Dynamic structure modelling for Causal Block Diagrams

Master Thesis by Yves Maris

Promotors: Fernando Barros, Hans Vangheluwe

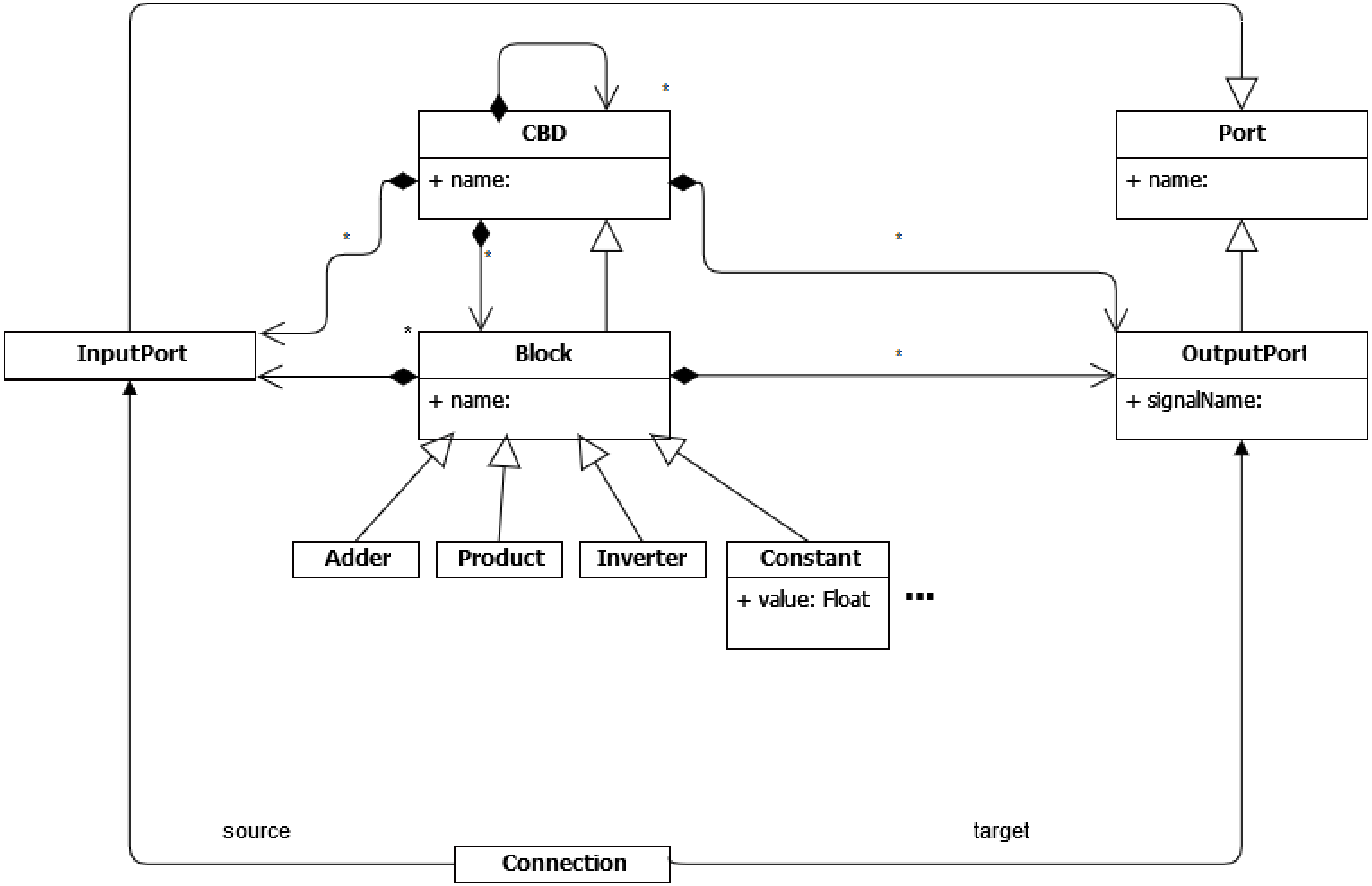
Overview

- Background: Causal Block Diagrams
  - Syntax
  - Semantics
- Problem statement
- Dynamic structure CBD
  - Syntax
  - Semantics
- Implementation
- Case study: elevator model

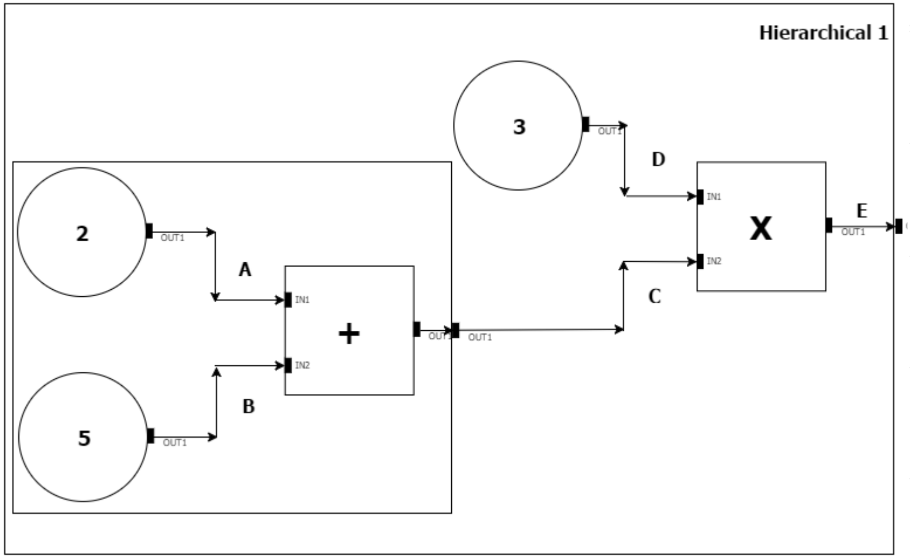
Overview

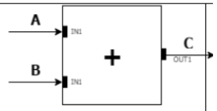
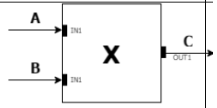
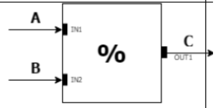
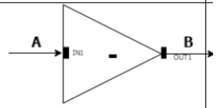
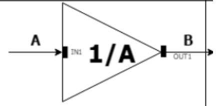
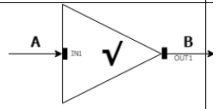
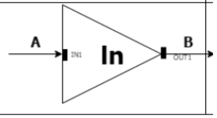
- Background: Causal Block Diagrams (CBD)
  - Syntax
  - Semantics
- Problem statement
- Dynamic structure CBD
  - Syntax
  - Semantics
- Implementation
- Case study: elevator model

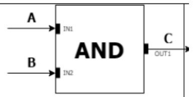
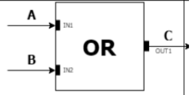
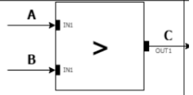
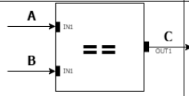
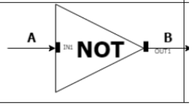
Causal Block Diagrams Abstract Syntax

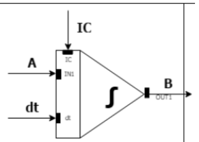
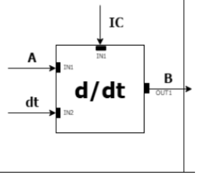


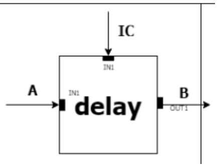
Causal Block Diagrams Visual Syntax



Name	Block
Adder block	
Product block	
Modulo block	
NegatorBlock	
InverterBlock	
SquarerootBlock	
Ln block	

Name	Block
AND block	
OR block	
Greater than block	
Equals block	
NOT Block	

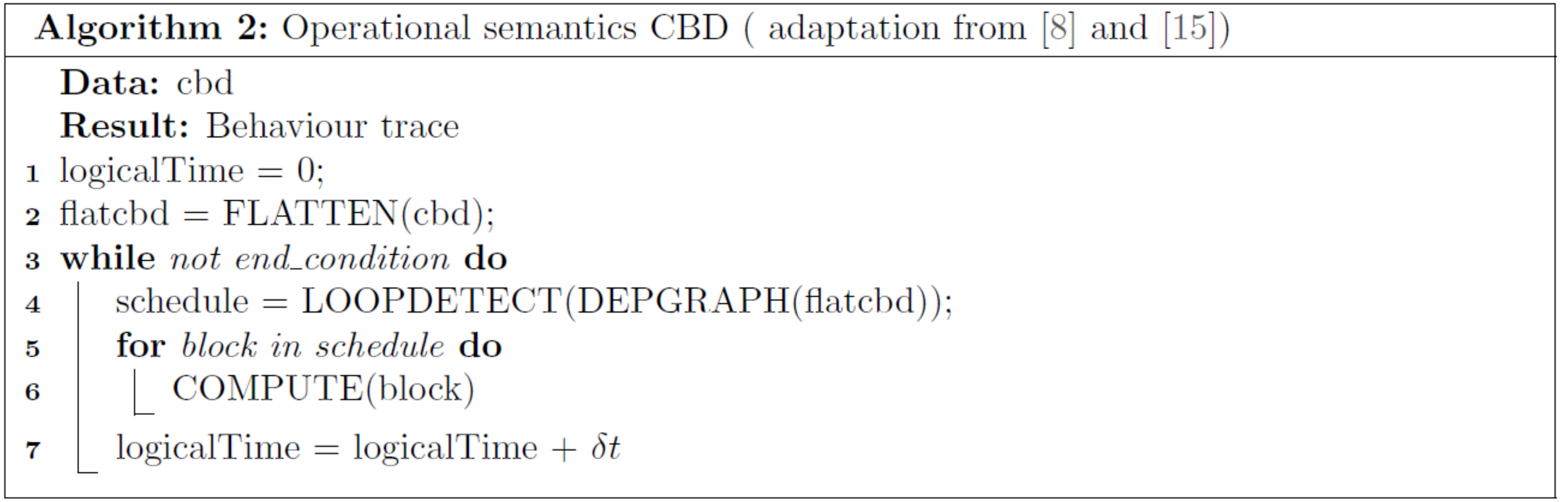
Name	Block
Integrator block	
Derivator block	

