

1st Asian Pacific Summer School on Formal Methods

Course 12: Static Analysis of C programs with Frama-C

Virgile Prevosto

CEA List

August 30, 2009



long n
for (i = 0; i < n; i++)
C[i] = 0;
tmp2 = 0;
for (k = 0; k < n; k++)
tmp2 += C[k];

tmp2[i] = 0; for (k = 0; k < n; k++) tmp2[i] += m2[i][k] * tmp2[k]; /* The [i,j] coefficient of the matrix product MC2*TMP2, that is, *MC2*(TMP1) = MC2*(MC1*M1) = MC2*M1*MC1. Final rounding: tmp2[i] is now represented on 9 bits: if (tmp2[i] < -256) tmp2[i] = -256; else if (tmp2[i] > 255) tmp2[i] = 255; else tmp2[i] = tmp2[i];

Presentation

Data-flow Analysis

Abstract Interpretation

Abstract Interpretation in Practice

long ra
for 0
C1) if m
tmp2 =
of the

tmp2[0] = 1 << (n0 - 1) else if (tmp1[0]) >= 1 << (n0 - 1) tmp2[0] = (1 << (n0 - 1) + abs(tmp2[0] - tmp1[0])) / 2; Then the second part takes the first part, i.e.,
tmp1[0][k] = 0; k = 0; k++) tmp1[0][k] += mc2[0][k] * tmp2[0][k]; /* The [i][j] coefficient of the matrix product MC2*TMP2, that is, *MC2*(TMP1) = MC2*(M1*M1) = MC2*M1*M1
i = 1; tmp1[0][i] >>= 2; */ Final rounding: tmp2[0][i] is now represented on 9 bits. *if (tmp1[0][i] < -256) m2[0][i] = -256; else if (tmp1[0][i] > 255) m2[0][i] = 255; else m2[0][i] = tmp1[0][i];



Presentation

Data-flow Analysis

Abstract Interpretation

Abstract Interpretation in Practice

long ra
for 0 <=
C1) if (m
tmp2 =
e of the

tmp2[0] = 1 << (Nb1 - 1) else if (tmp1[0]) >= 1 << (Nb1 - 1) tmp2[0] = (1 << (Nb1 - 1) - 1) else tmp2[0] = tmp1[0]; /* Then the second part looks like the first one: */
tmp1[0][k] = 0; k = 0; k++) tmp1[0][k] += mc2[0][k] * tmp2[0][k]; /* The [i,j] coefficient of the matrix product MC2*TMP2, that is, *MC2*(TMP1) = MC2*(M1C1) = MC2*M1 *MC1
l = 1; tmp1[0][l] >= 1; */ Final rounding: tmp2[0][l] is now represented on 9 bits. *if (tmp1[0][l] < -256) m2[0][l] = -256; else if (tmp1[0][l] > 255) m2[0][l] = 255; else m2[0][l] = tm



Main objective

Statically determine some semantic properties of a program

- ▶ safety: pointer are all valid, no arithmetic overflow, ...
- ▶ termination
- ▶ functional properties
- ▶ dead code
- ▶ ...

Embedded code

- ▶ Much simpler than desktop applications
- ▶ Some parts are critical, *i.e.* a bug have severe consequences (financial loss, or even dead people)
- ▶ Thus a good target for static analysis



Polyspace Verifier Checks for (absence of) run-time error
 C/C++/Ada)

http:

[//www.mathworks.com/products/polyspace/](http://www.mathworks.com/products/polyspace/)

ASTRÉE Absence of error *without false alarm* in
 SCADE-generated code

http:

[//www.di.ens.fr/~cousot/projets/ASTREE/](http://www.di.ens.fr/~cousot/projets/ASTREE/)

Coverity Checks for various code defects (C/C++/Java)

<http://www.coverity.com>



a3 Worst-case execution time and Stack depth

<http://www.absint.com/>

FLUCTUAT Accuracy of floating-point computations and origin of rounding errors

<http://www-list.cea.fr/labos/fr/LSL/fluctuat/>

Frama-C A toolbox for analysis of C programs

<http://frama-c.cea.fr/>



- ▶ 90's: CAVEAT, an Hoare logic-based tool for C programs
- ▶ 2000's: CAVEAT used by Airbus during certification process of the A380
- ▶ 2002: Why and its C front-end Caduceus
- ▶ 2006: Joint project to write a successor to CAVEAT and Caduceus
- ▶ 2008: First public release of Frama-C



- ▶ 90's: CAVEAT, an Hoare logic-based tool for C programs
- ▶ 2000's: CAVEAT used by Airbus during certification process of the A380
- ▶ 2002: Why and its C front-end Caduceus
- ▶ 2006: Joint project to write a successor to CAVEAT and Caduceus
- ▶ 2008: First public release of Frama-C



long ra
t for 0
ct) if m
tmp2
e of the

tmp2[0] = 1 << (nbl - 1) else if (tmp1[0]) >= 1 << (nbl - 1) tmp2[0] = 1 << (nbl - 1) + abs(tmp2[0] - tmp1[0]); /* Then the second part takes the first one
tmp1[0] = 0; k = 0; k++) tmp1[0] += mc2[0][k] * tmp2[k]; /* The [i][j] coefficient of the matrix product MC2*TMP2, that is, *MC2*(TMP1) = MC2*(MC1*M1) = MC2*M1 *MC1
l = 1; tmp1[0] >= 1; /* Final rounding: tmp2[0] is now represented on 9 bits. *if (tmp1[0] < -256) tmp2[0] = -256; else if (tmp1[0] > 255) tmp2[0] = 255; else tmp2[0] = tmp1[0];

- ▶ 90's: CAVEAT, an Hoare logic-based tool for C programs
- ▶ 2000's: CAVEAT used by Airbus during certification process of the A380
- ▶ 2002: Why and its C front-end Caduceus
- ▶ 2006: Joint project to write a successor to CAVEAT and Caduceus
- ▶ 2008: First public release of Frama-C



- ▶ 90's: CAVEAT, an Hoare logic-based tool for C programs
- ▶ 2000's: CAVEAT used by Airbus during certification process of the A380
- ▶ 2002: Why and its C front-end Caduceus
- ▶ 2006: Joint project to write a successor to CAVEAT and Caduceus
- ▶ 2008: First public release of Frama-C



- ▶ A modular architecture
- ▶ Kernel:
 - ▶ CIL (U. Berkeley) library for the C front-end
 - ▶ ACSL front-end
 - ▶ Global management of analyzer's state
- ▶ Various plug-ins for the analysis
 - ▶ Value analysis (abstract interpretation)
 - ▶ Jessie (translation to Why)
 - ▶ Slicing
 - ▶ Impact analysis
 - ▶ ...



long ra
for 0 <=
C1) if (m
tmp2 =
e of the

tmp2[j] = 0; else if (tmp1[j] >= 0) tmp2[j] = (1 << (N81 - j)) * tmp1[j]; else tmp2[j] = tmp1[j]; /* Then the second part looks like the first one. */
tmp1[0][k] = 0; k = 0; k++) tmp1[0][k] += m2[0][k] * tmp2[k][j]; /* The [k] coefficient of the matrix product MC2*TMP2, that is, *MC2*(TMP1) = MC2*(M1)*M1 = MC2*M1*M1 = 1 * tmp1[0][j] >= 1. */ Final rounding: tmp2[0][j] is now represented on 9 bits. *if (tmp1[0][j] < -256) m2[0][j] = -256; else if (tmp1[0][j] > 255) m2[0][j] = 255; else m2[0][j] = tmp1[0][j];

- ▶ A modular architecture
- ▶ Kernel:
 - ▶ CIL (U. Berkeley) library for the C front-end
 - ▶ ACSL front-end
 - ▶ Global management of analyzer's state
- ▶ Various plug-ins for the analysis
 - ▶ Value analysis (abstract interpretation)
 - ▶ Jessie (translation to Why)
 - ▶ Slicing
 - ▶ Impact analysis
 - ▶ ...

if long ra
t for 0 <=
C1; if (m
tmp2 =
e of the

tmp2[0] = 1 << (nbl - 1) else if (tmp1[0]) >> 1 << (nbl - 1) tmp2[0] = 1 << (nbl - 1) + tmp1[0]; /* Then the second part looks like the first one. */
tmp1[0][k] = 0; k = 0; k++ tmp1[0][k] += mc2[0][k] * tmp2[0][k]; /* The [k] coefficient of the matrix product MC2*TMP2, that is, *MC2*(TMP1) = MC2*(M1) = MC2*(M1)*MC1
l = 1; tmp1[0][l] >>= 1; */ Final rounding: tmp2[0][l] is now represented on 9 bits. *if (tmp1[0][l] < -256) m2[0][l] = -256; else if (tmp1[0][l] > 255) m2[0][l] = 255; else m2[0][l] = tmp1[0][l];



- ▶ A modular architecture
- ▶ Kernel:
 - ▶ CIL (U. Berkeley) library for the C front-end
 - ▶ ACSL front-end
 - ▶ Global management of analyzer's state
- ▶ Various plug-ins for the analysis
 - ▶ Value analysis (abstract interpretation)
 - ▶ Jessie (translation to Why)
 - ▶ Slicing
 - ▶ Impact analysis
 - ▶ ...

long ra
for 0 <=
C1; if (m
tmp2 =
se of the

tmp2[j] = 0; if (i < (n1 - 1) || (i == (n1 - 1) && tmp1[j] >= 0)) tmp2[j] = (i < (n1 - 1) ? tmp1[j] : tmp1[j] + 1); Then the second part takes for the first part
tmp1[j] = 0; k = 0; k++ tmp1[j] += m2[0][k] * tmp2[k]; /* The [j] coefficient of the matrix product MC2*TMP2, that is *MC2*(TMP1) = MC2*(MC1*M1) = MC2*M1*MC1
i = 1; tmp1[j] >= 1; */ Final rounding: tmp2[j] is now represented on 9 bits: *if (tmp1[j] < -256) tmp2[j] = -256; else if (tmp1[j] > 255) tmp2[j] = 255; else tmp2[j] =



- ▶ A modular architecture
- ▶ Kernel:
 - ▶ CIL (U. Berkeley) library for the C front-end
 - ▶ ACSL front-end
 - ▶ Global management of analyzer's state
- ▶ Various plug-ins for the analysis
 - ▶ Value analysis (abstract interpretation)
 - ▶ Jessie (translation to Why)
 - ▶ Slicing
 - ▶ Impact analysis
 - ▶ ...



- ▶ A modular architecture
- ▶ Kernel:
 - ▶ CIL (U. Berkeley) library for the C front-end
 - ▶ ACSL front-end
 - ▶ Global management of analyzer's state
- ▶ Various plug-ins for the analysis
 - ▶ Value analysis (abstract interpretation)
 - ▶ Jessie (translation to Why)
 - ▶ Slicing
 - ▶ Impact analysis
 - ▶ ...



