Type Inference of Simulink Hierarchical Block Diagrams in Isabelle

Viorel Preoteasa[1]

joint work with Iulia Dragomir[2] and Stavros Tripakis[1,3]

[1]Aalto University, Finland
[2]Verimag, France
[3]University of California, Berkeley, USA

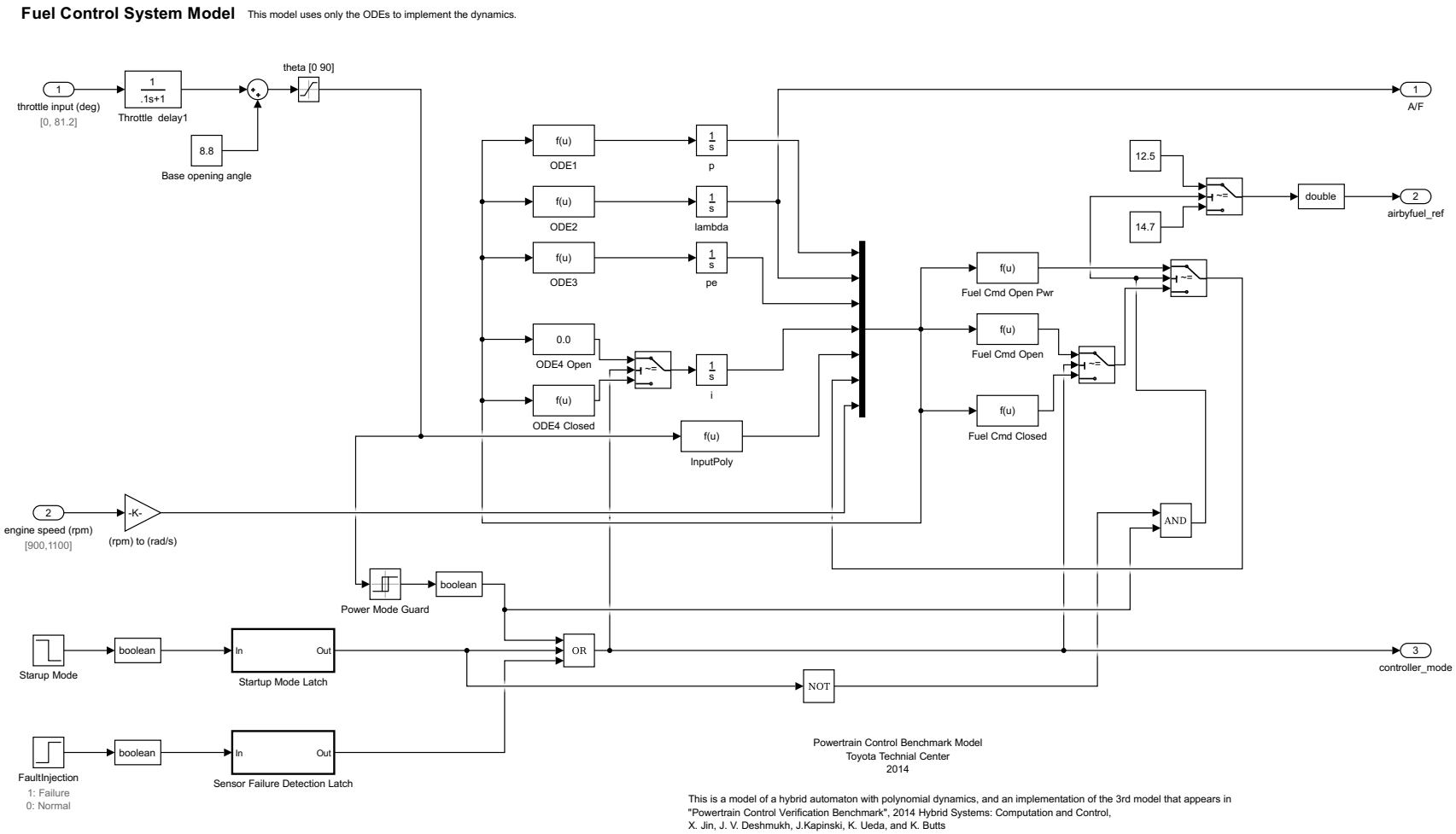
June 19, 2017

Overview

- About Simulink
- Refinement Calculus of Reactive Systems
- Simulink to RCRS
- Constant Blocks
- Generic Translation
- Conclusions

