# A pred-LL(*) Parsable Typed Higher-Order Macro System for Architecture Description Languages

**Christoph Hochrainer** Andreas Krall

# Outline

- Background
  - Macros
  - Architecture Description Language (ADL)
- Introduction
  - Vienna Architecture Description Language (VADL)
  - Instruction Set Architecture (ISA) Specification
- Macro System
- Evaluation

# Macros

# Macros

- User-defined **functions**, **transforming** one **program sequence** to another **program sequence**

```
macro circleArea(r) = (3.14*(r)*(r))

double area = circleArea( 2 )
```

# Macros

- User-defined **functions**, **transforming** one **program sequence** to another **program sequence**

```
macro circleArea(r) = (3.14*(r)*(r))

double area = circleArea( 2 )
```



macro expansion

```
double area = (3.14*(2)*(2))
```

# Macros

- User-defined **functions**, **transforming** one **program sequence** to another **program sequence**
- **Pattern-based** vs **procedural**

```
macro circleArea(r) = (3.14*(r)*(r))

double area = circleArea( 2 )
```



macro expansion

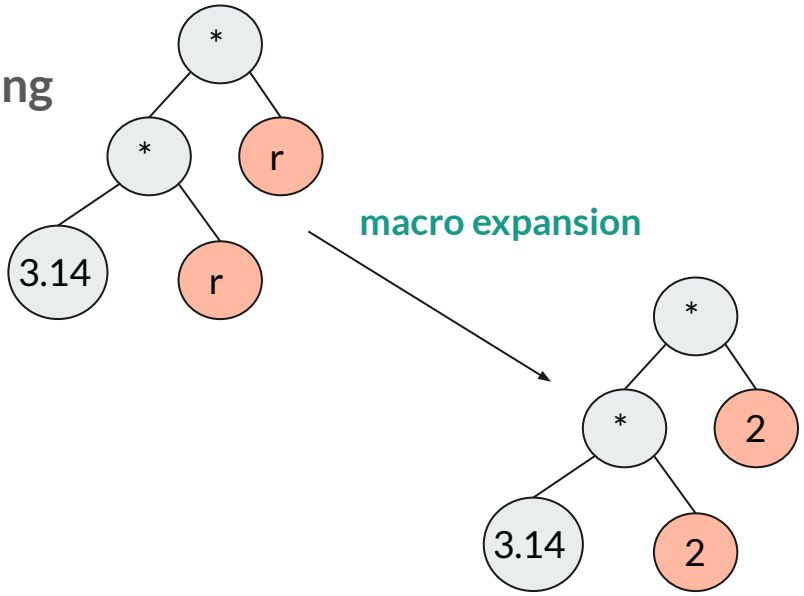
```
double area = (3.14*(2)*(2))
```

# Macros

- User-defined **functions**, **transforming** one **program sequence** to another **program sequence**
- **Pattern-based** vs **procedural**
- **Lexical** vs **syntactic**

```
macro circleArea(r) = (3.14*(r)*(r))

double area = circleArea( 2 )
```

