On the Implementability of Behavioural Systems
(Preliminary Report)

Siu O’Young & Theodore S. Norvell
Memorial University of Newfoundland
NECEC 1999
Goals

• Common formalism for
  * continuous
  * discrete
  * discrete event
  * hybrid
  systems

• Common definitions of
  * refinement
  * composition
  * implementability.
Specifying Behaviours

A specification is a pair of sets

\[(U, B)\]

where

- \(U\) is a universum — a set of possible things
- \(B\) is a set of acceptable things.

In modelling the behaviour of systems

- \(U\) is a set of behaviours

**Continuous behaviours**

\(U\) is the set of all functions from a continuous time domain \(T\) to a signal space \(W\)

\[U = (T \rightarrow W)\]

**Examples**

Classical signals and systems theory.

**Logical Behaviours**

\(U\) is the set of all finite sequences of symbols in a (finite)
alphabet $\Sigma$

$$U = \Sigma^*$$

Examples

- Finite State Automate
- Regular expressions
- Context-free Grammars

**Synchronous (or Reactive) systems**

$U$ is the set of all sequences over sets of symbols

$$U = (2^\Sigma)^\omega$$

**Batch Program Behaviours**

$U$ is the set of all pairs over a statespace $S$

$$U = S \times S$$

Examples

Statements in a computer language:

$$x := x + 1$$

modeled by

$$B = \{(s, t) \mid t.x = s.x + 1 \land \forall y \neq x \cdot t.y = s.y\}$$
Specification and Refinement

We say that \((U, B_I)\) refines \((U, B_S)\) iff
\[
B_I \subseteq B_S
\]
The idea is that \((U, B_S)\) is a specification:
- a description of acceptable behaviour.

And \((U, B_I)\) is an proposed implementation
- or a step towards an implementation

Refinement means \((U, B_I)\) has no unacceptable behaviours.
Composition

The composition of two systems \((U, B_0)\) and \((U, B_1)\) is the result of them acting together. The composition is defined as \((U, B_0 \cap B_1)\).

For example, informally we want to know if plant + controller meets specification?

This question becomes

\[ B_{PLANT} \cap B_{CONTROLLER} \subseteq B_{SPEC} \]
Implementability

Informally a specification is *implementable* if it is logically conceivable that it could be implemented. In particular the system must not force its input nor overdetermine its output.

- It is important to test a specification for implementability before trying to implement it with a real system.
- It is important that the operators of any programming language preserve implementability.

**Example:**

In *Batch Program Behaviours*

- Behaviours are input/output pairs and \((U, B)\) is implementable iff
  \[
  \forall s \cdot \exists t \cdot (s, t) \in B
  \]

**Example:**

In discrete event systems we partition the alphabet into input and output symbols

\[
\Sigma_{IN} \cup \Sigma_{OUT} = \Sigma \quad \Sigma_{IN} \cap \Sigma_{OUT} = \emptyset
\]
Now \((\Sigma^*, B)\) is implementable iff
\[
\forall s \in \bar{B} \cdot \forall \sigma \in \Sigma_{IN} \cdot s\sigma \in \bar{B}
\]
where \(\bar{B}\) is the set of all prefixes of strings in \(B\).
I.e.
\[
s \in \bar{B} \iff \exists t \cdot st \in B
\]
Breaking up the behaviours

We must be able to discriminate

- past from future and
- inputs from outputs

Assume there is a space of *pasts*  
\[ U_P \]

and a space of *futures*  
\[ U_F \]

and a partial function that puts them together  
\[ \oplus : U_P \times U_F \rightarrow U \]

Assume there is a space of *inputs*  
\[ U_I \]

and a space of *outputs*  
\[ U_O \]

and a partial function that puts them together  
\[ \otimes : U_I \times U_O \rightarrow U \]
Formalizing implementability

We say that a system \((U, B)\) is *implementable* iff regardless of the past and of the future inputs, there is always a possible output. I.e. iff

\[
\forall s \in \bar{B} \cdot \forall i \in I_s \cdot \exists o \in O_s \cdot i \otimes o \in B
\]

where \(\bar{B}\) is the set of all pasts of \(B\)

\[
\bar{B} = \{ s \in U_P \mid \exists t \in U_F \cdot s \oplus t \in B \}
\]

and \(I_s\) is the set of all inputs compatible with \(s\)

\[
I_s = \{ i \in U_I \cdot \exists o \in U_O \cdot \exists t \in U_F \cdot i \otimes o = s \oplus t \}
\]

and \(O_s\) is the set of all outputs compatible with \(s\)

\[
O_s = \{ o \in U_O \cdot \exists i \in U_I \cdot \exists t \in U_F \cdot i \otimes o = s \oplus t \}
\]