Combining Graded and Intersection Types for the Analyses of Resources

Project Summary

Type systems are used to automatically check security properties of large programs. This project will extend typing methodology to a large panel of properties currently unreachable by state-of-the-art techniques, enabling in particular the analysis of **quantitative properties of programs**.

We will develop a way to keep track of the extensional information inside types in order to perform the whole static analysis at the level of types. For this purpose, we will combine two (re)emerging type systems, namely graded types and intersection type systems, with the well established techniques from the field of abstract interpretation such as widening.

Graded type systems formally embed a first order structure within types, while intersection types will help to circumvent the unconditional non-compositionality of fine grained resource analyses. This is how we plan to tackle the long running problem of applying abstract interpretation result in functional programming.

People Involved in the Project

Affiliation	Family Name	First Name	Position	P.M.	Role & Responsibilities
University Paris 13	BREUVART	Flavien	MCF	25 (60%)	Scientific Coordinator DS,GT,IT,(RA,FP,DT)
University Paris 13	ARIAS	Jaime	IR2	6 (15%)	Engineer FP, Implementation
University Paris 13	MAZZA	Damiano	CR1	4 (10%)	PhD Co-Advisor IT, DS, GT, (RA)
University Paris 13	SEILLER	Thomas	CR1	4 (10%)	Scientific Collaborator RA, DT, DS, (GT)
University Paris 13	MANZONETTO	Giulio	MCF	4 (10%)	PhD Co-Advisor IT, DS, RA, (FP)
University P.Diderot	REGIS-GIANAS	Yann	MCF	4 (10%)	External Collaborator FP, DT, (AI), Implem
Inria Paris	FERET	Jérôme	CR1	2 (5%)	External Consultant AI, RA, (DS), Implem

GT: Graded Types, DS: Denotational Semantics, IT: Intersection Types RA: Resource Analysis
FP: Functional Programming AI: Abstract Interpretation DT: Dependant Types
without parenthesis = "expert on" with parenthesis = "have experience with"

Evolution Compared to the Pre-Proposal

Since submission of the preproposal, Jame Arias was appointed as Research Engineer in the team LoVe of the LIPN. Having a rich and fruitful experience in functional programming, he is enthusiastic in being part of the project and being in charge of WP-2. Due to his arrival, the need of WP-2 for a post-doc largely decreased, and we choose to reassign the 1-year-post-doc to WP-3.

In addition, the requested amount has slightly decreased due to some expanses being over-evaluated in the pre-proposal.

I Context, Positioning and Objectives of the Proposal

I.1 Objectives and Scientific Hypotheses

Overview: Type systems are used to automatically check security properties of large programs. This project will extend typing methodology to a large panel of properties currently unreachable by state-of-the-art techniques, enabling in particular the analysis of quantitative properties of programs. This will be achieved by expanding an emerging notion of graded types (GT) and borrowing methods from the community working on abstract interpretation (AI).

Functional programming languages often use complex type systems describing the *intentional* behaviour of programs (understand structure). These types are statically inferred before the compilation, but often use meta-information provided by the programmer in the form of type annotations or choices of primitives.

However, to describe *extensional* behaviour of programs, analyses using AI are substantially more refined. By extensional behaviour, we mean the information regarding the execution, such as the complexity, the sequenciality, the probabilistic behaviour or any other qualitative or quantitative information linked with the actual implementation and not the program structure. The analyses of extensional behaviours are later referred as *resource analyses* coherently with the literature.

AI techniques for resource analysis are very efficient, but not compositional, and thus not adapted to analyse the higher order structure of functional programs. This means, in particular, that most of the *intentional* information provided by the programmer is lost in those extensional analyses. This project will develop the notion of GT that offers a way to keep track of the extensional information inside types in order to perform the whole static analysis at type level. Here, the *extensional* information (later called resource) is embedded as a first order structure in the higher order types; thus only using AI's methods on first order objects.

More precisely, there exist some classes of graded type system whose members are parametric over a first order structure. This parametricity embeds a natural notion of Galois relation (approximation order) on structures so that they can play the role of domains of AI. Moreover, one can describe a functor from those structures to traditional AI's domains[1] preserving this Galois relation.

In order to apply AI's methodologies, we need to define a notion of parametric GT with decidable (abstract) and complete (concrete) graded structures related by abstraction relations. As discussed further, the current parametric definitions of GT make quite good abstract domains, but already lack expressive power to tackle complex analyses, and *a fortiori* to define complete domains. Fortunately, specific (non-parametric) extensions in the literature are able to perform some more complex analyses and even to achieve some forms of completeness. Those type extensions either use dependency or intersections (IT). We aim at including them into a common parametric definition of GTs in which we could perform full fledged AI.

This generalisation of GTs will be performed through the lens of denotational semantics. Indeed, denotational and categorical semantics offer a common language of choice for the communities involved. As we will see in the next section, the most important tools of this proposal (AI, dependant types, IT, GT) are all originated from considerations over denotational semantics. In particular, both domain and category theories are fundamental for these tools and the associated advancements. Notice that the denotational semantics (and in particular domain and category theories) and its uses for resource analysis are central in the research of the PI.

In order to have a concrete goal, we will focus our attention on three specific applicative cases: the analyses of dead-code, throwable exceptions, and sequenciality. Nonetheless, we aim at defining a global theory not directly related to a specific application; as such, our theoretical results need to be as general

[1] The resulting type system can be seen as a domain.

as possible, following a mathematical methodology.

Those three cases are classic applications of abstract interpretations that have practical use and that are more and more descriptive. Dead-code analysis basically looks for variables that are no more used (in order to process a garbage collection step). Throwable exceptions analysis is a disguised dead-code analysis that have to deal with exceptions (caught exceptions, one exception shadowing the other...). Finally, the sequenciality has to recover the full sequenciality of the possible traces (in order to optimise scheduling for example), which is obviously more complex and more undecidable than the previous cases. Given the other research interests of the PI and the members of his team, two other kinds of study may be performed as byproducts: the complexity analysis, and the probabilistic version of previous analyses.

I.2 Originality and Relevance in Relation to the State of the Art

I.2.1 Abstract interpretation for functional programming : an historical debate.

Abstract interpretation (AI). The AI methodologies for static analyses were developed by R. Cousot & P. Cousot in the 80's [27]. Their fruitful methods consist in interpreting a language by two domains: a concrete one and an abstract one. Programs and their approximations are associated to points of these domains, with an exact matching in the concrete domain and a collapse in the abstract one. However, an abstract domain is created in order to relate its concrete one, in a way that conserves or approximates the wanted property (ideally through a Galois connection [28]).

The analysis then takes place in the abstract domain, which is typically structured into a posets with arbitrary coherent joins (where points are not coherent if they cannot be from the same program). This way, one is able to compute the interpretations of programs via fixpoints, and potentially with convergence accelerators such as widening or narrowing.

Compositionality looses information. The research of concrete domains for functional programming is older than even the idea of abstract interpretation, through the denotational semantics thread. Those researches where quite fruitful with the discover of Scott domains [64], Girard coherent spaces [37] or HO game semantics [44], which the PI is expert on.

The abstraction of such domains will structurally pass through the abstraction of their most important operator: the application/composition. Unfortunately, the compositionality is structurally incomplete: without the knowledge of the composed function, we cannot statically compute the control flow of the program and thus perform a refined analysis. As result, these domains becomes too complex to be efficiently abstracted for practical purposes.

Abstracting abstract machines. Due to this stepback, most refined abstract analyses of functional programs are whole program analyses that do not target the program itself but a transformation into its first-order abstract machine [42] (or equivalently through its defunctionalysation). These complex and rough transformations, however, tend to break all intentional information on the program and seriously hinder the upcomming analysis process.

Intersection type: an information preserving abstraction? With such constraints, an attentive reader may wonder how concrete semantics manage to be static, compositional *and* complete. Such s paradox is even more dreadful when applying Stone duality, which will transform Scott domains into systems of intersection types (IT). We will later explain that those are two different "kinds" of compositionality.

IT have been intensively studied since the 80's [26] and are now showing several applications and generalisations [53]. Their main trait is that they are complete in the strongest sens: for any behaviour[2] of a program, there is an IT that reflects it. This means, however, that the type inference is undecidable,

[2]Strictly speaking, we only consider observable and "commutative" (we do not know the order of the events) behaviours.

and that there is potentially infinitely [3] many independent ways to type the same program (one for each of its behaviours...).

From most general to more general type. Abramsky showed in 1990 [1] that compositional abstract interpretations were the logical relation or (equivalently) the realizability models, which are mathematical abstractions for types. But this notion of compositionality, is a relaxation of the above mentioned notion since IT are such type systems.

The IT, at first, seems useless: a type derivation alone does not give much information on the program and amassing derivations is highly non efficient. From another perspective, they contain a formidable idea: that we can achieve completeness by dropping the need for a type system to furnish "most general types". In fact, what is really useful for static analysis is the "more general type" so that one can refine the analyse as far as his computational power allows it. The idea is for a library supplier to perform a pre-analysis that composes with generic applications; (even if non-standard applications shall need further analyses).

