Interactive Debugging for Extensible Languages in Multi-Stage Transformation Environments

2nd International Workshop on Executable Modeling

Domenik Pavletic and Kim Haßlbauer

2016

Agenda

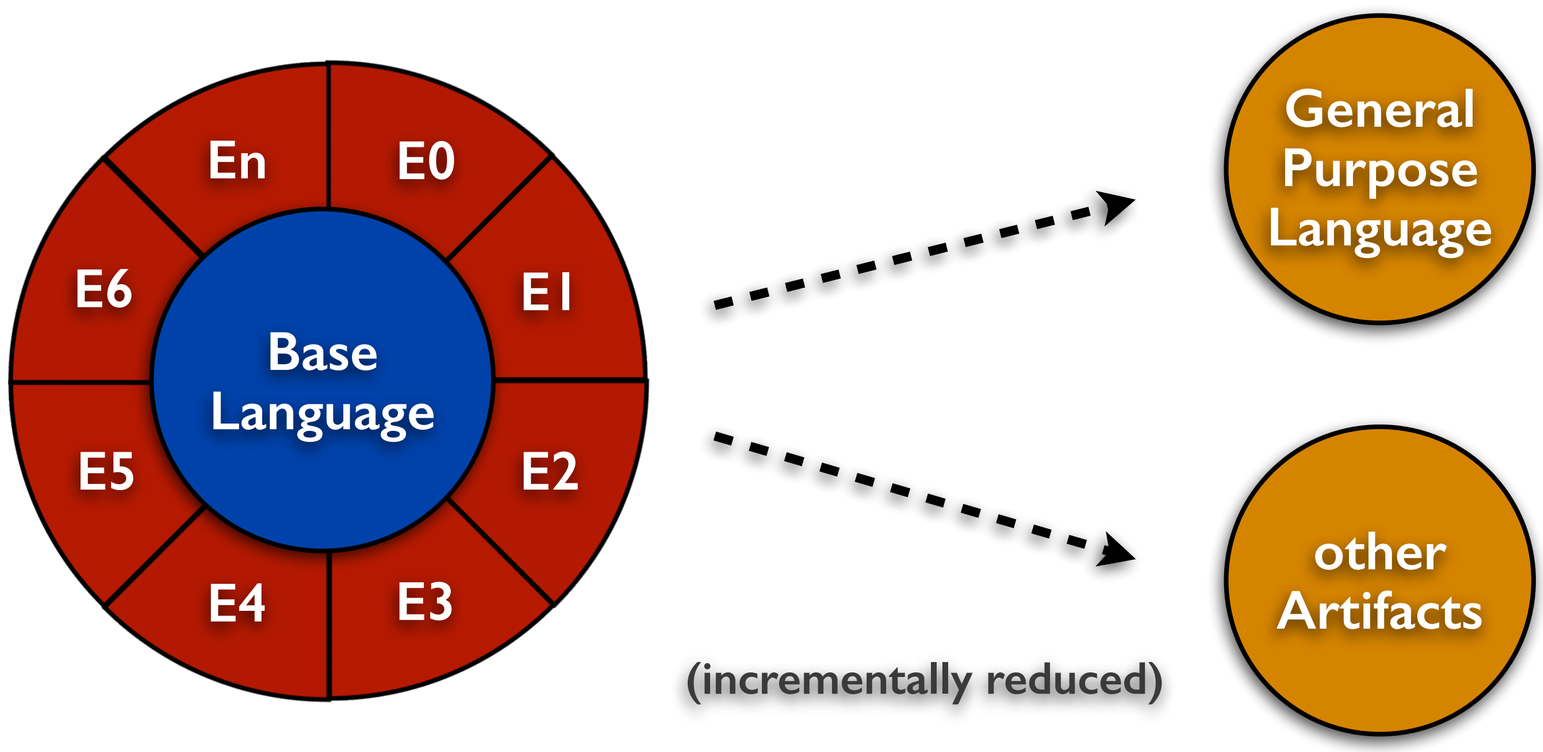
* Extensible Languages
* Bugs in Extensible Language Programs
* The MuLDer Framework
* mbeddr multi-level Debugger
* Conclusion

