Table of contents

PART B

1 Concept and objectives, progress beyond state-of-the-art, S/T methodology and work plan

1.1 Concept and project objective(s)

The project aims at the construction of a formally verified complexity preserving compiler from a large subset of C to some typical micro-controller assembly, of the kind traditionally used in embedded systems.

The work comprises the definition of cost models for the input and target languages, and the machine-checked proof of preservation of complexity (concrete, not asymptotic) along compilation. In particular, the compiler will also return tight and certified cost annotations for the source program (expressed in terms of clock-cycles) for program slices with O(1) complexity. It is then up to the user, possibly assisted by automatic tools, to use this (trusted) information to state and to prove precise complexity assertions on the program. To this aim, he can exploit the tools provided by the ongoing research on techniques for invariants generation and cost inference for imperative programs, with two additional benefits: the guarantee that the inferred intensional properties will carry over to assembly code; and the adoption of a cost model that is absolutely precise, being induced from the generated assembly language.

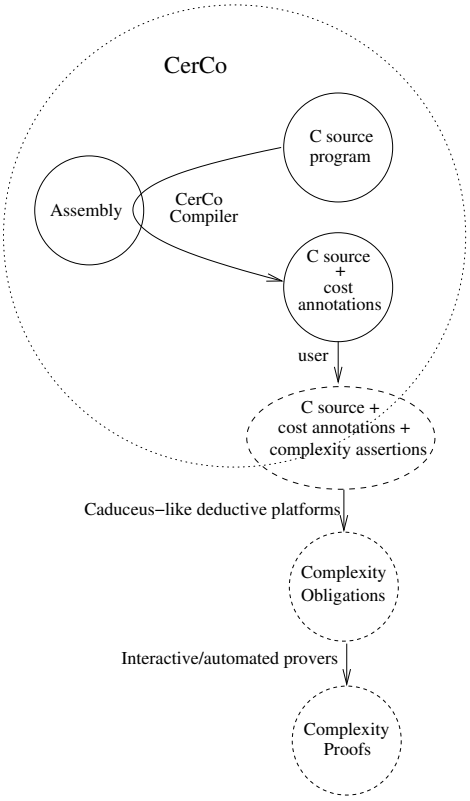


Figure 1: Data Flow

The main focus of the current project is on the certified, cost annotating compiler. As for the way cost annotations are used, we shall develop a proof-of-concept prototype, by interfacing to already existing tools, to show how this information can be exploited in a useful manner. We will develop abstract interpretation techniques to infer automatically complexity bounds and, in particular, we will test these techniques on the C code generated by the compilers of synchronous languages, such as Lustre or Esterel.

The major breakthrough of the project is the possibility to give a tight performance estimate for the executable code (for computing platforms such as microcontrollers, not relying on operating systems), a task that is currently regarded as highly visionary in the compiler community. The essential paradigm shift consists in creating the (certified) infrastructure allowing to draw conclusions on the target code, while comfortably reasoning on the source. The compiler will be open source, and all proofs will be public domain.

1.2 Progress beyond state of the art

Automatic verification of properties of software components has reached a level of maturity allowing complete correctness proofs of entire compilers; that is of the semantic equivalence between the generated assembly code and its source program. For instance, in the framework of the Verifix Project a compiler from a subset of Common Lisp to Transputer code was formally checked in PVS (see [Dold and Vialard]). [Strecker] and [Klein and Nipkow] certified bytecode compilers from a subset of Java to a subset of the Java Virtual Machine in Isabelle. In the same system, [Leinenbach et al.] formally verified a compiler from a subset of C to a DLX assembly code. [Chlipala] recently wrote in Coq a certified compiler from the simply-typed lambda calculus to assembly language, making an extensive use of dependent types. Perhaps, the most advanced project is Compcert, headed by Leroy, based on the use of Coq both for programming the compiler and proving its correctness. In particular, both the back-end ([Leroy]) and the front-end ([Leroy and Tristan]) of an optimising compiler from a subset of C to PowerPC assembly have been certified in this way.

However, very little is known about the preservation of intensional properties of programs, and in particular about their (concrete) complexity. The theoretical study of the complexity impact of program transformations between different computational models has so far been confined to very foundational devices. Here we propose to address a concrete case of compilation from a typical high-level language to assembly. It is worth remarking that it is unlikely to have constant-time transformations between foundational models: for instance coding a multitape Turing machine into a single tape one could introduce a polynomial slow-down. Thus, complexity is architecture dependent, and the claim that you may pass from one language to another, preserving the performance of your algorithms, must be taken with the due caution. In particular, as surprising as it may be, very little is known about the complexity behaviour of a compiled code with respect to its source; as a matter of fact, most industries producing robots or devices with strong temporal constraints (such as, e.g. photoelectric safety barriers) still program such devices in assembly.

The tacit assumption that the complexity of algorithms is preserved along compilation, while plausible under the suitable assumptions, is not supported by any practical or theoretical study. For instance, a single register is usually used to point to activation records, implicitly delimiting their number; you may take more registers to this purpose, but unless you fix a priori their number (hence fixing the size of the stack), you cannot expect to access data in activation records in constant time. In particular, the memory model assumed by Leroy assumes an infinite memory, where allocation requests always succeed, that clearly conflicts with the reality of embedded software, where one has to work within precise (often relatively small) memory bounds. If working in restricted space on one side allows us to properly weight memory access as a unit cost, on the other side it introduces a subtle interplay between space complexity, time complexity and correctness that will be one of the crucial issues of the project.

Even admitting (as we hope to prove) that in a confined universe we may actually preserve complexity, the main interest of the project is in producing a (certified) computational cost (in terms of clock cycles) for all instruction slices of the source program with O(1) complexity, thus providing precise values for all constants appearing in the cost function for the source. This opens the possibility of computing time constraints for executable code by reasoning directly on the high level input language. In particular, we are not aiming to help analyse the complexity (or termination) of programs (that has to be guessed by the user, as he guesses invariants in axiomatic semantics), but we shall build the necessary infrastructure to reflect a high-level, abstract complexity analysis of the source on a concrete instantiation of its target code.

Such instantiation depends on the target architecture: for instance some microcontrollers lack the multiplication instruction as a primitive operation, preventing to count such an operation with a fixed cost. Moreover, if we are interested in a really tight complexity measure, we cannot expect to have a uniform cost for input instructions since, due to register allocation and optimisations, their actual cost depends on their surrounding context. In other

words, we have to face the non compositional nature (in terms of the source structure) of most compiler techniques and optimizations.

The Compcert project represents the current baseline for any future work on compiler certification, comprising the one we plan to do. We will improve on Compcert in two directions: by assuming a formal model where resources (memory) are constrained; and by preserving complexity of O(1) operations, also tracing the way they are mapped to assembly to reflect actual computational costs on the source code. Both improvements greatly increase the exploitation potentials, in particular in the domain of embedded systems and real time computation.

1.2.1 The CerCo approach

The complexity of a program only depends on its control-flow structure, and in particular on its cycles (procedures calls are just a special case of cycles). Proving that a compiler preserves complexity amounts to proving that it preserves (up to local modifications, like loop unrolling, in-line expansion, etc.) the control-flow structure of the source[1] and, less trivially, that all other instructions are compiled into assembly code whose execution takes a bounded number of clock-cycles (i.e. with O(1) complexity). The interest of the project lies in the possibility to compute these costs directly on the target code then refer them back to the source program, allowing the possibility to make precise and trusted temporal assertions about execution from reasoning on the source code.

As already mentioned, the main problem in the backward translation of costs from target code to source code is the fact that, apart from the overall control flow structure, all remaining structure of the input is usually irremediably lost during compilation: optimizations can move instructions around, change the order of jumps, and in general perform operations that are far from compositional w.r.t. the high level syntactic structure of the input program. So there is no hope to compute costs on an instruction-by-instruction basis of the source language, since the actual cost of the executable is not compositional in these figures. We have to find another, eventually coarser, level of granularity where the source can be sensibly annotated by target costs.

We regard a C program as a collection of mutually defined procedures. The flow inside each procedure is determined by branching instructions like if-then-else; "while" loops can be regarded as a special kind of tail recursive procedures. The resulting flow can thus be represented as a directed acyclic graph (DAG). We call a path of the directed acyclic graph an *execution path*.

```
void quicksort(int t[], int l, int r) {
  if (l < r) {
    int v = t[l];
    int m = l;
    int i = l + 1;
    while (i <= r) {
      if (t[i] < v) { m++; swap(t, i, m); }
      i++;
    }
    swap(t, l, m);
    quicksort(t, l, m - 1);
    quicksort(t, m + 1, r);
  }
}
```

Figure 2: Quicksort

As a simple example, consider the quicksort program of Fig. 2. This algorithm performs in-place sorting of input array t whose bounds are l and r; initially l is expected to be zero, while r is the length of the array minus one.

[1] This requires, in turn, the preservation of semantics: the right conditions must be tested, and procedures must be called with the correct parameters.

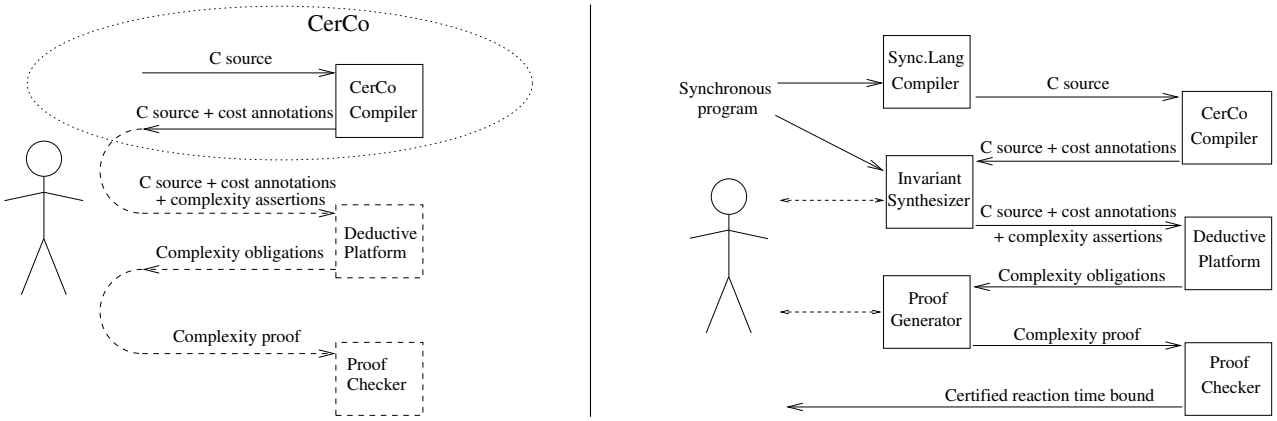


Figure 3: Interaction and automation diagrams

The outermost conditional terminates when the bounds of the array are illegal. (Sorting an empty array will end the recursive behaviour of the algorithm.) The variable v is the so called pivot: a selected element of the array that will be compared with all other elements. Bigger elements will be moved (by the swap function) to the end of the array (the upper part), while smaller elements are placed at the beginning of the array (the lower part). Then the pivot is placed between the lower and the upper part of the array, in position m, its position in the resulting sorted array; all elements before the pivot are smaller and all elements following it are bigger. The algorithm completes the sorting with recursive calls on the lower and on the upper parts of the array.

In the body of the `quick_sort` procedure there are only two execution paths, corresponding to the two cases $l < r$ and $l \geq r$. The latter is a trivial path, immediately leading to termination. The former leads to the while loop (that is regarded as a procedure call), the call to swap, and the two recursive calls. Similarly, the body of the while loop is composed by two paths, corresponding to the two conditions $i \leq r$ and $i > r$.

All operations performed along any of these paths takes some constant time c. The complexity magnitude of the program only depends on the loops and recursive calls met along its execution paths, but not on their associated constants. On the other hand, if we want to give tight performance bounds to the execution time, we have to compute the real constants on the executable.

The compiler must be able to return a set of pairs (p_i, c_i), where each p_i is an execution path, and c_i is its actual cost[2]. It is then up to the user to guess (and to prove, assisted by interactive tools) the complexity of the program: the compiler only provides the infrastructure required to map a complexity analysis on the source into a faithful analog on the target. This approach looks compatible with most local optimizations. Moreover, since we work on a cycle-by-cycle (procedure-by-procedure) basis, the approach should scale up well.

