Synthesizing Safe State Machines from Esterel

Steffen Prochnow, Claus Traulsen, Reinhard v. Hanxleden

Christian-Albrechts Universität zu Kiel
Department of Computer Science and Applied Mathematics
Real-Time Systems and Embedded Systems Group
rvh@informatik.uni-kiel.de

LCTES'06, Ottawa, June 2006

Introduction

- Esterel: Synchronous language for programming reactive systems [Berry 1984]
- Statecharts: Visual formalism to model reactive systems [Harel 1987]
- Safe State Machines (SSMs): Statechart dialect with fully synchronous model of computation [André 1996/2003]

Introduction

- Esterel: Synchronous language for programming reactive systems [Berry 1984]
- Statecharts: Visual formalism to model reactive systems [Harel 1987]
- Safe State Machines (SSMs): Statechart dialect with fully synchronous model of computation [André 1996/2003]
- Visual formalism facilitates understanding complex behaviors

Introduction

- Esterel: Synchronous language for programming reactive systems [Berry 1984]
- Statecharts: Visual formalism to model reactive systems [Harel 1987]
- Safe State Machines (SSMs): Statechart dialect with fully synchronous model of computation [André 1996/2003]
- Visual formalism facilitates understanding complex behaviors—*but there are also drawbacks wrt textual formalisms!*

Introduction

- Esterel: Synchronous language for programming reactive systems [Berry 1984]
- Statecharts: Visual formalism to model reactive systems [Harel 1987]
- Safe State Machines (SSMs): Statechart dialect with fully synchronous model of computation [André 1996/2003]
- Visual formalism facilitates understanding complex behaviors—*but there are also drawbacks wrt textual formalisms!*

The work presented here aims to make the best of both worlds accessible to the system developer

Overview

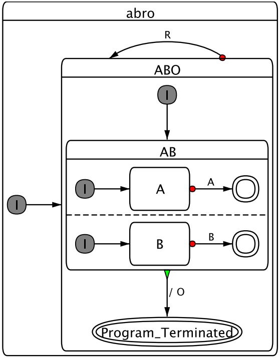
Example: ABRO/ABCRO

```
module ABRO:
input A, B, R;
output O;
loop
  [
    await A
  ||
    await B
  ];
  emit O;
each R
end module
```

Example: ABRO/ABCRO

```
module ABRO:
input A, B, R;
output O;
loop
  [
    await A
  ||
    await B
  ];
  emit O;
each R
end module
```

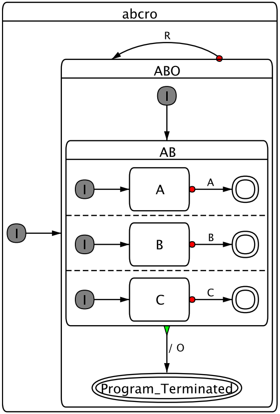
$\Longleftrightarrow$



Example: ABRO/ABCRO

```
module ABCRO:
input A, B, C, R;
output O;
loop
  [
    await A
  ||
    await B
  ||
    await C
  ];
  emit O;
each R
end module
```

$\Longleftrightarrow$



Which is faster?

Editing Esterel:

1. move cursor to position
2. type ", C"
3. move cursor to position
4. type "`|| await C`"

Which is faster?

Editing Esterel:

1. move cursor to position
2. type ", C"
3. move cursor to position
4. type "`|| await C`"

Editing Safe State Machine:

1. make room: shift neighbor states, enlarge parent state
2. click on "add state"
3. move mouse to location and place new state
4. click on "add state"
5. move mouse to location and place new state
6. double click on new state, toggle terminal field
7. click on "initial state"
8. move mouse to location and place new initial state
9. click on "transition"
10. move mouse to location of initial state
11. press left mouse button and keep pressed until reaching state
12. click on "transition"
13. move mouse to location of state
14. press left mouse button and keep pressed until reaching terminal state
15. double click on transition
16. write "C" in trigger field
17. press "OK"
18. click on "delimiter line"
19. move mouse to location and place delimiter line

Which is more tracable?