Extensible Languages

the context

Extensible Languages

enable syntactical and semantic extensions



Extensible Languages

mixed-language programs (mbeddr example)

State Machines

C

Requirements

Variability

Decision Tables

```
File Edit Search View Go To Code Build Run Tools Version Control Wi
ADemoModule ×
enum MODE { FAIL: AUTO: MANUAL: }
statemachine Counter {
  in start() <no binding>
     step(int[0..10] size) <no binding> trace R2
  out resetted() <no binding> {resettable}
  vars int[0..10] currentVal = 0
       int[0..10] LIMIT = 10
  states (initial = start)
    state start {
      on start [ ] -> countState { }
    }
    state
      on step [currentVal + size > LIMIT] -> start { send resetted(); }
      on step [currentVal + size <= LIMIT] -> countState {
                                              + size;
        send incremented();
} } }
MODE nextMode(MODE mode, int8_t speed) {
  return MODE, FAIL
```

```
start  ^inEvents (cdesignpaper.screenshot.ADemoModule)
step   ^inEvents (cdesignpaper.screenshot.ADemoModule)
```

Error: wrong number of arguments

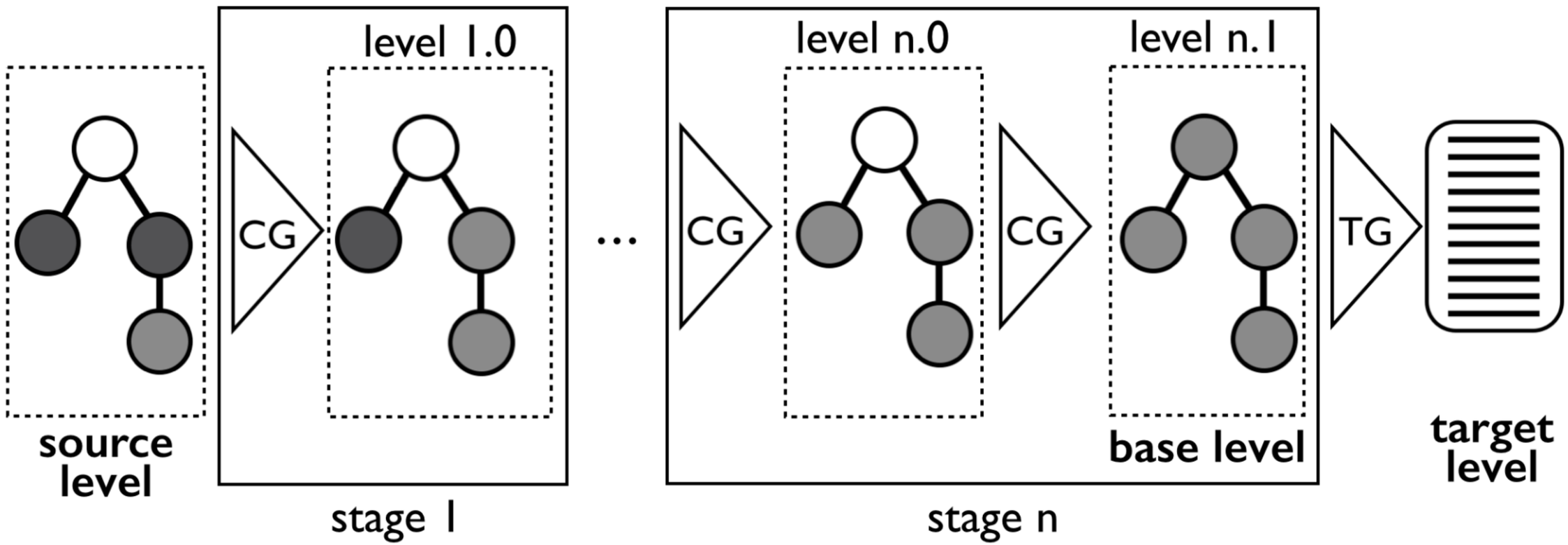
	mode == AUTO	mode == MANUAL
speed < 50	AUTO	MANUAL
speed >= 50	MANUAL	MANUAL

trace R1;

```
cdesignpaper
components
statemachine
test.ex.ext.statemachine
test.ex.ext.yices
test.ts.cc.fm
test.ts.ext.statemachine
test.ts.requirements
com.mbeddr.cc.reqtrace
com.mbeddr.components
com.mbeddr.ext.statemachi
com.mbeddr.statemachines
Modules Pool
```

