A Theoretical and Empirical Study of EFSM Dependence

Kelly Androutsopoulos[1], **Nicolas Gold**[1], **Mark Harman**[1], **Zheng Li**[1] and **Laurence Tratt**[2]
[1]King's College London, CREST, Department of Computer Science, Strand, London, United Kingdom.
[2]Bournemouth University, Poole, Dorset, BH12 5BB, United Kingdom.

Abstract

Dependence analysis underpins many activities in software maintenance such as comprehension and impact analysis. As a result, dependence has been studied widely for programming languages, notably through work on program slicing. However, there is comparatively little work on dependence analysis at the model level and hitherto, no empirical studies. We introduce a slicing tool for Extended Finite State Machines (EFSMs) and use the tool to gather empirical results on several forms of dependence found in ten EFSMs, including well-known benchmarks in addition to real-world EFSM models. We investigate the statistical properties of dependence using statistical tests for correlation and formalize and prove four of the empirical findings arising from our empirical study. The paper thus provides the maintainer with both empirical data and foundational theoretical results concerning dependence in EFSM models.

Keywords: EFSM, Slicing, Dependence Analysis

1 Introduction

Dependence analysis is the name given to a class of techniques that have repeatedly proved to be useful as underlying support for a number of software maintenance activities. Through dependence analysis, the software maintainer can approach the problem of understanding the potential and actual interactions between parts of a system. These interactions may be subtle, complex and non-obvious. However, because a change potentially has an impact upon the transitively dependent parts of the system, understanding these dependences is a valuable part of the process of assessing and managing the maintenance and evolution process.

Hitherto, the overwhelming majority of work on dependence analysis for maintenance has concerned the implementation level of abstraction [6, 31]. This has produced a great many of empirical and theoretical results and applications to Software Maintenance, including program comprehension [10], impact analysis [11], dependence cluster analysis [16], testing and debugging [17], reintegration of changed version of a system [7], refactoring [20], and reverse engineering [9].

However, there is comparatively little work on dependence analysis and slicing at the model level of abstraction and even less concerned with slicing of state based models. This creates a need for the literature on theoretical and empirical results to catch up with development practice; developers' interest is increasingly tending to move up the abstraction chain to design levels of abstraction. This migration is driven, *inter alia*, by the growing popularity of model based development methods, architectures and testing techniques and the need to control size and complexity.

As the software maintenance community knows only too well, today's new development technique is tomorrow's maintenance problem. This observation is also true of work on maintenance for model based levels of abstraction, with much work in the maintenance community concerned with maintenance of modelling notations [14]. Since dependence analysis has provided a valuable suite of maintenance techniques at the implementation level, there is no reason to believe that it will not also provide useful information to the maintainer, working at the model level.

However, the paucity of empirical results on dependence for state based models means that the software maintainer has little base line data on dependence in state based models, leaving a gap in the existing literature. This paper aims to address this gap. It provides empirical results on slicing and dependence for a set of state based models, including text book benchmark systems as well as real world production industrial systems. The results show some interesting statistical correlations and relationships between different techniques for assessing dependence. These empirical observations are analyzed, first using statistical tests, and then formalized as theorems about state based model dependence. The empirical data, together with the proof of the theorems, establishes practical and theoretical underpinnings to the dependence analysis of state based systems.

In order to produce the results in this paper, we implemented an EFSM slicing tool, the CREST EFSM slicer, that supports various forms of forward and backward EFSM slicing, according to several previously introduced definitions of data and control dependence for EFSM and reactive systems. Like program slicing, EFSM slicing is a de-

pendence analysis based on a user-specified slicing criterion. The criterion captures the transition and variables of interest within the EFSM, while the process of slicing consists of following dependencies to locate those states and transitions that are relevant to the slicing criterion. Tracing the dependencies from the slicing criterion to parts of the EFSM that could be affected when the criterion transition is changed is called forward slicing. This has applications to the software maintenance problems of impact analysis and ripple effect computation [29]. On the other hand, tracing the dependence from the criterion to those states and transitions that could potentially affect the criterion is called backward slicing. This has applications to the software maintenance problems of comprehension, reintegration and refactoring [22].

In this paper we present results for both forward and backward slices of state based models for several recently introduced formulations of control dependence. Using the CREST EFSM slicing tool, we construct all possible slices of a suite of ten EFSMs, taken from text book benchmark examples and real world production EFSM models. The primary contributions of the paper are as follows:

1. Because they are design level abstractions, state based models can be considerably smaller than the programs that implement them. However, our empirical findings reveal that current definitions of dependence lead to slice sizes that are notably larger than the existing benchmark data for program slice size [4]. This suggests that more work may be required to find alternative definitions of dependence for state based models.

2. Forward slice sizes tend to be larger than backward slice size for EFSMs. This also appears to suggest differences in dependence at the model level compared to studies of dependence at the program level of abstraction [3].

3. Four of the novel findings arising from the empirical results are formalised and proved.

2 Slicing State-based Model

Slicing of EFSMs involves dependence analysis, in particular control and data dependence. One of the challenges with slicing EFSMs is how to correctly account for control dependence. This is because EFSMs can be non-terminating (i.e. without an EXIT state) which breaks traditional control depedence used in program dependence analysis. Moreover, there is a choice of whether control dependence should be *sensitive* or *insensitive* to non-termination. This decision determines whether slicing may remove any non-termination. This has lead to numerous definitions of control dependence [2].

For this empirical study, we consider Ranganath et al. [27, 28] definitions of control dependence, i.e. Non-termination Sensitive Control Dependence (NTSCD), and Non-termination Insensitive Control Dependence (NTICD), which have been adapted and given in terms of transitions of EFSMs, rather than nodes of a Control Flow Graph (CFG). Also, we have defined a new control dependence definition in [2], called Unfair Non-termination Insensitive Control Dependence (UNTICD) that overcomes the limitation's of NTICD.

In this section, we first define the syntax of EFSMs. Then we define three types of paths that are used in the three definitions of control dependence. We also define data dependence and out notion of a slice.

