Novel Automatic Solver for Program Analysis

Acronyme **Acronym**	SOPRANO		
Titre du projet	Nouveau prouveur automatique pour l'analyse de programmes		
Catégorie **Category**	Projet Collaboratif en Partenariat Public-Privé Cooperative Project between Academia and Industry		
Type de recherche **Type of research**	Recherche Fondamentale Basic Research		
Défi 7	Société de l'information et de la communication		
Aide demandée **Grant requested**	868k€	**Durée du projet** **Project duration**	42 months

Contents

Abstract

Modern societies crucially rely on digital infrastructures, and it is becoming clear that high-quality software can be obtained only with the help of proper software verification tools. Today most major verification approaches rely on automatic external solvers. These solvers, however, do not fill the current and future needs for verification: lack of satisfying model generation, lack of reasoning on difficult theories (e.g. floating-point arithmetic), lack of extensibility for specific or new needs. The SOPRANO project aims at solving these problems and preparing the next generation of verification-oriented solvers by gathering experts from academia and industry.

We will design a new framework for the cooperation of solvers, focused on model generation and borrowing principles from SMT (current standard) and CP (well-known in optimization). Our main scientific and technical objectives are the following. The first objective is to design a new collaboration framework for solvers, centered around synthesis rather than satisfiability and allowing cooperation beyond that of Nelson-Oppen while still providing minimal interfaces with theoretical guarantees. The second objective is to design new decision procedures for industry-relevant and hard-to-solve theories. The third objective is to implement these results in a new open-source platform. The fourth objective is to ensure industrial-adequacy of the techniques and tools developed through periodical evaluations from the industrial partners.

Our main approach is to combine principles coming from both SMT and CP. Roughly speaking, we seek to add to SMT the extensibility of CP and its native handling of domains, and to CP the elegant communication interfaces of SMT and its ability to reason over formulas with complex boolean structures (conflict analysis) and quantifiers. In a canonical SMT solver, the Boolean Theory enjoys a privileged status. We want to explore how to break this privileged status, in order to allow theory solvers to contribute more directly to each part of the solving process. For that, we will develop a notion of first-class domain and first-class conflict analysis.

The major results of the project includes scientific, technological, and industrial benefits. The project has the potential to deliver significant breakthroughs in automated solving and program analysis. The major outcome will be a paradigm shift in the combination of automated solvers, going beyond current standard cooperation frameworks in terms of extensibility, model-synthesis abilities, and richness of communication between theories. The resulting solvers will be more widely applicable, easier to tune for specific applications, and potentially more efficient, by encompassing the best trade-offs from SMT and CP. The major technological output will be the open-source platform implementing the results, including the cooperation mechanism. OCamlPro and AdaCore, the industrial partners, will take full advantage of the project to improve their lines of products and services. Finally, traditional industrial partners of CEA and UPSud will directly benefit from improvements of the verification tools of the project members.

The Consortium is composed of CEA (software verification tools and CP solving), University of Paris-Sud (SMT solving and floating-point theory), Inria Rennes (CP solving, floating-point theory), OcamlPro (software editor and SMT solver developer), Adacore (software solution for Ada programming language).

Modifications since the pre-proposition. The only modification is the addition of Guillaume Melquiond (Univ. Paris-Sud), a leading expert in automatic proof of programs manipulating floating-point numbers.

Partner	Name	First name	Position	Implication	Responsability
CEA	BOBOT	François	Research engineer	42 p.m.	Project leader SMT solver Deductive verification
	BARDIN	Sébastien	Research engineer	11 p.m.	Bitvector solver Array theory CP solver
	BRUNO	Marre	Research engineer	10 p.m.	CP solver Floating-point
UPSud	CONCHON	Sylvain	Professor	7 p.m.	Scientific coordinator SMT solver Theory combination
	CONTEJEAN	Evelyne	CR researcher	7 p.m.	SMT solver Rewriting engine
	MELQUIOND	Guillaume	CR researcher	11 p.m.	Floating-point Proof assistant
OcamlPro	LE FESSANT	Fabrice	Scientific advisor	4 p.m.	Scientific coordinator Software editor
	IGUERNELALA	Mohamed	Research engineer	28 p.m.	SMT solver
Inria	ACHER	Mathieu	Assistant professor	9 p.m.	Scientific coordinator CP solver
	GOTLIEB	Arnaud	DR researcher	9 p.m.	CP solver Floating-point solver Bitvector solver
Adacore	KANIG	Johannes	Research engineer	3 p.m.	Scientific coordinator Program verification
	MOY	Yannick	Research engineer	3 p.m.	Program verification Software editor
	DROSS	Claire	Research engineer	4 p.m.	SMT solver Quantified theory

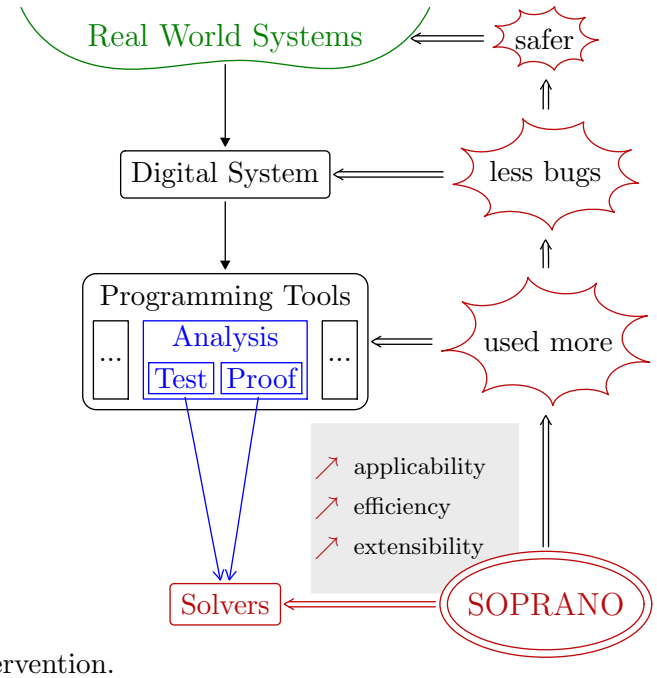
UPSud: University Paris-Sud with Inria

Inria: University Rennes 1 with Inria

1 Context, Position and Objectives

1.1 Context

Context. Modern societies rely in a crucial way on digital infrastructures, from the Web to medical health-care systems, e-voting, and smart cars. In that perspective, buggy or insecure software systems can have dramatic impacts in terms of financial losses, privacy leaks, human casualties, or even ecological disasters. Formal methods in general, and formal verification in particular (including both proof and testing), aim at developing theoretically-sound methods and tools for designing and implementing high-quality software, where quality can be understood in terms of safety, security, or performance.



The dominant verification method in the industry is testing, which has provided satisfactory results. However, the cost of testing has been ever increasing, dwarfing by an order of magnitude the cost of the actual development. As software is getting larger and more complex, these costs will keep increasing.

Formal methods have been proposed as an alternative to testing, with the potential to save cost while increasing the obtained guarantees. However, a prerequisite to the cost-effectiveness and acceptance of formal methods is a high degree of automation, that is, the ability to perform a comprehensive verification with few human intervention.

While formal methods are now a standard in the hardware industry, adoption in the software industry has been so far only very limited. The crucial difference between these two industries is precisely the lack of adequate automated reasoning techniques applicable to software, compared to those that apply to hardware. Yet it is becoming clear that high-quality software can be obtained in a cost-effective way only with the help of a solid bedrock of proper verification and analysis tools, and a few impressive success stories have already been obtained for example at Microsoft (Static Driver Verifier [3]) or at Airbus (with the Frama-C verification platform [27] developed by CEA, Inria, and University of Paris-Sud). The SPARK technology[1] developed by AdaCore has been successfully used in a variety of contexts (the iFacts air traffic control system in UK, a secure workstation by SecuNet, etc.).

Since the early 2000's, there is a significant trend in the research community toward building software verification technologies upon automated decision procedures for first-order theories ("solvers"), which contribute to more automation of these technologies, thus lowering the barrier for their adoption. Besides weakest-precondition calculi dating back to the 1970's [35], most major recent verification approaches heavily rely on external solvers, to name a few: Bounded Model Checking [18], Dynamic Symbolic Execution [39], CEGAR Model Checking [45, 54]. Automatic solvers for theories expressive enough to model significant parts of program behaviors have become the major building block of verification and analysis tools, which are themselves basic building blocks for high-quality software. Improving such solvers in terms of efficiency, expressiveness of the input language, and ease of extensibility, will directly lead to more efficient and applicable software verification tools. In turn, it will broaden the adoption of formal methods in software development and help to raise the overall quality of software systems.

Example: Industrial problem. Current SMT solvers, such as those integrated into industrial verification tools, are extremely efficient for some kinds of problems, e.g. linear arithmetic on real

[1] `http://www.spark-2014.org/`

numbers, but have some blind spots, typically floating-point numbers or nonlinear arithmetic. This is a problem for fully automated analysis, because any such blind spot will result in false alarms requiring careful examination by an expert. An example is the following excerpt of critical industrial code provided by AdaCore (written in Ada), which implements a simple division between floating-point numbers while carefully trying to avoid division by zero and numeric overflow:

```
function protectedFloatDivide(left, right : in Float) return Float is
   begin
      -- Check for fractional denominator
      if right < 1.0 then
         -- Guard against divide by zero
         if right = 0.0 then
            return Float'Last;
         -- Guard against overflow where left/right > Float'Last
         elsif right < (left / Float'Last ) then
            return Float'Last;
         else
            return left / right;
         end if;
      else
         return left / right;
      end if;
   end protectedFloatDivide;
```

It happens that the above code is incorrect, because it does not take into account the case where the divisor is a very small (so-called *subnormal*) floating-point number. Current solvers based on SMT technology are unable to produce useful results on this code because of the presence of division, which is a nonlinear operation. Note that the bug can be found quickly using a technique called *bit-blasting*, which encode floating-point numbers and operations into boolean variables and boolean circuits. But the proof that no overflow occur in a corrected version of the function takes several minutes. Such a long time for a single proof is unacceptable in automatic tools that have to process hundreds or thousands of such queries while delivering results in a reasonable time.

Similar problems arise with nonlinear arithmetic, modular integers or bitvectors. The current technologies are not yet powerful enough for these crucial domains. Finding a satisfactory solution to these kinds of problems is a powerful driving force for the SOPRANO project.

1.2 Problem and objectives

The scientific problem. Satisfiability Modulo Theory (SMT) [38] with the Nelson-Oppen communication scheme (NO) [59] is the current *de facto* standard in verification-oriented solvers. Especially, NO makes it possible to combine solvers for disjoint theories T_1 and T_2 into a solver for the combined theory $T_1 \uplus T_2$. This is particularly interesting for software verification where constraints are built over basic data types, typically integers and arrays. Contributing to this fruitful line of research, University of Paris-Sud has developed the Alt-Ergo prover [21], commercialized by OCamlPro and used both by the verification toolkit of AdaCore and by Frama-C. Yet SMT solvers show the following shortcomings:

- The cooperation mechanism is centered around satisfiability (proving that a formula has a solution) while verification is mostly interested in synthesis (finding a solution if any), either for test - the solution is typically the test input looked for, or for proof - generation of counter-examples when a property does not hold.
- Search (including decisions and conflict analysis) is done at the boolean level of atomic predicates, losing high-level information and sometimes making reasoning unduly complex.
- NO does not allow fine-grained cooperation between decision procedures since only (dis-)equalities of variables are shared, theories must be disjoint, finite-domain theories are not natively supported and must be encoded into the boolean part of the formula, blurring its high-level structure.