- ▶ A modular architecture
- ▶ Kernel:
 - ▶ CIL (U. Berkeley) library for the C front-end
 - ▶ ACSL front-end
 - ▶ Global management of analyzer's state
- ▶ Various plug-ins for the analysis
 - ▶ Value analysis (abstract interpretation)
 - ▶ Jessie (translation to Why)
 - ▶ Slicing
 - ▶ Impact analysis
 - ▶ ...



- ▶ A modular architecture
- ▶ Kernel:
 - ▶ CIL (U. Berkeley) library for the C front-end
 - ▶ ACSL front-end
 - ▶ Global management of analyzer's state
- ▶ Various plug-ins for the analysis
 - ▶ Value analysis (abstract interpretation)
 - ▶ Jessie (translation to Why)
 - ▶ Slicing
 - ▶ Impact analysis
 - ▶ ...



- ▶ A modular architecture
- ▶ Kernel:
 - ▶ CIL (U. Berkeley) library for the C front-end
 - ▶ ACSL front-end
 - ▶ Global management of analyzer's state
- ▶ Various plug-ins for the analysis
 - ▶ Value analysis (abstract interpretation)
 - ▶ Jessie (translation to Why)
 - ▶ Slicing
 - ▶ Impact analysis
 - ▶ ...



- ▶ A modular architecture
- ▶ Kernel:
 - ▶ CIL (U. Berkeley) library for the C front-end
 - ▶ ACSL front-end
 - ▶ Global management of analyzer's state
- ▶ Various plug-ins for the analysis
 - ▶ Value analysis (abstract interpretation)
 - ▶ Jessie (translation to Why)
 - ▶ Slicing
 - ▶ Impact analysis
 - ▶ ...



Presentation

Data-flow Analysis

Abstract Interpretation

Abstract Interpretation in Practice

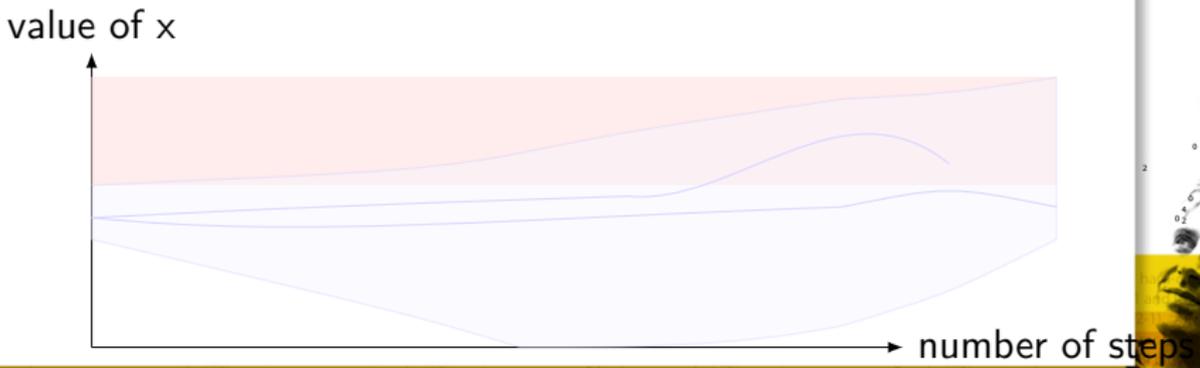
long ra
for 0 <=
C1) if (m
tmp2 =
e of the

tmp2[0] = 1 << (Nb1 - 1) else if (tmp1[0]) >= 1 << (Nb1 - 1) tmp2[0] = (1 << (Nb1 - 1) - 1) else tmp2[0] = tmp1[0]; /* Then the second part looks like the first one: */
tmp1[0][k] = 0; k = 0; k++ tmp1[0][k] += mc2[0][k] * tmp2[0][k]; /* The [i,j] coefficient of the matrix product MC2*TMP2, that is, *MC2*(TMP1) = MC2*(M1)*M1) = MC2*(M1)*M1
l = 1; tmp1[0][l] >= 1; /* Final rounding: tmp2[0][l] is now represented on 9 bits. */ if (tmp1[0][l] < -256) m2[0][l] = -256; else if (tmp1[0][l] > 255) m2[0][l] = 255; else m2[0][l] = tm



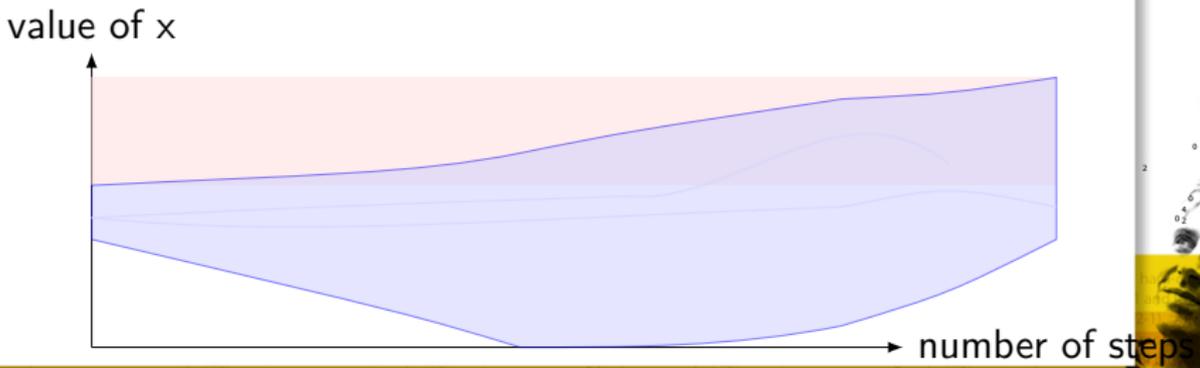
Concrete semantics

- ▶ Formalisation of **all** possible behaviors of a program
- ▶ Function which associate to a program an element of the **concrete domain** of interest
- ▶ Trace semantics: associate to each program point the values that the variables can take at this point



Concrete semantics

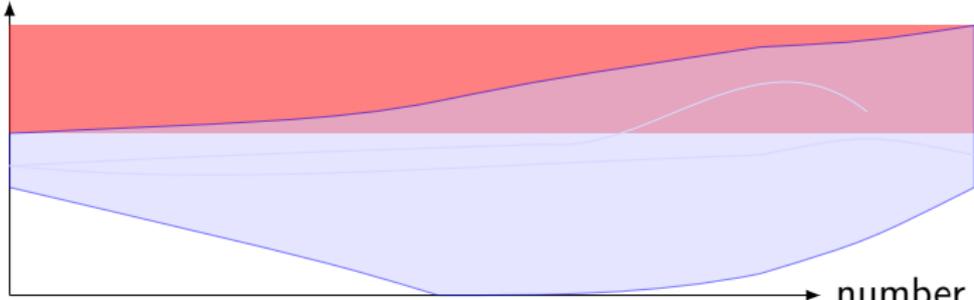
- ▶ Formalisation of **all** possible behaviors of a program
- ▶ Function which associate to a program an element of the **concrete domain** of interest
- ▶ Trace semantics: associate to each program point the values that the variables can take at this point



Concrete semantics

- ▶ Formalisation of **all** possible behaviors of a program
- ▶ Function which associate to a program an element of the **concrete domain** of interest
- ▶ Trace semantics: associate to each program point the values that the variables can take at this point

value of x

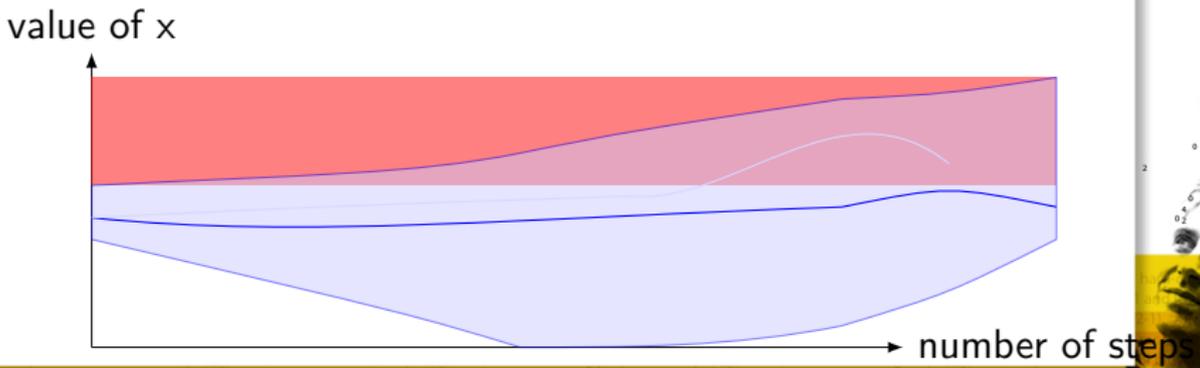


number of steps



Concrete semantics

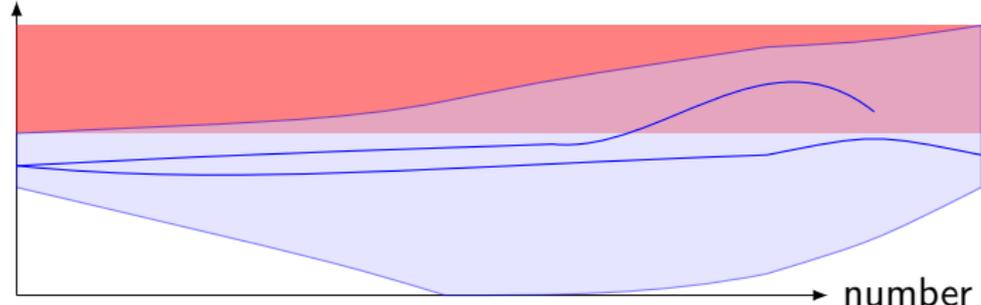
- ▶ Formalisation of **all** possible behaviors of a program
- ▶ Function which associate to a program an element of the **concrete domain** of interest
- ▶ Trace semantics: associate to each program point the values that the variables can take at this point



Concrete semantics

- ▶ Formalisation of **all** possible behaviors of a program
- ▶ Function which associate to a program an element of the **concrete domain** of interest
- ▶ Trace semantics: associate to each program point the values that the variables can take at this point

value of x



number of steps



```
int fact(int x) {  
    int z = 1, y = 1;  
    if (x < 4) { x = 4; }  
    while (y <= x) {  
        z = z * y;  
        y++;  
    }  
    return z;  
}
```



```

int fact(int x) {
    int z = 1, y = 1;
    if (x < 4) { x = 4; }
    while (y <= x) {
        z = z * y;
        y++;
    }
    return z;
}
    
```