Name	Block
Delay block	

Overview

- Background: Causal Block Diagrams (CBD)
  - Syntax
  - Semantics
- Problem statement
- Dynamic structure CBD
  - Syntax
  - Semantics
- Implementation
- Case study: elevator model

Causal Block Diagrams semantics



Algorithm 2: Operational semantics CBD (adaptation from [8] and [15])

```
Data: cbd
Result: Behaviour trace
logicalTime = 0;
flatcbd = FLATTEN(cbd);
while not end_condition do
    schedule = LOOPDETECT(DEPGRAPH(flatcbd));
    for block in schedule do
        COMPUTE(block)
    logicalTime = logicalTime + $\delta t$
```

Background: flattening

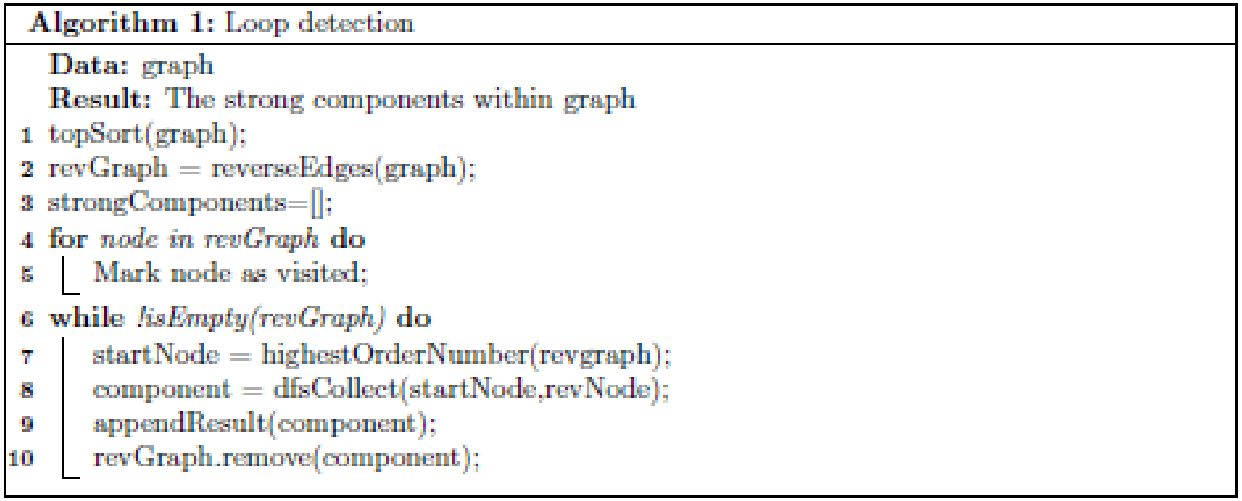
- Purely syntactical
- Needed for loop detection

Background: Evaluation Order

- Dependenties beween blocks
- Topological sort

Loop detection

- Strong component algorithm
- Use gausian solver for implicit solution



Algorithm 1: Loop detection

```
Data: graph
Result: The strong components within graph
topSort(graph);
revGraph = reverseEdges(graph);
strongComponents=[];
for node in revGraph do
    Mark node as visited;
while !isEmpty(revGraph) do
    startNode = highestOrderNumber(revgraph);
    component = dfsCollect(startNode,revNode);
    appendResult(component);
    revGraph.remove(component);
```

Overview

- Background: Causal Block Diagrams (CBD)
  - Syntax
  - Semantics
- Problem statement
- Dynamic structure CBD
  - Syntax
  - Semantics
- Implementation
- Case study: elevator model

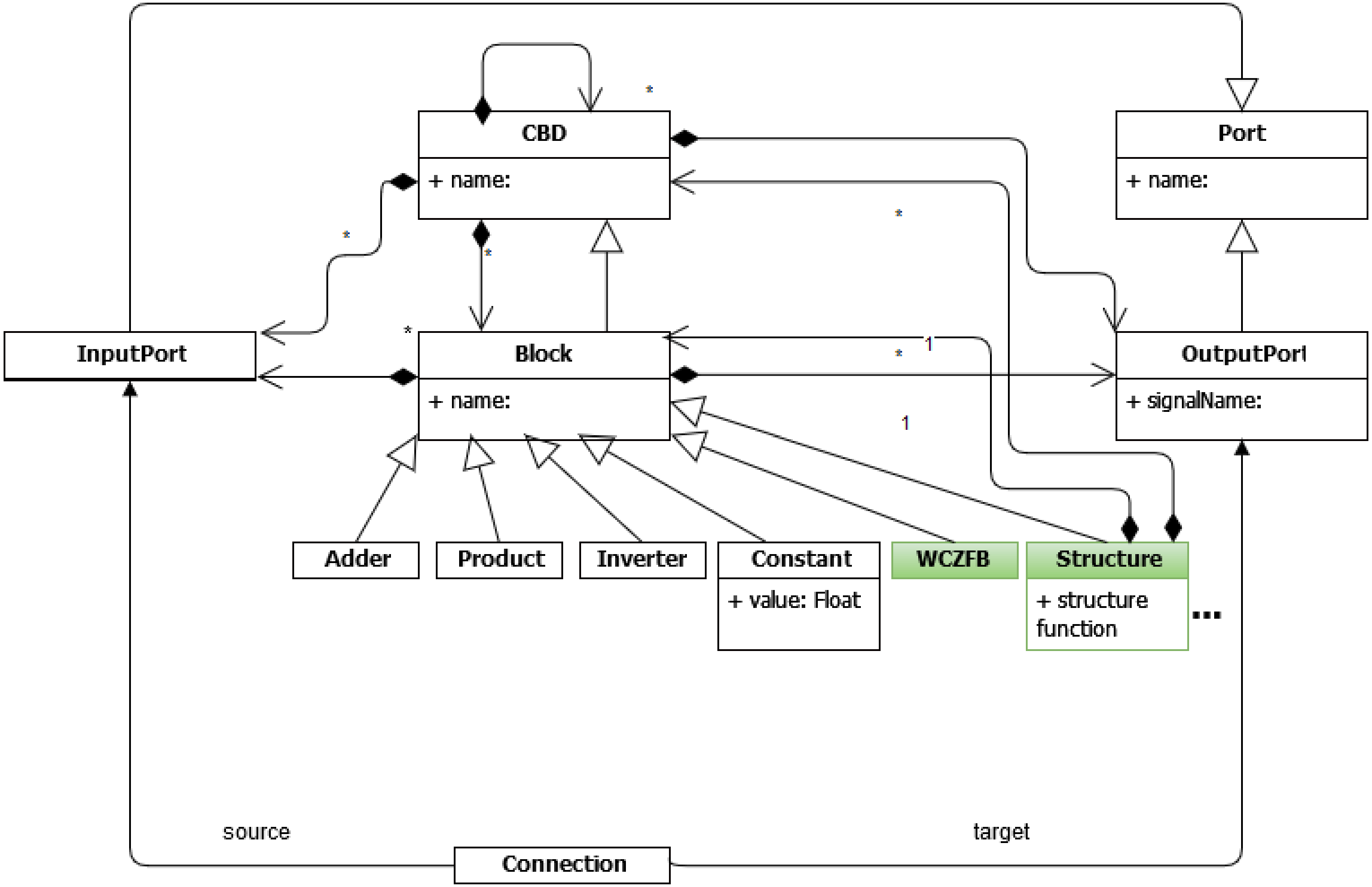
Problem Statement

- Expressiveness limited by fixed structure
- Changing model during simulation
- Staying consistent with CBD constructs

Schedule

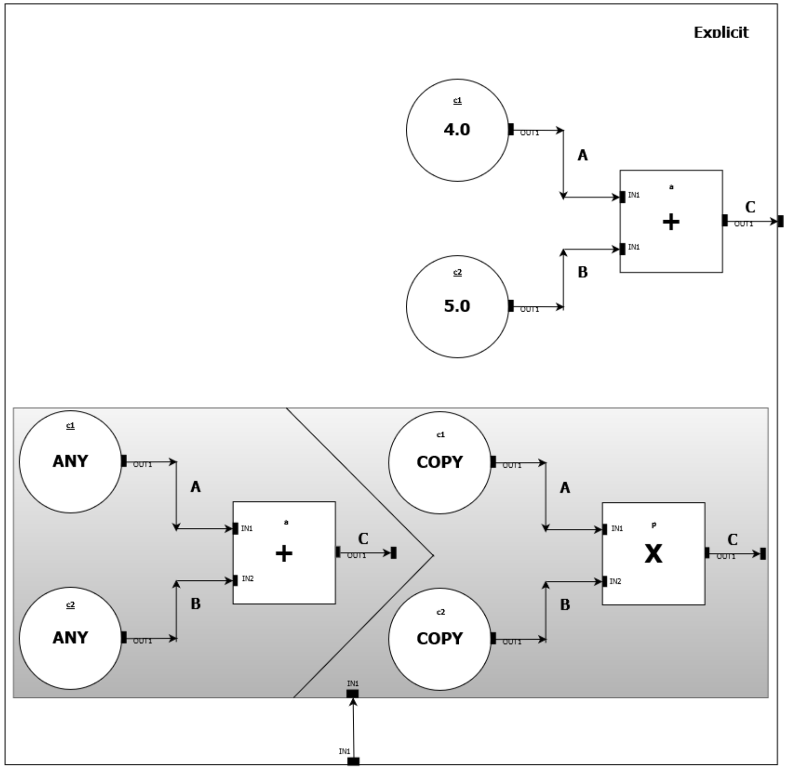
- Background: Causal Block Diagrams
- Problem statement
- Dynamic structure CBD
  - Syntax
  - Semantics
- Implementation
- Case study: elevator model

Abstract Syntax



Visual syntax: explicit representation

- Reuse of existing graph transformation syntax
- Left-hand side before
- Right-hand side after
- Trigger trough input port

