics

4 Verification of Control Flow Automata
Short Introduction to Control Flow Automata

Requirement for verification: formal semantics of programs

Formal Semantics

Formalization as a mathematical model of the meaning of programs

- Denotational
- Operational
- Axiomatic

Operational Semantics

Labeled transition system describing the possible computational steps

First Step Towards an Operational Semantics

Program text $\rightarrow$ Graph-based representation
Short Introduction to Control Flow Automata

Requirement for verification: formal **semantics** of programs

**Formal Semantics**
Formalization as a mathematical model of the meaning of programs

Denotational  Operational  Axiomatic

**Operational Semantics**
Labeled transition system describing the possible computational steps

**First Step Towards an Operational Semantics**
Program text $\rightarrow$ Graph-based representation
Short Introduction to Control Flow Automata

Requirement for verification: formal **semantics** of programs

**Formal Semantics**
Formalization as a mathematical model of the meaning of programs

- Denotational
- Operational
- Axiomatic

**Operational Semantics**
Labeled transition system describing the possible computational steps

**First Step Towards an Operational Semantics**
Program text $\rightarrow$ Graph-based representation
Short Introduction to Control Flow Automata

Requirement for verification: formal semantics of programs

Formal Semantics
Formalization as a mathematical model of the meaning of programs

Denotational  Operational  Axiomatic

Operational Semantics
Labeled transition system describing the possible computational steps

First Step Towards an Operational Semantics
Program text $\rightarrow$ Graph-based representation
Control flow automaton
x = 1;
if (y ≤ 10) {
y = 10;
} else {
    while (x < y) {
        x = 2 * x;
y = y - 1;
    }
} x = y + 1;
x = 1;
if (y ≤ 10) {
  y = 10;
} else {
  while (x < y) {
    x = 2 * x;
    y = y - 1;
  }
} x = y + 1;
Labeled Directed Graphs

Definition

A labeled directed graph is a triple \( G = \langle V, \Sigma, \rightarrow \rangle \) where:

- \( V \) is a finite set of vertices,
- \( \Sigma \) is a finite set of labels,
- \( \rightarrow \subseteq V \times \Sigma \times V \) is a finite set of edges.

Notation for edges: \( v \xrightarrow{\sigma} v' \) instead of \( (v, \sigma, v') \in \rightarrow \)

A path in \( G \) is a finite sequence \( v_0 \xrightarrow{\sigma_0} v'_0, \ldots, v_k \xrightarrow{\sigma_k} v'_k \) of edges such that \( v'_i = v_{i+1} \) for each \( 0 \leq i < k \).

Notation for paths: \( v_0 \xrightarrow{\sigma_0} v_1 \cdots v_k \xrightarrow{\sigma_k} v'_k \)
A labeled directed graph is a triple \( G = \langle V, \Sigma, \rightarrow \rangle \) where:

- \( V \) is a finite set of vertices,
- \( \Sigma \) is a finite set of labels,
- \( \rightarrow \subseteq V \times \Sigma \times V \) is a finite set of edges.

Notation for edges: \( v \xrightarrow{\sigma} v' \) instead of \( (v, \sigma, v') \in \rightarrow \)

A path in \( G \) is a finite sequence \( v_0 \xrightarrow{\sigma_0} v'_0, \ldots, v_k \xrightarrow{\sigma_k} v'_k \) of edges such that \( v'_i = v_{i+1} \) for each \( 0 \leq i < k \).

Notation for paths: \( v_0 \xrightarrow{\sigma_0} v_1 \cdots v_k \xrightarrow{\sigma_k} v'_k \)
A labeled directed graph is a triple $G = \langle V, \Sigma, \rightarrow \rangle$ where:

- $V$ is a finite set of vertices,
- $\Sigma$ is a finite set of labels,
- $\rightarrow \subseteq V \times \Sigma \times V$ is a finite set of edges.

Notation for edges: $v \xrightarrow{\sigma} v'$ instead of $(v, \sigma, v') \in \rightarrow$

A path in $G$ is a finite sequence $v_0 \xrightarrow{\sigma_0} v'_0, \ldots, v_k \xrightarrow{\sigma_k} v'_k$ of edges such that $v'_i = v_{i+1}$ for each $0 \leq i < k$.

Notation for paths: $v_0 \xrightarrow{\sigma_0} v_1 \cdots v_k \xrightarrow{\sigma_k} v'_k$
A control flow automaton is a quintuple \( \langle Q, q_{in}, q_{out}, X, \rightarrow \rangle \) where:

- \( Q \) is a finite set of \textit{locations},
- \( q_{in} \in Q \) is an \textit{initial location} and \( q_{out} \in Q \) is an \textit{exit location},
- \( X \) is a finite set of \textit{variables},
- \( \rightarrow \subseteq Q \times \mathcal{O}_p \times Q \) is a finite set of \textit{transitions}.

\( \mathcal{O}_p \) is the set of operations defined by:

\[
\begin{align*}
cst & ::= c \in Q \\
var & ::= x \in X \\
expr & ::= \text{cst} \mid \text{var} \mid expr \, \circ \, expr, \text{ with } \circ \in \{+, -, \ast\} \\
guard & ::= expr \, \triangleleft \, expr, \text{ with } \triangleleft \in \{<, \leq, =, \neq, \geq, >\} \\
\mathcal{O}_p & ::= guard \mid \text{var} \, := \, expr
\end{align*}
\]
A control flow automaton is a quintuple \( \langle Q, q_{in}, q_{out}, X, \rightarrow \rangle \) where:

- \( Q \) is a finite set of locations,
- \( q_{in} \in Q \) is an initial location and \( q_{out} \in Q \) is an exit location,
- \( X \) is a finite set of variables,
- \( \rightarrow \subseteq Q \times Op \times Q \) is a finite set of transitions.

\( Op \) is the set of operations defined by:

\[
\begin{align*}
cst & ::= c \in Q \\
var & ::= x \in X \\
expr & ::= cst \mid var \mid expr \bullet expr, \text{ with } \bullet \in \{+,-,\ast\} \\
guard & ::= expr \uparrow expr, \text{ with } \uparrow \in \{<,\leq,=,\neq,\geq,>\} \\
Op & ::= guard \mid var := expr
\end{align*}
\]
Control Flow Automata: Syntax

\[ Q = \{ q_1, q_2, q_3, q_6, q_7, q_8, q_{11}, q_{12} \} \]

\[ q_{in} = q_1 \]

\[ q_{out} = q_{12} \]

\[ X = \{ x, y \} \]

\[ \rightarrow = \begin{cases} (q_1, x := 1, q_2), \\ (q_2, y \leq 10, q_3), \\ (q_2, y > 10, q_6), \\ (q_3, y := 10, q_{11}), \\ \ldots \end{cases} \]
Control flow automata can model:

- flow of control (program points),
- numerical variables and numerical operations,
- non-determinism (uninitialized variables, boolean inputs).

Control flow automata cannot model:

- pointers
- recursion
- threads
- ...

But they are complex enough for verification... ... and for learning!
Programs as Control Flow Automata

Control flow automata can model:

- flow of control (program points),
- numerical variables and numerical operations,
- non-determinism (uninitialized variables, boolean inputs).

Control flow automata cannot model:

- pointers
- recursion
- threads
- ...

But they are complex enough for verification... ... and for learning!
Control flow automata can model:

- flow of control (program points),
- numerical variables and numerical operations,
- non-determinism (uninitialized variables, boolean inputs).

Control flow automata cannot model:

- pointers
- recursion
- threads
- ...

But they are complex enough for verification... ... and for learning!
Goal

Check that “nothing bad can happen”.

Bad behaviors specified e.g. as assertion violations in the original program.

An assertion violation can be modeled as a location:

\[
\text{assert}(x > 0) \implies \text{if } (x > 0) \text{ then } \{ \text{BAD: } \}
\]

Goal (refined)

Check that there is no “run” that visits a location \(q\) contained in a given set \(Q_{BAD} \subseteq Q\) of bad locations.
Verification of Safety Properties

Goal

Check that “nothing bad can happen”.

Bad behaviors specified e.g. as assertion violations in the original program

An assertion violation can be modeled as a location:

\[
\text{assert}(x > 0) \quad \implies \quad \text{if } (x > 0) \text{ then } \{ \text{BAD: } \}
\]

Goal (refined)

Check that there is no “run” that visits a location \(q\) contained in a given set \(Q_{BAD} \subseteq Q\) of bad locations.
Verification of Safety Properties

Goal

Check that “nothing bad can happen”.

Bad behaviors specified e.g. as assertion violations in the original program

An assertion violation can be modeled as a location:

```
assert(x > 0)  ⇒  if (x > 0) then { BAD: }
```

Goal (refined)

Check that there is no “run” that visits a location $q$ contained in a given set $Q_{BAD} \subseteq Q$ of bad locations.
Runs: Examples

\((q_1, 0, 0)\)
\(\downarrow\)
\(x := 1\)
\(\rightarrow\)
\((q_2, 1, 0)\)
\(y \leq 10\)
\(\downarrow\)
\((q_3, 1, 0)\)
\(x \geq y\)
\(\rightarrow\)
\((q_6, 1, 0)\)
\(y := y - 1\)
\(\rightarrow\)
\((q_7, 1, 10)\)
\(x := 2 \times x\)
\(\rightarrow\)
\((q_8, 11, 10)\)
\(x := y + 1\)
Runs: Examples

\[(q_1, -159, 27)\]
\[
\downarrow x := 1
\]
\[(q_2, 1, 27)\]
\[
\downarrow y > 10
\]
\[(q_6, 1, 27)\]
\[
\downarrow x < y
\]
\[(q_7, 1, 27)\]
\[
\downarrow x := 2 \times x
\]
\[(q_8, 2, 27)\]
\[
\downarrow y := y - 1
\]
\[(q_6, 2, 26)\]
A labeled transition system is a quintuple $\langle C, \text{Init}, \text{Out}, \Sigma, \rightarrow \rangle$ where:

- $C$ is a set of configurations
- $\text{Init} \subseteq C$ and $\text{Out} \subseteq C$ are sets of initial and exit configurations
- $\Sigma$ is a finite set of actions
- $\rightarrow \subseteq C \times \Sigma \times C$ is a set of transitions

\[
\text{Post}(c, \sigma) = \left\{ c' \in C \mid c \xrightarrow{\sigma} c' \right\} \quad \text{Post}(c) = \bigcup_{\sigma \in \Sigma} \text{Post}(c, \sigma)
\]

\[
\text{Post}(U, \sigma) = \bigcup_{c \in U} \text{Post}(c, \sigma) \quad \text{Post}(U) = \bigcup_{c \in U} \text{Post}(c)
\]
A labeled transition system is a quintuple \( \langle C, \text{Init}, \text{Out}, \Sigma, \rightarrow \rangle \) where:
- \( C \) is a set of configurations
- \( \text{Init} \subseteq C \) and \( \text{Out} \subseteq C \) are sets of initial and exit configurations
- \( \Sigma \) is a finite set of actions
- \( \rightarrow \subseteq C \times \Sigma \times C \) is a set of transitions

\[
\begin{align*}
\text{Post}(c, \sigma) &= \left\{ c' \in C \mid c \xrightarrow{\sigma} c' \right\} \\
\text{Post}(U, \sigma) &= \bigcup_{c \in U} \text{Post}(c, \sigma) \\
\text{Post}(c) &= \bigcup_{\sigma \in \Sigma} \text{Post}(c, \sigma) \\
\text{Post}(U) &= \bigcup_{c \in U} \text{Post}(c)
\end{align*}
\]
A labeled transition system is a quintuple \( \langle C, \text{Init}, \text{Out}, \Sigma, \rightarrow \rangle \) where:

- \( C \) is a set of configurations
- \( \text{Init} \subseteq C \) and \( \text{Out} \subseteq C \) are sets of initial and exit configurations
- \( \Sigma \) is a finite set of actions
- \( \rightarrow \subseteq C \times \Sigma \times C \) is a set of transitions

\[
\begin{align*}
\text{Pre}(c, \sigma) &= \left\{ c' \in C \mid c' \xrightarrow{\sigma} c \right\} \\
\text{Pre}(U, \sigma) &= \bigcup_{c \in U} \text{Pre}(c, \sigma) \\
\text{Pre}(c) &= \bigcup_{\sigma \in \Sigma} \text{Pre}(c, \sigma) \\
\text{Pre}(U) &= \bigcup_{c \in U} \text{Pre}(c)
\end{align*}
\]
Consider a finite set $x$ of variables. A **valuation** is a function $v : x \rightarrow \mathbb{R}$.

### Expressions: $\llbracket e \rrbracket_v$

<table>
<thead>
<tr>
<th>Expression</th>
<th>Evaluation</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[c]_v$</td>
<td>$c$</td>
<td>$c \in \mathbb{Q}$</td>
</tr>
<tr>
<td>$[x]_v$</td>
<td>$v(x)$</td>
<td>$x \in x$</td>
</tr>
<tr>
<td>$[e_1 + e_2]_v$</td>
<td>$[e_1]_v + [e_2]_v$</td>
<td></td>
</tr>
<tr>
<td>$[e_1 - e_2]_v$</td>
<td>$[e_1]_v - [e_2]_v$</td>
<td></td>
</tr>
<tr>
<td>$[e_1 \times e_2]_v$</td>
<td>$[e_1]_v \times [e_2]_v$</td>
<td></td>
</tr>
</tbody>
</table>

### Guards: $v \models g$

<table>
<thead>
<tr>
<th>Guard</th>
<th>Evaluation</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v \models e_1 &lt; e_2$</td>
<td>$[e_1]_v &lt; [e_2]_v$</td>
<td></td>
</tr>
<tr>
<td>$v \models e_1 \leq e_2$</td>
<td>$[e_1]_v \leq [e_2]_v$</td>
<td></td>
</tr>
<tr>
<td>$v \models e_1 = e_2$</td>
<td>$[e_1]_v = [e_2]_v$</td>
<td></td>
</tr>
<tr>
<td>$v \models e_1 \neq e_2$</td>
<td>$[e_1]_v \neq [e_2]_v$</td>
<td></td>
</tr>
<tr>
<td>$v \models e_1 \geq e_2$</td>
<td>$[e_1]_v \geq [e_2]_v$</td>
<td></td>
</tr>
<tr>
<td>$v \models e_1 &gt; e_2$</td>
<td>$[e_1]_v &gt; [e_2]_v$</td>
<td></td>
</tr>
</tbody>
</table>
Consider a finite set $X$ of variables. A **valuation** is a function $\nu : X \rightarrow \mathbb{R}$.

### Expressions:

| $\mathllbracket c \mathrrbracket_\nu$ | $c$ if $c \in \mathbb{Q}$ |
| $\mathllbracket x \mathrrbracket_\nu$ | $\nu(x)$ if $x \in X$ |
| $\mathllbracket e_1 + e_2 \mathrrbracket_\nu$ | $\mathllbracket e_1 \mathrrbracket_\nu + \mathllbracket e_2 \mathrrbracket_\nu$ |
| $\mathllbracket e_1 - e_2 \mathrrbracket_\nu$ | $\mathllbracket e_1 \mathrrbracket_\nu - \mathllbracket e_2 \mathrrbracket_\nu$ |
| $\mathllbracket e_1 \times e_2 \mathrrbracket_\nu$ | $\mathllbracket e_1 \mathrrbracket_\nu \times \mathllbracket e_2 \mathrrbracket_\nu$ |

### Guards:

| $\nu \models \mathllbracket e_1 < e_2 \mathrrbracket$ | if $\mathllbracket e_1 \mathrrbracket_\nu < \mathllbracket e_2 \mathrrbracket_\nu$ |
| $\nu \models \mathllbracket e_1 \leq e_2 \mathrrbracket$ | if $\mathllbracket e_1 \mathrrbracket_\nu \leq \mathllbracket e_2 \mathrrbracket_\nu$ |
| $\nu \models \mathllbracket e_1 = e_2 \mathrrbracket$ | if $\mathllbracket e_1 \mathrrbracket_\nu = \mathllbracket e_2 \mathrrbracket_\nu$ |
| $\nu \models \mathllbracket e_1 \neq e_2 \mathrrbracket$ | if $\mathllbracket e_1 \mathrrbracket_\nu \neq \mathllbracket e_2 \mathrrbracket_\nu$ |
| $\nu \models \mathllbracket e_1 \geq e_2 \mathrrbracket$ | if $\mathllbracket e_1 \mathrrbracket_\nu \geq \mathllbracket e_2 \mathrrbracket_\nu$ |
| $\nu \models \mathllbracket e_1 > e_2 \mathrrbracket$ | if $\mathllbracket e_1 \mathrrbracket_\nu > \mathllbracket e_2 \mathrrbracket_\nu$ |
Semantics of Operations

The semantics $[\text{op}]$ of an operation $\text{op}$ is defined as a binary relation between valuations before $\text{op}$ and valuations after $\text{op}$:

$$[\text{op}] \subseteq (X \rightarrow \mathbb{R}) \times (X \rightarrow \mathbb{R})$$