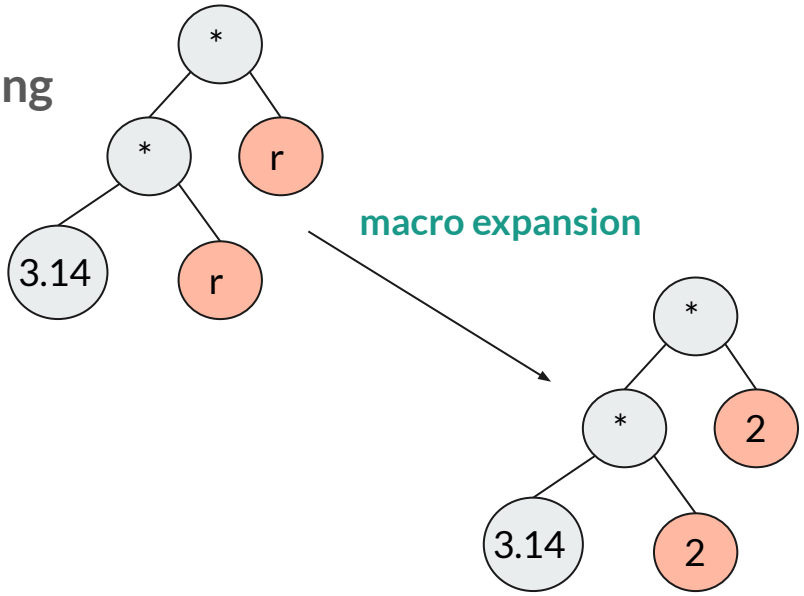


# Macros

```
macro circleArea(r) = (3.14*(r)*(r))

double area = circleArea( 2 )
```

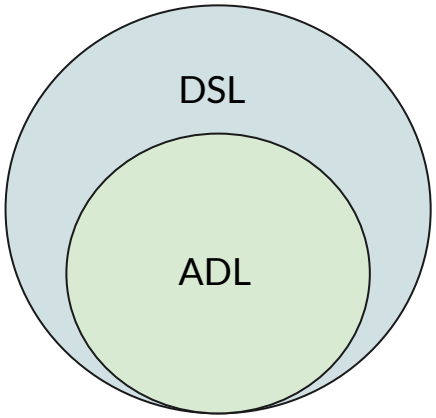
- User-defined **functions**, **transforming** one **program sequence** to another **program sequence**
- **Pattern-based** vs **procedural**
- **Lexical** vs **syntactic**



# Architecture Description Languages

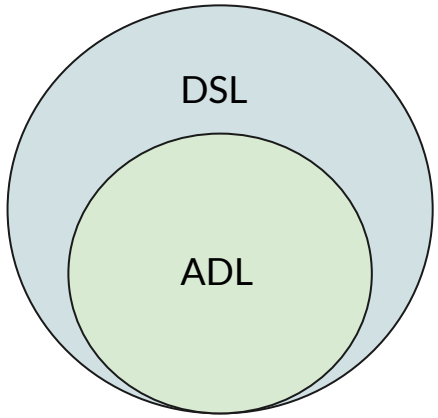
# Architecture Description Languages

- Family of **Domain Specific Languages**



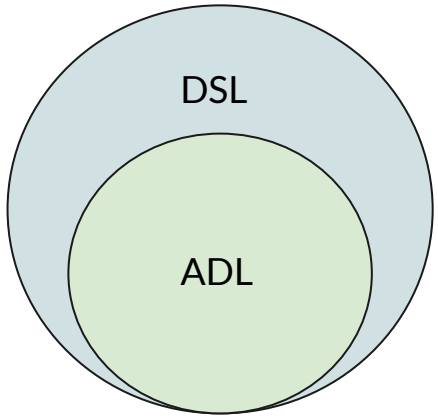
# Architecture Description Languages

- Family of **Domain Specific Languages**
- **Description** of how a system looks and behaves



# Architecture Description Languages

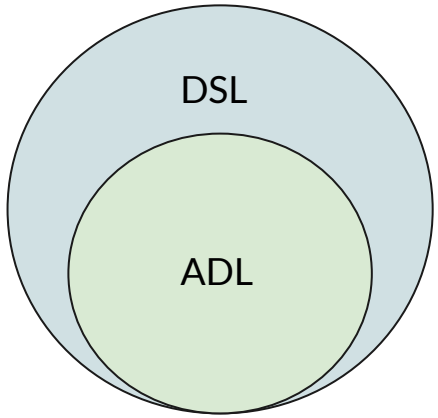
- Family of **Domain Specific Languages**
- **Description** of how a system looks and behaves




**Description languages are usually NOT executable!**

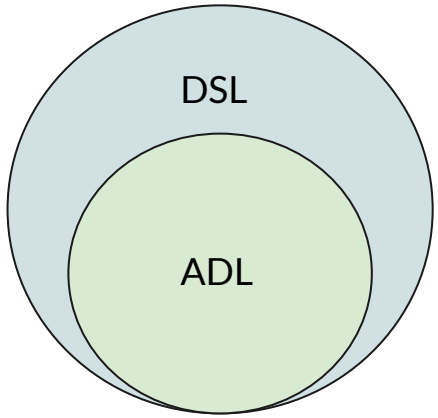
# Architecture Description Languages

- Family of **Domain Specific Languages**
- **Description** of how a system looks and behaves
- Used to generate or verify **artifacts**



# Architecture Description Languages

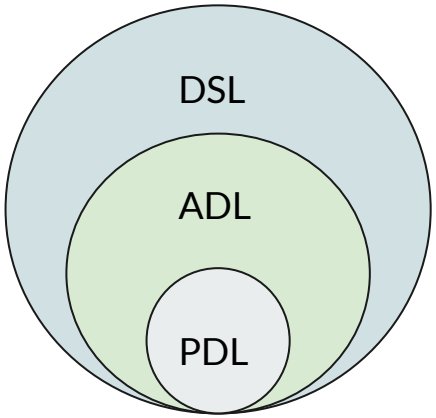
- Family of **Domain Specific Languages**
- **Description** of how a system looks and behaves
- Used to generate or verify **artifacts**
- Can be part of a **host-language** or **standalone**



