



Heterogeneous Function Composition in Embedded Software Synthesis to Eliminate Direct Relations Between Components

Pieter J. Mosterman

Senior Research Scientist
Design Automation Department

Adjunct Professor
School of Computer Science



© 2013 The MathWorks, Inc.

Agenda

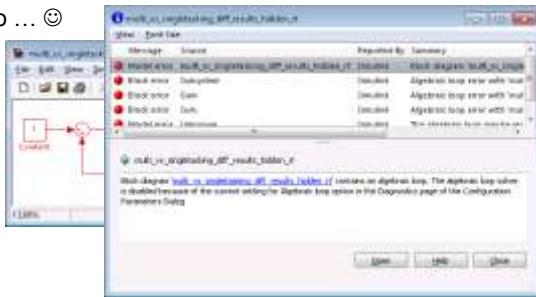
- ▪ Simulink preliminaries
 - Executing a Simulink model
 - Dealing with hierarchy
 - Heterogeneous composition
 - An implementation in Simulink
 - Conclusions



3

The problem!

- Demo ... ☺



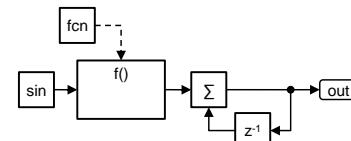
- What happened?!

2



Basic Simulink syntax

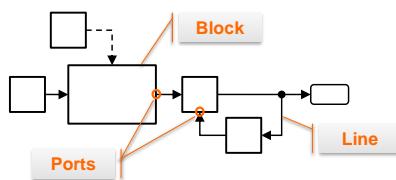
- An example model ...



4

Basic Simulink syntax

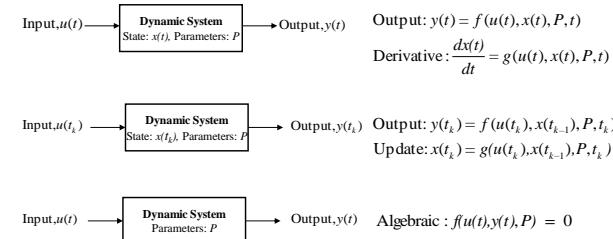
- Network of **blocks** with directed **lines** connected between **ports**



5

The execution hierarchy

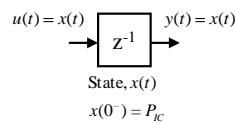
- The lines reflect input/output relations
- The (nonvirtual) blocks are dynamic systems



6

Example block implementation

- A unit delay



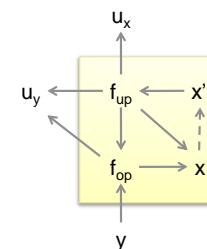
```
class UnitDelayBlock : public Block {
public:
    ErrorStatus BlockDrawIcon() {
        // Draw 'z^-1' on the icon
    }
    BlockParameterData BlockGetParameterData() {
        // Return initial_condition as block data
    }
    ErrorStatus BlockOutput() {
        // Implement y(t) = x(t)
    }
    ErrorStatus BlockUpdate() {
        // Implement x(t) = u(t)
    }
private:
    double initial_condition;
};
```

7

The basic structure of a discrete time block

- Data dependencies between

- Input signals, output signals, current state, new state, update function and output function

