Towards Slicing Communicating Extended Automata

Sébastien Labbé and Jean-Pierre Gallois

CEA-LIST, Laboratoire Logiciels pour la Sûreté des Procédés,
F-91191 Gif-sur-Yvette, France
{sebastien.labbe,jean-pierre.gallois}@cea.fr

Abstract. Slicing is a well-established program analysis technique that has applications in debugging, program understanding and model reduction. This paper presents an approach to slicing formal specifications based on communicating extended automata.

1 Introduction

Almost thirty years ago, Weiser introduced the concept of program slicing [5] as a static analysis technique, aimed at facilitating the debugging and understanding of programs expressed in classical imperative programming languages. Informally, a slice of a program P is a set of statements of P, which are relevant to the computations performed at some point(s) of interest in P.

In recent years, the slicing literature has become extensive, therefore some useful surveys have been published [4]. Considering the evolution of specification languages, and their wide use in formal methods, we address the problem of slicing formal specifications based on communicating extended automata. Finding a solution is not straightforward because some significant differences lie between the features of communicating extended automata and those of imperative programs, mainly the unique end node property and communications via channels. In this paper we state the new definitions that form the cornerstone of our approach to slicing specifications based on communicating extended automata. Prospects of our work appear in several fields of formal methods, including simulation, test-case generation and model checking.

2 Communicating Extended Automata

Extended input/output Labelled Transition Systems (ELTS) [3] are automata used to specify the behavior of reactive systems. ELTS extend classical labelled transition systems by enabling communications via channels, and assignments of values to variables.

For the remainder of this paper, we assume the reader is familiar with some first-order logic notions: terms and quantifier-free formulas of a first order language. Let $\mathscr{L}$ be a first order language, C a set of channel symbols, V a set of variables, and τ a silent action. We note $\mathscr{T}_{\mathscr{L}}(V)$ the set of $\mathscr{L}$-terms, and $\mathscr{F}_{\mathscr{L}}(V)$ the set of quantifier-free $\mathscr{L}$-formulas, whose variables are elements of V.

Definition 1 (Communication actions). *The* $\mathscr{L}$-communication-actions *defined over* C *and* V *are the elements of* $Act_{\mathscr{L}}(C,V)$*, such that:*

$$Act_{\mathscr{L}}(C,V) = \{\tau\} \cup \{c?x \mid c \in C \wedge x \in V\} \cup \{c!t \mid c \in C \wedge t \in \mathscr{T}_{\mathscr{L}}(V)\}$$

Definition 2 (Extended input/output LTS (ELTS)). *An* ELTS *defined over* $\mathscr{L}$*,* V *and* C *is a tuple* $(S, s_0, T, V, C)_{\mathscr{L}}$ *where* S *is a set of states,* $s_0 \in S$ *is the initial state and* $T \subseteq S \times Act_{\mathscr{L}}(C,V) \times \mathscr{F}_{\mathscr{L}}(V) \times \mathscr{T}_{\mathscr{L}}(V)^V \times S$ *is a transition relation, where* $\mathscr{T}_{\mathscr{L}}(V)^V$ *is the set of functions from* V *to* $\mathscr{T}_{\mathscr{L}}(V)$*.*

3 Slicing Formal Specifications

One of the main approaches to calculating program slices is dependence-based, i.e it involves the computation of dependence relations between the program statements. In a *dependence graph*, nodes represent the program statements and edges represent the dependence relations. When all the dependence relations are transitive, slicing is a simple reachability problem in the dependence graph [4].

In our framework, we consider formal specifications as several ELTS that run concurrently, and communicate with each other and with their environment. Formally, a *specification* $\mathscr{S}_{\mathscr{L}}$ is a set of ELTS, each of which is defined over the first-order language $\mathscr{L}$. Our approach to slicing formal specifications based on ELTS is inspired on previous works on dependence-based program slicing. We extend to ELTS the classical notions of control dependence and data dependence, then we define an extra dependence relation to handle the data flows across communication channels. Having calculated the three sets of dependences (i.e the dependence graph), our solution to a slicing problem is the solution to a reachability problem in the dependence graph. For the remainder of this section, we assume $\mathscr{E} = (S, s_0, T, V, C)_{\mathscr{L}}$ is an ELTS, and we call *path* in $\mathscr{E}$ a sequence of consecutive transitions in T.

The traditional definition of control dependence assumes the structure under analysis satisfies the unique end node property, and thus is not directly applicable to ELTS. Ranganath et al. [2] addressed this issue in a recent work on which our definition of control dependence is based.