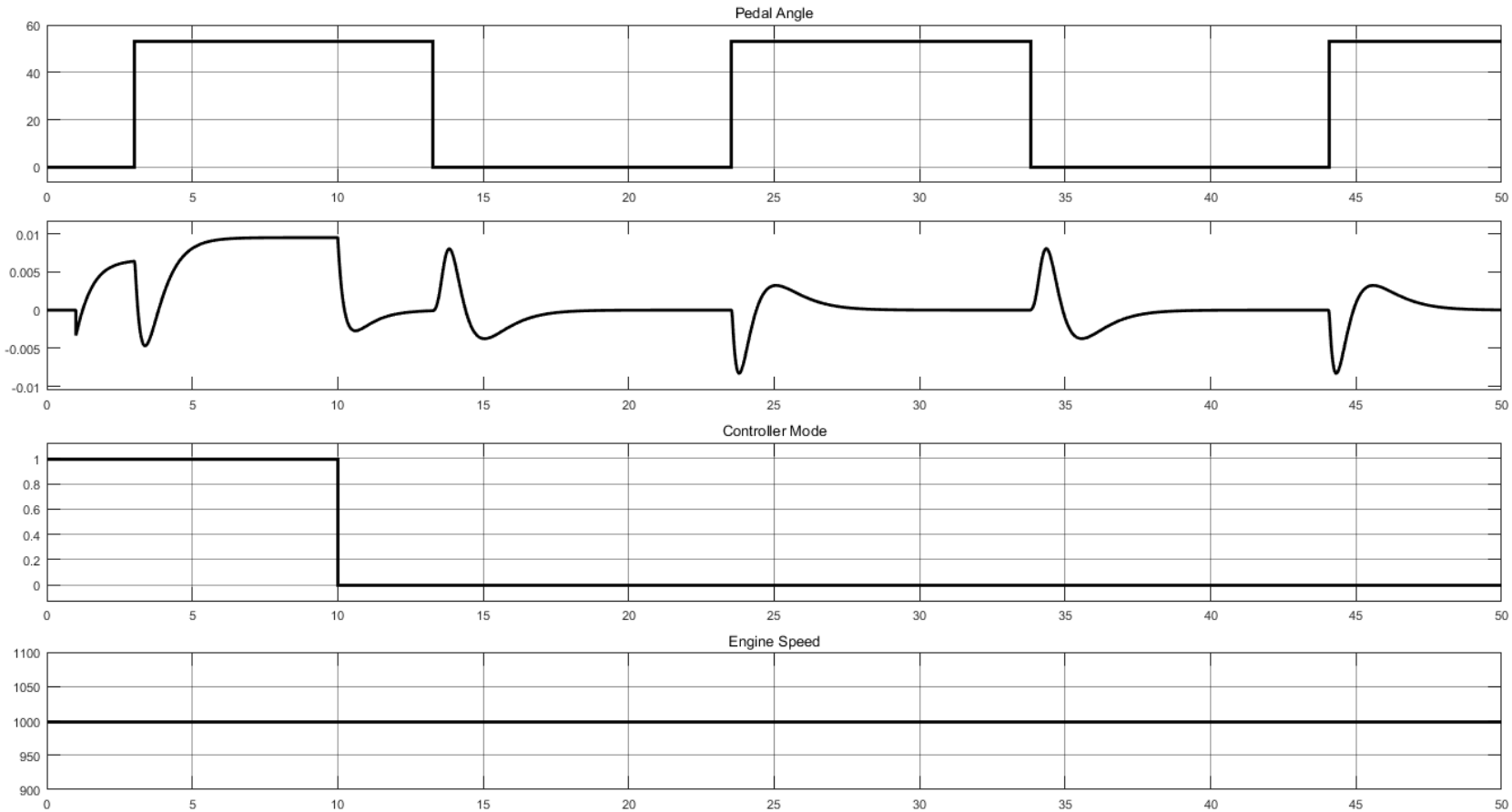
Simulink

Simulink is a tool for modeling and simulating cyber-physical systems



Simulink

One can perform simulations of the system and display signals of interest.