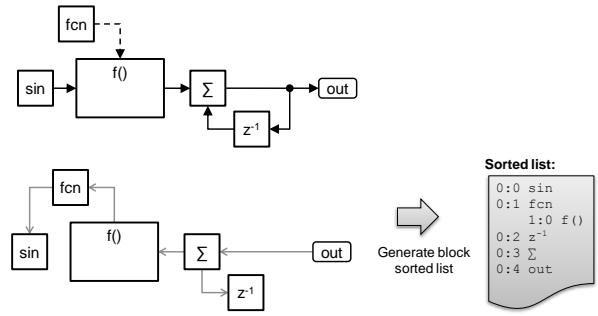


8



Data dependencies to create a sorted list for efficient execution

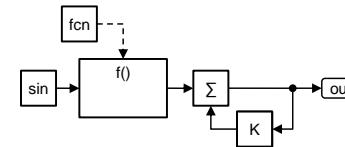
- Static dependency analysis



9

Algebraic dependencies

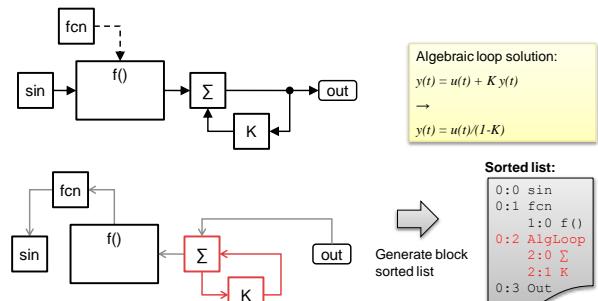
- What if we replace the delay block by a gain?



10

Algebraic dependencies

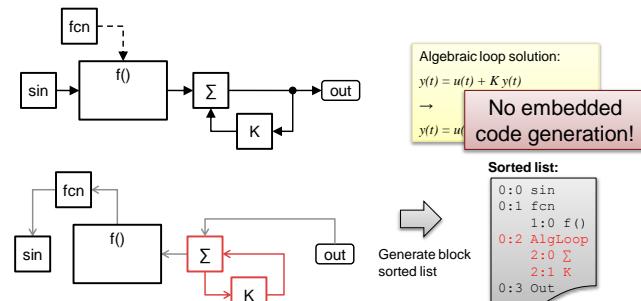
- Strongly connected components



11

Algebraic dependencies

- Strongly connected components



12

Agenda

- - Simulink preliminaries
 - Executing a Simulink model
 - Dealing with hierarchy
 - Heterogeneous composition
 - An implementation in Simulink
 - Conclusions

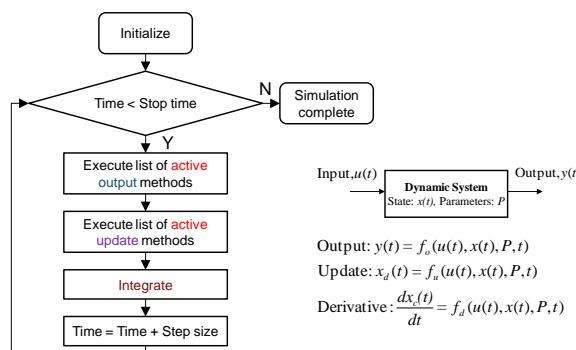
13

Let's assume single-tasking execution

- Allows multi-rate systems
- All blocks run in a single task
 - Single execution time line
 - Base rate at greatest common denominator
 - Blocks execute when they have a sample hit
 - No data integrity issues

14

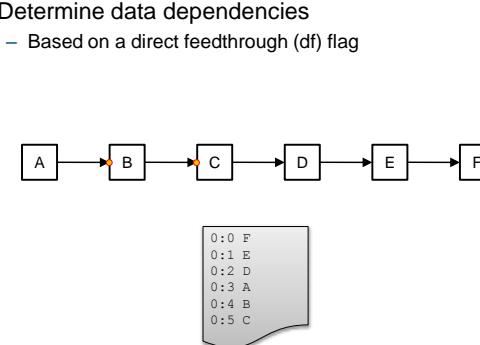
A simulation algorithm for networks of dynamic systems