1.2.2 User interaction flow

The left part of Fig. 3 shows the interaction diagram for the final user of the system (the right part is a planned case study for a possible automation of the process and will be discussed later). The interaction is done in several steps. We illustrate them using the quicksort program above.

1. The user writes her code in C (see Fig. 2) and compiles it with our CerCo (Certified Complexity) compiler.
2. The CerCo compiler outputs both the object code and an annotated copy of the C source (Fig. 4). Each loop and function body is annotated with the cost of one iteration, along all its possible execution paths. The cost is expressed as a function of the state of the program, which comprises the value of every variable. (In the example, we use a textual annotation for simplicity, but we expect to produce a more structured output.)

[2] A more flexible result would consist in returning pairs (p_i, a_i) where a_i is the sequence of assembly instructions corresponding to p_i; this would allow to take space into consideration, as well as time.

```
void quick_rec(int t[], int l, int r) {
  /* Cost annotation for quick_rec body (1 cycle only)
     @ if (l<r) time += 21;
     @ if (l>=r) time += 6; */
  if (l < r) {
    int v = t[l];
    int m = l;
    int i = l + 1;
    while (i <= r) {
      /* Cost annotation for while (1 cycle only)
         @ if (t[i] < v) time += 4; else time += 5; */
      if (t[i] < v) { m++; swap(t, i, m); }
      i++;
    }
    swap(t, l, m);
    quick_rec(t, l, m - 1);
    quick_rec(t, m + 1, r);
  }
}
```

Figure 4: Cost annotated C code (generated by the compiler)

The leftmost column of Fig. 5 shows the original source code. Colours are used to relate source code statements with their respective (human readable) assembly instructions, reported in the central column. That assembly was produced by gcc[3] with a moderate level of optimizations for an Intel 386 family microprocessor. We used l, r, t, v and m to mention locations (registers and/or stack frame temporaries) in which the corresponding C variables are placed, while $r1, \ldots, r9$ are other register or temporaries that have no direct mapping to C. The calling convention puts the first three parameters in $r1$, $r2$ and $r3$, and it is up to the callee to eventually store them in local temporaries. Assignment is denoted with `<-`, addition and multiplication with `+` and `*`; the jump instruction is followed by the target address, and when the jump is conditional a C like expression follows (but its evaluation is performed early by the `cmp` instruction, that sets a CPU flag recording the result of the comparison). The only tricky expression is "`*(r8 + r7 * 4)`", that exploits an advanced addressing mechanism corresponding to array indexing (4 is the size of an array cell in bytes, $r7$ is the index and $r8$ is the address at which the array starts). It amounts to the C statement "`t[l]`" that computes the pivot.

The rightmost column shows two possible execution paths, with a precise estimation of their cost (here 6 and 21 CPU cycles plus the cost of function calls) and the algebraic conditions characterizing these paths.

More precisely

- The CerCo compiler avoids intra-procedural optimisations and loop optimisations that may change the number of iterations performed in a non trivial way.
- Some intra-procedural or loop optimisations (like the `while` to `repeat` pre-hoisting optimisation applied by gcc in Fig. 5) can be allowed, provided that the compiler records them precisely.
- Once the assembly code is produced, the assembly-level control flow graph is analysed in order to compute the cost of execution paths. Fig. 5 shows two of them in the rightmost column; the analysis of the `while` loop has been omitted, but is similar.

3. The user computes (by hand or semi-automatically) the complexity invariants of each cycle and (recursive) function, and he adds them to the C code as special comments[4] (Fig. 6). The quicksort complexity invariant

[3] GNU compiler collection, version 4.2
[4] Again, more interactive forms of annotations can be considered.

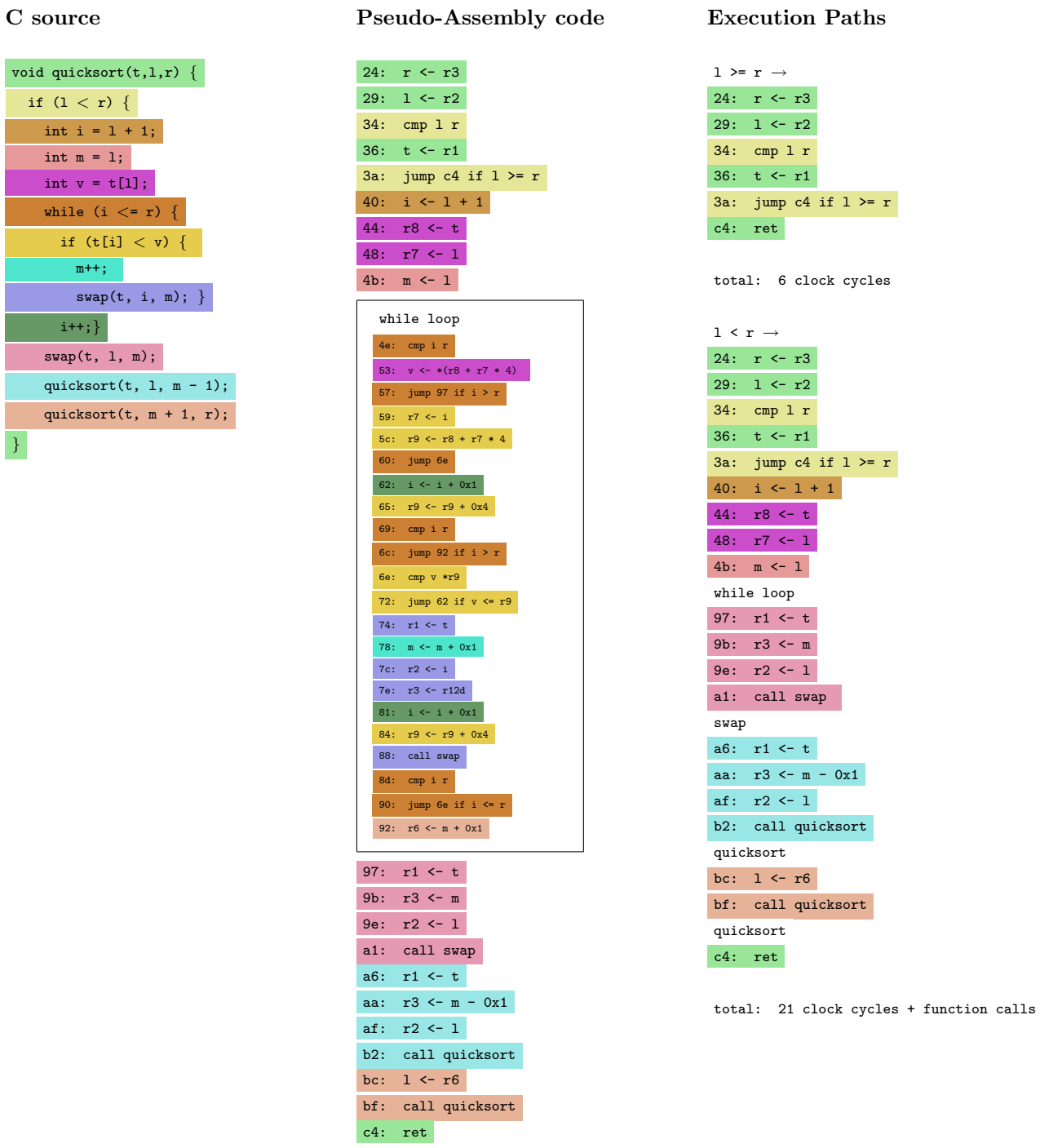


Figure 5: Automatic inference of cost annotations from assembly code

```
/* User provided invariant for the quicksort function.
   @ ensures (time <= \old(time) + max(0,((1+r)-l))*(max(0,(r-l))*5+21) + 6) */
void quick_rec(int t[], int l, int r) {
  /* Cost annotation for quick_rec body (1 cycle only)
     @ if (l<r) time += 21;
     @ if (l>=r) time += 6; */
  if (l < r) {
    int v = t[l];
    int m = l;
    int i = l + 1;
    /* @ label L
       User provided invariant for the while loop. To perform i iterations
       we need at most (i-l)*5 clock cycles.
       @ invariant time <= \at(time,L) + (i-l) * 5 &&
       Additional invariants of the loop
       @ l <= m && l+1 <= i && m <= r && i <= r+1 && m < i && l < r */
    while (i <= r) {
      /* Cost annotation for while (1 cycle only)
         @ if (t[i] < v) time += 4; else time += 5; */
      if (t[i] < v) { m++; swap(t, i, m); }
      i++;
    }
    swap(t, l, m);
    quick_rec(t, l, m - 1);
    quick_rec(t, m + 1, r);
  }
}
```

Figure 6: Invariants annotated C code. The invariants are user provided.

First complexity obligation: case where $l < r$. The number of clock cycles spent in one iteration of the recursive function must be greater or equal than the sum of the number of clock cycles spent in the function body, in the `while` loop and in the two recursive calls:

$$\begin{array}{l}\forall i, l, m, r.\ l < r \wedge l + 1 \leq r \wedge r < i \leq r + 1 \wedge m \leq r \wedge l + 1 \leq i \wedge l \leq m \Rightarrow \\ (i - l) * 5 + 1 + \\ max(0, 1 + r - (m + 1)) * (max(0, r - (m + 1)) * 5 + 21) + 6 + \\ max(0, 1 + (m - 1) - l) * (max(0, m - 1 - l) * 5 + 21) + \\ 6 \\ \leq max(0, 1 + r - l) * (max(0, r - l) * 5 + 21) + 6\end{array}$$

Second complexity obligation: case $l \leq r$. The number of clock cycles spent in one iteration must be greater than the number of clock cycles required when the `if` statement fails.

$$l \leq r \Rightarrow 6 \leq max(0, 1 + r - l) * (max(0, r - l) * 5 + 21) + 6$$

A few others complexity obligations are automatically generated but, being trivial, are automatically proved by the system.

Figure 7: Complexity obligations (automatically generated). The user should prove every complexity obligation.

states the maximum number of clock cycles required by its execution on an array delimited by l and r. Since the procedure can be called with wrong indices ($r < l$) the formula has to take into account that the $r-l$ could be negative using the max function to raise that difference to zero when needed. The literature suggests that this quicksort implementation (where the pivot v is chosen deterministically) has a quadratic complexity in the worst case. Cleaning up the formula from multiplicative and additive constants one obtains the expected asymptotic complexity $(r-l)^2$.

The coefficients are those returned by the cost-annotating compiler. Similarly, the user has to give a complexity invariant for the inner while loop.

4. The user and compiler annotated C code is fed into an already existing tool (in this example, Caduceus, [Filliâtre and Marché]) that produces one complexity obligation for each execution path (Fig. 7).

5. The user should prove all complexity obligations. The proofs are the certificate that the user provided complexity invariant is correct. In many cases, the obligations can be proved automatically using a general purpose automatic theorem prover or an ad-hoc procedure. For instance, to prove the complexity obligations of Fig. 7, we must show that a system of inequations holds, which may be done automatically. When an automatic proof is not possible, the user can resort to an interactive proof.

The right part of Fig. 3 describes a planned case study for the automation of the complexity proof. We start with a synchronous program which is compiled to C code. The CerCo compiler then produces suitable cost annotations which are used by an invariant synthesizer to build complexity assertions on the C code. The synthesizer can take advantage of the high level control flow information contained in the source synchronous program. The deductive platform (Caduceus) generates complexity obligations which are passed to an ad-hoc proof generator to produce a machine-checked proof from which we can extract a certified bound on the reaction time of the original synchronous program. The proof generator can also take advantage of the high level information coming from the original source program, and user interaction can be used to drive the generator in critical cases.

1.2.3 Certification: tools and techniques

In order to trust the process described in the previous section, we need to trust the CerCo compiler. I.e. we need to fulfil the following requirements:

1. the compiled assembly program respects the semantics of the C program

2. we need to know that the number of iterations performed by the C and assembly programs are the same, i.e. to prove that the compiler preserves the complexity

3. we need to prove that the cost annotations generated by the compiler really correspond to the number of clock cycles spent by the hardware

For this reason, we plan to[5]:

1. develop an untrusted CerCo compiler prototype in high level programming language;

2. provide an executable formal specification of the target microprocessor;

3. provide an executable formal specification of the C language;

4. develop an executable version of the CerCo compiler in a language suitable to formal correctness proofs;

5. give a machine checkable proof that the latter implementation satisfies all the requirements mentioned above.

