Universitat Politècnica de València

DSIC
Departamento de Sistemas Informáticos y Computación

Doctorado en Informática
RD 99/2011

Rewriting Logic Techniques for Program Analysis and Optimization

Ph.D. Student
Julia Sapiña Sanchis
jsapina@dsic.upv.es

Supervisor
María Alpuente Frasnedo
alpuente@dsic.upv.es

Motivation

- According to recent Cambridge University research, the global cost of debugging software has risen to $312.000 millions by 2013.
- On average, software developers spend 50% of their programming time finding and fixing bugs.
- Execution traces are an important source of information for program understanding and debugging.
- However, software systems commonly generate large and complex execution traces whose analysis is extremely time-consuming and even unfeasible to perform without adequate tool support.
- Trace slicing is an automated transformation technique that can drastically reduce the size and complexity of execution traces by tracking dependences and causality along the traces and by removing irrelevant information that does not affect or is affected by the observed data.
- By greatly reducing the size of execution traces while keeping all the relevant information, the effort required to find and correct an error can be significantly lowered because many irrelevant inspections that occur during diagnosis and bug localization can be automatically avoided.

General Objective

- New tools and techniques for program debugging and optimization based on automated transformation of programs and computations.

Specific Objectives

- Further develop transformation techniques for the inspection and analysis of rewriting logic (RWL) computations, with particular emphasis on efficiency.
- Apply these techniques to real languages whose RWL semantics is formalized in the 𝕂-Framework.
- Develop the first rewriting-based, universal debugger that can inspect any program by just loading the correspondent language semantics.

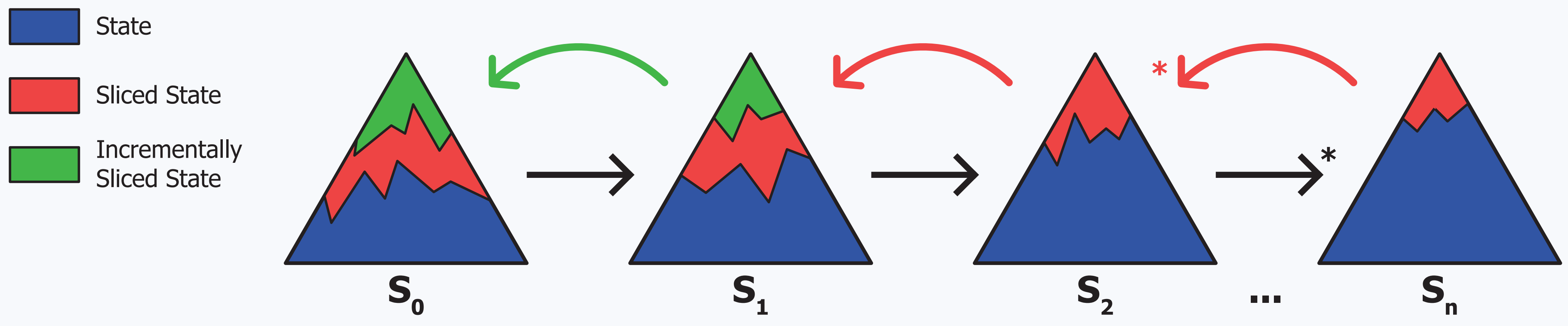
Research Plan

- **Stage 1 (in progress)**. Research on dependence analysis and slicing techniques for rewriting logic computations.
- **Stage 2.** Acquaitance with the 𝕂-Framework.
- **Stage 3.** Formal development in 𝕂 of a universal debugger.
- **Stage 4.** Implementation of the universal debugger and experimental evaluation.
- **Stage 5.** Applications of universal debugging and transformation.

