Complexity Metrics for Hierarchical State Machines

Mathew Hall[*]

Department of Computer Science, University of Sheffield, UK
m.hall@dcs.shef.ac.uk

Abstract. Automatically generated state machines are constrained by their complexity, which can be reduced via hierarchy generation. A technique has been demonstrated for hierarchy generation, although evaluation of this technique has proved difficult.

There are a variety of metrics that can be used to provide indicators of how complicated a state machine or statechart is, one such example is cyclomatic complexity (the number of edges - the number of states + 2). Despite this, the existing complexity metric for statecharts does not operate on the hierarchy, instead providing an equivalent cyclomatic complexity for statecharts by ignoring it.

This paper defines two new metrics; Top Level Cyclomatic Complexity and Hierarchical Cyclomatic Complexity. These metrics assess the complexity of a hierarchical machine directly, as well as allowing for comparison between the original, flat state machine and its hierarchical counterpart.

1 Introduction

One of the problems that precludes the widespread use of state machines is their inefficient growth of complexity with size; as the number of states increase, the likelihood of duplication of parts of the machine increases. Complexity impacts understandability [1], meaning state machines get less understandable as they grow, jeopardising their utility in reverse engineering scenarios.

Adding a hierarchy to a machine reduces its complexity by providing abstraction, breaking the machine into multiple levels of behavioural detail. State hierarchies were originally proposed by Harel in statecharts [4] which add "superstates", which contain more superstates or simple states to represent a hierarchy.

Search-based hierarchy generation has been used to generate groupings for software module dependency graphs based on the number of edges between groups of modules [9]. An information-theoretic method has also been used [6] to accomplish the same goal. The former method has also been used to produce hierarchies for state machines [3].

Published results of these techniques both exhibit a common flaw; in the absence of an expert, evaluation of results is difficult. Metrics allow results to

[*] Supported by EPSRC grant EP/F065825/1 REGI: Reverse Engineering State Machine Hierarchies by Grammar Inference.

be compared and ranked without the requirement of a pre-determined solution to compare them to (or human participation to provide ranking). Structural properties affect the cognitive complexity of software [10]; therefore metrics that express these properties numerically express the complexity of the artefact under investigation.

Statechart specific metrics have been defined, although none take into account the hierarchy of the states, but simply the connectivity between them. This work proposes two metrics for this purpose; principally to facilitate evaluation of automatic hierarchy generation techniques. In addition to these metrics, this work also formally defines a hierarchical state machine as an extension of a labeled transition system.

2 Background

One oft-used metric is McCabe Cyclomatic Complexity (CC) [8]. Defined as $E - N + 2$, where E is the number of edges in a graph, and N is the number of nodes, it offers an estimate of how complex an artefact is. Originally applied to software systems (where the graph is the control-flow graph) it gives an approximate indicator of the difficulty developers will have when maintaining a module.

CC has been applied to statecharts in two ways. One such way is to calculate the sum of the CC of the code that implements each state [11]. Another is to perform the normal CC calculation, ignoring superstates, operating only on simple states, known as structural cyclomatic complexity (SCC) [2].

The value of the latter, if applied in the context of reverse engineering hierarchies would rarely change, as the number of simple states is constant. The number of transitions, however may reduce complexity [5]. The transition count is reduced whenever every state in a superstate has one or more transitions in common. These common transitions are replaced by a single transition from the superstate, resulting in fewer overall edges.

Despite this reduction of cyclomatic complexity, this metric is reduced to counting the number of edges in the machine when comparing results; rather than analysing the abstraction in the form of redistribution of complexity.

This work is applied to hierarchical state machines (HSMs). These can be viewed as an extension of a labeled transition system (LTS), a generalisation of a state machine, consisting of only states and transitions with an associated label. These extensions are defined below.

3 Hierarchical State Machines

A hierarchical machine adds a structure to the states, encoded in sets, to the machine. States represent superstates, which may contain superstates themselves, as well as zero or more simple states (a superstate must never be empty).

The standard used in this work also takes superstate transitions from the similar statechart formalism. It is possible for a transition to occur from not

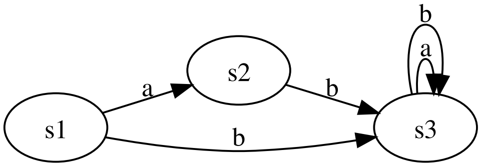

Fig. 1. The example state machine L

only a single state, but also a superstate. These transitions are shorthand for a transition occurring from every state in a superstate.

