SWEET – A Tool for WCET Flow Analysis (Extended Abstract)

Björn Lisper

School of Innovation, Design, and Engineering, Mälardalen University,
SE-721 23 Västerås, Sweden

1 Introduction

Worst-Case Execution Time (WCET) analysis [14] aims to estimate the longest possible execution time for a piece of code executing uninterrupted on a particular hardware. Such WCET estimates are used when analysing real-time systems with respect to possible deadline violations. For safety-critical real-time systems, *safe* (surely not underestimating) estimates are desirable. Such estimates can be produced by a *static WCET analysis* that takes all possible execution paths and corresponding hardware states into account.

Static WCET analysis has been around for 20 years, and a number of tools have emerged such as aiT [5], Otawa [2], Bound-T [8], Heptane [3], TuBound [11], and Chronos [10]. Most tools today use the so-called "Implicit Path Enumeration Technique" (IPET) [12]. In IPET, execution times are estimated from local WCET bounds for small program fragments (typically basic blocks). Each such fragment p is given an *execution counter* $\#p$ recording its number of executions: the execution time for a path is then approximated from above by the sum $\sum_p WCET(p) \times \#p$, where $WCET(p)$ is the local WCET bound for p. WCET estimation can now be formulated as maximising this sum subject to *program flow constraints* on the execution counters. If these constraints are linear, then the WCET estimation becomes an Integer Linear Programming (ILP) problem that can be solved by a standard ILP solver. It turns out that very many important program flow constraints can be expressed as linear constraints.

Thus, in the IPET model the WCET estimation problem is nicely decomposed into three distinct parts: the *low-level analysis*, which computes local WCETs using hardware timing models, the *program flow analysis* that derives program flow constraints ("Flow Facts") from the program code, and the final *calculation* where the ILP problem is solved to produce the WCET bound. Notably the program flow analysis will not need any information about hardware timing, but can be based entirely on the functional semantics of the code. Flow Facts can indeed be seen as a special kind of loop invariants.

2 SWEET

SWEET (SWEdish Execution Time tool) is a tool that derives Flow Facts automatically. SWEET can compute a variety of Flow Facts, from simple loop

T. Margaria and B. Steffen (Eds.): ISoLA 2014, Part II, LNCS 8803, pp. 482–485, 2014.

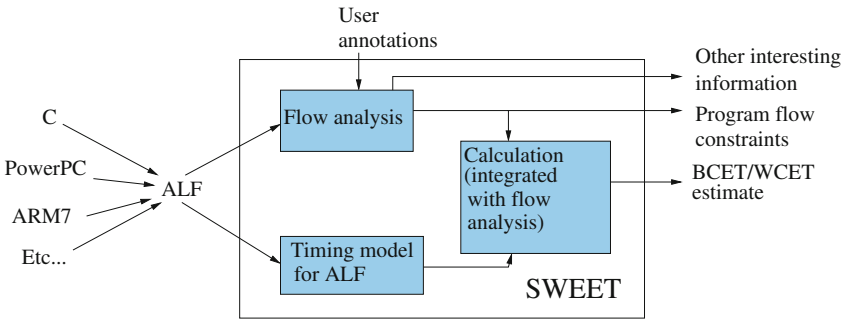

Fig. 1. The structure of SWEET

iteration bounds to complex infeasible path constraints. It can analyze a variety of code formats through translation into the intermediate format ALF [6]. SWEET is open source: comprehensive information about the tool is found online [13]. In Fig. 1 the structure of SWEET is shown.

An earlier version of SWEET ("NIC-SWEET") was integrated into a research compiler, and could analyze code generated by that compiler. This version of SWEET was a full WCET analysis tool using the IPET model. The current version ("ALF-SWEET") is a specialised program flow analysis tool.

SWEET uses *Abstract Execution* (AE) [7] to derive Flow Facts. AE can be seen as a very context-sensitive value analysis, where different loop iterations are analysed separately. This gives the analysis a flavor of symbolic execution, executing the program in the abstract domain using abstract states rather than concrete states. AE is based on the theory of abstract interpretation: thus it is safe, and computed Flow Facts will never underestimate the set of possible program paths. SWEET currently uses an abstract domain of bounded intervals, but AE also works with other abstract domains.

Abstract states reaching a condition may contain concrete states for which the condition evaluates to true and false, respectively. Then the abstract state is split into a different abstract state for each outcome of the condition. To curb the potential explosion of states SWEET offers a variety of *merge strategies*, where abstract states are merged in certain program points using their least upper bound operator. By selecting the proper strategy, the tradeoff between precision and analysis speed can be fine-tuned.

AE is a potentially very general technique to derive Flow Facts. It can in principle deal with loops of any form, as long as the abstract domain can express the semantics of the loop conditions accurately enough. AE can also bound recursion depth. The price to pay for this generality is a risk of nontermination. The current implementation in SWEET has some limitations: recursion is not handled, as well some forms of unstructured loops. The use of interval domain also yields some limitations. SWEET currently handles nontermination by allowing the user to set a timeout where the analysis is aborted.

The environment of the analysed code may be important to know for the analysis. For instance, the values of some variables may be confined to certain ranges. SWEET provides *abstract input annotations*, where such constraints can be specified. The AE can use this information to compute tighter Flow Facts.

SWEET uses *recorders* and *collectors* to compute Flow Facts during the AE [7]. Recorders are attached to abstract states, and contain information that is successively accumulated into the collectors during the abstract execution. Collectors are pertinent to *scopes* (typically loops), and their final values are used to produce Flow Facts for that scope. For instance, to compute an upper loop bound the recorder is the execution counter for the loop header, and the collector is a number containing the highest value of this counter seen for any abstract state in the loop so far. Other, more complex Flow Facts are generated using more elaborate recorders and collectors.