15

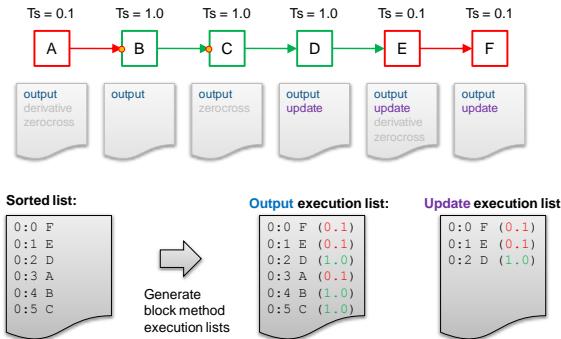
First generate the sorted list

- Determine data dependencies
 - Based on a direct feedthrough (df) flag



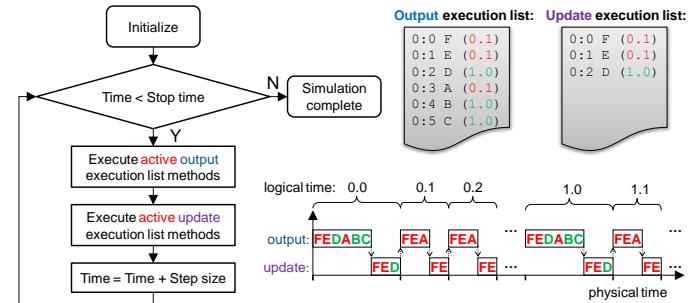
16

Generate execution lists for all of the block methods



17

Repeatedly evaluate the execution lists



18

Agenda

- ▪ Simulink preliminaries
- Executing a Simulink model
- Dealing with hierarchy
- Heterogeneous composition
- An implementation in Simulink
- Conclusions

19

Hierarchy

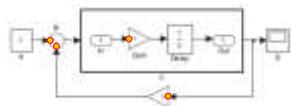
- **Virtual** blocks to organize graphical hierarchy
 - Referential transparency; no semantic bearing
- **Nonvirtual** blocks to organize
 - Execution hierarchy
 - Data scope hierarchy

20



A graphical hierarchy

- Group blocks in a **virtual** subsystem
 - Subsystem C does not appear in the sorted list



Sorted list:

0:0 Delay
0:1 D
0:2 A
0:3 E
0:4 B
0:5 Gain

Generate block method execution lists

Output execution list:

0:0 Delay
0:1 D
0:2 A
0:3 E
0:4 B
0:5 Gain

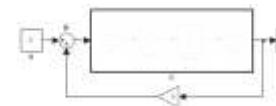
Update execution list:

0:0 Delay
0:1 D
0:2 A
0:3 E
0:4 B
0:5 Gain

21

Let's create a component ...

- Make the virtual subsystem **nonvirtual**
 - Becomes a dynamic system in its own right



Input, $u(t)$ → **Dynamic System**
State: $x(t)$, Parameters: P → Output, $y(t)$

Output: $y(t) = f_o(u(t), x(t), P, t)$
Update: $x_d(t) = f_u(u(t), x(t), P, t)$

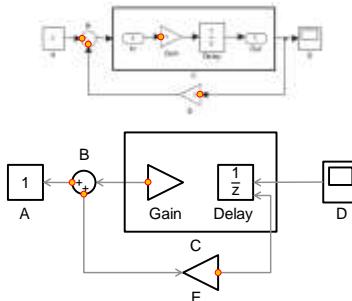
Derivative: $\frac{dx_e(t)}{dt} = f_d(u(t), x(t), P, t)$

- Does that affect sorting?

22

Let's start with a virtual subsystem ...

- Dependencies derived from **df** flag



23



Make subsystem an atomic component

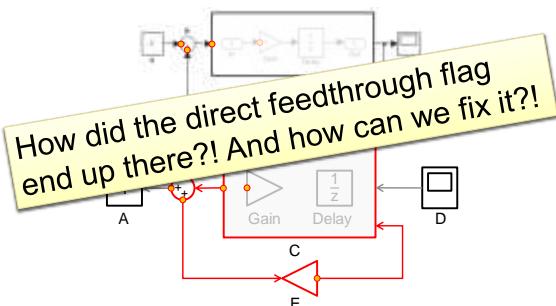
- Direct feedthrough moves to component level!



24

Make subsystem an atomic component

- Now we have a dependency cycle!



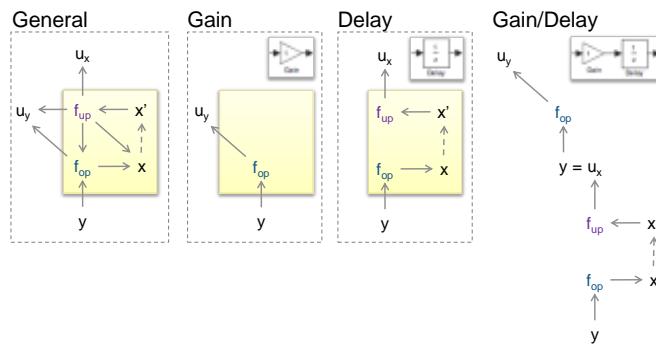
25

Agenda

- Simulink preliminaries
- Executing a Simulink model
- Dealing with hierarchy
- Heterogeneous composition
- An implementation in Simulink
- Conclusions

