Resource Usage Analysis and its Application to Resource Certification (Part I)

Germán Puebla[1]

joint work with

Elvira Albert[2], Puri Arenas[2],
Samir Genaim[2], and Damiano Zanardini[1]

[1] Technical University of Madrid, Spain

[2] Complutense University of Madrid, Spain

September 3–4 2009

Resource Usage and Security

- The aim of security analysis of a program is to guarantee that the execution of the program does not perform certain undesired behaviours.

Resource Usage and Security

- The aim of security analysis of a program is to guarantee that the execution of the program does not perform certain undesired behaviours.
- Classical notions of undesired behaviour include:
  - Access to private data
  - Tampering with data not related to program (i.e. removing files)

Resource Usage and Security

- The aim of security analysis of a program is to guarantee that the execution of the program does not perform certain undesired behaviours.
- Classical notions of undesired behaviour include:
  - Access to private data
  - Tampering with data not related to program (i.e. removing files)
- Also, using too many resources can be undesired as well. E.g.:
  - Consuming too much CPU
  - Taking up too much memory
  - Performing too many billable events

Resource Usage and Security

- The aim of security analysis of a program is to guarantee that the execution of the program does not perform certain undesired behaviours.
- Classical notions of undesired behaviour include:
  - Access to private data
  - Tampering with data not related to program (i.e. removing files)
- Also, using too many resources can be undesired as well. E.g.:
  - Consuming too much CPU
  - Taking up too much memory
  - Performing too many billable events
- From this point of view, resource usage is a security property.

Security of Mobile Code

- Today, the use of mobile code is widespread. E.g., running applets downloaded from the web.
- In fact, most software we install and run is actually downloaded from the web.

Security of Mobile Code

- Today, the use of mobile code is widespread. E.g., running applets downloaded from the web.
- In fact, most software we install and run is actually downloaded from the web.
- The current approach to security of mobile code (e.g. Java bytecode) is a combination of:
  - static checking of certain properties, such as the bytecode verification process
  - dynamic checking of operations which are still potentially unsecure, such as array indexing.

Security of Mobile Code

- Today, the use of mobile code is widespread. E.g., running applets downloaded from the web.
- In fact, most software we install and run is actually downloaded from the web.
- The current approach to security of mobile code (e.g. Java bytecode) is a combination of:
  - static checking of certain properties, such as the bytecode verification process
  - dynamic checking of operations which are still potentially unsecure, such as array indexing.
- We would like to extend the security policies which are statically verified in order to include resource usage.

Static Checking of Resource Usage

- It the static checker succeeds to prove that the program complies with the resource usage policy, the program is loaded and executed.
- Otherwise, the program is rejected as being potentially dangerous.

Static Checking of Resource Usage

- It the static checker succeeds to prove that the program complies with the resource usage policy, the program is loaded and executed.
- Otherwise, the program is rejected as being potentially dangerous.
- Several approaches can be used to perform static checking of resource usage:
  - Type systems: elegant and efficient. Generally restricted to certain complexity classes.

Static Checking of Resource Usage

- It the static checker succeeds to prove that the program complies with the resource usage policy, the program is loaded and executed.
- Otherwise, the program is rejected as being potentially dangerous.
- Several approaches can be used to perform static checking of resource usage:
  - Type systems: elegant and efficient. Generally restricted to certain complexity classes.
  - Program logics extended to include resource usage. Precise, but often requires user intervention.

Static Checking of Resource Usage

- It the static checker succeeds to prove that the program complies with the resource usage policy, the program is loaded and executed.
- Otherwise, the program is rejected as being potentially dangerous.
- Several approaches can be used to perform static checking of resource usage:
  - Type systems: elegant and efficient. Generally restricted to certain complexity classes.
  - Program logics extended to include resource usage. Precise, but often requires user intervention.
  - Recurrence relations. Not restricted, but approximate. Quite promising results.