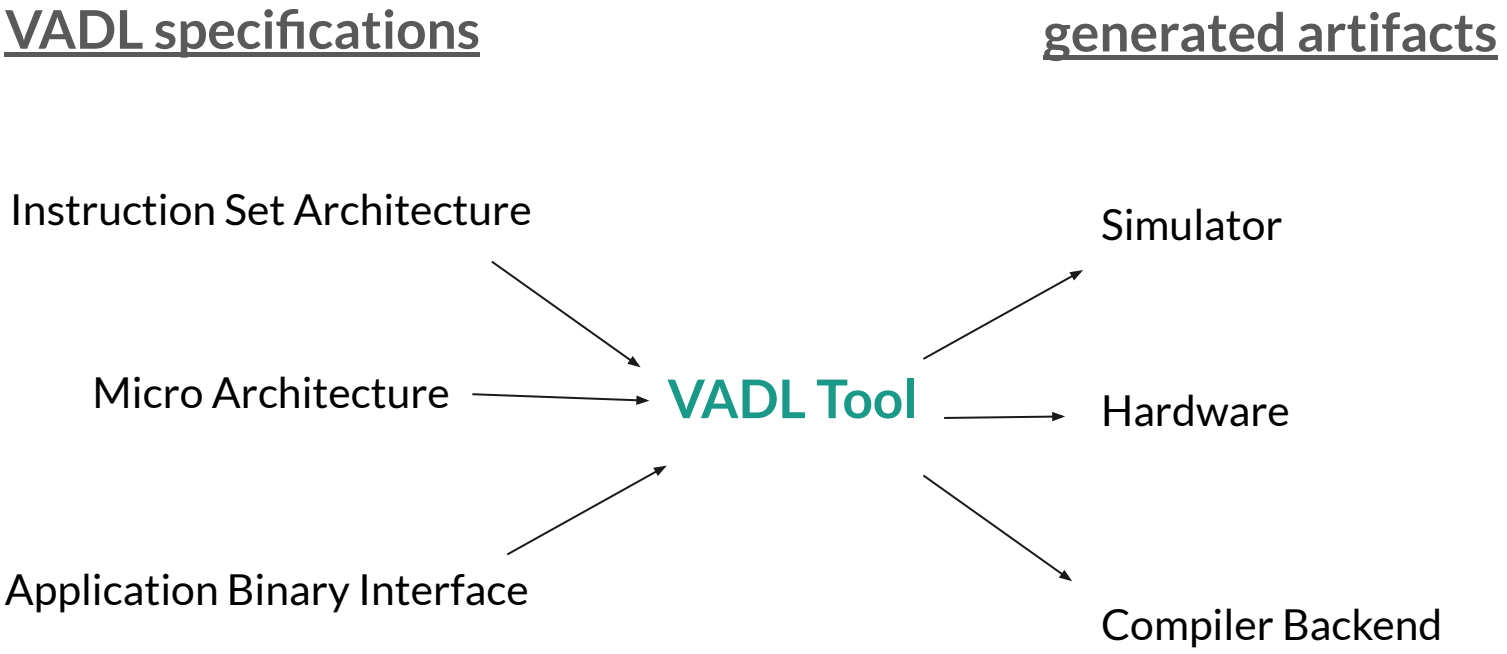
# Vienna Architecture Description Language

