25 Years of Ciao

Manuel Hermenegildo[1,2] M. Carro[1,2] P. López-García[1,3] J. Morales[1]

J. Arias[1] I. García-Contreras[1,2] M. Klemen[1] N. Stulova[1]

[1] IMDEA Software Institute

[2] T. U. of Madrid (UPM)

[3] Spanish Research Council (CSIC)

LOPSTR'18 — Frankfurt, Sep 6, 2018

25 (More) Years of Ciao

Manuel Hermenegildo[1,2] M. Carro[1,2] P. López-García[1,3] J. Morales[1]

J. Arias[1] I. García-Contreras[1,2] M. Klemen[1] N. Stulova[1]

[1] IMDEA Software Institute

[2] T. U. of Madrid (UPM)

[3] Spanish Research Council (CSIC)

LOPSTR'18 — Frankfurt, Sep 6, 2018

Going back to the beginning...

- Context, early 90's: **many languages/systems, each with one extension**.
  - Parallelism/concurrency: &-Prolog, MUSE, Andorra, GHC, CC, ...
  - Equations, functions, CLP(X), HO unification, ...
  - Control rules: Andorra, iterative deepening, tabling, ...

But systems typically only good at that one thing.

- Key observation: no need for a whole separate system for each extension.
  - E.g., if you have stack sets, microtasks, goal queues, etc. you can implement any concurrency/parallelism model.
  - E.g., if you have attributed variables: CLP(Q)/CLP(R) (Holzbaur), delays, ...

Ciao principles I: **Language definition/extension is library-based.**

- Start: small, very *extensible* LP kernel – a language-building language.
- Build gradually extensions in layers on top of it.
- Bring in the *most useful features* from the different programming paradigms.

- Much development within the ACCLAIM project (same as, e.g., Oz).
- This approach is also taken nowadays by, e.g., Racket.

[TPLP'12, PADL'13, LOPSTR'11, DCG-PhD'04, NovaSci'99, ILPS'95, PPCP'94, CL'00, ENTCS'00]
[PADL'12, ICLP'11, ICLP'08, EuroPar'96, NGC'91, NGC'89, ICPP'88, ICLP'87, ICLP'86]

- Context, early 90's:
  A tendency to restrict languages (generally for performance).
  - Elimination of unification: Mercury, GHC, CC, Erlang, ...
  - Elimination of non-determinism/search: GHC, CC, Erlang, ...

  Static languages, strong typing:
  - ML, Haskell | Gödel, Mercury.

- At the same time:
  Abstract interpretation-based global analysis becoming practical.
  - Motivated by auto-parallelization and other optimizations (*dynamic language*).
  - Efficient fixpoint (PLAI), with techniques for dealing with dynamic languages, interactive development (e.g., incremental analysis).
  - Powerful domains (sharing/aliasing, cost, convex hulls, polyhedra, CLP).
  - Global analysis supported by module system.
  - Already using assertions (`entry`, `trust`, `true`) to communicate w/analyzer.

Ciao principles II:
High performance via optimization, not language restriction.

- No need to eliminate unification or tabling or backtracking or constraints, etc.

Ciao principles III:
Combine the best of the dynamic and static language approaches.

- Provide the flexibility of dynamic languages:
  - Dynamic typing, dynamic load, dynamic program modification, meta-programming, top level, call (eval), scripts, ...
- But with *guaranteed safety and efficiency.*

Enablers:

- **Optimization** via analysis, partial evaluation, parallelization, profiling, separate/incr. compilation, small executables, embeddability, ...
- **AbsInt-based checking of optional assertions** – safe approximations – The Ciao assertions model

[ICLP'09, LPAR'06, SCP'05, SAS'03, PBH00a, LOPSTR'99, LNCS'99, AADEBUG'97]

Ciao principles II: **High performance via optimization, not language restriction.**

- No need to eliminate unification or tabling or backtracking or constraints, etc.

Ciao principles III: **Combine the best of the dynamic and static language approaches.**

- Provide the flexibility of dynamic languages:
  - Dynamic typing, dynamic load, dynamic program modification, meta-programming, top level, call (eval), scripts, ...
- But with *guaranteed safety and efficiency.*

- Approach not particularly in line with the trends at the time!
  - Strong typing was a religion, dynamic languages a **bad** thing.
  - Reflected in the languages developed: ML, Haskell | Goedel, Mercury.

→ "Ciao: (first?) dynamic language with safety assurances,
trying to survive in a world dominated by strong typing."

- However, idea quite popular now: gradual typing, Racket, liquid Haskell, etc.

[ICLP'09, LPAR'06, SCP'05, SAS'03, PBH00a, LOPSTR'99, LNCS'99, AADEBUG'97]

Outline

Part I The Ciao language

Part II The Ciao assertions model

Part III Using CiaoPP as a multi-language analyzer/verifier

Part IV Conclusion and some recent work

Outline

Part I The Ciao language

Part II The Ciao assertions model

Part III Using CiaoPP as a multi-language analyzer/verifier

Part IV Conclusion and some recent work

A Modular Language-Building Language

Ciao makes it very easy to build *syntactic and semantic extensions* in a flexible and scalable way.

