Correctly Slicing Finite State Machines

Torben Amtoft\(^1\), Kelly Androutsopoulos\(^2\), and David Clark\(^3\)

\(^1\) Department of Computer Science, Kansas State University, Manhattan KS 66506, USA
\(^2\) Department of Computer Science, Middlesex University, London, NW4 4BT, United Kingdom
\(^3\) CREST, Department of Computer Science, University College London, WC1E 6BT, United Kingdom.

Abstract. We consider slicing extended finite state machines. Extended finite state machines (EFSMs) combine a finite state machine with a store and can model a range of computational phenomena, from high-level software to cyber-physical systems. They are essentially interactive and may be nondeterministic so standard techniques for slicing, developed for control flow graphs of programs with a functional semantics, are not immediately applicable. They may be placed in parallel but in this paper we consider them only as stand-alone models. This paper provides the first proofs of correctness for control and data dependence based slicing of EFSMs.

We express the semantic correctness of slicing as follows. We demand that the sliced machine simulates the original machine, so an “observable” step taken by the latter can also be done by the former. In the other direction, we cannot hope for a perfect simulation, but demand that for each observable step by the sliced machine, either the original machine simulates it or (i) it gets stuck, or (ii) it loops. To ensure correctness, it suffices to demand that the set of transitions in the slice satisfies two conditions: it must be closed under the well-known notion of data dependence, and it must have the “weak commitment” property highlighted by Danicic et alia. If the slice also has the “strong commitment” property, the case (ii) above can be ruled out, meaning that the original machine will simulate the sliced machine except that it may get stuck. We provide statements of correctness for each of the properties “weak commitment” and “strong commitment”. We also give algorithms to compute the least sets satisfying each of the properties “weak commitment” or “strong commitment”.

1 Introduction

The goal of program slicing is to remove the parts of a program that are irrelevant in a given context [Wei84,Tip95]. Slicing has a wide variety of applications that include compiler optimizations, debugging, model checking and comprehension [Tip95].

One might imagine, given the long history of both slicing and finite state machines, that the correct slicing of finite state machines is a closed question or, if not closed, a relatively uninteresting one. However, neither consideration is true when aiming to produce correct, executable slices, that is to say reduced “programs” for which any nodes not relevant to the restricted semantics have been removed and the “program”
has been “rewired” to maintain executability. Once extended with a store, i.e. a set of
program variables, that can be updated via commands attached to transitions between
the primitive states, Extended Finite State Machines (EFSMs) can directly model a wide
variety of software systems. What makes them particularly interesting from a theoretical
perspective is that their interactive, atomic event driven semantics offers a simplified
laboratory in which to analyse the correctness of slicing in the presence of interaction
with an environment.

Work on program slicing has largely focused on programs with a functional seman-
tics, and notions of correctness assume interaction only at input. Consequently, slic-
ing interactive programs using traditional algorithms produces incorrect slices because
the input sequence of a program is part of the implicit state and not described by any
variable. A correct algorithm for programs with a single exit point has been described
previously [SHD97]. The limitation in that work is the single exit assumption on which
the definition of control dependence rests.

Like EFSMs, modern programs may have multiple exit points or even no exit points
so as to program indefinitely non-terminating, reactive systems. This has required the
development of new definitions of control dependence for programming languages
[RAB+05, RAB+07, Amt08]. Definitions of control dependency began to proliferate un-
til Danicic and others extracted order out of chaos by focusing on the desirable proper-
ties of the resulting dependence relations rather than the control dependence definitions
themselves [DBH+11]:

– **Weak Commitment Closure (WCC)** means that each node has *at most one* “next
observable”, where an observable is a node relevant to the slicing criterion.

– **Strong Commitment Closure (SCC)** in addition demands that from a node that has
a “next observable”, there is no way to infinitely avoid that observable.

EFSMs, being models rather than programs, also allow non-determinism. An early
attempt to handle non-determinism by Hatcliff et alia considers a multi-threaded lan-
guage but does not prove any correctness results [HCD+99].

Any correct algorithm for the slicing of finite state model based formalisms has to
consider all of the above issues: interaction with the environment; non-determinism; and
flexible control dependence. This paper is the first that successfully and generally deals
with all these issues to produce correct algorithms for slicing EFSMs. In the process,
we establish that the commitment closure approach to control dependence of Danicic
et alia is flexible enough to provide correctness for interactive programming languages
with modern control structures and non-determinism. For EFSMs it appears natural
to let slices be sets of transitions rather than sets of nodes (atomic states). The major
restriction is that we only consider stand-alone EFSMs that do not generate events.

2 Related Work

Relevant related work in the arena of program slicing has already been mentioned in the
introduction. The state of the art in slicing any form of state based models (prior to this
paper) is represented in a recent survey [ACH+13a]. A critical line of research leading
to this paper has been contributions on slicing correctly in the presence of interaction with environmental events.

Ganapathy et alia defined correctness for slicing reactive programs, where events (in particular generated events) are part of the slicing criterion [GR02]. However, not all interactive systems generate events, and the drawback of their work is the size of their slices: they include all states and transitions that reach the event of interest. Labbe et alia presented polynomial algorithms for slicing Input/Output Symbolic Transition Systems (IOSTSs) [LG08]. However, they consider only control dependence that is sensitive to non-termination and, unlike WCC is not backwardly compatible with traditional forms of slicing.