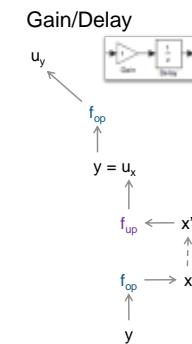
26

Gain followed by delay



27

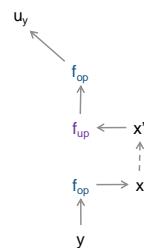
Gain followed by delay



28

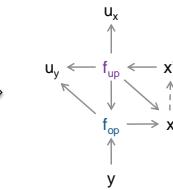
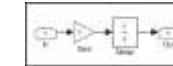
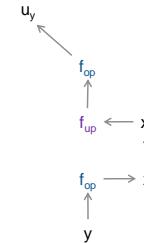
Gain followed by delay

Gain/Delay



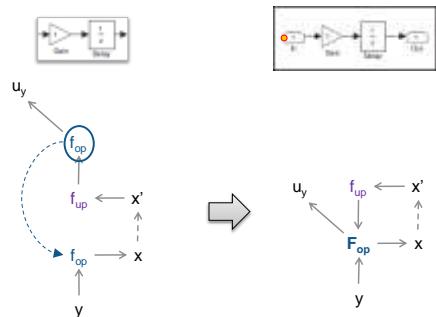
29

How do we create a component?



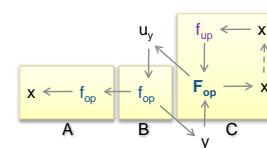
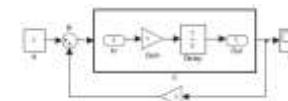
30

Homogeneous composition



31

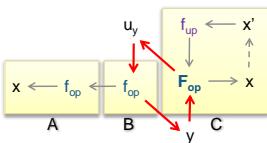
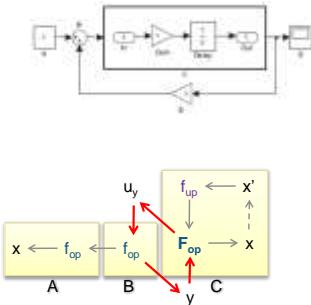
Put the component in context



32



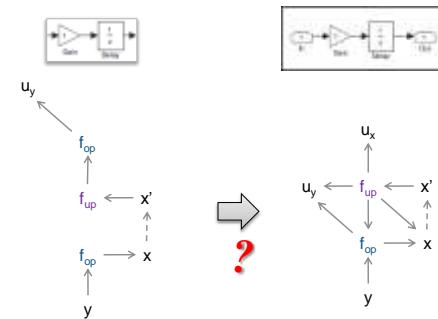
We have a dependency cycle!



33



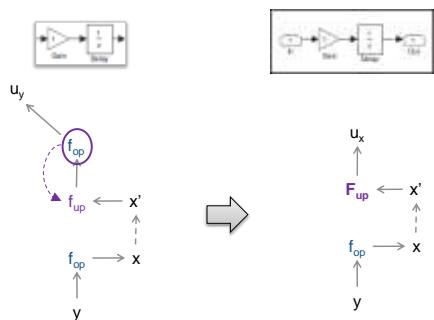
Is there another way ... ?



34



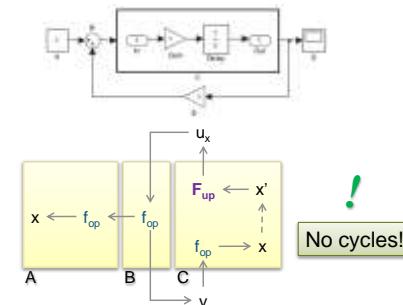
Heterogeneous composition!



35



Putting it in context again



36



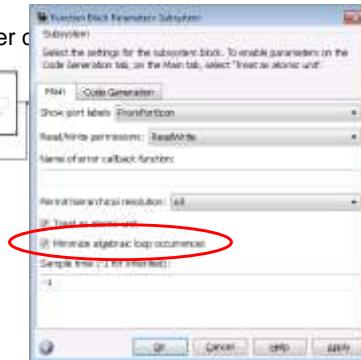
Agenda

- Simulink preliminaries
- Executing a Simulink model
- Dealing with hierarchy
- ▶ Heterogeneous composition
- An implementation in Simulink
- Conclusions

37

How do we fit this into the model compilation machinery?