$z \in \{24, 120, 720, \dots\}$



- ▶ Control-flow graph (CFG) of the program
- ▶ Each edge has an associated a transfer function $f_{i,j} : L \rightarrow L$
- ▶ System of equations $l_i = \bigcup_{\{e_j \in I_j\}} \{f_{j,i}(e_j)\}$
- ▶ Solved by successive iterations (Kleene)

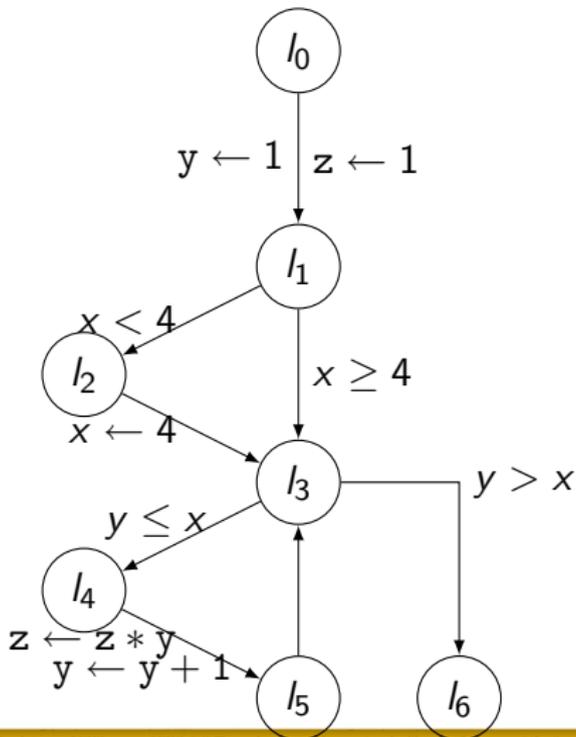


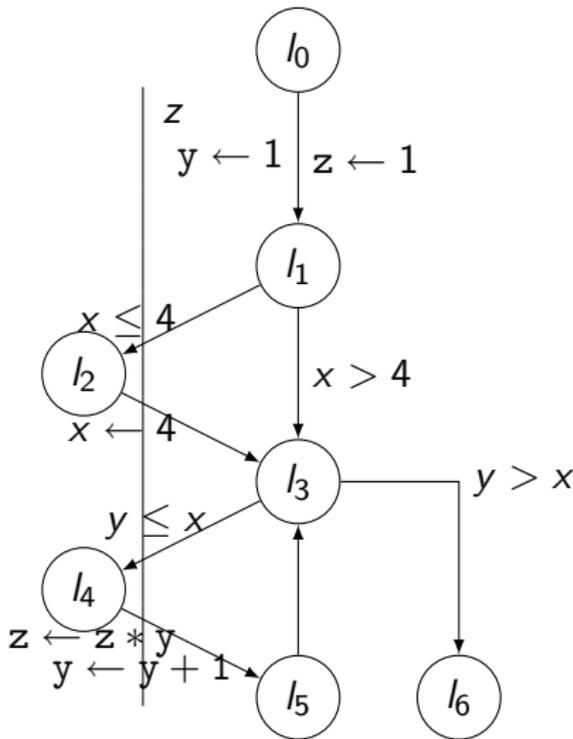
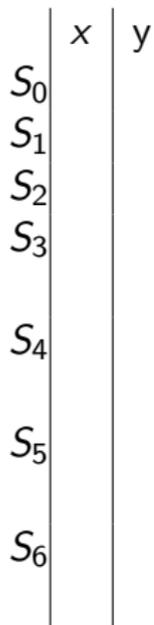
long n
for 0 <=
C1; if (n
tmp2 =
of the

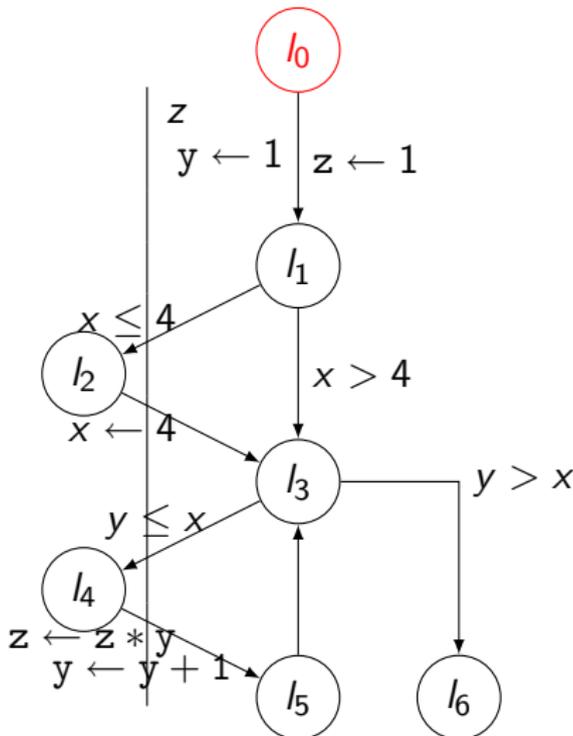
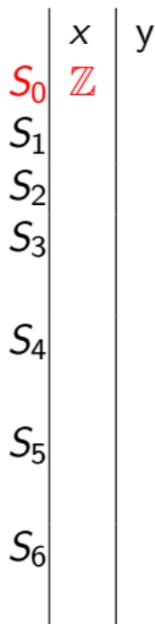
tmp2[0] = 1 << (n-1) else if (tmp1[0]) << (n-1) else tmp2[0] = 1 << (n-1) + tmp1[0]; /* Then the second part takes like the first one */
tmp1[0] = 0; k = 5; k <= 8; k <= 8; tmp1[0] = mc2[0][k] * tmp2[0][k]; /* The [i][j] coefficient of the matrix product MC2*TMP2, that is, *MC2*(TMP1) = MC2*(MC1*M1) = MC2*M1 * MC1
i = 1; tmp1[0] >= 1; /* Final rounding: tmp2[0] is now represented on 9 bits: if (tmp1[0] < 256) tmp2[0] = 256; else if (tmp1[0] > 255) tmp2[0] = 255; else tmp2[0] = tmp1[0];

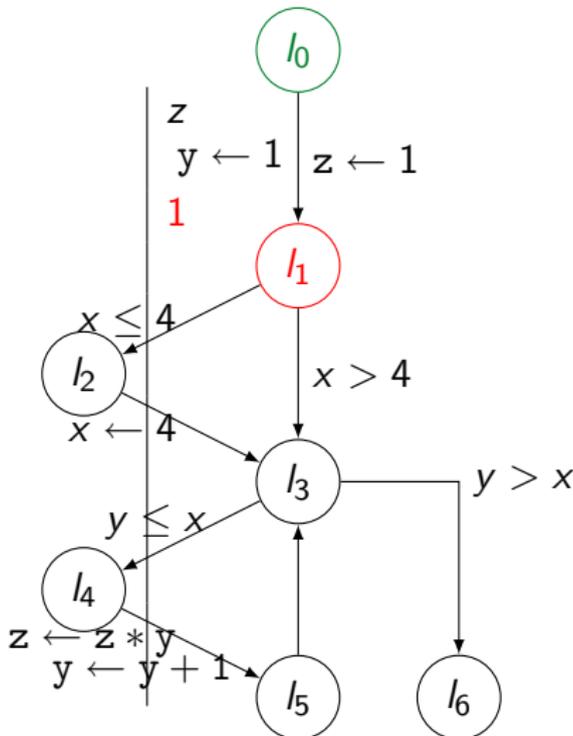
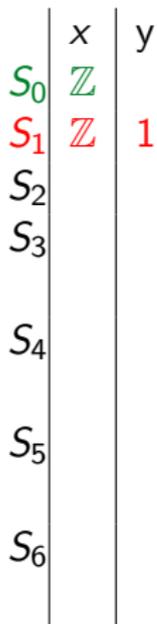
```

int fact(int x) {
    int z = 1, y = 1;
    if (x < 4) { x = 4; }
    while (y <= x) {
        z = z * y;
        y++;
    }
    return z;
}
    
```