# Vienna Architecture Description Languages (VADL)

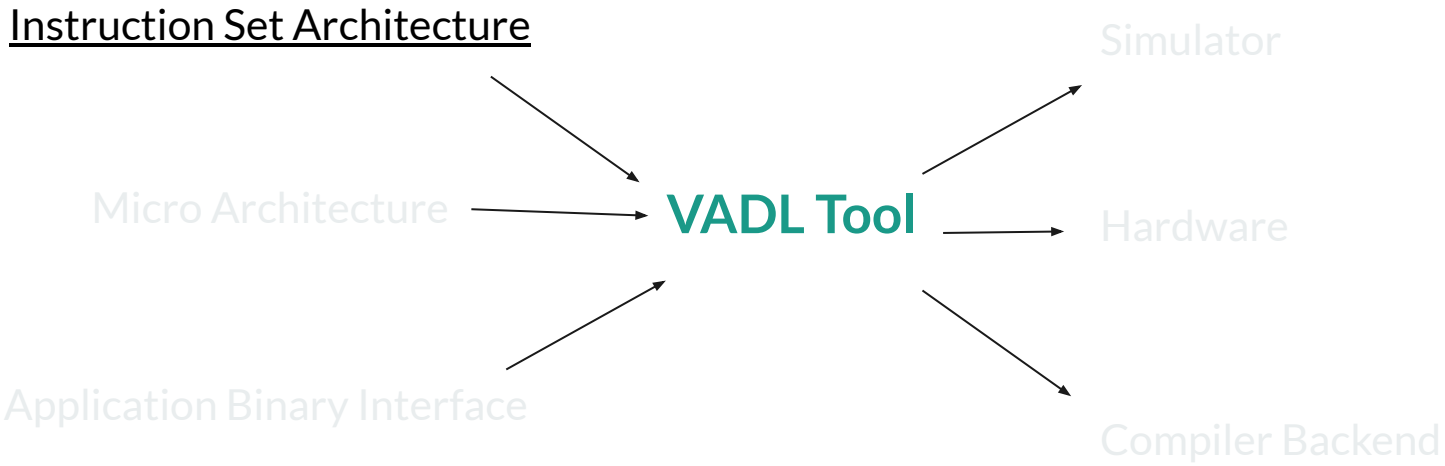
- Family of **Domain Specific Languages**
- **Description** of how a system looks and behaves
- Used to generate or verify **artifacts**
- Can be part of a **host-language** or **standalone**
- **Processor Description Language (PDL)**
- **Work in progress**



# Vienna Architecture Description Languages (VADL)



# Vienna Architecture Description Languages (VADL)



# VADL ’s Instruction Set Architecture Specification

# VADL 's Instruction Set Architecture

```
instruction ADD : F =
{
        X(rd) := X(rs1) + X(rs2)
}
```

# VADL 's Instruction Set Architecture

```
instruction ADD : F =
{
        X(rd) := X(rs1) + X(rs2)
}
```

# VADL 's Instruction Set Architecture

```
instruction ADD : F =
{
        X(rd) := X(rs1) + X(rs2)
}
```

```
assembler ADD =
( "ADD"
, register(rd), ","
, register(rs1), ","
, register(rs2), ","
)
```

# VADL 's Instruction Set Architecture

```
instruction ADD : F =
{
        X(rd) := X(rs1) + X(rs2)
}
```

```
assembler ADD =
( "ADD"
, register(rd), ","
, register(rs1), ","
, register(rs2), ","
)
```

```
encoding ADD : F =
{ opcode = 0b011'0011
, funct     = 0b000
}
```

# VADL 's Instruction Set Architecture

```
instruction ADD : F =
{
        X(rd) := X(rs1) + X(rs2)
}
```

```
encoding ADD : F =
{ opcode = 0b011'0011
, funct     = 0b000
}
```

```
assembler ADD =
( "ADD"
, register(rd), ","
, register(rs1), ","
, register(rs2), ","
)
```

and now repeat for SUB, AND, OR, ...

# VADL 's Instruction Set Architecture

```
instruction ADD : F =
{
        X(rd) := X(rs1) + X(rs2)
}
```

```
encoding ADD : F =
{ opcode = 0b011'0011
, funct     = 0b000
}
```

```
assembler ADD =
( "ADD"
, register(rd), ","
, register(rs1), ","
, register(rs2), ","
)
```

and now repeat for SUB, AND, OR, ...

NO!

# VADL 's Macro System

# VADL 's Macro System

```
instruction ADD : F =
{
        X(rd) := X(rs1) + X(rs2)
}
```

# VADL 's Macro System

```
instruction ADD : F =
{
        X(rd) := X(rs1) + X(rs2)
}
```

# VADL 's Macro System

```
instruction $id : F =
{
        X(rd) := X(rs1) $op X(rs2)
}
```

# VADL 's Macro System

```
instruction $id : F =
{
        X(rd) := X(rs1) $op X(rs2)
}
```

# VADL 's Macro System

```
model BinaryInstruction( id : Id, op : BinOp ) : IsaDef =
{
        instruction $id : F =
        {
                X(rd) := X(rs1) $op X(rs2)
        }
}
```

# VADL 's Macro System

```
model BinaryInstruction( id : Id, op : BinOp ) : IsaDef =
{
        instruction $id : F =
        {
                X(rd) := X(rs1) $op X(rs2)
        }
}

$BinaryInstruction( ADD ;  + )
$BinaryInstruction( SUB   ;  -  )
$BinaryInstruction( AND ;  & )
$BinaryInstruction( OR    ;  |   )
```