Korel et alia made a key observation on how stuttering event sequences affect the correctness of slicing finite state machines [KSTV03]. A stuttering event sequence is a sequence of events whereby not all events trigger transitions. A non-stuttering event sequence may become stuttering after slicing. In Section 4 we explain this using Example 1. Korel and co-authors define a new notion of correctness to consider stuttering event sequences upon which we build. In fact our approach can be viewed as a formalisation and validation of the correctness notion in their paper [KSTV03] but extended to possibly non-terminating systems.

## 3 Extended Finite State Machines (EFSMs)

An Extended Finite State Machine (EFSM) $M$ is a finite state automaton where transitions have guards, may manipulate a common store, and may be triggered by external events. We shall formally define their syntax (Section 3.1) and semantics (Section 3.3), and also define the notion of a slice (Section 3.2). In Section 4 we shall discuss what it means for a slice to be semantically correct, and propose a suitable definition.

### 3.1 Syntax of EFSMs

An EFSM is a tuple $(\hat{S}, \hat{T}, \hat{E}, \hat{V})$ where $\hat{S}$ is a finite set of states, $\hat{T}$ is a finite set of transitions, $\hat{E}$ is a finite set of events (some of which may take a parameter), and $\hat{V}$ is a finite set of variables. A state $n$ (or $m$) $\in \hat{S}$ is considered atomic, and one state in $\hat{S}$ may be designated as the initial state. A transition $t$ (or $u$) $\in \hat{T}$ has a source state $S(t) \in \hat{S}$, a target state $T(t) \in \hat{S}$, and a label of the form $\hat{e}[g]/a$. Here $\hat{e} = E(t)$ is either of the form $e$ where $e$ is an unparametrized event in $\hat{E}$, of the form $e(v_b)$ where $e$ is a parametrized event in $\hat{E}$ and $v_b$ is a bound variable (not occurring in $\hat{V}$), or of the form $\varepsilon$ (the transition is “spontaneous”).

Also, $g = G(t)$ is a guard and $a$ is an action which we may assume is either of the form $v := A$ with $v \in \hat{V}$ or $\text{skip}$; here $g$ and $A$ may refer to the variables in $\hat{V}$ and also to $v_b$ if it exists. We define $D(t)$, the variables used by $t$, as $\{v\}$ if $a$ is $v := A$ and $\emptyset$ if $a$ is $\text{skip}$; we define $A(t)$, the (arithmetic) expression mentioned in $t$, as $A$ if $a$ is $v := A$ and $0$ (arbitrarily) if $a$ is $\text{skip}$. We define $U(t)$, the variables used by $t$, as $(fv(G(t)) \cup fv(A(t))) \setminus bv(\hat{e})$ where $bv(e(v_b)) = \{v_b\}$ and $bv(e) = bv(\varepsilon) = \emptyset$. When depicting an EFSM, all parts of a label are optional: one may omit $\hat{e}$ if it is $\varepsilon$, omit $g$ if
Fig. 1: Our running example EFSM.

it is true, and omit a if it is skip. A transition where all parts of its label are empty is called an \(\varepsilon\)-transition. Note that we do not allow for transitions that produce events.

Fig. 1 depicts a non-trivial EFSM (with the initial state \(S_1\)) that shall serve as our main running example; since our general focus is on control aspects rather than on data aspects (which are already well understood) we ignore the labels of the transitions.

We say that \(t\) is self-looping if \(S(t) = T(t)\), and that \(u\) is a successor of \(t\) if if \(S(u) = T(t)\). In Fig. 1, \(t_{11}\) is self-looping, and \(t_5\) is a successor of \(t_1\).

We say that \([t_1..t_k]\) \((k \geq 0)\) is a path (of length \(k\)) if for all \(j \in 1...k - 1\), \(t_{j+1}\) is a successor of \(t_j\). If \(k \geq 1\), we say that \([t_1..t_k]\) is a path from \(S(t_1)\) to \(T(t_k)\). We say that \(n\) occurs in \([t_1..t_k]\) if there exists \(j \in 1...k\) such that \(n = S(t_j)\) or \(n = T(t_j)\). With \(L\) as set of transitions, we say that a path \([t_1..t_k]\) is outside \(L\) iff \(t_j \notin L\) for all \(j \in 1...k\). In Fig. 1, \([t_5, t_4, t_3]\) is a path from \(S_2\) to \(S_4\) where the states \(S_2, S_3, S_1\) and \(S_4\) occur but which is outside say \(\{t_1, t_6\}\). We say that \([t_1..t_k..]\) is an infinite path from (or say that it “loops from”) \(S(t_1)\) if \(t_{j+1}\) is a successor of \(t_j\) for all \(j \geq 1\); we say that the path avoids \(n\) if \(n \neq S(t_j)\) for all \(j \geq 1\). In Fig. 1, the path \([t_4, t_3, t_7, t_4, t_3, t_7, \ldots]\) is an infinite path from \(S_3\) that avoids all of \(S_2, S_5\), and \(S_6\).

### 3.2 Slicing EFSMs

In general, a slice is a subpart of the source program. In our setting, this raises the question: should we consider a slice to be a subset of the states (together with the transitions between these states), or to be a subset of the transitions (together with the states involved in these transitions). The former option is a special case (where a slice contains a transition iff it contains its source and also its target) of the latter which we therefore choose:

**Definition 1 (Slice Set)** A slice set for an EFSM \((\hat{S}, \hat{T}, \hat{E}, \hat{V})\) is a subset of \(\hat{T}\).

