

Correctness of Slicing Finite State Machines

Torben Amtoft

Kansas State University

January 27, 2014

Motivating Slicing

Deterministic Setting

Goal
Methods

Non-Determinism

Goal
Method

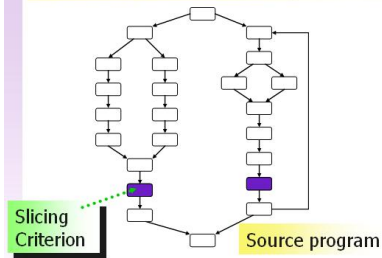
EFSM Development

Adapting Definitions
Slicing Algorithms

Conclusion

What is Slicing?

Static Backwards Slicing



Pick one or more program points of interest, called the **slicing criterion**

Motivating Slicing

Deterministic Setting

Goal
Methods

Non-Determinism

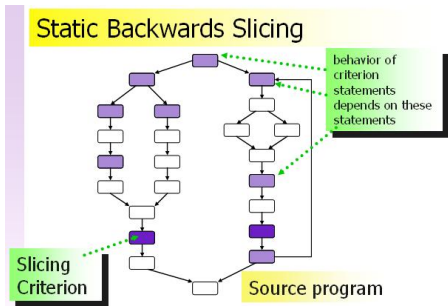
Goal
Method

EFSM Development

Adapting Definitions
Slicing Algorithms

Conclusion

What is Slicing?



Walk backwards to find the nodes (the **slice set**) that the nodes in the slicing criterion **depend** on

- ▶ through **data dependence**, or
- ▶ through **control dependence**

Remove nodes not in the slice set.

Motivating Slicing

Deterministic Setting

Goal
Methods

Non-Determinism

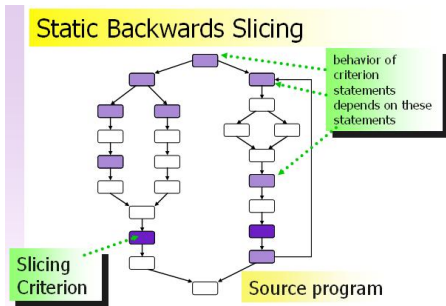
Goal
Method

EFSM Development

Adapting Definitions
Slicing Algorithms

Conclusion

What is Slicing?



Motivating Slicing

Deterministic Setting

Goal
Methods

Non-Determinism

Goal
Method

EFSM Development

Adapting Definitions
Slicing Algorithms

Conclusion

Applications include

- ▶ compiler optimizations
- ▶ debugging
- ▶ model checking
- ▶ protocol understanding

Outline of Talk

1. Summarize slicing of deterministic programs
 - ▶ for various kinds of control flow graphs
 - ▶ with focus on correctness properties
2. Discuss how to extend to non-determinism
 - ▶ restate desired correctness properties
3. Application: extended finite state machines (EFSM)
 - ▶ outline technical details
 - ▶ sketch algorithm
 - ▶ give example slices

Motivating Slicing

Deterministic Setting

Goal
Methods

Non-Determinism

Goal
Method

EFSM Development

Adapting Definitions
Slicing Algorithms

Conclusion

Initial Assumptions

We assume **for now** a **deterministic** setting, and consider a control-flow graph (CFG) where nodes are either

- ▶ assignments
 - ▶ with one successor
 - ▶ to be replaced by **skip** if sliced away
- ▶ conditionals
 - ▶ with two successors
 - ▶ to be replaced by suitable **goto** if sliced away
- ▶ end node, with no successors

There may be a **post-processing** phase which

- ▶ may **re-wire** the CFG, removing **skip** nodes etc
- ▶ is **not** the focus of this work

Motivating Slicing

Deterministic Setting

Goal
Methods

Non-Determinism

Goal
Method

EFSM Development

Adapting Definitions
Slicing Algorithms

Conclusion

Correctness of slicing (early work: [Ball & Horwitz, 1993] and [Hatcliff et al, 2000]) may be phrased as **simulation**:

- ▶ the **observables** are the nodes in the slice set
- ▶ the equivalences are **modulo relevant variables**

Weak Correctness:

Each observable action by the original program can be simulated by the sliced program

In **deterministic** setting, this implies (if **end** observable)

*Each observable action by the sliced program can be simulated by the original program **unless** original program does unobservable loop*

Strong correctness: in addition to weak correctness,

Each observable action by the sliced program can be simulated by the original program