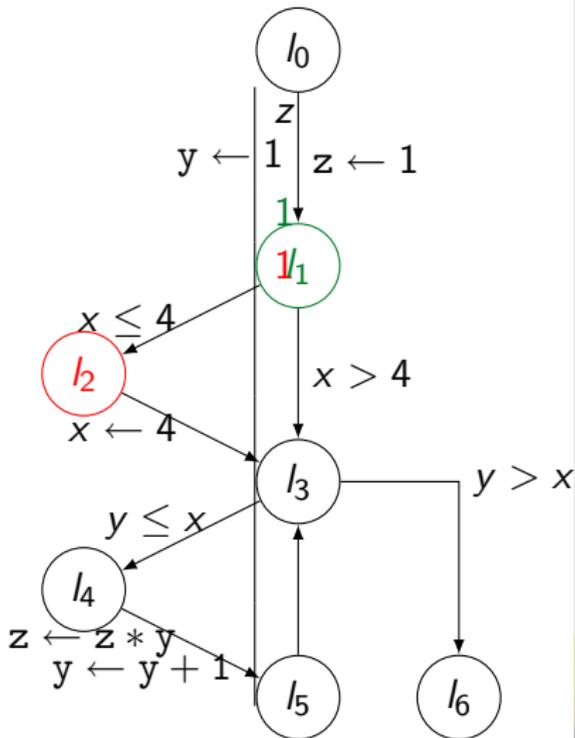




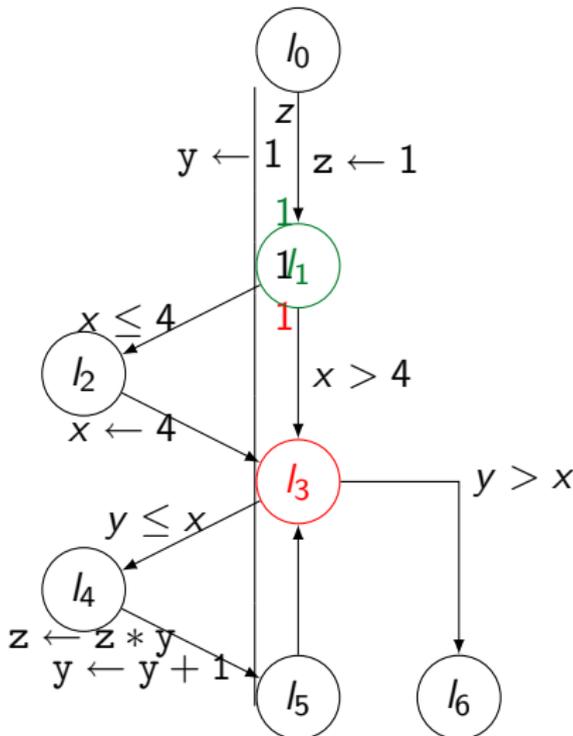




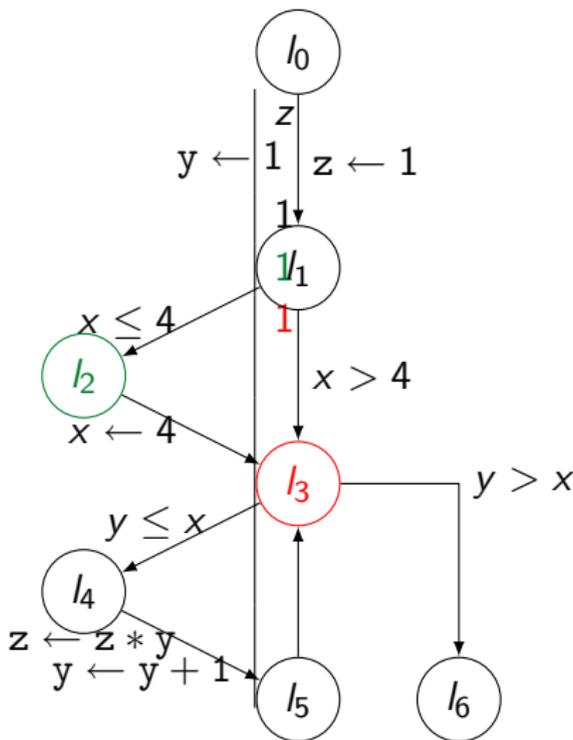
	x	y
S_0	\mathbb{Z}	
S_1	\mathbb{Z}	1
S_2	$-\infty..4$	1
S_3		
S_4		
S_5		
S_6		



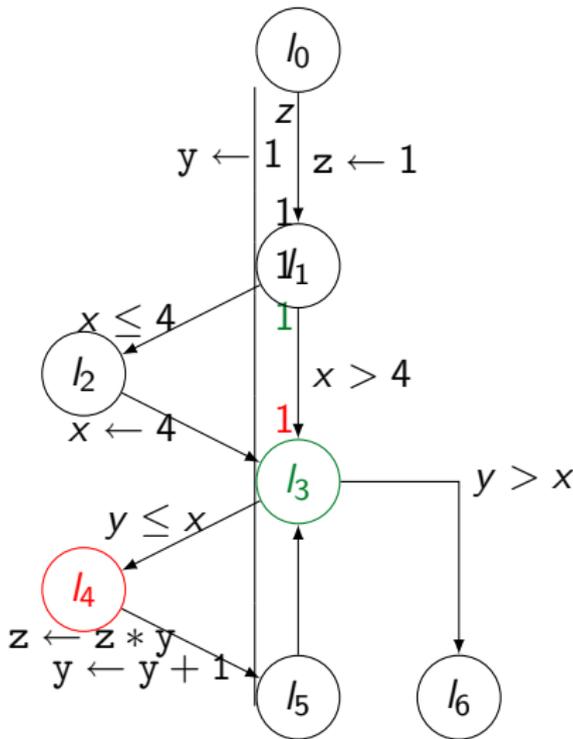
	x	y
S_0	\mathbb{Z}	
S_1	\mathbb{Z}	1
S_2	$-\infty..4$	1
S_3	$4..+\infty$	1
S_4		
S_5		
S_6		



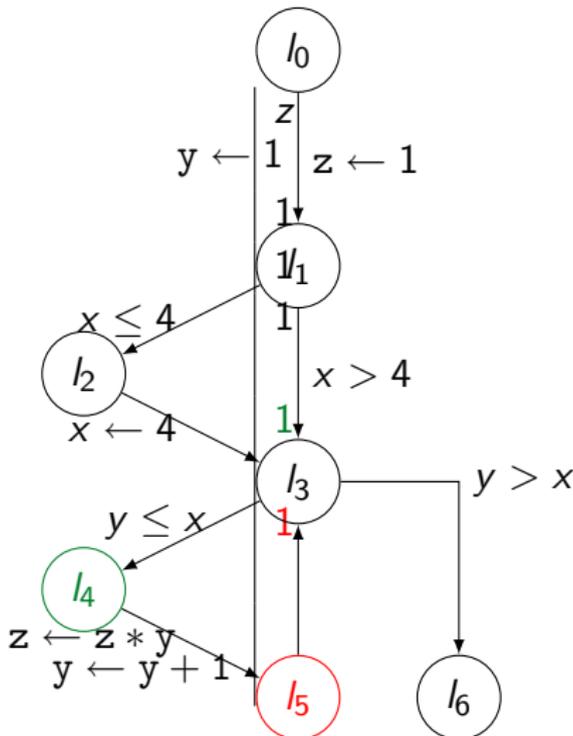
	x	y
S_0	\mathbb{Z}	
S_1	\mathbb{Z}	1
S_2	$-\infty..4$	1
S_3	$4..+\infty$	1
S_4		
S_5		
S_6		



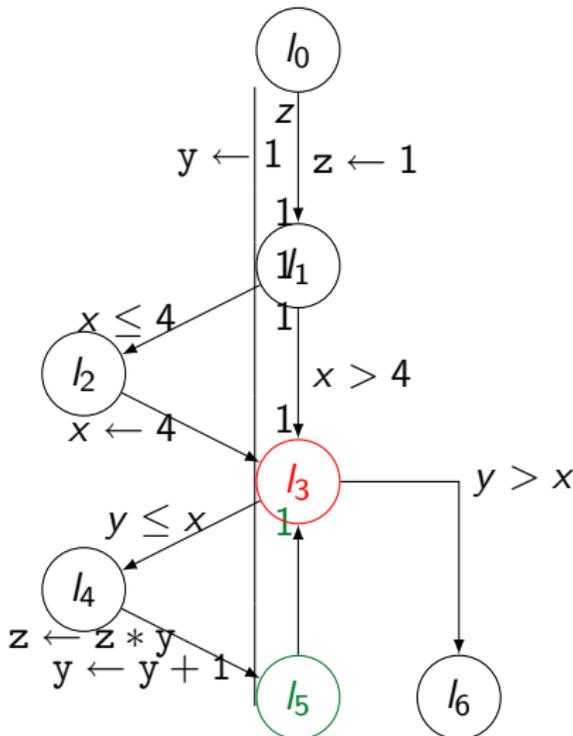
	x	y
S_0	\mathbb{Z}	
S_1	\mathbb{Z}	1
S_2	$-\infty..4$	1
S_3	$4..+\infty$	1
S_4	$4..+\infty$	1
S_5		
S_6		



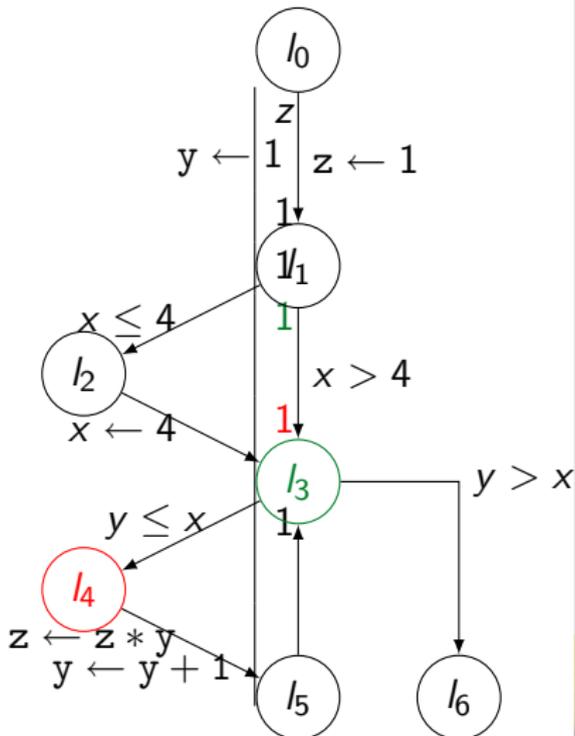
	x	y
S_0	\mathbb{Z}	
S_1	\mathbb{Z}	1
S_2	$-\infty..4$	1
S_3	$4..+\infty$	1
S_4	$4..+\infty$	1
S_5	$4..+\infty$	2
S_6		



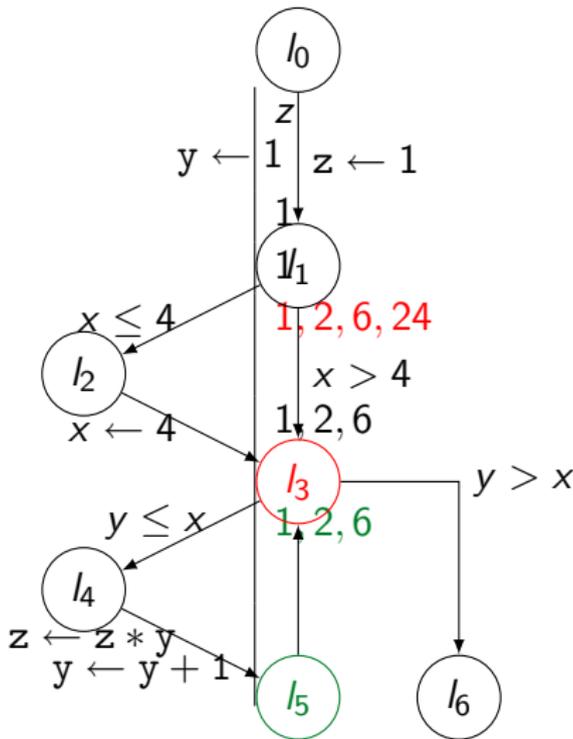
	x	y
S_0	\mathbb{Z}	
S_1	\mathbb{Z}	1
S_2	$-\infty..4$	1
S_3	$4..+\infty$	1,2
S_4	$4..+\infty$	1
S_5	$4..+\infty$	2
S_6		



