Strings & Software Model Checking

Philipp Rümmer
Uppsala University

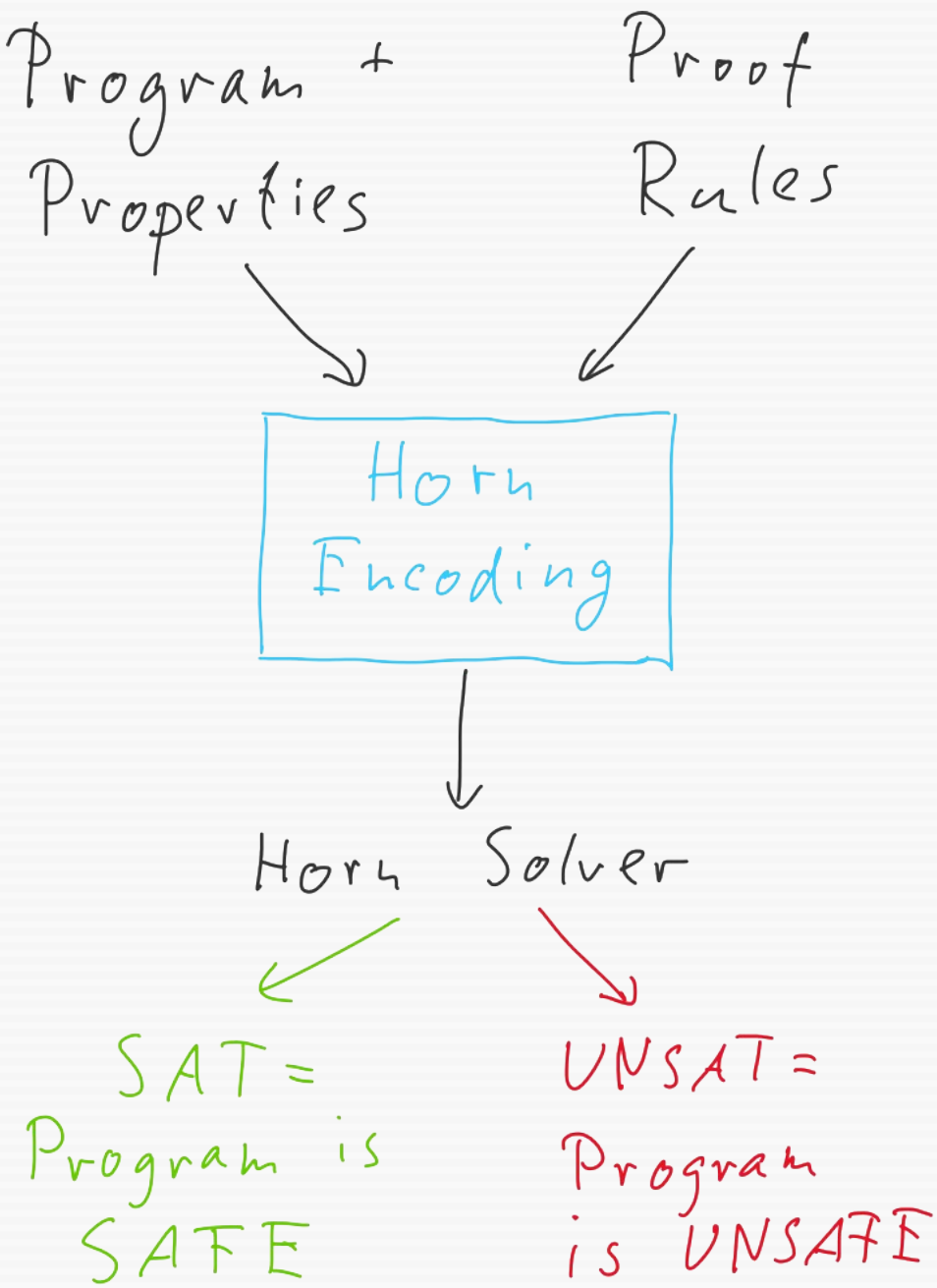
30 August 2019
Taipei, Taiwan

Outline

- Constrained Horn Clauses
- JayHorn Architecture
- JayHorn Approach to Handling Heap
- Demos & Examples

- Decision Procedures for Strings
- Strings in Software Model Checking

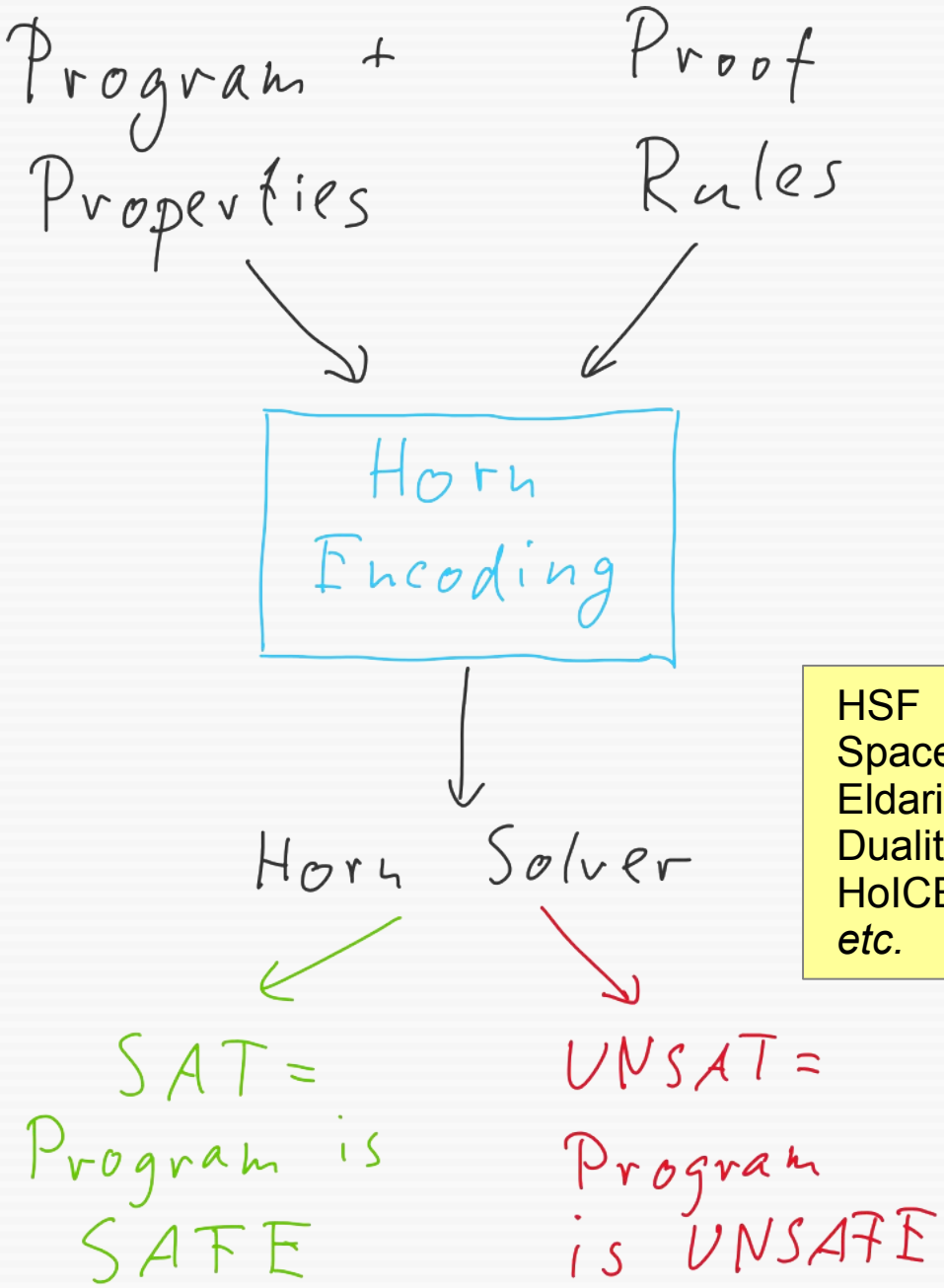
Verification Engines



Verification Engines

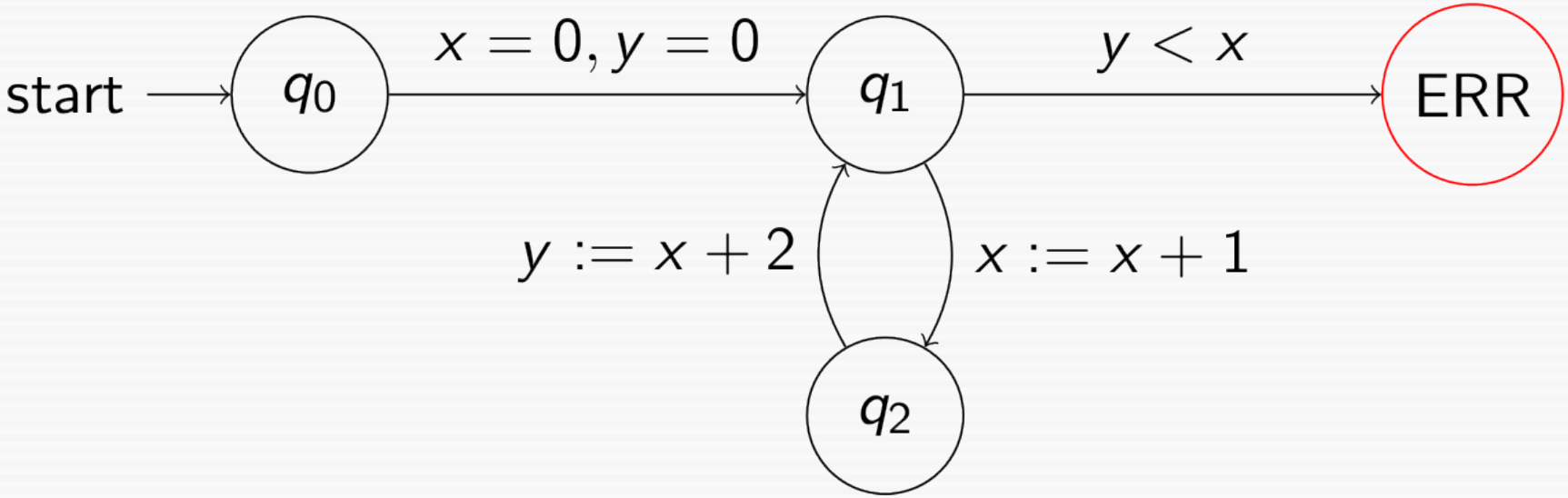
Software programs
Networks of timed automata
BIP models
etc.

Floyd-Hoare
Design by contract
Owicki-Gries
Rely Guarantee
etc.

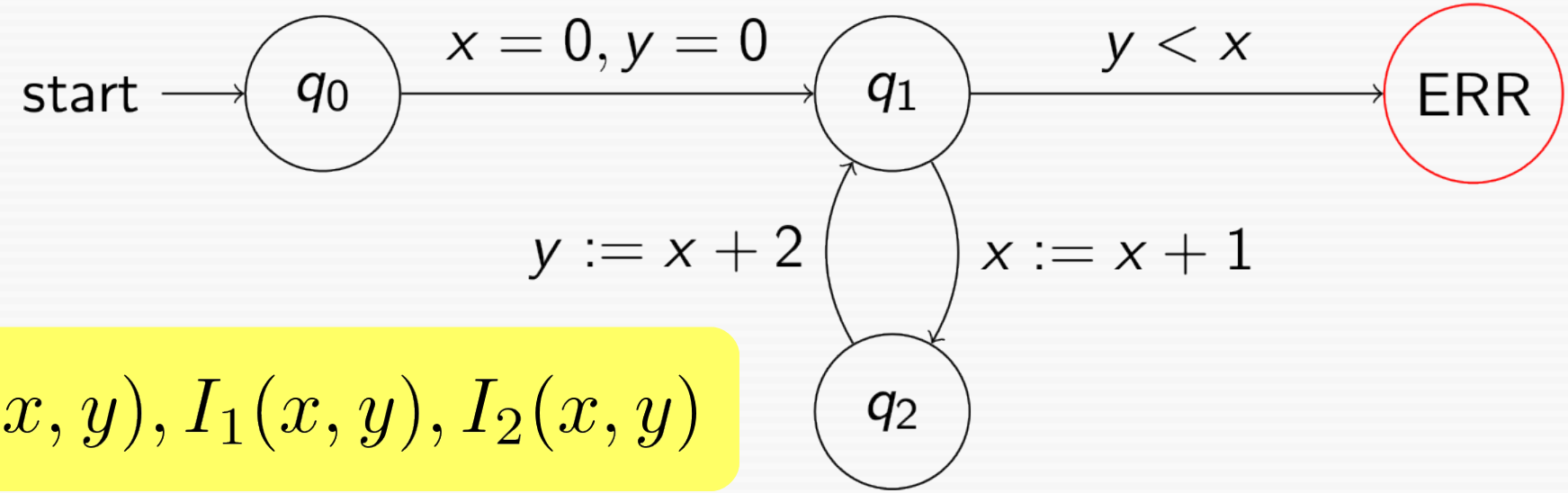


HSF
Spacer
Eldarica
Duality
HoICE
etc.

Ex 1: Floyd-style invariants

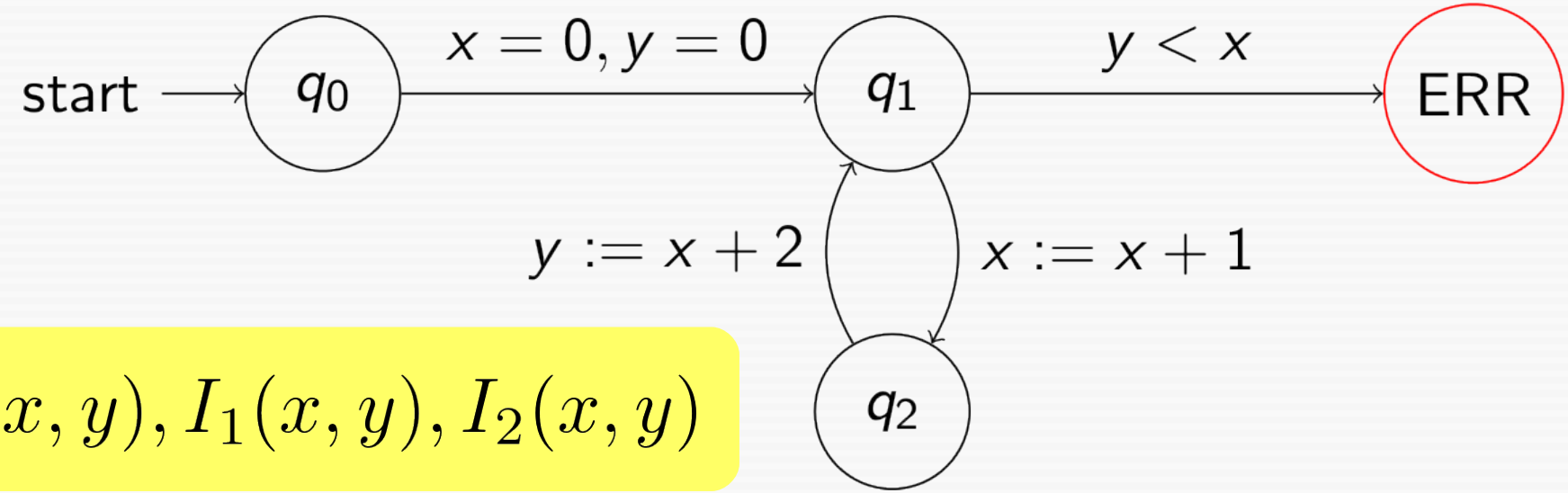


Ex 1: Floyd-style invariants



$$I_0(x, y), I_1(x, y), I_2(x, y)$$

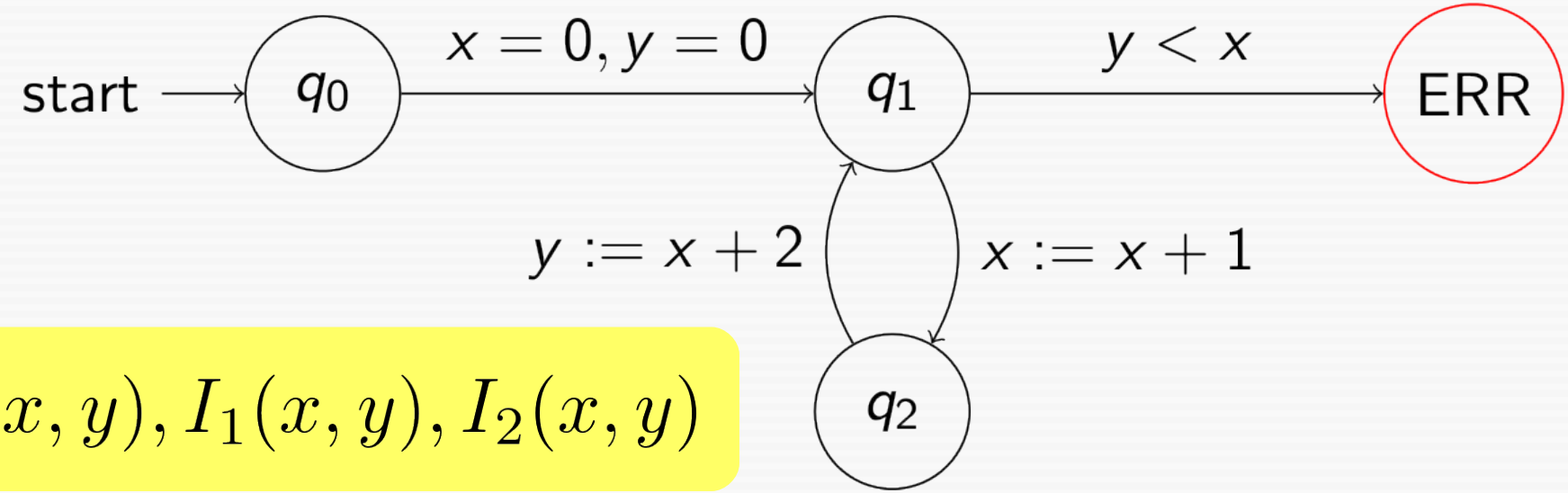
Ex 1: Floyd-style invariants



$I_0(x, y), I_1(x, y), I_2(x, y)$

- When the program is in q_0, $I_0(x, y)$ holds

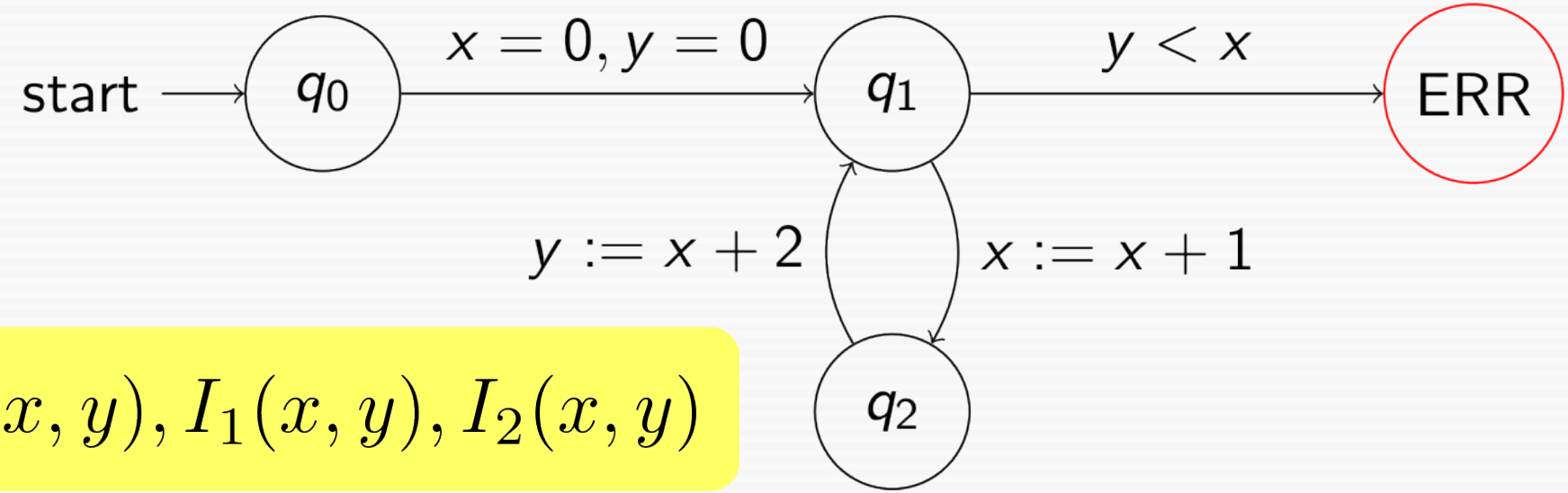
Ex 1: Floyd-style invariants



$I_0(x, y), I_1(x, y), I_2(x, y)$

- When the program is in q_0, $I_0(x, y)$ holds
- When the program is in q_0 and $I_0(x, y)$ holds, then after transition to q_1 the formula $I_1(x, y)$ holds

