Circuit timing analysis, linear maps, and semantic morphisms

Conal Elliott

Tabula

IFIP WG 2.8, Nov. 2012

Outline

Timing analysis

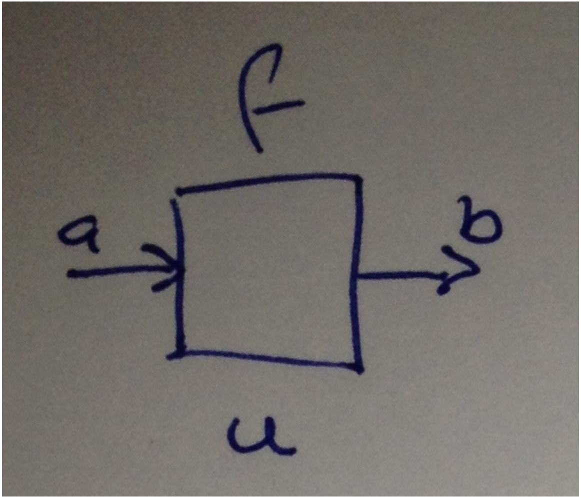
Linear transformations

Semantics and implementation

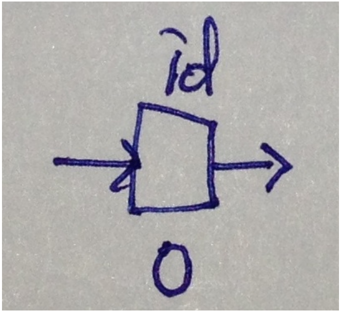
Timing analysis

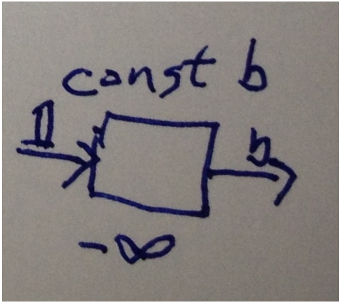
Simple timing analysis

Computation with a time delay:

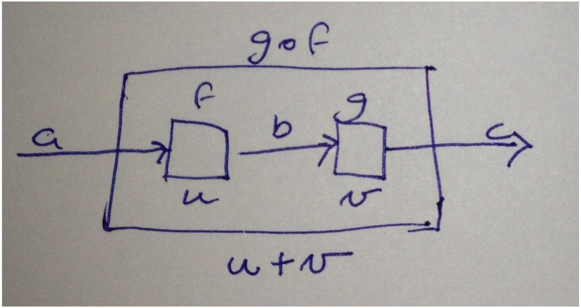


Trivial timings

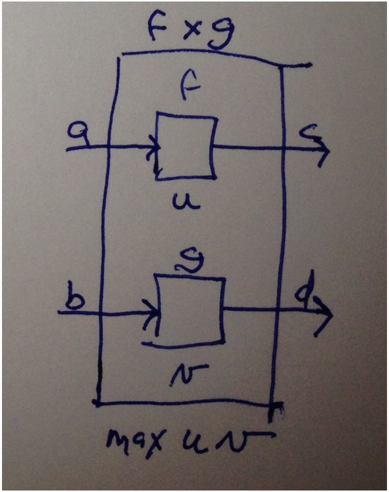




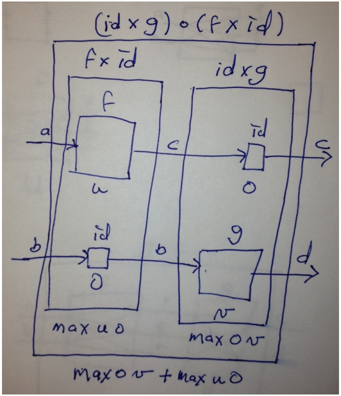
Sequential composition

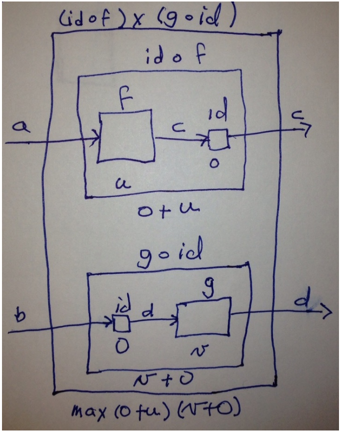


Parallel composition



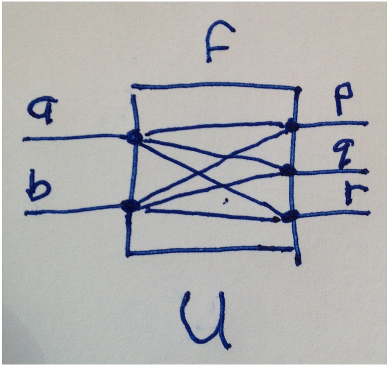
But ...





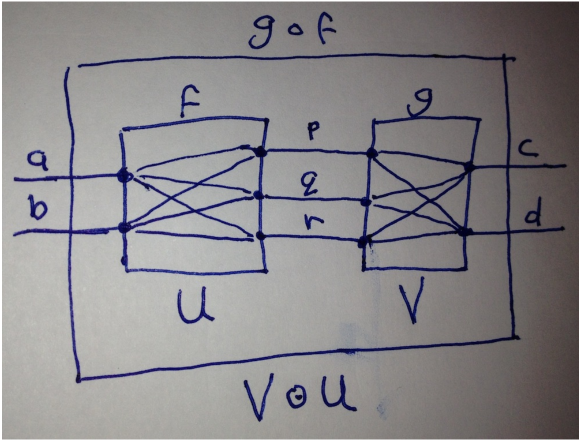
Oops: Same circuit $(f \times g)$, different timings.

Multi-path analysis

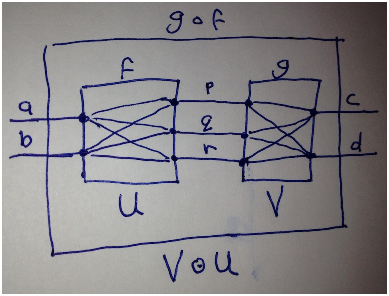


- Max delay for *each* input/output pair
- How do delays *compose*?

How do delays *compose*?



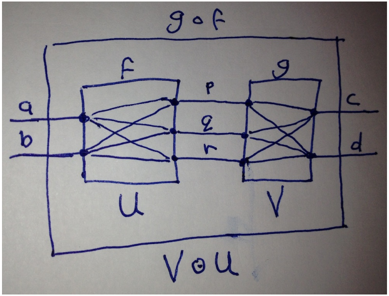
How do delays *compose*?



$V \odot U = W$, where

$$W_{i,k} = \operatorname*{Max}_j \left(U_{i,j} + V_{j,k}\right)$$

How do delays *compose*?

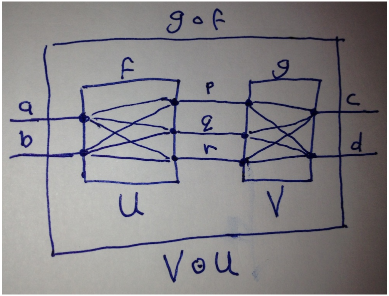


$V \odot U = W$, where

$$W_{i,k} = \operatorname*{Max}_j \left(U_{i,j} + V_{j,k}\right)$$

Look familiar?

How do delays *compose*?



$V \odot U = W$, where

$$W_{i,k} = \operatorname*{Max}_{j} \left(U_{i,j} + V_{j,k} \right)$$

Look familiar? Matrix multiplication?

MaxPlus algebra

type $Delay = MaxPlus\ Double$

data $MaxPlus\ a = MP\ a$

instance $Ord\ a \Rightarrow AdditiveGroup\ (MaxPlus\ a)$ **where**
  $MP\ a \mathbin{\hat{+}} MP\ b = MP\ (a\ \text{`}max\text{`}\ b)$

instance $(Ord\ a, Num\ a) \Rightarrow VectorSpace\ (MaxPlus\ a)$ **where**
  type $Scalar\ (MaxPlus\ a) = a$
  $a \cdot MP\ b = MP\ (a + b)$

Oops – We also need a zero.
VectorSpace is overkill. Module over a semi-ring suffices.

