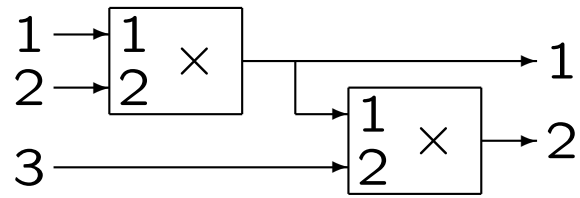
Towards an executable denotational semantics for causal block diagrams

Ben Denckla – Denckla Consulting
Pieter J. Mosterman – The MathWorks
Hans Vangheluwe – McGill University

What is the meaning of 

?

We'd like it to be $\lambda(x, y, z) \to (x \times y, x \times y \times z)$.

E.g., $(3, 4, 5) \to (12, 60)$.

How can we formalize these semantics?

We formalize by cascading two translational semantics and a traditional denotational semantics.

block diagram language

$\downarrow$ (translation)

BdAppLang

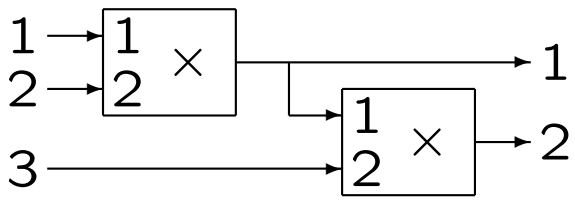
$\downarrow$ (translation)

AppLang

$\downarrow$ (denotation)

Haskell

block diagram language to BdAppLang translation



↓

diagram blocks

name	**function**	**inputs**	
mula	×	1	2
mulb	×	mula	3

diagram outputs

mula
mulb

BdAppLang to AppLang translation

diagram blocks

name	**function**	**inputs**	
mula	$\times$	1	2
mulb	$\times$	mula	3

diagram outputs

mula
mulb

$$\downarrow$$

$\lambda toplevel \rightarrow$
  letrec
    $mula = \times\ (toplevel_1, toplevel_2)$
    $mulb = \times\ (mula, toplevel_3)$
  in $(mula, mulb)$

AppLang to Haskell denotation

$\lambda toplevel \rightarrow$
  letrec
    $mula = \times\ (toplevel_1, toplevel_2)$
    $mulb = \times\ (mula, toplevel_3)$
  in $(mula, mulb)$

$\downarrow$ (abbreviated)

$m\ [\![\lambda i \rightarrow x]\!]\ e = \lambda v \rightarrow m\ x\ (ue\ i\ v\ e)$
$m\ [\![f\ a]\!]\ e = (m\ f\ e)\ (m\ a\ e)$
$m\ [\![\text{letrec}\ ds\ \text{in}\ x]\!]\ e = m\ x\ (\mathit{fix}\ \lambda e' \rightarrow ue\ (md\ ds\ e')\ e)$
$m\ [\![(a, b)]\!]\ e = (m\ a\ e, m\ b\ e)$
$m\ [\![i]\!]\ e = e\ i$
$m\ [\![\text{int}\ n]\!]\ e = n$

I've implemented this all in Haskell.

$$
\begin{array}{l}
Dia\ blockList\ outputList \\
\quad \text{where} \\
\quad blockList = [\\
\qquad Bld\ \text{``}mula\text{''}\ tupleMul\ [Din\ 0, Din\ 1], \\
\qquad Bld\ \text{``}mulb\text{''}\ tupleMul\ [Blo\ \text{``}mula\text{''}\ 0, Din\ 2]] \\
\quad outputList = [Blo\ \text{``}mula\text{''}\ 0, Blo\ \text{``}mulb\text{''}\ 0]
\end{array}
$$

If bd is the BdAppLang block diagram above, trd translates from BdAppLang to AppLang, and $mPro$ gives the meaning of a program in AppLang,

$$mPro\ (App\ (trd\ bd)\ (tixn\ [3, 4, 5])) = (12, 60)$$

But this is all kind of boring!

Instead of just arithmetic, we want to do signal processing, controls, and continuous-time simulation via numerical approximation of ODEs!

We can!

E.g., feedback already supported via fixed-point semantics.

With lazy lists (streams) and implicit state added, lots of applications are supported.

But it is all based on the small, clean semantics presented here with a boring arithmetic example.