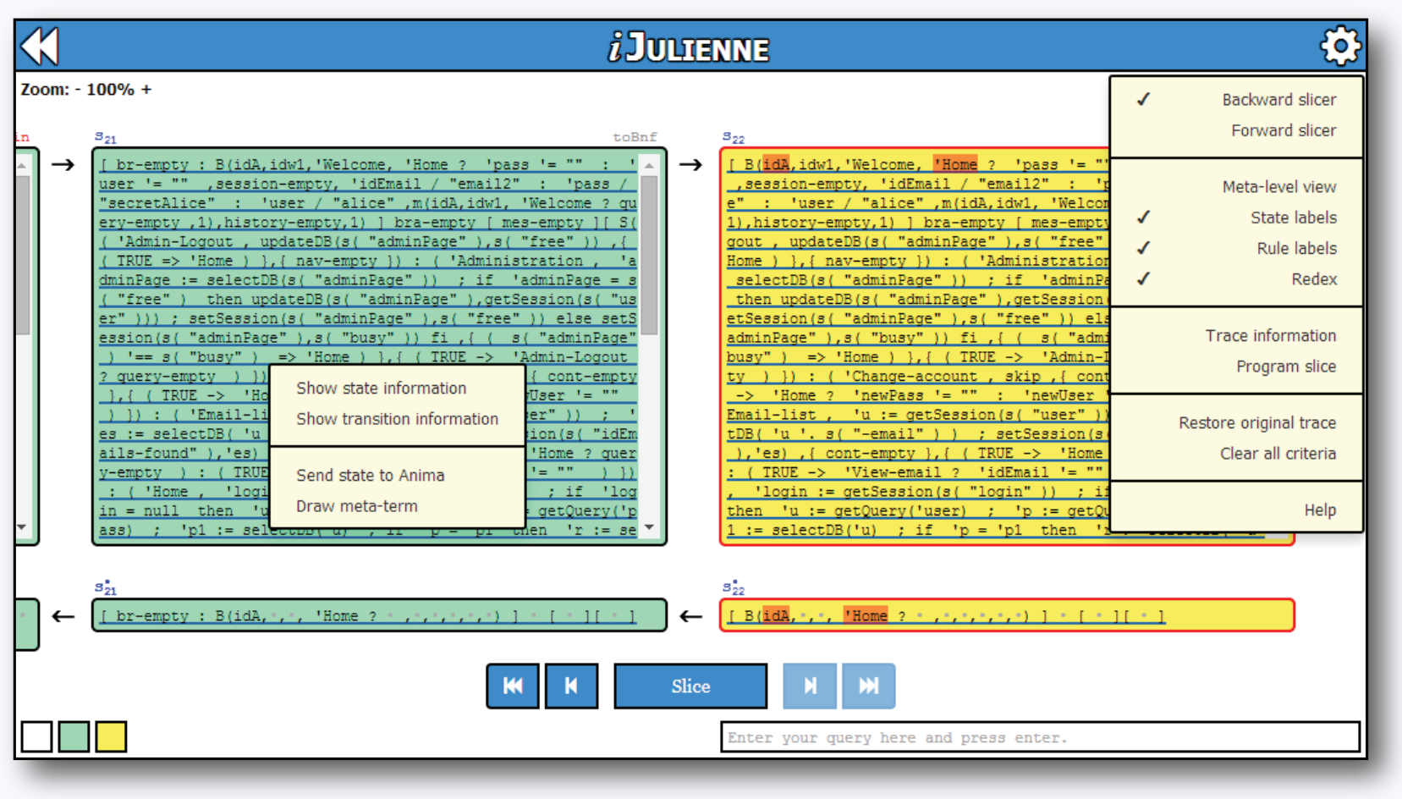
Expected Results

- Contribute to advance the state of the art in trace analysis of rewriting logic computations (journal articles and conferences).
- Significantly improve the time required to debug a program by providing the user with efficient techniques and tools to perform the necessary inspection.
- Provide existing and future programming languages with a generic, extensible, modular, and language-independent analysis and debugging framework.

Backward Slicing[1]

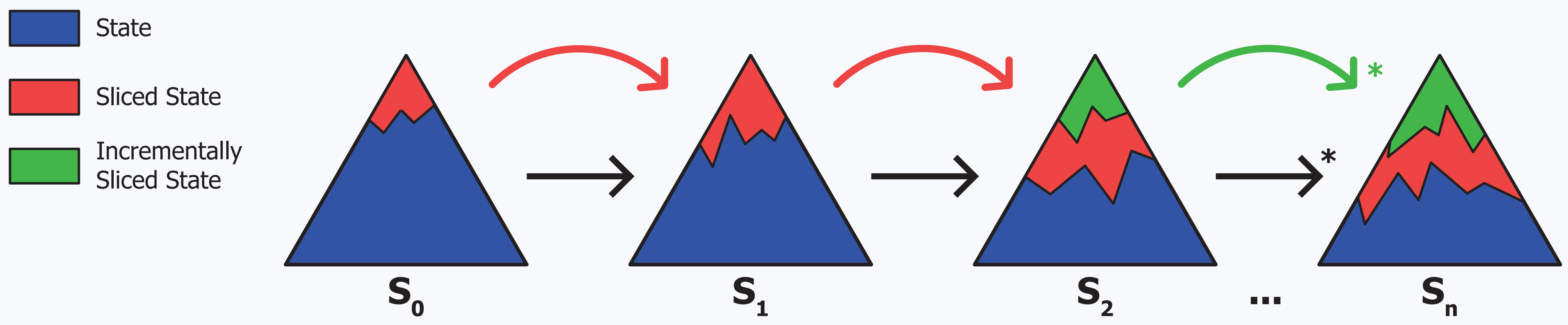


- First backward trace slicing technique for Rewriting Logic specifications.
- We trace back the antecedents of the observed data.
- The slicing criterion can be refined or completely redefined at any state, resulting in significantly smaller trace slices (incremental slicing).
- Implemented using Maude in the ***iJulienne*** Online Trace Analyzer, which is publicly available at safe-tools.dsic.upv.es/iJulienne.