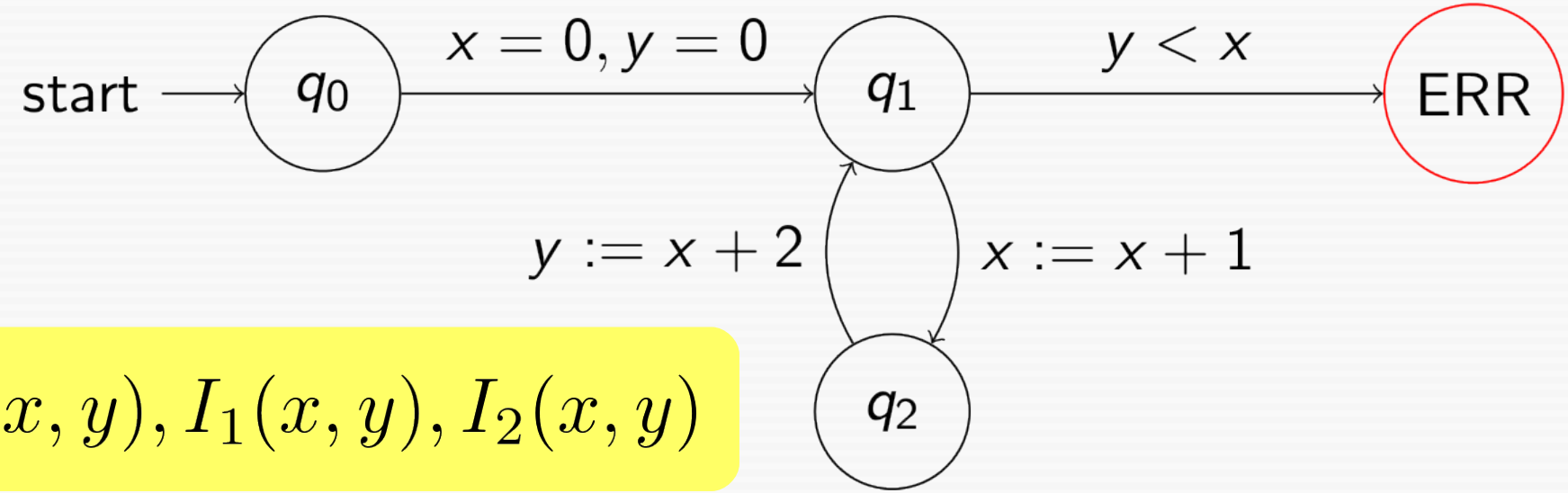
Ex 1: Floyd-style invariants



$I_0(x, y), I_1(x, y), I_2(x, y)$

- When the program is in q_0, $I_0(x, y)$ holds
- When the program is in q_0 and $I_0(x, y)$ holds, then after transition to q_1 the formula $I_1(x, y)$ holds
- etc.

Ex 1: Floyd-style invariants



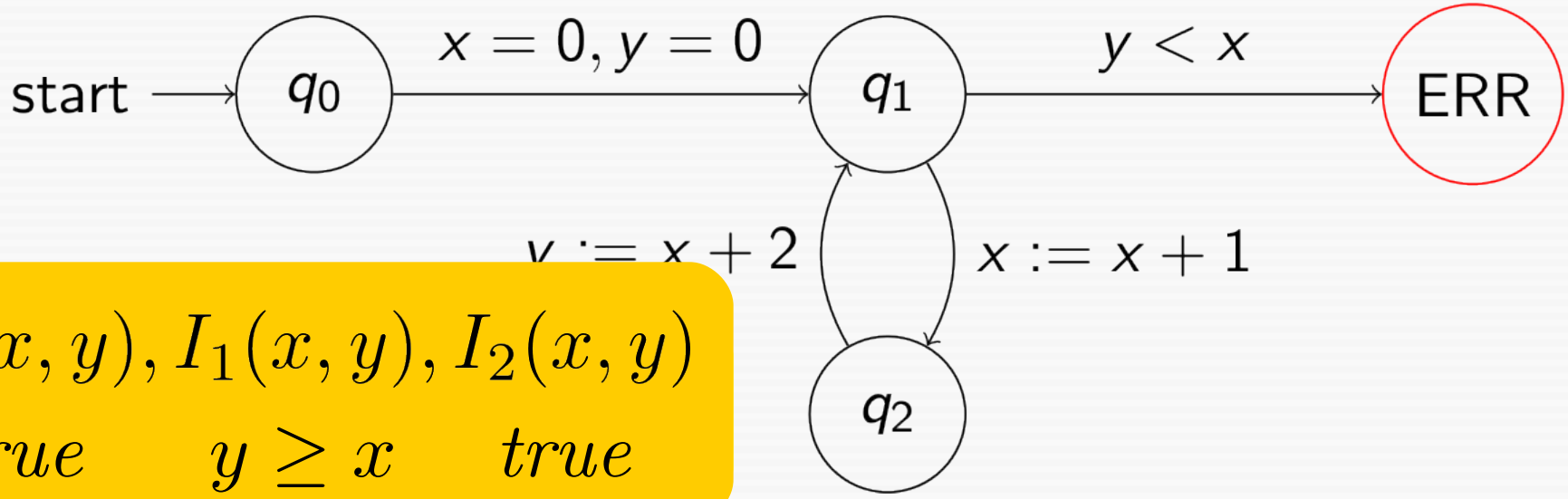
$I_0(x, y), I_1(x, y), I_2(x, y)$

- When th … $y)$ holds
- When th … $I_0(x, y)$ holds, t … $_1$ the formula
- etc.

Constraints:

$\forall x, y.\ true \rightarrow I_0(x, y)$
$\forall x, y.\ I_0(x, y) \rightarrow I_1(0, 0)$
$\forall x, y.\ I_1(x, y) \rightarrow I_2(x + 1, y)$
$\forall x, y.\ I_2(x, y) \rightarrow I_1(x, x + 2)$
$\forall x, y.\ I_1(x, y) \wedge y < x \rightarrow false$

Ex 1: Floyd-style invariants



$I_0(x,y)$	$I_1(x,y)$	$I_2(x,y)$
$true$	$y \geq x$	$true$

- When th … $y)$ holds
- When th … $I_0(x,y)$ holds, t … $_1$ the formula
- etc.

Constraints:

$\forall x, y.\ true \rightarrow I_0(x,y)$
$\forall x, y.\ I_0(x,y) \rightarrow I_1(0,0)$
$\forall x, y.\ I_1(x,y) \rightarrow I_2(x+1,y)$
$\forall x, y.\ I_2(x,y) \rightarrow I_1(x,x+2)$
$\forall x, y.\ I_1(x,y) \wedge y < x \rightarrow false$

In Machine-Readable Format

```
(set-logic HORN)

(declare-fun I0 (Int Int) Bool)
(declare-fun I1 (Int Int) Bool)
(declare-fun I2 (Int Int) Bool)

(assert (forall ((x Int) (y Int)) (I0 x y)))
(assert (forall ((x Int) (y Int)) (=> (I0 x y) (I1 0 0))))
(assert (forall ((x Int) (y Int)) (=> (I1 x y) (I2 (+ x 1) y))))
(assert (forall ((x Int) (y Int)) (=> (I2 x y) (I1 x (+ x 2)))))
(assert (forall ((x Int) (y Int)) (=> (and (I1 x y) (< y x)) false)))

(check-sat)
(get-model)
```

In Machine-Readable Format

```
(set-logic HORN)

(declare-fun I0 (Int Int) Bool)
(declare-fun I1 (Int Int) Bool)
(declare-fun I2 (Int Int) Bool)

(assert (forall ((x Int) (y Int)) (I0 x y)))
(assert (forall ((x Int) (y Int)) (=> (I0 x y) (I1 0 0))))
(assert (forall ((x Int) (y Int)) (=> (I1 x y) (I2 (+ x 1) y))))
(assert (forall ((x Int) (y Int)) (=> (I2 x y) (I1 x (+ x 2)))))
(assert (forall ((x Int) (y Int)) (=> (and (I1 x y) (< y x)) false)))

(check-sat)
(get-model)
```

```
i0(X, Y)    :- 1=1.
i1(X', Y') :- i0(X, Y), X'=0, Y'=0.
i2(X', Y)  :- i1(X, Y), X'=X+1.
i1(X, Y')  :- i2(X, Y), Y'=X+2.
false      :- i1(X, Y), Y < X.
```

More formally ...

Definition
Suppose
- $\mathcal{L}$ is some constraint language (e.g., integers);
- $\mathcal{R}$ is a set of relation symbols;
- $\mathcal{X}$ is a set of first-order variables.

Then a *Horn clause* is a formula $C \wedge B_1, \ldots, B_n \rightarrow H$ where
- C is a constraint in $\mathcal{L}$ (without symbols from $\mathcal{R}$);
- each B_i is a literal of the form $r(t_1, \ldots, t_m)$;
- H is either $false$, or of the same form as the B_i.

More formally ...

Definition
Suppose
- $\mathcal{L}$ is some constraint language (e.g., integers);
- $\mathcal{R}$ is a set of relation symbols;
- $\mathcal{X}$ is a set of first-order variables.

Then a *Horn clause* is a formula $C \wedge B_1, \ldots, B_n \rightarrow H$ where

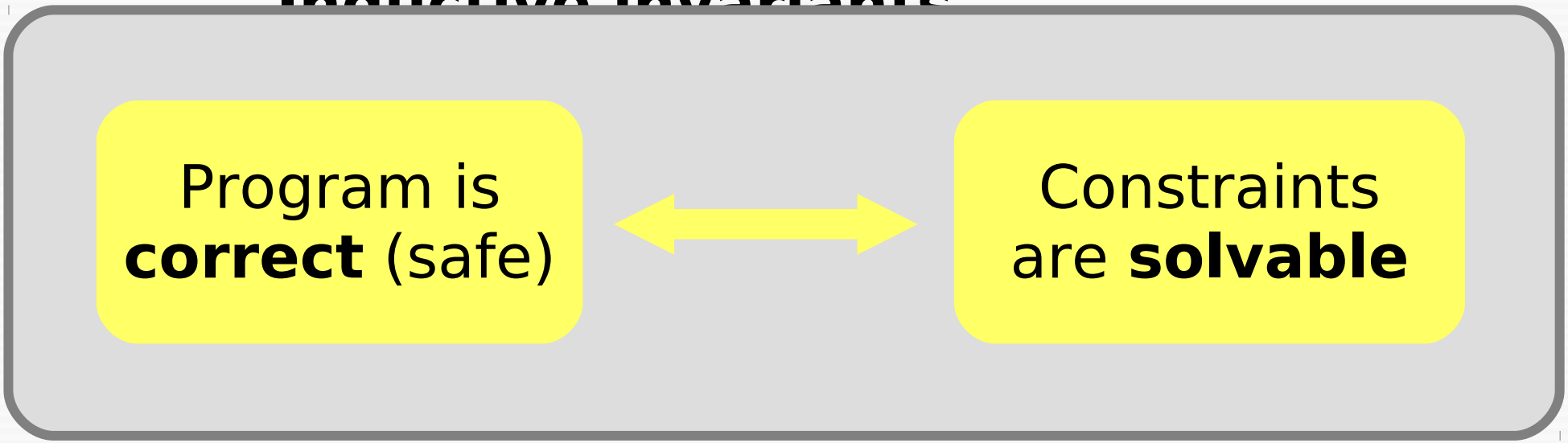
Definition
- A set $\mathcal{C}$ of Horn clauses is *(syntactically)*
- *solvable* if the $\mathcal{R}$ -symbols can be replaced with formulae such that all clauses become valid.

General Encoding Strategy

- Choose a set of relation symbols representing
  Inductive invariants
- for instance
  one per control location
  one per process
  one per module
  one per class/class instance
- Add **constraints** on inductive invariants:
  initiation, consecution
- Add safety constraints:
  invariants exclude **error states**

General Encoding Strategy

- Choose a set of relation symbols representing
  Inductive invariants



- Add **constraints** on inductive invariants:
  initiation, consecution
- Add safety constraints:
  invariants exclude **error states**

Ex 2: Functions

$$f(x) = \begin{cases} x - 10, & \text{if } x > 100 \\ f(f(x+11)), & \text{if } x \leq 100 \end{cases}$$

Ex 2: Functions

$$f(x) = \begin{cases} x - 10, & \text{if } x > 100 \\ f(f(x+11)), & \text{if } x \leq 100 \end{cases}$$

Verify $x \leq 100 \rightarrow f(x) = 91$

Ex 2: Functions

$$f(x) = \begin{cases} x - 10, & \text{if } x > 100 \\ f(f(x+11)), & \text{if } x \leq 100 \end{cases}$$

Verify $x \leq 100 \rightarrow f(x) = 91$

Partial correctness

Ex 2: Functions

$$f(x) = \begin{cases} x - 10, & \text{if } x > 100 \\ f(f(x+11)), & \text{if } x \leq 100 \end{cases}$$

Verify $x \leq 100 \rightarrow f(x) = 91$

```
int f(int x) {
  if (x > 100) {
    int t0 = x - 10;
    return t0;
  } else {
    int t0 = x + 11;
    int t1 = f(t0);
    int t2 = f(t1);
    return t2;
  }
}
```

Partial correctness

Full Encoding

```
i0(X0, X)       :- X0=X.                          % int f(int x) {
i1(X0, X)       :- i0(X0, X), X > 100.            %   if (x > 100) {
i2(X0, T0)      :- i1(X0, X), T0=X-10.            %     int t0 = x - 10;
post_f(X0, T0)  :- i2(X0, T0).                    %     return t0;
i3(X0, X)       :- i0(X0, X), X =< 100.           %   } else {
i4(X0, T0)      :- i3(X0, X), T0=X+11.            %     int t0 = x + 11;
i5(X0, T1)      :- i4(X0, T0), post_f(T0, T1).    %     int t1 = f(t0);
i6(X0, T2)      :- i5(X0, T1), post_f(T1, T2).    %     int t2 = f(t1);
post_f(X0, T2)  :- i6(X0, T2).                    %     return t2;
                                                  %   }
                                                  % }

false :- post_f(X, R), X =< 100, \+(R = 91).  % Assertion
```

Ex 3: Concurrency

```
int N;

int i = 0, x = 1;
assume(N > 0);

while (i < N) {
  x *= 2;
  ++i;
}

assert(x > 1);
```

Ex 3: Concurrency

```
int N;

int i = 0, x = 1;
assume(N > 0);

while (i < N) {
  x *= 2;
  ++i;
}

assert(x > 1);
```

Loop invariant:

`x >= 2 | (x = 1 & i < N)`

Ex 3: Concurrency

```
int N;
```

```
int i = 0, x = 1;
assume(N > 0);

while (i < N) {
  x *= 2;
  ++i;
}

assert(x > 1);
```

Loop invariant:

`x >= 2 | (x = 1 & i < N)`

```
while (1) {
  N++;
}
```

Ex 3: Concurrency

```
int N;
```

```
int i = 0, x = 1;
assume(N > 0);

while (i < N) {
  x *= 2;
  ++i;
}

assert(x > 1);
```