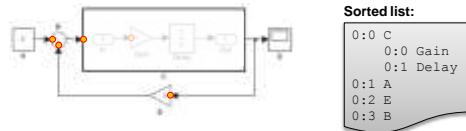
- Create sorted list after clearing df flag



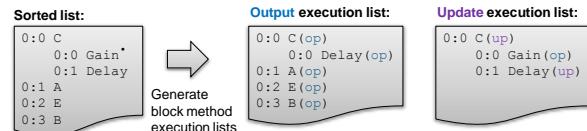
38

How do we fit this into the model compilation machinery?

- Create sorted list after clearing df flag



- Mark where output for update should be used



39

What does the generated code look like?

```

56 /* Model output function */
57 void gaindelay_output(void)
58 {
59     real_T rtb_E;
60
61     /* Outputs for atomic SubSystem: '<Root>/C' */
62     gaindelay_C();
63
64     /* end of Outputs for SubSystem: '<Root>/C' */
65
66     /* Gain: '<Root>/B' */
67     rtb_E = gaindelay_B_E_Gain * gaindelay_B.Delay;
68
69     /* Sum: '<Root>/B' incorporates:
70      * Constant: '<Root>/A'
71      */
72     gaindelay_B.B = gaindelay_B_A_Value + rtb_E;
73 }

```

Output execution list:

```

0:0 C(op)
0:0 Delay(op)
0:1 A(op)
0:2 E(op)
0:3 B(op)

```

Update execution list:

```

0:0 C(up)
0:0 Gain(op)
0:1 Delay(up)

```

40



What does the generated code look like?

```

56 /* Model output function */
57 void gaindelay_output(void)
58 {
59     real_T rtb_E;
60
61     /* Outputs for atomic SubSystem: '<Root>/C' */
62     gaindelay_C();
63
64     /* end of Output */ 37 /* Outputs for atomic system: '<Root>/C' */
65
66     /* Gain: '<Root>' 39 */
67     rtb_E = gaindelay_E;
68     /* UnitDelay: '<S1>/Delay' */
69     rtb_E += gaindelay_B.Delay = gaindelay_DWork.Delay_DSTATE;
70
71     /* Sum: '<Root>/A' */
72     /* Constant: '<Root>/A' */
73
74     gaindelay_B.B = gaindelay_B.A_Value + rtb_E;
75 }

```

Output execution list:

0:0 C(op)
0:0 Delay(op)
0:1 A(op)
0:2 E(op)
0:3 B(op)

Update execution list:

0:0 C(up)
0:0 Gain(op)
0:1 Delay(up)

41

```

44 /* Update for atomic system: '<Root>/C' */
45 void gaindelay_C_Update(void)
46 {
47     real_T rtb_Gain;
48
49     /* Gain: '<S1>/Gain' */
50     rtb_Gain = gaindelay_B.Gain * gaindelay_B.B;
51
52     /* Update for UnitDelay: '<S1>/Delay' */
53     gaindelay_BWork.Delay_DSTATE = rtb_Gain;
54 }

```

Output execution list:

0:0 C(op)
0:0 Delay(op)
0:1 A(op)
0:2 E(op)
0:3 B(op)

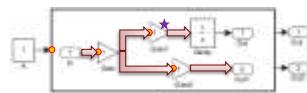
Update execution list:

0:0 C(up)
0:0 Gain(op)
0:1 Delay(up)

42

The general analysis

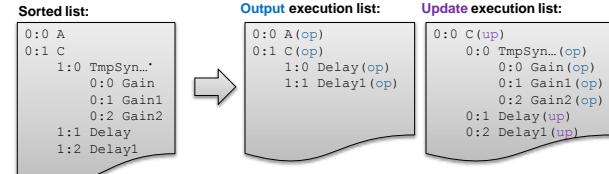
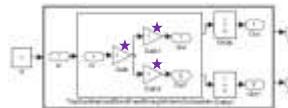
- For an input block
 - Depth-first search for a **df** path to output
 - If a port **without df** is reached, mark visited nodes for potential move of output method
- If output found
 - Clear **all** blocks visited from the initial input port