# VADL 's Macro System

```
instruction ADD_eq : F =
{
        if ( check_condition( EQ ) ) then
                X(rd) := X(rs1) + X(rs2)
}
```

# VADL 's Macro System

```
instruction ADD_eq : F =
{
        if ( check_condition( EQ ) ) then
            X(rd) := X(rs1) + X(rs2)
}
```

# VADL 's Macro System

```
instruction ADD_eq : F =
{
        if ( check_condition( EQ ) ) then
            X(rd) := X(rs1) + X(rs2)
}
```

There are 15 different condition flags for AAarch32!

e.g. NE, LS, GE, LT, GT, …

# VADL 's Macro System

```
model BinaryInstruction( id : Id, op : BinOp, cond :  Id, postfix : Str ) : IsaDef
{
        instruction ExtendId( $id, "_", $postfix ) : F =
        {
                if ( check_condition( $cond ) ) then
                        X(rd) := X(rs1) $op X(rs2)
        }
}
```

- ExtendId is a lexical extension to our macro system
- ExtendId : ( Id,  Str [,  Str]* )  –>  Id

# VADL 's Macro System

```
model BinaryInstruction( id : Id, op : BinOp, cond :  Id, postfix : Str ) : IsaDef
{
        // ...
}

$BinaryInstruction( ADD ; + ; EQ ; "eq" )
$BinaryInstruction( ADD ; + ; NE ; "ne" )
$BinaryInstruction( ADD ; + ; LE  ; "le"  )
$BinaryInstruction( ADD ; + ; LT  ; "lt"  )
...
```

# VADL 's Macro System

```
model BinaryInstruction( id : Id, op : BinOp, cond :  Id, postfix : Str ) : IsaDef
{
        // ...
}

$BinaryInstruction( ADD ; + ; EQ ; "eq" )
$BinaryInstruction( ADD ; + ; NE ; "ne" )
$BinaryInstruction( ADD ; + ; LE  ; "le"  )
$BinaryInstruction( ADD ; + ; LT  ; "lt"   )
...
```

NO!

# VADL 's Higher-Order Macros and Type Composition

# VADL 's Macro System

```
model BinaryInstruction( id : Id, op : BinOp, cond :  Id, postfix : Str ) : IsaDef =
{
        // ...
}
```

# VADL 's Macro System

```
record InstInfo ( id : Id , op : BinOp )
record CondInfo ( id : Id, postfix : Str )

model BinaryInstruction( inst : InstInfo, cond :  CondInfo ) : IsaDef =
{
        // ...
}
```

# VADL 's Macro System

```
record InstInfo ( id : Id , op : BinOp )
record CondInfo ( id : Id, postfix : Str )

model BinaryInstruction( inst : InstInfo, cond :  CondInfo ) : IsaDef =
{
        instruction ExtendId( $inst.id, "_", $cond.postfix ) : F =
        {
                if ( check_condition( $cond.id ) ) then
                        X(rd) := X(rs1) $inst.op X(rs2)
        }
}
```

# VADL 's Macro System

```
record InstInfo ( id : Id , op : BinOp )
record CondInfo ( id : Id, postfix : Str )

model BinaryInstruction( inst : InstInfo, cond :  CondInfo ) : IsaDef =
{
        // ...
}

$BinaryInstruction( ( ADD ; + ) ; ( EQ ; "eq" ) )
$BinaryInstruction( ( ADD ; + ) ; ( EQ ; "ne" ) )
$BinaryInstruction( ( ADD ; + ) ; ( EQ ; "le" ) )
$BinaryInstruction( ( ADD ; + ) ; ( EQ ; "lt" ) )
...
```

# VADL 's Macro System

```
record InstInfo ( id : Id , op : BinOp )
record CondInfo ( id : Id, postfix : Str )

model-type InstType = ( InstrInfo, CondInfo ) -> IsaDef

model BinaryInstruction( inst : InstInfo, cond :  CondInfo ) : IsaDef =
{
        // ...
}
```

# VADL 's Macro System

```
record InstInfo ( id : Id , op : BinOp )
record CondInfo ( id : Id, postfix : Str )

model-type InstType = ( InstrInfo, CondInfo ) -> IsaDef

model BinaryInstruction( inst : InstInfo, cond :  CondInfo ) : IsaDef = { /* … */ }

model CondWrapper ( instModel : InstType, info : InstrInfo ) : IsaDef =
{
        $instModel( $info ;  ( “eq” ; EQ )  )
        $instModel( $info ;  ( “ne” ; NE )  )
        // …
}
```

