Automated Instantiation of Control Flow Tracing Exercises

ThEdu '21

Clemens Eisenhofer Martin Riener

Technische Universität Wien, Austria

Context

- Introduction to programming course (CS 1)
- Understanding and writing Java code
- Course consists of 3 parts:
  - Lab classes - discussion of homework
  - Programming exams
  - *Auto-graded tests about given code*

Automatically Graded Tests

- Done via Moodle (e-Learning platform)
  - What is the result/output of the code?
  - Which of these code fragments produces . . . as result/output?
  - Which programs have the same (concrete) results/outputs?
  - Does the result/output change if we replace line . . . by . . . ?
  - Which starting value of . . . produces the given result?
  - . . .
- Question pool is limited
  - Writing new ones is error-prone and tedious
  - Hard to guarantee same difficulty
  - Should have didactic value

Automatically Graded Tests: Example

Consider the following code fragment:

```
int a = 0;
int b = 0;
int limit = /* ? */;
int inc =   /* ? */;
int start = /* ? */;
for (int i = start; i < limit; i += inc) {
    // Loop A
    if (i % 2 == 0)
        a++;
    if (i % 3 == 0)
        a++;
}

for (int i = start; i < limit; i += inc) {
    // LOOP B
    if (i % 2 == 0)
        b++;
    else if (i % 3 == 0)
        b++;
}

System.out.print(a+" "+b);
```

Which of the values for limit, inc, start lead to different outputs of a and b in the end?

Select one or more:

- ☐ int limit = 10; int inc = 5; int start = 3;
- ☐ int limit = 17; int inc = 5; int start = 1;
- ☐ int limit = 15; int inc = 5; int start = 3;
- ☐ int limit = 14; int inc = 3; int start = 3;

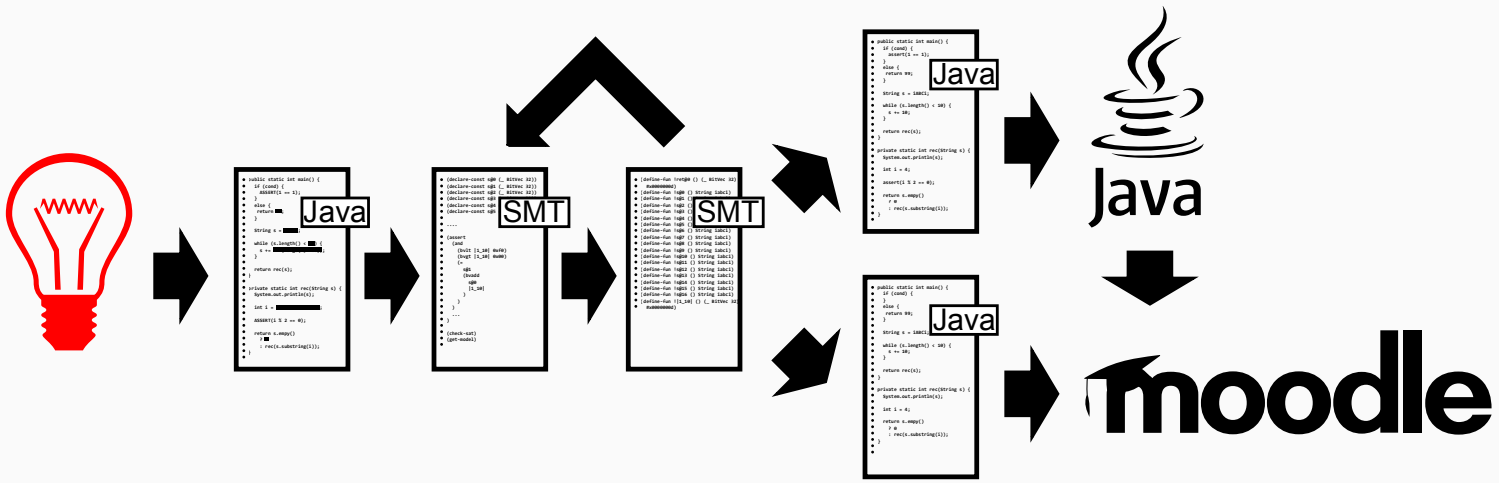
Aim of Tatsu

- Generate multiple instances of a program/exercises
- As little additional (manual) effort required as possible
- Exercises should be approx. equally difficult

```
int a = 0;
int b = 0;
int limit = 20;
int inc = 2;
int start = 0;

for (int i = start; i < limit; i += inc) {
   // Loop A
   if (i % 2 == 0) a++;
   if (i % 3 == 0) a++;
}

for(int i = start; i < limit; i += inc) {
   // LOOP B
   if (i % 2 == 0) b++;
   else if (i % 3 == 0) b++;
}

System.out.print(a+" "+b);
```