[5] See next section for a more articulated description of the methodology

The untrusted compiler will be written in the OCaml programming language, developed and distributed by INRIA, France's national research institute for computer science. OCaml is a general-purpose programming language, especially suited for symbolic manipulation of tree-like data structures, of the kind typically used during compilation. It is a simple and efficient language, designed for program safety and reliability and particularly suited for rapid prototyping.

For the certification of the compiler we plan to use the Matita Interactive Theorem Prover, developed at the Computer Science Department of the University of Bologna. Matita is based on the Calculus of Inductive Constructions, the same foundational paradigm as INRIA's Coq system, and it is partially compatible with it. It adopts a tactic based editing mode. Proof objects (XML-encoded) are produced for storage and exchange. Its graphical interface, inspired by CtCoq and Proof General, supports high quality bidimensional rendering of proofs and formulae, transformed on-the-fly to MathML markup. In spite of its young age it has already been used for complex formalizations, including non trivial results in Number Theory and problems from the Poplmark challenge [POPLmark]. An executable specification for all models of Freescale 8bit ALUs (Families HC05/HC08/RS08/HCS08) and memories (RAM, ROM, Flash) has also already been formalised in Matita.

For the management of cost annotations and proof obligation synthesis we plan to interface with the Caduceus verification tool for C programs, developed by the Computer Science Laboratory of the University of Paris sud. Caduceus is built on top of Why, a general-purpose verification condition generator, exploiting Dijkstra's weakest precondition calculus. The Why tool allows the declaration of logical models (types, functions, predicates and axioms) that can be used in programs and annotations; moreover, it can be interfaced to a wide set of existing provers for verification of the resulting conditions. In particular, we will rely on Alt-Ergo, which includes a decision procedure for linear arithmetic.

We plan to support almost every ANSI C construct (functions, pointers, arrays and structures) and data-types (integers of various sizes) except function pointers, explicit jumps (goto) and pointer aliasing or casting. These features do not seem to pose major additional challenges to the technology we plan to develop, but could be time expensive to implement, formalize and prove correct. Moreover, they are not currently supported by Caduceus, posing additional problems for the development of the proof-of-concept prototype of the whole application. We could also support floating point numbers, but the kind of micro-controller we are targeting (8 and 16 bits processors) do not usually provide instructions to efficiently process them. Moreover floating point numbers are seldom used in embedded software for that very reason, making them a feature of ANSI C of limited interest in our scenario.

We stress that the proposed approach handles costa annotations for C programs as a special case of standard annotations for imperative programs whose management we plan to automatize with tools such as Caduceus. As explained above, the choice of the tool does have an impact on the fragment of ANSI C constructs we will handle, and future advances in this domain could enlarge the fragment of ANSI C under consideration. On the other hand, tools like Caduceus pose no limitation on the invariants, which can be freely described in a computational and very expressive logic. Hence, every technique to automatically infer invariants and cost annotations can be exploited (and often automatized) in Caduceus.

1.3 S/T methodology and associated work plan

1.3.1 Overall strategy and general description

The objectives detailed in Section 1.2 will be pursued by the CerCo Consortium through the actuation of a workplan described in this section, consisting of a total of 6 Work-Packages (WPs) spanning a temporal frame of 36 months. Several Consortium partners participate in each WP, according to their specific expertise, know-how and business interests.

WPs are independent, yet tightly related. Their execution, to be successful, calls for a significant amount of interaction, information exchange and coordination. Each WP is led by one of the Consortium partners, whose role is to coordinate the work inside the WP and interfacing and communicating with the other WPs. Each WP is further broken down into Tasks, each of them is responsible for a specific portion of the work.

Work packages In this section, we provide a brief overview of WP activities, outlining the required inputs and expected outputs, as well as the foreseen interactions among WPs. The description of the work to be carried out within each WP, split over the different Tasks, and including information on the specific involvement and contribution of each Consortium partner along the 36-months lifetime of the project will be presented in a later section (Tables 1.3.5). A Pert showing the dependencies between the tasks is provided in Fig. 8.

WP1: Project Management

WP2: Compiler Prototype (untrusted)

- Task T2.1 Architectural design
- Task T2.2 Intermediate languages and data structures
- Task T2.3 Implementation
- Task T2.4 Integration, validation and testing

WP3: Verified Compiler - front end

- Task T3.1 Formal semantics of C
- Task T3.2 Functional encoding in the Calculus of Inductive Construction
- Task T3.3 Formal semantics of intermediate languages
- Task T3.4 Correctness proofs

WP4: Verified Compiler - back end

- Task T4.1 Formal semantics of machine code
- Task T4.2 Functional encoding in the Calculus of Inductive Construction
- Task T4.3 Formal semantics of intermediate languages
- Task T4.4 Correctness proofs

WP5: Interfaces and interactive components

- Task T5.1 Management of complexity assertions
- Task T5.2 Automation of complexity proofs
- Task T5.3 Case studies

WP6: Dissemination and exploitation

- Task T6.1 User validation and exploitability
- Task T6.2 Contribution to portfolio and concertation activities at FET-Open level

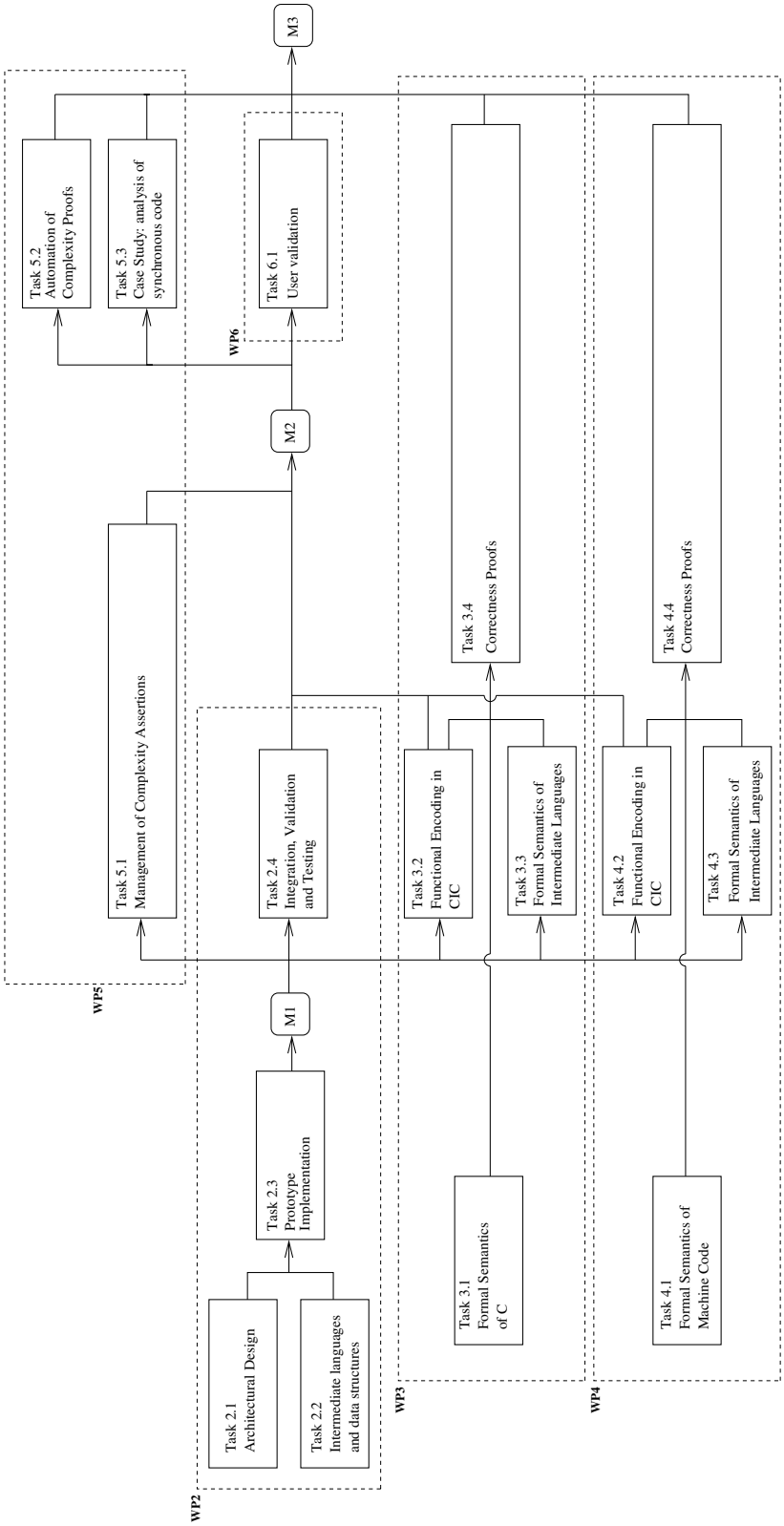


Figure 8: Pert Diagram

The work is organized in four main technical Workpackages WP2-WP5, plus two WPs devoted to Project Management (WP1) and Dissemination and exploitation activities (WP6).

Work Package WP1 collects the main management activities. The aim of WP1 is to efficiently support technical and administrative activities and to facilitate good communication and co-operation among the partners, in order to ensure the successful fulfilment of all project objectives and the achievement of deliverables. As better explained in Section 2.1 Implementation, the Project Coordinator and the Project Manager will be responsible for the project management and coordination activities which include: supervision of project progresses and events, coordination between the work packages, consistency check of the contributions from project partners, check content and timing of project deliverables, monitoring of the project financial situation. The Project Coordinator will be supported by WP leaders, which will coordinate the working activities of WPs and will organise WP deliverables.

Work Package WP2 aims at building a functional prototype of the cost annotating compiler (milestone M1). The compiler will be *untrusted*, meaning that no proof will be given that the machine code and the cost annotations returned by the compiler are correct; it will be written in a high-level, comfortable programming language particularly tailored to compiler construction, such as OCaml. This untrusted prototype compiler is the first milestone of the project, since it will embody the main architectural design of the final compiler, the format of cost annotations, and the way of computing them. At the same time, it will allow to start experimenting with the management of cost annotations, the declaration of complexity assertions, the generation of complexity obligations and their interactive solution (tasks covered by WP5).

All partners will contribute expertise to Task T2.1 where major design choices will be taken and where the format of cost annotations will be thoroughly discussed. A selection of optimizations compatible with complexity preservation will also be identified in that task.

UPD will be in charge of the actual implementation of the untrusted compiler (Task T2.3). UPD and UEDIN will preliminary agree in Task T2.2 on the intermediate languages and data structures for the front-end, in order to make them easily encodable in CIC (Task T3.3). Similarly UPD and UNIBO will reach an agreement on languages and data structures for the back-end.

Task T2.4 will be performed by UPD, driven by the feedback collected from UEDIN and UNIBO in Tasks T3.2, T3.3, T4.2 and T4.3.

Work packages WP3 and WP4 are entirely devoted to the correctness proof of the compiler. Due to the dimension of the work, this has been split in two major WPs, reflecting the natural decomposition of a compiler into a front-end (WP3) and a back-end (WP4). The first step of the correctness proof consists in rewriting the compiler into the internal language of a proof assistant - the Matita Interactive Theorem Prover, in our case - i.e. in a simplified language more suited than OCaml to formal reasoning (Tasks T3.2 and T4.2). These two tasks will be respectively performed by UEDIN and UNIBO with the collaboration of UPD who wrote the untrusted compiler and is responsible, in Task T2.4, of keeping it in sync with the certified one. The complexity of the tasks relies in the fact that the language of Matita (the Calculus of Inductive Constructions) although quite expressive from a logical point of view, is very rudimentary from a programming perspective.

The formalization of the source and target language semantics in Tasks T3.1 (UEDIN) and T4.1 (UNIBO) will be performed during the fist year, in parallel with Task T2.3 (UPD), anticipating the certification of the compiler.

At the end of the second year we plan to have a full prototype of the system (M2), already written in a language suited to be formally checked for correctness, but still lacking (complete) proofs (part of the formalization will be already completed during the second year, as part of Tasks T3.3, T3.4, T4.3 and T4.4). Having such a prototype will allow to start, during the third year, an intensive dissemination activity and the user validation phase.

Starting from month 18, Tasks T3.4 and T4.4 are devoted to the correctness proof and will be respectively performed by UEDIN and UNIBO with frequent interactions with UPD to clarify the program logic and related invariants. The outcome of the two tasks will be the main contribution to the final, trusted, prototype of CerCo (M3).

