

# Modeling Kernel Language (MKL)

A formal and extensible approach to  
equation-based modeling languages

Guest Talk, TU Berlin, Germany

March 3, 2011

**David Broman**

Department of Computer and Information Science  
Linköping University, Sweden  
david.broman@ida.liu.se



Linköpings universitet

2

## Agenda

David Broman  
david.broman@liu.se

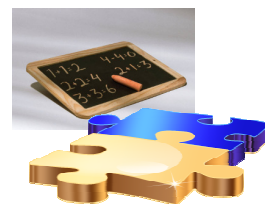
### Part I What is an EOO Language?

$$\begin{aligned}J_1\dot{\omega}_1 &= M_v - M_1 \\ J_2\dot{\omega}_2 &= M_h - M_2 \\ \omega_1 &= -r\omega_2 \\ M_1 &= -r^{-1}M_2\end{aligned}$$

### Part II Why MKL?



### Part III Expressiveness, Extensibility, and Formalization



**Part I**  
What is an EOO  
language?

**Part II**  
Why MKL?

**Part III**  
Expressiveness, Extensibility  
and Formalization



Linköpings universitet

## Part I

### What is an EOO language?

$$J_1 \dot{\omega}_1 = M_v - M_1$$

$$J_2 \dot{\omega}_2 = M_h - M_2$$

$$\omega_1 = -r\omega_2$$

$$M_1 = -r^{-1}M_2$$



**Part I**  
What is an EOO  
language?

**Part II**  
Why MKL?

**Part III**  
Expressiveness, Extensibility  
and Formalization



## What is Modeling and Simulation?

experiment on...

Simulation



$$\begin{aligned} J_1 \dot{\omega}_1 &= M_v - M_1 \\ J_2 \dot{\omega}_2 &= M_h - M_2 \\ \omega_1 &= -r\omega_2 \\ M_1 &= -r^{-1}M_2 \end{aligned}$$

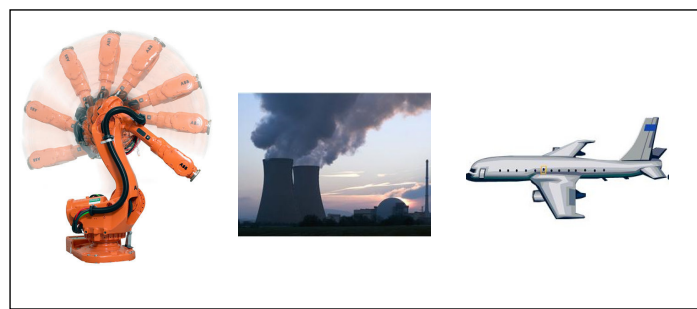
**Mathematical Model**  
Differential-Algebraic  
Equations (DAEs)

Model



Modeling

answer questions  
about...



System



**Part I**  
What is an EOO  
language?

**Part II**  
Why MKL?

**Part III**  
Expressiveness, Extensibility  
and Formalization



## Domain-Specific Language (DSL)

- **Primarily domain:** Modeling of physical systems
- **Multiple physical domains:** e.g., mechanical, electrical, hydraulic



## Models and Objects

- **Object in e.g., Java, C++:** object = data + methods
- **Objects in EEO languages:** object = data + equations



**Part I**  
What is an EEO language?

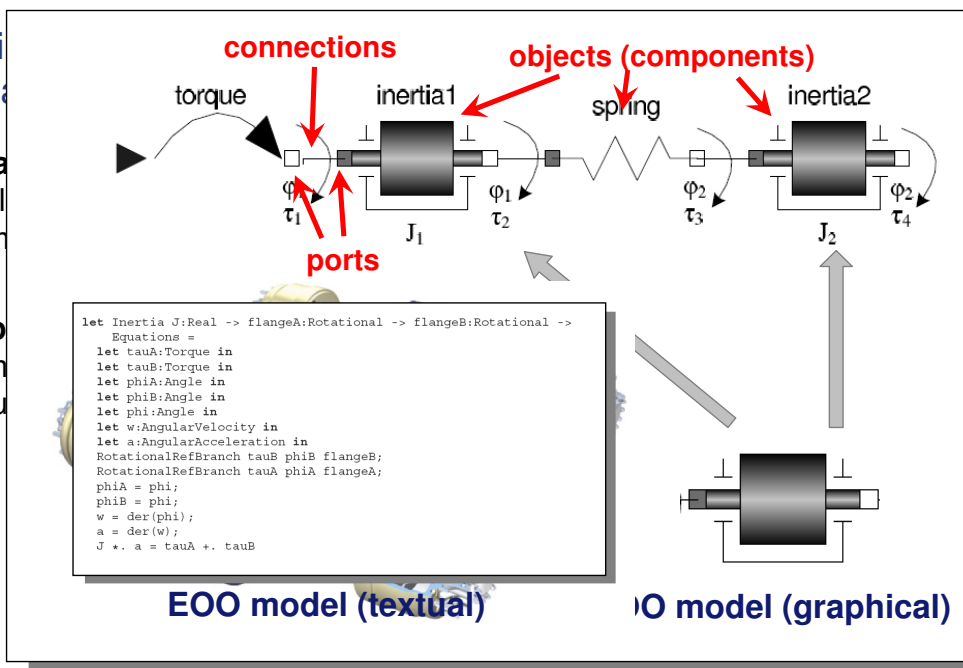
**Part II**  
Why MKL?

**Part III**  
Expressiveness, Extensibility and Formalization



## Domain-Specific Language (DSL)

- **Primarily domain:** Modeling of physical systems
- **Multiple physical domains:** e.g., mechanical, electrical, hydraulic



## Models and Objects

- **Object in e.g., Java, C++:** object = data + methods
- **Objects in EEO languages:** object = data + equations



**Part I**  
What is an EEO language?

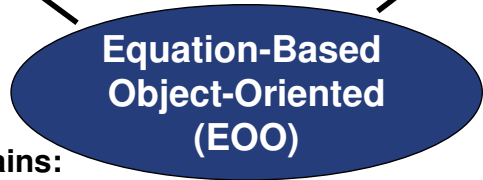
**Part II**  
Why MKL?

**Part III**  
Expressiveness, Extensibility and Formalization



Domain-Specific Language (DSL)

- **Primarily domain:** Modeling of physical systems
- **Multiple physical domains:** e.g., mechanical, electrical, hydraulic



Models and Objects

- **Object in e.g., Java, C++:** object = data + methods
- **Objects in EOO languages:** object = data + equations

Acausality

- **At the equation-level**  
 $u = R * i$
- **At the object connection level**



**Part I**  
What is an EOO language?

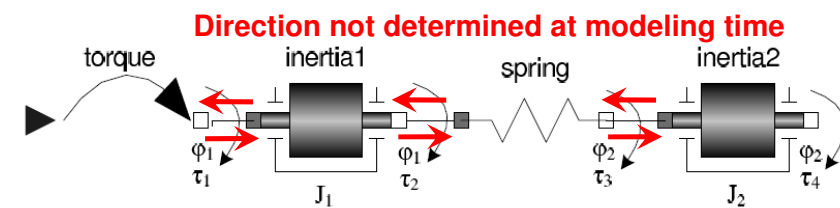
**Part II**  
Why MKL?