Extensible Languages

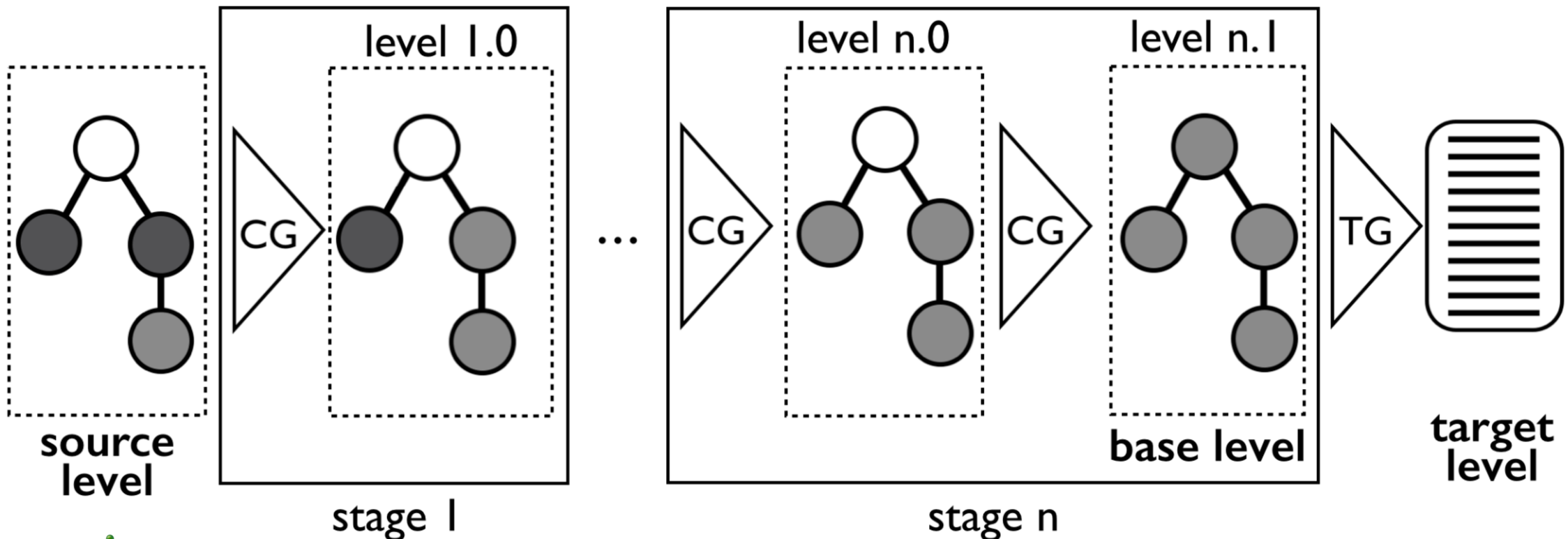
high-level code gets incrementally reduced

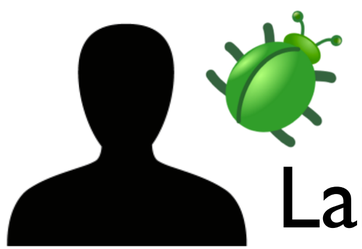


Who can introduce Bugs that manifest at **Runtime**?

Who can introduce Bugs?

