Dynamics for WCET

David Griffin and Alan Burns

Abstract
This paper examines the link between recent advances in dynamical systems, the mathematical study of systems which evolve under a a fixed rule, and existing WCET analysis techniques. In particular it focuses on the fact that all WCET techniques implicitly use a common dynamical system, through which it is possible to translate information from one technique such that it is of use to another technique.

1. Introduction

One of the basic principals that govern static analysis techniques is to model the state of a computer system as it executes instructions based on what instructions are feasible from flow analysis; in fact this is the direct approach of plain model checking [3], an exhaustive search over all possible states. However, as this can very quickly lead to the state space explosion problem, many static analysis techniques implement some kind of countermeasure, trading the tightness of the result for the ability to consider many concrete states of the computer system with a single abstract state and thus arrive at a result in a reasonable amount of time. Two such methods are abstract interpretation [2], which lazily combines states at join points, and symbolic model checking [1], which uses a prespecified simplified state space.

One unintended consequence of the different types of technique available is that they are not either readily combinable or easily comparable. They are not combinable in the sense that whilst one method could be used to derive a bound on the WCET for a subroutine and this embedded into the analysis of a greater system using a different method, the embedding happens as a "black box" and hence it is reasonable to assume that some information is lost. For instance, the subroutine may not be used in a particular way by the calling routine, and hence a tighter bound could have been obtained with this knowledge. The different techniques are not easily comparable in the sense that whilst there is lively debate about the results and possible causes for the failures of a given technique on a certain class of problems [6, 7], this discussion is based upon the observed results of each method and not by comparing the inner workings of the methods as there is no common ground on which to conduct a direct comparison.

Given that all of these techniques are solving the same problem using similar principals, the lack of a common base upon which the methods can be combined or compared seems nonsensical. A common base would enable, for example, the easy embedding of highly accurate model checking on a bottlenecked subroutine within a greater program which used abstract interpretation for the majority given the improved scaling. It would also enable to determine exactly where and when one technique stops working as well as another, and hence may lead to more accurate comparison of the merits of different techniques.

To rectify this the topic of dynamical systems is applied to the WCET problem. Dynamical systems is a branch of mathematics that deals with how a deterministic system progresses with the passage of time. However, current work by Giunti and Mazzola [4] indicates that time may be substituted with any monoid, which in turn enables other the theory of dynamical systems to be applied to other problems. Section 2 takes the current state of the art theory and applies it to computer systems and specifically the WCET problem, defining a dynamical system for use in the WCET problem. Section 3 proceeds to show how abstract interpretation, model checking and symbolic model checking can all be mapped into this dynamical system. Analysis of how techniques may be compared or combined is presented in Section 4

2. Dynamics for WCET

Before any techniques can be explained in terms of dynamical systems, it is first necessary to define an appropriate dynamical system. This section provides a brief introduction to and definitons of the concepts necessary to define a dynamical system, in addition to the WCET dynamical system. A dynamical system, as defined by Gientu and Mazzola [4], is defined as a tuple of three items:

- T: a monoid (typically written additively) defining a form of time for the system
- M: a set of states which the system may be in
- $\Phi : T \times M \to M$: a time evolution function which determines how states progress given the passage of time

The first item to define is an appropriate definition of time. A monoid is a set T with a binary operator $+$ that has the properties of closure ($\forall a, b \in T, a + b \in T$), associativity ($(a + b) + c = a + (b + c) \forall a, b, c \in T$) and an identity element ($\exists 0 \in T$ such that $a + 0 = a = 0 + a \; \forall a \in T$). One important note is that the binary operator is written as $+$ by convention of dynamical systems; normally $+$ is reserved for commutative ($a + b = b + a \; \forall \; a, b \in T$) operators, but this need not be true in this case. Examples of monoids used in computer science are the natural numbers including zero under addition or the transition monoid that can be used to describe a finite state automaton.

In WCET analysis, existing techniques essentially track the progression of a computer systems state as the system executes a series of instructions. Hence the appropriate measure of “time” for the WCET dynamical system is intuitively the lists of instructions being executed, as these are what advance the computer systems state. With this in mind, a set T_0 can be defined as the set of all lists of instructions, and the addition operator $+$ can be defined as list extension. It is then trivial to verify that T_0 under $+$ is indeed a monoid:

Lemma 1. $T = (T_0, +)$ *is a monoid.*

Proof. The following properties hold:

- **Closure**: For $a, b \in T$, a and b are lists of instructions, so $a + b$ is a list of instructions, so $a + b \in T$.
- **Associativity**: For $a, b, c \in T$, $(a + b) + c$ is a list of the elements of a, followed by the elements of b, followed by the elements of c. This is also a description of $a + (b + c)$, hence $(a + b) + c = a + (b + c)$.