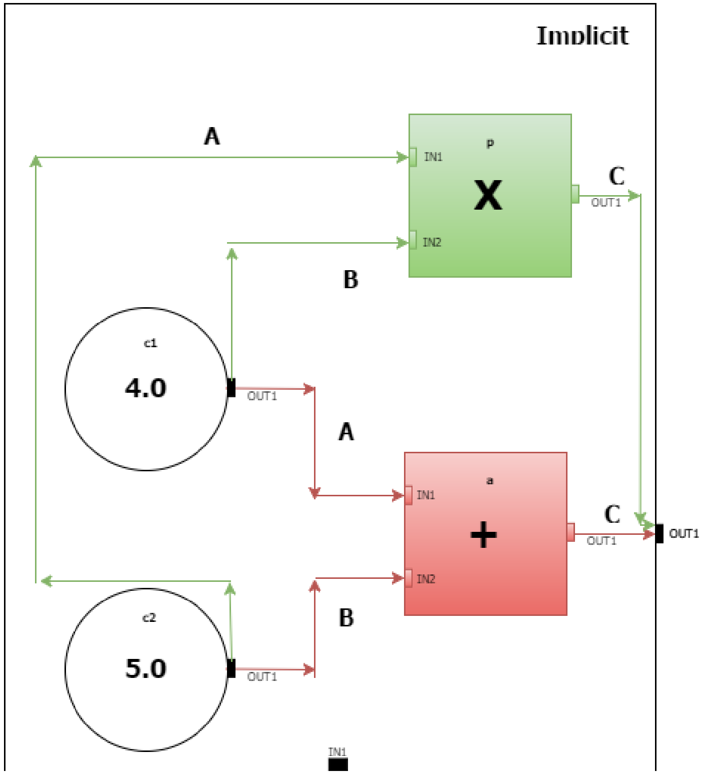


Visual syntax: implicit representation

– Trigger received trough CBD input port

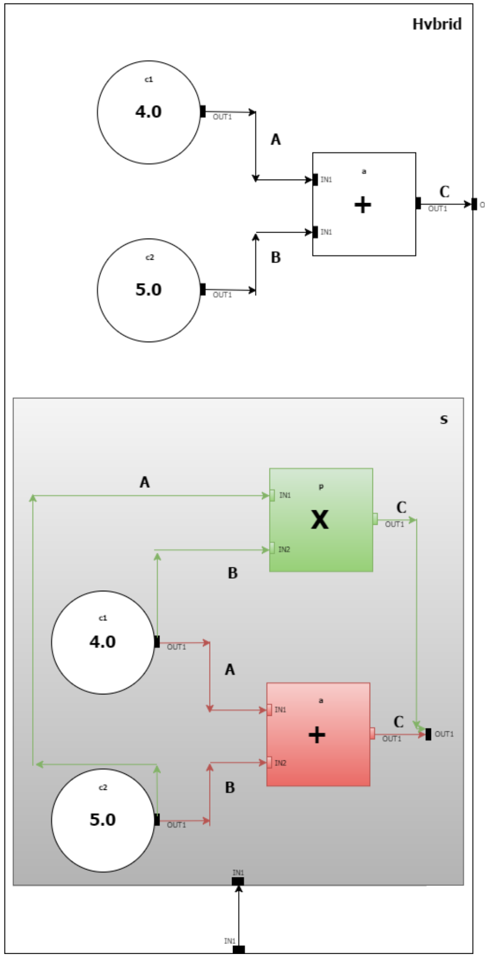
– Added structures: green

– Removed structures: red



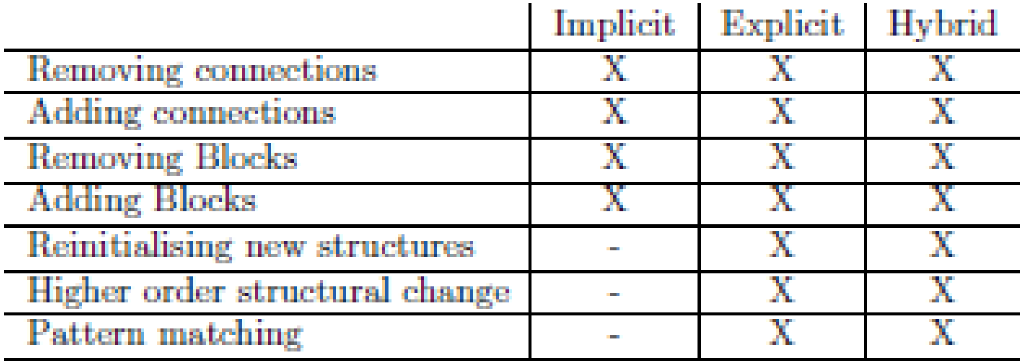
Visual syntax: hybrid representation

- Mix between implicit and explicit representation
- Implicit representation separated by block
- Simulated model isolated



Visual syntax: comparison

- Implicit representation not expressive enough
- Hybrid representation
  - Same expressiveness as explicit representation
  - More compact



	Implicit	Explicit	Hybrid
Removing connections	X	X	X
Adding connections	X	X	X
Removing Blocks	X	X	X
Adding Blocks	X	X	X
Reinitialising new structures	-	X	X
Higher order structural change	-	X	X
Pattern matching	-	X	X

Overview

- Background: Causal Block Diagrams (CBD)
  - Syntax
  - Semantics
- Problem statement
- Dynamic structure CBD
  - Visual syntax
  - Semantics
- Implementation
- Case study: elevator model

Related work

- Hybrid CBD

- Mustafiz, Sadaf, et al. "Towards Modular Language Design Using Language Fragments: The Hybrid Systems Case Study." *Information Technology: New Generations*. Springer International Publishing, 2016. 785-797.

  - Uses signal crossing

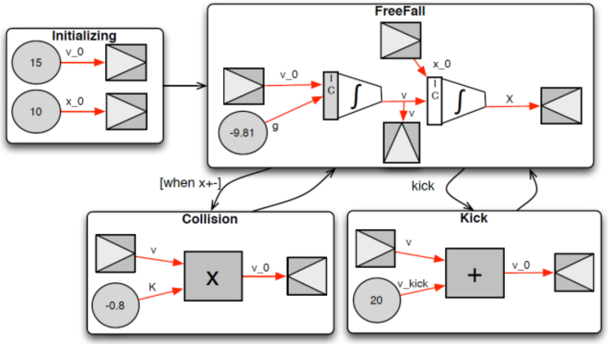
- Dynamic structure DEVS (DSDEVS)

- Barros, Fernando J. "Modeling formalisms for dynamic structure systems."*ACM Transactions on Modeling and Computer Simulation (TOMACS)* 7.4 (1997): 501-515.

- Heterogeneous flow systems

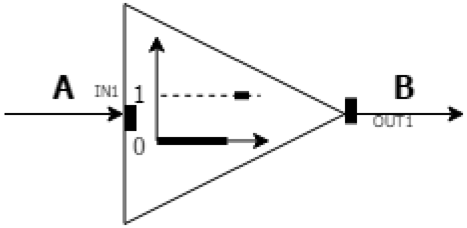
- Barros, Fernando J. "Dynamic structure multiparadigm modeling and simulati *Simulation (TOMACS)* 13.3 (2003): 259-275.

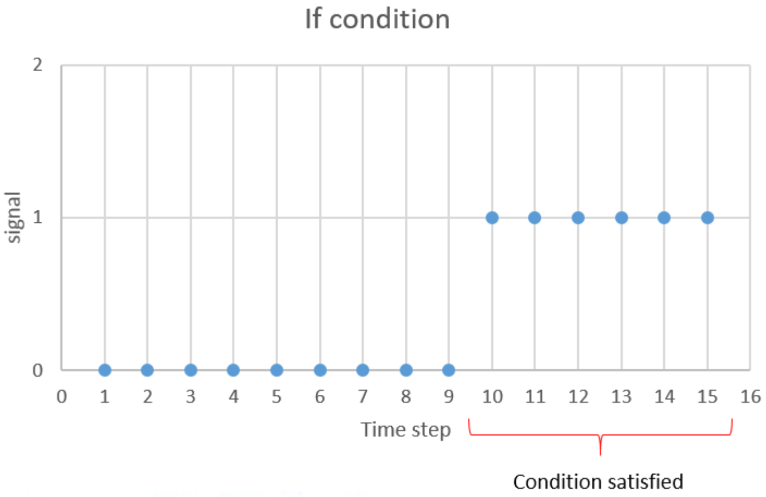
  - Use of model executive

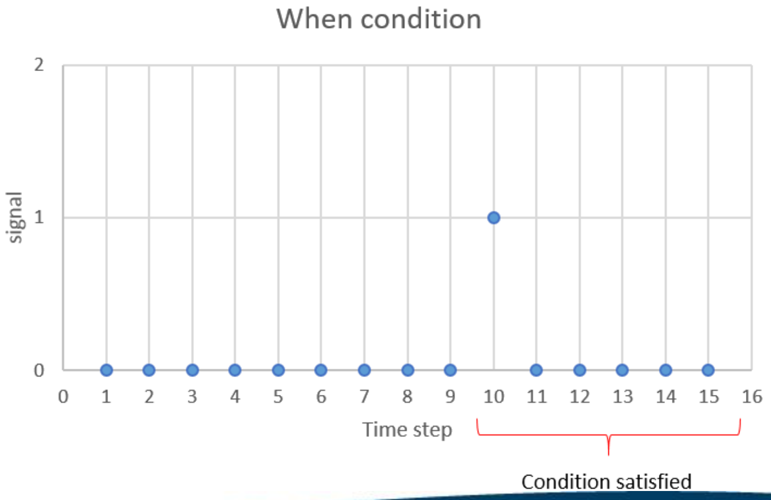


Triggering a change: zero crossing

- Piecewise constant signal to “event”
- Pre and post condition
- Implemented using basic blocks