Dependency What if we try to amalgamate several proofs from IT? This is a way to look at what is done by dependent types systems. However, most existing dependent type systems have so rigid dependency disciplines that the full higher order proof have to be given together with the type, thus deceasing the possibility of automation. More precisely, this rigidity is an hindrance because it disallows any "more general type" theorem. We will thus look, in this proposal, at systems which dependency only appears on first order objects called resources, and in which we can perform such joins over (coherent) types.

Abstracting abstract interpretation Stone duality links ITs with domains by connecting sets of types to points of the domain, thus forming a complete lattice. But if we aim at amalgamating different type derivations, the points of the abstract domain should be the type derivations themselves. Which, unfortunately, is inconsistent with the absence of "most general" types. More concretely, our issue is that we cannot compute the type of recursion by a fixpoint analysis if we do not have a "most general" type, whose presence is inconsistent with completeness.

In order to bypass this issue, we won't consider a AI analysis on the type system (and on the program), but only on a the first order resource information carried by types. The space where this information lives have to be a domain that kind of represents the abstract domain of the abstracted abstract machine we were discussing; only without the hindrance of the technical encoding.

Notice, however, that this approach does not intend to rule out every problem in the abstraction of functional programs. In particular, we do not offer any new approach to the analysis of recursive structures. For now, we prefer to focus on one point, which happens to be more than complex enough; but in the long run we will combine the analysis with other approaches.

I.2.2 Various attempts to perform type-level automated resource analysis.

In the literature, type systems that can answer specific resource analyses are numerous and diversified. We will specifically present some of them that share some similarities that we will try to exploit. We will therefore elude type systems which inference has no objective of being automatisable (*e.g.*, calculus of construction, IT...).

Sizes-types. Size types [43] form one of the first success story of type-level resource analyses. The idea is simply to precise the size of your data-structures along with their types. For example, $\texttt{List}[3]$ is the type of a list of size 3. Then you can have function types such as $tl : \texttt{List}[3] \to \texttt{List}[2]$. Of course, we will need polymorphism over resources so that we prefer to note $tl : \texttt{List}[i+1] \to \texttt{List}[i]$. Similarly, the append operation have the type $append : \texttt{List}[i] \to \texttt{List}[j] \to \texttt{List}[i+j]$.

This presentation of size type is already useful. In fact, one of the main ingredient of the success of *generalised algebraic data types* (GADTs) [60] is simply the fact that they can encode size-types, using

[3] For simply typed λ-terms, there is a finite number of IT, which is why they are so useful in such framework

unification to resolve the type equations (see for example the GADT encoding of AVLs). However, this is not sufficient to fully automatise the inference of size-types. The first issue is the undecidability of the exact size in the general case, but this is kind of acceptable: we can look for a lower (or upper) bound, with trivial answer in unknown cases. The real issue for us is the second one: this is not compositional in the general case.

The non-compositionality comes from the difficulty, if not the impossibility, to unify two arbitrary functions $\texttt{List}[f(i,j,k...)]$ and $\texttt{List}[g(i,j,k...)]$. In order to bypass this issue (that will be recurrent for our problem), people had to differentiate *data* from *co-data*. Data are inductive types on which we can perform inductive recursion and *forward analyses* for other functionals, whose types are of the form $\texttt{List}[i]\to\texttt{List}[f(i)]$. Co-data, are the opposite objects: they are coinductive types on which we can perform coinductive recursions and *backward analyses* for other functionals, whose types are of the form $\texttt{Stream}[f(i)]\to\texttt{Stream}[i]$ (where $\texttt{Stream}[i]$ means, here that the "infinite" stream is used i times).

Once this consideration accepted, one can remove the dependency in i and only write functions. For example we write $\texttt{List}\to\texttt{List}\to\texttt{List}[f(i,j)]$ for $\texttt{List}[i]\to\texttt{List}[j]\to\texttt{List}[f(i,j)]$ so that composition of types is only performed by compositions of those functions. This simplification, however, tends to fail for programs of higher order (more that 2), we will rediscuss it.

Information flow. In the 90's, people also experimented the use of type systems for the analysis of information flow [58]. A simple, and important, example for those, is the use of type annotations characterising a "security level". One can annotate l a secured program and h a non-secured one. For example the program $\lambda xy.x : l.int\to l.int\to l.int$ has a secured output whenever its inputs are secured.

Stated like this, such type system seems quite weak as there may be many types for the same program. For example we also have $\lambda xy.x : h.int\to l.int\to h.int$. This weakness is resolved by using a backward control flow analysis. Since secured program can always be cast into a non-secured one, the resource l is thus less "precise". This means, firstly, that there is a "most precise" annotation,[4] and, secondly, that the resource attached to the return type is not necessary anymore, as one only needs to know how to get a secured output. Thus the principal type becomes $\lambda xy.x : l.int\to h.int\to int$. Notice, however, that for inherently unsecured output (not depending of the input), we cannot write them this way and need, for example, a third symbol in the input (we will come back to this point).

Multilinear types. Linear types are yet another example of type extracting extensional information. The principle is simple, but restrictive: no linear argument can be duplicated; tracking linear arguments can help the compiler for optimisation or security checks [69]. A more refine version of linear types is given by multilinear ones, that are tracking the multiplicity of use of each argument. This system, once again, performs a backward analysis in order to be compositional, with the following example of application rule:[5]

$$\frac{x : 1.\texttt{int} \vdash \lambda f.f\ (f\ x) : 2.(1.\texttt{int}\to\texttt{int})\to\texttt{int} \qquad x : 1.\texttt{int} \vdash \lambda y.y+x : 1.\texttt{int}\to\texttt{int}}{x : 3.\texttt{int} \vdash (\lambda f.f\ (f\ x))\ (\lambda y.y+x)\ : \texttt{int}}\ 2*1+1=3$$

In this example, the program $\lambda f.f\ (f\ x)$ has the type $2.(1.\texttt{int}\to\texttt{int})\to\texttt{int}$; this means that it uses twice its input f, which, itself use its own argument once. In the end, the argument x is used 3 times: once in the call $\lambda f.f\ (f\ x)$ and once for each of the calls to f. Notice that to get the annotations of the conclusion sequent, we only need to perform a simple operation on the hypotheses'.

Fuzz. [33] A more recent and more impressive achievement in this direction is the language Fuzz, that is able to enforce programs to noise the data sufficiently to conserve privacy (referred as differential privacy).

[4] It may seems weird to use the term "most precise" where type theorist always want the "most general" type, but think of it as a duality between the orders given by resource approximation and polymorphism substitution. Later on, we will fuse the two by reversing the polymorphism...

[5] This corresponds to call-by-name evaluation.

Their type checking algorithm[6] may not always succeed, but is able to use an SMT solver to resolve most conflict. We won't describe it in details as the type system of Fuzz is extremely rich, but it is basically using both a backward (for linearity) and a forward (for noise) analyses in the very style we described above; the main differences being the way both analyses interact and the dependency the resource can have regarding the inputs.

Fuzz project is extremely impressive and brought hope regarding the feasibility of CoGITARe. Indeed, from a certain point of view, our objective is to generalise the ideas behind Fuzz, and make them systematically applicable to other frameworks.

I.2.3 Graded types : Unifying and generalising graded type systems.

Graded (co)monads. In recent years, we have seen a theoretical breakthrough on the application of graded algebraic structures [] to semantic models of computation [54, 55].

In 2014, following this breakthrough,[7] two groups gave simultaneously a general framework to work with type-level backward analyses: the graded comonads[8] [20, 36].[9] For any ordered semiring $\mathcal{S}$, they where able to give a graded monad $!_{\mathcal{S}}$ and a type system performing the backward analysis in a call-by-name language. This systems are obtained by a parameterisation, or gradation, of the exponential of linear logic. In particular, the sum of the semiring corresponds to the contraction and the multiplication correspond to the digging:

$$\frac{\Gamma, x : !_i A, y : !_j A \vdash t : B}{\Gamma, x : !_{i+j} A \vdash t[x/y] \ : B}\ \text{contr} \qquad \frac{\Gamma, \vdash t : B}{\Gamma, x : !_0 A \vdash t \ : B}\ \text{weak} \qquad \frac{\Gamma, x : !_i !_j A \vdash t : B}{\Gamma, x : !_{i*j} A \vdash t \ : B}\ \text{digg} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma, x : !_1 A \vdash t \ : B}\ \text{der}$$

The functional types, in these systems, are of the form $i.\sigma \to \tau$ where i is an element from the semiring $\mathcal{S}$ associated to the specific considered system. The semiring $\mathcal{S}$ can have various shapes representing different kind of resources (Booleans for security level, natural numbers for multilinearity, monotonous functions for size types...). They also furnished some new potential of applications.

Surfing on the same breakthrough, an equivalent version for forward analysis was published the same year [46]. Even if graded monads and comonads are kind of dual, the monads are somehow simpler and only require an ordered monoid, not a semiring, for the parameters.

