universität wien

CASM-IR: Uniform ASM-Based Intermediate Representation for Model Specification, Execution, and Transformation

Philipp Paulweber and Emmanuel Pescosta and Uwe Zdun

{philipp.paulweber,uwe.zdun}@univie.ac.at, epescosta@casm-lang.org

Research Group Software Architecture
Faculty of Computer Science
University of Vienna
Vienna, Austria

ABZ'18
June 6, 2018
Southampton, UK

universität wien

Intermediate Representation

Model Specification

WHY?

Execution

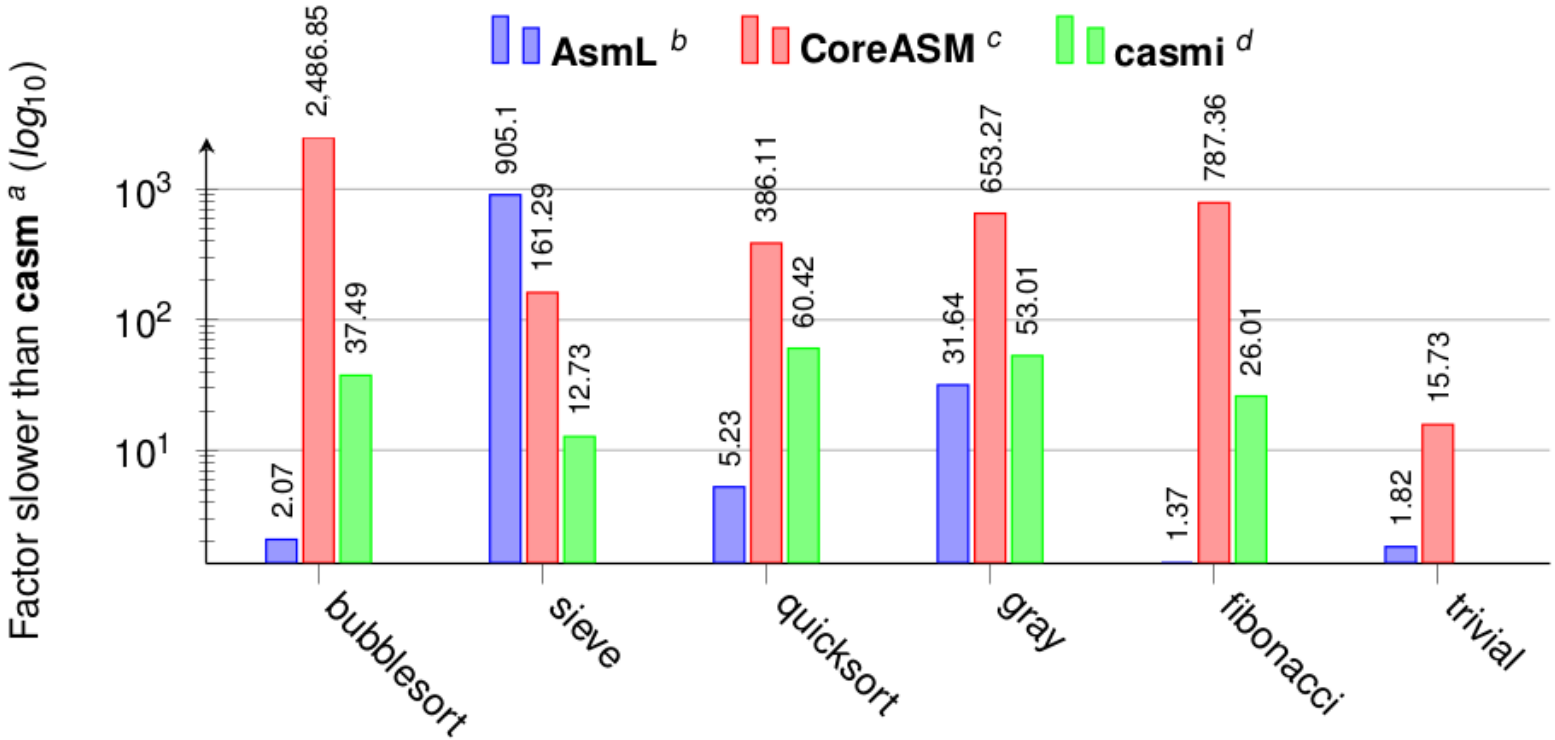
Transformation

Motivation

- Research Context & Goals
  - reusable ASM specifications, retargetable artifacts
  - optimized artifacts (code size, execution speed)
  - framework to experiment with different ASM run-time implementations
  - provide the ability to execute ASM specifications in environments with limited resources (e.g. micro-controller)
- Earlier Research
  - proposed retargetable transformation approach [1]
  - discovered huge ASM-related optimization potential [2]
    - run-time implementation and specification transformation related

[1] P. Paulweber and U. Zdun, "A Model-Based Transformation Approach to Reuse and Retarget CASM Specifications," in *Abstract State Machines, Alloy, B, TLA, VDM, and Z - 5th International Conference, ABZ 2016*, Lecture Notes in Computer Science 9675, pp. 250–255, Springer, 2016
[2] R. Lezuo, P. Paulweber, and A. Krall, "CASM - Optimized Compilation of Abstract State Machines," in *SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems (LCTES)*, pp. 13–22, ACM, 2014

Motivation Cont'd



a) CASM Compiler without Optimizations [2]
b) Microsoft .NET-based Compiler [3]
c) Java-based Interpreter [4]
d) CASM AST-based Interpreter [2]

[2] R. Lezuo, P. Paulweber, and A. Krall, "CASM - Optimized Compilation of Abstract State Machines," in *SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems (LCTES)*, pp. 13–22, ACM, 2014

