### PART A

1. Line of intervention  
   **Main line/Linea Principale**

2. Research project title  
   **Resource Awareness in Programming: Algebra, Rewriting, and Analysis**

3. Duration of the project (months)  
   **24 months**

4. Strategic emerging Topics - 5. Related Cluster

<table>
<thead>
<tr>
<th>Strategic emerging topic:</th>
<th>CIRCULAR ECONOMY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster:</td>
<td>Digital, Industry and Space</td>
</tr>
<tr>
<td>Sub Cluster:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2. Globally attractive, secure and dynamic data-agile economy by developing and enabling the uptake of the nextgeneration computing and data technologies and infrastructures, including space infrastructure and data), enabling the European single market for data with the corresponding data spaces; and a trustworthy artificial intelligence ecosystem.</td>
</tr>
</tbody>
</table>

6. Main ERC field  
   **PE - Physical Sciences and Engineering**
Software is ubiquitous in our lives and there is virtually no aspect of society where software does not play a key role. This, however, is in striking contrast with the extraordinary difficulty of certifying safety and reliability of software systems, a crucial task to ensure the sustainability of our digital ecosystem. In fact, the massive presence of software and its exponential growth rate, its heterogeneity (think about data-driven computation), and the growing awareness of its social, environmental, and ethical impact (as embodied by privacy, security, and energy issues) are outlining a new world of software safety criteria that go significantly beyond traditional input-output-based analysis. In the last decade, researchers have realised that the aforementioned phenomena can be uniformly understood in terms of a paradigm shift in Computer Science, whereby data and information are treated as physical resources, rather than as logical entities. Resource-awareness is indeed at the heart of many intensional behaviours – such as information leakage and data privacy – as well as to the energy consumption and environmental impact of computation. Resource-awareness is not only crucial at the syntactic level of data manipulation: it is also vital to make semantic program analysis sustainable and in line with the exponential growth rate of software. To face the new challenges of software safety and sustainability, consequently, resource-awareness has to play a key role in programming language syntax, semantics, and analysis.

Traditional theories of programs are, by their very foundation, inherently resource-agnostic. And even if theories of resources in program syntax and semantics – such as theories of monoidal syntax, on the syntax side, and of coeffects, on the semantic side – have been recently proposed, they all focus on specific traits of resource-awareness lacking, above all, to truly model resources both syntactically and semantically. All of that has also prevented the development of resource-aware program analyses, which is crucial to ensure software safety.

By exploiting the synergies of resource-awareness in syntax, semantics and analysis, the goal of Resource Awareness in Programming (RAP) is to develop novel techniques, methodologies and automatic tools to measure, certify or refute nowadays desirable properties of modern software systems. In particular, RAP goals are: the introduction of algebraic resource-aware semantics for resource-sensitive syntax, viz. string diagram and monoidal syntax; the development of resource-aware rewrite systems for monoidal syntax and its operational properties; the introduction of novel resource-aware program analyses in the form of resource-based abstract interpretation and program logic.

<table>
<thead>
<tr>
<th>Associated Investigator</th>
<th>item A.1</th>
<th>item A.2</th>
<th>item B</th>
<th>item C</th>
<th>item D</th>
<th>item E</th>
<th>item F</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZANASI Fabio</td>
<td>42.002</td>
<td>69.750</td>
<td>0</td>
<td>0</td>
<td>16.763</td>
<td>0</td>
<td>3.430</td>
<td>131.945</td>
</tr>
<tr>
<td>GAVAZZO Francesco</td>
<td>55.375</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8.306</td>
<td>0</td>
<td>6.300</td>
<td>69.981</td>
</tr>
<tr>
<td>BALDAN Paolo</td>
<td>41.975</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6.296</td>
<td>0</td>
<td>3.000</td>
<td>51.271</td>
</tr>
<tr>
<td>GIACOBazzi Roberto</td>
<td>38.028</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5.704</td>
<td>0</td>
<td>3.000</td>
<td>46.732</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>177.380</td>
<td>69.750</td>
<td>0</td>
<td>0</td>
<td>37.069</td>
<td>0</td>
<td>15.730</td>
<td>299.929</td>
</tr>
</tbody>
</table>