2.1 Extended Finite State Machine

An Extended Finite State Machine (EFSM) M is a tuple (S, T, E, V) where S is a set of states, T is a set of transitions, E is a set of events, and V is a store represented by a set of variables. Transitions have a source state $source(t) \in S$, a target state $target(t) \in S$ and a label $lbl(t)$. Transition labels are of the form $e_1[g]/a$ where $e_1 \in E$, g is a guard (we assume a standard conditional language) and a a sequence of actions (we assume a standard expression language including assignments). All parts of a label are optional.

EFSMs are possibly non-deterministic. States of S are atomic. Actions can involve store updates. A *self-looping transition* is a transition t where the source of t is the same as the target of t. A set of distinct transitions may have an identical source and an identical target. Transitions which share the same source state are said to be *siblings*. A *final transition* is one whose target is an EXIT state that has no outgoing transitions. An ε transition is one with no event, guard or action.

2.2 Paths in EFSM

Since a path is commonly presented as a (possibly infinite) sequence of nodes, a node is in a path if it is in the sequence. A transition is in a path if its source state is in the path and its target state is both in the path and immediately follows its source state. There are three types of paths that can be used to define different kinds of control dependence.

Definition 1 (Maximal Path)**.** *A maximal path is any path that terminates in a final transition, or is infinite.*

Sink-bounded paths are given in terms of control sinks.

Definition 2 (Control Sink)**.** *A control sink in an EFSM is a set of transitions $\mathcal{K}$ that form a strongly connected component (SCC) such that, for each transition t in $\mathcal{K}$ each successor of t is also in $\mathcal{K}$.*

Definition 3 (Sink-bounded Path)**.** *A maximal path π is sink-bounded iff (i) there exists a control sink $\mathcal{K}$ such that $\mathcal{K} \cap \pi \neq \emptyset$ and, (ii) if π is infinite, then all transitions in $\mathcal{K}$ occur infinitely often.*

The second clause of Definition 3 defines a form of fairness and hence we refer to it as the *fairness condition*.

Definition 4 (Unfair Sink-bounded Path [2])**.**
A maximal path π is unfair sink-bounded iff there exists a control sink $\mathcal{K}$ such that: π contains a transition from $\mathcal{K}$.

The definition of Unfair Sink-bounded Paths drops the fairness condition in Definition 3. For non-terminating systems this means that control dependence can be calculated within control sinks.

2.3 Dependence Analysis of EFSM

The following definition specifies control dependence independently of any of the specific definitions of paths from Section 2.2. The *PATHs* function can be substituted with the three types of paths defined therein, yielding different types of control dependence.

Definition 5 (Control Dependence (CD))**.**
$T_i \xrightarrow{CD} T_j$ means that a transition T_j is control dependent on a transition T_i iff T_i has at least one sibling T_k such that:

1. *for all paths $\pi \in$ PATHs$(target(T_i))$, the $source(T_j)$ belongs to π;*
2. *there exists a path $\pi \in$ PATHs$(source(T_k))$ such that the $source(T_j)$ does not belong to π.*

Table 1 shows the corresponding relations between the path type and control dependences. For example, *NTSCD* is given in terms of *maximal paths*, so by replacing *PATH*s with the *MaximalPath* function, Definition 5 will yield the corresponding *NTSCD* definition.

Table 1. Types of control dependence

Name	Path type
Non-Termination Insensitive Control Dependence (NTSCD)	Maximal Path
Non-Termination Sensitive Control Dependence (NTICD)	Sink-bounded Path
Unfair Non-Termination Insensitive Control Dependence (UNTICD)	Unfair Sink-bounded Path

Definition 6 (Data Dependence (DD))**.**
$T_i \xrightarrow{DD} T_j$ means that transitions T_i and T_j are data dependent with respect to a variable v if:

1. *$v \in D(T_i)$, where $D(T_i)$ is a set of variables defined by transition T_i, i.e. variables defined by actions and by the event of T_i;*
2. *$v \in U(T_j)$, where $U(T_j)$ is a set of variables used in a condition and actions of transition T_j;*
3. *there exists a path in an EFSM from the $source(T_i)$ to the $target(T_j)$ whereby v is not modified by any of the intermediate transitions.*

To illustrate these definitions consider Figure 1 which is an EFSM of the door control component of an elevator control system [30]. The door component controls the elevator door: it opens the door, waits for the passengers to enter or leave the elevator, and then shuts the door. All the control dependencies for this EFSM using the three types of control dependence are given in Figure 2.

NTSCD	$T3 \rightarrow T4, T5, T6$	$T5 \rightarrow T9, T10$
	$T6 \rightarrow T7, T8$	$T8 \rightarrow T9, T10$
	$T10 \rightarrow T11, T12$	$T12 \rightarrow T4, T5, T6$
NTICD	No dependences	
UNTICD	$T5 \rightarrow T9, T10$	$T6 \rightarrow T7, T8$
	$T8 \rightarrow T9, T10$	$T10 \rightarrow T11, T12$
	$T12 \rightarrow T4, T5, T6$	
DD	$T1 \rightarrow T2, T3$	$T2 \rightarrow T2, T3$
	$T5 \rightarrow T11$	$T8 \rightarrow T11$
	$T11 \rightarrow T11$	

Figure 2. Dependence for Figure 1.

2.4 EFSM Slicing

Slices for an EFSM are constructed based on dependence analysis with respect to a slicing criterion. A slicing criterion is a pair (t, v) where transition $t \in T$ and variable set $v \subseteq V$. It designates the point in the evaluation immediately after the execution of the action contained in transition t.

Definition 7 (Slice)**.**
A slice of an EFSM M, is another EFSM M', some of whose transitions may be $\varepsilon-$transitions. The transitions that are not $\varepsilon-$transitions are in the set of transitions that are directly or indirectly (transitive closure) data and control dependent on the slicing criterion c.

