A Sequentially Constructive Circuit Semantics for Esterel

Alexander Schulz-Rosengarten,
Reinhard von Hanxleden, Kiel University
and Michael Mendler, Bamberg University
Program Classes

Berry Constructive (BC) Esterel
- Circuit semantics → Delay insensitive circuits

Sequentially Constructive Circuit (SCC) Semantics
Overview

Visibility

Visibility in circuits

Visibility on source level
module OffOn:

Program Classes

BC

Acyclic BC

Acyclic

SCC

SC
module OffOn:
output S, T, U;
present S then emit T end;
emit S;
present S then emit U end
module OffOn:
output S, T, U;
present S then
  emit T end;
emit S;
present S then
  emit U end

bool S = T = U = false;
if S {
  T = true 
};
S = true;
if S {
  U = true 
};

Signals encoded as booleans
Proposal

Constructive coherence law:
A signal is present/absent iff it must/cannot be emitted

Sequentially Constructive Coherence Law:
A signal is present/absent iff it must/cannot be emitted concurrently or sequentially preceding

We say that an emit E is **SC-visible** to a present test P if:
1) E is concurrent to P or
2) E sequentially precedes P
SC-Visibility in Circuits
SCC Emit

!s
SCC Present Test
s? P, Q
SCC Sequence

\[ P ; Q \]
SCC Parallel

P || Q
Further SCC Rules

**Signal**

**Loop**

**Pause**

**Suspend**

**Trap**

**Exit**

**Nothing**
module OffOn:
output S, T, U;
present S then emit T end;
emit S;
present S then emit U end
Applying SC-visibility to Esterel on source level.

**Static Single Assignment Form**

```plaintext
module OffOn:
output S, T, U;
present S then emit T end;
emit S end;
present S then emit U end
end;
```

**Source-to-Source Transformation**

SCC to BC

```plaintext
module OffOn:
output S_0, S_1, T, U;
present S_0 then emit S_0 end;
present S_1 then emit S_1 end;
```

```plaintext
emit S_0
```
emit $S_0$

Concurrent SSA for Esterel

```plaintext
module ST:
output S, T;
[
  present S then
  emit S end
||
  present T else
  emit S end
];
present S then
emit T end
```
Concurrent SSA for Esterel

emit $S_0$

Concurrent Read

Concurrent Merge

Sequential Merge

“Signal” Initialization
Concurrent SSA for Esterel

emit $S_0$

S = $\Psi(S_3, S_4)$

T = true

T1 = $\Phi(T0, T)$

entry

fork

T0 = false

S2 = false

entry

S2 | S1

S0 = true

S1 = true

S3 = $\Phi(S2, S0)$

S4 = $\Phi(S1, S2)$

exit

S = $\Psi(S3, S4)$

true

T = true

T1 = $\Phi(T0, T)$

exit

join

entry

fork

T0 = false

S2 = false

entry

S2 | S1

S0 = true

S1 = true

S3 = $\Phi(S2, S0)$

S4 = $\Phi(S1, S2)$

exit

S = $\Psi(S3, S4)$

true

T = true

T1 = $\Phi(T0, T)$

exit

join

entry

fork

T0 = false

S2 = false

entry

S2 | S1

S0 = true

S1 = true

S3 = $\Phi(S2, S0)$

S4 = $\Phi(S1, S2)$

exit

S = $\Psi(S3, S4)$

true

T = true

T1 = $\Phi(T0, T)$

exit

join

entry

fork

T0 = false

S2 = false

entry

S2 | S1

S0 = true

S1 = true

S3 = $\Phi(S2, S0)$

S4 = $\Phi(S1, S2)$

exit

S = $\Psi(S3, S4)$

true

T = true

T1 = $\Phi(T0, T)$

exit

join

entry

fork

T0 = false

S2 = false

entry

S2 | S1

S0 = true

S1 = true

S3 = $\Phi(S2, S0)$

S4 = $\Phi(S1, S2)$

exit

S = $\Psi(S3, S4)$

true

T = true

T1 = $\Phi(T0, T)$

exit

join

entry

fork

T0 = false

S2 = false

entry

S2 | S1

S0 = true

S1 = true

S3 = $\Phi(S2, S0)$

S4 = $\Phi(S1, S2)$

exit

S = $\Psi(S3, S4)$

true

T = true

T1 = $\Phi(T0, T)$

exit

join

entry

fork

T0 = false

S2 = false

entry

S2 | S1

S0 = true

S1 = true

S3 = $\Phi(S2, S0)$

S4 = $\Phi(S1, S2)$

exit

S = $\Psi(S3, S4)$

true

T = true

T1 = $\Phi(T0, T)$

exit

join

entry
emit $S_0$

Concurrent SSA for Esterel
Concurrent SSA for Esterel

```plaintext
module ST-SSA:
  output S, T;
  signal S0, S1 in
  [  
    present S1 then
      emit S0 end
    ||
      emit S1
  ];
  present S0 or S1 then
    emit S end;
  present S then
    emit T end
```

emit S₀
Summary

• Sequentially Constructive Circuit (SCC) semantics for Esterel

• Source-to-source transformation from SCC programs into equivalent Berry constructive programs

• Claim conservativeness
  – BC => SCC
  – Preserving semantics
  – Proof sketch (WIP)

The End