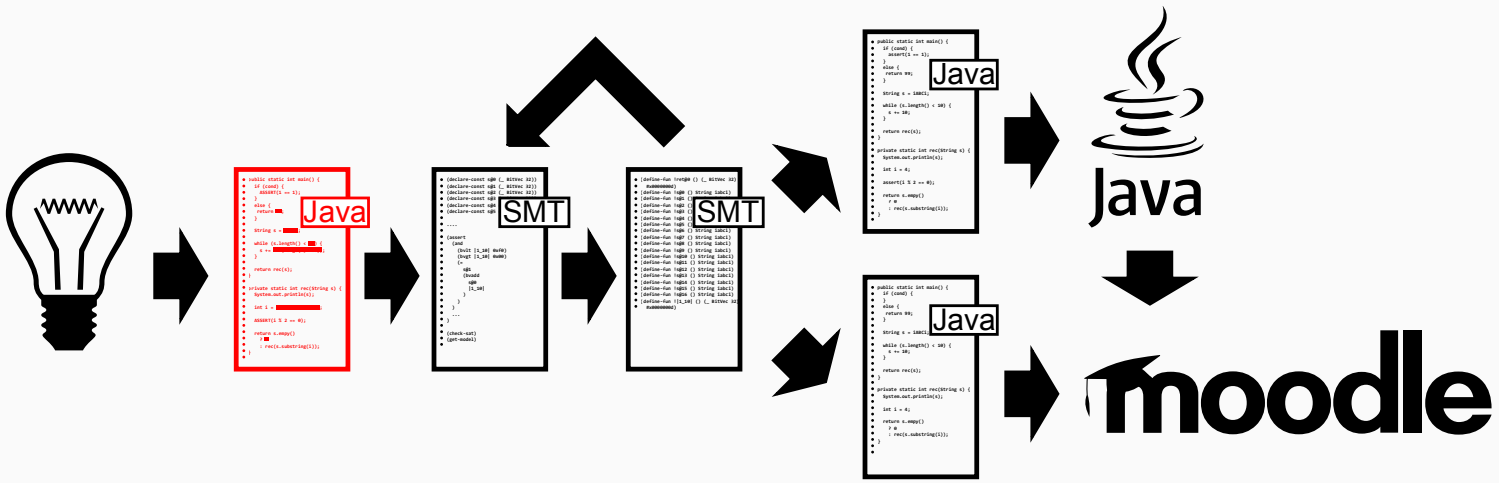
```
int a = 0;
int b = 0;
int limit = 1743;
int inc = 13;
int start = 17;

for (int i = start; i < limit; i += inc) {
   // Loop A
   if (i % 2 == 0) a++;
   if (i % 3 == 0) a++;
}

for(int i = start; i < limit; i += inc) {
   // LOOP B
   if (i % 2 == 0) b++;
   else if (i % 3 == 0) b++;
}

System.out.print(a+" "+b);
```

From Idea to Moodle



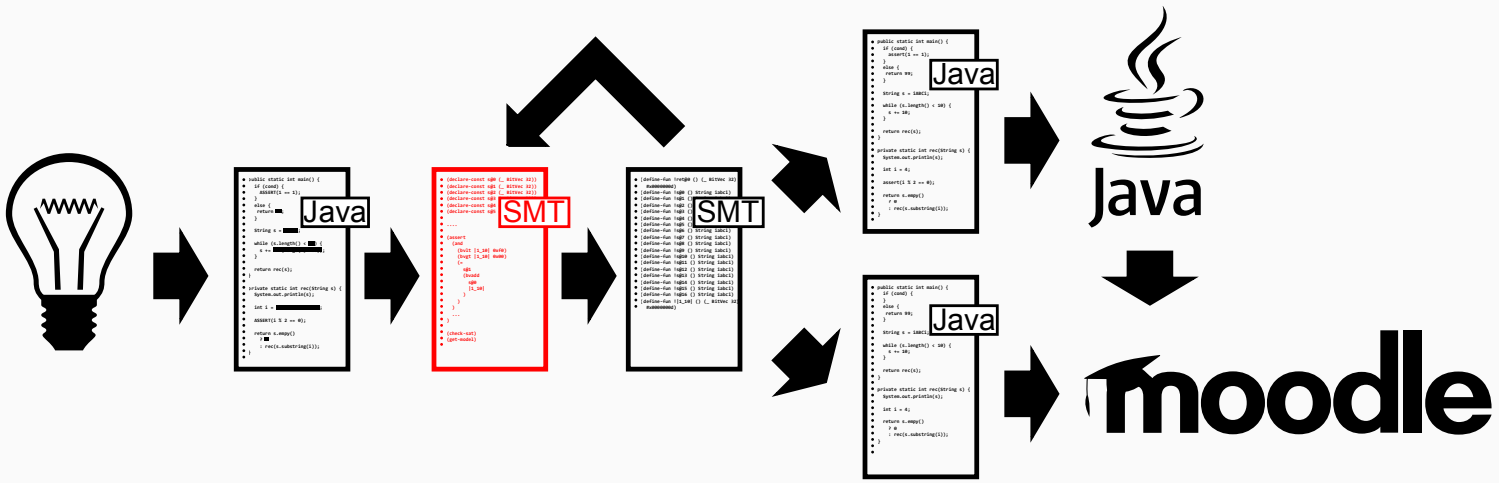

```java
int a = 0;
int b = 0;
int limit = 20;
int inc = 2;
int start = 0;

for (int i = start; i < limit; i += inc) {
   // Loop A
   if (i % 2 == 0) a++;
   if (i % 3 == 0) a++;
}

for(int i = start; i < limit; i += inc) {
   // LOOP B
   if (i % 2 == 0) b++;
   else if (i % 3 == 0) b++;
}

System.out.print(a+" "+b);
```