N.B. The Item D and TOTAL columns will be filled in automatically
- item A.1: enhancement of months/person of permanent and temporary employees
- item A.2: cost of contracts of non-employees, specifically to recruit
- item B: cost of equipment and tools
- item C: cost of consulting and other services
- item D: overhead
- item E: materials cost
- item F: other costs

**PART B**

**B.1**

1. State of the art

Traditional program syntax, semantics, and analysis are – by their very foundation in terms of cartesian categories, classic, and intuitionistic logic – resource-agnostic. Nonetheless, there have been proposals to
In the last decades, there has been an increasing interest both in syntactic and semantic resource awareness, although these two sides have been studied separately. Concerning syntactic resource awareness, researchers have progressively realised that traditional approaches to programming language syntax do not cope well with non-classical notions of computation, such as those originating in quantum computing, as well as with the diagrammatic languages used by engineers for designing different kinds of networks, such as those for electrical circuits, signal flow graphs, and Bayesian networks. String diagrams [Sel10] emerged as a unifying, resource-aware formal syntax [BD08, BE15, BSZ15, MCG19, BHP+19, PF21] originating in the theory of monoidal categories. The use of monoidal syntax allows for an explicit representation of variables copying and erasing which is instead implicit in traditional cartesian syntax. For instance, the two distinct string diagrams below are both represented as the traditional term \( f(g(x, x), y) \).

Monoidal syntax thus allows for treating data as physical entities that, in general, cannot be freely copied and erased.

On the semantic side, resource-awareness has been identified as crucial to model a large family of intensional program behaviour, including those related to information-flow [ABH+99], differential privacy [RP10], and code analysis [OLE19]. Resource-awareness also plays a crucial role in quantitative and metric reasoning [Gav18], to the point that abstract quantitative and resource-based reasoning are equivalent [DG22a]. All of that has led to the development of several semantic theories and techniques to deal with resource-awareness and quantitative reasoning. Among those, of particular relevance are coeneffectful static semantics [OLE19], program distances [Gav18, DG22], and quantitative rewriting [GD23] and equational theories [MPP16].

Perhaps surprisingly, syntactic and semantic resource-awareness have not been combined into general theories of resources, this way constraining the applicability and effectiveness of resource-sensitive reasoning. On the one hand, semantics of string diagrams and monoidal syntax are given by means of ordinary equational theories and rewriting [Bur93, Mim14, BGK+22], this way lacking semantic resource-awareness. On the other hand, quantitative equational theories and rewriting – which are both prominent examples of resource-aware semantics – are defined for cartesian syntax only. All of that has not only severely constrained resource-aware semantic reasoning, but it has also prevented the development of resource-aware techniques of program analysis, especially those akin to abstract interpretation and program logic.

Program analysis has been studied for over half a century as a major method to aid programmers and software engineers to produce reliable software. In order to make decidable or simply tractable, otherwise undecidable (or way too complex) program properties, the basic idea is to work on approximations of the original program. A successful example of approximation is abstract interpretation [Cou21], based on the simple but effective idea that extracting properties of programs means (over-)approximating their semantics. This approach is sound, i.e., a program deemed correct will be so, but it might raise false alarms (i.e., false positives) due to strict over-approximation. A further example is incorrectness logic [OHe20], a proof system for detecting undesired behaviours (e.g. bugs) in programs. Being based on an under-approximating of the program semantics, such program logic suffers from the opposite problem of abstract interpretation: it may raise no alarms even when the program does not behave correctly, i.e., it may display false negatives. Program logics have been proposed which deal both with correctness and incorrectness, like [BGGR21] combining incorrectness logic and abstract interpretation and [ZDS22] investigating a monoidal generalisation of Hoare’s logic as a unifying foundation for correctness and incorrectness reasoning. Some work has been done in the direction of quantifying the imprecision of program analysis by leveraging abstract domains enhanced with a notion of distance [LPL09, CML+19, CDG22].