Our definitions will often be implicitly parameterized with respect to a given EFSM, and with respect to a fixed slice set which is often called \(L\) while its members are called “observables”. We shall aim for \(L\) to be the least set (cf. Section 6) that contains the slicing criterion, that is the transition(s) that we are ultimately interested in, and in
addition satisfies certain conditions (given in Section 5) designed to ensure semantic correctness (as defined in Section 4).

As will be formalized in Section 3.3, slicing amounts to keeping the transitions in $L$ as they are, whereas transitions not in $L$ are replaced by $\varepsilon$-transitions.

![Fig. 2: A simplified EFSM for ATM operations.](image1)

![Fig. 3: The result of slicing the EFSM from Fig. 2 with respect to $T_1$ and $T_3$ and $T_4$.](image2)

**Example 1** Consider the EFSM in Fig. 2 which models a simple ATM system, and assume that the slicing criterion consists of $T_4$. Then the slice set $L$ must also contain $T_1$ and $T_3$ (as these transitions provide definitions of the $x$ which is used by $T_4$). The result of slicing wrt. $L$ is the EFSM depicted in Fig. 3.

The presence of $\varepsilon$-transitions may enable the further optimization of slices [ACH+13b]. For example, in Fig. 3, since $S_3$ is reachable by an $\varepsilon$-transition from $S_2$ and from $S_4$, these three states may be merged. We shall not further address this optimization.

### 3.3 Semantics of EFSMs

We shall present a semantics that facilitates reasoning about slicing. Our development is relative to a given EFSM $(\hat{S}, \hat{T}, \hat{E}, \hat{V})$ and a fixed slice set $L$.

A configuration $C$ is a pair $(n, s)$ where $n \in \hat{S}$ is a state and $s$ is a store which maps variables $v, w \in \hat{V}$ (we shall also use the letter $w$ to range over variables) to values. The domain of values is unspecified but we assume an expression language with $[A]s$ denoting the value resulting from evaluating expression $A$ in store $s$; similarly we assume a guard language with $[B]s \in \{true, false\}$ denoting the value of the boolean expression $B$ wrt. store $s$.

In our definitions, we shall use the subscript 1 to refer to the original EFSM, and subscript 2 to refer to the sliced EFSM. Thus $E_1(t) = E(t)$ and $G_1(t) = G(t)$, etc. If
where each element of an event sequence $t$ contains $z$ to denote that $t$ is a value; the empty event sequence is denoted $\varepsilon$. We shall use Example 1, where $\mathcal{L}$ contains $T_1$ and $T_3$ and $T_4$, to illustrate the definitions. First a one-step move:

**Definition 2** We write
$$i \vdash t : (n, s) \xrightarrow{E} (n', s')$$
to denote that $(n, s)$ in the $i$-semantics $(i = 1, 2)$ through transition $t$ moves to $(n', s')$ while consuming the event sequence $E$ (which will be either empty or a singleton). This happens when $t$ is such that all the conditions listed below hold:

- $S(t) = n$ and $T(t) = n'$
- either $E_n(t) = \varepsilon = E$ or there exists $e \in \bar{E}$ such that either $E_n(t) = e = E$ (for some $z$ and $v_b$)
- $[g]s = true$
- if $D_i(t) = \emptyset$ then $s' = s$ but if $D_i(t)$ is a singleton $\{v\}$ then $s' = s[v \mapsto [A]s]$

where $g$ and $A$ are defined as follows: if $E = e(z)$ and $E_n(t) = e(v_b)$ then $g = G_i(t)[z/v_b]$ and $A = A_i(t)[z/v_b]$ but otherwise $g = G(t)$ and $A = A_i(t)$.

In Example 1, when $E =$ deposit(20) we have (for $i = 1, 2$ and for all stores $s$):
$$i \vdash T_3 : (S_3, s) \xrightarrow{E} (S_4, s[x \mapsto s(x) + 20]).$$

Next a multi-step move:

**Definition 3** We write $i \vdash \pi : C \xrightarrow{E} C'$ iff with $\pi = [t_1..t_{k-1}]$ there exists $C_1...C_k$

$k \geq 1$ with $C = C_1$ and $C'= C_k$ such that for all $j \in 1...k-1$, $i \vdash t_j : C_j \xrightarrow{E_j} C_{j+1}$ for some $E_j$ where $E = E_1...E_{k-1}$.

In Example 1, we have (for all stores $s$)

1. $[T_3, T_5, T_4] : (S_3, s) \xrightarrow{E_1} (S_4, s[x \mapsto s(x) + 20])$ where $E_1 =$ deposit(30), done, withdraw(10)

2. $[T_3, T_5, T_4] : (S_3, s) \xrightarrow{E_2} (S_4, s[x \mapsto s(x) + 20])$ where $E_2 =$ deposit(30), withdraw(10)

We are often interested in unobservable moves:

**Definition 4** We write $i \vdash C \xrightarrow{E} C'$ iff $i \vdash \pi : C \xrightarrow{E} C'$ where $\pi$ is outside $\mathcal{L}$.

Of special interest is moves where all but the last step are unobservable:

**Definition 5** We write $i \vdash t : C \xrightarrow{E} C'$ if $t \in \mathcal{L}$ and there exists $C_1$, and $E_1, E_2$ with $E = E_1E_2$, such that $i \vdash C \xrightarrow{E_1} C_1$ and $i \vdash t : C_1 \xrightarrow{E_2} C'$.