- A few theories essential for verification still do not have any satisfactory decision procedure: floating-point arithmetic, nonlinear arithmetic, modular arithmetic, bitvectors, arrays over variables with domains, etc.

On the other hand, a few teams including CEA and Inria are investigating the use of Constraint-Programming (CP) for verification purpose, with promising results on floating-point arithmetic [53, 1, 16] or bitvectors [5]. However, synthesis in CP is mostly limited to finite domains, and while it is rather easy to combine different families of constraints, all information between constraints is shared without the minimal interface and theoretical guarantees of NO.

Objectives. SOPRANO is a 42-month basic research project gathering experts from academia and industry in order to prepare the next generation of software verification-oriented solvers. Our main scientific and technical objectives are the following:

- **Obj1:** design a **new collaboration framework for solvers**, centered around synthesis rather than satisfiability and allowing cooperation beyond that of NO (richer communications, non-disjoint theories) while still providing minimal interfaces with theoretical guarantees [CEA, Inria, UPSud];
- **Obj2:** design **new decision procedures** for industry-relevant and hard-to-solve theories such as floating-point arithmetic, nonlinear arithmetic and arrays [CEA, Inria, UPSud];
- **Obj3:** implement these results in a **new open-source platform**[CEA, UPSud, OCamlPro];
- **Obj4:** ensure **industrial-adequacy** of the techniques and tools developed.

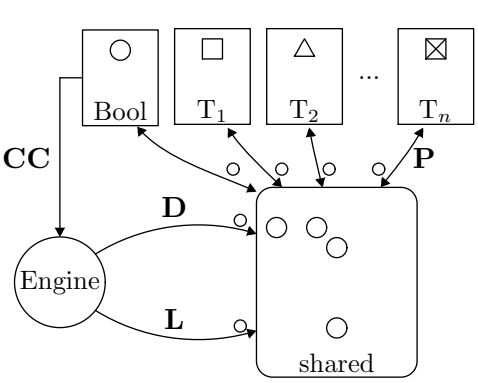
We also add two less technical objectives in order to maximize the project's impact: progress beyond the state of the art (**Obj-progress**), and visibility and dissemination (**Obj-visibility**). Success criteria for these objectives are listed in Table 1.

Objective	Criteria
Obj1	• ability to demonstrate advantages over standard approaches (richer communications, richer theories, more genericity, etc.) • implementation of the combination framework
Obj2	• design and implementation of a few theory solvers • experimental evaluation w.r.t. to existing approaches (efficiency, robustness)
Obj3	• availability and robustness of the open-source platform • experimental evaluation on standard benchmarks (SMT-COMP) • experimental evaluation on industrial-problem formulas from partners
Obj4	• connection with some verification tools from partners - GATEL and Frama-C (CEA), Why3 (UPSud), SPARK (AdaCore) • improve the current version of the Alt-Ergo solver (for a quick return on investment)
Obj-progress	• publications in top-level conferences and journals in automated reasoning or formal verification (IJCAR, LICS, LPAR, CP, CPAIOR, POPL, CAV, ETAPS, VMCAI, etc.) • research reports
Obj-visibility	• dissemination activities (open-source prototype, open-access benchmarks, tutorials, etc.) • participation and talks at seminars and workshops

Table 1: Success criteria

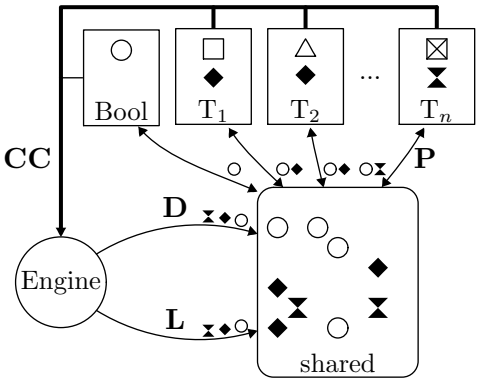
1.3 Method

Our main approach is **to combine principles coming from both SMT and CP**. Roughly speaking, we seek to add to SMT the extensibility of CP and its native handling of domains, and to CP the elegant communication interfaces of SMT and its ability to reason over formulas with complex boolean structures and quantifiers. We will also borrow the essential deduction principles of both approaches and integrate them in a unified framework, especially domain filtering (CP, forward reasoning) and learning (SMT, backward reasoning). Moreover, we focus on **verification-oriented formulas** for



SMT

In a canonical SMT solver, only boolean facts (○) are shared among theories. They come from the original formula, theory propagation (**P**), or equality and disequalities from the Nelson-Oppen combination. Moreover decisions (**D**) are done only on literals, and learning (**L**) produces only clauses containing literals involved in the conflict. Finally the search engine (CDCL) gets information only from the boolean theory (**CC**). So even if theories can compute precise specific informations (□, △, ⊠), it needs to encode them into booleans for the communications.



Proposed Method

In the proposed method, we want to remove the bottleneck due to boolean communications, by allowing every theory to use the same operations granted to the boolean theory. So instead of sharing information only on the domain of booleans, theories can agree on common domains (◆, ⧗). Moreover the engine asks the theories for choices on these domains (**CC**) and decision are done on them (**D**). During learning, the theory that share domains also share a common way for computing the conflict (**CC**) and specific constraints are learnt (**L**) from the conflict.

Figure 1: Comparison of SMT with the proposed method

which the consortium has the strongest expertise. Finally, we follow a **pragmatic industry-driven approach**, including tight collaboration with industrial partners and feedback from experiments, in order to keep the project focused on realistic problems and help us to explore relevant trade-offs between completeness and efficiency.

We describe hereafter how we will address the main technical objectives.

New collaboration framework for solvers. In a canonical SMT solver, the Boolean Theory enjoys a privileged status: theories communicate through boolean facts, including initial literals and (dis-)equalities from NO. The solving mechanism (propagation, decision, learning) operates mostly at this level, while each theory solver maintains its own private information store. We want to explore how to break this privileged status of the Boolean Theory, in order to allow theory solvers to contribute more directly to each part of the solving process. Our method is sketched in Figure 1. Especially, we plan to:

- develop a notion of **first-class domain**, meaning that solvers will share CP-like domains of variables (such as intervals or congruence) rather than being limited to (dis-)equalities and literals. Moreover different domains can be used at once, both because different data types require different specialized domain and because different domains can be complementary (e.g. we can use intervals, bitvectors, and floating-point *binades* for representing interesting facts about floating-point numbers). As already noted by several authors [5, 62], the abstract domains from *Abstract Interpretation* [26] provide an adequate tool for a generic notion of domains in CP. We will start from this notion. We also expect that reduced products and other domain-combination methods will prove useful.

- develop a notion of **first-class conflict analysis**, allowing theories to exchange rich conflicting facts (e.g. polynomials for linear rational arithmetic). Two theories that use the same domains

must also share a common conflict analysis language. This does not prevent a theory from having a second, more general language (e.g. for both nonlinear and linear arithmetics), as long as it can convert between both languages or add a "cut" to the proof (roughly, if t is a nonlinear term, one can pose $t = x$ and use x for the linear version).

- state the **properties and requirements for each domain and conflict analysis**. Since we allow theories to communicate with rich domains, as in CP, one theory solver could invalidate the guarantee properties of another theory solver. Verifying that all the solver properties still hold whenever a new theory solver is added does not scale. It is a well-known problem in software analysis, and we will apply the rely/guarantee approach [19]. For each domain and conflict analysis, we will define which properties it ensures and what it requires. Then we can prove that the requirements hold when we define a theory solver that uses them. So each theory solver is proved separately and the addition of a new theory solver only requires a local check.

- **center the combination scheme around synthesis** in order to simplify both witness construction and solver combination (agree on an explicit solution rather than on an implicit one), in the vein of model-based theory combination [31], where the Nelson-Oppen combination is guided by the (dis-)equalities holding on the internal models built by the theory solvers. We want to go a step further by relying on an incrementally-built model shared by solvers in order to ensure model agreement across theories.

Dedicated solvers. We will instantiate the previous framework for theories of interest, such as combining floats with rational numbers, and developing domain-aware decision procedures for arrays, bitvectors or modular arithmetic. We provide a few representative examples of the analyses we plan to explore:

- Linear arithmetic: the natural domains for rational and natural numbers are the domains of intervals and congruences. The constraints learnt by conflict are polynomials [49] and they are computed from the original polynomials the same way learnt clauses are computed from the original clauses. These techniques are well-known, hence the Linear arithmetic case will help us for preliminary design of the combination framework.

- Sets and arrays: considering a set $S = \{x, y, z\}$ and an element $t \in S$, then the domain of t, denoted by $\mathcal{D}(t)$, can be refined by intersecting it with $\mathcal{D}(x) \cup \mathcal{D}(y) \cup \mathcal{D}(z)$, similarly to [4]. This domain-based reasoning is orthogonal to the standard (expensive) case-split analysis on sets and arrays, and it captures *disjunctive information* that cannot always be recovered from conflict analysis.

- Bitvectors: bitvectors are standardly solved using bit-blasting [11], by associating a boolean variable to each bit of a bitvector variable and a boolean circuit to each bitvector operation. Once equipped with proper preprocessing, this straightforward approach works better that one could have expected. Yet, some realistic examples yield intractable boolean formulas. Bardin *et al.* proposes a promising CP-based approach for bitvectors [5] through an encoding into integers, together with dedicated domains and propagators. The present framework will allow us to explore extensions of this work, especially the design of dedicated learning mechanisms.

- Communication between integers, bitvectors, and floats: these three data types can be seen as different interpretations of the same objects, hence they can naturally be equipped with the same domains (e.g. intervals, congruence, trivalued bitvectors), allowing tight collaborations between their dedicated solvers through reduced products [5].

Platform implementation. The prover will be **extensible** using dynamic plugins in order to make it possible for the users to define theories that correspond best to their problems. Moreover, even the most basic theories will be designed so that they can be replaced for easy future prototyping. Since we want the communication between theories to be unhindered, we cannot restrict the interface given

to the theories. So the theories will separate themselves using well-defined module APIs on which rely/guarantee reasoning can be applied.

Second, we also aim at implementing all the features or APIs needed for an **easy integration** in our partners tools. This ranges from specific command-line options, timeout, step number limit (for better repeatability), specific output, specific APIs. All these specific developments in the solvers will be open-source and kept general enough for reusability.

Third, since many tradeoffs must be made during the implementation of a prover, we will use **benchmarks** coming from industrial problems to guide us.

Finally, since all the needed theories for program verification will not be available until the end of the project, we will rely on **early prototyping** to test our intuitions and early ideas. This prototype will be implemented upon the Alt-Ergo prover which is already used industrially and maintained by OcamlPro. These developments will not contain the combination framework because it is too invasive. The first point that will be experimented is incomplete domain propagation for floating-point numbers and evaluation of its impact on industrial programs. The second one is model output; it will be restricted to a few theories but it will allow to evaluate the communication between the provers and the program verification tools used by the partners (SPARK, Why3, Frama-C).

1.4 Challenges, Risks and Fallback

SOPRANO is clearly a highly challenging project. We identify four major challenges, each of them requiring significant progress with respect to the current state of the art:

Chal1: identify a sweet spot of theory combination, going beyond NO and its derivatives, with richer communication between theories and finer cooperation;

Chal2: design effective and generic learning mechanisms suitable for both SMT-like and CP-like decision procedures, and identify the right compromise between propagation and learning;

Chal3: address several industry-relevant and hard-to-solve theories, including floating-point arithmetic, nonlinear arithmetic, and bitvectors;

Chal4: understand how quantifiers can be handled in our synthesis-based cooperation framework.