- **Identity**: The empty list [] provides an identity element as $a + [] = a = [] + a \; \forall \; a \in T$. Hence $0 = []$.

$\square$

The second item of to define in a dynamical system is the set of states which the dynamical system can occupy. Evidently this should be related to the set of states that the computer system can occupy. The set of states for the dynamical system must also comprise some additional elements, as it must also be able to provide input to the system. Hence, an appropriate set of states for the dynamical system is as follows: $M = \{(p, i) \,|\, p \in P, i \in I\}$ with P being the set of all possible computer system states, and I being the set of all lists of inputs to the system. This definition is enough to commence work on defining the time evolution function.

The time evolution function Φ is the final item to define to create a dynamical system. The time evolution function dictates how the system progresses given the passage of time; given that the time that this system is using is the instructions that the system is executing, a simple definition of the the time evolution function would be:

$$\Phi(t, (x_1, i_1)) = (x_2, i_2)$$

with x_2 the state of the computer system after executing the instructions in t, and i_2 the list of inputs that were not consumed by the instruction in t. Unfortunately, this simple definition does not take into account flow analysis; it permits that at any point any instruction may be executed, when this is clearly not the case. As static analysis cannot work without some notion of flow analysis, it is necessary to include it here. To this end it is necessary to extend the set of states for the dynamical system so that there is a notion of invalid states. Hence the final definition of the set of states M can be defined as follows:

$$\begin{aligned} M_0 &= \{(p, i) \,|\, p \in P, i \in I\} \\ M &= M_0 \cup \{X_{(p,i)} | (p, i) \in M_0\} \end{aligned}$$

where the states of the form $X_{(p,i)}$ are the invalid states, with $X_{(p,i)}$ resulting from trying to execute an instruction which cannot be executed on the state (p, i).

Using the new states it is possible to define a time evolution function which takes into account program flow as follows:

$$\begin{aligned} \Phi([], (x_1, i_1)) &= (x_1, i_1) \\ \Phi(t, (x_1, i_1)) &= \begin{cases} \Phi(tail(t), (x_2, i_2)) & \text{if the instruction } head(t) \\ & \text{could be executed on state } (x_1, i_1), \\ & \text{where } (x_2, i_2) \text{ is the result of} \\ & \text{executing } head(t) \text{ on } (x_1, i_1) \\ X_{(x_1,i_1)} & \text{otherwise} \end{cases} \\ \Phi(t, X_{(x_1,i_1)}) &= X_{(x_1,i_1)} \end{aligned}$$

This time evolution function has the property that attempting to execute an invalid instruction results in the dynamic system entering a dead end, which is equivalent to how existing systems work. It only remains to show that Φ meets the requirements to be a valid time evolution function, that is that it respects both the 0 and the addition operator of T.

Lemma 2. *Φ is a valid time evolution function*

Proof. $\Phi(0, m) = m \; \forall \; m \in M$: If $m = (x_1, i_1)$, given that $0 = []$, then clearly not executing any instructions must leave the computer system in the same state as it was before and it cannot have processed any inputs. If $m = X_(x_1, i_1)$, $\Phi([], m) = m$ by definition. Hence Φ respects the 0 of T.

$\Phi(t_2, \Phi(t_1, m)) = \Phi(t_1 + t_2, m) \; \forall \; t_1, t_2 \in T$: Let $x = \Phi(t_1, m)$. In the case that all the instructions of t are valid, given the recursive definition of Φ, when evaluating $\Phi(t_1 + t_2, m)$ after $len(t_1)$ applications of Φ, the remainder of the calculation will be $\Phi(t_2, x)$. If not all of the instructions are valid, with the first invalid instruction being executed on state (x_j, i_j), then the result of both calculations will be $X_{(x_j, i_j)}$. Hence $\Phi(t_2, \Phi(t_1, m)) = \Phi(t_1 + t_2, m) \; \forall \; t_1, t_2 \in T$ and Φ respects the addition operator of T. □

With the proof that Φ is a valid time evolution function, the following result is obvious:

Theorem 1. *(T, M, Φ) is a dynamical system*

Proof. Follows from previous results. □

One important concept in dynamical systems which is useful for the WCET problem is that of *Orbits*. In dynamical systems, orbits partition the state space; The common use of the word orbit in dynamical systems is referred to as a periodic orbit. Importantly this means that when using a dynamics based approach, the WCET problem can be split into finding the WCET of the orbits of the potential starting states. This means that a large volume of possible states can be removed from consideration altogether, in a similar fashion to the suggestions of Lim et. al. [5] who determined that narrowing down the potential beginning and finishing states would result in a tighter WCET bound with a lower computational cost. In addition, if it can be proved that two start states are not in the same orbit (that is it is not possible for them to converge on a single processor state) it is possible to easily parallelise computing the WCET whilst also having a guarantee that work will not be duplicated.