Slices are computed by gathering transitions by way of a backward traversal of the EFSMs dependence graph, starting at the slicing criterion t. Therefore, these slices are referred to as *backward* slices (denoted by $\overleftarrow{\mathcal{S}}(M, t)$, where M is an EFSM and t is the transition from the slicing criterion). A *forward* slice (denoted by $\overrightarrow{\mathcal{S}}(M, t)$) consists of all transitions dependent on the slicing criterion, which is constructed by way of a forward traversal of EFSM's dependence graph.

3 Empirical Study

3.1 Motivation

For EFSM slicing to be considered of practical use, it is important to establish that the slices produced from the current types of dependence analysis are not so large as to confer no advantage over understanding the whole of the original model being analysed. Three types of control dependence have been defined for EFSMs in Section 2, each of which produces different types of slices, capturing different features of EFSMs. Comparison of the size of these types of slices is an appropriate method to examine the effects on size of different types of dependence. This leads to two research questions:

1. What is the typical forward and backward slice size using different types of control dependence for EFSMs?

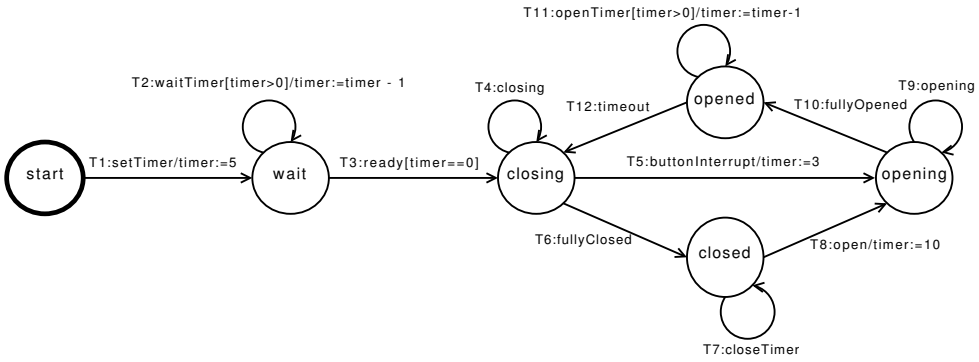

Figure 1. An EFSM specification for the door control of the elevator system.

2. Is there a correlation between the slice size observed using different types of control dependence for EFSMs?

3.2 Metrics

This section formalizes the metrics used in the paper to measure slice size in terms of the percentage of transitions that transitively depend upon a criterion. Dependence could be either control dependence or data dependence or both. Data dependence includes the data dependence for all variables in the criterion. For a model M, t' is a transition dependent on t (i.e., $t' \in T \wedge t \rightarrow t'$), the size of slice with respect to t is:

$$|\mathcal{S}(M,t)| = \frac{\sum t'}{|M|}$$

Note that the slicing criterion t is in the slice $\mathcal{S}(M,t)$ iff t is dependent on itself. This is slightly different to program slicing where the criterion is always in the slice. For example, if a variable is defined and used in t, there exists a self-looping data dependence on t. Certainly, it is also possible that t is transitively control dependent on itself. If no transition is dependent on t, then $|\mathcal{S}(M,t)| = 0$. To remove the effect of size-zero slices, the average slice size for a model is computed only for transitions that have a non-empty set of dependences.

For a model M, NT is subset of transitions of M with non-zero slice size (i.e., $NT \subseteq T$ and $\forall t \in NT, |\mathcal{S}(M,t)| > 0$) .Thus, the average slice size of M is:

$$\mathsf{Avg}(M) = \frac{\sum_{t \in NT} |\mathcal{S}(M,t)|}{|NT|} \quad (1)$$

3.3 Subjects

The ten EFSM models employed as subjects are described in Table 2. The table provides each model's size in terms of the number of transitions and the number of states. It also provides a brief description of each subject. The penultimate column separates the six models known to contain EXIT states from the four known to be free of EXIT states.

The models are drawn from a variety of sources. The first six models were used in the research of model-based slicing with traditional dependence analysis introduced by Korel et al. [23], which requires models to contain both a START state and an EXIT state, and where every path must end in an EXIT state. The last four models are free of EXIT states. INRES [8] and Lift [30] come from previous model-based studies. TCP [33] and TCSbin are extracted from SDL specifications, and TCSbin is an industry model from Motorola. A simplification is adopted in the EFSM extraction for these two models that includes the omission of History State and All state defined in SDL.

Inspection of the models' state machines reveals that their structure varies even if they share some common characteristics such as all containing EXIT states. Typically, only one of the first six models, FuelPump, has a CFG-like structure, i.e., all transitions from the START state to the EXIT state almost form a straight line, while the other five of the six models contain a few SCCs. Generally models free of EXIT states contain large control sinks. All transitions of the Lift model, illustrated in Figure 1, apart from T1, T2 and T3, form a large control sink. All transitions of the INRES, TCP, TCSbin models, apart from one transition from the START state, also form a large control sink. A transition from the START state has no event, condition or action as it indicates only the initial state of a model.

Two groups are defined based on control sinks. $M' = \{$ATM, Cashier, CruiseControl, FuelPump, PrinTok, VendingMachine$\}$, where all models are free of control sinks. $M'' = \{$INRES, TCP, TCSbin$\}$, where all models are large control sinks except for the transition from the START state. Note that Lift is not in M' or M''.

3.4 Implementation and Tool

Slices with respect to each transition in a model are computed based on control dependence (using one of the three definitions) and data dependence. To help draw out the effect of the three types of control dependence, slices are also computed using only NTSCD, NTICD, UNTICD and data dependence respectively. Additionally, both forward and backward slicing are considered in the compu-

Table 2. Experimental Models.

Models	Number of States	Number of Transitions	Number of Variables	EXIT State	Brief Description
ATM	9	23	8	Yes	Automated Teller Machine [23]
Cashier	12	21	10	Yes	Cashier Machine
CruiseControl	5	17	18	Yes	Cruise Control System [21]
FuelPump	13	25	12	Yes	Fuel Pump System [21]
PrinTok	11	89	5	Yes	Print Token
VendingMachine	7	28	7	Yes	Vending Machine system
INRES	8	18	8	No	INRES protocol [8]
TCP	12	57	31	No	TCP Standard(RFC793) [33]
TCSbin	24	65	61	No	Telephony Control Protocol (Motorola)
Lift	6	12	1	No	Lift System [30]
Total	107	355	161		

tation. Therefore, there are 14 types of slices constructed for each transition over ten models, i.e., {forward, backward} $\times$ {NTICD, UNTICD, NTSCD, DD, DD+NTICD, DD+UNTICD, DD+NTSCD}.