**Example 2** In Example 1 we have (for all $s$ with $s(l) = "\text{"}$)

1. $[T_3] : (S_2, s) \xrightarrow{E_1} (S_4, s[y \mapsto s(x) + 20, l \mapsto \text{English}])$

2. $[T_3] : (S_2, s) \xrightarrow{E_2} (S_4, s[x \mapsto s(x) + 20])$

where $E_1 =$ language(English), deposit(20) and $E_2 =$ deposit(20).
A configuration is *stuck* if no transitions apply:

**Definition 6** We say that \((n, s)\) is \(i\)-stuck if \([G_i(t)]s = \text{false for all } t\text{ with } S(t) = n\).

In Example 1, \((S2, s)\) is stuck if \(s(l) \neq \text{""} \).

We shall be interested in moves that never reach a certain state \(m\), either because they get stuck or because they never terminate:

**Definition 7** We say that \((n, s)\) gets \(i\)-stuck avoiding \(m\) if there exists \(n' \neq m\) and \(s'\) such that \((n', s')\) is \(i\)-stuck, and there exists \(\pi\) where \(m\) does not occur such that \((n', s') \xrightarrow{E} (n', s'))\).

**Definition 8** We say that \(C\) \(i\)-loops avoiding \(m\) if for all \(j \geq 1\) there exists \(t_j\) where \(m\) does not occur such that for all \(k \geq 1\), for some \(E_k, C_k\): \(i \vdash \pi : (n, s) \xrightarrow{E_k} (n', s')\).

In Example 1, for all \(s\) and for \(i = 1, 2\), it will be the case that \((S3, s)\) \(i\)-loops avoiding \(S2\), but *not* that \((S3, s)\) \(i\)-loops avoiding \(S4\).

### 4 Slicing an EFSM: What does Correctness Mean?

We shall now develop definitions of what it means for slicing, with respect to a given slice set \(L\), to be semantically correct. Naively, we would like that each observable move by the original EFSM can be done by the sliced EFSM, and vice versa:

\[
\begin{align*}
\text{if } 1 & \vdash^t C \xrightarrow{E} C' \text{ then } 2 \vdash^t C \xrightarrow{E} C' \\
\text{if } 2 & \vdash^t C \xrightarrow{E} C' \text{ then } 1 \vdash^t C \xrightarrow{E} C'
\end{align*}
\]

But this demand is too restrictive, for several reasons to be detailed in the following.

Let us first look at Example 1, and the event sequence

\[
\text{enterCard, withdraw(10), deposit(20), withdraw(30)}
\]

which for the sliced EFSM (Figure 3) gives \(x\) the value 80, after several spontaneous transitions from \(S4\) to \(S3\) (and one from \(S2\) to \(S3\)). For the original EFSM (Figure 2), this sequence can be “padded” with \textit{dones} (and \textit{languages}) to give an event sequence which also for the original EFSM gives \(x\) the value 80.

Together with Example 2, this suggests that we should phrase correctness as follows

\[
\begin{align*}
\text{if } 1 & \vdash^t C \xrightarrow{E_1} C' \text{ then } 2 \vdash^t C \xrightarrow{E_2} C' \text{ where } E_2 = \text{filter}_L(E_1), \text{ and} \\
\text{if } 2 & \vdash^t C \xrightarrow{E_2} C' \text{ then } 1 \vdash^t C \xrightarrow{E_1} C' \text{ for some } E_1 \text{ with } E_2 = \text{filter}_L(E_1). \\
\end{align*}
\]

Here \(\text{filter}_L(E)\) returns a subsequence of \(E\) which includes \(e\) or \(e(z)\) iff there exists \(t \in L\) with \(E(t) = e\) or \(E(t) = e(v_b)\).

In effect, we assume that the sliced EFSM executes with a “skip semantics” when given an event that is sliced away; the sliced EFSM will ignore such events, rather than block on them. On the other hand, we can *not* allow to assume a general skip semantics.

Consider the event sequence: \text{enterCard, language("English")},
withdraw(10), deposit(20), done, withdraw(30) which will take the sliced EFSM (Figure 3) from $S_1$ to $S_4$, with $x$ as 80 since (skipping language ("English") and done) it can consume deposit(20) after a spontaneous transition from $S_4$ to $S_3$. This sequence will also take the original EFSM (Figure 2) from $S_1$ to $S_4$, provided it is allowed to skip the deposit(20) event, but then $x$ will end up being 60.

We conclude that our correctness results will apply to a semantics where an EFSM blocks on events that may be expected at some point, but skips events that can never be expected. Our approach can be viewed as a formalization and validation of the correctness notion presented by Korel et al. [2003] in [KSTV03].

For consistency, we need the following requirement: If $e \in \hat{E}$ is such that $E(t_1) \in \{ e, e(v_b) \}$ and also $E(t_2) \in \{ e, e(v_b) \}$ then either $t_1, t_2 \in \mathcal{L}$ or $t_1, t_2 \notin \mathcal{L}$.

Our tentative correctness demand is still too restrictive, as the sliced EFSM has to produce exactly the same store as the original. But since slicing removes transitions that are not “relevant”, the sliced EFSM is likely to disagree with the original EFSM on variables that are not relevant:

Definition 9 We say that $v$ is relevant for $t$ wrt. $\mathcal{L}$, written $v \in Rv(t)$ (or $v \in RV(t)$ when $\mathcal{L}$ is given from context), iff there exists $t' \in \mathcal{L}$ such that $v \in U(t')$, and there exists a path $[t_1..t_k]$ with $k \geq 1$ and $t = t_1$ and $t' = t_k$ such that for all $j \in 1..k-1$, $v \notin D(t_j)$.