Yet, we believe that we will be able to overcome most of these difficulties for the following reasons.

First, we strongly believe that the original approach sketched in Section 1.3 has the potential to significantly improve the current combination schemes (**Chal1**) and provide the right tools for designing more efficient domain-aware solvers (**Chal3**) and more generic learning mechanisms (**Chal2**).

Second, we rely on a strong consortium with international experts in the design and implementation of verification-oriented solvers, we stand upon strong prior results, and we deliberately focus on solving verification-oriented formulas for which the consortium has the strongest expertise. Relevant anterior results from partners include SMT solving [21, 13] (UPSud), integration of abstract interpretation into verification-oriented CP-solvers [52, 33] (CEA, Inria), decision procedures for floating-point arithmetic [23, 53, 1, 16, 30, 15] (CEA, Inria, UPSud), an original CP approach for bitvectors [5] (CEA), a new decision procedure for nonlinear arithmetic [22, 30] and combination of rational arithmetic and floats [23] (UPSud), an original combination of SMT and CP for arrays [4] in the FDCC prototype (Inria & CEA) (**Chal1**, **Chal2**, **Chal4**).

Third, we follow an industry-driven approach, including tight collaboration with industrial partners and feedback from experiments, in order to keep the project focused on the most industry-relevant problems. This helps mitigating the risk of wasting time with hard-to-solve theories by restricting ourselves to realistic instances, which may be easier to handle (**Chal3**). Especially, since program verification is an unsatisfiable problem, we can take a pragmatic approach and in some cases sacrifice completeness for the sake of fast termination (**Chal2**, **Chal3**, **Chal4**).

1.5 Results

The project has the potential to deliver significant breakthroughs in automated solving and program analysis. The major outcome will be a paradigm shift in the combination of automated solvers, going

beyond current standard cooperation frameworks in terms of extensibility, model-synthesis abilities, and richness of communication between theories. The resulting solvers will be more widely applicable, easier to tune for specific applications, and potentially more efficient, by encompassing the best trade-offs from SMT and CP. This will result in a wider use of verification and program analysis tools, yielding in turn safer, more secure, and more efficient programs and digital infrastructures. The proposed approach is very different from state-of-the-art, but we believe that it is the best way to overcome the limitations pointed out in the introduction and to prepare the future.

1.6 State of the Art

The field of automatic reasoning evolved in many different directions, from the development of very powerful higher-order logics aiming at the mechanization of mathematics, to the design of efficient decision procedures tailored to propositional logic (SAT DPLL [29]). Between these two extremes, SMT and CP are interested in efficient decision procedures for particular first-order theories with more or less restricted forms of quantification.

DPLL and SAT. A SAT solver has two possible outcomes: either a solution is found, or a proof of unsatisfiability is constructed. While the satisfiability of boolean formulas (SAT) is theoretically hard, modern SAT solvers are able to tackle huge practical problems efficiently. These achievements are in part due to the way the search is done, namely conflict-driven clause learning (CDCL) [64], which combines decision and learning tightly.

Satisfiability Modulo Theory. The extension of the SAT problem to specific theories is called Satisfiability Modulo Theories (SMT). SMT solvers commonly use the DPLL(T) approach [60, 38]: a SAT solver works on the propositional part of the formula, and decision procedures are used to validate the solution found from the perspective of specific theories. Multiple theories can be dealt with through combination frameworks such as Nelson-Oppen [59] or Shostak [21]. Nelson-Oppen makes it possible to combine two solvers for disjoint theories T_1 and T_2 into a solver for the combined theory $T_1 \uplus T_2$. The key point is that solvers must agree on (dis-)equalities between shared variables. NO can be smoothly integrated into DPLL(T) by adding to the initial formula all (dis-)equalities between proxy-variables and letting the SAT engine reasons about them.

Several refinements have been proposed along the years. Split-On-Demand [6] allows theory solvers to request splitting on new clauses (eg. $x <= 4 \vee x > 4$) sent to the SAT solver. Yet, splitting still occurs at the boolean level. Model-based combination [31] takes advantage of the internal (partial) models choosen by each theory solver for guiding the agreement on (dis-)equalities in Nelson-Oppen. The domains are still kept private, and the communication is done through (dis-)equalities.

Modern SMT solvers integrates a wide range of very efficient and specialized engines. Program analysis, however, often requires relatively basic reasoning on many different theories. Thus, the difficulty does not lie in the reasoning itself, but in the communication between theories. Yet, in the current SMT framework, these communications are restricted to the boolean level. As a consequence, many encodings are necessary, and some relevant information may be lost during the process.

S. Conchon and E. Contejean (UPSud) developed Alt-Ergo, an SMT solver based on CC(X) [21]. Contrary to usual SMT solvers, the CC(X) framework centralizes all the equalities in one place. Alt-Ergo also supports AC(X), an extension of CC(X) designed by M. Iguernelala; this extension natively supports associative and commutative operators. Finally, Alt-Ergo integrates a powerful new decision procedure for linear arithmetic [13], where decisions are performed on integer variables (still, inside the theory solver). Alt-Ergo is used as a backend for several verification tools, including the SPARK technology[2] (AdaCore) and the Frama-C verification platform [27] (CEA, UPsud), whose Qed module [24] can considerably simplify the formulas sent to Alt-Ergo.

Constraint Propagation. Much attention has been devoted in the CP community to the usage of domains as a way to facilitate communication between theories. By reducing program verification problems to showing that a constraint system over finite domains is satisfiable or unsatisfiable, the question of building efficient and effective solvers emerged 15 years ago, in particular through the early

[2] `http://www.spark-2014.org/`

and pioneering work of Marre [51] and Gotlieb *et al* [42]. More recently, by fostering the design of industry-strength constraint solvers, some works demonstrated the potential of CP to address heavy program verification tasks [20, 41].

From the seminal work of Howe and King from the University of Kent [46] and Leconte and Berstel from ILOG [50] emerged the idea of exploiting abstractions to enhance current constraint propagation in CP solvers dedicated to program verification. This effort led to many improvements for hard-to-solve theories such as floating-point arithmetic [1, 8, 9, 16, 17, 53, 55], linear and nonlinear constraints on bounded or natural integers [34, 40, 43],

The integration of these results in CP-solves highly increases their capabilities and efficiency, but because of the strong collaborations needed between theories, it leads to complex tools hard to maintain and extend. Another issue comes from the difficulty of turning constraint-based search procedures into learning-by-failure procedures. Indeed, although clause-learning is very popular in SAT and SMT solving, CP-based search procedures still do not learn from unsatisfiable cores which are massively produced during the search.

CEA and Inria have carried out pioneering work in both verification-oriented CP solvers (ACI V3F, 2003–2006) and the combination of Abstract Interpretation and CP (ANR CAVERN, 2008–2011). They developed respectively the GATeL tool for automatic testing of SCADE models [51, 52] and the INKA/Euclid tools for automatic testing of C programs [40, 41, 42]. Recent works include CP-based methods for floating-point arithmetic [16, 17, 53, 1], non-linear arithmetic [34], bitvectors [5], modular integers [43] and preliminary results on combination of SMT and CP for arrays [4]. Finally, CEA and Inria initiate the creation of the CSTVA workshop (Constraints in Software Testing, Verification and Analysis) which brings together people of distinct communities to discuss fruitful ideas around the use of constraints in program verification tasks.

Beyond the Current Practice. SMT solvers keep growing with the addition of new decision procedures. Some have proposed to adapt the CDCL approach to theories beyond propositional logic [25]. De Moura and Jovanović [48] revisited a complete but impractical technique for solving nonlinear problems called CAD; they turned it into a usable decision procedure using the CDCL approach for focusing the search and reducing the amount of computations. New decision procedures on floating-point numbers also used this approach [44]. In the meantime, a proposal extended the use of CDCL to the whole SMT framework and called it Model-Constructing Satisfiability (MCSat) calculus [32, 47]. This approach, however, keeps a strong link with propositional logic: it is still based on CNF formulas and keeps the domains inside the theories. For its part, the CP community extended the use of global constraints which can be considered as theories [10]. But these works have no mechanism of learning, or are restricted in a way similar to DPLL(T) [61]. *We propose to overcome the shortcomings of the MCSat framework with ideas of the CP community: making the domain a first-class citizen and letting theories communicate in a natural way.*

We cal also notice a growing trend toward the convergence of separated subfields of verification and automated reasoning. Abstract Interpretation revisited SAT/SMT [37] and CP [62, 5]. Yet, these works do not properly take into account learning. In the meantime, Bardin and Gotlieb combined propagation (CP) and reasoning (theory of array used in SMT) [4]. *The SOPRANO project provides a natural opportunity to push further such connections and take advantage of them.*

1.7 Position of the Project

1.7.1 Adequacy to ANR Call For Project 2014

This project falls into Challenge 7 of the ANR CfP 2014, entitled “Information and Communication Society”. It is especially relevant to Axis 7.2.2 “Software Science and Technology” (p. 52 of CfP) focusing on technologies and tools for building better software, since the new generation of automatic solvers we want to design is primarily geared toward program analysis. They will be first used in “automatic verification tools” for safety or security (such as provers or advanced testers), but such solvers can also be of interest in “optimized compilation” or in advanced type systems for high-level “programming languages”. Simply put, Axis 7.2.2 aims at improving tools and frameworks for building high-quality software systems, and automatic solvers is a current, powerful, and pervasive trend for

building them. In the same line, the project is also relevant to Axis 7.2.3 "Digital Security" (p. 52 of CfP) since better automatic solvers yield better automatic verification tools for security-related properties, such as automatically finding vulnerabilities or checking the absence of information leakage in a piece of software ("verification and resilience of software-intensive systems"). Finally, on a broader scope, the project will also address basic issues falling into Axis 7.2.1 "Foundations of Digital Systems" (p. 51 of CfP), such as exploring new combination frameworks for automatic solvers and designing learning algorithms for CP.

SOPRANO is a cooperative project gathering partners from both academia and industry. The project addresses fundamental issues, but in the same time is firmly based on industry-relevant problems. Indeed, program analysis and verification have proved to be domains where cutting-edge commercial products often rely on fundamental results in logic and theoretical computer science (see hardware model checking at Intel or IBM and software model checking at Microsoft).

1.7.2 Position w.r.t. anterior research projects from members of the consortium

SOPRANO stands upon a strong record of fruitful collaborative works between the project's partners. We summarize them hereafter, and we give a more precise description of the Hi-Lite project (2010–2013). SOPRANO build on these former results in order to prepare the next generation of verification-oriented solvers.

SMT-based verification: The development of the Alt-Ergo SMT solver was initiated in the ANR A3PAT project (2005–2009) (UPSud), in order to help proof assistants with trustworthy decision procedures. The tool was later consolidated in ADT Alt-Ergo (2009–2011) (UPSud), towards its industrial use in avionics by Airbus. In the same time, RNTL CAT (2006–2009) (CEA, UPSud, Inria) and ANR U3CAT (2009–2012) (CEA, UPSud, Inria) explored the usage of Alt-Ergo as a backend solver for the Frama-C verification platform, through the Why3 language, while ANR Decert (2009–2012) (UPSud, CEA, Inria) aimed at designing and implementing new efficient cooperating decision procedures (in particular for fragments of arithmetics), to standardize output interfaces based on proof certificates and to integrate SMT provers with skeptical proof assistants and larger verification contexts.