2. Detailed description of the project: methodologies, objectives, and results that the project aims to achieve; indicate deliverables and milestones outlining the project coherence as to the strategic themes, indicating clear and innovative objectives, setting out the project sector relevance and its positioning with reference to the state of art, describing the role and contribution of each research unit

**Introduction and Motivation**

Resource-awareness is at the heart of modern software safety and sustainability. Treating data as consumable resources both in program syntax and semantics is the key to make software systems reliable and trustworthy, as witnessed by the recent development of resource-aware type systems [OLE19] to ensure data privacy [PD10] and avoid information leakage [ABH+99]. Resource-awareness is also crucial to understand the energy consumption and environmental impact of computation, as first theorised by Landauer [Lan61], as well as to obtain better analyses of complex software systems, such as quantum and cyber-physical systems.
The benefits of semantic and syntactic resource-awareness actually go significantly beyond the aforementioned ones: in fact, they are not separate concepts. Quite the contrary, they are highly interconnected notions. On the one hand, languages employing a resource-aware syntax can be better analysed relying on resource-aware semantics; on the other hand, by injecting resource constraints into the syntax of a language – via, e.g., suitable type systems – better resource aware semantics and analyses can be obtained.

For those reasons, several resource-aware approaches to program syntax and semantics have been recently proposed (see State of the Art). All such proposals, however, cover only specific syntax or semantic issues, leaving program reasoning largely resource-agnostic. For instance, string diagrams and monoidal syntax constitute one of the most successful approaches to syntactic resource-awareness; their semantics, however, is mostly given by means of resource-agnostic equational and rewriting theories, this way considerably restricting their applicability. Conversely, quantitative equational and rewriting theories provide general forms of resource-aware semantics that are however given for resource-agnostic, cartesian syntax only. This makes reasoning highly nontrivial, as one has to endow languages with ad hoc type systems to cope with the lack of syntactic resource-sensitivity.

All of that witnesses how the lack of combined syntactic and semantic resource-awareness deeply impact the effectiveness of program reasoning. But this is not the end of the story; in fact, it has also prevented the development of resource-aware theories of program analysis, such as abstract interpretation and program logic. Indeed, resource-awareness is not only crucial in reasoning and modelling; it is also vital to verification by making program analysis sustainable and in line with the exponential growth rate of software. In fact, current program analysis mostly focuses on exact properties of software that are meant to guarantee that programs will behave as expected in any possible situation. This makes analysis expensive and time-consuming, as program behaviours have to be tested even for scenarios that are not practically feasible, and whenever the analysis relies on an approximated semantics no quantitative guarantee is provided about the quality of the result. By making program semantics resource-aware, analysis can be dependent on the available resources, such as security permissions of an external observer, the available hardware, or (quantitative) precision of the observation. This naturally leads to improvements of the analysis, both in terms of efficiency and of applicability: for instance, if made resource aware, the same analysis can work for different programs up-to a given resource or error threshold – and improving its efficiency; alternatively, efficiency of the analysis can be traded for precision, in a measurable way.

All of that makes resource-aware program syntax, semantics, and analysis not ready to face the new challenges posed by software safety, reliability, and sustainability issues, where having uniform accounts to resource-awareness is crucial. The goal of Resource Awareness in Programming (RAP) is to develop novel techniques combining resource-aware syntax, semantics, and analysis to guarantee software safety, reliability, and sustainability. All of that will be achieved both by designing novel resource-aware theories of programs and by combining them into a general framework for resource-based reasoning.