Swap Example

```
1  CASM init swap
2
3  function x : -> Integer
4  function y : -> Integer
5
6  rule swap =
7  {
8      x := y
9      y := x
10     program( self ) := undef
11 }
```

Output in e.g. C?

Swap Example Cont'd

universität wien

```
1  CASM init swap
2
3  function x : -> Integer
4  function y : -> Integer
5
6  rule swap =
7  {
8      x := y
9      y := x
10     program( self ) := undef
11 }
```

```
1  // generated run-time structures
2  struct Integer x;
3  struct Integer y;
4  struct Agent program;
5
6  void swap( void )
7  {
8      struct Integer tmp;
9      tmp = x;
10     x = y;
11     y = tmp;
12     program = { 0 };
13 }
14
15 int main( int argc, char** argv )
16 {
17     program = { &swap, 1 };
18     while( program.defined )
19     {
20         (*program.value)();
21     }
22     return 0;
23 }
```

Swap Example Cont'd

```
1  CASM init swap
2
3  function x : -> Integer
4  function y : -> Integer
5
6  rule swap =
7  {
8      x := y
9      y := x
10     program( self ) := undef
11 }
```

```
1 int main( int argc, char** argv )
2 {
3     return 0;
4 }
```

Swap Example Cont'd

universität wien

```
CASM init swap

function x : -> Integer
function y : -> Integer

rule swap =
{
    x := y
    y := x
    program( self ) := undef
}
```

```
// generated run-time elements
// generated ASM function state

void swap( void )
{
    // generated calls of
    // run-time elements
}

void kernel( void )
{
    // single agent kernel calls
    // besides setup the 'swap'
}

int main( int argc, char** argv )
{
    // generated run-time setup
    // code and function state
    // initialization
    kernel();
    // generated runt-time tear-
    // down code
}
```

Swap Example Cont'd

```
CASM init swap          // CASM

function x : -> Integer
function y : -> Integer

rule swap =
{
  x := y
  y := x
  program( self ) := undef
}
```

