The SR Domain

Stephen Edwards
Edward A. Lee

http://www.eecs.berkeley.edu/~sedwards/

University of California, Berkeley
The SR Domain

- A specification scheme
  - Synchronous model of time
    * Predictable temporal behavior
    * Easier to design
    * Easier to analyze
  - Heterogeneous: compiler cannot see inside blocks
    * Mixing languages made easy
    * Allows separate compilation
    * Large designs are tractable
- Deterministic
  - Guaranteed by fixed-point semantics
- Fast, predictable execution time
  - Chaotic iteration-based scheme
  - Fully static scheduling
SR Systems

Zero-delay blocks compute continuous functions

Instantaneous communication with feedback

Single driver, multiple receiver wires with values from flat CPOs

- Block functions may change between instants for time-varying behavior
- Block functions may be specified in any language
Zero Delay and Feedback

How to maintain determinism?

Which goes first?
Need an order-invariant semantics

Contradictory!
Need to attach meaning to such systems.
Dealing with Feedback

Why bother at all?
Answer: Heterogeneity

- Cycles are usually broken by delay elements at the lowest level
- Some schemes (e.g., Lustre) insist on this
- False feedback often appears at higher levels
- Data dependent cycles can appear when sharing resources
- Virtually all cycles are “false,” yet must be dealt with.
Fixed-point Semantics are Natural for Synchronous Specifications with Feedback

Why a fixed point?

Self-reference:
The essence of a cycle

\[ f(x_t) = x_t \]

System function \quad Vector of signals
(composition of block functions) \quad at time \( t \)
(zero delay)

fixed point \iff stable state

determinism \iff unique solution
The Least Fixed Point of What?

Interpret as

\[ B(I, f(I)) = f(I) \]

Take LFP
Unique Least Fixed Point Theorem

Recall:
A monotonic function on a complete partial order (with $\bot$) has a unique least fixed point.

What does it mean to make the system function $f$ monotonic and the signal values a CPO?
Vector of Signals is a CPO

Values along an upward path grow more defined.

```
1
  \downarrow
  "Undefined"
```

```
1 0
```

More Defined

Incomparable

Less Defined

```
11 01 10 00
```

vector-valued extension

```
11 1 1 1 0 0
```

Formally, $x \sqsubseteq y$ if $y$ is at least as defined as $x$. 
**Adding \(\bot\) Is Enough**

Any set \(\{a_1, a_2, \ldots, a_n, \ldots\}\) can easily be “lifted” to give a flat partial order:

\[
\begin{array}{c}
a_1 \\
\downarrow \\
a_2 \\
\downarrow \\
a_3 \\
\downarrow \\
\vdots \\
\downarrow \\
a_n \\
\downarrow \\
\vdots
\end{array}
\]

A CPO for signals with pure events:

- \(\text{absent}\)
- \(\text{present}\)

\[
\begin{array}{c}
\text{absent} \\
\downarrow \\
\text{present}
\end{array}
\]

A CPO for valued events:

- \(\text{absent}\)
- \(v_1\)
- \(v_2\)
- \(\vdots\)
- \(v_n\)

Why not \(\text{absent} \sqsubseteq \text{present}\)?

- \(\text{present}\)
- \(A\) then \(\ldots\)
- \(\text{else} \ldots\)
- \(\text{end}\)

Violates monotonicity
Monotonic Block Functions

Giving a more defined input to a monotonic function always gives a more defined output.

Formally, $x \sqsubseteq y$ implies $f(x) \sqsubseteq f(y)$.

A monotonic function never recants (“changes its mind”).
Many Languages Use Strict Functions, Which Are Monotonic

A strict function:

\[ g(\ldots, \perp, \ldots) = (\perp, \ldots, \perp) \]

Outside:
A strict monotonic function

Inside:
Simple “function call” semantics

Most common imperative languages only compute strict functions.

Danger: Cycles of strict functions
deadlock—fixed point is all \( \perp \)—need some non-strict functions.
A Simple Way to Find the Least Fixed Point

\[ \bot \subseteq f(\bot) \subseteq f(f(\bot)) \subseteq \cdots \subseteq \text{LFP} = \text{LFP} = \cdots \]

For each instant,

1. Start with all signals at \( \bot \)
2. Evaluate all blocks (in some order)
3. If any change their outputs, repeat Step 2

\[
\begin{array}{cccc}
  f_0 & a & f_1 & b & f_2 & c \\
  \downarrow & & & & & \\
  (a, b, c) & = & (\bot, \bot, \bot) \\
  f_0(\bot, \bot, \bot) & = & (0, \bot, \bot) \\
  f_1(0, \bot, \bot) & = & (0, 1, \bot) \\
  f_2(0, 1, \bot) & = & (0, 1, 0) \\
  f_2(f_1(f_0(0, 1, 0))) & = & (0, 1, 0)
\end{array}
\]
The Dependency Graph

Transform into single-output functions:

```
    A  1  2
   /\   /
  /   \ /   \  
 B  3  4  D  6  7
    \   /   
     \ /    
      \   /
       \ /
       \  
        \|
       5

\downarrow

1  2  3  4  5  6  7
```
The Scheduling Algorithm

1. Decompose into strongly-connected components

2. Remove a head (set of vertices) from each SCC, leaving a tail

3. Recurse on each tail
Evaluating SCCs

Split a strongly-connected graph into a head and tail:

Good heads break T’s strong connectivity.
Schedules

1 2
4 5

head

( 1 2 . ( 4 . 5 ) 6 ( 0 . 3 )

SCC

5 4 5 6 3 0 3 1 2 5 4 5 6 3 0 3 1 2 5 4 5 6 3 0 3
Finding Good Heads

Must break strong connectivity—remove a border of a set of vertices:

border of \{ A, B, C \}

(vertices with incoming edges)
Choosing Good Border Sets

Heuristic: “Grow” a set starting from a vertex and greedily include the best border vertex:

```
<table>
<thead>
<tr>
<th>Set</th>
<th>Border</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>1 5</td>
<td>2 3</td>
</tr>
<tr>
<td>1 5 2</td>
<td>3</td>
</tr>
<tr>
<td>1 5 2 3</td>
<td>7</td>
</tr>
<tr>
<td>1 5 2 3 7</td>
<td>4 6</td>
</tr>
<tr>
<td>1 5 2 3 7 4</td>
<td>6</td>
</tr>
</tbody>
</table>
```

2 is better (3 would increase border)
The Cost of Using the Heuristic

Increase in Cost of Schedule

Fraction of Runs

Number of Outputs

0% 5% 10% 15% 20% 25%

0 20 40 60 80

0% 50% 100% 150%
Asymptotic Schedule Cost

Number of Outputs

Optimal Schedule Cost

$n^2$

$n^{1.5}$

$n$
Conclusions

- Deterministic specification scheme combining synchrony and heterogeneity
- Semantics: the least fixed point of a continuous function on a CPO
- Iterative execution scheme based on recursive divide-and-conquer
- Exact scheduling practical for small graphs
- Heuristic practical for very large graphs
- Execution time for random graphs growing slower than $n^{1.5}$