Objectives

By exploiting the synergies of resource awareness in syntax, semantics and analysis, RAP goal is to develop novel techniques, methodologies and automatic tools to measure, certify or refute nowadays desirable properties of modern software systems.

More precisely, RAP will make resource-aware the following three-steps process exploited in traditional approaches to program compilation/analysis.

- Programs are regarded as terms, i.e., syntactic entities, of a language. A set of axioms over terms defines a notion of program equivalence which respects the intended semantics.
- Programs are transformed by means of term rewriting. By taking the axioms in 1) as rewriting rules, program transformation is guaranteed to preserve the semantics, namely programs are transformed into semantically equivalent programs.
- Then program-transformation can be safely exploited by the compiler, to optimise/speed-up the execution of a programme, or by the analyser to make the analysis feasible or more efficient, for instance, by transforming the programme in a normal form like, e.g., the one by Bohm and Jacopini.

Making resource-aware each of the three above points is the main goal of each of the three work packages of RAP. These are shortly described below, but detailed descriptions are given in the next section:

- (WP1) To make point 1) above resource-aware, we need to replace “terms” by string diagrams, i.e., monoidal resource-aware syntax. Moreover, rather than considering program “equivalence” one should focus on program distances. Finding an effective way to axiomatise such distances on string diagrams is the main challenge of this WP.
- (WP2) The development of resource-aware rewriting for string diagrams, and the study of operational and algorithmic properties will allow to transform programs into not semantically “equivalent”, but “sufficiently close”, programs.
- (WP3) The development of resource-aware abstract interpretation – namely a novel theory of program analysis parameterised on the available semantic resources – and the study of its meta-theory will allow to devise sound analysis with quantitative precision guarantees. RAP will also develop resource-aware program logic and relate it to resource-aware abstract interpretation by means of resource-sensitive soundness and completeness results.

Methodology

To achieve its goals, RAP will follow a structural approach to resource-awareness, whereby program semantics and analysis will be given with respect to general notions of semantic resources. More
specifically, whereas syntactic resource-awareness is achieved by imposing strict control on copying and discarding of syntactic elements, semantic resource-awareness is obtained by enriching models on collections of available resources, which can be uniformly modelled as suitable algebraic and categorical structures. Examples of resources RAP is interested in include physical and computational resources, tolerable errors and degrees of precision in observation, and security levels. Those will allow RAP to model, e.g., semantics and analysis with limited precision as well as analysis that are sound only up to a given class of errors. All these examples can be abstractly modelled by enriching semantics (e.g. rewriting and equations) and analysis (e.g. abstract interpretation and logical satisifiability) on algebraic structures such as quantales [HST14] and preorder semiring [OLE19,DG22a], or even on suitable monoidal categories [HST14].

Parameterising semantics and analysis on suitable algebras and categories of resources has been proven to be an effective strategy to achieve resource-awareness when dealing with, e.g., program equivalence [Gav18,DG22a] and type systems [OLE19]. RAP will follow the same methodology to develop the aforementioned resource aware theories of algebra, rewriting, and analysis, this way building upon a firm mathematical foundation. Consequently, all three RAP research lines will be pursued relying on an unique vocabulary and mathematical framework, this way easing contaminations between them and collaborations between their associated research units.

Work Packages, Milestones, and Deliverables

RAP consists of four cooperating research units (UNIPI, UNIBO, UNIVR, UNIPD) whose work is structured into three work packages, each aiming to achieve a milestone realising the research lines previously outlined. UNIBO will lead WP1, and it will work on it together with UNIPI. UNIPI will also lead WP2, whereas UNIVR will lead WP3 closely collaborating with UNIPD, UNIPI, and UNIBO on it. Each WP, whose detailed description is given below, aims to achieve a milestone. To ensure a uniform and continuous progress, each WP is divided into specific tasks, and any task is associated with a deliverable based on technical articles published, or aimed for publication, in international journals and/or conference proceedings; in addition to that, software artefacts based on RAP case studies will be produced as further deliverables. RAP timeline and workplan are given in B1.3. Below, a detailed description of each WP, its tasks, and the milestones it aims to achieve is given.

