A Model-Based Transformation Approach to Reuse and Retarget CASM Specifications

Philipp Paulweber and Uwe Zdun

{philipp.paulweber,uwe.zdun}@univie.ac.at

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

ABZ'16
May 27, 2016
Linz, Austria

CASM

Reuse/Retarget

WHY?

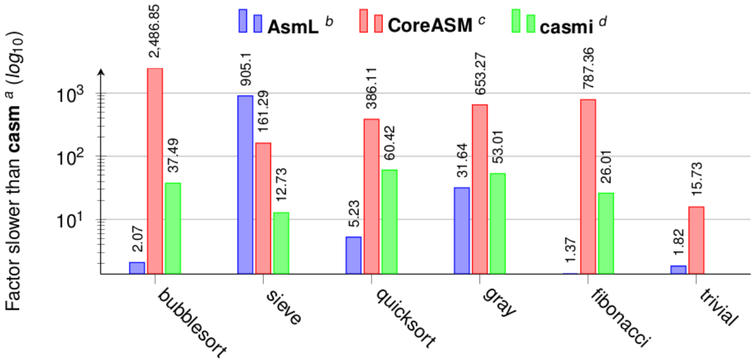
Models

Transformation

Why CASM?

- originally introduced by Lezuo, Barany and Krall [1]
- purpose:
  - specify machine languages
  - enable efficient (fast) execution
  - verified instruction set simulation [2]
- language:
  - subset of rules from CoreASM
  - statically typed (optimizations [3])
  - numeric and symbolic execution
- designed for:
  - small updates (partial updates, update-set)
  - large number of (machine) steps

Why CASM?

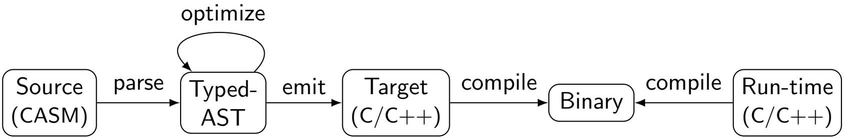


a) CASM Compiler without Optimizations [3]
b) Microsoft .NET-based Compiler [4]
c) Java-based Interpreter [5]
d) CASM AST-based Interpreter [3]

[3] 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

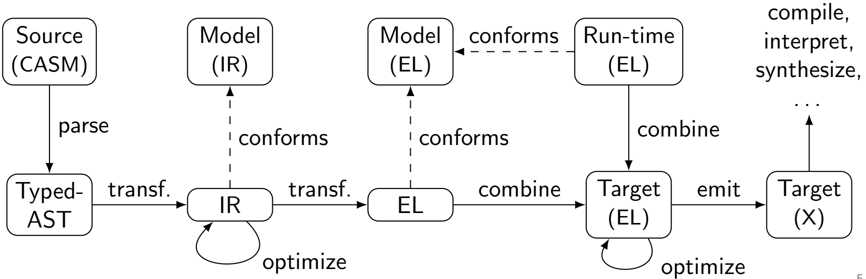
Why Reuse/Retarget?

- current tools
  - fixed execution environment (Java/JVM, .NET, . . .)
  - closed source projects (AsmL, CASM [3])
- embed specified ASMs in different contexts
  - software high-level (C, Java, Python, . . .)
  - software low-level (LLVM, . . .)
  - hardware high-level (VHDL, Verilog, SystemVerilog, . . .)
  - hardware low-level (Netlist, . . .)
- compiler design proposed by Lezuo, Paulweber and Krall [3]:



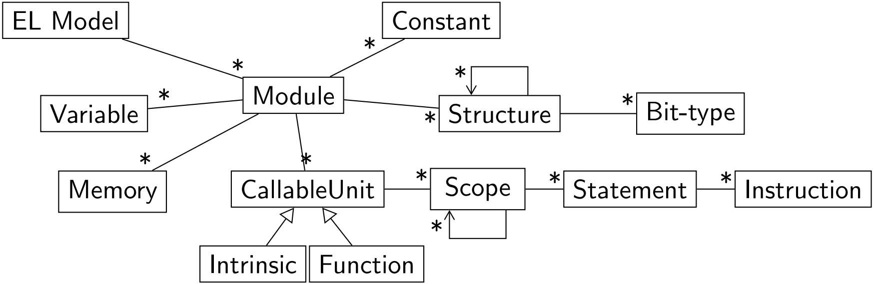
Why Models and Transformations?