CP-based verification: (CEA, Inria) The ACI V3F project (2003–2006) [12] initiated the study of constraint-based testing and verification techniques for embedded C and C++ programs, leading to the development of specialized constraint solvers. The ANR CAVERN project (2008–2011) (Constraints and Abstractions for program VERificatioN) proposed to explore and extend the capabilities of Constraint Programming for the automated testing of imperative programs using notions from Abstract Interpretation.

FUI Hi-Lite (2010–2013): (UPSud, Adacore, CEA) Hi-Lite's goal was to promote the use of formal methods in developing high-integrity software. It has resulted in a complete redesign of the SPARK language and a reimplementation of the associated verification tools. The difference is impressive: the new SPARK 2014 language is both richer and more flexible; the formal verification tools are easier to use and much more automated; the associated methodology allows applying SPARK incrementally to legacy projects. The resulting toolset has been released commercially[3] for the first time in April 2014, one year after the end of the project. The increased usage of specification, however, leads to new proof challenges as described in the objectives of SOPRANO.

We describe hereafter three current projects involving partners of the consortium. They are complementary of SOPRANO in the sense that they all seek to extend the use of automatic provers, but they mostly take solvers as blackbox and do not try to improve them.

[3] `http://www.spark-2014.org/about`

Joint Laboratory ProofInUse (2014–2017). (UPSud, Adacore) This Joint Laboratory[4] shares the resources and knowledge between Vals/Toccata (UPSud) and AdaCore and aims at significantly increasing the number of industrial customers of the SPARK technology. To do so, it focuses on improving automation in program verification, but at a higher level than SOPRANO does (at the level of Why3, which produces the formulas sent to automatic provers).

ANR BWare (2013–2016). (UPSud, OCamlPro) This is an industrial research project funded by the INS program of ANR. This project aims to provide a mechanized framework to support the automated verification of proof obligations coming from the development of industrial applications using the B method and requiring high guarantees of confidence. One of its goal is to integrate the SMT solver Alt-Ergo as a back-end of Atelier B.

ANR Cafein (2013–2016). (CEA, UPSud) This is an industrial research project funded by the INS program of ANR. It addresses the formal verification of functional properties at specification level, for safety critical reactive systems. One of the goals of the project is to improve the level of automation of formal verification, by adapting and combining existing verification techniques such as SMT-based induction and abstract interpretation for invariant discovery.

1.7.3 Position w.r.t. national and international research teams

National level. Our consortium is centered around teams from "Plateau de Saclay" (CEA, UPSud, OcamlPro, AdaCore) specialized in formal methods, software verification, and automated reasoning. These teams have already a strong record of close and fruitful collaborations (through seminars, collaborative research projects, research networks such as Systematic or Digiteo, recruitment of former PhD students, etc.), leading to international-level achievements and even a few industrial success stories: the development of Frama-C (CEA, UPSud) and its industrialization at Airbus, the creation of OCamlPro, the industrialization of Alt-Ergo (UPSud, OcamlPro, AdaCore). SOPRANO will allow to support and consolidate this ecosystem.

The research on SMT solvers in France is very active at LORIA, Nancy. The VeriDis team develops the veriT solver, which is a concurrent of Alt-Ergo. The VeriDis and the CASSIS teams explore extensions of the Nelson-Oppen combination framework, with ideas different from the ones exposed in Section 1.3. We will make sure to communicate with these teams during the project. SOPRANO also regroups most of (the few teams) involves in CP-based verification, and we are used to collaborating tightly with the major remaining team led by Michel Rueher at University of Nice.

From an industrial point of view, industrial users of the verification tools developed by the consortium (including major French companies, such as Airbus, Dassault, Athos, Esterel, and EDF) will benefit from the enhanced capabilities of the new solver developed in SOPRANO.

International level. The two major SMT solvers are probably Z3 - developed at Microsoft R&D (RISE team), and CVC4 - developed at New-York University (NYU) and University of Iowa (UIowa). (Note that the Microsoft RISE team also develops verification tools in addition to Z3.) The Z3 and CVC4 teams are continuously trying to push their approaches beyond the state of the art. We can cite for example: extension of SMT solvers to least-fixpoint of Horn clauses [2] (Microsoft), or the model construction calculus approach [32, 47] (SRI, NYU, Microsoft). In Europe (Spain), Robert Nieuwenhuis and his team developed the SAT-solver Barcelogic which was recognized as one of the most powerful SAT-solver at the time. Extended with theories such as uninterpreted functions and difference logic ten years ago, Barcelogic has recently been commercialized and released as another SMT-solver.

The Australian team G12 led by Peter Stuckey and Kim Marriot has released the Zinc CP-solver [58] which is a general-purpose constraint optimization language. By implementing dedicated filtering procedures for modular integers and nonlinear polynomial constraints, Zinc is a an interesting environment for solving optimization problems. At Uppsala University in Sweden, Joseph Scott, Pierre Flener, and Justin Pearson have proposed a dedicated solver for solving problems with string

[4] `http://www.spark-2014.org/proofinuse`

manipulation [63]. This solver, built on top of Gecode, implements dedicated filtering procedures based on abstract domain computations over strings.

We already have contacts with most of these teams, and we will make sure to communicate with them along the project.

It must be noticed as well that several recent works establish connections between separate subfields of automated reasoning and/or verification. We can cite for example CP and Abstract Interpretation [62, 5] or SMT and Abstract Interpretation [37, 56]. The SOPRANO project provides a natural opportunity to study such connections and take advantage of them.

2 Scientific and Technical Program

2.1 General Description

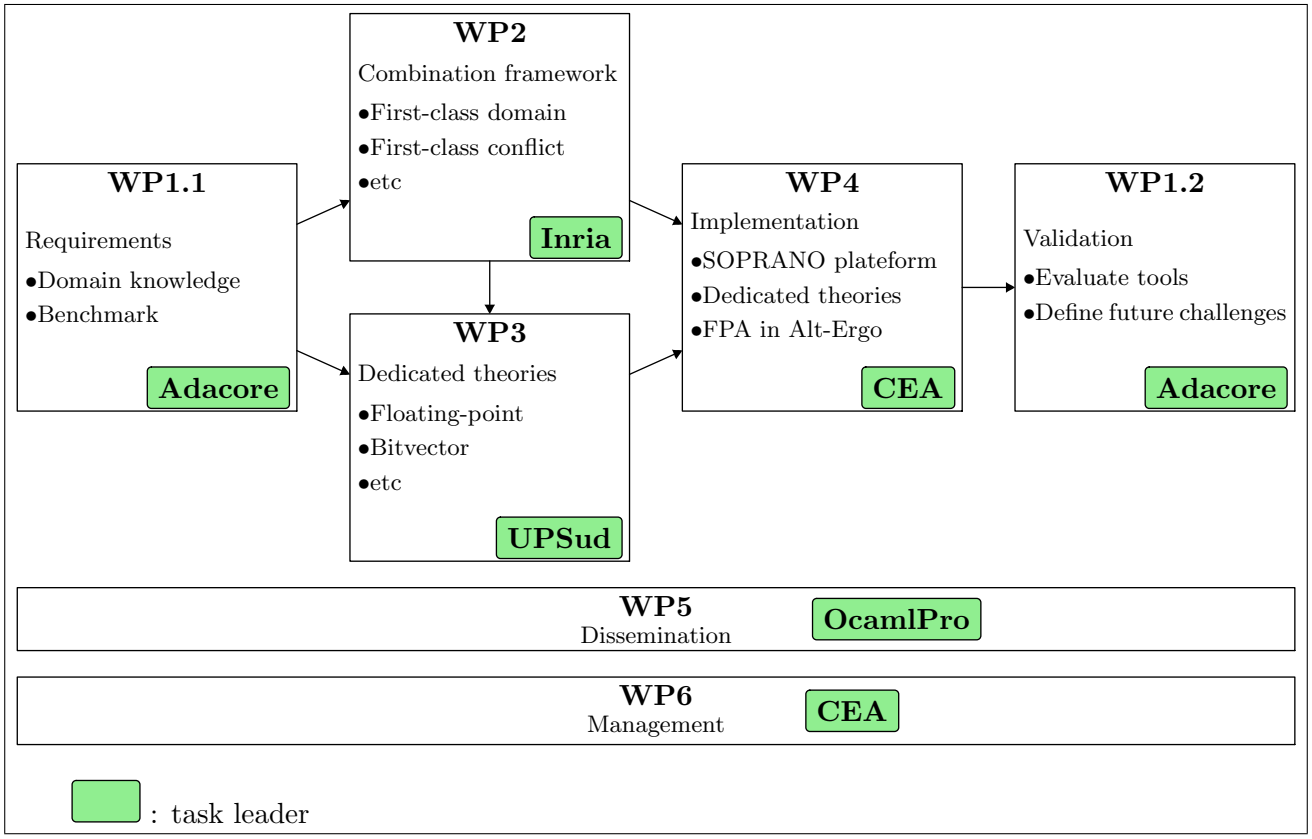


Figure 2: Workpackage workflow

The project is divided into four technical tasks (WP1 to WP4), plus a dissemination task (WP5), and a management task (WP6). The core technical tasks are WP2 (Combination), WP3 (Dedicated solvers), and WP4 (Implementation), respectively dedicated to objectives **Obj1**, **Obj2**, and **Obj3**. Requirement analysis and industrial validation will be performed in WP1 (resp. WP1.1 and WP1.2), addressing **Obj4**, while WP5 addresses the **Obj-visibility** objective. Finally, all the WPs participate in achieving **Obj-progress**. The major scientific and technical output of the project is the new solver combination scheme (WP2) and its implementation (WP4). The logical workflow among tasks is depicted in Figure 2, while Figure 3 shows the involvement of each partner in each task. A brief description of each task, with its leader, is given hereafter.

WP1: Requirements and validation [AdaCore]. The goal of WP1.1 is to identify industrial needs in terms of verification-oriented automatic solvers. They will be expressed mainly through a representative set of benchmarks and comparison of existing tools. Validation of the prototypes developed during the project tools will be conducted in WP1.2 at the end of the project. Adacore and OcamlPro have a prominent role here. We will also involve external partners, such as Dassault.

WP1 will first provide useful inputs to core technical tasks, especially the limitation of existing approaches and the identification of the crucial parts of the theories. During the project, WP1 will provide continuous guidance in order to keep focusing on industry-relevant problems. Finally, at the end of the project, WP1 will help to assess the viability and maturity of the proposed solutions.

WP2: Combination [INRIA]. WP2 aims at developing the new collaboration framework. We will start by reviewing state-of-the-art combination techniques, especially those of the tools compared in WP1. We will then design a framework following guidelines of Section 1.3 in order to overcome (part of) the limitations identified during WP1.

WP2 will provide to WP3 the framework in which the theories will be designed and to WP4 the formalization of the engine to be implemented.

WP3: Dedicated Theories [UPSud]. This task deals with the development of theories that have been identified in WP1 as useful for program verification. One already identified theory is floating-point arithmetic. For this theory, WP1 will help identify which kind of reasoning is needed. Other possible theories include linear and nonlinear arithmetic, as well as bitvectors.

WP3 will provide to WP4 the formalization of the dedicated theories to implement.

WP4: Implementation [CEA]. This task aims at creating a tool that fits the requirements expressed in WP1, using the framework and theories designed in WP2 and WP3. To provide early feedback on the choices made during the design of the dedicated theories, some of them will be implemented in the Alt-Ergo solver already used industrially. However Alt-Ergo suffers from the same limitation as the other solvers; in order to implement all the dedicated theories and the extensible framework, a new solver will be implemented during the project. It will be tested with the benchmark gathered during WP1.