WP1: Quantitative Equational Reasoning on Resource-Sensitive Syntax

Originating in the theory of monoidal categories, string diagrams are a graphical, yet completely formal language [Sel10] that, in the last years, has emerged as a unifying, resource-aware formal syntax [BD08, BE15, BSZ15, MCG19, BHP+19, PF21]. Similarly to traditional syntax, string diagrams are primarily manipulated using equational theories. For instance, a great amount of research has been devoted to finding complete axiomatization for diagrammatic calculi of quantum processes [BD08, JPV18]; another example is the equational theory of signal flow graphs, which allows to study semantic refinement [BHPS17], realisability [BSZ15], as well as controllability [FSR16] of cyber-physical systems.

So far, research on these equational theories has focussed on exact equality. The goal of this work package is to develop a quantitative algebra for string diagrams, allowing to reason more generally about distance between systems. In a nutshell, instead of reasoning about resource-sensitive computational processes in an exact domain — where we may only ask whether two processes are equal or not — our framework will allow to consider them in a domain enriched with a metric, where we may check statements like "A is at distance at most x from B".

In order to motivate and inform the theoretical developments (in T1.1), we will enrich in a quantitative setting string diagrammatic approaches to different computational models: signal flow graphs (T1.3), neural networks (T1.4), and the resource calculus (T1.2) which will then be used as a case study for the remaining WPs.

TASK 1.1 Axiomatising distances in string diagrams. String diagrams usually come with an axiomatisation provided by what is called in categorical jargon a “monoidal theory”. By enriching such theories over algebraic structures that carry quantitative information, such as quantales, we expect to find a systematic way to specify distances for string diagrams. We will borrow inspiration from the seminal work in [MPP16] that only deals with resource-agnostic syntax. In tackling this question we will also rely on the team's expertise on monoidal algebra [BSZ17, BSZ18] and categorical semantics with metric spaces [BBKK14,DG22]. It will also be helpful to study the analogies with approaches to quantitative reasoning using linear logic [DP22].

TASK 1.2 Resource calculus. The calculus of resources [Pie18, BHP+19] is a simple model of computation focusing on resource consumption. Resources are discrete and bounded quantities that are exploited by linear dynamical systems. By means of a sound and complete axiomatisation, one can prove that two systems consume the same amount of resources under the assumption that the resources available in the environment are unlimited. However, when the environment has a bounded quantity of resources, two systems with the same behaviour would be regarded as different. Our idea is to define a distance on systems measuring how many resources are necessary to observe differences. Our aim is to identify such distance and, exploiting the general framework developed in T1.1, axiomatise it. This would serve as case
studies for the remaining workpackages in T2.3 and T3.3.

TASK 1.3 Cyber-physical systems. We will develop a quantitative semantics and equational theory to study circuit optimisation and controllability for signal flow graphs, building from the work already done in an exact setting in [BSZ15, FSR16]. The literature on control theory (see e.g. [KB14] for an overview) offers a number of metrics to compare and control these models. The novelty of our approach will be to cast these metrics in an equational setting, developing axiomatisations and thus offering purely algebraic ways to specify and prove their properties. Since the algebraic theories of the resource calculus [BHP+19] in T1.2 and the one for signal flow graphs [BSZ15] are closely related, we expect that also the axiomatization of their distances would be closely connected.

TASK 1.4 Machine learning with neural networks. We will build on the recent categorical framework of [CGG+22], which formalises training of neural networks by gradient-descent as composition of appropriate string diagrams in a category of lenses. We will develop quantitative equational theories for these diagrams, with the aim of measuring uncertainty in training — questions such as where a parameter might lie, how close are we to the correct model, how accurate are the predictions of the trained model. Such a framework for quantitative reasoning will inform verification of machine learning properties, by creating guarantees that, for instance, the uncertainty in learning is reduced as training is increased, and that this process converges in the limit.