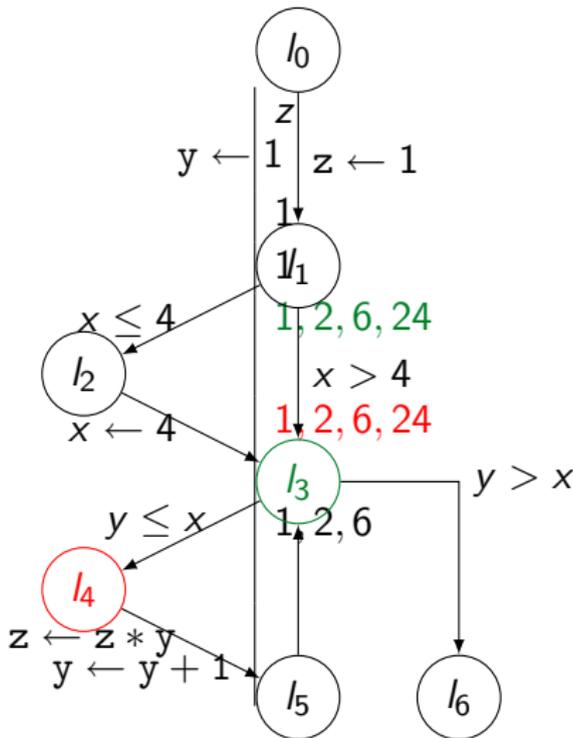
	x	y
S_0	\mathbb{Z}	
S_1	\mathbb{Z}	1
S_2	$-\infty..4$	1
S_3	$4..+\infty$	1,2
S_4	$4..+\infty$	1,2
S_5	$4..+\infty$	2
S_6		



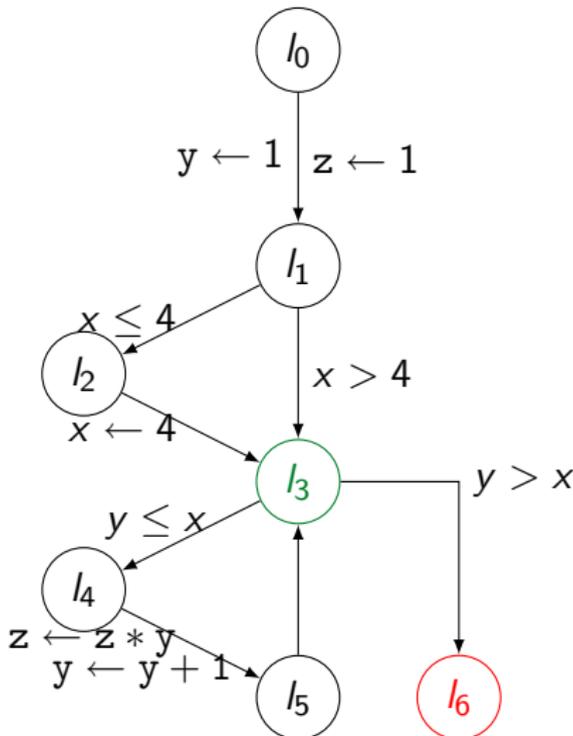
	x	y
S_0	\mathbb{Z}	
S_1	\mathbb{Z}	1
S_2	$-\infty..4$	1
S_3	$4..+\infty$	1..5
S_4	$4..+\infty$	1..4
S_5	$4..+\infty$	2..5
S_6		



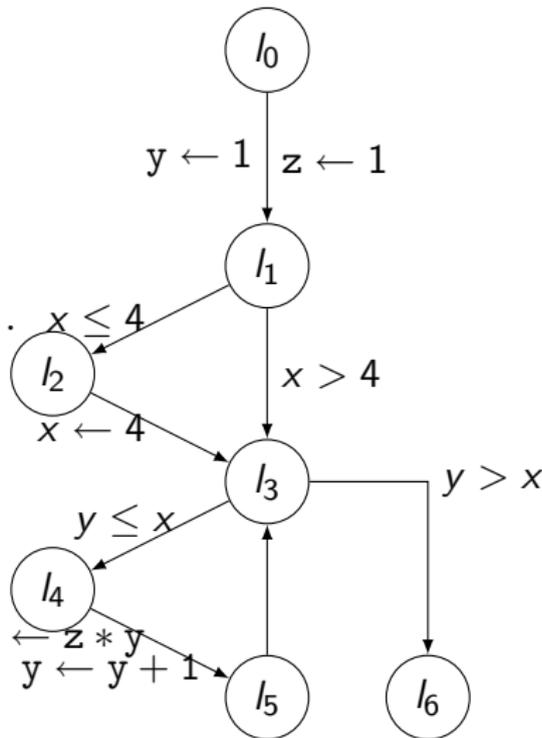
	x	y
S_0	\mathbb{Z}	
S_1	\mathbb{Z}	1
S_2	$-\infty..4$	1
S_3	$4..+\infty$	1.5
S_4	$4..+\infty$	1.5
S_5	$4..+\infty$	2.5
S_6		



	x	y	z
S_0	\mathbb{Z}		
S_1	\mathbb{Z}	1	1
S_2	$-\infty..4$	1	1
S_3	$4..+\infty$	1.5	1, 2, 6, 24
S_4	$4..+\infty$	1.5	1, 2, 6, 24
S_5	$4..+\infty$	2.5	1, 2, 6
S_6	$4..+\infty$	5	24



	x	y	z
S_0	\mathbb{Z}		
S_1	\mathbb{Z}	1	1
S_2	$-\infty..4$	1	1
S_3	$4..+\infty$	$1..$	$1, 2, 6, 24 \dots$
S_4	$4..+\infty$	$1..$	$1, 2, 6, \dots$
S_5	$4..+\infty$	$2..$	$1, 2, 6, \dots$
S_6	$4..+\infty$	$4..$	$1, 2, 6, \dots$



Rice's Theorem

Any non-trivial semantic property of a program is undecidable.

Example

Halting problem: it cannot be decided statically if a given program will always terminate or not

Approximations

Even if the general case is unreachable, it is possible to devise analyses that give useful information

```

long n;
for (i = 0; i < n; i++)
    C[i] = 0;
tmp2 =
... of the

```

```

tmp2[0] = 1; // (N-1) else if (tmp1[0]) >= 1; // (N-1) - 1; else tmp2[0] = tmp1[0]; /* Then the second part takes the first one:
tmp1[0] = 0; k = 0; k++ tmp1[0][k] += mc2[0][k] * tmp2[0][k]; /* The [k] coefficient of the matrix product MC2*TMP2, that is, *MC2*(TMP1) = MC2*(MC1*M1) = MC2*M1*MC1
l = 1; tmp1[0][l] >= 1; /* Final rounding: tmp2[0][l] is now represented on 9 bits: *if (tmp1[0][l] < -256) m2[0][l] = -256; else if (tmp1[0][l] > 255) m2[0][l] = 255; else m2[0][l] = tm

```



Rice's Theorem

Any non-trivial semantic property of a program is undecidable.

Example

Halting problem: it cannot be decided statically if a given program will always terminate or not

Approximations

Even if the general case is unreachable, it is possible to devise analyses that give useful information

```

long n;
for (i = 0; i < n; i++)
    tmp2 =
    ...

```

```

tmp2[i] = (i < (n-1) ? tmp[i] : 0); // The [i] coefficient of the matrix product MC2*TMP2, that is, *MC2*(TMP1) = MC2*(MC1*M1) = MC2*M1*MC1
tmp1[i] = 0; k = 0; k++ tmp1[i] += mc2[i][k] * tmp2[k]; // The [i] coefficient of the matrix product MC2*TMP2, that is, *MC2*(TMP1) = MC2*(MC1*M1) = MC2*M1*MC1
i = i + 1; tmp1[i] >>= 1; // Final rounding, tmp2[i] is now represented on 9 bits. *if (tmp1[i] < -256) m2[i] = -256; else if (tmp1[i] > 255) m2[i] = 255; else m2[i] = tmp1[i];

```



Rice's Theorem

Any non-trivial semantic property of a program is undecidable.

Example

Halting problem: it cannot be decided statically if a given program will always terminate or not

Approximations

Even if the general case is unreachable, it is possible to devise analyses that give useful information



Presentation

Data-flow Analysis

Abstract Interpretation

Abstract Interpretation in Practice

long ra
for 0 ->
C1) if (u
tmp2 =
e of the

tmp2[0] = 1 << (N8 - 1) else if (tmp1[0]) >= 1 << (N8 - 1) tmp2[0] = (1 << (N8 - 1) - 1) else tmp2[0] = tmp1[0]; /* Then the second part looks like the first one: */
tmp1[0][k] = 0; k = 8; k--> tmp1[0][k] += mc2[0][k] * tmp2[0][k]; /* The [i,j] coefficient of the matrix product MC2*TMP2, that is, *MC2*(TMP1) = MC2*(M1)*M1 = MC2*M1*M1
l = 1; tmp1[0][l] >= 1; /* Final rounding: tmp2[0][l] is now represented on 9 bits. */ if (tmp1[0][l] < -256) m2[0][l] = -256; else if (tmp1[0][l] > 255) m2[0][l] = 255; else m2[0][l] = tm



- ▶ Ensuring termination of the analysis
- ▶ Use abstract values
- ▶ Allows approximations
- ▶ may lead to false alarm

Abstract interpretation

- ▶ Formalized by Patrick and Radhia Cousot [POPL'77]
- ▶ Give relations between concrete and abstract domains (**Galois connection**)
- ▶ Termination (**widening**)
- ▶ Mixing information from distinct abstractions (**reduced product**)



- ▶ Ensuring termination of the analysis
- ▶ Use abstract values
- ▶ Allows approximations
- ▶ may lead to false alarm

Abstract interpretation

- ▶ Formalized by Patrick and Radhia Cousot [POPL'77]
- ▶ Give relations between concrete and abstract domains (**Galois connection**)
- ▶ Termination (**widening**)
- ▶ Mixing information from distinct abstractions (**reduced product**)

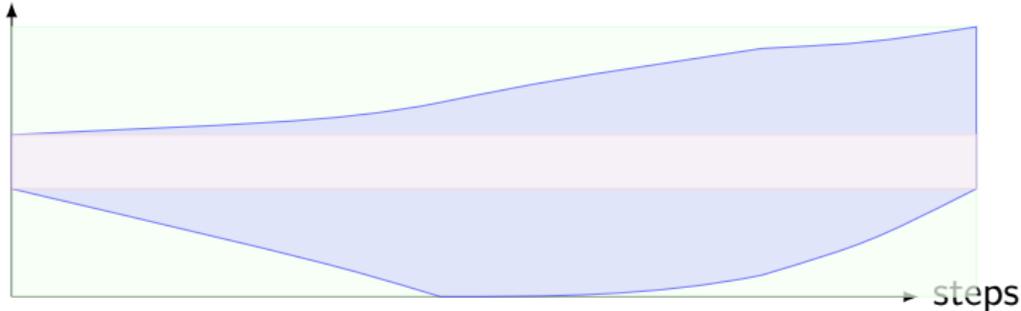


The approximation can be either

correct: All concrete behavior are represented by the abstraction

complete: All abstract behaviors are the representation of a concrete trace
but not both

values

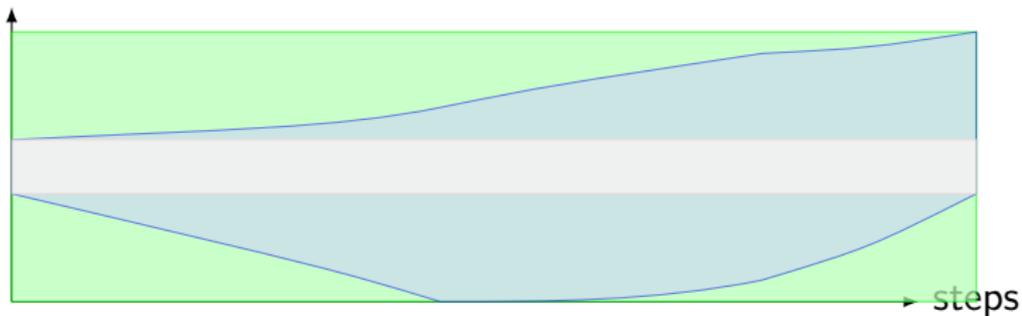


The approximation can be either

correct: All concrete behavior are represented by the abstraction

complete: All abstract behaviors are the representation of a concrete trace
 but not both

values



`(long n; for (k = 0; k < n; k++) tmp1[k] += mc2[k][k] * tmp2[k]);` /* The [k][k] coefficient of the matrix product MC2*TMP2, that is, *MC2*[TMP1] = MC2*[MC1*M1] = MC2*[M1]*MC1
 i.e. tmp1[k][k] += mc2[k][k] * tmp2[k]; */ Final rounding: tmp2[k][k] is now represented on 9 bits. *if (tmp1[k][k] < -256) m2[k][k] = -256; else if (tmp1[k][k] > 255) m2[k][k] = 255; else m2[k][k] = tmp1[k][k];