For a state $n$ we define $RV_{\mathcal{L}}(n) = \bigcup_{S(t)=n} RV_{\mathcal{L}}(t)$.

Observe that in Fig 3, $x$ is not relevant for $T_1$ (as $x$ is defined there) and thus not for $S_1$, but $x$ is relevant for $T_3$ and $T_4$ (and thus also for $S_3$) since it is used there.

Definition 10 $(n_1, s_1) \not\sim (n_2, s_2)$ iff $n_1 = n_2$ and $s_1(v) = s_2(v)$ for all $v \in RV(n_1)$.

We can now phrase the property that no behavior is lost when slicing; the sliced EFSM can do anything the original can do:

Definition 11 (Completeness) Slicing with respect to a slice set $\mathcal{L}$ is complete iff the following property holds for all $C_1, C_2$ with $C_1 Q C_2$:

\[
\text{if } 1 \vdash^t C_1 \overset{E_1}{\rightarrow} C_1' \text{ then } 2 \vdash^t C_2 \overset{E_2}{\rightarrow} C_2' \text{ for some } C_1', E_2, C_2' \text{ with } C_1' Q C_2', \text{ and } E_2 = \text{filter}_\mathcal{L}(E_1).
\]

The converse property, that the sliced EFSM cannot do more than what the original can do, may appear desirable but will severely restrict the ability to do non-trivial slicing. For example, in Figs 2 and 3, if the initial store maps 1 (which is relevant for $S_2$) to “Spanish” then the original EFSM will be stuck at $S_3$, while the sliced EFSM will still move to $S_4$. We shall therefore settle for a more relaxed version of correctness:

Definition 12 (Strong Soundness) Slicing with respect to a slice set $\mathcal{L}$ is strongly sound iff the following property holds for all $C_1, C_2$ with $C_1 Q C_2$:

\[
\text{if } 2 \vdash^t C_2 \overset{E_2}{\rightarrow} C_2' \text{ then either}
\]

- $1 \vdash^t C_1 \overset{E_1}{\rightarrow} C_1'$ for some $C_1', E_1$ with $C_1' Q C_2'$ and $E_2 = \text{filter}_\mathcal{L}(E_1)$, or

- $C_1$ gets $l$-stuck avoiding $S(t)$. 
In Figs. 2 and 3, for all \( s \) there exists \( s' \) such that \( 2 \vdash T_4 (S_2, s) \Rightarrow \rightarrow (S_4, s') \), but if \( s(1) \neq "\) then \((S_2, s)\) gets stuck avoiding \( S(T_4) = S_3 \).

In many situations, we may want a somewhat weaker version of correctness:

**Definition 13 (Weak Soundness)** Slicing with respect to a slice set \( L \) is weakly sound iff the following property holds for all \( C_1, C_2 \) with \( C_1 \sqsubseteq C_2 \):

if \( 2 \vdash t C_2 \Rightarrow \rightarrow C_2' \) then either

- \( 1 \vdash t C_1 \Rightarrow \rightarrow C_1' \) for some \( C_1', E_1 \) with \( C_1' \sqsubseteq C_2 \) and \( E_2 = \text{filter}_L(E_1) \), or
- \( C_1 \) gets 1-stuck avoiding \( S(t) \), or
- \( C_1 \) I-loops avoiding \( S(t) \).

For example, assume that in Fig. 2 we add a self-looping transition at \( S_2 \), with guard \( l = "\text{English}" \) and action \( \text{skip} \). We might like to slice away that transition but this would violate strong soundness, whereas weak soundness would permit it.

## 5 Slicing an EFSM: How to get Correctness?

We shall develop conditions on the slice set \( L \) that ensure completeness and/or strong/weak soundness. In Section 5.1 we introduce the standard notion of data dependence which suffices for completeness. For soundness, in addition to data dependence we need two crucial conditions: in Section 5.3 we introduce the condition “weak commitment closed” (WCC) which suffices for weak soundness, and in Section 5.4 we introduce the condition “strong commitment closed” (SCC) which ensures strong soundness; both are expressed using the notion of “next observable” introduced in Section 5.2.

### 5.1 Data Dependence

Our definition is standard except that it relates transitions rather than states:

**Definition 14 (Data Dependence)** We say that \( t' \) is data dependent on \( t \), written \( t \rightarrow \text{dd} t' \), iff there exists a variable \( v \in D(t) \cap U(t') \) and a path \([t_1..t_k]\) \((k \geq 0)\) from \( T(t) \) to \( S(t') \) such that for all \( j \in 1..k \), \( v \notin D(t_j) \).

For example, in Fig. 2, \( T_4 \) is data dependent on \( T_1 \) and \( T_3 \), and also on itself which is uninteresting for the purpose of computing a slice set that is closed under \( \rightarrow \text{dd} \):

**Definition 15** Say \( L \) is closed under \( \rightarrow \text{dd} \) iff \( t \in L \) whenever \( t \rightarrow \text{dd} t' \) and \( t' \in L \).

