ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

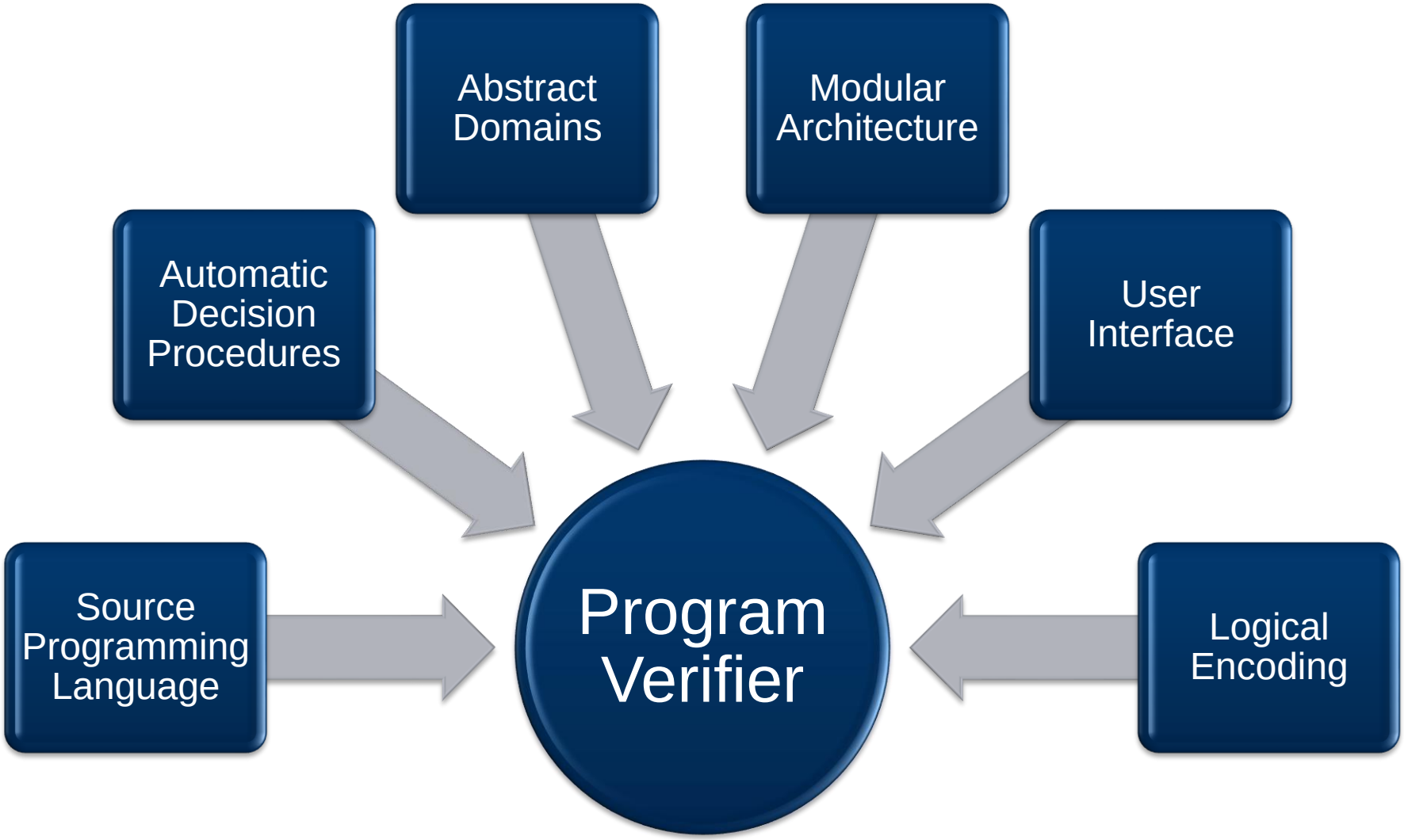
inf | Informatik Computer Science

# Boogie: A Modular Reusable Verifier for Object-Oriented Programs

M. Barnett, B.E. Chang, R. DeLine, B. Jacobs, K.R.M. Leino

Lorenzo Baesso
ETH Zurich

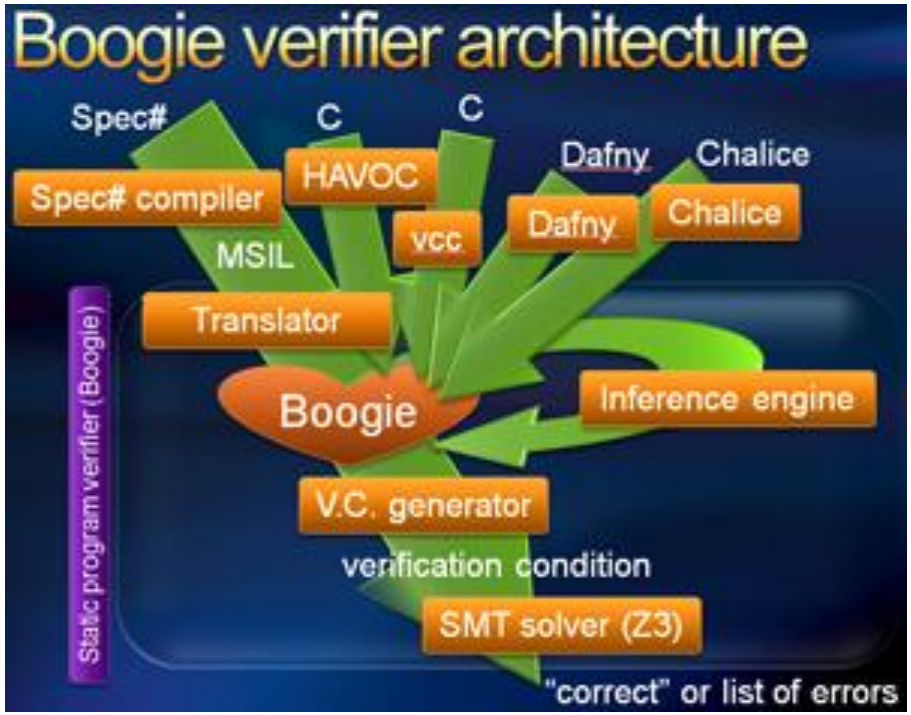
# Motivation


Abstract Domains
Modular Architecture
Automatic Decision Procedures
User Interface
Source Programming Language
Program Verifier
Logical Encoding

# Boogie: An Intermediate Verification Language

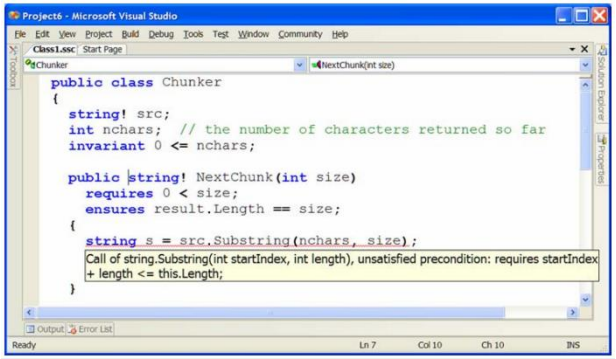
*"intended as a layer on which to build program verifiers for other languages"*