Examples with $X = \{x, y\}$

$$[x \cdot y \leq 10] = \{(v, v') | v(x) \times v(y) \leq 10\}$$

$$[x := 3 \cdot x] = \{(v, v') | v'(x) = 3 \times v(x) \land v'(y) = v(y)\}$$

Operations:

$$(v, v') \in [g] \quad \text{if} \quad v \models g \quad \text{and} \quad v' = v$$

$$(v, v') \in [x := e] \quad \text{if} \quad \begin{cases} v'(x) = [e]_v \\ v'(y) = v'(y) \quad \text{for all } y \neq x \end{cases}$$
The semantics $[\text{op}]$ of an operation $\text{op}$ is defined as a binary relation between valuations before $\text{op}$ and valuations after $\text{op}$:

$$[\text{op}] \subseteq (X \to \mathbb{R}) \times (X \to \mathbb{R})$$

Examples with $X = \{x, y\}$

$$[x \cdot y \leq 10] = \{(v, v') | v(x) \cdot v(y) \leq 10\}$$

$$[x := 3 \cdot x] = \{(v, v') | v'(x) = 3 \cdot v(x) \land v'(y) = v(y)\}$$

Operations:

$$(v, v') \in [g] \text{ if } v \models g \text{ and } v' = v$$

$$(v, v') \in [x := e] \text{ if } \begin{cases} v'(x) = [e]_v \\ v'(y) = v'(y) \text{ for all } y \neq x \end{cases}$$
Semantics of Operations

The semantics $[\text{op}]$ of an operation $\text{op}$ is defined as a binary relation between valuations before $\text{op}$ and valuations after $\text{op}$:

$$[\text{op}] \subseteq (\mathbb{X} \rightarrow \mathbb{R}) \times (\mathbb{X} \rightarrow \mathbb{R})$$

Examples with $\mathbb{X} = \{x, y\}$

$$[x \cdot y \leq 10] = \{ (v, v') \mid v(x) \times v(y) \leq 10 \}$$

$$[x := 3 \cdot x] = \{ (v, v') \mid v'(x) = 3 \times v(x) \land v'(y) = v(y) \}$$

Operations:

$$(v, v') \in [g] \quad \text{if} \quad v \models g \quad \text{and} \quad v' = v$$

$$(v, v') \in [x := e] \quad \text{if} \quad \left\{ \begin{array}{l} v'(x) = [e]_v \\ v'(y) = v'(y) \quad \text{for all} \ y \neq x \end{array} \right.$$
Operational Semantics of Control Flow Automata

Definition

The interpretation of a control flow automaton \( \langle Q, q_{in}, q_{out}, X, \rightarrow \rangle \) is the labeled transition system \( \langle C, Init, Out, \circ \rightarrow \rangle \) defined by:

- \( C = Q \times (X \rightarrow \mathbb{R}) \)
- \( Init = \{ q_{in} \} \times (X \rightarrow \mathbb{R}) \) and \( Out = \{ q_{out} \} \times (X \rightarrow \mathbb{R}) \)
- \((q, v) \xrightarrow{\circ} (q', v')\) if \( q \xrightarrow{\circ} q' \) and \((v, v') \in \llbracket \circ \rrbracket\)

Two kinds of labeled directed graphs

Control Flow Automata

Use: program source codes
- Syntactic objects
- Finite

Interpretations (LTS)

Use: program behaviors
- Semantic objects
- Uncountably infinite
Operational Semantics of Control Flow Automata

Definition

The interpretation of a control flow automaton \( \langle Q, q_{in}, q_{out}, X, \rightarrow \rangle \) is the labeled transition system \( \langle C, Init, Out, \mathcal{O}, \rightarrow \rangle \) defined by:

- \( C = Q \times (X \rightarrow \mathbb{R}) \)
- \( Init = \{q_{in}\} \times (X \rightarrow \mathbb{R}) \) and \( Out = \{q_{out}\} \times (X \rightarrow \mathbb{R}) \)
- \( (q, v) \xrightarrow{\mathcal{O}} (q', v') \) if \( q \xrightarrow{\mathcal{O}} q' \) and \( (v, v') \in \llbracket \mathcal{O} \rrbracket \)

Two kinds of labeled directed graphs

Control Flow Automata

Use: program source codes
- Syntactic objects
- Finite

Interpretations (LTS)

Use: program behaviors
- Semantic objects
- Uncountably infinite
A control path is a path in the control flow automaton:

\[ q_0 \xrightarrow{\text{op}_0} q_1 \cdots q_{k-1} \xrightarrow{\text{op}_{k-1}} q_k \]

An execution path is a path in the labeled transition system:

\[(q_0, v_0) \xrightarrow{\text{op}_0} (q_1, v_1) \cdots (q_{k-1}, v_{k-1}) \xrightarrow{\text{op}_{k-1}} (q_k, v_k)\]

A run is an execution path that starts with an initial configuration:

\[(q_{\text{in}}, v_{\text{in}}) \xrightarrow{\text{op}_0} (q_1, v_1) \cdots (q_{k-1}, v_{k-1}) \xrightarrow{\text{op}_{k-1}} (q_k, v_k)\]
Control Paths, Execution Paths and Runs

A **control path** is a path in the control flow automaton:

\[ q_0 \xrightarrow{\text{op}_0} q_1 \cdots q_{k-1} \xrightarrow{\text{op}_{k-1}} q_k \]

An **execution path** is a path in the labeled transition system:

\[
(q_0, v_0) \xrightarrow{\text{op}_0} (q_1, v_1) \cdots (q_{k-1}, v_{k-1}) \xrightarrow{\text{op}_{k-1}} (q_k, v_k)
\]

A **run** is an execution path that starts with an initial configuration:

\[
(q_{in}, v_{in}) \xrightarrow{\text{op}_0} (q_1, v_1) \cdots (q_{k-1}, v_{k-1}) \xrightarrow{\text{op}_{k-1}} (q_k, v_k)
\]
Execution Path: Example

\[ x := 1 \]
\[ y \leq 10 \]
\[ y > 10 \]
\[ y := y - 1 \]
\[ x \geq y \]
\[ y := 10 \]
\[ x := 2 \times x \]
\[ x := y + 1 \]
\[ (q_1, -159, 27) \]
\[ (q_2, 1, 27) \]
\[ (q_6, 1, 27) \]
\[ (q_7, 1, 27) \]
\[ (q_8, 2, 27) \]
\[ (q_6, 2, 26) \]

Grégoire Sutre  
Software Verification  
Control Flow Automata  
VTSA’08 41 / 286
Outline — Control Flow Automata

3 Syntax and Semantics

4 Verification of Control Flow Automata
Forward Reachability Set Post*

Set of all configurations that are reachable from an initial configuration

\[ \text{Post}^* = \bigcup_{\rho: \text{run}} \{(q, v) \mid (q, v) \text{ occurs on } \rho\} \]

\[ = \bigcup_{i \in \mathbb{N}} \text{Post}^i(\text{Init}) \]

\[ = \bigcup_{q_{in} \xrightarrow{\text{op}_0} \cdots \xrightarrow{\text{op}_{k-1}} q} \{q\} \times ([\text{op}_{k-1}] \circ \cdots \circ [\text{op}_0])[(X \rightarrow \mathbb{R})] \]
Forward Reachability Set Post* on Running Example

$q_1 : \mathbb{R} \times \mathbb{R}$
$q_2 : \{1\} \times \mathbb{R}$
$q_3 : \{1\} \times ] - \infty, 10]$;
$q_6 : \{1\} \times ]10, +\infty[ \cup \{2\} \times ]9, +\infty[ \cup \{4\} \times ]8, +\infty[ \cup \ldots$

- $x := 1$
- $y \leq 10$
- $y > 10$
- $y := y - 1$
- $x := x + 1$
- $y := 10$
- $x \geq y$
- $x < y$
- $x := 2 \times x$
- $y$
Forward Reachability Set Post* on Running Example

$q_1 : \mathbb{R} \times \mathbb{R}$
$q_2 : \{1\} \times \mathbb{R}$
$q_3 : \{1\} \times ]-\infty, 10]$ 
$q_6 : \{1\} \times ]10, +\infty[ \cup \{2\} \times ]9, +\infty[ \cup \{4\} \times ]8, +\infty[ \cup \ldots$

$q_6 : \exists i \in \mathbb{N} \cdot \left\{ x = 2^i \land y + i > 10 \land i \geq 1 \implies 2^{i-1} < y + 1 \right\}$
Backward Reachability Set $\text{Pre}^*$

Set of all configurations that can reach an exit configuration

\[ \text{Pre}^* = \bigcup_{i \in \mathbb{N}} \text{Pre}^i(Out) \]

\[ = \bigcup \{q\} \times (\circ \circ \circ \circ \circ \circ (X \rightarrow \mathbb{R})) \]

\[ = \bigcup \{q\} \times (\circ \circ \circ \circ \circ \circ (X \rightarrow \mathbb{R})) \]
Verification of Control Flow Automata

Goal (Repetition)
Check that there is no run that visits a location $q$ contained in a given set $Q_{BAD} \subseteq Q$ of bad locations.

Define the set $Bad$ of bad configurations by: $Bad = Q_{BAD} \times (X \rightarrow \mathbb{R})$.

Goal (Equivalent Formulation)
Check that $Post^*$ is disjoint from $Bad$.

Undecidability
The location reachability and configuration reachability problems are both undecidable for control flow automata.

Proof by reduction to location reachability in two-counters machines.
Two-Counters (Minsky) Machines

Finite-state automaton extended with:

- two counters over nonnegative integers
- test for zero, increment and guarded decrement

Reachability is undecidable for this class.

Any two-counters machine can (effectively) be represented as a control flow automaton in this restricted class:

- two variables: \( X = \{c_1, c_2\} \)
- allowed guards: \( x = 0 \) and \( x \neq 0 \) for each \( x \in X \)
- allowed assignments: \( x := x + 1 \) and \( x := x - 1 \) for each \( x \in X \)
Two-Counters Machines as Control Flow Automata

Two-Counters (Minsky) Machines

Finite-state automaton extended with:
- two counters over nonnegative integers
- test for zero, increment and guarded decrement
Reachability is undecidable for this class.

Any two-counters machine can (effectively) be represented as a control flow automaton in this restricted class:
- two variables: \( X = \{c_1, c_2\} \)
- allowed guards: \( x = 0 \) and \( x \neq 0 \) for each \( x \in X \)
- allowed assignments: \( x := x + 1 \) and \( x := x - 1 \) for each \( x \in X \)
**Definition**

An **invariant** is any set $\text{Inv} \subseteq C$ such that $\text{Post}^* \subseteq \text{Inv}$.

**Idea:**

1. Compute an invariant $\text{Inv}$ (easier to compute than $\text{Post}^*$)
2. If $\text{Inv}$ is disjoint from $\text{Bad}$ then $\text{Post}^*$ is also disjoint from $\text{Bad}$

Rest of the lecture:

**Computation of precise enough invariants**
**Definition**

An invariant is any set $\text{Inv} \subseteq C$ such that $\text{Post}^* \subseteq \text{Inv}$.

**Idea:**

1. Compute an invariant $\text{Inv}$ (easier to compute than $\text{Post}^*$)
2. If $\text{Inv}$ is disjoint from $\text{Bad}$ then $\text{Post}^*$ is also disjoint from $\text{Bad}$

Rest of the lecture:

**Computation of precise enough invariants**
Computational model for programs: control flow automata
  - syntax
  - semantics

Undecidability in general of model-checking for control flow automata

Tentative solution: computation of invariants
Part III

Data Flow Analysis
Outline — Data Flow Analysis

5. Classical Data Flow Analyses
6. Basic Lattice Theory
7. Monotone Data Flow Analysis Frameworks
Outline — Data Flow Analysis

5  Classical Data Flow Analyses

6  Basic Lattice Theory

7  Monotone Data Flow Analysis Frameworks
Short Introduction to Data Flow Analysis

Tentative Definition
Compile-time techniques to gather run-time information about data in programs without actually running them.

Applications
Code optimization
- Avoid redundant computations (e.g. reuse available results)
- Avoid superfluous computations (e.g. eliminate dead code)

Code validation
- Invariant generation

Conservative approximations
A variable \( x \) is **live** at location \( q \) if there exists a control path starting from \( q \) where \( x \) is used before it is modified.
A variable $x$ is live at location $q$ if there exists a control path starting from $q$ where $x$ is used before it is modified.
A variable \( x \) is \textit{live} at location \( q \) if there exists a control path starting from \( q \) where \( x \) is used before it is modified.
A variable $x$ is **live** at location $q$ if there exists a control path starting from $q$ where $x$ is used before it is modified.

**Example:**

1. $x := 1$
2. $y := x + 3$
3. $x \geq y$
4. $x := 0$

$x$ **live**, $y$ **live**

**Example:**

1. $x := 1$
2. $y := y + 3$
3. $x \geq 0$
4. $x := 0$

$x$ **not live**, $y$ **live**
A variable $x$ is **live** at location $q$ if there exists a control path starting from $q$ where $x$ is used before it is modified.
Live Variables Analysis: Running Example

$q_1$: $x := 1$

$q_2$: $y \leq 10$

$q_3$: $y > 10$

$q_4$: $y := 10$

$q_5$: $x \geq y$

$q_6$: $x < y$

$q_7$: $x := 2 \times x$

$q_8$: $y := y - 1$

$q_{11}$: $x := y + 1$

$q_{12}$
Live Variables Analysis: Running Example

$q_1$ $\xrightarrow{x := 1}$ $q_2$

$q_2$ $\xrightarrow{y \leq 10}$ $q_3$ $\xrightarrow{x \geq y}$ $q_11$ $\xrightarrow{x := y + 1}$ $q_{12}$

$q_2$ $\xrightarrow{y > 10}$ $q_6$ $\xrightarrow{x < y}$ $q_7$ $\xrightarrow{x := 2x}$ $q_8$

$q_6$ $\xrightarrow{y := y - 1}$ $q_7$ $\xrightarrow{x < y}$ $q_8$

$q_8$ $\xrightarrow{y := 10}$ $q_7$ $\xrightarrow{x < y}$ $q_8$

$q_7$ $\xrightarrow{x := y}$ $q_6$ $\xrightarrow{y := y - 1}$ $q_8$

$q_8$ $\xrightarrow{y := 10}$ $q_7$ $\xrightarrow{x < y}$ $q_8$

$q_8$ $\xrightarrow{y := 10}$ $q_7$ $\xrightarrow{x < y}$ $q_8$

$q_8$ $\xrightarrow{y := 10}$ $q_7$ $\xrightarrow{x < y}$ $q_8$

$q_8$ $\xrightarrow{y := 10}$ $q_7$ $\xrightarrow{x < y}$ $q_8$
0: Initialization
1: Local information
2: Propagation (← )

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>q1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q12</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

x := 1
y ≤ 10
y > 10
y := y - 1
x ≥ y
x < y
x := 2 * x
x := y + 1
x := y
y := 10
0: Initialization
1: Local information
2: Propagation (←)

\[
\begin{array}{c|c|c}
\text{q} & x & y \\
\hline
q_1 & & \\
q_2 & \leq 10 & > 10 \\
q_3 & & \geq y \\
q_6 & x < y & y := y - 1 \\
q_7 & & x := 2 * x \\
q_8 & & \\
q_{11} & x := 1 & y := 10 \\
q_{12} & & x := y + 1
\end{array}
\]
Live Variables Analysis: Running Example

\begin{itemize}
\item \textbf{0 : Initialization}
\item \textbf{1 : Local information}
\item \textbf{2 : Propagation (←)}
\end{itemize}

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|}
\hline
\textbf{State} & \textbf{x} & \textbf{y} \\
\hline
\textbf{q}_1 & & \\
\textbf{q}_2 & \textbullet & \textbullet \\
\textbf{q}_3 & & \\
\textbf{q}_6 & \textbullet & \textbullet \\
\textbf{q}_7 & & \\
\textbf{q}_8 & & \\
\textbf{q}_{11} & & \\
\textbf{q}_{12} & & \\
\hline
\end{tabular}
\end{table}
Live Variables Analysis: Running Example

0: Initialization
1: Local information
2: Propagation (←)

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>q₁</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q₂</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q₃</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q₆</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q₇</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q₈</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q₁₁</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q₁₂</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

x := 1
y := 10
y ≤ 10
y > 10
y := y - 1
y ≥ y
x ≥ y
x < y
x := 2 * x
x := y + 1
Live Variables Analysis: Running Example

0: Initialization
1: Local information
2: Propagation (←)

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>q1</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>q2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q12</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Live Variables Analysis: Running Example

0: Initialization
1: Local information
2: Propagation (←→)

