15 April 2021

Re: SUPE-D-20-01593 "Making Legacy Fortran Code Type Safe through Automated Program Transformation"

Dear Editor,

Below are the reviewers' comments with my responses.

Kind regards,

Wim Vanderbauwhede

Reviewer #1:

The topic chosen is very interesting.

The related work could be more elaborate.

In the contribution the author is referring to ordinary Fortran complier, which version of the compiler the author refers to and whether it has been tested on various versions of Fortran compilers.

> I have added the compilers used and their versions in §3

At the end of section 4.3 a summary of the entire section can be given in a couple of lines.

> The final three lines of §4.3 are in fact the summary

The complexity of the algorithms discussed for converting the given program to type safe can be discussed.

> I have added the complexity of all algorithms

The overhead incurred to convert a Fortran 77 to Fortran 90 and to ensure that it is type safe needs to be discusses.

> We have added a discussion of this with the algorithms for run-time checking. Essentially, there is no performance overhead after conversion from Fortran 77 to Fortran 90; the run-time type checks introduce the overhead of an if-condition to be checked, but in practice this is negligible.

A suggestion to have an optimized compiler which can do the above mentioned job with backward compatibility.

> I am not quite sure what the reviewer means here by "backward compatibility". As explained in the paper, we start from Fortran 77 code but need to generate Fortran 90 code to guarantee the type safety. It is not possible to have complete type-safety in Fortran 77.

I would like to accept this paper with minor revisions like analyzing the complexity of the algorithm and the overhead required for conclusion.

Reviewer #4

I have read through this paper and I find it very interesting as the majority of legacy code was in FORTRAN-77 and COBOL.

Although the paper is very interesting, I think it needs to be shortened as it is too long.

> I have moved most of the formal notation to Appendices, this has made the paper both shorter and more readable.

Also, the list of references does not seem to be complete nor adequate.

> I have added six more references, in particular to applications used for verification.

I recommend accepting the paper, but after submitting a carefully revised version of it.

Reviewer #5

Good morning dear Author,

first of all congratulations on the excellent work submitted to the analysis.

In my analysis, I noted the following points below which made me decline to approve the publication of your work:

1) The text made me understand that it is a guide for migration from Fortran77 to Fortran90;

> I am sorry that the reviewer misunderstood: the paper is not a guide for migration from Fortran77 to Fortran90. As stated in the abstract, the paper presents an analysis of the type safety of FORTRAN 77 and the novel program transformation and type checking algorithms required to convert FORTRAN 77 subroutines and functions into pure, side-effect free subroutines and functions in Fortran 90. The migration to Fortran 90 is required to support the features needed for type safety, but not the goal in itself. I have clarified this in §3.

2) The objectives of the work are not clear enough;

> I have added a sentence to the abstract to make clear that the objective is to reduce errors by increasing type safety

3) The experiments are not conclusive enough to prove the applicability of the study.

> I have added a list of the programs on which the approach was verified in §3

Reviewer #6:

The subject of this paper is highly relevant and the compiler as outlined in it will undoubtedly be very useful in making type safe Fortran routines and functions available for accelerators that enhance HPC systems.

It would have been nice when this objective somehow was reflected in the title of the paper, but with the title being already quite long I can understand that it has been omitted.

There are very few typos in the text and these can be easily hunted down by the author. So, I will refrain from indicating them in this review.

There are also some errors in the content that the author could easily find by himself, but as they may be overlooked and of are importance for the understanding, or ease of reading

we point them out, for sofar as catched by the reviewer.

1. Page 8, line 5 should be: x_out,m = f x_in,k

> OK

2. Page 9, line 36: ...a character as (integer,1). => ...a character as (character,1).

> (integer,1) is correct, we define a character as a one-byte integer

3. Page 10, line 15: Tuple = (τ× ...×τ_i× ...×τ_k) => Tuple =(τ_1× ...×τ_i× ... ×τ_k).

> OK

4. Page 13, line 40: As explained below... => As explained above...

> OK

5. Page 21, Algorithm 1: By placing the notation of the argument lists starting at line 27 before the transformed

version f', f' could be written much more compact.

> I agree but I would prefer to keep the syntax Fortran-like, in particular the order of the arguments would be different if I wrote f'(a",y)

6. Page 21., line 46: I/O direction => INTENT

> OK

7. Page 26, In the storage associations yl(2) is associated with xc(1), xc(2) and not xl(1), xl(2)

yl(3) is associated with xc(3), xc(4) and not zl(1), zl(2)

zl(3) is associated with xc(7), xc(8) and not z1c, which does not appear in the common of the caller.

zl(4) mght be associated with zx in the caller, but zx is not declared.

In all probability the common block in the caller should have read: 'common yc, xc, z1c, z2c'

> Thank you for pointing out this mistake, I have corrected it

By the way, it seems that here a convention is assumed in which variables from the caller are suffixed with a 'c' while the

corresponding variables in the callee are suffixed with an 'l'. Later on this convention seems to be maintained in the

subscripts of expressions, e.g., 'cseq_c' and 'cseq_l'. It would be nice to mention such a convention explicitly and beforehand.

> I have added the convention before the start of §4.6.1

8. Page 26, line 37: 'decl' should be italic.

> OK

9. Page 27, Algorithm 7, - formulas (2) and (3) the subscript 'e' of idx_l,e is not explained.

> I've added an explanation

- the \dot operator is not defined. Although one can assume that 'adding to the set' is meant, it should be stated explicitly.

> I've added the definition

- I can not find the removal of members from cseq_l in the while loop. So, in this form it should run indefinitely.

> I have added the removal of the members form cseq_l and cseq_r

10. In example 14, in function 't3 function f2(f)' the 'external f' statement is missing.

> OK

What makes this paper less appealing is the formal approach in section 4. For instance, the the definition of a pure function as given in subsection 4.2 could be given in a few sentences and could be as unambiguous as the formalism displayed in pages 7 and 8.

> I have removed the notation from 4.2

This superfluous use of set theoretical formalism is even more present in subsections 4.4.1--4.4.8. Virtually nothing in the content of these subsections is used in the following sections, except in a very restricted form in at the end of Algorithm 11.

> I have moved most of the formal notation to Appendices, so that the interested reader can refer to them but they do not break the flow of the paper.

I can think of no excuse to lessen the readability of the paper by inserting these superfluous parts that may discourage readers to continue to the more practical sections of your paper.

My recommendation is therefore to revise it in the sense that only the really necessary notation is used and to introduce it properly before it is applied.

> The reviewer's suggestion has been very helpful. The readability of the paper has much improved by following them.