- research interests:
  - investigate optimization potential of ASMs
  - efficient target code generation for ASM specifications
- IR (CASM Intermediate Representation) Model
  - contains run-time behavior
  - focus on ASM-based analyses and transformations
  - possible optimizations: Redundant Lookup/Update Elimination [3]
- EL (Emitting Language) Model
  - CASM unaware computational description
  - CASM run-time behavior implemented once in the EL model
  - focuses to ease to emit to different languages

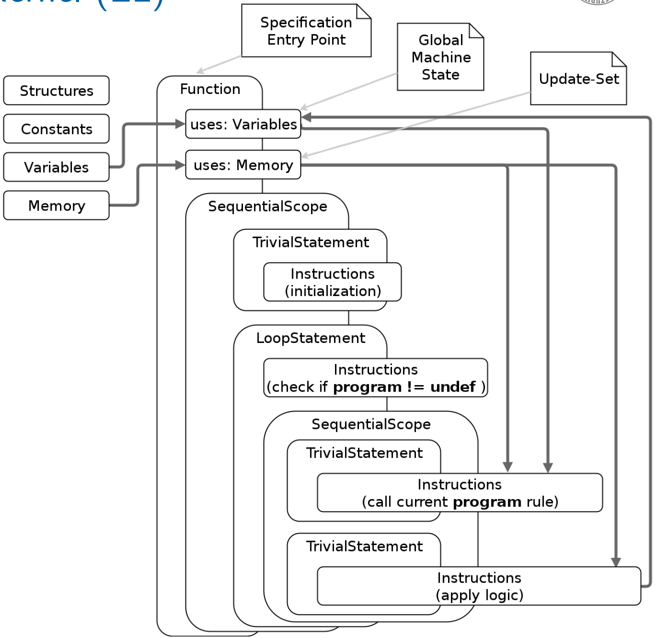


EL Model Design

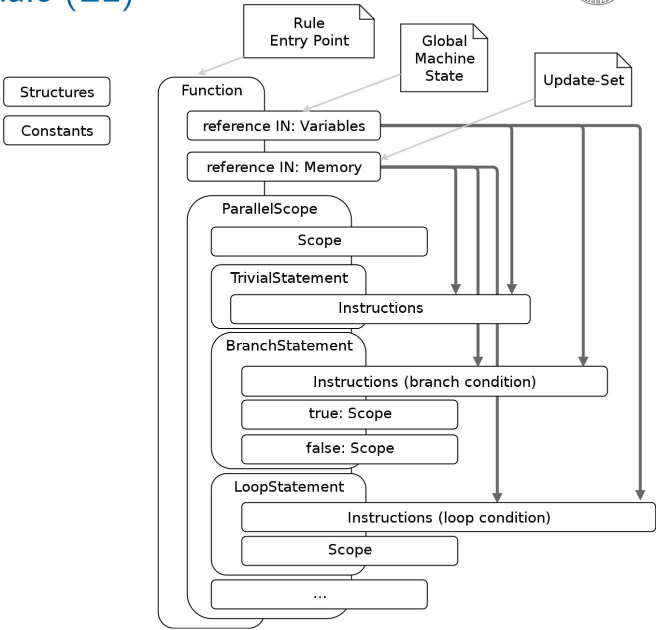
- initially based on LLVM, but inappropriate
- statically typed (bit-precise, structured)
- parallel and sequential scopes
- fixed-size memory block components
- allocation `id` concept



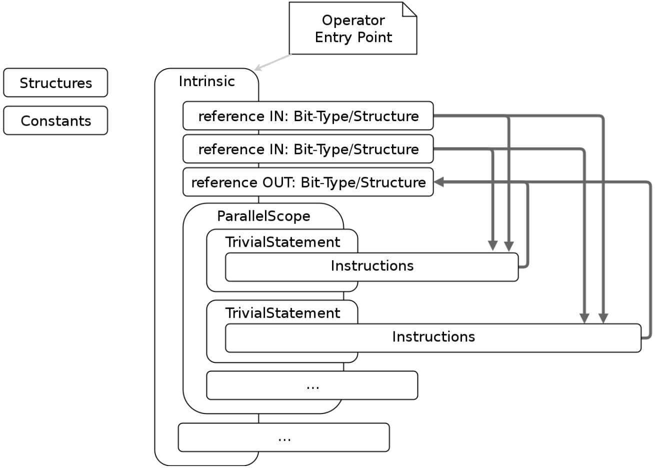
CASM Kernel (EL)



