Seventh Framework Programme
"Ideas" specific programme

European Research Council

Grant agreement for Starting Grant

Project acronym:	**STATOR**
Project full title:	STATic analysis with ORiginal methods
Grant agreement no:	306595
Duration:	60 months
Principal investigator:	**David Monniaux**
Host institution:	Université Joseph Fourier (Grenoble, France)

Annex I — "Description of Work"

Date of preparation of Annex I: October 24, 2012

1 The Principal Investigator

1.1 Curriculum Vitae

1.1.1 Current positions

- CNRS (national centre for scientific research): *directeur de recherche de deuxième classe* (senior researcher, second class), VERIMAG laboratory, Grenoble, France (laboratory hosted by Université Joseph Fourier).
- École polytechnique, Paris, a prestigious national institution: *professeur chargé de cours d'exercice incomplet* (associate professor, part-time: no research work, approx. 90 hr of classes taught per year).

1.1.2 Past positions

2003–2009 École polytechnique, *chargé d'enseignement d'exercice incomplet* (assistant professor, part-time)

2002–2007 CNRS, chargé de recherche, in P. Cousot's group at ENS-Paris; mostly worked on Astrée.

2002 Short post-doc at SRI International (California) on the SAL project: analysis of partial differential equations by means of polynomial systems.

2001–2002 École normale supérieure, Paris: postdoctoral research assistant in P. Cousot's group; beginning of work the Astrée static analyser.

Static analysis of software consists in automatically inferring properties about all its possible executions. It is distinct from testing, sometimes called dynamic analysis, which considers a limited number of executions, and thus, except for very small systems (small finite state), cannot guarantee the absence of errors. Some static analyses are *unsound*, in the sense that if they list no possible incorrect executions, there is no guarantee that such executions cannot happen. These are sometimes called bug-finding tools, because such tools are designed to provide a limited number of warnings that in practice have a high probability of being actual bugs; examples of such tools include the commercial Coverity product. Airbus needed to demonstrate the safety of their systems to certification authorities, which cannot be done with bug-finding tools.

In contrast, sound analysis provides an exhaustive list of the possible errors of the class that they are designed to detect: in Astrée, division by zero, array access out of bounds, arithmetic overflow, user-specified assertion failure.

1999–2001 Université Paris-9 Dauphine: research and teaching assistant

1997,1998 Summer internships at SRI International with John Rusby and Natarajan Shankar: work on the analysis of cryptographic protocols, leading to publications [CSFW'99, SAS'99, JTIT'02]. The SAS'99 publication is still highly cited in the field of analysis of cryptographic protocols.

1995–1999 École normale supérieure de Lyon: *élève normalien*, a highly selective salaried student position.

1.1.3 Academic degrees

2009 Habilitation to direct research (computer science), Université Joseph Fourier (Grenoble)

1999–2001 Ph. D. in computer science, Université Paris-9 Dauphine, summa cum laude, on the static analysis of probabilistic programs; work conducted at the computer science laboratory of the École normale supérieure in Paris, with Patrick Cousot as advisor.

1999 *Agrégation* in mathematics (a national competitive exam)
1998 Research master in computer science (rank 1), Université Paris-7 Denis Diderot
1998 *Magistère* in computer science, École normale supérieure de Lyon
1995 Selected by competitive exam (national rank 24) for École normale supérieure de Lyon (after 2 years of college)

1.1.4 Projects and funding

There is and there will be no funding overlap with the ERC grant requested and any other source of funding for the same activities and costs that are foreseen in this project.

Current projects and funding

2012–2015 French ANR project VERASCO: A 5-site project spread between Paris, Rennes and Grenoble, with total funding approx. 1 million€; I manage VERIMAG's part (164,000€). Project VERASCO tackles an ambitious objective: the integration of a simple static analyser into the formally certified compiler CompCert, as well as the formal certification of this analyser within the Coq system.
A crucial difference with STATOR is that VERASCO aims at writing formally proved implementations of existing techniques (e.g. polyhedral analysis) while STATOR tackles the creation of brand new techniques; however some questions are common, such as how to make abstract domains interact in a clean and modular fashion.

Past projects

2009–2012 French ANR project ASOPT: A 5-site project spread between Paris and Grenoble with total funding approx. 1 million€. I was the principal investigator for the VERIMAG site, managing a budget of 136,000€. ASOPT investigates the relationships between static analysis one the one hand and mathematical programming and other techniques from mathematics on the other hand; my main contributions in this project, in addition to managing it for VERIMAG, are new quantifier elimination algorithms (LPAR'08, CAV'10), an application of floating-point linear programming to exact arithmetic decision procedures (CAV'09), an algorithm for the exact computation of transfer functions (POPL'09, LMCS'10), and an algorithm for the exact computation of strongest invariants in certain lattices, with a Π_2^p-complexity result (ESOP'11).
2004–2006 French project APRON: Participant. This project considered static analysis of numerical constraints; I contributed e.g. results on numerical filters (CAV'05).
2005–2007 French RNTL project Thésée: Participant. A follow-up to Astrée.
2002–2004 French RNTL project Astrée: Participant. This project provided funding for part of the development of the Astrée static analyser.
2000–2002 European project DAEDALUS: Participant.

1.1.5 Supervision of students and post-docs

PhD students: Julien Le Guen (2009–2012), Julien Henry (2011–2014), Alexis Fouilhé (2012–2015)
Postdocs: Thomas Gawlitza (2009–2010), Diego Caminha (2011–2012)

Note: In France, in order to be (at least officially) Ph.D advisor for a student, one has to have an habilitation to direct research, a diploma granted after an established post-PhD research record; in this case the habilitation was granted in 2009.

I also welcome students on internships from masters programs, as well as undergraduates from e.g. IIT-Indore, ENSIMAG, ENS-Paris, ENS-Lyon and ENS-Cachan.

1.1.6 Invited talks, lecturing at summer schools

2010 Invited speaker at the NSF-sponsored workshop on Usable Verification, held at Microsoft Research

2010 Invited speaker at the International mini-conference in EECS, Tōhoku University, Sendai, Japan

2011 Lecturing at formal methods summer school, sponsored by the US National Science Foundation, Atherton, California (other lecturers included Ken McMillan, Leonardo de Moura, Natarajan Shankar, etc.)

2011 Lecturing at the EPFL (*École polytechnique fédérale de Lausanne*, Switzerland) Summer research institute (other lecturers included J Strother Moore, Tony Hoare, Sharad Malik, Joseph Halpern, etc.)

2012 Lecturing at the *Verification Technology, Systems & Applications* (VTSA) summer school, 2012, at Max Planck Institut für Informatik, Saarbrücken, Saarland, Germany

1.1.7 Program committees

International conferences: VMCAI 2005, CAV 2010, PEPM 2010, ESOP 2012, SAS 2012
Workshops: AFM 2007, AFM 2008, AFM 2009, SMT 2010, NSV-3, CA-CAV 2011, ACCA 2011 (chair), WING 2012

1.1.8 Important publications

My early article on the analysis of cryptographic protocols (SAS'99) is highly cited (137 citations), since it introduced abstract interpretation with automata techniques in this field. The book *Validation of stochastic systems: a guide to current research* (2004), p. 424, describes the papers on the analysis of probabilistic programs published during my thesis as "mandatory reading on abstraction of Markov decision processes". Several articles describing the Astrée system and written by the Astrée team (2002 book chapter, PLDI'03...) are highly cited as examples of a industrial-strength static analysis system as well as for astute techniques for augmenting precision while keeping costs reasonable.

1.2 Early achievements track-record

6 single-authored articles in international peer-reviewed journals; 23 articles in proceedings of international peer-reviewed conferences (15 as single author), including 2 in POPL, 1 in PLDI, 3 in ESOP, 3 in CAV, for the most prestigious conferences; 2 in proceedings of international workshops. Citation numbers from Google Scholar, H-index=16; total 1115 citations. Full list of publications at http://www-verimag.imag.fr/~monniaux/biblio/David_Monniaux.html or at DBLP.

1.2.1 Industrialised software developments

Astrée,[1] static analyser for proving that software cannot encounter runtime errors (particularly critical embedded software for aerospace applications). I designed alone the first version of the system, and then developed it along with B. Blanchet, P. and R. Cousot, L. Mauborgne, A. Miné, X. Rival. Now distributed through Absint GmbH[2] from Saarbrücken, Germany, Astrée has been licensed to leading semiconductor, automotive and avionic companies, including Airbus.