# VADL 's Macro System

```
model BinaryInstruction( inst : InstInfo, cond :  CondInfo ) : IsaDef = { /* ... */ }

model CondWrapper ( instModel : InstType, info : InstrInfo ) : IsaDef = { /* ... */ }


$CondWrapper( BinaryInstruction ; ( ADD ;   +  ) )
$CondWrapper( BinaryInstruction ; ( SUB  ;   -  ) )
$CondWrapper( BinaryInstruction ; ( AND  ;  &  ) )
$CondWrapper( BinaryInstruction ; ( OR    ;  |  ) )
...
```

# Evaluation

# Evaluation

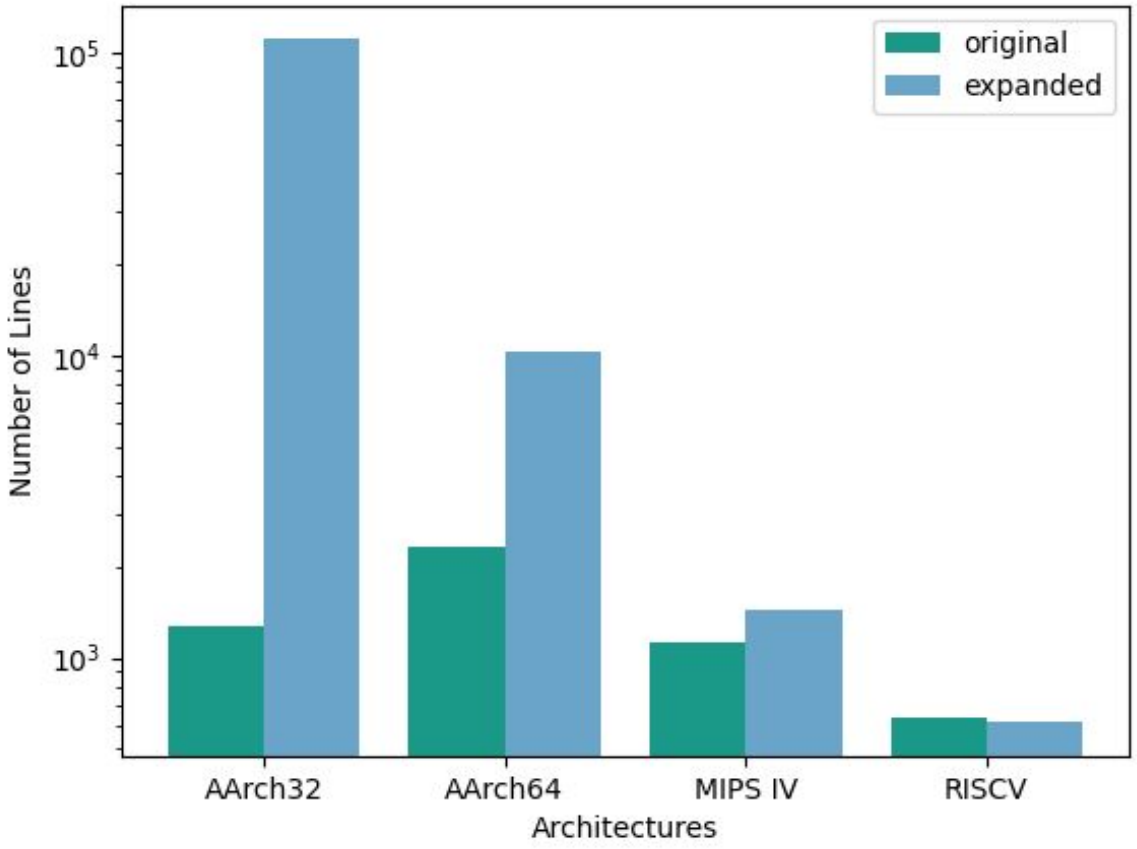
- Integration into the VADL tool
- Instruction set architecture specification for
  - AArch64
  - AArch32
  - RISC-V
  - MIPS IV