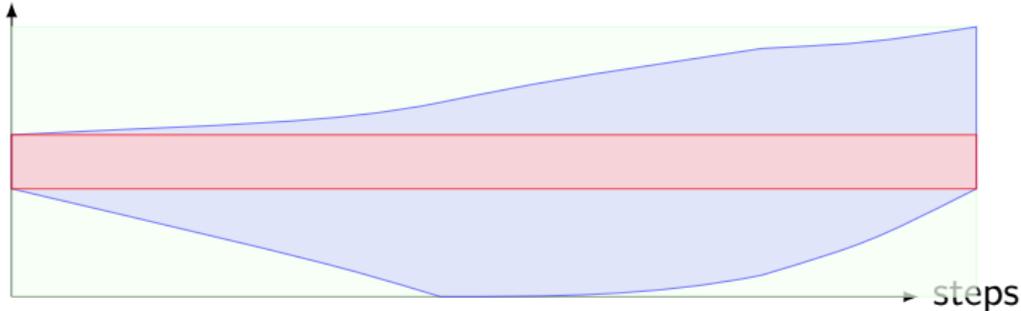
The approximation can be either

correct: All concrete behavior are represented by the abstraction

complete: All abstract behaviors are the representation of a concrete trace

but not both

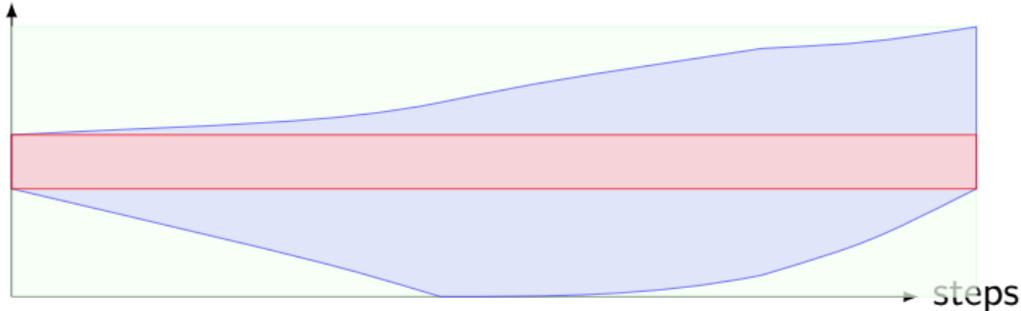
values

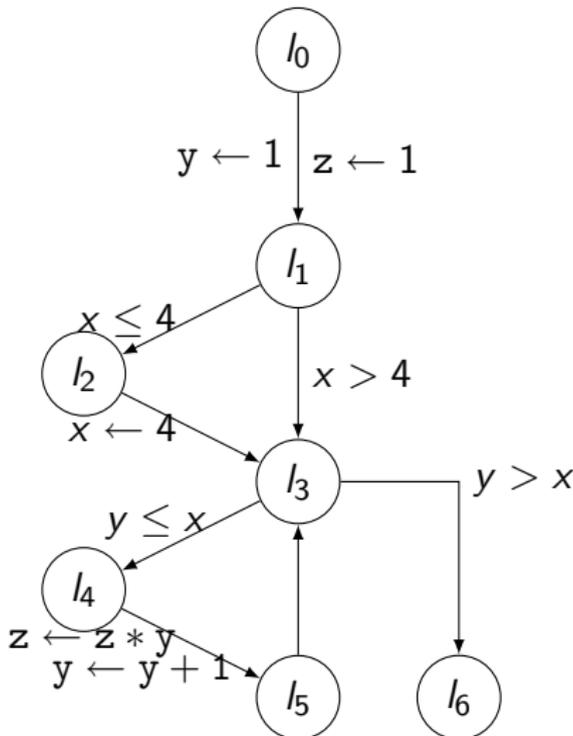
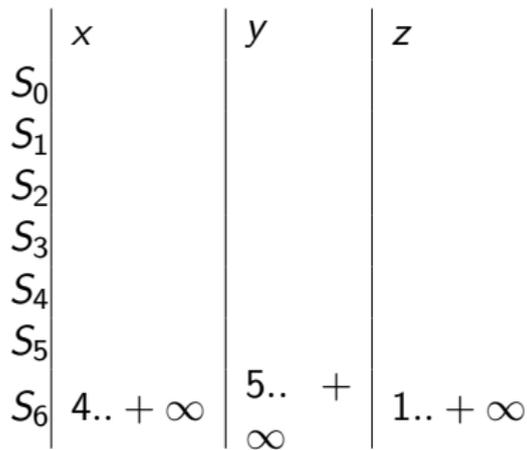


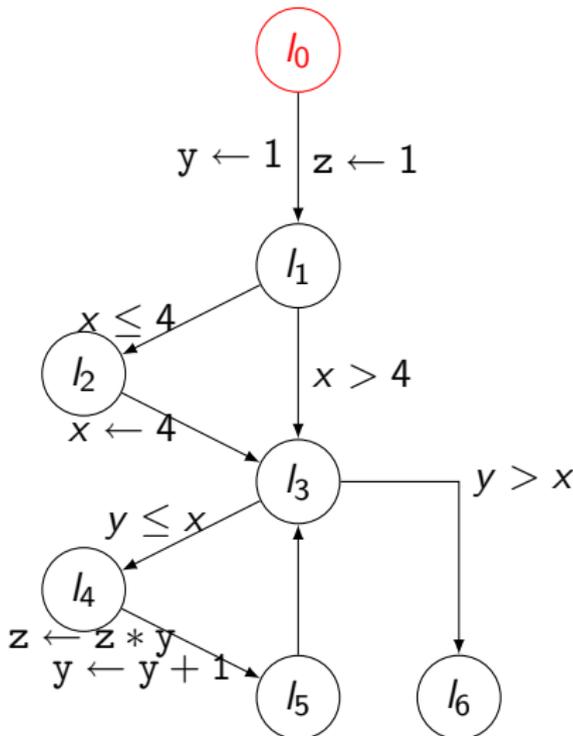
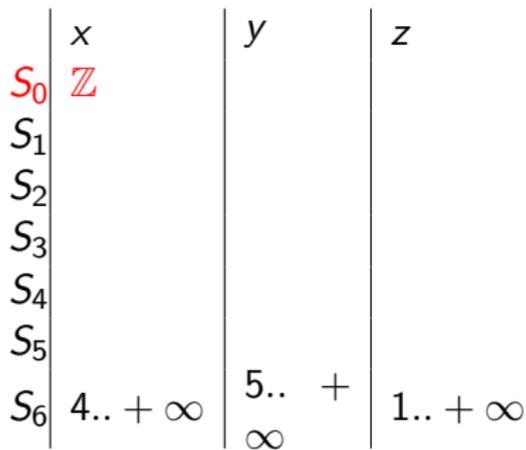
The approximation can be either

- correct:** All concrete behavior are represented by the abstraction
- complete:** All abstract behaviors are the representation of a concrete trace
but not both