1.2.2 Articles in international peer-reviewed journals

D. Monniaux. *Analysis of cryptographic protocols using logics of belief: an overview.* Journal of Telecommunications and Information Technology, 4:57–67, 2002. 3 citations.
D. Monniaux. *Abstracting cryptographic protocols with tree automata.* Science of Computer Programming, 47(2-3):177–202, 2003. Journal version of [SAS'99]. 33 citations.
D. Monniaux. *Abstract interpretation of programs as Markov decision processes.* Science of Computer Programming, 58(1-2):179–205, October 2005. Journal version of [SAS'03]. 22 citations
D. Monniaux. *The pitfalls of verifying floating-point computations.* TOPLAS, 30(3):12, May 2008. 57 citations. *This article summarises fine points of floating-point implementations, creating difficulties for the soundness of program proofs or static analyses, as well as solutions.*
D. Monniaux. *A minimalistic look at widening operators.* Higher order and symbolic computation, 22(2):145-154, December 2009. 4 citations
D. Monniaux. *Automatic modular abstractions for template numerical constraints.* Logical Methods in Computer Science, June 2010. 5 citations

1.2.3 Book chapters

B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. *Design and implementation of a special-purpose static program analyzer for safety-critical real-time embedded software.* In *The Essence of Computation: Complexity, Analysis, Transformation*, LNCS 2566, pp. 85–108. Springer, 2002. 119 citations. *Describes an early version of the Astrée analyser. As for all articles in the Astrée project, authors are listed alphabetically.*

1.2.4 Articles in international peer-reviewed conference proceedings

Due to lack of space, only the most important are listed; see others online.

D. Monniaux. *Decision procedures for the analysis of cryptographic protocols by logics of belief.* In 12th Computer Security Foundations Workshop, pages 44–54. IEEE, 1999. 19 citations

D. Monniaux. *Abstracting cryptographic protocols with tree automata.* In Sixth International Static Analysis Symposium (SAS'99), LNCS 1694, pp. 149–163. Springer, 1999. 137 citations. *This article, from early graduate research work, introduced abstractions using regular tree languages in the analysis of cryptographic protocols in the Dolev-Yao model, a direction followed by many other researchers in this field.*

D. Monniaux. *Abstract interpretation of probabilistic semantics.* In Seventh International Static Analysis Symposium (SAS'00), LNCS 1824, pp. 322–339. Springer, 2000. 59 citations *Introduced a notion of forward static analysis, by abstraction of sets of measures, on probabilistic programs.*

[1] http://www.astree.ens.fr/
[2] http://www.absint.de/astree/

D. Monniaux. *An abstract Monte-Carlo method for the analysis of probabilistic programs (extended abstract).* In 28th Symposium on Principles of Programming Languages (POPL '01), pp. 93–101. ACM, 2001. 34 citations *Replaced exact computations in analysis algorithms by Monte-Carlo (randomized) algorithms.*

D. Monniaux. *Backwards abstract interpretation of probabilistic programs.* In European Symposium on Programming Languages and Systems (ESOP '01),LNCS 2028, pp. 367–382. Springer, 2001. 16 citations *Introduced backward analysis of probabilistic programs by bounding sets of measurable functions.*

D. Monniaux. *An abstract analysis of the probabilistic termination of programs.* In 8th International Static Analysis Symposium (SAS'01), LNCS 2126, pp. 111–126. Springer, 2001. 12 citations

B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. *A static analyzer for large safety-critical software.* In PLDI, pages 196–207. ACM, 2003. 304 citations *The main description of the Astrée static analyser. As for all articles in the Astrée project, authors are listed alphabetically.*

D. Monniaux. *Abstract interpretation of programs as Markov decision processes.* In Static Analysis Symposium (SAS'03), LNCS 2694, pp. 237–254. Springer, 2003. 22 citations *Unifying view of probabilistic and nondeterministic programs as MDPs, and their analysis.*

D. Monniaux. *Compositional analysis of floating-point linear numerical filters.* In Computer-aided verification (CAV'05), LNCS 3576, pp. 199–212. Springer, 2005. 8 citations *General analysis of floating-point feedback linear filters.*

P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. *The ASTRÉE analyzer.* In ESOP, LNCS 3444, pp. 21–30, Springer, 2005. 142 citations *A later description of the Astrée system. As for all articles in the Astrée project, authors are listed alphabetically.*

D. Monniaux. *Optimal abstraction on real-valued programs.* In Static analysis (SAS '07), LNCS 4634, pp. 104–120. Springer, 2007. 6 citations *Strongest invariants in certain abstract domains can be computed exactly by reduction to quantifier elimination.*

P. Cousot, R. Cousot, J. Feret, A. Miné, L. Mauborgne, D. Monniaux, and X. Rival. *Varieties of static analyzers: A comparison with ASTRÉE.* In Theoretical Aspects of Software Engineering (TASE'07). IEEE, 2007. 22 citations *Authors listed alphabetically.*

D. Monniaux. *Verification of device drivers and intelligent controllers: a case study.* In EMSOFT, pp. 30–36. ACM & IEEE, 2007. 9 citations *Formalisation of a USB OHCI intelligent controller, and static analysis of the combination of driver and controller.*

P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. *Combination of abstractions in the Astrée static analyzer.* In Advances in Computer Science - ASIAN 2006. Secure Software and Related Issues, LNCS 4435, pages 272–300. Springer, 2008. 39 citations

D. Monniaux. *A quantifier elimination algorithm for linear real arithmetic.* In LPAR (Logic for Programming Artificial Intelligence and Reasoning), LNCS 5330, pp. 243–257. Springer, 2008. 25 citations

D. Monniaux. *On using floating-point computations to help an exact linear arithmetic decision procedure.* In Computer-aided verification (CAV), LNCS 5643, pp. 570–583. Springer, 2009. 6 citations *On replacing the exact rational simplex algorithm in DPLL(T) by a floating-point one.*

D. Monniaux. *Automatic modular abstractions for linear constraints.* In POPL (Principles of programming languages). ACM, 2009. 25 citations

D. Monniaux. *Quantifier elimination by lazy model enumeration.* In Computer-aided verification (CAV), LNCS 6174, pp. 585–599. Springer, 2010. 9 citations *Ideas of lazy generation of constraints brought into quantifier elimination.*

T. Gawlitza and D. Monniaux. *Improving strategies via SMT solving.* In ESOP, LNCS 6602, pp. 236–255. Springer, 2011. *Exact computation of strongest invariants in certain domains by policy iteration guided by SMT-solving; proof of* Π_p^2*-completeness.*

D. Monniaux and P. Corbineau. *On the generation of Positivstellensatz witnesses in degenerate cases.* In Interactive Theorem Proving (ITP), LNCS 6898, pp. 249–264. Springer, 2011 *Generation of easily checkable unsatisfiability witnesses for systems of polynomial constraints.*

D. Monniaux and L. Gonnord. *Using bounded model checking to focus fixpoint iterations.* In Static analysis (SAS), LNCS 6887, pp. 369–385. Springer, 2011. *Iterations in classical abstract domains guided by SMT-solving.*

2 State of the art and objectives

2.1 Scientific and societal challenge

Static analysis tools are already suitable for proving the absence of certain classes of bugs in industrial applications. Among notable industrial examples are the PolySpace Verifier[3], Microsoft's Device driver verifier[4], and Astrée, a project in which I was one of the main developers in 2002–2007[5]. Interestingly, these three systems originated from advanced research projects, and not from evolution of compilers or other commonly used development tools.

Despite all their successes, these three tools work well on limited classes of software, but they tend to be slow and to produce too many false alarms when applied in other contexts. A common caricature is that analyzers based on accelerated Kleene iterations (see below) will terminate but produce false alarms, while those based on predicate abstraction will run out of time and memory due to excessive CEGAR (see below). This limits the applicability of these techniques in most application areas.

At the same time, there is growing concern about the safety of embedded software, e.g. in medical devices (the US Food and Drugs administration now mandates the use of static analysis for the software driving certain devices, such as infusion pumps[6]), not to mention the many security bugs in networked applications, which now tends to mean most applications. Clearly, there is societal interest in better automated analysis methods, but we do not envision larger use unless scientific breakthroughs are made. **The purpose of the STATOR project is to take up this challenge by providing original and innovative techniques.**

STATOR involves both theoretical work (design of new algorithms and techniques) as well as implementation and experimental work; we have thus divided the work packages into Tx (theory) and Iy (implementation and experiments).

[3]PolySpace was designed by the late Alain Deutsch, originally from INRIA, then founder of the PolySpace start-up company, which has since then been bought by The Mathworks.