A tool has been developed using Python to implement all three types of control dependence analysis and data dependence analysis, construct the dependence graph for a model, and compute the slice with respect to a criterion. The statistical package SPSS is used for statistical analysis.

4 Results and Discussion

In this section the empirical results related to the forward and backward slice over all ten subjects are discussed. Also, four novel findings arising from the empirical results are formalised and proved.

4.1 Slice Size

Table 3. Average slice size.

Dependence	Forward Slices # T	Forward Slices Avg	Backward Slices # T	Backward Slices Avg
DD+NTSCD	276	87.45%	345	70.46%
DD+NTICD	220	61.99%	278	49.48%
DD+UNTICD	267	83.20%	335	66.83%
DD	161	35.67%	174	33.15%
NTSCD	205	86.10%	336	53.63%
NTICD	92	78.67%	167	44.59%
UNTICD	190	82.21%	313	51.00%

Table 3 presents the results of average slice size for 14 types of slices for all transitions over all ten subjects. In the table, #T is the number of transitions with non-zero slice size and Avg is the average slice size of all transitions over all ten models.

Note that the average forward slice size for all transitions (including transitions with size-zero slice) is the same as that of backward slices, because forward and backward slicing are dual operations. Thus, if T_i is in the backward slice taken with respect to T_j then T_j is in the forward slice taken with respect to T_i. However, the average of forward slices and backward slices measured by Avg is not equal, as the number of non-zero forward slices is different than the number of non-zero backward slices.

Table 2 shows that there are 355 transitions in total over all ten subjects. In Table 3, it can be seen that some transitions do not have forward slices and some do not have backward slices, but it is very uncommon that a transition has neither a forward slice nor a backward slice. Inspection of the data reveals three exceptions, which are the three transitions from the START state to initial state of the three models in M''. The only function of these three transitions is the initialisation of initial state of a model.

The average slice size reported in Table 3 shows that the average slice size using NTICD is smaller than that using UNTICD or NTSCD. Further inspection of the data reveals that the slice using NTICD is contained within the slice using UNTICD with respect to the same criterion. This result reflects the formally-proved property that *the transitive closure of NTICD is contained in the transitive closure of UNTICD* [2]. Furthermore, the average slice size using UNTICD is smaller than that using NTSCD, suggesting a new property that *the transitive closure of UNTICD is contained in the transitive closure of NTSCD*.

Studies of backward program-based slice size indicate that a typical backward slice may be as much as a third of the program [5]. In Table 3, over all ten models, the smallest average backward slice size using DD+NTICD slices contains over half of the original program. As Avg measures the average slice size for those non-zero slice transitions, the value is larger than the average slice size for all possible transitions. It may be unfair to compare model-based Avg to average backward program-based slice size. Therefore, the average backward model-based slice size for all possible transitions over all ten subjects is measured. Numerically, they are 38.42%, 67.99% and 62.58% for DD+NTICD, DD+UNTICD, and DD+NTSCD respectively. The smallest (i.e. 38.42% for DD+NTICD) is slightly larger than a typical backward slice size (i.e., one third of the program). The other two types of slices are significantly larger, as UN-

TICD and NTSCD capture more dependencies within control sinks which NTICD does not capture.

Let us now consider the slice size difference between forward slicing and backward slicing. The data in Table 3 shows that the average slice size using forward slicing is larger than the average slice size using backward slicing, but the number of transitions have forward slice is smaller than that of backward slice. That is, more transitions tend to have backward slice rather than forward slice, but once a transition has a forward slice, the size tends to be large.

Binkley and Harman [3] reported that the distribution of small forward slice is larger than the distribution of small backward slice for programs. This is not contrary to the conclusion presented in this paper, because Avg only involves non-zero slices in this paper. Binkley and Harman [3] also pointed out that there must be a few large forward slices in programs, but these tend to be uninteresting slices, since they occur primarily when the slicing criterion is near the entry to a procedure. However, these large forward slices may be interesting in state-based models, as the slice criterion producing a large forward slice could occur anywhere in a state machine. This is because an EFSM does not to have an entry and an exit node, as is required for program's CFGs. Also, the structure of an EFSM used to specify reactive systems is often just a control sink. In this case, a transition with a large forward slice would be interesting as it is not necessary for it to occur near the START state and it could also have a large impact on the model.

Figure 3 presents the average slice size for each model. Different types of control dependences and slice directions are considered in slice construction. Thus, Figure 3 shows six bar charts $\{$forward, backward$\} \times \{$NTICD, UNTICD and NTSCD$\}$. For each bar chart, separate slices are constructed (in the following order) for: both data and control dependence; control dependence only; and data dependence only. Therefore, for each model the three bars represent the Avg of three types of slices.

Some interesting results emerge from Figure 3. For each model, the average backward slice size is less than or equal to the forward slice size. FuelPump has the smallest slice size when using only control dependence in slice construction. As explained in Section 3.3, the state machine of FuelPump has a CFG-like structure which results in small slice size. This also provides evidence that SCCs in models tend to increase the slice size. PrinTok has the largest forward slice size, close to 100%, i.e. all transitions are forward transitive control dependent upon each other. In such a case, forward slice can not reduce the model size. A further discussion and analysis is presented in Section 4.2.

In Figure 3, it can be observed that the properties formally shown in [2] are true for all ten models.

- The average slice size of slices using only NTICD for models in M'' is 0 that confirms that there is no NTICD in control sink, as each model m'' in M'' is a large control sink.
- The average slice size of slices using only NTICD is the same as that of slices using only UNTICD for M' that confirms that UNTICD is the same as NTICD out of control sinks, as all models in M' do not contain control sinks.
- The average slice size of slices using only NTSCD is the same as that of slices using only UNTICD for M'' that confirms that UNTICD is the same as NTSCD in control sinks, as each model m'' in M'' is a large control sink.

