Composing Reo Connector Animations

José Proença   David de Oliveira Costa

Centrum voor Wiskunde en Informatica

CIC 2006

Outline

Outline

Motivation

- Simple REO connectors can have complex behaviour
- Formal Semantics exist:
  - Coinductive Calculus for Component Connectors;
  - Constraint Automata;
  - Connector Colouring;

Motivation

But

- Often connectors are not very *intuitive*
- Hard to design or debug

Enhance static representations of REO Connectors
$\Rightarrow$ using Animations.

Motivation

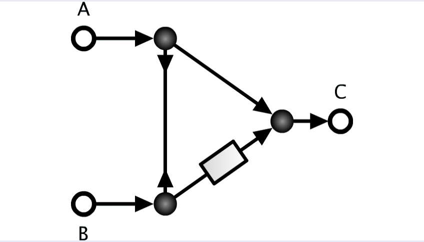
But

- Often connectors are not very *intuitive*
- Hard to design or debug

Enhance static representations of REO Connectors
$\Rightarrow$ using Animations.

Example
Ordering Connector

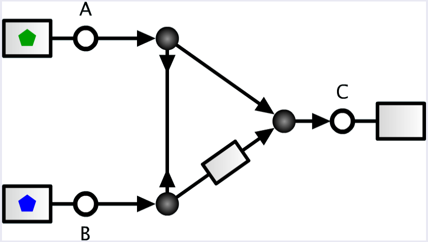
Connector



Example

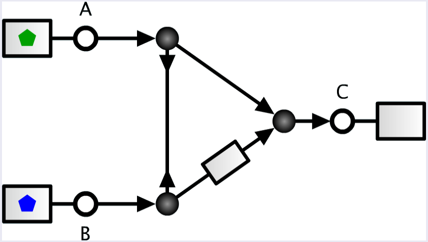
Ordering Connector

Connector with components

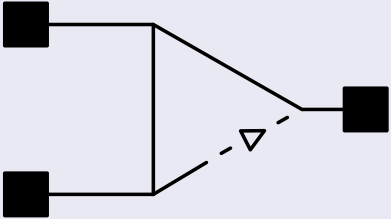


Example
Ordering Connector

Connector with components

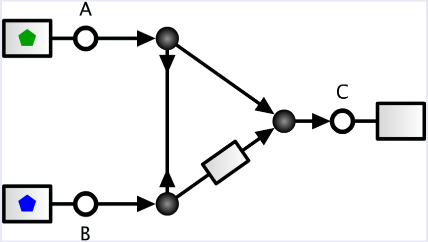


Colouring

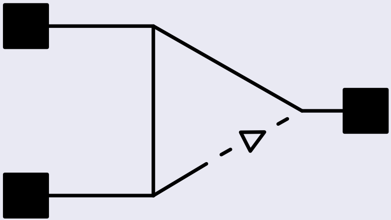


Example
Ordering Connector

Connector with components



Colouring



See animation

Graphical Elements

Graphical elements presented in animations:

- data
- synchrony
- exclusion
- buffering
- replication
- discrimination

Graphical Elements

Graphical elements presented in animations:

- data
- synchrony
- exclusion
- buffering
- replication
- discrimination

Tokens move over channels and components to represent dataflow.

Graphical Elements

Graphical elements presented in animations:

- data
- synchrony
- exclusion
- buffering
- replication
- discrimination

The part of the connector where dataflow occurs in a synchronous slice is coloured in red.

Graphical Elements

Graphical elements presented in animations:

- data
- synchrony
- exclusion
- buffering
- replication
- discrimination

A blue triangle represents the propagation of impossibility of dataflow.

Graphical Elements

Graphical elements presented in animations:

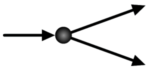
- data
- synchrony
- exclusion
- buffering
- replication
- discrimination

FIFO1 channels can buffer data, which is represented by placing the token inside the channel.

Graphical Elements

Graphical elements presented in animations:

- data
- synchrony
- exclusion
- buffering
- replication
- discrimination



Data (token) is replicated at each node with more than one output.

Graphical Elements

Graphical elements presented in animations:

- data
- synchrony
- exclusion
- buffering
- replication
- discrimination

The filter channel allows only the flow of some specific kind of data, which is represented by colouring the center with the same colour of tokens that are *allowed*.