[4]Originally developed as the SLAM project in Tom Ball's group at Microsoft Research, now integrated into a commercial product.

[5]Astrée originated from a CNRS / ENS-Paris research project. It is now sold through Absint GmbH from Saarbrücken, Germany. I was the architect of the first version, and one of the main developers and main contributors of scientific solutions for subsequent versions.

[6]http://preview.tinyurl.com/5wlaxht

2.2 State of the art

Let us now give a short introduction to static analysis, with a few simple examples that illustrate some of the limitations of current methods and our insights about what will be done within STATOR.

Static analysis of software consists in automatically inferring properties about all its possible executions. It is distinguished from testing, sometimes called dynamic analysis, which considers a limited number of executions, and thus, except for very small systems (small finite state or small bounded execution time), cannot guarantee the absence of errors. Some static analyses are *unsound*, in the sense that if they list no possible incorrect executions, there is no guarantee that such executions cannot happen. These are sometimes called bug-finding tools, because such tools are designed to provide a limited number of warnings that in practice have a high probability of being actual bugs; examples of such tools include the commercial Coverity product.[7] In contrast, sound analysis methods provide an exhaustive list of the possible errors of the class that they are designed to detect, e.g. division by zero, array access out of bounds, arithmetic overflow, user-specified assertion failure. **In this project, we shall focus on sound methods, though some results may also apply to bug-finding methods.**

It is a well-known fact that any terminating, automatic and sound analysis is bound to sometimes introduce false alarms, that is, warnings about problems that cannot happen in any program execution.[8] A major issue in static analysis is to reduce the number of false alarms, and more generally improve the precision of the analysis, while keeping time and memory costs low.

Schematically, there exist two families and traditions of techniques for proving properties of programs. One tradition is based on Floyd / Hoare logic: inductive loop invariants (and/or function specifications) are provided by the user, as well as proofs that they are correct. For instance, for a program

```
int t[100];
for(int i=0; i<10; i++) { ... }
```

the user has to provide an annotation for an inductive loop invariant, for instance $0 \leq i \leq 10$; this invariant entails that no array access occurs out of bounds.

A formula I defines an inductive relation for a transition relation τ and a set of initial states defined by predicate ι if and only if $\forall\sigma\ \iota(x) \Rightarrow I(x)$ and $\forall\sigma,\sigma'\ I(x) \wedge \tau(x,x') \Rightarrow I(x')$ where σ,σ' denote program states; in other words, if and only if $\iota(\sigma) \wedge \neg I(\sigma)$ and $I(\sigma) \wedge \tau(\sigma,\sigma') \wedge \neg I(\sigma')$ are unsatisfiable. It follows by induction that I is always true along a program execution. For a sufficiently restricted language of formulas for describing invariants, initial conditions and transition relations (thus a sufficiently restricted class of program constructs), for instance linear integer arithmetic (also known as Presburger arithmetic), it is possible to use *decision procedures* [9, 30] in order to test whether such formulas truly are unsatisfiable. When the formulas are propositional and quantifier-free, this problem boils down to SAT-solving, the canonical NP-complete problem. SAT-solving has been extended to quantifier-free formulas over linear integer and real arithmetic, plus uninterpreted functions and other extensions, as *satisfiability modulo theory* (SMT), generally implemented by DPLL(T) techniques [14, 24, 30]. Most modern decision procedures are variants of

[7]http://www.coverity.com/

[8]In recursion theory. this is known as Rice's theorem: given any property on program input-outputs that is neither uniformly true nor false, there is no algorithm that decides this property on the program code source. Basically, solving such problems entails solving Turing's halting problem.

SMT-solving. SMT-solvers have also been extended to semidecision procedures for undecidable logics (such as quantified arithmetic + uninterpreted functions).

Modern tools based on Floyd / Hoare logic, such as the Frama-C suite, therefore check the verification conditions induced by invariant annotations by calling SMT-solvers, with manual proofs as a possibility for more complex properties. Because of the tediousness of providing invariants, such tools often have some form of invariant inference; for instance some of the tools operating over Microsoft Research's Boogie intermediate representation will try to construct invariants out of plausible predicates extracted from the source code. Stronger forms of automation are however needed in but the most simple cases.

The second tradition came from data-flow analysis, which propagates properties along the program, with control-flow joins corresponding to meet operators in a lattice (a property is ensured to be true at the end of a test if it is true from both branches). Similarly, interval analysis propagates interval bounds on all numerical variables: comparisons restrict such intervals, while control-flow joins correspond to convex hull of intervals (if from the left branch of the test, we know $x \in [l_1, h_1]$ and from the right branch $x \in [l_2, h_2]$, then $x \in [\min(l_1, l_2), \max(h_1, h_2)]$ at the join). Both approaches are generalized in *abstract interpretation.*[9]

Static analysis by abstract interpretation, in a nutshell, finds inductive invariants by restricting the search to an *abstract domain* of properties, generally a *lattice* (a partially ordered set with least upper bound $\sqcup$ and greatest lower bound $\sqcap$ operators). For instance, when considering numerical variables, one may look for loop invariants given by convex polyhedra, or equivalently systems of linear (in)equalities; in this case, $\sqcup$ is convex hull and $\sqcap$ is intersection.

There exist many abstract domains in the literature; for instance, for numerical values, one may restrict the domain of polyhedra to those with predefined outgoing normal vectors for the faces and get *template linear constraint domains* [41], let these predefined directions be $\pm x \pm y$ and $\pm x$ for any couple of variables x, y and get *octagons* [32, 35] (we mean that all constraints expressed are of the form $\pm x \pm y \leq C$ where C is a constant), further restrict them to $x - y$ and $\pm x$ and get *difference bound matrices*, even restrict them to $\pm x$ and obtain intervals. More restricted domains are less expressive, but typically have lower time and memory costs; furthermore, due to the non-monotonicity of analysis with widenings (see below), less precise domains may occasionally lead to more precise results. Choosing a good abstract domain for a class of problems and programs is thus delicate.

Classically, abstract interpretation works by propagating abstract information along the control-flow graph, either forward (obtaining a super-set of the states reachable from program start) or backward (obtaining a super-set of the states co-reachable from a property of interest), or a combination thereof. Consider for instance the following program, where choose(a,b) denotes a nondeterministically chosen integer within $[a, b]$:

```
1 x = choose(10, 20);
2 y = choose(-1, 1);
3 if (choose(0, 1) != 0) {
4   z = y + x;
5 } else {
6   z = y - x;
```

[9]A little note on notation: the convention in abstract interpretation is to order the abstract lattice according to the set of states being represented, $a \sqsubseteq b$ meaning that b represents more states than a, while in data-flow analysis it tends to be ordered according to the set of true properties, thus exactly in the inverse order.

```
7 }
8 assert(z != 0);
```

With forward analysis, the “then” branch yields z $\in [9, 21]$ and the “else” branch z $\in [-21, -9]$. The convex hull $[-21, 21]$ of these two intervals is computed at line 8. Since this interval contains 0, the analyzer fails to prove that the assertion z $\neq 0$ truly holds. With backward analysis, we start from the property of interest, that is, z $= 0$ at line 8, and then move backwards into both branches of the if-then-else; but since we know nothing about x and y, interval analysis fails to propagate further information backwards. Can we do better?

- We could use logic-based methods: in this case, SMT-solving would show automatically that $10 \leq x \leq 20 \wedge -1 \leq y \leq 1 \wedge (z = y + x \vee z = y - x) \wedge z = 0$ has no solution. This, however, would not extend easily to cases where the postcondition $z \neq 0$ does not appear directly, but for instance is the precondition of a loop further down the program.

- We could use a more expressive abstract domain, such as convex polyhedra: but for this example, any domain that expresses only convex properties will fail to prove z $\neq 0$.

- We could run a forward interval analysis (which obtains z $\in [9, 21]$ and z $\in [-21, -9]$ respectively in the test branches), then run a backward analysis from z $= 0$, and take intersections between the results of the forward and backward: thus both branches get $\emptyset$. What is obtained at each program point is a superset of the states reachable from program start and co-reachable from the property of interest z $= 0$. Since it is $\emptyset$, we conclude that z $= 0$ is unreachable. This illustrates another idea that we shall apply: **a combination of several simple analyses can perform as well or better than more complex analyses**, a fact already acknowledged within the Astrée project [6, 7].

- We could analyse `assert(z != 0);` in the context of both branches, thus differently depending on the history of the computation trace; this is known as *trace partitioning* [40]. The number of contexts to track is in the worst case exponential in the number of tests, thus it has to be heuristically limited.