WP2: Resource-Aware Rewriting and Operational Reasoning

The goal of WP2 is to develop a theory of resource-aware rewrite systems and to study their applications to resource-sensitive equational theories. Traditional rewriting [BKB+03] studies discrete transformations between objects in a resource-agnostic way, modelling transformations as binary relations—called reductions—between objects.

To model quantitative and resource-sensitive forms of rewriting where reductions may consume resources and produce errors, Gavazzo and Di Florio [GD23] have recently introduced a general theory of resource-sensitive, quantitative rewrite systems. In a quantitative rewrite system, reduction is modelled by means of relations enriched on quantales, i.e. abstract models of distances and resources [HST14]. Even if nontrivial results have already been proved (such as resource-aware refinements of major confluence theorems), the theory of quantitative and resource-aware rewrite systems is still in its infancy and a large body of classic rewriting results still lacks resource-aware counterparts. Moreover, such a theory has been studied for cartesian, first-order syntax only, oftentimes requiring to introduce ad hoc restrictions to ensure consistency of resource consumption somehow mimicking constraints given by string diagrams and monoidal syntax. The goal of WP2 is to further develop a theory of quantitative and resource-aware rewrite systems, both in general and on resource-sensitive syntax, and to study its applications to resource-aware equational theories.

Task 2.1. Quantitative Rewriting, Beyond Cartesian Syntax. The goal of this task is to extend the current theory of quantitative rewriting beyond first-order and cartesian syntax. To achieve such a goal, this task will develop a general theory of quantitative rewriting parametric on suitable categories and notions of syntax, modelled as monads and initial algebras. Categories will be used to specify the nature of syntactic expressions: for instance, categories of presheaves will be used to model expressions with binders, and thus higher-order languages; whereas categories of spans will be used to model string diagrams, and thus monoidal syntax. Once a category is fixed, syntactic terms will be defined through initial algebras of suitable endofunctors, a standard methodology in formal semantics. Finally, quantitative and resource-aware rewriting relations will be modelled by extending the aforementioned categories to suitable quantalooids [HST14] or monoidal allegories — the latter capturing general notions of quantitative rewriting relations — and by defining concrete reductions throughout relational extensions of syntax endofunctors.

Task 2.2. Quantitative and Resource-Aware Word Problems. The goal of this task is to apply quantitative and resource-aware rewriting to study operational properties of resource-sensitive equational theories. Indeed, quantitative reductions exploit the computational content of resource-aware equational deduction, so that one can use rewriting to algorithmically approximate equationally-defined resources and distances. The problem of approximating resources given by equational theories is precisely the resource-aware refinement of the well-known word problem for equational systems [BKB+03]. Contrary to their resource-agnostic counterparts, resource-aware word problems are completely unexplored, although they naturally appear in several fields, ranging from program semantics to optimal transport [Vi08]. For instance, the well-known Wasserstein-Kantorovich distance between probability distributions [Vi08] can be defined in terms of a suitable quantitative equational theory [MP16]. The celebrated duality result by Kantorovich [Vi08] has an operational interpretation in terms of quantitative rewriting that asserts the existence of a quantitative rewriting strategy computing the Wasserstein-Kantorovich distance. Similar results can be obtained for many other distances, such as the Hamming, Hausdorff, and Levenshtein distance [GD23], this way suggesting a deep connection between quantitative rewriting, resource-aware word problems, and algorithms for computing distances. All of that gives rise to new exciting questions, such as: do quantitative confluence and termination implies the existence of algorithms to compute distances between terms? Under which conditions, if any, there exists a rewriting strategy computing (or approximating) optimal distances? Some preliminary answers have been given by Gavazzo and Di Florio [GD23]; but those touch just the tip of the iceberg. Task 2.2 will be devoted to the study of quantitative and resource-aware word problems, aiming to
answer the aforementioned questions.