language user - source level

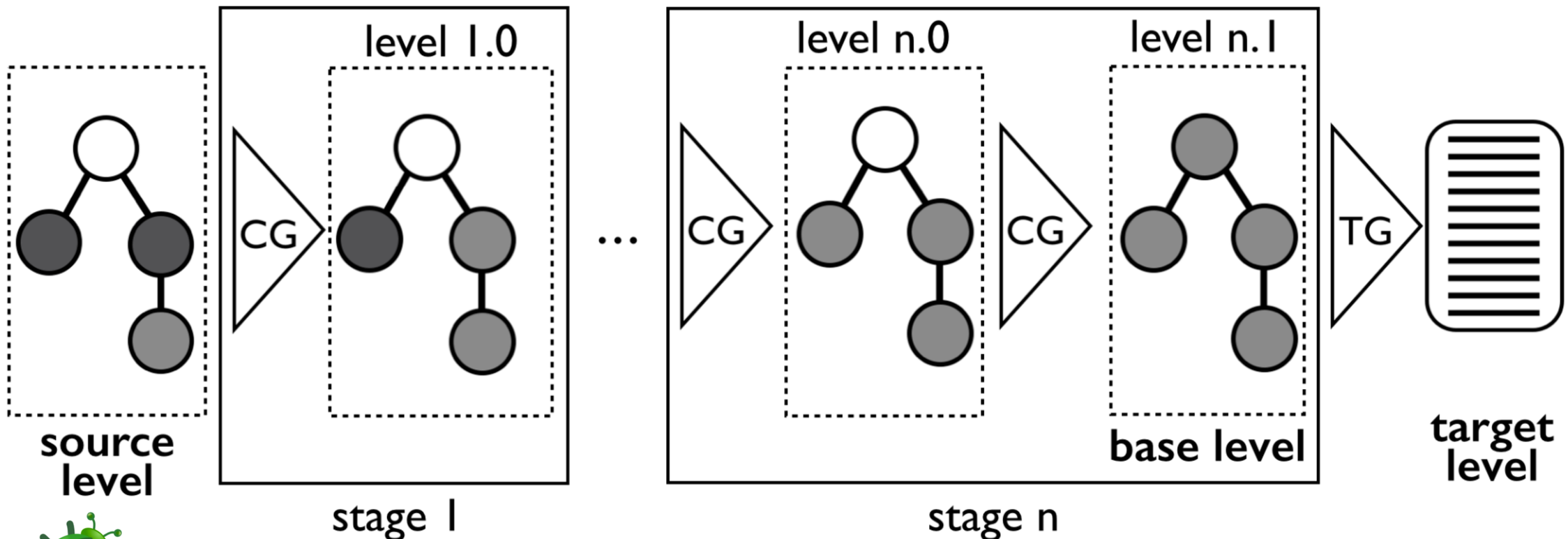


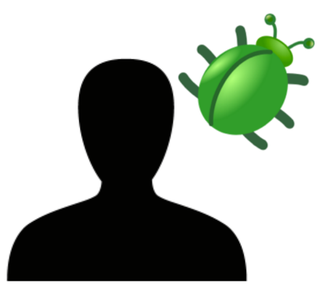


Language user **manually** introduces **source level bug**

Who can introduce Bugs?

language engineer - intermediate/base level





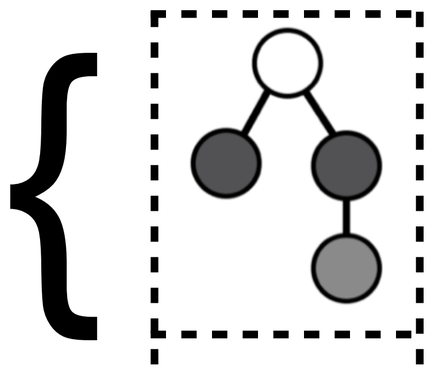
Language engineer introduces bug through **faulty transformation**

Multi-Level Debuggers can analyze both **bug** categories

Multi-Level Debuggers

two types of users

Language User



Source Level

Language Engineer

n Intermediate Levels

Base Level

Target Level (text)

(later demonstrated)

The MuLDer Framework

(The Multi-Level Debugger Framework)