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>q₁</td>
<td>•</td>
<td></td>
</tr>
<tr>
<td>q₂</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>q₃</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>q₆</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>q₇</td>
<td></td>
<td>•</td>
</tr>
<tr>
<td>q₈</td>
<td></td>
<td>•</td>
</tr>
<tr>
<td>q₁₁</td>
<td></td>
<td>•</td>
</tr>
<tr>
<td>q₁₂</td>
<td></td>
<td>•</td>
</tr>
</tbody>
</table>

x := 1
y ≤ 10

y > 10

y := y - 1

x ≥ y

x := 2 * x

x := y + 1
0: Initialization
1: Local information
2: Propagation (←)

```
<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>q1</td>
<td></td>
<td>.</td>
</tr>
<tr>
<td>q2</td>
<td>.</td>
<td></td>
</tr>
<tr>
<td>q3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q7</td>
<td>.</td>
<td></td>
</tr>
<tr>
<td>q8</td>
<td></td>
<td>.</td>
</tr>
<tr>
<td>q9</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>q10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q11</td>
<td>.</td>
<td></td>
</tr>
<tr>
<td>q12</td>
<td></td>
<td>.</td>
</tr>
</tbody>
</table>
```

- $x := 1$
- $y \leq 10$
- $y > 10$
- $x := 2 \times x$
- $x := y + 1$
- $y := 10$
- $y := y - 1$
- $x < y$
Live Variables Analysis: Running Example

0: Initialization
1: Local information
2: Propagation (←)

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>q1</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>q2</td>
<td></td>
<td>●</td>
</tr>
<tr>
<td>q3</td>
<td></td>
<td>●</td>
</tr>
<tr>
<td>q6</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>q7</td>
<td></td>
<td>●</td>
</tr>
<tr>
<td>q8</td>
<td></td>
<td>●</td>
</tr>
<tr>
<td>q11</td>
<td></td>
<td>●</td>
</tr>
<tr>
<td>q12</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

x := 1
y <= 10
y > 10
y := y - 1
x := 2*x

x := y + 1

x := 1
y := 10
x < y
x >= y
y := 10
Live Variables Analysis: Running Example

0: Initialization
1: Local information
2: Propagation (←)

$q_1$
$q_2$
$q_3$
$q_6$
$q_7$
$q_8$
$q_{11}$
$q_{12}$

$x := 1$
$y \leq 10$
$y > 10$
$y := y - 1$

$y := 10$
$x \geq y$
x < y

$x := 2 \times x$

$x := y + 1$

|$x$ | $y$
|---|---|
$q_1$ | • |
$q_2$ | • | • |
$q_3$ | |
$q_6$ | • | • |
$q_7$ | • | • |
$q_8$ | • | • |
$q_{11}$ | • |
$q_{12}$ | • |
0: Initialization
1: Local information
2: Propagation (←)

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>q₁</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>q₂</td>
<td></td>
<td>● ●</td>
</tr>
<tr>
<td>q₃</td>
<td></td>
<td>●</td>
</tr>
<tr>
<td>q₆</td>
<td>●</td>
<td>● ●</td>
</tr>
<tr>
<td>q₇</td>
<td>●</td>
<td>● ●</td>
</tr>
<tr>
<td>q₈</td>
<td>●</td>
<td>● ●</td>
</tr>
<tr>
<td>q₁₁</td>
<td></td>
<td>●</td>
</tr>
</tbody>
</table>

- x := 1
- y := 10
- y > 10
- y := y - 1
- x ≥ y
- x ≤ 10
- x := 2 * x
- x := y + 1
- x := y
Live Variables Analysis: Running Example

0: Initialization
1: Local information
2: Propagation (←)

\begin{center}
\begin{tabular}{|c|c|}
\hline
\textbf{q} & \textbf{x} \textbf{y} \\
\hline
\textbf{q}_1 & \bullet \textbf{y} \\
\textbf{q}_2 & \bullet \textbf{y} \\
\textbf{q}_3 & \textbf{y} \\
\textbf{q}_4 & \textbf{y} \textbf{y} \\
\textbf{q}_5 & \textbf{y} \\
\textbf{q}_6 & \textbf{y} \textbf{y} \\
\textbf{q}_7 & \textbf{y} \\
\textbf{q}_8 & \textbf{y} \\
\textbf{q}_9 & \textbf{y} \\
\textbf{q}_{10} & \textbf{y} \\
\textbf{q}_{11} & \textbf{y} \\
\textbf{q}_{12} & \textbf{y} \\
\hline
\end{tabular}
\end{center}
Live Variables Analysis: Formulation

Control Flow Automaton: \( \langle Q, q_{in}, q_{out}, X, \rightarrow \rangle \)

System of equations: variables \( L_q \) for \( q \in Q \), with \( L_q \subseteq X \)

\[
L_q = \bigcup_{q \xrightarrow{op} q'} \text{Gen}_\text{op} \cup (L_{q'} \setminus \text{Kill}_\text{op})
\]

\[
L(q_{out}) = \emptyset
\]

\[
\text{Gen}_\text{op} = \begin{cases} 
\text{Var}(g) & \text{if } op = g \\
\text{Var}(e) & \text{if } op = x := e
\end{cases}
\]

\[
\text{Kill}_\text{op} = \begin{cases} 
\emptyset & \text{if } op = g \\
\{x\} & \text{if } op = x := e
\end{cases}
\]

\[
f_{op}(X) = \text{Gen}_\text{op} \cup (X \setminus \text{Kill}_\text{op})
\]
Live Variables Analysis: Formulation

Control Flow Automaton: \( \langle Q, q_{in}, q_{out}, X, \rightarrow \rangle \)

System of equations: variables \( L_q \) for \( q \in Q \), with \( L_q \subseteq X \)

\[
L_q = \bigcup_{q \xrightarrow{\text{op}} q'} f_{\text{op}}(L_{q'}) \\
L(q_{out}) = \emptyset
\]

\[
Gen_{\text{op}} = \begin{cases} Var(g) & \text{if } \text{op} = g \\ Var(e) & \text{if } \text{op} = x := e \end{cases} \\
Kill_{\text{op}} = \begin{cases} \emptyset & \text{if } \text{op} = g \\ \{x\} & \text{if } \text{op} = x := e \end{cases}
\]

\[
f_{\text{op}}(X) = Gen_{\text{op}} \cup (X \setminus Kill_{\text{op}})
\]
If $x$ is not live at location $q_2$ then we may remove the assignment $x := e$ on the edge from $q_1$ to $q_2$.

This is sound since the analysis is conservative.
A expression $e$ is available at location $q$ if every control path from $q_{\text{in}}$ to $q$ contains an evaluation of $e$ which is not followed by an assignment of any variable $x$ occurring in $e$. 
A expression $e$ is available at location $q$ if every control path from $q_{in}$ to $q$ contains an evaluation of $e$ which is not followed by an assignment of any variable $x$ occurring in $e$. 
A expression $e$ is **available** at location $q$ if every control path from $q_{in}$ to $q$ contains an evaluation of $e$ which is not followed by an assignment of any variable $x$ occurring in $e$. 

$x-1$ available, $x*y$ not available
A expression $e$ is available at location $q$ if every control path from $q_{in}$ to $q$ contains an evaluation of $e$ which is not followed by an assignment of any variable $x$ occurring in $e$.

$x - 1$ available, $x \times y$ not available
Available Expressions Analysis: Definition

Definition

A expression $e$ is available at location $q$ if every control path from $q_{in}$ to $q$ contains an evaluation of $e$ which is not followed by an assignment of any variable $x$ occurring in $e$.
Available Expressions Analysis: Other Example

q₁ → q₂
a := c * d

q₂ → q₁, q₃
b + 1 ≤ 10, b + 1 > 10

q₂ → q₆

q₃ → q₂, q₄
a ≥ b

c := 5

q₄ → q₂

q₆ → q₇, q₈
a < b

q₇ → q₈
b := 2 * a

q₈ → q₇

q₁₁ → q₁, q₁₂

a := 2 * a

q₁₂ → q₁₁

a := 2 * a
Available Expressions Analysis: Other Example

\[
\begin{align*}
q_1 & \quad a := c \cdot d \\
q_2 & \quad b + 1 \leq 10 \\
q_3 & \quad b + 1 > 10 \\
q_4 & \quad a \geq b \\
q_5 & \quad a < b \\
q_6 & \quad a := b + 1 \\
q_7 & \quad b := 2 \cdot a \\
q_8 & \quad a := 2 \cdot a
\end{align*}
\]
Available Expressions Analysis: Other Example

0 : Initialization
1 : Local information
2 : Propagation (→)

<table>
<thead>
<tr>
<th></th>
<th>c*d</th>
<th>b+1</th>
<th>2*a</th>
</tr>
</thead>
<tbody>
<tr>
<td>q₁</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q₂</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q₃</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q₄</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q₅</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q₆</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q₇</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q₈</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q₉</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q₁₀</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q₁₁</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q₁₂</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Grégoire Sutre
Software Verification
Data Flow Analysis
VTSA'08 60 / 286
Available Expressions Analysis: Other Example

Grégoire Sutre
Software Verification
Data Flow Analysis

0 : Initialization
1 : Local information
2 : Propagation (→)

<table>
<thead>
<tr>
<th></th>
<th>c*d</th>
<th>b+1</th>
<th>2*a</th>
</tr>
</thead>
<tbody>
<tr>
<td>q1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q13</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q14</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
\begin{align*}
q_1 & : a := c \times d \\
q_2 & : b + 1 \leq 10 \\
q_3 & : b + 1 > 10 \\
q_4 & : a := b + 1 \\
q_5 & : c := 5 \\
q_6 & : a \geq b \\
q_7 & : a < b \\
q_8 & : b := 2 \times a \\
q_9 & : a := 2 \times a
\end{align*}
\]
0: Initialization
1: Local information
2: Propagation (→)

<table>
<thead>
<tr>
<th></th>
<th>c*d</th>
<th>b+1</th>
<th>2*a</th>
</tr>
</thead>
<tbody>
<tr>
<td>q1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q2</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>q3</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>q6</td>
<td>●</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>q7</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>q8</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>q11</td>
<td>●</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>q12</td>
<td>●</td>
<td>●</td>
<td></td>
</tr>
</tbody>
</table>
Available Expressions Analysis: Other Example

0 : Initialization
1 : Local information
2 : Propagation (→)

<table>
<thead>
<tr>
<th></th>
<th>c*d</th>
<th>b+1</th>
<th>2*a</th>
</tr>
</thead>
<tbody>
<tr>
<td>q1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q12</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ a := c \times d \]
\[ b + 1 \leq 10 \]
\[ b + 1 > 10 \]
\[ c := 5 \]
\[ a \geq b \]
\[ a < b \]
\[ a := b + 1 \]
\[ b := 2 \times a \]
\[ a := 2 \times a \]
Available Expressions Analysis: Other Example

0: Initialization
1: Local information
2: Propagation (→)

<table>
<thead>
<tr>
<th></th>
<th>c*d</th>
<th>b+1</th>
<th>2*a</th>
</tr>
</thead>
<tbody>
<tr>
<td>q1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q12</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

a := c*d

b+1 ≤ 10
b+1 > 10
a ≥ b
a < b
b := 2*a
a := 2*a

c := 5

Grégoire Sutre
Software Verification
Data Flow Analysis
VTSA’08 60 / 286
Available Expressions Analysis: Other Example

0 : Initialization
1 : Local information
2 : Propagation (→)

\[ a := c \times d \]
\[ b + 1 \leq 10 \]
\[ b + 1 > 10 \]
\[ a := b + 1 \]
\[ a \geq b \]
\[ a < b \]
\[ c := 5 \]
\[ b := 2 \times a \]
\[ a := 2 \times a \]

<table>
<thead>
<tr>
<th></th>
<th>c \times d</th>
<th>b + 1</th>
<th>2 \times a</th>
</tr>
</thead>
<tbody>
<tr>
<td>q_1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q_2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q_3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q_6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q_7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q_8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q_11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q_{12}</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Grégoire Sutre
Software Verification
Data Flow Analysis
VTSA’08 60 / 286
Available Expressions Analysis: Other Example

0: Initialization
1: Local information
2: Propagation (→)

<table>
<thead>
<tr>
<th></th>
<th>c*d</th>
<th>b+1</th>
<th>2*a</th>
</tr>
</thead>
<tbody>
<tr>
<td>q1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q12</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>a := c*d</th>
<th>b+1 ≤ 10</th>
<th>b+1 &gt; 10</th>
<th>a := b+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>q1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q11</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q12</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Grégoire Sutre
Software Verification
Data Flow Analysis
VTSA’08 60 / 286
Available Expressions Analysis: Other Example

0 : Initialization
1 : Local information
2 : Propagation (→)

<table>
<thead>
<tr>
<th></th>
<th>c*d</th>
<th>b+1</th>
<th>2*a</th>
</tr>
</thead>
<tbody>
<tr>
<td>q1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q3</td>
<td>●</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>q4</td>
<td>●</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>q6</td>
<td>●</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>q7</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>q8</td>
<td>●</td>
<td></td>
<td>●</td>
</tr>
<tr>
<td>q9</td>
<td></td>
<td></td>
<td>●</td>
</tr>
<tr>
<td>q10</td>
<td>●</td>
<td>●</td>
<td></td>
</tr>
</tbody>
</table>

a := c*d
b+1 ≤ 10
b+1 > 10
a := b+1
a := 2*a
b := 2*a
a ≥ b
c := 5
a < b
0 : Initialization
1 : Local information
2 : Propagation (→)

<table>
<thead>
<tr>
<th></th>
<th>c*d</th>
<th>b+1</th>
<th>2*a</th>
</tr>
</thead>
<tbody>
<tr>
<td>q1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q12</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

a := c*d
b+1 ≤ 10
b+1 > 10
a := b+1
a ≥ b
a < b
b := 2*a
a := 2*a
c := 5
Available Expressions Analysis: Other Example

0: Initialization
1: Local information
2: Propagation (→)

<table>
<thead>
<tr>
<th></th>
<th>c*d</th>
<th>b+1</th>
<th>2*a</th>
</tr>
</thead>
<tbody>
<tr>
<td>q1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q3</td>
<td>•</td>
<td>•</td>
<td></td>
</tr>
<tr>
<td>q4</td>
<td>•</td>
<td>•</td>
<td></td>
</tr>
<tr>
<td>q5</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>q6</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>q7</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>q8</td>
<td>•</td>
<td>•</td>
<td></td>
</tr>
<tr>
<td>q9</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Grégoire Sutre
Software Verification
Data Flow Analysis
VTSA'08
Available Expressions Analysis: Other Example

\[
\begin{align*}
q_1 & \quad a := c \cdot d \\
q_2 & \quad b + 1 \leq 10, \quad b + 1 > 10 \\
q_3 & \quad c := 5 \\
q_6 & \quad a \geq b \\
q_7 & \quad a < b \\
q_8 & \quad b := 2 \cdot a \\
q_{11} & \quad a := 2 \cdot a \\
q_{12} & \quad \text{Initialization} \\
\end{align*}
\]

0: Initialization
1: Local information
2: Propagation (→)

<table>
<thead>
<tr>
<th></th>
<th>c \cdot d</th>
<th>b + 1</th>
<th>2 \cdot a</th>
</tr>
</thead>
<tbody>
<tr>
<td>q_1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q_2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q_3</td>
<td>\bullet</td>
<td>\bullet</td>
<td></td>
</tr>
<tr>
<td>q_4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q_5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q_6</td>
<td>\bullet</td>
<td>\bullet</td>
<td></td>
</tr>
<tr>
<td>q_7</td>
<td>\bullet</td>
<td>\bullet</td>
<td>\bullet</td>
</tr>
<tr>
<td>q_8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q_{11}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q_{12}</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Available Expressions Analysis: Other Example

\[ a := c \times d \]

\[ b + 1 \leq 10 \]

\[ b + 1 > 10 \]

\[ a := b + 1 \]

\[ a \geq b \]

\[ a < b \]

\[ c := 5 \]

\[ a := 2 \times a \]

\[ b := 2 \times a \]

\[ q_1 \]

\[ q_2 \]

\[ q_3 \]

\[ q_6 \]

\[ q_7 \]

\[ q_{11} \]

\[ q_{12} \]

\[ q_8 \]

\[ q_{11} \]

\[ q_{12} \]

0 : Initialization
1 : Local information
2 : Propagation (→)

<table>
<thead>
<tr>
<th>q</th>
<th>( c \times d )</th>
<th>( b + 1 )</th>
<th>( 2 \times a )</th>
</tr>
</thead>
<tbody>
<tr>
<td>q₁</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>q₂</td>
<td>●</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q₃</td>
<td>●</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>q₆</td>
<td>●</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>q₇</td>
<td>●</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>q₈</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>q₁₁</td>
<td></td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>q₁₂</td>
<td>●</td>
<td>●</td>
<td></td>
</tr>
</tbody>
</table>
Available Expressions Analysis: Other Example

0 : Initialization
1 : Local information
2 : Propagation (→)

\[
a := c \times d
\]
\[
b + 1 \leq 10
\]
\[
b + 1 > 10
\]
\[
a := b + 1
\]
\[
c := 5
\]
\[
a \geq b
\]
\[
a < b
\]
\[
b := 2 \times a
\]
\[
a := 2 \times a
\]