  We have developed a more systematic approach by having the relevant traces selected by SMT-solving [37]. This combination of abstract interpretation and logical methods (SMT-solving) has the best of both worlds: it can infer loop invariants, and it deals with traces only as needed, as opposed to trace partitioning, with may carry information about useless contexts. **Such combinations of exact logic-based methods with abstraction are a major feature of STATOR.** An additional benefit of this method is that it performs abstract computations over long sequences of straight-line code, which has benefits for precision.

Let us illustrate this last remark. If we apply forward interval analysis to this program

```
int x, y, z;
x = choose(0, 1);
y = x;
z = x-y;
```

then we get $x \in [0, 1]$, $y \in [0, 1]$, $z \in [-1, 1]$. This is because we proceeded line by line. Had we used convex polyhedra instead, we would have kept the $y = x$ relationship and obtained $z = 0$; but there is a simpler alternative: to perform lines 3 and 4 in one step $(x, y, z) \mapsto (x, x, x - x)$. This *symbolic propagation* and simplification [33] may be precomputed, for instance by translating the program to single static assignment (SSA) form [31]. Another important idea: **simple abstractions can be made more precise by suitable program transformations, simplifications, and pre-analyses**. For instance, one can get better results by analysing increasing subsets of the program than the full program directly, either with respect to statements [20] or variables [31, 38].

Let us now see how abstract interpretation deals with loops, on the simple example **for**(**int** i=0; i<100000; i++) {}. Interval analysis starts from the initial state $i \in [0, 0]$, applies one loop iteration and adds back the initial states to obtain $[0, 1]$, does it again and obtains $[0, 2]$. It is easy to see that this naive method will converge only after 100000 iterations, which is impractical. *Widening* operators extrapolate from the initial iterations and guess a larger element; in this example, since the lower bound 0 is stable and the higher bound moves, classical widening will try $[0, +\infty)$. This interval is an inductive invariant for this loop, but obviously not the best one we could hope for. Once we have an inductive invariant, we obtain another one by applying the loop iteration and adding back the initial states once more (*narrowing* step); in this example, we obtain $[0, 100000]$ which is the strongest invariant and thus the best interval we could get.

Yet, on more complex examples, the widening and narrowing technique may produce invariants that are too weak to prove the desired properties. For instance, on a variant of the above example, classical widening and narrowing produces $[0, +\infty)$: **for**(**int** i=0; i!=100000; i++) {} (the != operator foils the narrowing operation). The same applies to

```
int i=0;
while (true) {
  if (choose(0, 1)) {
    i=i+1;
    if (i >= 100000) i=0;
  }
}
```

Of course, for such simple examples, there exist classical workarounds. We could for instance detect syntactically the most common loop structures, as many compilers do, and immediately compute the invariants, which will work our simplest examples only. We could replace the classical widening by a variant, such as *widening with thresholds* or *limited widening* [7, 22], which basically tracks for each variable the set of con/stants to which it is compared or assigned, propagates this set through assignments, and widens to the next "magical constant" instead of to $\pm\infty$. Widening design is somewhat of a black art and many combinations are possible; for instance there exist many variations on the widening operators on convex polyhedra [2].

An especially counterintuitive characteristic of analysis with widening operators is that the results are non-monotonic in the specification of the system: that is, replacing parts of the system by more imprecise specifications may ultimately produce stronger invariants and fewer false alarms. For instance, in the previous example, by replacing **int** i=0; ($i \in [0, 0]$) by **int** i=choose(0,99999); ($i \in [0, 99999]$), a larger set of initial states, we obtain a stronger invariant $i \in [0, 99999]$ instead of $i \in [0, +\infty)$. Another example is that in polyhedral analysis, removing some of the variables and replacing them by "I don't know" may lead to stronger invariants [38].

Because of the pitfalls of Kleene iterations accelerated with widening operators, alternate techniques have been proposed in the recent years: direct computation of the strongest inductive invariant in the abstract domain [1, 17, 21, 36], upward policy iteration [17, 18], downward policy iteration [11, 16]... **The STATOR project will work on refining such advanced iteration techniques.**

In comparison, predicate abstraction works on finite models. For a given family $\pi_1, \ldots, \pi_n$ of predicates, the state σ is abstracted as the vector $a = (\pi_1(\sigma), \dots, \pi_n(\sigma))$ of the truth values of these predicates; thus a program has at most 2^n abstract states.[10] A transition is put between two abstract states a and a' if there is at least one couple (σ, σ') of concrete states, respectively abstracted by a and a', such that the concrete system has a transition from σ to σ'. The set of reachable states in the abstract system is computed as in model-checking, and yields an inductive invariant for the concrete systems. If the family $\pi_1, \dots, \pi_n$ is badly chosen, this inductive invariant may be insufficient to prove the concrete property even if this property is true: that is, there exist an abstract counterexample, meaning a trace of abstract states from initial states to "bad" states, but there is no actual concrete execution trace that corresponds to it. *Counterexample-guided abstraction refinement* (CEGAR) attempts finding more useful predicates. It is often implemented by extracting predicates from a Craig interpolant of the proof of absence of a concrete counterexample. Again, it is worth pointing that such abstraction, checking and interpolation techniques make heavy use of decision procedures. **The STATOR project will also use decision procedures, but instead of pure predicate abstraction will combine the use of these procedures with abstract interpretation techniques such as those discussed above.**

2.3 Theoretical objectives

In the last years, new techniques were proposed as alternatives to the usual Kleene iterations with widenings: guided iterations [20], min-policy iteration [11]; max-policy iteration [18]; modular invariants by reduction to quantifier elimination [36]; path-focusing [37]; path max-policies [17]; reduction to numerical programming [21]. Some of these techniques are highly enticing in theory: for instance, path max-policy iteration, which I introduced together with T. Gawlitza, computes least invariants in a way that is optimal from a certain complexity point of view. More generally, we witness a convergence of techniques based on abstract interpretation with techniques based on logic: while some have proposed the integration of SMT solving in abstract interpretation analyzers [17, 37], others have recast the DPLL [23] and Nelson-Oppen [12] procedures into the abstract interpretation framework; and relations are being sketched between the widening (extrapolation) operators and Craig interpolation techniques. Finally, early results in my group have shown that cheap results from abstraction can considerably speed up precise, but expensive, logical methods.

The main scientific axis of the proposal is the development of original techniques for invariant generation, breaking out of the traditional accelerated Kleene iteration scheme. A key idea is to leverage the good aspects from both abstract interpretation and logical methods: abstract interpretation is good at generalising partial results while maintaining compact representation, while exact logical methods, such as SMT-solving, are good at avoiding or recovering precision loss.

WP T1: Implicit representations

As explained above, classical abstract interpretation techniques, e.g. polyhedral analysis, lose precision due to join operations, which are applied at join points in the control flow (end of tests and heads of loops, in structured programs). Partitioning techniques have been applied [40], but they are quickly very costly, thus heuristics are used to limit the number of partitions.

[10] This is a somewhat naive and simple presentation of predicate abstraction; current systems try to avoid such exponential enumerations.

In 2011, we proposed a per-path analysis [37], which amounts to trace partitioning without incurring the space blowup, combining abstract interpretation with logical decision procedures (SMT-solving). This technique simulates the iterations of an exponentially larger transition graph, without ever expanding it, by keeping it *implicit* (this graph is deduced from the solution set of SMT formulas). Initial experimental results are encouraging [25]. Similarly, we proposed a policy iteration algorithm for solving exponentially large systems of min-max equations given implicitly, whose solution is the least inductive invariant of the program in the abstract domain being considered [17].

These techniques are very new; this is only the beginning of their development. There remain many scientific issues to be solved, which will be considered within STATOR.

1. We shall develop improvements to the path focusing technique. The key idea of this technique is to take "big steps" in the control flow of the program, between abstraction and widening points, which form a subset of the original control points. It is unclear how best to select such points [8]. While it is hopeless to request a rule that would produce best results on any program, we expect that we can distinguish useful cases where there is optimal placement, or at least suitable **placement heuristics**.

   Self-loops are particularly interesting since they are amenable to **acceleration**; we shall thus study how to reuse the extant work on **acceleration** (exact or abstract) [19]. Finally, techniques that select subsets of the control graph for intermediate analysis [20] may be lifted to our path graph (we already have some implementation results), but it is yet unclear how to do so while still maintaining sparseness and avoiding enumerating exponential sets of paths.