Textual:

```
1,2c1,2
< module ABRO:
< input A, B, R;
---
> module ABCRO:
> input A, B, C, R;
5c5
<   [ await A || await B ];
---
>   [ await A || await B || await C ];
```

Which is more tracable?

Textual:

```
1,2c1,2
< module ABRO:
< input A, B, R;
---
> module ABCRO:
> input A, B, C, R;
5c5
<   [ await A || await B ];
---
>   [ await A || await B || await C ];
```

Graphical:

```
1c1
< #    Model of type
     Document saved by /
     home/esterel/
     EsterelStudio
   −5.2/bin/estudio.exe
        [11/18/2005
        10:39:01]
−−−
> #    Model of type
     Document saved by /
     home/esterel/
     EsterelStudio
   −5.2/bin/estudio.exe
        [11/18/2005
        10:40:03]
161c161
< {115
−−−
> {295
227c227
< AT   107 145
−−−
> AT   197 145
243c243
< [E]
−−−
> [V105 E E]
344a345,519
> NODE init.2   init
> ATTRIB
> {[]
> ""
>
> {0
> 0
> }
```

```
> {
> {"helvetica"
> 12
> 1
> "Blue"
> }
> {"helvetica"
> 12
> 1
> "Blue"
> }}<false>
> }
> AT   170 35
> END # of init.2
>
> NODE state.4 state
> ATTRIB
> {""
> <halt>
> <no>
> ""
>
> {30
> 30
> 0
> 0
> 1
> "1 0 0 1 0 0"
> 0
> 0
> 0
> 0
> 0
> 0
> "1 0 0 1 0 0"
> }
> {[]
```

```
> []
> []
> }
> {[]
> []
> []
> }
> {<false>
> ""
> }
> {
> {"helvetica"
> 12
> 1
> "Blue"
> }
> {"Black"
> 1
> "none"
> 0
> }"White"
>
> {"helvetica"
> 12
> 4
> "Blue"
> }
> {"helvetica"
> 12
> 4
> "Blue"
> }
> {"helvetica"
> 12
> 4
> "Blue"
```

(Only first 100 of 287 lines shown)

Overview

From Esterel to Safe State Machines

Step 1: Transform Esterel program into SSM with textual macro states

From Esterel to Safe State Machines

Step 1: Transform Esterel program into SSM with textual macro states

Step 2: Iteratively apply reduction rules to transform Esterel constructs into graphical components

From Esterel to Safe State Machines

Step 1: Transform Esterel program into SSM with textual macro states

Step 2: Iteratively apply reduction rules to transform Esterel constructs into graphical components

Step 3: Optimize SSM

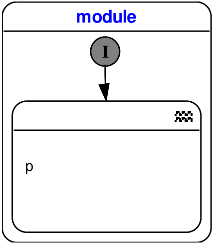
Step 1: Transform Esterel to SSM

```
module mod_name:
  input I_1, ..., I_n;
  output O_1, ..., O_m;
  p
end module
```

Step 1: Transform Esterel to SSM

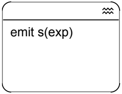
```
module mod_name:
  input I_1, ..., I_n;
  output O_1, ..., O_m;
  p
end module
```