From Idea to Moodle



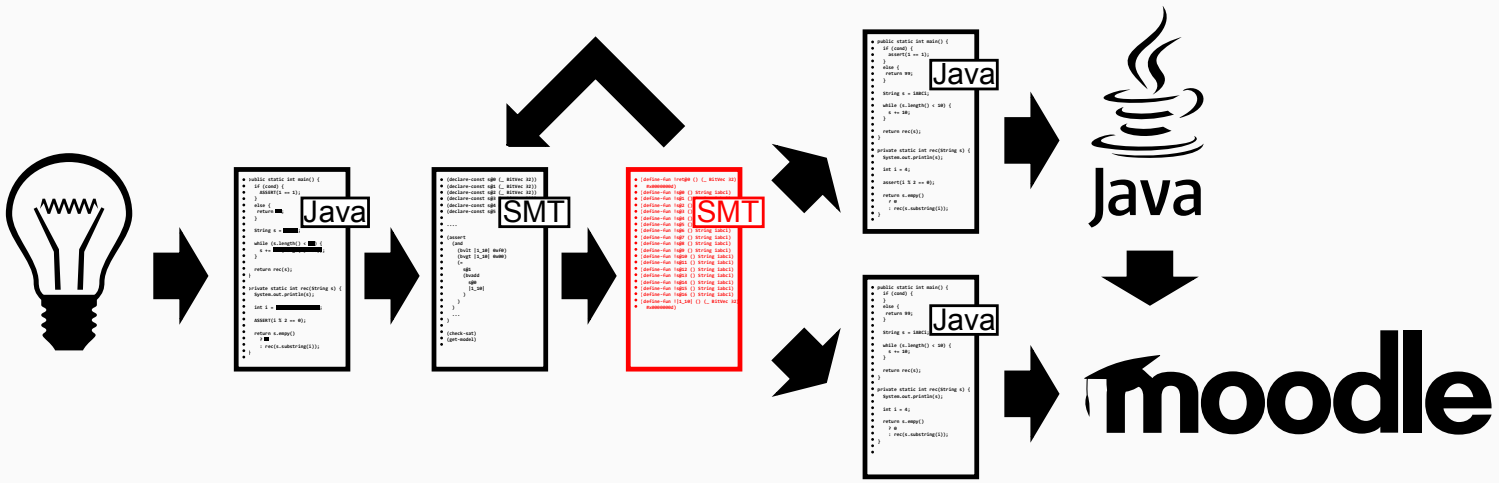

```
int a = 0;
int b = 0;
int limit = INT(range(10, 20));
int inc = INT(range(2, 5));
int start = INT(range(0, 3));

LOOP(range(1, 20));
for (int i = start; i < limit; i += inc) {
   // Loop A
   if (i % 2 == 0) a++;
   if (i % 3 == 0) a++;
}

LOOP(range(1, 20));
for(int i = start; i < limit; i += inc) {
   // LOOP B
   if (i % 2 == 0) b++;
   else if (i % 3 == 0) b++;
}
ASSERT(a != b);
System.out.print(a+" "+b);
```

From Idea to Moodle



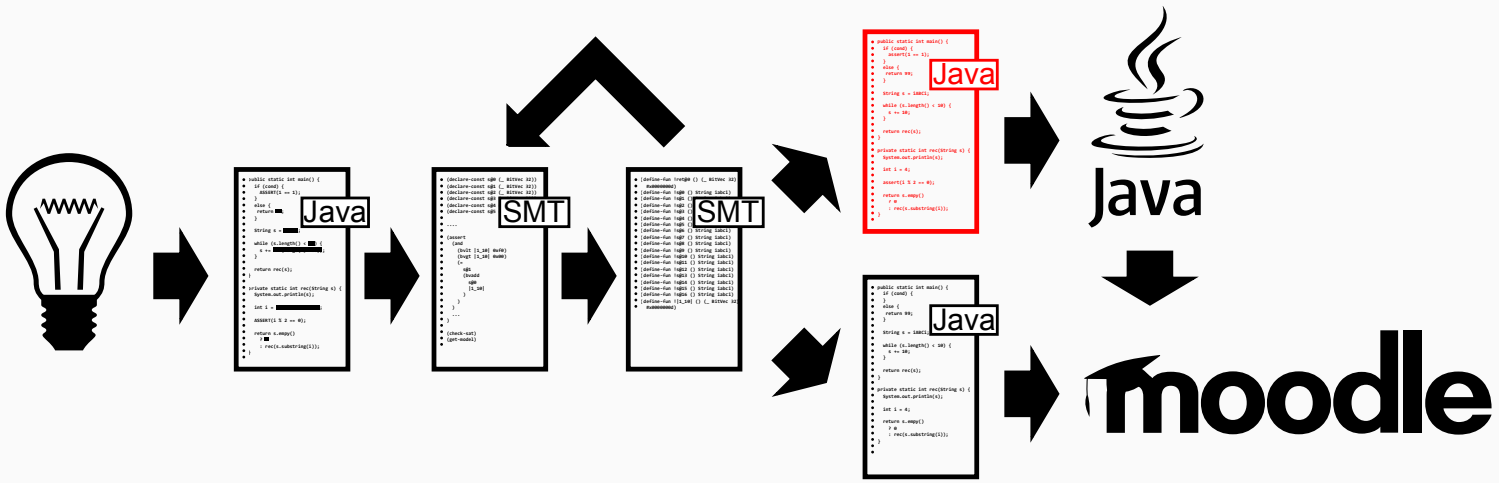

```
(declare-fun b@0@10 () (_ BitVec 32))
(declare-fun a@0@20 () (_ BitVec 32))
(declare-fun i@1@2 () (_ BitVec 32))
(declare-fun i@1@10 () (_ BitVec 32))
(declare-fun b@0@2 () (_ BitVec 32))
..........
(assert (let ((a!1 (and (ite (=
    (bvsrem i@0@9 #x00000002) #x00000000)
      (= a@0@19 (bvadd a@0@18 #x00000001))
      (= a@0@19 a@0@18))
      (ite (= (bvsrem i@0@9 #x00000003)
                                #x00000000)
    ..........
  (and (bvsge |3_13| #x0000000a)
       (bvsle |3_13| #x00000014)
       ..........
       a!26
       (not (= a@0@20 b@0@10))))))))))))))
(check-sat)
(get-model)
```