Our Approach to Static Checking of Resource Usage

- In this course we will study an automatic approach to static checking of resource usage policies based on:
  - Obtaining an upper bound on the resource usage of programs
  - Comparing this upper bound with a user-provided resource usage policy.

Our Approach to Static Checking of Resource Usage

- In this course we will study an automatic approach to static checking of resource usage policies based on:
  - Obtaining an upper bound on the resource usage of programs
  - Comparing this upper bound with a user-provided resource usage policy.
- This approach has important advantages:
  - The upper bounds computed can be used for many applications besides resource certification.

Our Approach to Static Checking of Resource Usage

- In this course we will study an automatic approach to static checking of resource usage policies based on:
  - Obtaining an upper bound on the resource usage of programs
  - Comparing this upper bound with a user-provided resource usage policy.
- This approach has important advantages:
  - The upper bounds computed can be used for many applications besides resource certification.
  - It is based on a fully automatic static analysis.

Our Approach to Static Checking of Resource Usage

- In this course we will study an automatic approach to static checking of resource usage policies based on:
  - Obtaining an upper bound on the resource usage of programs
  - Comparing this upper bound with a user-provided resource usage policy.
- This approach has important advantages:
  - The upper bounds computed can be used for many applications besides resource certification.
  - It is based on a fully automatic static analysis.
  - The user does not need to provide upper bounds for intermediate program parts whose resource usage is not to be checked.

First COSTA System Demo

Costa Demo

On checking of upper bounds.

Kinds of Resource Usage Analysis

- In programs with data-structures, not all input with the same size have the same cost. Several alternatives are possible
  - worst case $\longrightarrow$ upper bound
  - average case $\longrightarrow$ requires probabilistic study
  - best case $\longrightarrow$ lower bound

Kinds of Resource Usage Analysis

- In programs with data-structures, not all input with the same size have the same cost. Several alternatives are possible
  - worst case $\longrightarrow$ upper bound
  - average case $\longrightarrow$ requires probabilistic study
  - best case $\longrightarrow$ lower bound
- Two classes of upper bounds can be considered:
  - non-asymptotic (or concrete, or micro-analysis)
  - asymptotic (or macro-analysis)

Kinds of Resource Usage Analysis

- In programs with data-structures, not all input with the same size have the same cost. Several alternatives are possible
  - worst case $\longrightarrow$ upper bound
  - average case $\longrightarrow$ requires probabilistic study
  - best case $\longrightarrow$ lower bound
- Two classes of upper bounds can be considered:
  - non-asymptotic (or concrete, or micro-analysis)
  - asymptotic (or macro-analysis)
- Analysis results can be
  - platform-independent
  - platform-dependent $\longrightarrow$ WCET

State of the Art in Automatic Cost Analysis

- Work on automatic cost analysis dates back to 1975, with the seminal work of Wegbreit.
- His system, metric was able to compute:
  - interesting results, but for
  - restricted class of functional programs

State of the Art in Automatic Cost Analysis

- Work on automatic cost analysis dates back to 1975, with the seminal work of Wegbreit.
- His system, metric was able to compute:
  - interesting results, but for
  - restricted class of functional programs
- Also, the seminal work on abstract interpretation [Cousot & Cousot'77] mentions performance analysis as an application.

State of the Art in Automatic Cost Analysis

- Work on automatic cost analysis dates back to 1975, with the seminal work of Wegbreit.
- His system, metric was able to compute:
  - interesting results, but for
  - restricted class of functional programs
- Also, the seminal work on abstract interpretation [Cousot & Cousot'77] mentions performance analysis as an application.
- Since then, a number of analyses and systems have been built which extend the capabilities of cost analysis:
  - functional programs [Le Metayer'88, Rosendahl'89, Wadler'88, Sands'95, Benzinger'04]
  - logic programs [Debray and Lin'93, Navas et al'07]
  - imperative [Adachi et al'79, Gulwani el al'09]
  - object oriented [Albert et al'07]