CASM Rule (EL)


Rule
Entry Point
Global
Machine
State
Update-Set
Structures
Constants
Function
reference IN: Variables
reference IN: Memory
ParallelScope
Scope
TrivialStatement
Instructions
BranchStatement
Instructions (branch condition)
true: Scope
false: Scope
LoopStatement
Instructions (loop condition)
Scope
...

CASM Operator (EL)



EL Model to Text

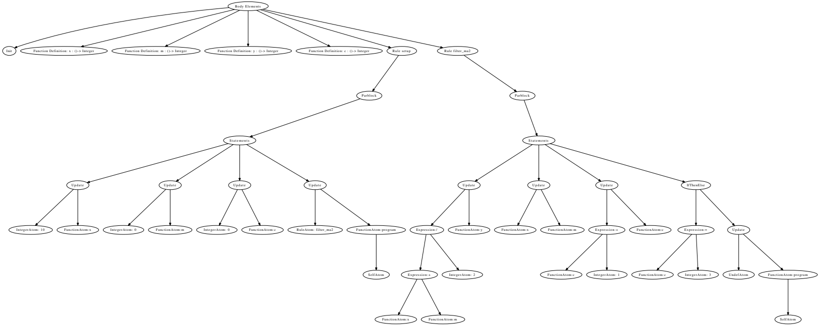
- planned are several different software and hardware back-ends of the EL model
- main focus for now is `C` and `VHDL`
- C
  - full sequential implementation of the EL elements
  - simple element translation
- VHDL
  - mixed sequential/parallel implementation
  - sequential logic (asynchronous design, 4-phase handshake with bundled data)

CASM Running Example

```
CASM filter_specification

init setup

function x : -> Integer // initially undef
function m : -> Integer // initially undef
function y : -> Integer // initially undef
function c : -> Integer // initially undef

rule setup =
{ // par
    m := 0
    x := 10
    c := 0
    program( self ) := @filter_ma2
} // endpar

rule filter_ma2 =
{
    m := x
    y := ( x + m ) / 2

    c := c + 1
```

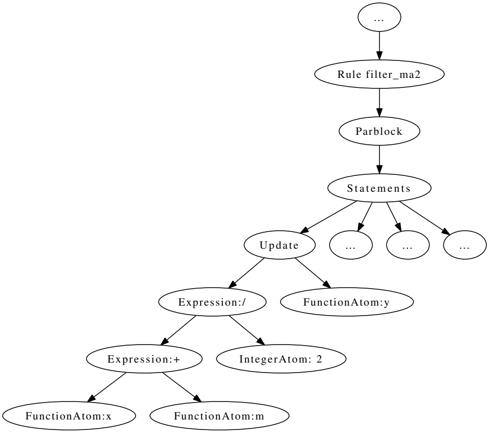
CASM AST Running Example

- full typed AST (rendered with dot - graphviz version 2.26.3)



CASM AST Running Example (cont'd)

- snippet of `filter_ma2` rule and `y` function update



CASM IR Running Example

```
// ...
lbl3, .function x
lbl4, .function m
lbl5, .function y
lbl11, .integer 2
// ...
lbl29, .rule filter_ma2
lbl30,   .par
lbl31,     .statement
lbl32,       .location , lbl5 (Integer)
lbl33,       .location , lbl3 (Integer)
lbl34,       .lookup , lbl33 (Integer)
lbl35,       .location , lbl4 (Integer)
lbl36,       .lookup , lbl35 (Integer)
lbl37,       .add , lbl34 (Integer), lbl36 (Integer)
lbl38,       .div , lbl37 (Integer), lbl11 (Integer)
lbl39,       .update , lbl32 (Integer), lbl38 (Integer)
// ...
```