Outline

The main idea

Using Domain Specific Languages (DSL)

- Simple DSL's to draw and animate connectors;
- Conversion to a standard animation tool.

We can go further:

- using the Connector Colouring semantics
- compositionally build the animations

The main idea

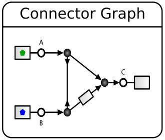
Using Domain Specific Languages (DSL)

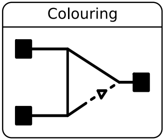
- Simple DSL's to draw and animate connectors;
- Conversion to a standard animation tool.

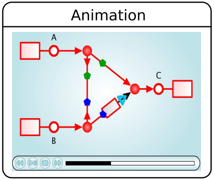
We can go further:

- using the Connector Colouring semantics
- compositionally build the animations

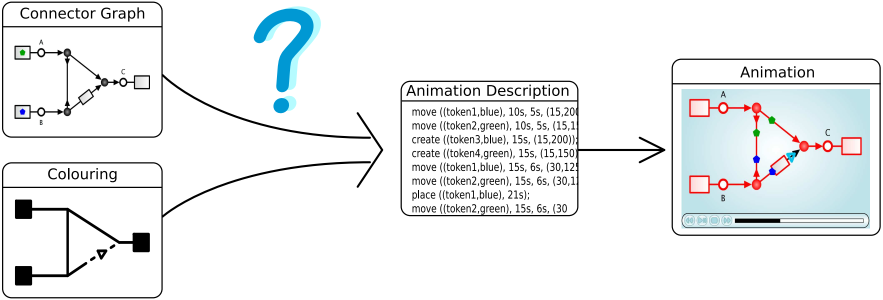
Generating Animations



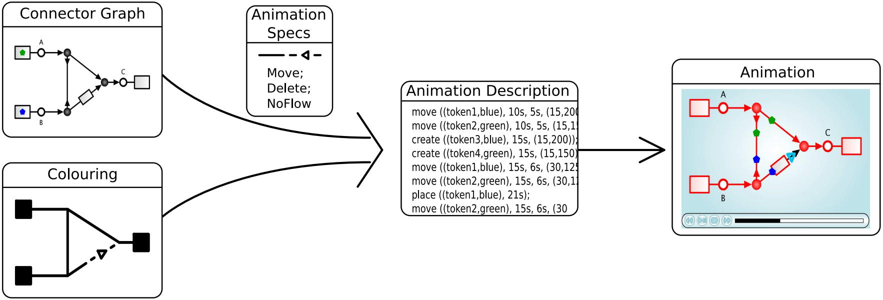




Generating Animations



Generating Animations



Outline

Animation Specifications
Definition

Let C be a connector:

$$\mathit{Colouring}_{\mathit{CT}(\mathrm{C})} \to \mathit{Set}\ [\mathit{ActionSpec}]$$

$$\begin{array}{rcl} \mathit{ActionSpec} & = & \mathit{Move\ Location\ Location} \\ & \mid & \mathit{Place\ Location} \\ & \mid & \mathit{Delete\ Location} \\ & \mid & \mathit{Create\ Location} \\ & \mid & \mathit{Copy\ Location} \\ & \mid & \mathit{NoFLow\ Location\ Location} \end{array}$$

Animation Specifications

Example

Synchronous Drain Channel

$(n_1^i, n_2^i)_{SyncDrain}$  $\{c_1 : ——, c_2 : - - \triangleright - -, c_3 : - - \triangleleft - -\}$

Animation Specification

$anim(c_1) = \{$ [Move `n1 SyncDrain`
, Delete `SyncDrain`
, Move `n2 SyncDrain`
, Delete `SyncDrain`] $\}$

$anim(c_2) = \{$ [NoFlow `n1 n2`] $\}$

$anim(c_3) = \{$ [NoFlow `n2 n1`] $\}$

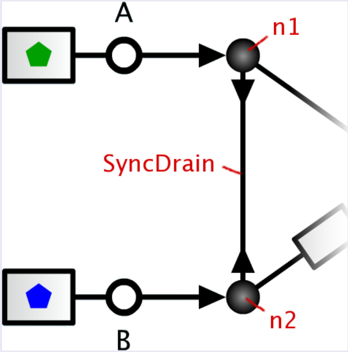
Animation Specifications
Example

Synchronous Drain Channel

$(n_1^i, n_2^i)_{SyncDrain}$  $\{c_1: \text{———}, c_2: - - \triangleright - -, c_3: - - \triangleleft - -\}$