Modularity. As for traditional (co)monads, modularity of graded (co)monads (putting together several of them) is not trivial, but is essential for fine grained analyses. One has to go throw a complex distributive law involving the semiring/monoid. A first work in this direction was achieved by the PI and other authors [34], but this is not the end of the story.

Not general enough. These graded systems share wonderful properties, but are unfortunately quite poor, both in term of kinds of resources represented and in term of expressive power (i.e., programs that we can typecheck). Here is a non exhaustive list of issues:

- Graded comonads are not able to observe non-commutative properties, such as the order in which an argument in used.
- Graded monads contract the different sources of the flow linearly; for example the size-type $\texttt{List}[i]\to\texttt{List}[j]\to\texttt{List}[2*i*j]$ can be represented by the graded type $\texttt{List}\to\texttt{List}\to(2*)\texttt{List}$, but it is impossible to represent $\texttt{List}[i]\to\texttt{List}[j]\to\texttt{List}[2i+3j]$.
- Graded comonad works poorly in call by value, and graded monads works poorly in call-by-name programming languages.

[6] Technically, they do not furnish an inference algorithm, they a type checking algorithm without type annotations; which is similar in practice.

[7] The above papers where written in 2014

[8] None of the two groups was using this name, it only appeared later on.

[9] Notice that Damiano Mazza was part of one of these groups.

- The graded (co)monad is not so canonical, in the sens that one can imagine other kind of slightly different systems that do not enter these classes [59].

Game-like types. Preliminary works towards those issues are currently being performed by the PI and Ghica. The resulting system is an intermediate one that can simulate the graded monads in CbV and the graded comonads in CbN and falls into the above issues that we temporarily call game-like types. In this framework, we can define, for example, a type system that develop the execution order of a term, so that the tail call fixpoint has the type $TCFix : l.(h.\tau \to (\alpha h + \beta).\tau) \to \alpha^* \beta.\tau$ where l, h and k are labels tracking the usage of each arguments and where α and β are resource-polymorphic variables (that do not use h).

However, there is another issue in which our new system fall over: the drastic lack of expressivity for terms of functional order $n > 2$. This is not so problematic for applications to OCaml, whose programs are mostly of order 2, but it is a big obstacle for other languages such as Haskell: we show in next subsection how we intend to tackle this issue.

Semantics. The semantics of graded monads and comonads are now well known and well understood [18, 32, 34, 54, 55], but none of them let any door for generalisations. Two preliminary attempts are currently investigated investigated by the PI: the semantic analysis of game-like types that is also performed with Dan Ghica; and a categorical study of the gradation of strong monads, that subsume both standard monads and comonads [5], with Thomas Seiller.

I.2.4 Resource dependency and approximations : two ways to add expressivity

In Section I.2.1, we where explaining that dependency should allows us to aggregate "coherent" behaviours. An other way to aggregate behaviours is to add an order on resources, allowing approximant types that aggregates all their approximations. Those two dynamics should be formatted into the intentional and the extensional faces of the same concept which we would like to use for subtyping.

Powerful systems are dependant : Fuzz, BLL and linear dependant types. In addition to Fuzz [33], there is at least 2 different systems from which the graded monads are inspired but that are fairly more expressive. Those are the Girard's Bounded linear logic [38] and Bucciarelli & Ehrhard's linear dependant types [21, 41]. The three of them are able to precisely characterise properties statically (even if only Fuzz is inferenceable). However, each of them happened to use some kind of dependency over resources. Even more surprising: each of them use a different one !

This means that we have to get a very abstract vision of dependency, here. In Fuzz, it is possible to have resources depending on first order terms, but at least there is no binder of resource variables inside resources. In BLL, however, in addition to contain a resource, every annotation somehow contain a binder on a resource variable appearing in the annotated term. In linear dependant types, every resource is defined over a context and resource annotations change the context...

Preliminary works on the logic. An unpublished preliminary work on this subject can be find in the PI's thesis. This work is not complete but contains the basics of our philosophy: the resources are now morphism of a category, with dependency expressed by actions over all resources of a given type. For example, if we want to refine the type of $\lambda f.f\ (f\ id) : ((\tau \to \tau) \to \tau \to \tau) \to \tau \to \tau$ with the number of use of each argument, then the f could be used with functional arguments of different types, thus enforcing a more complex type. The most general grading (which is not the most general graded type) should be something like $\lambda f.f\ (f\ id) : \forall i \in [1, \phi(1)].(\phi(i).(i.\tau \to \tau) \to \psi(i).\tau \to \tau) \to \phi^2(1).\tau \to \tau$. Then $\forall i \in [1, \phi(1)]$ is both a resource[10] and a binder of resource variable. We can either approximate the elements inside the multiset of the multiset itself (e.g. by using segments): this is a sole algebraic operation.

[10] $[1, \phi(1)]$ denotes a multiset with two elements, thus $\forall i \in [1, \phi(1)]$ says that the argument is used twice

Semantics. Semantics of dependant types have been extensively studied [45]. But those are very constrained and cannot be adapted easily. In particular, modelling dependency in presence of effects or linearity is one of the most important challenge of the area. In our case, the situation is different as the dependency only target resources. We already have early result on denotational semantics obtained by slicing models of linear logic [10]. One of our goal will be to formalise and generalise them in order to fit our framework. An other goal is to situate ourselves with regard to the study of dependant effectfull models and participate in the field with a proposition of locally linear category.

Refinement and liquid types Refinement types [31], and the more recent liquid types [63] also aim at automatically capturing refined properties of programmes via the inference of dependant types. Their approach is nonetheless quite different as they mainly consider intentional or size properties. They are not able, for example, to qualify the frequency of use of some resource. Moreover, the dependency is a type dependency, not a resource dependency, which is both more powerful and more difficult to infer. Nonetheless, we intend to study the links between the two approaches at the end of the project.

I.2.5 Intersection types: a concrete graded structure ?

The whole idea of this project can trace its roots back to an article from the PI on the use of non-standard relational models to model graded comonads [18]. The interesting point of this study is that the adequation of the interpretation is completely standard up to an additional requirement: the adequation of the semiring grading the considered monads with respect to an emerging algebraic structure inside the model.

A natural Galois connection. This emerging structure is very close to the structure we where trying to model: this is an ordered right-lax semiring interpreting an ordered semiring. Moreover, the "interpretation" is a natural generalisation of Galois relations to ordered (right-lax) semiring. Which means that we are somehow interpreting the (abstract) type system into the (concrete) model via a Galois relation at the level of the embedded algebraic structure rather than the logic/model, which is much simpler.

A concrete graded structure. It is also interesting to remark that, through the lens of stone duality, our models corresponds to specific (non-idempotent) intersection types. We believe, in fact, that these IT systems can be reducible to specific (and complex) GT systems which gradations follow the formers, empirically observed, algebraic structures.

Intersection types in the modern world. IT have recently been subject to a regain in interest [30, 53], as several results highlighted their capability to precisely and efficiently describe the behaviour of programs, even with practical analyses in mind [47]. We believe, and we are not alone [41], that the analytic power of IT and GT are stems from the same fundamental source and should be studied together.

I.2.6 Deteriorating abstract graded types

Categorical semantics and Curry Howard Their graded systems all have perfectly clean categorical semantics, generalising the well known notions of monads and comonads. Moreover, they forms a Curry Howard isomorphism with associated logics. This means that we can work on the same object at any level: operational, categorical or logical. Those are healthy properties that we require as much as possible, but that we may have to bypass in the future.

BLL case. For example, in Girard's Bounded linear logic [38], composition of terms are subject, not just to type restrictions, but to complex grading restrictions that are difficult to account for at categorical level. These restrictions are not inherent to the analysis, but are due to the level of approximation performed by the system, in the sens that the system is an abstraction of a more complex (and concrete) system that recovers these properties.

Subject reduction. Once we are convinced that logical properties only are of importance for the concrete system, we can have bolder thoughts. In particular, GT systems do not necessarily need to respect subject reduction if their concretisation does respect it. What we need is for the types to make sense and for the gradation to stands for correct extensional information. The subject reduction may be essential for the proof of correctness, we only need such a proof in the concrete system.

Derivation rules. In fact, we can be even bolder: We could accept typing sequents without type derivation, with the condition that it is the image of a derivable sequent in a more concrete system. This means that we can use a statements that are not provable in the system but only in a more precise/concrete one.

Galois Relation and widening. In practical uses of AI, a full Galois relation is often impossible to obtain. Thus people works with weaker relations together with a widening operator that reach fixpoints via overapproximations. We expect the same situation in our framework.

All these are deteriorations of our abstract systems that we will need to perform in order to reach efficiency. But we will be careful to conserve important properties through the connection with the concrete systems that are mathematically as sound as possible.

I.3 Methodology and Risk Management

As describe in more details in Section II.2, we distinguish three work packages.

- **WP-1. Generalising GT and integrating methods of AI.** aims at delivering the main objective of this proposal: the theoretical foundation of AI via graded and intersection types.
- **WP-2. Towards the grading of fully fledged programming languages.** is a byproduct aiming at a long term objective : we are investigating the integration with real-life types system and operative semantics and we are building a prototype.
- **WP-3. Fruitful interaction with the semantics of type systems.** is another important byproduct: the reciprocal enrichments with denotational semantics.