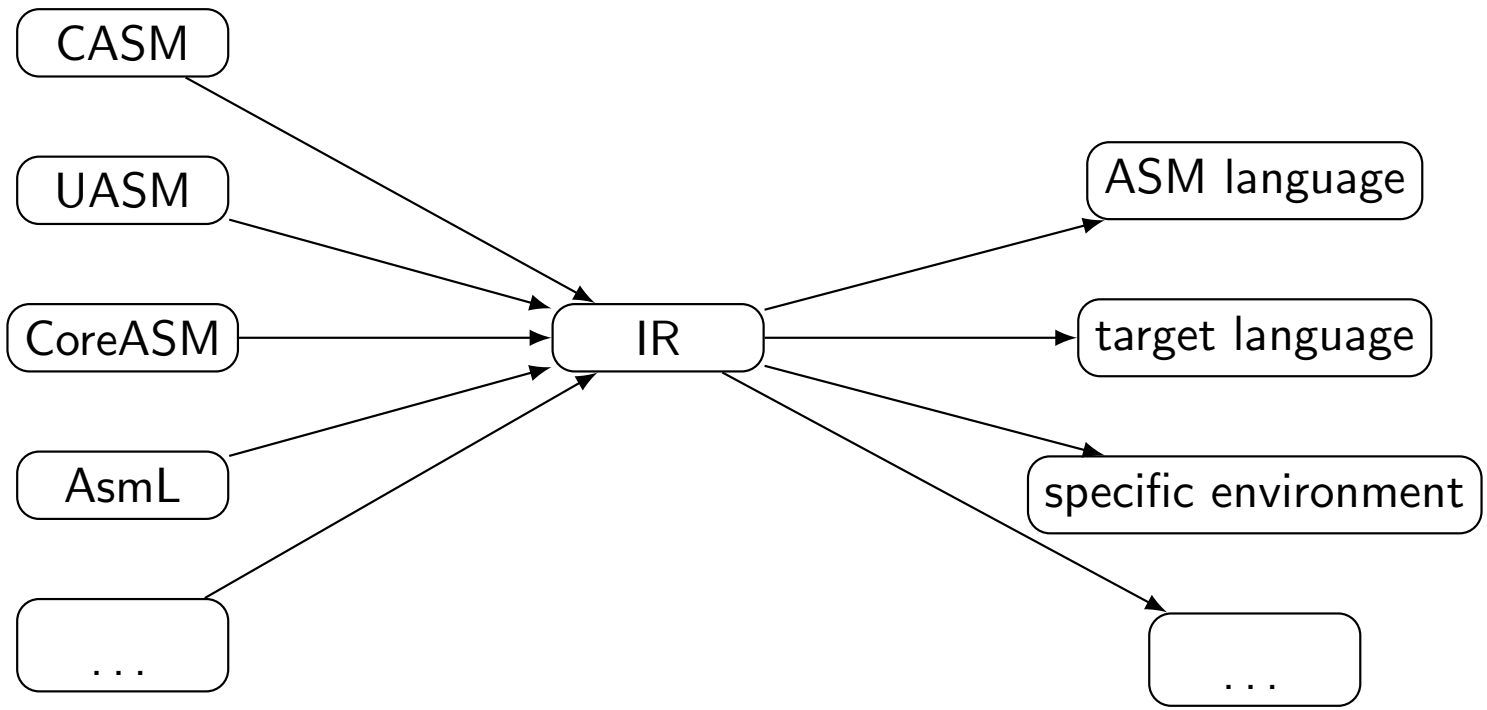
```
var x as Integer          // AsmL
var y as Integer

swap()
  x := y
  y := x

Main()
    swap()
  step
    // terminates after this step
```

```
CoreASM swap        // CoreASM
use StandardPlugins
init swap

function x : -> Integer
function y : -> Integer

rule swap =
  par
    x := y
    y := x
    program( self ) := undef
  endpar
```