**Part III**  
Expressiveness, Extensibility and Formalization

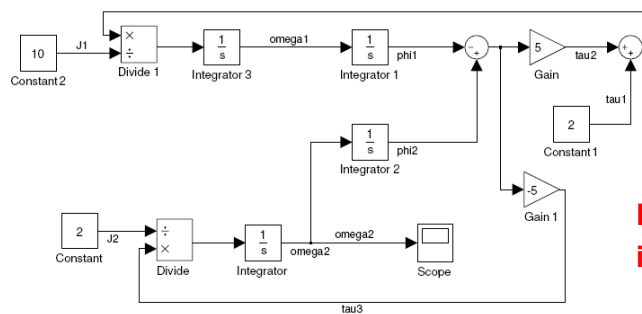


Domain-Specific Language (DSL)

- **Primarily domain:** Modeling of physical systems
- **Multiple physical domains:** e.g., mechanical, electrical, hydraulic



- Variables**
- Potential
  - Flow



**Part I**  
What is an EOO language?

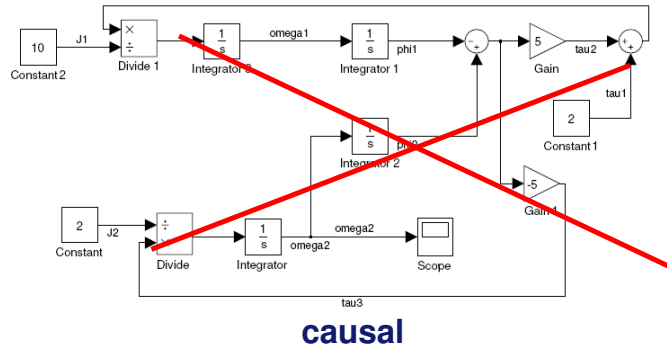
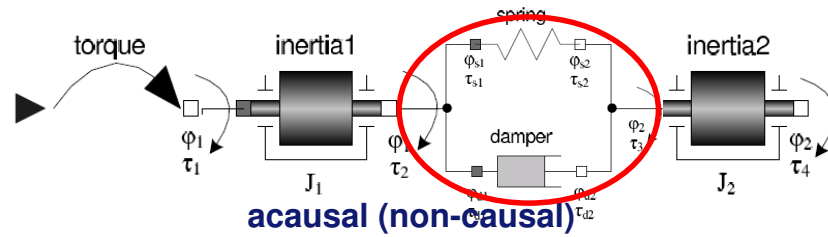
**Part II**  
Why MKL?

**Part III**  
Expressiveness, Extensibility and Formalization



Domain-Specific Language (DSL)

- Primarily domain: Modeling of physical systems
- Multiple physical domains: e.g., mechanical, electrical, hydraulic



Objects

Java, C++:  
Methods

Equation-based languages:  
Equations

Level



**Part I**  
What is an EOO language?

**Part II**  
Why MKL?

**Part III**  
Expressiveness, Extensibility and Formalization



Linköpings universitet

Domain-Specific Language (DSL)

- Primarily domain: Modeling of physical systems
- Multiple physical domains: e.g., mechanical, electrical, hydraulic

- Modelica
- VHDL-AMS
- gPROMS

Equation-Based Object-Oriented (EOO)

Models and Objects

- Object in e.g., Java, C++:  
object = data + methods
- Objects in EOO languages:  
object = data + equations

Acausality

- At the equation-level  
 $u = R * i$
- At the object connection level



**Part I**  
What is an EOO language?

**Part II**  
Why MKL?

**Part III**  
Expressiveness, Extensibility and Formalization



Linköpings universitet

## Part II

### Why MKL?



**Part I**  
What is an EOO  
language?



**Part II**  
Why MKL?

**Part III**  
Expressiveness, Extensibility  
and Formalization



## Expressiveness

### Expressiveness – ease and possibility of expressing complex models or tasks

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**



**Part I**  
What is an EOO  
language?



**Part II**  
Why MKL?

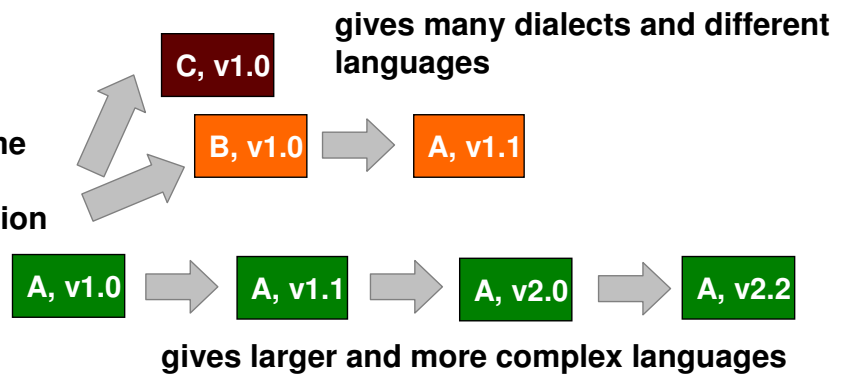
**Part III**  
Expressiveness, Extensibility  
and Formalization



## Extensibility – mechanisms to add new language features

### Uses

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



**Part I**  
What is an EOO  
language?



**Part II**  
Why MKL?

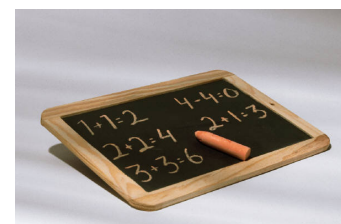
**Part III**  
Expressiveness, Extensibility  
and Formalization



## Formalization – precise semantics “meaning” of the language

### Language Specifications of state-of-the-art are informally defined

- hard to interpret unambiguously when developing compilers
- hard to reason about when extending the language
- hard to formalize e.g. Modelica due to size and complexity



**Part I**  
What is an EOO  
language?



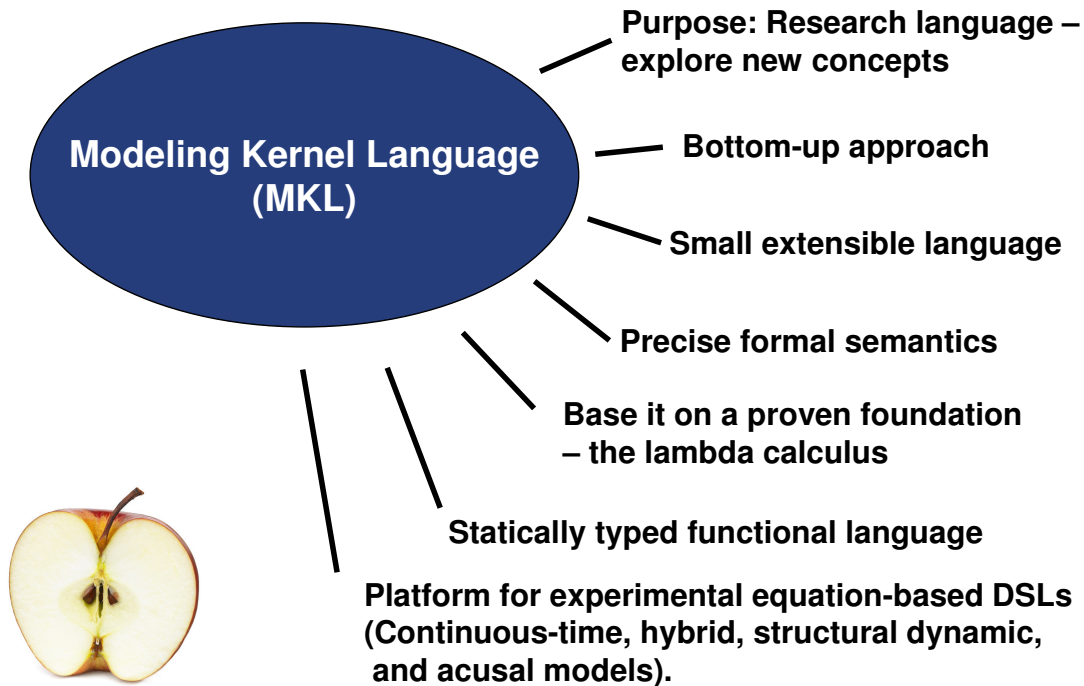
**Part II**  
Why MKL?