$\Longrightarrow$

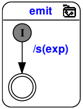


Step 2: Reduce to Fully Graphical SSM

Signal emission

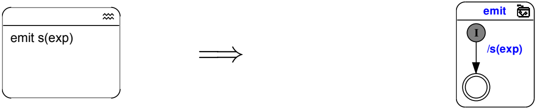


$\Longrightarrow$

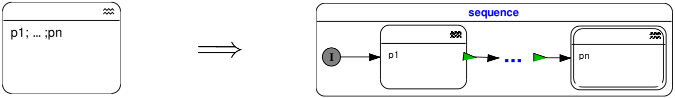


Step 2: Reduce to Fully Graphical SSM

Signal emission

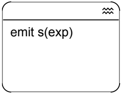


Sequence

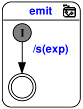


Step 2: Reduce to Fully Graphical SSM

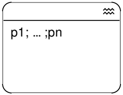
Signal emission



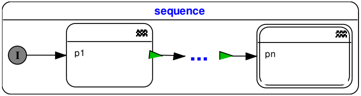
$\Longrightarrow$



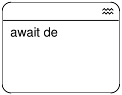
Sequence



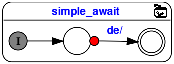
$\Longrightarrow$



Signal awaiting

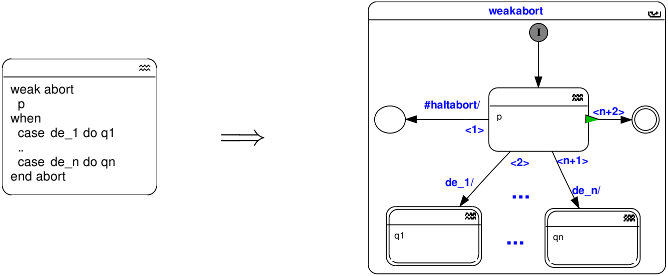


$\Longrightarrow$



Step 2: Reduce to Fully Graphical SSM

Weak Abortion



+ 19 additional rules

Translation of traps not trivial—see paper

Example: ABRO

Applying Rule (module)

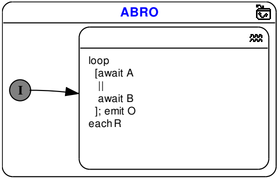
```
module ABRO:
input A, B, R;
output O;
loop
  [
    await A
  ||
    await B
  ];
  emit O;
each R
end module
```

Example: ABRO

Applying Rule (module)

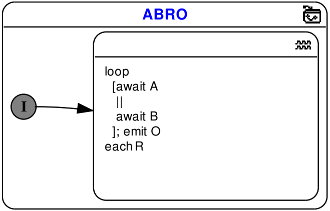
```
module ABRO:
input A, B, R;
output O;
loop
  [
    await A
  ||
    await B
  ];
  emit O;
each R
end module
```

$\Longrightarrow$



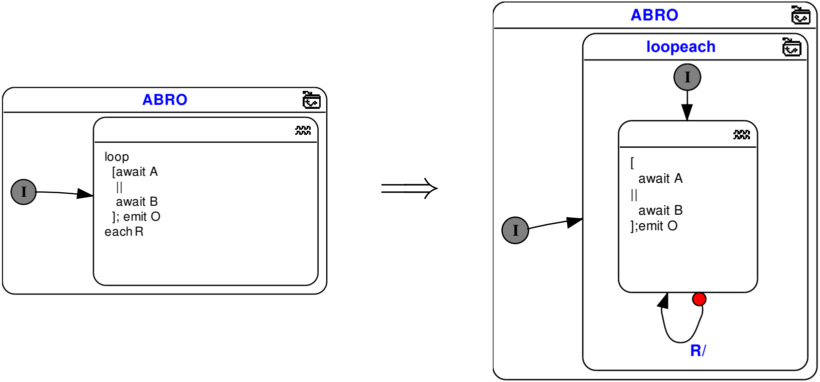
Example: ABRO

Applying Rule (loopeach)



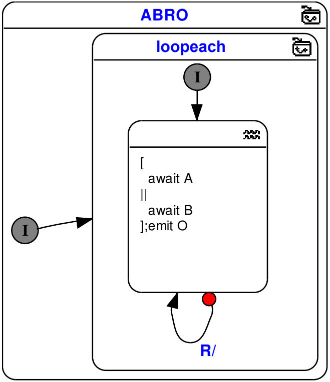
Example: ABRO

Applying Rule (loopeach)

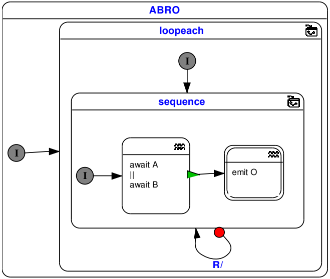


Example: ABRO

Applying Rule (sequence)