Microsoft Research

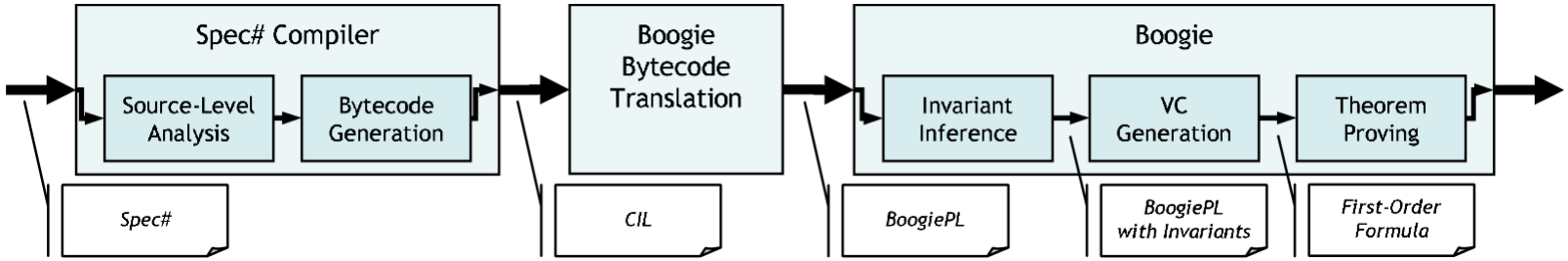


# Boogie Novel Aspects

- Nice integration into Visual Studio and **design-time feedback**.



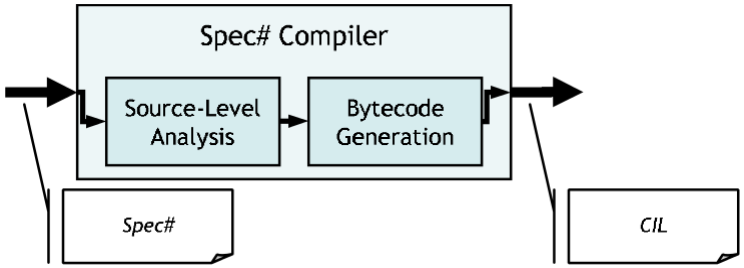

- Well defined **interfaces** and **modular architecture**.




- Distinct **proof obligation** and **verification phases**.

- **Abstract interpretation** and **verification condition generation**.

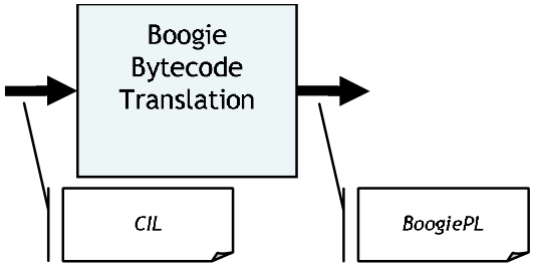
# From Spec# to the Common Intermediate Language




```
public class Example {
 int x ;
 string! s;
 invariant s.Length >= 12;
 public Example(int y) requires y > 0; {...}

 public static void M (int n) {
  Example e = new Example(100/n);
  int k = e.s.Length;
  for (int i = 0; i < n; i++) { e.x += i; }
  assert k == e.s.Length;
 }
}
```

```
.class public Example {
 ...

 .method public static void M(int32) cil managed
{
  .maxstack 1
  ldarg.0
  ...
  ldc.i4.x
  newobj instance void Example::.ctor(int)
  stloc.0
  ...
}
```

# Boogie Bytecode Translation (1/2)




CIL

- Object code with a bytecode-style format

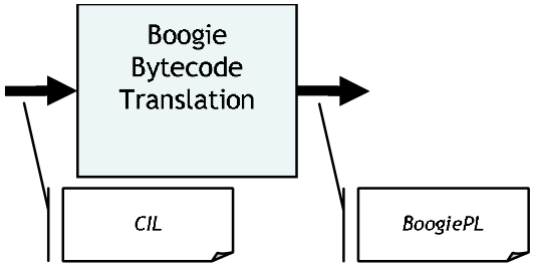
Abstract Syntax Tree

- Assert, assume, goto statements
- Loop and object invariants
- ...

BoogiePL

- Supports: procedures, pre- and postconditions, etc.
- Lacks: expressions with side effects, heap, classes and interfaces, call-by-reference parameter passing and structured control-flow.