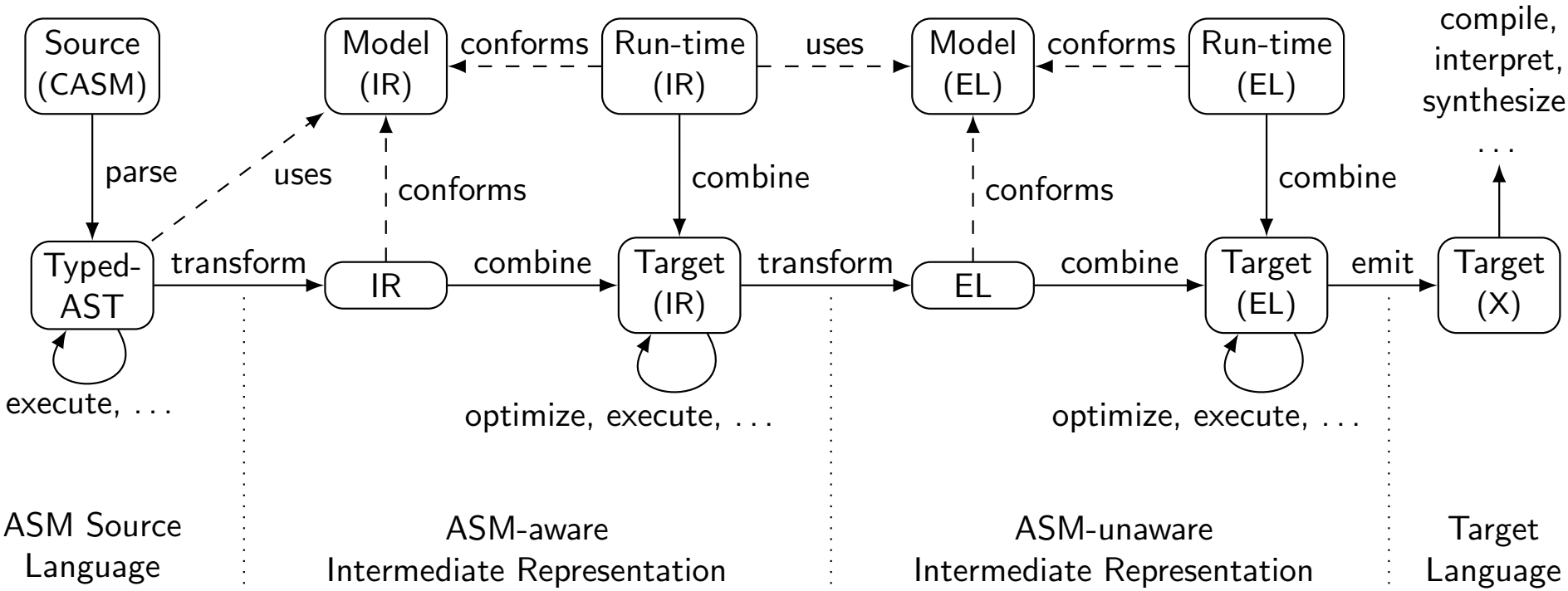
```
asm swap                // AsmetaL
import ../STDL/StandardLibrary

signature:
  dynamic controlled x : Integer
  dynamic controlled y : Integer

definitions:
  main rule swap =
    par
      x := y
      y := x
    endpar
```

Introduce IR

- uniform ASM-based IR
- generalize ASM-related optimizations – analyze & transform passes
- provide foundation for ASM language engineers



Introduce IR Cont'd



[1] P. Paulweber and U. Zdun, "A Model-Based Transformation Approach to Reuse and Retarget CASM Specifications," in *Abstract State Machines, Alloy, B, TLA, VDM, and Z - 5th International Conference, ABZ 2016*, Lecture Notes in Computer Science 9675, pp. 250–255, Springer, 2016

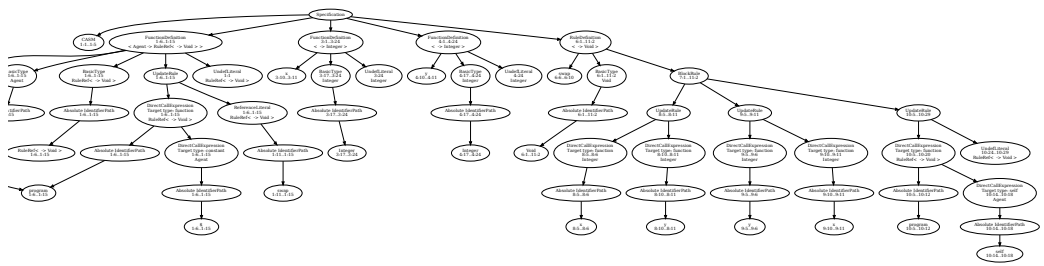
Swap Example Cont'd

```
1  CASM init swap
2
3  function x : -> Integer
4  function y : -> Integer
5
6  rule swap =
7  {
8      x := y
9      y := x
10     program( self ) := undef
11 }
```