In order to have a concrete goal, we will focus our attention on three specific cases: the analyses of dead-code, throwable exceptions, and game-like. Nonetheless, we aim at defining a global theory not directly related to a specific application; as such, our theoretical results need to be as general as possible, following a mathematical methodology.

The risks intrinsic to any research project will be managed by relying on the experience of the researchers in their respective fields, by weekly (if not daily) interacting with internal members, and by organising bi-monthly meetings to monitor the progress of the project with the advice of both internal and external members of the team. Risks more specifics to each task are described along with those in Section II.2.

The coordinator will set up an online repository of documents, containing all relevant material written by the team members including papers, notes, drafts, slides. The website of the group will include a wiki section allowing for more informal contributions and collaborative work.

II Project Organisation and Means Implemented

II.1 Scientific Coordinator and its Team

The members of the CoGITARe team are all young researchers (2-12 years after their PhD defense). Considering the different stages of career, each member is an internationally recognised expert in his field of expertise and will contribute to the success of the project by bringing the specific competences that are required to complete the different tasks. We will make a distinction between the members of LIPN, and the external experts whose implication we need to explain.

II.1.1 Scientific coordinator

The PI was recently employed as a *maître de conférences* at LIPN, Université Paris 13. He accessed to this position only 10 months after his PhD defense. He has a total of 6(+2 submitted) publications[11] in different top-ranked conferences (LICS, ICFP, FoSSaCS, CSL, FSCD and TLCA), and 1(+2) publications in international journals. The LICS and CSL communities, in particular, selected the PI for the Kleene Award (best student papers) in 2014 for their joint meeting.[12]

A more fine grained analysis of his publication record witnesses his lively interactions with the community and his opening to a broad variety of subjects in contrast with his youth. Indeed, he is well balanced between 3(+1) autonomous papers and 4(+2) collaborative papers with 8 different collaborators. Moreover, his research is spanning 4 different main themes:[13] denotational semantics [6, 7, 8, 9, 11, 15, 16, 17, 18, 19, 34], lambda calculus [7, 8, 9, 11, 16, 17], GT [6, 15, 18, 19, 34] and probabilistic higher order programming [12, 13, 14]. In addition to these main themes, this record also shows that he is well versed in numerous minor themes that are fundamental for the project: IT [6, 9, 12, 16, 17], realisability proofs [7, 12, 13, 14, 15, 34], category theory [6, 8, 11, 12, 17, 18, 19, 34], domain theory [7, 11, 18], resource analysis [6, 8, 12, 15, 18, 34], dependant types [6, 15, 19], functional programming [12, 15, 19, 34].

Due to his youth, the PI lacks experience in administration and coordination of project, as well as in student supervision. However, he will receive the assistance of D. Mazza, G. Manzonetto and T. Seiller in his team regarding these points. Moreover, he will stay in contact with the SAIC of Paris13 that is experienced in helping with the administration of ANR projects.

II.1.2 Local project team

The project will give a new impulsion to the younger generation of the PI's team inside the LIPN[14] that was augmented by 3 members recently.

The members of this project team, being part of an official research team, have common weekly seminars and weekly working sessions. As such, the management of the local team will mainly occur throw these events. In particular, we intend to reserve a part of both seminars and working sessions to be oriented toward CoGITARe. In addition, the PI will have independent collaborations (and thus meetings) with each member.

Jaime Arias(IR2-CNRS) [Tasks 2.1.2, 2.1.4, 2.2.1, 2.2.2, 2.2.3, 2.2.4] is expert in software engineering and has a background in formal methods for the specification and verification of systems (e.g., Linear Logic, Session Types, Model Checking, etc). He recently got a research engineer position assigned to the

[11] The number between parentheses represent submitted articles.

[12] Notice that for this exceptional join meeting, they where more than 200 submissions and that among ∼70 accepted papers, 9 were student papers; which testify the special competitivity in 2014 even for an A+ conference.

[13] For completeness, we also include works in progress.

[14] Administratively, the team of the PI is LoVe, but the team is in the middle of a scission procedure that should be effective by the end of the project; we thus only consider, herein the "logic" part of the team.

coordination and development of software projects in the PI team.

In the last years, he has been involved in several international research projects where he showcased the theoretical results through prototypical and robust tools [4] using functional programming languages such as Ocaml, Haskell and ReactiveML. In particular, he is working on a library, called ReactiveSessions, for the specification of communication-based software featuring declarative, reactive, timed and contextual behaviours by extending the model of session types and taking advantage of the Ocaml type system [24]. Another example of his work is the implementation of an open source system for the automatic creation of interactive scores by using formal specifications [68].

Damiano Mazza (CR1-CNRS) [Tasks 1.1.3, 1.1.4, 1.2.2, 2.1.4, 3.1.1, 3.2.1] is a recognised expert in denotational semantics, with important contributions to graded and intersection type theories [20, 53]. Among the local members of the team, he is the most knowledgeable one regarding the CoGITARe's subjects, being able to work on most theoretical tasks.

With his additional experience in supervising several PhD students (always with impressive success) and his recently acquired *habilitation*, he is the perfect choice for the co-supervision of the first PhD-student. He is also the coordinator of the ELICA ANR project on related topics and will help the PI with administrative tasks.

Giulio Manzonetto (MCF) [Tasks 1.1.3, 1.2.2, 1.2.3, 2.1.4, 3.1.3] has a publication record witnessing his active scientific production and lively interactions within the community and with the PI in particular. A large panel of his research is focused on resource aware type systems as well as resource aware denotational and operational semantics [50, 52]. More precisely, he has a deep experience with intersection types [22] and with methods of approximations of functional programs [52] that will largely benefit the CoGITARe project.

He has administration and coordination skills, since he was principal investigator of the NWO project Calmoc, *directeur des études* at the IUT of Villetaneuse (Department R&T) and he supervised Master and Phd students, and postdocs; with his recently acquired *habilitation* he will help the PI in supervising the second PhD student.

Thomas Seiller (CR1-CNRS) [Tasks 1.1.2, 2.1.4, 3.1.1, 3.1.2, 3.2.1, 3.2.2] is a recognised expert in Mathematical Foundations of Computer Science, as witnessed by several publications in top tier journals and conferences. After holding a Marie Curie Individual Fellowship at the University of Copenhagen, he recently joined the CNRS as a CR1 research scientist. His main contributions have been in the study of mathematical models of computer programs and their dynamics, and can be understood as showing how concepts from computer science (e.g. computability, determinism, complexity constraint) correspond to notions in mathematics (e.g. sheaf condition, norm condition, subalgebras).

Since September 2017, the PI and Seiller started collaborating on the subject of categorical semantics of programming languages. The CoGITaRe project will benefit from Seiller's expertise on both categorical semantics and realisability models, and more particularly quantitative models of linear logic [65, 66]. In particular, Seiller's preliminary investigations on modelling dependent types in realisability models will provide grounds for a fruitful collaboration on **??**. In fact, the PI and Thomas Seiller already started collaborating in this direction with promising early results.

II.1.3 Non-local members of the team

The main weaknesses of the local team regarding the project are:

- their quasi nonexistent research experience with standard AI,
- their lack of experience regarding OCaml compiler (and practical functional programming in general),
- and their not-so-strong experience on dependent types.

In fact, even as a whole, the LIPN currently does not cover efficiently any of these areas. That is why the PI went looking for experts outside Paris13 whose role is mainly consultative.

Yann Regis-Gianas(MCF) [Tasks 1.1.2, 1.2.3, 2.1.4, 2.2.2, 2.2.3, 2.2.4] is a member of IRIF (Paris Diderot). He is an expert of implementations of functional programming languages [60], compiler certification [2], dependently type systems [25, 61], relational semantic analysis [40], and has some experience in abstract interpretation [39] and verification. He will be the main referent regarding the interaction with functional programming community.

Jérôme Feret (CR1-Inria) [Tasks 2.1.4, 1.2.3, 1.2.4] is an INRIA researcher in the project-team Antique, located at *École Normal Supérieure*. He graduated from École normale supérieure and did his PhD on the subject of static analysis of mobile systems there, under the supervision of Patrick Cousot. He has authored 56 publications (3600 citations, H-index of 26 according to Google Scholar). He is one of the developer of the ASTRÉE analyzer and of static analysis, causality analysis, and model reduction tools for rule-based languages.

He is interested in extending the scope of abstract interpretation to new topics [29]. He has designed abstractions for mobile systems, embedded systems, numerical computations, and models of signalling pathways. He was the laureate of the Junior ANR Chair of Excellence AbstractCell (2009-2013) and he is coordinating the ANR AnaStaSec (2015-2018). Due to his expertise in AI, he will be the main referent regarding the interaction with AI community.

II.1.4 Extended, international, circle of collaborators

Beyond the team members directly involved in the project, there are several researchers we wish to interact with. We present here a non-exhaustive list of people expected to collaborate on some tasks or provide expertise.