43

The synthesis part of the algorithm

- Move **marked df** blocks into an atomic subsystem

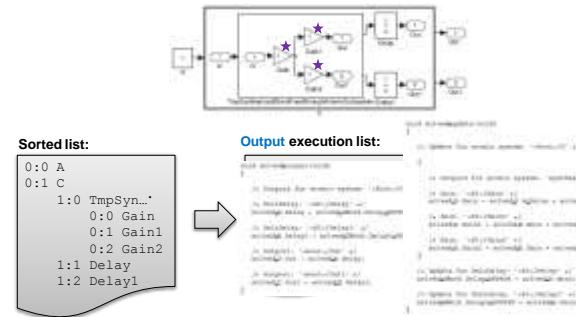


44



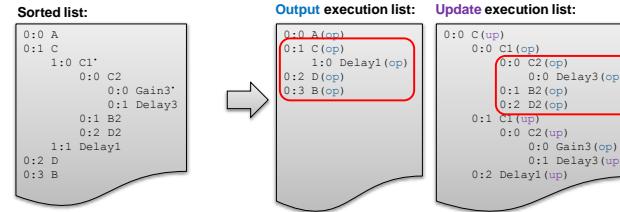
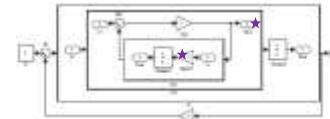
The synthesis part of the algorithm

- Move marked df blocks into an atomic subsystem



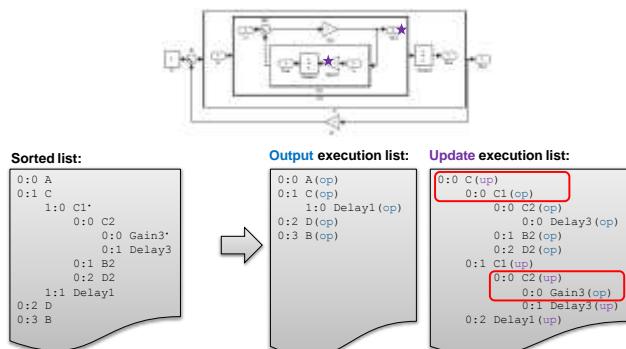
45

Nested cyclic dependencies



46

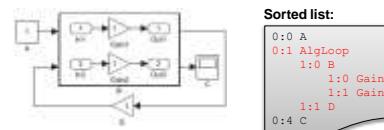
Nested cyclic dependencies



47

Scope of applicability

- Applies to **derivative** and **zerocrossing** as well
- But, the **df** path has to be breakable



- In general, four basic data dependency classes



48

Agenda

- Simulink preliminaries
- Executing a Simulink model
- Dealing with hierarchy
- Heterogeneous composition
- ▶ An implementation in Simulink
- Conclusions

Conclusions

- Simulink as a network of dynamic systems
 - Method set to generate behavior (output, update, ...)
- Lists for the execution phases
 - `df` to sort according to data dependencies
 - Block methods and sample times to create lists
- Atomic subsystems for componentization
 - Heterogeneous composition to avoid dependencies
 - Heterogeneous execution lists (flat and hierarchical)
 - Scoped the class of systems where this applies

49

50

References

Pieter J. Mosterman and John E. Ciolfi, "[Using Interleaved Execution to Resolve Cyclic Dependencies in Time-Based Block Diagrams](#)," in *Proceedings of 43rd IEEE Conference on Decision and Control (CDC'04)*, pp. 4057-4062, December 14 - 17, Atlantis, Paradise Island, Bahamas, 2004

Ben Denckla and Pieter J. Mosterman, "[An Intermediate Representation and Its Application to the Analysis of Block Diagram Execution](#)," in *Proceedings of the 2004 Summer Computer Simulation Conference (SCSC'04)*, pp. 167-172, July 25 - 29, San Jose, CA, 2004

<http://msdl.cs.mcgill.ca/people/mosterman/publications.html>

51

Acknowledgments

John E. Ciolfi
MathWorks

Ben Denckla
Independent Thinker

Many thanks for their continuing collaboration!

52