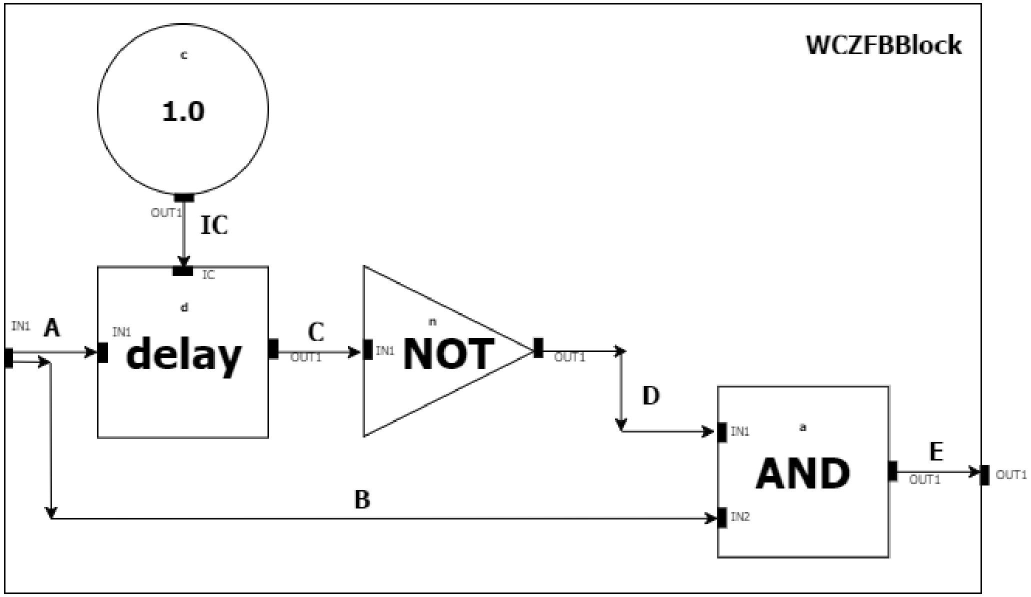






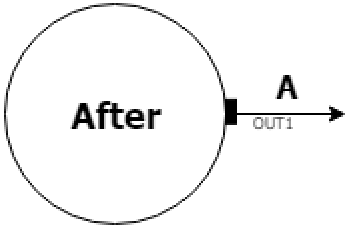
Triggering a change: zero crossing

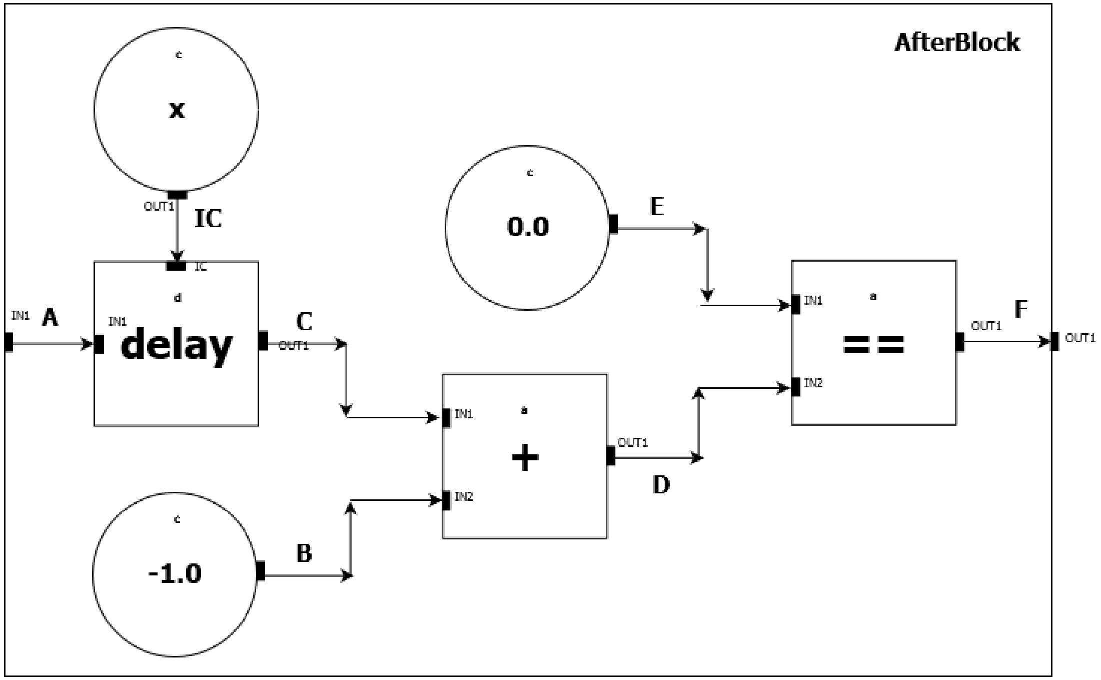
- Signal must cross zero from below
  - Previous iteration: condition must be not satisfied
  - Current iteration: condition must be satisfied



Triggering a change: timed event

- Generating “event” after a fixed number of timesteps
- Value determined when intialised





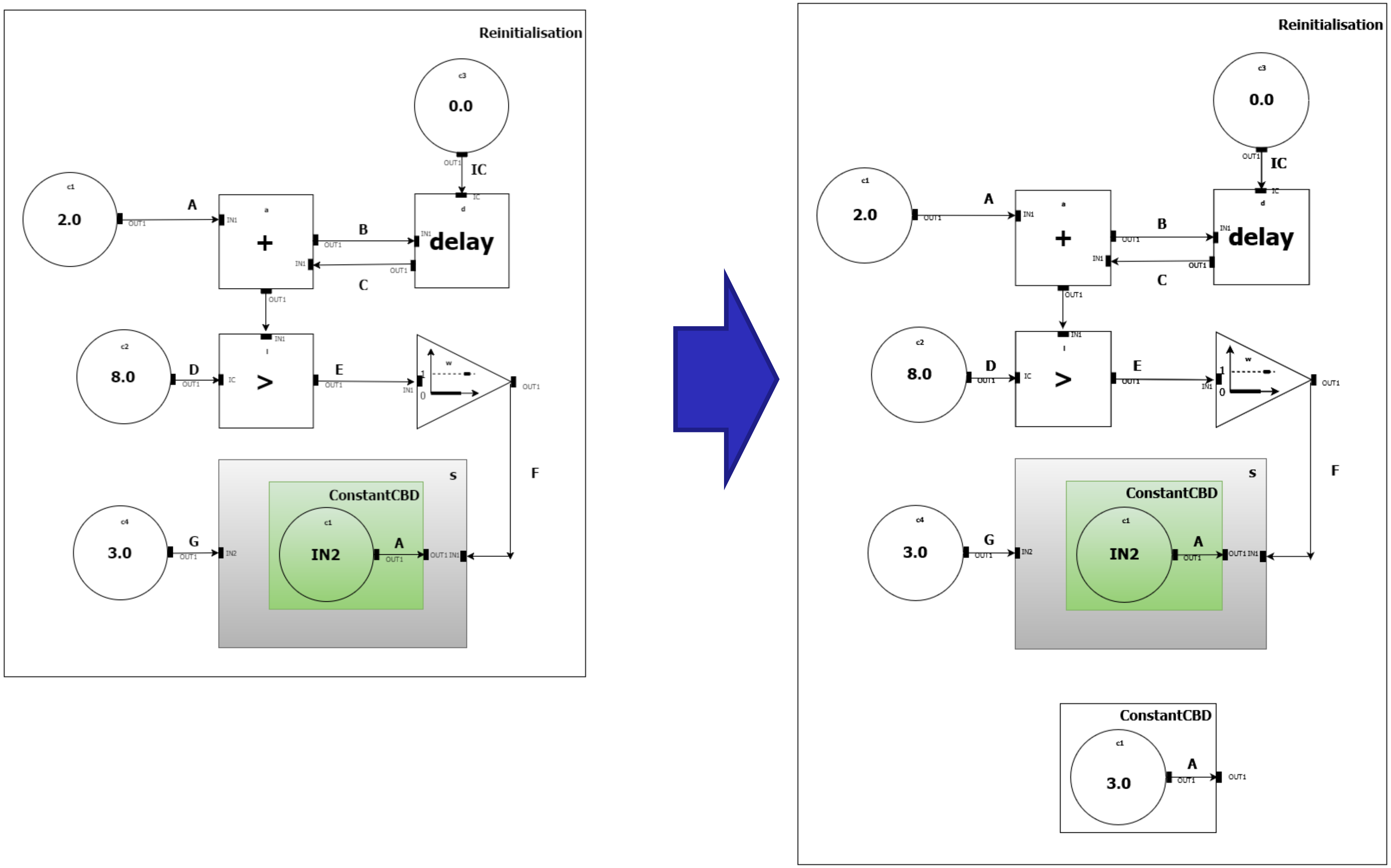
Modeling a change: structure block

- Features of a dynamic structure formalism
  - Identification of existing structures
  - Creation of new structures
  - Removal of existing structures
  - Initialisation of values

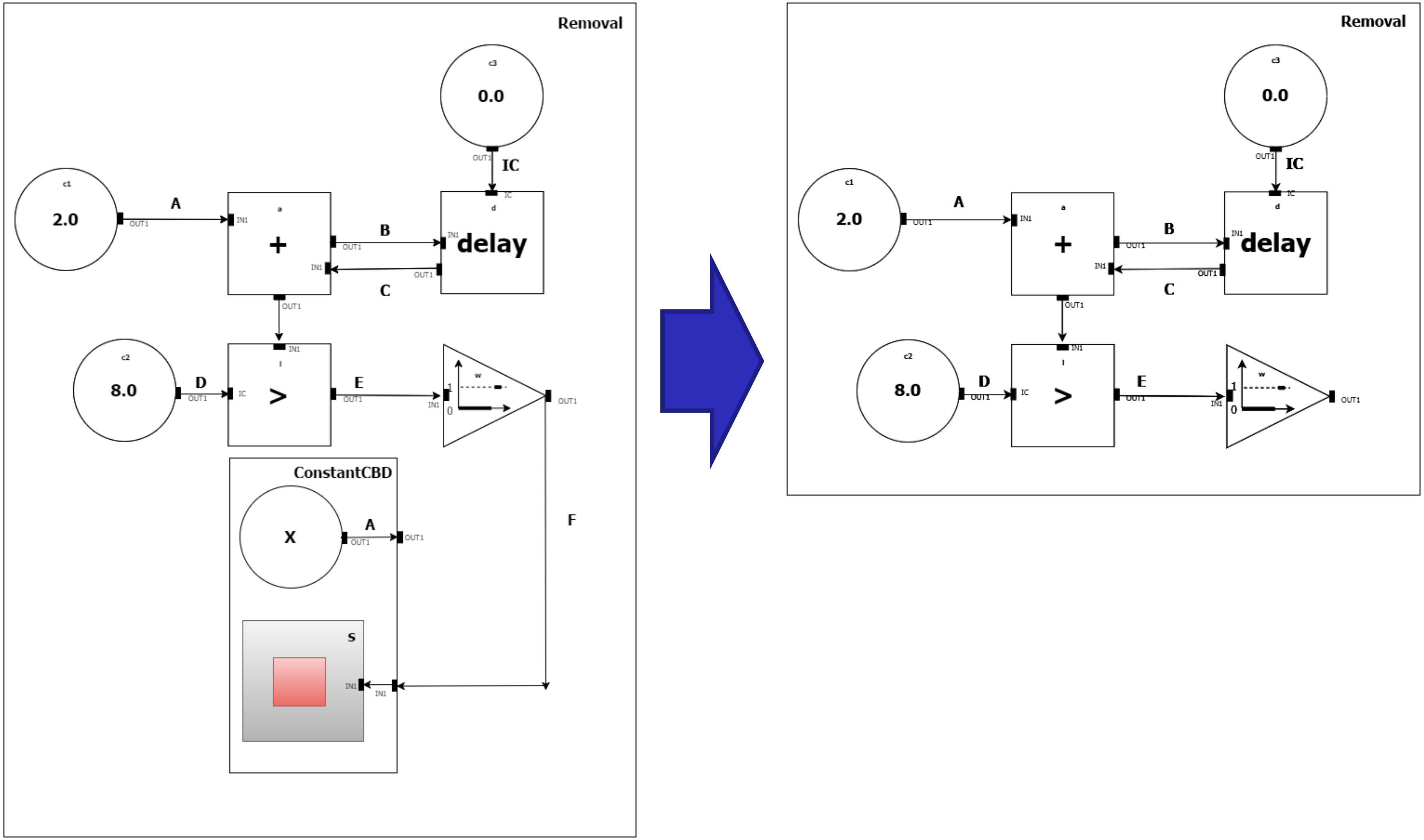
Modeling a change: structure block

- Structure block = adapted CBD specification
- Structure function for modelling change
- Multiple input/ no output ports
  - Default 1: event that triggers a change
  - Other input ports for initialisation of values
- Changes apply only to one CBD! (hierarchical)