EL Running Example

```
lbl17, .const_struct (Structure( Bit(64), Bit(1) )) ( 2, true ) // ...
lbl59, .intrinsic casmrt_location_x // ...
lbl65, .intrinsic casmrt_location_m // ...
lbl71, .intrinsic casmrt_location_y // ...
lbl83, .intrinsic casmrt_update_Integer // ...
lbl134, .intrinsic casmrt_lookup_Integer // ...
lbl145, .intrinsic casmrt_add_Integer_Integer_Integer // ...
lbl166, .intrinsic casmrt_div_Integer_Integer_Integer // ...
// ...
lbl217, .function casm_rule_filter_ma2
lbl218, .reference refs, in lbl218 (Interconnect)
lbl219, .reference uset, in lbl219 (Memory)
lbl220, .par
lbl221,   .statement
lbl222,     .alloc (Bit(48))
lbl223,     .call, lbl71, lbl222
lbl224,     .alloc (Bit(48))
lbl225,     .call, lbl59, lbl224
lbl226,     .alloc (Structure(Bit(64),Bit(1)))
lbl227,     .call, lbl134, lbl218, lbl219, lbl224, lbl226
lbl228,     .alloc (Bit(48))
lbl229,     .call, lbl65, lbl228
lbl230,     .alloc (Structure( Bit(64), Bit(1) ))
lbl231,     .call, lbl134, lbl218, lbl219, lbl228, lbl230
lbl232,     .alloc (Structure( Bit(64), Bit(1) ))
lbl233,     .call, lbl145, lbl226, lbl230, lbl232
lbl234,     .alloc (Structure( Bit(64), Bit(1) ))
lbl235,     .call, lbl166, lbl232, lbl17, lbl234
lbl236,     .call, lbl83, lbl219, lbl222, lbl234
```

EL Running Example (cont'd)

```
// ...
lbl145, casmrt_add_Integer_Integer_Integer
lbl146, .reference a, in (Structure( Bit(64), Bit(1) ))
lbl147, .reference b, in (Structure( Bit(64), Bit(1) ))
lbl148, .reference t, out (Structure( Bit(64), Bit(1) ))
lbl149, .par
lbl150,   .statement
lbl151,     .extract , lbl146, lbl101 value (Bit(64))
lbl152,     .load , lbl151 (Bit(64))
lbl153,     .extract , lbl147, lbl101 value (Bit(64))
lbl154,     .load , lbl153 (Bit(64))
lbl155,     .adds , lbl152, lbl154
lbl156,     .extract , lbl148, lbl101 value (Bit(64))
lbl157,     .store , lbl155, lbl156
lbl158,   .statement
lbl159,     .extract , lbl146, lbl106 isdef (Bit(1))
lbl160,     .load , lbl159
lbl161,     .extract , lbl147, lbl106 isdef (Bit(1))
lbl162,     .load , lbl161
lbl163,     .land , lbl160, lbl162
lbl164,     .extract , lbl148, lbl106 isdef (Bit(1))
lbl165,     .store , lbl163, lbl164
// ...
```

C Running Example

```
// Function 'lbl217'
void casm_rule_filter_ma2
( uint64_t** lbl218 /*refs in*/, Update* lbl219 /*uset in*/ )
{ // par 'lbl220'
    // stmt 'lbl221'
    {
        uint64_t lbl222;// alloc
        casmrt_location_y( (uint64_t*)&lbl222 ); // call 1
        uint64_t lbl224;// alloc
        casmrt_location_x( (uint64_t*)&lbl224 ); // call 1
        Integer lbl226;// alloc
        casmrt_lookup_Integer(
            (uint64_t**)lbl218, (Update*)lbl219, (uint64_t)lbl224, (Integer
        uint64_t lbl228;// alloc
        casmrt_location_m( (uint64_t*)&lbl228 ); // call 1
        Integer lbl230;// alloc
        casmrt_lookup_Integer(
            (uint64_t**)lbl218, (Update*)lbl219, (uint64_t)lbl228, (Integer
        Integer lbl232;// alloc
        casmrt_add_Integer_Integer_Integer(
            (Integer*)&lbl226, (Integer*)&lbl230, (Integer*)&lbl232 ); // c
        Integer lbl234;// alloc
        casmrt_div_Integer_Integer_Integer(
            (Integer*)&lbl232, (Integer*)&lbl17, (Integer*)&lbl234 ); // ca
        casmrt_update_Integer(
            (Update*)lbl219, (uint64_t)lbl222, (Integer*)&lbl234 ); // call
    }
// ...
```