At this point it hasn't been specified exactly what is contained within the tracked state of the computer system, other than it is comparable to what is tracked within existing systems. The state which should be tracked is simply what needs to be simulated to devise a WCET bound. An important point is that this does not contain a current "working" WCET bound for computation up until that state, as this would mean that different WCET analyses produce different states if the estimate of WCET differs, which is not desirable for combining techniques. Instead, each state should be associated with a WCET bound derived from some technique. Then when combining techniques the WCET bound can be taken as the lowest bound found thus far associated with that state.

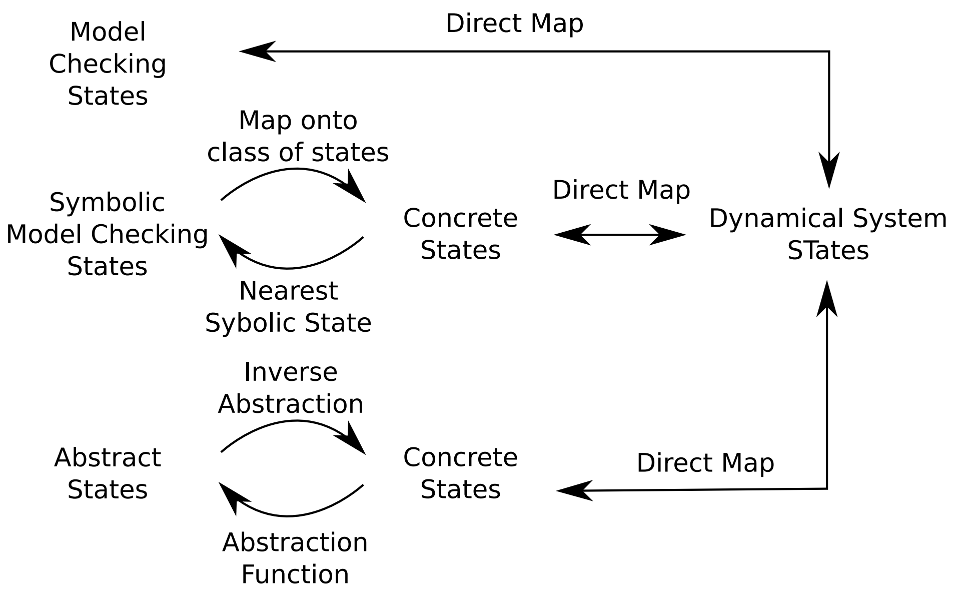

Figure 1. Illustrating how to map existing techniques into a dynamical system

3. Existing Techniques as Dynamics

As previously mentioned, existing techniques are implicitly using dynamics to find WCET. This section details how existing techniques may be translated to and from pure dynamical systems, therefore enabling mixing of techniques without losing information.

Plain model checking [3] can be thought of a an exhaustive search which determines for each reachable state the property "it this state a part of the worst case"; if the property cannot be disproved, then the search expands to all valid successor states until the wosrt case is found. This enables very tight bounds to be computed at the expense of a potentially large problem. Plain model checking is simply a direct map into the dynamical system, with any evaluated state being mapped onto the corresponding state in the dynamical system. Similarly, to convert the dynamical system for usage in model checking, any states which are required by the model checker can be used from the dynamical system. However, the caveat with this conversion is that other methods may not produce as accurate information as model checking alone could, and hence when converting from another method there may be a drop in precision when compared against model cheking throughout the WCET estimation.

Symbolic model checking [1] defines a simplified abstract machine which represents many concrete states in one symbolic state, but otherwise follows the same principals of model checking. This trades the precision of the WCET bound for a decrease in the cost of performing the WCET estimation, with the magnitude of the trade defined by the degree of abstraction. Data from symbolic model checking can be converted to a dynamical system by first mapping the symbolic state into the concrete states it defines, and mapping these concrete states into the dynamical system. Similarly the inverse method would be to map the states of the dynamical system into concrete states and then construct abstract states such that the estimated bound for the abstract state holds for all concrete states within the abstract state.

Abstract interpretation [2] identifies sets of states in the analysis which are similar and performs an abstraction step which enables the states to be represented as a single abstract state, with a corresponding loss of precision. In a sense this is similar to the approach used by symbolic model checking, but the abstraction is performed may be performed in a lazy manner rather than prespecified. Hence a similar approach to symbolic model checking in mapping is appropriate. When mapping to the dynamical system, the abstract state is mapped onto the concrete states it represents and these states mapped into the dynamical system. The inverse method needs to take into account the abstraction used by abstraction interpretation; hence the appropriate method is determine the relevant states in the dynamical system, find the associated concrete states and on these perform any necessary abstractions.