Motivating Slicing

Deterministic Setting

Goal

Methods

Non-Determinism

Goal

Method

EFSM Development

Adapting Definitions

Slicing Algorithms

Conclusion

Basic Dependence Relations

It is standard to demand the slice set to be

- ▶ **closed** under **data dependence**
- ▶ **closed** under *some kind of* **control dependence**

```
int c = 0;
if (_result1 <= 50) {
    a = a + 1;
    b = b + 1;
    c = c + 1;
}
```

Control Dependence

Data Dependence

Defining data dependence (DD) is non-controversial:

b is data dependent on a if there is a path from a to b, and a variable defined in a and used in b but not redefined in the interior of that path

The proper notion of control dependence depends on

- ▶ the correctness criterion aimed for
- ▶ the kind of CFGs that are considered

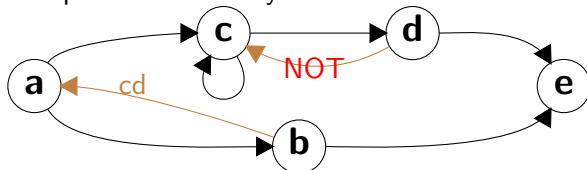
Classical Definitions of Control Dependence

Assume the CFG has **unique end node** e . We say:

b *post-dominates* a iff
all paths from a to e contain b

b is **control dependent** on a if

- ▶ a is **not strictly** postdominated by b
- ▶ there is a path from a to b where **all** nodes except a are postdominated by b

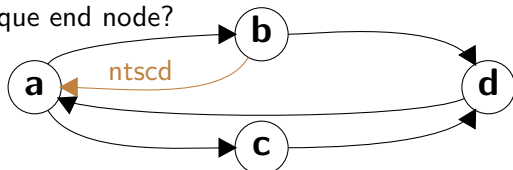


This ensures **weak** correctness. To get **strong** correctness, use **strong** post-dominance [Podgurski & Clarke]:

b *strongly post-dominates* a iff
all maximal paths from a contain b

Control Dependence for Reactive Systems, I

What if **not** unique end node?

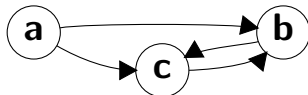


For **strong** correctness, Ranganath et al proposed [ESOP'05 & TOPLAS'07] a **conservative extension** of the **strong** version of control dependence:

b is **NTSCD**-control dependent on a iff from

- ▶ one of a 's successors, b **cannot** be avoided forever
- ▶ another of a 's successors, b **may** be avoided forever

This ensures **strong correctness** **provided** the CFG is **reducible** (forward edges form a DAG; for back edges, the target dominates the source)



Otherwise, we must add a certain “order dependence”

Motivating Slicing

Deterministic Setting

Goal
Methods

Non-Determinism

Goal
Method

EFSM Development

Adapting Definitions
Slicing Algorithms

Conclusion

To get strong correctness, slices must include **all** nodes that influence guards of potential loops

- ▶ great, if slicing to preserve liveness properties
- ▶ not so great, if slicing for program understanding

Hence we may want to go for **weak** correctness. For that, the relevant condition [Amtoft, IPL'08] is that the slice set should be closed under a **ternary** relation:

c & b are **weakly order dependent** on a iff

- ▶ $\text{path}[a..b] \not\supseteq c$ and $\text{path}[a..c] \not\supseteq b$
- ▶ a has successor x such that either b is reachable from x and all $[x..c]$ contain b , or c is reachable from x and all $[x..b]$ contain c .

Conservative extension: for a CFG with an end node which is part of slicing criterion, weak order dependence gives the same closure as standard control dependence.

Motivating Slicing

Deterministic Setting

Goal

Methods

Non-Determinism

Goal

Method

EFSM Development

Adapting Definitions

Slicing Algorithms

Conclusion

Making Sense of Chaos

Danicic et al [TCS 2011] observed, in a setting that generalizes most previous frameworks for slicing, that

- ▶ the key to get weak correctness is to ensure that the slice set is **weak commitment-closed (WCC)**: each node has at most one next observable

Motivating Slicing

Deterministic Setting

Goal

Methods

Non-Determinism

Goal

Method

EFSM Development

Adapting Definitions

Slicing Algorithms

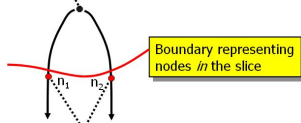
Conclusion

Making Sense of Chaos

Danicic et al [TCS 2011] observed, in a setting that generalizes most previous frameworks for slicing, that

- ▶ the key to get weak correctness is to ensure that the slice set is **weak commitment-closed (WCC)**: each node has at most one next observable

A conditional that is not in the slice...