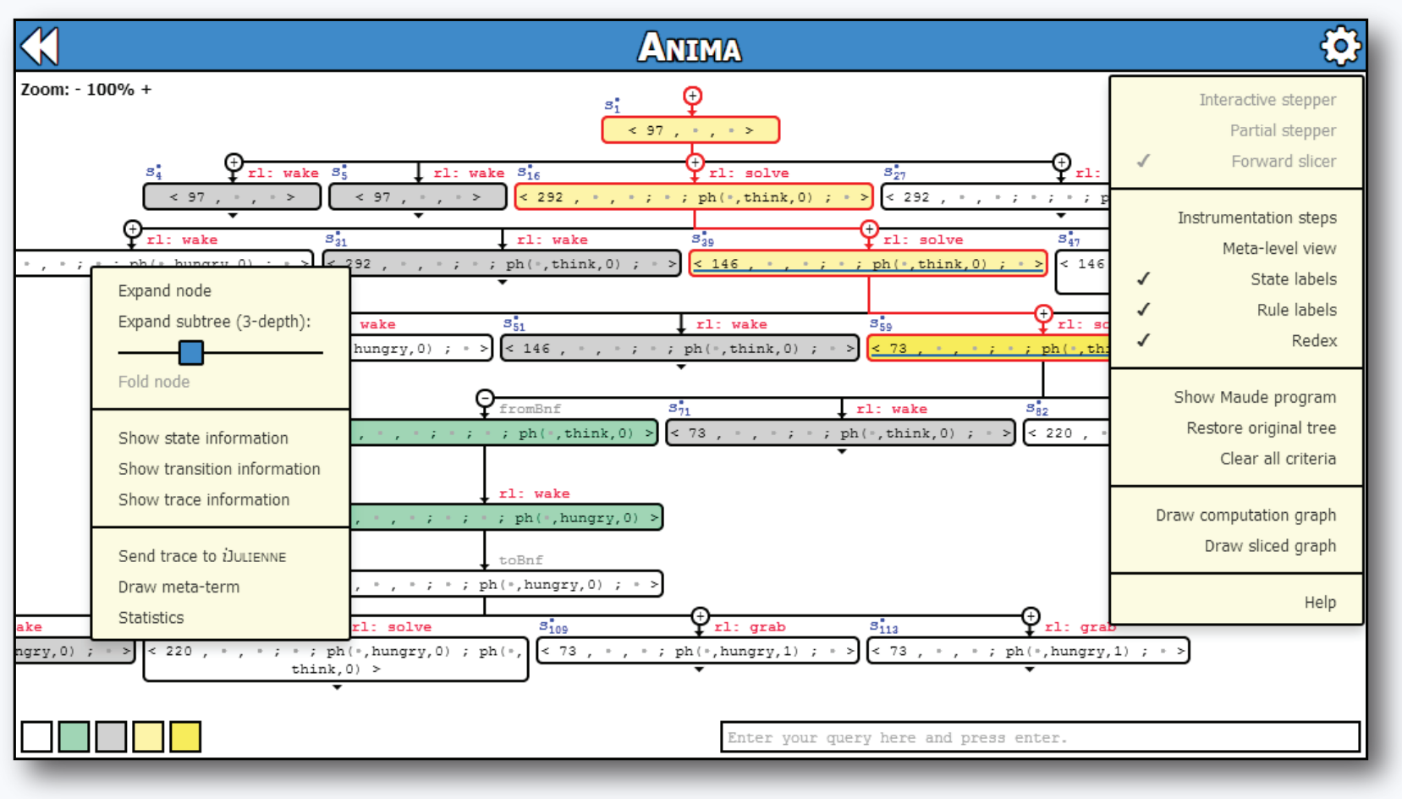


Navigation through the trace slice of a webmail application example in *i*Julienne.

