DEVS Flattening with muModelica and pyDEVS

Jesse Doherty

Outline

- Introduction
- Motivations
- Tools
- Solution
- Conclusion

Introduction

- DEVS
  - Atomic

$$\langle S, ta, \delta_{\text{int}}, X, \delta_{ext}, Y, \lambda \rangle$$

  - Coupled

$$\langle X_{self}, Y_{self}, D, \{M_i\}, \{I_i\}, \{Z_{ij}\}, select \rangle$$

Introduction

- DEVS
  - Closed under coupling, through flattening
  - Closure Procedure

$$\langle X_{self}, Y_{self}, D, \{M_i\}, \{I_i\}, \{Z_{ij}\}, select \rangle \longrightarrow \langle S, ta, \delta_{\mathrm{int}}, X, \delta_{ext}, Y, \lambda \rangle$$

Motivation

- Why do we use a coupled DEVS solver?

Motivation

- Why do we use a coupled DEVS solver?
  - Solver can be parallelized
  - Solving the original system seems more satisfying
  - Solving through flattening still requires an atomic solver

Motivation

- Why would we want to flatten?

Motivation

- Why would we want to flatten?
  - Static analysis
  - Optimizations
  - Tools become less complex

How?

- Seems simple
  - take some cross products
  - find some minimums
  - keep track of some time
  - forward some transition functions

Problems

- Questions come up quickly
  - how do we specify DEVS
  - how do we represent them
  - how do we transform them
  - how do we solve them

Tools

- Modelica
- muModelica
- Devs in Modelica
- Python Devs

Modelica

- Object oriented model description language
  - not a programming language
- Highly structured
- Suitable for high-level model description

muModelica

- Modelica compiler originally intended to target octave code
- Written in python
- Extendable
- Provides an AST of input code

DEVS in Modelica

- Set of Modelica classes used to represent DEVS components
  - Events
  - State
  - Port
  - Atomic DEVS
  - Coupled DEVS
- More structured than pydevs representation

DEVS in Modelica

- Functionality added to muModelica to DEVS semantics and output pydevs code
- Some restrictions
  - submodels must be explicitly listed
  - atomic DEVS' states are expected to have a sequential state component (though not enforced)

PyDEVS

- All seen before

Ideal Solution

- For each coupled DEVS
  - produce a new atomic DEVS with
    - state equivalent to a combination of all sub model states
    - transition, output and ta functions are an inlining of component functions
  - discard original AST and produce AST for just the new atomic DEVS

Initial Solution

- Maintain original AST structure
- Create new flattened versions of coupled models
- State of these flattened versions would consist of a list of instances of component models, and an elapsed time for each
- For each function, perform appropriate logic and forward the function to the needed component models

Current Solution

- While producing python code for modelica models, also produce python code for flattened DEVS models

- Benefits:
  - Simpler to implement, direct access to python language
- Drawbacks:
  - loose access to structure of flattened model

Encountered Problems

- muModelica AST can be clumsy when dealing with DEVS structure
- Need a useful way of representing combined states, for use in analysis

Conclusion

- Flattening might be useful
- More specialized tools for dealing with DEVS structure would be needed produce useful analysis