<table>
<thead>
<tr>
<th></th>
<th>(c \times d)</th>
<th>(b + 1)</th>
<th>(2 \times a)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(q_1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(q_2)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(q_3)</td>
<td>.</td>
<td>.</td>
<td></td>
</tr>
<tr>
<td>(q_4)</td>
<td>.</td>
<td>.</td>
<td></td>
</tr>
<tr>
<td>(q_5)</td>
<td>.</td>
<td>.</td>
<td></td>
</tr>
<tr>
<td>(q_6)</td>
<td>.</td>
<td>.</td>
<td></td>
</tr>
<tr>
<td>(q_7)</td>
<td>.</td>
<td>.</td>
<td></td>
</tr>
<tr>
<td>(q_8)</td>
<td>.</td>
<td>.</td>
<td></td>
</tr>
<tr>
<td>(q_{11})</td>
<td>.</td>
<td></td>
<td>.</td>
</tr>
<tr>
<td>(q_{12})</td>
<td>.</td>
<td></td>
<td>.</td>
</tr>
</tbody>
</table>
Available Expressions Analysis: Formulation

Control Flow Automaton: $\langle Q, q_{in}, q_{out}, X, \rightarrow \rangle$

System of equations: variables $A_q$, with $A_q \subseteq \text{SubExp}(\rightarrow)$

$$A_q = \bigcap_{q' \xrightarrow{\text{op}} q} \text{Gen}_{\text{op}} \cup (A_{q'} \setminus \text{Kill}_{\text{op}})$$

$$A(q_{in}) = \emptyset$$

$\text{Gen}_{\text{op}} = \begin{cases} \text{SubExp}(g) & \text{if } \text{op} = g \\ \{ f \in \text{SubExp}(e) \mid x \notin \text{SubExp}(e) \} & \text{if } \text{op} = x := e \end{cases}$

$\text{Kill}_{\text{op}} = \begin{cases} \emptyset & \text{if } \text{op} = g \\ \{ e \in \text{SubExp}(\rightarrow) \mid x \in \text{Var}(e) \} & \text{if } \text{op} = x := e \end{cases}$

$$f_{\text{op}}(X) = \text{Gen}_{\text{op}} \cup (X \setminus \text{Kill}_{\text{op}})$$
Control Flow Automaton: \( \langle Q, q_{\text{in}}, q_{\text{out}}, X, \rightarrow \rangle \)

System of equations: variables \( A_q \), with \( A_q \subseteq \text{SubExp}(\rightarrow) \)

\[
A_q = \bigcap_{q' \xrightarrow{\text{op}} q} f_{\text{op}}(A_{q'})
\]

\[
A(q_{\text{in}}) = \emptyset
\]

\[
\text{Gen}_{\text{op}} = \begin{cases} 
\text{SubExp}(g) & \text{if } \text{op} = g \\
\{ f \in \text{SubExp}(e) \mid x \not\in \text{SubExp}(e) \} & \text{if } \text{op} = x := e
\end{cases}
\]

\[
\text{Kill}_{\text{op}} = \begin{cases} 
\emptyset & \text{if } \text{op} = g \\
\{ e \in \text{SubExp}(\rightarrow) \mid x \in \text{Var}(e) \} & \text{if } \text{op} = x := e
\end{cases}
\]

\[
f_{\text{op}}(X) = \text{Gen}_{\text{op}} \cup (X \setminus \text{Kill}_{\text{op}})
\]
Available Expressions Analysis: Applications

Code Optimization

Avoid recomputation of an expression

If $e$ is available at location $q_1$ then we may reuse its value to evaluate the operation on the edge from $q_1$ to $q_2$.

This is sound since the analysis is conservative.
A variable $x$ is **constant** at location $q$ if we have $v(x) = v'(x)$ for any two reachable configurations $(q, v)$ and $(q, v')$ in Post*.
A variable $x$ is **constant** at location $q$ if we have $v(x) = v'(x)$ for any two reachable configurations $(q, v)$ and $(q, v')$ in Post$^*$. 

Grégoire Sutre

Software Verification

Data Flow Analysis

VTSA’08 63 / 286
A variable $x$ is constant at location $q$ if we have $v(x) = v'(x)$ for any two reachable configurations $(q, v)$ and $(q, v')$ in Post*.
A variable $x$ is constant at location $q$ if we have $v(x) = v'(x)$ for any two reachable configurations $(q, v)$ and $(q, v')$ in $\text{Post}^*$. 

$x := 7$

$y := x - 3$

$x := 2$

$y := 2 \times x$

$x$ not constant, $y$ constant
A variable $x$ is **constant** at location $q$ if we have $\nu(x) = \nu'(x)$ for any two reachable configurations $(q, \nu)$ and $(q, \nu')$ in $\text{Post}^*$. 

$x := 7$

$y := x - 3$

$x := 2$

$y := 2 \times x$

$x$ not constant, $y$ constant

$x := 2$

$y := x - 3$

$y := 2 \times z$

$x$ constant, $y$ not constant
x := 1

y ≤ 10

y > 10

y := y - 1

x ≥ y

x < y

x := 2 * x

x := y + 1

x := 1
Constant Propagation Analysis: Running Example

0 : Initialization
1 : Propagation (→)

<table>
<thead>
<tr>
<th>q1</th>
<th>q2</th>
<th>q3</th>
<th>q6</th>
<th>q7</th>
<th>q8</th>
<th>q11</th>
<th>q12</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>y</td>
<td>y ≥ y</td>
<td>x &lt; y</td>
<td>x ≥ y</td>
<td>y := y - 1</td>
<td>x := 2 * x</td>
<td>x := y + 1</td>
</tr>
</tbody>
</table>

y := 10
y ≤ 10
y > 10

x := 1

x := 1

y := 10

Grégoire Sutre
Software Verification
Data Flow Analysis
VTSA’08 64 / 286
Constant Propagation Analysis: Running Example

0: Initialization
1: Propagation (→)

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>q₁</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>q₂</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q₃</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q₆</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q₇</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q₈</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q₁₁</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q₁₂</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

x := 1
y ≤ 10
y > 10
y := y - 1
x ≥ y
x < y
x := 2 * x
x := y + 1
x := 2 * x
x := y - 1
x := 1
Constant Propagation Analysis: Running Example

0 : Initialization
1 : Propagation (→)

\[
\begin{array}{|c|c|c|}
\hline
 & x & y \\
q_1 & T & T \\
q_2 & 1 & T \\
q_3 & & \\
q_4 & & \\
q_5 & & \\
q_6 & & \\
q_7 & & \\
q_8 & & \\
q_9 & & \\
q_{10} & & \\
q_{11} & & \\
q_{12} & & \\
\hline
\end{array}
\]

\[
x := 1 \\
y := 10 \\
x \leq 10 \\
y > 10 \\
x = y \\
x := 2 * x \\
x := y + 1 \\
y := y - 1 \\
x \geq y \\
x < y \\
\]
Constant Propagation Analysis: Running Example

0: Initialization
1: Propagation (→)

<table>
<thead>
<tr>
<th>State</th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>q₁</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>q₂</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q₃</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q₆</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q₇</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q₈</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q₁₁</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q₁₂</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Constant Propagation Analysis: Running Example

0 : Initialization
1 : Propagation (→)

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>q1</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>q2</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q3</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q12</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Constant Propagation Analysis: Running Example

0: Initialization
1: Propagation (→)

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>q_1</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>q_2</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q_3</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q_6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q_7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q_8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q_{11}</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>q_{12}</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Constant Propagation Analysis: Running Example

0: Initialization
1: Propagation ($\rightarrow$)

<table>
<thead>
<tr>
<th>State</th>
<th>$x$</th>
<th>$y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_1$</td>
<td>$\top$</td>
<td>$\top$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>1</td>
<td>$\top$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>1</td>
<td>$\top$</td>
</tr>
<tr>
<td>$q_6$</td>
<td>$1$</td>
<td>$10$</td>
</tr>
<tr>
<td>$q_7$</td>
<td>$1$</td>
<td>$10$</td>
</tr>
<tr>
<td>$q_8$</td>
<td>$1$</td>
<td>$10$</td>
</tr>
<tr>
<td>$q_{11}$</td>
<td>$1$</td>
<td>$10$</td>
</tr>
<tr>
<td>$q_{12}$</td>
<td>$1$</td>
<td>$10$</td>
</tr>
</tbody>
</table>
Constant Propagation Analysis: Running Example

0: Initialization
1: Propagation (→)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>q₁</td>
<td>Q₂</td>
</tr>
<tr>
<td>q₂</td>
<td>Q₃</td>
</tr>
<tr>
<td>q₃</td>
<td>Q₆</td>
</tr>
<tr>
<td>q₆</td>
<td>Q₇</td>
</tr>
<tr>
<td>q₇</td>
<td>Q₈</td>
</tr>
<tr>
<td>q₈</td>
<td>Q₁₁</td>
</tr>
<tr>
<td>q₁₁</td>
<td>Q₁₂</td>
</tr>
<tr>
<td>q₁₂</td>
<td>Q₁₀</td>
</tr>
<tr>
<td>q₁₀</td>
<td>Q₉</td>
</tr>
<tr>
<td>q₉</td>
<td>Q₈</td>
</tr>
<tr>
<td>q₈</td>
<td>Q₇</td>
</tr>
<tr>
<td>q₇</td>
<td>Q₆</td>
</tr>
<tr>
<td>q₆</td>
<td>Q₅</td>
</tr>
<tr>
<td>q₅</td>
<td>Q₄</td>
</tr>
<tr>
<td>q₄</td>
<td>Q₃</td>
</tr>
<tr>
<td>q₃</td>
<td>Q₂</td>
</tr>
<tr>
<td>q₂</td>
<td>Q₁</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>q₁</th>
<th>Q₂</th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>q₁</td>
<td>Q₂</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>q₂</td>
<td>Q₃</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q₃</td>
<td>Q₆</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q₆</td>
<td>Q₇</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q₇</td>
<td>Q₈</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q₈</td>
<td>Q₁₀</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>q₁₀</td>
<td>Q₉</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>q₉</td>
<td>Q₈</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>q₈</td>
<td>Q₇</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>q₇</td>
<td>Q₆</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>q₆</td>
<td>Q₅</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>q₅</td>
<td>Q₄</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>q₄</td>
<td>Q₃</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>q₃</td>
<td>Q₂</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>q₂</td>
<td>Q₁</td>
<td>1</td>
<td>10</td>
</tr>
</tbody>
</table>

\(x := 1\)
\(y := 10\)
\(y > 10\)
\(y := y - 1\)
\(x := 2 * x\)
\(x := y + 1\)
Constant Propagation Analysis: Running Example

\[ x := 1 \]
\[ y \leq 10 \]
\[ y := 10 \]
\[ x := 2 \times y \]
\[ x := y + 1 \]
\[ x < y \]
\[ x \geq y \]
\[ y := y - 1 \]

\begin{tabular}{|c|c|c|}
\hline
State & x & y \\
\hline
q1 & T & T \\
q2 & 1 & T \\
q3 & 1 & T \\
q4 & 1 & 10 \\
q5 & 11 & 10 \\
\hline
\end{tabular}

0 : Initialization
1 : Propagation (→)
Constant Propagation Analysis: Running Example

0 : Initialization
1 : Propagation (→)

<table>
<thead>
<tr>
<th>State</th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>q₁</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>q₂</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q₃</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q₆</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q₁₁</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>q₁₂</td>
<td>11</td>
<td>10</td>
</tr>
</tbody>
</table>

- x := 1
- y := y - 1
- y := 10
- x := 2 * x
- x := y + 1
- x := 1
- y := 10
- x ≥ y
- y > 10
- x < y
Constant Propagation Analysis: Running Example

0 : Initialization
1 : Propagation (→)

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>q₁</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>q₂</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q₃</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q₆</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q₇</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q₁₁</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>q₁₂</td>
<td>11</td>
<td>10</td>
</tr>
</tbody>
</table>
Constant Propagation Analysis: Running Example

0 : Initialization
1 : Propagation (→)

- **q1**: $x := 1$
- **q2**: $y \leq 10$
- **q3**: $y > 10$
- **q6**: $y := y - 1$
- **q11**: $x \geq y$
- **q12**: $x := y + 1$
- **q1**: $x \leq y$
- **q2**: $x := 2 \times x$
- **q3**: $y := 10$
- **q6**: $y := y - 1$
- **q7**: $y := y - 1$
- **q8**: $y := y - 1$

<table>
<thead>
<tr>
<th>State</th>
<th>$x$</th>
<th>$y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_1$</td>
<td>$\top$</td>
<td>$\top$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>1</td>
<td>$\top$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>1</td>
<td>$\top$</td>
</tr>
<tr>
<td>$q_6$</td>
<td>1</td>
<td>$\top$</td>
</tr>
<tr>
<td>$q_7$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$q_8$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$q_{11}$</td>
<td>1</td>
<td>10, $\top$</td>
</tr>
<tr>
<td>$q_{12}$</td>
<td>11</td>
<td>10</td>
</tr>
</tbody>
</table>
Constant Propagation Analysis: Running Example

0: Initialization
1: Propagation (→)

<table>
<thead>
<tr>
<th>q</th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>q₁</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>q₂</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q₃</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q₆</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q₇</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q₈</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q¹₁</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q¹₂</td>
<td>11</td>
<td>10</td>
</tr>
</tbody>
</table>
Constant Propagation Analysis: Running Example

0: Initialization
1: Propagation (→)

<table>
<thead>
<tr>
<th>State</th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>q1</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>q2</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q3</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q6</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q11</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q12</td>
<td>11,2</td>
<td>10, T</td>
</tr>
</tbody>
</table>

x := 1
y ≤ 10
y > 10
y := y - 1
y := 10
x ≥ y
x := 2 * x
x := y + 1

Grégoire Sutre
Software Verification
Data Flow Analysis
VTSA’08 64 / 286
Grégoire Sutre
Software Verification
Data Flow Analysis

Constant Propagation Analysis: Running Example

0 : Initialization
1 : Propagation (→)

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>q1</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>q2</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q3</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q6</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q11</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q12</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>
Constant Propagation Analysis: Running Example

0 : Initialization
1 : Propagation (→)

\begin{array}{|c|c|c|}
\hline
q & x & y \\
\hline
q_1 & T & T \\
q_2 & 1 & T \\
q_3 & 1 & T \\
q_6 & 1 & T \\
q_7 & 1 & T \\
q_11 & 1 & T \\
q_{12} & T & T \\
\hline
\end{array}

- \text{x := 1}
- \text{y \leq 10}
- \text{y > 10}
- \text{y := y - 1}
- \text{x := 2 * x}
- \text{x := y + 1}
- \text{x := 1}
- \text{y := 10}
- \text{x := y}
Constant Propagation Analysis: Running Example

0: Initialization
1: Propagation (→)

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>q1</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>q2</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q3</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q4</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q5</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q6</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q7</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q8</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q9</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q10</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q11</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q12</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>
0 : Initialization
1 : Propagation (→)

<table>
<thead>
<tr>
<th>q</th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>q_1</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>q_2</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q_3</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q_6</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q_7</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q_8</td>
<td>2</td>
<td>T</td>
</tr>
<tr>
<td>q_{11}</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q_{12}</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>
Constant Propagation Analysis: Running Example

0: Initialization
1: Propagation (→)

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>q₁</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>q₂</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q₃</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q₆</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q₇</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q₈</td>
<td>2</td>
<td>T</td>
</tr>
<tr>
<td>q₁₁</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q₁₂</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

Grégoire Sutre
Software Verification
Data Flow Analysis
VTSA'08 64 / 286
Constant Propagation Analysis: Running Example

0 : Initialization
1 : Propagation (→)

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>q₁</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>q₂</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q₃</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q₆</td>
<td>1, 2</td>
<td>T</td>
</tr>
<tr>
<td>q₇</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q₈</td>
<td>2</td>
<td>T</td>
</tr>
<tr>
<td>q₁₁</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q₁₂</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>
Constant Propagation Analysis: Running Example

${\text{0: Initialization}}$

${\text{1: Propagation (→)}}$

<table>
<thead>
<tr>
<th></th>
<th>$x$</th>
<th>$y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_1$</td>
<td>$T$</td>
<td>$T$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>1</td>
<td>$T$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>1</td>
<td>$T$</td>
</tr>
<tr>
<td>$q_6$</td>
<td>$T$</td>
<td>$T$</td>
</tr>
<tr>
<td>$q_7$</td>
<td>1</td>
<td>$T$</td>
</tr>
<tr>
<td>$q_8$</td>
<td>2</td>
<td>$T$</td>
</tr>
<tr>
<td>$q_{11}$</td>
<td>1</td>
<td>$T$</td>
</tr>
<tr>
<td>$q_{12}$</td>
<td>$T$</td>
<td>$T$</td>
</tr>
</tbody>
</table>
Constant Propagation Analysis: Running Example

0 : Initialization
1 : Propagation (→)

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>q₁</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>q₂</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q₃</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q₆</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>q₇</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q₈</td>
<td>2</td>
<td>T</td>
</tr>
<tr>
<td>q₁₁</td>
<td>1, T</td>
<td>T</td>
</tr>
<tr>
<td>q₁₂</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

\(x := 1\)
\(x := 2 \times x\)
\(x := y + 1\)
\(y := y - 1\)
\(x > y\)
\(x \geq y\)
\(y \leq 10\)
\(y > 10\)
Constant Propagation Analysis: Running Example

0: Initialization
1: Propagation (→)