Work Package WP5 will develop a proof of concept prototype of a system to draw complexity assertions on the execution time of a C program. The system will exploit the output of the CerCo compiler and will be based on existing tools (like Caduceus) for the management of cost annotations and the synthesis of proof obligations (Task T5.1). The task will be leaded by UPD with contributions from UEDIN which has expertise as users of interactive theorem provers. In an additional task T5.2 UPD will focus on the automation of proofs of complexity obligations with contributions from UNIBO which has expertise in implementation of proof automation.

The synthesis of the annotating invariants will benefit from previous and ongoing research at UPD on the production of complexity bounds. A first approach that has been experimented on first-order functional programs amounts to combine a traditional termination proof with a numerical interpretation of the functions (called quasi-interpretations) that allows to bound the size of the computed data. A second approach which is currently under development at UPD addresses the complexity of higher-order functional programs with side-effects using the ideas of (elementary/light) linear logic. In this case, complexity bounds are extracted from the typing of the program. Of course, in both cases abstraction entails a loss of precision of the analysis and one challenge is to find classes of programs for which automatic analyses do provide practically useful bounds (cf. case study in task T5.3).

This final task T5.3 will be leaded by UPD and will concern the application of the techniques developed in task T5.2 (automation of proofs) to the C code generated by available compilers for synchronous languages such as Lustre or Esterel. This is a relevant class of C programs for which it is indeed important to get concrete complexity bounds. As a matter of fact, synchronous languages are built around a notion of instant (or phase) and in concrete applications one has to check that the duration of an instant is less than the time between two input events. A preliminary analysis reveals that the structure of the generated C code is rather simple. Then the planned case study aims at automating the process of producing and verifying the invariants which are needed to bound the duration of an instant for the generated C code.

Work Package WP6 is to manage the knowledge generated by the project and IPRs and to bring the technological advances to the scientific community and potential users and is further detailed in Section 3.2. In addition to the standard scientific papers and reports, we will also create software (the certified CerCo compiler itself and the proof-of-concept exploitation system developed in WP5) and test it on a significant case study that we expect to be immediately interesting also to private parties. All our software will be developed under an open license. The Consortium Agreement will contain the policy, rights and obligations on this matter. All partners will contribute to this work package that will be leaded by UNIBO.

Task T6.1, performed during the last year, will asses the technologies developed in the project in order to ensure fulfilment of actual requirement of the targeted exploitation communities and it will be based on the Untrusted CerCo Compiler (M2).

Self assessment and validation As explained in Section 1.1, the first outcome of the project is the CerCo cost-preserving compiler, first implemented in OCaml (Tasks T2.1, T2.2, T2.3 of WP2) and then in CIC (Tasks T2.4 of WP2, T3.2 of WP3 and T4.2 of WP4). The compiler will be fully certified for cost-preservation in WP3 and WP4, which are the main instrument for validation of WP2. In particular, in WP3 and WP4, we expect to detect bugs in the cost-preserving compiler or algorithms whose proof of correctness is too complex to be formalized; feedback will be immediately provided to WP2 in order to fix the bug or change the algorithms before propagating the changes back to WP3 and WP4. In particular, UNIBO and UEDIN will be responsible for the proofs and for bug detection, while UPD will be for bug fixing.

The deliverables provided in WP3 and WP4 are mostly self-validating, since they are formal proofs that are checked for correctness by the Matita interactive theorem prover. An additional validation of the methodology adopted in the proofs will be achieved by submitting them to international journals and by presenting them to international peer-reviewed conferences.

The second outcome of the project is the cost-annoting feature of the CerCo compiler. The concrete exploitability of the cost-annotations is investigated in the Tasks T5.1, T5.2, T5.3, T6.1. In particular, Tasks T5.1 and T5.2 are aimed at providing a first layer of techniques and tools to extend cost-annotations to exact computational complexity proofs for realistic programs and constitute the main self assessment exercise for cost-annotations. Difficulties faced in the exploitation of the cost annotations must be reflected on the data structures and algorithms used to compute

them in the compiler (WP2) and are likely to require modifications to the optimizations implemented in the compiler (WP2) and proved correct (in WP3 and WP4). Thus, once again, validation feedback needs to be threaded to every Work Package and intra-WP collaboration triggers bi-lateral collaborations between partners.

Tasks T5.3 and T6.1 are aimed at validating (in the particular scenario of synchronous languages for T5.3) also the techniques exploited in T5.1 and T5.2, in particular by identifying methodological weaknesses and potential solutions to improve exploitability in the long term. Task 6.1 will also target potential communities of users both in academia and in industry in order to disseminate the project results and collect timely feedback used for self-assessment.

Risk analysis and contingency plan A project like CerCo can encounter a number of adverse situations. We define a risk as a product between an adverse event and its consequences on the projects achievements of its objectives. A correct procedure to minimise the overall risk will be taken into account in order to minimise the possible occurrence of adverse events in the construction of the project.

Technical Risks

Risk	Probability	Remedial actions
Intra-procedural optimizations and optimizations altering in non trivial way the control flow of the program.	Low/Medium	Optimization are important but not crucial for the functional behaviour of the compiler: we may be ready to pay the improved semantic and complexity reliability with a small loss in performance. Hence, particularly problematic optimizations can be skipped.
During certification a bug in the untrusted compiler code is spotted.	High	The untrusted compiler will also be extensively tested by traditional techniques, minimizing the risk. Moreover, the certification starts six months before delivery of the untrusted CerCo compiler to spot most errors before user validation. Essential parts of the compiler will be certified first. Man-power can be shifted from certification of non crucial parts to bug-fixing and re-certification of the minimal functional core.
A show-stopper issue is faced during certification, undermining the goal of a fully certified compiler. E.g. the formal semantics of the source language turns out to be unfaithful and proofs must be redone from scratch.	Low	With the delivery of the second milestone, at month 24, we already provide a fully functional CerCo compiler, that can be tested with conventional techniques. Man-power previously assigned to certification is shifted to testing.
Matita turns out to be inadequate for the task.	Low/Medium	We will switch to the Coq proof assistant. Matita is based on the same logic of the Coq proof assistant and it is not difficult to manually port scripts from one system to the other one. Moreover, the two systems can directly share definitions and proof objects encoded in an XML dialect for the Calculus of Inductive Constructions.
Lack of portability of the compiler to different architectures	Low	Preservation of cost models could be very sensitive to the target architecture and thus the compiler could be unsuited to support different back-ends. Since we plan to deliver just a single back-end during the project, this will not affect the schedule. Generative programming techniques (as used in gcc) can be considered to improve portability.

Sensitivity of the cost annotations to the target architecture	High	Cost annotations are sensitive to the target architecture and they affect the user provided cost invariants and the proof obligations. Thus even deploying an application on similar devices could require major work to update the invariants and redo the proofs. The problem is made less severe by increasing the level of automation, e.g. integrating invariant generators and automatic theorem provers. These topics will be addressed during the project time-frame only in the use-cases and thus they do not affect the schedule. Further research will be required in the long term to improve robustness of costs invariants.
Lack of methodology to help the user in identifying the cost invariants	Medium	Guessing cost invariants tight enough for the user scenario can be very difficult. Invariant generation techniques can be exploited and further research will be needed after the end of the project. In the project time frame we will focus our case-studies on the compilation of high level synchronous programs. For this restricted domain, we expect to be able to automatically generate cost invariants without any major user intervention. If this expectation turns out to be wrong, we plan to devise a new set of case-studies or to shift man-power to improve the management of cost invariants.

Consortium Risks

Risk	Probability	Remedial actions
Not to be able to intervene with correction just in time	Low	To ask to WP leader to prepare periodically reports based on specific forms defined at the beginning of the project
Researchers might leave	Low/Medium	All work to be regularly documented and stored
Divergence among partners on project running	Low	Consortium agreement rules every conflict situation. The research of consensus is the first objective. However, after a reasonable amount of time has been allowed to illustration and defence of conflicting positions, in order to avoid deadlock in project operational progress, the approval of a two-third majority of Partners will be considered conclusive.
Bad consortium communication	Medium	Improve team building among members; improve communication facilities; increase face-to-face or telephone communications when possible

Management Risks

Risk	Probability	Remedial actions
Overestimate work load	Medium	Study, implement and certify more compiler optimizations. Alternatively, put more effort on WP5 to improve the proof of concept prototype of exploitation of the cost annotations or perform more case studies on more expressive synchronous languages, such as Lucid-Synchrone.
Underestimate work load in implementation	Low	Manpower can be reassigned from certification to implementation, certifying only core parts of the compiler.
Underestimate work load in certification	Medium	Certify only core parts of the compiler.
Unrealistic Time Schedule for milestone M1	Low	Tasks T3.3, T4.3 and partially T4.2 can be started before completion of deliverable D2.2 and perform in parallel with the late task.
Unrealistic Time Schedule for milestone M2	Low	This is likely to happen only if T5.1 is late, since other tasks contributing to M2 are to be completed six months in advance. In that case, task T5.2 and T5.3 will be suppressed reassigning man-power to the late task and to task T6.1 that has to be shortened to achieve in time milestone M3.
Inaccurate budget allocation	Medium	Identify necessary re-allocations among partners.

1.3.2 Timing of work: packages and their components

We show the timing of work in the Figure 9.

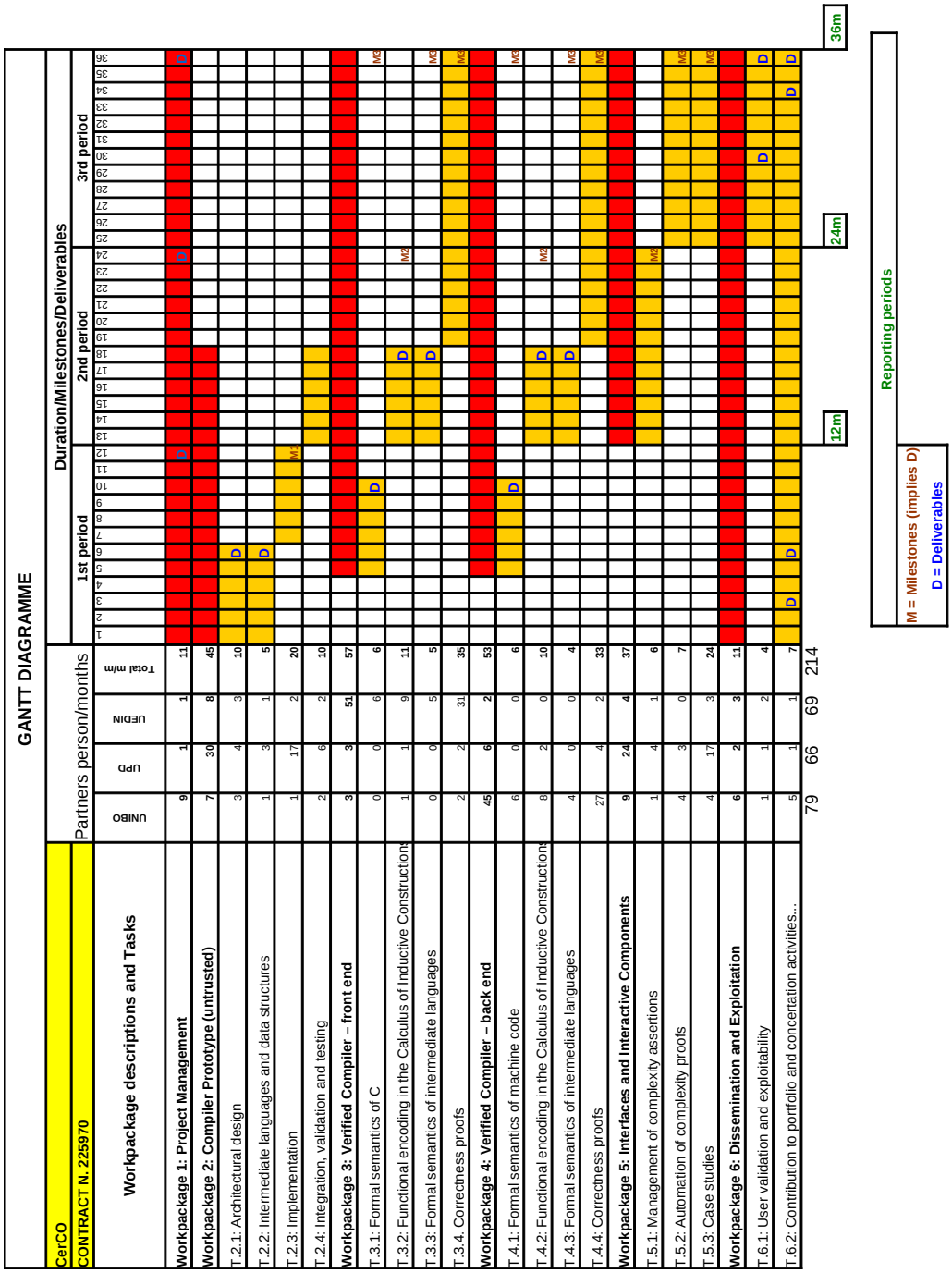


Figure 9: Gantt diagram

1.3.3 Work package list/overview

See Workplan Table WT1: List of work packages.