An example LTS L, given by $L = (Q, T) = (\{s_1, s_2, s_3\}, T\})$ where $T = \{(s_1, a, s_2),\ (s_1, b, s_3),\ (s_2, b, s_3),\ (s_3, a, s_3),\ (s_3, b, s_3)\}$ is shown in figure 1. Q encodes the states in the machine, while T encodes the transitions. T is a set of transitions, each $t \in T$ is a tuple - (s, l, d) where s is the source of the transition (any element of Q), l is the label for the transition and d is the destination (again, $d \in Q$).

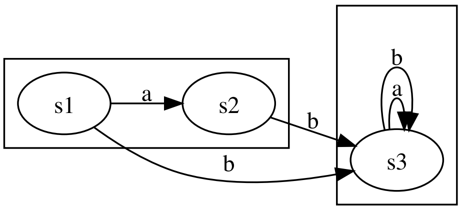

Fig. 2. One possible HSM for L

An HSM is defined as (H, T); figure 2 gives an example hierarchy for L. In an HFSM the set of states is expanded to the set of superstates and basic states, which encodes the hierarchy in nested sets. In the example $H = \{\{s_1, s_2\}, \{s_3\}\}$.

The set of transitions T is also modified, such that the source of any transition s may be any element of one of the sets in H, although the destination d must still be a basic state (i.e. any element of Q).

Not shown in this example, there is a common transition that could be replaced by a single superstate transition,$(\{s_1, s_2\}, b, s_3)$.

4 Metrics for HFSMs

Two metrics are defined, $TLCC$, which provides an overview of the abstracted machine, and HCC which conveys the complexity of the whole hierarchy. These metrics are defined for HSMs as defined in section 3.

4.1 Top-Level Cyclomatic Complexity

Top-level cyclomatic complexity is the result of viewing the HFSM at the maximum level of abstraction. It is equivalent to the CC of the result of transforming every top-level superstate into a single state, its edges being those that left the superstate it is made from.

Abstracting this to an HSM, E is taken to be the size of the number of inter-edges of a machine and N is taken to be the number of top-level superstates of the machine, $|H|$. As will be shown in section 4.2, it can also be used to operate on superstates themselves; in this case the immediate superstates become the top-level states used in the calculation.

This results in the definition:

$$TLCC(H) = \sum_{c \in H} |inter(c)| - |H| + 2$$

Where:

$$inter(c) = \{(s, l, d) \in edges(c) : !in(c, d)\}$$

$inter(c)$ gives all the edges that leave a given state or superstate, that is, their destination is not within c or its substates. Conversely, the edges that do not leave a superstate can be obtained by $edges(c) \setminus inter(c)$.

$edges(c)$ is all the edges of a superstate or state c, as well as the edges of the states that make up a superstate (and so on). It is given by

$$edges(c) = \{(s, l, d) \in T : in(c, s)\}$$

in is a recursive predicate that indicates membership of a state or superstate in another superstate:

$$in(c, s) = \begin{cases} true, & \text{if } c = s \\ true, & \text{if } \exists a.(in(a, s)) : a \in c \\ false, & \text{otherwise} \end{cases}$$

Figure 4 shows the effective graph $TLCC$ operates on when calculating for figure 3. In this example $TLCC = 20 - 3 + 2 = 19$. This illustrates the loss of

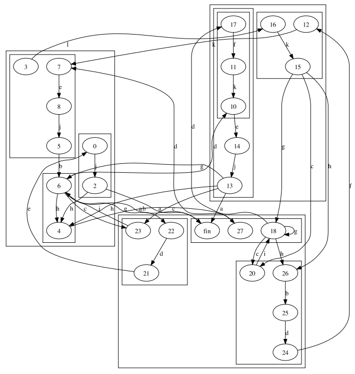

Fig. 3. An example HSM

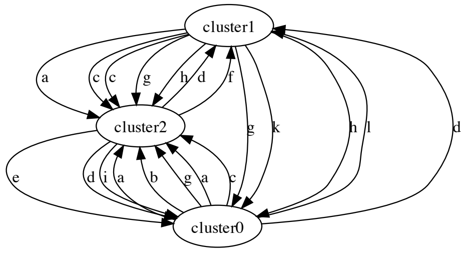

Fig. 4. The effective HSM $TLCC$ operates on

abstraction and also highlights a possible source of complexity; repeated edges. It may be that the overall CC for a machine is less than the $TLCC$ for it purely because the number of nodes (states) to be considered is reduced, but the number of edges only reduces when a transition can be formed from a superstate, which may not arise in some cases.