# Boogie Bytecode Translation (2/2)




**Main Functionalities:**

- Encoding the **heap**, **object allocation** and **fields**
  - 2D array named Heap, allocation bit, fields into Heap.
- Translating **call-by-reference parameters**
  - Variables are passed as in-parameters, copied into local variables, used, copied into out-parameters, updated with out-parameters values.
- Translating **methods** and **method calls**
  - Method declaration/implementation -> BoogiePL procedure/implementation
  - Method call -> associated procedure/additional BoogiePL procedure
- Generating **frame conditions for methods and loops**
  - Postconditions on the procedures (modifies clauses)
  - Loop invariants (havoc clauses)

# BoogiePL: Theory

## Spec#

```
public class Example {
 int x ;
 string! s;
 invariant s.Length >= 12;
 public Example(int y)
     requires y > 0; {...}

 public static void M (int n) {
  Example e = new Example(100/n);
  int k = e.s.Length;
  for (int i = 0; i < n; i++) {
     e.x += i;
  }
  assert k == e.s.Length;
 }
}
```

## BoogiePL

```
const System.Object : name;
const Example : name;
axiom Example <: System.Object;
function typeof(obj : ref)
     returns (class : name);

const allocated : name;
const Example.x : name;
const Example.s : name;

function StringLength(s : ref )
     returns (len : int);
```

# BoogiePL: Imperative Part (1/2)

## Spec#

```
public class Example {
 int x ;
 string! s;
 invariant s.Length >= 12;
 public Example(int y)
     requires y > 0; {...}

 public static void M (int n) {
  Example e = new Example(100/n);
  int k = e.s.Length;
  for (int i = 0; i < n; i++) {
     e.x += i;
  }
  assert k == e.s.Length;
 }
}
```

## BoogiePL

```
var Heap : [ref , name]any;

procedure Example..ctor(this : ref, y : int);
     requires ... ∧ y > 0;
     modifies Heap;
     ensures ...;

procedure Example.M(n : int);
     requires ...;
     modifies Heap;
     ensures ...;
```

# BoogiePL: Imperative Part (2/2)

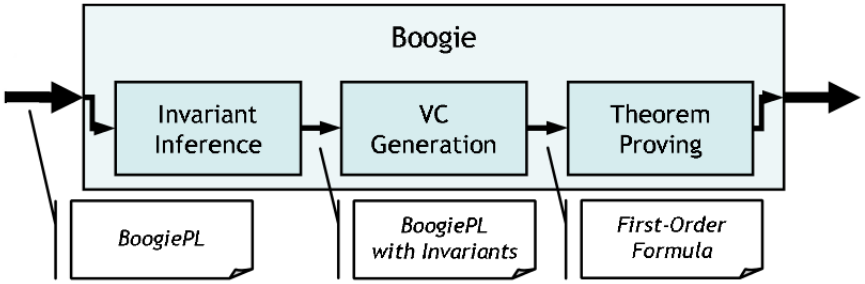
## Spec#

```
public class Example {
 int x ;
 string! s;
 invariant s.Length >= 12;
 public Example(int y)
     requires y > 0; {...}

 public static void M (int n) {
  Example e = new Example(100/n);
  int k = e.s.Length;
  for (int i = 0; i < n; i++) {
     e.x += i;
  }
  assert k == e.s.Length;
 }
}
```

## BoogiePL

```
implementation Example.M(n : int)
{
var e : ref where e = null ∨ typeof(e) <: Example;
var k : int, i : int, tmp : int, PreLoopHeap : [ref, name]any;

Start :
      assert n ≠ 0;
      tmp := 100/n;
      havoc e;
      assume e≠null∧typeof(e)=Example∧ Heap[e,allocated]=false;
      Heap[e, allocated] := true;
      call Example..ctor(e, tmp);
      assert e ≠ null;
      k := StringLength(cast(Heap[e, Example.s], ref));
      i := 0;
      PreLoopHeap := Heap;
      goto LoopHead;

LoopHead :
      goto LoopBody, AfterLoop :

LoopBody :
      assume i < n;
      assert e ≠ null;
      Heap[e, Example.x] := cast(Heap[e, Example.x], int) + i;
      i := i+ 1;
      goto LoopHead;

AfterLoop :
      assume ¬(i < n);
      assert e ≠ null;
      assert k = StringLength(cast(Heap[e, Example.s], ref));
      return;
}
```