SWEET can compute different kinds of Flow Facts specified by a combination of attributes telling the type of bound (upper/lower/infeasible), where to put execution counters, and Flow Fact context. The Flow Facts can thus be context sensitive (call strings), and they can pertain to different scopes (e.g., an execution counter for the loop body in a nested loop can be relative to either the inner or outer loop). SWEET has an expressive language for expressing these Flow Facts. In addition, SWEET can generate Flow Facts in the annotation formats of the commercial WCET analysis tools aiT and RapiTime.

In order to keep SWEET portable across different formats it analyses the intermediate format ALF [6]. Other languages and formats can be analysed if translated into ALF. To facilitate this, ALF is designed to faithfully represent high-level languages (like C) as well as machine code. Currently two translators from C to ALF exist, as well as a translator from PowerPC binaries to ALF.

The current version of SWEET lacks a low-level analysis. It can however use simple timing models for ALF to obtain WCET estimates. This estimation is done directly in the AE by treating time as a variable being incremented for each executed statement [4]. The AE thus computes an interval bounding the execution time. This interval also bounds the execution time from below, thus providing a Best Case Execution Time (BCET) estimate. Such simple cost models are way to coarse to provide both safe and tight WCET/BCET bounds, but they can nevertheless be useful to provide approximate bounds, for instance for early source-level timing estimation [1].

SWEET can also provide information from its rich set of internal analyses supporting the AE. Such analyses include a conventional value analysis, data flow analysis, construction of control flow and call graphs, and program slicing.

3 Conclusions

We have presented SWEET, a tool for generating precise Flow Facts. It is designed for maximal interoperability. It can be used both as a standalone analysis tool, or as a "plugin" providing Flow Facts to other tools: indeed, SWEET is an important component in the Open Timing Analysis Platform [9]. Its main use is however as a vehicle for program analysis research targeting real-time code.

References

1. Altenbernd, P., Ermedahl, A., Lisper, B., Gustafsson, J.: Automatic generation of timing models for timing analysis of high-level code. In: Faucou, S. (ed.) Proc. 19th International Conference on Real-Time and Network Systems (RTNS 2011), Nantes, France (Sepember 2011)
2. Ballabriga, C., Cassé, H., Rochange, C., Sainrat, P.: OTAWA: An open toolbox for adaptive WCET analysis. In: Min, S.L., Pettit, R., Puschner, P., Ungerer, T. (eds.) SEUS 2010. LNCS, vol. 6399, pp. 35–46. Springer, Heidelberg (2010)
3. Colin, A., Puaut, I.: A modular and retargetable framework for tree-based WCET analysis. In: Proc. 13th Euromicro Conference on Real-Time Systems (ECRTS 2001) (June 2001)
4. Ermedahl, A., Gustafsson, J., Lisper, B.: Deriving WCET bounds by abstract execution. In: Healy, C. (ed.) Proc. 11th International Workshop on Worst-Case Execution Time Analysis (WCET 2011), Porto, Portugal (July 2011)
5. Ferdinand, C., Heckmann, R., Franzen, B.: Static memory and timing analysis of embedded systems code. In: 3rd European Symposium on Verification and Validation of Software Systems (VVSS 2007), Eindhoven, The Netherlands, pp. 153–163. No. 07-04 in TUE Computer Science Reports (March 2007)
6. Gustafsson, J., Ermedahl, A., Lisper, B., Sandberg, C., Källberg, L.: ALF – a language for WCET flow analysis. In: Holsti, N. (ed.) Proc. 9th International Workshop on Worst-Case Execution Time Analysis (WCET 2009), pp. 1–11. OCG, Dublin (2009)
7. Gustafsson, J., Ermedahl, A., Sandberg, C., Lisper, B.: Automatic derivation of loop bounds and infeasible paths for WCET analysis using abstract execution. In: Proc. 27th IEEE Real-Time Systems Symposium (RTSS 2006), pp. 57–66. IEEE Computer Society, Rio de Janeiro (2006)
8. Holsti, N., Saarinen, S.: Status of the Bound-T WCET tool. In: Proc. 2nd International Workshop on Worst-Case Execution Time Analysis, WCET 2002 (2002)
9. Huber, B., Puffitsch, W., Puschner, P.: Towards an open timing analysis platform. In: Healy, C. (ed.) Proc. 11th International Workshop on Worst-Case Execution Time Analysis (WCET 2011), Porto, Portugal (July 2011)
10. Li, X., Liang, Y., Mitra, T., Roychoudhury, A.: Chronos: A timing analyzer for embedded software. Science of Computer Programming 69(1-3), 56–67 (2007)
11. Prantl, A., Schordan, M., Knoop, J.: TuBound – a conceptually new tool for worst-case execution time analysis. In: Kirner, R. (ed.) Proc. 8th International Workshop on Worst-Case Execution Time Analysis (WCET 2008), Prague, Czech Republic, pp. 141–148 (July 2008)
12. Puschner, P.P., Schedl, A.V.: Computing maximum task execution times – a graph-based approach. Journal of Real-Time Systems 13(1), 67–91 (1997)
13. SWEET home page (2011), `http://www.mrtc.mdh.se/projects/wcet/sweet/`
14. Wilhelm, R., Engblom, J., Ermedahl, A., Holsti, N., Thesing, S., Whalley, D., Bernat, G., Ferdinand, C., Heckmann, R., Mitra, T., Mueller, F., Puaut, I., Puschner, P., Staschulat, J., Stenström, P.: The worst-case execution time problem — overview of methods and survey of tools. ACM Transactions on Embedded Computing Systems (TECS) 7(3), 1–53 (2008)