1.3.4 Deliverable list

The project is organized to provide a completely measurable and verifiable assessment of the state of advancement of the work, through a detailed list of intermediate deliverables. Most of the technical deliverables are executable prototypes, permitting an easy verification of their functionality. Some of them are also formal proofs developed in Matita, hence granted to be correct in themselves and proving the corresponding prototype to be bug free. Finally, all deliverables that are prototypes will come together with a report that provides the amount of information that is necessary to understand the design decisions and techniques used in the prototype in order to foster application of the same techniques to similar scenarios.

For details, see Workplan Table WT2: List of Deliverables.

1.3.5 Work package descriptions

See Workplan Table WT3: Work package description.

1.3.6 Efforts for the full duration of the project

See Workplan Table WT6: Project Effort by Beneficiary and Work Package
and Workplan Table WT7: Project Effort by Activity type per Beneficiary.

1.3.7 List of milestones and planning of reviews

We have one milestone every 12 months. After the first year we plan to have a functional prototype of a cost annotating compiler, written in OCaml. At the end of the second year, a mature version of the compiler will have been encoded in CIC together with an executable specification (interpreter) of the source, intermediate and target language. At the end of the project we will deliver a completely certified version of the cost annotating compiler, together with a proof of concept implementation of a system that exploits the cost annotations to verify the actual complexity of compiled programs.

See Workplan Table WT4: List of milestones and Workplan Table WT5: Tentative schedule of Project Reviews.

2 Implementation

2.1 Management structure and procedures

The Consortium is aware that management activities are extremely important for the successful realisation of the project as well as for a transparent accountability of the European contribution. The following section details the management structure foreseen for the CerCo project, the distribution of responsibilities, communication flow, decision making procedures and conflict management.

2.1.1 Management structure

The organisation structure of the Consortium will include the following components: Project Coordinator, Project Manager, Advisory Board. They will be assisted by WP leaders. A general schema is presented in Figure 10.

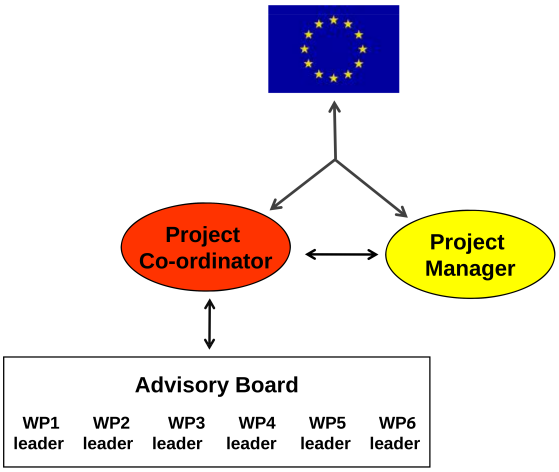


Figure 10: Project management interactions

Project Coordinator The project will be coordinated by the University of Bologna, Italy.
The Coordinator will be Dr. Claudio Sacerdoti Coen of the Department of Computer Science who will be the ultimate responsible for the overall coordination of the Project, especially but not limited to the scientific point of view. Dr. Claudio Sacerdoti Coen already acted as WP leader in Project IST-2001-33562 Mowgli, he coordinates the Strategic Project DAMA (Dimostrazione Assistita per la Matematica e l'Apprendimento) of the University of Bologna, and he participated to several European, National and local projects.

The Coordinator shall:

- act as an intermediary between the Contractors and the European Commission in order to keep constantly informed both the EC about any aspect that may affect the work progress and the Consortium on EC indications;
- receive the financial contribution from the Commission and ensure timely budget allocation to the Contractors consistently with the program of activities and the decisions taken by the appropriate Bodies;
- supervise the scientific, technical, financial and administrative progress of the Project;
- receive from the partners any proposal for modification/re-planning of the activities agreed;

- submit to the Commission reports, Project Deliverables and financial statements prepared and duly certified by the Contractors;
- keep accurate records identifying the budget share allocated to each Contractor and inform the EC of the distribution of funds and the date of transfer to the partners on an annual basis;
- organize and ensure appropriate communication among the partners;
- chair the meetings of the Advisory Board;

The coordinator will be supported by the Project Management Team and, for matters of general relevance, by the Advisory Board.

Project Manager The Project Manager will be appointed within the Project Management Team of the EU Research Dept. of the University of Bologna, that will ensure high quality management for the project. The European Research Department at UNIBO is dedicated to legal, financial and administrative support of European research projects and was set up in 1994. Since then, the European Research Department has gained a considerable experience in the management of international and European projects and has been recently strengthened in terms of skills and human resources. At present the Project Management Team at the European Research Department at UNIBO is managing about 40 projects funded within FP6 and more than 70 projects funded within FP7. For most of the projects coordinated by UNIBO (about 15), the Project Management Team is responsible for the overall project management activities, including day-by-day management, monitoring of activities, reporting and financial management. For the remaining projects, which involve UNIBO as a partner, the Team is dealing mainly with financial and administrative management. The Project Management Team is also involved in management activities within other European Commission Programme projects, such as Culture2000, Life, Interreg and Daphne.

The PM will be working in close cooperation with the Coordinator and will be in charge of the following tasks:

- assist the coordinator in the day-by-day management of the project, organize the procedures for internal communication within the consortium, as well as for the collection of reports and deliverables
- assist the Coordinator in the financial management of the project, collect financial statements from the partners,
- support the organization, preparation and follow up of periodical meetings,
- support the partners with reference to procedures requested by the European Commission.

Advisory Board The Advisory Board (AB) consists of one representative of each partner and is chaired by the Coordinator. It is responsible for discussing the general RTD direction of the project and for ensuring the completion of the work plan within the scheduled time frame. The AB will meet at least 4 times during the project (a kick off meeting and further general meetings at month 12, 24 and 36).

In particular, Advisory Board shall be responsible for:

- decisions concerning the work plan and its major changes;
- allocation of the budget to the work plan and any financial and budget-related matters;
- decisions with regard to any amendment of the terms of the EC contract and Consortium Agreement which should prove necessary;
- decisions concerning possible premature completion/termination of the project
- settling any disputes arising from project implementation
- IPR-related matters
- press releases and joint publications by the partners with regard to the project.

All decisions will be taken by consent; in case a voting is necessary, each representative shall have one vote and may appoint a substitute to attend and vote at any meeting of the AB.

As far as the work plan implementation is concerned, the AB will be operatively supported by Work Package Leaders appointed within the WP leading institutions.

The following tasks are in charge of the WP Leaders:

- coordination and monitoring of the progress of the tasks foreseen in the respective WP;
- organization, collection and quality control of the foreseen deliverables;
- coordination with the other WP Leaders and the Coordinator in order to ensure exchange of information.
- information of the Coordinator and of the other partners of any event within the WP that may affect the foreseen scheduling of the work

2.1.2 Management procedures

Meetings

The AB will meet at least 4 times during the project duration (36 months). Meetings shall be convened by the Coordinator with at least 30 calendar days prior notice, accompanied by an agenda. The agenda shall be deemed to be accepted unless one of the members notifies the coordinator and the other members in writing of additional points to the agenda, at the latest two working days before the date of the meeting. During the meeting, scientific presentation of the results obtained will be held by the consortium members. Minutes of the meetings shall be published on the web site within 30 calendar days after the date of the meeting. The minutes shall be considered as accepted by the other members if, within fifteen calendar days from publication, no member has objected in a traceable form to the Coordinator. Each meeting will deliver, further to the minutes, copies of associated scientific lectures. Copy of the presentations will be also available in the Web page.

Communication

The communication among partners is very important, since it not only allows the smooth realization of the project activities, but furthermore encourages exchange and research creativity. Further to project meeting, e-mail and telephone (conference calls) will be the preferred communication tool among partners. An Internet website will be set up at the project start to facilitate exchange of information among partners and to enable partners to upload and download project relevant data and information in an easy way. The project web-site will include a section with access restricted to project partners.

Monitoring

The project plan is structured in work packages grouping activities and tasks and following the logical phases of the project. Each work package will have specific deliverables and a verifiable end-point which represents an important milestone in the overall project. This structure will enable adequate and effective monitoring by the project partners and by the Commission. Possible problems and relative corrective measures will be early detected and will be brought to the attention of the AB.

Decision making and conflict resolution

In general, it is expected that possible conflicting views will be solved bilaterally within the WP or task where they may emerge. In the exceptional case when conflicts cannot be solved, the AB may be called to solve the conflict. The AB will make a final binding decision, if necessary, by voting. All pending conflicts will be solved within reasonable time frames and the AB shall make a decision within 20 working days.

A Consortium Agreement,detailing responsibilities of partners, financial provisions and other contractual provisions (including conflict resolution) will be agreed upon and signed by all partners during the first months of the project.

2.2 Beneficiaries

2.2.1 UNIBO

The Department of Computer Science of the University of Bologna covers a wide range of research areas in the field of Information and Communication Technologies, with particular emphasis on distributed and real time system, and theory and implementation of programming languages. The HELM team, active since year 2000 and coordinated by Prof. Asperti, is focused on the study and implementation of tools and techniques for the automation of formal reasoning, and the certification of software properties, resulting in the recent release of the Matita interactive theorem prover.

Role in the project and main tasks

UNIBO is the Project coordinator, hence responsible for the Project Management activities of WP1. It is also responsible for WP4, namely the development of the certified back-end of the CerCo compiler and WP6 dissemination and exploitation. It will also actively participate to the design and implementation of the untrusted prototype compiler (WP2) and the management of complexity assertions and automation of complexity proofs (WP5).

Relevant previous experience (Formalization and Management of Knowledge)

- European projects: EU Project IST-2001-33562 MoWGLI (coordinator: Prof. Andrea Asperti; WP leader: Dr. Claudio Sacerdoti Coen), EU Project IST-2001-37057 MKM-NET (member), TMR-Network LINEAR (member)
- Recent national and local projects: PRIN 2002-06 McTafi (national, member), DAMA (Strategic Project of Universiy of Bologna; coordinator: Dr. Claudio Sacerdoti Coen), "Formalization of Formal Topology by means of the interactive theorem prover Matita" (Strategic Project of the University of Padova; member)

Staff members involved (UNIBO)

Claudio Sacerdoti Coen	Researcher University of Bologna
Andrea Asperti	Professor University of Bologna
Research fellow (24 months)	to be hired with project fundings
PostDoc (24 months)	to be hired with project fundings

Key person

Claudio Sacerdoti Coen was born in 1976. He got a Ph.D. in Computer Science by the University of Bologna in 2004. After a Post-Doc at INRIA-Futurs (FR), he became a Ricercatore (permanent researcher) in Computer Science at the University of Bologna, where he has been teaching Operating Systems and Logic. His main research interests are the development of interactive theorem provers, their application to the formalization of mathematics, and knowledge management of formal mathematics. He is the author of over 30 peer-reviewed publications in mathematical knowledge management, type theory and automated reasoning. Recent publications relevant to the project comprise:

1. A.Asperti, W.Ricciotti, C.Sacerdoti Coen, E.Tassi. A compact kernel for the calculus of inductive constructions. In Special Issue on Interactive Proving and Proof Checking of the Academy Journal of Engineering Sciences (Sadhana) of the Indian Academy of Sciences. SADHANA. vol. 34(1), pp. 71–144, 2009.
2. A.Asperti, C.Sacerdoti Coen, E.Tassi, S.Zacchiroli. User Interaction with the Matita Proof Assistant. Journal of Automated Reasoning, Special Issue on User Interfaces for Theorem Proving, 39(2), pp.109–139, 2007.
3. C.Sacerdoti Coen. Declarative Representation of Proof Terms. In Special Issue on Programming Languages for Mechanized Mathematical Systems, Journal of Automated Reasoning, to appear in 2009.