- not explicit defined `program` function
- not explicit defined updates to functions
- run-time operations not visible

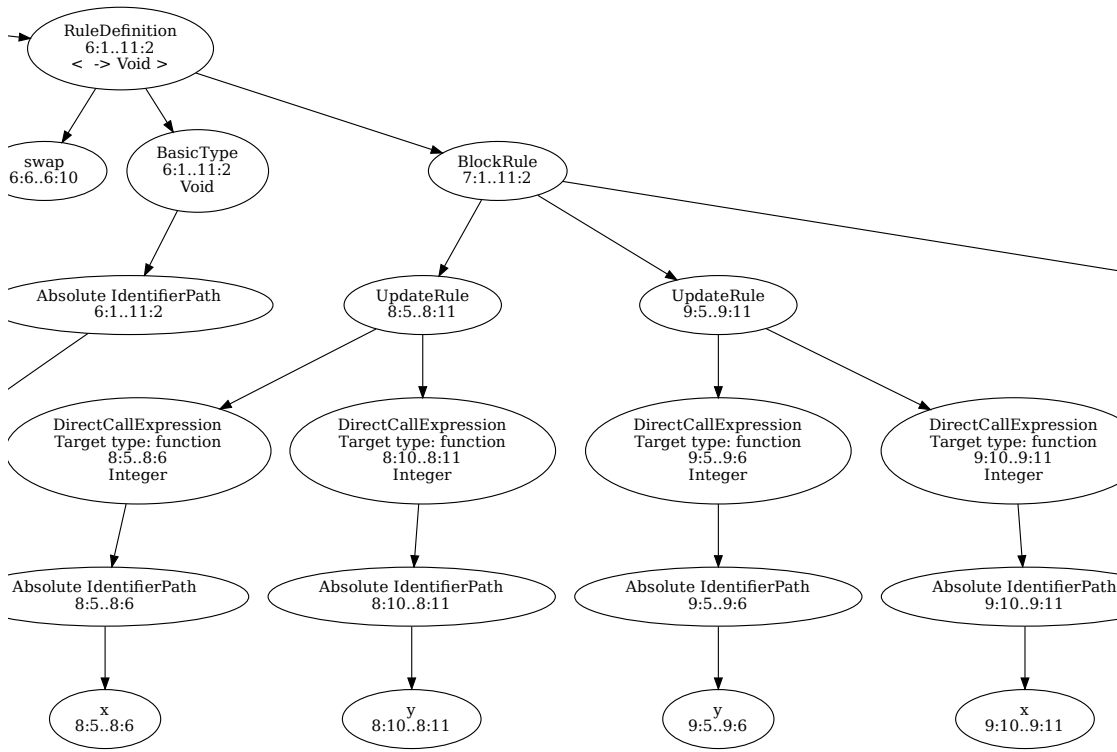
Swap Example Cont'd

```
1  CASM init swap
2
3  function x : -> Integer
4  function y : -> Integer
5
6  rule swap =
7  {
8      x := y
9      y := x
10     program( self ) := undef
11 }
```