**Theorem 1 (Completeness)** Assume that \( L \) is closed under \( \rightarrow \text{dd} \). If

- \( 1 \vdash t C_1 \Rightarrow \rightarrow C_1' \) and
- \( C_1 \sqsubseteq C_2 \)

then there exists \( C_2' \) such that with \( E_2 = \text{filter}_L(E_1) \) we have

- \( 2 \vdash t C_2 \Rightarrow \rightarrow C_2' \) and
- \( C_1' \sqsubseteq C_2' \)
5.2 Next Observable

Following recent trends in the theoretical foundation of slicing [Amt08,DBH+11] we shall employ the concept of “next observable”, but in our setting where a slice set consists of transitions rather than states it needs to be phrased in a somewhat different way:

Definition 16 For a slice set $\mathcal{L}$, for each state $n$ we define $\text{obs}_\mathcal{L}(n)$ (written $\text{obs}(n)$ when $\mathcal{L}$ is given by the context) as the set of states $n'$ such that

- there exists $t \in \mathcal{L}$ with $S(t) = n'$, and
- there exists a path outside $\mathcal{L}$ from $n$ to $n'$.

For example, consider the EFSM in Fig. 1, and assume that $\mathcal{L} = \{t5, t6\}$. Then for all $n$ we have $\text{obs}_\mathcal{L}(n) \subseteq \{S(t5), S(t6)\} = \{S2, S3\}$. In particular, $\text{obs}_\mathcal{L}(S1) = \{S2, S3\}$ whereas $\text{obs}_\mathcal{L}(S2) = \{S2\}$ (since there is no path outside $\mathcal{L}$ from $S2$ to $S3$) but $\text{obs}_\mathcal{L}(S3) = \{S2, S3\}$ (since $[t4, t1]$ is a path outside $\mathcal{L}$ from $S3$ to $S2$).

5.3 Weak Commitment Closure

In order to obtain (weak) soundness, we cannot allow a set $\text{obs}_\mathcal{L}(n)$ to contain two (or more) states. To see this, consider Fig. 1 with $\mathcal{L} = \{t5, t6\}$ so that $\text{obs}_\mathcal{L}(S1) = \{S2, S3\}$. Then the sliced EFSM may move to $S3$ (through $\varepsilon$-transitions) and perform the observable transition $t6$, while such a move may be impossible for the original EFSM (if $G(t3)$ is false) which could instead move to $S2$ (if $G(t1)$ is true) and perform $t5$. This motivates the notion of “weak commitment closed” (WCC):

Definition 17 (WCC) We say that $\mathcal{L}$ satisfies WCC iff for each state $n$, $\text{obs}_\mathcal{L}(n)$ is either empty or a singleton.

We see that in Fig. 1, the set $\{t5, t6\}$ does not satisfy WCC.

Theorem 2 (Weak Soundness) Assume that $\mathcal{L}$ is closed under $\rightarrow_{\text{dd}}$ and satisfies WCC. If $2 \vdash C_2 \Rightarrow C'_2$ and $C_1 \models E_2$ then with $m = S(t)$ there are 3 possibilities:

1. there exists $C'_1$ with $C'_1 \models E_1$ and $E_1$ with $E_2 = \text{filter}_\mathcal{L}(E_1)$, such that $1 \vdash C_1 \Rightarrow C'_1$
2. $C_1$ gets 1-stuck avoiding $m$
3. $C_1$ 1-loops avoiding $m$.

5.4 Strong Commitment Closure

In order to obtain strong soundness, we cannot allow a state $n$ to be part of an infinite part that avoids $\text{obs}_\mathcal{L}(n)$. To see this, consider Fig. 1 but this time with $\mathcal{L} = \{t9\}$. This trivially satisfies WCC as $\text{obs}_\mathcal{L}(n)$ will always be either $\emptyset$ or $\{S5\}$, in particular $\text{obs}_\mathcal{L}(S1) = \{S5\}$ but there is an infinite path from $S1$ that avoids $S5$. Thus the sliced EFSM may move to $S5$ and perform the observable transition $t9$, while such a move may be impossible for the original EFSM (if $G(t2)$ is false) which could instead (for certain values of the store) cycle infinitely between $S1$, $S2$, and $S3$. This would violate strong soundness, and motivates the notion of “strong commitment closed” (SCC):
**Definition 18** We say that \( L \) satisfies SCC iff for each state \( n \), either

- \( \text{obs}_L(n) \) is empty, or
- \( \text{obs}_L(n) \) is a singleton \( n' \), and there is no infinite path from \( n \) that avoids \( n' \).

Clearly a slice set that satisfies SCC will also satisfy WCC.

**Theorem 3 (Strong Soundness)** Assume \( L \) is closed under \( \rightarrow_{\text{dd}} \) and satisfies SCC. If \( 2 \vdash t \) then with \( m = S(t) \) there are 2 possibilities:

1. there exists \( C'_1 \) with \( C'_1 \rightarrow C'_2 \) and \( E_1 \) with \( E_2 = \text{filter}_L(E_1) \), such that \( 1 \vdash t \) \( \rightarrow \)
2. \( C_1 \) gets 1-stuck avoiding \( m \).

### 6 Computing Least Slices

For a given EFSM, there may be many slice sets that satisfy WCC and are closed under data dependence.

**Example 3** Consider the EFSM given in Figure 4. If the slicing criterion is transition \( t_8 \), and \( t_8 \) is data dependent on \( t_3 \) and \( t_6 \) (but no other data dependencies exist), then a superset is closed under data dependence and satisfy WCC iff it contains \( t_2 \) and \( t_5 \). (That is, any of \( t_1 \), \( t_4 \) or \( t_7 \) may or may not be there, so there are 8 possible supersets.)

