CerCo: Follow-up of second review (also part of D1.3)

During the second project review the reviewers have made the following main recommendations. We describe here the actions taken in response.

1) "The consortium should concentrate on positioning the project with respect to the state of the art which requires a detailed literature survey. In particular, the consortium should clearly formulate what distinguishes the project from existing work and which results are clear improvements."

In response to the suggestion, we got in touch with some research groups and EU Projects that work in the mainstream of WCET analysis. Particularly useful where the interactions with the forthcoming TACLe COST Action, which we plan to join, and the EU Project PROARTIS. The two events organized during the third period also allowed us to discuss our project goals and objectives with a larger audience.

From the project planning phase, the differences in methodology w.r.t. the state of the art have been clear, as well as the long term promises of the methodology (reduced trusted code base, improved precision of the bounds, parametricity, integration of a larger set of techniques in time analysis). These, however, are all long term goals and, starting with a new methodology from scratch, it was also obvious that in the short term we would not have any improvement with the consolidated techniques, that have evolved for decades.

The interaction with the WCET experts, however, has indeed allowed us to understand a strong points of our methodology that we did not consider. Industry seems to have a strong interest in performing time analysis during early development phases, something we can already provide almost for free.

WCET analysis has traditionally been used in the verification phase of a system, after all components have been built. Since redesigning a software system is very costly, designers usually choose to over-specify the hardware initially and then just verify that it is indeed sufficiently powerful. However, as systems' complexity rises, these initial safety margins can prove to be very expensive. Undertaking lightweight (but less precise) analysis in the early stages of the design process has the potential to drastically reduce total hardware costs. Obviously current WCET technology that depends on the generation of the object code is unsuitable for this task. Embracing the CerCo approach, instead, we can already reason at early stages by axiomatizing the cost of unimplemented components or by reasoning parametrically over those costs. The CerCo plug-in can already combine together actual costs computed on the generated object code and axiomatized costs, and it smoothly supports progressive program refinement.

To understand the current interest in early development analysis, one out of four work packages of the forthcoming TACLe cost action will be entirely devoted to the topic. This will be the work package that members of CerCo are going to join and contribute to.

On the negative side, both the literature survey and direct interaction with fields experts show that the major concern of the WCET community at the moment is represented by the so-called WCET Walls, i.e. showstoppers to static analysis. The first wall is the current evolution of high performance hardware that is hitting the embedded systems market too. Simple and predictable microprocessors and systems are being replaced with multi-cores and, soon, whole systems-on-chip with several computing units, each one with several multi-cores, a non uniform memory space and a network-on-chip. The computations running in parallel on different nodes greatly affect each other execution costs. If static analysis considers programs in isolation, it will always need to assume the worst scenario and, because of the extremely high jitters of these architectures, the bounds obtained will be so large to be practically useless. The second wall is the progressive disappearing of detailed information to build the clock cycle precise hardware models required by static analysis.

A lot of pressure is currently put by researcher on hardware manufacturers to produce alternative hardware that is not oriented at maximizing average case performance, but at being predictable. As

far as we were told, these efforts have failed so far and it may be that in the future no static analysis at all will be any longer useful. CerCo is clearly affected by this issue and does not contribute to the problem. On the other hand, we tried to understand to which extent the CerCo technology could embrace and cope with some of the solutions currently proposed to mitigate the problem, like randomization of hardware components (e.g. caches) to eradicate the dependency of execution cost from execution history. The results in this direction seem very promising: the labelling approach at the heart of CerCo can already deal with both deterministic and probabilistic cost models and embracing the PROARTIS proposal on probabilistic caches could allow CerCo to deal with the currently deployed hardware.

2) "The consortium should as soon as possible start with the verification of the compiler." "The reviewers are worried that the whole verification tasks cannot be successfully completed by the end of the third project period."

The verification of the compiler had already started and an estimation of the time required had been made by multiplying the amount of code to be verified by the ratio time-per-line obtained from CompCert. The worries by the reviewers were well founded because indeed we were not able to complete the formalization for the end of the project. Two facts concurred to the delay: 1) our preliminary certification of a toy compiler (for a While language) that implemented the labelling approach show that the effort required for the intensional properties was minimal compared to the part that deals with a classical forward simulation. That resulted to be completely false when we started the proof for the C compiler. The main difference was due to function calls and function pointers. In the toy compiler, the final result was obtained from two independent proofs: one syntactic preservation of certain invariants, that were a property of the static code and not of the execution; a proof of forward simulation were the set of observables were augmented with labels, posing no additional problems. In presence of function calls, a whole new set of invariants need to be preserved. In particular, we need to grant that a function returns immediately after its calling point, which is a non trivial property in all back-end languages (and that can be falsified by any compiler pass without affecting functional correctness). The property is hard to prove because it is a global property of part of a sub-trace. Function pointers pose additional issues because with function pointers a call can jump anywhere in the code. Therefore, what used to be a syntactic property on the code (and not on executions), now becomes a property on executions too. To cope with both problems at once, we completely revised the global proof sketch introducing a new style of semantics that maps programs to *structured traces*. A structured trace is no longer a plain stream of observables, but a "stream" of observables with additional structure and invariants that capture all the information required to infer a cost model on the object code and backward propagate it to the source code. In particular, after introducing a notion of similarity between structured traces, we proved that all that is needed to complete the proof is: a) a forward simulation result for each pass, where the notion of simulation is based on the new notion of similarity; b) the proof of existence of structured traces in the first place, i.e. the fact that every C program execution has the good structure. We also integrated in the structure the assumption that the run does not go out of stack, which is necessary because we assume a realistic target language with finite memory. This necessity was also pointed out by the reviewers during the meeting, and we were able to cope with it inside the labelling approach. 2) the ratio time-per-line was extraordinarily higher for the proof of correctness of the assembler because of the enormous amounts of instructions and because of the complexity of the operational semantics. CompCert does not implement any assembler.