2.2.2 UPD

The University Paris Diderot is the major pluri-disciplinary University in France. The University gathers 26,000 students, 1,800 researchers and academic staff and 125 research teams covering a wide range of fields. The laboratory PPS (Proofs, Programs, and Systems) is a joint laboratory of University Paris Diderot and the CNRS (Centre National de la Recherche Scientifique) whose research spans from mathematical foundations to open source software development. The main scientific themes of the laboratory are: (1) Mathematical structures of programming, in particular game semantics, linear logic, and algebraic methods, (2) Proof and rewriting theory, in particular type theory, extraction of programs from proofs via realisability interpretations, explicit substitutions, and formal proof developments in Coq (3) Concurrency and modelling, in particular models of concurrent programming, probabilistic systems, and molecular biology modelling. (4) Software tools, in particular web languages and programming and environments for open source software development and distribution.

Role in the project and main tasks

The UPD site will lead the Work Package 2 on the development of an (unverified) prototype compiler. This activity should be completed during the first 18 months of the project. Starting from month 12, it will take up coordination of Work Package 5 (management of complexity assertions and automation of complexity proofs).

Relevant previous experience (resource analysis)

- EU Project IST-2001-33149 on *Mobile Resource Guarantees* (2003) (member).
- Control of Resources and Interference in Synchronous Systems (2003-2006), French national ANR programme on Security and Informatics (coordinator).
- Parallelism and Security (2006-2009). French ANR programme on Security and Informatics (member).

Staff members involved (UPD)

Roberto Amadio	Professor University Paris Diderot
Yann Régis-Gianas	Lecturer/Assistant professor University Paris Diderot
PostDoc (12 months)	to be hired during year 1
PhD Student (36 months)	to be hired

Key person at UPD

Roberto Amadio is Professor at the University of Paris Diderot. Previously he has held positions as Professor at the University of Aix-Marseille 1 and as Research Fellow of CNRS in Nancy and Nice. He holds a PhD from the University of Pisa and an Habilitation from the University of Nancy. He is the author of a monograph on Domains and Lambda-Calculi and of over 50 publications on domain theory, semantics of λ-calculus and type theory, process calculi, models of migration, protocol analysis and verification, mobile code and resource control. He is a member of the steering committees of the conferences *Concurrency Theory*, *Computer Security Foundation Symposium*, and *European Joint Conferences on Theory and Practice of Software* (ETAPS). He is also responsible for the University Paris Diderot of the *Master Parisien de Recherche en Informatique* which is the leading Research Master in Informatics in France. He is currently teaching a course on syntax analysis and compilers.

Recent publications related to the project comprise:

1. R. Amadio, S. Coupet-Grimal, S. Dal Zilio, L. Jakubiec. A functional scenario for bytecode verification of resource bounds. In Proc. *Computer Science Logic*, Springer LNCS 3210 , 2004.
2. R. Amadio. Synthesis of max-plus quasi-interpretations. *Fundamenta Informaticae*, 65(1-2):29–60, 2005.
3. R. Amadio, S. Dal-Zilio. Resource control for synchronous cooperative threads. *Theoretical Computer Science*, 358:229–254, 2006.

2.2.3 UEDIN

The School of Informatics in the University of Edinburgh is one of the largest and most successful computing departments in the UK, as borne out by the latest UK Research Assessment Exercise (2008) which showed the School had a a significantly larger volume of research rated internationally excellent (3*) or world leading (4*) than any other university in the UK. The School has world-class research institutes in the areas of theoretical computer science, computing systems architecture, artificial intelligence and cognitive science.

This project will be carried out in the Laboratory for Foundations of Computer Science (LFCS) (http://www.lfcs.inf.ed.ac.uk), a research group in theoretical computer science, with expertise in areas including Logic and Semantics (Prof. Gordon Plotkin), Complexity (Dr. Kousha Etessami, Dr. Mary Cryan), Programming Languages (Prof. Philip Wadler) and Databases (Prof. Peter Buneman). The LFCS has a long history of work on computer assisted proof, including the development of foundational type theories such as the Edinburgh Logical Framework, and the Extended Calculus of Constructions. The LFCS has also worked on tools for machine proof, including the proof assistant LEGO, and Proof General, a generic front-end for proof assistants.

Also part of Edinburgh School of Informatics is the Institute for Computing Systems Architecture, with expertise in advanced compilers (Prof. Michael O'Boyle).

Role in the project and main tasks

The UEDIN site will lead the Work Package 3 on the certification of the front end of the CerCo compiler. It will also actively contribute to the development of the unverified prototype compiler in WP2.

Relevant previous experience (resource analysis)

- EU Project IST-2001-33149 MRG on *Mobile Resource Guarantees* (2003) (member).
- EU Project IST-15095 MOBIUS on *Mobility, Ubiquity and Security* (2005) (member)
- EPSRC funded project EP/C537068/1 ReQueST on *Resource Quantification in eScience Technologies* (2005)

Staff members involved (UEDIN)

Robert Pollack	Research Fellow, University of Edinburgh
Post Doc (24 months)	to be hired during year 1
PhD Student (36 months)	to be hired

Key person at UEDIN

Dr. Robert Pollack is a world expert in computer assisted proof and proof assistants. He has a long history of contributions in the application of type theory. In the late 1980s he developed the *LEGO* system at Edinburgh. More recently, Dr. Pollack has worked on several formalization projects, including a formalization of the fundamental theorem of algebra and investigations into ways to represent modularity and binding structure. He was on the steering committee of the EU Coordination Action 510996 TYPES, and involved in several previous EU actions.

For 17 years he worked as a software engineer specialized in real-time systems, in particular industrial process control and air traffic control. On these topics he had been Consultant to the Transportation Systems Center of the U.S. Department of Transportation, and chief software engineer of a startup company building process control computers.

Recent publications related to the project comprise:

1. Brian E. Aydemir, Arthur Charguraud, Benjamin C. Pierce, Randy Pollack, Stephanie Weirich. Engineering formal metatheory. In *Principles of Programming Languages* (POPL), pp. 3–15, IEEE 2008.
2. Thierry Coquand, Randy Pollack, Makoto Takeyama. A Logical Framework with Dependently Typed Records. Fundam. Inform. 65(1-2): 113-134, 2005.

2.3 Consortium as a whole

The participants to the consortium group together the minimal set of skills, know how and resources capable of achieving the project objectives. The project lies at the intersection between complexity, formal checking and compilers, requiring not only experts of the three fields, but also people with some knowledge and sensibility towards all the different topics.

The main research field of UNIBO is currently on tools and techniques for the automation of formal reasoning, and especially on the development of the new Interactive Theorem Prover Matita and its applications to correctness proofs for critical software applications. The group has direct experience on the formalization of micro-controllers in Matita, as testified by an executable specification for all models of Freescale 8bit micro-controllers available at the address http://matita.cs.unibo.it/library.shtml. The formalization captures all ALU families (Families HC05/HC08/RS08/HCS08) and all kind of memories that can be installed (RAM, ROM, Flash). The formalization comprise the computational cost of every machine operation, allowing to reason at the level of granularity of single clock cycles. An OCaml emulator running at reasonable speed has also been automatically extracted from the executable specification. In addition, the UNIBO site has past didactical and experimental experience in compiler construction, and good knowledge of complexity theory. In particular, Andrea Asperti held the chair in Programming Languages and Compilers in the period 1992-2000, writing scientific and didactical monographs on the topics,[6] and is currently teaching Computability and Complexity Theory.

The present research focus of UPD is mostly on complexity aspects, and resource control for real time systems, but they are expert in compiler construction, and comprise members with good experience in the use of interactive provers. In particular, Roberto Amadio is currently teaching a course on Syntax Analysis and Compilers at the University of Paris Diderot and he recently coordinated the National Project Criss, comprising the development of a small compiler for a first order functional language, producing bytecode annotated with relevant complexity information using shape analysis. This compiler was programmed in OCaml.

Finally, UEDIN offers much experience in the theory and implementation of programming languages, with particular emphasis on the formal engineering of their metatheory. Randy Pollack is the author of LEGO, a well known Interactive Prover of the nineties (no longer maintained) that was used by many students and researchers. Prior to that, Pollack had been an industrial software engineer specializing in real-time systems. He is currently working in the Mobility and Security Group at UEDIN, which has participated in 2 recent EU projects on formal certification of resource properties.[7] Pollack was also on the steering committee of the recently completed EU Coordination Action TYPES. In addition, UEDIN has significant research groups in areas of complexity and compilers.

All the partners have previously worked, alone or together, in European projects of different nature and sizes, including RTD projects. The above mentioned projects have successfully completed, demonstrating the capabilities of the CerCo partners of carrying out projects of European dimension, as well as their positive attitude to collaborative work. In many cases, the partners CerCo Consortium have acted as project coordinators; as such, they are well aware of execution and management policies of EU-funded projects.

2.3.1 Sub-contracting

Subcontractors will only be used for production of Certificates of Financial Statements, as required by FP7 rules. Other subcontracting has not resulted to be necessary since all competences required for carrying out the action are represented within the consortium.

2.3.2 Funding for beneficiaries from third countries

NOT APPLICABLE

[6] A.Asperti and S. Guerrini. The Optimal Implementation of Functional Programming Languages. Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, 1998. - A.Asperti and R.Davoli. Esperimenti di compilazione in ambiente Unix. Edizioni Pitagora, 1994.

[7] IST-2001-33149 MRG on *Mobile Resource Guarantees*, and IST-15095 MOBIUS on *Mobility, Ubiquity and Security*

2.3.3 Additional beneficiaries/Competitive calls

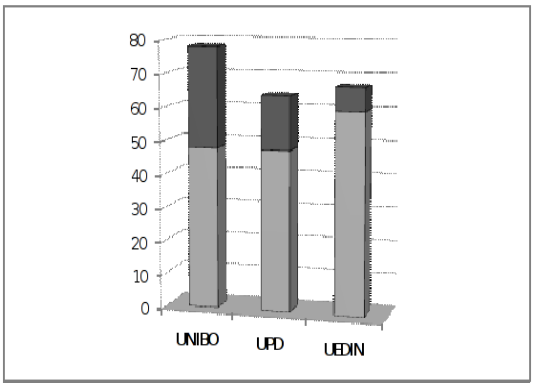
NOT APPLICABLE

2.3.4 Third parties

NOT APPLICABLE

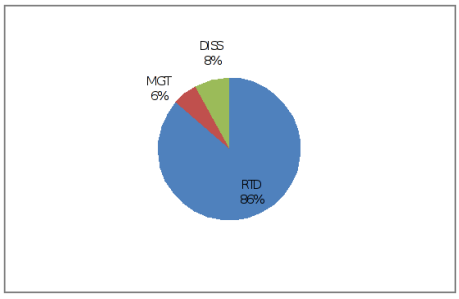
2.4 Resources to be committed

The overall budget of the CerCo project for the 36 months reported on the A3 form shows that the total costs of the project is 1,523,743 Euro, and the requested grant from the EU is 1,164,533 Euro. The total effort dedicated to the project illustrated in the graph below is equal to 214 person-months. The dark grey represents the share not funded person-months by EC.



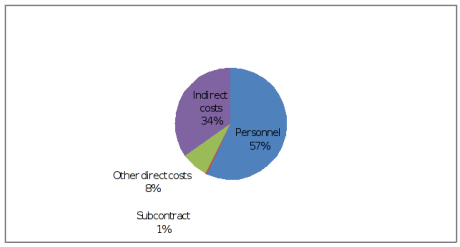
Person months distribution among partners

Breakdown by type of activity
A percentage of total costs equal to 86.31% will be allocated to RTD activities (including WP2, WP3, WP4 and WP5), excluding dissemination activities (WP6), which account for 7,98% of total costs. Management activities (WP1) are budgeted at 5,70% of the total costs. These costs include: the staff specifically dedicated to the project, involved in the daily coordination and the reporting of the project, costs for the certificates on the financial statement (when applicable).



Breakdown by type of activity

Breakdown by cost factors The above mentioned resources will be integrated to give to CerCo the necessary critical mass to achieve the project milestones and deliverables. All the resources described have been estimated analytically per costs category. In the graph here included is depicted the overall budget distributed among the main categories of eligible costs.