2. We so far focused on numerical abstractions, because they are the most immediate to test, and also because of the availability of the APRON library. Yet, analysis of realistic programs requires a memory model, dealing with **pointers, aliasing, and dynamic allocation**. Abstract domains for dealing with memory and pointers have long been known [13], with recent trends in shape analysis including separation logic, e.g. [10, 43] and analysis of low-level representations, e.g. [3]. Also, it is possible to model memory within SMT formulas by using uninterpreted functions (memory can be seen as a function from addresses to values) or their mutable version, often known as the theory of arrays [30].

   Yet, the integration of both techniques is not so obvious. It is known to developers of Floyd-Hoare based proof assistants for imperative programs that applying such an encoding naively quickly results in unbearable computation times for solving SMT formulas. It will be necessary to carefully convert the aliasing and pointer information from the abstract domain into logical representations that are a simple as possible; for instance, if a variable is not aliased, it can be mapped to a scalar variable in the formula instead of an array reference; if memory regions can be proved to be disjoint, they may be mapped to two different arrays; and so on. This suggests that rough pointer information may be used to simplify the formulas. While some techniques are already known in this respect, we expect that sizable work still has to be done.

   With respect to pointers, some recent tools use variants of separation logic to analyse the heap [4, 39], but it still somewhat unclear how best to map abstractions to SMT formulas, and how to analyse separation properties efficiently.

3. Many techniques described in this proposal use SMT-solving or other exact logical techniques. It is known that such techniques may perform very badly, taking up exponential time.

Early experimental results from PhD student Julien Henry and post-doc Diego Caminha, on static analysis for worst-case execution time, show that much can be gained by injecting the results of rough static analysis (e.g. rough bounds over some variables) into the SMT formulas, so as to prevent the SMT solver from exploring fruitless parts of the search space. On one example, we have been able to move from the SMT solver failing to terminate after one night of computation to termination within seconds. This is similar to the practice, in operation research, of adding additional constraints ("cuts") that do not change the solution set, but make solving faster. A related idea is the generation of "learned clauses" in modern DPLL SAT solvers [24, 30].

The most obvious suggestion arising from this observation would be to run simple and fast static analyses and use the invariants obtained as additional constraints to the SMT solver or other expensive analyses; we expect such an approach to be ready in the short run. A more ambitious endeavour is a more intimate combination of the abstract and precise analyses: for instance, path splittings may be triggered only when there is evidence that abstract joins resulted in imprecision. Such **dynamic partitioning** has been proposed by Jeannet [28, 29], but without the link to exact formulas and SMT-solving. Very recent work indicates that abstraction may be integrated directly into the decision procedure [12, 23].

We intend to explore such combinations of exact techniques, dynamic splitting, and abstraction.

WP T2: Modularity

There exist modular techniques for static analysis, for instance computing polyhedra between the input and output variables of functions. We intend to adapt these techniques to our path analyses. It is not obvious how to do so properly: the reason we use path analysis is to gain precision with respect to execution context, thus reusing analysis results obtained for any execution context seems counter-productive.

A common criticism at techniques using SMT-solving is that they will have difficulty scaling. Obviously, it is impossible to take a 500,000 line control loop (as analysed by Astrée), compile it into a SMT formula and expect the SMT solver not to blow up; at the same time, it is also infeasible to apply octagon analysis [32, 34], whose cubic running time is better than the exponential in SAT/SMT, to all variables in program above toy size — in both cases, it is necessary to cut the problem into manageable pieces, to be analysed rather independently. More generally, we would like **modular analysis**: ideally, a function or module would be analysed once and for all, and the analysis results reused at points of call.

With respect to modularity, most existing modular analysis techniques consider that suitable units are the functions (in the case of imperative or functional programs) or classes (in the case of object programs). Yet, in the case of control and multithreaded programs, suitable units seem to be interplaying variables and data structures, along with the code that affect them.

We shall therefore develop techniques that identify clusters of related variables and code fragments, then run analyses **modularly and iteratively** (start with analysing each cluster with a rough abstraction of the environment, and then refine). An extension of such techniques is modularization of the analysis of multi-threaded programs; we shall also investigate this, keeping in mind that the analysis of parallel programs is a research topic in itself and that too much work in this direction would spread our resources too thin.

WP T3: Alternative iteration methods

Recently proposed methods for finding numerical invariants express the problem of finding an invariant as a system of monotonic numerical equations or inequalities involving minimum and maximum operators, as well as normal arithmetic expressions. The min and max operators are analogous to the choices made by the players in game theory, and thus policy iteration methods inspired by game theory have been proposed. Min-policy iteration [11] replaces the original analysis problem by a sequence of simpler problems, whose values, in the game-theoretic sense, define a descending sequence of inductive invariants; in general this sequence does not converge to the least inductive invariant definable in the considered abstract domain, but in some cases it can be proved it does so. Max-policy iteration [18] instead considers successive problems whose values under-approximate the least inductive invariant in the domain, and terminates on the least inductive invariant. For both techniques, the number of problems to solve is exponential in the size of the code (roughly, in the number of if-then-else).

We recently developed techniques where explicit "min" and "max" equations (which may be large) are replaced by shorter implicit descriptions using logical formulas [17]. SMT-solving is used to select the next policy. In STATOR, we shall investigate the following problems:

1. Can policy iteration techniques be extended to non-numerical values, taken in general lattices, for instance for shape analysis?

2. For non-linear program constructs and constraints, can we use suitable relaxations yielding problems that can be solved numerically?

3. How can solving methods based on policy iteration be combined with "normal" Kleene iterations? (For instance, Kleene iterations for abstractions of the memory layout, and policy iterations for the numerical contents of memory.)

WP T4: Finite state vs numeric state

Most static analysis techniques split the program state into a control state (often taken within a finite set of control points, sometimes including a call stack) and a data state, and treat the control state specially (often, distinguishing one set of data states per control state). This is a good option for many programs, but other programs encode part of the actual control into the data (e.g. Booleans or other variables expressing the state of a finite automaton, as in parsers or reactive programs); thus this dichotomy is not very satisfactory. Furthermore, other parts of the data can be considered as control: for instance, if pointer p points either to variable x or y, it can be considered as a Boolean, and an assignment `*p = foo;` can be rewritten as **if** (b) then x=foo; **else** y=foo;. Some tools, such as the Polyspace Verifier, do such a transformation before analyzing the data, but this approach often results in exponential blowup.

Again, a key idea is to have a close-knit collaboration between logic-based techniques (SMT-solving and BDDs) and abstract domains for numerical quantities or memory shapes. Another is to keep implicit representations for as much data as possible, as opposed to explicit representations that enumerate potentially exponential sets of configurations. We shall thus investigate the following problems:

1. The integration of Booleans and other finite-state variables, or other discrete components such as pointer aliasing information, into the data state considered by the abstraction, without incurring blowup. Despite the recent work by Bertrand Jeannet on BDD-APRON [27, 42], there is

considerable work to do in this respect: which information to track or not, which relationships to keep or not, how to group similar states in the analysis, etc.

2. The combination with strategy iteration. We proved that abstract reachability for template linear constraint domains and a linear arithmetic transition relation is Π_2^p-complete [17], and provided an algorithm that seems to perform well in practice (though further experiments are needed). The generalisation of the problem with Booleans inside the state is in NEXPTIME and is PSPACE-hard: can we get a better characterisation, and practical algorithms? Since the PSPACE-complete Boolean reachability problem is often practically solved by BDD techniques without the worst-case exponential memory cost, we expect that similar techniques will apply to our problem.

2.4 Experimentation objectives

As static analysis attempts solving in practice a problem that is unsolvable in general, we cannot expect all-encompassing results; at worst, all we can prove is *correctness* of the method (it does not give wrong results, such as claiming that a certain fault cannot occur), but not *completeness* (all true properties are proved). As a consequence, for any static analysis technique, we should ask the following questions:

1. Is it complete for a certain class of programs and/or properties?
2. If so, is this class representative of real programs?
3. How does it perform in practice on real programs?

Question 1 is theory, but 2 and 3 may only be addressed by experimentation. It is worth mentioning again that many techniques published in research articles in the program analysis field are never tested on anything but toy examples. Often, an advanced technique is shown to be better than the previous state of the art by providing examples on which it performs better (faster or with better results), but are these examples representative of problems that occur in the "real world"? Two examples: experiments have shown that Andersen's pointer analysis performs as well in practice as a fully precise flow-insensitive analysis, despite being theoretically weaker [5]; we have failed to find programs on which the predicated global value numbering from Gargi [15] performs better than a simpler non-predicated algorithm, except the case example in Gargi's article.