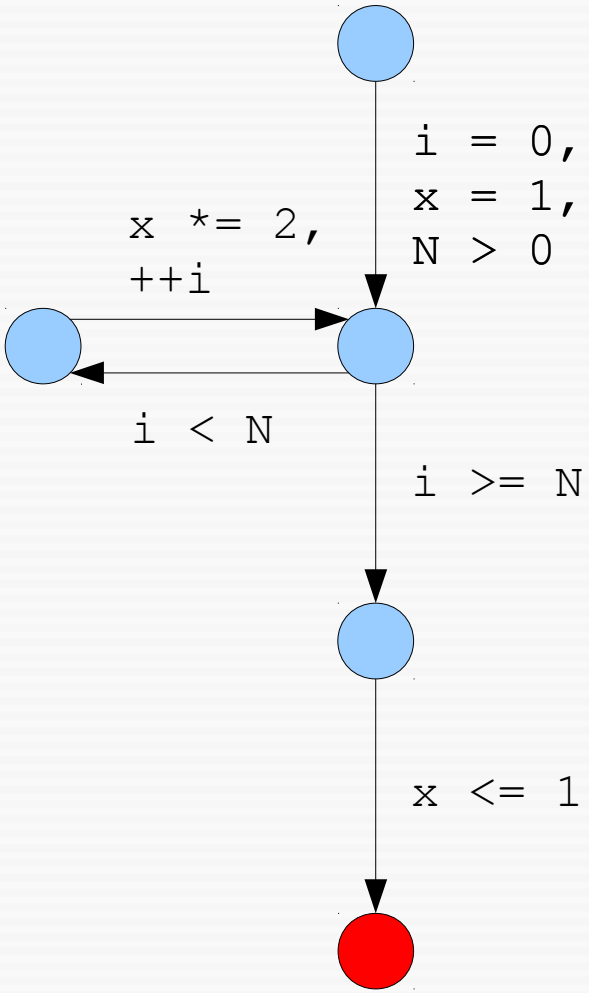
Loop invariant:

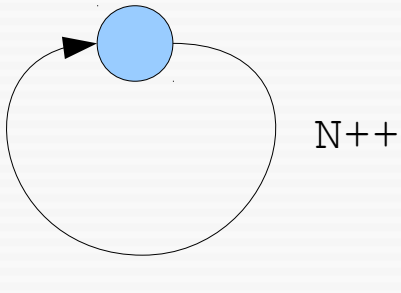
`x >= 2 | (x = 1 & i < N)`

```
while (1) {
  N++;
}
```

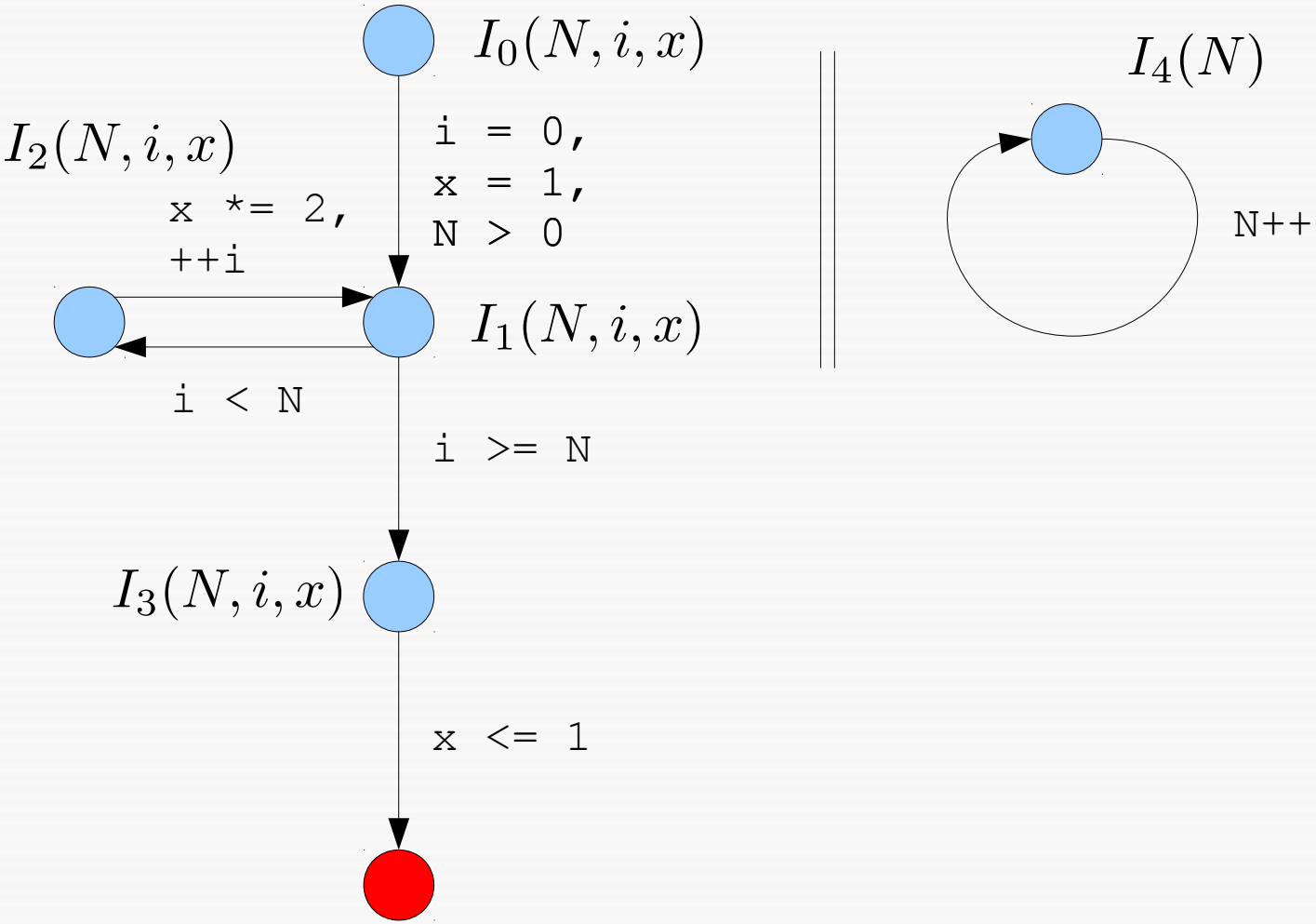
Can invariant still be used?

Owicki-Gries

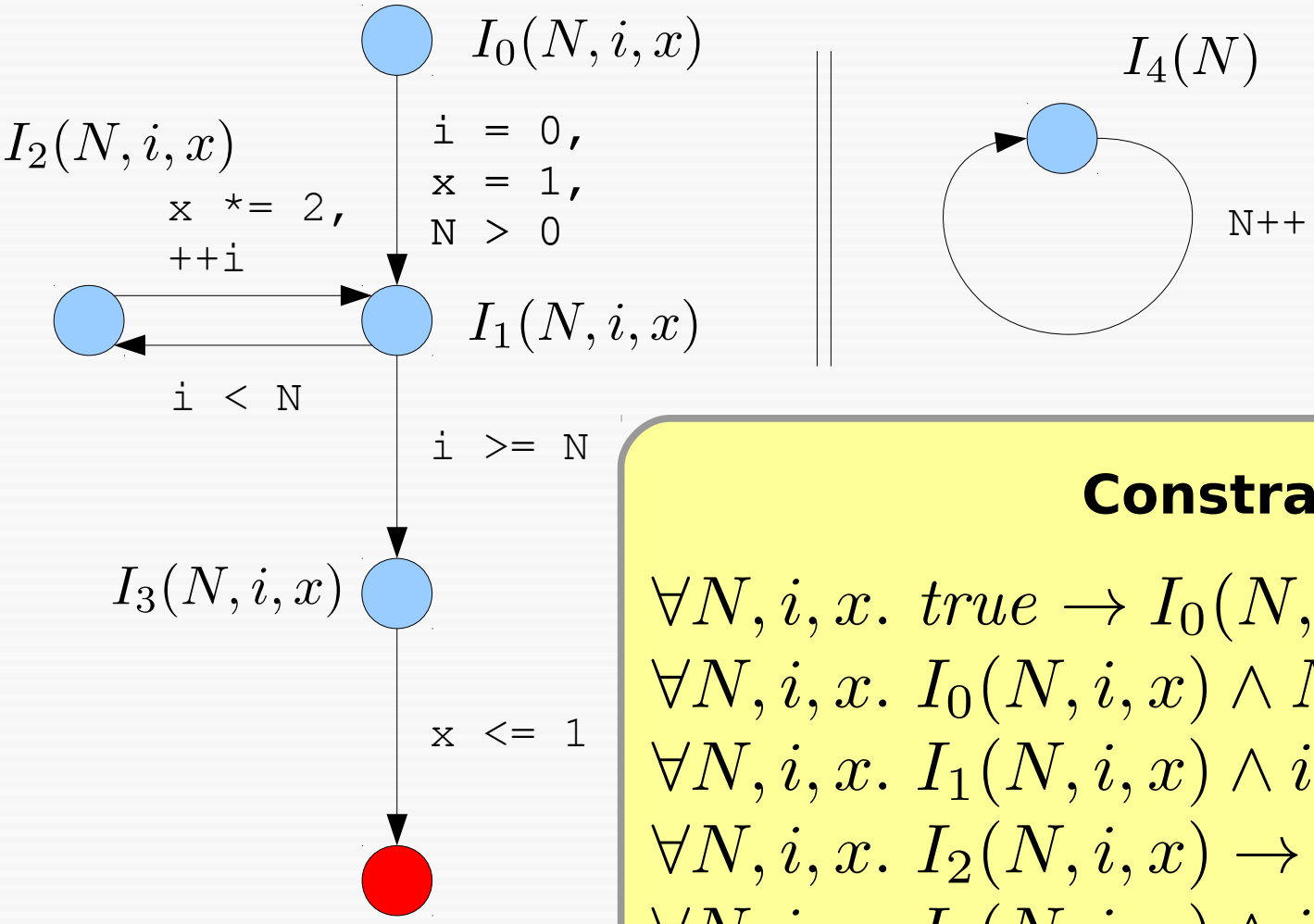




Owicki-Gries



Owicki-Gries

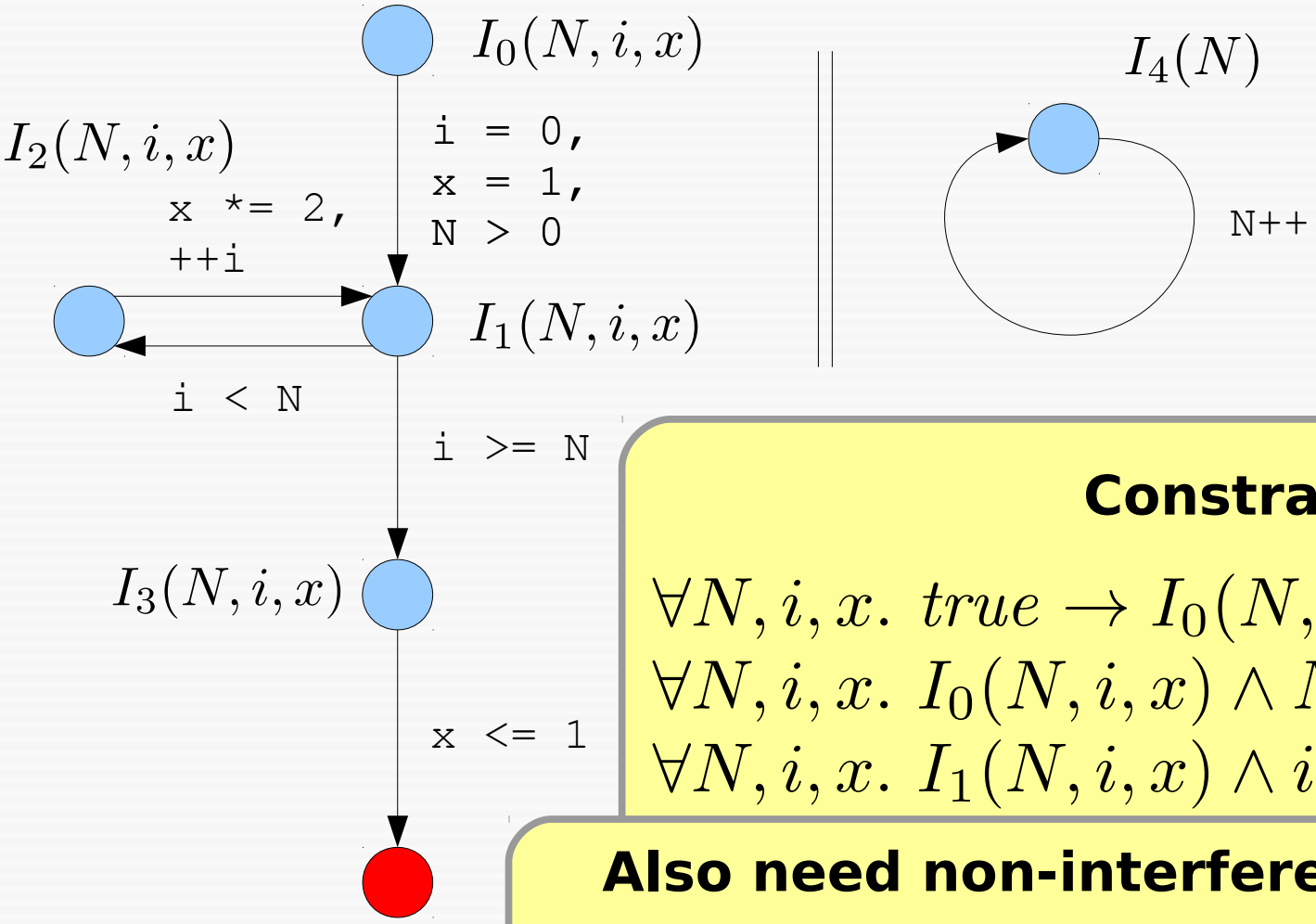


Constraints:

$$\forall N, i, x.\ \mathit{true} \rightarrow I_0(N,i,x)$$
$$\forall N, i, x.\ I_0(N,i,x) \wedge N > 0 \rightarrow I_1(N,0,1)$$
$$\forall N, i, x.\ I_1(N,i,x) \wedge i < N \rightarrow I_2(N,i,x)$$
$$\forall N, i, x.\ I_2(N,i,x) \rightarrow I_1(N,i+1,2x)$$
$$\forall N, i, x.\ I_1(N,i,x) \wedge i \geq N \rightarrow I_3(N,i,x)$$
$$\forall N, i, x.\ I_3(N,i,x) \wedge x \leq 1 \rightarrow \mathit{false}$$

$$\forall N.\ I_4(N) \rightarrow I_4(N+1)$$

Owicki-Gries



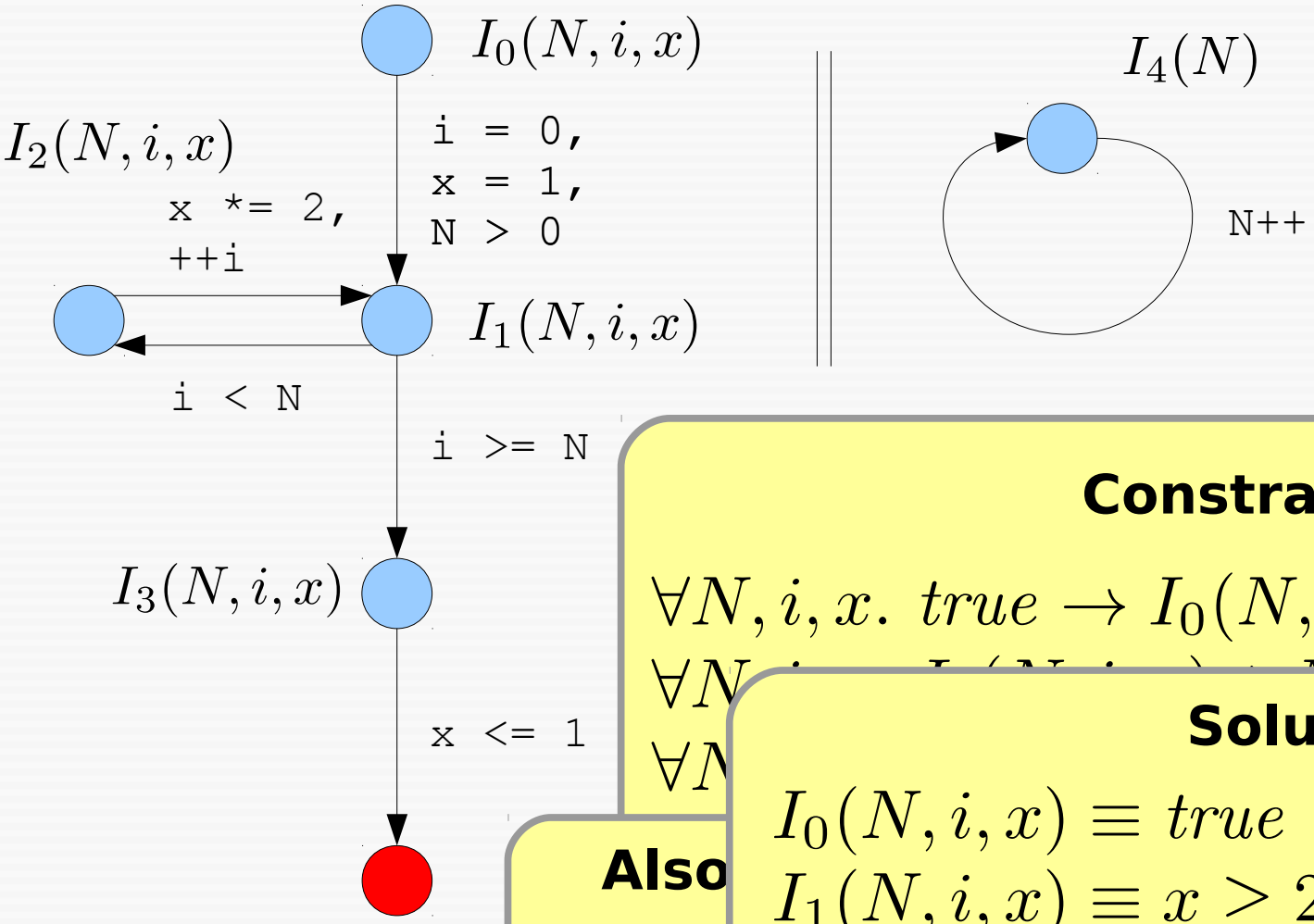
Constraints:

$$\forall N, i, x.\ true \rightarrow I_0(N,i,x)$$
$$\forall N, i, x.\ I_0(N,i,x) \wedge N > 0 \rightarrow I_1(N,0,1)$$
$$\forall N, i, x.\ I_1(N,i,x) \wedge i < N \rightarrow I_2(N,i,x)$$

Also need non-interference constraints!

$$\forall N, i, x.\ I_0(N,i,x) \wedge I_4(N) \rightarrow I_0(N+1,i,x)$$
$$\forall N, i, x.\ I_1(N,i,x) \wedge I_4(N) \rightarrow I_1(N+1,i,x)$$
$$\ldots$$

Owicki-Gries



Constraints:

$\forall N, i, x.\ true \rightarrow I_0(N,i,x)$

$\forall N$

$\forall N$

Also

$\forall N, i,$

$\forall N, i,$

$\cdots$

Solution:

$I_0(N,i,x) \equiv true$

$I_1(N,i,x) \equiv x \geq 2 \vee (x = 1 \wedge i < N)$

$I_2(N,i,x) \equiv x \geq 1 \wedge i < N$

$I_3(N,i,x) \equiv x \geq 2$

$I_4(N,i,x) \equiv true$

Other Horn Encodings

- **Owicki-Gries**
- **Rely-guarantee**
- **Various forms of thread communication**
- **Parameterised systems**
- **Synchronous programs**
- **Timed systems**
- **Regression verification**
- **Games**
- **Networks, SDN**
- …

CHC-COMP 2018

Arie Gurfinkel

Philipp Ruemmer, Grigory Fedyukovich, Adrien Champion

1st Competition on Solving Constrained Horn Clauses

UNIVERSITY OF WATERLOO

Report on the Second Edition of the CHC Competition

Grigory Fedyukovich

April 7, Prague

Results: **LIA-Nonlin**

Solver	Score	#SAT	#UNSAT	Avg time
Spacer	**270**	153	117	5.04
Eldarica	**234**	131	103	15.93
Ultimate Unihorn Automizer	**177**	96	81	36.94
Hoice	**176**	110	66	9.85
PCSat	**123**	81	42	24.69
Ultimate Tree Automizer	**73**	29	44	4.85

* **283** instances total

Data Structures and Heap?