WP4 will implement the results from WP2 and WP3, and it will provide the prototypes for validation during WP1.2.

WP5: Dissemination [OcamlPro]. WP5 deals with dissemination activities (open-access benchmarks, tutorials, workshops, etc.) in order to ensure maximal visibility to the project's results.

WP6: Management [CEA]. WP6 deals with the management and the coordination of the project as well as issues related to the consortium agreement.

2.2 Project Management

CEA is in charge of the general management of the project as well as reporting to ANR. Task leaders are in charge of the management of their tasks and report to the project leader. General meetings involving all participants will take place twice a year. We plan periodical steering meetings (email, phone, visio). Technical meetings focused on particular issues will be organized on-demand during the project. We also plan a 2- or 3-day midterm seminar in order to evaluate the progress and anticipate difficulties. Regarding day-to-day work, we will set up a collaborative working environment (website, git, and mailing list), and we plan a few visits of 2-4 days between partners.

2.3 Description by task

<table>
<tr><th colspan="6">WP 1. Requirements and validation</th></tr>
<tr><td>responsable</td><td colspan="5">Adacore</td></tr>
<tr><td>participants</td><td colspan="5">All</td></tr>
<tr><td>start</td><td colspan="2">T0+0</td><td>end</td><td colspan="2">T0+42</td></tr>
<tr><td>goal</td><td colspan="5">This task is divided into two parts: a requirement analysis at the beginning of the project (WP1.1), and a validation step at the end of the project (WP1.2).
WP1.1: Requirement analysis. As a first action, the industrial partner will summarize in a report the shortcomings of the existing provers, which will be addressed in this project. This report will help direct the work on the proof tools. AdaCore will then prepare a set of benchmarks taken from SPARK programs which are representative of the problems to be solved in the project. Some work will be needed to adapt the existing benchmarks to a format that is suitable for Alt-Ergo and the new prover. For example, SPARK currently uses real numbers to express properties of floating-point numbers, but a benchmark for floating-point support in provers should use floating-point operations directly. Finally, the benchmark will be integrated into the testing frameworks at OCamlPro and CEA.
W1.2: Validation. The benchmark will be used continuously by OCamlPro, Inria, and CEA to measure their progress. AdaCore will also regularly assess the tools. This will happen in a more formal way twice over the duration of the collaboration, once at T0 + 24 months as a mid-term evaluation, and once at T0 + 42 months to establish the final report.</td></tr>
<tr><td>method</td><td colspan="5">Diversity in the benchmarks. Every set of benchmarks will contain simplified problems and real-life examples. While the latter ones may be highly challenging, the simplified experiments (“in-vitro”) should be easier to reach, and should already provide us with interesting feedback and lessons.</td></tr>
<tr><td rowspan="5">deliverables</td><td>D1.1</td><td>T0+3</td><td>requirement analysis</td><td colspan="2">Adacore</td></tr>
<tr><td>D1.2</td><td>T0+9</td><td>set of benchmarks</td><td colspan="2">Adacore</td></tr>
<tr><td>D1.3</td><td>T0+12</td><td>benchmark environment</td><td colspan="2">OcamlPro</td></tr>
<tr><td>D1.4•</td><td>T0+24</td><td>midterm experimental evaluation</td><td colspan="2">Adacore</td></tr>
<tr><td>D1.5•</td><td>T0+42</td><td>final experimental evaluation</td><td colspan="2">Adacore</td></tr>
</table>

<table>
<tr><th colspan="6">WP 2. Combination</th></tr>
<tr><td>responsable</td><td colspan="5">Inria</td></tr>
<tr><td>participants</td><td colspan="5">CEA, UPSud, and Inria</td></tr>
<tr><td>start</td><td colspan="2">T0+3</td><td>end</td><td colspan="2">T0+36</td></tr>
<tr><td>goal</td><td colspan="5">The goal is to design a new collaboration framework for solvers, centered around synthesis rather than satisfiability and allowing cooperation beyond that of Nelson-Oppen (richer communications, non-disjoint theories) while still providing minimal interfaces with theoretical guarantees.
The Nelson-Oppen framework and derivatives achieve soundness by ensuring the theories agree on the equalities and disequalities between shared formulas. With hypotheses on the theories like disjointness, politeness, this agreement ensures the existence of a model (but does not build it explicitly). In the first deliverable, we will survey the existing combination techniques (UPSud and Inria). In order to remove the hypotheses of current combination frameworks, we will build concretely the models, and the theories will directly agree on the models. For that, the domains need to be a field specific concrete value. Moreover, since this agreement is an iterative process, an event engine must be devised. The second work deliverable aims at describing such combination, and how the theories will manipulate the domain and guide the search. Inria and CEA will bring their expertise in CP solvers for the domain interface and the event engine; UPSud will bring its combination framework expertise.
Since it will not be practical to try one model at a time, we will need to generalize the impossibility of some models by reasoning. Since the domains allow the theories to communicate, reasoning must also be an inter-theory process. The third deliverable aims at describing a parametric reasoning framework (CEA, UPSud, Inria).
During this work package, common theories such as booleans, uninterpreted function, linear arithmetic, or algebraic datatypes will be designed as instances of the framework. That will show whether the framework is general enough and give the plan for their implementation in WP4.4.
Finally the last deliverable aims at consolidating this framework, and taking into account the experience of WP3 and WP4 (CEA, UPSud, Inria, OCamlPro).
OCamlPro will bring a practical point-of-view during all this work package and also test early ideas on domains for floating-point arithmetic in the Alt-Ergo prover.</td></tr>
<tr><td>method</td><td colspan="5">Event-driven engine. We will integrate in the combination framework an event-driven engine as can be found in CP solvers. This makes it possible to give a general definition of propagation between theories but also possibly inside theories.
Learning framework. We will design a framework that makes it possible for different theories to communicate during the conflict-analysis phase.
No special case. We will design the collaboration framework without using special cases for common theories. In previous framework, some theories, such as boolean connectives or uninterpreted functions, are treated specially in order to be integrated. Other theories do not benefit from this special treatment, and their expressive power is limited as a consequence.
Join operations. We will design ways for a generic theory, like arrays or if-then-else, to propagate domains of other theories through theirs own symbols using join operations like in abstract interpretation. For example, the domain of $ite(c, x, y)$ is included in the union of the domain of x and the domain of y</td></tr>
<tr><td rowspan="4">deliverables</td><td>D2.1</td><td>T0+12</td><td colspan="2">survey of combination techniques</td><td>UPSud</td></tr>
<tr><td>D2.2•</td><td>T0+18</td><td colspan="2">combination framework I (first-class domain)</td><td>Inria</td></tr>
<tr><td>D2.3</td><td>T0+24</td><td colspan="2">combination framework II (first-class conflict)</td><td>CEA</td></tr>
<tr><td>D2.4</td><td>T0+36</td><td colspan="2">consolidation</td><td>UPSud</td></tr>
</table>

<table>
<tr><th colspan="6">WP 3. Dedicated theories</th></tr>
<tr><td>responsable</td><td colspan="5">UPSud</td></tr>
<tr><td>participants</td><td colspan="5">CEA, UPSud, OcamlPro, Inria</td></tr>
<tr><td>start</td><td colspan="2">T0+3</td><td>end</td><td colspan="2">T0+36</td></tr>
<tr><td>goal</td><td colspan="5">The goal is to design new decision procedures for industry-relevant and hard-to-solve theories such as floating-point arithmetic, nonlinear arithmetic, and arrays. The floating-point theory is the most critically theory missing in Alt-Ergo. However, some preliminary intuitions of the industrial needs, which will be checked by WP1, show that a complete decision procedure for floating-point operations is less needed than an incomplete but fast domain propagation. In order to verify that statement and speed up the time-to-market of this feature, it will be firstly designed for the Alt-Ergo combination framework $AC(X)$ and implemented during WP4.1. Then when WP2.2 starts describing the new combination framework, a floating-point solver that can be integrated to this framework will be designed. Some other interesting theories will be designed during the other phases and be implemented during WP4.4 and WP4.3.
One major goal of the work package is to enforce as much as possible communication between these theories, like using the key and data domains in the array theories [4], or creating a fast reduce-product of all the domains related to natural numbers (interval, modulus, bitvector [5], floating-point).
Another major goal is to reduce the number of theory axiomatization in industrial problems sent to the prover. Axiomatization using first-order quantification is very useful for describing a theory that the solver does not know using built-in theory. However it is never satisfactory since the quantifier engine is very general, and it requires a great knowledge of how the axioms are instantiated. One of the partners worked on formalizing how solvers instantiate and how to prove soundness and completeness of axiomated theories [36]. We will evaluate during the third part of this work package, whether coding an actual theory in the framework is harder than writing an axiomatization in this formalization, whether domains make such a theory more efficient or simpler to code, and whether libraries can be designed for simplifying this coding.</td></tr>
<tr><td>method</td><td colspan="5">Domain for floating-point numbers. We will reuse the experience of the prototype of the integration of floating-point arithmetic from [23] in $AC(X)$. However we will focus more strongly on precise floating-point domain propagation like in [53].
Bitvector. Bitvectors will not use direct bit-blasting as usually done; they transform bitvector constraints into boolean constraints, leading to a huge increase in the formula size. We will use instead a lazy version that extends [5], which uses a domain of bitvector and direct propagation of bitvector constraints. The conflict-analysis phase deduces constraints on bitvectors, which are in the worse case similar to the clause learnt from bit-blasting, but without making explicit the intermediary huge SAT formula.
Nonlinear arithmetic. We will first use a pragmatic but incomplete approach for nonlinearity and if needed more complicated techniques in the spirit of [48].</td></tr>
<tr><td rowspan="3">deliverables</td><td>D3.1•</td><td>T0+12</td><td colspan="2">FPA solver</td><td>UPSud</td></tr>
<tr><td>D3.2•</td><td>T0+24</td><td colspan="2">theory solver for our combination mechanisms I</td><td>Inria</td></tr>
<tr><td>D3.3</td><td>T0+36</td><td colspan="2">theory solver for our combination mechanisms II</td><td>UPSud</td></tr>
</table>

<table>
<tr><th colspan="5">WP 4. Implementation</th></tr>
<tr><td>responsable</td><td colspan="4">CEA</td></tr>
<tr><td>participants</td><td colspan="4">CEA, OcamlPro, INRIA, UPSud</td></tr>
<tr><td>start</td><td colspan="2">T0+6</td><td>end</td><td>T0+40</td></tr>
<tr><td>goal</td><td colspan="4">The project partly aims at improving the automatic reasoning used in the tools from the industrial partners. A prototype will be delivered during the project but the more competitive version will appear only at the end of the project. In order to improve the industrial tools on a shorter term, some already-used techniques will be implemented into Alt-Ergo as soon as the project starts (OcamlPro, UPSud). Moreover it will make it possible to test some of the new techniques in-situ.
We aim for the implementation to be as extensible as the framework designed during WP2 could be. Moreover the tools will accept dynamic plugins for new theories. One technical problem with the extensibility of such tools is that the parser and type-checker must also be extended which is not easy to do. Fortunately, the Why3 notions of theory help to solve this problem. The general type system and the ability to define new prefix, infix, or mixfix operators allow to define the signature of all the usual theories. So we want the Why3 language to be one of the input languages, and each plugin will define a Why3 theory for the surface language and link it to term constructors for the propagation and reasoning part. We also aim to implement all the features or APIs needed for an easy integration in our partners tools. This ranges from specific command-line options, timeout, step number limit (for better repeatability), specific output, specific APIs. All these specific development in the solvers will be open-source and kept general enough for reusability.</td></tr>
<tr><td>method</td><td colspan="4">Two-way communication with theories. The implementation is linked to the theoretical part of the project not only because it is the concretization of the former, but also because the former guides the latter. That is why the time of the implementation task overlaps with the corresponding fundamental task.
Testsuite. We will continuously use the benchmark given by WP1 as a testsuite for developing and guiding the heuristics and optimizations.
Polyglot. We will add a variety of input language (SMT-LIB2 [7], Why3 [14], Alt-Ergo [21]) for simplifying the use of the tools.</td></tr>
<tr><td rowspan="6">deliverables</td><td>D4.1</td><td>T0+12</td><td>extension of Alt-Ergo I (FPA)</td><td>OcamlPro</td></tr>
<tr><td>D4.2•</td><td>T0+12</td><td>preliminary implementation of the SOPRANO-solver, roadmap</td><td>CEA</td></tr>
<tr><td>D4.3</td><td>T0+24</td><td>extension of Alt-Ergo II</td><td>OcamlPro</td></tr>
<tr><td>D4.4</td><td>T0+24</td><td>implementation of SOPRANO-solver II (incl. D2.2 and D2.3)</td><td>CEA</td></tr>
<tr><td>D4.5•</td><td>T0+36</td><td>implementation of SOPRANO-solver III (incl. D3.2 and D3.3)</td><td>CEA</td></tr>
<tr><td>D4.6</td><td>T0+42</td><td>consolidation of prototypes</td><td>OcamlPro</td></tr>
</table>