For these reasons, it seems necessary to us to assess the precision and efficiency of the static analysis methods developed within STATOR by running them on examples from a variety of sources, and comparing them with each other as well as with earlier techniques.

3 Methodology

As explained in the objectives section, we intend to obtain feedback from experiments in order to judge which techniques are useful or efficient, and possibly new directions of research. Furthermore, we are well aware that the time and resources necessary for successful software development, especially debugging and reengineering, should not be underestimated. Our plan is therefore to start implementation-related work right at the start of the project, so that groundwork has already been made at the time when new techniques are ready to be implemented.

3.1 WP T: Theory work

While investigations on a few of our research directions have already begun, some will require more time to be mature. We therefore propose an indicative schedule, with milestones.

WP T1: Implicit representations

Work is already underway by myself, Laure Gonnord and PhD student Julien Henry, with the occasional help of colleagues, but this is only the beginning: many important issues have not been tackled.

We propose the following milestones:

- $t_0 + 12$ Simple pointer analysis and encoding into SMT formulas.
- $t_0 + 36$ Improved pointer analysis.
- $t_0 + 12$ Initial study on the use of use of results from abstract interpretation to speed up SMT computations.
- $t_0 + 48$ A more general framework for the interaction of exact computations with abstraction.

WP T2: Modularity

- $t_0 + 24$ Study on finding clusters of code and data.
- $t_0 + 48$ Study on multithreaded code.
- $t_0 + 54$ Modularisation of the analysis.

WP T3: Alternative iteration methods

- $t_0 + 24$ Study of relaxation techniques.
- $t_0 + 36$ Combinations with Kleene iterations.
- $t_0 + 54$ Policy iteration for non-numerical domains (speculative).

WP T4: Finite state vs numeric state

- $t_0 + 12$ Initial study on the combination of Boolean and other discrete data into the control state used by numerical reachability analysis.
- $t_0 + 24$ Study on iteration techniques on such systems.
- $t_0 + 48$ Study on the combination with strategy iteration.
- $t_0 + 54$ General framework for combination with Booleans into the control state.

3.2 WP I: Experimental work

Our proposal involves the development of a research tool. We divide this work into:

1. the development of an analyser basis, implementing classical techniques as well as recent ones (state of the art)
2. the implementation of the new techniques developed during the project
3. comparative experiments.

WP I1: Development and maintenance of the analyser basis

A static analyser implementing common techniques (say, those not specific to avionic applications described in the Astrée system [6, 7]) and capable of analyzing C programs will be implemented.

The reason why so many articles are published with results only on toy examples is that an analyser that runs decently on real programs is a huge investment. During the Astrée project, we were able to get an analyser running with reasonable performance on narrow class of code in a matter of months. However, adaptations for running on less restricted classes of programs took much effort. We will thus allocate sufficient resources to this workpackage, taking into account the tendency to underestimate development times (especially debugging).

Another observation in that project is that one should not lock the system early on in a supposedly "sufficient" narrow class of programs, for instance a small subset of C reputedly enough for safety-critical applications: real-life programs seldom fit perfectly in such small classes. We thus feel that our implementation should handle a full language, or at least a very large subset thereof (e.g. one could treat C without longjmp()), with graceful degradation for constructs not yet handled well: maybe precision or efficiency would decrease, but the program should not be refused.

The analyzer will detect assertion violations, array access out of bounds, bad pointer dereferences, arithmetic overflow and other violations of the C standard [26]. Depending on development speed and availability of resources, a more expressive annotation system may be developed, but will not be a priority.

Depending on frontend choices, it might be able to run on C++, Fortran or Lustre/Scade. The output will be initially in logging files; depending on resources and ease of implementation, we might implement export or interface modules for GUI presentation (e.g. click on a variable to see the invariant), though this will not be a priority.

We will strive to make implementation as orthogonal and modular as possible, so as to make it easy to implement new techniques or to use other front-ends. (For instance, our current experiments with LLVM as front-end show that it is not easy to stick to the LLVM bitcode representation as intermediate code; another layer is probably preferable.)

- $t_0 + 6$ Basic intra-procedural static analyser based on LLVM; extends extant code.
- $t_0 + 6$ Study about various possibilities of front-ends, and discussion of the best possible choices.
- $t_0 + 12$ Rework of the analyser to abstract away the front-end.
- $t_0 + 18$ Interprocedural analysis.
- $t_0 + 30$ Infrastructure for procedure summaries. Modular analysis with classical abstract domains.
- $t_0 + 48$ Interface with some tools providing GUIs or other presentation interface, such as Frama-C.

WP I2: Implementation of new techniques into the analyser

Once reflexion on algorithms has been sufficiently advanced, they will be implemented into the analyser.

WP I2-1 Implementation of results from T1:

- $t_0 + 18$ Simple pointer analysis and encoding into SMT formulas.

- $t_0 + 42$ Improved pointer analysis.
- $t_0 + 60$ Implementation of the general framework for interaction of abstract and exact analysis.

WP I2-2 Implementation of results from T2:

- $t_0 + 30$ Implementation of clusterised analysis.
- $t_0 + 60$ Implementation of the techniques of for modularisation of the analysis.

WP I2-3 Implementation of results from T3:

- $t_0 + 30$ Implementation of relaxation techniques.
- $t_0 + 42$ Implementation of combinations with Kleene iterations.
- $t_0 + 60$ Implementation of policy iteration for non-numerical domains (speculative).

WP I2-4 Implementation of results from T4:

- $t_0 + 18$ Implementation of the combination of Boolean and other discrete data into the control state used by numerical reachability analysis.
- $t_0 + 36$ Implementation of iteration techniques on such systems.
- $t_0 + 60$ Implementation of the combination with strategy iteration.
- $t_0 + 60$ Implementation of the general framework for combination with Booleans into the control state.

WP I3: Experimentation on real programs

We intend to experiment our new techniques on real programs, particularly the two following classes:

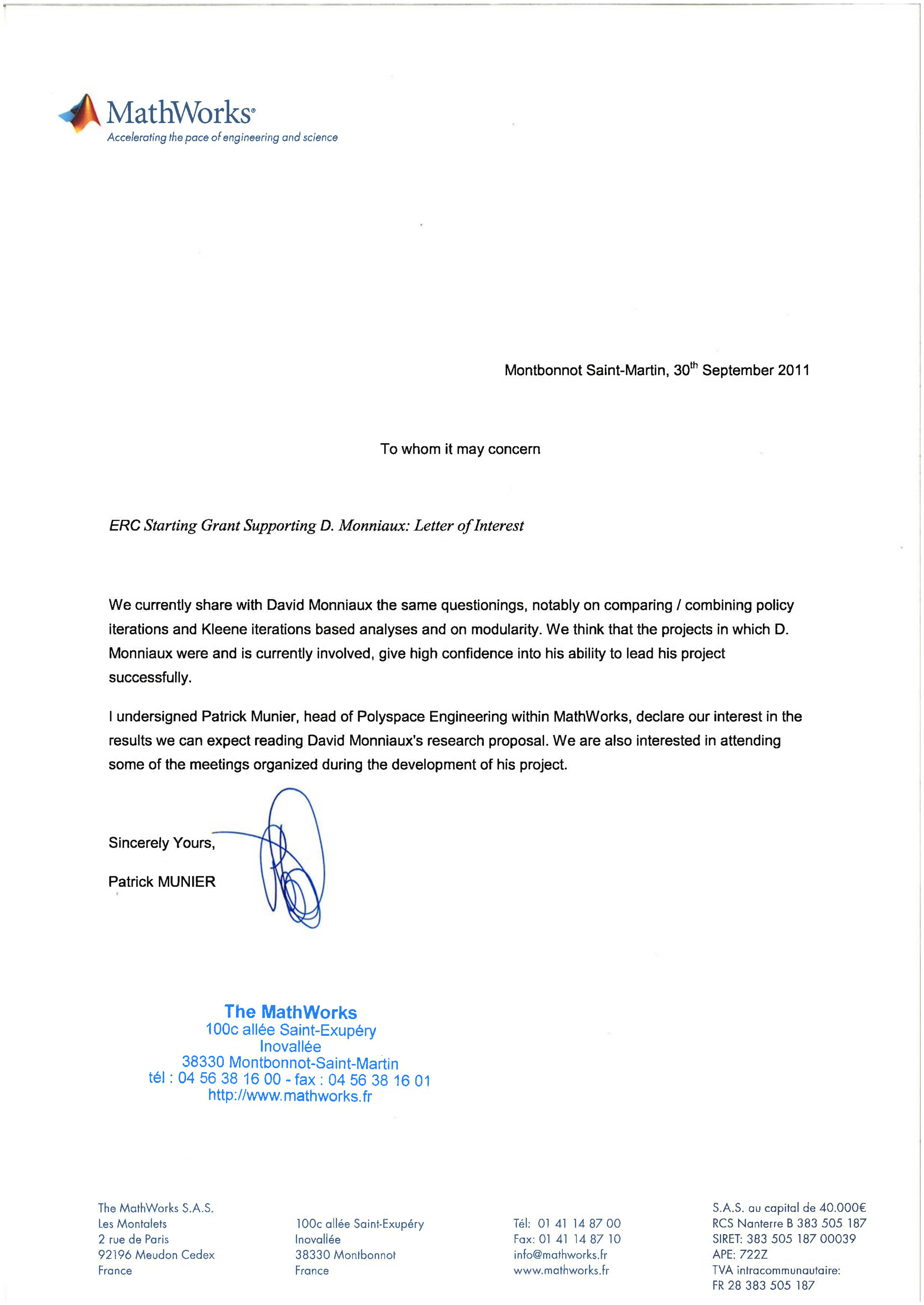
Benchmarks from literature or compiler performance suites.