- Addresses shortcomings of traditional Prolog `expand_term`, etc.:
  - Expansions defined for *semantic* points: goals, terms, heads, bodies, ... (not just a global `expand_term`) → *much easier coding*.
  - All operators, expansions, flags, etc. are *module-local*.
  - Dynamic and static code clearly separated, e.g.:
    - Syntax expansion code does not necessarily end up in executables.
    - Program syntax does not necessarily affect what is read.
  - Mechanisms for defining compositions of extensions.
  - New types of operators.
  - Higher-order syntax (e.g., `X(a)`), ...

→ Any extensions can be *activated* or *deactivated on per-module basis*.

→ The concept of *packages*.

[TPLP'12, PADL'13, LOPSTR'11, CL'00, ENTCS'00, CiaoManual'18, DCG-PhD'04, NovaSci'99, ILPS'95, PPCP'94]

A Modular Language-Building Language (Contd.)

Fundamental enabler –Ciao's module/class system.

Allows also:

- Modular program development, separate/incremental compilation.
- Modular (scalable) global analysis for detecting errors and optimizing.
- Also, building small, fast executables and embeddability (non-needed parts of the language and libraries are not included).

- All these mechanisms are easily accessible to the programmer for building extensions, restrictions (language subsets), DSLs, etc.

- Ciao is itself built in layers over a small (LP-based) *kernel*.
  - Built-ins are *in libraries* (and can be redefined or not loaded).
  - Same with all language features (loops, conditionals, functions, ',' ...).

[TPLP'12, PADL'13, LOPSTR'11, CL'00, ENTCS'00, DCG-PhD'04, CiaoManual'18, NovaSci'99, ILPS'95, PPCP'94]

Supporting Traditional Logic Programming

Is it still a Prolog system?

- Yes, indistinguishable to the naked eye!
- As ISO-Prolog compliant as other popular Prologs.
- Quite compatible with de-facto standards (e.g., SICStus).
- Standard predicates, libraries, etc.

However, inside:

- No "builtins:" Prolog support is in libraries, which *can be unloaded*.
- All Prolog libraries loaded automatically for Prolog programs.

- This allows having, e.g., *pure LP* modules (no cut, no assert, ...).
- Also, other computation rules: breadth-first, iterative-deepening, Andorra, *tabling*, *fuzzy* rules, ASP, etc.

All through the mechanism of packages, loadable on a per-module basis:

```
:- module(_,exports).            % All traditional built-ins available
:- module(_,exports,[]).         % Just the pure kernel
:- module(_,exports,[packages]). % Kernel+packages
:- module(_,exports,[iso]).      % Pure ISO
```

[ICLP'18, TR'95, CiaoManual'18, PPDP'16, FLOPS'12, ICLP'10, ICLP'09, PADL'09, PADL'08]

Supporting the Best Features of Other Paradigms

Multi-paradigm:

- *Constraint programming:* clpr, clpq, Leuven CHR, fd, ...
- *Functional programming:*
  - Function definitions, function calls, functional syntax for predicates.
  - *Higher-order* and *lazyness* for functions and predicates.
- *Concurrency, parallelism, distributed execution*.
- *Imperative features*: mutables, assignment, loops, cases, arrays, etc.
- *Objects:* a naturally embedded notion of classes and objects.

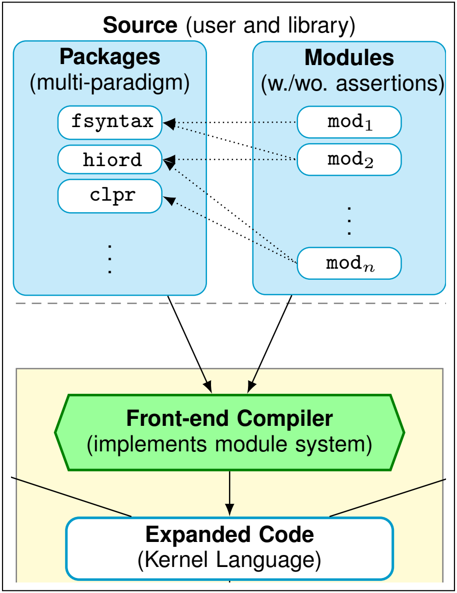
+ many other packages:

- Records, named argument positions.
- Logical interface to databases. Persistence.

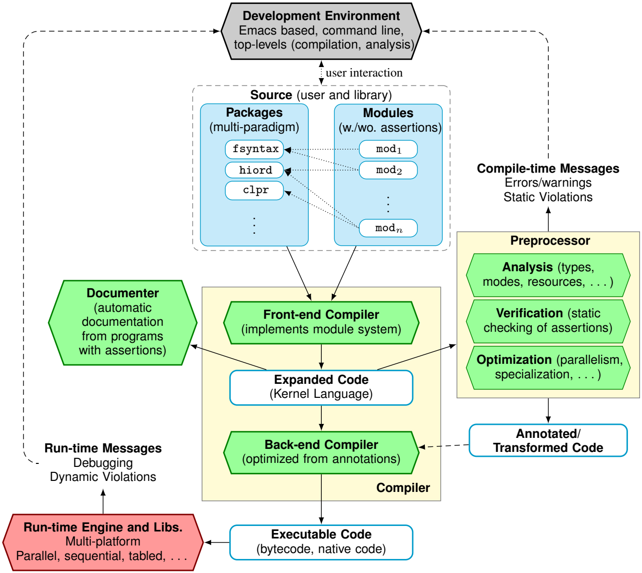
...