In Section 6.1, we shall present an algorithm that always returns the least such set. In the above example, this will be \( \{t_2, t_3, t_5, t_6, t_8\} \) which is constructed as follows: we first close \( \{t_8\} \) under data dependence which yields \( \{t_3, t_6, t_8\} \) (in bold in Fig. 4), so that the observable states (the sources of these transitions) are \( \{S_3, S_5, S_7\} \) (filled in Fig. 4); we then do a backwards search from these states and see that \( S_2 \) has two next observables \( (S_3 \text{ and } S_5) \) so we need to add \( t_2 \) and \( t_5 \), after which no more transitions need to be added. Note that even though also \( S_1 \) has two next observables, it is important that \( S_2 \) is considered first, as otherwise \( t_1 \) is needlessly added.

![Fig. 4](image-url) An EFSM where the current slice set \( \{t_3, t_6, t_8\} \) (in bold) does not satisfy WCC, as the corresponding set of observable states \( \{S_3, S_5, S_7\} \) (filled) allows some states \( (S_1 \text{ and } S_2) \) to have two next observables.

Similarly, for a given EFSM there may be many slice sets that satisfy SCC and are closed under data dependence.
Example 4 Consider the EFSM given in Figure 5. If the slicing criterion is transition $t_4$ (and there are no data dependencies), then the following supersets satisfy SCC:

$\{t_3, t_4\}$, $\{t_1, t_3, t_4\}$, $\{t_2, t_3, t_4\}$, and $\{t_1, t_2, t_3, t_4\}$.

In Section 6.2, we shall present an algorithm that always returns the least such set. In the above example, this will be $\{t_3, t_4\}$ which is constructed as follows: with the initial observable state being $S_3$ (filled in Fig. 5), we again do a backwards search from $S_3$ and see that $S_2$ can avoid $S_3$ and hence we need to add $t_3$, after which no more transitions need to be added. Note that even though also $S_1$ can avoid $S_3$, it is important that $S_2$ is considered first, as otherwise $t_1$ is needlessly added.

Fig. 5: An EFSM where the current slice set $\{t_4\}$ (in bold) does not satisfy SCC, as the corresponding set of observable states $\{S_3\}$ (filled) allows some states ($S_1$ and $S_2$) to avoid that observable.

To prepare for our algorithms, we now address how to ensure data dependence; we shall assume a pre-computed table $\mathcal{DD}$ such that $\mathcal{DD}(t, u)$ is true iff $t \rightarrow_{dd} u$ holds.

Lemma 19 The table $\mathcal{DD}$ can be computed in time $O(a^2)$ where $a$ is the number of transitions in the EFSM.

To do so, for each transition $t$ where $D(t)$ is non-empty and thus a singleton $v$, we do a depth-first search to find how far this definition “propagates”: we find the transitions $u$ that use $v$ and which can be reached thru transitions that do not redefine $v$.

We can use the table $\mathcal{DD}$ to add transitions while preserving the property of being closed under data dependence:

Lemma 20 Given a table $\mathcal{DD}$ such that $\mathcal{DD}(t, u)$ holds iff $t \rightarrow_{dd} u$, we can write a function $\mathcal{DD}_{\text{close}}$ that given a slice set $L$ closed under data dependence, and another slice set $L_2$ with $L \cap L_2 = \emptyset$, returns the smallest slice set $L'$ that contains $L \cup L_2$ and is closed under data dependence. The running time of that function is in $O(a, |L'| - |L|)$ where $a$ is the number of transitions in the EFSM.

6.1 Computing Least $\mathcal{WCC}$-Satisfying Slices

In Figure 6 we present an algorithm that for a given slice set $L$ returns the least superset of $L$ that satisfies $\mathcal{WCC}$ and is closed under data dependence. The algorithm works by adding transitions to $L$ until no new transitions need to be added. In each iteration, the algorithm computes in $B$ the states that are sources of transitions in $L$, and does from $B$ a backwards breadth-first search thru transitions not in $L$, with $V$ being the states that
have been visited so far. For each $n \in V$, the array entry $X[n]$ is defined as the state in $B$ that can be reached (thru transitions not in $L$) from $n$; that state will be a next observable of $n$. The current frontier of the search is called $C$, the exploration of which builds up $C_{\text{new}}$, the next frontier. If there is a transition $t \notin L$ to a node $m \in C$, from a node $n$ that already belongs to $V$ but with $X[n] \neq X[m]$, we detect that $n$ has two next observables and we infer that $t$ has to be included in the slice set (as motivated in Example 3 and justified by the correctness result).

---

**Input:** An EFSM $M$; a set $L$ of transitions in $M$

**Output:** the least $\to_{\text{dd}}$-closed superset of $L$ that satisfies $WCC$

```plaintext
1. $L := \text{DDclose}(\emptyset, L)$
2. repeat
   /* $L$ is closed under $\to_{\text{dd}}$, and a subset of any $\to_{\text{dd}}$-closed superset of $L$ that satisfies $WCC$ */
   3. $L_{\text{new}} := \emptyset$
   4. $B := \{ n \mid \exists t \in L : n = S(t) \}$
   5. foreach $n \in B$ do
      6. $X[n] := n$
   7. $V := B$
   8. $C := B$
   9. while $C \neq \emptyset$ and $L_{\text{new}} = \emptyset$ do
      10. $C_{\text{new}} := \emptyset$
      11. foreach $m \in C$ do
          12. foreach transition $t \notin L$ with $T(t) = m$ do
              13. $n := S(t)$
              14. if $n \in V$ then
                  15. if $X[n] \neq X[m]$ then
                      16. $L_{\text{new}} := L_{\text{new}} \cup \{ t \}$
                  17. else
                      18. $V := V \cup \{ n \}$
                      19. $C_{\text{new}} := C_{\text{new}} \cup \{ n \}$
                      20. $X[n] := X[m]$
              21. $C := C_{\text{new}}$
              22. $L := \text{DDclose}(L, L_{\text{new}})$
      23. until $L_{\text{new}} = \emptyset$
```

---

Fig. 6: Computing the least superset of $L$ that is closed under data dependence and satisfies $WCC$.