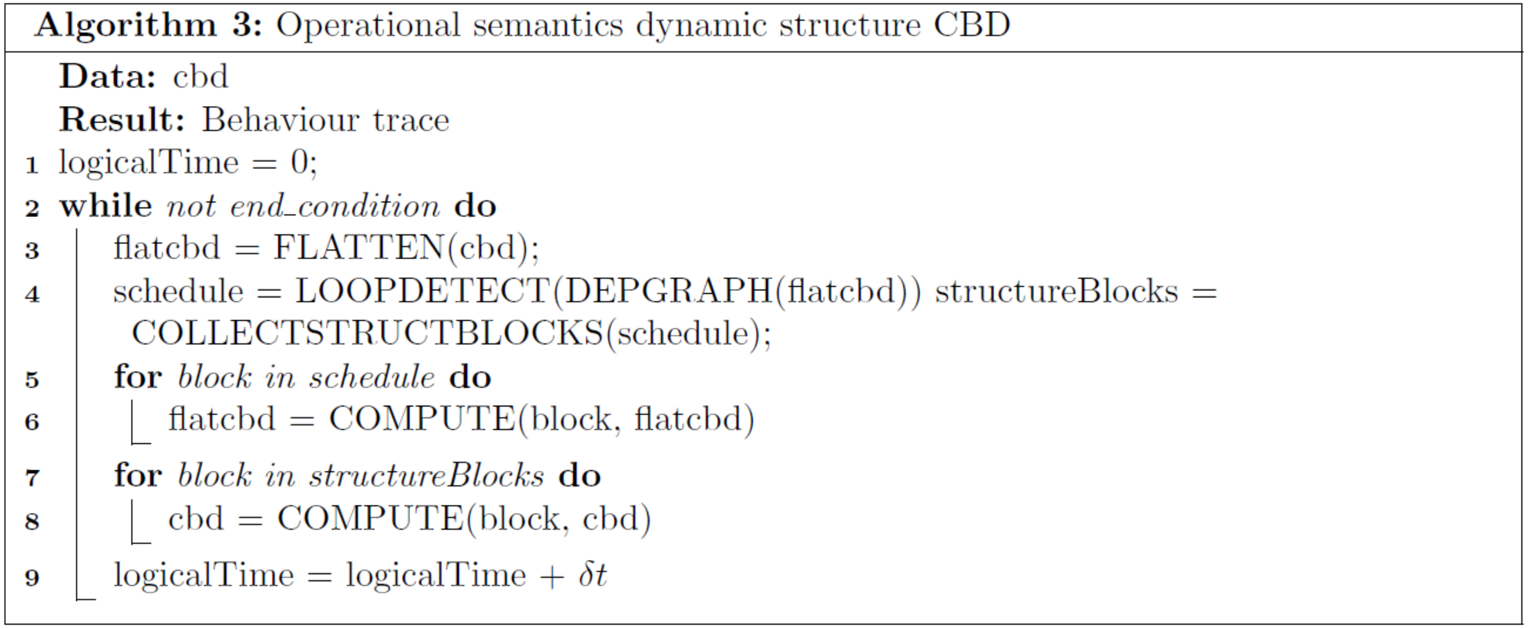
Modeling a change: examples



Modeling a change: examples



Adapted operational semantics



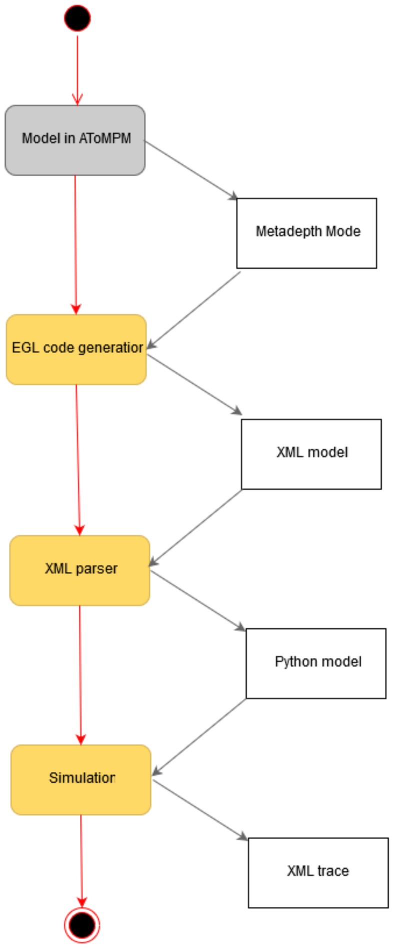
Algorithm 3: Operational semantics dynamic structure CBD

```
Data: cbd
Result: Behaviour trace
logicalTime = 0;
while not end_condition do
    flatcbd = FLATTEN(cbd);
    schedule = LOOPDETECT(DEPGRAPH(flatcbd)) structureBlocks =
     COLLECTSTRUCTBLOCKS(schedule);
    for block in schedule do
        flatcbd = COMPUTE(block, flatcbd)
    for block in structureBlocks do
        cbd = COMPUTE(block, cbd)
    logicalTime = logicalTime + $\delta t$
```

Overview

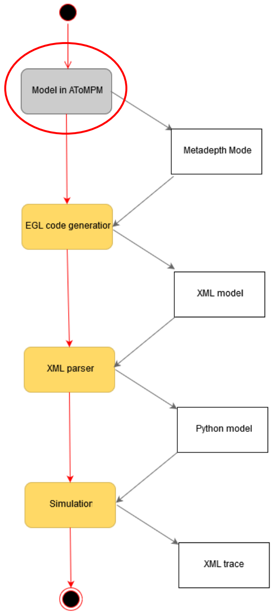
- Background: Causal Block Diagrams (CBD)
  - Syntax
  - Semantics
- Problem statement
- Dynamic structure CBD
  - Syntax
  - Semantics
- Implementation
- Case study: elevator model