Boundary representing nodes in the slice

First node in slice encountered along path of non-relevant nodes

Motivating Slicing

Deterministic Setting

Goal

Methods

Non-Determinism

Goal

Method

EFSM Development

Adapting Definitions

Slicing Algorithms

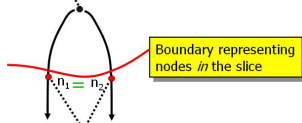
Conclusion

Making Sense of Chaos

Danicic et al [TCS 2011] observed, in a setting that generalizes most previous frameworks for slicing, that

- ▶ the key to get weak correctness is to ensure that the slice set is **weak commitment-closed (WCC)**: each node has at most one next observable

A conditional that is not in the slice...



Boundary representing nodes in the slice

First node in slice encountered along path of non-relevant nodes

Motivating Slicing

Deterministic Setting

Goal

Methods

Non-Determinism

Goal

Method

EFSM Development

Adapting Definitions

Slicing Algorithms

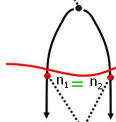
Conclusion

Making Sense of Chaos

Danicic et al [TCS 2011] observed, in a setting that generalizes most previous frameworks for slicing, that

- ▶ the key to get weak correctness is to ensure that the slice set is **weak commitment-closed (WCC)**: each node has at most one next observable

A conditional that is not in the slice...



Boundary representing nodes in the slice

First node in slice encountered along path of non-relevant nodes

- ▶ the key to get strong correctness is to ensure that the slice set is **strong commitment-closed (SCC)**: each node either has **no** next observable, or **one** next observable which **no infinite** path can miss

Correctness for Non-Determinism, I

For **weak correctness**, our previous attempt

*Each observable action by the original program
can be simulated by the sliced program*

while still **necessary** does **no longer suffice** as it allows *increased* non-determinism, giving the sliced program freedom to do actions the original program would not do.

What in a deterministic version was implied by the above, we now need to explicitly state:

*Each observable action by the sliced program
can be simulated by the original program
unless original program does unobservable loop
or original program gets stuck*

with a line added to allow for **no** feasible choices

Motivating Slicing

Deterministic Setting

Goal
Methods

Non-Determinism

Goal
Method

EFSM Development

Adapting Definitions
Slicing Algorithms

Conclusion

We stated **Weak Correctness for Non-Determinism**:

1. Each observable action by the original program can be simulated by the sliced program
2. Each observable action by the sliced program can be simulated by the original program **unless** original program does unobservable loop **or** original program gets stuck.

If we disallow the removal of unobservable loops we get **Strong Correctness for Non-Determinism**:

1. Each observable action by the original program can be simulated by the sliced program
2. Each observable action by the sliced program can be simulated by the original program **unless** original program gets stuck

Choice (debatable?): slicing may remove “stuckness”.

Motivating Slicing

Deterministic Setting

Goal
Methods

Non-Determinism

Goal
Method

EFSM Development

Adapting Definitions
Slicing Algorithms

Conclusion

Computing Slice Sets: Basic Approach

Q: which control dependencies are suitable for non-determinism?

A: we probably need to invent some quite sophisticated ones... but which???

What we shall require about the slice set is: **not** that it is closed under some kind of control dependence, but that

- ▶ it is closed under data dependence
- ▶ it satisfies **WCC**, and perhaps even **SCC**

We expect to be able to prove weak correctness from

WCC: no node has two “next observable”s

and to prove strong correctness from

SCC: each node either has no next observable, or one which no infinite path can miss

We shall now work out this agenda for a concrete setting.

Motivating Slicing

Deterministic Setting

Goal
Methods

Non-Determinism

Goal
Method

EFSM Development

Adapting Definitions
Slicing Algorithms

Conclusion

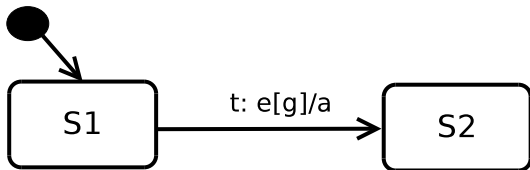
Extended Finite State Machines, Definition

To model reactive systems, an **EFSM** has

- ▶ a number of **states**
- ▶ **labeled transitions** between states

Each transition is triggered

- ▶ when **guard** is true
- ▶ possibly consuming **event** from environment
- ▶ possibly doing **action** on store



Extended Finite State Machines, Slicing

We want the slice set to contain **transitions**, rather than nodes, as this is where the real action takes place.