# Boogie: Invariant Inference



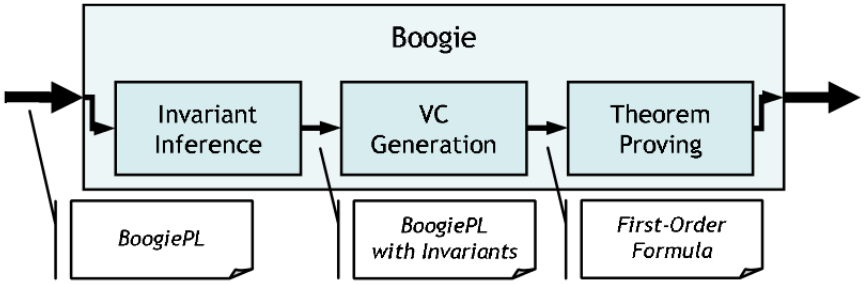

**General idea:**

- An abstract interpreter is used to infer invariants (assume statements) and plug them into the original BoogiePL code.

**Things to consider:**

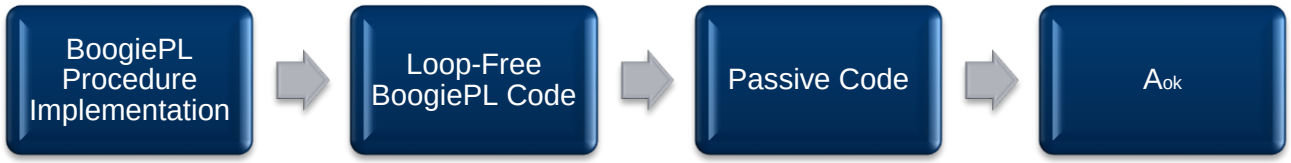
- Expressiveness (variables and function symbols).
- Exploration strategies (trade-off precision for efficiency).
- Well known abstract domains (base domains).
- Combination of abstract domains (coordination abstract domain).

# Boogie: Verification Condition Generation

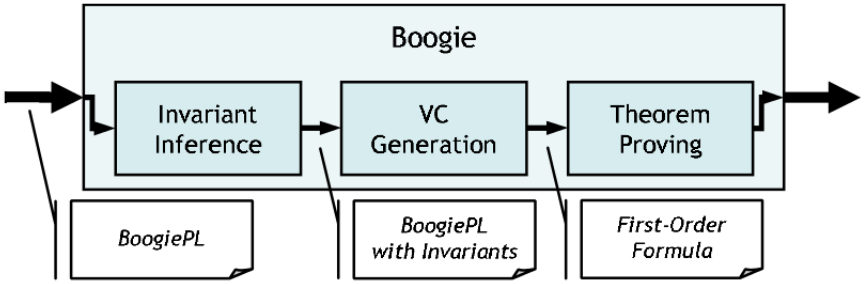



**Characteristics:**

- Requires the proof obligations to be declared in BoogiePL.
- One VC for every BoogiePL procedure implementation.
- Standard weakest-precondition calculus: $A_{ok}$ = wp(S, R).