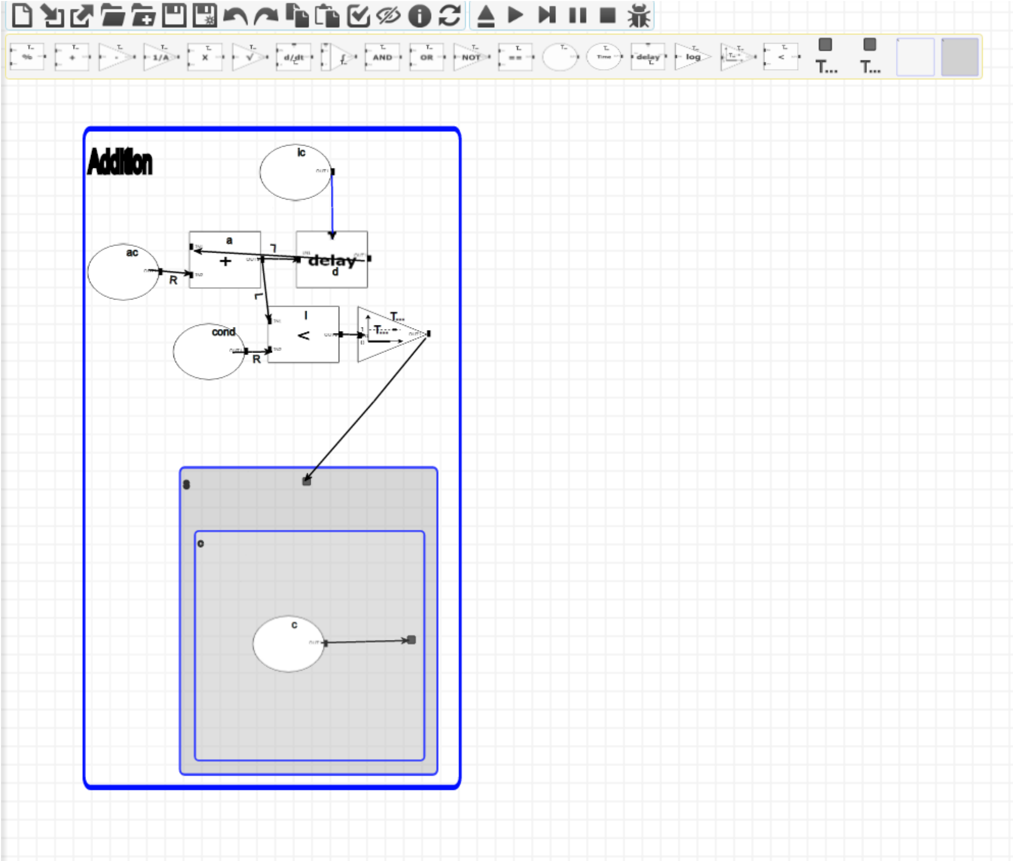
Implementation



Implementation: AToMPM

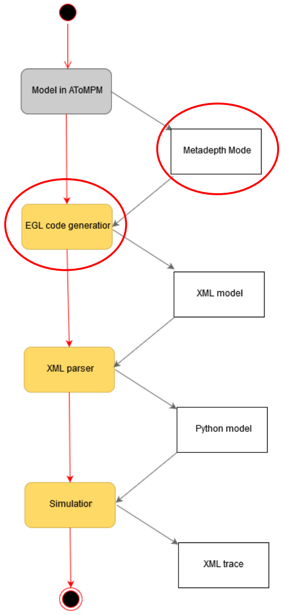
– Visual modeling enviroment

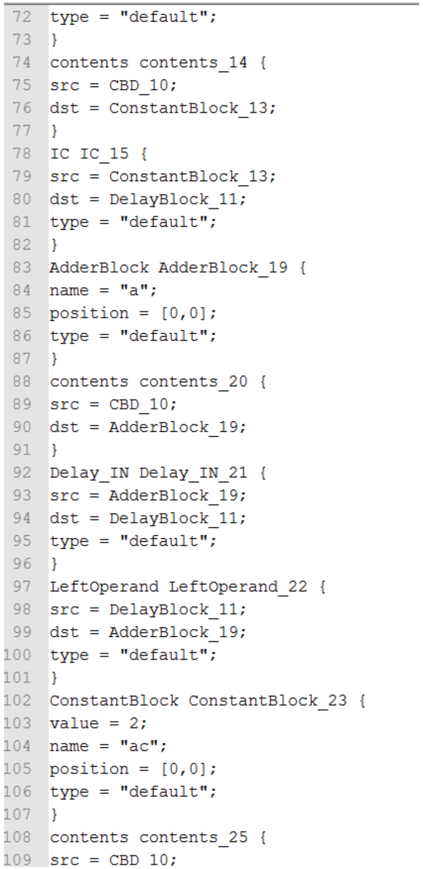




Metadepth

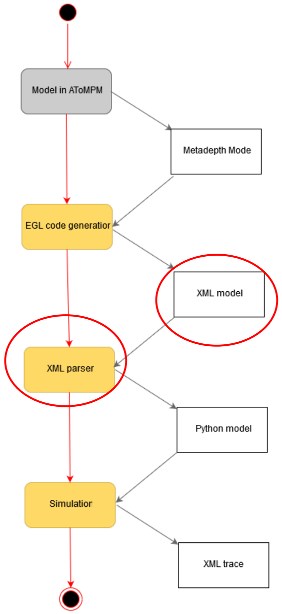
- Exported AToMPM model using built in functionality
- Supports EGL for code generation

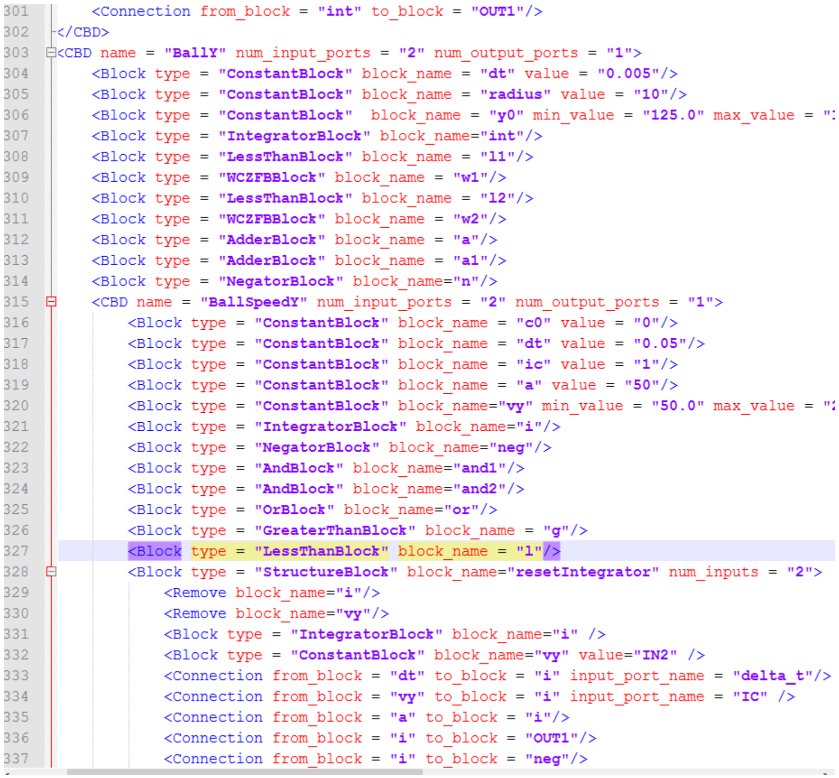



```
type = "default";
}
contents contents_14 {
src = CBD_10;
dst = ConstantBlock_13;
}
IC IC_15 {
src = ConstantBlock_13;
dst = DelayBlock_11;
type = "default";
}
AdderBlock AdderBlock_19 {
name = "a";
position = [0,0];
type = "default";
}
contents contents_20 {
src = CBD_10;
dst = AdderBlock_19;
}
Delay_IN Delay_IN_21 {
src = AdderBlock_19;
dst = DelayBlock_11;
type = "default";
}
LeftOperand LeftOperand_22 {
src = DelayBlock_11;
dst = AdderBlock_19;
type = "default";
}
ConstantBlock ConstantBlock_23 {
value = 2;
name = "ac";
position = [0,0];
type = "default";
}
contents contents_25 {
src = CBD 10;
```

XML

- Readable models
- Consistent declarations
- Remove tags



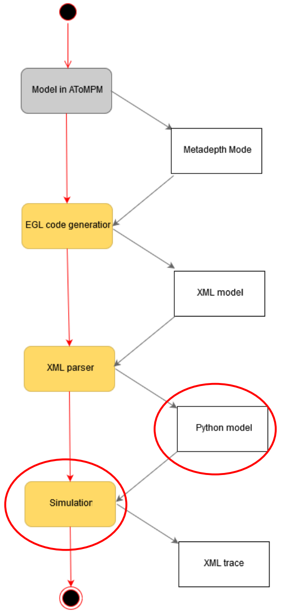


```
    <Connection from_block = "int" to_block = "OUT1"/>
</CBD>
<CBD name = "BallY" num_input_ports = "2" num_output_ports = "1">
    <Block type = "ConstantBlock" block_name = "dt" value = "0.005"/>
    <Block type = "ConstantBlock" block_name = "radius" value = "10"/>
    <Block type = "ConstantBlock"  block_name = "y0" min_value = "125.0" max_value = "
    <Block type = "IntegratorBlock" block_name="int"/>
    <Block type = "LessThanBlock" block_name = "l1"/>
    <Block type = "WCZFBBlock" block_name = "w1"/>
    <Block type = "LessThanBlock" block_name = "l2"/>
    <Block type = "WCZFBBlock" block_name = "w2"/>
    <Block type = "AdderBlock" block_name = "a"/>
    <Block type = "AdderBlock" block_name = "a1"/>
    <Block type = "NegatorBlock" block_name="n"/>
    <CBD name = "BallSpeedY" num_input_ports = "2" num_output_ports = "1">
        <Block type = "ConstantBlock" block_name = "c0" value = "0"/>
        <Block type = "ConstantBlock" block_name = "dt" value = "0.05"/>
        <Block type = "ConstantBlock" block_name = "ic" value = "1"/>
        <Block type = "ConstantBlock" block_name = "a" value = "50"/>
        <Block type = "ConstantBlock" block_name="vy" min_value = "50.0" max_value = "
        <Block type = "IntegratorBlock" block_name="i"/>
        <Block type = "NegatorBlock" block_name="neg"/>
        <Block type = "AndBlock" block_name="and1"/>
        <Block type = "AndBlock" block_name="and2"/>
        <Block type = "OrBlock" block_name="or"/>
        <Block type = "GreaterThanBlock" block_name = "g"/>
        <Block type = "LessThanBlock" block_name = "l"/>
        <Block type = "StructureBlock" block_name="resetIntegrator" num_inputs = "2">
            <Remove block_name="i"/>
            <Remove block_name="vy"/>
            <Block type = "IntegratorBlock" block_name="i" />
            <Block type = "ConstantBlock" block_name="vy" value="IN2" />
            <Connection from_block = "dt" to_block = "i" input_port_name = "delta_t"/>
            <Connection from_block = "vy" to_block = "i" input_port_name = "IC" />
            <Connection from_block = "a" to_block = "i"/>
            <Connection from_block = "i" to_block = "OUT1"/>
            <Connection from_block = "i" to_block = "neg"/>
```

Universiteit Antwerpen

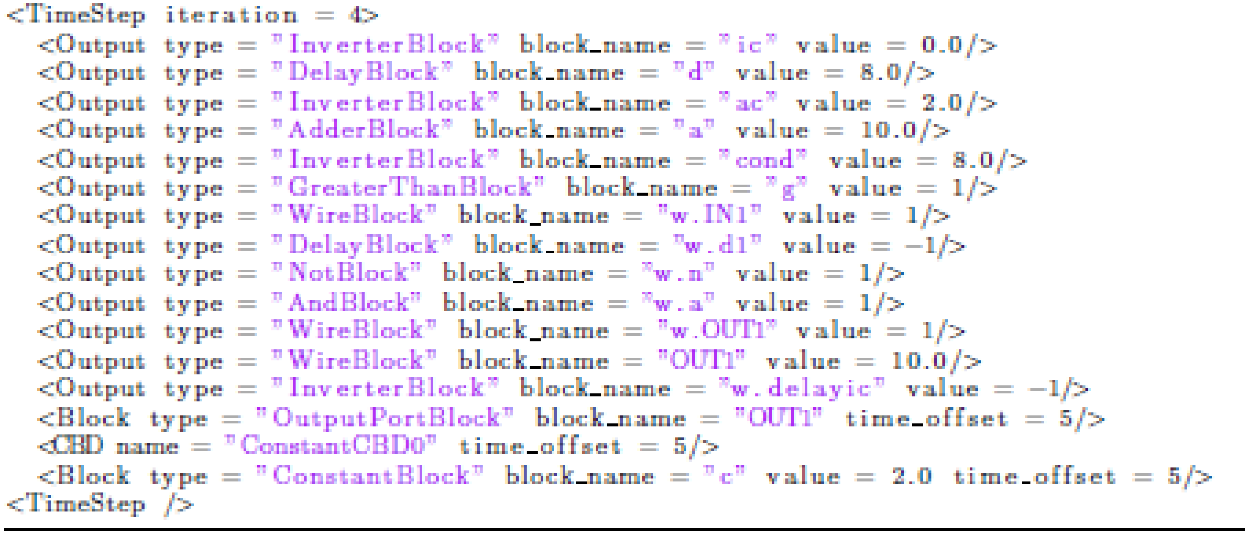
Python

- Parser generates code including structure functions
- Simulator implemented in python
  - Extended CBD simulator from MOSIS course