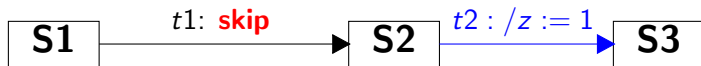
If a transition is **not** part of the slice set its

- ▶ guard becomes *true*
- ▶ action becomes **skip**

Example: the slicing criterion $t2$ does not depend on $t1$



and hence $t1$ is an “ ϵ -transition” in the sliced program:

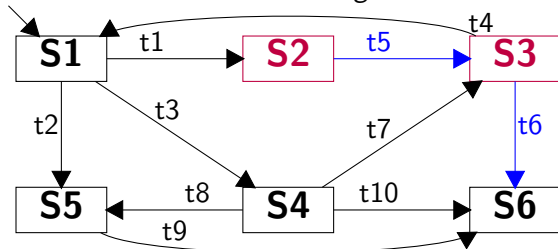


which is **less** likely to be **stuck** than the original program.

How to modify definitions developed for CFGs?

node a has node b as next observable if

- ▶ *there is a path from a to b of transitions not in slice set*
- ▶ *a transition from b belongs to the slice set*



- ▶ the next observables of S1 are S2 and S3
- ▶ the slice set does thus not satisfy WCC
- ▶ and indeed, the sliced EFSM might do t6 while
 - ▶ the original EFSM can't do t6 (due to guard for t3)
 - ▶ but may be able to do t5.

Motivating Slicing

Deterministic Setting

Goal
Methods

Non-Determinism

Goal
Method

EFSM Development

Adapting Definitions
Slicing Algorithms

Conclusion

If the slice set is weakly committed then

- ▶ if the original EFSM can do an observable step it can be simulated by the sliced EFSM
- ▶ if the sliced EFSM can do an observable step then either
 1. it can be simulated by the original EFSM, or
 2. the original EFSM may get **stuck**, or
 3. the original EFSM may enter an unobservable loop where (3) is ruled out if slice set strongly committed.

Motivating Slicing

Deterministic Setting

Goal
Methods

Non-Determinism

Goal
Method

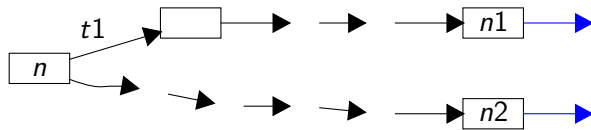
EFSM Development

Adapting Definitions
Slicing Algorithms

Conclusion

Finding Least Set Satisfying WCC: Theory

Observe: if WCC does **not** hold then the situation is



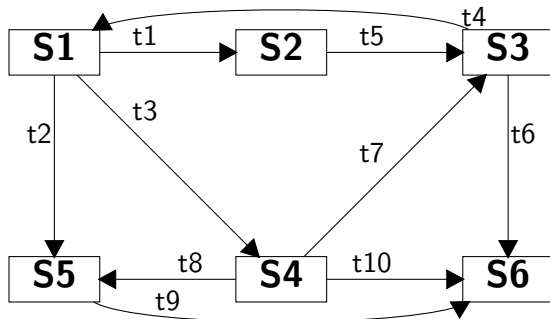
and $t1$ will belong to any superset satisfying WCC.

For a given slicing criterion, there thus exists a **least superset that satisfies WCC** (and is closed under DD) and we can write an algorithm to iteratively find this set:

- ▶ from the observables, do a **backwards** breadth-first search through transitions **not** in slice set.
- ▶ if some node n is reached from **two** observables then add to the slice set the transition(s) from n .

Running time: **quadratic** in number of transitions (excluding time to compute DD, an orthogonal issue).