4.2 Correlation of Slice Sizes

To provide more rigour, a statistical analysis of the correlation between different types of slices was conducted in this section.

Table 4 reports the Pearson correlation between the slice size with respect to each transition using only control dependence for all models. NTICD, UNTICD and NTSCD are considered respectively. The value in the table is a correlation coefficient (R value) and ranges from -1.0 to 1.0. Where -1.0 is a perfect negative (inverse) correlation, 0.0 is no correlation, and 1.0 is a perfect positive correlation. If R=1 then two slice sizes have a linear correlation. As both forward slices and backward slices are measured, if two sets of slices are constructed using two different types of control dependence in a model, and R=1 for both forward and backward slicing, the two slice sizes for each transition must be equal.

The result in Table 4 shows that the R value between the slices using NTICD and UNTICD is 1 for both forward and backward slicing for models in M'. That is for each transition t in m', where $m' \in M'$, slice size is the same using NTICD as that using UNTICD. This reflects the property that UNTICD and NTICD dependences for transitions outside of control sink are the same. Similarly, the R value of 1 between the slices using UNTICD and NTSCD for both forward and backward slicing for models in M'' reflects the property that UNTICD and NTSCD dependence for transitions within control sink are the same.

It is also interesting that CruiseControl and PrinTok have R value of 1 for all slices using three types of control dependence, which indicates that the sizes of slices taken with respect to each transition using only NTICD, UNTICD and NTSCD are the same. Inspection of the data reveals that they are actually identical. Furthermore, the two models have similar structure, i.e. each state has a transition leading to an EXIT state. Such transitions generally handle errors (i.e., in any state of a model, if an error occurs, go to the EXIT).

CruiseControl and PrinTok represent typical SDL models in which a type of state, *All State*, represents all possible

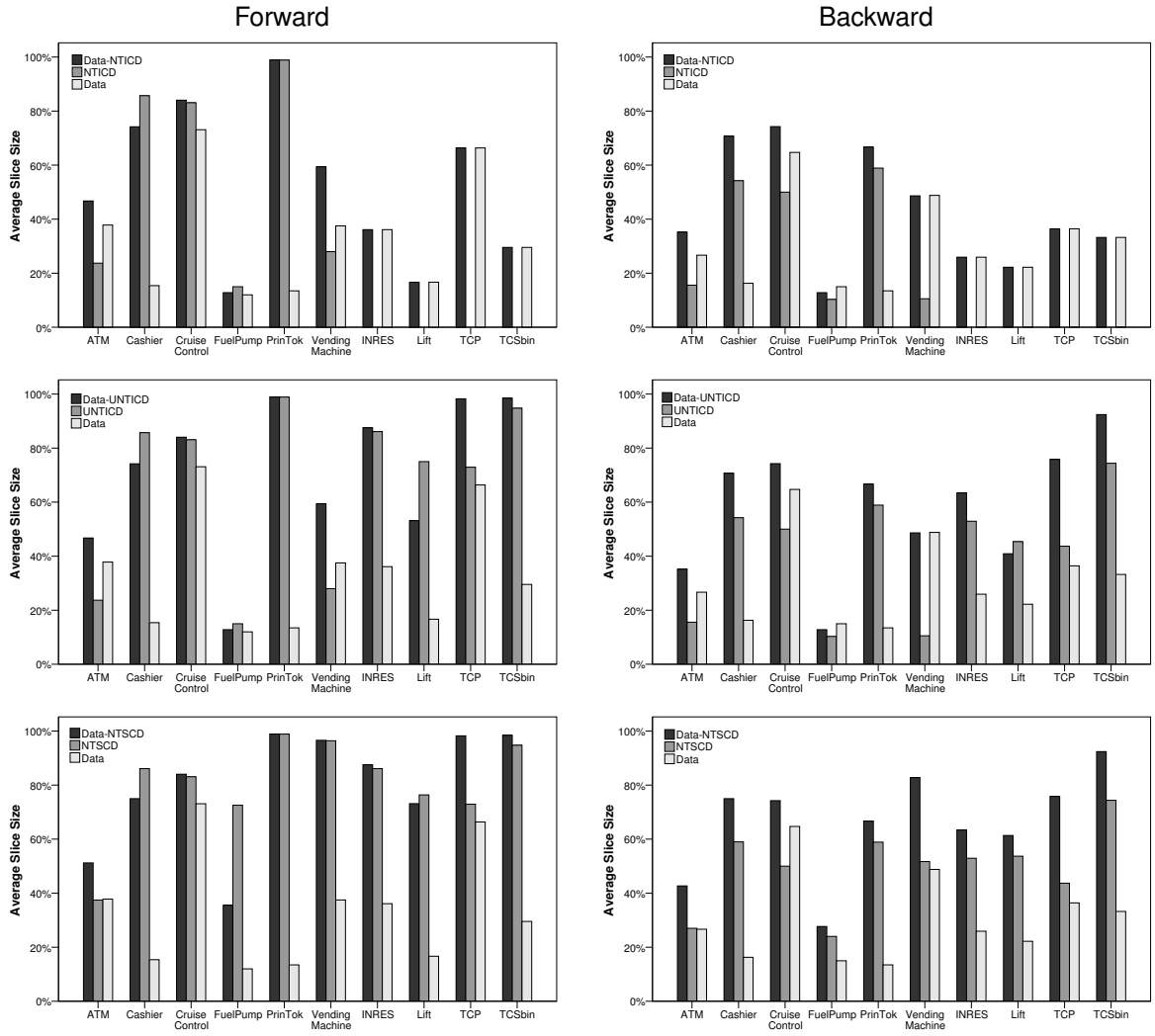

Figure 3. Average forward slice size for all models based on NTICD, NTSCD and UNTICD respectively

states in the model. A typical SDL model will contain the exception handling using *All State*, but in most case studies, *All State* is ignored. The result presented reveals that for any model with exception handling using *All State*, slices using NTICD, UNTICD and NTSCD would be the same. Note that the average forward slice size of PrinTok is almost 100%. The investigation reports a worst case for forward slicing, as in such model, each transition forward controls all other transitions. A further inspection of the state machine of PrinTok reveals that if the EXIT state is removed as well as all transitions from each state to EXIT state, the remaining transitions form a large control sink except the transition from START state.