Data Structures and Heap?

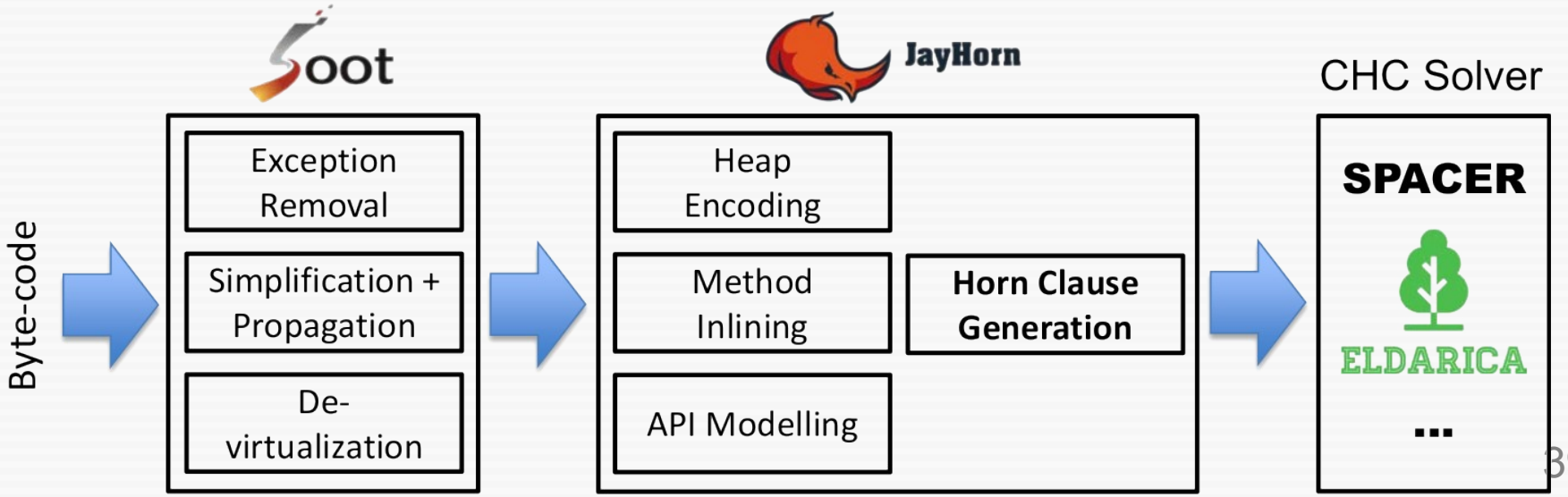
- Encoding using McCarthy Arrays
  - Precise, relatively complete
  - Hard to infer invariants automatically
- Refinement types, etc.
  - Incomplete
  - Easier to automate
- (Separation logic, ownership systems, dynamic frames, etc.)

Data Structures and Heap?

- Encoding using McCarthy Arrays
  - Precise, relatively complete
  - Hard to infer invariants automatically
- Refinement types, etc.
  - Incomplete
  - Easier to automate
- (Separation logic, ownership systems, dynamic frames, etc.)

JayHorn

- A Horn-based verification tool for Java
- Fully implemented in Java
- Fully automatic
- Open source, MIT licence



Demo

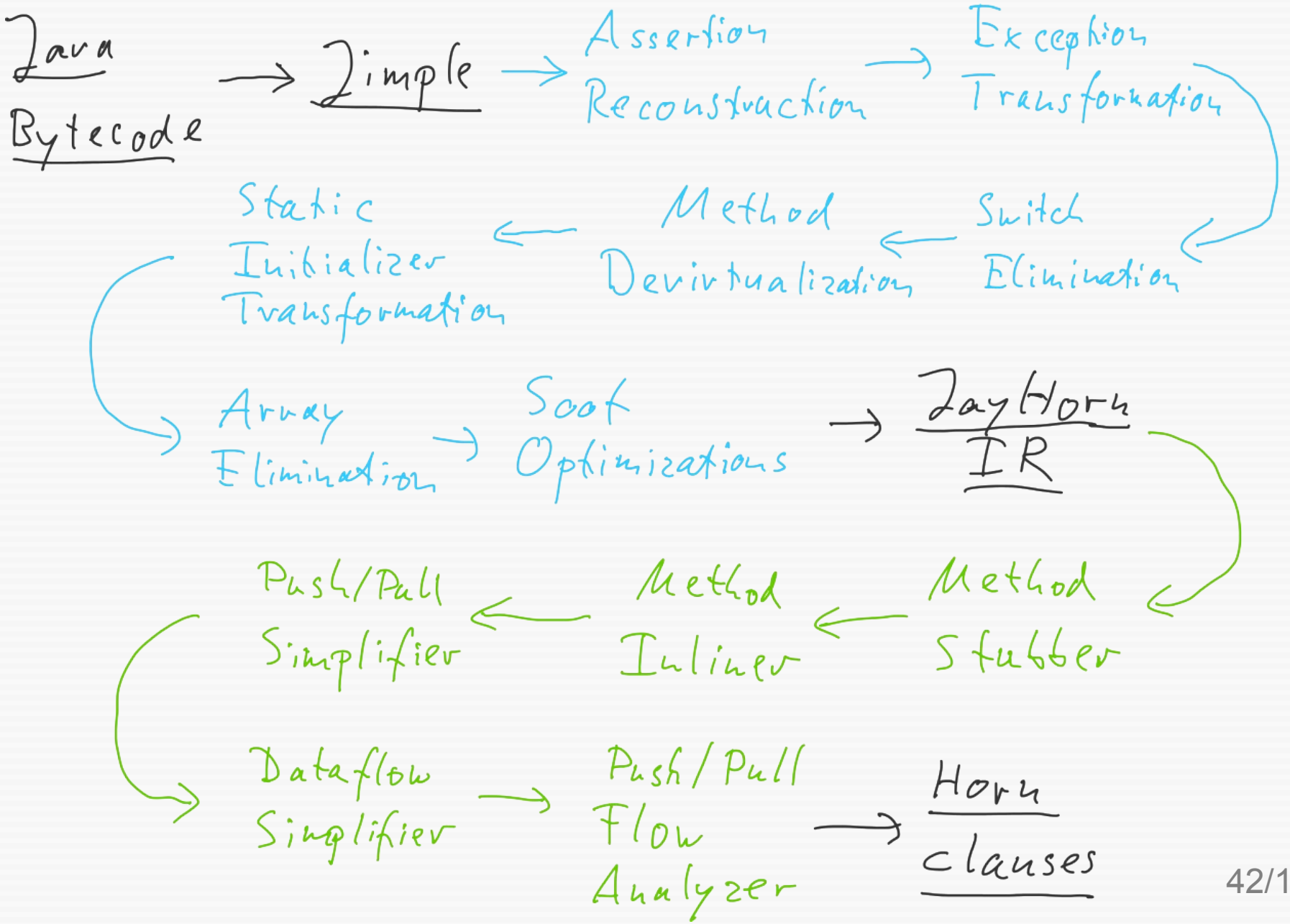
```
import org.sosy_lab.sv_benchmarks.Verifier;

public class McCarthy91 {
    private static int f(int n) {
        if (n > 100)
            return n - 10;
        else
            return f(f(n + 11));
    }

    public static void main(String[] args) {
        int x = Verifier.nondetInt();
        int y = f(x);
        assert(x > 101 || y == 91);
    }
}
```

Demo

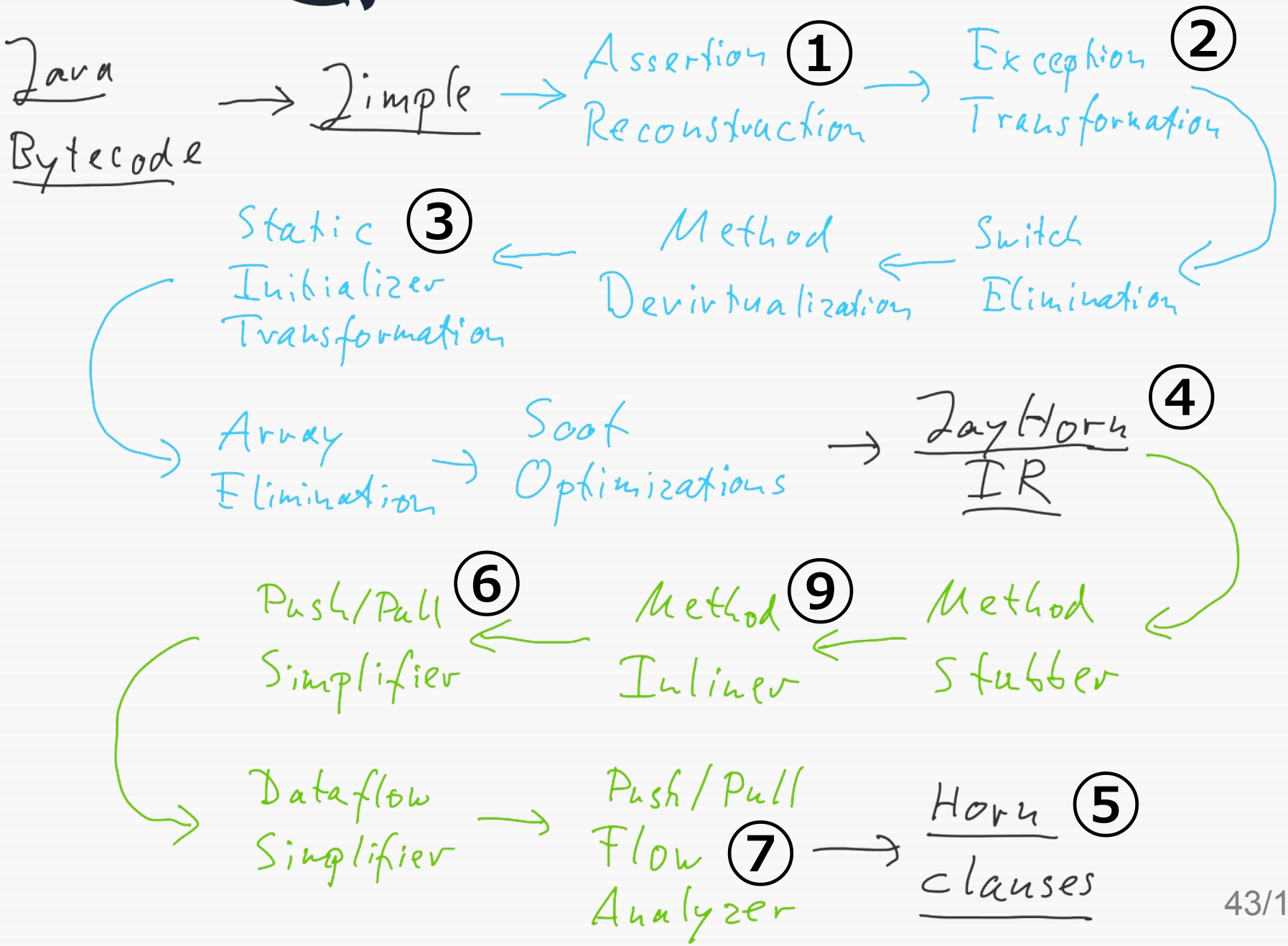
```
import org.sosy_lab.sv_benchmarks.Verifier;

public class McCarthy91 {
    private static int f(int n) {
        if (n > 100)
            return n - 10;
        else
            return f(f(n + 11));
    }

    public static void main(String[] args) {
        int x = Verifier.nondetInt();
        int y = f(x);
        assert(x > 101 || y == 91);
    }
}
```

https://github.com/sosy-lab/sv-benchmarks/blob/master/java/common/org/sosy_lab/sv_benchmarks/Verifier.java

Data-Flow


Java Bytecode
Jimple
Assertion Reconstruction
Exception Transformation
Switch Elimination
Method Devirtualization
Static Initializer Transformation
Array Elimination
Soot Optimizations
JayHorn IR
Method Stubber
Method Inliner
Push/Pull Simplifier
Dataflow Simplifier
Push/Pull Flow Analyzer
Horn clauses

Data-Flow


Java Bytecode
Jimple
Assertion Reconstruction
1
Exception Transformation
2
Switch Elimination
Method Devirtualization
Static Initializer Transformation
3
Array Elimination
Soot Optimizations
JayHorn IR
4
Method Stubber
Method Inliner
9
Push/Pull Simplifier
6
Dataflow Simplifier
Push/Pull Flow Analyzer
7
Horn clauses
5

From Java to Jimple

```
public class Test {
    int x, y;
    public static void main(String[] args) {
        Test t = new Test();
        while (t.x < 10)
            t.x++;
        assert(t.x == 10);
    }
}
```

```
<Test: void main(java.lang.String[])>
public static void main(java.lang.String[])
{
    java.lang.String[] r0;
    Test r1, $r2;
    int $i0, $i1, $i2, $i3;
    boolean $z0;
    java.lang.AssertionError $r3;
    r0 := @parameter0: java.lang.String[];
    $r2 = new Test;
    specialinvoke $r2.<Test: void <init>()>();
    r1 = $r2;

 label1:
    $i0 = r1.<Test: int x>;
    if $i0 >= 10 goto label2;
    $i2 = r1.<Test: int x>;
    $i3 = $i2 + 1;
    r1.<Test: int x> = $i3;
    goto label1;

 label2:
    $z0 = <Test: boolean $assertionsDisabled>;
    if $z0 != 0 goto label3;
    $i1 = r1.<Test: int x>;
    if $i1 == 10 goto label3;
    $r3 = new java.lang.AssertionError;
    specialinvoke $r3.<java.lang.AssertionError: void <init>()>();
    throw $r3;

 label3:
    return;
}
```

```
public class Test {
    int x, y;
    public static void main(...) {
        Test t = new Test();
        while (t.x < 10)
            t.x++;
        assert(t.x == 10);
    }
}
```

```
<Test: void main(java.lang.String[])>
public static void main(java.lang.String[])
{
    java.lang.String[] r0;
    Test r1, $r2;
    int $i0, $i1, $i2, $i3;
    boolean $z0;
    java.lang.AssertionError $r3;
    r0 := @parameter0: java.lang.String[];
    $r2 = new Test;
    specialinvoke $r2.<Test: void <init>()>();
    r1 = $r2;

 label1:
    $i0 = r1.<Test: int x>;
    if $i0 >= 10 goto label2;
    $i2 = r1.<Test: int x>;
    $i3 = $i2 + 1;
    r1.<Test: int x> = $i3;
    goto label1;

 label2:
    $z0 = <Test: boolean $assertionsDisabled>;
    if $z0 != 0 goto label3;
    $i1 = r1.<Test: int x>;
    if $i1 == 10 goto label3;
    $r3 = new java.lang.AssertionError;
    specialinvoke $r3.<java.lang.AssertionError: void <init>()>();
    throw $r3;

 label3:
    return;
}
```

```
public class Test {
    int x, y;
    public static void main(...) {
        Test t = new Test();
        while (t.x < 10)
            t.x++;
        assert(t.x == 10);
    }
}
```

+ Several further methods

```
<Test: void main(java.lang.String[])>
public static void main(java.lang.String[])
{
    java.lang.String[] r0;
    Test r1, $r2;
    int $i0, $i1, $i2, $i3;
    boolean $z0;
    java.lang.AssertionError $r3;
    r0 := @parameter0: java.lang.String[];
    $r2 = new Test;
    specialinvoke $r2.<Test: void <init>()>();
    r1 = $r2;

 label1:
    $i0 = r1.<Test: int x>;
    if $i0 >= 10 goto label2;
    $i2 = r1.<Test: int x>;
    $i3 = $i2 + 1;
    r1.<Test: int x> = $i3;
    goto label1;

 label2:
    $z0 = <Test: boolean $assertionsDisabled>;
    if $z0 != 0 goto label3;
    $i1 = r1.<Test: int x>;
    if $i1 == 10 goto label3;
    $r3 = new java.lang.AssertionError;
    specialinvoke $r3.<java.lang.AssertionError: void <init>()>();
    throw $r3;

 label3:
    return;
}
```

```
public class Test {
    int x, y;
    public static void main(...) {
        Test t = new Test();
        while (t.x < 10)
            t.x++;
        assert(t.x == 10);
    }
}
```

Load instruction

Store instruction

```
<Test: void main(java.lang.String[])>
public static void main(java.lang.String[])
{
    java.lang.String[] r0;
    Test r1, $r2;
    int $i0, $i1, $i2, $i3;
    boolean $z0;
    java.lang.AssertionError $r3;
    r0 := @parameter0: java.lang.String[];
    $r2 = new Test;
    specialinvoke $r2.<Test: void <init>()>();
    r1 = $r2;

 label1:
    $i0 = r1.<Test: int x>;
    if $i0 >= 10 goto label2;
    $i2 = r1.<Test: int x>;
    $i3 = $i2 + 1;
    r1.<Test: int x> = $i3;
    goto label1;

 label2:
    $z0 = <Test: boolean $assertionsDisabled>;
    if $z0 != 0 goto label3;
    $i1 = r1.<Test: int x>;
    if $i1 == 10 goto label3;
    $r3 = new java.lang.AssertionError;
    specialinvoke $r3.<java.lang.AssertionError: void <init>()>();
    throw $r3;

 label3:
    return;
}
```

```
public class Test {
    int x, y;
    public static void main(...) {
        Test t = new Test();
        while (t.x < 10)
            t.x++;
        assert(t.x == 10);
    }
}
```