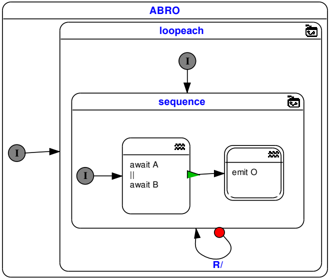


$\Longrightarrow$

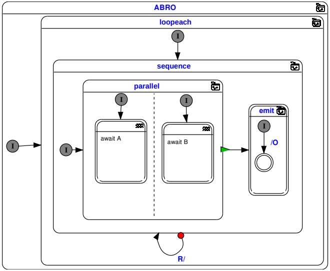


Example: ABRO

Applying Rules (parallel) + (emit)

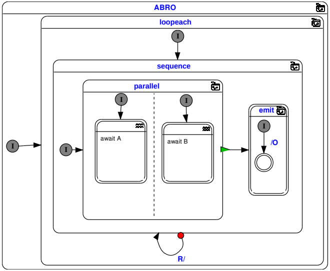


⟹

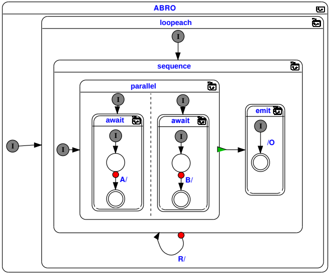


Example: ABRO

Applying Rule (simple await)



⟹



Step 3: Optimizations

Motivation

- Automatic synthesis produces "verbose" modules
- However, also human modelers (esp. novices) may produce sub-optimal models

Step 3: Optimizations

Motivation

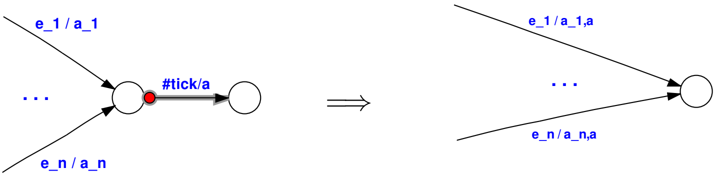
- Automatic synthesis produces "verbose" modules
- However, also human modelers (esp. novices) may produce sub-optimal models

Notes:

- It may be a matter of style/opinion what "optimal" means
- However, consistency in style is desirable in any case—and standardized optimization rules help to achieve this

Step 3: Optimizations

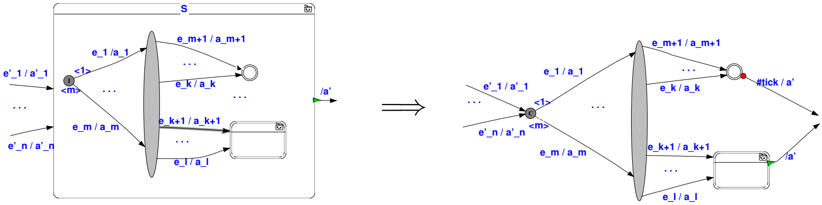
Removing Simple States



Precondition: State must be transient

Step 3: Optimizations

Flattening Hierarchy



Preconditions:

- no abort originate from S
- S has no local signals

+ further rules to remove conditionals, combine terminal states, remove normal terminations

Overview

Experimental Results

Model	Esterel	Safe State Machines									Time	
		Before Optimization				After Optimization						
	Lines of Code (LoC)	States/Pseudo-states	Transitions	Total Graphical Elements (GEs)	GEs/LoC	States/Pseudo-states	Transitions	Total Graphical Elements (GEs)	GEs/LoC	SSM Reduction Factor	Transformation [ms]	Optimization [ms]
ABRO	7	12/8	12	32	4.57	8/4	8	20	2.86	0.63	861	56
SCHIZOPHRENIA	10	13/9	14	36	3.60	4/3	6	13	1.30	0.36	838	81
REINCARNATION	25	27/17	28	72	2.88	5/2	6	13	0.52	0.18	642	83
JACKY1	27	31/19	33	83	3.07	11/5	12	28	1.04	0.34	913	93
RUNNER	55	39/24	42	105	1.91	21/11	25	57	1.04	0.54	725	160
TOKENRING3	79	77/61	91	229	2.90	15/20	38	73	0.92	0.32	733	278
GREYCOUNTER	82	211/148	254	613	7.48	42/50	106	198	2.41	0.32	789	593
ABCD	101	231/130	250	611	6.05	78/41	97	216	2.14	0.35	943	731
TOKENRING10	247	245/194	294	733	2.97	43/62	122	227	0.92	0.31	1267	736
MEJIA	555	374/246	414	1034	1.86	127/76	181	384	0.69	0.37	3085	1266
TCINT	687	475/285	543	1303	1.90	163/81	221	465	0.68	0.36	3382	1310
ATDS-100	948	961/558	1092	2611	2.75	352/184	504	1040	1.10	0.40	7046	2760
WW	1088	342/228	386	965	0.89	102/85	177	364	0.33	0.38	4470	1053
TOKENRING50	1207	1205/954	1454	3613	2.99	203/302	602	1107	0.92	0.31	6608	8148
TOKENRING100	2407	2405/1904	2904	7213	3.00	403/602	1202	2207	0.92	0.31	20910	25528
MCA200	7269	5159/3931	5947	15037	2.07	179/925	1794	2898	0.40	0.19	61510	100594