4.3 Properties of Control Dependence

From the empirical studies and subsequent data analysis, we have observed the following:

1. For a self-looping transition T in any of the ten EFSM models, the forward slice with respect to T using only NTSCD, NTICD or UNTICD is null.

2. In any of the ten EFSM models M, if two transitions T_i and T_j have the same source and target states (i.e., $source(T_i) = source(T_j)$ and $target(T_i) = target(T_j)$), then the forward slice using NTSCD with respect to T_i will be the same as the forward slice using NTSCD with respect to T_j. Similarly for forward slicing with NTICD and UNTICD.

3. In any of the ten EFSM models, if T_j is directly control dependent (either NTSCD, NTICD or UNTICD) on T_i, then the source state of T_j is always in the shortest path from the set of paths from T_i. The shortest path from a set of paths PATHs denotes the path with the minimum sequence of nodes.

4. For each transition T in the CruiseControl and

Table 4. Pearson correlation between the slice size for each model.

Model	Dependence	Forward NTICD	Forward UNTICD	Forward NTSCD	Backward NTICD	Backward UNTICD	Backward NTSCD
ATM	NTICD	-	1.000	.652	-	1.000	.941
	UNTICD	1.000	-	.652	1.000	-	.941
	NTSCD	.652	652.	-	.941	.941	-
Cashier	NTICD	-	1.000	.898	-	1.000	1.000
	UNTICD	1.000	-	.898	1.000	-	1.000
	NTSCD	.898	.898	-	1.000	1.000	-
CruiseControl	NTICD	-	1.000	1.000	-	1.000	1.000
	UNTICD	1.000	-	1.000	1.000	-	1.000
	NTSCD	1.000	1.000	-	1.000	1.000	-
FuelPump	NTICD	-	1.000	.786	-	1.000	-.509
	UNTICD	1.000	-	.786	1.000	-	-.509
	NTSCD	.786	.786	-	-.509	-.509	-
PrinTok	NTICD	-	1.000	1.000	-	1.000	1.000
	UNTICD	1.000	-	1.000	1.000	-	1.000
	NTSCD	1.000	1.000	-	1.000	1.000	-
VendingMachine	NTICD	-	1.000	.360	-	1.000	.224
	UNTICD	1.000	-	.360	1.000	-	.224
	NTSCD	.360	.360	-	.224	.224	-
INRES	NTICD	-	x	x	-	x	x
	UNTICD	x	-	1.000	x	-	1.000
	NTSCD	x	1.000	-	x	1.000	-
Lift	NTICD	-	x	x	-	x	x
	UNTICD	x	-	.813	x	-	1.000
	NTSCD	x	.813	-	x	1.000	-
TCP	NTICD	-	x	x	-	x	x
	UNTICD	x	-	1.000	x	-	1.000
	NTSCD	x	.	-	x	1.000	-
TCSbin	NTICD	-	x	x	-	x	x
	UNTICD	x	-	1.000	x	-	1.000
	NTSCD	x	1.000	-	x	1.000	-

PrinTok models, the slices, either forward or backward, using NTICD, UNTICD or NTSCD with respect to T are the same. Both of these models have a similar structure, where each state, except for the START state, has a transition that leads to the EXIT state.

We generalise each observation to a corresponding property and provide a proof. These four properties can simplify the model graph and thus reduce the cost of computing control dependence for large models. For example, Proposition 4.3 helps by not computing control dependence for any transition whose source state is not on the shortest path from the slicing criterion.

Proposition 4.1. *For an EFSM M, if $T_i \in M$ is a self-looping transition, then there is no transition T_j that is control dependent (NTSCD, NTICD or UNTICD) on T_i.*

Proof. If T_i is a self-looping transition in M and T_k is a sibling of T_i, then $source(T_k) = source(T_i) = target(T_i)$ (by definition of sibling and self-looping transition), and thus $PATHs(source(T_k)) = PATHs(target(T_i))$. Assume there is a transition T_j that is control dependent (either NTSCD, NTICD or UNTICD) on T_i. Then, $source(T_j)$ belongs to all $PATHs(target(T_i))$ (by clause 1 of Definition 5), and there exists a path in $PATHs(source(T_k))$ that $source(T_j)$ does not belong to (by clause 2 of Definition 5). However, $PATHs(source(T_k)) = PATHs(target(T_i))$ and hence clause 2 of Definition 5 will always be false. Therefore, we have shown by contradiction that there is no such T_j that is control dependent on a self-looping transition T_i. □

Proposition 4.2. *For an EFSM M, if two transitions T_i and T_j have the same source and target states, and $T_i \xrightarrow{CD} T_l$ (using NTSCD, NTICD or UNTICD) then $T_j \xrightarrow{CD} T_l$ (using NTSCD, NTICD or UNTICD respectively).*

Proof. Let T_i and T_j be two transitions in an EFSM M, where $source(T_i) = source(T_j)$ and $target(T_i) = target(T_j)$, and if T_k is a sibling of T_i, then T_k is also the sibling of T_j. Assume that there exists a transition T_l where $T_i \xrightarrow{\text{CD}} T_l$, by Definition 5, and not $T_j \xrightarrow{\text{CD}} T_l$. Since, $T_i \xrightarrow{\text{CD}} T_l$, the $target(T_l)$ is on all PATHs($target(T_i)$) and there exists a path from T_k where $source(T_l)$ does not belong to. However, T_i and T_j have identical source and target states, so the PATHs($target(T_i)$) = PATHs($target(T_j)$), and identical sibling transition T_k, thus both clauses in the Definition 5 are true for T_j, that is $T_j \xrightarrow{\text{CD}} T_l$. Therefore, we have shown by contradition that if T_i and T_j have the same source and target states, and $T_i \xrightarrow{\text{CD}} T_l$ then $T_j \xrightarrow{\text{CD}} T_l$. □

