Modelyze: Embedding Equation-Based DSLs

SYNCHRON’13
Dagstuhl, Germany, November 20, 2013

David Broman
broman@eecs.berkeley.edu
EECS Department
UC Berkeley, USA
and
Linköping University, Sweden

Modelyze contributors
David Broman
Jeremey G. Siek
Hokeun Kim

Agenda

Part I
Modelyze Overview

Part II
Formal Semantics

Part III
Modelyze Demo
Part II

Modelyze Overview

Problem: Expressiveness and Analyzability

Cannot express all modeling or analysis needs.
Limited to what the modeling language can provide.

Language versions:
- \(A, v1.0\) → \(A, v1.1\) → \(A, v2.0\) → \(A, v2.2\)

Standard library versions:
- \(L, v1.0\) → \(L, v1.1\) → \(L, v2.0\) → \(L, v2.2\)

Modelica: A new language definition approximately every second year

Uses
- Simulation
- Optimization
- Code generation for real-time
- Model export
- Grey-box system identification etc.

\(C, v1.0\) gives many dialects and different languages (e.g. Mosilab, Optimica)

\(B, v1.0\) → \(A, v1.1\)

\(A, v1.0\) → \(A, v1.1\) → \(A, v2.0\) → \(A, v2.2\)
What is Modelyze?

Modelyze (MODEL and analyZEE)

Purpose: Research language – addresses the expressiveness and analyzability problem by making the language extensible.

Small, simple, host language for embedding domain-specific languages (DSL) of different models of computation (MoC).

Key aspect: Both the DSL and models in the DSL are defined in Modelyze.

Gradually typed functional language (call-by-value).

Novelty: Typed symbolic expressions.

Formal semantics for a core of the language.

Proven type soundness for the core.

Prototype implementation (interpreter).

Evaluated for series of equation-based DSLs.

Experimental DSLs

Extensible DSLs for physical modeling

Differential-Algebraic Equations (DAE)

Acausal connections (Electrical and Mechanical domain)

ModelyzeDAE → ModelyzeEOO

ModelyzeHC → ModelyzeHEOO

EOO + HC = HEOO

Ongoing work on combining heterogeneous DSLs for CPS

ModelyzeMA

Master algorithm according to formalized FMI interface (see Broman et al. EMSOFT’13)

ModelyzeHEOO

ModelyzeSync

ModelyzeDE
Overview of the Compilation and Simulation Process

Type checking and collapsing the instance hierarch come “for free”. Part of Modelyze (host language)

Run-time semantics described in Modelyze libraries (meta-programming)

Compile-time parts in Modelica

Compile-time part in Modelyze

Run-time changes

Part I
Modelyze Overview

Part II
Formal Semantics

Part III
Modelyze Demo

Related Work

Implementing DSLs
Compiler construction
- JastAdd (Ekman & Hedin, 2007)
- MetaModelica (Pop & Fritzson, 2006)

Preprocessing and template metaprogramming
- C++ Templates (Veldhuizen, 1995)
- Template Haskell (Sheard & Peyton Jones, 2002)
- Stratego/XP (Bravenboer et al., 2008)

Embedded DSLs
- Haskell DSELs, e.g., Fran (Elliot & Hudak, 1997), Lava (Bjesse et al. 1998), and Paradise (Augustsson, 2008)
- FHM (Nilsson et al., 2003)
- ForSyDe (Sander & Jantsch, 2004)
- Pure embedding (Higher-order functions, polymorphism, lazy evaluation, type classes) (Hudak, 1998)

Combining Dynamic and Static Typing
- Gradual Typing (Siek & Taha, 2007)
- Soft Typing (Cartwright & Fagan, 1991)
- Dynamic type with typecase (Abadi et al., 1991)
- Typed Scheme, Racket (Tobin-Hochstadt, Felleisen, 2008)
- Thorn, like types (Wrigstad et al., 2010)

Representing Code and Data type
- Dynamic languages LISP, Mathematica
- MetaML <T> (Taha & Sheard, 2000)
- GADT (Peyton Jones et al., 2006; Xi et al., 2003; Cheney & Ralf, 2003)
- Open Data types (Löeh & Hinze, 2006)
- Pattern Calculus (Jay, 2009)
- Syntactic library (Axelsson, 2012)
Part I
Modelyze Overview