Jean-Vincent Loddo (LIPN) [51] Experienced OCaml programmer, he will help in supervising WP-2.
Micaela Mayero (LIPN) Expert in Coq proof assistants and dependant types. She can help with WP-1, and with the formalisation in Coq of especially technical proofs.
Dan Ghica (Birmingham): [35, 36] Currently in a collaboration with the PI regarding WP-1. He is a specialist in denotational semantics, GT and higher order languages for circuit design.[15]
Jan Midtgaard (Southern Denmark): [57, 67] One of the few leading expert on functional program analyses based on abstract interpretation. He can help in the articulation of WP-1 and WP-2.
Marco Gaboardi (Buffalo): [3, 20, 33, 34, 48, 62] Previous collaboration; one of the leaders of practical applications of GTs. He can help in the articulation of WP-1 to WP-2.
Paul-André Mellies, Thomas Ehrhard, Antonio Bucciarelli & Michele Pagani (IRIF): [18, 21, 23, 32, 41, 54, 55, 56] They have close relationships with the LIPN and the PI in particular. Having a rich theoretical experience on different kinds of graded types, they could help with WP-4.
Shin-ya Katsumata (Tokyo) [3, 32, 34, 46] Previous collaboration; one of the leaders of theoretical foundations of GTs. He can help in the articulation of WP-1 and WP-4.
Charles Grellois (LIS) [41, 49] Close but never formal collaborator of the PI. He is a specialist in denotational semantics, IT, and HO-recursive schemes; he can help in the the articulation of WPs 1 and 4.
Simona Ronchi Della Rocca & Mariangiola Dezani (Torino) [23] Recognised IT experts.
Alexis Ghyselen (LIX) A recently employed PhD student that is working on graded types.
Arnaud Spiwack (Tweag I/O), Thomas Blanc & Pierre Chambart (OCamlPro) T. Blanc is a former CIFRE (PhD student), advised by P. Chambard, on the use of global abstract interpretation analyses on OCaml programs; Arnaud Spiwack is currently working on graded-style linear type systems. They will maintain a link with the private companies OCamlPro and Tweag I/O.
Ugo Dal Labo (Bologna) [48, 49], Tarmo Uustalu (Talin) [34] & Dominic Orchard (Kent) [34, 59] Previous

[15] notice that the later requires heavy resources analysis at compile type, where GTs come in action.

collaborators of the PI and experts of GTs.

II.1.5 Employed researchers

A post-doc financed by CoGITARe The post-doc will work on the highly theoretical Task 3.1.2. However, depending on the candidate or the timeline, we may redistribute her/him over some other tasks.

A PhD financed by CoGITARe We intend to find this student by April 2019. We intend to place her/him on Task 1.1.1 for the duration of her/his master internship. Then, the thesis should begin in September, so the end of her/his funding should coincide with the end of the project. The advising will be a joint work from the PI and Damiano Mazza.

We believe that the thesis subject need to be adapted with respect to the student ambitions, capabilities and preferences. Among other possibilities, we distinguish two sets of tasks that we believe they should suit a PhD student: Tasks 1.1.3, 1.1.4, 2.2.1 and 2.2.2 (and maybe 2.2.4), for an applied thesis; and Tasks 3.1.3, 1.2.2 and 1.2.3 (and maybe 1.2.4) for a more theoretical one.

A PhD financed by LIPN On the condition that the first PhD starts with a healthy dynamic, the PI and G. Manzonetto will propose a candidate for "école doctorale"'s PhD fundings at the start of the second year. Notice that the policy of LIPN is to prioritise already active projects, thus favouring our application.

The internship should begin around April 2020 on Task 2.1.2.. Then, as for the other student, we will propose two possible sets of tasks: For an applied thesis, Tasks 2.1.2, 2.1.4 and 2.2.2 are suitable and can be followed by a strengthened collaboration with start-ups for the last year. As for a theorical thesis, Task 2.1.3 gives an interesting basis that can either be followed by denotational studies or by the integration of refinement types.

II.2 Means of Achieving the Objectives

We are beginning with a more precise description of each WP. These 3 WPs are further separated in 21 different tasks. This seems to be a lot, but it is actually manageable for several reasons:

- Some tasks at the beginning of the project[16] are already work in progress. Some of them may actually be close to their conclusion by the beginning of the project.
- Some tasks are partially redundant and target a unique deliverable. We separated them for practical reasons such as packages dependency[17] or version control.[18]
- Some tasks[19] are fuses: those are risky tasks that are not required for the continuation of the project. As long as no strong result is view, we can stop them anytime to concentrate our workforces to critical tasks.

II.2.1 WP-1. Generalising GT and integrating methods of AI.

As the title suggests, we can separate tasks of WP-1 into two sub-WPs: the generalisations of GT (WP-1.1) and the integration of AI methods (WP-1.2).

Task 1.1.1 : Game-like types. Game-like types are the result of a preliminary work. They seem to cover several weaknesses of graded monads and comonads while simulating both of them in specific contexts. As such, their development is important to get a uniform study. As a preliminary, advanced work, there are few risks involved.

[16] Tasks 1.1.1, 1.1.2, 1.1.3, 3.1.1 and 3.1.2
[17] Tasks 1.1.1-3.1.1, 1.1.2-3.1.2, 1.1.3-3.1.3-3.1.2 and 1.1.4-1.2.2
[18] Tasks 2.2.2, 2.2.3 and 2.2.4
[19] Tasks 1.2.4, 2.2.3, 3.2.1 and 3.2.2

Task 1.1.2: Dependant GT. See Section I.2.4 for more details. Adding dependency over resources is important to get more completeness/expressive power. At the end of his thesis, the PI did some preliminary work that can serve as bases, hence a medium risk is involved.

Task 1.1.3: IT as GT. See Section I.2.5 for more details. The whole project follows the remark that (a generalisation of) IT systems seems to act as "concrete" graded types. Here we intend to formalise this intuition.

Task 1.1.4: Deteriorating GT. See Section I.2.6 for more details. The best applications are deteriorations of rigid and theoretical structures. Here we look at which deteriorations are safe regarding our objectives; for example, subject reduction should be weakenable in presence of a suitable Galois relation.

Task 1.2.1: Learning about AI. The concrete knowledge of the PI regarding abstract interpretation is relatively poor. The PI thus intends to spend some time learning real competency on this field.

Task 1.2.2: Galois relations over GT. See Section I.2.5 for more details. While noticing that IT systems seem to act as "concrete" graded types, the "concreteness" stands for a natural Galois relations arising between types from graded comonads and IT. We intend to develop this relation (before looking at its degraded versions).

Task 1.2.3: Widening operators. With our notion of abstract domains, we should be able to interpret recursive calls through the use of widening operators. This should be the first, and most important, transfer of technology from AI. Then we could work on other related technologies such as narrowing.

Task 1.2.4: "GTising" traditional abstractions. Once these fundamental notions are transferred, we will look at transferring the traditional classes of domains; hopping that those correspond to classes of GT. This may be a risky move as we cannot predict the requirements at the actual time.

II.2.2 WP-2. Towards the grading of fully fledged programming languages

For the second WP, we can also separate tasks into two sub-WPs: the generalisation toward real languages (WP-2.1) and the implementation (WP-2.2).

Task 2.1.1: Learning about type inference. The PI is knowledgeable regarding type inference using unification or focusing, but the project requires to fully understand subtleties of type inference which he is not expert of (type refinement, use of SMT solvers...).

Task 2.1.2: Modularity and interaction with computational monads. Real languages, and OCaml in particular, use effects by default. In particular, they use I/O operations, exceptions and references. Those are monads and thus very particular forms of graded monads. This means that dealing with them should account to use modularity between GTs. The author already works on this question for the particular case of graded (co)monads [34], but this should be generalised.

Task 2.1.3: Polymorphism and type dependency. We expect our work to ultimately interact fruitfully with user-level type annotations, but before that we have to interact safely with those. The goal of this task is to define an acceptable level of polymorphism and (if possible) type dependency (different from resource dependency) for the tool to be applicable in real languages.

Task 2.1.4: Design of the full type system. This task simply intends to put all the pieces we defined earlier together in a complete, parametric system and to extract a language from there. This should not be a risky or too complex task, but it has to be a joint work of the full team.

Task 2.2.1: Implementation of a minimal analyser. First, we intend to implement a small prototype that analyses one of our three canonical cases of study on a minimal language (PCF potentially enriched with exceptions). The choice of the case will depend on our advancement and confidence hitherto.

Task 2.2.2: Implementation of a specific analyser. The second version should be targeting one of the OCaml intermediate compiling languages (potentially ignoring some minor of complex features).

Task 2.2.3: Implementation of a complete analyser. The last implementation (which is kind of optional) should implement a modular system in which we could choose the performed analyses. In particular our three canonical examples should be implementable.

Task 2.2.4: Benchmarks and comparison to real implementations. In the end of the project, we will perform some real benchmarks and compare with existing ones in OCaml or, if not available, in comparable C programs.

II.2.3 WP-3. Fruitful interaction with the semantics of type systems

The third WP is separated into two natural sub-WPs: the study of denotational semantics of GT and the spinoffs.