From Idea to Moodle



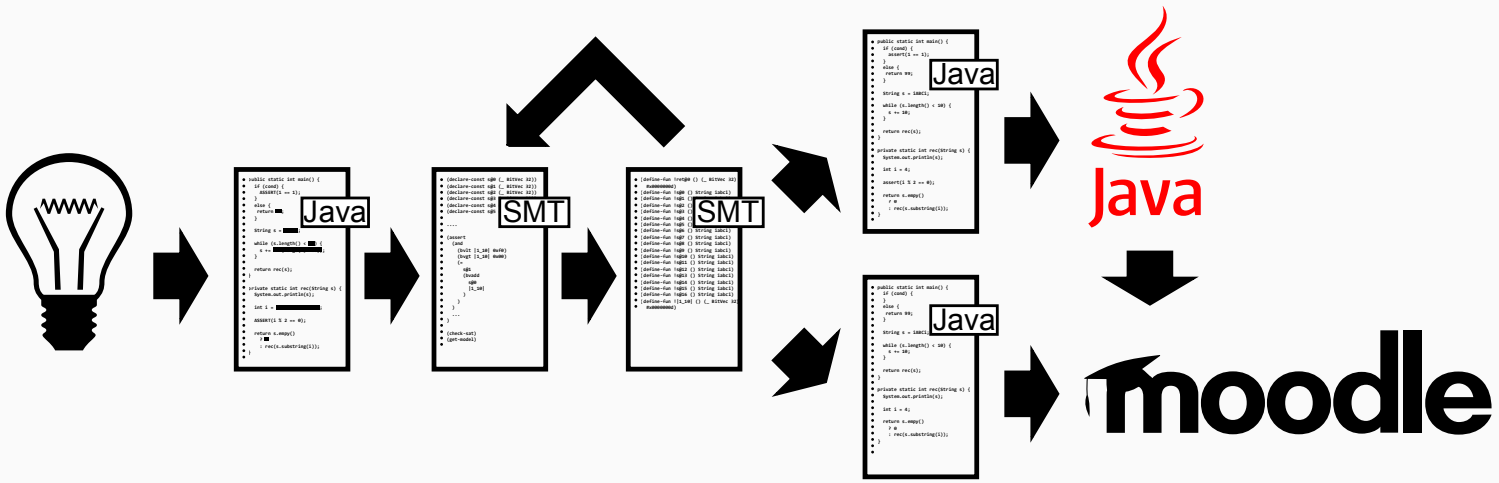

```
(define-fun b@0@5 () (_ BitVec 32)
  #x00000005)
(define-fun a@0@17 () (_ BitVec 32)
  #x0000000c)
(define-fun b@0@10 () (_ BitVec 32)
  #x0000000a)
(define-fun a@0@15 () (_ BitVec 32)
  #x0000000b)
(define-fun i@1@7 () (_ BitVec 32)
  #x0000000e)
(define-fun a@0@20 () (_ BitVec 32)
  #x0000000e)
..............
(define-fun i@1@1 () (_ BitVec 32)
  #x00000002)
(define-fun i@0@5 () (_ BitVec 32)
  #x0000000a)
(define-fun a@0@11 () (_ BitVec 32)
  #x00000008)
```

From Idea to Moodle



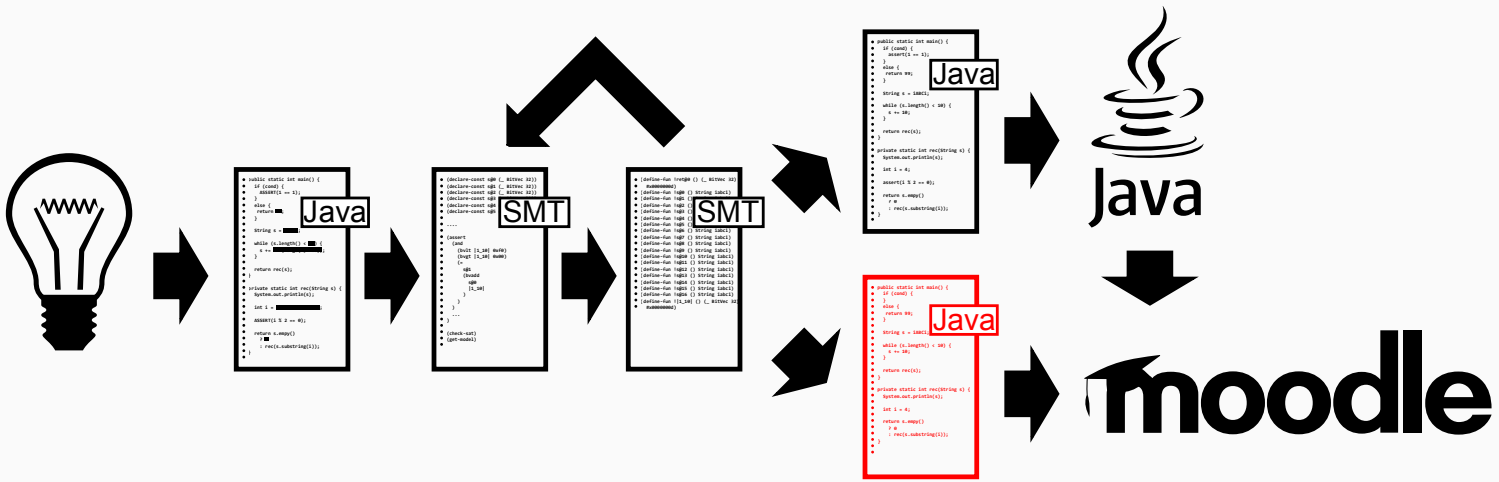

```
int a = 0;
int b = 0;
int limit = 16;
int inc = 2;
int start = 0;
int __cnt_7_1 = 0;
for (int i = start; i < limit; i += inc) {
    __cnt_7_1++;
    if (i % 2 == 0)
        a++;
    if (i % 3 == 0)
        a++;
}
if (!(((__cnt_7_1 >= 1) &&
                    (__cnt_7_1 <= 20))))
    throw new AssertionError("1");
.........
if (!(a != b))
    throw new AssertionError("0");
__out += a + " " + b;
```