**Part III**  
Expressiveness, Extensibility  
and Formalization



# What is MKL?

David Broman  
david.broman@liu.se



**Part I**  
What is an EOO language?



**Part II**  
Why MKL?

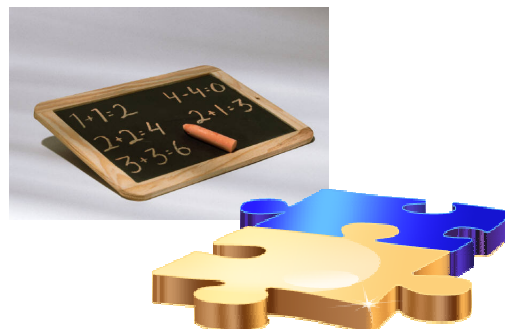
**Part III**  
Expressiveness, Extensibility and Formalization



David Broman  
david.broman@liu.se

## Part III

### Expressiveness, Extensibility, and Formalization



**Part I**  
What is an EOO language?

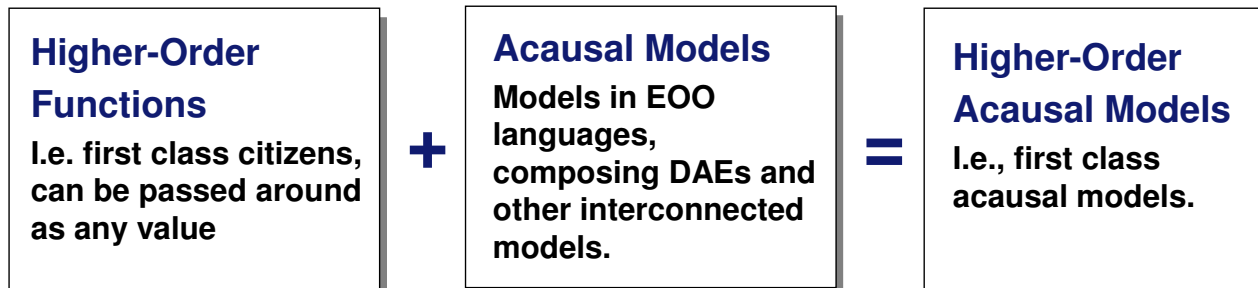
**Part II**  
Why MKL?



**Part III**  
Expressiveness, Extensibility and Formalization




## Higher-Order Acausal Models (HOAM)



**Part I**  
What is an EOO language?

**Part II**  
Why MKL?

 **Part III**  
Expressiveness, Extensibility and Formalization



**DEFINITION 3** (Higher-Order Acausal Model (HOAM)).  
*A higher-order acausal model is an acausal model, which can be*

1. *parametrized with other HOAMs.*
2. *recursively composed to generate new HOAMs.*
3. *passed as argument to, or returned as result from functions.*

**Replaces several of Modelica's constructs with one concept, e.g.,**

- **Conditional components**
- **For-equations**
- **Redeclare construct**

**Part I**  
What is an EOO language?

**Part II**  
Why MKL?

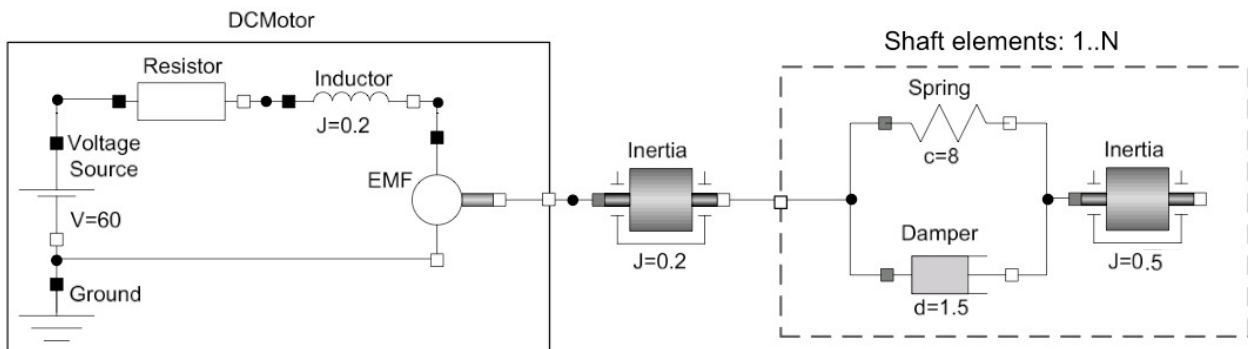
 **Part III**  
Expressiveness, Extensibility and Formalization



# HOAM – Example

David Broman  
david.broman@liu.se

## Example of a mechatronic system with a DC motor and a flexible shaft



```
let MechSys =
  let r1:Rotational in
  let r2:Rotational in
  let r3:Rotational in
  DCMotor r1;
  Inertia 0.2 r1 r2;
  FlexibleShaft 120 r2 r3
```

**Creates a flexible shaft with 120 shaft elements.**

How is this model defined?

**Part I**  
What is an EOO language?

**Part II**  
Why MKL?

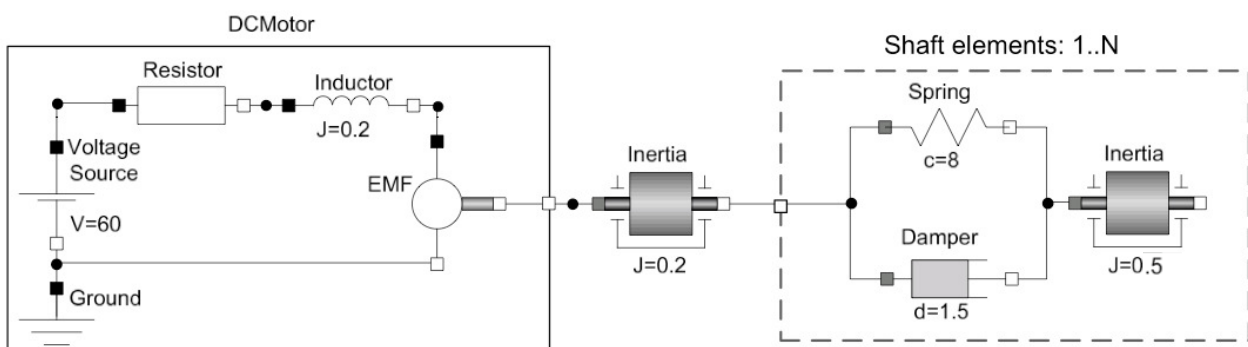
**Part III**  
Expressiveness, Extensibility and Formalization



