Exposé

Richard Stewing[*]

29 Jun 2019

1 Introduction

The main concern in the area of Real Time Systems is to decide whether or not a given System is scheduable. In order to decide this problem, it is necessary to know the Worst Case Execution Time (WCET) of each job in the system. The computation of the WCET in general is undecidable, as it is not even possible to decide termination of a given problem. This is famously known as the Halting Problem. Since the computation of the WCET is a requirement to decide schedulability, many techniques have been employed. This includes Program Path Analysis and Static Analysis such as Cache or Value Analysis. Most of these approaches lack in precision and therefore overestimate the WCET. Examples of such an overestimation is the Cache Analysis which lead to the Industry-best-practice to deactivate the cache in order to have a higher degree of predictability. Another industry best practice is to simply test the program and a buffer to the measurement in order to have a safe upper bound for the WCET. The insufficiency for critical systems of this approach is apparent to any conscious person.

We want to propose an additional approach, building on the large amount of work on dependent types. The conceptional approach is to tie an abstract execution time to each value in the computation. An important distinction to make at this point is that the whole power of dependent types are not used. We are only going to consider execution time and size descriptions for data as natural numbers. The goal is to practically evaluate this approach as the means to probably approximate the WCET and further describe possibilities to use the information encoded as types in the program at run-time to further improve scheduability.

2 An Introductory Example

For better understanding, we are now gonna present a first simplistic approach for describing the WCET in the types of values. As described above, the basic idea is to attach a simple value to the types value. Consider this example in the Idris Programming Language [1]:

```
data In : (a : Type) -> (t : Nat) -> Type where
  InC : (x : a) -> In a t
```

[*]richard.stewing@udo.edu

[1]`https://www.idris-lang.org/`

For anyone not familiar with Generic Algebraic Data Types, the here declared type constructor *In* maps each type *a* and execution time *t* to a new type. A value of this type is constructed using the value constructor *InC*.

This new data type can be used to create value that have there execution time directly attached to them in the type.

```
zero : In Nat 0
zero = InC 0
```

It should be noted that this is equally correct as

```
zero : In Nat 10
zero = InC 0
```

This shows that the correctness of the execution time of such base values can not be verified with this approach. A couple of simplistic functions could look like this:

```
add : In Nat t -> In Nat t' -> In Nat (t+t'+1)
add (InC x) (InC y) = InC (x+y)

mult : In Nat t -> In Nat t' -> In Nat (t+t'+1)
mult (InC x) (InC y) = InC (x*y)
```

As a first take this might look fine, assuming the correctness of the given execution time. This can even be used to calculate the execution time of a combined function. [2]

```
multAndAdd : In Nat t -> In Nat t' -> In Nat t'' -> In Nat (t+t'+t''+2)
multAndAdd {t} {t'} {t''} x y z = rewrite p t t' t'' in add (mult x y) z
```

The proof only considers the execution time and the two calculated execution times, one from the call $add(mult(x)(y))$ and the one expected by the type of *multAndAdd*, are indeed the same.

This small example shows that it is indeed possible two tie an execution time to a value. The correctness and feasibility of this approach still needs to be evaluated.

[2]The proof term p is omitted for brevity.