This metric effectively quantifies the cyclomatic complexity of the top level of abstraction; capturing the "first look" complexity a human perceives when starting to work with the hierarchical machine. This metric still does not attain the goal of quantifying the overall complexity, however.

4.2 Hierarchical Complexity

To combat the shortcomings of TLCC as well as the lack of hierarchy consideration of SCC a metric is proposed; this metric captures the complexity of the whole hierarchy, applying a reduced weight to nodes appearing lower in the hierarchy. Although they contribute less to the complexity, they are still counted; additionally, different lower-level hierarchy arrangements result in different values, which is not the case for the metrics discussed in sections 2 and 4.1

This metric is given by:

$$HCC(S,T,d) = \frac{TLCC(H,T)}{d} + \sum_{c \in H} HCC(c,T,d+1)$$

As it is recursive, HCC is calculated with an initial depth 1, to apply it to a HSM $X = (H,T)$ the calculation starts with $HCC(H,T,1)$. The final parameter is the depth of the current superstate. The example given in figure 4 has a HCC of $19 + (2+1+0)/2 + (0+1+1)/3 + (0+0+0)/3 + (1+0)/3 + 1/4 = 21.75$. This contrasts to its lower SCC of 14.

5 Conclusions and Future Work

This paper presents two novel metrics designed specifically for statecharts. The first, $TLCC$ provides an estimate of the complexity the abstraction at its most simple level. The second, HCC, allows a single metric to capture the quality of abstraction offered by a particular hierarchy for a state machine.

Both metrics produce higher values than SCC, although this is because they compare different things. SCC is lower as the number of states is always higher, whereas for $TLCC$ the number of nodes often drops, resulting in a higher overall value.

Although the metrics are now defined, they still need to undergo validation; a method for validation of similar metrics exists proposed [7], this could easily be applied to these metrics.

This work is similar to fitness functions for search-based clustering algorithms (as mentioned in section 2); where the goal is to provide an objective value of a candidate solution to direct a search. The difference between the two techniques is the goal; fitness functions seek to approximate an absolute measure of quality, whereas metrics are often used for relative comparison. These goals are similar, however and these proposed metrics could also be used to optimise a clustering, or the fitness functions used as metrics.

References

1. Cruz-Lemus, J.A., Maes, A., Genero, M., Poels, G., Piattini, M.: The impact of structural complexity on the understandability of UML statechart diagrams. Information Sciences 180(11), 2209–2220 (2010)
2. Genero, M., Miranda, D., Piattini, M.: Defining metrics for UML statechart diagrams in a methodological way. In: Jeusfeld, M.A., Pastor, Ó. (eds.) ER Workshops 2003. LNCS, vol. 2814, pp. 118–128. Springer, Heidelberg (2003)
3. Hall, M., McMinn, P., Walkinshaw, N.: Superstate identification for state machines using search-based clustering. In: Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation, pp. 1381–1388 (July 2010)
4. Harel, D.: Statecharts: A visual formalism for complex systems. Science of Computer Programming 8, 231–274 (1987)
5. Kumar, A.: Schaem: A method to extract statechart representation of FSMs. In: IEE International Advance Computing Conference 2009 (January 2009)
6. Lutz, R.: Evolving good hierarchical decompositions of complex systems. Journal of systems architecture 47(7), 613–634 (2001)
7. Genero, M., David Mir, M.P.: Empirical validation of metrics for UML statechart diagrams.... Quantitative Approaches in Object-oriented... (January 2002)
8. McCabe, T.: A complexity measure. IEEE Transactions on Software Engineering 2(4), 308–320 (1976)
9. Mitchell, B.S., Mancoridis, S.: On the automatic modularization of software systems using the bunch tool, vol. 32, pp. 193–208 (2006)
10. Poels, G., Dedene, G.: Measures for assessing dynamic complexity aspects of object-oriented conceptual schemes. In: Laender, A.H.F., Liddle, S.W., Storey, V.C. (eds.) ER 2000. LNCS, vol. 1920, pp. 499–512. Springer, Heidelberg (2000)
11. Yacoub, S., Ammar, H., Robinson, T.: Dynamic metrics for object oriented designs. In: Proceedings of Sixth International Software Metrics Symposium 1999, pp. 50–61 (1999)