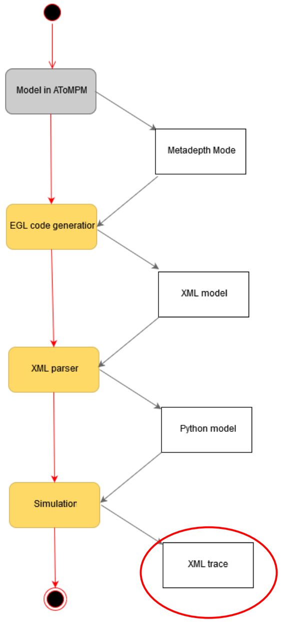


Debugging

- Test driven development
- Traces



```
<TimeStep iteration = 4>
  <Output type = "InverterBlock" block_name = "ic" value = 0.0/>
  <Output type = "DelayBlock" block_name = "d" value = 8.0/>
  <Output type = "InverterBlock" block_name = "ac" value = 2.0/>
  <Output type = "AdderBlock" block_name = "a" value = 10.0/>
  <Output type = "InverterBlock" block_name = "cond" value = 8.0/>
  <Output type = "GreaterThanBlock" block_name = "g" value = 1/>
  <Output type = "WireBlock" block_name = "w.IN1" value = 1/>
  <Output type = "DelayBlock" block_name = "w.d1" value = −1/>
  <Output type = "NotBlock" block_name = "w.n" value = 1/>
  <Output type = "AndBlock" block_name = "w.a" value = 1/>
  <Output type = "WireBlock" block_name = "w.OUT1" value = 1/>
  <Output type = "WireBlock" block_name = "OUT1" value = 10.0/>
  <Output type = "InverterBlock" block_name = "w.delayic" value = −1/>
  <Block type = "OutputPortBlock" block_name = "OUT1" time_offset = 5/>
  <CBD name = "ConstantCBD0" time_offset = 5/>
  <Block type = "ConstantBlock" block_name = "c" value = 2.0 time_offset = 5/>
<TimeStep />
```



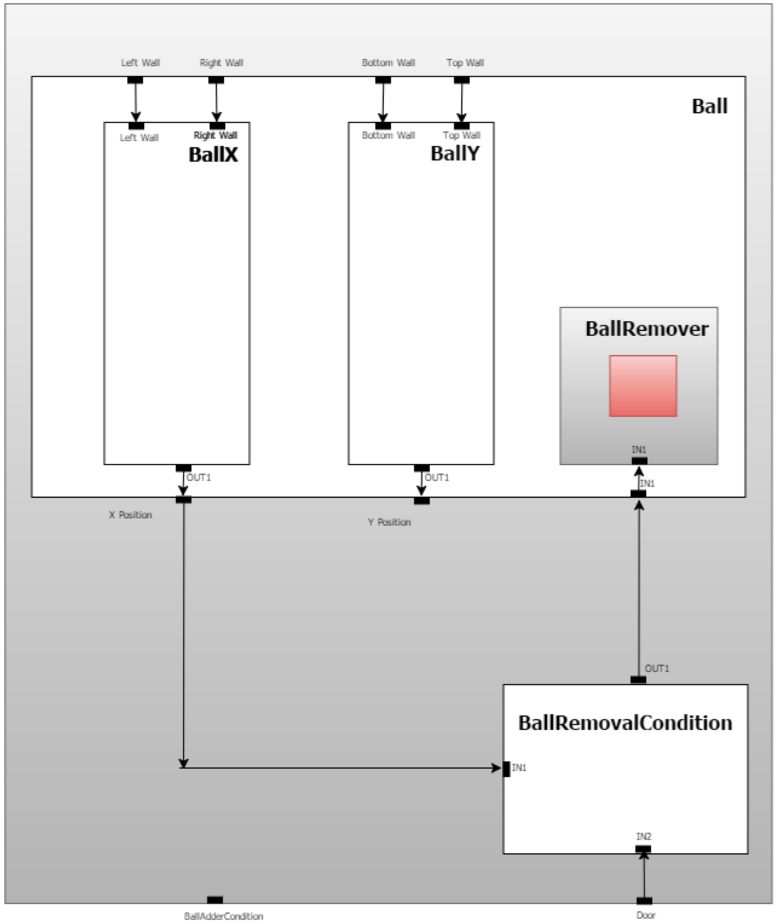
Overview

- Background: Causal Block Diagrams (CBD)
  - Syntax
  - Semantics
- Problem statement
- Dynamic structure CBD
  - Syntax
  - Semantics
- Implementation
- Case study: elevator model

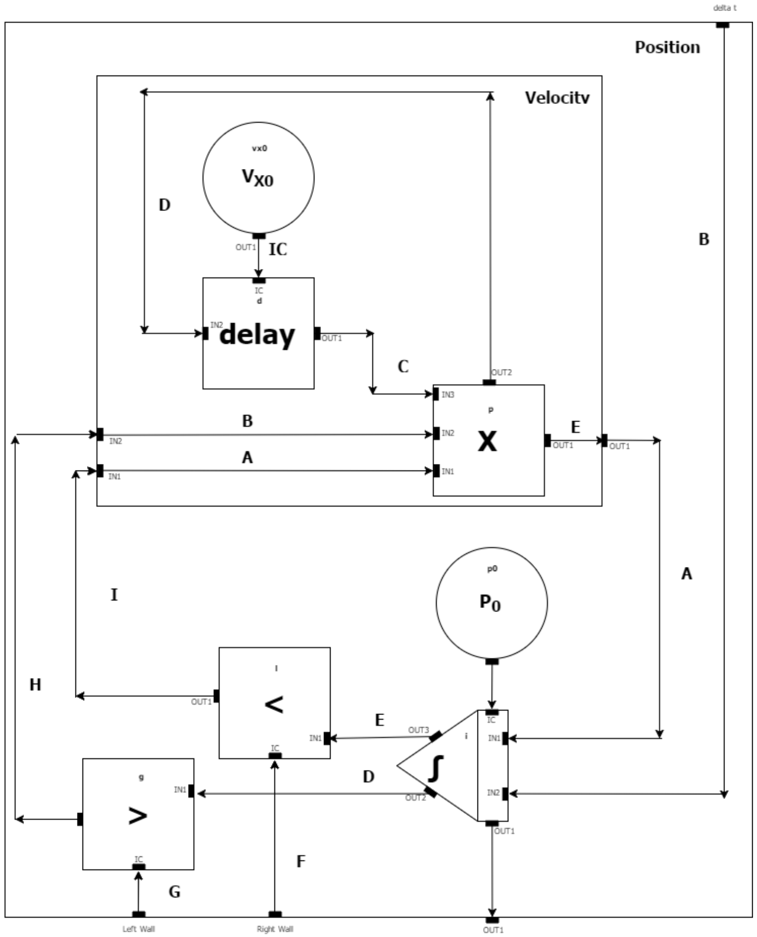
Case Study

- Balls in elevator
- Doors open when elevator reaches floor
- Balls can enter and leave elevator trough door