Forward Slicing[1]

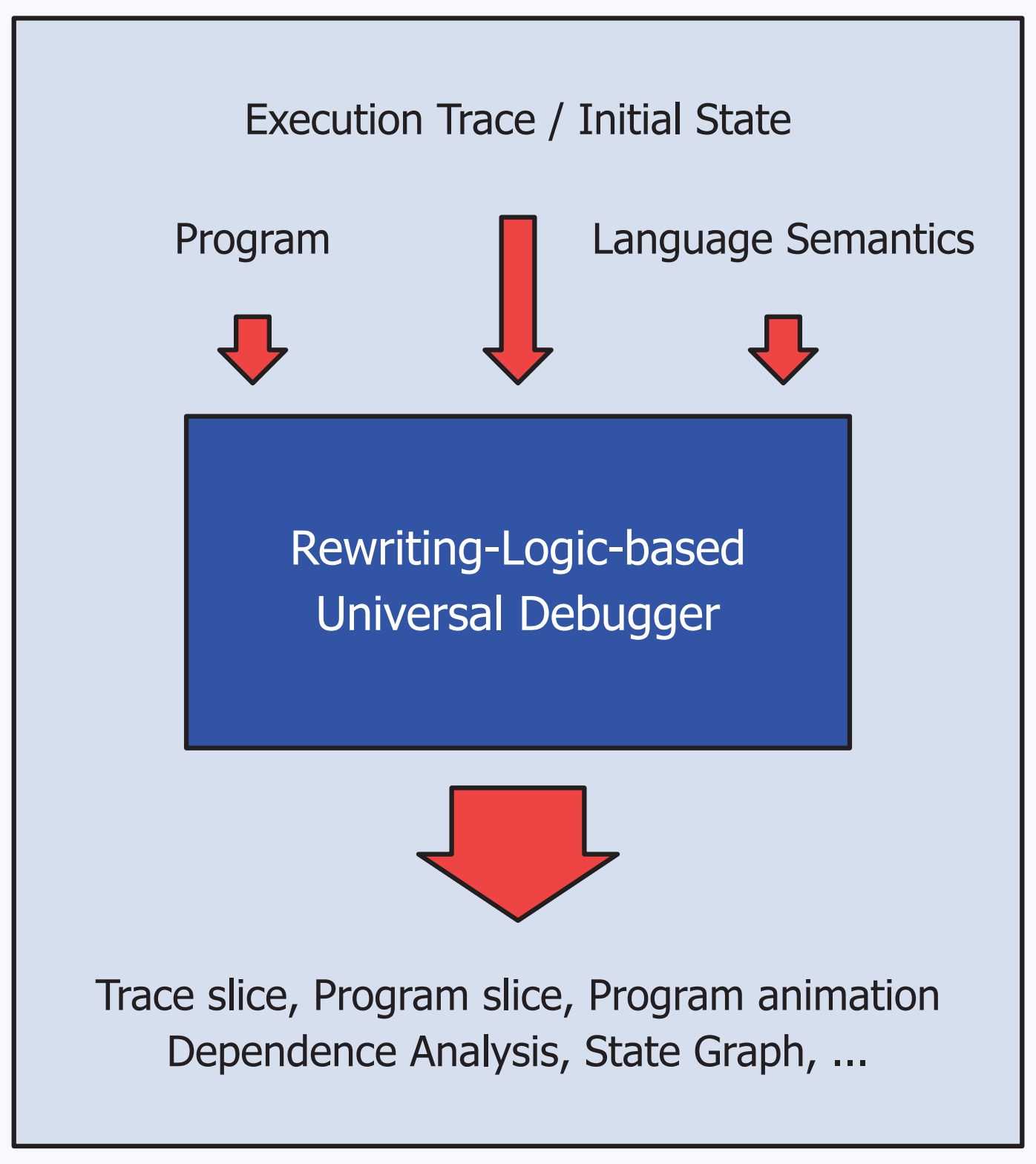


- First forward trace slicing technique for Rewriting Logic specifications.
- Descendants of the observed data are traced forth.
- The slicing criterion can be refined or completely redefined at any state, resulting in significantly smaller trace slices (incremental slicing).
- Implemented using Maude in the **Anima** Online Stepper, which is publicly available at safe-tools.dsic.upv.es/anima.



Forward slicing exploration of the Philosophers problem with Anima.

Universal Debugging


Execution Trace / Initial State
Program
Language Semantics
Rewriting-Logic-based Universal Debugger
Trace slice, Program slice, Program animation
Dependence Analysis, State Graph, ...

Universal Debugger Architecture.