Finding Least Set Satisfying WCC: Example



Motivating Slicing

Deterministic Setting

Goal
Methods

Non-Determinism

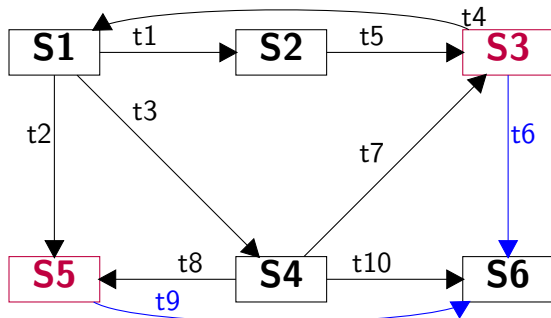
Goal
Method

EFSM Development

Adapting Definitions
Slicing Algorithms

Conclusion

Finding Least Set Satisfying WCC: Example



Starting with slicing criterion t_6, t_9 we

Motivating Slicing

Deterministic Setting

Goal
Methods

Non-Determinism

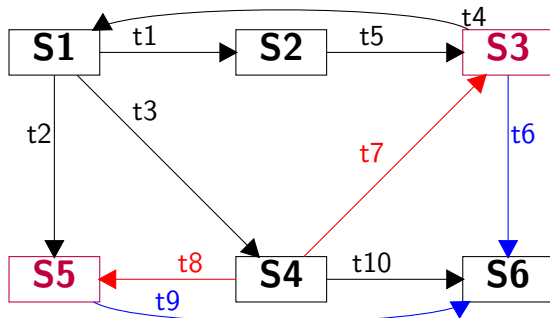
Goal
Method

EFSM Development

Adapting Definitions
Slicing Algorithms

Conclusion

Finding Least Set Satisfying WCC: Example



Starting with slicing criterion t_6, t_9 we

1. add t_8 and t_7

Motivating Slicing

Deterministic Setting

Goal
Methods

Non-Determinism

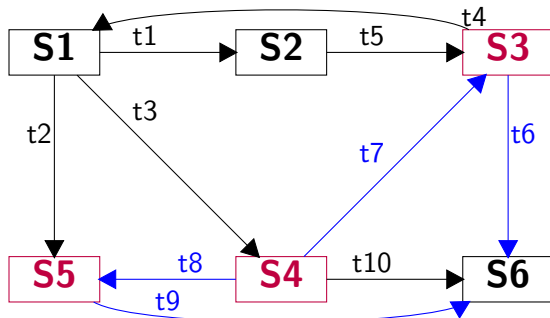
Goal
Method

EFSM Development

Adapting Definitions
Slicing Algorithms

Conclusion

Finding Least Set Satisfying WCC: Example



Starting with slicing criterion t_6, t_9 we

1. add t_8 and t_7

Motivating Slicing

Deterministic Setting

Goal
Methods

Non-Determinism

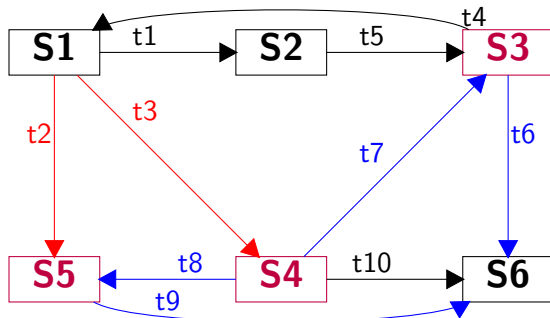
Goal
Method

EFSM Development

Adapting Definitions
Slicing Algorithms

Conclusion

Finding Least Set Satisfying WCC: Example



Starting with slicing criterion t_6, t_9 we

1. add t_8 and t_7
2. add t_2 and t_3

Motivating Slicing

Deterministic Setting

Goal
Methods

Non-Determinism

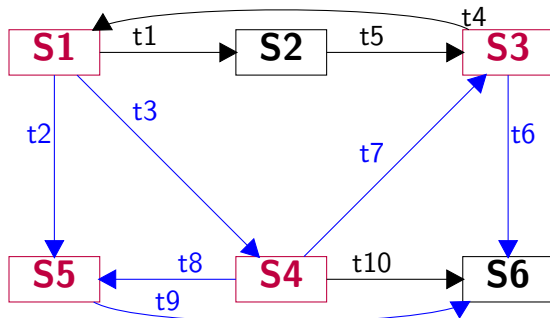
Goal
Method

EFSM Development

Adapting Definitions
Slicing Algorithms

Conclusion

Finding Least Set Satisfying WCC: Example



Motivating Slicing

Deterministic Setting

Goal
Methods

Non-Determinism

Goal
Method

EFSM Development

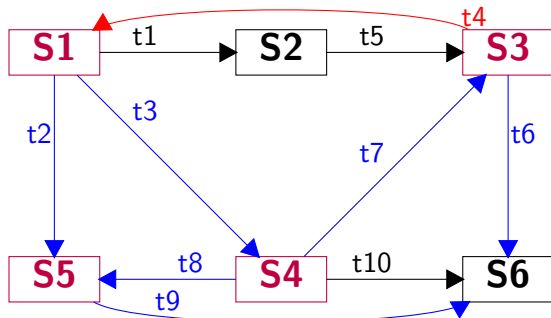
Adapting Definitions
Slicing Algorithms

Conclusion

Starting with slicing criterion t_6, t_9 we

1. add t_8 and t_7
2. add t_2 and t_3

Finding Least Set Satisfying WCC: Example



Starting with slicing criterion t_6, t_9 we

1. add t_8 and t_7
2. add t_2 and t_3
3. add t_4