Refinement Calculus of Reactive Systems (RCRS)

- We represent Simulink diagrams as *monotonic predicate transformers*
- Monotonic predicate transformers map sets of output values into sets of input values
- This allows modeling different aspects about Simulink
  - non-deterministic specifications based on LTL, including liveness properties
  - refinement to show that a diagram is refinement of a specification.
  - systems that are not input-receptive – they cannot respond to all possible values of the inputs.
- Implementation: RCRS framework
  - http://rcrs.cs.aalto.fi/

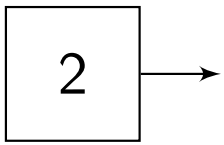
Simulink to RCRS

- We use two basic predicate transformers
  - assert $\{x, y \mid x + y > 0\}$
    - if inputs x, y are such that $x + y > 0$ then the output is unchanged x, y, otherwise this component *fails*
  - update $[x, y \rightsquigarrow x + y, x - y, x \cdot y]$
    - for input x, y the output is the tuple $(x + y, x - y, x \cdot y)$
- Three composition operations
  - Serial composition $S \circ T$
  - Parallel composition $S \parallel T$
  - Feedback composition feedback(S)

Simulink to RCRS

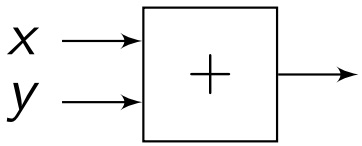
Basic blocks

- Constant



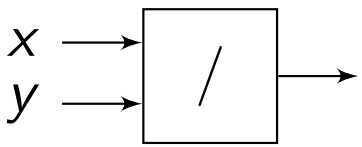
$$\text{Const} = [() \rightsquigarrow 2]$$

- Addition



$$\text{Add} = [x, y \rightsquigarrow x + y]$$

- Division

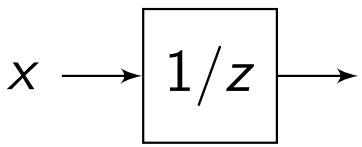


$$\text{Div} = \{x, y \mid y \neq 0\} \circ [x, y \rightsquigarrow x/y]$$

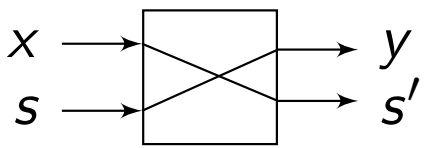
Simulink to RCRS

Basic blocks with state

- Unit delay



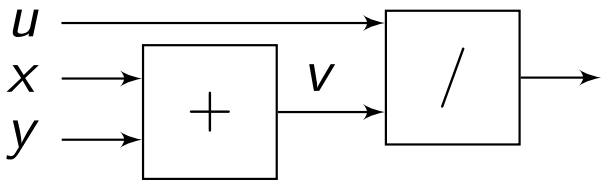
$$\text{UnitDelay} = [x, s \rightsquigarrow s, x]$$



$$s' := x;\ y := s$$

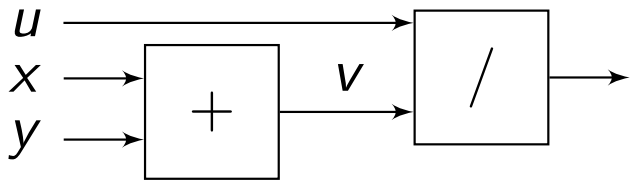
For initial state $s_0 = 0$, and input $x_0, x_1, \ldots$, output is $0,\ x_0,\ x_1, \ldots$.