Task 3.1.1: Join semantics for graded monads and comonads. Two parallel works in progress: an investigation on the semantics of game-like types (Sec.I.2.3), and an investigation on the graded monads of profunctors.

Task 3.1.2: Semantics of dependant resources. Preliminary results in the PI's thesis on slices of linear categories (Sec.I.2.4), but still a lot of work to do.

Task 3.1.3: Concrete GT and categories. Semantic counterpart of Task 1.1.3: we aim at defining a clean, categorical notion of concrete GT system. This is a formalisation of an intuitive conscept in the context of the semantics studied in Tasks 3.1.1 and 3.1.2, without much risk involved.

Task 3.2.1: Locally linear categories. The eventuality of a decisive result makes the task important, but we cannot wage on it. Thus, we will perform a categorical formalisation and a study of the slices of linear categories and its parallel with locally cartesian close category and decide afterward whether to continue in this direction or not.

Task 3.2.2: Forthcoming spinoffs. Other potential spinoffs may emerge along with the study. With this task, we make clear the attention we give to any possible feedback. Nonetheless, this task is subject to the discovery of such spinoffs and the state of advancement of main tasks.

II.2.4 Timeline and deliverables

The Gantt diagram is given in Figure 1. As for the deliverables, in addition to publications in each (pairs of) task, we will develop a prototype. As described in WP 2.2, we are forseeing three successive versions.

II.2.5 Scientific justification of requested resources

172 419 € are dedicated to salaries: in addition to the post-doc (50 400€) and the PhD student (103 000€) whose roles are described in Section II.1.5, we are counting here two 5-months internships for the two PhD students (6 500€) and a decharge for the PI (12 519€).

8 000 € are dedicated to material. In addition to the two laptops (3 000€) for the PI and one student, we are investing in a more effective computer with a large RAM for the development and the benchmarks (2 500€); we are also budgeting a sum for buying books for the PI whose library is still non-existent (2 500€).

40 000 € are dedicated to missions. One part is more specifically associated to invitations of foreign collaborators (7 000) and another is to participate and send the students to summer schools (5 000€; see Section III.3), the remaining part (28 000€) will be used for attending conferences or workshops, or to visit collaborators.

A summary of the budget is described in Figure 2

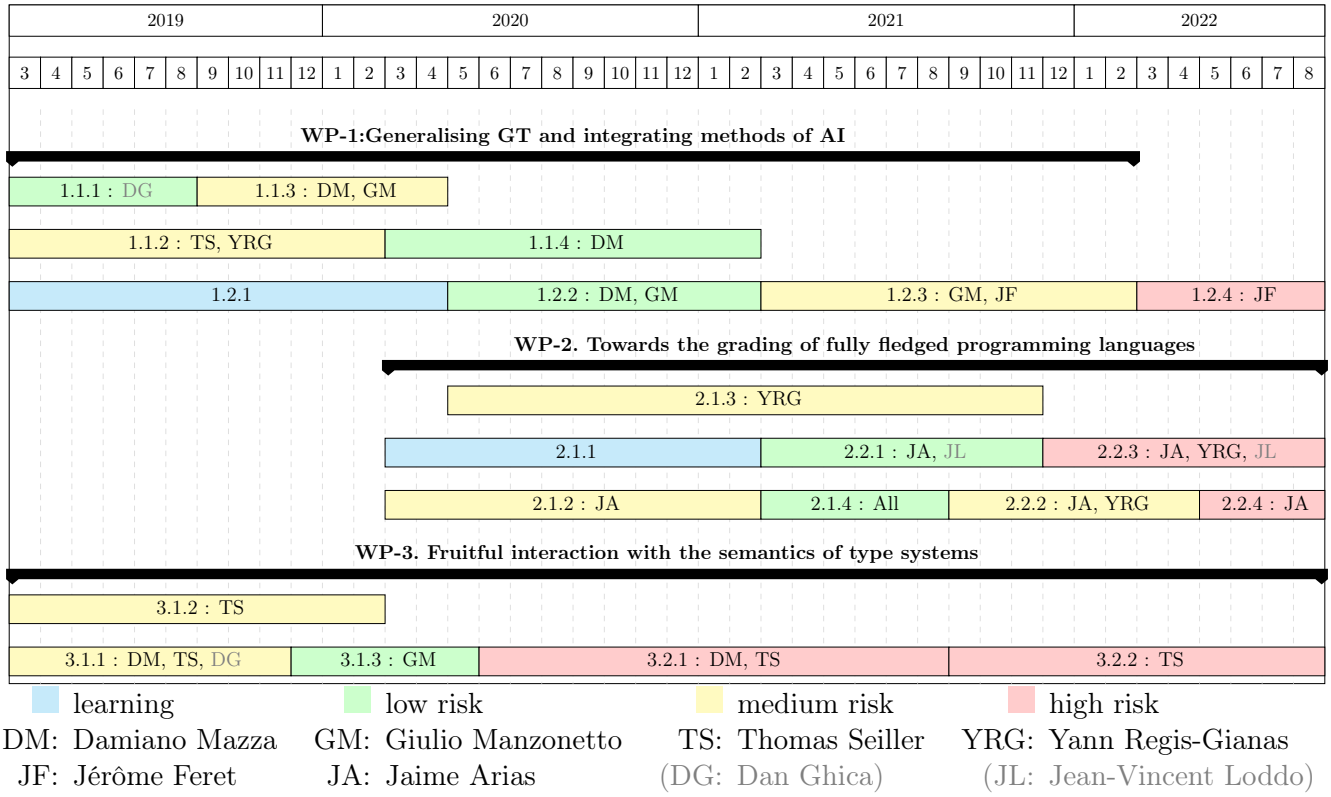

Figure 1: Gantt diagram

Staff expenses	post-doc	50 400€
	PhD	103 000€
	internships (x2)	6 500€
	décharge	12 519€
	sub-total	172 419€
Instruments and material costs	Laptops (x2)	3 000€
	Books	2 500€
	Computer (for Benchmarks)	2 500€
	sub-total	8 000€
Building and ground costs		0€
Outsourcing / subcontracting		0€
General and administrative costs & other operating expenses	invitations	7 000€
	summer school (2 students + 1 teacher)	5 000€
	other travel costs	28 000€
	Administrative management structure costs	0€
	sub-total	40 000€
Total		220 419€ +8%
Requested		238 053€

Figure 2: Detailed budget

III Impact and Benefits of the Project

III.1 Impact of the proposal

On the Scientific Community. We will be mostly conducting fundamental research with applied objectives, therefore, our proposal will lead for a large part to advances of a theoretical nature. As such, the outcomes of the proposal will directly benefit researchers working on logic in computer science, type systems, abstract interpretation and static analysis in general.

At the scale of specialised scientific communities, this proposal has an interdisciplinary nature since it connects with each other logic, functional programming and abstract interpretation communities. Even if pairwise interactions are quite rich (especially for logic and functional programming), projects relating each of them tend to become sparser and sparser due to the above-mentioned main barriers in the application of abstract interpretation to functional programming languages.

As discussed in the risk management section, the different milestones are of different risk intensities, but have impact proportional to those risks. The two first WPs have lower risks involved, and thus less impact. But it does not mean that this impact is null if the other WPs fails. In fact, even if lesser that the two others, this impact is already noticeable.

The WP-1, in addition to allow us to advance in our objectives, will have a large impact in the fundamental and practical approaches of graded types. Indeed, graded types, in their different and non-unified forms, recently triggered quite an interest of the community for specific but powerful applications [33, 41].

The WP-2 offers a new approach to the traditional barrier hindering the application of abstract interpretation to functional programming. As such, even if we are not able to fully break it with WP-3, we will definitely be able to enrich the theoretical understanding of this barrier.

Ultimately, we plan to develop a software prototype implementing an abstract analyser using this theory. If this prototype prove to be efficient, it will have a major impact on verification and compilation of functional programming languages. In fact, we do not need to be more efficient than existing tools to have a large impact. This is because we introduce two novel features: the partial compositionality of the method, and, more importantly, the future possibility of taking types annotations into account.

The WP-3 also may have huge impact with low probability. In fact, we do not intend to spare much time on this point if initial investigation fails, but the possibility of a positive result is worth a few months. Indeed, the question of unifying linearity (or effects) and dependant types (or GADTs), is one of the main open question of dependant type community: including Coq, Agda and HOTT communities, but also semanticists and theoretists for functional languages with some restricted dependency (such as GADTs). The main issue with this open question is that it is intrinsically ill-defined: one has to give a notion of effect for higher order terms appearing inside types while those are disappearing after compilation. Our answer may be an unexpected one: those effects are triggered at compile time and influence the full analysis of the program (including our abstract interpretation).

On the PI's Academic Career. The PI is a young researcher who obtained his PhD thesis in Oct. 2015, and who has a permanent position at Paris 13 since January 2015. The next five to ten years, then, will be absolutely crucial for the PI's career, and will ultimately decide whether the talent demonstrated by the PI can be fruitful for the French and international research communities in computer science.