[PPDP'14, FLOPS'06, ASIAN'04, EuroPar'04, ICLP'02, CICLOPS'02, ICLP'99, ICLP'95, ProDe'96, CLNet'95, PPCP'94, ALP'94, CICLOPS'12]

Ciao Overview: Language Extensions



Ciao Architecture Overview



Development Environment
Emacs based, command line,
top-levels (compilation, analysis)

user interaction

Source (user and library)

Packages (multi-paradigm)
- fsyntax
- hiord
- clpr
- ⋮

Modules (w./wo. assertions)
- $\texttt{mod}_1$
- $\texttt{mod}_2$
- ⋮
- $\texttt{mod}_n$

Compile-time Messages
Errors/warnings
Static Violations

Preprocessor

Analysis (types, modes, resources, . . .)

Verification (static checking of assertions)

Optimization (parallelism, specialization, . . .)

Documenter (automatic documentation from programs with assertions)

Front-end Compiler (implements module system)

Expanded Code (Kernel Language)

Back-end Compiler (optimized from annotations)

Compiler

Annotated/ Transformed Code

Run-time Messages
Debugging
Dynamic Violations

Run-time Engine and Libs.
Multi-platform
Parallel, sequential, tabled, . . .

Executable Code (bytecode, native code)

Demo: properties, types, predicates, functions, higher order, constraints,

breadth-first search, tabling, ...

Outline

Part I The Ciao language

Part II The Ciao assertions model

Part III Using CiaoPP as a multi-language analyzer/verifier

Part IV Conclusion and some recent work

Dynamic vs. Static — An Almost Religious Argument!

Dynamic languages (Prolog, Lisp/Scheme, Python, Javascript, ...)

- Dynamic checking of types (and many other properties):
  - `..., A is B+C, ...`
    `B` and `C` checked to be `numexpr` by `is/2` at run time.
  - `..., arg(N,T,A), ...`
    `N` checked to be `nat` & $\leq$ `arity(T)` by `arg/3` (array bounds).
- Need to use tags (*boxing* of data) to identify type, var/nonvar, etc.
- Flexibility, compactness, rapid prototyping, scripting, ...

Static languages (ML, Haskell, Mercury, Gödel, ...)

- Compiler checks statically *types*.
- No dynamic checks needed for types.
- Safety guarantees (types), scalability, performance, large systems, ...

- Some languages (e.g., C) are neither (even if still very useful!):
  no checking of, e.g., array bounds at compile time or run time...

[AADEBUG'97, LNCS'99, LOPSTR'99, PBH00a, SAS'03, SCP'05, LPAR'06, ICLP'09]

Solving the Dynamic vs. Static Dilemma

The Ciao Approach:

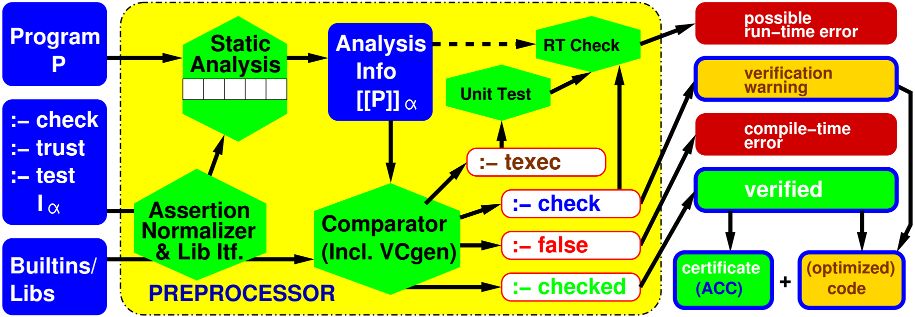
- Provide the flexibility of dynamic languages, but with
- *Guaranteed safety, reliability, and efficiency.*

- Use of *voluntary assertions* to express desired properties (incl. types).
  - Can be added up front, gradually, or not at all.
- Use of *advanced program analysis* (abstract interpretation) for:
  - Guaranteeing the properties as much as possible at compile-time.
  - Achieving high performance:
    - Eliminating run time checks at compile time.
    - Unboxing.
    - Specialization, slicing, ...
    - Automatic parallelization.

- Integrated Approach to Specification, Verification, Testing, Debugging, and Optimization.

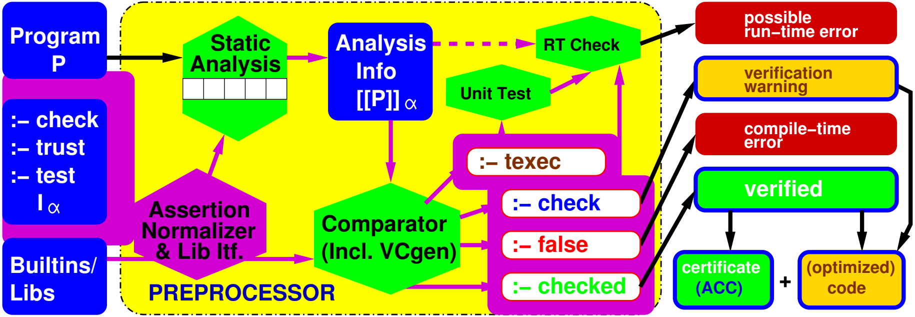
[AADEBUG'97, LNCS'99, LOPSTR'99, PBH00a, SAS'03, SCP'05, LPAR'06, ICLP'09]

The Ciao Integrated Approach to Specification, Debugging, Verification, Testing, and Optimization (Mostly Mid 90's!)