Simulink to RCRS – Compositions



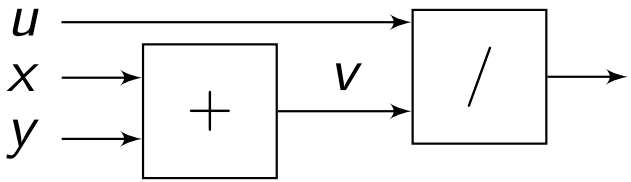
$$
\begin{aligned}
& (\mathsf{Id} \parallel \mathsf{Add}) \circ \mathsf{Div} \\
= \quad & \\
& ([u \rightsquigarrow u] \parallel [x, y \rightsquigarrow x + y]) \circ \{u, v \mid v \neq 0\} \circ [u, v \rightsquigarrow u/v] \\
= \quad & \\
& \{u, x, y \mid x + y \neq 0\} \circ [u, x, y \rightsquigarrow u/(x + y)]
\end{aligned}
$$

Typing



- We like to treat every block *locally*, and have Isabelle *infer* the typing of the diagram.
- If there are no explicit types in the diagram we do not want to introduce them, but use *type variables* instead.
- By default in Isabelle the basic blocks have some typing
  - Add : $'a \times 'a \xrightarrow{\circ} 'a \;\; (= ('a \to \text{bool}) \to ('a \times 'a \to \text{bool}))$
  - Id : $'b \xrightarrow{\circ} 'b$
  - Div : $'c \times 'c \xrightarrow{\circ} 'c$
  - (Id $\parallel$ Add) : $'b \times 'a \times 'a \xrightarrow{\circ} 'b \times 'a$
  - (Id $\parallel$ Add) $\circ$ Div : $'a \times 'a \times 'a \xrightarrow{\circ} 'a$

Typing



- $A = (\mathrm{Id} \parallel \mathrm{Add}) \circ \mathrm{Div} : 'a \times 'a \times 'a \xrightarrow{\circ} 'a$
- In this example the inferred type is $'a \times 'a \times 'a \xrightarrow{\circ} 'a$ where $'a$ is a type variable.
- We can instantiate $'a$ with different concrete types int, real, nat, ...
  - $A : \mathrm{int} \times \mathrm{int} \times \mathrm{int} \xrightarrow{\circ} \mathrm{int}$
  - $A : \mathrm{real} \times \mathrm{real} \times \mathrm{real} \xrightarrow{\circ} \mathrm{real}$
  - $A \circ [x \rightsquigarrow \mathrm{Suc}(x)] : \mathrm{nat} \times \mathrm{nat} \times \mathrm{nat} \xrightarrow{\circ} \mathrm{nat}$ – here the type inference instantiate $'a$ with nat because Suc is a function on natural numbers.

Challenges

- Typing of atomic components is based on type variables, and Isabelle infers the types of the compositions.
- When specific types are used in the diagram, we use the corresponding Isabelle type.
- All works well for many Simulink diagrams.
- However, since Simulink is not as strongly typed as Isabelle, there are problems for some diagrams.
- Next we will explore some of these problems and their solutions.
- More details and more problems are discussed in the paper.

Constant Blocks

Constant blocks are the simplest blocks in Simulink, however they require a careful implementation.

- Isabelle numerical constants (2, 3, 3.5) are polymorphic
- By default a numerical constant in a term has the type $'a$.
- Numerical types (nat, int, real) are disjoint in Isabelle.
- To use numerical constants of type nat, for example:
  - we must specify the type explicitly 3 : nat
  - or we must use the constant in a context where a nat is expected Suc(3).

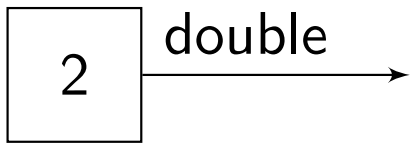
Constant Blocks

The basic ideas of implementing a constant block are

- Constant block that specify an output type are implemented as such.
  - This results in simple and intuitive implementation.
- Constant blocks that do not specify an output type are implemented using type variables.
  - Some details need to be considered.