State of the Art in Automatic Cost Analysis

- Work on automatic cost analysis dates back to 1975, with the seminal work of Wegbreit.
- His system, metric was able to compute:
  - interesting results, but for
  - restricted class of functional programs
- Also, the seminal work on abstract interpretation [Cousot & Cousot'77] mentions performance analysis as an application.
- Since then, a number of analyses and systems have been built which extend the capabilities of cost analysis:
  - functional programs [Le Metayer'88, Rosendahl'89, Wadler'88, Sands'95, Benzinger'04]
  - logic programs [Debray and Lin'93, Navas et al'07]
  - imperative [Adachi et al'79, Gulwani el al'09]
  - object oriented [Albert et al'07]
- Other approaches exist not based on this approach:
  - type systems [Hofmann and Jost'03]
  - quantitative abstract interpretation [Sotin et al'06]

A Classical Approach to Cost Analysis

A classical approach [Wegbreit'75] to cost analysis consists of:

1. expressing the cost of a program part in terms of other program parts, thus obtaining *recurrence relations*
2. solving the relations by obtaining a *closed-form* for the cost in terms of the input arguments

A Classical Approach to Cost Analysis

A classical approach [Wegbreit'75] to cost analysis consists of:

1. expressing the cost of a program part in terms of other program parts, thus obtaining *recurrence relations*
2. solving the relations by obtaining a *closed-form* for the cost in terms of the input arguments

The current situation is that

- Most work has concentrated on the first phase, usually for simple programming languages
- The difficulties of the second phase have been mostly overseen
- Practical usage of cost analysis requires both!
- In COSTA we address both phases.

A Simple Example

- For the method:

```
public int funLinear (int entry) {
    if (entry <= 0)
        return 1;
    else
        return entry * funLinear (entry - 1);
}
```

- The corresponding Recurrence Relation for number of bytecode instructions is:

```
funLinear(I)I(A,B)= 4                                [B=<0]
funLinear(I)I(A,B)= 10+funLinear(I)I(A,C) [B>=1,C=B-1,A>=1]
```

- Whose (exact) Closed Form is:

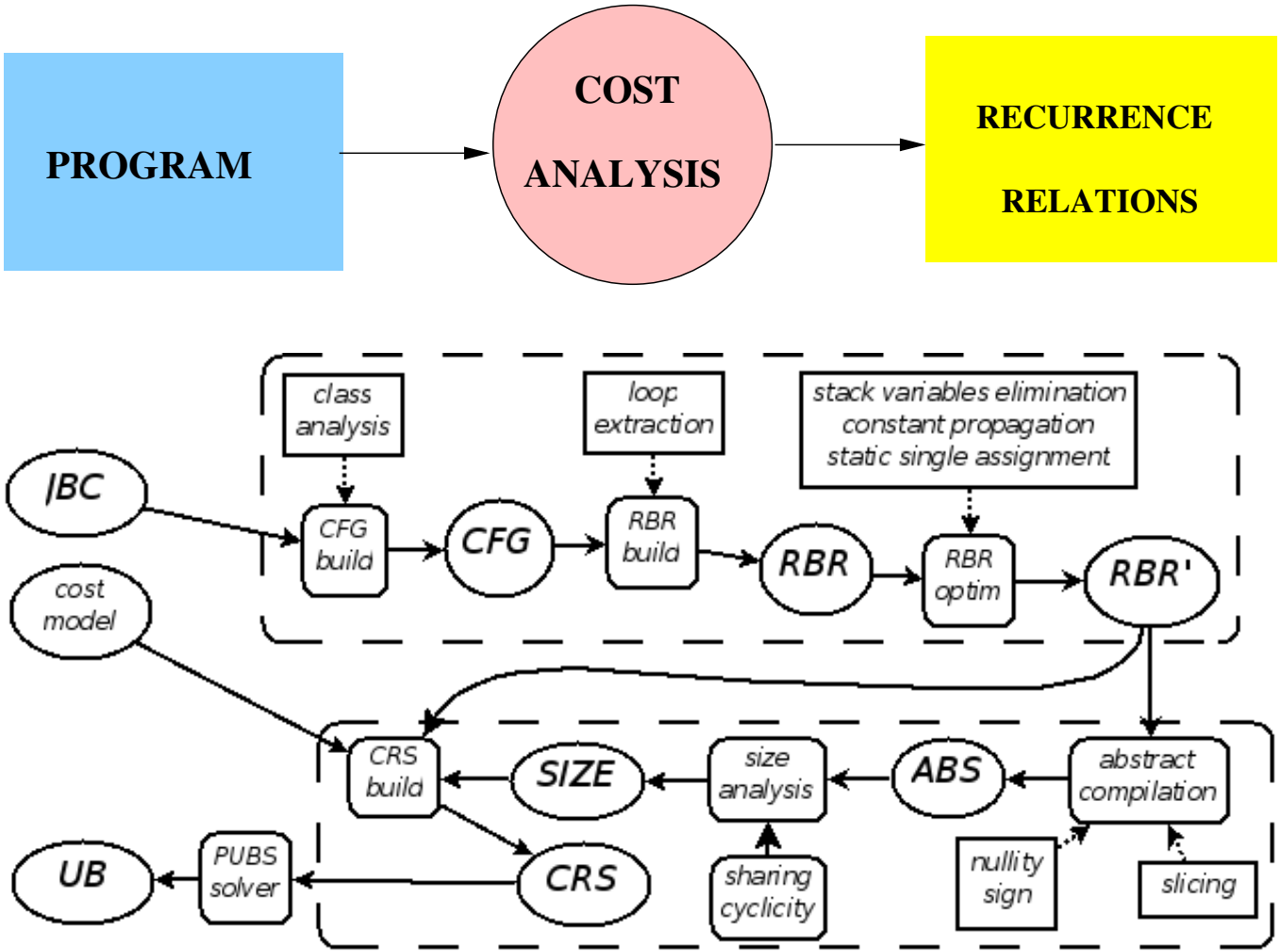
```
funLinear(I)I(A,B)=10*nat(B)+4
```

A Simple Example (Cont.)

- However, the actual recurrence relation automatically generated from the bytecode is:

```
funLinear(I)I(A,B)= 2 + f8(A,B,B)                          []
f8(A,B,C)= f2(A,B)                                         [C>=1]
f8(A,B,C)= f3(A,B)                                         [C=<0]
f2(A,B) = 5 + f5(A,B,B,A,C)                                [C=B-1,A>=1]
f3(A,B)=  2 + f7(A,B,1)                                    []
f5(A,B,C,D,E)=1+funLinear(I)I(D,E)+normal(A,B,C,F,G)  []
f7(A,B,C)= 0                                               []
normal(A,B,C,D,E) = f6(A,B,C,D)                            []
f6(A,B,C,D)= 2 + f7(A,B,E)                                 []
```

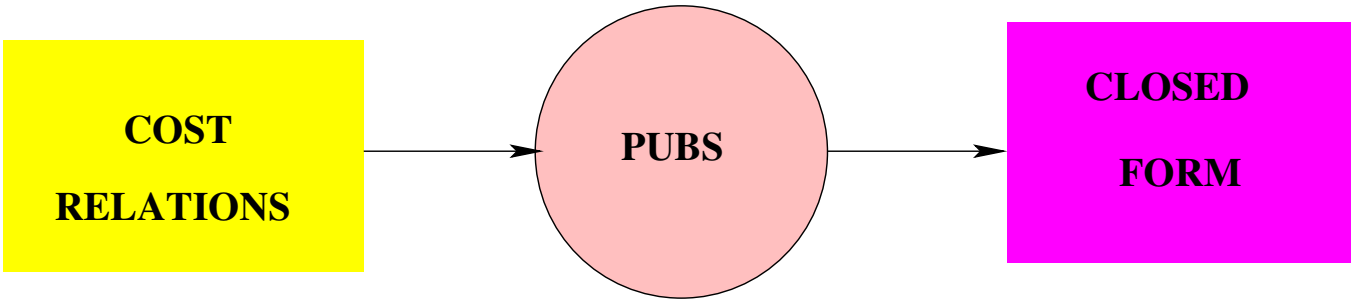
Step 1: Generating Recurrence Relations



Underlying techniques (static analyses in the first phase)