[AADEBUG'97, LNCS'99, ICLP'99, LOPSTR'99, PBH00a, SAS'03, LPAR'04, ICLP'09, TPLP'18, FOPARA'12, ICLP'10]

The Assertion Language



- Assertions optional, can be added at any time. Provide partial spec.
- *Sets* of pre/post/global triples (+ “status” field, documentation, ...).
- Used everywhere, for many purposes (incl. doc gen., foreign itf).
- System makes it worthwhile for the programmer to include them.
- Part of the programming language and “runnable.”

[ESOP’96, AADEBUG’97, ILPS-WS’97, LNCS’00, ICLP’09, PPDP’14, LOPSTR-Informal’18]

The Assertion Language (Subset)

```
:- pred Pred [:Precond] [=> Postcond] [+ CompProps ] .
```

Each typically a “mode” of use; the set *covers the valid calls*.

```
:- pred quicksort(X,Y) : list(int) * var => sorted(Y) + (is_det,not_fails).
:- pred quicksort(X,Y) : var * list(int) => ground(X) + non_det.
```

Properties; from libraries or user defined (in the source language):

```
:- regtype color := green | blue | red.
:- regtype list(X) := [] | [ X|list].          ≡   list(_,[]). list(X,[H|T]) :- X(H), list(X,T).
:- prop sorted := [] | [ _ ] | [X,Y|Z] :- X > Y, sorted([Y|Z]).
```

Types/shapes, cost, data sizes, aliasing, termination, determinacy, non-failure, ...

Program-point Assertions:

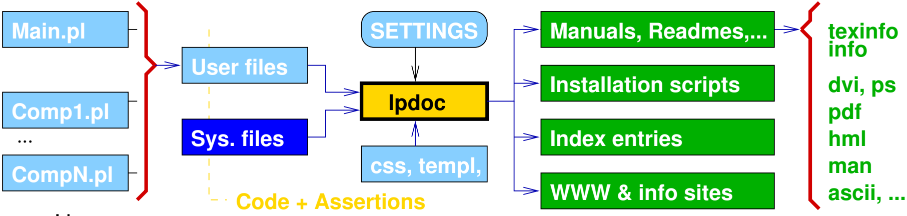
- Inlined with code: `..., check( int(X), X>0, mshare([[X]]) ), ....`

Assertion Status (so far “to be checked” – check status – default)

- Also: **trust** (guide analyzer), **true**/**false** (analysis output), **test**, etc.

[ESOP’96, AADEBUG’97, ILPS-WS’97, LNCS’00, ICLP’09, PPDP’14, LOPSTR-Informal’18]

Autodocumenter: LPdoc



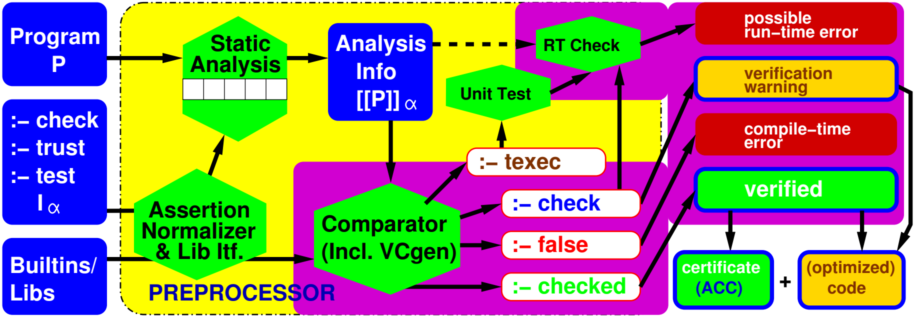
- Uses:
  - All the information that the compiler has.
  - **Assertions.**
  - Doc declarations and “active comments:”

```
:- doc(title,"Complex numbers library").
:- doc(summary,"Provides an ADT for complex numbers.").

%! \title   Complex numbers library
%! \summary Provides an ADT for complex numbers
```

  - Markup language (close to LaTeX/texinfo) + markdown.
  - All Ciao manuals, tutorials, and sites generated with LPdoc.

[CL2000]

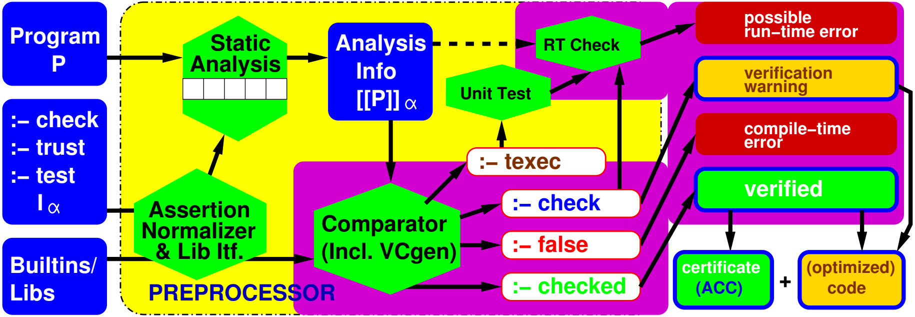
Integrated Static/Dynamic Debugging and Verification