Constant Blocks with Output Type

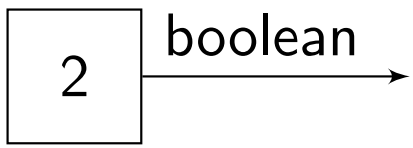
Constant block of type double:



$$\text{Const} = [() \rightsquigarrow 2 : \text{real}]$$

- Double is translated into the type real of Isabelle.

Constant block of type Boolean:

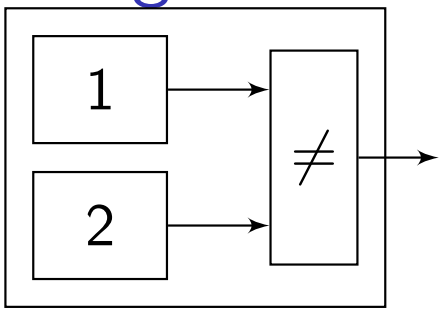


$$\text{Const} = [() \rightsquigarrow 2 : \text{bool}]$$

- The numerical constants are not available for the type bool.
- So, we need some additional definitions:

$$\begin{aligned} 0 : \text{bool} &:= \text{false} \\ 1 : \text{bool} &:= \text{true} \\ 2 : \text{bool} &:= \text{true} \\ &\ldots \end{aligned}$$

Challenge 1: Constant Blocks without Output Type



Compare

We try to use the same approach, but without types:

- ConstA $= [() \rightsquigarrow 1]$, and ConstA : unit $\xrightarrow{\circ} 'a$
- ConstB $= [() \rightsquigarrow 2]$, and ConstB : unit $\xrightarrow{\circ} 'b$
- unit is the singleton type of empty tuple ().

Compare is defined by

- Compare $=$ (ConstA $\parallel$ ConstB) $\circ [x, y \rightsquigarrow x \neq y]$ and Compare : unit $\xrightarrow{\circ}$ bool

After simplifications:

$$\text{Compare} = [() \rightsquigarrow 1 : 'a \neq 2 : 'a] : \text{unit} \xrightarrow{\circ} \text{bool}$$

Challenge 1: Constant Blocks without Output Type

The definition:

$$\text{Compare} := [() \rightsquigarrow 1 : 'a \neq 2 : 'a] : \text{unit} \xrightarrow{\circ} \text{bool}$$

is inconsistent in Isabelle, and it is not accepted.
$'a$ is "trapped" inside the definition, i.e. $'a$ is not part of the type of Compare

$$\begin{array}{ccccc} \text{Compare} & = & [() \rightsquigarrow 1 : \text{nat} \neq 2 : \text{nat}] & = & [() \rightsquigarrow \text{true}] \\ \| & & & & \not\parallel \\ \text{Compare} & = & [() \rightsquigarrow 1 : \text{bool} \neq 2 : \text{bool}] & = & [() \rightsquigarrow \text{false}] \end{array}$$

Challenge 1: Constant Blocks without Output Type

The solution is to add parameters to constant blocks that will record the type of the constant

$$\begin{aligned} \mathsf{ConstA}(u : 'a) &:= [() \rightsquigarrow 1 : 'a] \\ \mathsf{ConstB}(v : 'b) &:= [() \rightsquigarrow 2 : 'b] \\ \mathsf{Compare}(u, v) &:= (\mathsf{ConstA}(u) \parallel \mathsf{ConstB}(v)) \circ [x, y \rightsquigarrow x \neq y] \end{aligned}$$