**Task 2.3 Quantitative Rewriting for the Resource Calculus**. Quantitative rewriting and its results on resource-aware word problems will be specialised to the quantitative axiomatisation of the resource calculus, developed in T1.2. This will allow, on the one hand, to find an algorithmic way to bound distances among resource calculus terms, on the other to devise some program transformation strategies that will be essential in T3.3 for the analysis of resource calculus programs.

**WP3: Resource Aware Program Analysis: Logics and Abstract Interpretation**

This workpackage will study the notion of resource-aware abstract interpretation and investigate the corresponding program logics. It is known that program logic and abstract interpretation are deeply related concepts. The first specifies the logic of assertions for proving properties of programs, such as program correctness or incorrectness. The latter provides effective algorithms for extracting invariants from programs. The definition of a novel notion of resource-aware abstract interpretation will therefore lead us to reshaping the known program logics, such as classical Hoare’s logic for partial and total correctness of programs and O’Heam’s incorrectness logic [OHe20] for proving the presence of bugs in programs.

The fundamental challenge is to understand the interplay between the precision of an abstract interpreter over a program and the resources that the program may use. The term resources should be intended in its most general form: it might refer to the data structures used by the program, the expressiveness of its basic operators, the environment where the program is supposed to run, the way the program is written, etc. A clear understanding of such interplay would enable to derive, for a given abstract interpreter, suitable constraints on the way programs should be built to reduce (and ideally minimise) the imprecision of its analyses. Moreover, methods and tools could be designed to refactor the code in such a way that the abstract interpreter will expose the least of imprecision, i.e., will report the least number of false-alarms. In general terms, the rationale is to let the code depend on the structure of the analyzer, thus giving to the programmer, who is the ultimate user of a program analysis, the control on its precision. This is similar to what happens for a typed programming language where it is a duty of the programmer to write code which complies with a typing discipline.

While it is known how to refine the analyzer to improve precision [BGGR22], almost nothing is known on how to refactor the code to improve the precision of its static analysis. We are convinced that the foundations of resource-aware syntax, semantics and equational reasoning of WP1 and WP2 can have a significant impact here to achieve this goal.

We know that, for any non-straightforward abstract interpretation, there always exists a program P for which this abstract interpretation yields one or more false alarms [GLR15]. We also know that the semantic equivalence induced by a non-straightforward abstract interpreter violates program extensionality, and that in any Turing complete language we can always foil a non-straightforward abstract interpreter. Here, non-straightforward means abstract interpreters that are not the concrete interpreter and are able to distinguish at least two programs [BGG+20]. What we still don’t know is how the resulting imprecision – which makes the program equivalence induced by an abstract interpreter intensional – depends upon the structure and the way the program is built, in a word, depends upon its resources. Preliminary results [CDG22] have been obtained by embedding metric spaces in abstract domains and abstract interpreters, but no relation has been established between the resources of the program and the structure of the metric space required to control the imprecision of abstract interpreters.

**TASK 3.1. Resource Aware Abstract Interpretation**. We plan to devise techniques for identifying classes of programs for which an abstract interpreter yields a bounded number of false alarms, suitably organised in hierarchies determined by the precision of the analysis (preliminary results reveal a suggestive correspondence with languages in Chomsky's hierarchy). We will study the algebraic properties of these classes and corresponding automata and resource-sensitive rewriting methods for refactoring programs in such a way that a fixed abstract interpreter will be guaranteed to produce false alarms under a given bound. This task will benefit from the results delivered in WP1 and WP2, respectively, for the quantitative reasoning in the design of metrics for measuring the imprecision of abstract interpreters and for code rewriting in equational theories of programs for abstract interpretation driven program refactoring.