# HOAM – Example

David Broman  
david.broman@liu.se

## Example of a mechatronic system with a DC motor and a flexible shaft



```
let Inductor L:Real -> p:Electrical -> n:Electrical -> Equations=
  let i:Current in
  let v:Voltage in
  ElectricalBranch i v p n;
  L *. (der i) = v
```

**Part I**  
What is an EOO language?

**Part II**  
Why MKL?

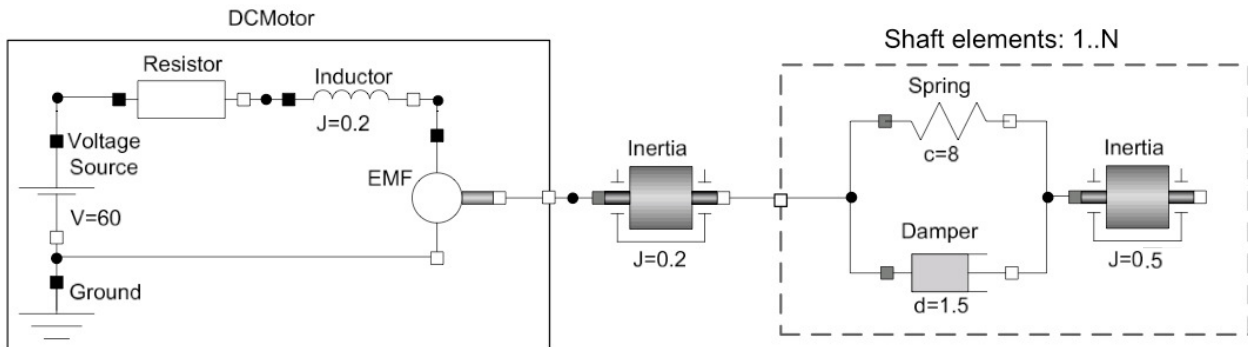
**Part III**  
Expressiveness, Extensibility and Formalization



# HOAM – Example

David Broman  
david.broman@liu.se

## Example of a mechatronic system with a DC motor and a flexible shaft



```
let ShaftElement flangeA:Rotational -> flangeB:Rotational ->
    Equations =
    let r1:Rotational in
    Spring 8. flangeA r1;
    Damper 1.5 flangeA r1;
    Inertia 0.5 r1 flangeB
```

One shaft element is created by standard components.

**Part I**  
What is an EOO  
language?

**Part II**  
Why MKL?

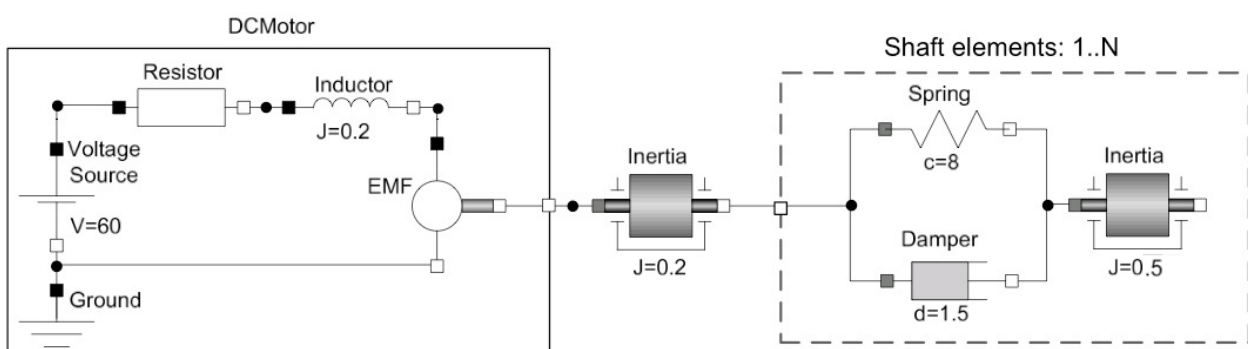
**Part III**  
Expressiveness, Extensibility  
and Formalization



# HOAM – Example

David Broman  
david.broman@liu.se

## Example of a mechatronic system with a DC motor and a flexible shaft



```
let FlexibleShaft n:Int -> flangeA:Rotational ->
    flangeB:Rotational -> Equations =
    if n == 1 then
    ShaftElement flangeA flangeB
    else
    let r1:Rotational in
    ShaftElement flangeA r1;
    FlexibleShaft (n-1) r1 flangeB
```

The flexible shaft is recursively defined by creating ShaftElements.

The recursion terminates after n steps (in the example 120 steps)

**Part I**  
What is an EOO  
language?

**Part II**  
Why MKL?

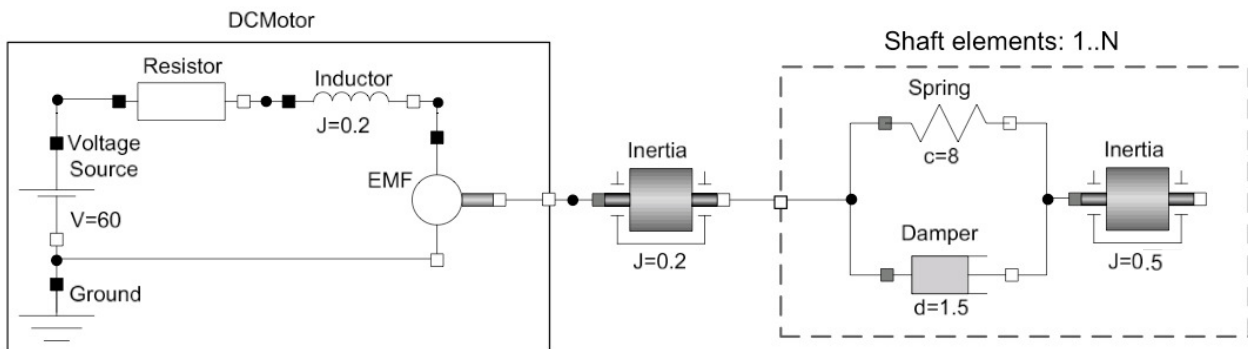
**Part III**  
Expressiveness, Extensibility  
and Formalization



# HOAM – Example

David Broman  
david.broman@liu.se

## Example of a mechatronic system with a DC motor and a flexible shaft



```
let MechSys =
  let r1:Rotational in
  let r2:Rotational in
  let r3:Rotational in
  DCMotor r1;
  Inertia 0.2 r1 r2,
  FlexibleShaft 120 r2 r3
```

Do we always need a special recursive model?