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>q₁</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>q₂</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q₃</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q₆</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>q₇</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q₈</td>
<td>2</td>
<td>T</td>
</tr>
<tr>
<td>q₁₁</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>q₁₂</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

x := 1
y := y - 1
y := 10
x := 2 * x
x := y + 1
x := y
y <= 10
y > 10
x <= y
x < y
x >= y
Constant Propagation Analysis: Running Example

0: Initialization
1: Propagation (→)

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>q₁</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>q₂</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q₃</td>
<td>1</td>
<td>T</td>
</tr>
<tr>
<td>q₆</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>q₇</td>
<td>1, T</td>
<td>T</td>
</tr>
<tr>
<td>q₈</td>
<td>2</td>
<td>T</td>
</tr>
<tr>
<td>q₁₁</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>q₁₂</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>
Constant Propagation Analysis: Running Example

0: Initialization
1: Propagation (→)

\[
\begin{array}{c|c|c}
\text{q} & x & y \\
\hline
q_1 & \top & \top \\
q_2 & 1 & \top \\
q_3 & 1 & \top \\
q_6 & \top & \top \\
q_7 & \top & \top \\
q_8 & 2 & \top \\
q_{11} & \top & \top \\
q_{12} & \top & \top \\
\end{array}
\]
Constant Propagation Analysis: Running Example

0: Initialization
1: Propagation (→)

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>q1</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>q2</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>q3</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>q6</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>q7</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>q8</td>
<td>2, T</td>
<td>T</td>
</tr>
<tr>
<td>q11</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>q12</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>
Constant Propagation Analysis: Running Example

\[ q_1 \]
\[ x := 1 \]
\[ y \leq 10 \]
\[ y > 10 \]
\[ y := y - 1 \]
\[ y := 10 \]
\[ x \geq y \]
\[ x := 2 \times x \]
\[ x := y + 1 \]

\[ q_2 \]
\[ q_3 \]
\[ q_6 \]
\[ q_7 \]
\[ q_8 \]
\[ q_{11} \]
\[ q_{12} \]

0 : Initialization
1 : Propagation (\(\rightarrow\))

<table>
<thead>
<tr>
<th></th>
<th>(x)</th>
<th>(y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(q_1)</td>
<td>(\top)</td>
<td>(\top)</td>
</tr>
<tr>
<td>(q_2)</td>
<td>1</td>
<td>(\top)</td>
</tr>
<tr>
<td>(q_3)</td>
<td>1</td>
<td>(\top)</td>
</tr>
<tr>
<td>(q_6)</td>
<td>(\top)</td>
<td>(\top)</td>
</tr>
<tr>
<td>(q_7)</td>
<td>(\top)</td>
<td>(\top)</td>
</tr>
<tr>
<td>(q_8)</td>
<td>(\top)</td>
<td>(\top)</td>
</tr>
<tr>
<td>(q_{11})</td>
<td>(\top)</td>
<td>(\top)</td>
</tr>
<tr>
<td>(q_{12})</td>
<td>(\top)</td>
<td>(\top)</td>
</tr>
</tbody>
</table>
Extend $\mathbb{R}$ with a new element $\top$ to account for non-constant values

Extend $+$, $-$ and $\times$ such that $\top$ is absorbent

$$\top + r = r + \top = \top \quad \forall r \in \mathbb{R}$$
$$\top - r = r - \top = \top \quad \forall r \in \mathbb{R}$$
$$\top \times r = r \times \top = \top \quad \forall r \in \mathbb{R}$$

Extend $[e]_v$ to valuations from $X$ to $\mathbb{R} \cup \{\top\}$

**Domain of data flow “information”**

$$D = X \rightarrow (\mathbb{R} \cup \{\top\})$$
Constant Propagation Analysis: Formulation

\[ \mathbb{D} = \mathbb{X} \rightarrow (\mathbb{R} \cup \{\top\}) \]

System of equations: variables \( C_q \) for \( q \in Q \), with \( C_q \in \mathbb{D} \)

\[ C_q = \bigotimes_{q' \xrightarrow{\text{op}} q} f_{\text{op}} (C_{q'}) \]

\[ C(q_{\text{in}}) = \lambda x. \top \]

\[ \nu \otimes \nu' = \lambda y. \begin{cases} 
\nu(y) & \text{if } \nu(y) = \nu'(y) \\
\top & \text{otherwise}
\end{cases} \]

Functions \( f_{\text{op}} \)

\[ f_{x:=e}(\nu) = \lambda y. \begin{cases} 
\nu(y) & \text{if } y \neq x \\
[e]_\nu & \text{if } y = x
\end{cases} \]

\[ f_g(\nu) = \nu \]
Constant Propagation Analysis: Formulation

\[ \mathbb{D} = x \rightarrow (\mathbb{R} \cup \{\top\}) \]

System of equations: variables \( C_q \) for \( q \in Q \), with \( C_q \in \mathbb{D} \)

\[ C_q = \bigotimes_{q' \xrightarrow{\text{op}} q} f_{\text{op}}(C_{q'}) \]

\[ C(q_{\text{in}}) = \lambda x. \top \]

\[ v \otimes v' = \lambda y. \begin{cases} v(y) & \text{if } v(y) = v'(y) \\ \top & \text{otherwise} \end{cases} \]

Functions \( f_{\text{op}} \)

\[ f_{x:=e}(v) = \lambda y. \begin{cases} v(y) & \text{if } y \neq x \\ [e]_v & \text{if } y = x \end{cases} \]

\[ f_g(v) = v \]
For each variable $y$ occurring in $e$, if $y$ is constant at location $q_1$ then we may replace $y$ with its constant value in $e$.

This is sound since the analysis is conservative.
Common Form of Data Flow Equations

- Domain $\mathbb{D}$ of data flow “information”
  - sets of variables, sets of expressions, valuations, . . .

- Variables $D_q$ for $q \in Q$, with value in $\mathbb{D}$
  - $D_q$ holds data-flow information for location $q$

$$D_q = \bigwedge f(D_q')$$

- “Confluence” operator $\bigwedge$ on $\mathbb{D}$ to merge data flow information
  - $\cup$, $\cap$, $\otimes$, . . .

- Functions $f : \mathbb{D} \rightarrow \mathbb{D}$ to model the effect of operations
Common Form of Data Flow Equations

- Domain $\mathbb{D}$ of data flow “information”
  - sets of variables, sets of expressions, valuations, . . .

- Variables $D_q$ for $q \in Q$, with value in $\mathbb{D}$
  - $D_q$ holds data-flow information for location $q$

\[
D_q = \bigwedge f(D_{q'})
\]

- “Confluence” operator $\bigwedge$ on $\mathbb{D}$ to merge data flow information
  - $\cup$, $\cap$, $\otimes$, . . .

- Functions $f : \mathbb{D} \to \mathbb{D}$ to model the effect of operations
Outline — Data Flow Analysis

5 Classical Data Flow Analyses

6 Basic Lattice Theory

7 Monotone Data Flow Analysis Frameworks
A partial order on a set $L$ is any binary relation $\sqsubseteq \subseteq L \times L$ satisfying for all $x, y, z \in L$:

\[
\begin{align*}
    x & \sqsubseteq x & \text{(reflexivity)} \\
    x \sqsubseteq y & \land y \sqsubseteq x \implies x = y & \text{(antisymmetry)} \\
    x \sqsubseteq y & \land y \sqsubseteq z \implies x \sqsubseteq z & \text{(transitivity)}
\end{align*}
\]

A partially ordered set is any pair $(L, \sqsubseteq)$ where $L$ is a set and $\sqsubseteq$ is a partial order on $L$.

There can be $x$ and $y$ in $L$ such that $x \not\sqsubseteq y$ and $y \not\sqsubseteq x$. 
A partial order on a set $L$ is any binary relation $\sqsubseteq \subseteq L \times L$ satisfying for all $x, y, z \in L$:

- Reflexivity: $x \sqsubseteq x$
- Antisymmetry: $x \sqsubseteq y \land y \sqsubseteq x \implies x = y$
- Transitivity: $x \sqsubseteq y \land y \sqsubseteq z \implies x \sqsubseteq z$

A partially ordered set is any pair $(L, \sqsubseteq)$ where $L$ is a set and $\sqsubseteq$ is a partial order on $L$.

There can be $x$ and $y$ in $L$ such that $x \nsubseteq y$ and $y \nsubseteq x$. 
Consider a partially ordered set \((L, \sqsubseteq)\) and a subset \(X \subseteq L\).

**Greatest Lower Bound**

A **lower bound** of \(X\) is any \(b \in X\) such that \(b \sqsubseteq x\) for all \(x \in X\).

A **greatest lower bound** of \(X\) is any \(glb \in X\) such that:

1. \(glb\) is a lower bound of \(X\),
2. \(glb \sqsupseteq b\) for any lower bound \(b\) of \(X\).

If \(X\) has a greatest lower bound, then it is *unique* and written \(\sqcap X\).
Consider a partially ordered set \((L, \sqsubseteq)\) and a subset \(X \subseteq L\).

**Greatest Lower Bound**

A lower bound of \(X\) is any \(b \in X\) such that \(b \sqsubseteq x\) for all \(x \in X\).

A greatest lower bound of \(X\) is any \(glb \in X\) such that: […]

If \(X\) has a greatest lower bound, then it is *unique* and written \(\bigcap X\).

**Least Upper Bound**

An upper bound of \(X\) is any \(b \in X\) such that \(b \sqsupseteq x\) for all \(x \in X\).

A least upper bound of \(X\) is any \(lub \in X\) such that:

1. \(lub\) is an upper bound of \(X\),
2. \(lub \sqsubseteq b\) for any upper bound \(b\) of \(X\).

If \(X\) has a least upper bound, then it is *unique* and written \(\bigcup X\).
Lower and Upper Bounds: Examples

$$(\mathbb{R}, \leq)$$

$$\bigcup \{0, \sqrt{2}, 4\} = 4 \quad \bigcap \left\{ \frac{1}{2^n} \mid n \in \mathbb{N} \right\} = 0$$

But $$\{\ldots, -2, -1, 0, 1, 2, \ldots\}$$ has no upper bound and no lower bound.

$$(\mathcal{P}(-1, 0, 1), \subseteq)$$

$$\bigcup \{0\}, \{1\} = \{0, 1\}$$

$$\bigcap \{-1\}, \{0, 1\} = \{-1, 0, 1\}$$

$$\bigcup \{-1, 0\}, \{0, 1\} = \{0\}$$
Lower and Upper Bounds: Examples

$$(\mathbb{R}, \leq)$$

\[
\bigcup \{0, \sqrt{2}, 4\} = 4 \quad \bigcap \left\{ \frac{1}{2^n} \middle| n \in \mathbb{N} \right\} = 0
\]

But \(\ldots, -2, -1, 0, 1, 2, \ldots\) has no upper bound and no lower bound.

\[\left(\mathcal{P}(\{-1, 0, 1\}), \subseteq\right)\]

\[
\bigcup \{\{0\}, \{1\}\} = \{0, 1\} \\
\bigcup \{-1\}, \{0, 1\}\} = \{-1, 0, 1\} \\
\bigcap \{-1, 0\}, \{0, 1\}\} = \{0\}
\]
A lattice is any partially ordered set \((L, \sqsubseteq)\) where every finite subset \(X \subseteq L\) has a greatest lower bound and a least upper bound.

A complete lattice is any partially ordered set \((L, \sqsubseteq)\) where every subset \(X \subseteq L\) has a greatest lower bound and a least upper bound.

The least element \(\bot\) and greatest element \(\top\) are defined by:

\[
\bot = \bigcap L = \bigcup \emptyset \quad \text{and} \quad \top = \bigcup L = \bigcap \emptyset
\]

Example

\((\mathbb{R}, \leq)\) is a lattice, but it is not a complete lattice.
Definition

A lattice is any partially ordered set \((L, \sqsubseteq)\) where every finite subset \(X \subseteq L\) has a greatest lower bound and a least upper bound.

Definition

A complete lattice is any partially ordered set \((L, \sqsubseteq)\) where every subset \(X \subseteq L\) has a greatest lower bound and a least upper bound.

The least element \(\bot\) and greatest element \(\top\) are defined by:

\[
\bot = \bigcap L = \bigcup \emptyset \quad \text{and} \quad \top = \bigcup L = \bigcap \emptyset
\]

Example

\((\mathbb{R}, \leq)\) is a lattice, but it is not a complete lattice.
A lattice is any partially ordered set \((L, \sqsubseteq)\) where every finite subset \(X \subseteq L\) has a greatest lower bound and a least upper bound.

A complete lattice is any partially ordered set \((L, \sqsubseteq)\) where every subset \(X \subseteq L\) has a greatest lower bound and a least upper bound.

The least element \(\bot\) and greatest element \(\top\) are defined by:

\[
\bot = \bigcap L = \bigcup \emptyset \quad \text{and} \quad \top = \bigcup L = \bigcap \emptyset
\]

Example

\((\mathbb{R}, \leq)\) is a lattice, but it is not a complete lattice.
Fixpoints

Let $f : L \rightarrow L$ be a function on a partially ordered set $(L, \sqsubseteq)$.

**Definition**

A **fixpoint** of $f$ is any $x \in L$ such that $f(x) = x$.

**Definition**

A **least fixpoint** of $f$ is any $lfp \in X$ such that:

1. $lfp$ is a fixpoint of $f$,
2. $lfp \sqsubseteq x$ for any fixpoint $x$ of $f$.

If $f$ has a least fixpoint, then it is *unique* and written $\text{lfp}(f)$.

**Definition**

A **greatest fixpoint** of $f$ is any $gfp \in X$ such that:

1. $gfp$ is a fixpoint of $f$,
2. $gfp \sqsupseteq x$ for any fixpoint $x$ of $f$. 

Fixpoints

Let $f : L \rightarrow L$ be a function on a partially ordered set $(L, \sqsubseteq)$.

**Definition**

A fixpoint of $f$ is any $x \in L$ such that $f(x) = x$.

**Definition**

A least fixpoint of $f$ is any $\text{lfp} \in X$ such that:

1. $\text{lfp}$ is a fixpoint of $f$,
2. $\text{lfp} \sqsubseteq x$ for any fixpoint $x$ of $f$.

If $f$ has a least fixpoint, then it is unique and written $\text{lfp}(f)$.

**Definition**

A greatest fixpoint of $f$ is any $\text{gfp} \in X$ such that:

1. $\text{gfp}$ is a fixpoint of $f$,
2. $\text{gfp} \sqsubseteq x$ for any fixpoint $x$ of $f$. 
Fixpoints

Let $f : L \rightarrow L$ be a function on a partially ordered set $(L, \sqsubseteq)$.

**Definition**

A **fixpoint** of $f$ is any $x \in L$ such that $f(x) = x$.

**Definition**

A **least fixpoint** of $f$ is any $lfp \in X$ such that: [...] If $f$ has a least fixpoint, then it is *unique* and written $lfp(f)$.

**Definition**

A **greatest fixpoint** of $f$ is any $gfp \in X$ such that:

1. $gfp$ is a fixpoint of $f$,
2. $gfp \sqsupseteq x$ for any fixpoint $x$ of $f$.

If $f$ has a greatest fixpoint, then it is *unique* and written $gfp(f)$. 
Knaster-Tarski Fixpoint Theorem

A function $f : L \to L$ on a partially ordered set $(L, \sqsubseteq)$ is monotonic if for all $x, y \in L$:

$$x \sqsubseteq y \implies f(x) \sqsubseteq f(y)$$

**Theorem**

*Every monotonic function $f$ on a complete lattice $(L, \sqsubseteq)$ has a least fixpoint $\text{lfp}(f)$ and a greatest fixpoint $\text{gfp}(f)$. Moreover:*

$$\text{lfp}(f) = \bigcap \{ x \in L \mid f(x) \sqsubseteq x \}$$

$$\text{gfp}(f) = \bigcup \{ x \in L \mid f(x) \sqsupseteq x \}$$
Order Duality

If \((L, \sqsubseteq)\) is a partially ordered set then so is \((L, \sqsupseteq)\).

If \((L, \sqsubseteq)\) is a complete lattice then so is \((L, \sqsupseteq)\).

\[
\bigwedge_{(L, \sqsubseteq)} = \bigvee_{(L, \sqsubseteq)} \quad \perp_{(L, \sqsubseteq)} = \top_{(L, \sqsubseteq)}
\]

\[
\bigvee_{(L, \sqsubseteq)} = \bigwedge_{(L, \sqsubseteq)} \quad \top_{(L, \sqsubseteq)} = \perp_{(L, \sqsubseteq)}
\]

For any monotonic function \(f : L \rightarrow L\) on a complete lattice \((L, \sqsubseteq)\),

\[
\text{lfp}_{(L, \sqsubseteq)}(f) = \text{gfp}_{(L, \sqsubseteq)}(f)
\]

\[
\text{gfp}_{(L, \sqsubseteq)}(f) = \text{lfp}_{(L, \sqsubseteq)}(f)
\]

We shall focus on least fixpoints.
Order Duality

If \((L, \sqsubseteq)\) is a partially ordered set then so is \((L, \sqsupseteq)\).