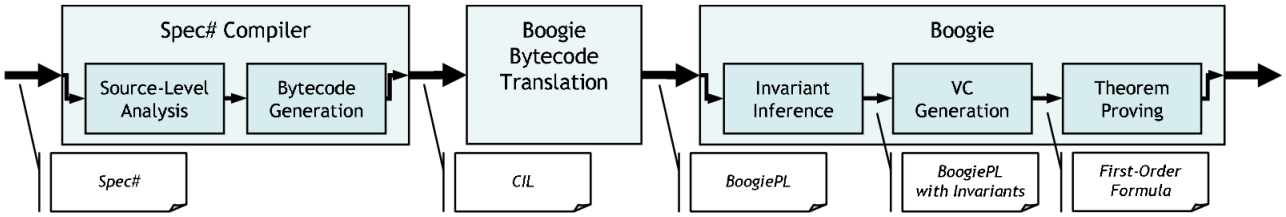
# Boogie: Theorem Proving




**Several alternatives:**

- Simplify (http://kindsoftware.com/products/opensource/Simplify/)
- Zap (under development)
- Z3 (http://z3.codeplex.com/)
- CVC3/CVC4, E-prover, SPASS, veriT, etc.

# Conclusion




**Features:**

- Design-time feedback (nicely integreted).
- Wide support of source programming languages.
- Modularity (well defined inner/outer interfaces).

**Future Work:**

- Verification results could be used to optimize the code (higher performance).
- Combination of abstract interpreter and theorem prover.
- More detailed error messages.

# References


- Boogie: A Modular Reusable Verifier for Object-Oriented Programs
  - M. Barnett, B.E. Chang, R. DeLine, B. Jacobs, K.R.M. Leino (2005)
- BoogiePL: A typed procedural language for checking object-oriented programs
  - R. DeLine, K.R.M. Leino (2005)
- Translating Java Bytecode to BoogiePL
  - Alex Suzuki (2006)
- This is Boogie 2
  - K.R.M. Leino (2008)

# Boogie Example (1/2)

boogie Microsoft Research

http://rise4fun.com/Boogie

Is this property always true?

```
1  type ref, name;
2
3  var heap : [ref, name]int;
4
5  procedure swap_int_field(obj : ref, f1 : name, f2 : name);
6    ensures heap[obj,f1] == old(heap[obj,f2]);
7    ensures heap[obj,f2] == old(heap[obj,f1]);
8    //modifies heap;
9
10 implementation swap_int_field(obj: ref, f1 : name, f2 : name) {
11   var tmp : int;
12   entry:
13     tmp := heap[obj,f1];
14     heap[obj,f1] := heap[obj,f2];
15     heap[obj,f2] := tmp;
16     return;
17 }
```

home permalink

'▶' shortcut: Alt+B

| | | Description |
|---|---|---|
| ⊗ | 1 | command assigns to a global variable that is not in the enclosing method's modifies clause: heap |
| ⊗ | 2 | command assigns to a global variable that is not in the enclosing method's modifies clause: heap |

```
input(14,17): Error: command assigns to a global variable that is not in the enclosing method's modifies clause: heap
input(15,17): Error: command assigns to a global variable that is not in the enclosing method's modifies clause: heap
2 type checking errors detected in input
```

# Boogie Example (2/2)

boogie Microsoft Research

http://rise4fun.com/Boogie

Is this property always true?

```
1  type ref, name;
2
3  var heap : [ref, name]int;
4
5  procedure swap_int_field(obj : ref, f1 : name, f2 : name);
6    ensures heap[obj,f1] == old(heap[obj,f2]);
7    ensures heap[obj,f2] == old(heap[obj,f1]);
8    modifies heap;
9
10 implementation swap_int_field(obj: ref, f1 : name, f2 : name) {
11   var tmp : int;
12   entry:
13     tmp := heap[obj,f1];
14     heap[obj,f1] := heap[obj,f2];
15     heap[obj,f2] := tmp;
16     return;
17 }
```

home permalink

'▶' shortcut: Alt+B

```
Boogie program verifier finished with 1 verified, 0 errors
```

# Other Verifiers

- Javanni - a Verifier for JavaScript
  - http://cloudstudio.ethz.ch/comcom/#Javanni


- Boogaloo - the Boogie Interpreter
  - http://cloudstudio.ethz.ch/comcom/#Boogaloo

# Abstract Syntax Tree Example

Euclidean Algorithm (GCD):

```
while b ≠ 0
      if a > b
            a := a − b
      else
            b := b − a
return a
```