- class analysis reduces the possible classes the object o might be an instance of when executing the expression $o.m()$
- sign analysis and null pointer analysis help in removing some branches (e.g., exceptional branches due to null references) from the program control
- slicing removes from the internal representation variables which are not relevant to the cost/termination behavior, thus simplifying the following tasks
- sharing and cyclicity analyses are required for correctness of subsequent analyses
- field-sensitive value analysis to handle numeric fields by transforming them to local variables when it is safe
- size analysis infers the relations between the sizes of the program variables at different program points
- more auxiliary analyses: *loop extraction*, *static single assignment*, *constant propagation*

Step 2: Finding Closed Form Upper Bounds



PUBS: **P**ractical **U**pper **B**ound **S**olver

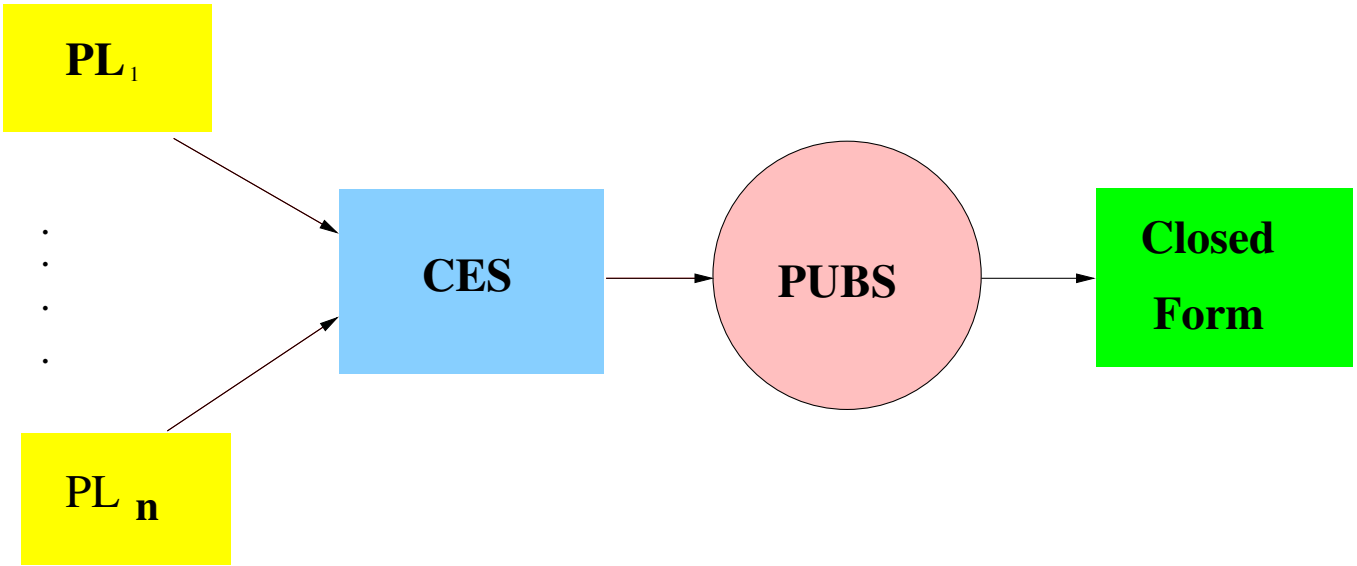
The main features of our approach are:

- Regard cost relations as (nondeterministic) programs.
- Provide an operational semantics based on evaluation trees.
- Reason on the shape and contents of evaluation trees.
- Perform partial evaluation for obtaining direct recursion.
- Compute ranking functions for bounding the height of trees.
- Obtain loop invariants for maximizing expressions.