MaxPlus algebra

$$\textbf{type } Delay = MaxPlus\ Double$$

$$\textbf{data } MaxPlus\ a = -\infty \mid Fi\ a$$

$$\textbf{instance } Ord\ a \Rightarrow AdditiveGroup\ (MaxPlus\ a)\ \textbf{where}$$
$$0 = -\infty$$
$$MP\ a \mathbin{\hat{+}} MP\ b = MP\ (a \text{ `}max\text{` } b)$$
$$-\infty \mathbin{\hat{+}} \_ = -\infty$$
$$\_ \mathbin{\hat{+}} -\infty = -\infty$$

$$\textbf{instance } (Ord\ a, Num\ a) \Rightarrow VectorSpace\ (MaxPlus\ a)\ \textbf{where}$$
$$\textbf{type } Scalar\ (MaxPlus\ a) = a$$
$$a \cdot MP\ b = MP\ (a + b)$$
$$\_ \cdot -\infty = -\infty$$

Linear transformations

Representation?

How might we represent linear maps/transformations $a \multimap b$?

Representation?

How might we represent linear maps/transformations $a \multimap b$?

- Matrices
- Functions
- What else?

Matrices

$$\begin{bmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nm} \end{bmatrix}$$

Static typing?

Statically sized matrices

$$\textbf{type}\ Mat\ m\ n\ a = Vec\ m\ (Vec\ n\ a)$$

$$(\circ) :: (IsNat\ m, IsNat\ o) \Rightarrow$$
$$\quad Mat\ n\ o\ D \rightarrow Mat\ m\ n\ D \rightarrow Mat\ o\ m\ D$$
$$no \circ mn = crossF\ dot\ (transpose\ no)\ mn$$

$$crossF :: (IsNat\ m, IsNat\ o) \Rightarrow$$
$$\quad (a \rightarrow b \rightarrow c) \rightarrow Vec\ o\ a \rightarrow Vec\ m\ b \rightarrow Mat\ o\ m\ c$$
$$crossF\ f\ as\ bs = (\lambda a \rightarrow f\ a \mathbin{<\$>} bs) \mathbin{<\$>} as$$

$$dot :: (Ord\ a, Num\ a) \Rightarrow$$
$$\quad Vec\ n\ a \rightarrow Vec\ n\ a \rightarrow a$$
$$u\ \text{`}dot\text{`}\ v = sum\ (zipWithV\ (*)\ u\ v)$$

Generalizing

$$\textbf{type}\ Mat\ m\ n\ a = m\ (n\ a)$$

$$\begin{aligned}&(\circ) :: (Functor\ m, Applicative\ n, Traversable\ n, Applicative\ o) \Rightarrow\\&\qquad Mat\ n\ o\ D \to Mat\ m\ n\ D \to Mat\ o\ m\ D\\&no \circ mn = crossF\ dot\ (sequenceA\ no)\ mn\end{aligned}$$

$$\begin{aligned}&crossF :: (Functor\ m, Functor\ o) \Rightarrow\\&\qquad\qquad (a \to b \to c) \to o\ a \to m\ b \to Mat\ o\ m\ c\\&crossF\ f\ as\ bs = (\lambda a \to f\ a \mathbin{<\$>} bs) \mathbin{<\$>} as\end{aligned}$$

$$\begin{aligned}&dot :: (Foldable\ n, Applicative\ n, Ord\ a, Num\ a) \Rightarrow\\&\qquad n\ a \to n\ a \to a\\&u\ `dot`\ v = sum\ (liftA2\ (*)\ u\ v)\end{aligned}$$

Represent via type family (old)

$$\textbf{class}\ \mathit{VectorSpace}\ v \Rightarrow \mathit{HasBasis}\ v\ \textbf{where}$$
$$\quad \textbf{type}\ \mathit{Basis}\ v :: *$$
$$\quad \mathit{coord} :: v \rightarrow (\mathit{Basis}\ v \rightarrow \mathit{Scalar}\ v)$$

Linear map as memoized function from basis:

$$\textbf{newtype}\ a \multimap b = L\ (\mathit{Basis}\ a \rightarrow^{M} b)$$

See *Beautiful differentiation* (ICFP 2009).