Motivating Slicing

Deterministic Setting

Goal
Methods

Non-Determinism

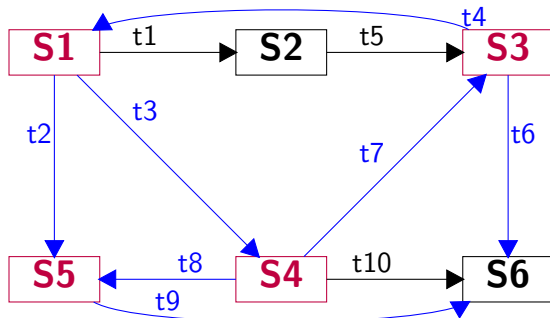
Goal
Method

EFSM Development

Adapting Definitions
Slicing Algorithms

Conclusion

Finding Least Set Satisfying WCC: Example



Motivating Slicing

Deterministic Setting

Goal
Methods

Non-Determinism

Goal
Method

EFSM Development

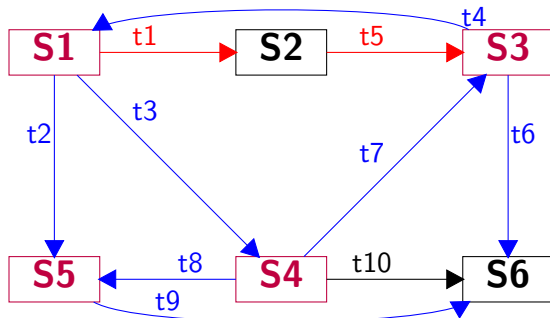
Adapting Definitions
Slicing Algorithms

Conclusion

Starting with slicing criterion t_6, t_9 we

1. add t_8 and t_7
2. add t_2 and t_3
3. add t_4

Finding Least Set Satisfying WCC: Example



Starting with slicing criterion t_6, t_9 we

1. add t_8 and t_7
2. add t_2 and t_3
3. add t_4
4. add t_1

Motivating Slicing

Deterministic Setting

Goal
Methods

Non-Determinism

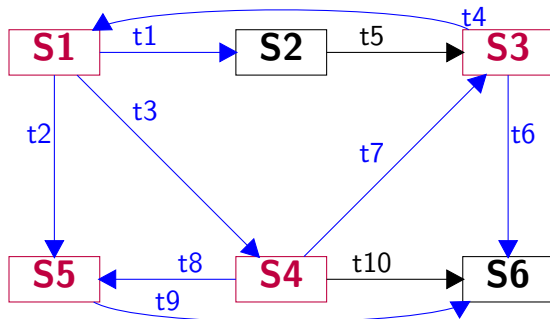
Goal
Method

EFSM Development

Adapting Definitions
Slicing Algorithms

Conclusion

Finding Least Set Satisfying WCC: Example



Motivating Slicing

Deterministic Setting

Goal
Methods

Non-Determinism

Goal
Method

EFSM Development

Adapting Definitions
Slicing Algorithms

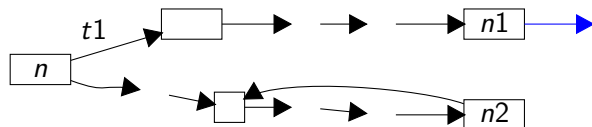
Conclusion

Starting with slicing criterion t_6, t_9 we

1. add t_8 and t_7
2. add t_2 and t_3
3. add t_4
4. add t_1

Finding Least Set Satisfying SCC: Theory

Observe: if WCC holds but SCC does **not** then we have



and $t1$ will belong to any superset satisfying SCC.

For a given slicing criterion, there thus exists a **least** superset that satisfies SCC (and is closed under DD) and we can write an algorithm to iteratively find this set:

- ▶ from the observables, do a **backwards** breadth-first search through transitions **not** in slice set.
- ▶ if some node n is reached from **two** observables, **or may avoid its observable**, then add transition(s) towards observable from n .

Running time: **quadratic** in number of transitions (excluding time to compute DD, including time to precompute which nodes may avoid which nodes).