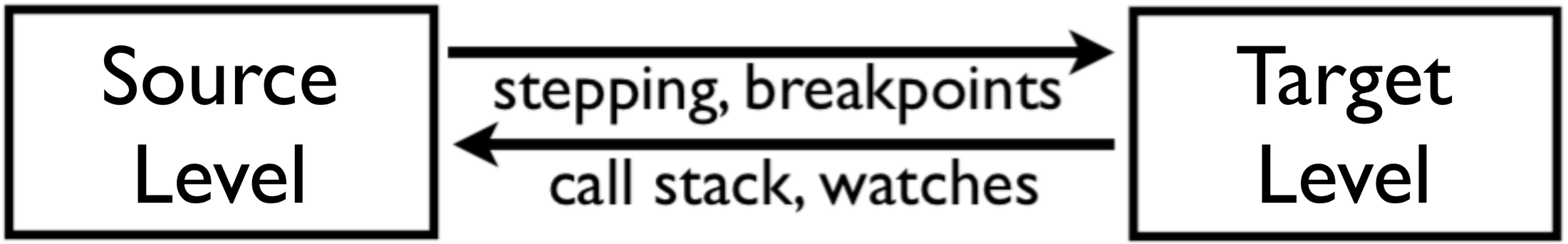
MuLDer Framework

overview

- Based on **mbeddr** Platform + JetBrains **MPS** (customized version)
- For languages supporting „call stack / stepping“-based debug approach (e.g., imperative)
- Debugging support is built per language construct
- Implementation is restricted to MPS, approach is workbench independent

MuLDer Framework

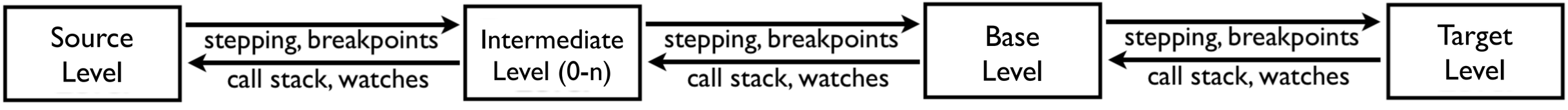
initial source-level debugger framework



- No support for **alternative transformations**
- Hard to establish a **mapping** between both levels
- **Changing** in low-level generators require **updates** in debugger definitions

MuLDer Framework

incremental approach with MuLDer



- ✓ Multi-Level debugging support
- ✓ Support for **alternative transformations**
- ✓ Easy to establish a **mapping** between both levels
- ✓ Debugger definitions are independent from **changing** low-level generators

MuLDer Framework

approach

1. Specify **debug semantics** of language constructs
2. Annotate transformations (M2M/M2T) to map between **high-level** and **generated code**

MuLDer Framework

specify debug semantics: implement interfaces

Function ×

Warning: the node is in a read-only model. Your changes won't be saved

```
concept Function extends     FunctionSignature
                implements  IDocumentable
                            IStatementListContainer
                            IFunctionLike
                            ILOCCountProvider
                            ICanMangleNames
                            IMayActAsMainFunction
                            IFunctionLikeReducedToSingleFunction
                            IDataFlowAnalyzerEntryPoint
                            Callable
                            ProviderScope
                            ControlFlowProvider
                            Steppable

instance can be root: false
alias: function
short description: a C function
```

MuLDer Framework

specify debug semantics: provide specification

Function ×

Warning: the node is in a read-only model. Your changes won't be saved

```
concept Function extends     FunctionSignature
                implements   IDocumentable
                             IStatementListContainer
```

Function_CallableSpec ×

Warning: the node is in a read-only model. Your changes won't be saved

Callable

concept:	Function

stack frame name:	(node, watchable)->string {
	  node.*name*;
	}

```
instance can be root: false
alias: function
short description: a C function
```

MuLDer Framework

annotate transformation rules (M2M)

```
exported int32 main( ) {
  return test[tc];
} main (function)
testcase tc {
  boolean trueValue = true;
  assert(true && trueValue);
}
```

```
@StackFrame2StackFrame main → main
exported int32 main( ) {
  return blockexpr {
    int32 failures = 0;
    @OutlineStackFrame: blockStatement2Testcase
    {
      boolean trueValue = true;
      if (!(true && trueValue)) {
        failures++;
      } if
    }
    yield failures;
  };
} main (function)
```

```
concept     TestcaseRef  --> content node:
inheritors  false            <TF @OutlineStackFrame: blockStatement2Testcase
condition   <always>         {
                               $COPY_SRCL$[ int32 bla = 123; ]
                             }
                             TF>
```

MuLDer Framework

annotate transformation rules (M2T)

```
@StackFrameProvider
text gen component for concept Function {
    (context, buffer, node)->void {
        ...
        append { };
               @IdentifierProvider
        append ${node.mangledName() };
        append {(};
        ...
    }
}
```