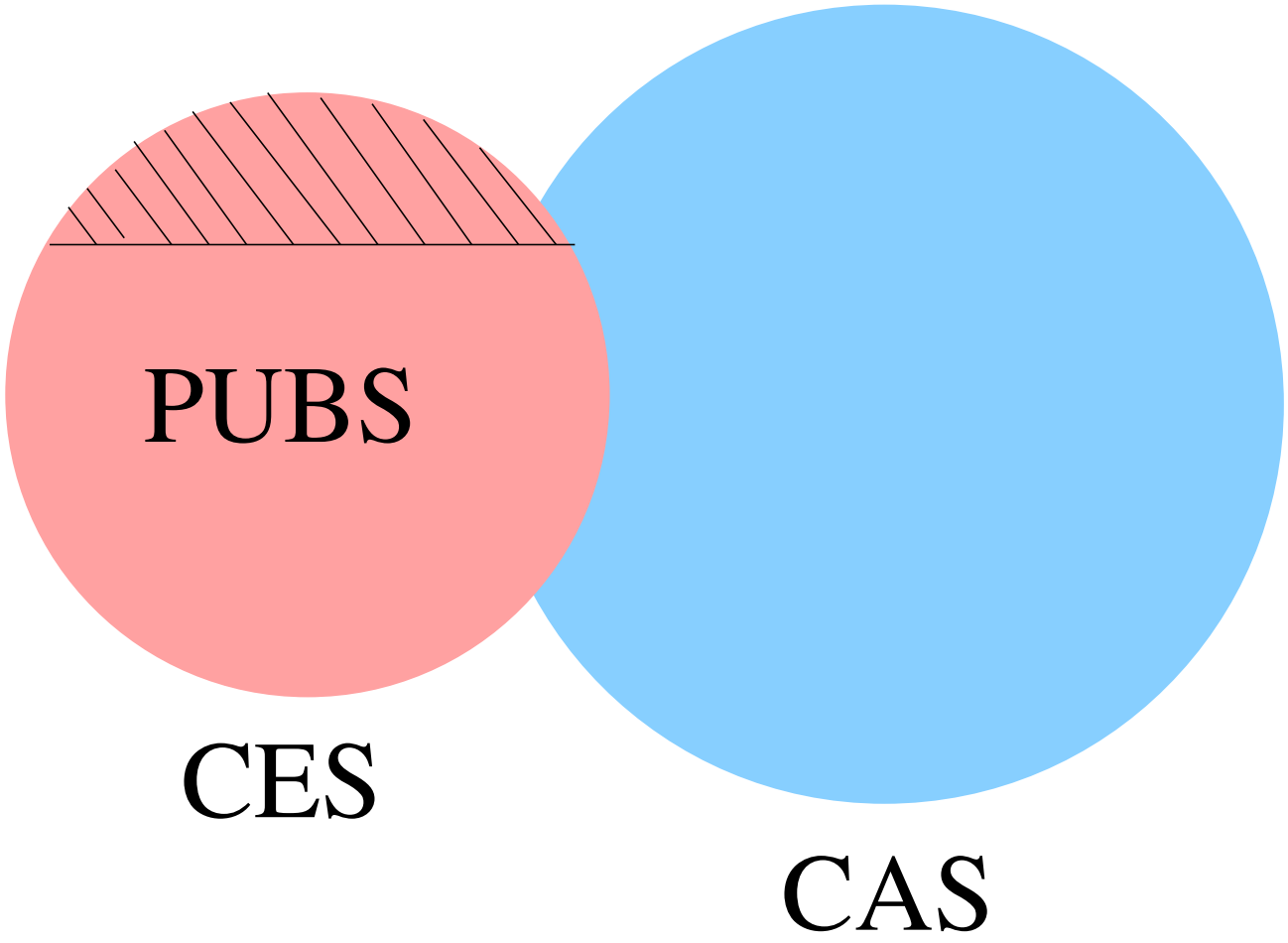
Underlying techniques (Upper Bounds Solver)