C Running Example (cont'd)

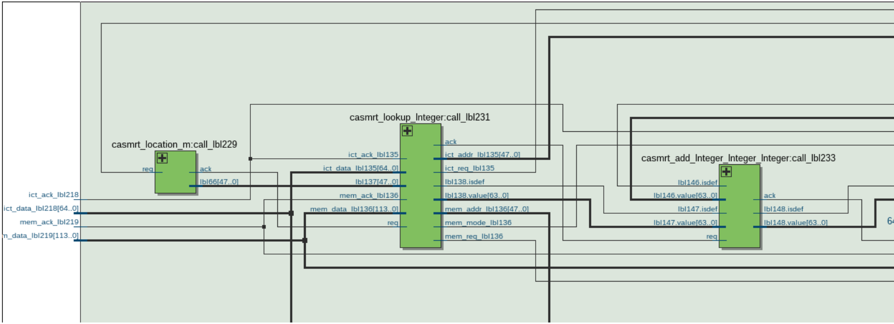
```
// Intrinsic 'lbl145'
static inline void casmrt_add_Integer_Integer_Integer
( Integer* lbl146 /*a in*/, Integer* lbl147 /*b in*/
, Integer* lbl148 /*t out*/
)
{ // par 'lbl149'
    // stmt 'lbl150'
    {
        uint64_t* lbl151 = &(lbl146->value); // extract (T2) 'a'
        uint64_t lbl152 = *lbl151; // load
        uint64_t* lbl153 = &(lbl147->value); // extract (T2) 'b'
        uint64_t lbl154 = *lbl153; // load
        uint64_t lbl155 = (uint64_t)((int64_t)lbl152 + (int64_t)lbl154);
        uint64_t* lbl156 = &(lbl148->value); // extract (T2) 't'
        *lbl156 = lbl155; // store 'lbl157'
    }
    // stmt 'lbl158'
    {
        uint8_t* lbl159 = &(lbl146->isdef); // extract (T2) 'a'
        uint8_t lbl160 = *lbl159; // load
        uint8_t* lbl161 = &(lbl147->isdef); // extract (T2) 'b'
        uint8_t lbl162 = *lbl161; // load
        uint8_t lbl163 = (lbl160 & lbl162);
        uint8_t* lbl164 = &(lbl148->isdef); // extract (T2) 't'
        *lbl164 = lbl163; // store 'lbl165'
    }
}
```

VHDL Running Example

```
-- Intrinsic 'lbl145'
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use work.Structure.all;
use work.Constants.all;
use work.Variables.all;
use work.Instruction.all;
entity casmrt_add_Integer_Integer_Integer is port
( req : in  std_logic
; ack : out std_logic
; lbl146 : in struct_Integer -- a in
; lbl147 : in struct_Integer -- b in
; lbl148 : out struct_Integer -- t out
);
end casmrt_add_Integer_Integer_Integer;
architecture \@casmrt_add_Integer_Integer_Integer@\ of casmrt_add_Integer_I
  signal req_lbl149 : std_logic := '0'; -- '.par'
  signal ack_lbl149 : std_logic := '0';
  signal req_lbl150 : std_logic := '0'; -- '.statement'
  signal ack_lbl150 : std_logic := '0';
  signal sig_lbl150 : std_logic := '0';
  signal sig_lbl151 : std_logic := '0'; -- .extract
  signal     lbl151 : std_logic_vector( 63 downto 0 ); -- .extract
  signal sig_lbl152 : std_logic := '0'; -- .load
  signal     lbl152 : std_logic_vector( 63 downto 0 ); -- .load
  signal sig_lbl153 : std_logic := '0'; -- .extract
  signal     lbl153 : std_logic_vector( 63 downto 0 ); -- .extract
  signal sig_lbl154 : std_logic := '0'; -- .load
  signal     lbl154 : std_logic_vector( 63 downto 0 ); -- .load
```

VHDL Running Example (cont'd)