**Part I**  
What is an EOO language?

**Part II**  
Why MKL?

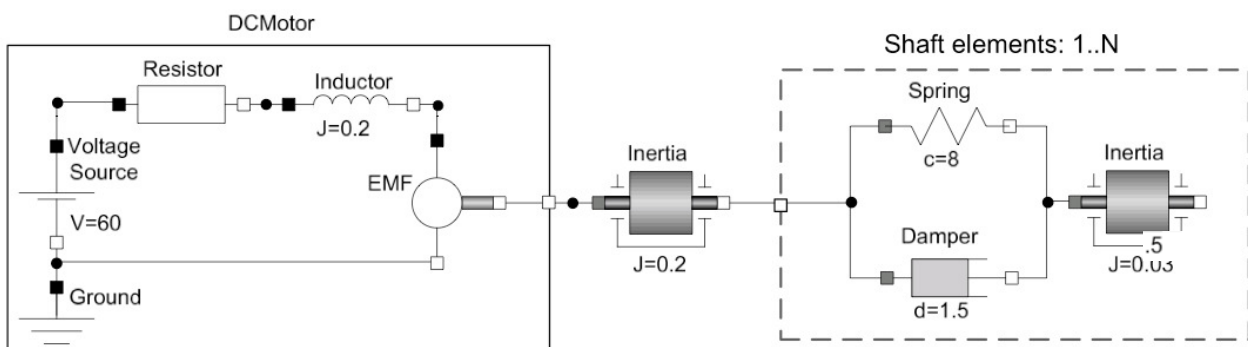
**Part III**  
Expressiveness, Extensibility and Formalization



# HOAM – Example

David Broman  
david.broman@liu.se

## Example of a mechatronic system with a DC motor and a flexible shaft



```
let MechSys =
  let r1:Rotational in
  let r2:Rotational in
  let r3:Rotational in
  DCMotor r1;
  Inertia 0.2 r1 r2;
  (serializeRotational 120 ShaftElement) r2 r3
```

Higher-order function that can compose any mechanical component in series

**Part I**  
What is an EOO language?

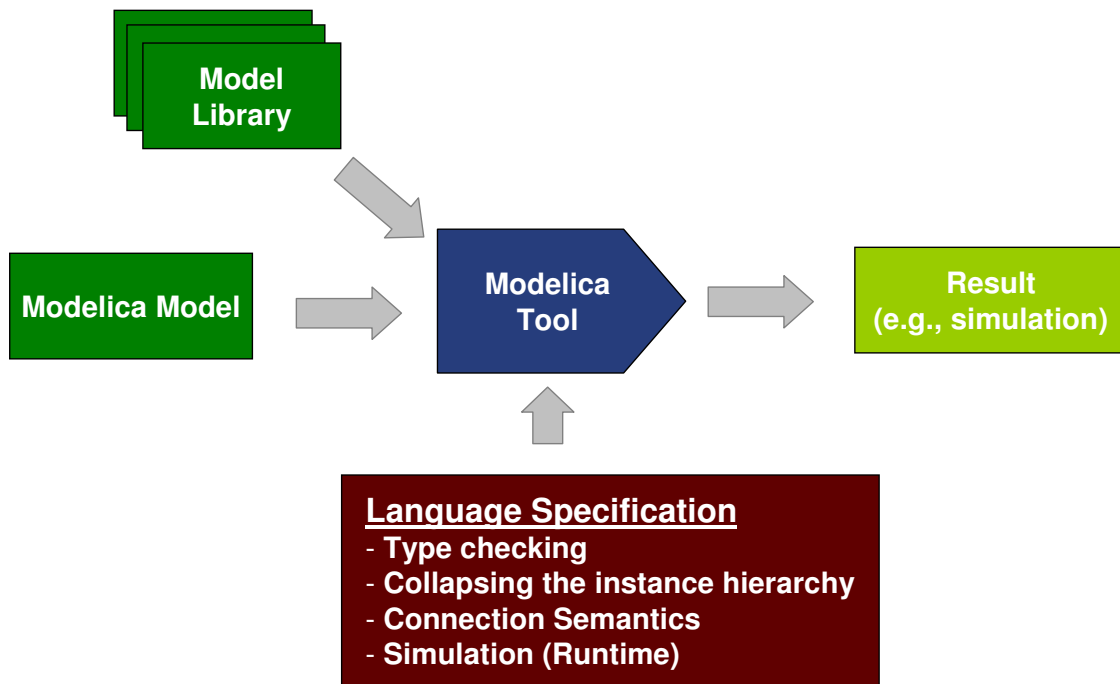
**Part II**  
Why MKL?

**Part III**  
Expressiveness, Extensibility and Formalization



# Modelica Environment

David Broman  
david.broman@liu.se



**Part I**  
What is an EOO language?

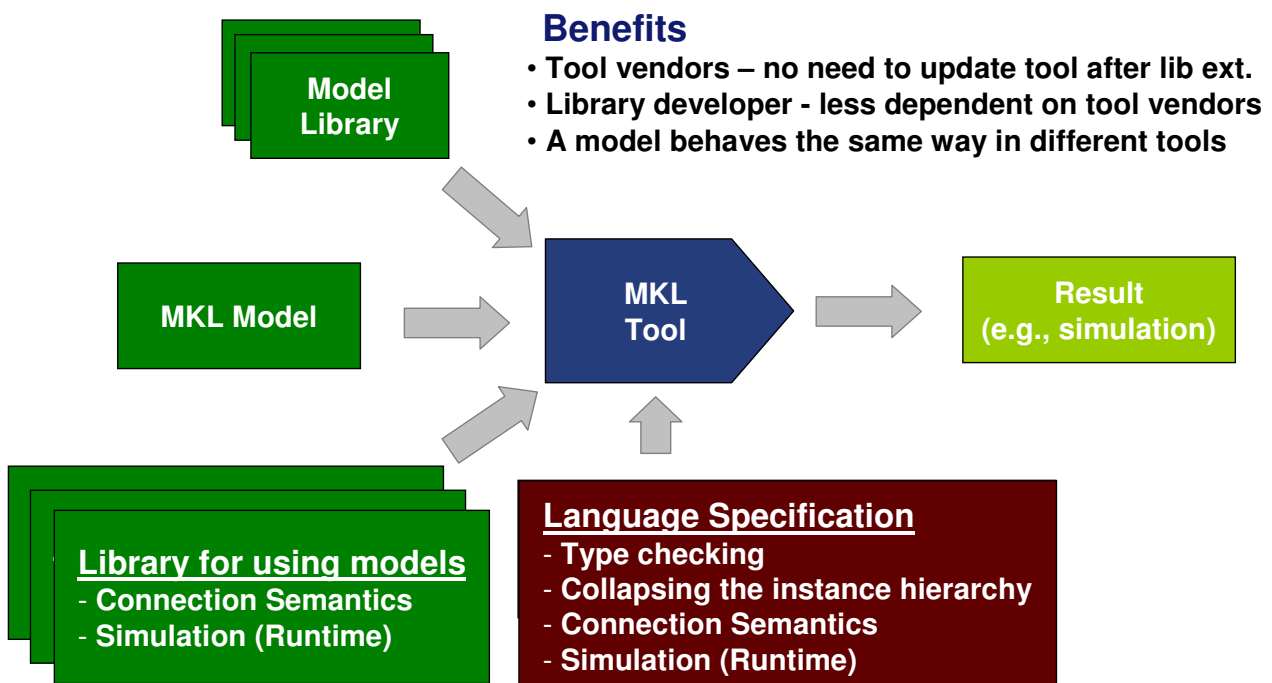
**Part II**  
Why MKL?