- Partial Evaluation: automatic transformation used to put the Cost Equation Systems in directly recursive form. This allows solving the relations compositionally.
  - In PE terms, we use monovariant offline partial evaluation
- Ranking Functions: used to obtain an upper bound on the number of iterations of each recursive relation.
  - In termination, RF are widely used, but its use in cost analysis is new.
- Loop Invariants: allow us to know (or have a safe approximation of) the values of the variables which appear in cost expressions
  - This makes it possible to maximize expressions in order to obtain cost upper bounds

PUBS as a Back-end to Different Cost Analyses



PUBS vs Computer Algebra Systems



COSTA: Main Publications

- *Cost Analysis of Java Bytecode.* – ESOP 2007
  - Generation of cost relations for Java Bytecode.
- *Automatic Inference of Upper Bounds for Recurrence Relations in Cost Analysis.* – SAS 2008.
  - Generation of closed form upper bounds for cost relations (which may originate from any programming language).
- *Termination Analysis of Java Bytecode.* – FMOODS'08.
  - Termination of bytecode by using CLP techniques.
- *Live Heap Space Analysis for Languages with Garbage Collection.* – ISMM'07, ISMM'09
  - Inference of heap escape usage considering a garbage collector
- *Field-Sensitive Value Analysis by Field Insensitive Analysis.* – FM'09
  - Analysis to infer size relations in the presence of numeric fields
- *Asymptotic Resource Usage Bounds.* – APLAS'09
  - Obtaining asymptotic bounds (upper, lower, and average case)

Systems and Contact

- Two systems, with web interfaces.
- COSTA: **COS**T AND **T**ERMINATION **A**NALYZER
  - can be tried out at `http://costa.ls.fi.upm.es/costa`

Systems and Contact

- Two systems, with web interfaces.
- COSTA: **COS**T AND **T**ERMINATION **A**NALYZER
  - can be tried out at `http://costa.ls.fi.upm.es/costa`
- PUBS: **P**RACTICAL **U**PPER **B**OUNDS **S**OLVER
  - The recurrence relation solver, language independent can be tried out at `http://clip.dia.fi.upm.es/Systems/PUBS`

Systems and Contact

- Two systems, with web interfaces.
- COSTA: **COS**T AND **T**ERMINATION **A**NALYZER
  - can be tried out at `http://costa.ls.fi.upm.es/costa`
- PUBS: **P**RACTICAL **U**PPER **B**OUNDS **S**OLVER
  - The recurrence relation solver, language independent can be tried out at `http://clip.dia.fi.upm.es/Systems/PUBS`
- The COSTA web site, based on `trac`, can be found at
  `http://costa.ls.fi.upm.es`
  (or google "The COSTA System")

Systems and Contact

- Two systems, with web interfaces.
- COSTA: **COS**T AND **T**ERMINATION **A**NALYZER
  - can be tried out at `http://costa.ls.fi.upm.es/costa`
- PUBS: **P**RACTICAL **U**PPER **B**OUNDS **S**OLVER
  - The recurrence relation solver, language independent can be tried out at `http://clip.dia.fi.upm.es/Systems/PUBS`
- The COSTA web site, based on `trac`, can be found at
  `http://costa.ls.fi.upm.es`
  (or google "The COSTA System")
- Both COSTA and PUBS will shortly be released under the General Public License.
- Further information on the systems can be requested at
  `costa @ fi.upm.es`
- For information about the upcoming release and other issues, you may consider joining the list
  `costa-users @ listas.fi.upm.es`