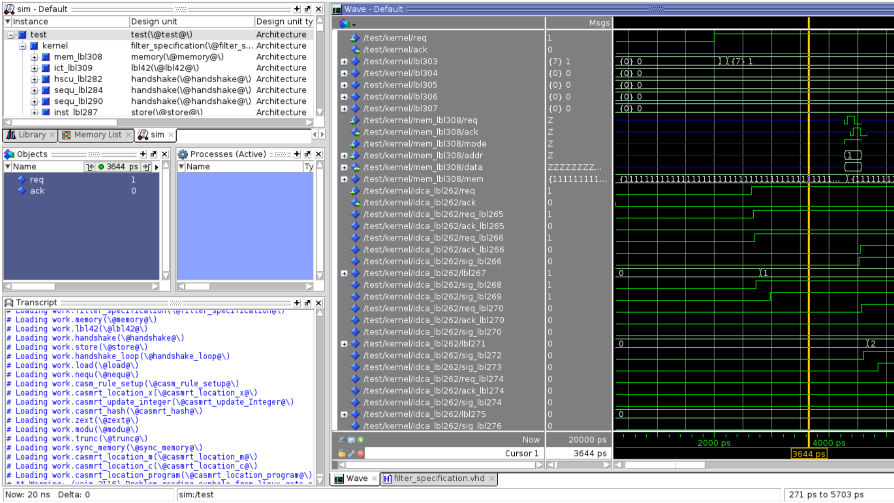
- Quartus Prime: Version 15.1.0 Build 185 10/21/2015 SJ Lite Edition



HDL Simulator Running Example

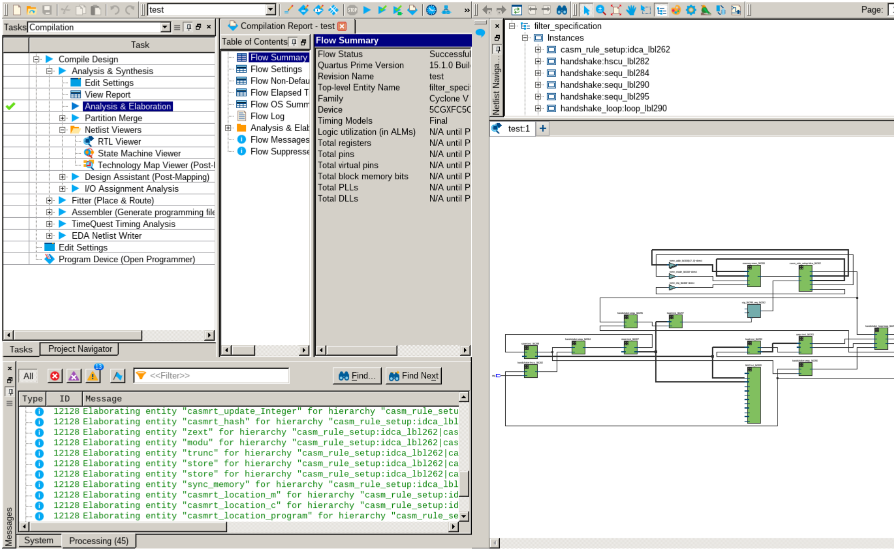
universität wien

- ModelSim Altera Starter Edition 10.4b Revision: 2015-05-27



RTL Viewer Running Example

- Quartus Prime: Version 15.1.0 Build 185 10/21/2015 SJ Lite Edition



Preliminary Results Running Example

- CASM input specification
  - about 20 lines of code (running example)
- C Back-end:
  - about 500 lines of code (running example)
  - about 80 times faster than old design (evaluation with large machine steps)
  - almost a direct mapping of the EL module
- VHDL Back-end:
  - about 2250 lines of code (running example)
  - creates valid entity composition hierarchy of REQ/ACK chains
  - simulation model still under evaluation (signaling issues etc.)

Conclusion

- new transformation approach for (C)ASMs
- compiler implemented in C++
- sources will be available soon (GPLv3)
  - GitHub: github.com/casm-lang
  - website: casm-lang.org

References

[1] R. Lezuo, G. Barany, and A. Krall, "CASM: Implementing an Abstract State Machine based Programming Language," in *Software Engineering (Workshops)*, pp. 75–90, 2013.

[2] R. Lezuo and A. Krall, "Using the CASM Language for Simulator Synthesis and Model Verification," in *Proceedings of the 2013 Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools*, p. 6, ACM, 2013.

[3] 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.

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

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

Special thanks to Florian Hahn for the help with the CASM front-end.

Thank you for your attention!