From Idea to Moodle



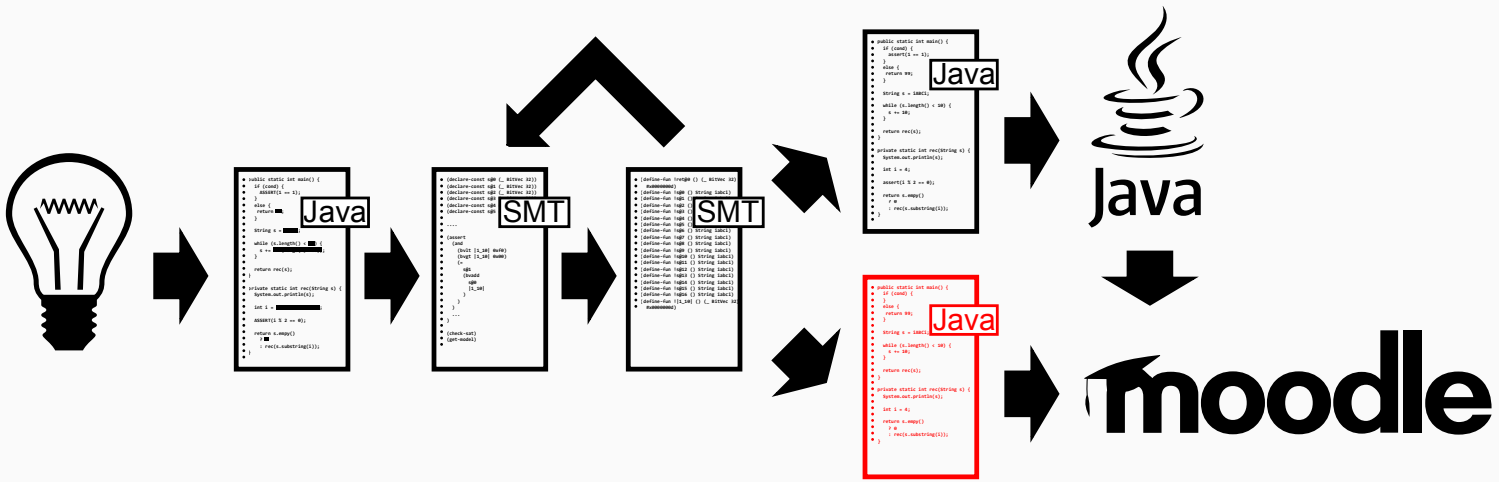

```
Returned value: null
Printed value: “11 8”
Process finished with exit code 0.
```

From Idea to Moodle




```java
int a = 0;
int b = 0;
int limit = 16;
int inc = 2;
int start = 0;

for (int i = start; i < limit; i += inc) {
    // Loop A
    if (i % 2 == 0) a++;
    if (i % 3 == 0) a++;
}

for (int i = start; i < limit; i += inc) {
    // LOOP B
    if (i % 2 == 0) b++;
    else if (i % 3 == 0) b++;
}
System.out.print(a+" "+b);
```

From Idea to Moodle




```
int a = 0;
int b = 0;
int limit = 20;
int inc = 2;
int start = 0;
for (int i = start;
    i < limit; i += inc) {
    // Loop A
    if (i % 2 == 0)
        a++;
    if (i % 3 == 0)
        a++;
}
```

```
int a = 0;
int b = 0;
int limit = 10;
int inc = 5;
int start = 0;
for (int i = start;
    i < limit; i += inc) {
    // Loop A
    if (i % 2 == 0)
        a++;
    if (i % 3 == 0)
        a++;
}
```

```
int a = 0;
int b = 0;
int limit = 10;
int inc = 4;
int start = 2;
for (int i = start;
    i < limit; i += inc) {
    // Loop A
    if (i % 2 == 0)
        a++;
    if (i % 3 == 0)
        a++;
}
```

```
int a = 0;
int b = 0;
int limit = 17;
int inc = 5;
int start = 1;
for (int i = start;
    i < limit; i += inc) {
    // Loop A
    if (i % 2 == 0)
        a++;
    if (i % 3 == 0)
        a++;
}
```

```
int a = 0;
int b = 0;
int limit = 10;
int inc = 2;
int start = 2;
for (int i = start;
    i < limit; i += inc) {
    // Loop A
    if (i % 2 == 0)
        a++;
    if (i % 3 == 0)
        a++;
}
```

```
int a = 0;
int b = 0;
int limit = 16;
int inc = 4;
int start = 0;
for (int i = start;
    i < limit; i += inc) {
    // Loop A
    if (i % 2 == 0)
        a++;
    if (i % 3 == 0)
        a++;
}
```