**TASK 3.2. Resource Aware Correctness and Incorrectness Logic**. Unveiling a quantitative relation between the resources of the source code and the precision of an abstract interpreter, naturally discloses the interplay between what can be proved in a program logic for correctness/incorrectness properties and the way the program is built. This naturally leads to answering fundamental questions in program verification such as: how far is an approximate invariant from an optimal/suboptimal one in proving program correctness, or, dually, how far is an assertion for proving program incorrectness from the detection of most of the bugs in the program under verification.

**TASK 3.3. Reachability Analysis for resource calculus programs**. The resource calculus represents a perfect case study for the work developed in T3.1. Indeed, such language is not Turing-complete but it has the same expressivity of vector addition systems (also known as Petri nets) for which, decidability and complexity of several decision problems has been broadly studied. In particular, the reachability problem is known to be decidable, but with an extremely high complexity [LS19,CL+20]. In this task, we will face the challenge of improving the efficiency of the analysis by admitting a bounded quantity of false alarms, and we will use the
rewriting strategy developed in T2.3 to refactor the code in such a way to minimise the number of false alarms.

3. Detailed description of the project team and planning; indicating the research team components – PI and associated PIs - and their relative expertise/track record, gender equality of the composition, the interrelation and coherence of the team components. RUs- and the feasibility of the project, thus outlining the congruity between objectives, timing and costs

Structure and Organisation

RAP consists of four cooperating research units (UNIPI, UNIBO, UNIVR, UNIPD) whose work is structured into three work packages (WPs). Each WP is further divided into tasks, and at the end of each task, a deliverable will be released. All deliverables consist of technical articles published, or aimed for publication, in top-class international journals and/or conference proceedings; in addition to that, software artefacts for the quantitative and resource-based analysis of systems will be produced as further deliverables for (specific) tasks. To coordinate research units and their scientific production, each research unit has an associated PI who is responsible for it; additionally, each WP has a WP Leader (WPL) who is responsible for coordinating the technical work in the WP, ensuring that the objectives are achieved and monitoring the production of the deliverables. The associated PI of UNIPI, UNIBO, and UNIVR are the WPLs for WP1, WP2, and WP3, respectively. The PI of the project is furthermore responsible for the overall project organisation as well as for coordinating research units and promoting their collaboration.

Workplan

All RAP research units will work both synchronously and asynchronously focusing on each WP both in parallel, hence avoiding bottleneck and deadlock, and through several collaborations connecting WPs one another. More specifically, foundational developments of each WP – namely resource-aware equational theories, rewriting, and abstract interpretation and program logic – are causally independent and dedicated research units will asynchronously work on them: UNIBO and UNIPI will be the most active on both WP1 and WP2, with UNIBO leading WP1 and UNIPI leading WP2; UNIVR will lead WP3 closely collaborating with UNIPD and UNIPI, as well as with UNIBO on specific tasks. Applications and case studies, instead, will be pursued synchronously: after WP1 will complete its first case study, both WP2 and WP3 will apply the foundational results achieved that far, as well as results specifically crafted, on it. The GANTT below shows the RAP timeline with respect to WPs and their tasks, and highlights connections between them.
To ensure a uniform and continuous progress of the project, a Project Board (PB) will be employed. The PB will be composed of the associated PIs and headed by the project PI. The PB will meet virtually every three months and physically every six. In each meeting, progresses of the project will be discussed and the correct and expected exchanges, both among WPS and among sites, will be monitored. In addition to these meetings, a kick-off meeting consisting of tutorial talks and discussions will be held at the beginning of the project, and a middle-term and final meeting will be held at the middle and end of the project, respectively. To guarantee visibility of the project, RAP kick-off and final meetings will be held as part of larger events, such as established conferences on similar topics or events organized by members of the consortium. All members, in fact, have extensively served in their scientific community, chairing and organizing major conferences in their research fields.