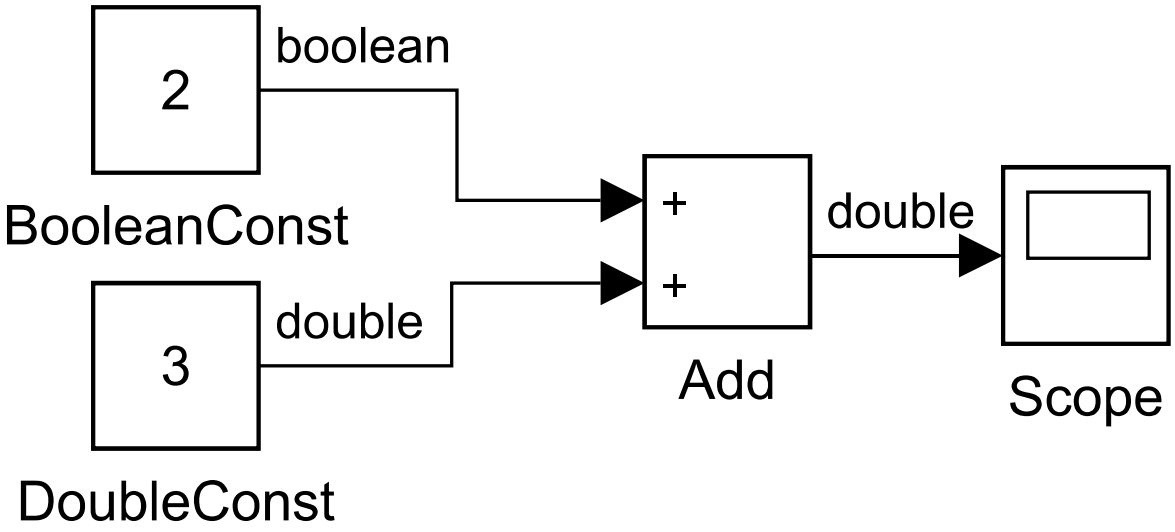
Compare has the type

$$\mathsf{Compare} : 'a \times 'a \to (\mathsf{unit} \xrightarrow{\circ} \mathsf{bool})$$

$$\mathsf{Compare}(u : \mathsf{bool}, v) \neq \mathsf{Compare}(s : \mathsf{nat}, t)$$

Challenge 2: Mixing Boolean and Double Values

Next diagram is accepted by Simulink.



- Add block has the output explicitly specified to be double
- The constant 2 has the output specified to be Boolean
- The constant 3 has the output specified to be double

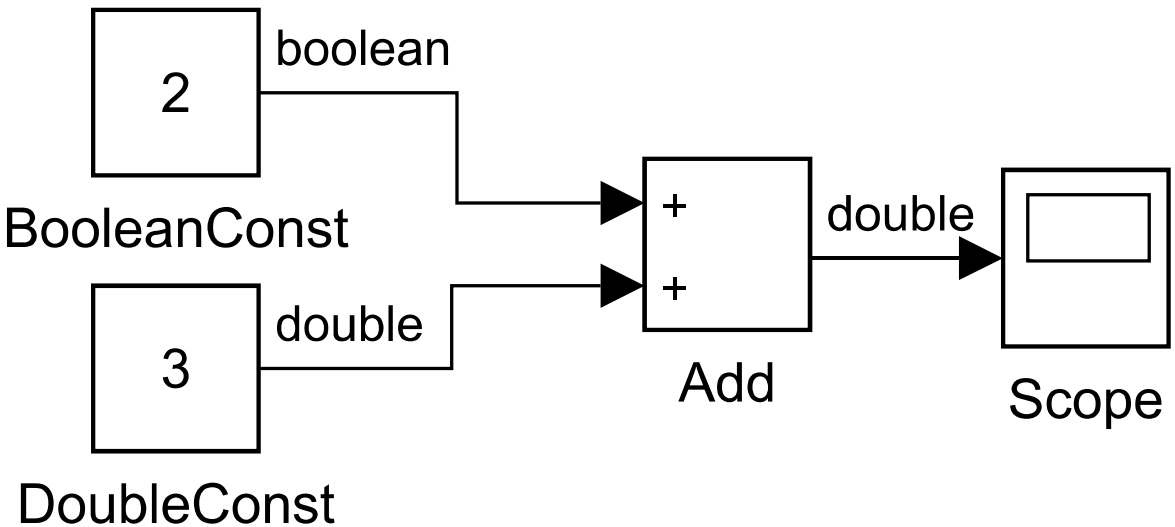
The constant 2 is converted to Boolean true (as $2 \neq 0$), and then true is converted again to double 1.

In Simulink this diagram outputs $3 + 1 = 4$.