WP 5. Dissemination				
responsable	OcamlPro			
participants	CEA, UPSud, OcamlPro, and Inria			
start	T0+6		end	T0+42
goal	WP5 focuses on the dissemination of the results obtained during the project. Moreover, in order to encourage other solver developers to focus on problems that matter for program verification, we will make the benchmarks produced during WP1 readily available. Last but not least, we will present the theoretical work done during WP2 and WP3 at conferences or in journals.			
method	**Open source.** We will publish the framework and theories implemented during this project with an open-source license. Tutorials and a mailing-list will be provided for a community to form. **Benchmarks.** We will add our benchmarks to existing benchmark databases, and make them available from the website of the project. **Publications.** We will produce early technical reports that could be later turned into publications.			
deliverables	D5.1	T0+18	publicly-available database of benchmark	OcamlPro
	D5.2	T0+20	website for SOPRANO-solver	CEA
	D5.3	T0+42	report about dissemination activities	OcamlPro

WP 6. Management				
responsable	CEA			
participants	CEA, UPSud, OcamlPro, and Inria			
start	T0+0		end	T0+42
goal	WP6 coordinates the project, deals with consortium agreement issues, sets up a cooperative framework (git, wiki, mailing list), and reports to ANR. We plan to have periodical steering meetings (email, phone, visio); general meetings will occur every 6 months. Technical meetings focused on particular issues will be organized on-demand during the project. We also plan a 2- or 3-day midterm seminar in order to review progress and difficulties. This seminar will be an opportunity to invite a few external experts.			
deliverables	D6.1	T0+3	project website, mailing lists, code repositories	CEA
	D6.2	T0+6	ANR starting report	CEA
	D6.3	T0+12	consortium agreement	CEA
	D6.4	T0+18	ANR midterm report	CEA
	D6.5	T0+21	midterm seminar	CEA
	D6.6	T0+42	ANR final report	CEA

2.4 Planning, summary

The list of deliverables is summarized in Table 2. A Gantt chart of the project is presented in Figure 3. Staff involvement is described in Table 3 and Table 4.

ID	MS	Date	Type	Description	Leader
D1.1		T0+3	R	requirement analysis	Adacore
D6.1		T0+3	W	project website, mailing lists, code repositories	CEA
D6.2		T0+6	R	ANR starting report	CEA
D1.2		T0+9	R	set of benchmarks	Adacore
D1.3		T0+12	P	benchmark environment	OcamlPro
D2.1		T0+12	R	survey of combination techniques	UPSud
D3.1	•	T0+12	R	FPA solver	UPSud
D4.1		T0+12	P	extension of Alt-Ergo I(FPA)	OcamlPro
D4.2	•	T0+12	P	preliminary implementation of the SOPRANO-solver, roadmap	CEA
D6.3		T0+12	CA	consortium agreement	CEA
D2.2	•	T0+18	R	combination framework I (first-class domain)	Inria
D5.1		T0+18	W	publicly-available database of benchmark	OcamlPro
D6.4		T0+18	R	ANR midterm report	CEA
D5.2		T0+20	W	website for SOPRANO-solver	CEA
D6.5		T0+21	S	midterm seminar	CEA
D1.4	•	T0+24	R	midterm experimental evaluation	Adacore
D2.3		T0+24	R	combination framework II (first-class conflict)	CEA
D3.2	•	T0+24	R	theory solver for our combination mechanisms I	Inria
D4.3		T0+24	P	extension of Alt-Ergo II	OcamlPro
D4.4		T0+24	P	implementation of SOPRANO-solver II (incl. D2.2 and D2.3)	CEA
D2.4		T0+36	R	consolidation	UPSud
D3.3		T0+36	R	theory solver for our combination mechanisms II	UPSud
D4.5	•	T0+36	P	implementation of SOPRANO-solver III (incl. D3.2 and D3.3)	CEA
D1.5	•	T0+42	R	final experimental evaluation	Adacore
D4.6		T0+42	P	consolidation of prototypes	OcamlPro
D5.3		T0+42	R	report about dissemination activities	OcamlPro
D6.6		T0+42	R	ANR final report	CEA

MS: milestone R: report, P: prototype, W: website, CA: consortium agreement, S: seminar

Table 2: Deliverables sorted by date

Partner	WP1	WP2	WP3	WP4	WP5	WP6	Total
CEA	3	12	12	30⋆	2	4⋆	63
UPSud	2	12	28⋆	10	2	1	55
Inria	2	13⋆	20	18	0	1	54
OcamlPro	3	4	11	28	5⋆	1	52
Adacore	7⋆	0	0	0	2	1	10
Total	17	41	71	86	11	8	234

⋆: WP leader

Table 3: Task Effort in PM

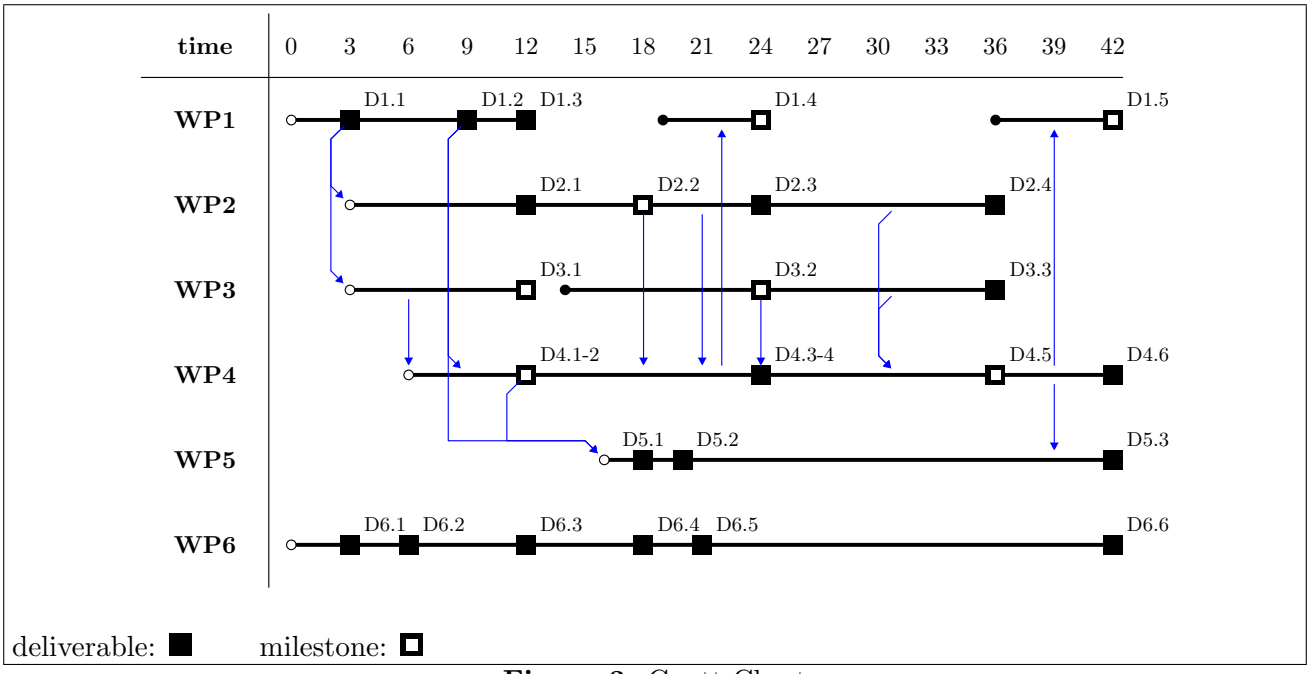


Figure 3: Gantt Chart

Partner	Name	First name	Position	Implication	Responsibility
CEA	BOBOT	François	Research engineer	42 p.m.	project leader leader WP4, 6 WP1, 2, 3, 4, 5
	BARDIN	Sébastien	Research engineer	11 p.m.	WP2, 3
	BRUNO	Marre	Research engineer	10 p.m.	WP2, 3, 4
UPSud	CONCHON	Sylvain	Professor	7 p.m.	WP2, 5
	CONTEJEAN	Evelyne	CR researcher	7 p.m.	WP2, 3
	MELQUIOND	Guillaume	CR researcher	11 p.m.	WP1, 2, 3, 4
	(to hire)		Post-doc	18 p.m.	WP3, 4
	(to hire)		Internship	6 p.m. ×2	WP3, 4
OcamlPro	LE FESSANT	Fabrice	Scientific advisor	4 p.m.	WP1, 5
	IGUERNELALA	Mohamed	Research engineer	28 p.m.	leader WP5 WP1, 3, 4, 5
	(to hire)		Internship	5 p.m. ×4	WP3,4,5
Inria	ACHER	Mathieu	Assistant professor	7 p.m.	leader WP2 WP2, 3
	GOTLIEB	Arnaud	DR researcher	7 p.m.	WP1, 2, 3
	(to hire)		PhD student	36 p.m.	WP2, 3, 4
Adacore	KANIG	Johannes	Research engineer	3 p.m.	leader WP1 WP1, 5
	MOY	Yannick	Research engineer	3 p.m.	WP1, 5
	DROSS	Claire	Research engineer	4 p.m.	WP1, 5

Table 4: Staff involvement

2.5 Consortium

CEA LIST. The LSL laboratory has been conceiving and realizing software verification tools for many years, and counts several successful industrial transfers: the Caveat proof tool for C programs is used by Airbus since 1998 (qualified for the DO-178B in 2008), its successor the open source Frama-C platform (co-developed with UPSud, based on the Alt-Ergo prover) is used at Airbus and NASA, the GATeL testing tool for SCADE programs is used by Esterel Technologies. François Bobot (coordinator), Bruno Marre, and Sébastien Bardin will participate in the project, bringing expertise in verification-oriented CP solving [52], including constraints for floating-point arithmetic [53], bitvectors [5], and combination of CP and SMT for arrays [4]. François Bobot has also an expertise in SMT since he worked on Alt-Ergo [13] (while a PhD student) and on the leading SMT solver CVC4.