# Evaluation

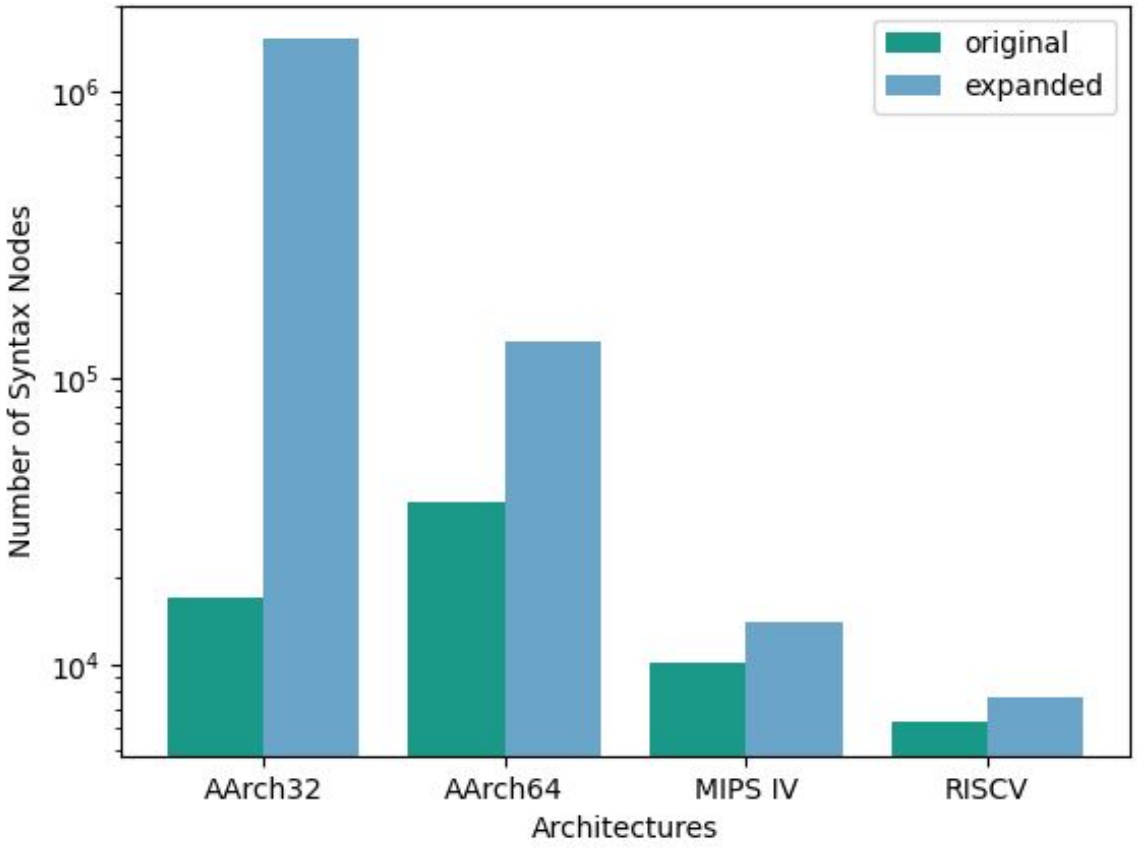
- Integration into the VADL tool
- Instruction set architecture specification for
  - AArch64
  - AArch32
  - RISC-V
  - MIPS IV