Free / open-source software We see several advantages to such examples: the source code is available without having to enter lengthy negotiations with organisations producing proprietary software, there are no non-disclosure agreements preventing the open discussion of bugs found, the availability of the code enables comparisons between tools, and there is public interest in the reliability of such software.

Control software Such software may be obtained under non-disclosure agreements (NDAs) from companies in the aerospace, nuclear or automotive areas.

We will conduct the following measurements:

1. Memory and time performance.
2. Precision of the invariants. The simplest is to compare using the inclusion ordering: for instance, $x \in [0, 1000]$ is more precise than $x \in [0, 10000]$. A more subtle comparison, in case of a domain A representing finer properties than another domain B (e.g. A is convex integer polyhedra plus congruences, B is convex real polyhedra) is to translate elements of A into elements of B before comparison: this measures what is gained by computing in A when the property of interest is in B.
3. Precision with respect to a client analysis, such as the runtime error and assertion checking analyses described above.



MathWorks
Accelerating the pace of engineering and science

Montbonnot Saint-Martin, 30th September 2011

To whom it may concern

ERC Starting Grant Supporting D. Monniaux: Letter of Interest

We currently share with David Monniaux the same questionings, notably on comparing / combining policy iterations and Kleene iterations based analyses and on modularity. We think that the projects in which D. Monniaux were and is currently involved, give high confidence into his ability to lead his project successfully.

I undersigned Patrick Munier, head of Polyspace Engineering within MathWorks, declare our interest in the results we can expect reading David Monniaux's research proposal. We are also interested in attending some of the meetings organized during the development of his project.

Sincerely Yours,

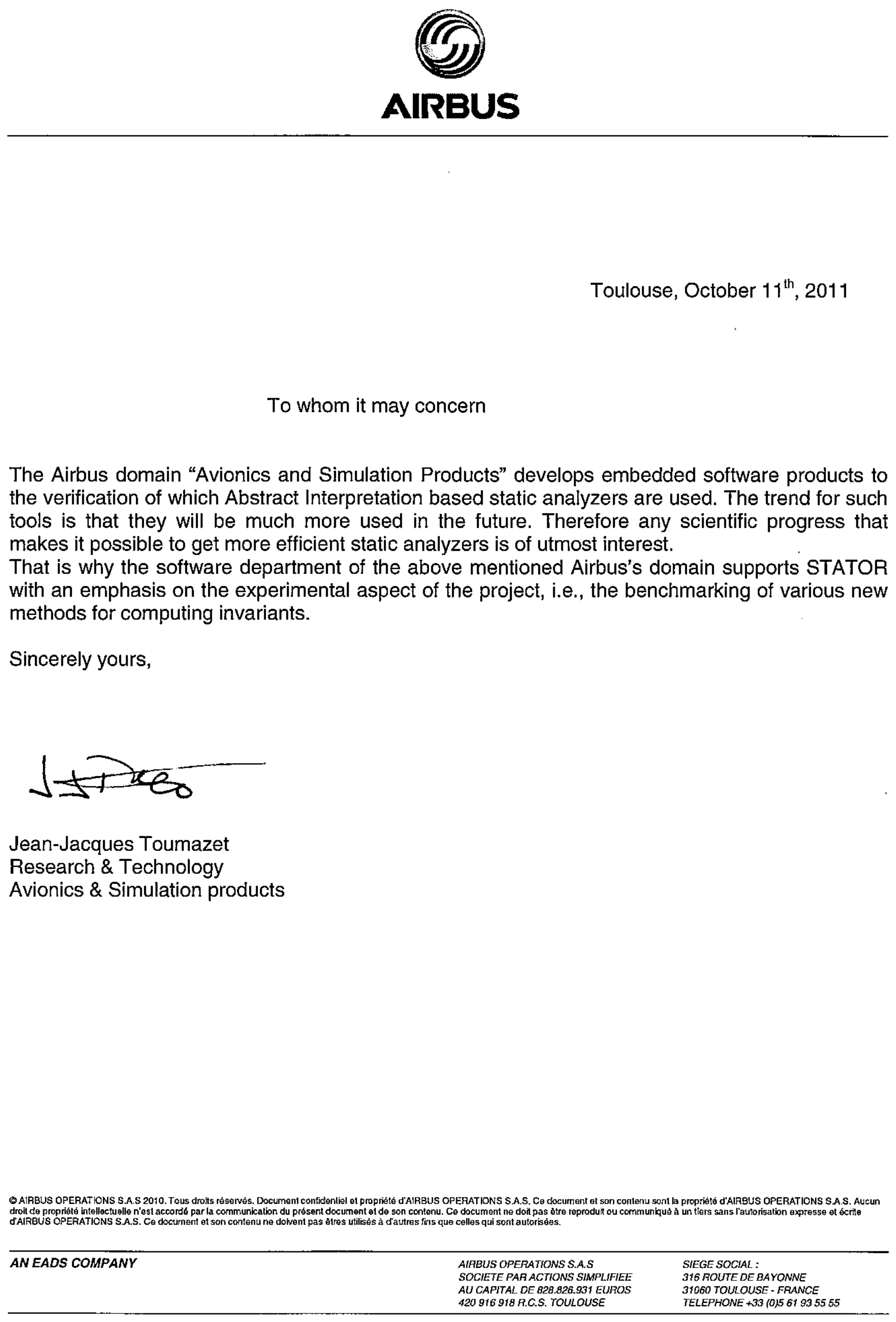
Patrick MUNIER

[illegible]

The MathWorks
100c allée Saint-Exupéry
Inovallée
38330 Montbonnot-Saint-Martin
tél : 04 56 38 16 00 - fax : 04 56 38 16 01
http://www.mathworks.fr

The MathWorks S.A.S. Les Montalets 2 rue de Paris 92196 Meudon Cedex France	100c allée Saint-Exupéry Inovallée 38330 Montbonnot France	Tél: 01 41 14 87 00 Fax: 01 41 14 87 10 info@mathworks.fr www.mathworks.fr	S.A.S. au capital de 40.000€ RCS Nanterre B 383 505 187 SIRET: 383 505 187 00039 APE: 722Z TVA intracommunautaire: FR 28 383 505 187

Figure 1: Letter of interest from Polyspace (The MathWorks)



AIRBUS

Toulouse, October 11th, 2011

To whom it may concern

The Airbus domain "Avionics and Simulation Products" develops embedded software products to the verification of which Abstract Interpretation based static analyzers are used. The trend for such tools is that they will be much more used in the future. Therefore any scientific progress that makes it possible to get more efficient static analyzers is of utmost interest.
That is why the software department of the above mentioned Airbus's domain supports STATOR with an emphasis on the experimental aspect of the project, i.e., the benchmarking of various new methods for computing invariants.

Sincerely yours,

[illegible]

Jean-Jacques Toumazet
Research & Technology
Avionics & Simulation products

Figure 2: Letter of interest from Airbus

We envision this workpackage as ongoing: we do not plan to run all experiments at a certain period in time, rather to run them as techniques, implementation and availability of case examples progresses, so as to obtain feedback as soon as possible.

3.3 WP D Dissemination and collaboration

I will establish a web site for the project, to which reports, articles and other material will be posted.