	Definition	Sufficient condition
P is prt. correct w.r.t. $\mathcal{I}_\alpha$ if	$\alpha(\llbracket P \rrbracket) \leq \mathcal{I}_\alpha$	$\llbracket P \rrbracket_{\alpha^+} \leq \mathcal{I}_\alpha$
P is complete w.r.t. $\mathcal{I}_\alpha$ if	$\mathcal{I}_\alpha \leq \alpha(\llbracket P \rrbracket)$	$\mathcal{I}_\alpha \leq \llbracket P \rrbracket_{\alpha^=}$
P is incorrect w.r.t. $\mathcal{I}_\alpha$ if	$\alpha(\llbracket P \rrbracket) \not\leq \mathcal{I}_\alpha$	$\llbracket P \rrbracket_{\alpha^=} \not\leq \mathcal{I}_\alpha$, or $\llbracket P \rrbracket_{\alpha^+} \cap \mathcal{I}_\alpha = \emptyset \wedge \llbracket P \rrbracket_\alpha \neq \emptyset$
P is incomplete w.r.t. $\mathcal{I}_\alpha$ if	$\mathcal{I}_\alpha \not\leq \alpha(\llbracket P \rrbracket)$	$\mathcal{I}_\alpha \not\leq \llbracket P \rrbracket_{\alpha^+}$

[AADEBUG'97, LNCS'99, LOPSTR'99, PBH00a, SAS'03, PPDP'05, LPAR'06, PEPM'08, ICLP'09]

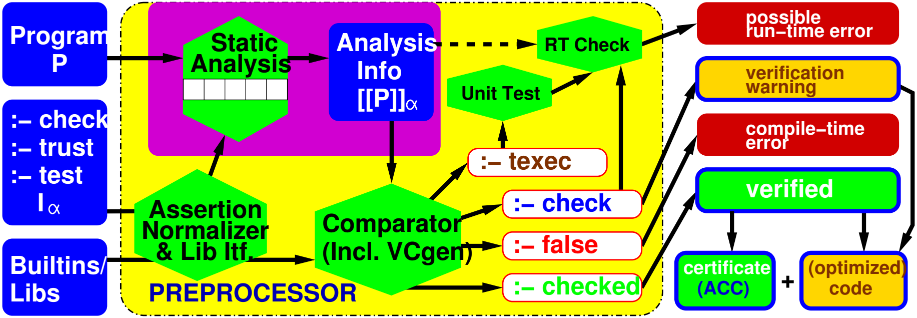
Integrated Static/Dynamic Debugging and Verification



- Based throughout on the notion of *safe approximation* (abstraction).
- Run-time checks generated for *parts* of asserts. not verified statically.
- Diagnosis (for both static and dynamic errors).
- Comparison not always trivial: e.g., resource debugging/certification
  - Need to compare functions.
  - “Segmented” answers.

[AADEBUG’97, LNCS’99, LOPSTR’99, PBH00a, SAS’03, PPDP’05, LPAR’06, PEPM’08, ICLP’09]

The Analyses



Program P

:– check
:– trust
:– test
I_α

Builtins/ Libs

Static Analysis

Analysis Info $[[P]]_\alpha$

RT Check

Unit Test

Assertion Normalizer & Lib Itf.

Comparator (Incl. VCgen)

:– texec

:– check

:– false

:– checked

PREPROCESSOR

possible run–time error

verification warning

compile–time error

verified

certificate (ACC)

+

(optimized) code

- Modular, parametric, polyvariant abstract interpretation.
- Accelerated, incremental fixpoint.
- Properties:
  - Shapes, data sizes, sharing/aliasing, determinacy, exceptions, termination, ...
  - Resources (time, memory, energy, ...), (user-defined) resources.

[LOPSTR'07] [JLP'92, TOPLAS'99, SAS'96, TOPLAS'00, FTfJP'07, LOPSTR-Informal'18, ICLP'18]
[POPL'94, ESOP'96, ENTCS'00, JLP'97, TOPLAS'96, TOPLAS'95, LOPSTR'01, LPAR'06, PEPM'08]
[ICLP'88, NACLP'89, ICLP'91, ICLP'97, SAS'02, FLOPS'04, LOPSTR'04, PADL'06, ICLP'08]
[VMCAI'08, LCPC'08, PASTE'08, CC'08, ISMM'09, NGC'10, LCPC'08] [PLDI'90, PASCO'94, JSC'96, SAS'94, ILPS'97]
[CLEI'06, ICLP'07, PPDP'08, NASA FM'08, Bytecode'09, ICLP'10, ICLP'13, LOPSTR'13, TPLP'14, ICLP'16, FLOPS'16]

Demo: assertions, static errors (types, data sizes, procedure cost, non-determinacy, ...), run-time check generation, certification, unit tests...