Animation Specification

$anim(c_1) = \{ [$ Move `n1 SyncDrain`
, Delete `SyncDrain`
, Move `n2 SyncDrain`
, Delete `SyncDrain` $] \}$
$anim(c_2) = \{ [$ NoFlow `n1 n2` $] \}$
$anim(c_3) = \{ [$ NoFlow `n2 n1` $] \}$



Outline

Composing Animation Specifications

Lets look at an example

$(n_1^i, n_2^i)_{SyncDrain}$  $\{c_1 : \text{———}, c_2 : - - \triangleright - -, c_3 : - - \triangleleft - -\}$

$(m_1^i, m_2^o)_{Sync}$  $\{c_4 : \text{———}, c_5 : - - \triangleright - -, c_6 : - - \triangleleft - -\}$

Example

$anim(c_1) = \{$ [Move `n1 SyncDrain`
, Delete `SyncDrain`
, Move `n2 SyncDrain`
, Delete `SyncDrain`] $\}$

$anim(c_4) = \{$ [Move `m1 m2`] $\}$

$anim(c_7) = ?$

Composing Animation Specifications

Lets look at an example

$(n_1^i, n_2^i)_{SyncDrain}$  $\{c_1 : ——, c_2 : - - \triangleright - -, c_3 : - - \triangleleft - -\}$

$(m_1^i, m_2^o)_{Sync}$  $\{c_4 : ——, c_5 : - - \triangleright - -, c_6 : - - \triangleleft - -\}$

Example

 $c_7 : ——$

$anim(c_1) = \{ [$ Move `n1 SyncDrain`
, Delete `SyncDrain`
, Move `n2 SyncDrain`
, Delete `SyncDrain` $] \}$

$anim(c_4) = \{ [$ Move `m1 m2` $] \}$

$anim(c_7) = ?$

Composing Animation Specifications

Lets look at an example

$(n_1^i, n_2^i)_{SyncDrain}$  $\{c_1 : \text{———}, c_2 : - - \triangleright - -, c_3 : - - \triangleleft - -\}$

$(m_1^i, m_2^o)_{Sync}$  $\{c_4 : \text{———}, c_5 : - - \triangleright - -, c_6 : - - \triangleleft - -\}$

Example

 c_7 : ———

$anim(c_1)$ = { [Move `n1 SyncDrain`
, Delete `SyncDrain`
, Move `n2 SyncDrain`
, Delete `SyncDrain`] }

$anim(c_4)$ = { [Move `m1 m2`] }

$anim(c_7) = ?$

Composing Animation Specifications

Lets look at an example

$(n_1^i, n_2^i)_{SyncDrain}$  $\{c_1 : \text{———}, c_2 : - - \triangleright - -, c_3 : - - \triangleleft - -\}$

$(m_1^i, m_2^o)_{Sync}$  $\{c_4 : \text{———}, c_5 : - - \triangleright - -, c_6 : - - \triangleleft - -\}$

Example

 $c_7 : \text{————}$

$anim(c_1) = \{$ [Move `n1 SyncDrain`
, Delete `SyncDrain`
, Move `n2 SyncDrain`
, Delete `SyncDrain`] $\}$

$anim(c_4) = \{$ [Move `m1 m2`] $\}$

$\boxed{anim(c_7) = ?}$

Composing Animation Specifications

$$
\begin{aligned}
anim(c_7) &= anim(c_1 \cup c_4) \\
&= anim(c_1) \cup anim(c_4) \\
&= \{\, [\,\text{Move } \texttt{n1 SyncDrain} \\
&\quad ,\ \text{Delete } \texttt{SyncDrain} \\
&\quad ,\ \text{Move } \texttt{n2 SyncDrain} \\
&\quad ,\ \text{Delete } \texttt{SyncDrain}\,] \\
&\quad [\,\text{Move } \texttt{m1 m2}\,]\,\}
\end{aligned}
$$