Proposition 4.3. *For an EFSM M, if $T_i \xrightarrow{CD} T_j$ (either NTSCD, NTICD, or UNTICD), $source(T_j)$ must belong to the shortest path of type PATH in PATHs($target(T_i)$).*

Proof. If $T_i \xrightarrow{\text{CD}} T_j$, then the $source(T_j)$ belongs to all PATHs($target(T_i)$), by Definition 5. Since, the shortest path from $target(T_i)$ of type PATH also belongs to PATHs($target(T_i)$), $source(T_j)$ must also belong to the shortest path in PATHs($target(T_i)$). □

Proposition 4.4. *For an EFSM M, if all states $s \in M$ where $s \neq$ START have a transition T_i where $source(T_i) = s$ and $target(T_i) =$ EXIT, then the set of transitions that are*

directly control dependent on T_i are the same for all types of control dependence, i.e. NTSCD, NTICD and UNTICD.

Proof. Assume an EFSM M with all states $s \in M$, where $s \neq$ START, and each has a transition T_i where $source(T_i) = s$ and $target(T_i) =$ EXIT. Then, for each state s, the shortest maximal path, the shortest sink-bounded path and the shortest unfair sink-bounded path are the same, i.e. the path $\{T_i\}$. Since all types of PATHS are the same, then the control dependences produced for NTICD, UNTICD and NTSCD are the same, by Definition 5. □

5 Related Work

Androutsopoulos et al. [2] briefly surveyed the definitions of control dependence for slicing finite machines (FSM). Heimdahl et al. [19, 18] were the first to present a control dependence definition for RSML, a tabular notation that is based on hierarchical FSMs. It differs from the traditional notion as it defines control flow in terms of events rather than transitions. Korel et al. [23] give a definition of control dependence for EFSMs in terms of post dominance that requires execution paths to lead to an EXIT state. Ranganath et al. [27, 28] give two versions of control dependence for non-terminating programs: NTSCD and NTICD. The difference between these definitions lies in the choice of paths. Labbé et al. [24] adapt Ranganath et al.'s NTSCD definition for communicating automata [12]. Oja [25] also adopts Ranganath et al.'s definition of control dependence, i.e. NTSCD, and decisive order dependence.

To the best of our knowledge, no empirical results have been obtained by testing these different control dependence definitions and analysing their effect on the size of slices.

Approaches for slicing state-based models that include some experimental results are discussed. Ramesh et al. [26] present two static backward slicing algorithms that compute slices of Esterel programs (FSM language) and VHDL programs used for developing synchronous reactive systems. The slicing criterion is an event or a set of events. Besides the standard control (i.e. given with respect to an exit state) and data dependence, Ramesh et al. introduce novel dependencies that arise due to concurrency and event generation: signal dependence, interference control dependence and time dependence. The Esterel and VHDL slicers have been tested by applying them to a number of programs in order to observe the amount of reduction due to slicing. The experimental results indicate that the size of the slices (measured by number of statements) depends on the slicing criterion, i.e. if different slicing criteria are chosen then the size of slices will be different.

Guo and Roychoudhury [13] present an approach for debugging Statecharts [15] by using dynamic slicing. First Java code is automatically generated from the statecharts while using appropriate tags to store the model-code association information. Then, subject to an error being detected, dynamic slicing, using the JSlice [32] tool, is applied to the Java code. The resulting slice is then mapped back to the statechart model, while maintaining the hierarchical and concurrent structure. They experimentally test the size of the slices, both at the code (measured by lines of code) and model (measured by number of model elements) levels, produced by dynamic slicing. The sizes of the model slices are significantly smaller, because a single model element requires a couple of lines of code to implement.

6 Summary and Future Work

This paper is geared toward providing empirical results on slicing and dependence for state based models that to be useful for understanding and analysing large and complex models. Three types of control dependence that tackle the issue of non-termination in EFSMs are empirically studied. The results over ten EFSMs show that the slice size of EFSMs is notably larger than that for program slice size. The analysis of the empirical data also reveals new properties of control dependence and the corrsponding formal proofs are given. A typical EFSM with *All State* where all slices with respect to a transition constructed using three types of control dependence are identical. A specific structure of EFSM with the worst case of forward slicing is also presented, where the forward slice with respect to any transition is the whole EFSM.

Amtoft [1] has recently presented a new control dependence definition, called Weak Order Dependence (WOD), that can be applied to irreducible CFGs and argues that it also captures traditional control dependence. We plan to implement this definition and experimentally test how big the slice sizes are when slicing is applied.

Acknowledgements

This research work is supported by EPSRC Grant EP/F059442/1. The authors also wish to thank Bogdan Korel for providing some of the case study models. Author order is alphabetical.

References

[1] T. Amtoft. Slicing for modern program structures: a theory for eliminating irrelevant loops. *Inf. Process. Lett.*, 106(2):45–51, 2008.

[2] K. Androutsopoulos, D. Clark, M. Harman, Z. Li, and L. Tratt. Control dependence for extended finite state machines. In M. Chechik and M. Wirsing, editors, *Fundamental Approaches to Software Engineering (FASE)*, volume 5503 of *Lecture Notes in Computer Science*, pages 216–230. Springer, 2009.

[3] D. Binkley and M. Harman. Forward slices are smaller than backward slices. In 5^{th} *IEEE International Workshop on Source Code Analysis and Manipulation*, pages 15–24, Los Alamitos, California, USA, 2005. IEEE Computer Society Press.

[4] D. W. Binkley, N. Gold, and M. Harman. An empirical study of static program slice size. *ACM Transactions on Software Engineering and Methodology*, 16(2):1–32, 2007.

