Experimentation with a Prototype OpenModelica Compiler in Julia

John Tinnerholm, Adrian Pop, and Martin Sjölund
Motivation
Motivation

• Integration with the Julia ecosystem
• Provide a standard-compliant Modelica environment in Julia
  – Larger OpenSource community
  – Supporting VSS/Multi-mode DAE models within standard Modelica
• Outsourcing implementation language development
• Enable separate use of OpenModelica Compiler packages
  – Utilizing Julia package manager
  – Tool development without a monolithic compiler
Julia

- A new programming language by Jeff Bezanson, Stefan Karpinski and Viral B. Shah
- A language for Numerical and Symbolic Computation
- Many libraries for Linear Algebra, Differential Equations, Fast Fourier Transforms etc.
- H. Wilkinson Prize for Numerical Software in 2019
- Already in use for equation-based-modelling, Modia.jl
  - Support for Multi-Mode DAE Models, however non standard-compliant
- OMJulia for interoperability with OpenModelica
The MetaModelica to Julia translator
The MetaModelica to Julia translator

- Translated from the old OpenModelica frontend
- Maps MetaModelica packages to Julia modules
- Difficulties with certain syntactical constructs
  - Circular packages
  - Uniontypes with static methods

```julia
uniontype Equation
record EQ_IF
  Exp ifExp;
  list<EquationItem> equationTrueItems;
  list<tuple<Exp, list<EquationItem>>> elseifBranches;
  list<EquationItem> equationElseItems;
end EQ_IF;

record EQ_EQUALS
  Exp leftSide;
  Exp rightSide;
end EQ_EQUALS;
// ...
```
The MetaModelica to Julia translator

• Continuous integration of features
• Supporting parallel development of the existing compiler
MetaModelica via Metaprogramming

- Julia’s AST macros makes language extensions easy
- Structural elements such as inheritance, uniontypes match and matchcontinue are provided

@Uniontype Equation begin
@Record EQ_IF begin
  @Record EQ_EQUALS begin
  end
  # ...
end
MetaModelica.jl

- Compiler runtime
  - MetaModelica.jl
  - Pattern-matching
  - Immutable List
  - Uniontypes

- Possible to use as a standalone package

```julia
@Uniontype Equation begin
@Record EQ_IF begin
  ifExp::Exp
  equationTrueItems::List{EquationItem}
  elseIfBranches::List{
    Tuple{Exp,List{EquationItem}}}
  equationElseItems::List{EquationItem}
end

@Record EQ_EQUALS begin
  leftSide::Exp
  rightSide::Exp
end
# ...
end
```
OMParser.jl

• The AST is constructed in ANTLR using C the foreign function interface of Julia
  – Capable of parsing any Modelica library (tested with MSL 3.2.3)
• Somewhat smaller memory footprint
  – Better data sharing
OMCompiler.jl

- Translated from the current frontend
- Able to parse and translate:
  - Absyn IR
  - SCode IR
  - DAE IR
OMCompiler.jl

- Generated Julia DAE IR compatible with existing MetaModelica DAE IR
- Somewhat smaller memory footprint

dae = DAE.DAE_LIST(Cons {
  DAE.Element
}DAE.COMP("HelloWorld", Cons {
  DAE.Element
}DAE.VAR(DAE.CREF_IDENT("x", DAE.T_REAL(Nil {
    Any
})), Nil {
    Any
})), DAE.VARIABLE(), DAE.BIDIR(), DAE.NON_PARALLEL(), DAE.PUBLIC(), DAE.T_REAL(Nil {
    Any
})), nothing, Nil {
    Any
... 
    Any
})), SOME {
  SCode.Comment
} (SCode.COMMENT(nothing, nothing)), Nil {
    Any
}))))
Future work
Future work

• Translating the new high performance frontend
• A Simulation runtime
• OMBackend.jl
• Investigate possible integration with:
  – Modia.jl
  – DifferentialEquations.jl
About the dragon?
Future work

• Dragon
  – LLVM
• Cogwheels
  – Modelica
• Colors?
  – Julia
Questions?

www.liu.se
References


References

References


References

References

https://doi.org/10.1145/3049797.3049806

References

https://github.com/OpenModelica/MetaModelica.jl
https://github.com/JKRT/OMCompiler.jl
https://github.com/OpenModelica/Absyn.jl
...