General rule

$anim(c_1 \cup \ldots \cup c_n) = anim(c_1) \cup \ldots \cup anim(c_n)$

Composing Animation Specifications

$$
\begin{aligned}
anim(c_7) &= anim(c_1 \cup c_4) \\
&= anim(c_1) \cup anim(c_4) \\
&= \{\ [\ \text{Move}\ \texttt{n1 SyncDrain} \\
&\qquad ,\ \text{Delete}\ \texttt{SyncDrain} \\
&\qquad ,\ \text{Move}\ \texttt{n2 SyncDrain} \\
&\qquad ,\ \text{Delete}\ \texttt{SyncDrain}\ ] \\
&\qquad [\ \text{Move}\ \texttt{m1 m2}\ ]\ \}
\end{aligned}
$$

General rule

$anim(c_1 \cup \ldots \cup c_n) = anim(c_1) \cup \ldots \cup anim(c_n)$

Composing Animation Specifications

$$
\begin{aligned}
anim(c_7) &= anim(c_1 \cup c_4) \\
&= anim(c_1) \cup anim(c_4) \\
&= \{\, [\,\text{Move } \texttt{n1 SyncDrain} \\
&\qquad ,\ \text{Delete } \texttt{SyncDrain} \\
&\qquad ,\ \text{Move } \texttt{n2 SyncDrain} \\
&\qquad ,\ \text{Delete } \texttt{SyncDrain}\,] \\
&\qquad [\,\text{Move } \texttt{m1 m2}\,]\,\}
\end{aligned}
$$

General rule

$$anim(c_1 \cup \ldots \cup c_n) = anim(c_1) \cup \ldots \cup anim(c_n)$$

Outline

Animation Description

- Lower lever:

What tokens	When
How long	Where

- Multiple Steps, each derived from a colouring;
- Steps can be combined, by incrementing the When value;
- Each Animation Step:

  actions, absence of flow, changes to the environment

$$
\begin{array}{rcl}
\mathit{Action} & = & \mathit{move}\ (\mathit{Token}, \mathit{When}, \mathit{Duration}, \mathit{From}, \mathit{To}) \\
 & | & \mathit{place}\ (\mathit{Token}, \mathit{When}, \mathit{Where}) \\
 & | & \mathit{delete}\ (\mathit{Token}, \mathit{When}) \\
 & | & \mathit{create}\ (\mathit{Token}, \mathit{When}, \mathit{Where})
\end{array}
$$

Animation Description

- Lower lever:

  What tokens When
  How long Where

- Multiple Steps, each derived from a colouring;
- Steps can be combined, by incrementing the When value;
- Each Animation Step:

  actions, absence of flow, changes to the environment

$$
\begin{array}{rcl}
\mathit{Action} & = & \mathit{move}\ (\mathit{Token}, \mathit{When}, \mathit{Duration}, \mathit{From}, \mathit{To}) \\
& | & \mathit{place}\ (\mathit{Token}, \mathit{When}, \mathit{Where}) \\
& | & \mathit{delete}\ (\mathit{Token}, \mathit{When}) \\
& | & \mathit{create}\ (\mathit{Token}, \mathit{When}, \mathit{Where})
\end{array}
$$

Animation Description

- Lower lever:

What tokens	When
How long	Where

- Multiple Steps, each derived from a colouring;
- Steps can be combined, by incrementing the When value;
- Each Animation Step:

  actions, absence of flow, changes to the environment

$$
\begin{array}{lcll}
\mathit{Action} & = & \mathit{move} & (\mathit{Token}, \mathit{When}, \mathit{Duration}, \mathit{From}, \mathit{To}) \\
 & | & \mathit{place} & (\mathit{Token}, \mathit{When}, \mathit{Where}) \\
 & | & \mathit{delete} & (\mathit{Token}, \mathit{When}) \\
 & | & \mathit{create} & (\mathit{Token}, \mathit{When}, \mathit{Where})
\end{array}
$$

Animation Description

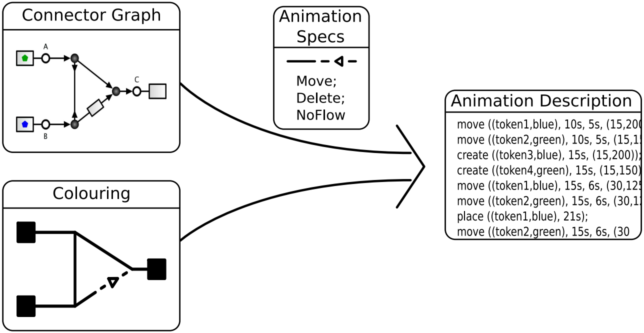
- Lower lever:

What tokens	When
How long	Where

- Multiple Steps, each derived from a colouring;
- Steps can be combined, by incrementing the When value;
- Each Animation Step:
  actions, absence of flow, changes to the environment

$$
\begin{array}{rcl}
Action & = & move\ (Token, When, Duration, From, To) \\
 & | & place\ (Token, When, Where) \\
 & | & delete\ (Token, When) \\
 & | & create\ (Token, When, Where)
\end{array}
$$

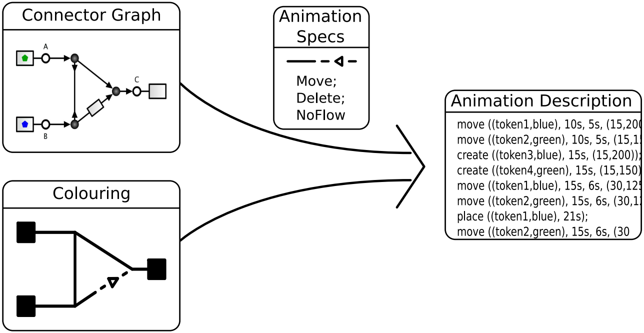
Obtaining Animation Descriptions



Missing
time information

Parameterised transformation on:
speed, fading time, time between steps

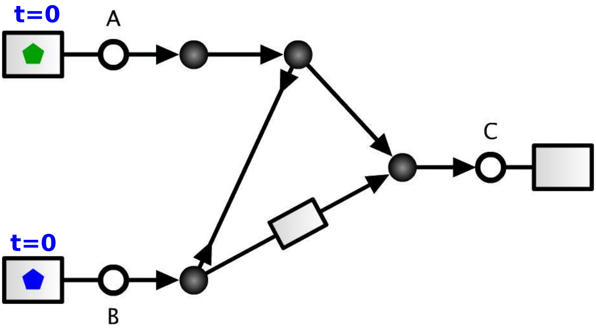
Obtaining Animation Descriptions



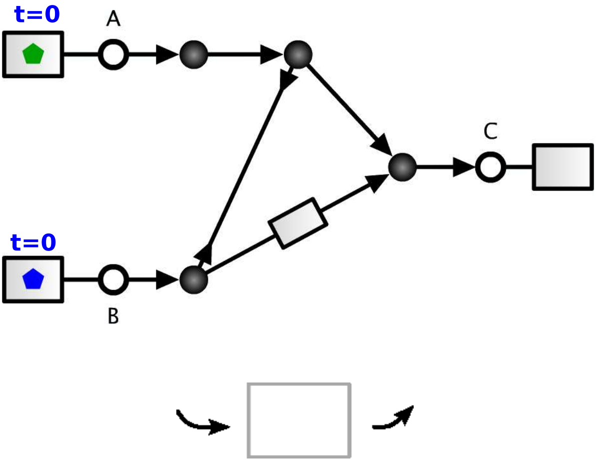
Missing
time information

Parameterised transformation on:
speed, fading time, time between steps

The Algorithm
A breath-first traversal

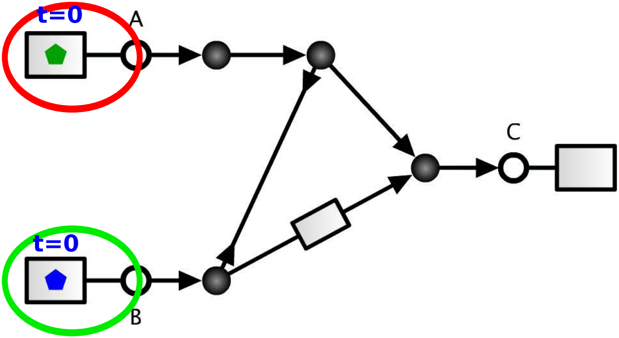


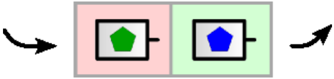
The Algorithm
A breath-first traversal



The Algorithm

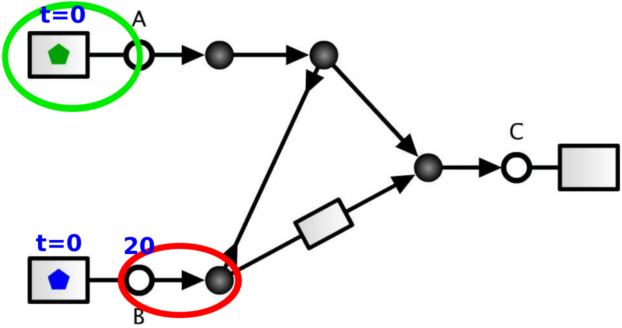
A breath-first traversal

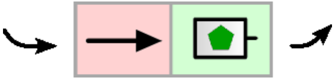




The Algorithm

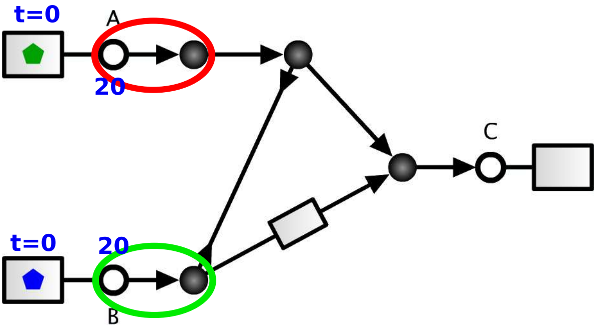