Part II
Formal Semantics

Part III
Modelyze Demo

Declarative Mathematical Model

Using function abstraction to define the model

Unknowns are given types but not bound to values

Equations and initial values are defined declaratively, just as the mathematical equations

---

Pendulum Example

\[
\begin{align*}
-T \cdot \frac{x}{l} &= mx'' \\
-T \cdot \frac{y}{l} - mg &= my'' \\
x^2 + y^2 &= l^2
\end{align*}
\]

---

\[
\begin{align*}
\text{def} & \quad \text{Pendulum}(m:\text{Real}, l:\text{Real}, \text{angle}:\text{Real}) = \\
& \quad \text{def} x,y,T:\text{Real}; \\
& \quad \text{init} x (l*sin(\text{angle})); \\
& \quad \text{init} y (-l*cos(\text{angle})); \\
& \quad -T*x/l = m*x''; \\
& \quad -T*y/l - m*g = m*y''; \\
& \quad x^2. + y^2. = l^2.;
\end{align*}
\]

---

---
Part I
Modelyze Overview

Part II
Formal Semantics

Part III
Modelyze Demo

Declarative Mathematical Model

Which parts are part of the host language (Modelyze)?

Unknowns are internally represented as typed symbols

Fresh (unique) symbol

Tagged with a type

Symbolic type

Release the user from annotation burden

Symbols cannot be bound to values, so \( x^2 \) would crash at runtime

Use quasi-quoting to mix symbolic expressions and program code?

Using MetaML syntax < > for quotation and ~ for anti-quoting (escape)

\[
(\text{fun } t \rightarrow <t>l^2. ) >
\]

Heavy annotation burden for the end-user
Symbol Lifting Analysis (SLA)

Symbol Lifting Analysis (SLA): During type checking, lift expressions that cannot be safely evaluated at runtime into symbolic expressions (data).

\[ \Gamma \vdash_L e \leadsto e': \tau \]

Rewritten to prefix curried form

\[((/ \times) \ 1)\]

where

\( (/) : \text{Real} \rightarrow \text{Real} \rightarrow \text{Real} \)

\( x : \text{Real} \)

\( l : \text{Real} \)

\[((\text{lift} (/) : \text{Real} \rightarrow \text{Real} \rightarrow \text{Real}) \ @ \ x) \ @ (\text{lift} \ l : \text{Real}))\]

Division cannot be performed, lift expression to type \( \langle \text{Real} \rightarrow \text{Real} \rightarrow \text{Real} \rangle \).

Term \( \text{lift} \ e : \tau \) wraps \( e \) and results in type \( \langle \tau \rangle \)

---

Part I
Modelyze
Overview

Part II
Formal
Semantics

Part III
Modelyze
Demo

Pattern Matching on Symbolic Expressions

Dynamic symbolic type \( <?> \)

Accumulator Sets of symbolic type \( <\text{Real}> \)

Query for all unknowns in a model instance

\[ \text{def getUnknowns}(\text{exp} : <?>, \text{acc} : (\text{Set} <\text{Real}>) ) \rightarrow (\text{Set} <\text{Real}>) = \{ \]

match \( \text{exp} \) with

| \( e1 \  e2 \rightarrow \text{getUnknowns}(e2, \text{getUnknowns}(e1, \text{acc})) \)
| \( \text{sym} : \text{Real} \rightarrow \text{Set}.\text{add} \ \text{exp} \ \text{acc} \)
| _ \rightarrow \text{acc} \}

Uniform data structure, no boilerplate code (matching on symbolic applications)

Match all symbols of type \( <\text{Real}> \)
i.e., unknowns in the model.