①

```
<Test: void main(java.lang.String[])>
public static void main(java.lang.String[])
{
    java.lang.String[] r0;
    Test r1, $r2;
    int $i0, $i1, $i2, $i3;
    boolean $z0;
    java.lang.AssertionError $r3;
    r0 := @parameter0: java.lang.String[];
    $r2 = new Test;
    specialinvoke $r2.<Test: void <init>()>();
    r1 = $r2;

 label1:
    $i0 = r1.<Test: int x>;
    if $i0 >= 10 goto label2;
    $i2 = r1.<Test: int x>;
    $i3 = $i2 + 1;
    r1.<Test: int x> = $i3;
    goto label1;

 label2:
```

```
public class Test {
    int x, y;
    public static void main(...) {
        Test t = new Test();
        while (t.x < 10)
            t.x++;
        assert(t.x == 10);
    }
}
```

Reconstructed assertion

```
    $i1 = r1.<Test: int x>;
    if $i1 == 10 goto label3;
    $assert_9 = 0;
    staticinvoke <JayHornAssertions: void assertion(boolean)>($assert_9);

 label3:
    return;
}
```

```
<Test: void main(java.lang.String[])>
public static void main(java.lang.String[])
{
    java.lang.String[] r0;
    Test r1, $r2;
    int $i0, $i1, $i2, $i3;
    boolean $z0;
    java.lang.AssertionError $r3;
    r0 := @parameter0: java.lang.String[];
    $r2 = new Test;
    specialinvoke $r2.<Test: void <init>()>();
    r1 = $r2;

 label1:
    $i0 = r1.<Test: int x>;
    if $i0 >= 10 goto label2;
    $i2 = r1.<Test: int x>;
    $i3 = $i2 + 1;
    r1.<Test: int x> = $i3;
    goto label1;

 label2:


    $i1 = r1.<Test: int x>;
    if $i1 == 10 goto label3;
    $assert_9 = 0;
    staticinvoke <JayHornAssertions: void assertion(boolean)>($assert_9);


 label3:
    return;
}
```

```
public class Test {
    int x, y;
    public static void main(...) {
        Test t = new Test();
        while (t.x < 10)
            t.x++;
        assert(t.x == 10);
    }
}
```

②

```
<Test: void main(java.lang.String[])>
public static void main(java.lang.String[])
{
    java.lang.String[] r0;
    Test r1, $r2;
    int $i0, $i1, $i2, $i3;
    boolean $z0;
    java.lang.AssertionError $r3;
    r0 := @parameter0: java.lang.String[];
    $r2 = new Test;
    specialinvoke $r2.<Test: void <init>()>();
    $helper1 = <JayHornAssertions: java.lang.Throwable lastExceptionThrown>;
    $assert_11 = $helper1 == null;
    staticinvoke <JayHornAssertions: void assertion(boolean)>($assert_11);
    r1 = $r2;

 label1:
    $i0 = r1.<Test: int x>;
    if $i0 >= 10 goto label2;
    $i2 = r1.<Test: int x>;
    $i3 = $i2 + 1;
    r1.<Test: int x> = $i3;
    goto label1;

 label2:
    $i1 = r1.<Test: int x>;
    if $i1 == 10 goto label3;
    $assert_9 = 0;
    staticinvoke <JayHornAssertions: void assertion(boolean)>($assert_9);

 label3:
    return;
}
```

```
public class Test {
    int x, y;
    public static void main(...) {
        Test t = new Test();
        while (t.x < 10)
            t.x++;
        assert(t.x == 10);
    }
}
```

Exception passing through variables; assert absence of exceptions

```
<Test: void main(java.lang.String[])>
public static void main(java.lang.String[])
{
    java.lang.String[] r0;
    Test r1, $r2;
    int $i0, $i1, $i2, $i3;
    boolean $z0;
    java.lang.AssertionError $r3;
    r0 := @parameter0: java.lang.String[];
    $r2 = new Test;
    specialinvoke $r2.<Test: void <init>()>();
    $helper1 = <JayHornAssertions: java.lang.Throwable lastExceptionThrown>;
    $assert_11 = $helper1 == null;
    staticinvoke <JayHornAssertions: void assertion(boolean)>($assert_11);
    r1 = $r2;

 label1:
    $i0 = r1.<Test: int x>;
    if $i0 >= 10 goto label2;
    $i2 = r1.<Test: int x>;
    $i3 = $i2 + 1;
    r1.<Test: int x> = $i3;
    goto label1;

 label2:
    $i1 = r1.<Test: int x>;
    if $i1 == 10 goto label3;
    $assert_9 = 0;
    staticinvoke <JayHornAssertions: void assertion(boolean)>($assert_9);

 label3:
    return;
}
```

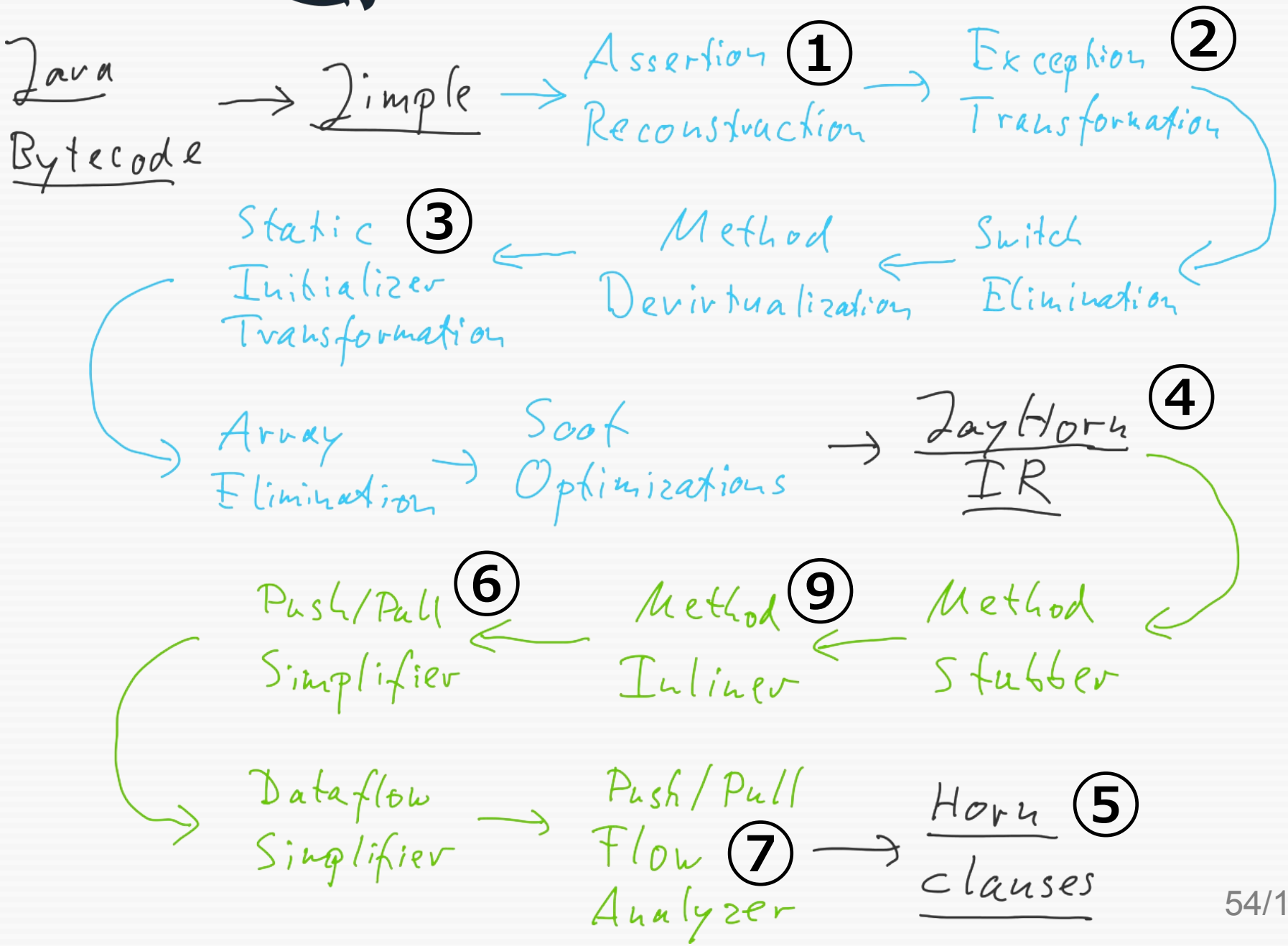
```
public class Test {
    int x, y;
    public static void main(...) {
        Test t = new Test();
        while (t.x < 10)
            t.x++;
        assert(t.x == 10);
    }
}
```

```
<Test: void main(java.lang.String[])>
public static void main(java.lang.String[])
{
    java.lang.String[] r0;
    Test r1, $r2;
    int $i0, $i1, $i2, $i3;
    boolean $z0;
    java.lang.AssertionError $r3;
    r0 := @parameter0: java.lang.String[];
③   staticinvoke <Test: void <clinit>()>();
    staticinvoke <JayHornAssertions: void <clinit>()>();
    $r2 = new Test;
    specialinvoke $r2.<Test: void <init>()>();
    $helper1 = <JayHornAssertions: java.lang.Throwable lastExceptionThrown>;
    $assert_11 = $helper1 == null;
    staticinvoke <JayHornAssertions: void assertion(boolean)>($assert_11);
    r1 = $r2;

 label1:
    $i0 = r1.<Test: int x>;
    if $i0 >= 10 goto label2;
    $i2 = r1.<Test: int x>;
    $i3 = $i2 + 1;
    r1.<Test: int x> = $i3;
    goto label1;

 label2:
    $i1 = r1.<Test: int x>;
    if $i1 == 10 goto label3;
    $assert_9 = 0;
    staticinvoke <JayHornAssertions: void assertion(boolean)>($assert_9);

 label3:
    return;
```

```
public class Test {
    int x, y;
    public static void main(...) {
        Test t = new Test();
        while (t.x < 10)
            t.x++;
        assert(t.x == 10);
    }
}
```

Static initialisers as normal methods

Data-Flow


Java Bytecode
Jimple
Assertion Reconstruction
1
Exception Transformation
2
Switch Elimination
Method Devirtualization
Static Initializer Transformation
3
Array Elimination
Soot Optimizations
JayHorn IR
4
Method Stubber
Method Inliner
9
Push/Pull Simplifier
6
Dataflow Simplifier
Push/Pull Flow Analyzer
7
Horn clauses
5

④ From Jimple to JayHorn IR

- Intermediate representation similar to Jimple
- But simpler, e.g.:
    - Methods become C-like functions
    - No exceptions

④ From Jimple to JayHorn IR

- Intermediate representation similar to Jimple
- But simpler, e.g.:
  - Methods become C-like functions
  - No exceptions

Main difference:

```
load   →   pull
store  →   push
```

```
<Test: void main(java.lang.String[])>
public static void main(java.lang.String[])
{
    java.lang.String[] r0;
    Test r1, $r2;
    int $i0, $i1, $i2, $i3;
    boolean $z0;
    java.lang.AssertionError $r3;
    r0 := @parameter0: java.lang.String[];
    staticinvoke <Test: void <clinit>()>();
    staticinvoke <JayHornAssertions: void <clinit>()>();
    $r2 = new Test;
    specialinvoke $r2.<Test: void <init>()>();
    $helper1 = <JayHornAssertions: java.lang.Throwable lastExceptionThrown>;
    $assert_11 = $helper1 == null;
    staticinvoke <JayHornAssertions: void assertion(boolean)>($assert_11);
    r1 = $r2;

 label1:
    $i0 = r1.<Test: int x>;
    if $i0 >= 10 goto label2;
    $i2 = r1.<Test: int x>;
    $i3 = $i2 + 1;
    r1.<Test: int x> = $i3;
    goto label1;

 label2:
    $i1 = r1.<Test: int x>;
    if $i1 == 10 goto label3;
    $assert_9 = 0;
    staticinvoke <JayHornAssertions: void assertion(boolean)>($assert_9);

 label3:
    return;
```

```
public class Test {
    int x, y;
    public static void main(...) {
        Test t = new Test();
        while (t.x < 10)
            t.x++;
        assert(t.x == 10);
    }
}
```

```
<Test: void main(java.lang.String[])>
public static void main(java.lang.String[])
{
    java.lang.String[] r0;
    Test r1, $r2;
    int $i0, $i1, $i2, $i3;
    boolean $z0;
    java.lang.AssertionError $r3;
    r0 := @parameter0: java.lang.String[];
    staticinvoke <Test: void <clinit>()>();
    staticinvoke <JayHornAssertions: void <clinit>()>();
    $r2 = new Test;
    specialinvoke $r2.<Test: void <init>()>();
    $helper1 = <JayHornAssertions: java.lang.Throwable lastExceptionThrown>;
    $assert_11 = $helper1 == null;
    staticinvoke <JayHornAssertions: void assertion(boolean)>($assert_11);
    r1 = $r2;

 label1:
    $i0 = r1.<Test: int x>;
    if $i0 >= 10 goto label2;
    $i2 = r1.<Test: int x>;
    $i3 = $i2 + 1;
    r1.<Test: int x> = $i3;
    goto label1;

 label2:
    $i1 = r1.<Test: int x>;
    if $i1 == 10 goto label3;
    $assert_9 = 0;
    staticinvoke <JayHornAssertions: void assertion(boolean)>($assert_9);

 label3:
    return;
```

```
public class Test {
    int x, y;
    public static void main(...) {
        Test t = new Test();
        while (t.x < 10)
            t.x++;
        assert(t.x == 10);
    }
}
```

```
<Test: void main(java.lang.String[])>
public static void main(java.lang.String[])
{
    java.lang.String[] r0;
    Test r1, $r2;
    int $i0, $i1, $i2, $i3;
    boolean $z0;
    java.lang.AssertionError $r3;
    r0 := @parameter0: java.lang.String[];
    staticinvoke <Test: void <clinit>()>();
    staticinvoke <JayHornAssertions: void <clinit>()>();
    $r2 = new Test;
    specialinvoke $r2.<Test: void <init>()>();
    $helper1 = <JayHornAssertions: java.lang.Throwable lastExceptionThrown>;
    $assert_11 = $helper1 == null;
    staticinvoke <JayHornAssertions: void assertion(boolean)>($assert_11);
    r1 = $r2;

 label1:
    $i0 = r1.<Test: int x>;
    if $i0 >= 10 goto label2;
    $i2 = r1.<Test: int x>;
    $i3 = $i2 + 1;
    r1.<Test: int x> = $i3;
    goto label1;

 label2:
    $i1 = r1.<Test: int x>;
    if $i1 == 10 goto label3;
    $assert_9 = 0;
    staticinvoke <JayHornAssertions: void assertion(boolean)>($assert_9);

 label3:
    return;
```

```
public class Test {
    int x, y;
    public static void main(...) {
        Test t = new Test();
        while (t.x < 10)
            t.x++;
        assert(t.x == 10);
    }
}
```

```
r1_x, r1_y := pull(Test, r1)
$i0 := r1_x
```

```
<Test: void main(java.lang.String[])>
public static void main(java.lang.String[])
{
    java.lang.String[] r0;
    Test r1, $r2;
    int $i0, $i1, $i2, $i3;
    boolean $z0;
    java.lang.AssertionError $r3;
    r0 := @parameter0: java.lang.String[];
    staticinvoke <Test: void <clinit>()>();
    staticinvoke <JayHornAssertions: void <clinit>()>();
    $r2 = new Test;
    specialinvoke $r2.<Test: void <init>()>();
    $helper1 = <JayHornAssertions: java.lang.Throwable lastExceptionThrown>;
    $assert_11 = $helper1 == null;
    staticinvoke <JayHornAssertions: void assertion(boolean)>($assert_11);
    r1 = $r2;

 label1:
    $i0 = r1.<Test: int x>;
    if $i0 >= 10 goto label2;
    $i2 = r1.<Test: int x>;
    $i3 = $i2 + 1;
    r1.<Test: int x> = $i3;
    goto label1;

 label2:
    $i1 = r1.<Test: int x>;
    if $i1 == 10 goto label3;
    $assert_9 = 0;
    staticinvoke <JayHornAssertions: void assertion(boolean)>($assert_9);

 label3:
    return;
```

```
public class Test {
    int x, y;
    public static void main(...) {
        Test t = new Test();
        while (t.x < 10)
            t.x++;
        assert(t.x == 10);
    }
}
```

```
r1_x, r1_y := pull(Test, r1)
$i0 := r1_x
```

```
r1_x, r1_y := pull(Test, r1)
r1_x := $i3
push(Test, r1, [r1_x, r1_y])
```

```
<Test: void main(java.lang.String[])>
public static void main(java.lang.String[])
{
    java.lang.String[] r0;
    Test r1, $r2;
    int $i0, $i1, $i2, $i3;
    boolean $z0;
    java.lang.AssertionError $r3;
    r0 := @parameter0: java.lang.String[];
    staticinvoke <Test: void <clinit>()>();
    staticinvoke <JayHornAssertions: void <clinit>()>();
    $r2 = new Test;
    specialinvoke $r2.<Test: void <ini
    $helper1 = <JayHornAssertions: jav
    $assert_11 = $helper1 == null;
    staticinvoke <JayHornAssertions: v
    r1 = $r2;

 label1:
    $i0 = r1.<Test: int x>;
    if $i0 >= 10 goto label2;
    $i2 = r1.<Test: int x>;
    $i3 = $i2 + 1;
    r1.<Test: int x> = $i3;
    goto label1;

 label2:
    $i1 = r1.<Test: int x>;
    if $i1 == 10 goto label3;
    $assert_9 = 0;
    staticinvoke <JayHornAssertions: void assertion(boolean)>($assert_9);

 label3:
    return;
```

```
public class Test {
    int x, y;
    public static void main(...) {
        Test t = new Test();
        while (t.x < 10)
            t.x++;
        assert(t.x == 10);
    }
}
```

```
havoc(r1_x, r1_y)
assume inv_Test(r1, r1_x, r1_y)
[...]
```

```
r1_x, r1_y := pull(Test, r1)
$i0 := r1_x
```

```
r1_x, r1_y := pull(Test, r1)
r1_x := $i3
push(Test, r1, [r1_x, r1_y])
```

```
<Test: void main(java.lang.String[])>
public static void main(java.lang.String[])
{
    java.lang.String[] r0;
    Test r1, $r2;
    int $i0, $i1, $i2, $i3;
    boolean $z0;
    java.lang.AssertionError $r3;
    r0 := @parameter0: java.lang.String[];
    staticinvoke <Test: void <clinit>()>();
    staticinvoke <JayHornAssertions: void <clinit>()>();
    $r2 = new Test;
    specialinvoke $r2.<Test: void <ini
    $helper1 = <JayHornAssertions: jav
    $assert_11 = $helper1 == null;
    staticinvoke <JayHornAssertions: v
    r1 = $r2;

 label1:
    $i0 = r1.<Test: int x>;
    if $i0 >= 10 goto label2;
    $i2 = r1.<Test: int x>;
    $i3 = $i2 + 1;
    r1.<Test: int x> = $i3;
    goto label1;

 label2:
    $i1 = r1.<Test: int x>;
    if $i1 == 10 goto label3;
    $assert_9 = 0;
    staticinvoke <JayHornAssertions: void assertion(boolean)>($assert_9);

 label3:
    return;
```

```
public class Test {
    int x, y;
    public static void main(...) {
        Test t = new Test();
        while (t.x < 10)
            t.x++;
        assert(t.x == 10);
    }
}
```

```
havoc(r1_x, r1_y)
assume inv_Test(r1, r1_x, r1_y)
[...]
```

```
r1_x, r1_y := pull(Test, r1)
$i0 := r1_x
```

```
r1_x, r1_y := pull(Test, r1)
r1_x := $i3
push(Test, r1, [r1_x, r1_y])
```

```
[...]
assert inv_Test(r1, r1_x, r1_y)
```