**Example 5** Let us apply the algorithm in Figure 6 to the EFSM in Figure 1, to find the least set that satisfies $WCC$ and contains $t6$ and $t9$.

In the first iteration, $B = \{ S3, S5 \}$. Since $S3$ can be reached from $S4$ by $t7$, and $S5$ can be reached from $S4$ by $t8$, there is a conflict at $S4$ so we add $t8$ (or $t7$) to $L$.
In subsequent iterations, with \( B = \{S3, S4, S5\} \), we may add \( t7 \) (due to the conflict as \( S4 \)), as well as \( t2 \) and \( t3 \) (due to a conflict at \( S1 \)).

We now have \( B = \{S1, S3, S4, S5\} \), and may add \( t4 \) due to a conflict at \( S3 \). The next iteration adds \( t1 \) since from \( S1 \) one can reach \( S3 \) through the path \([t1, t5]\). The following iteration adds \( t10 \) since from \( S4 \) one can reach \( S3 \) through the path \([t10, t12, t5]\).

We are left with \( L = \{t1, t2, t3, t4, t6, t7, t8, t9, t10\} \), illustrated in Figure 7 at which point no new transitions can be added, since \( S(t5) \) does not have other observables than \( S3 \). And \( L \) does indeed satisfy WCC: for all states \( n \) we have \( obs_L(n) = \{n\} \), except \( obs_L(S2) = obs_L(S6) = \{S3\} \).

**Theorem 4 (WCC Algorithm Complexity)** The algorithm in Figure 6 (including the construction of the table \( DDstar \)) can be implemented to run in time \( O(a^2) \) where \( a \) is the number of transitions.

**Theorem 5 (WCC Algorithm Correctness)** Assuming that the table \( DDstar \) correctly computes data dependence, the algorithm in Figure 6 returns in \( L \) the least superset of \( L \) that is closed under data dependence and satisfies WCC.

6.2 Computing Least SCC-Closed Slices

It is possible to extend the algorithm of Figure 6 into an algorithm for computing the least superset of \( L \) that satisfies SCC (and is closed under data dependence); in the beginning of the inner `for` loop, we insert a conditional:

\[
\text{If } \text{LoopAvoids}(n, X[m]) \text{ Then}
\]

\[
Lnew := Lnew \cup \{t\}
\]
Here $\text{LoopAvoids}$ is a pre-computed table such that $\text{LoopAvoids}(n, m)$ is true iff there is a loop from $n$ that avoids $m$. It can constructed as follows: for each $m$, ignore the transitions involving $m$, and do a depth-first search; then $\text{LoopAvoids}(n, m)$ is set to true iff a back edge is reachable from $n$. The resulting algorithm, including the construction of the tables $\text{DDstar}$ and $\text{LoopAvoids}$, can be implemented to run in time $O(a^2)$ where $a$ is the number of transitions.

Example 6 Let us apply the resulting algorithm to the EFSM in Figure 1, to find the least set that satisfies $\text{SCC}$ and contains $t9$.

In the first iteration, $B = \{S5\}$. Since $t2$ has source $S1$ and target $S5$, and there is a loop from $S1$ that avoids $S5$, we add $t2$. Similarly, since $t8$ has source $S4$ and target $S5$, and there is a loop from $S4$ that avoids $S5$, we add $t8$. Now, $B = \{S1, S4, S5\}$. Since $S4$ can be reached from $S1$ by $t3$, also $t3$ is added; since $S1$ can be reached from $S4$ by $[t7, t4]$, also $t7$ is added. Since $t4$ has target $S1$ and there is a loop that avoids $S1$ from $S3$, we add $t4$. Now, $B = \{S1, S3, S4, S5\}$. Since $S3$ can be reached from $S1$ by the path $[t1, t5]$, from $S3$ by the path $[t6, t12, t5]$, and from $S4$ by the path $[t10, t12, t5]$, also $t1$, $t6$, $t10$ are added. Also, $t12$ is added because there is a loop from $S6$ (via $t11$) that avoids $S3$.

We are left with $L = \{t1, t2, t3, t4, t6, t7, t8, t9, t10, t12\}$ at which point no new transitions can be added. And $L$ does indeed satisfy $\text{SCC}$: $\text{obs}_L(S1) = \{S1\}; \text{obs}_L(S2) = \{S3\}; \text{obs}_L(S3) = \{S3\}; \text{obs}_L(S4) = \{S4\}; \text{obs}_L(S5) = \{S5\}; \text{obs}_L(S6) = \{S6\}$ but no infinite path from $S2$ avoids $S3$.

7 Conclusions

We have proposed algorithms for slicing extended finite state machines and stated theorems that the resulting slices have well-defined semantic properties. Our development adapts to an interactive, non-deterministic setting a general methodology for describing the slicing of deterministic programs. It is left to future work to try our approach on realistic EFSMs and thus measure its practical usefulness. Future work also includes allowing transitions to produce events.

References