Therefore, this project is designed to make the later as likely as possible. In particular, its fundational nature is a deliberated orientation aiming for a long term, more applied, project. Indeed, if the results of the 42-months CoGITARe project are successful, the PI intends to concretise his long term project as an ERC Starting Grant. In addition, this project will be the basis for the PI's *habilitation*, that the PI intends to start writing at the end of the project.

CoGITARe is also the opportunity, for the PI, to interact more intensively with the community and

to strengthen his position and visibility as a specialist of graded type systems.

On the LoVe team. It has been promised in the recent HCERES evaluation that the team LoVe (formerly LCR) is going through a break-up into two new teams : a team on specification and verification, and a team on logic and programming. The CoGITARe project is involving a large part of the later. It will then be an intellectual and financial engine necessary for driving safely the team through this critical junction that is this break-up. In particular, it will forge new openings toward logic, type systems and functional programming that are the targeted focus of the future team.

III.2 ANR-Plan d'Action 2018.

The CoGITARe project contributes directly to the Société de l'information et de la communication challenge (number B.7) of the ANR program along two axes, and it will have indirect repercussions in a third axe of the same challenge:

Axe 2: *Sciences et technologies des logiciels.* As a project focused on the verification and analysis of programs, we can say without doubts that CoGITARe is part of the second axis.

Axe 1: *Socle Fondements du numérique.* From its fundamental nature and its deep interactions with logic and semantics, it is also part of the third axis. In particular, WP-3 carries an original look over type theory and abstract interpretation; with expected implications in type theory, logic and semantics.

Axe 7: *Infrastructures de communication hautes performances.* To a lesser degree, this project interconnects the seventh axis. Indeed, we aim at capturing temporal and spacial bounds, as well as sequenciality and causal-independence; all these analyses being essential for an efficient distribution of resources over complex architectures.

Overall, our project can be integrated as a fundamental approach to the objectives of *Infrastructures de communication hautes performances (réseau, calcul et stockage), Sciences et technologies logicielles.*

III.3 Dissemination

Our results are mainly intended for communication to the research community. The contributions of the project will be published as scientific articles, in world-class journals and conferences. Conferences of particular relevance include POPL, LICS, ESOP, SAS, CSL, ICALP and FoSSaCS. Before the actual publication, our results will be the subject of technical reports, and be made available online through the open access archive HAL. We will moreover design and maintain a web site relative to the project, where we will announce and keep track of the publications and from where the prototypes will be rendered available.

In order to favour the dissemination of CoGITARe ideas, the members of the project will present their work in local, national and international scientific meetings. In addition, the PI will take an active role in the emergence of this new and dynamic community around grading system which he is already part of. In particular, we plan to actively participate on in organisation of workshop FOPARA.

CoGITARe's objective is to consolidate the theoretical basis for an emerging research direction, only then will we be able to engage into a more ambitious project with actual applicative goals. However, in the longer term, we anticipate that the most promising techniques developed during the project will be implemented and embodied in a syntactic analyser running on an intermediate compilation language of a main functional programming languages (in particular OCaml and Haskell). In this perspective, we will maintain a constant interaction with private companies (*OCamlPro* and *Tweag I/O*) interested in the advancements and results of the project.

Finally, the PI believes that another important dissemination strategy is to teach in summer schools the fundamental principles at work in the project, in order to spread them among the next generation of researchers on higher-order programming languages. Therefore, CoGITARe will fund every year the

participation of a project member to teach in one summer school of the area such as the École de Printemps d'Informatique Théorique, the European Summer School in Logic, the International School on Rewriting, or the Oregon Programming Languages Summer School (OPLSS). The later will be spotted in particular due to its importance in the area. In fact, a participation to OPLSS will be mandatory for the PhD students before their defense.

[1] S. Abramsky, "Abstract interpretation, logical relations and kan extensions," *J. Log. Comput.*, vol. 1, no. 1, pp. 5–40, 1990. DOI: 10.1093/logcom/1.1.5.
[2] R. M. Amadio and Y. Régis-Gianas, "Certifying and reasoning on cost annotations of functional programs," in *FOPARA: 72-89*, 2011.
[3] A. A. de Amorim, M. Gaboardi, *et al.*, "A semantic account of metric preservation," in *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL*, G. Castagna and A. D. Gordon, Eds., ACM, 2017, pp. 545–556, ISBN: 978-1-4503-4660-3. [Online]. Available: http://dl.acm.org/citation.cfm?id=3009890.
[4] J. Arias, M. Gúzman, and C. Olarte, "A symbolic model for timed concurrent constraint programming," *Electronic Notes in Theoretical Computer Science*, vol. 312, pp. 161–177, 2015. DOI: 10.1016/j.entcs.2015.04.010.
[5] K. Asada, "Arrows are strong monads," in *Proceedings of the 3rd ACM SIGPLAN Workshop on Mathematically Structured Functional Programming, MSFP@ICFP 2010, Baltimore, MD, USA, September 25, 2010.*, V. Capretta and J. Chapman, Eds., ACM, 2010, pp. 33–42, ISBN: 978-1-4503-0255-5. DOI: 10.1145/1863597.1863607.
[6] F. Breuvart, "Generalised dependant linear types: adding dependancy to graded comonads," Work in progress, Draft.
[7] ——, "Refining properties of filter models: sensibility, approximability and reducibility," submited.
[8] ——, "The resource lambda calculus is short-sighted in its relational model," in *Typed Lambda Calculi and Applications, 11th International Conference, TLCA 2013, Eindhoven, The Netherlands, June 26-28, 2013. Proceedings*, vol. 7941, Springer, 2013, pp. 93–108.
[9] ——, "On the characterization of models of $\mathcal{H}^*$," in *Joint Meeting CSL-LICS*, 2014.
[10] ——, "Dissecting denotational semantics: from the well-established $\mathcal{H}^*$ to the more recent quantitative coeffects," PhD thesis, Université Paris Diderot, 2015.
[11] ——, "On the characterization of models of H*: the semantical aspect," *Logical Methods in Computer Science*, vol. 12, no. 2, 2016, Invited.
[12] F. Breuvart and U. Dal Lago, "Intersection type systems for probabilistic λ-calculi," submited.
[13] F. Breuvart, U. Dal Lago, and A. Herrou, "On higher-order probabilistic subrecursion," journal, submited.
[14] ——, "On higher-order probabilistic subrecursion," in *20th International Conference, FOSSACS, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS*, ser. Lecture Notes in Computer Science, vol. 10203, 2017, pp. 370–386.
[15] F. Breuvart and D. Ghica, "Scheduled types: capturing sequenciality," Work in progress.
[16] F. Breuvart, G. Manzonetto, *et al.*, "New results on morris's observational theory: the benefits of separating the inseparable," in *1st International Conference on Formal Structures for Computation and Deduction, FSCD 2016, June 22-26, 2016, Porto, Portugal*, ser. LIPIcs, vol. 52, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016, 15:1–15:18.
[17] F. Breuvart, G. Manzonetto, and D. Ruoppolo, "Relational graph models at work," journal, submited.
[18] F. Breuvart and M. Pagani, "Modelling Coeffects in the Relational Semantics of Linear Logic," in *CSL*, 2015.
[19] F. Breuvart and T. Seiller, "Monads over profunctor : a semantics for arrows and more," Work in progress.
[20] A. Brunel, M. Gaboardi, *et al.*, "A core quantitative coeffect calculus," in *23rd European Symposium on Programming, ESOP, Held as Part of ETAPS*, Z. Shao, Ed., ser. Lecture Notes in Computer Science, vol. 8410, Springer, 2014, pp. 351–370, ISBN: 978-3-642-54832-1. DOI: 10.1007/978-3-642-54833-8_19.
[21] A. Bucciarelli and T. Ehrhard, "On phase semantics and denotational semantics: the exponentials," *Ann. Pure Appl. Logic*, vol. 109, no. 3, pp. 205–241, 2001.
[22] A. Bucciarelli, T. Ehrhard, and G. Manzonetto, "Not enough points is enough," in *Computer Science Logic, 21st International Workshop, CSL 2007, 16th Annual Conference of the EACSL, Lausanne, Switzerland, September 11-15, 2007, Proceedings*, J. Duparc and T. A. Henzinger, Eds., ser. Lecture Notes in Computer Science, vol. 4646, Springer, 2007, pp. 298–312, ISBN: 978-3-540-74914-1. DOI: 10.1007/978-3-540-74915-8_24.
[23] A. Bucciarelli, D. Kesner, and S. R. D. Rocca, "The inhabitation problem for non-idempotent intersection types," in *8th IFIP TC 1/WG 2.2 International Conference, TCS*, 2014, pp. 341–354. DOI: 10.1007/978-3-662-44602-7_26.
[24] M. Cano, J. Arias, and J. A. Pérez, "Session-based concurrency, reactively," in *37th IFIP WG 6.1 International Conference on Formal Techniques for Distributed Objects, Components, and Systems (FORTE 2017), Neuchâtel, Switzerland, June 19-22, 2017*, ser. Lecture Notes in Computer Science, vol. 10321, Springer, 2017, pp. 74–91. DOI: 10.1007/978-3-319-60225-7_6.
[25] G. Claret, L. D. C. González-Huesca, *et al.*, "Lightweight proof by reflection using a posteriori simulation of effectful computation," in *ITP2013: 67-83*.
[26] M. Coppo and M. Dezani-Ciancaglini, "A new type assignment for λ-terms," *Archiv für mathematische Logik und Grundlagenforschung*, vol. 19, no. 1, pp. 139–156, 1978.
[27] P. Cousot and R. Cousot, " Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints," in *POPL*, 1977, pp. 238–252.
[28] P. Cousot and R. Cousot, "Comparing the galois connection and widening/narrowing approaches to abstract interpretation," in *Programming Language Implementation and Logic Programming, 4th International Symposium, PLILP'92, Leuven, Belgium, August 26-28, 1992, Proceedings*, M. Bruynooghe and M. Wirsing, Eds., ser. Lecture Notes in Computer Science, vol. 631, Springer, 1992, pp. 269–295, ISBN: 3-540-55844-6. DOI: 10.1007/3-540-55844-6_142.
[29] P. Cousot, R. Cousot, *et al.*, "Why does astrée scale up?" *Formal Methods in System Design*, vol. 35, no. 3, pp. 229–264, 2009.
[30] A. Dudenhefner and J. Rehof, "Intersection type calculi of bounded dimension.," in *POPL*, 2017, pp. 653–665.