⑤ JayHorn IR to Horn Clauses

Three kinds of predicates:

- **State invariants** `loc_l`
  for every control location `l`
  → like in Ex 1
- **Pre-/post-conditions** `pre_m`, `post_m`
  for every method `m`
  → like in Ex 2
- **Class/instance invariants** `inv_C`
  for every class `C`

JayHorn IR Instructions

- `x := t` $loc_l(\ldots, x, \ldots) \rightarrow loc_{l'}(\ldots, t, \ldots)$
- `assume phi` $\phi \wedge loc_l(\ldots) \rightarrow loc_{l'}(\ldots)$
- `assert phi` $loc_l(\ldots) \rightarrow loc_{l'}(\ldots)$
 $\neg\phi \wedge loc_l(\ldots) \rightarrow false$
- `x := pull(C, p)`
- `push(C, p, [x])`

JayHorn IR Instructions

- `x := t` $loc_l(\ldots, x, \ldots) \rightarrow loc_{l'}(\ldots, t, \ldots)$
- `assume phi` $\phi \wedge loc_l(\ldots) \rightarrow loc_{l'}(\ldots)$
- `assert phi` $loc_l(\ldots) \rightarrow loc_{l'}(\ldots)$
 $\neg\phi \wedge loc_l(\ldots) \rightarrow false$
- `x := pull(C, p)` $loc_l(\ldots) \wedge inv_C(\ldots) \rightarrow loc_{l'}(\ldots)$
- `push(C, p, [x])` $loc_l(\ldots) \rightarrow loc_{l'}(\ldots)$
 $loc_l(\ldots) \rightarrow inv_C(\ldots)$

JayHorn IR Instructions

- `x := t` $loc_l(\ldots, x, \ldots) \rightarrow loc_{l'}(\ldots, t, \ldots)$
- `assume phi` $\phi \wedge loc_l(\ldots) \rightarrow loc_{l'}(\ldots)$
- `assert phi` $loc_l(\ldots) \rightarrow loc_{l'}(\ldots)$
 $\neg\phi \wedge loc_l(\ldots) \rightarrow false$
- `x := pull(C, p)` $loc_l(\ldots) \wedge inv_C(\ldots) \rightarrow loc_{l'}(\ldots)$
- `push(C, p, [x])` $loc_l(\ldots) \rightarrow loc_{l'}(\ldots)$
 $loc_l(\ldots) \rightarrow inv_C(\ldots)$

Heap is completely gone at this point!

In the Example

```
public class Test {
    int x, y;
    public static void main(String[] args) {
        Test t = new Test();
        while (t.x < 10)
            t.x++;
        assert(t.x == 10);
    }
}
```

In the Ex

Invariant **`inv_Test(p, x, y)`** has to hold for **all states** of **all `Test` objects** reachable in the program

```
public class Test {
    int x, y;
    public static void main(String[] args) {
        Test t = new Test();
        while (t.x < 10)
            t.x++;
        assert(t.x == 10);
    }
}
```

In the Ex

Invariant **`inv_Test(p, x, y)`** has to hold for **all states** of **all `Test` objects** reachable in the program

```
public class Test {
    int x, y;
    public static void main(String[] args) {
        Test t = new Test();
        while (t.x < 10)
            t.x++;
        assert(t.x == 10);
    }
}
```

$x \subseteq [0, 10] \implies$ `inv_Test(t, x, y)`

In the Ex

Invariant **`inv_Test(p, x, y)`** has to hold for **all states** of **all `Test` objects** reachable in the program

```
public class Test {
    int x, y;
    public static void main(String[] args) {
        Test t = new Test();
        while (t.x < 10)
            t.x++;
        assert(t.x == 10);
    }
}
```

$x \subseteq [0, 10] \Longrightarrow$ `inv_Test(t, x, y)`

`inv_Test(t, x, y)` will be **too weak** to prove the assertion

In the Ex

Invariant **`inv_Test(p, x, y)`** has to hold for **all states** of **all `Test` objects** reachable in the program

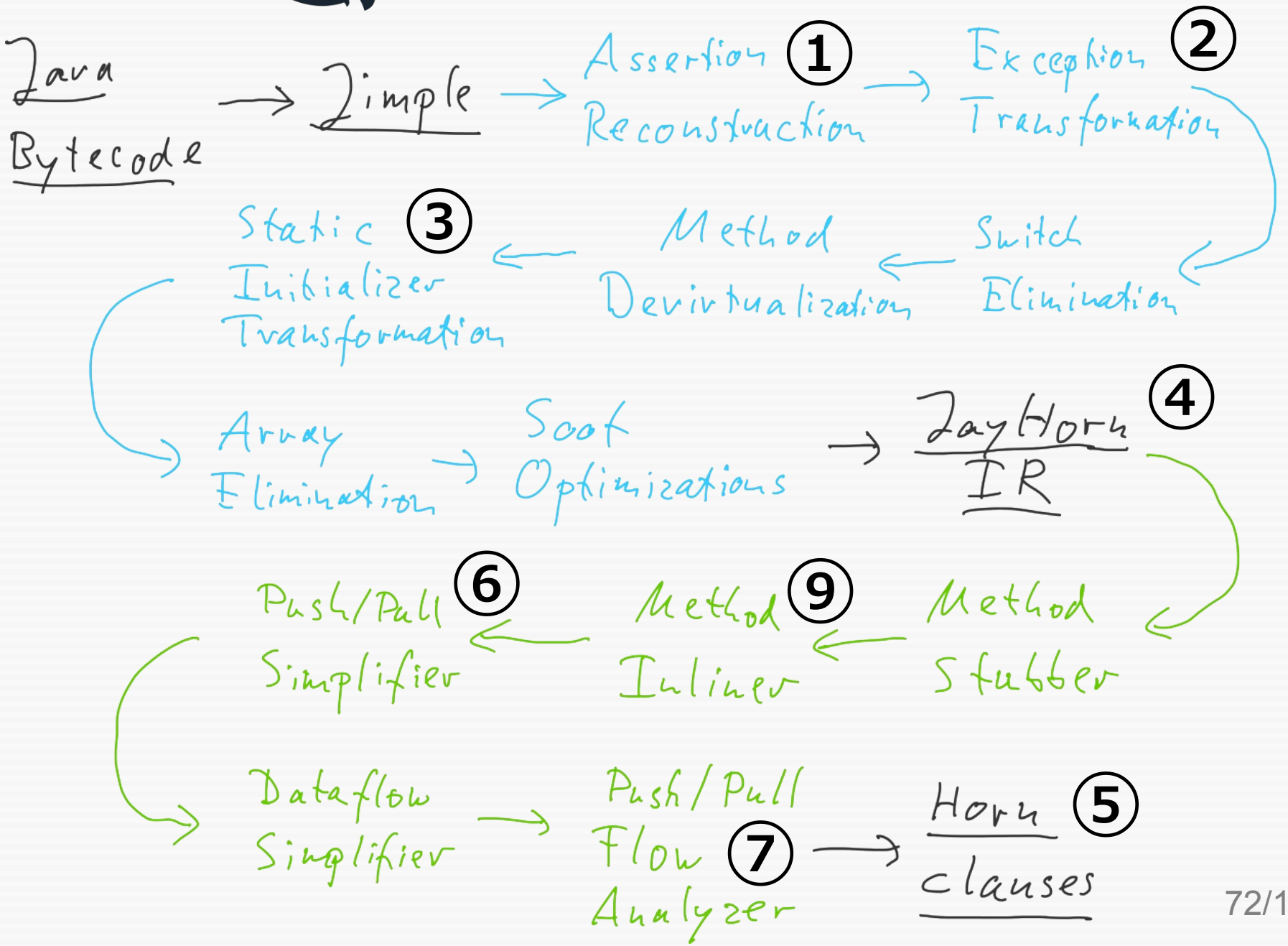
```
public class Test {
    int x, y;
    public static void main(String[] args) {
        Test t = new Test();
        while (t.x < 10)
            t.x++;
        assert(t.x == 10);
    }
}
```

$x \subseteq [0, 10] \Longrightarrow$ `inv_Test(t, x, y)`

NB: loop condition does not help, since state invariants only see **local variables**

`inv_Test(t, x, y)` will be **too weak** to prove the assertion

Data-Flow


Java Bytecode
Jimple
Assertion Reconstruction
1
Exception Transformation
2
Switch Elimination
Method Devirtualization
Static Initializer Transformation
3
Array Elimination
Soot Optimizations
JayHorn IR
4
Method Stubber
Method Inliner
9
Push/Pull Simplifier
6
Dataflow Simplifier
Push/Pull Flow Analyzer
7
Horn clauses
5

```
<Test: void main(java.lang.String[])>
public static void main(java.lang.String[])
{
    java.lang.String[] r0;
    Test r1, $r2;
    int $i0, $i1, $i2, $i3;
    boolean $z0;
    java.lang.AssertionError $r3;
    r0 := @parameter0: java.lang.String[];
    staticinvoke <Test: void <clinit>()>();
    staticinvoke <JayHornAssertions: void <clinit>()>();
    $r2 = new Test;
    specialinvoke $r2.<Test: void <init>()>();
    $helper1 = <JayHornAssertions: java.lang.Throwable lastExceptionThrown>;
    $assert_11 = $helper1 == null;
    staticinvoke <JayHornAssertions: void assertion(boolean)>($assert_11);
    r1 = $r2;

 label1:
    r1_x, r1_y := pull(Test, r1)
    $i0 := r1_x
    if $i0 >= 10 goto label2;
    r1_x, r1_y := pull(Test, r1)
    $i2 := r1_x
    $i3 = $i2 + 1;
    r1_x, r1_y := pull(Test, r1)
    r1_x := $i3
    push(Test, r1, [r1_x, r1_y])
    goto label1;

 label2:
    [...]
```

```
public class Test {
    int x, y;
    public static void main(...) {
        Test t = new Test();
        while (t.x < 10)
            t.x++;
        assert(t.x == 10);
    }
}
```

```
<Test: void main(java.lang.String[])>
public static void main(java.lang.String[])
{
    java.lang.String[] r0;
    Test r1, $r2;
    int $i0, $i1, $i2, $i3;
    boolean $z0;
    java.lang.AssertionError $r3;
    r0 := @parameter0: java.lang.String[];
    staticinvoke <Test: void <clinit>()>();
    staticinvoke <JayHornAssertions: void <clinit>()>();
    $r2 = new Test;
    specialinvoke $r2.<Test: void <init>()>();
    $helper1 = <JayHornAssertions: java.lang.Throwable lastExceptionThrown>;
    $assert_11 = $helper1 == null;
    staticinvoke <JayHornAssertions: void assertion(boolean)>($assert_11);
    r1 = $r2;

 label1:
    r1_x, r1_y := pull(Test, r1)
    $i0 := r1_x
    if $i0 >= 10 goto label2;
    r1_x, r1_y := pull(Test, r1)
    $i2 := r1_x
    $i3 = $i2 + 1;
    r1_x, r1_y := pull(Test, r1)
    r1_x := $i3
    push(Test, r1, [r1_x, r1_y])
    goto label1;

 label2:
    [...]
```

```
public class Test {
    int x, y;
    public static void main(...) {
        Test t = new Test();
        while (t.x < 10)
            t.x++;
        assert(t.x == 10);
    }
}
```