Definition 3 (Control dependence $(tr' \xrightarrow{cd} tr)$**).** *A transition* $tr \in T$ *is* control dependent *on a transition* $tr' \in T$ *if* tr' *has at least two successors,* tr'_1 *and* tr'_2*, such that: for all maximal paths from* tr'_1 *in* $\mathscr{E}$*,* tr *always occurs, and there exists a maximal path from* tr'_2 *in* $\mathscr{E}$ *on which* tr *does not occur.*

Let $tr = (s_1, a, f, \sigma, s_2)$ be a transition in T, and x be a variable in V. We say x is *defined* at tr if either $\sigma(x) \neq x$, or $a = c?x$ for some $c \in C$; and x is *used* at tr if x is a variable of f, or there is a term $t \in \mathscr{T}_{\mathscr{L}}(V)$ such that x is a variable of term t, and either $\sigma(y) = t$ for some $y \in V$, or $a = c!t$ for some $c \in C$.

Definition 4 (Data dependence $(tr' \xrightarrow{dd} tr)$**).** *A transition* $tr \in T$ *is* data dependent *on a transition* $tr' \in T$ *if there exists a variable* $x \in V$ *and a path* $p = \langle tr', \ldots, tr \rangle$ *in* $\mathscr{E}$ *such that* x *is defined at* tr'*,* x *is used at* tr*, and for all* $tr'' \in (p \backslash \{tr'\})$*,* x *is not defined at* tr''*.*

Informally, there is a communication dependence between two transitions in two different ELTS if there exists a channel that potentially allows a data flow between these two transitions. Let $\mathscr{S}_{\mathscr{L}}$ be a specification; let $\mathscr{E} = (S, s_0, T, V, C)_{\mathscr{L}}$ and $\mathscr{E}' = (S', s'_0, T', V', C')_{\mathscr{L}}$ be two ELTS in $\mathscr{S}_{\mathscr{L}}$.

Definition 5 (Communication dependence ($tr' \xrightarrow{com} tr$)). *A transition in T, $tr = (s_1, a, f, \sigma, s_2)$, is* communication dependent *on a transition in T', $tr' = (s'_1, a', f', \sigma', s'_2)$, if there exists a communication channel $c \in C \cap C'$ such that $a = c?x$ for some $x \in V$, and $a' = c!t$ for some $t \in \mathscr{T}_{\mathscr{L}}(V')$.*

A transition tr is *dependent* on a transition tr' ($tr' \xrightarrow{d} tr$), if either $tr' \xrightarrow{cd} tr$, $tr' \xrightarrow{dd} tr$, or $tr' \xrightarrow{com} tr$. Informally, a slice of a specification is a set of transitions, on which the slicing criterion (itself a set of transitions) is *transitively* dependent.

Definition 6 (Slice of a specification). *Let $\mathscr{S}_{\mathscr{L}} = \{\mathscr{E}_0, \ldots, \mathscr{E}_k\}$ be a specification, for some $k \geq 0$; for all $0 \leq i \leq k$, we note T_i the set of transitions in $\mathscr{E}_i$. Let $Crit \subseteq \bigcup_{0 \leq i \leq k} \{T_i\}$; the following set is a slice wrt. Crit:*

$$Slice_{Crit} = \{tr' \mid tr \in Crit \wedge tr' \xrightarrow{d}{}^{*} tr\}$$

4 Open Issues and Conclusion

Due to the lack of space, many important issues were not discussed here, in particular the design of methods to perform on ELTS the dataflow analyses required to compute the dependences defined above; whether the dependence relations are always transitive on ELTS (it involves the precision of our definition of a slice); how to handle shared variables across ELTS etc. These are some of the issues that will be addressed in future work.

The main result of this paper is that the benefits of slicing can be obtained on formal specifications based on communicating extended automata. We are currently implementing this technique in the Agatha tool [1], allowing us to perform slicing for specification debugging and understanding, as a straightforward application. We will then investigate the advantages of slicing for specification validation and simulation, and test-case generation.

References

1. C. Bigot, A. Faivre, J.-P. Gallois, A. Lapitre, D. Lugato, J.-Y. Pierron, and N. Rapin. Automatic test generation with agatha. In *TACAS*, pages 591–596, 2003.
2. 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.
3. N. Rapin, C. Gaston, A. Lapitre, and J.-P. Gallois. Behavioural unfolding of formal specifications based on communicating extended automata. In *ATVA*, 2003.
4. F. Tip. A survey of program slicing techniques. *J. Prog. Lang.*, 3(3), 1995.
5. M. Weiser. Program slicing. In *ICSE*, pages 439–449, 1981.