Breakdown by cost factor

The costs represented will cover:

Personnel costs (866,818€, 56,89%): they represent the main share of the budget. The allocation of person-months to the different partners reflects the activities they will carry out within the project. The overall effort of the project is 214 person months, including new personnel hired specifically for the implementation of the project and permanent personnel working in the different partner organizations, which constitutes the main own contribution element provided by participants to complement the EC contribution.

The person-month distribution by partner and WPs is quite balanced and reflects their role in the project:

- UNIBO (Participant n. 1, coordinator): 79 p-m in total, 70 p-m for RTD, 9 p-m management
- UPD (participant n. 2): 66 p-m in total, 65 p-m for RTD, 1 p-m management
- UEDIN (participant n. 3): 66 p-m in total, 65 p-m for RTD, 1 p-m management

Subcontracting costs (9,400€, 0,62%): they include costs related to the production of Certificates of Financial Statements, as required by FP7 rules. Other subcontracting has not resulted to be necessary since all competences required for carrying out the action are represented within the consortium.

Other direct costs (120,918€, 7,94%): they include:

- Travel& Subsistence (83,511€) costs provide for each partner the necessary budget for participating to the project coordination meetings foreseen in Europe. A relevant share of this resources are specifically assigned to the General Assembly meetings. In particular, annual or semester meetings have been planned to coordinate the CerCo activities: thus, considering the kick-off meeting, this leads to at most 7 coordination meetings for the overall project. We also foresee temporary visits and exchange of researchers between the participating sites, in particular in order to acquire competences on the tools used or developed by the visited site. A share of the T&S costs is allocated to dissemination activities and will allow partners to participate to conferences in order to present the contents and results of the project. Furthermore a small budget share has been kept by the Co-ordinator for management purposes (possible need to visit partners and/or the European Commission).

  Travels outside EU will be subject to prior authorization by the Project Officer.

- Equipment costs (4,000€) include only depreciation of some equipments which are needed for the project, while mostly partners will perform their research using in-home available instrumentation. In particular equipment includes a network server to be hosted by the coordinator and to be dedicated only to the project. The server will be used for dissemination (hosting the project Web site) and communication between partners (hosting the mailing list server and the message archive, but also acting as the central repository for the distributed versioning system used by partners to share and maintain the history of documents and prototypes). Moreover, we will set up on the server a continuous building system to run regression tests overnight in order to have early detection of (re)introduction of bugs or loss of features.

- Other specific costs (33,407€): conference fees, costs for the hosting of meetings, costs for dissemination activities such as the organization of workshops, the setting up and maintenance of the CerCo web site, and costs related to dissemination materials (flyers, posters, etc.).

Indirect costs (526,607€, 34,56%): they are calculated considering the indirect cost models chosen by each participant, according to the provisions of FP7 financial rules, and each specific accounting principle.

The following table details the overall budget of the project per WP and per main cost factors.

WP	n. p-m	Personnel	Subcontr.	Travels	Equipm.	Consumable	Other specific costs	Indirect costs	Total Cost	Total EC Contribution
WP1	11	48,419	9,400	0	0	0	0	29,087	86,906	—
Tot.MGT	11	48,419	9,400	0	0	0	0	29,087	86,906	86,906
WP2	45	172,496	0	18,178	4,000	0	4,375	117,648	316,697	—
WP3	57	237,949	0	25,258	0	0	7,871	104,013	375,091	—
WP4	53	221,448	0	10,794	0	0	3,587	142,053	377,882	—
WP5	37	138,788	0	16,589	0	0	787	89,373	245,537	—
WP6	11	47,718	0	12,692	0	0	16,787	44,433	121,630	—
Tot.RTD	203	818,399	0	83,511	4,000	0	33,407	497,520	1,436,837	1,077,627
TOTAL	214	866,818	9,400	83,511	4,000	0	33,407	526,607	1,523,743	1,164,533

Partner's resources which will complement the EC contribution As already said, the main own contribution provided by participants are person months of permanent personnel paid partner institutions. Moreover, all CerCo partners will use and share their own laboratories and facilities to carry on the foreseen research activities.

3 Potential impact

3.1 Strategic impact

In spite of the many recent, astonishing achievements in proof checking (proofs of the four colors theorem [Mackenzie, Gonthier] and prime number theorem [Avigad et al.], ongoing proof of the Kepler conjecture [Hales], etc.) and program verification (see the long list of results mentioned in Section 1.2) the field of computer assisted reasoning, with its long term vision of a fully dependable, formally checked information society is still one of the most visionary field of computer science. The impressive results obtained in this area are only partially due to the advancements in the tools for assisted reasoning, but especially to a growing confidence in their potentialities.

Compilers, filling the gap between humans and machines from the programming point of view, are one of the main tools at the base of the information technologies. We cannot hope to build truly trustworthy, dependable and long-lasting systems over shaky foundations: the semantics of compilation must be better understood. There is some history of work in certified compilers [Dave], and a series of workshop meetings on the topic (Compiler Optimization Meets Compiler Verification, 2003 – 2009). Only very recently has end-to-end certification of a realistic compiler been attempted [Leroy09, Leroy, Leroy et al.].

This previous work is focused on denotational aspects of computation. Our proposal is new in addressing intensional aspects, such as preserving space or time bounds of the source in compiled code. This is still perceived by the compiler community as a highly visionary goal, that would provide a major milestone in this area.

Our proposal contributes to the long-term community goal of formally checked reliable computer systems both directly, by our original work on a certified complexity preserving compiler, and more generally by large scale use of state-of-the-art tools and techniques for software certification. Our proposal takes such technologies past the proof-of-concept phase to the maturity level that could allow a substantial technology transfer towards industries.

The direct, long term, impact of the project on real-time systems (in particular reactive systems) will be that of dramatically increasing the trust on response time of the systems, while at the same time simplifying the code by removing some of the checks for approaching deadlines. Moreover, we envision a future where compilers (certified or not) will give back to the user guarantees and informations on the intensional behaviour of the produced binaries (like time, space and power consumption), to be exploited by semi-automated reasoning tools.

The certified compiler developed in the project is just a first step towards this direction. Being the first example of a compiler that provides intensional guarantees, we do not expect to be able to immediately exploit the provided cost annotations in an effective way. This requires an effective convergence of three communities, that are the compiler implementation, proof assistance and invariant generation ones. Moreover, further work is required to target processors for non embedded systems (which implement many hardware optimizations) and to enlarge the class of inter-procedural and loop optimizations that the compiler can handle while tracking the intensional properties. Together with the need of better understanding the issues related to multiple backends for different architectures, the time-frame required to bring this technology to mainstream can be estimated in a decade.

It would be certainly interesting to address advanced features of general purpose processors such as cache memory. However, we believe that such a task is not feasible in the time and resources allocated to our project.

Finally, we recall that Task 5.2 is dedicated to tools and techniques for automatic/computer-assisted inference of the global execution cost of C programs.

3.1.1 Contribution at the European level towards the expected impacts listed in the work programme

The project responds to the increasing expectation for trustworthy, dependable and long-lasting systems by developing reliable compilers not only from the point of view of behaviour but also of performance. Building certified compilers (coming with a machine checked proof of their semantics preservation) is an emerging topic whose relevance is destined to grow in coming years. It looks of high importance for ICT to have methods and tools for producing certifiable systems, and automatic checking of complex invariants (such as preservation of complexity) is a critical step and a major scientific challenge.

Expected impacts	Contribution of the project
ICT-relevant, visionary, high quality, long-term research of a foundational nature, involving bright new ideas of high-risk — high-payoff, aiming at a breakthrough, a paradigm shift, or at the proof of a novel scientific principle.	The main breakthrough envisaged by the project is the possibility to give a precise performance estimation for the executable (a task that is currently regarded as highly visionary in the compiler community), by creating a (certified) infrastructure allowing to draw conclusions on the target code, while comfortably reasoning on the source. Disposing of a such a tool would allow e.g. to shift programming of critical systems with strong temporal requirements from assembly to a high-level language, with a major beneficial impact on reusability and maintainability. At the same time, it would provide an occasion for rethinking the nature of compilation, measuring the actual impact of the many optimisation phases, and providing a better theoretical status for the many heuristics of current use.
Research refining the visionary ideas that have gone past the proof-of-concept phase to bring them to the maturity level where they could be taken up by the mainstream ICT programme objectives.	The realization of a tool offering precise and certified performance bounds on the generated code requires an essential paradigm shift of a foundational nature, emphasizing the role of *execution paths* and their preservation/modification along the the process of compilation. This poses new and interesting semantic challenges, and requires a detailed comparison between different semantic styles to discover the most suitable approach, also in view of its formal encoding and automatic checking. It is also clear that the knowledge gained by such a work could be profitably reused for the treatment high level program transformation techniques, and in general for the study of properties of program.

3.1.2 European dimension

The project lies at the intersection between complexity, formal checking and compilers, requiring a complex synergy between experts of the three fields, and providing an original forum for a radical interdisciplinary exploration of complexity issues from a computer assisted viewpoint. It is clear that having a geographically distributed team always poses additional problems with respect to a tightly integrated one. However, due to the dimension of the project and its visionary nature it is difficult to imagine a single agency to carry out the work. Even worse, in this case, due to the cost of the project, the results of the work would presumably be covered by copyright, and the software would hardly be open source, to the detriment of the diffusion of scientific knowledge of a foundational nature, and the industrial take up of innovative research results, that should be among the main goals of the European Union[8].

3.1.3 Related national and international research activities

During the development of the project we shall devote a particular attention to the evolution of related research efforts.

A closely related one is INRIA's CompCert, the certified compiler written in Coq by X. Leroy and his team. As for CerCo, the source language of CompCert is a large subset of C. The target language of CompCert is PowerPC assembly and several intermediate data structures and phases of the back-end are tightly bound to this choice. For instance, instruction selection is performed in the first phase of the back-end and pseudo-registers are partitioned into floating point and standard registers to match the actual machine registers in later phases. In CompCert we plan to address simpler microprocessors of the type frequently used for embedded systems. Moreover, the additional constraints imposed by the preservation of complexity could also prevent sharing of data structures and intermediate languages among the two projects. Nevertheless, the work done in Compcert will be a major source of inspiration.

[8]This is for instance the case of the Compcert verified compiler, which is distributed under the terms of the INRIA Non-Commercial License Agreement: a non-free license only granting rights to use the software for educational, research or evaluation purposes, but not for commercial uses.

Another INRIA's project we plan to collaborate with is Proval, in charge of the development the Why-Caduceus deductive platform, in view of the possible use of this tool for the proof-of-concept prototype of WP5. In particular, Caduceus and Why allow to generate proof obligation for C programs annotated with preconditions, postconditions and loop invariants. The obligations can be feed to several automated or interactive theorem provers to be solved. As a proof of concept for the exploitation of CerCo, we plan in WP5 to use these or similar systems to generate proof obligations for complexity invariants derived from the cost annotations produced by the CerCo compiler. We will also explore in Task T5.2 the possibility of helping the user in proving these obligations by increasing automation.

Our project is also related to the CMU/Penn Manifest Security Initiative. Its research objectives are "to develop the theoretical foundations for manifestly secure software and to demonstrate its feasibility in practice". As a matter of fact, a possible application of CerCo is to proof carrying code techniques, allowing to specify the computational cost of untrusted and potentially malicious extensions, along with a proof of their complexity.

The Poplmark Challenge, proposed by the University of Pennsylvania, is a set of benchmarks designed to evaluate mechanized theorem proving tools in the setting of the metatheory of programming languages. The benchmarks are drawn from the metatheory of a simple dialect of the lambda calculus, hence are focused on small scale formalization of programming languages. Even though our effort aims at a much larger scale certification, we expect that the experience gathered in the Poplmark Challenge might turn useful, particularly in the issues of name binding and complex induction principles.

INRIA-Microsoft's Mathematical Components project aims to demonstrate that formalized mathematical theories can, like modern software, be built out of components. Their effort is not only toward the formalization of finite group theory up to the odd order theorem, an ambitious challenge and a good test case, but also in the development of modular tools to make it possible to formalize such huge proofs. To pursue this last objective they started from SSReflect, a proof shell extending the Coq system developed by Gonthier to carry out the formalization of the Four Colour Theorem. SSReflect comes with a modular library of theorems conceived to reason about structures equipped with a decidable equality and a highly flexible set of commands specifically designed to carry out huge proofs in that domain. While structures with a decidable equality appear in a quite small part of mainstream mathematics, they are ubiquitous in the field of programming, and consequently in the implementation of a compiler. For that reason the technology developed by the Mathematical Components team seems an interesting device we may adopt, possibly tailoring it to our needs collaborating with them.