Motivating Slicing

Deterministic Setting

Goal
Methods

Non-Determinism

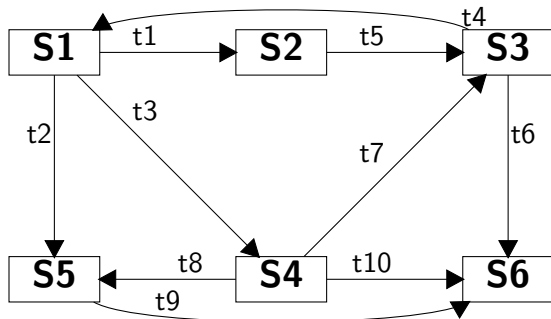
Goal
Method

EFSM Development

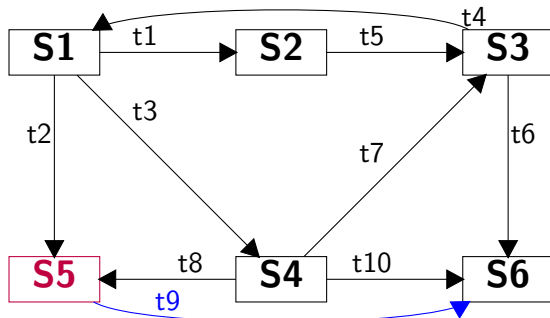
Adapting Definitions
Slicing Algorithms

Conclusion

Finding Least Set Satisfying SCC: Example



Finding Least Set Satisfying SCC: Example



Starting with slicing criterion t_9 we

Motivating Slicing

Deterministic Setting

Goal
Methods

Non-Determinism

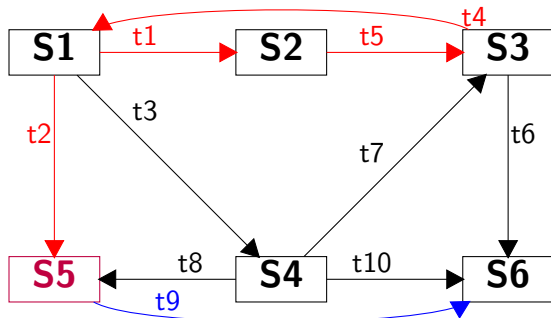
Goal
Method

EFSM Development

Adapting Definitions
Slicing Algorithms

Conclusion

Finding Least Set Satisfying SCC: Example



Starting with slicing criterion t_9 we

1. add t_2

Motivating Slicing

Deterministic Setting

Goal
Methods

Non-Determinism

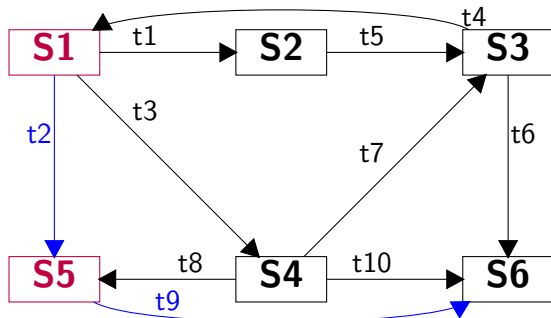
Goal
Method

EFSM Development

Adapting Definitions
Slicing Algorithms

Conclusion

Finding Least Set Satisfying SCC: Example



Motivating Slicing

Deterministic Setting

Goal
Methods

Non-Determinism

Goal
Method

EFSM Development

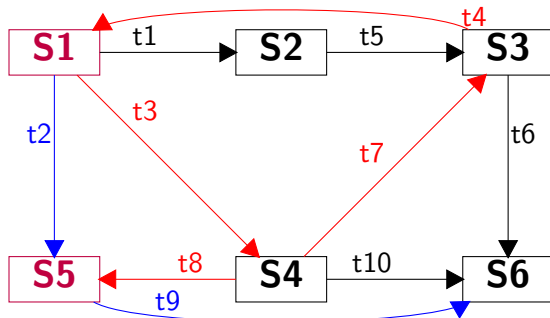
Adapting Definitions
Slicing Algorithms

Conclusion

Starting with slicing criterion t_9 we

1. add t_2

Finding Least Set Satisfying SCC: Example



Starting with slicing criterion t_9 we

1. add t_2
2. add t_8

Motivating Slicing

Deterministic Setting

Goal
Methods

Non-Determinism

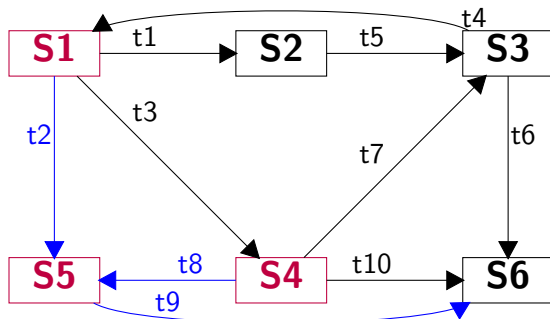
Goal
Method

EFSM Development

Adapting Definitions
Slicing Algorithms

Conclusion

Finding Least Set Satisfying SCC: Example



Starting with slicing criterion t_9 we

1. add t_2
2. add t_8

Motivating Slicing

Deterministic Setting

Goal
Methods

Non-Determinism

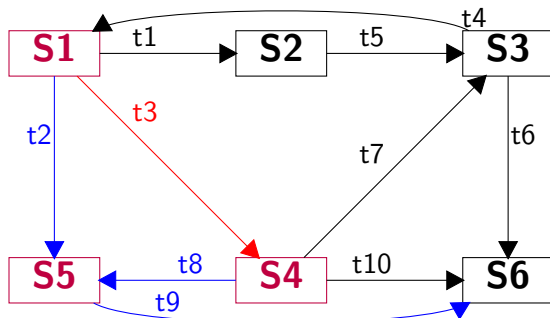
Goal
Method

EFSM Development

Adapting Definitions
Slicing Algorithms

Conclusion

Finding Least Set Satisfying SCC: Example