[5] D. W. Binkley and M. Harman. A large-scale empirical study of forward and backward static slice size and context sensitivity. In *IEEE International Conference on Software Maintenance*, pages 44–53, Los Alamitos, California, USA, Sept. 2003. IEEE Computer Society Press.

[6] D. W. Binkley and M. Harman. A survey of empirical results on program slicing. *Advances in Computers*, 62:105–178, 2004.

[7] D. W. Binkley, S. Horwitz, and T. Reps. Program integration for languages with procedure calls. *ACM Transactions on Software Engineering and Methodology*, 4(1):3–35, 1995.

[8] C. Bourhfir, R. Dssouli, E. Aboulhamid, and N. Rico. Automatic executable test case generation for extended finite state machine protocols protocols.

[9] G. Canfora, A. Cimitile, and A. De Lucia. Conditioned program slicing. *Information and Software Technology Special Issue on Program Slicing*, 40(11 and 12):595–607, 1998.

[10] K. Gallagher and L. Layman. Are decomposition slices clones? In 11^{th} *International IEEE Workshop on Program Comprehension (IWPC'03)*, pages 285–286. IEEE, May 2003.

[11] K. B. Gallagher and J. R. Lyle. Using program slicing in software maintenance. *IEEE Transactions on Software Engineering*, 17(8):751–761, Aug. 1991.

[12] C. Gaston, P. L. Gall, N. Rapin, and A. Touil. Symbolic execution techniques for test purpose definition. In *Proc. Testing of Communicating Systems*, pages 1–18, 2006.

[13] L. Guo and A. Roychoudhury. Debugging statecharts via model-code traceability. In *Leveraging Applications of Formal Methods, Verification and Validation, Third International Symposium, ISoLA 2008*, pages 292–306, Port Sani, Greece, 2008.

[14] H. Gustavsson, B. Lings, B. Lundell, A. Mattsson, and M. Beekveld. Simplifying maintenance by using XSLT to unlock UML models in a distributed development environment. In *IEEE International Conference on Software Maintenance (ICSM'07)*, pages 465–468, Paris, France, 2007.

[15] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, June 1987.

[16] M. Harman, D. Binkley, K. Gallagher, N. Gold, and J. Krinke. Dependence clusters in source code. *ACM Transactions on Programming Languages and Systems*. to appear.

[17] M. Harman, Y. Hassoun, K. Lakhotia, P. McMinn, and J. Wegener. The impact of input domain reduction on search-based test data generation. In *ACM Symposium on the Foundations of Software Engineering (FSE '07)*, pages 155–164, Dubrovnik, Croatia, September 2007. Association for Computer Machinery.

[18] M. P. E. Heimdahl, J. M. Thompson, and M. W. Whalen. On the effectiveness of slicing hierarchical state machines: A case study. In *EUROMICRO '98: Proceedings of the 24th Conference on EUROMICRO*, pages 10435–10444, Washington, DC, USA, 1998. IEEE Computer Society.

[19] M. P. E. Heimdahl and M. W. Whalen. Reduction and slicing of hierarchical state machines. In *Proc. Fifth ACM SIGSOFT Symposium on the Foundations of Software Engineering*. Springer–Verlag, 1997.

[20] R. Komondoor and S. Horwitz. Semantics-preserving procedure extraction. In *Proceedings of the 27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL-00)*, pages 155–169, N.Y., Jan. 19–21 2000. ACM Press.

[21] B. Korel, G. Koutsogiannakis, and L. H. Tahat. Model-based test prioritization heuristic methods and their evaluation. In *A-MOST '07: Proceedings of the 3rd international workshop on Advances in model-based testing*, pages 34–43, New York, NY, USA, 2007. ACM.

[22] B. Korel and J. Rilling. Dynamic program slicing in understanding of program execution. In 5^{th} *IEEE International Workshop on Program Comprenhesion (IWPC'97)*, pages 80–89, Los Alamitos, California, USA, May 1997. IEEE Computer Society Press.

[23] B. Korel, I. Singh, L. Tahat, and B. Vaysburg. Slicing of state based models. In *IEEE International Conference on Software Maintenance (ICSM'03)*, pages 34–43, Los Alamitos, California, USA, Sept. 2003. IEEE Computer Society Press.

[24] S. Labbé and J.-P. Gallois. Slicing communicating automata specifications: polynomial algorithms for model reduction. *Formal Aspects of Computing*, 2008.

[25] V. Ojala. A slicer for UML state machines. Technical Report HUT-TCS-25, Helsinki University of Technology Laboratory for Theoretical Computer Science, Espoo, Finland, 2007.

[26] S. Ramesh, A. Kulkarni, and V. Kamat. Slicing tools for synchronous reactive programs. *SIGSOFT Softw. Eng. Notes*, 29(4):217–220, 2004.

[27] V. P. Ranganath, T. Amtoft, A. Banerjee, M. B. Dwyer, and J. Hatcliff. A new foundation for control-dependence and slicing for modern program structures. In *ESOP*, pages 77–93, 2005.

[28] V. P. Ranganath, T. Amtoft, A. Banerjee, J. Hatcliff, and M. B. Dwyer. A new foundation for control dependence and slicing for modern program structures. *ACM Trans. Program. Lang. Syst.*, 29(5):27, 2007.

[29] X. Ren, O. Chesley, and B. G. Ryder. Identifying failure causes in java programs: An application of change impact analysis. *IEEE Transactions on Software Engineering*, 32(9):718–732, 2006.

[30] F. Strobl and A. Wisspeintner. Specification of an elevator control system – an autofocus case study. Technical Report TUM-I9906, Technische Univerität München, 1999.

[31] F. Tip. A survey of program slicing techniques. *Journal of Programming Languages*, 3(3):121–189, Sept. 1995.

[32] T. Wang and A. Roychoudhury. JSlice: dynamic slicing tool for Java.

[33] R. Y. Zaghal and J. I. Khan. EFSM/SDL modeling of the original tcp standard (RFC793) and the congestion control mechanism of TCP Reno. Technical Report TR2005-07-22, Internetworking and Media Communications Research Laboratories, Department of Computer Science, Kent State University, March 2005.