The new proofs that deal with structured traces took a significant amount of time and we had to revise the definition of structured traces several times. At a certain point it has been clear that we would not have been able to complete the formalization.

As contemplated in the DoW, we started counter-measures. At the time of the DoW, the idea was to concentrate on some passes (the most difficult ones) leaving other unproved. Instead, we chose another strategy. We decided to isolate as much as possible the parts of the proof that either were

novel and peculiar of the labelling approach, or that were interesting because they pointed at some benefits from our proof and design choices. At the end of the project, all CerCo specific parts have been proved and, indeed, at least for the back-end we have provided two generic layers to reduce a proof of forward simulation in the sense of structured traces to a standard proof of simulation (with a few very simple additional syntactic invariants to be checked). The first layer is fully generic because it works on a fully abstract notion of transition system equipped with a structured traces semantics. The second layer works on our generic representation of back-end languages and it applies to any concrete pass that instantiates a generic pass data structure. Thanks to these generic layers, we now know how to apply the labelling approach to an existent formally verified compiler after re-writing all back-end languages and passes using our generic representations.

To summarize, while the compiler has not been fully certified, only functional properties preservation is left to be proved and it has been isolated in such a way that the missing parts of the proof should now be standard. Moreover, the non standard parts have been made as reusable as possible.

3) “The consortium should develop a code extractor for MATITA which allows obtaining executable code from MATITA specifications such that an executable version of the trusted CerCo compiler can be automatically generated. This extraction will be crucial to apply the CerCo approach to practical case studies and to present it to industry.”

We have worked on two code extractors, one targeting the OCaml language and the Hindley-Milner type system and one targeting the flavor of System F_omega that is implemented by extensions of GHC. The first one has been used to extract the CerCo compiler. It shares code with the one of Coq and produces code of the same quality. As discussed in D5.2, the quality of the code extracted from the CerCo compiler, measure using several orthogonal axis, ranges from average to poor. This is partly a consequence of extracting from a very rich type system (we fully exploited the dependently typed discipline) to a very poor one (Hindley-Milner). For that reason we started extraction to System F_omega (GHC), but we have also identified several additional improvements to code extraction to recover code quality. For example, we would like to experiment with the extraction of dependent records to first class OCaml modules to recover typing information, and we would like to exploit polymorphic variants and GADTs during extraction. It seems that the CerCo compiler code represents a very good test bench for studying the improvement of code extraction and in the middle term period we will continue research in that direction.

4) “The consortium should concentrate on further developing and publishing the dependent labelling approach. In particular, the approach should be extended to further compiler optimizations, apart from loop peeling and hoisting, and be applied to more modern processor architectures, eventually aiming at multi-level caches and pipelines.”

Not being able to divert more manpower from the formalization, we hired a new Post-Doc from UNIBO to work on this topic. Because of lack of time, it was not possible to let him work both on new optimizations and modern process architectures. We decided that the latter were more relevant at the moment. The Post-Doc first concentrated on pipelines, but at the end he came up with a fine grained analysis of dependent labelling that classifies cost models according to the possibility of backward propagating them to the source code. In particular, a certain class of cost models, that covers at least simple version of pipelines, can be propagated by exposing at the same time on the source code an abstract representation of the hardware state. For caches the situation is harder, but combining the same technique with probabilistic time analysis it should be possible to analyze on the source code randomized caches like the ones developed in the PROARTIS project. Further research is needed on the topic and we absolutely need a working prototype to test the capabilities of invariant generators and automated provers to reason on these more complex cost models. Nevertheless, this research line is on its way and no major showstopper has been found so far,

confirming our intuition that precise source code time analysis should be possible even in presence of complex hardware architectures. Moreover, all alternative possibilities already discusses in last year response to the reviewers still apply, with a degradation of precision.

5) The reviewers required a revision of some deliverables and a few addenda. The new versions and addenda were submitted to the EU as expected.