Starting with slicing criterion t_9 we

1. add t_2
2. add t_8
3. add t_3

Motivating Slicing

Deterministic Setting

Goal
Methods

Non-Determinism

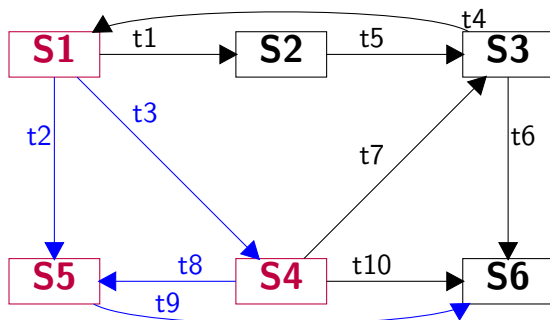
Goal
Method

EFSM Development

Adapting Definitions
Slicing Algorithms

Conclusion

Finding Least Set Satisfying SCC: Example



Starting with slicing criterion t_9 we

1. add t_2
2. add t_8
3. add t_3

Motivating Slicing

Deterministic Setting

Goal
Methods

Non-Determinism

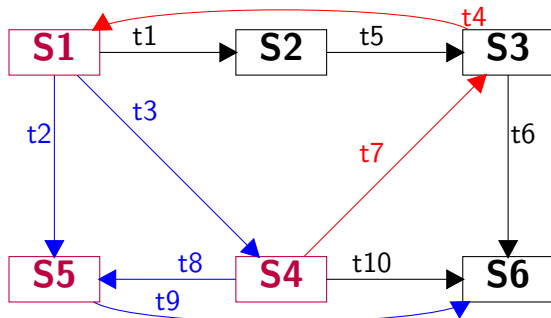
Goal
Method

EFSM Development

Adapting Definitions
Slicing Algorithms

Conclusion

Finding Least Set Satisfying SCC: Example



Starting with slicing criterion t_9 we

1. add t_2
2. add t_8
3. add t_3
4. add t_7

Motivating Slicing

Deterministic Setting

Goal
Methods

Non-Determinism

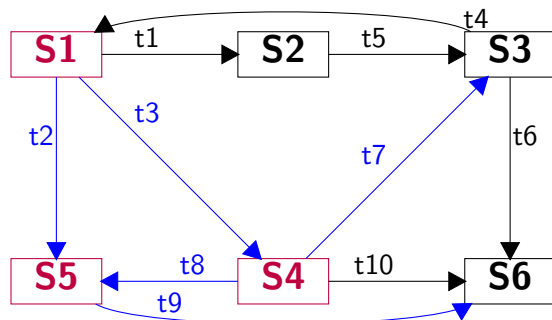
Goal
Method

EFSM Development

Adapting Definitions
Slicing Algorithms

Conclusion

Finding Least Set Satisfying SCC: Example



Starting with slicing criterion t_9 we

1. add t_2
2. add t_8
3. add t_3
4. add t_7

Motivating Slicing

Deterministic Setting

Goal
Methods

Non-Determinism

Goal
Method

EFSM Development

Adapting Definitions
Slicing Algorithms

Conclusion

We would like to see if our ideas could be extended to handle **concurrent programs**, taking inspiration from [Hatcliff et al, SAS 1999] which

- ▶ considers **multi-threading** with synchronization through monitors
- ▶ defines **various dependencies**, not just data and control but also *divergence*, *interference*, *synchronization*, *ready*
- ▶ proposes **bisimulation** as correctness property but does **not** work out the details

Motivating Slicing

Deterministic Setting

Goal
Methods

Non-Determinism

Goal
Method

EFSM Development

Adapting Definitions
Slicing Algorithms

Conclusion