\( \text{getUnknowns}((\text{Pendulum}(5,3,45*\pi/180), \text{Set}.\text{empty})) \)

Returns a set with 3 symbols (representing \( x, y, \) and \( T \)).
Static Error Checking at the DSL Level

Syntactically correct model (host syntax)

```
def ModifiedPendulum(m:Real,l:Real,angle:Real) = {
def x,y,T:Real;
init x (l*sin(angle));
init y;  //Error: Missing initial value
-T*x/l = m*x'';
-T*y/l - m*g = m*y'';
x^2. + y^2. = l^2.;
}
```

Static type error instead of dynamic error during translation/pattern matching.

```
def ModifiedPendulum(m:Real,l:Real,angle:Real) = {
def x,y,T:Real;
init x (l*sin(angle));
init y;  //Error: Missing initial value
-T*x/l = m*x'';
-T*y/l - m*g = m*y'';
x^2. + y^2. = l^2.;
}
```

Quite intuitive error messages at the DSL level.

```
modifiedpendulum.moz 4:10-4:10 error: Missing argument of type 'Real'.
```

Mechatronic Control Example (ModelyzeEOO)

Control Components

Electrical Components

Mechanical Components

Part I
Modelyze Overview

Part II
Formal Semantics

Part III
Modelyze Demo
Nodes are represented as symbols. “Wiring” components together.

```python
def CPS() = {
    def s1, s2, s3, s4:Signal;
    def r1, r2, r3, r4:Rotational;
    ConstantSource(1.0, s1);
    Feedback(s1, s4, s2);
    PID(3.0, 0.7, 0.1, 10.0, s2, s3);
    DCMotor(s3, r1);
    IdealGear(4.0, r1, r2);
    serialize(5.0, r2, r3, ShaftElement);
    Inertia(0.3, r3, r4);
    SpeedSensor(r4, s4);
}
```

Higher-order model (higher-order function)

Hierarchies of model components.

```python
def DCMotor(V:Voltage, flange:Rotational) = {
    def e1, e2, e3, e4:Electrical;
    SignalVoltage(V, e1, e4);
    Resistor(200.0, e1, e2);
    Inductor(0.1, e2, e3);
    EMF(1.0, e3, e4, flange);
    Ground(e4);
}
```

Unkowns and behaviour equation
Part II
Formal Semantics

\[
\Gamma \vdash_L e_1 \sim e_1^\prime : \langle \tau_{11} \rightarrow \tau_{12} \rangle \\
\Gamma \vdash_L e_2 \sim e_2^\prime : \tau_2 \\
[\tau_2^\prime : \tau_2] = [e_2^\prime] \\
\langle \tau_{11} \rangle \sim [\tau_2] \\
\Gamma \vdash_L e_1 e_2 \sim e_1^\prime e_2^\prime : \langle \tau_{12} \rangle \quad \text{(L-APP)}
\]

Intermediate Languages

To enable formalization and proving type soundness, we define three intermediate languages.

- Both are gradually typed (Mixing static and dynamic typing).

- Translation step from surface language to core language. Includes: parsing, syntactic transformation, pattern compilation etc.

- Translating from the core to Lifted language. Includes: symbolic lifting analysis (SLA).

- Dynamic aspects made explicit by cast insertion. Also vital for proving type safety.
Proposition 3 (Symbolic Lifting Preserves Types). If $\Gamma \vdash_L e \leadsto e' : \tau$ then $e'$ is well typed in $\Gamma$ at type $\tau$.

Proposition 4 (Cast Insertion Preserves Types). If $\Gamma \vdash_C e \leadsto e' : \tau$ then $\Gamma \vdash e' : \tau$.

Lemma 3 (Progress). If $\vdash e : \tau$ then $e \in \text{Values}$, or for all $S$ there exists $S'$ and $e'$ such that $e \mid S \rightarrow e' \mid S'$, or $e = \text{error}$.

Lemma 7 (Preservation). If $\Gamma \vdash e : \tau$ and $e \mid S \rightarrow e' \mid S'$ then $\Gamma \vdash e' : \tau$. 
Part III
Modelyze Demo

Conclusions

Equation-based extensible modeling using an embedded DSL approach.

Symbol lifting analysis (SLA) used to release the annotation burden from the end user

Gradually typed symbols used for pattern matching and DSL-level error reporting

Thanks for listening!

See journal preprint:

Open source implementation: http://www.eecs.berkeley.edu/~broman/