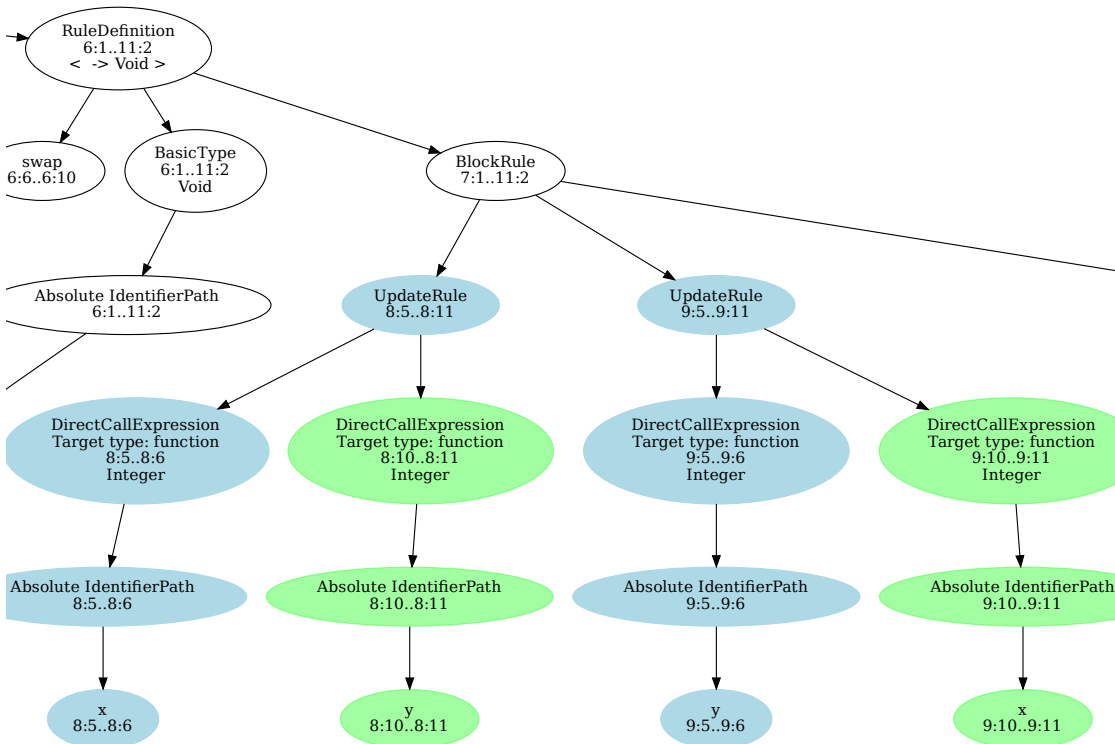
Swap Example Cont'd

```
1  CASM init swap
2
3  function x : -> Integer
4  function y : -> Integer
5
6  rule swap =
7  {
8      x := y
9      y := x
10     program( self ) := undef
11 }
```



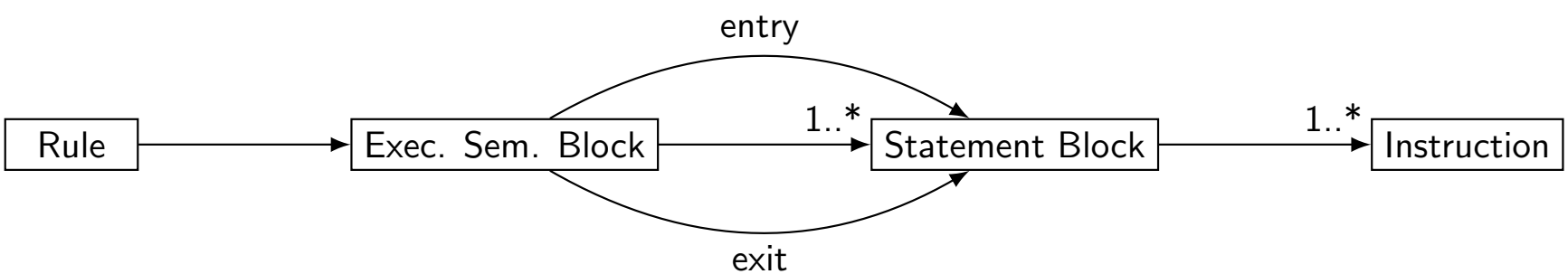
Swap Example Cont'd

```
1  CASM init swap
2
3  function x : -> Integer
4  function y : -> Integer
5
6  rule swap =
7  {
8      x := y
9      y := x
10     program( self ) := undef
11 }
```



