Semantic Type Annotation
ActionWebs Meeting Presentation

Ben Lickly\textsuperscript{1}  Man-Kit Leung\textsuperscript{1}  Thomas Mandl\textsuperscript{2}  Edward A. Lee\textsuperscript{1}
Elizabeth Latronico\textsuperscript{2}  Charles Shelton\textsuperscript{2}  Stavros Tripakis\textsuperscript{1}

\textsuperscript{1}University of California, Berkeley
\textsuperscript{2}Bosch Research, LLC

December 17, 2009
1. Introduction
   - Actor-oriented Models
   - Motivation

2. Example
   - Car-tracking Demo

3. Problem Formulation
   - Theory
   - Design

4. Conclusion
Actor-oriented Models

This model shows a simple adaptive cruise control system, illustrating model-integrated control strategies. A leading car model produces information that is observed with possible flaws by a following car. If the following car detects flaws, it uses a conservative strategy. Otherwise, it tracks the leading car closely.

Simulate a car that attempts to detect faults in communication and adapt its behavior.

Simulate a wireless network that corrupts the data when the fault input is true.

Simulation of the driver of the leading car. Output is the driver's desired speed.

Simulate a car that matches the desired speed using feedback control with a specified time constant.

Simulate faults.

Author: Xiaoljun Liu and Edward A. Lee
What Questions Do We Want to Ask?

- Types
- Units
- Dimensions
- Constant / Non-constant
- Product configuration
- Code ownership
- Code Maturity Level
- ......
What Questions Do We Want to Ask?

- Types
- Units
- Dimensions
- Constant / Non-constant
- Product configuration
- Code ownership
- Code Maturity Level
- .....
Approach

**Ptolemy II**: Our modeling and simulation environment

**Pthomas**: A framework on top of Ptolemy II, allowing analysis of these semantic properties

Types of analysis:
- Inference
- Validity Checking
Calculus Review

\[
\int acceleration(t) \, dt = speed(t)
\]

\[
\int speed(t) \, dt = position(t)
\]
Continuous Director

DimensionSystemSolver
Double click to Resolve Properties

ConstNonconstSolver
Double click to Resolve Properties

PropertyRemover
Double click to Remove Properties

• DimensionSystemSolver::Constraint: Const.output == Acceleration

PropertyLatticeAttribute
Demo

Continuous Director

DimensionSystemSolver
  Double click to Resolve Properties

ConstNonconstSolver
  Double click to Resolve Properties

PropertyRemover
  Double click to Remove Properties

Integrator
  Acceleration → Speed
  Const

Display
  Speed

*DimensionSystemSolver::Constraint: Const.output == Acceleration*
DimensionSystemSolver
- Double click to Resolve Properties

ConstNonconstSolver
- Double click to Resolve Properties

PropertyRemover
- Double click to Remove Properties

Continuous Director

PropertyLatticeAttribute

AddSubtract

Integrator

Const

Display

- DimensionSystemSolver::Constraint: Const.output == Acceleration
Demo

Continuous Director

DimensionSystemSolver
Double click to Resolve Properties

ConstNonconstSolver
Double click to Resolve Properties

PropertyRemover
Double click to Remove Properties

AddSubtract

Const
Acceleration

Integrator

Display

PropertyLatticeAttribute

DimensionSystemSolver::Constraint: Const.output == Acceleration
This model shows a simple adaptive cruise control system, illustrating model–integrated control strategies. A leading car model produces information that is observed with possible flaws by a following car. If the following car detects flaws, it uses a conservative strategy. Otherwise, it tracks the leading car closely.

Simulate a wireless network that corrupts the data when the fault input is true.

Simulate a car that attempts to detect faults in communication and adapt its behavior.

Author: Xiaoljun Liu and Edward A. Lee
This model shows a simple adaptive cruise control system, illustrating model–integrated control strategies. A leading car model produces information that is observed with possible flaws by a following car. If the following car detects flaws, it uses a conservative strategy. Otherwise, it tracks the leading car closely.

Simulate a car that matches the desired speed using feedback control with a specified time constant.

Simulate a wireless network that corrupts the data when the fault input is true.

Simulate a car that attempts to detect faults in communication and adapt its behavior.

Author: Xiaoljun Liu and Edward A. Lee
Car simulator. This model takes as input a desired speed and implements a simple proportional controller with the specified time constant to achieve that speed. It outputs the acceleration, speed, and position of the car.

- initialPosition: 10.0
- initialSpeed: 0.0
- timeConstant: 10.0
- DimensionSystemSolver::constraint: timeConstant >= Time
Car simulator. This model takes as input a desired speed and implements a simple proportional controller with the specified time constant to achieve that speed. It outputs the acceleration, speed, and position of the car.
Background

- **Lattice**
  A partially ordered set in which every pair of elements has both a least upper bound and a greatest lower bound.