If \((L, \sqsubseteq)\) is a complete lattice then so is \((L, \sqsupseteq)\).

\[
\begin{align*}
\bigwedge_{(L, \sqsubseteq)} &= \bigvee_{(L, \sqsubseteq)} \\
\bigvee_{(L, \sqsubseteq)} &= \bigwedge_{(L, \sqsubseteq)} \\
\bot_{(L, \sqsubseteq)} &= \top_{(L, \sqsupseteq)} \\
\top_{(L, \sqsubseteq)} &= \bot_{(L, \sqsupseteq)}
\end{align*}
\]

For any monotonic function \(f : L \rightarrow L\) on a complete lattice \((L, \sqsubseteq)\),

\[
\begin{align*}
\text{lfp}_{(L, \sqsubseteq)}(f) &= \text{gfp}_{(L, \sqsupseteq)}(f) \\
\text{gfp}_{(L, \sqsubseteq)}(f) &= \text{lfp}_{(L, \sqsupseteq)}(f)
\end{align*}
\]

We shall focus on least fixpoints.
An ascending chain in a partially ordered set \((L, \sqsubseteq)\) is any infinite sequence \(x_0, x_1, \ldots\) of elements of \(L\) satisfying \(x_i \sqsubseteq x_{i+1}\) for all \(i \in \mathbb{N}\).

A partially ordered set \((L, \sqsubseteq)\) satisfies the ascending chain condition if every ascending chain \(x_0 \sqsubseteq x_1 \sqsubseteq \cdots\) of elements of \(L\) is eventually stationary.

**Examples**

\((\mathbb{R}, \leq)\) does not satisfy the ascending chain condition.

\((\mathbb{N}, \geq)\) satisfies the ascending chain condition.
An ascending chain in a partially ordered set \((L, \sqsubseteq)\) is any infinite sequence \(x_0, x_1, \ldots\) of elements of \(L\) satisfying \(x_i \sqsubseteq x_{i+1}\) for all \(i \in \mathbb{N}\).

A partially ordered set \((L, \sqsubseteq)\) satisfies the ascending chain condition if every ascending chain \(x_0 \sqsubseteq x_1 \sqsubseteq \cdots\) of elements of \(L\) is eventually stationary.

**Examples**

\((\mathbb{R}, \leq)\) does not satisfy the ascending chain condition.

\((\mathbb{N}, \geq)\) satisfies the ascending chain condition.
Kleene Iteration

Consider a partially ordered set \((L, \sqsubseteq)\) and \(f : L \to L\) monotonic.

The **Kleene iteration** \((f^i(\bot))_{i \in \mathbb{N}}\) is an ascending chain:

\[
\bot \sqsubseteq f(\bot) \sqsubseteq \cdots \sqsubseteq f^i(\bot) \sqsubseteq f^{i+1}(\bot) \sqsubseteq \cdots
\]

For every \(k \in \mathbb{N}\), if \(f^k(\bot) = f^{k+1}(\bot)\) then \(f^k(\bot)\) is the least fixpoint of \(f\).

**Correction and termination**

1. For every monotonic \(f\), if \(\text{LFP}(f)\) terminates then it returns \(\text{lfp}(f)\).

2. If \(L\) satisfies the ascending chain condition then \(\text{LFP}(f)\) always terminates (on monotonic \(f\)).
Kleene Iteration

Consider a partially ordered set \((L, \sqsubseteq)\) and \(f : L \to L\) monotonic.

The **Kleene iteration** \((f^i(\bot))_{i \in \mathbb{N}}\) is an ascending chain:

\[
\bot \sqsubseteq f(\bot) \sqsubseteq \cdots \sqsubseteq f^i(\bot) \sqsubseteq f^{i+1}(\bot) \sqsubseteq \cdots
\]

For every \(k \in \mathbb{N}\), if \(f^k(\bot) = f^{k+1}(\bot)\) then \(f^k(\bot)\) is the least fixpoint of \(f\).

**Correction and termination**

1. For every monotonic \(f\), if \(\text{LFP}(f)\) terminates then it returns \(\text{lfp}(f)\).

2. If \(L\) satisfies the ascending chain condition then \(\text{LFP}(f)\) always terminates (on monotonic \(f\)).
For any set \( S \), the pair \((P(S), \subseteq)\) is a complete lattice, where \( \subseteq = \subseteq \).

\( \sqcap \), \( \sqcup \), \( \bot \) and \( \top \) satisfy:

\[
\begin{align*}
\sqcap &= \cap \\
\sqcup &= \cup \\
\bot &= \emptyset \\
\top &= S
\end{align*}
\]

If \( S \) is finite then \((P(S), \subseteq)\) satisfies the ascending chain condition.
For any set $S$ and complete lattice $(L, \sqsubseteq)$, the pair $(S \to L, \sqsubseteq)$ is a complete lattice, where $\sqsubseteq$ is defined by:

$$f \sqsubseteq g \text{ if } f(x) \sqsubseteq g(x) \text{ for all } x \in S$$

$\sqcap$, $\sqcup$, $\bot$ and $\top$ satisfy:

$$\sqcap X = \lambda x . \sqcap \{ f(x) | f \in X \} \quad \bot = \lambda x . \bot$$

$$\sqcup X = \lambda x . \sqcup \{ f(x) | f \in X \} \quad \top = \lambda x . \top$$

If $S$ is finite and $(L, \sqsubseteq)$ satisfies the ascending chain condition then $(S \to L, \sqsubseteq)$ satisfies the ascending chain condition.
Outline — Data Flow Analysis

5 Classical Data Flow Analyses

6 Basic Lattice Theory

7 Monotone Data Flow Analysis Frameworks
Common Form of Data Flow Equations (Recall)

- Domain \( \mathbb{D} \) of data flow “information”
  - sets of variables, sets of expressions, valuations, . . .

- Variables \( D_q \) for \( q \in Q \), with value in \( \mathbb{D} \)
  - \( D_q \) holds data-flow information for location \( q \)

\[
D_q = \bigwedge f(D_{q'})
\]

- “Confluence” operator \( \bigwedge \) on \( \mathbb{D} \) to merge data flow information
  - \( \cup, \cap, \otimes, \ldots \)

- Functions \( f : \mathbb{D} \rightarrow \mathbb{D} \) to model the effect of operations
Monotone Framework

- Complete lattice \((L, \sqsubseteq)\) of data flow facts
- Set \(F\) of monotonic transfer functions \(f : L \rightarrow L\)

Partial order \(\sqsubseteq\) compares the precision of data flow facts:

- \(\phi \sqsubseteq \psi\) means that \(\phi\) is more precise than \(\psi\).
- \(\sqcap X\) is the most precise fact consistent with all facts \(\phi \in X\).

Conservative Approximation

- \(\phi \sqsubseteq \psi\) means that \(\psi\) soundly approximates \(\phi\).
- If \(\phi \sqsubseteq \psi\) then it is sound, but less precise, to replace \(\phi\) by \(\psi\).
Monotone Framework

- Complete lattice \((L, \sqsubseteq)\) of data flow facts
- Set \(\mathcal{F}\) of monotonic transfer functions \(f : L \rightarrow L\)

Partial order \(\sqsubseteq\) compares the precision of data flow facts:

- \(\phi \sqsubseteq \psi\) means that \(\phi\) is more precise than \(\psi\).
- \(\bigvee X\) is the most precise fact consistent with all facts \(\phi \in X\).

Conservative Approximation

- \(\phi \sqsubseteq \psi\) means that \(\psi\) soundly approximates \(\phi\).
- If \(\phi \sqsubseteq \psi\) then it is sound, but less precise, to replace \(\phi\) by \(\psi\).
Monotone Framework

- Complete lattice $(L, \sqsubseteq)$ of data flow facts
- Set $\mathcal{F}$ of monotonic transfer functions $f : L \to L$

Partial order $\sqsubseteq$ compares the precision of data flow facts:

- $\phi \sqsubseteq \psi$ means that $\phi$ is more precise than $\psi$.
- $\bigsqcup X$ is the most precise fact consistent with all facts $\phi \in X$.

Conservative Approximation

$\phi \sqsubseteq \psi$ means that $\psi$ soundly approximates $\phi$.

If $\phi \sqsubseteq \psi$ then it is sound, but less precise, to replace $\phi$ by $\psi$. 
Semantic Definition of Liveness

A variable $x$ is **live** at location $q$ if there exists an execution path starting from $q$ where $x$ is used before it is modified.

Consider a control flow automaton with variables $X = \{x, y, z\}$.

Complete lattice $(L, \sqsubseteq)$ of data flow facts: $(\mathcal{P}(X), \subseteq)$

The fact $\{x, z\}$ means: _the variables that are live are among $\{x, z\}$_.

i.e. _the variable $y$ is not live._

The fact $\{x\}$ is **more precise** than $\{x, z\}$, but incomparable with $\{y\}$.

The fact $\{x, z\}$ **soundly approximates** the fact $\{x\}$.
Data Flow Instances

Data Flow Instance

- Monotone framework \( \langle (L, \sqsubseteq), \mathcal{F} \rangle \)
- Control flow automaton \( \langle Q, q_{in}, q_{out}, X, \rightarrow \rangle \)
- Transfer mapping \( f : \text{Op} \rightarrow \mathcal{F} \)
- Initial data flow value \( i \in L \)

Notation for transfer mapping: \( f_{\text{op}} \) instead of \( f(\text{op}) \)

Two possible directions for data flow analysis: forward and backward

Transfer functions \( f_{\text{op}} \) must be defined in accordance with the direction of the analysis.
Data Flow Instances

Data Flow Instance

- Monotone framework $\langle (L, \sqsubseteq), \mathcal{F} \rangle$
- Control flow automaton $\langle Q, q_{in}, q_{out}, X, \rightarrow \rangle$
- Transfer mapping $f : \mathcal{Op} \rightarrow \mathcal{F}$
- Initial data flow value $i \in L$

Notation for transfer mapping: $f_{\text{op}}$ instead of $f(\text{op})$

Two possible directions for data flow analysis: forward and backward

Transfer functions $f_{\text{op}}$ must be defined in accordance with the direction of the analysis.
Data Flow Equations

Consider a data flow instance \( \langle (L, \sqsubseteq), F, Q, q_{\text{in}}, q_{\text{out}}, X, \rightarrow, f, \iota \rangle \).

System of equations: variables \( A_q \) for \( q \in Q \), with \( A_q \in L \)

### Forward Analysis

\[
A_q = l_q \sqcup \bigcup_{q' \xrightarrow{\text{op}} q} f_{\text{op}}(A_{q'}) \quad l_q = \begin{cases} \iota & \text{if } q = q_{\text{in}} \\ \bot & \text{otherwise} \end{cases}
\]

### Backward Analysis

\[
A_q = l_q \sqcup \bigcup_{q' \xleftarrow{\text{op}} q} f_{\text{op}}(A_{q'}) \quad l_q = \begin{cases} \iota & \text{if } q = q_{\text{out}} \\ \bot & \text{otherwise} \end{cases}
\]
Data Flow Equations

Consider a data flow instance \( \langle (L, \sqsubseteq), \mathcal{F}, Q, q_{\text{in}}, q_{\text{out}}, X, \rightarrow, f, \iota \rangle \).

System of equations: variables \( A_q \) for \( q \in Q \), with \( A_q \in L \)

**Forward Analysis**

\[
A_q = I_q \sqcup \bigsqcup_{q' \xrightarrow{\text{op}} q} f_{\text{op}}(A_{q'}) \\
I_q = \begin{cases} 
\iota & \text{if } q = q_{\text{in}} \\
\bot & \text{otherwise}
\end{cases}
\]

**Backward Analysis**

\[
A_q = I_q \sqcup \bigsqcup_{q \xrightarrow{\text{op}} q'} f_{\text{op}}(A_{q'}) \\
I_q = \begin{cases} 
\iota & \text{if } q = q_{\text{out}} \\
\bot & \text{otherwise}
\end{cases}
\]
Minimal Fixpoint (MFP) Solution

The system of data flow equations may have several solutions...

We are interested in the “least solution” to the data flow equations.

Complete lattice \((L, \sqsubseteq)\) extended to \((Q \rightarrow L, \sqsubseteq)\)

The forward minimal fixpoint solution \(\mathbf{MFP}\) of the data flow instance is the least fixpoint of the monotonic function \(\overrightarrow{\Delta}\) on \((Q \rightarrow L)\):

\[
\overrightarrow{\Delta}(a) = \lambda q. \begin{cases}
\text{i} & \text{if } q = q_{in} \\
q' \xrightarrow{\text{op}} q & \text{otherwise}
\end{cases}
\]

\[

\overrightarrow{\Delta}(a) = \lambda q. \begin{cases}
\text{i} & \text{if } q = q_{in} \\
q' \xrightarrow{\text{op}} q & \text{otherwise}
\end{cases}
\]

\[

\overrightarrow{\Delta}(a) = \lambda q. \begin{cases}
\text{i} & \text{if } q = q_{in} \\
q' \xrightarrow{\text{op}} q & \text{otherwise}
\end{cases}
\]

\[

\overrightarrow{\Delta}(a) = \lambda q. \begin{cases}
\text{i} & \text{if } q = q_{in} \\
q' \xrightarrow{\text{op}} q & \text{otherwise}
\end{cases}
\]

Grégoire Sutre
Software Verification
Data Flow Analysis
VTSA’08
Minimal Fixpoint (MFP) Solution

The system of data flow equations may have several solutions. . . 

We are interested in the “least solution” to the data flow equations.

Complete lattice \((L, \sqsubseteq)\) extended to \((Q \rightarrow L, \sqsubseteq)\)

The **forward minimal fixpoint solution** \(\text{MFP}\) of the data flow instance is the **least fixpoint** of the monotonic function \(\Delta\) on \((Q \rightarrow L)\):

\[
\Delta(a) = \lambda q. \begin{cases} 
\bot & \text{if } q = q_{\text{in}} \\

f_{\text{op}}(a(q')) & \text{otherwise} 
\end{cases}
\]
Minimal Fixpoint (MFP) Solution

The system of data flow equations may have several solutions. . .

We are interested in the “least solution” to the data flow equations.

Complete lattice \((L, \sqsubseteq)\) extended to \((Q \rightarrow L, \sqsubseteq)\)

The **backward minimal fixpoint solution** \(\Delta\) of the data flow instance is the **least fixpoint** of the monotonic function \(\Delta\) on \((Q \rightarrow L)\):

\[
\Delta(a) = \lambda q. \begin{cases}
\bot & \text{if } q = q_{out} \\
\bigcup \limits_{q \xrightarrow{\text{op}} q'} f_{\text{op}}(a(q')) & \text{otherwise}
\end{cases}
\]
Consider a data flow instance \( \langle (L, \sqsubseteq), \mathcal{F}, Q, q_{\text{in}}, q_{\text{out}}, X, \rightarrow, f, i \rangle \).

Constraint system: variables \( A_q \) for \( q \in Q \), with \( A_q \in L \)

Forward Analysis

\[
\overrightarrow{(CS)} \quad \begin{cases} 
A_{q_{\text{in}}} \sqsubseteq i \\
A_{q'} \sqsubseteq f_{\text{op}}(A_q) \quad \text{for each } q \xrightarrow{\text{op}} q'
\end{cases}
\]

By Knaster-Tarski Fixpoint Theorem,

\[
\overrightarrow{\text{MFP}} = \prod \left\{ a \in Q \rightarrow L \mid a \models (CS) \right\}
\]

Any solution to \( (CS) \) is a sound approximation of \( \overrightarrow{\text{MFP}} \).
Constraint-Based Formulation

Consider a data flow instance \( \langle (L, \sqsubseteq), \mathcal{F}, Q, q_{\text{in}}, q_{\text{out}}, X, \rightarrow, f, \iota \rangle \).