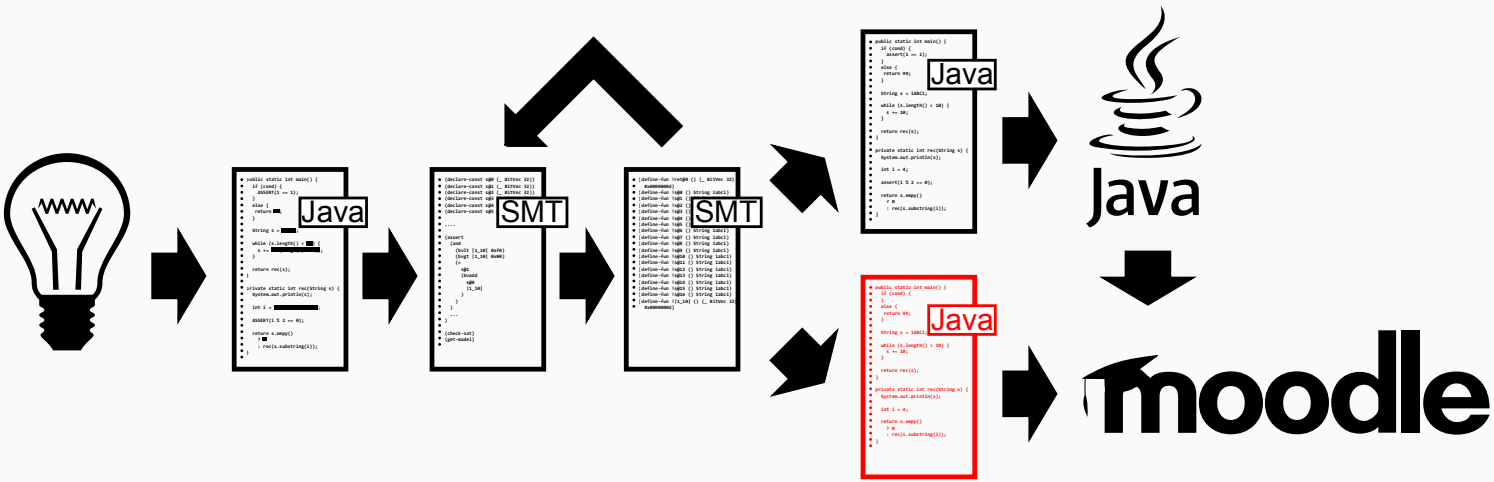
```
int a = 0;
int b = 0;
int limit = 16;
int inc = 5;
int start = 2;
for (int i = start;
    i < limit; i += inc) {
    // Loop A
    if (i % 2 == 0)
        a++;
    if (i % 3 == 0)
        a++;
}
```

```
int a = 0;
int b = 0;
int limit = 13;
int inc = 5;
int start = 1;
for (int i = start;
    i < limit; i += inc) {
    // Loop A
    if (i % 2 == 0)
        a++;
    if (i % 3 == 0)
        a++;
}
```

```
int a = 0;
int b = 0;
int limit = 14;
int inc = 5;
int start = 1;
for (int i = start;
    i < limit; i += inc) {
    // Loop A
    if (i % 2 == 0)
        a++;
    if (i % 3 == 0)
        a++;
}
```

```
int a = 0;
int b = 0;
int limit = 10;
int inc = 5;
int start = 1;
for (int i = start;
    i < limit; i += inc) {
    // Loop A
    if (i % 2 == 0)
        a++;
    if (i % 3 == 0)
        a++;
}
```

From Idea to Moodle



From Idea to Moodle

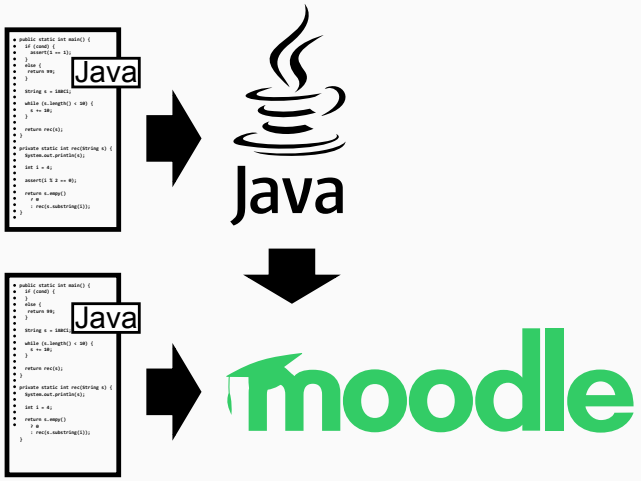
Consider the following code fragment:

```
int a = 0;
int b = 0;
int limit = /* ? */;
int inc =   /* ? */;
int start = /* ? */;
for (int i = start; i < limit; i += inc) {
    // Loop A
    if (i % 2 == 0)
        a++;
    if (i % 3 == 0)
        a++;
}

for (int i = start; i < limit; i += inc) {
    // LOOP B
    if (i % 2 == 0)
        b++;
    else if (i % 3 == 0)
        b++;
}

System.out.print(a+" "+b);
```

Which of the values for limit, inc, start lead to different outputs of a and b in the end?

Select one or more:

- ☐ int limit = 10; int inc = 5; int start = 3;
- ☐ int limit = 17; int inc = 5; int start = 1;
- ☐ int limit = 15; int inc = 5; int start = 3;
- ☐ int limit = 14; int inc = 3; int start = 3;



The supported language

- Subset of the Java programming language
- Variables (including common operations)

  `boolean`, `byte`, `short`, `int`, `long`, `int[]`

  (`int[][]`, `String`, `String[]`)
- `if-else`, `while`, `do-while`, `for`, ternary-operator
- `return`, `break`, `continue`
- function calls, recursion
- *Tatsu specific keywords/annotations* (Java compatible syntax)

The supported language

- Subset of the Java programming language
- Roughly speaking:
- Variables (including common operations)

  `boolean`, `byte`, `short`, `int`, `long`, `int[]`

  (`int[][]`, `String`, `String[]`)
- `if-else`, `while`, `do-while`, `for`, ternary-operator
- `return`, `break`, `continue`
- function calls, recursion
- *Tatsu specific keywords/annotations* (Java compatible syntax)

Additional keywords/annotations

- `ASSERT(x);`

  ... eliminates all instantiations where x is not satisfied when reached

  e.g., `ASSERT((x % 2) == 0);`

Additional keywords/annotations

- Placeholders: `INT(v)`, `INTARRAY(l, v)`, `BOOLEAN()`, ...
  - ... represents a constant of the corresponding type that is constrained by its arguments

  e.g., `INT(range(0, 100))` - an integer constant in $[0, 100]$

  e.g., `INTARRAY(range(0, 100), range(3, 5)` - an integer array with 3, 4, or 5 elements being in $[0, 100]$

