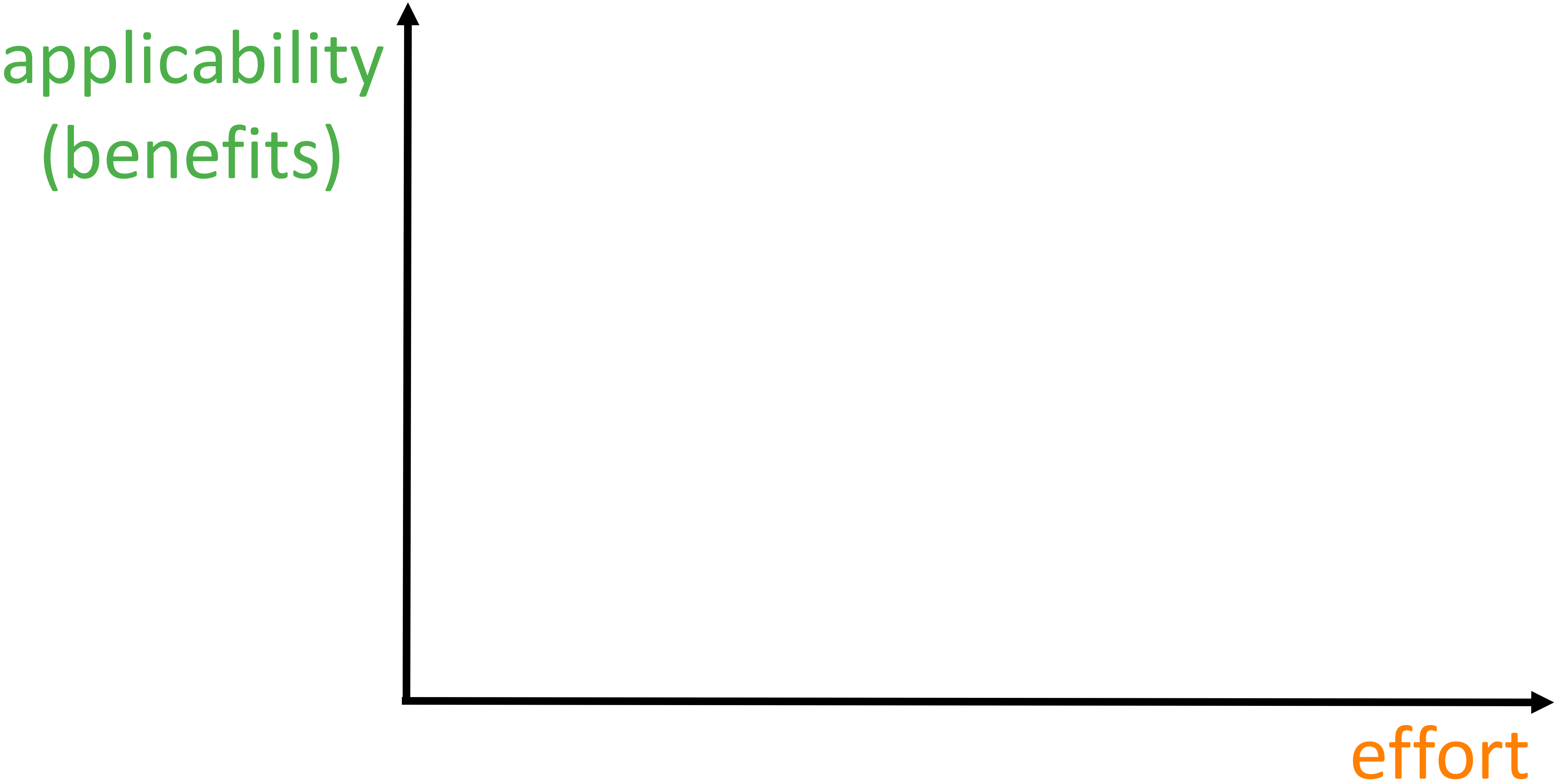
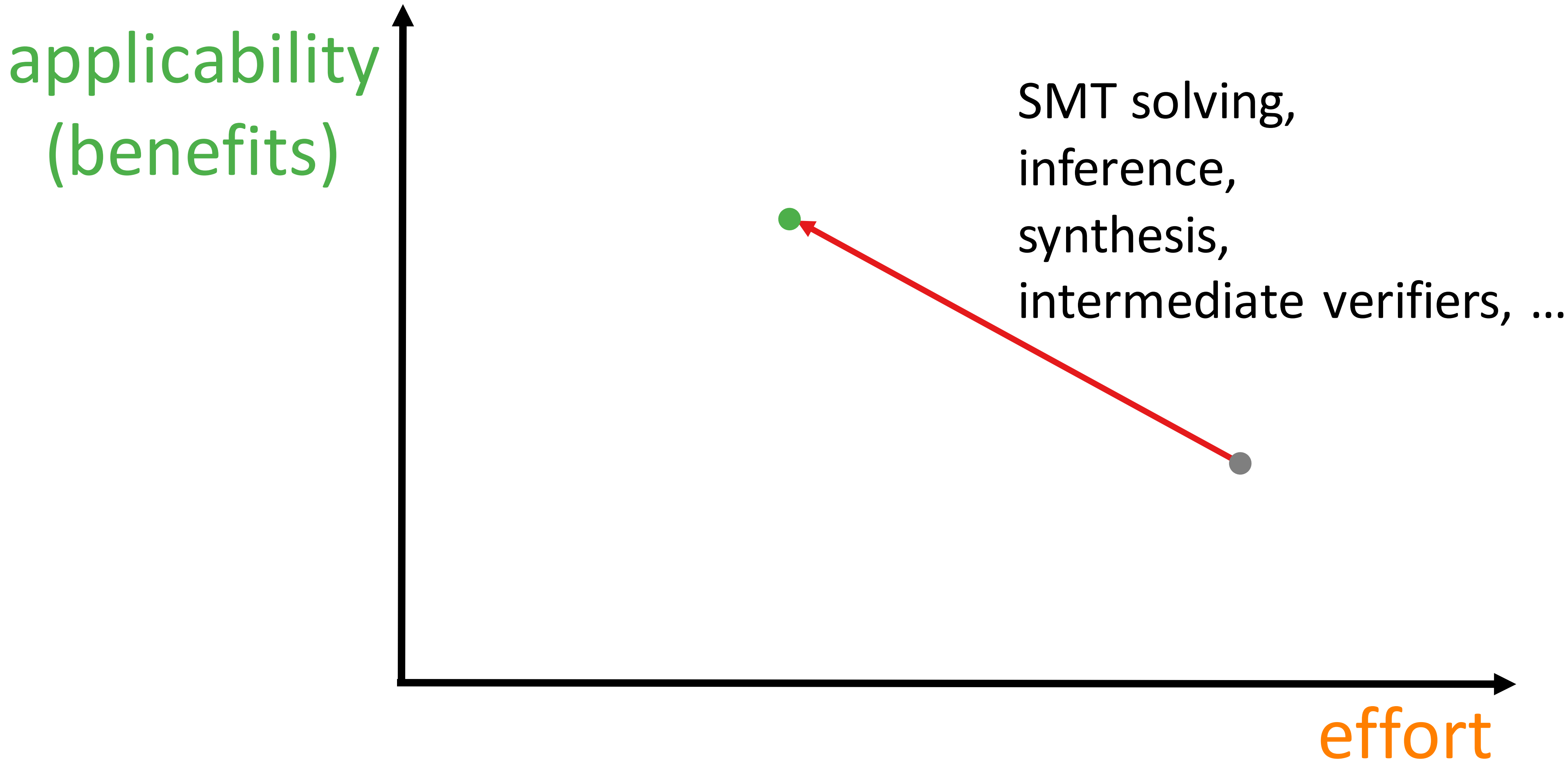
Practical Formal Methods: Are We There Yet?