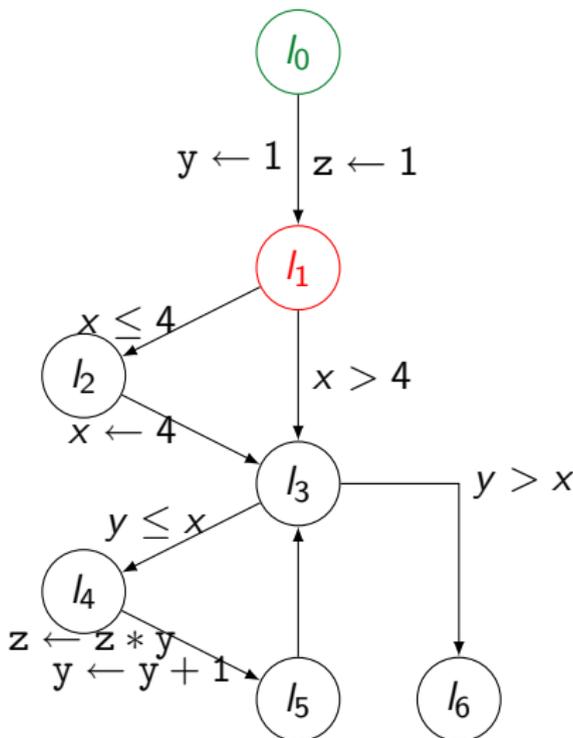
values



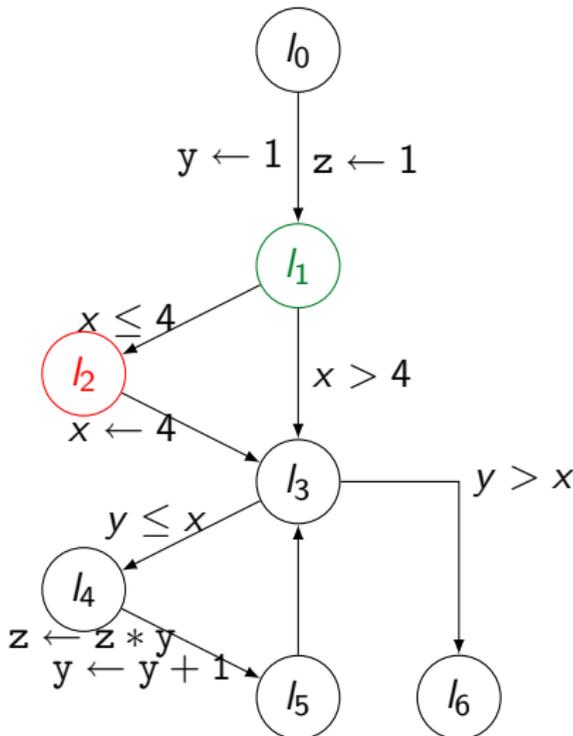




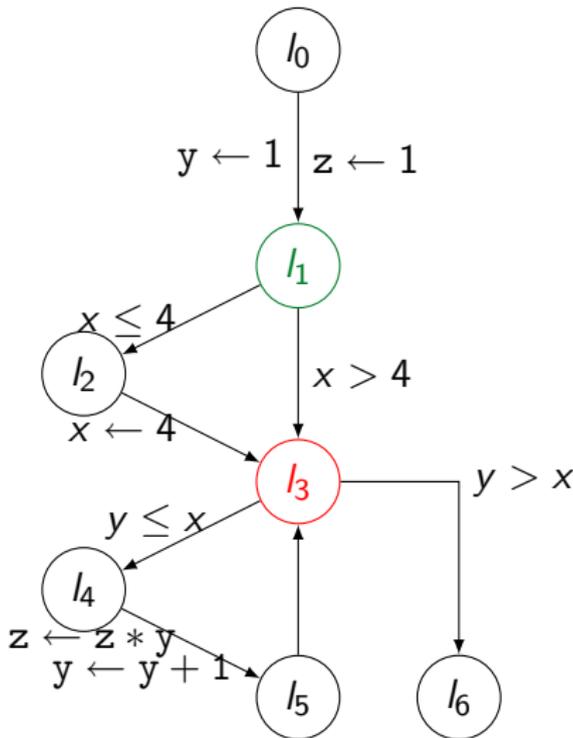
S_0	x	y	z
S_1	\mathbb{Z}	1	1
S_2			
S_3			
S_4			
S_5			
S_6	$4.. + \infty$	$5.. + \infty$	$1.. + \infty$



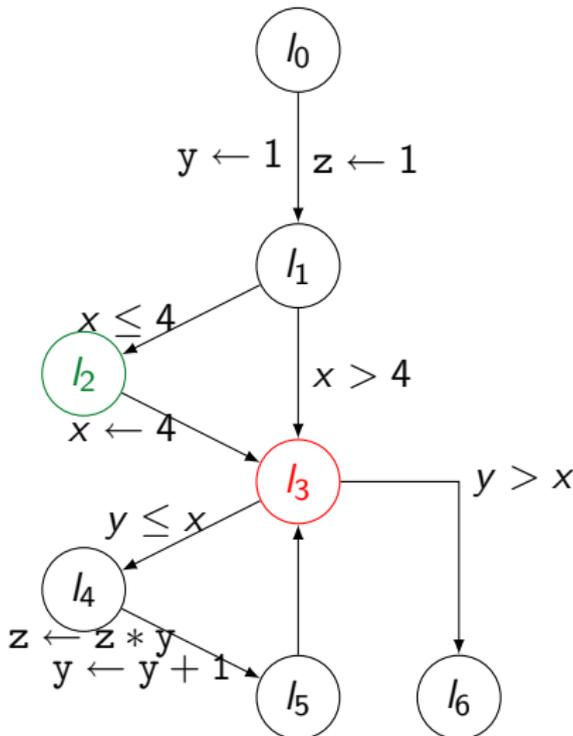
S_0	x	y	z
S_1	\mathbb{Z}	1	1
S_2	$-\infty..4$	1	1
S_3			
S_4			
S_5			
S_6	$4.. + \infty$	$5.. + \infty$	$1.. + \infty$



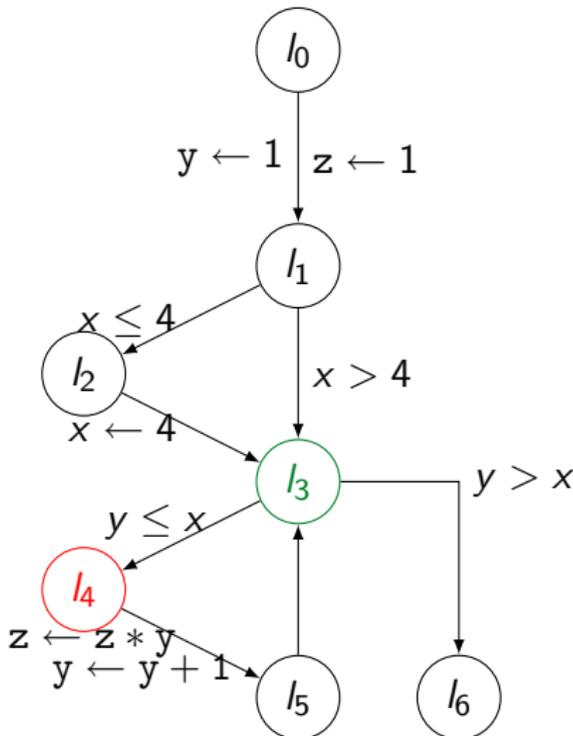
	x	y	z
S_0	\mathbb{Z}		
S_1	\mathbb{Z}	1	1
S_2	$-\infty..4$	1	1
S_3	$4..+\infty$	1	1
S_4			
S_5		$5..+\infty$	
S_6	$4..+\infty$	∞	$1..+\infty$



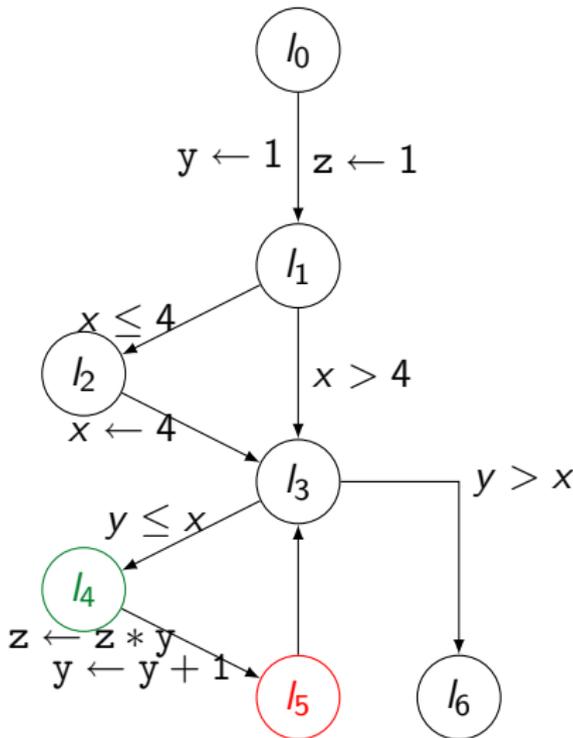
	x	y	z
S_0	\mathbb{Z}		
S_1	\mathbb{Z}	1	1
S_2	$-\infty..4$	1	1
S_3	$4..+\infty$	1	1
S_4			
S_5		$5..+\infty$	
S_6	$4..+\infty$	∞	$1..+\infty$



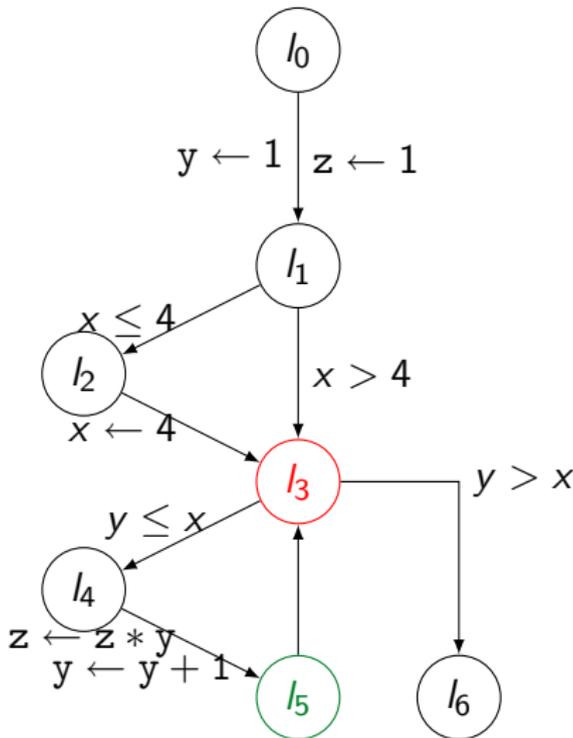
	x	y	z
S_0	\mathbb{Z}		
S_1	\mathbb{Z}	1	1
S_2	$-\infty..4$	1	1
S_3	$4..+\infty$	1	1
S_4	$4..+\infty$	1	1
S_5		$5..+\infty$	
S_6	$4..+\infty$	∞	$1..+\infty$



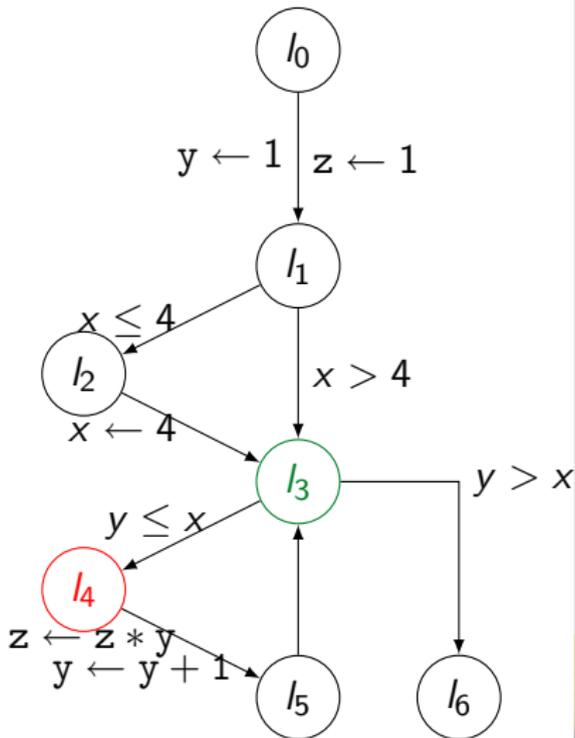
S_0	\mathbb{Z}	y		z	
S_1	\mathbb{Z}	1		1	
S_2	$-\infty..4$	1		1	
S_3	$4..+\infty$	1		1	
S_4	$4..+\infty$	1		1	
S_5	$4..+\infty$	2		1	
S_6	$4..+\infty$	$5..$	$+$	$1..+\infty$	
		∞			



	x	y	z
S_0	\mathbb{Z}		
S_1	\mathbb{Z}	1	1
S_2	$-\infty..4$	1	1
S_3	$4..+\infty$	$1..+\infty$	1
S_4	$4..+\infty$	1	1
S_5	$4..+\infty$	2	1
S_6	$4..+\infty$	$5..+\infty$	$1..+\infty$