Additional keywords/annotations

- `LOOP(x);`, `@REC(x)`
  - ... determines how many recursive calls/loop iterations should be considered $\rightarrow$ User-defined upper bounds
  - e.g., `LOOP(range(0, 5))` - the the following loop might iterate between $[0, 5]$ times

Additional keywords/annotations

- `ASSERT(x);`
  - ... eliminates all instantiations where x is not satisfied when reached

  e.g., `ASSERT((x % 2) == 0);`
- Placeholders: `INT(v)`, `INTARRAY(l, v)`, `BOOLEAN()`, ...
  - ... represents a constant of the corresponding type that is constrained by its arguments

  e.g., `INT(range(0, 100))` - an integer constant in $[0, 100]$

  e.g., `INTARRAY(range(0, 100), range(3, 5)` - an integer array with 3, 4, or 5 elements being in $[0, 100]$
- `LOOP(x);`, `@REC(x)`
  - ... determines how many recursive calls/loop iterations should be considered $\rightarrow$ User-defined upper bounds

  e.g., `LOOP(range(0, 5))` - the the following loop might iterate between $[0, 5]$ times
- and some more ...

Encoding

- Generating logical representation:
  - Add assertions for the placeholders
  - Eliminate `break`, `continue`, and `return` by program transformation
  - Loop unwinding and function call inlining
  - Conversion to a variant of SSA form
  - Optimization during transformation
- SSA form $\rightarrow$ logical representation $\rightarrow$ SMT solver
- Generated model $\rightarrow$ AST transformation

Unwinding

- Loops are unwinded a given number of times:

```
LOOP(range(10, 15));
while (loop-condition) {
  stmts;
}
```

```
ASSERT(loop-condition);
stmts;
...                                          } 10×
ASSERT(loop-condition);
stmts;
if (loop-cond) {
  stmts;
  if (loop-cond) {
    ...                                      } 5×
    if (loop-cond) ASSERT(false);
  }
}
```

Inlining

- Function calls are inlined a given number of times:

```
@REC(5)
static int recFunc(int x) {
  stmts1;
  z = recFunc(y);
  stmts2;
  return val;
}

static void func() {
  recFunc(INT(range(0, 10)));
}
```

```
static void func() {
  x_1 = INT(range(0, 10));
  stmts1;
  x_2 = y_1;          } 5×
  stmts1;
  ...
  ASSERT(false);
  ...
  ret_5 = val_5;
  z_4 = ret_5;        } 5×
  stmts2;
  ...
  stmts2;
}
```

Transformation

- `break`, `continue`, and `return` are eliminated by transformation:

```
LOOP(...);
while (loopCond) {
  if (ifCond) {
    stmts1;
    break;
  }
  stmts2;
}
```

```
LOOP(...);
while (loopCond && !_ifCond) {
  if (ifCond) {
    _ifCond = ifCond;
    stmts1;
  }
  if (!_ifCond) {
    stmts2;
  }
}
```

Variable encoding

- Every state is represented by its own SMT constant (“SSA”):

  `x@[version]@[writecount]`

- Allows reconstruction of complete data-flow

  $\rightarrow$ Useful for post-processing

Variable encoding

- Non-Arrays:

Java	boolean	byte	short	int	long	String
SMT	Bool	BitVec 8	BitVec 16	BitVec 32	BitVec 64	String

e.g., `int i = INT(range(0, 100)); i = i + 1;`

$\rightarrow$ `i@0@0` $\geq$ `0` $\wedge$ `i@0@0` $\leq$ `100` $\wedge$ `i@0@1 = i@0@0 + 1`

- Arrays (Pointer, Length) + Entry in global "heap-array(s)":

Java	int[ ]	int[ ][ ]	String[ ]
SMT	(BitVec 32, BitVec 32)		
	Array BitVec 32	3 · (Array BitVec 32)	Array String

Variable encoding - Arrays

e.g., `int[] y = new int[4]; y[0] = 17;`

$\rightarrow$:

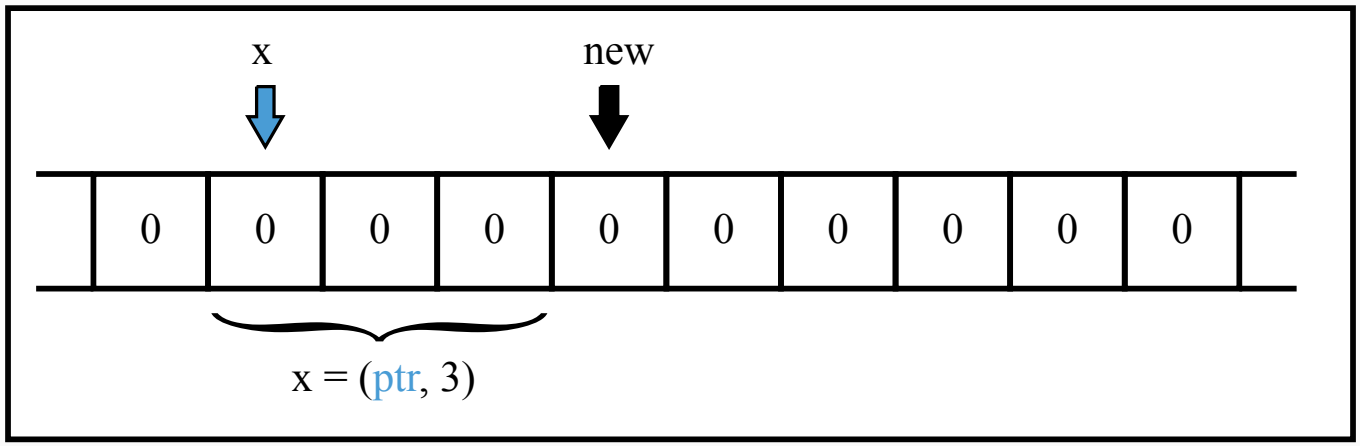
$$\texttt{y@0@0.ptr} = \texttt{new@1} \wedge \texttt{y@0@0.len} = 4 \wedge$$
$$\texttt{new@2} = \texttt{new@1} + 4 \wedge$$
$$\texttt{gIntArr@0@1} = \texttt{store}(\texttt{gIntArr@0@0}, \texttt{y@0@0.ptr}, 13)$$