**Part III**  
Expressiveness, Extensibility and Formalization



# MKL Environment

David Broman  
david.broman@liu.se



## Benefits

- Tool vendors – no need to update tool after lib ext.
- Library developer - less dependent on tool vendors
- A model behaves the same way in different tools

**Part I**  
What is an EOO language?

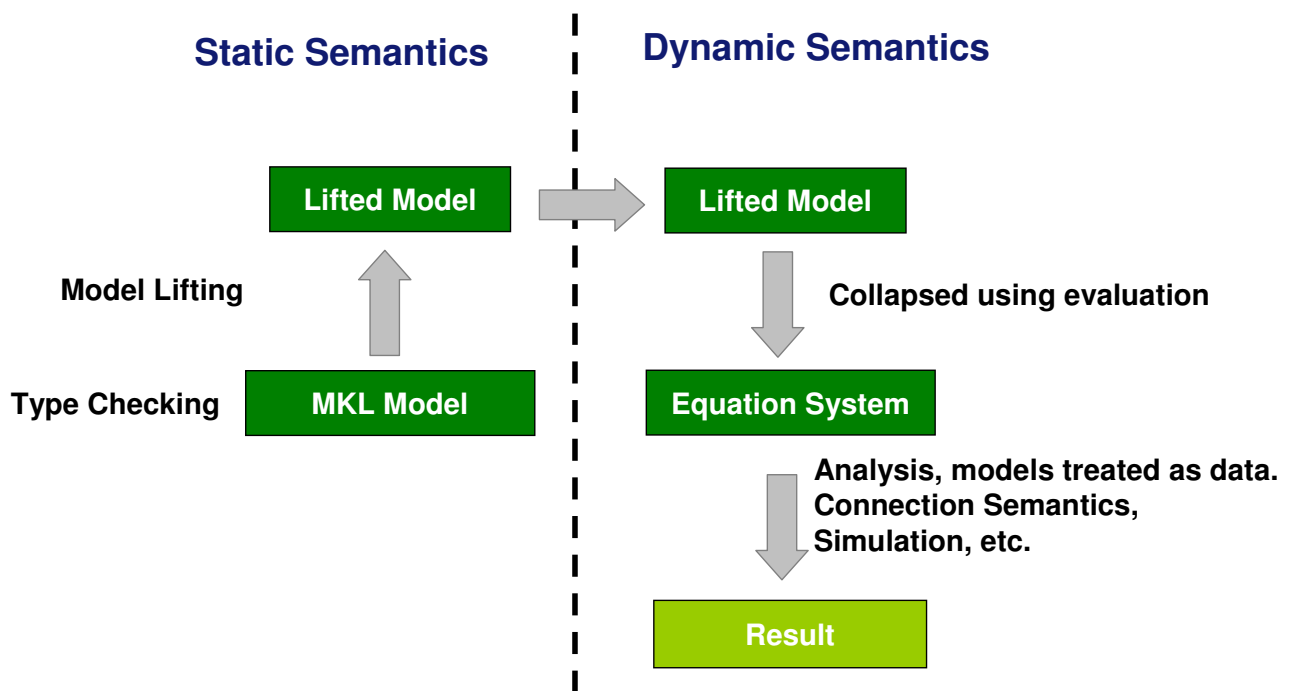
**Part II**  
Why MKL?

**Part III**  
Expressiveness, Extensibility and Formalization



# Intensional Analysis and Model Lifting

David Broman  
david.broman@liu.se



**Part I**  
What is an EOO  
language?

**Part II**  
Why MKL?

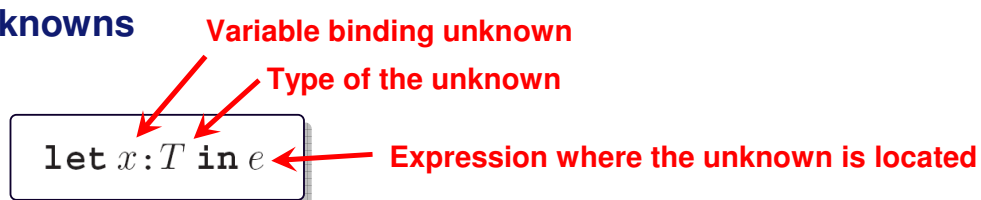
**Part III**  
Expressiveness, Extensibility  
and Formalization



# Unknowns and Model Types

David Broman  
david.broman@liu.se

## Unknowns



## Types

$T ::= \text{Int} \mid \text{Real} \mid \text{Bool} \mid \text{String} \mid T \rightarrow T \mid () \mid$   
 $[T] \mid (T_1, \dots, T_n) \mid \text{Set } T \mid T \Rightarrow T \mid \{T\} \mid$   
 $\langle T \rangle \mid \langle \rangle$

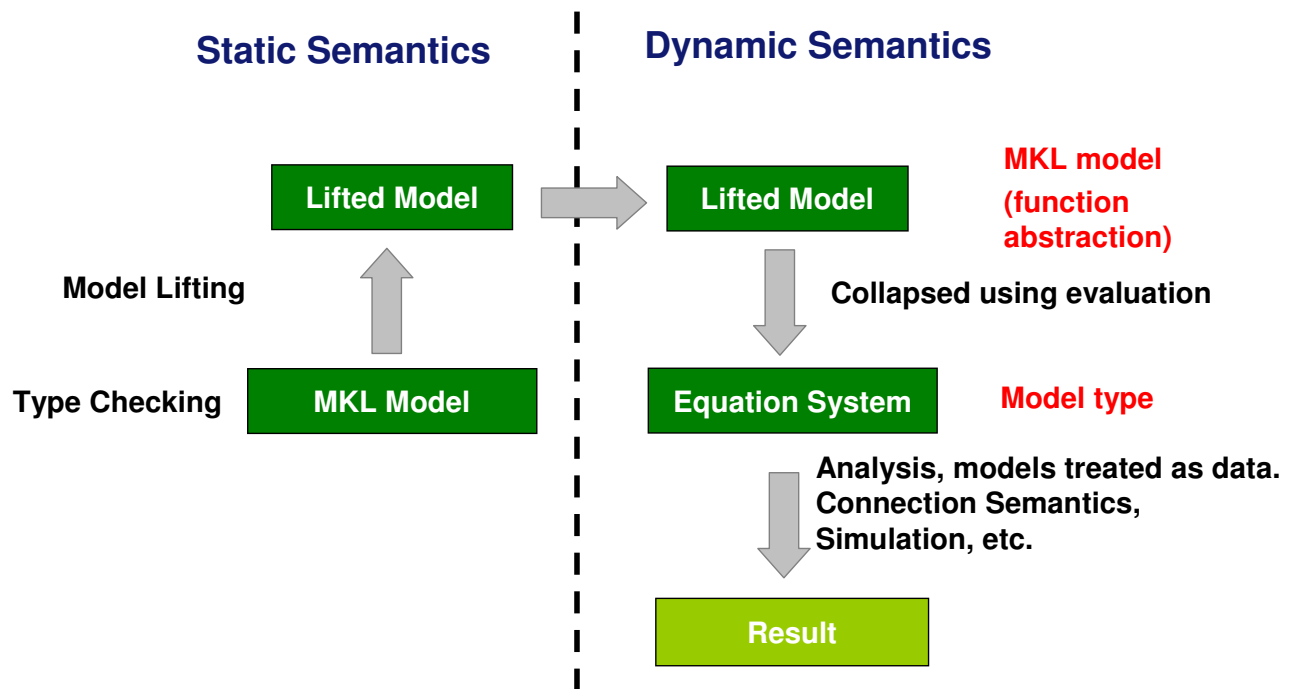
- Any model type** points to  $\langle T \rangle$ .
- Specific model type** points to  $\langle \rangle$ .