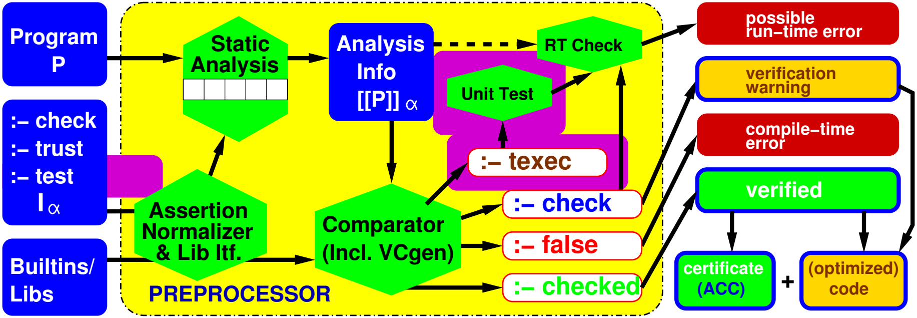
Discussion: Comparison with *Classical* Types

"Traditional" Types	**Ciao Assertion-based Model**
"Properties" limited by decidability	Much more general property language
May need to limit prog. lang.	No need to limit prog. lang.
"Untypable" programs rejected	Run-time checks introduced
(Almost) Decidable	Decidable + Undecidable (approximated)
Expressed in a different language	Expressed in the source language
Types must be defined	Types can be defined or inferred
Assertions are only of type "check"	"check", "trust", ...
Type signatures & assertions different	Type signatures *are* assertions

- But quite popular now: gradual typing, Racket, liquid Haskell, etc.
- Some key issues:
  Safe / Sound approximation *Suitable assertion language*
  Abstract Interpretation *Powerful abstract domains*
- Works best when properties and assertions can be expressed in the source language (i.e., source lang. supports *predicates*, *constraints*).

[AADEBUG'97, LNCS'99, ICLP'99, LOPSTR'99, PBH00a, SAS'03, SCP'05, LPAR'06, ICLP'09]

Integration of Testing



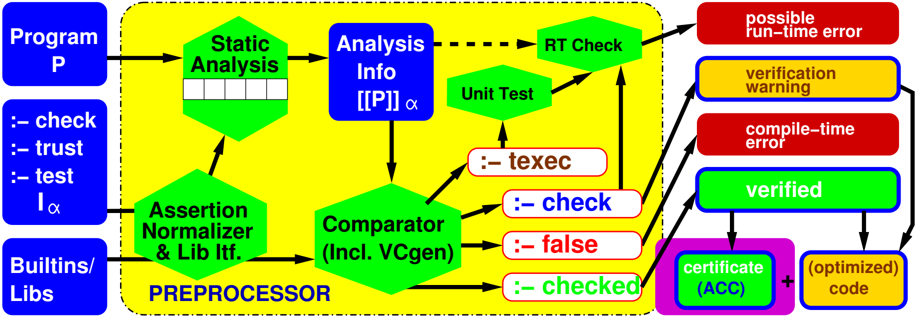
Many interactions within the integrated framework:

- (Unit) tests are part of the assertion language:

  `:- test Pred [:Precond] [=>Postcond] [+CompExecProps].`
- Parts of unit tests that can be verified at compile-time are deleted.
- Unit testing uses the run time assertion-checking machinery.
- Unit tests also provide test cases for the run-time checks.
  - Assertions checked by unit testing, even if not conceived as tests.

[ICLP'09]

Abstraction-based Certification, Abstraction-Carrying Code



PRODUCER

$[\![P]\!]_\alpha$ = Analysis = **lfp**(*analysis_step*)

Certificate $\subset [\![P]\!]_\alpha$
Certificate → Safety Policy

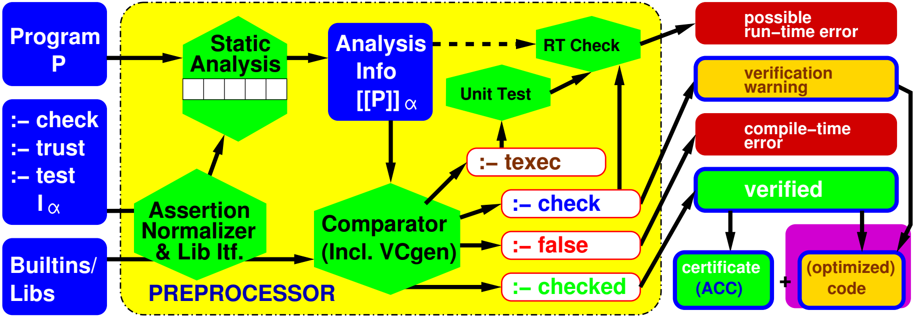
CONSUMER

Checker = *analysis_step*

- Interesting extensions: reduced certificates, incrementality, ...

[LPAR'04, COCV'04, PPDP'05, TPLP'12]

Optimization



- Source-level optimizations:
  - Partial evaluation, (multiple) (abstract) specialization, ...
- Low-level optimizations (e.g., dynamic check elimination, unboxing):
  - Use of specialized instructions.
  - Optimized native code generation.

→ obtaining close-to-C performance for dynamic languages.

- Parallelization. Granularity control.