Constraint system: variables \( A_q \) for \( q \in Q \), with \( A_q \in L \)

**Backward Analysis**

\[
\begin{align*}
\overset{←}{(CS)} & \quad \begin{cases}
A_{q_{\text{out}}} \sqsupseteq \iota \\
A_{q'} \sqsupseteq f_{\text{op}}(A_q) \quad \text{for each } q' \xrightarrow{\text{op}} q
\end{cases}
\end{align*}
\]

By Knaster-Tarski Fixpoint Theorem,

\[
\overset{←}{\text{MFP}} = \bigsqcap \{ a \in Q \rightarrow L \mid a \models (CS) \}
\]

Any solution to \( (CS) \) is a sound approximation of \( \overset{←}{\text{MFP}} \).
Live Variables Analysis (Revisited)

Control Flow Automaton: $\langle Q, q_{in}, q_{out}, X, \to \rangle$

**Monotone Framework**
- Complete lattice $(L, \sqsubseteq)$ of data flow facts: $(\mathcal{P}(X), \subseteq)$
- Set $\mathcal{F}$ of monotonic transfer functions:

$$\mathcal{F} = \{ \lambda \phi \cdot \text{gen} \cup (\phi \setminus \text{kill}) \mid \text{gen}, \text{kill} \in L \}$$

**Data Flow Instance**
- Initial data flow value: $\emptyset$
- Transfer mapping: $f_{op}(\phi) = \text{Gen}_{op} \cup (\phi \setminus \text{Kill}_{op})$

Backward analysis
Available Expressions Analysis (Revisited)

Control Flow Automaton: \( \langle Q, q_{in}, q_{out}, X, \rightarrow \rangle \)

### Monotone Framework

- Complete lattice \((L, \sqsubseteq)\) of data flow facts: \((\mathcal{P}(SubExp(\rightarrow)), \supseteq)\)
- Set \(F\) of monotonic transfer functions:

\[
F = \{ \lambda \phi. gen \cup (\phi \setminus kill) \mid gen, kill \in L \}
\]

### Data Flow Instance

- Initial data flow value: \(\emptyset\)
- Transfer mapping: \(f_{op}(\phi) = Gen_{op} \cup (\phi \setminus Kill_{op})\)

Forward analysis
Constant Propagation Analysis (Revisited)

Control Flow Automaton: \( \langle Q, q_{in}, q_{out}, X, \rightarrow \rangle \)

Constant Propagation Lattice for a Single Variable

\[
\begin{array}{cccc}
\top & -2 & -1 & 0 & 1 & 2 & \cdots \\
\downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \\
\bot & -2 & -1 & 0 & 1 & 2 & \cdots \\
\end{array}
\]

\[
(R \cup \{\bot, T\}, \sqsubseteq)
\]

<table>
<thead>
<tr>
<th>(\phi)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>(T)</td>
<td>(R)</td>
</tr>
<tr>
<td>(r \in R)</td>
<td>{r}</td>
</tr>
<tr>
<td>(\bot)</td>
<td>(\emptyset)</td>
</tr>
</tbody>
</table>

Monotone Framework

- Complete lattice \((L, \sqsubseteq)\) of data flow facts: \((X \rightarrow (R \cup \{\bot, T\}), \sqsubseteq)\)
- Set \(\mathcal{F}\) defined as the set of all monotonic transfer functions on \(L\).
Constant Propagation Analysis (Revisited)

Control Flow Automaton: \( \langle Q, q_{in}, q_{out}, X, \rightarrow \rangle \)

**Monotone Framework**
- Complete lattice \( (L, \sqsubseteq) \) of data flow facts: \( (X \rightarrow (\mathbb{R} \cup \{\bot, \top\}), \sqsubseteq) \)
- Set \( \mathcal{F} \) defined as the set of all monotonic transfer functions on \( L \).

**Data Flow Instance**
- Initial data flow value: \( \top \)
- Transfer mapping:

\[
f_x := e(\phi) = \lambda y. \begin{cases} 
\phi(y) & \text{if } y \neq x \\
[e]_\phi & \text{if } y = x 
\end{cases}
\]

\( f_g(\phi) = \phi \)

Forward analysis
Constant Propagation Analysis (Revisited)

Extension of $\llbracket e \rrbracket$ to valuations in $x \rightarrow (\mathbb{R} \cup \{ \bot, T \})$

For $r \in \mathbb{R} \cup \{ T \}$

- $T + r = r + T = T$
- $T - r = r - T = T$
- $T \times r = r \times T = T$

For $r \in \mathbb{R} \cup \{ \bot, T \}$

- $\bot + r = r + \bot = \bot$
- $\bot - r = r - \bot = \bot$
- $\bot \times r = r \times \bot = \bot$

Expressions:

- $\llbracket c \rrbracket_v = c$ [for $c \in \mathbb{Q}$]
- $\llbracket x \rrbracket_v = v(x)$ [for $x \in X$]
- $\llbracket e_1 + e_2 \rrbracket_v = \llbracket e_1 \rrbracket_v + \llbracket e_2 \rrbracket_v$
- $\llbracket e_1 - e_2 \rrbracket_v = \llbracket e_1 \rrbracket_v - \llbracket e_2 \rrbracket_v$
- $\llbracket e_1 \times e_2 \rrbracket_v = \llbracket e_1 \rrbracket_v \times \llbracket e_2 \rrbracket_v$
Extension of $[e]$ to valuations in $x \rightarrow (\mathbb{R} \cup \{\bot, T\})$

For $r \in \mathbb{R} \cup \{T\}$
\[
\begin{align*}
T + r &= r + T = T \\
T - r &= r - T = T \\
T \times r &= r \times T = T
\end{align*}
\]

For $r \in \mathbb{R} \cup \{\bot, T\}$
\[
\begin{align*}
\bot + r &= r + \bot = \bot \\
\bot - r &= r - \bot = \bot \\
\bot \times r &= r \times \bot = \bot
\end{align*}
\]

Expressions: $[e]_v$
\[
\begin{align*}
[c]_v &= c \quad [c \in Q] \\
[x]_v &= v(x) \quad [x \in X] \\
[e_1 + e_2]_v &= [e_1]_v + [e_2]_v \\
[e_1 - e_2]_v &= [e_1]_v - [e_2]_v \\
[e_1 \times e_2]_v &= [e_1]_v \times [e_2]_v
\end{align*}
\]
(Forward) MFP Computation by Kleene Iteration

Consider a data flow instance \( \langle (L, \sqsubseteq), \mathcal{F}, Q, q_{in}, q_{out}, X, \rightarrow, f, \iota \rangle \).

```plaintext
a \leftarrow \lambda q . \perp
repeat
  b \leftarrow a
  a \leftarrow \Delta(a)
until b = a
return a
```

Correction and termination
1. Returns \( \overrightarrow{\text{MFP}} \) when it terminates
2. Always terminates when \((L, \sqsubseteq)\) satisfies the ascending chain condition
(Forward) MFP Computation by Kleene Iteration

Consider a data flow instance \( \langle (L, \sqsubseteq), \mathcal{F}, Q, q_{in}, q_{out}, X, \rightarrow, f, \iota \rangle \).

\[
\begin{align*}
&\text{a} \leftarrow \lambda q. \bot \\
&\text{repeat} \\
&\quad b \leftarrow a \\
&\quad a \leftarrow \Delta(a) \\
&\text{until} \ b = a \\
&\text{return} \ a
\end{align*}
\]

**Correction and termination**

1. Returns MFP when it terminates
2. Always terminates when \((L, \sqsubseteq)\) satisfies the ascending chain condition

\[
\begin{align*}
&\text{foreach } q \in Q \\
&\quad a[q] \leftarrow \bot \\
&\quad a[q_{in}] \leftarrow \iota \\
&\quad a[q_{in}] \leftarrow \iota \\
&\text{repeat} \\
&\quad \text{foreach } q \in Q \\
&\quad \quad b[q] \leftarrow a[q] \\
&\quad \text{foreach } q \in Q \\
&\quad \quad a[q] \leftarrow \bigsqcup_{q' \rightarrow q} f_{op}(b[q']) \\
&\quad \text{until} \ (\forall q \in Q \cdot b[q] = a[q]) \\
&\text{return} \ a
\end{align*}
\]
Consider a data flow instance \(< (L, \sqsubseteq), \mathcal{F}, Q, q_{in}, q_{out}, X, \rightarrow, f, \iota)\>.

\[
\begin{align*}
a & \leftarrow \lambda q. \bot \\
\text{repeat} & \\
& \quad b \leftarrow a \\
& \quad a \leftarrow \Delta(a) \\
\text{until} & \quad b = a \\
\text{return} & \quad a
\end{align*}
\]

Correction and termination

1. Returns \(\xrightarrow{\text{MFP}}\) when it terminates
2. Always terminates when \((L, \sqsubseteq)\) satisfies the ascending chain condition

\[
\begin{align*}
\text{foreach } q \in Q \\
a[q] & \leftarrow \bot \\
a[q_{in}] & \leftarrow \iota \\
\text{repeat} & \\
& \quad \text{foreach } q \in Q \\
& \quad b[q] \leftarrow a[q] \\
& \quad \text{foreach } q \in Q \\
& \quad a[q] \leftarrow \bigcup_{q' \xrightarrow{\text{op}} q} f_{\text{op}}(b[q']) \\
\text{until} & \quad (\forall q \in Q \cdot b[q] = a[q]) \\
\text{return} & \quad a
\end{align*}
\]
Consider a data flow instance \( \langle (L, \sqsubseteq), \mathcal{F}, Q, q_{in}, q_{out}, X, \rightarrow, f, i \rangle \).

The foreach loop iterates over transitions in \( \rightarrow \).

Propagation of facts
- benefits from previous propagations
- records whether there was a change

Correct and always faster than Kleene iteration
Consider a data flow instance \( \langle (L, \sqsubseteq), \mathcal{F}, Q, q_{in}, q_{out}, X, \rightarrow, f, \iota \rangle \).

The foreach loop iterates over transitions in \( \rightarrow \).

Propagation of facts
- benefits from previous propagations
- records whether there was a change

Correct and always faster than Kleene iteration
(Forward) MFP Computation by Worklist Iteration

wl ← nil

foreach $q' \xrightarrow{op} q$
    wl ← cons((q, op, q'), w)

foreach $q \in Q$
    $a[q] \leftarrow \bot$
    $a[q_{in}] \leftarrow i$

while $wl \neq nil$
    $(q, op, q') \leftarrow \text{head}(w)$
    wl ← tail(w)

    new ← $f_{op}(a[q])$
    if new $\not\subseteq a[q']$
        $a[q'] \leftarrow a[q] \cup new$

    foreach $q' \xrightarrow{op'} q''$
        wl ← cons((q', op', q''), w)

return a
(Forward) MFP Computation by Worklist Iteration

wl ← nil

foreach q′ op → q
    wl ← cons((q, op, q′), wl)

foreach q ∈ Q
    a[q] ← ⊥
    a[q_{in}] ← ı

while wl ≠ nil
    (q, op, q′) ← head(wl)
    wl ← tail(wl)
    new ← f_{op}(a[q])
    if new ⊈ a[q′]
        a[q′] ← a[q] ⊔ new

    foreach q′ op′ → q''
        wl ← cons((q′, op′, q''), wl)

return a

Vs Round-Robin

😊 Less computations

😢 Overhead

Worklist structures

- LIFO
- FIFO
- Set
- ...
Optimization of MFP Computation with SCCs

1. Decompose control flow automaton into strongly connected components

2. Transitions between SCCs induce a partial order between SCCs

3. Compute the MFP solution component after component, following the partial order between SCCs

This optimization often pays off in practice

Further optimizations are possible…
Optimization of MFP Computation with SCCs

1. Decompose control flow automaton into strongly connected components

2. Transitions between SCCs induce a partial order between SCCs

3. Compute the MFP solution component after component, following the partial order between SCCs

This optimization often pays off in practice

Further optimizations are possible...
Loss of Precision with the MFP Solution

At $q_5$, we have $z = 3$
Loss of Precision with the MFP Solution

At $q_5$, we have $z = 3$

Loss of Precision

Cause: application of $\sqcup$ at $q_4$ to merge data flow information
Loss of Precision with the MFP Solution

\[ x := 1 \]
\[ y := 2 \]
\[ z := x + y \]
\[ x := 2 \]
\[ y := 1 \]

At \( q_5 \), we have \( z = 3 \)

<table>
<thead>
<tr>
<th></th>
<th>( x )</th>
<th>( y )</th>
<th>( z )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_1 )</td>
<td>( \top )</td>
<td>( \top )</td>
<td>( \top )</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>( 1 )</td>
<td>( \top )</td>
<td>( \top )</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>( \bot )</td>
<td>( \bot )</td>
<td>( \bot )</td>
</tr>
<tr>
<td>( q_4 )</td>
<td>( \bot )</td>
<td>( \bot )</td>
<td>( \bot )</td>
</tr>
<tr>
<td>( q_5 )</td>
<td>( \bot )</td>
<td>( \bot )</td>
<td>( \bot )</td>
</tr>
</tbody>
</table>

Loss of Precision

Cause: application of \( \sqcap \) at \( q_4 \) to merge data flow information
Loss of Precision with the MFP Solution

\[
x := 1 \quad x := 2
\]

\[
y := 2 \quad y := 1
\]

\[
z := x + y
\]

\[\begin{array}{ccc}
q_1 & x & y & z \\
q_2 & 1 & T & T \\
q_3 & \bot & \bot & \bot \\
q_4 & \bot & \bot & \bot \\
q_5 & \bot & \bot & \bot \\
\end{array}\]

At \(q_5\), we have \(z = 3\).

Loss of Precision

Cause: application of \(\sqcap\) at \(q_4\) to merge data flow information
Loss of Precision with the MFP Solution

At $q_5$, we have $z = 3$

Loss of Precision
Cause: application of $\sqcap$ at $q_4$ to merge data flow information
Loss of Precision with the MFP Solution

At $q_5$, we have $z = 3$

Loss of Precision

Cause: application of $\sqcup$ at $q_4$ to merge data flow information
### Loss of Precision with the MFP Solution

At $q_5$, we have $z = 3$

```latex
\begin{align*}
x &:= 1 \\
x &:= 2 \\
y &:= 2 \\
y &:= 1 \\
z &:= x+y
\end{align*}
```

<table>
<thead>
<tr>
<th></th>
<th>$x$</th>
<th>$y$</th>
<th>$z$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_1$</td>
<td>$\top$</td>
<td>$\top$</td>
<td>$\top$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>1</td>
<td>$\top$</td>
<td>$\top$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>2</td>
<td>$\top$</td>
<td>$\top$</td>
</tr>
<tr>
<td>$q_4$</td>
<td>1</td>
<td>2</td>
<td>$\top$</td>
</tr>
<tr>
<td>$q_5$</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>$\bot$</td>
</tr>
</tbody>
</table>

Loss of Precision

Cause: application of $\sqcup$ at $q_4$ to merge data flow information.
Loss of Precision with the MFP Solution

At $q_5$, we have $z = 3$

Loss of Precision

Cause: application of $\sqcup$ at $q_4$ to merge data flow information
Loss of Precision with the MFP Solution

At $q_5$, we have $z = 3$

<table>
<thead>
<tr>
<th></th>
<th>$x$</th>
<th>$y$</th>
<th>$z$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_1$</td>
<td>$\top$</td>
<td>$\top$</td>
<td>$\top$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$1$</td>
<td>$\top$</td>
<td>$\top$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$2$</td>
<td>$\top$</td>
<td>$\top$</td>
</tr>
<tr>
<td>$q_4$</td>
<td>$1 \sqcup 2$</td>
<td>$2 \sqcup 1$</td>
<td>$\top$</td>
</tr>
<tr>
<td>$q_5$</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>$\bot$</td>
</tr>
</tbody>
</table>

Loss of Precision

Cause: application of $\sqcup$ at $q_4$ to merge data flow information.
Loss of Precision with the MFP Solution

At $q_5$, we have $z = 3$

<table>
<thead>
<tr>
<th></th>
<th>$x$</th>
<th>$y$</th>
<th>$z$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_1$</td>
<td>$\top$</td>
<td>$\top$</td>
<td>$\top$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>1</td>
<td>$\top$</td>
<td>$\top$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>2</td>
<td>$\top$</td>
<td>$\top$</td>
</tr>
<tr>
<td>$q_4$</td>
<td>$\top$</td>
<td>$\top$</td>
<td>$\top$</td>
</tr>
<tr>
<td>$q_5$</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>$\bot$</td>
</tr>
</tbody>
</table>

Loss of Precision
Cause: application of $\sqcup$ at $q_4$ to merge data flow information
Loss of Precision with the MFP Solution

At $q_5$, we have $z = 3$

Loss of Precision
Cause: application of $\sqcap$ at $q_4$ to merge data flow information
At $q_5$, we have $z = 3$

Loss of Precision

Cause: application of $\sqcup$ at $q_4$ to merge data flow information
Loss of Precision with the MFP Solution

At $q_5$, we have $z = 3$

Loss of Precision

Cause: application of $\sqcup$ at $q_4$ to merge data flow information

<table>
<thead>
<tr>
<th></th>
<th>$x$</th>
<th>$y$</th>
<th>$z$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_1$</td>
<td>$\top$</td>
<td>$\top$</td>
<td>$\top$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>1</td>
<td>$\top$</td>
<td>$\top$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>2</td>
<td>$\top$</td>
<td>$\top$</td>
</tr>
<tr>
<td>$q_4$</td>
<td>$\top$</td>
<td>$\top$</td>
<td>$\top$</td>
</tr>
<tr>
<td>$q_5$</td>
<td>$\top$</td>
<td>$\top$</td>
<td>$\top$</td>
</tr>
</tbody>
</table>
Alternative Approach for Better Precision

Control Paths from $q_1$ to $q_5$

At $q_5$, we have $z = 3$
Alternative Approach for Better Precision

At $q_5$, we have $z = 3$
Alternative Approach for Better Precision

At $q_5$, we have $z = 3$
Alternative Approach for Better Precision

At $q_5$, we have $z = 3$

Control Paths from $q_1$ to $q_5$

$\begin{align*}
q_1 &\rightarrow (T, T, T) \rightarrow q_1 \\
q_1 &\rightarrow (1, T, T) \rightarrow q_2 \\
q_2 &\rightarrow (1, T, T) \rightarrow q_2 \\
q_2 &\rightarrow (1, 2, T) \rightarrow q_4 \\
q_4 &\rightarrow (1, 2, 3) \rightarrow q_5 \\
q_5 &\rightarrow (1, 2, 3) \rightarrow q_5 \\
q_5 &\rightarrow (2, 1, 3) \rightarrow q_5 \\
q_5 &\rightarrow (2, 1, 3) \rightarrow q_5
\end{align*}$
Meet Over All Paths (MOP) Solution

Consider a data flow instance \( \langle (L, \sqsubseteq), \mathcal{F}, Q, q_{in}, q_{out}, \mathbb{X}, \rightarrow, f, \iota \rangle \).