- The 𝕂-Framework is an operational, Rewriting Logic semantic framework in which programming languages can be defined using configurations, computations and rules.
- Examples of languages that have been semantically defined in the 𝕂-Framework are Beta, C, Esolang, Haskell, IMP, Java 1.4, Javascript, KOOL, Lambda, LLVM, OCAML, PHP, Python, Scheme, and Verilog.
- Once the semantics of a language is defined, it is possible to execute any piece of code by means of the generated rewriting system.
- Therefore, we can apply our techniques to develop the first universal rewriting-based debugger, which allows the behavior of any program to be analyzed.

Analyze a program written in **any language**
by just loading its semantics!

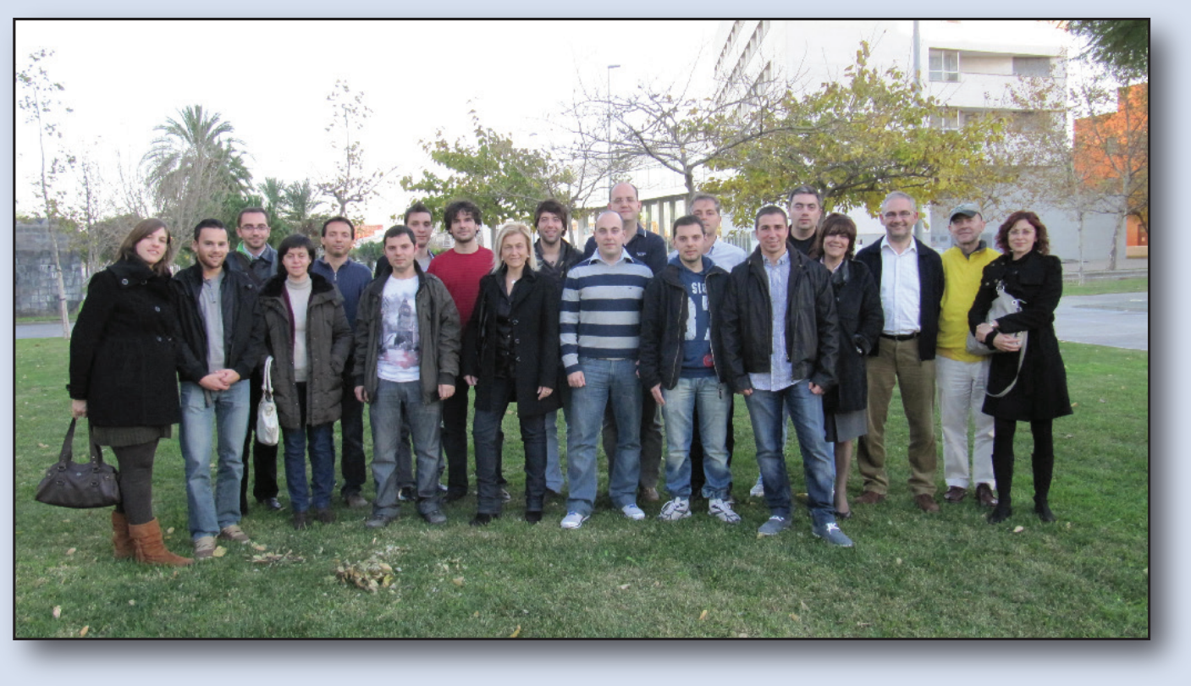
[1] Joint work with María Alpuente, Demis Ballis, and Francisco Frechina.

Publications

- M. Alpuente, D. Ballis, F. Frechina, and J. Sapiña. Slicing-Based Trace Analysis of Rewriting Logic Specifications with iJulienne. In Proc. of the 22nd European Symposium on Programming (ESOP 2013), volume 7792 of Lecture Notes in Computer Science (LNCS), pages 121-124. Springer-Verlag, 2013.
- M. Alpuente, D. Ballis, F. Frechina, and J. Sapiña. Parametric Exploration of Rewriting Logic Computations. In Proc. of the 5th International Symposium on Symbolic Computation in Software Science (SCSS 2013), volume 15 of EasyChair Proceedings in Computing (EPiC), pages 4-18. EasyChair, 2013.
- M. Alpuente, D. Ballis, F. Frechina, and J. Sapiña. Inspecting Rewriting Logic Computations (in a parametric and stepwise way). In Proc. of Specification, Algebra, and Software: A Festschrift Symposium in Honor of Kokichi Futatsugi (SAS 2014), volume 8373 of Lecture Notes in Computer Science (LNCS), pages 229-255. Springer-Verlag, 2014.

Acknowledgements

- Francisco Frechina.
- Demis Ballis.
- Members of the Extensions of Logic Programming (ELP) research group.
- UPV-FPI 2013 research grant.



Extensions of Logic Programming (ELP) research group.