[PADL'04, CASES'06] [TPLP'11, SAS'06, PPDP'06, LOPSTR'05, PEPM'03, PLILP'91, PEPM'99, JLP'99] [TCS'09, LOPSTR'07, TOPLAS'00] [TOPLAS'99, JLP'99, PLILP'96, CompLangJ'96, JLP'95, SAS'94, ICLP'91, NGC'91, ?] [DAMP'07, JSC'96, PASCO'94, PLDI'90]

Partial Evaluation

CiaoPP includes a state-of-the-art partial evaluator

- Partial *deduction* (i.e., fold/unfold), poly-controlled, efficient, slicing, conjunctive, ...
- Polyvariant, modular **abstract** specialization (i.e., specialization w.r.t. abstract values).

→ Fully interleaved **integration of partial evaluation within abstract interpreter** ("Abstract Interpretation with Specialized Definitions").

Shown strictly more powerful than any separate analysis/peval approach.

- Applied together with lower-level optimizations to parallelization, run-time check optimization, etc.

[TPLP'11, SAS'06, PPDP'06, LOPSTR'05, PEPM'03, PEPM'99, JLP'99, PLILP'91]

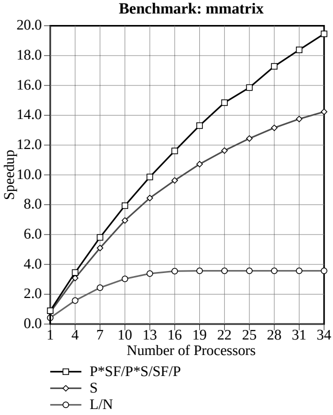
Optimizing Compilation

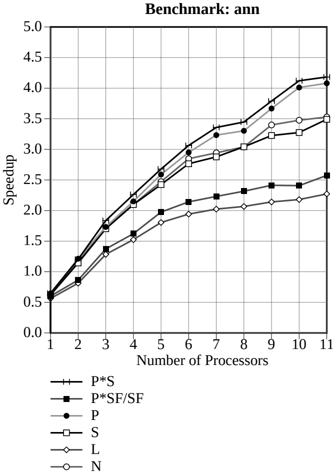
- Combine high-level instantiation, regular types, determinism, non-failure analyses with low-level analyses (choice-point analysis, heap usage, liveness, etc.). Used in multiple optimizations [PADL'04]:
  - Specialized unification, builtins, control structures; unboxing of argument and local variables (e.g., untagged int).
  - Same mechanism useful to reproduce hand-made instruction specializations in the bytecode emulator (reconstructed from a reduced set of unoptimized instructions) [TPLP'16, LOPSTR'06] → automatic generation of specialized emulators (e.g., with no non-determinism if not used).
- Emulator written in Ciao dialect [LOPSTR'06], multiple backends (e.g., Javascript [ICLP'12]).
- Case study for small devices (sound spatializer) [CASES'06]:

Compilation mode	Non-Specialized	Specialized
Bytecode	25.64	14.00
N.C. via C	21.59	11.99
Id. + semidet	19.59	11.53
Id. + mode/type	19.19	11.08
Id. + unboxing	6.97	**3.62**

[TPLP'16, LOPSTR'15, ICLP'12, PPDP'08, LOPSTR'06, CASES'06, ICLP'05, PADL'04]

Some Speedups (Using Different Abstract Domains)

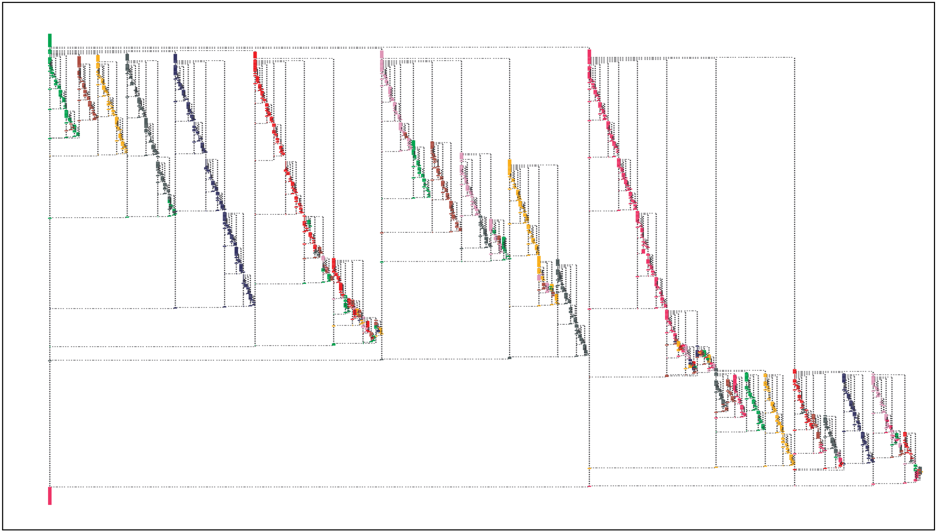




(ann: parallelizer parallelizing itself; 1-10 proc.: actual speedups on Sequent Symmetry; 10+ simulator projections from execution traces)