University of Paris-Sud. The LRI will be involved in this project through the VALS team which develops methods and tools for program verification. Sylvain Conchon (PR), Évelyne Contejean (CR CNRS), and Guillaume Melquiond (CR Inria) will participate to SOPRANO. Sylvain Conchon and Évelyne Contejean are both involved in the design and implementation of the Alt-Ergo SMT solver [21, 13]. Guillaume Melquiond is involved in the development of Gappa, an automated prover for floating-point properties [30], and Flocq, a formalization of floating-point arithmetic [15]. The role of VALS in the project will be to bring its competences on Alt-Ergo and on the design of new decision procedures dedicated to program verification, e.g. floating-point arithmetic [23] or nonlinear arithmetic [22].

Inria. The team-project DIVERSE from Inria Rennes will be involved. The team focuses on improving development methods and tools for highly-configurable systems, including Model-Driven Engineering techniques, variability modeling and testing, code-based testing and test configuration generation. Participation of Inria Rennes will be coordinated by Arnaud Gotlieb (CR Inria), who has developed a strong expertise in the use of CP for verification and testing [33], including floating-point arithmetic [1, 16] and recent work around cooperation between SMT and CP for arrays [4].

AdaCore is the leading provider of commercial software solutions for the Ada programming language. AdaCore's customers build large, long-lived applications where safety, security, and reliability are critical. SPARK is a subset of Ada which is amenable to formal verification. It has been used in the industry for the last 25 years to formally verify software in industrial applications in avionics, railway, and the security industry. The newest version SPARK 2014, being based on the Why3 framework for program verification, makes full use of the power of SMT solvers; it uses the SMT solver Alt-Ergo at its heart. SPARK is developed jointly by AdaCore and Altran UK. AdaCore will adapt the SPARK tools to be able to benefit from the solver developed in this project, provide benchmarks and feedback.

OCamlPro is a young Inria start-up devoted to the improvement of software code quality. OCamlPro has two main activities. First, the company promotes the use of functional programming languages with strong static type-systems, currently mainly the OCaml language. Second, the company develops verification tools to improve the quality of programs developed with standard (weakly typed) programming languages. Among these tools, OCamlPro improves and distributes the Alt-Ergo automatic theorem prover from University of Paris-Sud, which is used inside verification tools in the avionic industry to prove the correctness of critical code.

Quality of the consortium. The consortium shows three major strengths, which should allow to meet the project's ambitious goals.

- *Quality, diversity, and complementarity of partners.* All researchers and engineers involved in SOPRANO are high-quality experts working at international quality standards and used to collaborative projects. The consortium gathers experts whose competences span over all main aspects of the project: SMT, CP, development of solvers for verification purposes and industrial experiments. Moreover, partners range from academia to industry, including a company selling solvers (OCamlPro) and a company using solvers (AdaCore).
- *Strong prior results.* SOPRANO can stand upon strong prior results from the partners. To name a few: strong expertise in SMT solving (UPSud [21, 13]), strong expertise in CP for verification

purpose (CEA [52], Inria [33]), innovative works on floating-point arithmetic constraints (UPSud [30, 23], CEA [53], Inria [1, 16]) and bitvectors (CEA [5]), first ideas on combining CP and SMT for the theory of arrays (CEA & Inria [4]).

- *Strong prior collaborations.* Finally, the different partners are used to collaborating together with a strong record of fruitful collaborations, including the design and implementation of the open-source verification platform Frama-C (CEA, UPSud) and the industrialization of Alt-Ergo (UPSud, OCamlPro). In order to ensure strong collaboration within the project, we will ask for co-supervised PhD and Post-Doc grants between the academic partners.

2.6 Scientific Justification of Requested Resources

Why a 42 months project? We request a 42-month project for the following reasons. First, a project longer than 36 months is much more convenient to recruit *good* PhD students, as we can spend the first months searching for candidates. Second, SOPRANO contains a significant amount of prototype implementation and empiric evaluation. These activities require lots of time, and experiments often require a consolidation step after the first try. Note that we have planned regularly-spaced deliverables and milestones in order to ensure project progress.

	permanent (p.m.)	non-permanent (p.m.)	total cost (k€)	funding request (k€)
CEA	63	0	670	335
UPSud	25	(1 PD, 2 IS) 30	493	132
Inria	18	(PhD) 36	397	143
OcamlPro	32	20	440	198
Adacore	10	0	134	60
Total	148	86	2134	868

PD: Post-doc IS: Internship PhD: PhD student

Table 5: Total effort

CEA LIST

- **Permanent staff.** People involved in the project are François Bobot (100%), Sébastien Bardin (26%) and Bruno Marre (24%). This is a total of 63 person.months, for a cost of 386k€
- **Missions.** We ask for 17.5k€: project meeting and technical visits (7k€), international conferences and workshop (10.5k€)
- **Other.** We ask for 3k€ of furniture, including laptops and scientific books.
- **Structure cost.** 264k€

University of Paris-Sud

- **Permanent staff.** People involved in the project are : Evelyne Contejean, Guillaume Melquiond and Sylvain Conchon. This is a total of 25 person.months.
- **Non-permanent staff.** A Post-doc will be involved in the project for 18 person.months. There will also be two internships involved for 12 person.months.
- **Missions.** We will need 20k€ for project meetings, technical visits and international conferences.
- **Other.** We ask for 6k€ for furniture, including laptops.
- **Structure cost.** 5k€.

Inria

- **Permanent staff.** People involved in the project are Arnaud Gotlieb and Mathieu Acher. This is a total of 18 person.months.
- **Non-permanent staff.** A PhD student will be involved in the project for 36 person.months (i.e., for 3 years as typical in PhD thesis) and for 120k€.
- **Missions.** We will need 15k€ for project meetings, technical visits and international conferences. The PhD student involved in the project is likely to present papers and disseminate ideas at international venues.
- **Other.** We ask for 2,5k€ for furniture, including laptops and scientific books.
- **Structure cost.** 5k€

Adacore

- **Permanent staff.** The engineers Johannes Kanig, Claire Dross, and Yannick Moy are involved in the project, for a total of 10 person.months, and for a cost of 80k€.
- **Missions.** We will need 12k€ for project meetings, technical visits and international conferences.
- **Other.** We ask for 2k€ of other cost,
- **Structure cost.** The indirect cost of the permanent positions is around 40k€.

OcamlPro

- **Permanent staff.** Mohamed Iguernelala at OCamlPro is currently maintaining the official version of Alt-Ergo, and will spend a lot of time designing and prototyping improvements during this project. So, people involved in the project are Fabrice Le Fessant (Scientific Advisor, involved for 4 person.months) and one or two R&D Engineers (among which Mohamed Iguernelala), involved for 28 person.months, and a total cost of 233k€.
- **Non-permanent staff.** We plan to take master students as interns, for 20 person.months (i.e. 4 internships), for 10k€.
- **Missions.** We will need 10k€ for project meetings, technical visits and international conferences.
- **Other.** We ask for 3k€ for furniture, including laptops and related electronic devices.
- **Structure cost.** 173k€

3 Impact

3.1 Protection of Results

The basic guidelines for IP issues will be the following: each partner keeps its anterior ownership and has the whole property of results obtained alone. New results coming from cooperative work between partners will be shared among these partners. All the software developed inside the project, including the extensible platform, will be distributed among the developers under an open-source license (the exact license being yet to define, probably LGPL). A consortium agreement among all partners will be established at the very beginning of the project (D6.3). We provide in appendix a refined description of these principles *in French*. The description is only indicative.

3.2 Dissemination and Valorisation

The **short-term impact** of the project includes scientific, technological, and industrial benefits.

Science: SOPRANO will mainly deliver a new framework for solver combination, allowing finer cooperations while being less restrictive on the underlying theories. Other important outputs of the project include new dedicated solvers for verification-relevant and hard-to-solve theories such as floating-point arithmetic, and learning mechanisms adapted to CP-like reasoning. The combination of CP and SMT has the potential to lead to new insights benefiting both fields.

Technology: The major output will be an open-source platform implementing the results, including the cooperation mechanism. The platform will be licensed under LGPL (or equivalent) in order to ensure maximal dissemination, and the Intellectual Property will be shared between CEA, University of Paris-Sud, and OCamlPro. Besides, several existing solvers and verification tools of the partners will be updated, including GATeL and Frama-C (CEA), SPARK (AdaCore), Alt-Ergo (OCamlPro), Why3 (UPSud), and FDCC (Inria).

Industry: OCamlPro and AdaCore will take full advantage of the project to improve their lines of products and services. OCamlPro will improve the current version of Alt-Ergo with reasoning over floating-point arithmetic, hence getting a competitive advantage over other solvers. AdaCore will improve the SPARK tool to take full advantage of the new capabilities of solvers. The expected benefit for the users of the SPARK technology is faster and much more precise results than today on floating-point numbers, modular arithmetic, and more generally nonlinear arithmetic. Finally, traditional industrial partners of CEA and UPSud (including Airbus [57], Dassault [57], Esterel, Athos, EDF [28]) will directly benefit from improvements of the verification tools of the project members.

We will deploy a pro-active communication strategy in order to maximize the project's impact. Besides the **open-source platform**, we will setup an **open-access database of benchmarks** and OCamlPro will develop a **webservice-based version of our solving technology** for demonstration purposes. We also intend to participate to well-established automatic solver competition, for example the SMT competition. Finally, in order to maximize dissemination toward industry, we plan to have a **pool of industrial supporters** following the project (taken among traditional partners of the project members) and providing regular feedback. Especially, we plan a **midterm workshop** with these supporters.

In a **longer term**, we expect that SOPRANO will have a scientific impact beyond verification, since logic, constraints, and automated reasoning have a pervasive place in Computer Science. We can cite for example: compilation, type systems, databases, higher-order provers and mechanized mathematics, operational research, optimization and artificial intelligence. The project will also strengthen the position of France (and of "Plateau de Saclay") in formal methods and high-quality software engineering. Finally, SOPRANO will participate in building better software verification tools (more automated, more widely applicable, more efficient), broadening the use of formal methods in high-quality software engineering and, in the end, improving the overall quality of digital infrastructures.

References

[1] R. Bagnara, M. Carlier, R. Gori, and A. Gotlieb. Symbolic path-oriented test data generation for floating-point programs. In *ICST 2013*.

[2] T. Ball, N. Bjørner, L. de Moura, K. L. McMillan, and M. Veanes. Beyond first-order satisfaction: Fixed points, interpolants, automata and polynomials. In A. Donaldson and D. Parker, editors, *Model Checking Software*, number 7385 in LNCS, pages 1–6. Springer Berlin Heidelberg, Jan. 2012.

[3] T. Ball, B. Cook, V. Levin, and S. K. Rajamani. Slam and static driver verifier: Technology transfer of formal methods inside microsoft. In E. A. Boiten, J. Derrick, and G. Smith, editors, *IFM*, volume 2999 of *LNCS*, pages 1–20. Springer, 2004.

[4] S. Bardin and A. Gotlieb. FDCC: a combined approach for solving constraints over finite domains and arrays. In *Proc. of Constraint Programming, Artifical Intelligence, Operational Research (CPAIOR'12)*, May. 2012.

[5] S. Bardin, P. Herrmann, and F. Perroud. An alternative to SAT-Based approaches for bit-vectors. In *TACAS 2010*.

[6] C. Barrett, R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Splitting on demand in sat modulo theories. In M. Hermann and A. Voronkov, editors, *LPAR*, volume 4246 of *LNCS*, pages 512–526. Springer, 2006.

[7] C. Barrett, A. Stump, and C. Tinelli. The smt-lib standard: Version 2.0. In *Proceedings of the 8th International Workshop on Satisfiability Modulo Theories (Edinburgh, England)*, volume 13, page 14, 2010.