[31] T. S. Freeman and F. Pfenning, "Refinement types for ML," in *Proceedings of the ACM SIGPLAN'91 Conference on Programming Language Design and Implementation (PLDI)*, D. S. Wise, Ed., ACM, 1991, pp. 268–277, ISBN: 0-89791-428-7. DOI: 10.1145/113445.113468.
[32] S. Fujii, S. Katsumata, and P. Melliès, "Towards a formal theory of graded monads," in *Foundations of Software Science and Computation Structures - 19th International Conference, FOSSACS, Held as Part of ETAPS*, B. Jacobs and C. Löding, Eds., ser. Lecture Notes in Computer Science, vol. 9634, Springer, 2016, pp. 513–530. DOI: 10.1007/978-3-662-49630-5_30.
[33] M. Gaboardi, A. Haeberlen, *et al.*, "Linear dependent types for differential privacy," in *POPL*, 2013, pp. 357–370.
[34] M. Gaboardi, S. Katsumata, *et al.*, "Combining effects and coeffects via grading," in *ICFP*, 2016.
[35] D. R. Ghica and A. I. Smith, "Geometry of synthesis III: resource management through type inference," in *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL*, T. Ball and M. Sagiv, Eds., ACM, 2011, pp. 345–356. DOI: 10.1145/1926385.1926425.
[36] ——, "Bounded linear types in a resource semiring," in *23rd European Symposium on Programming, ESOP, Held as Part of ETAPS*, Z. Shao, Ed., ser. LNCS, vol. 8410, Springer, 2014, pp. 331–350. DOI: 10.1007/978-3-642-54833-8_18.
[37] J.-Y. Girard, "Normal functors, power series and λ-calculus," *Annals of pure and applied logic*, vol. 37, no. 2, pp. 129–177, 1988.
[38] J.-Y. Girard, A. Scedrov, and P. J. Scott, "Bounded linear logic: a modular approach to polynomial-time computability," *Theoretical computer science*, vol. 97, no. 1, pp. 1–66, 1992.
[39] T. Girka, D. Mentré, and Y. Régis-Gianas, "A Mechanically Checked Generation of Correlating Programs Directed by Structured Syntactic Differences," in *ATVA*, 2015.
[40] T. Girka, D. Mentré, and Y. Régis-Gianas, "Verifiable semantic difference languages," in *PPDP: 73-84*, 2017.
[41] C. Grellois and P. Melliès, "Indexed linear logic and higher-order model checking," in *Proceedings Seventh Workshop on Intersection Types and Related Systems, ITRS 2014, Vienna, Austria, 18 July 2014.*, J. Rehof, Ed., ser. EPTCS, vol. 177, 2014, pp. 43–52. DOI: 10.4204/EPTCS.177.4.
[42] D. V. Horn and M. Might, "Abstracting abstract machines," in *Proceeding of the 15th ACM SIGPLAN international conference on Functional programming, ICFP*, P. Hudak and S. Weirich, Eds., ACM, 2010, pp. 51–62, ISBN: 978-1-60558-794-3. DOI: 10.1145/1863543.1863553.
[43] J. Hughes, L. Pareto, and A. Sabry, "Proving the correctness of reactive systems using sized types," in *Proceedings of the 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, ser. POPL '96, St. Petersburg Beach, Florida, USA: ACM, 1996, pp. 410–423, ISBN: 0-89791-769-3. DOI: 10.1145/237721.240882.
[44] J. M. E. Hyland and C.-H. Ong, "On full abstraction for PCF: I, II, and III," vol. 163, no. 2, pp. 285–408, 2000.
[45] B. Jacobs, *Categorical logic and type theory*. Elsevier, 1999, vol. 141.
[46] S. Katsumata, "Parametric effect monads and semantics of effect systems," in *POPL*, 2014.
[47] N. Kobayashi and C.-H. L. Ong, "A type system equivalent to the modal mu-calculus model checking of higher-order recursion schemes," in *LICS*, 2009.
[48] U. D. Lago and M. Gaboardi, "Linear dependent types and relative completeness," 4, vol. 8, 2011. DOI: 10.2168/LMCS-8(4:11)2012. [Online]. Available: https://doi.org/10.2168/LMCS-8(4:11)2012.
[49] U. D. Lago and C. Grellois, "Probabilistic termination by monadic affine sized typing," in *26th European Symposium on Programming, ESOP, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS*, H. Yang, Ed., ser. Lecture Notes in Computer Science, vol. 10201, Springer, 2017, pp. 393–419. DOI: 10.1007/978-3-662-54434-1_15.
[50] J. Laird, G. Manzonetto, *et al.*, "Weighted Relational Models of Typed Lambda-Calculi," in *LICS*, 2013.
[51] J.-V. Loddo, *Marionnet Network Simulator*, www.marionnet.org.
[52] G. Manzonetto and M. Pagani, "Böhm Theorem for Resource Lambda Calculus through Taylor Expansion," 2011.
[53] D. Mazza and L. Pellissier, "Polyadic approximations, fibrations and intersection types," Accepted in POPL'18.
[54] P. Melliès, "A micrological study of negation," *Ann. Pure Appl. Logic*, vol. 168, no. 2, pp. 321–372, 2017. DOI: 10.1016/j.apal.2016.10.008.
[55] ——, "The parametric continuation monad," *Mathematical Structures in Computer Science*, vol. 27, no. 5, pp. 651–680, 2017. DOI: 10.1017/S0960129515000328. [Online]. Available: https://doi.org/10.1017/S0960129515000328.
[56] P. Melliès and N. Zeilberger, "Functors are type refinement systems," in *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL*, ACM, 2015, pp. 3–16. DOI: 10.1145/2676726.2676970.
[57] J. Midtgaard, "Control-flow analysis of functional programs," *ACM Computing Surveys*,
[58] F. Nielson, H. Nielson, and C. Hankin, "Principles of program analysis," *Springer*, 2015.
[59] T. Petricek, D. Orchard, and A. Mycroft, "Coeffects: a calculus of context-dependent computation," in *Proceedings of International Conference on Functional Programming*, ser. ICFP, Gothenburg, Sweden, 2014.
[60] F. Pottier and Y. Regis-Gianas, "Stratified type inference for generalized algebraic data types," in *POPL*, 2006.
[61] F. Pottier and Y. Régis-Gianas, "Stratified type inference for generalized algebraic data types," in *POPL: 232-244*, 2006.
[62] I. Radicek, G. Barthe, *et al.*, "Monadic refinements for relational cost analysis," *PACMPL*, vol. 2, no. POPL, 36:1–36:32, 2018. DOI: 10.1145/3158124. [Online]. Available: http://doi.acm.org/10.1145/3158124.
[63] P. M. Rondon, M. Kawaguchi, and R. Jhala, "Liquid types," in *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, ACM, 2008, pp. 159–169. DOI: 10.1145/1375581.1375602.
[64] D. Scott, *Continuous lattices*. Springer, 1972.
[65] T. Seiller, "From dynamic to static semantics, quantitatively."
[66] ——, "Interaction Graphs: Full Linear Logic," in *LICS '16,2016*, 2016, pp. 427–436.
[67] I. Sergey, D. Devriese, *et al.*, "Monadic abstract interpreters," in *ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '13*, H. Boehm and C. Flanagan, Eds., ACM, 2013, pp. 399–410, ISBN: 978-1-4503-2014-6. DOI: 10.1145/2491956.2491979.
[68] *Vmo-score*. [Online]. Available: https://vmo-score.github.io/.
[69] S. Zdancewic and A. C. Myers, "Secure information flow via linear continuations," *Higher-Order and Symbolic Computation*, vol. 15, no. 2-3, pp. 209–234, 2002. DOI: 10.1023/A:1020843229247.