**Part I**  
What is an EOO  
language?

**Part II**  
Why MKL?

**Part III**  
Expressiveness, Extensibility  
and Formalization





**Part I**  
What is an EOO language?

**Part II**  
Why MKL?

**Part III**  
Expressiveness, Extensibility and Formalization



## Model Lifting – an Example

### Before model lifting

```
let x:<Real> in
  x +. 1.
```

Unknown, with specific model type Real

What is the type of the expression?

### Definition of infix operators

```
let (+.) : Real -> Real -> Real = @@real_add
```

Curried form

Built-in operation

### After model lifting

```
let x:<Real> in
  (val (+.)) x (val 1.)
```

Model value expression

If e has type T  
then val e has type <T>

The type of the expression is <Real>

Model application, treated as data

**Part I**  
What is an EOO language?

**Part II**  
Why MKL?

**Part III**  
Expressiveness, Extensibility and Formalization



# Model with Initial Values

David Broman  
david.broman@liu.se

```
let LotkaVolterra =
  let growthRateRabbits = 0.04 in
  let deathRateRabbits = 0.0005 in
  let deathRateFoxes = 0.09 in
  let efficiencyGrowthFoxes = 0.1 in
  let rabbits:Population in
  let foxes:Population in
  Init rabbits 700.;
  Init foxes 10.;
  der(rabbits) = growthRateRabbits *. rabbits -.
    deathRateRabbits *. rabbits *. foxes;
  der(foxes) = efficiencyGrowthFoxes *. deathRateRabbits *.
    rabbits *. foxes -. deathRateFoxes *. foxes
```

Type alias

Derivatives defined as unknowns with specific model function type

```
let der : <Real -> Real>
```

Initial values (example of adding a language construct)

```
let Init : <Real -> Real -> Eqs>
```

**Part I**  
What is an EOO  
language?

**Part II**  
Why MKL?

**Part III**  
Expressiveness, Extensibility  
and Formalization



# Pattern Matching and Analyzing Systems of Equations

David Broman  
david.broman@liu.se

Pattern →  $p ::= \text{uk} : T \mid p_1 p_2 \mid \text{val } x : T \mid x$  ← Pattern variable

Unknown →  $\text{uk} : T$

Model application →  $p_1 p_2$

Model value →  $\text{val } x : T$

```
type InitValMap = (<Real> => Real)
```

```
let initValues eqs:Equations -> InitValMap =
  let get eqs:Equations -> acc:InitValMap -> InitValMap =
    match eqs with
    | e1 ; e2 -> get e2 (get e1 acc)
    | Init x (val v:Real) -> Map.add x v acc
    | _ -> acc
  in get eqs (Map.empty)
```

**Part I**  
What is an EOO  
language?

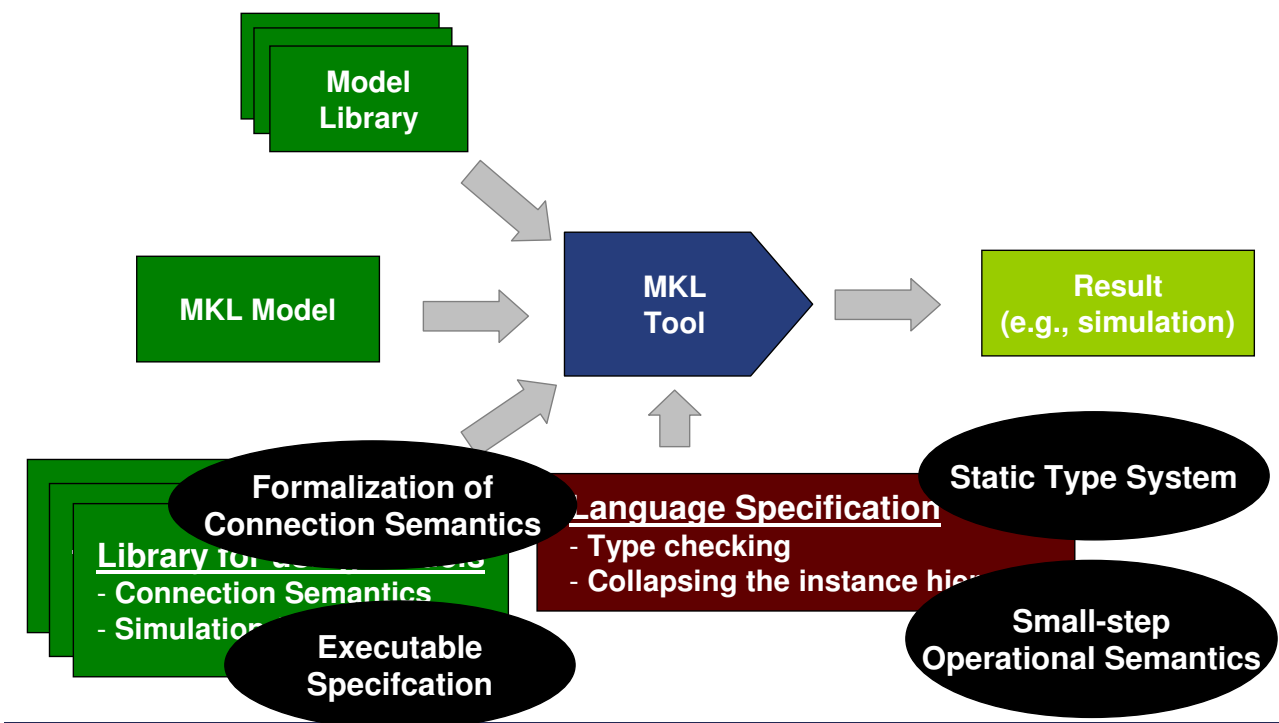
**Part II**  
Why MKL?

**Part III**  
Expressiveness, Extensibility  
and Formalization



# Formalization of Semantics

David Broman  
david.broman@liu.se



**Part I**  
What is an EOO language?

**Part II**  
Why MKL?

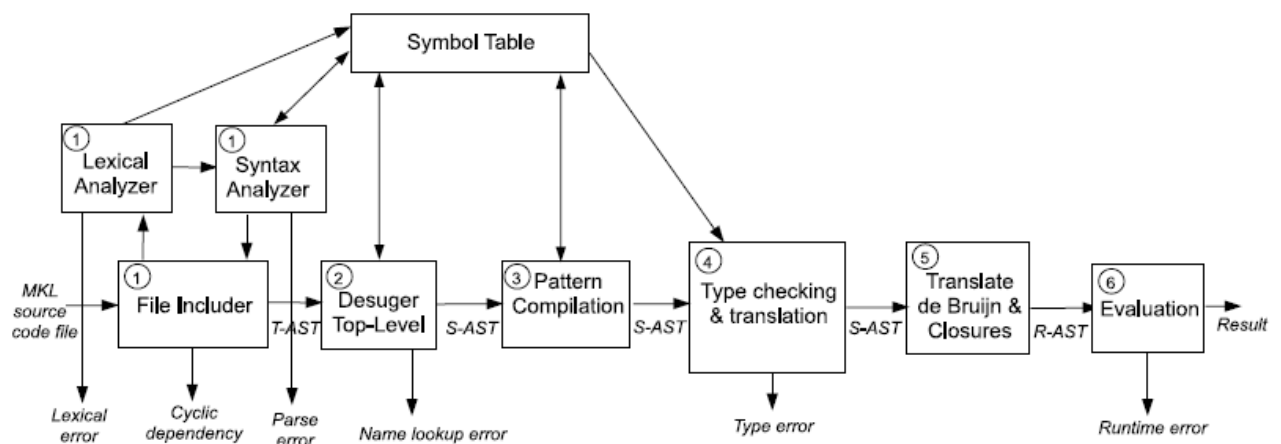
**Part III**  
Expressiveness, Extensibility and Formalization