Carlo A. Furia
Software Institute, USI
bugcounting.net

Practical or Impractical?

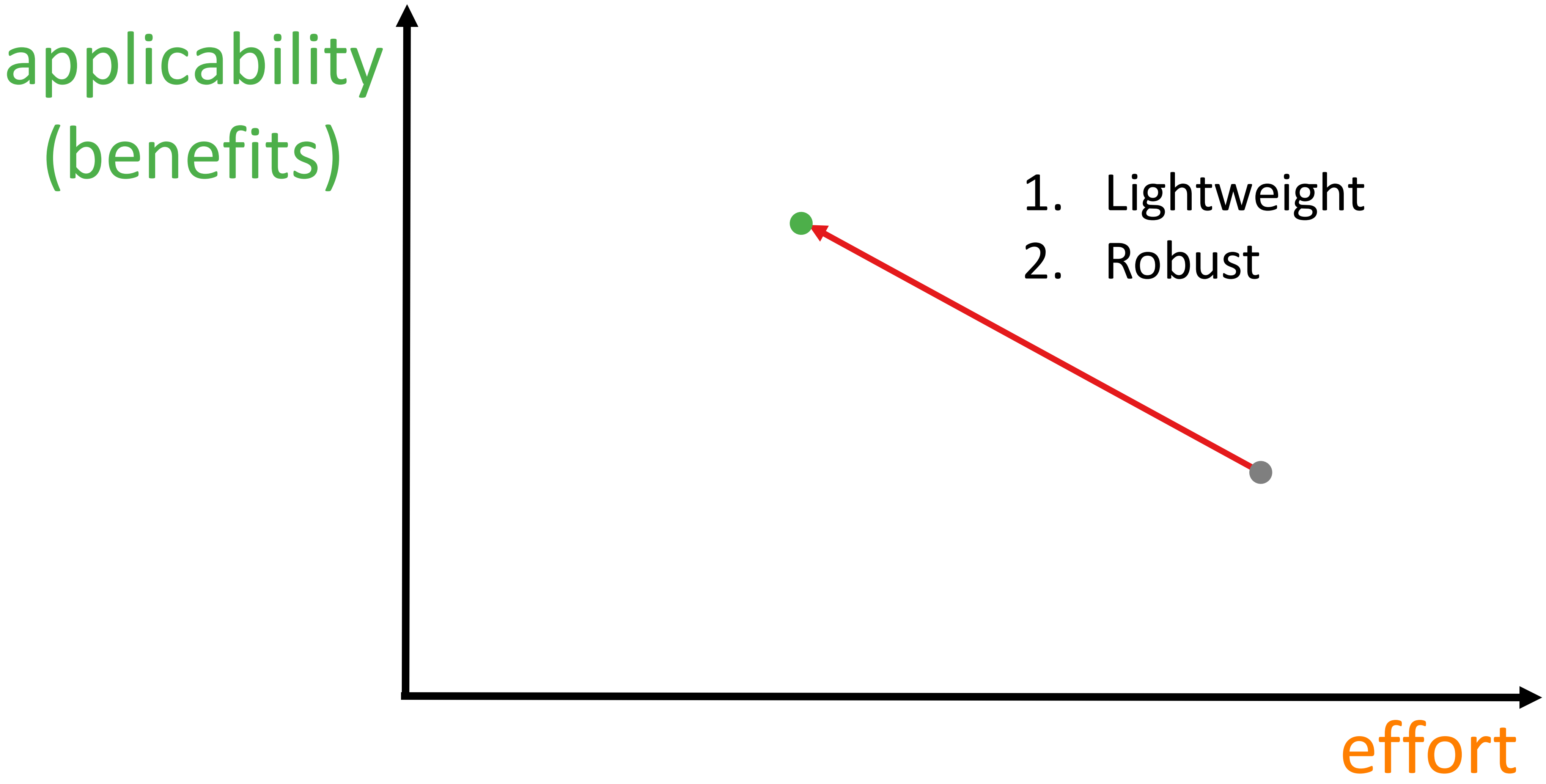


Increase automation



SMT solving,
inference,
synthesis,
intermediate verifiers, …

More Ways to Practicality



1. Lightweight
2. Robust

Exception Preconditions

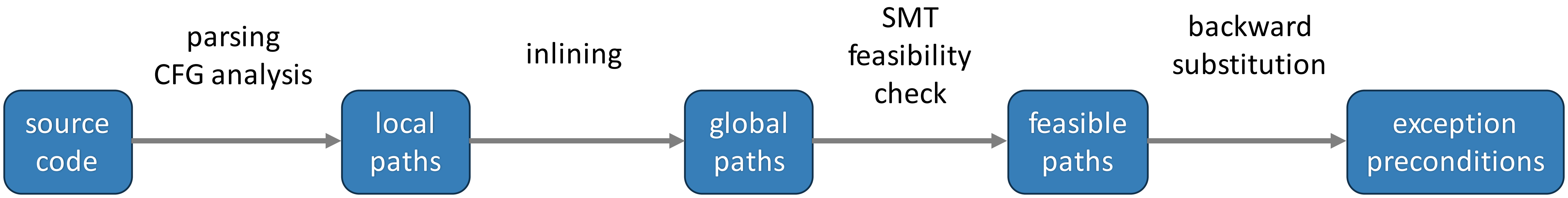
```
  throws: IndexOutOfBoundsException
    when: off >= 0 && len >= 0
          && bs.length < len + off
 example: [off = 0, len = 1, bs.length = 0]
```