A breath-first traversal

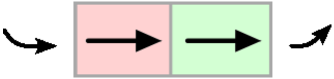




The Algorithm

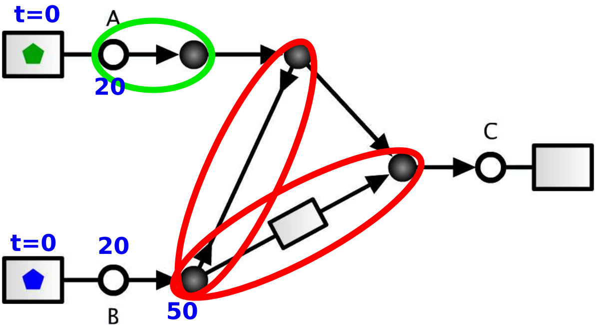
A breath-first traversal

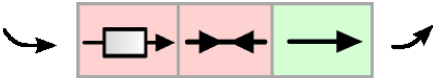




The Algorithm

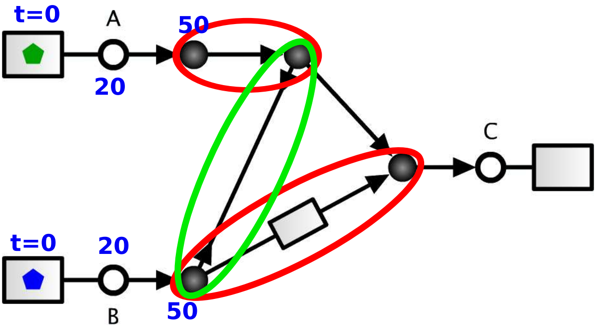
A breath-first traversal

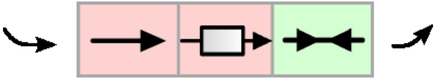




The Algorithm

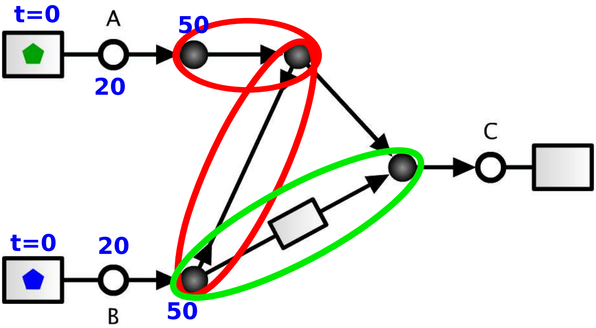
A breath-first traversal

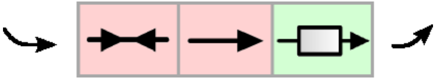




The Algorithm

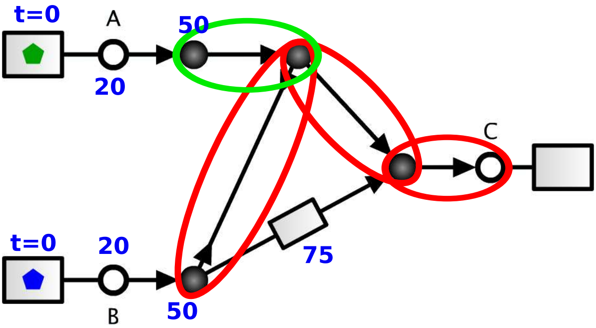
A breath-first traversal

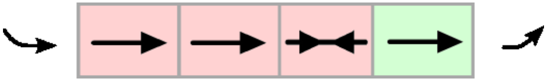




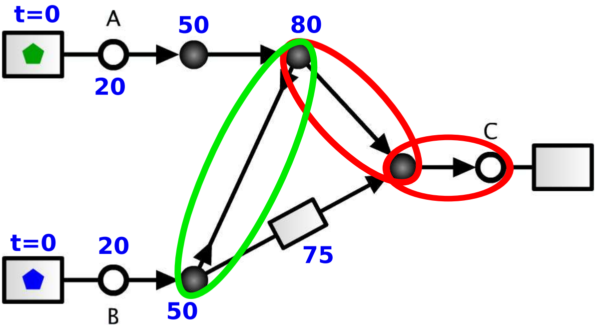
The Algorithm

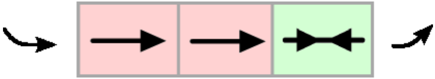
A breath-first traversal





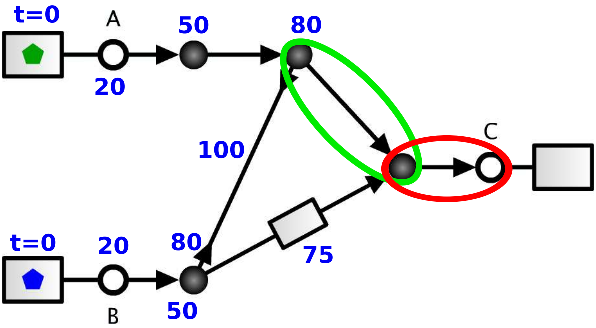
The Algorithm
A breath-first traversal

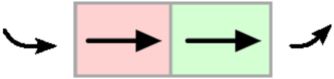




The Algorithm

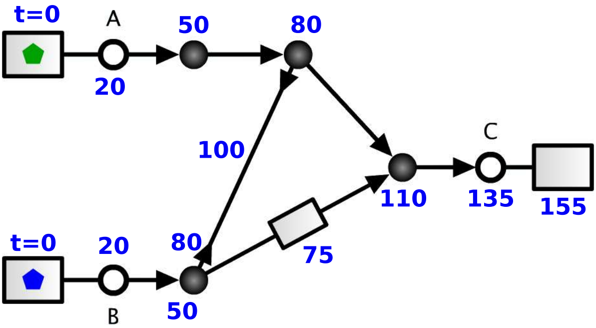
A breath-first traversal





The Algorithm

A breath-first traversal



Outline

Conclusion and Future Work

- Precise and *intuitive* representation of the behaviour of circuits;
- Animations produced by composing Animation Steps;
- Steps can be composed.

- Framework for developing Reo Connectors;
- Interaction: choosing existing components and non-deterministic choices during animation.

See animations online: `www.cwi.nl/~proenca/webreo`

Conclusion and Future Work

- Precise and *intuitive* representation of the behaviour of circuits;
- Animations produced by composing Animation Steps;
- Steps can be composed.

- Framework for developing Reo Connectors;
- Interaction: choosing existing components and non-deterministic choices during animation.

See animations online: www.cwi.nl/~proenca/webreo

Related work

- David Harel's statecharts (Statemate and Rhapsody tools)
  Reactive Animations — Flash animations interacting via TCP/IP with Rhapsody executing statecharts.
- *Animation of Behaviour Models*, by Magee et al.;
  extension of LTS with mappings from labels to actions and conditions;
  based on Timed Automata to allow composition;
  resulting XML used to generate JavaBeans.
- *Goal-Oriented Requirements Animation*, by Tran Van, van Lamsweerde, Massonet, Ponsard;
  good way to communicate with stakeholders;
  integrated in the FAUST formal analysis suite;
  multiple stakeholders can interact with the animation over the Internet.