Represent as GADT

$$
\begin{aligned}
&\textbf{data}\ a \multimap b\ \textbf{where} \\
&\quad \mathit{Dot} :: \mathit{InnerSpace}\ b \Rightarrow \\
&\qquad\qquad b \to (b \multimap \mathit{Scalar}\ b) \\
&\quad (:\triangle) :: \mathit{VS}_3\ a\ c\ d \Rightarrow \qquad \text{-- vector spaces with same scalar field} \\
&\qquad\qquad (a \multimap c) \to (a \multimap d) \to (a \multimap c \times d)
\end{aligned}
$$

Semantics and implementation

Semantics

$$
\begin{aligned}
[\![\cdot]\!] &:: (a \multimap b) \to (a \to b) \\
[\![Dot\ b]\!] &= dot\ b \\
[\![f :\!\vartriangle g]\!] &= [\![f]\!] \vartriangle [\![g]\!]
\end{aligned}
$$

where, on functions,

$$(f \vartriangle g)\ a = (f\ a, g\ a)$$

Recall:

$$
\begin{aligned}
&\textbf{data}\ a \multimap b\ \textbf{where} \\
&\quad Dot :: InnerSpace\ b \Rightarrow b \to (b \multimap Scalar\ b) \\
&\quad (:\!\vartriangle) :: VS_3\ a\ c\ d \quad \Rightarrow (a \multimap c) \to (a \multimap d) \to (a \multimap c \times d)
\end{aligned}
$$

Semantic type class morphisms

Category instance specification:	*Arrow* instance specification:
$\llbracket id \rrbracket \equiv id$ $\llbracket g \circ f \rrbracket \equiv \llbracket g \rrbracket \circ \llbracket f \rrbracket$	$\llbracket f \vartriangle g \rrbracket \equiv \llbracket f \rrbracket \vartriangle \llbracket g \rrbracket$ $\llbracket f \times g \rrbracket \equiv \llbracket f \rrbracket \times \llbracket g \rrbracket$

where

$$
\begin{aligned}
(\vartriangle) &:: Arrow\ (\leadsto) \Rightarrow (a \leadsto c) \to (a \leadsto d) \to (a \leadsto c \times d) \\
(\times) &:: Arrow\ (\leadsto) \Rightarrow (a \leadsto c) \to (b \leadsto d) \to (a \times b \leadsto c \times d)
\end{aligned}
$$

The *Category* and *Arrow* laws then follow.

Deriving a *Category* instance

One case:

$$
\begin{aligned}
& [\![(f :\!\vartriangle g) \circ h]\!] \\
\equiv\ & ([\![f]\!] \vartriangle [\![g]\!]) \circ [\![h]\!] \\
\equiv\ & [\![f]\!] \circ [\![h]\!] \vartriangle [\![g]\!] \circ [\![h]\!] \\
\equiv\ & [\![f \circ h \vartriangle g \circ h]\!]
\end{aligned}
$$

(where $f \circ h \vartriangle g \circ h \equiv (f \circ h) \vartriangle (g \circ h)$). Uses:

$$(f \vartriangle g) \circ h \equiv f \circ h \vartriangle g \circ h$$

Implementation:

$$(f :\!\vartriangle g) \circ h = f \circ h :\!\vartriangle g \circ h$$

Deriving a *Category* instance

$$
\begin{aligned}
&[\![Dot\ s \circ Dot\ b]\!] \\
\equiv\ & dot\ s \circ dot\ b \\
\equiv\ & dot\ (s \cdot b) \\
\equiv\ & [\![Dot\ (s \cdot b)]\!]
\end{aligned}
$$

$$
\begin{aligned}
&[\![Dot\ (a, b) \circ (f :\!\vartriangle g)]\!] \\
\equiv\ & dot\ (a, b) \circ ([\![f]\!] \vartriangle [\![g]\!]) \\
\equiv\ & add \circ (dot\ a \circ [\![f]\!] \vartriangle dot\ b \circ [\![g]\!]) \\
\equiv\ & dot\ a \circ [\![f]\!] \mathbin{\hat{+}} dot\ b \circ [\![g]\!] \\
\equiv\ & [\![Dot\ a \circ f \mathbin{\hat{+}} Dot\ b \circ g]\!]
\end{aligned}
$$