- **Monotonic Function**
  A function $f$ for which
  \[ x_1 \leq x_2 \implies f(x_1) \leq f(x_2) \]
Problem Statement

Given:

Lattice: $P$ \h(1)

Constants & Variables: $p_1, p_2, \cdots \in P$ \h(2)

Constraints of the form: $f(p_1, p_2, \cdots) \leq p_n$ (\(f\) monotonic) \h(3)

is there a satisfying assignment to variables?
Problem Statement

Given:

Lattice: \( P \) \hspace{1cm} (1)

Constants & Variables: \( p_1, p_2, \cdots \in P \) \hspace{1cm} (2)

Constraints of the form: \( f(p_1, p_2, \cdots) \leq p_n \) \hspace{1cm} (f monotonic) \hspace{1cm} (3)

is there a satisfying assignment to variables?

This problem has a linear time algorithm!
(Rehof and Mogensen, 1996)
Writing Actor Constraints

Problem Formulation

Integrator

\[ x \rightarrow \int \rightarrow y \]

Component | Elements | Constraints
---|---|---
Integrator | input \( x \), output \( y \) | \[ f_I(p_y) \leq p_x \]
\[ f_O(p_x) \leq p_y \]

\[ f_I(p_y) = \begin{cases} 
\text{Undef.} & \text{if } p_y = \text{Undef.} \\
\text{Speed} & \text{if } p_y = \text{Pos.} \\
\text{Accel.} & \text{if } p_y = \text{Speed} \\
\text{Unitless} & \text{if } p_y = \text{Time} \\
\text{Error} & \text{otherwise} 
\end{cases} \]

\[ f_O(p_x) = \begin{cases} 
\text{Undef.} & \text{if } p_x = \text{Undef.} \\
\text{Pos.} & \text{if } p_x = \text{Speed} \\
\text{Speed} & \text{if } p_x = \text{Accel.} \\
\text{Time} & \text{if } p_x = \text{Unitless} \\
\text{Error} & \text{otherwise} 
\end{cases} \]
User interface

Problem: $|C| \propto |M|$

1. **Default Constraints**
   - Set globally by the property solver (actors, connections, etc.)

2. **Actor-specific Constraints**
   - Uses an *adapter pattern* for actors

3. **Instance-specific Constraints**
   - Specified through model annotations
Future work

- Handling of conflicts
  - How can we present useful information in case of a type error? ✓

- Unit system
  - Unit conversion, Default units, Aliasing

- Extension to infinite lattices
Our framework (Pthomas) for analyzing model properties:

1. Requires minimal user specification (lattice and constraints)
2. Infers unspecified properties
3. Catches and reports design errors
4. Scales up efficiently to large models
Relational Constraint Problem (RCP)

\[ RCP : (P, C) \]

\( P \) is a partially ordered set, \( C \) is a set constraints of the form:

\[ r(P_x, P_y, \ldots) \]

where \( r \) is a relation (e.g. \( =, +, \times, \leq \)).

A solution is a satisfying assignment to property variables.
Definite Monotone Function Problem (DMFP)

Special case of RCP

\[ DMFP : (P, C_F) \]

\( P \) is a lattice, \( C_F \) is a set of definite inequalities:

\[ F(P_y, P_z, \ldots) \leq P_x \]

where \( F \) is a monotonic function.

Here, there is a unique least fixed point (LFP) solution.
Related work

- Constraint Satisfiability (Rehof and Mogensen)
  - Linear time algorithm for monotone function problem

- Hindley-Milner Type Theory
  - Sound, incomplete static check of programs before run time.

- Web Ontology Language (OWL), Eclipse Modeling Framework (EMF), Object Constraint Language (OCL)
  - Similarities: Ontology frameworks (concepts and relationships)
  - Differences: Expressiveness vs. Efficiency
Algorithm D (Rehof and Mogensen, 1996)

Pseudocode

\[ C_{\text{var}} \leftarrow \{ \tau \leq A \in C : A \text{ a variable} \} ; \]
\[ C_{\text{const}} \leftarrow \{ \tau \leq A \in C : A \text{ a constant} \} ; \]
\[ p(\beta) = \text{Undef. for all variables } \beta ; \]
\[ \text{while there are unsatisfied constraints in } C_{\text{var}} \text{ do} \]
\[ \quad \text{Let } \tau \leq \beta \text{ be one such constraint} ; \]
\[ \quad \beta \leftarrow \beta \lor \tau ; \]
\[ \text{end} \]
\[ \text{if there are unsatisfied constraints in } C_{\text{const}} \text{ then} \]
\[ \quad \text{Fail: There is no solution} ; \]
\[ \text{end} \]

For a finite lattice, this algorithm takes \( O(\text{height}(L) \times |C|) \).
Conflicts

- What is a conflict?
  - Unsatisfiable constraints

- How is it detected?

\[ C_{const} \leftarrow \{ \tau \leq A \in C : A \text{ a constant} \} ; \]

\[ \ldots \]

\[ \textbf{if} \ \text{there are unsatisfied constraints in } C_{const} \ \textbf{then} \]

\[ \quad \text{Fail: There is no solution} ; \]

\[ \textbf{end} \]