4. Analysis

With the previous explanation of how to map some common WCET techniques into the dynamical system, now follows an explanation of how to compare how the different techniques perform. Assuming that all techniques are correct and therefore do not underestimate WCET, there are two main parameters for measuring performance: the amount of effort spent to arrive at an answer, and the tightness of the derived WCET bound. The latter of these is simple to compare between two techniques: at a given point, whichever technique produces the lower bound is superior.

A measure of the amount of effort required can be obtained by examining the mapping into the dynamical system. A one-one mapping, as is the case with model checking, indicates that all possible states are being considered. Assuming that any abstraction does not add significant computational overhead - a fair assumption given that if this were the case then there would be little point in applying the abstraction - a one-many mapping implies that there is a degree of simplification in the state space. The exact degree of simplification at any point can be determined by the average number of states in the dynamical system that each state in the analysis map into. The higher this average, the greater the reduction in complexity.

Combining the two parameters in comparison is where difficulties lie. This is because whilst tighter WCET bounds are desirable, if a WCET bound cannot by a particular method due to the computational or time resources available to the tester being insufficient, the technique is useless. For this reason there needs to be an acceptance of the trade off between the desired tightness of the bound and the amount of resources available to compute that bound.

To maximise the usage of resources available, combining WCET estimation techniques may be desirable. Frequently a program will have bottlenecks which account for a large proportion of the total execution time of the program despite being a relatively small proportion of the program code. The proposed use of dynamical systems enables a common language that all techniques can be converted to and from. This means that a computationally inexpensive technique can be used to derive estimation of the state and WCET bounds for the majority of the program, with a more accurate but computationally expensive technique used for any bottlenecks.

For example, a monitoring system may spend much of its time in executing a small amount of code that collects measurements from various sensors, but has many ways in which it could be required to process that data. Given the size of the program, it is infeasible to perform model checking on the entire program. Instead, another technique which scales better, such as abstract interpretation could be used to find the initial conditions for the measuring code. Then, the abstract start states could be converted to a form appropriate for model checking, and then model checking used to find a tight bound on the measuring code. Finally, to complete the analysis the end states found by model checking could be converted to abstract states and used to resume abstract interpretation of the program after the model checking analysis of the measuring code has completed.

Initial comparison of the techniques examined in this paper reveals that abstract interpretation and symbolic model checking are perhaps more similar than current discussion would suggest. Given that both techniques operate in an abstracted state space, the major difference between the two techniques is how the abstracted state space comes into existence. In symbolic model checking, the abstracted state space is defined before analysis is started, whereas in abstract interpretation it is lazily computed depending on the abstraction rules available. The approach used in symbolic model checking has the advantage that it is possible to specify the level of complexity of the abstract space in advance. Lazy construction as in abstract interpretation has the advantage that the specification is simpler to construct and has the potential for simplification to happen on an "as necessary" basis, potentially increasing accuracy.

5. Conclusion and Further Work

This paper looked at addressing the disconnect between the various WCET analysis techniques by using recent advances in mathematics to provide a definition of the problem space in which all WCET techniques must operate. This was accomplished by defining the WCET problem as a dynamical system. A consequence of this is that as all WCET techniques can be expressed in terms of this dynamical system, it is possible to directly compare different techniques. An initial comparison suggests that current state of the art static analysis techniques have more in common than present literature would suggest.

Further work needs to be carried out on analysing the WCET problem from a pure dynamical systems point of view. Given that dynamical systems are used in many fields, there is a large number of results which may be applicable. One promising area in this regard is the application of chaos theory, as modern computer systems would appear to meet the requirements for considering a chaos theory perspective: that they are indeed sensitive to their initial conditions.

References

[1] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Inf. Comput.*, 98(2):142–170, 1992.

[2] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, California, 1977. ACM Press, New York, NY.

[3] E. A. Emerson and E. M. Clarke. Characterizing correctness properties of parallel programs using fixpoints. In *Proceedings of the 7th Colloquium on Automata, Languages and Programming*, pages 169–181, London, UK, 1980. Springer-Verlag.

[4] M. Giunti and C. Mazzola. Dynamical systems on monoids: Toward a general theory of deterministic systems and motion. 2010.

[5] S. Lim, Y. H. Bae, G. T. Jang, B. Rhee, S. L. Min, C. Y. Park, H. Shin, K. Park, S. Moon, and C. S. Kim. An accurate worst case timing analysis for risc processors. *IEEE Trans. Softw. Eng.*, 21(7):593–604, 1995.

[6] A. Metzner. Why model checking can improve WCET analysis. In *Computer Aided Verification*, volume 3114 of *Lecture Notes in Computer Science*, pages 334–357. Springer Berlin / Heidelberg, 2004.

[7] R. Wilhelm. Why AI + ILP is good for WCET, but MC is not, nor ILP alone. In B. Steffen and G. Levi, editors, *VMCAI*, volume 2937 of *Lecture Notes in Computer Science*, pages 309–322. Springer, 2004.