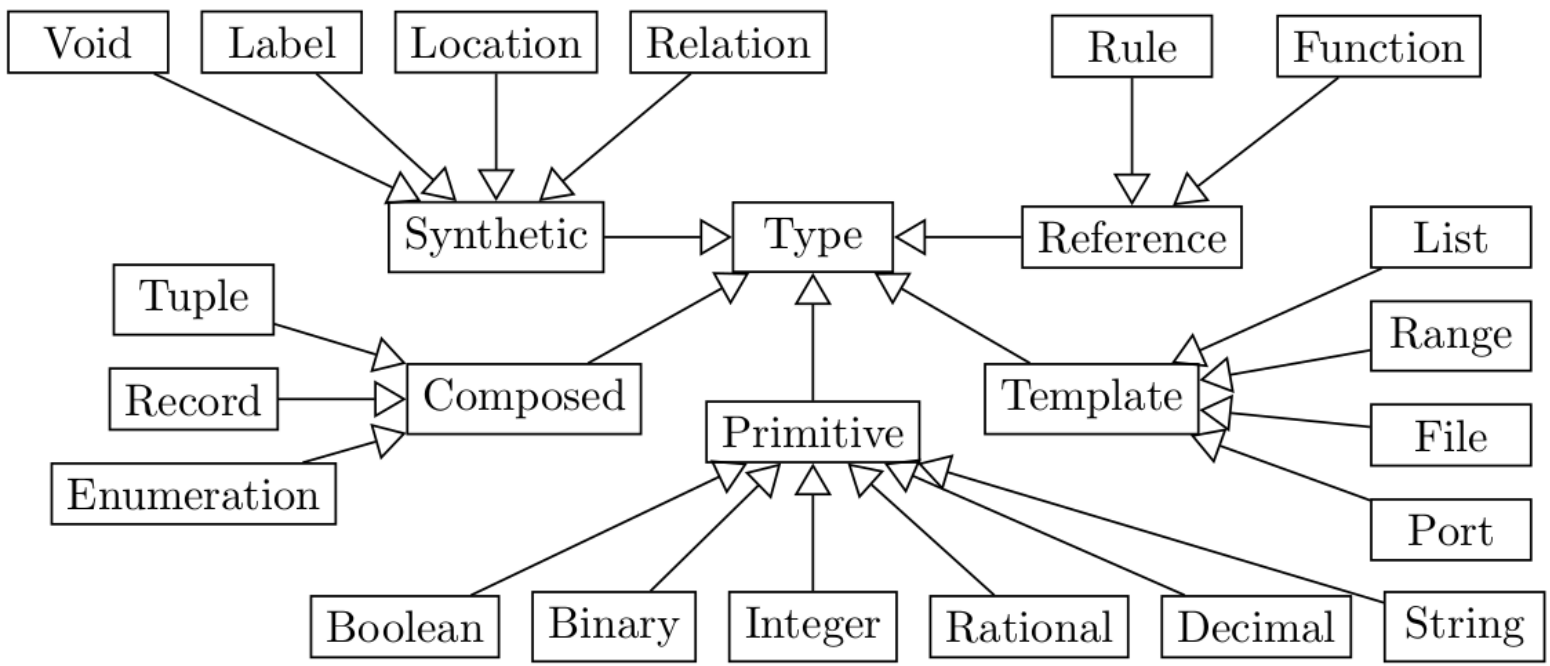
CASM-IR

- Instruction
  - expressions, terms, and ASM rules
  - three-address style
  - strongly typed
  - single static assignment (SSA) form
- Statement Block (SB)
  - list of instructions
  - sequential execution semantics
- Execution Semantics Block (ESB)
  - list of SBs and/or other ESBs
  - entry and exit section



CASM-IR Cont'd

- Typed ASM-based Model
- Defines Operator and Built-ins
- Rule, Function, and Constants
- Type System



Swap Example in CASM-IR

universität wien

```
CASM init swap

function x : -> Integer
function y : -> Integer

rule swap =
{
    x := y
    y := x
    program( self ) := undef
}
```

```
CASM-IR
a = { $ }
.agent = a
@swap < -> v>
@c0 = r< -> v> @swap
@c1 = a $
@c2 = r< -> v> undef
@c3 = a $
@program = <a -> r< -> v>>
@x = < -> i>
@y = < -> i>
@init -> v = {
  lbl0: entry
    fork par

  lbl1: %lbl0
    %r0 = location <a -> r< -> v>>
              @program, a @c1
    update loc %r0, r< -> v> @c0

  exit: %lbl0
    merge par
}
```