Variable encoding - Arrays

e.g., `int[] y = new int[4]; y[0] = 17;`

$\rightarrow$:

$$\texttt{y@0@0.ptr = new@1} \wedge \texttt{y@0@0.len = 4} \wedge$$
$$\texttt{new@2 = new@1 + 4} \wedge$$
$$\texttt{gIntArr@0@1 = store(gIntArr@0@0, y@0@0.ptr, 13)}$$

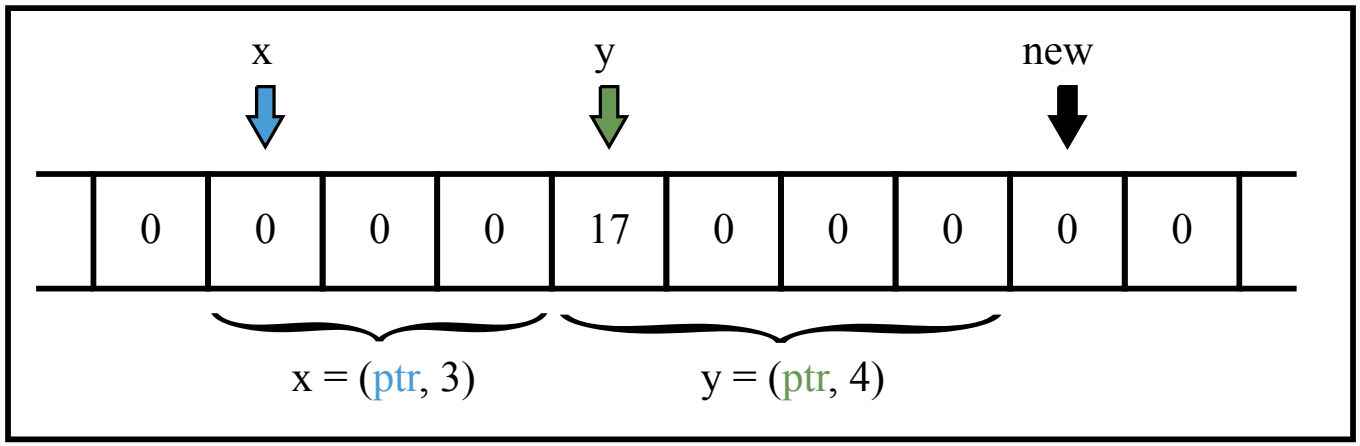


Variable encoding - Arrays

e.g., `int[] y = new int[4]; y[0] = 17;`

$\rightarrow$:

$$\texttt{y@0@0.ptr} = \texttt{new@1} \wedge \texttt{y@0@0.len} = 4 \wedge$$
$$\texttt{new@2} = \texttt{new@1} + 4 \wedge$$
$$\texttt{gIntArr@0@1} = \texttt{store}(\texttt{gIntArr@0@0}, \texttt{y@0@0.ptr}, 13)$$



Live Demo

- Switching to console ...

Conclusion and Future Work

- Tatsu is a open-source command-line tool and Java API to automatically instantiate code templates:
  `https://git.logic.at/ep1-tools/tatsu-generator`
- Generates total correct Java codes
- Plans for the future:
  - Support for Strings (and Characters) via integer arrays (C-like)
  - More Java features (especially object-orientation)
  - Automatically generating feedback for students via BMC
  - Generating arbitrary number of iterations/recursive calls by manually providing invariants

More Examples

```
@MAIN
static int[] start() {
  int[] input = INTARRAY(list(6), range(-25,25));
  ASSERT(__distinct(input, 6));
  return mystery(input, 0, input.length-1);
}

@REC(5)
static int[] mystery(int[] data, int from, int to) {
    if (from == to) return new int[] { data[from], data[from] };

    int[] left = mystery(data, from, (from + to) / 2);
    int[] right = mystery(data, (from + to) / 2 + 1, to);

    int r[] = new int[2];
    r[0] = left[0] < right[0] ? left[0] : right[0];
    r[1] = left[1] < right[1] ? right[1] : left[1];
        ASSERT(r[1] - r[0] > 5);
    return r;
}
```

More Examples

```
@MAIN
static int[] main() {
  int[] values = INTARRAY( range(8,12), range(1,30));
  ASSUME(__distinct(values, 100 ));
  ASSUME(__asc(values, 100));
  ASSUME(values[0] == 1 );
  ASSUME(values[values.length-1] == values.length );

  int[] instr = INTARRAY(list(10), range(0,12) );
  ASSUME(__distinct(instr, 100 ));

  shuffle(values, instr, 0);
  ASSERT(values[0] == 7);
  return values;
}

@REC(20)
static void shuffle(int[] values, int[] instr, int start) {
  if (start >= instr.length) return;
  int t = values[0];
  values[0] = values[instr[start]];
  values[instr[start]] = t;
  shuffle(values, instr, start+1);
}
```

Problem with Arrays

- Consider the following (unsat) code:

```
void test(int[] arr) {
    arr[0] = 2;
}

void main(){
    int[] arr = new int[1];
    arr[0] = 1;
    test(arr);
    ASSERT(arr[0] == 1);
}
```