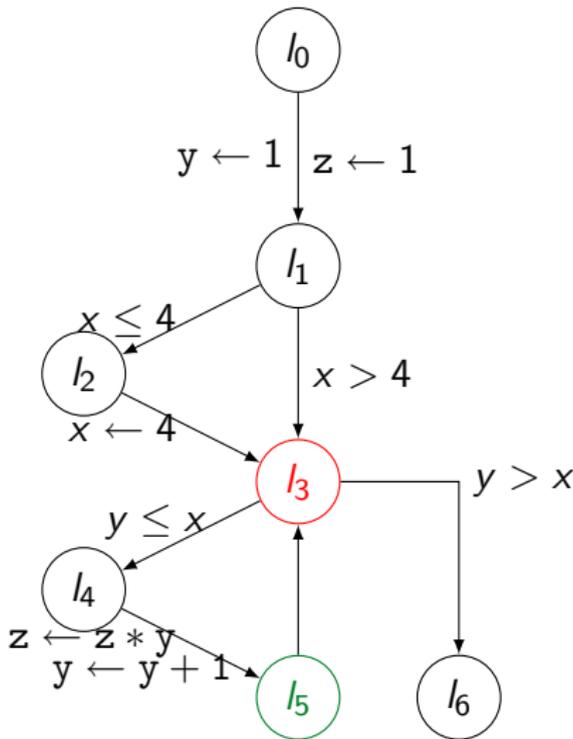


	x	y	z
S_0	\mathbb{Z}		
S_1	\mathbb{Z}	1	1
S_2	$-\infty..4$	1	1
S_3	$4..+\infty$	$1..+\infty$	1
S_4	$4..+\infty$	$1..+\infty$	1
S_5	$4..+\infty$	2	1
S_6	$4..+\infty$	$5..+\infty$	$1..+\infty$

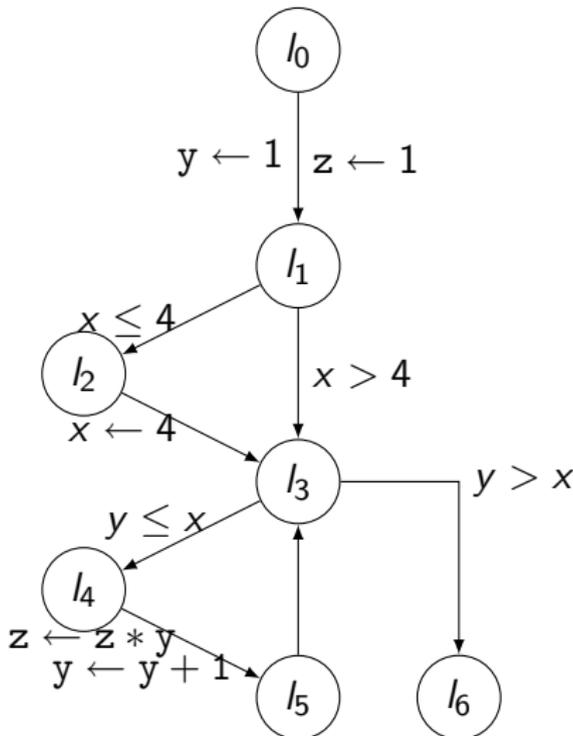


Example: intervals

	x	y	z
S_0	\mathbb{Z}		
S_1	\mathbb{Z}	1	1
S_2	$-\infty..4$	1	1
S_3	$4..+\infty$	$1..+\infty$	$1..+\infty$
S_4	$4..+\infty$	$1..+\infty$	
S_5	$4..+\infty$	$2..+\infty$	$1..+\infty$
S_6	$4..+\infty$	$5..+$	$1..+\infty$
		∞	



	x	y	z
S_0	\mathbb{Z}		
S_1	\mathbb{Z}	1	1
S_2	$-\infty..4$	1	1
S_3	$4..+\infty$	$1..+\infty$	$1..+\infty$
S_4	$4..+\infty$	$1..+\infty$	$1..+\infty$
S_5	$4..+\infty$	$2..+\infty$	$1..+\infty$
S_6	$4..+\infty$	$5..+$	$1..+\infty$
		∞	



Frama-c's integers abstraction uses intervals and modulo information.

- ▶ normal product: $x \in [0; 8] \wedge x \equiv 1[2]$
- ▶ but $0 \not\equiv 1[2]$, so $x \in [1; 7] \wedge x \equiv 1[2]$
- ▶ more generally the **reduced product** of two abstract domains allows to deduce more information than by doing two analyses separately



Frama-c's integers abstraction uses intervals and modulo information.

- ▶ normal product: $x \in [0; 8] \wedge x \equiv 1[2]$
- ▶ but $0 \not\equiv 1[2]$, so $x \in [1; 7] \wedge x \equiv 1[2]$
- ▶ more generally the **reduced product** of two abstract domains allows to deduce more information than by doing two analyses separately



Frama-c's integers abstraction uses intervals and modulo information.

- ▶ normal product: $x \in [0; 8] \wedge x \equiv 1[2]$
- ▶ but $0 \not\equiv 1[2]$, so $x \in [1; 7] \wedge x \equiv 1[2]$
- ▶ more generally the **reduced product** of two abstract domains allows to deduce more information than by doing two analyses separately



Presentation

Data-flow Analysis

Abstract Interpretation

Abstract Interpretation in Practice

long ra
for 0 =>
C1) if (u
tmp2 =
e of the

tmp2[0] = 1 << (Nb1 - 1) else if (tmp1[0]) >= 1 << (Nb1 - 1) tmp2[0] = (1 << (Nb1 - 1) - 1) else tmp2[0] = tmp1[0]; /* Then the second part looks like the first one: */
tmp1[0][k] = 0; k = 0; k++) tmp1[0][k] += mc2[0][k] * tmp2[0][k]; /* The [i,j] coefficient of the matrix product MC2*TMP2, that is, *MC2*(TMP1) = MC2*(M1C1) = MC2*M1*M1C1
i = 1; tmp1[0][i] >= 1; */ Final rounding: tmp2[0][i] is now represented on 9 bits. *if (tmp1[0][i] < -256) m2[0][i] = -256; else if (tmp1[0][i] > 255) m2[0][i] = 255; else m2[0][i] = tm



Frama-C use mainly 3 domains:

- ▶ floating-point values: intervals
- ▶ integral types: intervals and modulo information
- ▶ pointers: set of possible base addresses + an offset (which is an integer).
- ▶ a few other refinements for pointed values



In order to scale to realistic programs (100 kLOC or more), an efficient representation of the state of the program at each point is very important:

- ▶ Maximal sharing of the sub-expressions (hash-consing).
- ▶ Data structures allowing for fast search and insertion: variations over Patricia trees.
- ▶ some improvements of the Ocaml compiler itself have helped a lot.



It is possible to regain some precision (at the expense of the performances) by keeping at most n states separated before merging or widening.

Example

```
int main(int c) {
int x = 0;
int y = 0;
if (c<0) x++;
if (c<0) y++;
if (x == y) { x = y = 42; }
return 0;
}
```

running frama-c



It is possible to regain some precision (at the expense of the performances) by keeping at most n states separated before merging or widening.

Example

```
int main(int c) {
int x = 0;
int y = 0;
if (c<0) x++;
if (c<0) y++;
if (x == y) { x = y = 42; }
return 0;
}
```

$$c \in [-\infty; \infty]$$

$$x \in [0; 1]$$

running frama-c



It is possible to regain some precision (at the expense of the performances) by keeping at most n states separated before merging or widening.

Example

```
int main(int c) {
int x = 0;
int y = 0;
if (c<0) x++;
if (c<0) y++;
if (x == y) { x = y = 42; }
return 0;
}
```

$c \in]-\infty; \infty[$
 $x \in [0; 1]$
 $y \in [0; 1]$

running frama-c



It is possible to regain some precision (at the expense of the performances) by keeping at most n states separated before merging or widening.

Example

```
int main(int c) {
int x = 0;
int y = 0;
if (c<0) x++;
if (c<0) y++;
if (x == y) { x = y = 42; }
return 0;
}
```

$c \in]-\infty; \infty[$
 $x \in [0; 42]$
 $y \in [0; 42]$

running frama-c



It is possible to regain some precision (at the expense of the performances) by keeping at most n states separated before merging or widening.

Example

```
int main(int c) {
int x = 0;
int y = 0;
if (c < 0) x++;
if (c < 0) y++;
if (x == y) { x = y = 42; }
return 0;
}
```

$$c \in] - \infty ; 0 [\cup] 0 ; \infty [$$

$$x = 0 \cup 1$$

running frama-c



It is possible to regain some precision (at the expense of the performances) by keeping at most n states separated before merging or widening.

Example

```
int main(int c) {
int x = 0;
int y = 0;
if (c<0) x++;
if (c<0) y++;
if (x == y) { x = y = 42; }
return 0;
}
```

$c \in]-\infty; 0[\cup]0; \infty[$
 $x = 0 \cup 1$
 $y = 0 \cup 1$

running frama-c



It is possible to regain some precision (at the expense of the performances) by keeping at most n states separated before merging or widening.

Example

```
int main(int c) {
int x = 0;
int y = 0;
if (c<0) x++;
if (c<0) y++;
if (x == y) { x = y = 42; }
return 0;
}
```

$$\begin{array}{l}
 c \in]-\infty; 0[\vee [0; \infty[\\
 x = 42 \vee 42 \\
 y = 42 \vee 42
 \end{array}$$

running frama-c



It is possible to use the results of value analysis to produce more specialized results. This includes currently:

- ▶ semantic constant folding
- ▶ inputs and outputs of a function
- ▶ slicing
- ▶ impact analysis



```

long n;
for (i = 0; i < n; i++)
  tmp2 = ...
  of the
tmp2[i] = (i < (n-1) ? tmp1[i] : 0) + (i < (n-1) ? tmp1[i] : 0) + (i < (n-1) ? tmp1[i] : 0) + (i < (n-1) ? tmp1[i] : 0);
tmp1[i] = 0; k = 5; k--); tmp1[i] = m2[i][k] * tmp2[k]; /* The [i][k] coefficient of the matrix product M2*TMP2, that is, *MC2*(TMP1) = MC2*(MC1*M1) = MC2*M1 *MC1
= 1 * tmp1[i] >= 1; */ Final rounding: tmp2[i] is now represented on 3 bits: if (tmp1[i] < -256) m2[i] = -256; else if (tmp1[i] > 255) m2[i] = 255; else tmp2[i] =
  
```