Our project is complementary to the Embound Project of the sixth Programme Framework. While the main focus of Embound was on static analysis techniques for Worst Case Analysis of real time systems, our project is aimed at building a verified infrastructure allowing to translate a high level analysis of the source program into a faithful counterpart on its executable. Moreover, while Embound started from a functional based, domain specific language as Hume, ending into a virtual abstract machine (HAM), we start from C and end up to the hardware. The methodology is also different: Embound was based on resource bound computational techniques that in recent years have clearly proved their limitations: we impose no restriction on the kind of techniques used to prove the complexity bounds, focusing our attention on the translation of computational costs from the source to the target, and proving its correctness.

3.2 Plan for the use and dissemination of foreground

3.2.1 Dissemination and Innovation Activities

Dissemination of project results into knowledge, products, and exploitation are key indicators of the success of the project. The dissemination strategy will be explicit about the links between the research process and the dissemination process, with particular attention paid to the links between the project's outputs and the dissemination tools and between these tools and the potential users of the project's results.

The potential target audience of the CerCo project is composed of the following categories:

- Academic communities, which should be made aware of both the methodological and technological approach developed in CerCo. In particular, formal verification is still seen as highly innovative in the community of compiler construction, and complexity has been under-addressed so far in the theorem proving community.

Other communities, like that of Worst Case Execution Time, could find in our technology a solid ground for linking high level analysis to actual code execution.

- Other EC funded initiatives, related either to compilation and formal verification techniques (in particular for embedded systems) or to their exploitation areas, like the European Technology Platform/Joint technology Initiative on Embedded Systems ARTEMIS.

- European software houses active in the development of compilers or static analyzers for embedded systems, with potential interest in certification of their products.

In order to reach these stakeholders and ensure an effective dissemination, the project foresees the following dissemination mechanisms:

- Definition of a project coordinated image and preparation of the dissemination materials, such as the project logo and a coordinated set of project tools for reports and presentation of the project results, be used in different dissemination occasions.

- Preparation of a project web site containing a reserved area needed as collaboration platform among the project partners, as well as public pages, to be used to showcase the results which can be disseminated. In particular, the web site will contain the project description; partners profile and relevant contact details; scientific papers and slides of presentations to international meetings; public project deliverables; press kit: project fact sheet and flyer; project poster; links to relevant projects; research programmes and associations; notice of important European and international events. The content of the site will be updated regularly as well as at the accomplishment of every deliverable. Furthermore,a quarterly newsletter will be published on the web public section and electronically distributed to potential user communities.

- Participation in national and international scientific events (seminars, clustering activities and working groups); presentation of results at conferences, seminars, workshops, both through speeches or posters; publication of technical and research papers in well-known scientific and industrial journals, magazines, newspapers. Within the initial dissemination plan, partners will contribute to create a list of possible conferences and events connected with the topics dealt with by the project in order to exploit all possible dissemination opportunities.

- Presentations of the technologies developed in the project to industrial parties with potential short and long term interests. This could also be achieved by inviting industrial representatives to late project meetings.

3.2.2 Exploitation of the results

Academic exploitation Academic partners will take great advantage from the results of CerCo mainly in terms of increased technical know-how and scientific knowledge, increased visibility in the scientific community of reference, increased expertise, exploitable for institutional academic purposes (e.g. didactic activities). We also expect that the rest of the academic community will take up the innovative ideas of the project, building on top of them code analyses at a level of accuracy that could have not been previously possible. Finally, we expect elaboration and instantiation of our methodology to other kinds of high level languages (functional, logic) and to more complex scenarios involving, for instance, a real time operating system.

Industrial exploitation The long term industrial exploitation of the results of the CerCo project is envisaged mainly in the area of the embedded systems/software, in particular in the case of safety critical applications and time critical (realtime) applications. In the short term, software houses producing compilers for embedded systems could immediately benefit from the CerCo cost annotating technology or, more generally, by the know-how provided in the certification of compilers. It is worth noting that several of these software houses are located in Europe, such as the medium-sized System Engineer Group of Freescale, which is headquartered in Scotland, Raisonance and Cosmic Software, which are headquartered in France, the small-size Hightech, headquartered in Germany, just to name a few.

3.2.3 Management of Intellectual Property Rights (IPR)

Apart from the Guideline for IPR in FP7 Projects (the Project Manager is in contact with the IPR Helpdesk Service organized by the EC) that will be the basis for the project IPR standards, more specific rules regulating specific aspects of the property and protection needs will be defined in the Consortium Agreement. All software will be developed under open licences and proofs will be public domain. The partners will commit in the Consortium Agreement to avoid filing patents on the software. Nevertheless, a continuous patent survey, carried out by the Coordinator, will assist the Advisory Board in decisions related to the danger of infringing existent patents.

The Consortium Agreement will be a formally binding legal document, prepared by the Coordinator and signed, upon approval of the Advisory Board, by all the participants at the latest within three months after the start of the Project.

The Consortium Agreement will be mainly aimed at complementing and better clarifying the rules stated in the EU Contract and Annexes and will further specify the recommendations and guidelines of EU IPR HelpDesk to the sensible results of the Project. In particular, it will be focused among others on:

1. Decision process and voting ways of the Advisory Board
2. Confidentiality
3. Publication authorization
4. IPR and ownership rules
5. Access Rights
6. Participant obligations
7. Periodic reporting

3.2.4 Contributions to standards

NOT APPLICABLE

3.2.5 Contributions to policy developments

NOT APPLICABLE

3.2.6 Risk assessment and related communication strategy

NOT APPLICABLE

4 Ethical issues

We expect no ethical issue may arise in the project.

ETHICAL ISSUES TABLE

	YES	PAGE
Informed consent		
Does the proposal involve children?		
Does the proposal involve patients or persons not able to give consent?		
Does the proposal involve adult healthy volunteers?		
Does the proposal involve Human Genetic Material?		
Does the proposal involve Human biological samples?		
Does the proposal involve Human data collection?		
Research on Human embryo/foetus		
Does the proposal involve Hyman Embryos?		
Does the proposal involve Human Foeatal Tissue/Cells?		
Does the proposal involve Human Embryonic Stem Cells?		
Privacy		
Does the proposal involve processing of genetic information or personal data (eg. health, sexual lifestyle, ethnicity, political opinion, religious or philosophical conviction)		
Does the proposal involve tracking the location or observation of people?		
Research on Animals		
Does the proposal involve research on animals?		
Are those animals transgenic small laboratory animals?		
Are those animals transgenic farm animals?		
Are those animals cloned farm animals?		
Are those animals non-human primates?		
Research Involving Developing Countries		
Use of local resources (genetic, animal, plant etc)		
Impact on local community		
Dual Use		
Research having direct military application		
Research having the potential for terrorist abuse		
ICT Implants		
Does the proposal involve clinical trials of ICT implants?		

I CONFIRM THAT NONE OF THE ABOVE ISSUES APPLY TO MY PROPOSAL	YES	

References

[Asperti et al.] A.Asperti, W.Ricciotti, C.Sacerdoti Coen, E.Tassi. A compact kernel for the calculus of inductive constructions. In Special Issue on Iteractive Proving and Proof Checking of the Academy Journal of Engineering Sciences (Sadhana) of the Indian Academy of Sciences. SADHANA (BANGALORE). vol. 34(1), pp. 71 - 144 ISSN: 0256-2499, 2009.

[Asperti et al.] A.Asperti, Claudio Sacerdoti Coen, Enrico Tassi, Stefano Zacchiroli. User Interaction with the Matita Proof Assistant. Journal of Automated Reasoning, Special Issue on User Interfaces for Theorem Proving, 39(2), pp.109–139, 2007.

[Asperti et al.] A. Asperti, C. Sacerdoti Coen, E. Tassi, S. Zacchiroli. Crafting a Proof Assistant In Proceedings of Types 2006, Lecture Notes in Computer Science (LNCS), Vol. 4502, pp. 18–32, 2007.

[POPLmark] Aydemir and Bohannon and Fairbairn and Foster and Pierce and Sewell and Vytiniotis and Washburn and Weirich and Zdancewic. Mechanized metatheory for the masses: The POPLmark Challenge. International Conference on Theorem Proving in Higher Order Logics (TPHOLs) 2005.

[Aydemir et al.] B.Aydemir, A.Charguéraud, B.C.Pierce, R.Pollack, S.Weiric. Engineering Formal Methatheory. Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2008)

[Appel and Palsberg] A.W.Appel, J.Palsberg. Modern Compiler Implementation in Java. Cambridge University Press, 1998.

[Avigad et al.] J.Avigad, K.Donnelly, D.Gray, P.Raff. A formally verified proof of the prime number theorem. ACM Transactions on Computational Logic (TOCL). Volume 9, Issue 1 (December 2007)

[Chlipala] A.Chlipala. A Certified Type-Preserving Compiler from Lambda Calculus to Assembly Language. Proceedings of ACM SIGPLAN 2007 Conference on Programming Language Design and Implementation (PLDI'07). June 2007.

[Dave] Maulik A. Dave. Compiler verification: a bibliography. ACM SIGSOFT Software Engineering Notes, Volume 28, Issue 6 (Nov. 2003).

[Dold and Vialard] A. Dold, V. Vialard. A Mechanically Verified Compiling Specification for a Lisp Compiler. Proc. of the 21st Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2001), December 13-15, Bangalore, India. Springer LNCS 2245.

[Filliâtre and Marché] J-C. Filliâtre, C Marché. Multi-Prover Verification of C Programs. In Sixth International Conference on Formal Engineering Methods (ICFEM), volume 3308 of Lecture Notes in Computer Science, pages 15-29, Seattle, November 2004. Springer-Verlag.

[Gonthier] G. Gonthier. Formal Proof–The Four-Color Theorem. Notices of the AMS, Vol 55, no. 11, pp. 1382–1393.

[Goos and Zimmermann] G.Goos, W.Zimmermann. Verification of Compilers, Bernhard Steffen and Ernst Rdiger Olderog (Ed.), Correct System Design, p. 201-230, Springer, Nov 1999.

[Hales] T.C.Hales. Formalizing the Proof of the Kepler Conjecture. Proceedings of the 17th International Conference on Theorem Proving in Higher Order Logics (TPHOLs 2004), Park City, Utah, USA, September 14-17, 2004. K.Slind, A.Bunker, G.Gopalakrishnan (Eds.) LNCS 3223 Springer 2004.

[Klein and Nipkow] G.Klein, T.Nipkow. A machine-checked model for a Java-like language, virtual machine, and compiler. ACM Transactions on Programming Languages and Systems (TOPLAS), Volume 28 , Issue 4 (July 2006).

[Mackenzie] D.Mackenzie. What in the Name of Euclid Is Going On Here? Science 4 March 2005. Vol. 307. no. 5714, pp. 1402 - 1403.

[Muchnick] S.S.Muchnick. Advanced Compiler Design Implementation. Morgan Kaufmann, 1997.

[Leinenbach et al.] D.Leinenbach, W.J.Paul, E.Petrova. Towards the Formal Verification of a C0 Compiler: Code Generation and Implementation Correctnes. SEFM 2005: 2-12.

[Leroy] X.Leroy. Formal certification of a compiler back-end, or: programming a compiler with a proof assistant. Proceedings of Proceedings of the 33th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2006)

[Leroy09] X.Leroy. Formal verification of a realistic compiler. To appear in Communications of the ACM, 2009.

[Leroy et al.] X.Leroy, S.Blazy, Z.Dargaye. Formal verification of a C compiler front-end. Proceedings of Formal Methods 2006, LNCS 4085.

[Leroy and Tristan] X.Leroy, J-B Tristan. Formal verification of translation validators: A case study on instruction scheduling optimizations. Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2008)

[Peyton Jones et al.] S.Peyton Jones, N.Ramsey, F.Reig. C–: A portable assembly language that supports garbage collection. Invited talk at PPDP'99, LNCS Vol1702, pp.1-28.

[Strecker] M.Strecker. Formal Verification of a Java Compiler in Isabelle. Proceedings of the 18th International Conference on Automated Deduction, Copenhagen, Denmark, July 27-30, 2002. LNCS vol.2392, pp.63-77.