Challenge 2: Mixing Boolean and Double Values

The techniques described so far cannot handle this diagram because Boolean values are mixed with numerical values.

Isabelle gives a type mismatch error on the translation.

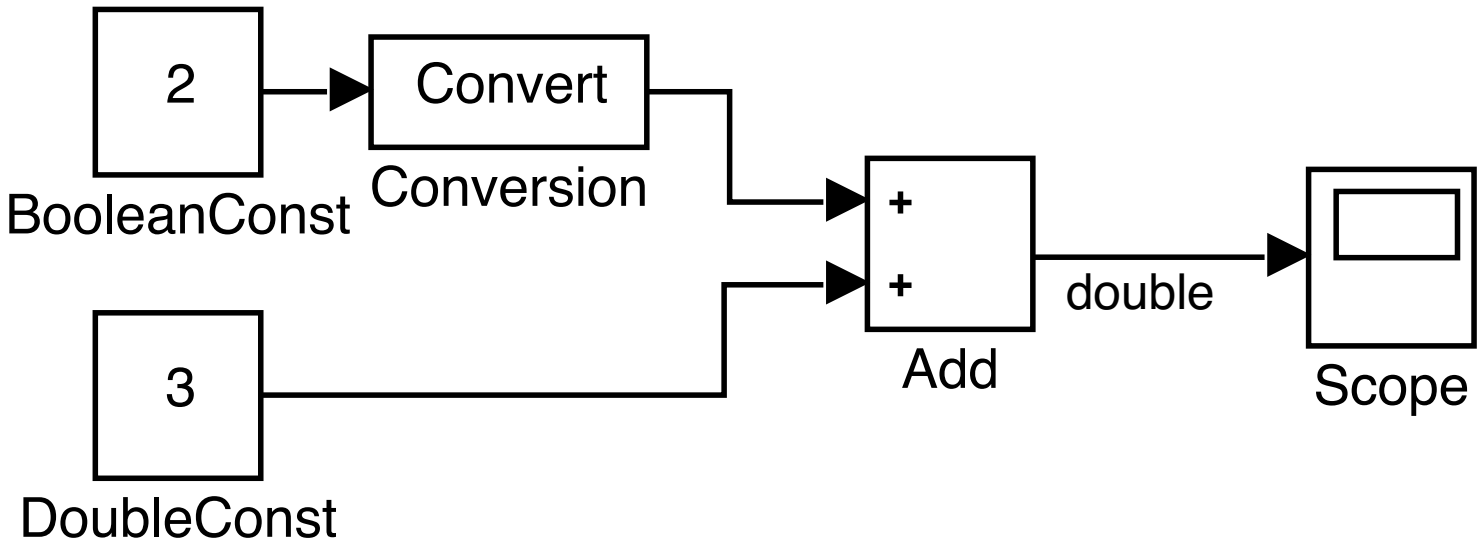


However, we want to be able to accept also this kind of diagrams.

Challenge 2: Solution I – Developer Interaction

Developer Interaction

- This case may be regarded as bad design, and the developer could add conversions



We don't need a conversion with explicit output type:

- Input of the conversion is Boolean because of the Boolean constant block
- Output is real because the Add block requires real.

Challenge 2: Solution II – Generic Translation

Generic Translation

- Implement Boolean constants and operations using numerical constant and operations.
- "Generic" – we use type variables.
- If the diagram has logical parts, not mixed with numbers, they can be preserved as logical.
- Details in the paper.

Conclusions

- Implemented in our RCRS tool.
- Our technique is compositional and modular.
- The treatment of atomic blocks is local. We do not need to know anything about the context of the blocks.
- This is important for expanding the library of blocks supported by our tool.
- We applied our translation to a fairly large (#70 basic blocks) industrial Simulink model provided by Toyota.
- Many of the cases discused in the paper are generalizations of problems discovered in the Toyota model.
- Althought the translation itself is quite involved, the result of the translation is intuitive, especially after simplifications.
- Using the Isabelle translation, we can easily translate the Simulink model into Python (for simulation and testing), Z3 (for verification), ...