**We are interested in runtime and code size!**

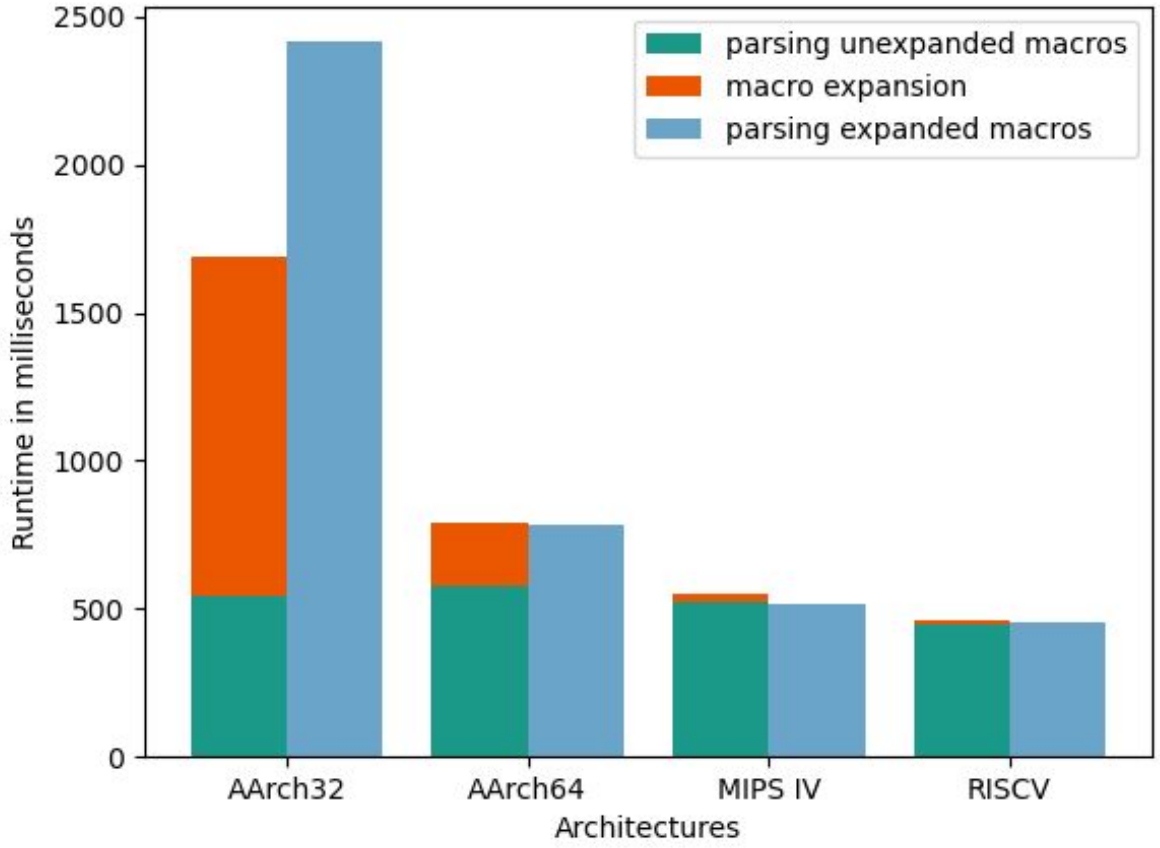
# Evaluation


original
expanded
$10^5$
$10^4$
$10^3$
Number of Lines
AArch32
AArch64
MIPS IV
RISCV
Architectures

# Evaluation


original
expanded
$10^6$
$10^5$
$10^4$
Number of Syntax Nodes
AArch32
AArch64
MIPS IV
RISCV
Architectures

# Evaluation


parsing unexpanded macros
macro expansion
parsing expanded macros
Runtime in milliseconds
2500
2000
1500
1000
500
0
AArch32
AArch64
MIPS IV
RISCV
Architectures

# Takeaways

- **Specification languages** can profit alot from **syntactical pattern-based** macro systems
- **Faster** and more **concise** specifications
- **model**: **Syntax-type safe higher-order** macro templates
- **record** and **model-type**: **Composable syntax types** and **type alias**

# Appendix

# Syntax Versions

VERSION 1: X(rs1) $(BinOp op) X(rs2) // ERROR at PARSE TIME

VERSION 2: X(rs1) $op X(rs2) // ERROR at EXPANSION

# Hygiene

```
model M ( /* ... */ ) : /* ... */ =
{
    instruction $id : F =
    {
        X(rd) := X(rs1) $op X(rs2)
    }
}
```

full control over identifier to reference them later

capture **X**

# Tables | Macro Elements

| | AArch32 | AArch64 | MIPS IV | RISCV | TriLen |
|---|---|---|---|---|---|
| Models | 90 | 142 | 64 | 4 | 21 |
| Placeholders | 1168 | 1270 | 311 | 30 | 178 |
| Instantiations | 225 | 322 | 163 | 24 | 56 |
| Records | 4 | 9 | 0 | 0 | 0 |
| Type-Alias | 4 | 8 | 0 | 0 | 0 |
| Match | 0 | 12 | 2 | 0 | 0 |

# Tables | Macro Parameters

| | | AArch32 | AArch64 | MIPS IV | RISCV | TriLen |
|---|---|---|---|---|---|---|
| Normal | Min | 0 | 0 | 1 | 4 | 2 |
| | Max | 9 | 8 | 5 | 6 | 8 |
| | Abs | 343 | 417 | 187 | 21 | 96 |
| | Avg | 3.81 | 2.93 | 2.92 | 5.25 | 4.57 |
| Flattened | Min | 0 | 0 | 1 | 4 | 2 |
| | Max | 14 | 15 | 5 | 6 | 8 |
| | Abs | 738 | 886 | 187 | 21 | 96 |
| | Avg | 8.2 | 6.23 | 2.92 | 5.25 | 4.57 |
| Higher-Order | Min | 0 | 0 | 0 | 0 | 0 |
| | Max | 3 | 2 | 0 | 0 | 0 |
| | Abs | 37 | 36 | 0 | 0 | 0 |
| | Avg | 0.41 | 0.25 | 0 | 0 | 0 |

# Syntax Types