Forward Meet Over All Paths Solution

\[
\overrightarrow{\text{MOP}} = \lambda q. \bigsqcup \left\{ f_{\iota} \circ \cdots \circ f_{\iota} \left| q_{in} \xrightarrow{\iota} q_{1} \cdots q_{k} \xrightarrow{\iota} q \right. \right\}
\]

Backward Meet Over All Paths Solution

\[
\overleftarrow{\text{MOP}} = \lambda q. \bigsqcup \left\{ f_{\iota} \circ \cdots \circ f_{\iota} \left| q \xrightarrow{\iota} q_{1} \cdots q_{k} \xrightarrow{\iota} q_{out} \right. \right\}
\]

More precise than MFP

\[
\overrightarrow{\text{MOP}} \sqsubseteq \overrightarrow{\text{MFP}} \quad \overleftarrow{\text{MOP}} \sqsubseteq \overleftarrow{\text{MFP}}
\]

Not Computable in General

\[
\text{MOP}(q) \not= 1 \text{ is undecidable for constant propagation}
\]
Meet Over All Paths (MOP) Solution

Consider a data flow instance \( \langle (L, \sqsubseteq), \mathcal{F}, Q, q_{in}, q_{out}, X, \rightarrow, f, \iota \rangle \).

**Forward Meet Over All Paths Solution**

\[
\overrightarrow{\text{MOP}} = \lambda q. \bigsqcup \left\{ f_{\circ \circ \circ \circ \circ k} \circ \cdots \circ f_{\circ \circ \circ \circ \circ 0}(\iota) \mid q_{in} \xrightarrow{\circ \circ \circ \circ \circ 0} q_1 \cdots q_k \xrightarrow{\circ \circ \circ \circ \circ k} q \right\}
\]

**Backward Meet Over All Paths Solution**

\[
\overleftarrow{\text{MOP}} = \lambda q. \bigsqcup \left\{ f_{\circ \circ \circ \circ \circ 0} \circ \cdots \circ f_{\circ \circ \circ \circ \circ k}(\iota) \mid q \xrightarrow{\circ \circ \circ \circ \circ k} q_1 \cdots q_k \xrightarrow{\circ \circ \circ \circ \circ 0} q_{out} \right\}
\]

**More precise than MFP**

\[
\overrightarrow{\text{MOP}} \sqsubseteq \overrightarrow{\text{MFP}}
\]

\[
\overleftarrow{\text{MOP}} \sqsubseteq \overleftarrow{\text{MFP}}
\]

**Not Computable in General**

\[
\overrightarrow{\text{MOP}}(q) \neq 1 \text{ is undecidable for constant propagation}
\]
MOP = MFP in Distributive Frameworks

A monotone framework \( \langle (L, \sqsubseteq), \mathcal{F} \rangle \) is distributive if every \( f \in \mathcal{F} \) is completely additive:

\[
f (\bigsqcup X) = \bigsqcup \{ f(\phi) \mid \phi \in X \} \quad \text{(for all } X \subseteq L)\]

**Theorem**

For any data flow instance over a distributive monotone framework,

\[
\overrightarrow{\text{MOP}} = \overrightarrow{\text{MFP}} \quad \overleftarrow{\text{MOP}} = \overleftarrow{\text{MFP}}
\]

**Intuition**

In a distributive framework, applying \( \bigsqcup \) “early” does not lose precision:

\[
f_{\text{op}_5} \circ f_{\text{op}_2}(\phi) \sqcup f_{\text{op}_3}(\psi) = f_{\text{op}_5} \circ f_{\text{op}_2}(\phi) \sqcup f_{\text{op}_5} \circ f_{\text{op}_3}(\psi)
\]
MOP = MFP in Distributive Frameworks

A monotone framework \( \langle (L, \sqsubseteq), \mathcal{F} \rangle \) is distributive if every \( f \in \mathcal{F} \) is completely additive:

\[
f (\bigsqcup X) = \bigsqcup \{ f(\phi) \mid \phi \in X \}
\]
(for all \( X \subseteq L \))

**Theorem**

*For any data flow instance over a distributive monotone framework,*

\[
\begin{align*}
\overrightarrow{\text{MOP}} &= \overrightarrow{\text{MFP}} \\
\overleftarrow{\text{MOP}} &= \overleftarrow{\text{MFP}}
\end{align*}
\]

**Intuition**

In a distributive framework, applying \( \bigsqcup \) “early” does not lose precision:

\[
f_{op5} \left( f_{op2}(\phi) \bigsqcup f_{op3}(\psi) \right) = f_{op5} \circ f_{op2}(\phi) \bigsqcup f_{op5} \circ f_{op3}(\psi)
\]
Examples of Distributive Monotone Frameworks

Gen / Kill Monotone Frameworks

- Complete lattice \((L, \sqsubseteq)\) of data flow facts:
  \[
  L = \mathcal{P}(S) \text{ for some set } S \sqsubseteq \subseteq \text{ or } \supseteq
  \]

- Set \(\mathcal{F}\) of monotonic transfer functions:
  \[
  \mathcal{F} = \{ \lambda \phi. \text{gen} \cup (\phi \setminus \text{kill}) \mid \text{gen, kill} \in L \}\]

All gen / kill monotone frameworks are distributive

Examples

- Live Variables
- Available Expressions
- Uninitialized Variables
- . . .
Sign Analysis: Monotone Framework

Control Flow Automaton: \( \langle Q, q_{in}, q_{out}, X, \rightarrow \rangle \)

(Simplified) Sign Lattice for a Single Variable: \((Sign, \sqsubseteq)\)

<table>
<thead>
<tr>
<th>(\phi)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\top)</td>
<td>(\mathbb{R})</td>
</tr>
<tr>
<td>(-)</td>
<td>({ r \in \mathbb{R} \mid r &lt; 0 })</td>
</tr>
<tr>
<td>(\geq)</td>
<td>({ r \in \mathbb{R} \mid r &gt; 0 })</td>
</tr>
<tr>
<td>0</td>
<td>({ 0 })</td>
</tr>
<tr>
<td>(\bot)</td>
<td>(\emptyset)</td>
</tr>
</tbody>
</table>

Monotone Framework

- Complete lattice \((L, \sqsubseteq)\) of data flow facts: \((X \rightarrow Sign, \sqsubseteq)\)
- Set \(\mathcal{F}\) defined as the set of all monotonic transfer functions on \(L\).
Sign Analysis: Data Flow Instance

Control Flow Automaton: \( \langle Q, q_{in}, q_{out}, X, \rightarrow \rangle \)

Monotone Framework
- Complete lattice \((L, \subseteq)\) of data flow facts: \((X \rightarrow \text{Sign}, \subseteq)\)
- Set \(\mathcal{F}\) defined as the set of all monotonic transfer functions on \(L\).

Data Flow Instance
- Initial data flow value: \(\top\)
- Transfer mapping:

\[
f_x := e(\phi) = \lambda y. \begin{cases} \phi(y) & \text{if } y \neq x \\ [e]_{\phi} & \text{if } y = x \end{cases}
\]

\[
f_g(\phi) = \phi
\]

Forward analysis
Sign Analysis: Transfer Mapping

Need to define $[e]$ for valuations $v$ in $x \rightarrow \{-, 0, +, \bot, \top\}$

### Expressions:

<table>
<thead>
<tr>
<th>$[c]_v$</th>
<th>$v(x)$</th>
<th>$[e_1 + e_2]_v$</th>
<th>$[e_1 - e_2]_v$</th>
<th>$[e_1 \times e_2]_v$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{sign}(c)$ [c ∈ Q]</td>
<td>$v(x)$ [x ∈ x]</td>
<td>$[e_1]_v \oplus [e_2]_v$</td>
<td>$[e_1]_v \ominus [e_2]_v$</td>
<td>$[e_1]_v \otimes [e_2]_v$</td>
</tr>
</tbody>
</table>

### “Abstract” Addition

<table>
<thead>
<tr>
<th>+</th>
<th>$\bot$</th>
<th>$-$</th>
<th>0</th>
<th>+</th>
<th>$\top$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\bot$</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>$\bot$</td>
</tr>
<tr>
<td>$-$</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>$\bot$</td>
</tr>
<tr>
<td>0</td>
<td>$\bot$</td>
<td>$-$</td>
<td>0</td>
<td>+</td>
<td>$\top$</td>
</tr>
<tr>
<td>+</td>
<td>$\bot$</td>
<td>$\top$</td>
<td>$+$</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>$\top$</td>
<td>$\bot$</td>
<td>$\top$</td>
<td>$\top$</td>
<td>$\top$</td>
<td>$\top$</td>
</tr>
</tbody>
</table>

### “Abstract” Subtraction

<table>
<thead>
<tr>
<th>$-$</th>
<th>$+$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\bot$</td>
<td>$\bot$</td>
</tr>
</tbody>
</table>

### “Abstract” Multiplication

<table>
<thead>
<tr>
<th>$\times$</th>
<th>0</th>
<th>+</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>+</td>
<td>0</td>
<td>+</td>
</tr>
</tbody>
</table>

### Sign Function

\[
\text{sign}(c) = \begin{cases} 
- & \text{if } c < 0 \\
0 & \text{if } c = 0 \\
+ & \text{if } c > 0 
\end{cases}
\]
Need to define $\llbracket e \rrbracket$ for valuations $v$ in $x \rightarrow \{-, 0, +, \bot, \top\}$

Expressions:

- $\llbracket c \rrbracket_v = \text{sign}(c)$ if $c \in \mathbb{Q}$
- $\llbracket x \rrbracket_v = v(x)$ if $x \in X$
- $\llbracket e_1 + e_2 \rrbracket_v = \llbracket e_1 \rrbracket_v \oplus \llbracket e_2 \rrbracket_v$
- $\llbracket e_1 - e_2 \rrbracket_v = \llbracket e_1 \rrbracket_v \ominus \llbracket e_2 \rrbracket_v$
- $\llbracket e_1 \times e_2 \rrbracket_v = \llbracket e_1 \rrbracket_v \otimes \llbracket e_2 \rrbracket_v$

“Abstract” Addition:

<table>
<thead>
<tr>
<th>$\oplus$</th>
<th>$\bot$</th>
<th>$-$</th>
<th>$0$</th>
<th>$+$</th>
<th>$\top$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\bot$</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>$\bot$</td>
</tr>
<tr>
<td>$-$</td>
<td>$\bot$</td>
<td>$-$</td>
<td>$-$</td>
<td>$\top$</td>
<td>$\top$</td>
</tr>
<tr>
<td>$0$</td>
<td>$\bot$</td>
<td>$-$</td>
<td>$0$</td>
<td>$+$</td>
<td>$\top$</td>
</tr>
<tr>
<td>$+$</td>
<td>$\bot$</td>
<td>$+$</td>
<td>$+$</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>$\top$</td>
<td>$\bot$</td>
<td>$\top$</td>
<td>$\top$</td>
<td>$\top$</td>
<td>$\top$</td>
</tr>
</tbody>
</table>

Sign ($\text{sign}(c)$):

- $-$ if $c < 0$
- $0$ if $c = 0$
- $+$ if $c > 0$

Tables also required for:
- “abstract” subtraction
- “abstract” multiplication
Sign Analysis: Transfer Mapping

Need to define $\llbracket e \rrbracket$ for valuations $\nu$ in $X \rightarrow \{-, 0, +, ⊥, ⊤\}$

Expressions:

<table>
<thead>
<tr>
<th>$\llbracket c \rrbracket \nu$</th>
<th>$\llbracket x \rrbracket \nu$</th>
<th>$\llbracket e_1 + e_2 \rrbracket \nu$</th>
<th>$\llbracket e_1 - e_2 \rrbracket \nu$</th>
<th>$\llbracket e_1 \ast e_2 \rrbracket \nu$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\llbracket c \rrbracket \nu$</td>
<td>$\llbracket x \rrbracket \nu$</td>
<td>$\llbracket e_1 \rrbracket \nu \oplus \llbracket e_2 \rrbracket \nu$</td>
<td>$\llbracket e_1 \rrbracket \nu \ominus \llbracket e_2 \rrbracket \nu$</td>
<td>$\llbracket e_1 \rrbracket \nu \otimes \llbracket e_2 \rrbracket \nu$</td>
</tr>
</tbody>
</table>

$\llbracket c \rrbracket \nu = \text{sign}(c)$ [for $c \in \mathbb{Q}$]

$\llbracket x \rrbracket \nu = \nu(x)$ [for $x \in X$]

$\llbracket e_1 + e_2 \rrbracket \nu = \llbracket e_1 \rrbracket \nu \oplus \llbracket e_2 \rrbracket \nu$

$\llbracket e_1 - e_2 \rrbracket \nu = \llbracket e_1 \rrbracket \nu \ominus \llbracket e_2 \rrbracket \nu$

$\llbracket e_1 \ast e_2 \rrbracket \nu = \llbracket e_1 \rrbracket \nu \otimes \llbracket e_2 \rrbracket \nu$

$\text{sign}(c) = \begin{cases} 
  - & \text{if } c < 0 \\
  0 & \text{if } c = 0 \\
  + & \text{if } c > 0 
\end{cases}$

Are these tables correct?

Tables also required for:

- “abstract” subtraction
- “abstract” multiplication
Sign Analysis: Transfer Mapping

Need to define \([e]\) for valuations \(\nu\) in \(X \rightarrow \{-, 0, +, \bot, \top\}\)

**Expressions:**

\[[c]_{\nu} = \text{sign}(c) \quad [c \in \mathbb{Q}]\]

\[[x]_{\nu} = \nu(x) \quad [x \in X]\]

\[[e_1 + e_2]_{\nu} = [e_1]_{\nu} \oplus [e_2]_{\nu}\]

\[[e_1 - e_2]_{\nu} = [e_1]_{\nu} \ominus [e_2]_{\nu}\]

\[[e_1 \ast e_2]_{\nu} = [e_1]_{\nu} \otimes [e_2]_{\nu}\]

\(\text{sign}(c) = \begin{cases} 
- & \text{if } c < 0 \\
0 & \text{if } c = 0 \\
+ & \text{if } c > 0
\end{cases}\)

**Are these tables correct?**

**Does this data flow instance really perform sign analysis?**

**Is the analysis correct?**

**Is it precise?**
What About Correctness of Data Flow Analyses?

\[
(L, \sqsubseteq) \quad F \quad \vdash \quad f : \mathcal{O}_P \rightarrow F
\]

Desired Analysis

Framework

\[
\langle Q, q_{\text{in}}, q_{\text{out}}, X, \rightarrow \rangle
\]

Solution

MFP MOP

Transfer

Program

Ideal Solution

Desired Analysis

Framework

Transfer

Program

Solution

MFP MOP
What About Correctness of Data Flow Analyses?

Framework

\( (L, \sqsubseteq) \)

\( \mathcal{F} \)

\( \nu \in L \)

\( f : \mathcal{O}_P \rightarrow \mathcal{F} \)

Desired Analysis

Solution

MFP MOP

Ideal Solution

\( \langle Q, q_{\text{in}}, X, \rightarrow \rangle \)

Transfer

\( \langle Q, q_{\text{in}}, q_{\text{out}}, X, \rightarrow \rangle \)

Program

Semantics

\[ \langle Q, q_{\text{in}}, X, \rightarrow \rangle \]
What About Correctness of Data Flow Analyses?

Desired Analysis

Ideal Solution

Semantics

Framework

Transfer

Solution

MFP MOP

\((L, \sqsubseteq)\)

\(\mathcal{F}\)

\(i \in L\)

\(f : \mathcal{O}_P \rightarrow \mathcal{F}\)

\(\langle Q, q_{in}, X, \rightarrow \rangle\)

\(\langle Q, q_{in}, q_{out}, X, \rightarrow \rangle\)

soundly approximates
Manual correctness proof for each analysis soundly approximates desired analysis solution.

Framework

Program

Solution

Ideal Solution

Desired Analysis

Semantics

Manual correctness proof for each analysis soundly approximates desired analysis solution.

Framework

Program

Solution

Ideal Solution

Desired Analysis

Semantics
Data flow facts have an intended meaning.

The transfer mapping is designed according to this intended meaning.

We need a formal link to relate data flow facts and transfer functions with the formal semantics.

Solution: Abstract Interpretation

« This paper is devoted to the systematic and correct design of program analysis frameworks with respect to a formal semantics. »

Data flow facts have an intended meaning.

The transfer mapping is designed according to this intended meaning.

We need a formal link to relate data flow facts and transfer functions with the formal semantics.

Solution: Abstract Interpretation

"This paper is devoted to the systematic and correct design of program analysis frameworks with respect to a formal semantics."