➡ Associates target level Functions with base level

mbeddr

multi-level Debugger

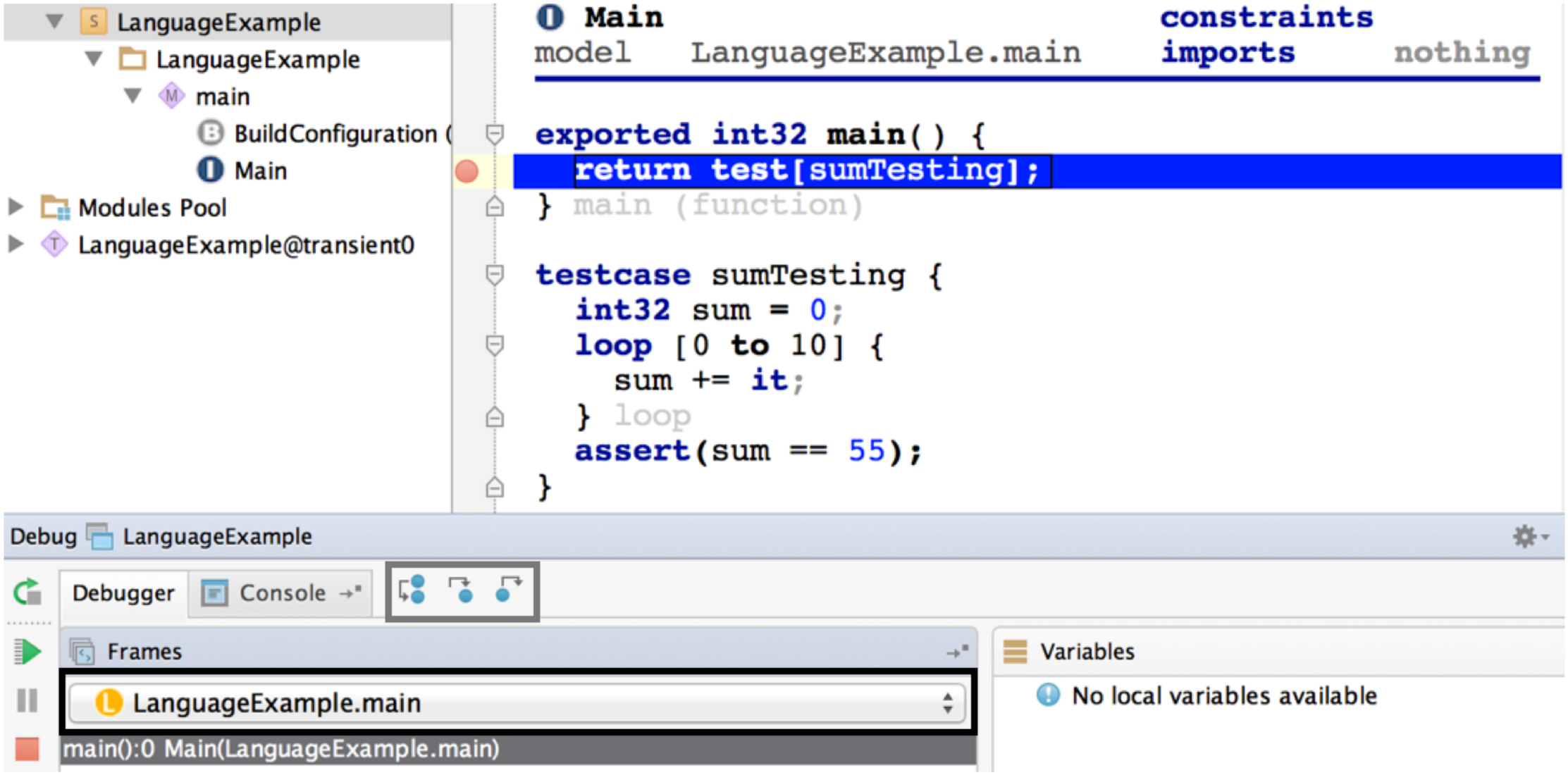
mbeddr multi-level debugger

different languages supported

- mbeddr C
- State machines
- Components
- Decision Tables
- …

mbeddr multi-level debugger

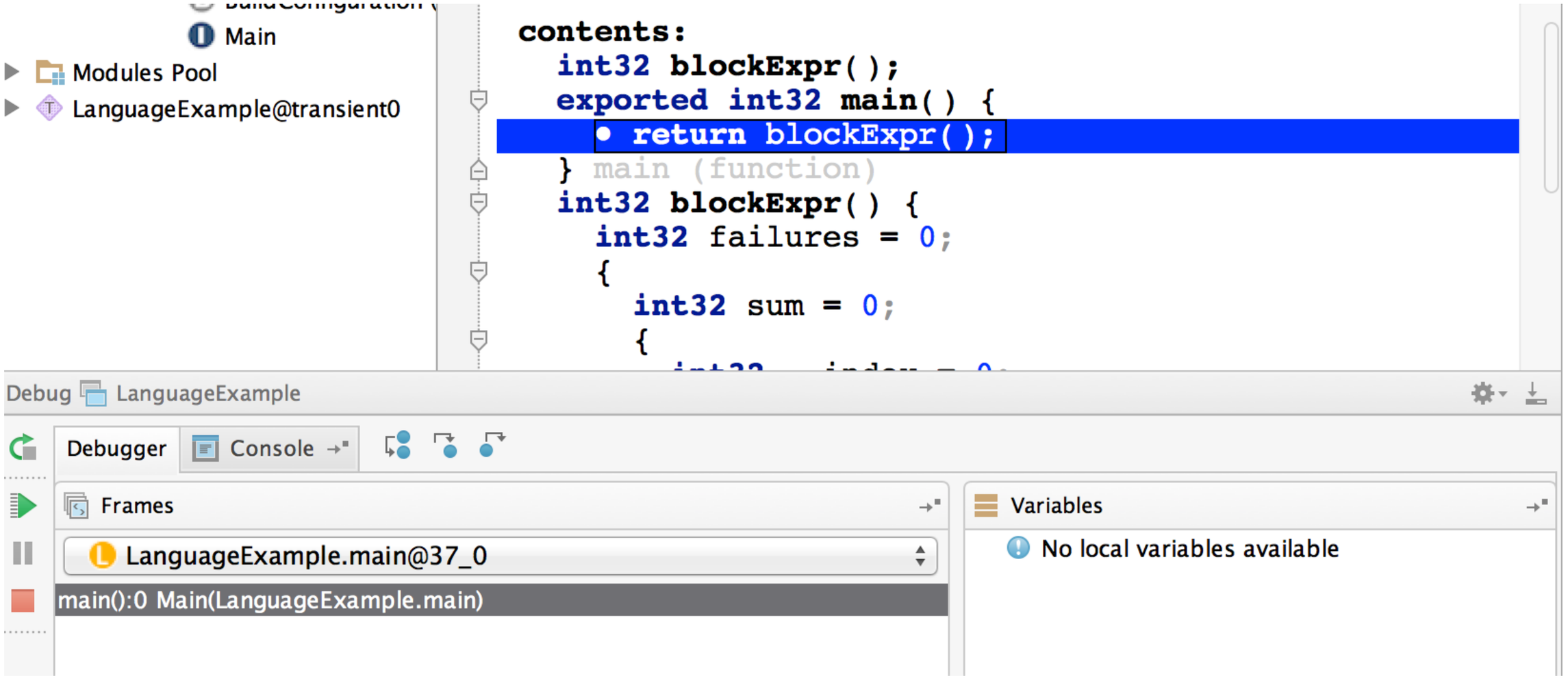
source-level debugging



➡ Debugging **source-level** code (high level)

mbeddr multi-level debugger

base-level debugging



➡ Debugging **base-level** code (C)

Conclusion

Conclusion

identified advantages

✓ **Alternative** transformations supported

✓ Building debuggers requires **little effort** due to the incremental approach

✓ Debugger implementation does not **depend** on low-level languages/generators

✓ Debuggers can be **used** by language users and language engineers

Conclusion

identified limitations

- Implementation **restricted** to MPS
- **Tracing** across all levels required (MPS modified)
- Debug **performance** decreases with each additional abstraction level
- Language must support the „stepping / call stack“ based debugging approach

return „any Questions?“;

https://github.com/DomenikP/MuLDer