[8] M. S. Belaid, C. Michel, and M. Rueher. Résolution de contraintes sur les nombres à virgule flottante par une approximation sur les nombres réels. In *Sixièmes Journées Francophones de Programmation par Contraintes*, 2010.

[9] M. S. Belaid, C. Michel, and M. Rueher. Boosting local consistency algorithms over floating-point numbers. In M. Milano, editor, *CP*, LNCS, pages 127–140. Springer Berlin Heidelberg, 2012.

[10] N. Beldiceanu. Global constraints as graph properties on a structured network of elementary constraints of the same type. In R. Dechter, editor, *CP 2000*, volume 1894 of *LNCS*, pages 52–66. Springer Berlin Heidelberg, 2000.

[11] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. *Symbolic model checking without BDDs*. Springer, 1999.

[12] B. Blanc, F. Bouquet, A. Gotlieb, B. Jeannet, T. Jéron, B. Legeard, B. Marre, C. Michel, and M. Rueher. The V3F project. In *Proceedings of Workshop on Constraints in Software Testing, Verification and Analysis (CSTVA'06)*, Nantes, France, Sep. 2006.

[13] F. Bobot, S. Conchon, E. Contejean, M. Iguernelala, A. Mahboubi, A. Mebsout, and G. Melquiond. A Simplex-based extension of Fourier-Motzkin for solving linear integer arithmetic. In *IJCAR*. 2012.

[14] F. Bobot, J.-C. Filliâtre, C. Marché, and A. Paskevich. Why3: Shepherd your herd of provers. In *Boogie 2011: First International Workshop on Intermediate Verification Languages*, Wrocław, Poland, August 2011.

[15] S. Boldo and G. Melquiond. Flocq: A unified library for proving floating-point algorithms in Coq. In E. Antelo, D. Hough, and P. Ienne, editors, *Proceedings of the 20th IEEE Symposium on Computer Arithmetic*, pages 243–252, Tübingen, Germany, 2011.

[16] B. Botella, A. Gotlieb, and C. Michel. Symbolic execution of floating-point computations. *STVR*, 2006.

[17] M. Carlier and A. Gotlieb. Filtering by ulp maximum. In *ICTAI*, pages 209–214. IEEE, 2011.

[18] E. M. Clarke, D. Kroening, and F. Lerda. A tool for checking ansi-c programs. In K. Jensen and A. Podelski, editors, *TACAS*, volume 2988 of *LNCS*, pages 168–176. Springer, 2004.

[19] J. W. Coleman and C. B. Jones. A structural proof of the soundness of rely/guarantee rules. *Journal of Logic and Computation*, 17(4):807–841, 2007.

[20] H. Collavizza, M. Rueher, and P. Van Hentenryck. CPBPV: A constraint-programming framework for bounded program verification. *Constraints Journal*, 15(2):238–264, 2010.

[21] S. Conchon, E. Contejean, J. Kanig, and S. Lescuyer. CC(X): Semantic combination of congruence closure with solvable theories. *ENTCS'08*.

[22] S. Conchon, M. Iguernelala, and A. Mebsout. A collaborative framework for non-linear integer arithmetic reasoning in Alt-Ergo. In *SYNASC 2013*.

[23] S. Conchon, G. Melquiond, C. Roux, and M. Iguernelala. Built-in treatment of an axiomatic floating-point theory for SMT solvers. In *SMT workshop*, 2012.

[24] L. Correnson. Qed. Computing what remains to be proved. In J. M. Badger and K. Y. Rozier, editors, *NASA Formal Methods*, volume 8430 of *LNCS*, pages 215–229. Springer, 2014.

[25] S. Cotton. Natural domain SMT: a preliminary assessment. In *Formal Modeling and Analysis of Timed Systems*. 2010.

[26] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In R. M. Graham, M. A. Harrison, and R. Sethi, editors, *POPL*, pages 238–252. ACM, 1977.

[27] P. Cuoq, F. Kirchner, N. Kosmatov, V. Prevosto, J. Signoles, and B. Yakobowski. Frama-C - a software analysis perspective. In G. Eleftherakis, M. Hinchey, and M. Holcombe, editors, *SEFM*, volume 7504 of *LNCS*, pages 233–247. Springer, 2012.

[28] P. Cuoq, F. Kirchner, B. Yakobowski, S. Labbé, N. Thuy, and P. Hilsenkopf. Formal verification of software important to safety using the Frama-C tool suite. In *NPIC*, July 2012.

[29] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.

[30] F. de Dinechin, C. Lauter, and G. Melquiond. Certifying the floating-point implementation of an elementary function using Gappa. *Transactions on Computers*, 60(2):242–253, 2011.

[31] L. de Moura and N. Bjørner. Model-based theory combination. *Electr. Notes Theor. Comput. Sci.*, 198(2):37–49, 2008.

[32] L. de Moura and D. Jovanović. A model-constructing satisfiability calculus. In *VMCAI 2013*.

[33] T. Denmat, A. Gotlieb, and M. Ducassé. An abstract interpretation based combinator for modelling while loops in constraint programming. In *CP 2007*.

[34] T. Denmat, A. Gotlieb, and M. Ducasse. Improving constraint-based testing with dynamic linear relaxations. In *18th IEEE International Symposium on Software Reliability Engineering (ISSRE' 2007)*, Trollhättan, Sweden, Nov. 2007.

[35] E. W. Dijkstra. *A discipline of programming*, volume 1. Prentice-Hall Englewood Cliffs, 1976.

[36] C. Dross, S. Conchon, J. Kanig, and A. Paskevich. Reasoning with triggers. In P. Fontaine and A. Goel, editors, *SMT@IJCAR*, volume 20 of *EPiC Series*, pages 22–31. EasyChair, 2012.

[37] V. D'Silva, L. Haller, and D. Kroening. Abstract satisfaction. In *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '14, page 139–150, New York, NY, USA, 2014. ACM.

[38] H. Ganzinger, G. Hagen, R. Nieuwenhuis, A. Oliveras, and C. Tinelli. DPLL(T): fast decision procedures. In R. Alur and D. A. Peled, editors, *Computer Aided Verification*, number 3114 in LNCS, pages 175–188. Springer Berlin Heidelberg, Jan. 2004.

[39] P. Godefroid. Test generation using symbolic execution. In D. D'Souza, T. Kavitha, and J. Radhakrishnan, editors, *FSTTCS*, volume 18 of *LIPIcs*, pages 24–33. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.

[40] A. Gotlieb. Euclide: A constraint-based testing framework for critical C programs. In *ICST*, pages 151–160. IEEE Computer Society, 2009.

[41] A. Gotlieb. TCAS software verification using constraint programming. *The Knowledge Engineering Review*, 27(3):343–360, Sep. 2012.

[42] A. Gotlieb, B. Botella, and M. Rueher. Automatic test data generation using constraint solving techniques. In *Proc. of Int. Symp. on Soft. Testing and Analysis (ISSTA'98)*, pages 53–62, 1998.

[43] A. Gotlieb, B. Marre, and M. Leconte. Constraint solving on modular integers. In *the 9th Int. Workshop on Constraint Modelling and Reformulation (ModRef'10)*, Sept. 2010.

[44] L. Haller, A. Griggio, M. Brain, and D. Kroening. Deciding floating-point logic with systematic abstraction. In *FMCAD*, 2012.

[45] T. A. Henzinger, R. Jhala, R. Majumdar, and G. Sutre. Lazy abstraction. In J. Launchbury and J. C. Mitchell, editors, *POPL*, pages 58–70. ACM, 2002.

[46] J. M. Howe and A. King. Specialising finite domain programs using polyhedra. In A. M. D. Moreira and S. Demeyer, editors, *ECOOP Workshops*, volume 1743 of *LNCS*, pages 258–259. Springer, 1999.

[47] D. Jovanović, C. Barrett, and L. de Moura. The design and implementation of the model constructing satisfiability calculus. In *Proceedings of 13th International Conference on Formal Methods in Computer-Aided Design, FMCAD 2013*, Porland, Oregon, USA, 2013.

[48] D. Jovanović and L. de Moura. Solving non-linear arithmetic. In *IJCAR 2012*.

[49] D. Jovanović and L. de Moura. Cutting to the chase solving linear integer arithmetic. In N. Bjørner and V. Sofronie-Stokkermans, editors, *Automated Deduction – CADE-23*, volume 6803, pages 338–353. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[50] M. Leconte and B. Berstel. Extending a CP solver with congruences as domains for program verification. In B. Blanc, A. Gotlieb, and C. Michel, editors, *Proceedings of the 1st workshop on Constraints in Software Testing, Verification and Analysis (CSTVA '06)*, pages 22–33, Nantes, France, 2006. IEEE Computer Society Press. Available at `http://www.irisa.fr/manifestations/2006/CSTVA06/`.

[51] B. Marre. Toward automatic test data set selection using algebraic specifications and logic programming. In K. Furukawa, editor, *Proc. of the Eight Int. Conf. on Logic Prog. (ICLP'91)*, pages 202–219, Paris, Jun. 1991. MIT Press.

[52] B. Marre and A. Arnould. Test sequences generation from lustre descriptions: Gatel. In *ASE 2000*.

[53] B. Marre and C. Michel. Improving the floating point addition and subtraction constraints. In *CP 2010*.

[54] K. L. McMillan. Lazy abstraction with interpolants. In T. Ball and R. B. Jones, editors, *CAV*, volume 4144 of *LNCS*, pages 123–136. Springer, 2006.

[55] M. R. Mohammed Said Belaid, Claude Michel. Approximating floating-point operations to verify numerical programs. In *14th GAMM-IMACS International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics*, 2010.

[56] D. Monniaux. Automatic modular abstractions for linear constraints. In Z. Shao and B. C. Pierce, editors, *POPL*, pages 140–151. ACM, 2009.

[57] Y. Moy, E. Ledinot, H. Delseny, V. Wiels, and B. Monate. Testing or formal verification: Do-178c alternatives and industrial experience. *IEEE Software*, 30(3):50–57, 2013.

[58] J. A. Navas, P. Schachte, H. Søndergaard, and P. J. Stuckey. Signedness-agnostic program analysis: Precise integer bounds for low-level code. In R. Jhala and A. Igarashi, editors, *APLAS 2012: Proceedings of the 10th Asian Symposium on Programming Languages and Systems*, volume 7705 of *Lecture Notes in Computer Science*, pages 115–130. Springer, 2012.

[59] G. Nelson and D. C. Oppen. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 1(2):245–257, 1979.

[60] R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Solving SAT and SAT modulo theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). *J. ACM*, 53(6):937–977, Nov. 2006.

[61] O. Ohrimenko, P. J. Stuckey, and M. Codish. Propagation via lazy clause generation. *Constraints*, 14(3):357–391, Sept. 2009.

[62] M. Pelleau, A. Miné, C. Truchet, and F. Benhamou. A constraint solver based on abstract domains. In R. Giacobazzi, J. Berdine, and I. Mastroeni, editors, *VMCAI*, volume 7737 of *LNCS*, pages 434–454. Springer, 2013.

[63] J. D. Scott, P. Flener, and J. Pearson. Bounded strings for constraint programming. In *Tools with Artificial Intelligence (ICTAI 2013)*. IEEE Computer Society, 2013.

[64] L. Zhang, C. F. Madigan, M. H. Moskewicz, and S. Malik. Efficient conflict driven learning in a boolean satisfiability solver. In *Proceedings of the 2001 IEEE/ACM international conference on Computer-aided design*, pages 279–285. IEEE Press, 2001.