Uses:

$$
\begin{aligned}
dot\ (a, b) &\equiv add \circ (dot\ a \times dot\ b) \\
(k \times h) \circ (f \vartriangle g) &\equiv k \circ f \vartriangle h \circ g \\
[\![f \mathbin{\hat{+}} g]\!] &\equiv [\![f]\!] \mathbin{\hat{+}} [\![g]\!]
\end{aligned}
$$

Deriving an *Arrow* instance

$$
\begin{aligned}
& [\![f \vartriangle g]\!] \\
\equiv\ & [\![f]\!] \vartriangle [\![g]\!] \\
\equiv\ & [\![f :\!\vartriangle g]\!]
\end{aligned}
$$

$$
\begin{aligned}
& [\![f \times g]\!] \\
\equiv\ & [\![f]\!] \times [\![g]\!] \\
\equiv\ & [\![f]\!] \circ \mathit{fst} \vartriangle [\![g]\!] \circ \mathit{snd} \\
\equiv\ & [\![\mathit{compFst}\ f]\!] \vartriangle [\![\mathit{compSnd}\ g]\!] \\
\equiv\ & [\![\mathit{compFst}\ f :\!\vartriangle \mathit{compSnd}\ g]\!]
\end{aligned}
$$

assuming

$$
\begin{aligned}
[\![\mathit{compFst}\ \ f]\!] &\equiv [\![f]\!] \circ \mathit{fst} \\
[\![\mathit{compSnd}\ g]\!] &\equiv [\![g]\!] \circ \mathit{snd}
\end{aligned}
$$

Composing with *fst* and *snd*

$$compFst :: VS_3\ a\ b\ c \Rightarrow a \multimap c \rightarrow a \times b \multimap c$$
$$compSnd :: VS_3\ a\ b\ c \Rightarrow b \multimap c \rightarrow a \times b \multimap c$$

Derivation:

$$dot\ a \circ fst \equiv dot\ (a, 0)$$
$$(f \vartriangle g) \circ fst \equiv f \circ fst \vartriangle g \circ fst$$

Implementation:

$$compFst\ (Dot\ a) = Dot\ (a, 0)$$
$$compFst\ (f :\vartriangle g) = compFst\ f \vartriangle compFst\ g$$

$$compSnd\ (Dot\ b) = Dot\ (0, b)$$
$$compSnd\ (f :\vartriangle g) = compSnd\ f \vartriangle compSnd\ g$$

Adding linear maps

$$
\begin{aligned}
& [\![Dot\ b \mathbin{\hat{+}} Dot\ c]\!] \\
\equiv\ & dot\ b \mathbin{\hat{+}} dot\ c \\
\equiv\ & dot\ (b \mathbin{\hat{+}} c) \\
\equiv\ & [\![Dot\ (b \mathbin{\hat{+}} c)]\!]
\end{aligned}
$$

$$
\begin{aligned}
& [\![(f :\!\vartriangle g) \mathbin{\hat{+}} (h :\!\vartriangle k)]\!] \\
\equiv\ & ([\![f]\!] \vartriangle [\![g]\!]) \mathbin{\hat{+}} ([\![h]\!] \vartriangle [\![k]\!]) \\
\equiv\ & ([\![f]\!] \mathbin{\hat{+}} [\![h]\!]) \vartriangle ([\![g]\!] \mathbin{\hat{+}} [\![k]\!]) \\
\equiv\ & [\![(f \mathbin{\hat{+}} h) \vartriangle (g \mathbin{\hat{+}} k)]\!]
\end{aligned}
$$

Other cases don't type-check.
Uses (on functions):

$$(f \vartriangle g) \mathbin{\hat{+}} (h \vartriangle k) \equiv (f \mathbin{\hat{+}} h) \vartriangle (g \mathbin{\hat{+}} k)$$

What next?

- Fancier timing analysis
- What else is linear?
- More examples of semantic type class morphisms