Summary

Graphical vs. textual modeling

- Graphical models nice to browse/simulate
- Textual models much faster to edit, better to maintain

Summary

Graphical vs. textual modeling

- Graphical models nice to browse/simulate
- Textual models much faster to edit, better to maintain

Enablers to take best of both worlds

1. Matching of Esterel semantics and SSM semantics
2. Ability to do automatic layout

Summary

Graphical vs. textual modeling

- Graphical models nice to browse/simulate
- Textual models much faster to edit, better to maintain

Enablers to take best of both worlds

1. Matching of Esterel semantics and SSM semantics
2. Ability to do automatic layout

Have implemented this in KIEL framework

- Edit Esterel code, simulate SSMs
- Facilitates tracability
- Allows designer to focus on **content**, rather than layout

Summary

Graphical vs. textual modeling

- Graphical models nice to browse/simulate
- Textual models much faster to edit, better to maintain

Enablers to take best of both worlds

1. Matching of Esterel semantics and SSM semantics
2. Ability to do automatic layout

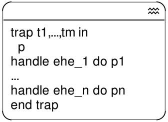
Have implemented this in KIEL framework

- Edit Esterel code, simulate SSMs
- Facilitates tracability
- Allows designer to focus on **content**, rather than layout

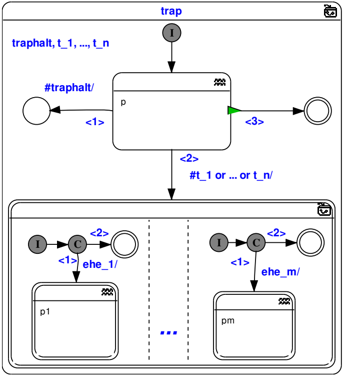
Thanks! Questions/Comments?

Translating Traps

The Trap Reduction Rule



⟹



Translating Traps

Detecting Instantaneous Loops

```
module LOOP_WITH_TRAP:
loop
  trap T in
    pause;
    exit T
  end trap
end loop
end module
```

$\Longrightarrow$

Translating Traps

Detecting Instantaneous Loops

```
module LOOP_WITH_TRAP:
loop
  trap T in
    pause;
    exit T
  end trap
end loop
end module
```

$\Longrightarrow$

```
module LOOP_WEAK_ABORT:
loop
  signal T in
    weak abort
      pause;
      emit T
    when immediate T
  end signal
end loop
end module
```

Problem: Compiler (falsely) assumes that loop body may be instantaneous

Translating Traps

Detecting Instantaneous Loops

```
module LOOP_WITH_TRAP:
loop
  trap T in
    pause;
    exit T
  end trap
end loop
end module
```

$\Longrightarrow$

```
module LOOP_WEAK_ABORT:
loop
  signal T in
    weak abort
      pause;
      emit T
    when immediate T
  end signal
||
 pause
end loop
end module
```

Solution: Add parallel pause statement

Translating Traps

Detecting Instantaneous Loops

```
module LOOP_WEAK_ABORT:
loop
  signal T in
    weak abort
      pause;
      emit T
    when immediate T
  end signal
||
  pause
end loop
end module
```

$\Longrightarrow$