Move pulls upward and pushes downward in the program ⑥

```
<Test: void main(java.lang.String[])>
public static void main(java.lang.String[])
{
    java.lang.String[] r0;
    Test r1, $r2;
    int $i0, $i1, $i2, $i3;
    boolean $z0;
    java.lang.AssertionError $r3;
    r0 := @parameter0: java.lang.String[];
    staticinvoke <Test: void <clinit>()>();
    staticinvoke <JayHornAssertions: void <clinit>()>();
    $r2 = new Test;
    specialinvoke $r2.<Test: void <init>()>();
    $helper1 = <JayHornAssertions: java.lang.Throwable lastExceptionThrown>;
    $assert_11 = $helper1 == null;
    staticinvoke <JayHornAssertions: void assertion(boolean)>($assert_11);
    r1 = $r2;
```

```
public class Test {
    int x, y;
    public static void main(...) {
        Test t = new Test();
        while (t.x < 10)
            t.x++;
        assert(t.x == 10);
    }
}
```

```
label1:
    r1_x, r1_y := pull(Test, r1)
    $i0 := r1_x
    if $i0 >= 10 goto label2;
    r1_x, r1_y := pull(Test, r1)
    $i2 := r1_x
    $i3 = $i2 + 1;
    r1_x, r1_y := pull(Test, r1)
    r1_x := $i3
    push(Test, r1, [r1_x, r1_y])
    goto label1;

label2:
    [...]
```

```
label1:
  r1_x, r1_y := pull(Test, r1)
  $i0 := r1_x
  if $i0 >= 10 goto label2;
  r1_x, r1_y := pull(Test, r1)
  $i2 := r1_x
  $i3 = $i2 + 1;
  r1_x, r1_y := pull(Test, r1)
  r1_x := $i3
  push(Test, r1, [r1_x, r1_y])
  goto label1;
```

```
<Test: void main(java.lang.String[])>
public static void main(java.lang.String[])
{
    java.lang.String[] r0;
    Test r1, $r2;
    int $i0, $i1, $i2, $i3;
    boolean $z0;
    java.lang.AssertionError $r3;
    r0 := @parameter0: java.lang.String[];
    staticinvoke <Test: void <clinit>()>();
    staticinvoke <JayHornAssertions: void <clinit>()>();
    $r2 = new Test;
    specialinvoke $r2.<Test: void <init>()>();
    $helper1 = <JayHornAssertions: java.lang.Throwable lastExceptionThrown>;
    $assert_11 = $helper1 == null;
    staticinvoke <JayHornAssertions: void assertion(boolean)>($assert_11);
    r1 = $r2;
```

```
public class Test {
    int x, y;
    public static void main(...) {
        Test t = new Test();
        while (t.x < 10)
            t.x++;
        assert(t.x == 10);
    }
}
```

```
label1:
    r1_x, r1_y := pull(Test, r1)
    $i0 := r1_x
    if $i0 >= 10 goto label2;
    r1_x, r1_y := pull(Test, r1)
    $i2 := r1_x
    $i3 = $i2 + 1;
    r1_x, r1_y := pull(Test, r1)
    r1_x := $i3
    push(Test, r1, [r1_x, r1_y])
    goto label1;

label2:
    [...]
```

```
label1:
  r1_x, r1_y := pull(Test, r1)
  $i0 := r1_x
  if $i0 >= 10 goto label2;
  r1_x, r1_y := pull(Test, r1)
  $i2 := r1_x
  r1_x, r1_y := pull(Test, r1)
  $i3 = $i2 + 1;
  r1_x := $i3
  push(Test, r1, [r1_x, r1_y])
  goto label1;
```

```
<Test: void main(java.lang.String[])>
public static void main(java.lang.String[])
{
    java.lang.String[] r0;
    Test r1, $r2;
    int $i0, $i1, $i2, $i3;
    boolean $z0;
    java.lang.AssertionError $r3;
    r0 := @parameter0: java.lang.String[];
    staticinvoke <Test: void <clinit>()>();
    staticinvoke <JayHornAssertions: void <clinit>()>();
    $r2 = new Test;
    specialinvoke $r2.<Test: void <init>()>();
    $helper1 = <JayHornAssertions: java.lang.Throwable lastExceptionThrown>;
    $assert_11 = $helper1 == null;
    staticinvoke <JayHornAssertions: void assertion(boolean)>($assert_11);
    r1 = $r2;
```

```
public class Test {
    int x, y;
    public static void main(...) {
        Test t = new Test();
        while (t.x < 10)
            t.x++;
        assert(t.x == 10);
    }
}
```

```
label1:
    r1_x, r1_y := pull(Test, r1)
    $i0 := r1_x
    if $i0 >= 10 goto label2;
    r1_x, r1_y := pull(Test, r1)
    $i2 := r1_x
    $i3 = $i2 + 1;
    r1_x, r1_y := pull(Test, r1)
    r1_x := $i3
    push(Test, r1, [r1_x, r1_y])
    goto label1;

label2:
    [...]
```

```
label1:
  r1_x, r1_y := pull(Test, r1)
  $i0 := r1_x
  if $i0 >= 10 goto label2;
  r1_x, r1_y := pull(Test, r1)
  r1_x', r1_y := pull(Test, r1)
  $i2 := r1_x
  $i3 = $i2 + 1;
  r1_x := $i3
  push(Test, r1, [r1_x, r1_y])
  goto label1;
```

```
<Test: void main(java.lang.String[])>
public static void main(java.lang.String[])
{
    java.lang.String[] r0;
    Test r1, $r2;
    int $i0, $i1, $i2, $i3;
    boolean $z0;
    java.lang.AssertionError $r3;
    r0 := @parameter0: java.lang.String[];
    staticinvoke <Test: void <clinit>()>();
    staticinvoke <JayHornAssertions: void <clinit>()>();
    $r2 = new Test;
    specialinvoke $r2.<Test: void <init>()>();
    $helper1 = <JayHornAssertions: java.lang.Throwable lastExceptionThrown>;
    $assert_11 = $helper1 == null;
    staticinvoke <JayHornAssertions: void assertion(boolean)>($assert_11);
    r1 = $r2;
```

```
public class Test {
    int x, y;
    public static void main(...) {
        Test t = new Test();
        while (t.x < 10)
            t.x++;
        assert(t.x == 10);
    }
}
```

```
label1:
    r1_x, r1_y := pull(Test, r1)
    $i0 := r1_x
    if $i0 >= 10 goto label2;
    r1_x, r1_y := pull(Test, r1)
    $i2 := r1_x
    $i3 = $i2 + 1;
    r1_x, r1_y := pull(Test, r1)
    r1_x := $i3
    push(Test, r1, [r1_x, r1_y])
    goto label1;

label2:
    [...]
```

```
label1:
  r1_x, r1_y := pull(Test, r1)
  $i0 := r1_x
  if $i0 >= 10 goto label2;
  r1_x, r1_y := pull(Test, r1)
  r1_x' := r1_x; r1_y := r1_y
  $i2 := r1_x
  $i3 = $i2 + 1;
  r1_x := $i3
  push(Test, r1, [r1_x, r1_y])
  goto label1;
```

```
<Test: void main(java.lang.String[])>
public static void main(java.lang.String[])
{
    java.lang.String[] r0;
    Test r1, $r2;
    int $i0, $i1, $i2, $i3;
    boolean $z0;
    java.lang.AssertionError $r3;
    r0 := @parameter0: java.lang.String[];
    staticinvoke <Test: void <clinit>()>();
    staticinvoke <JayHornAssertions: void <clinit>()>();
    $r2 = new Test;
    specialinvoke $r2.<Test: void <init>()>();
    $helper1 = <JayHornAssertions: java.lang.Throwable lastExceptionThrown>;
    $assert_11 = $helper1 == null;
    staticinvoke <JayHornAssertions: void assertion(boolean)>($assert_11);
    r1 = $r2;
```

```
public class Test {
    int x, y;
    public static void main(...) {
        Test t = new Test();
        while (t.x < 10)
            t.x++;
        assert(t.x == 10);
    }
}
```

```
label1:
    r1_x, r1_y := pull(Test, r1)
    $i0 := r1_x
    if $i0 >= 10 goto label2;
    r1_x, r1_y := pull(Test, r1)
    $i2 := r1_x
    $i3 = $i2 + 1;
    r1_x, r1_y := pull(Test, r1)
    r1_x := $i3
    push(Test, r1, [r1_x, r1_y])
    goto label1;

label2:
    [...]
```

```
label1:
  r1_x, r1_y := pull(Test, r1)
  $i0 := r1_x
  if $i0 >= 10 goto label2;
  r1_x, r1_y := pull(Test, r1)

  $i2 := r1_x
  $i3 = $i2 + 1;
  r1_x := $i3
  push(Test, r1, [r1_x, r1_y])
  goto label1;
```

```
<Test: void main(java.lang.String[])>
public static void main(java.lang.String[])
{
    java.lang.String[] r0;
    Test r1, $r2;
    int $i0, $i1, $i2, $i3;
    boolean $z0;
    java.lang.AssertionError $r3;
    r0 := @parameter0: java.lang.String[];
    staticinvoke <Test: void <clinit>()>();
    staticinvoke <JayHornAssertions: void <clinit>()>();
    $r2 = new Test;
    specialinvoke $r2.<Test: void <init>()>();
    $helper1 = <JayHornAssertions: java.lang.Throwable lastExceptionThrown>;
    $assert_11 = $helper1 == null;
    staticinvoke <JayHornAssertions: void assertion(boolean)>($assert_11);
    r1 = $r2;
```

```
public class Test {
    int x, y;
    public static void main(...) {
        Test t = new Test();
        while (t.x < 10)
            t.x++;
        assert(t.x == 10);
    }
}
```

```
label1:
    r1_x, r1_y := pull(Test, r1)
    $i0 := r1_x
    if $i0 >= 10 goto label2;
    r1_x, r1_y := pull(Test, r1)
    $i2 := r1_x
    $i3 = $i2 + 1;
    r1_x, r1_y := pull(Test, r1)
    r1_x := $i3
    push(Test, r1, [r1_x, r1_y])
    goto label1;

label2:
    [...]
```

```
label1:
  r1_x, r1_y := pull(Test, r1)
  $i0 := r1_x
  if $i0 >= 10 goto label2;
  r1_x, r1_y := pull(Test, r1)

  $i2 := r1_x
  $i3 = $i2 + 1;
  r1_x := $i3
  push(Test, r1, [r1_x, r1_y])
  goto label1;
```

```
<Test: void main(java.lang.String[])>
public static void main(java.lang.String[])
{
    java.lang.String[] r0;
    Test r1, $r2;
    int $i0, $i1, $i2, $i3;
    boolean $z0;
    java.lang.AssertionError $r3;
    r0 := @parameter0: java.lang.String[];
    staticinvoke <Test: void <clinit>()>();
    staticinvoke <JayHornAssertions: void <clinit>()>();
    $r2 = new Test;
    specialinvoke $r2.<Test: void <init>()>();
    $helper1 = <JayHornAssertions: java.lang.Throwable lastExceptionThrown>;
    $assert_11 = $helper1 == null;
    staticinvoke <JayHornAssertions: void assertion(boolean)>($assert_11);
    r1 = $r2;
```

```
public class Test {
    int x, y;
    public static void main(...) {
        Test t = new Test();
        while (t.x < 10)
            t.x++;
        assert(t.x == 10);
    }
}
```

```
label1:
    r1_x, r1_y := pull(Test, r1)
    $i0 := r1_x
    if $i0 >= 10 goto label2;
    r1_x, r1_y := pull(Test, r1)
    $i2 := r1_x
    $i3 = $i2 + 1;
    r1_x, r1_y := pull(Test, r1)
    r1_x := $i3
    push(Test, r1, [r1_x, r1_y])
    goto label1;

label2:
    [...]
```

```
label1:
  r1_x, r1_y := pull(Test, r1)
  $i0 := r1_x
  r1_x, r1_y := pull(Test, r1)
  if $i0 >= 10 goto label2;

  $i2 := r1_x
  $i3 = $i2 + 1;
  r1_x := $i3
  push(Test, r1, [r1_x, r1_y])
  goto label1;
```

```
<Test: void main(java.lang.String[])>
public static void main(java.lang.String[])
{
    java.lang.String[] r0;
    Test r1, $r2;
    int $i0, $i1, $i2, $i3;
    boolean $z0;
    java.lang.AssertionError $r3;
    r0 := @parameter0: java.lang.String[];
    staticinvoke <Test: void <clinit>()>();
    staticinvoke <JayHornAssertions: void <clinit>()>();
    $r2 = new Test;
    specialinvoke $r2.<Test: void <init>()>();
    $helper1 = <JayHornAssertions: java.lang.Throwable lastExceptionThrown>;
    $assert_11 = $helper1 == null;
    staticinvoke <JayHornAssertions: void assertion(boolean)>($assert_11);
    r1 = $r2;
```

```
public class Test {
    int x, y;
    public static void main(...) {
        Test t = new Test();
        while (t.x < 10)
            t.x++;
        assert(t.x == 10);
    }
}
```

```
label1:
    r1_x, r1_y := pull(Test, r1)
    $i0 := r1_x
    if $i0 >= 10 goto label2;
    r1_x, r1_y := pull(Test, r1)
    $i2 := r1_x
    $i3 = $i2 + 1;
    r1_x, r1_y := pull(Test, r1)
    r1_x := $i3
    push(Test, r1, [r1_x, r1_y])
    goto label1;

label2:
    [...]
```

```
label1:
  r1_x, r1_y := pull(Test, r1)
  $i0 := r1_x
  r1_x, r1_y := pull(Test, r1)
  if $i0 >= 10 goto label2;

  $i2 := r1_x
  $i3 = $i2 + 1;
  r1_x := $i3
  push(Test, r1, [r1_x, r1_y])
  goto label1;
```

...

```
<Test: void main(java.lang.String[])>
public static void main(java.lang.String[])
{
    java.lang.String[] r0;
    Test r1, $r2;
    int $i0, $i1, $i2, $i3;
    boolean $z0;
    java.lang.AssertionError $r3;
    r0 := @parameter0: java.lang.String[];
    staticinvoke <Test: void <clinit>()>();
    staticinvoke <JayHornAssertions: void <clinit>()>();
    $r2 = new Test;
    specialinvoke $r2.<Test: void <init>()>();
    $helper1 = <JayHornAssertions: java.lang.Throwable lastExceptionThrown>;
    $assert_11 = $helper1 == null;
    staticinvoke <JayHornAssertions: void assertion(boolean)>($assert_11);
    r1 = $r2;
    r1_x, r1_y := pull(Test, r1)

 label1:
    $i0 := r1_x
    if $i0 >= 10 goto label2;
    $i2 := r1_x
    $i3 = $i2 + 1;
    r1_x := $i3
    push(Test, r1, [r1_x, r1_y])
    goto label1;

 label2:
    [...]
```

```
public class Test {
    int x, y;
    public static void main(...) {
        Test t = new Test();
        while (t.x < 10)
            t.x++;
        assert(t.x == 10);
    }
}
```

Finally, complete computation happening on local variables!

Push/Pull Simplification Rules

$$\text{(I)}\quad \frac{\mathbf{pull}_c(x_1, \ldots, x_n, p)\ \ \mathbf{pull}_c(y_1, \ldots, y_n, p)}{\mathbf{pull}_c(x_1, \ldots, x_n, p)\ \ y_1 := x_1; \ldots; y_n := x_n}$$

$$\text{(II)}\quad \frac{\mathbf{push}_c(p, t_1, \ldots, t_n)\ \ \mathbf{push}_c(p, s_1, \ldots, s_n)}{\mathbf{push}_c(p, s_1, \ldots, s_n)}$$

$$\text{(III)}\quad \frac{\mathbf{push}_c(p, t_1, \ldots, t_n);\ \mathbf{pull}_c(x_1, \ldots, x_n, p)}{\mathbf{push}_c(p, t_1, \ldots, t_n)\ \ x_1 := t_1; \ldots x_n := t_n}$$

$$\text{(IV)}\quad \frac{\mathbf{pull}_c(x_1, \ldots, x_n, p)\ \ \mathbf{push}_c(p, x_1, \ldots, x_n)}{\mathbf{pull}_c(x_1, \ldots, x_n, p)}$$

$$\text{(V)}\quad \frac{x := t\ \ \mathbf{pull}_c(y_1, \ldots, y_n, p)}{\mathbf{pull}_c(y_1, \ldots, y_n, p)\ \ x := t} \qquad \text{(VI)}\quad \frac{\mathbf{push}_c(p, t_1, \ldots, t_n)\ \ x := t}{x := t\ \ \mathbf{push}_c(p, t_1, \ldots, t_n}$$

$$\text{(VII)}\quad \frac{\mathbf{push}_c(p, t_1, \ldots, t_n)\ \ \mathbf{pull}_c(x_1, \ldots, x_n, r)}{\mathbf{assert}(p \neq r)\ \ \mathbf{pull}_c(x_1, \ldots, x_n, r)\ \ \mathbf{push}_c(p, t_1, \ldots, t_n)}$$

Push/Pull Simplification Rules

$$\text{(I)}\quad \frac{\mathbf{pull}_c(x_1, \ldots, x_n, p)\ \ \mathbf{pull}_c(y_1, \ldots, y_n, p)}{\mathbf{pull}_c(x_1, \ldots, x_n, p)\ \ y_1 := x_1; \ldots; y_n := x_n}$$

$$\text{(II)}\quad \frac{\mathbf{push}_c(p, t_1, \ldots, t_n)\ \ \mathbf{push}_c(p, s_1, \ldots, s_n)}{\mathbf{push}_c(p, s_1, \ldots, s_n)}$$

$$\text{(III)}\quad \frac{\mathbf{push}_c(p, t_1, \ldots, t_n);\ \mathbf{pull}_c(x_1, \ldots, x_n, p)}{\mathbf{push}_c(p, t_1, \ldots, t_n)\ \ x_1 := t_1; \ldots x_n := t_n}$$

$$\text{(IV)}\quad \frac{\mathbf{pull}_c(x_1, \ldots, x_n, p)\ \ \mathbf{push}_c(p, x_1, \ldots, x_n)}{\mathbf{pull}_c(x_1, \ldots, x_n, p)}$$

Assuming distinct sets of variables

x not occurring in p, t1, ...

$$\text{(V)}\quad \frac{x := t\ \ \mathbf{pull}_c(y_1, \ldots, y_n, p)}{\mathbf{pull}_c(y_1, \ldots, y_n, p)\ \ x := t} \qquad \text{(VI)}\quad \frac{\mathbf{push}_c(p, t_1, \ldots, t_n)\ \ x := t}{x := t\ \ \mathbf{push}_c(p, t_1, \ldots, t_n}$$

$$\text{(VII)}\quad \frac{\mathbf{push}_c(p, t_1, \ldots, t_n)\ \ \mathbf{pull}_c(x_1, \ldots, x_n, r)}{\mathbf{assert}(p \neq r)\ \ \mathbf{pull}_c(x_1, \ldots, x_n, r)\ \ \mathbf{push}_c(p, t_1, \ldots, t_n)}$$

⑦ Adding Flow-Sensitivity

```
public class FlowSens {
    int x;
    public static void main(String[] args) {
        FlowSens t = new FlowSens();
        t.x = 1;
        if (args.length > 5)
            t.x += 10;
        else
            t.x += 20;
        f(t);
    }
    public static void f(FlowSens t) {
        assert t.x > 1;
    }
}
```

⑦ Adding Flow-Sensitivity

```
public class FlowSens {
    int x;
    public static void main(String[] args) {
        FlowSens t = new FlowSens();  push(t, 0)
        t.x = 1;
        if (args.length > 5)
            t.x += 10;
        else
            t.x += 20;
        f(t);
    }
    public static void f(FlowSens t) {
        assert t.x > 1;
    }
}
```

⑦ Adding Flow-Sensitivity

```
public class FlowSens {
    int x;
    public static void main(String[] args) {
        FlowSens t = new FlowSens();  push(t, 0)
        t.x = 1;                      push(t, 1)
        if (args.length > 5)
            t.x += 10;
        else
            t.x += 20;
        f(t);
    }
    public static void f(FlowSens t) {
        assert t.x > 1;
    }
}
```

⑦ Adding Flow-Sensitivity

```
public class FlowSens {
    int x;
    public static void main(String[] args) {
        FlowSens t = new FlowSens();   push(t, 0)
        t.x = 1;                       push(t, 1)
        if (args.length > 5)
            t.x += 10;                 push(t, 11)
        else
            t.x += 20;                 push(t, 21)
        f(t);
    }
    public static void f(FlowSens t) {
        assert t.x > 1;
    }
}
```

⑦ Adding Flow-Sensitivity

```
public class FlowSens {
    int x;
    public static void main(String[] args) {
        FlowSens t = new FlowSens();
        t.x = 1;
        if (args.length > 5)
            t.x += 10;
        else
            t.x += 20;
        f(t);
    }
    public static void f(FlowSens t) {
        assert t.x > 1;
    }
}
```

```
push(t, 0)
push(t, 1)

push(t, 11)

push(t, 21)

x := pull(t)
```

⑦ Adding Flow-Sensitivity

```
public class FlowSens {
    int x;
    public static void main(String[] args) {
        FlowSens t = new FlowSens();
        t.x = 1;
        if (args.length > 5)
            t.x += 10;
        else
            t.x += 20;
        f(t);
    }
    public static void f(FlowSens t) {
        assert t.x > 1;
    }
}
```

```
push_0(t, 0)
push_1(t, 1)

push_2(t, 11)

push_3(t, 21)

x := pull(t)
```

⑦ Adding Flow-Sensitivity

```
public class FlowSens {
    int x;
    public static void main(String[] args) {
        FlowSens t = new FlowSens();
        t.x = 1;
        if (args.length > 5)
            t.x += 10;
        else
            t.x += 20;
        f(t);
    }
    public static void f(FlowSens t) {
        assert t.x > 1;
    }
}
```

```
push_0(t, 0)
push_1(t, 1)

push_2(t, 11)

push_3(t, 21)

x := pull(t)[2,3]
```

Adding Flow-Sensitivity (2)

Definition
Statement

S = `push(p, x)`

is a **(data-)dependency** of statement

L = `x := pull(q)`

if there is a path from S to L such that

- p and q **may alias** (on that path);
- there is no further statement `push(p', x')` on the path such that p and p' **must alias**.