I plan to hold a regular static analysis seminar at VERIMAG, with participants from neighbouring groups involved in the same areas of research (INRIA-Grenoble, The Mathworks' PolySpace unit; see Fig. 1) and speakers invited from other locations. He also plans to have invited professors (e.g. in 2012 Helmut Veith comes); since the University has separate funding for such positions, they will not need ERC support except possibly for travel expenses. Such a seminar and invitations will support the dynamic work atmosphere desirable for productive advance research work.

I envision collaborations with other groups, be it on algorithmic research or on implementation work. For instance, other groups may choose to implement techniques outside of the main interests of the project onto the analyser developed within the project. Possible collaborations include:

- INRIA-Rocquencourt (X. Leroy) and Université Rennes 1 (S. Blazy, D. Pichardie), CompCert and VERASCO projects: in both STATOR and CompCert / VERASCO, good definitions of both concrete semantics and abstraction and iteration techniques are needed.
- INRIA-Saclay (C. Marché, J.-C. Filliâtre) and CEA (B. Monate), Frama-C: the Frama-C system is envisioned as a front-end.
- L. Gonnord at LIFL (Lille) and INRIA-CompSys, especially for acceleration techniques.
- B. Jeannet at INRIA-Alpes (numerical-Boolean combinations)
- T. Gawlitza at University of Sydney, Australia (policy iteration)
- H. Veith at TU-Wien, Austria (I obtained an invited professor position for him in 2012)
- P. Fontaine, Nancy; N. Bjørner and L. de Moura, Microsoft Research (decision procedures)
- G. Brat and A. Venet, NASA Ames, California
- J. Souyris and D. Delmas, Airbus, Toulouse (see Fig. 1)

These collaborations, including with industrial corporations such as Airbus and PolySpace / MathWorks, will not imply monetary fluxes.

It is my desire that the software developed in the course of the project be released under a free license. However, such a decision has to be made by the appropriate authorities, that is, CNRS and Université Joseph Fourier; I will therefore strongly recommend this option to them. A possibility is to have the software hosted on a repository such as forge.imag.fr, which should help collaborations with other groups.

3.4 Risks

The STATOR project involves only a reduced team, no specialised scientific hardware, little coordination with other projects. **We thus estimate that risks are fairly limited.** We however distinguish the following risks:

1. This is a research project, not an industrial development or implementation project. As a consequence, certain directions that look promising at the beginning may be dead ends; on the contrary, new ideas may emerge during the course of the project. It would be foolish or insincere to pretend to predict at the time of proposal submission the exact state of the research done 4 years from project start.

2. Hiring competent and diligent students, postdoctoral researchers or staff is not easy. A PhD student that seems promising at first may prove disappointing after a while.

3. We plan to rely on a number of external software packages (compiler frontends, BDD libraries, decision procedures, etc.), most of which are open-source and available under free licenses (e.g. APRON, PPL, CUDD), but some of which are closed-source but free for academic use (e.g. Yices, Z3). We cannot be sure whether the closed-source packages will continue being available under such licenses and will be maintained. We shall mitigate the risks by allowing multiple choices and avoiding being tied to one single technical solutions (e.g. using the SMTLIB2 format, one can choose among many SMT-solvers, some of which are proprietary and some of which are under a free license). The risks are lower with respect to open-source packages, because in the worst-case we could simply maintain them ourselves, but this would require supplemental effort.

4 Hosting institution

It is planned that research will take place at the VERIMAG laboratory. VERIMAG is a joint research unit of the *Centre national de la recherche scientifique* (my employer), also known as CNRS[11], Université Joseph Fourier and Grenoble-INP. *The project will be financially managed by the university, and the grant agreement should include a "special clause 10" for CNRS and Grenoble-INP.*

University Joseph Fourier Grenoble 1 (UJF)[12] is located in the middle of the Rhône-Alpes region, 2nd French region in terms of research activities. UJF Grenoble is a research intensive university in an international and high tech environment: 16,800 full time students, excluding doctoral students; 1,500 lecturers and researchers and 1 400 administrative and technical staff; 70 laboratories organised in four core areas; major international and national research centres are located in the Grenoble area, such as ESRF, ILL, EMBL, CEA; a high density of multinational firms in fields such as nano and micro-electronics and biotechnologies (ST microelectronics, Hewlett-Packard, BioMerieux, etc.); three major science parks (MINATEC, BIOPOLIS, MINALOGIC).

More specifically, UJF has a great experience in European framework programmes (FP), first projects were submitted in 1990, during FP3. It has since managed the participation in 69 projects under FP6 (2002-2006) and coordinates 5 of them. More recently, since the beginning of FP7, UJF laboratories have succeeded in 41 projects, 13 of them being coordinated by UJF, and 2 in the ERC program. 4 staff members are directly connected to European research projects at UJF (full or part-time of their activity): 2 at the Project Engineering office; 1 at the financial office and 1 at the personnel department.

UJF makes the following statement: "UJF will guarantee the independence of the researcher in four ways:

1. He will be in charge of the total budget of the project, as a dedicated financial line will be open for the project and he will be solely responsible for it.

[11]http://www.cnrs.fr/; VERIMAG is UMR 5104
[12]http://www.ujf-grenoble.fr/

2. He will be able to supervise students during their master studies, as well as young researchers at doctoral level.
3. He will have the entire possibility to publish in journals etc. without prior notice to the director of the laboratory."

VERIMAG[13] is a leading research center in embedded systems. VERIMAG develops programming and specification languages, modeling tools, as well as verification technology. Major contributions include Lustre, a synchronous data-flow programming language that evolved into the industrial language Scade,[14] used for many critical applications (e.g. avionics at Airbus). Notable researchers at VERIMAG include Joseph Sifakis, CNRS senior researcher, laureate of the 2007 Turing award for the development of model-checking, and Nicolas Halbwachs, CNRS senior researcher, co-inventor of polyhedral static analysis as well as the Lustre language.

5 Ethical and security sensitivity issues

5.1 Ethical issues

There are no ethical issues involved in this proposal.

Research on human embryo / foetus

Does the proposed research involve human Embryos?	NO
Does the proposed research involve human Foetal Tissues/ Cells?	NO
Does the proposed research involve human Embryonic Stem Cells (hESCs)?	NO
Does the proposed research on human Embryonic Stem Cells involve cells in culture?	NO
Does the proposed research on Human Embryonic Stem Cells involve the derivation of cells from Embryos?	NO
I CONFIRM THAT NONE OF THE ABOVE ISSUES APPLY TO MY PROPOSAL	YES

Research on Humans

Does the proposed research involve children?	NO
Does the proposed research involve patients?	NO
Does the proposed research involve persons not able to give consent?	NO
Does the proposed research involve adult healthy volunteers?	NO
Does the proposed research involve Human genetic material?	NO
Does the proposed research involve Human biological samples?	NO
Does the proposed research involve Human data collection?	NO
I CONFIRM THAT NONE OF THE ABOVE ISSUES APPLY TO MY PROPOSAL	YES

Privacy

[13]http://www-verimag.imag.fr/

[14]http://www.esterel-technologies.com/products/scade-suite/

Does the proposed research involve processing of genetic information or personal data (e.g. health, sexual lifestyle, ethnicity, political opinion, religious or philosophical conviction)?	NO
Does the proposed research involve tracking the location or observation of people?	NO
I CONFIRM THAT NONE OF THE ABOVE ISSUES APPLY TO MY PROPOSAL	YES

Research on Animals

Does the proposed research involve research on animals?	NO
Are those animals transgenic small laboratory animals?	NO
Are those animals transgenic farm animals?	NO
Are those animals non-human primates?	NO
Are those animals cloned farm animals?	NO
I CONFIRM THAT NONE OF THE ABOVE ISSUES APPLY TO MY PROPOSAL	YES

Research Involving non-EU Countries (ICPC Countries)

Is the proposed research (or parts of it) going to take place in one or more of the ICPC Countries?	NO
Is any material used in the research (e.g. personal data, animal and/or human tissue samples, genetic material, live animals, etc) :	
a) Collected in any of the ICPC countries?	NO
b) Exported to any other country (including ICPC and EU Member States)?	NO
I CONFIRM THAT NONE OF THE ABOVE ISSUES APPLY TO MY PROPOSAL	YES

Dual Use

Research having direct military use	NO
Research having the potential for terrorist abuse	NO
I CONFIRM THAT NONE OF THE ABOVE ISSUES APPLY TO MY PROPOSAL	YES

5.2 Security sensitivity issues

There are no security sensitivity issues involved in this proposal.