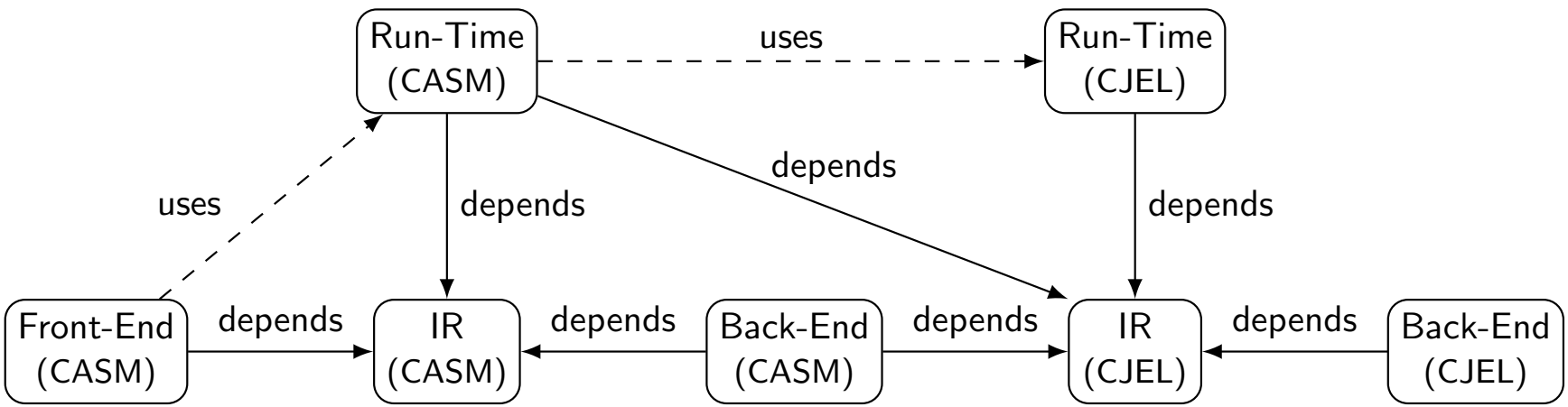
Swap Example in CASM-IR Cont'd

```
1  CASM init swap
2
3  function x : -> Integer
4  function y : -> Integer
5
6  rule swap =
7  {
8      x := y
9      y := x
10     program( self ) := undef
11 }
```

```
12 @swap -> v = {
13   lbl2: entry
14     fork par
15
16   lbl3: %lbl2
17     %r1 = location < -> i> @y
18     %r2 = lookup loc %r1
19     %r3 = location < -> i> @x
20     update loc %r3, i %r2
21
22   lbl4: %lbl2
23     %r4 = location < -> i> @x
24     %r5 = lookup loc %r4
25     %r6 = location < -> i> @y
26     update loc %r6, i %r5
27
28   lbl5: %lbl2
29     %r7 = location <a -> r< -> v>>
30               @program, a @c3
31     update loc %r7, r< -> v> @c2
32
33   exit: %lbl2
34     merge par
35 }
```

Applications

- Interpreter
- Compiler
- Language Server Protocol (LSP) Service (DEMO)



Conclusion

- uniform (C)ASM-based IR for execution and transformation
- CASM project implemented in C++11/14
  - sources/binaries (in part) are already available
  - used license is GPLv3 + linking exception (LE)
  - GitHub: github.com/casm-lang
  - website: casm-lang.org

References

[1] P. Paulweber and U. Zdun, "A Model-Based Transformation Approach to Reuse and Retarget CASM Specifications," in *Abstract State Machines, Alloy, B, TLA, VDM, and Z - 5th International Conference, ABZ 2016*, Lecture Notes in Computer Science 9675, pp. 250–255, Springer, 2016.

[2] R. Lezuo, P. Paulweber, and A. Krall, "CASM - Optimized Compilation of Abstract State Machines," in *SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems (LCTES)*, pp. 13–22, ACM, 2014.

[3] Y. Gurevich, B. Rossman, and W. Schulte, "Semantic Essence of AsmL," in *Formal Methods for Components and Objects*, pp. 240–259, Springer, 2004.

[4] R. Farahbod, V. Gervasi, and U. Glässer, "CoreASM: An Extensible ASM Execution Engine," *Fundamenta Informaticae*, vol. 77, no. 1-2, pp. 71–104, 2007.

universität
wien

Thank you for your attention!