Adding Flow-Sensitivity (3)

```
public class FlowSens {
    int x;
    public static void main(String[] args) {
        FlowSens t = new FlowSens();
        t.x = 1;
        if (args.length > 5)
            t.x += 10;
        else
            t.x += 20;
        f(t);
    }
    public static void f(FlowSens t) {
        assert t.x > 1;
    }
}
```

```
push_0(t, 0)
push_1(t, 1)

push_2(t, 11)

push_3(t, 21)

x := pull(t)[2,3]
```

g Flow-Sensitivity (3)

Ghost field to store the ID of the last push

```
public class FlowSens {
    int x; int pushID;
    public static void main(String[] args) {
        FlowSens t = new FlowSens();
        t.x = 1;
        if (args.length > 5)
            t.x += 10;
        else
            t.x += 20;
        f(t);
    }
    public static void f(FlowSens t) {
        assert t.x > 1;
    }
}
```

```
push_0(t, 0)
push_1(t, 1)

push_2(t, 11)

push_3(t, 21)

x := pull(t)[2,3]
```

g Flow-Sensitivity (3)

Ghost field to store the ID of the last push

Assign to the ghost field

```
public class FlowSens {
    int x; int pushID;
    public static void main(String[] args) {
        FlowSens t = new FlowSens();
        t.x = 1;
        if (args.length > 5)
            t.x += 10;
        else
            t.x += 20;
        f(t);
    }
    public static void f(FlowSens t) {
        assert t.x > 1;
    }
}
```

```
push(t, 0, 0)
push(t, 1, 1)

push(t, 11, 2)

push(t, 21, 3)

x := pull(t)[2,3]
```

g Flow-Sensitivity (3)

Ghost field to store the ID of the last push

Assign to the ghost field

```
public class FlowSens {
    int x; int pushID;
    public static void main(String[] args) {
        FlowSens t = new FlowSens();
        t.x = 1;
        if (args.length > 5)
            t.x += 10;
        else
            t.x += 20;
        f(t);
    }
    public static void f(FlowSens t) {
        assert t.x > 1;
    }
}
```

```
push(t, 0, 0)
push(t, 1, 1)

push(t, 11, 2)

push(t, 21, 3)
```

```
x, i := pull(t)
assume i==2 || i==3
```

Check that we read right version

g Flow-Sensitivity (3)

Ghost field to store the ID of the last push

Assign to the ghost field

```
public class FlowSens {
    int x; int pushID;
```

Possible class invariant:

```
pushID == 0 && x == 0 ||
pushID == 1 && x == 1 ||
pushID == 2 && x > 1  ||
pushID == 3 && x > 1
```

```
ng[] args) {
s();
```

```
push(t, 0, 0)
push(t, 1, 1)

push(t, 11, 2)

push(t, 21, 3)
```

```
        t.x += 20;
        f(t);
    }
    public static void f(FlowSens t) {
        assert t.x > 1;
    }
}
```

```
x, i := pull(t)
assume i==2 || i==3
```

Check that we read right version

⑧ Tupled References

```
01 public static class Node {
02   final Node next;
03   final int data;
04
05   public Node(Node next, int data) {
06     this.next = next;
07     this.data = data;
08   }
09 }
10
11 public static void main(String[] args) {
12   final int size = 10;
13   final int[] table = new int[size] ;
14   Node l1 = null;
15   Node l2 = null;
16   for (int i=0; i<args.length; i++) {
17       int d = Integer.parseInt(args[i]);
18       if (d >= 0 && d < size) {
19         l1 = new Node(l1, d);
20       } else {
21         l2 = new Node(l2, d);
22       }
23   }
24   while (l1 != null) {
25     table[l1.data] = table[l1.data] + 1;
26     l1 = l1.next;
27   }
28 }
```

⑧ Tupled References

```
01 public static class Node {
02   final Node next;
03   final int data;
04
05   public Node(Node next, int data) {
06     this.next = next;
07     this.data = data;
08   }
09 }
10
11 public static void main(String[] args) {
12   final int size = 10;
13   final int[] table = new int[size] ;
14   Node l1 = null;
15   Node l2 = null;
16   for (int i=0; i<args.length; i++) {
17       int d = Integer.parseInt(args[i]);
18       if (d >= 0 && d < size) {
19         l1 = new Node(l1, d);
20       } else {
21         l2 = new Node(l2, d);
22       }
23   }
24   while (l1 != null) {
25     table[l1.data] = table[l1.data] + 1;
26     l1 = l1.next;
27   }
28 }
```

Need to show absence of `ArrayIndexOutOfBoundsE`.

⑧ Tupled References

```
01 public static class Node {
02   final Node next;
03   final int data;
04
05   public Node(Node next, int data) {
06     this.next = next;
07     this.data = data;
08   }
09 }
10
11 public static void main(String[] args) {
12   final int size = 10;
13   final int[] table = new int[size] ;
14   Node l1 = null;
15   Node l2 = null;
16   for (int i=0; i<args.length; i++) {
17       int d = Integer.parseInt(args[i]);
18       if (d >= 0 && d < size) {
19         l1 = new Node(l1, d);
20       } else {
21         l2 = new Node(l2, d);
22       }
23   }
24   while (l1 != null) {
25     table[l1.data] = table[l1.data] + 1;
26     l1 = l1.next;
27   }
28 }
```

For this, we need an invariant about objects in the `l1` list

Need to show absence of `ArrayIndexOutOfBoundsE`.

⑧ Tupled References

```
01 public static class Node {
02   final Node next;
03   final int data;
04
05   public Node(Node next,
06     this.next = next;
07     this.data = data;
08   }
09 }
10
11 public static void main(
12   final int size = 10;
13   final int[] table = new     [size] ;
14   Node l1 = null;
15   Node l2 = null;
16   for (int i=0; i<args.length; i++) {
17       int d = Integer.parseInt(args[i]);
18       if (d >= 0 && d < size) {
19         l1 = new Node(l1, d);
20       } else {
21         l2 = new Node(l2, d);
22       }
23   }
24   while (l1 != null) {
25     table[l1.data] = table[l1.data] + 1;
26     l1 = l1.next;
27   }
28 }
```

Idea: we can distinguish `l1` and `l2` based on the **allocation site**

For this, we need an invariant about objects in the `l1` list

Need to show absence of `ArrayIndexOutOfBoundsE.`

Partitioning of Classes

Redefine references to be **tuples of immutable data** about objects, e.g.:

- Memory address
- (Dynamic) type
- Allocation site
- Values of immutable fields
- Constructor arguments

Partitioning of Classes

Redefine references to be **tuples of immutable data** about objects, e.g.:

- Memory address
- (Dynamic) type
- Allocation site
- Values of immutable fields
- Constructor arguments

This data becomes visible in state invariants, method contracts, and class invariants!

⑧ Tupled References

```
01 public static class Node {
02   final Node next;
03   final int data;
04
05   public Node(Node next, int data) {
06     this.next = next;
07     this.data = data;
08   }
09 }
10
11 public static void main(String[] args) {
12   final int size = 10;
13   final int[] table = new int[size] ;
14   Node l1 = null;
15   Node l2 = null;
16   for (int i=0; i<args.length; i++) {
17       int d = Integer.parseInt(args[i]);
18       if (d >= 0 && d < size) {
19         l1 = new Node(l1, d);
20       } else {
21         l2 = new Node(l2, d);
22       }
23   }
24   while (l1 != null) {
25     table[l1.data] = table[l1.data] + 1;
26     l1 = l1.next;
27   }
28 }
```

⑧ Tupled References

```
01 public static class Node {
02   final Node next;
03   final int data;
04
05   public Node(Node next, int data) {
06     this.next = next;
07     this.data = data;
08   }
09 }
10
11 public static void main(String[] args) {
12   final int size = 10;
13   final int[] table = new int[size] ;
14   Node l1 = null;
15   Node l2 = null;
16   for (int i=0; i<args.length; i++) {
17       int d = Integer.parseInt(args[i]);
18       if (d >= 0 && d < size) {
19         l1 = new Node(l1, d);
20       } else {
21         l2 = new Node(l2, d);
22       }
23   }
24   while (l1 != null) {
25     table[l1.data] = table[l1.data] + 1;
26     l1 = l1.next;
27   }
28 }
```

Possible state/loop invariant:
`l1 == null || l1.allocSite == 19`

⑧ Tupled

```
01 public static class Node {
02   final Node next;
03   final int data;
04
05   public Node(No
06     this.next =
07     this.data =
08   }
09 }
10
11 public static void main(String[] args) {
12   final int size = 10;
13   final int[] table = new int[size] ;
14   Node l1 = null;
15   Node l2 = null;
16   for (int i=0; i<args.length; i++) {
17       int d = Integer.parseInt(args[i]);
18       if (d >= 0 && d < size) {
19         l1 = new Node(l1, d);
20       } else {
21         l2 = new Node(l2, d);
22       }
23   }
24   while (l1 != null) {
25     table[l1.data] = table[l1.data] + 1;
26     l1 = l1.next;
27   }
28 }
```

Possible class invariant:

```
this.allocSite == 19 ==>
(next == null || next.allocSite == 19) &&
(0 <= data && data < 10)
```

Possible state/loop invariant:

```
l1 == null || l1.allocSite == 19
```

⑧ Tupled References

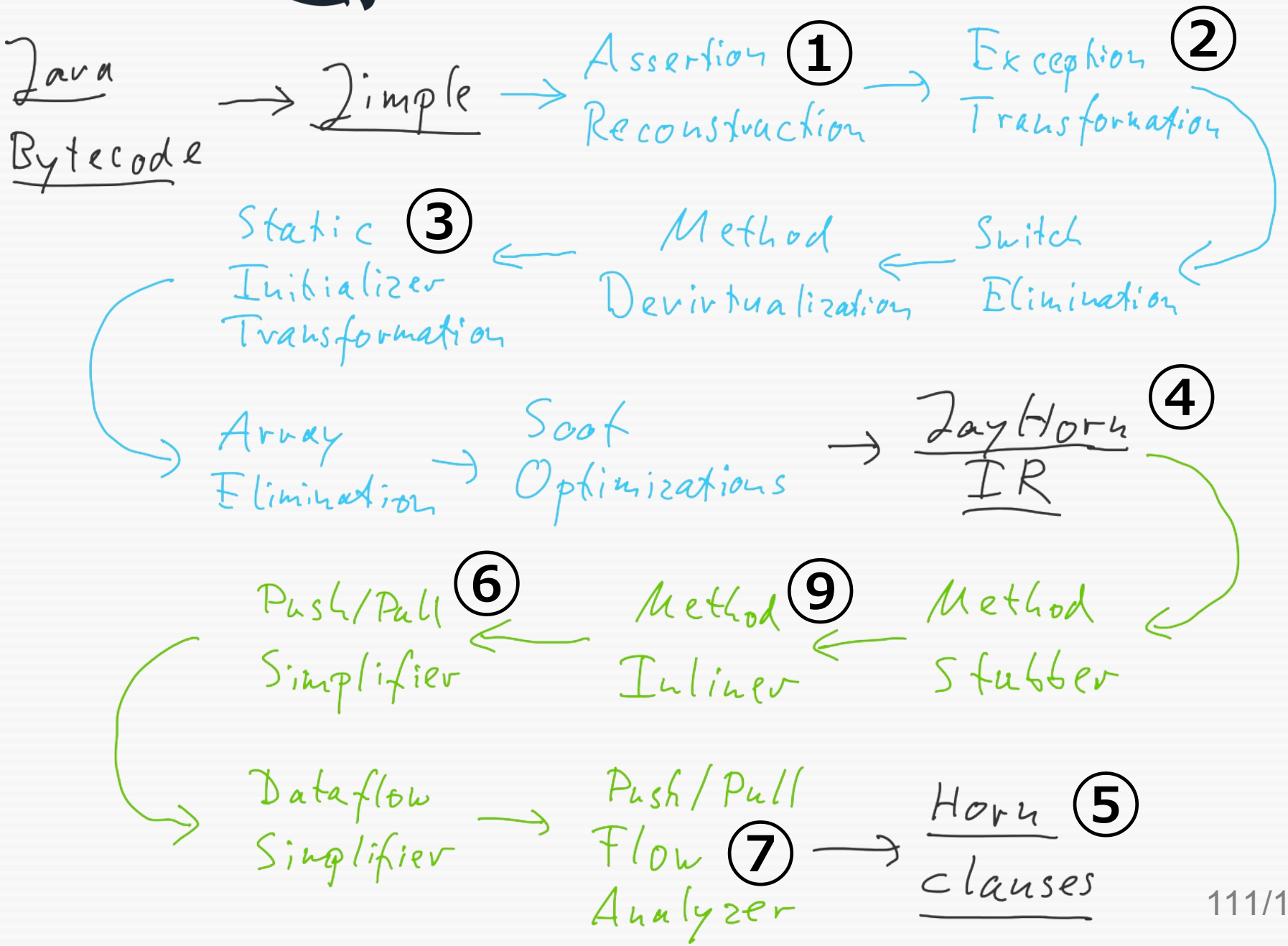
```
01 public static class Node {
02   final Node next;
03   final int data;
04
05   public Node(Node next, int data) {
06     this.next = next;
07     this.data = data;
08   }
09 }
10
11 public static void main(String[] args) {
12   final int size = 10;
13   final int[] table = new int[size] ;
14   Node l1 = null;
15   Node l2 = null;
16   for (int i=0; i<args.length; i++) {
17       int d = Integer.parseInt(args[i]);
18       if (d >= 0 && d < size) {
19         l1 = new Node(l1, d);
20       } else {
21         l2 = new Node(l2, d);
22       }
23   }
24   while (l1 != null) {
25     table[l1.data] = table[l1.data] + 1;
26     l1 = l1.next;
27   }
28 }
```

This example at the moment also needs option `-inline-size 50`

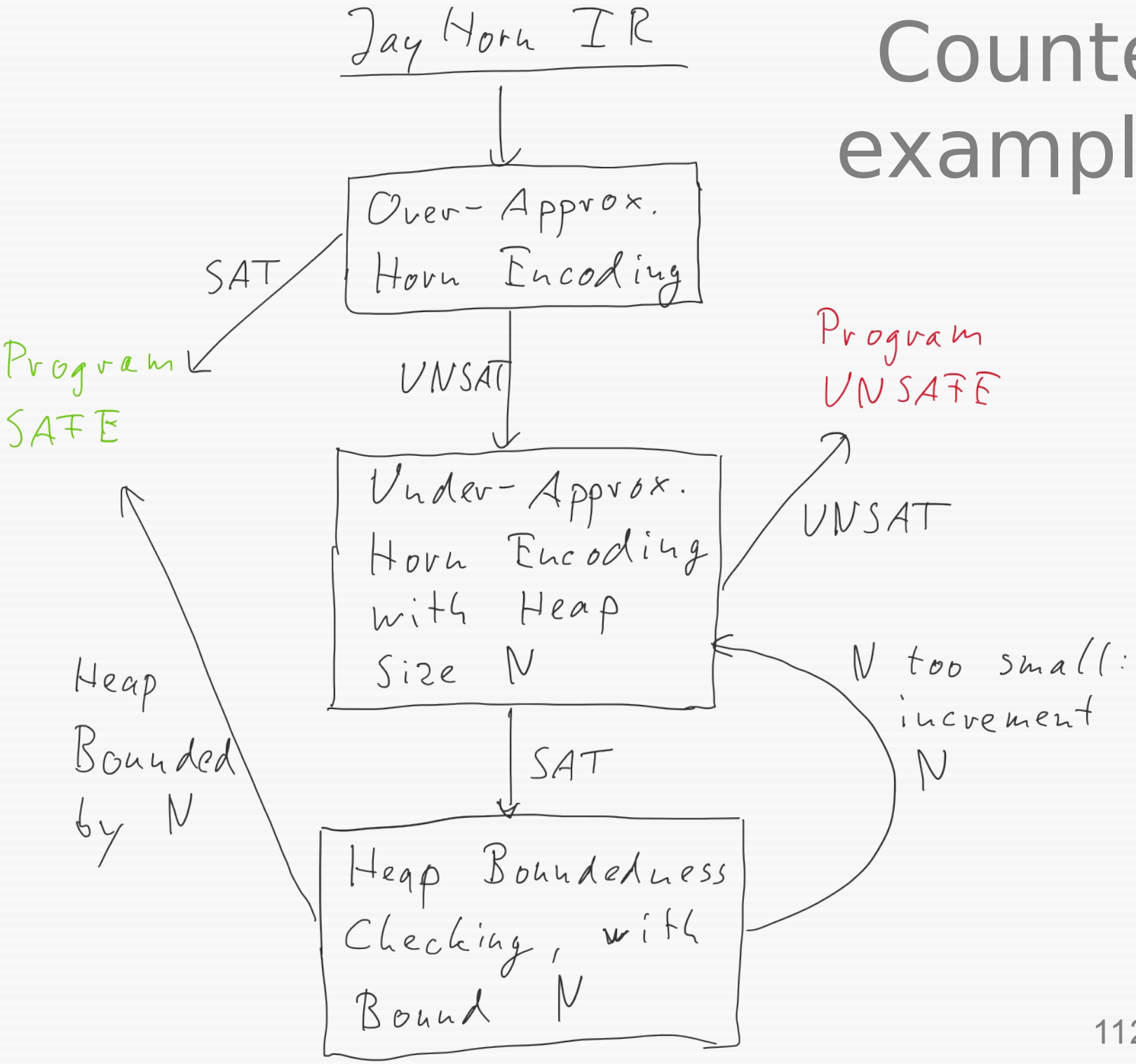
⑨ Method Inlining

- Simple, but extremely useful
  - Enables more effective `push`/`pull` movement (which is intra-proc.)
  - Reduces # of method contracts
- `-inline-size` <*n*>
  - Inline methods with at most <*n*> statements
- `-inline-count` <*n*>
  - Inline methods called at most <*n*> times

Data-Flow


Java Bytecode
Jimple
Assertion Reconstruction ①
Exception Transformation ②
Switch Elimination
Method Devirtualization
Static Initializer Transformation ③
Array Elimination
Soot Optimizations
JayHorn IR ④
Method Stubber
Method Inliner ⑨
Push/Pull Simplifier ⑥
Dataflow Simplifier
Push/Pull Flow Analyzer ⑦
Horn clauses ⑤

Counter-examples



Timeline

- Sturm und Drang: 2015 – 2017
  - Intensive implementation period
  - Loads of news ideas
- SV-COMP 2018
- Renewed activity: 2018 – now
  - Making JayHorn useful ... *(ongoing)*

The original JayHorn crowd:

- Core team:
  Temesghen Kahsai, Rody Kersten, PR, Huascar Sanchez, Martin Schäf
- Contributors:
  Daniel Dietsch, Valentin Wüstholz

Current team (since 2018):

- Main developers:
  Temesghen Kahsai, PR, Martin Schäf
- Contributors:
  Hossein Hojjat, Ali Shamakhi

Funding

- Vetenskapsrådet

UPPSALA UNIVERSITET

The Next Steps …

Java Features

- Full support for enums
- Improved array handing
- Sound machine arithmetic handling

- Java API modelling

- Concurrency

Support for Strings *(ongoing)*

- Need native Horn solver support:
  - Strings directly, or
  - Algebraic data-types
- Tupled references can be used to get effective value semantics
  - Same for boxed data-types (Integer, etc.)

Reworking the Heap Encoding

- There is also TriCera:
  the C version of JayHorn
- **Idea:**
  Make the heap encoding independent of JayHorn
- Long-term goal:
  an SMT-LIB theory of heap

Certificates

- JayHorn is a complicated system
  → to be trusted?

- **Idea:**
  Output JML annotations that can be checked by an independent tool

Case Studies ...

Conclusions

- **Constrained Horn Clauses:**
  A paradigm to build verifiers
- **JayHorn:**
  An automatic verifier for Java

- Download and try!
- We are also looking for further contributors!
- https://github.com/jayhorn/jayhorn