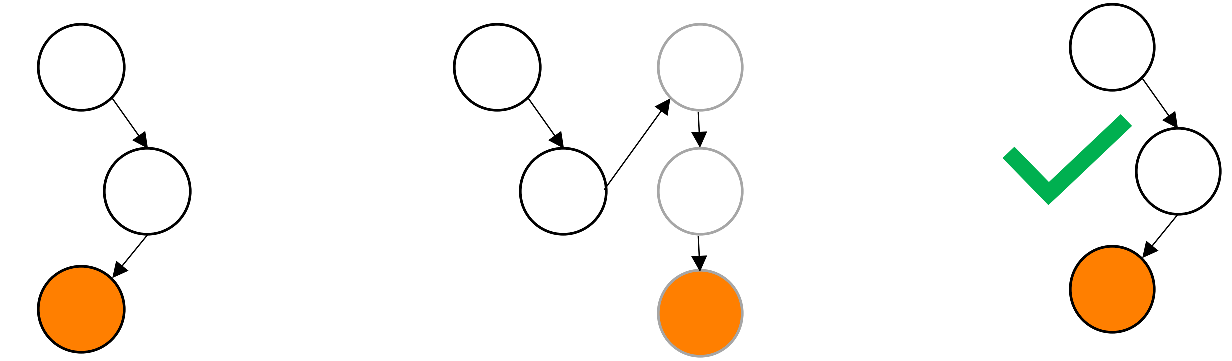
```
public static String bytes2base64(final byte[] bs,
    final int off, final int len, final char[] code) {
  if (off < 0)
    throw new IndexOutOfBoundsException();
  if (len < 0)
    throw new IndexOutOfBoundsException();
  if (off + len > bs.length)
    throw new IndexOutOfBoundsException();
  if (code.length < 64)
    throw new IllegalArgumentException();
  // ...
}
```

Lightweight Exception Precondition Inference

- Analysis does not require building project
- Flexible precision vs. recall trade-off

Wit: Exception Preconditions in Java

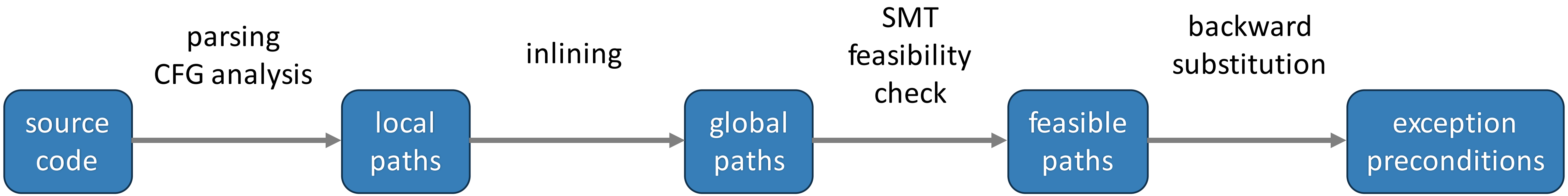




```
  throws: Error
    when: x < 0
 example: x = -1
```

Diego Marcilio

Lightweight Exception Preconditions Inference



source code

parsing
CFG analysis

local paths

inlining

global paths

SMT
feasibility
check

feasible paths

backward
substitution

exception preconditions

best-effort
parsing
of individual
classes

discard long paths,
modular analysis

unsupported
features,
SMT timeout

Precision: expres
Recall: maybes

Wit: Experimental Evaluation

projects	LOC	# expres	# maybes	Precision: expres	Precision: expres + maybes
5 JDK modules + 46 Java projects	6.1 M	30’487	31’043	100%	88%

Wit: Experimental Evaluation

projects	Recall: expres	Recall: expres + maybes
Apache Commons IO	9% (all) 57% (supported)	
9 open-source Java projects	49% (all) 71% (supported)	53% (all) 78% (supported)

Wit: Practical Usefulness

exception preconditions not already documented
72%
(out of 742 analyzed)

exception preconditions merged into official project documentation
71
(out of 90 submitted)

Robustness of Verifiers: Java 6...

```
@Require(forall j: int :: values[j] != 1)
@Ensure(return >= 0)
static int summary(int[] values) {
  int result = 0;
  for (int k = 0;
       k < values.length; k++) {
    invariant(result >= 0);
    if (values[k] == 0)
      result += 1;
    else if (values[k] == 1)
      result += -1;
    else if (values[k] > 0)
      result += values[k];
  }
  return result;
}
```

✓ The KeY Project

✓ OpenJML

Robustness of Verifiers: Java 6 to Java 17

🚫 The KeY Project

🚫 OpenJML

```
@Require(forall j: int :: values[j] != 1)
@Ensure(return >= 0)
static int summary(int... values) {
  var result = 0;
  for (var v: values)
    result += switch(v) {
      invariant(result >= 0);
      case 0: yield(1);
      case 1: yield(-1);
      default:
        if (v > 0) yield(v);
        else yield(0);
    }
  return result;
}
```

ByteBack:
Verification at the Level of JVM bytecode

Marco's talk about ByteBack is at 16:30 in A1 272 (Track A)

Marco Paganoni

✓

```
@Require(forall j: int :: values[j] != 1)
@Ensure(return >= 0)
static int summary(int… values) {
  var result = 0;
  for (var v: values)
    result += switch(v) {
      invariant(result >= 0);
      case 0: yield(1);
      case 1: yield(-1);
      default:
        if (v > 0) yield(v);
        else yield(0);
    }
  return result;
}
```