# How do we verify our solution?

David Broman  
david.broman@liu.se

## Prototype Implementation



**Part I**  
What is an EOO language?

**Part II**  
Why MKL?

**Part III**  
Expressiveness, Extensibility and Formalization



## Abstract Syntax (core of MKL)

Variables	$x, y \in \mathbb{X}$
Unknowns	$u \in \mathbb{U}$
Constants	$c \in \mathbb{C}$
Expressions	$e ::= x \mid \lambda x:\tau.e \mid e e \mid c \mid$ $u:\tau \mid \nu(\tau) \mid e @ e \mid \text{val } e:\tau \mid \text{decon}(e, d, e, e)$
Deconstruct patterns	$d ::= \text{uk}:\tau \mid x @ x \mid \text{val } x:\tau$
Values	$v ::= \lambda x:\tau.e \mid c \mid u:\tau \mid v @ v \mid \text{val } v:\tau$
Ground Types	$\gamma \in \mathbb{G}$
Types	$\tau ::= \gamma \mid \tau \rightarrow \tau \mid \langle \tau \rangle \mid \langle \rangle$

## Big-Step Semantics (selected rule)

$$\frac{e_1 \mid U_1 \Rightarrow \lambda x:\tau.e_3 \mid U_2 \quad e_2 \mid U_2 \Rightarrow v_1 \mid U_3 \quad [x \mapsto v_1]e_3 \mid U_3 \Rightarrow v_2 \mid U_4}{e_1 e_2 \mid U_1 \Rightarrow v_2 \mid U_4} \text{ (BS-APPABS)}$$

**Part I**  
What is an EOO  
language?

**Part II**  
Why MKL?



**Part III**  
Expressiveness, Extensibility  
and Formalization



## Small-Step Semantics and Type System

## Small-Step Semantics (selected rules)

## Computation Rules

$$e \mid U \longrightarrow e' \mid U'$$

$$(\lambda x:\tau_1.e_1)v_1 \mid U \longrightarrow [x \mapsto v_1]e_1 \mid U \text{ (E-APPABS)} \quad c_1 v_1 \mid U \longrightarrow \delta(c_1, v_1) \mid U \text{ (E-DELTA)}$$

$$\frac{u \notin U}{\nu(\tau_1) \mid U \longrightarrow u:\langle \tau_1 \rangle \mid U \cup \{u\}} \text{ (E-NEWUK)}$$

## Congruence Rules

$$e \mid U \longrightarrow e' \mid U'$$

$$\frac{e_1 \mid U \longrightarrow e'_1 \mid U'}{e_1 e_2 \mid U \longrightarrow e'_1 e_2 \mid U'} \text{ (E-APP1)} \quad \frac{e_2 \mid U \longrightarrow e'_2 \mid U'}{v_1 e_2 \mid U \longrightarrow v_1 e'_2 \mid U'} \text{ (E-APP2)}$$

## Type System (selected rule)

$$\Gamma \vdash_L e \rightsquigarrow e' : \tau$$

$$\frac{\Gamma \vdash_L e_1 \rightsquigarrow e'_1 : \tau_{11} \rightarrow \tau_{12} \quad \Gamma \vdash_L e_2 \rightsquigarrow e'_2 : \tau_2 \quad \tau_{11} \sim \tau_2}{\Gamma \vdash_L e_1 e_2 \rightsquigarrow e'_1 e'_2 : \tau_{12}} \text{ (L-APP)}$$

**Part I**  
What is an EOO  
language?

**Part II**  
Why MKL?



**Part III**  
Expressiveness, Extensibility  
and Formalization



## Main Lemmas

### Lemma 10.5 (Progress)

If  $\vdash e:\tau$  then  $e \in \text{Values}$  or for all  $U$  there exists  $U'$  and  $e'$  such that  $e \mid U \longrightarrow e' \mid U'$ .

### Lemma 10.8 (Preservation)

If  $\Gamma \vdash e:\tau$  and  $e \mid U \longrightarrow e' \mid U'$  then  $\Gamma \vdash e':\tau$ .

#### Part I

What is an EOO language?

#### Part II

Why MKL?



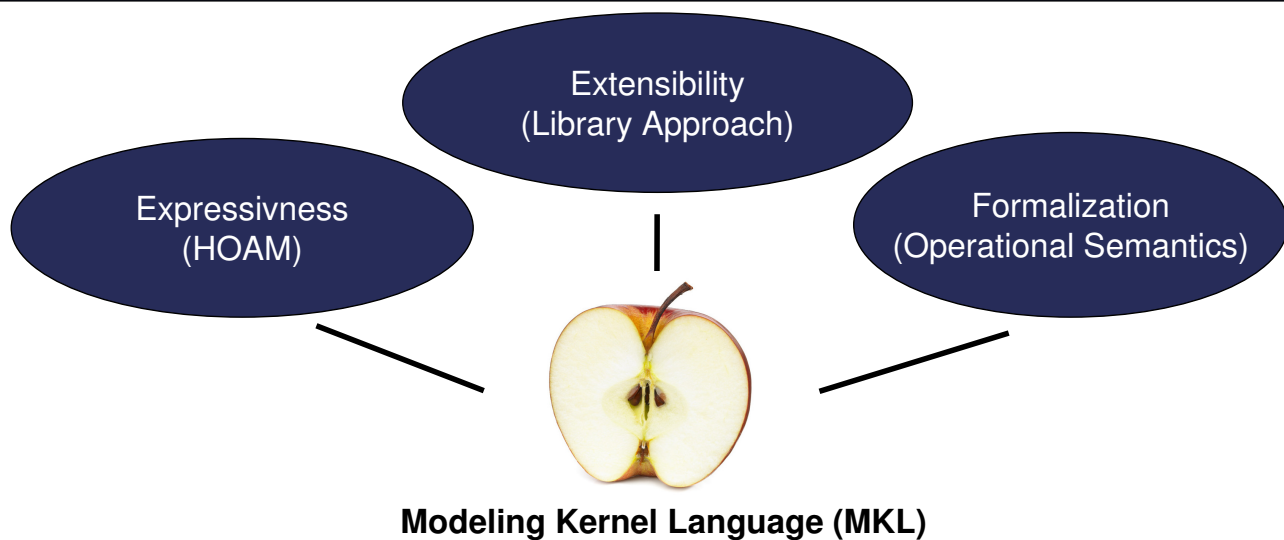
#### Part III

Expressiveness, Extensibility and Formalization



Linköpings universitet

# Conclusions



# Thanks for listening!

#### Part I

What is an EOO language?

#### Part II

Why MKL?

#### Part III

Expressiveness, Extensibility and Formalization



Linköpings universitet