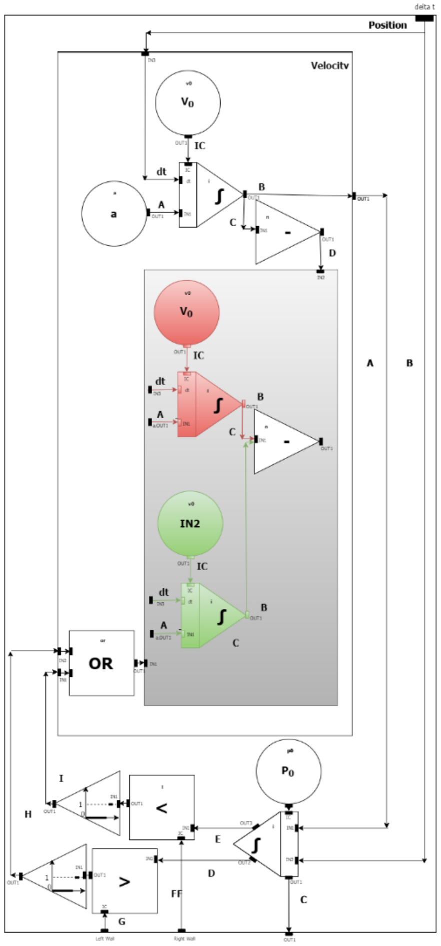
Modeling a Ball



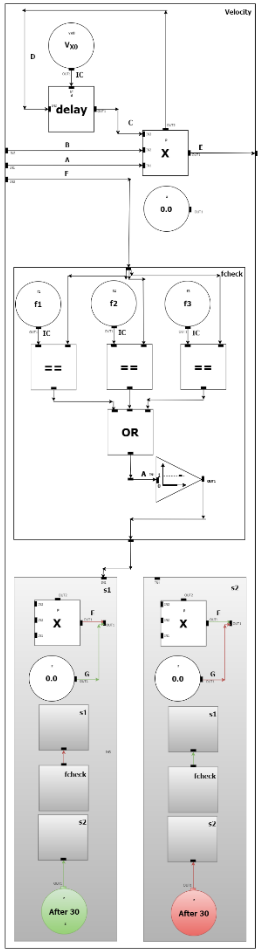
Position with constant velocity



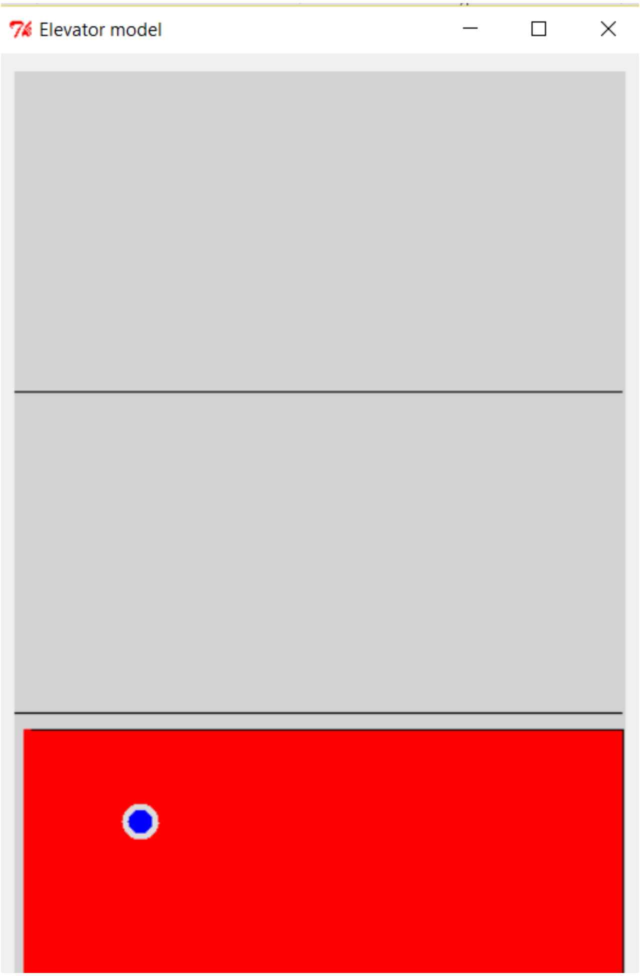
Position with constant acceleration



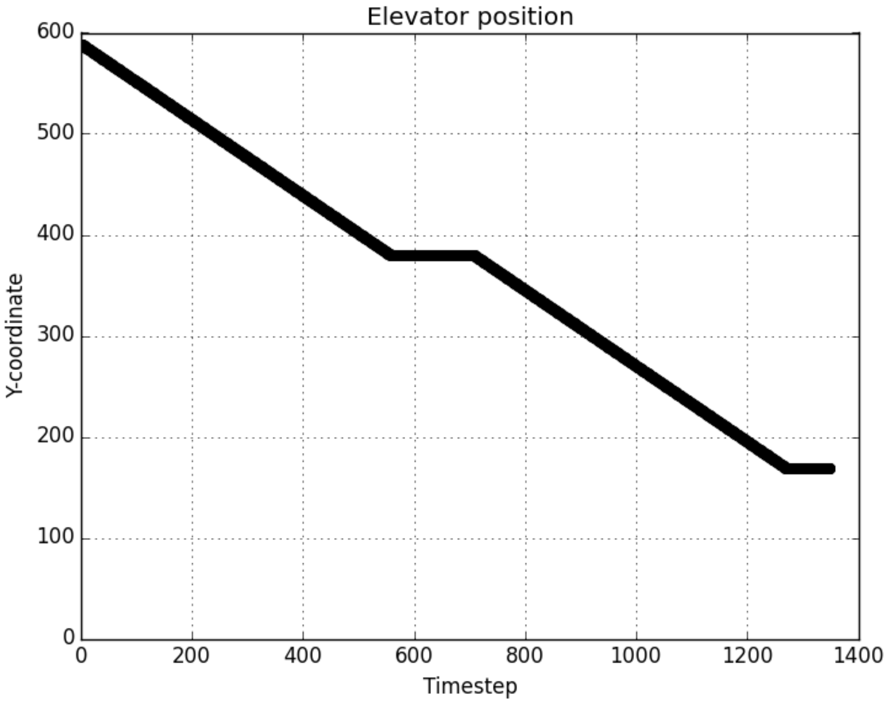
Velocity CBD for elevator wall



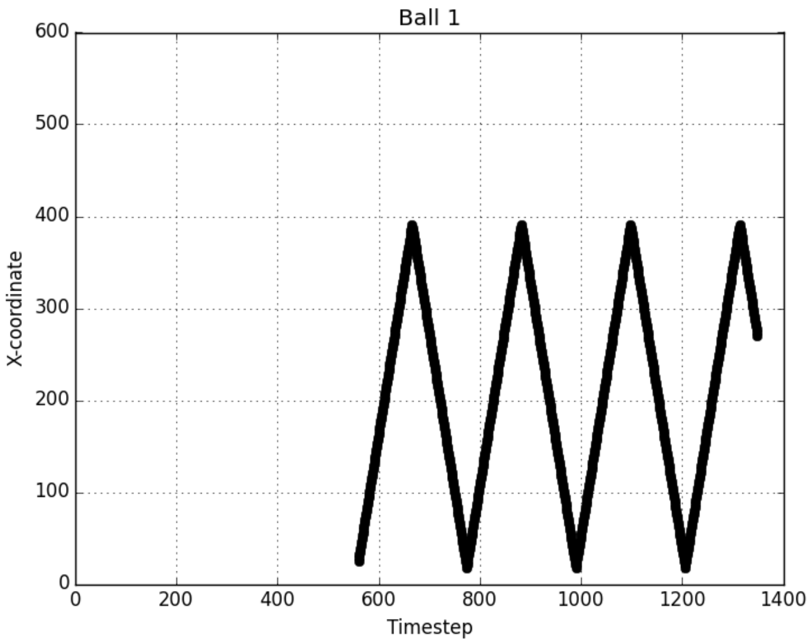
Demo

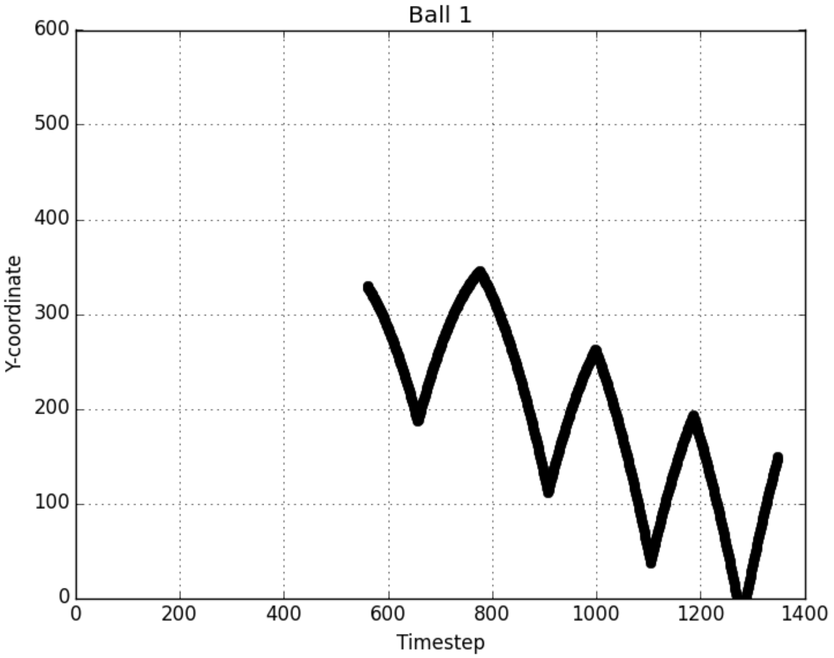


Execution plots

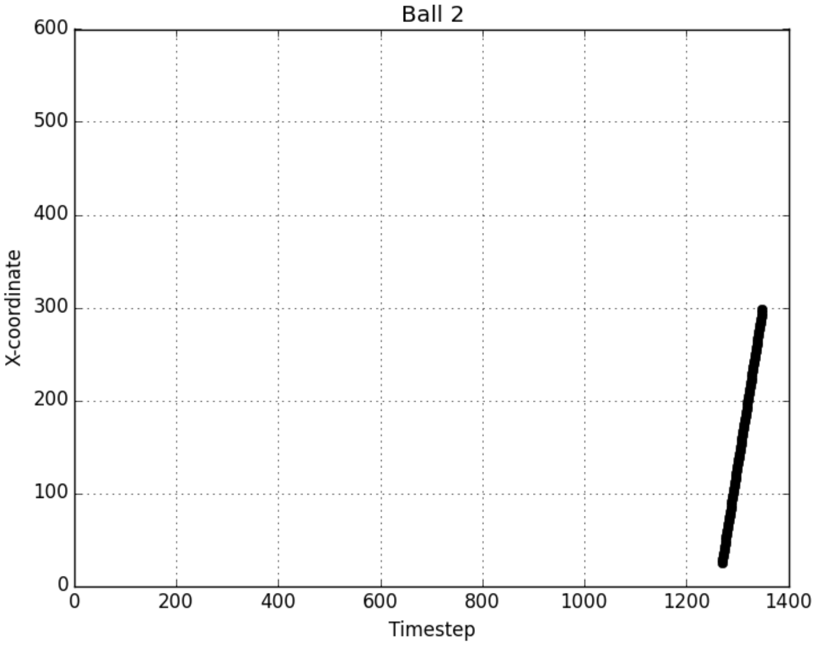


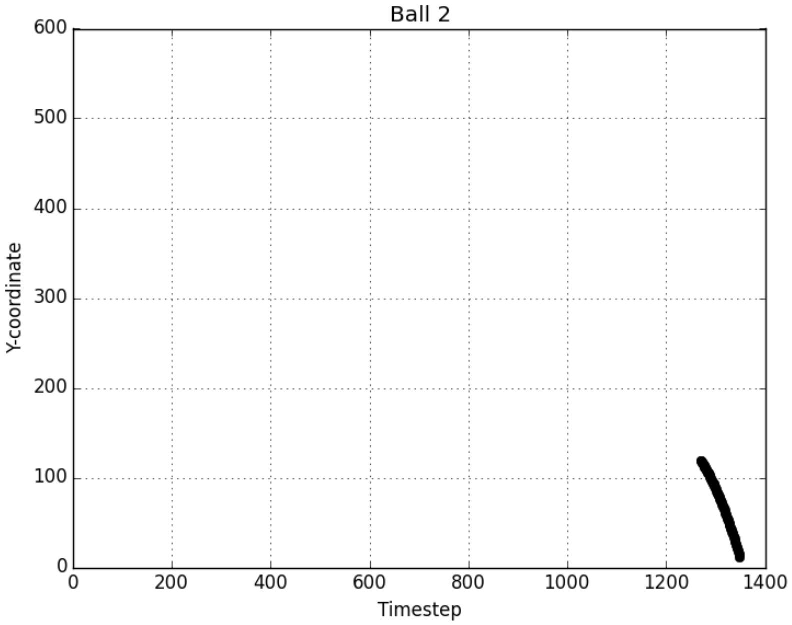
Execution plots





Execution plots





Future work

- Advanced scheduling of structure blocks
- Optimisation techniques
- Comparison to other methods (hybrid systems)

Questions?