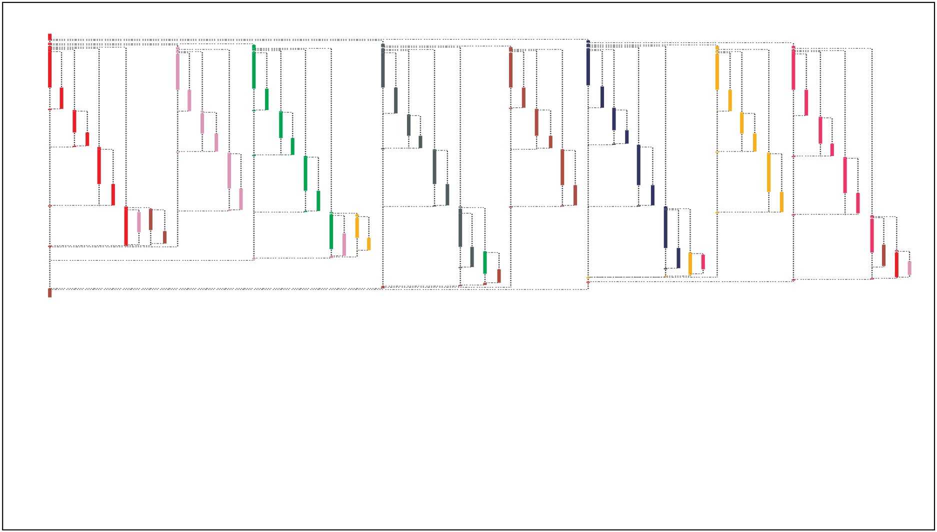
[TCS'09, LOPSTR'07, EuroPar'07, TOPLAS'00, TOPLAS'99, JLP'99, JLP'99, PLILP'96, CompLangJ'96, JLP'95, SAS'94]
[ICLP'91, PLILP'91, NGC'91, ICLP'90, NACLP'89]

8 Processors



[DAMP’07, JSC’96, ICLP’95, PASCO’94, PLDI’90]

8 Processors, with Granularity Control (Same Scale)



[DAMP'07, JSC'96, ICLP'95, PASCO'94, PLDI'90]

Outline

Part I The Ciao language

Part II The Ciao assertions model

Part III Using CiaoPP as a multi-language analyzer/verifier

Part IV Conclusion and some recent work

Early 2000's

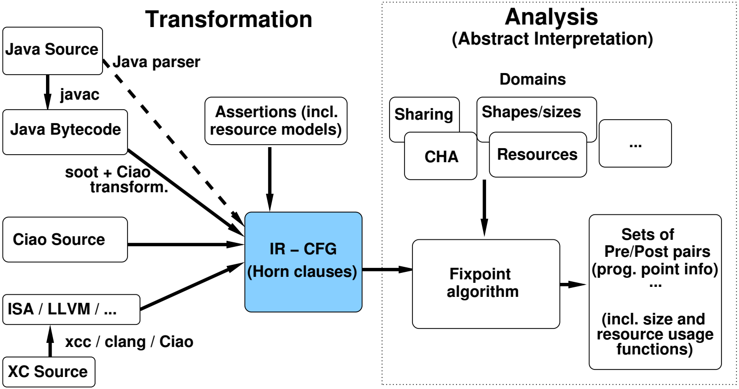
- Generalization of Ciao assertions model / CiaoPP to analyzing/verifying different languages by translation to Horn clauses. [LOPSTR'07]
  - Also allows analyzing/verifying multi-language applications.
  - First Constraint Horn Clause-based analyzer for non-logic languages?
- Extension to other properties (timing, energy).

→ Using CiaoPP as a general-purpose analysis and verification tool.

- The technique was already used early on (in our MA3 analyzer): we called it "abstract compilation" [ICLP'88].
- Approach quite popular now!

[LOPSTR'07, ICLP'88]

CiaoPP Intermediate Repr.: (Constraint) Horn Clauses



- Transformation:
  - Source: Program P in L_P + (possibly abstract) Semantics of L_P
  - Target: A (C) Horn Clause program capturing $[\![P]\!]$ (or, possibly, $[\![P]\!]^\alpha$)
- Block-based CFG. Each block represented as a *Horn clause*.
- Used for all analyses: aliasing, CHA/shape/types, data sizes, resources, etc.
- Allows supporting multiple languages.

[LOPSTR'07, ICLP'88]

Transformation Example - Java Source

Multiplying numbers from 1 to n (factorial)

source

```
public static int r(int n)
{
  int ans=1;
  while(n>0)
  {
    ans*=n;
    n-=1;
  }
  return ans;
}
```

Horn clause IR

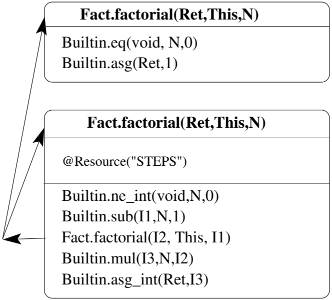
```
:- pred r/2 : num * var.

r(N,Ret) :-
        Ans .=. 1,
        r_1(N, Ans, Ret).

r_1(N, Ans, Ret) :-
        N .>. 0,
        mul(Ans, N, Ans1),
        N1 .=. N - 1,
        r_1(N1, Ans1, Ret).
r_1(N, Ans, Ret):-
        N .=<. 0,
        Ret .=. Ans.
```

The HC IR:

- Reduces everything to recursions, which simplifies analysis.
- Makes the semantics of loops, recursions, etc. precise: all variables, increments/decrements, scoping, etc. are explicit.
- Allows "small-step" and "big-step" variants.

[LOPSTR'07, ICLP'88, VMCAI'08, FTfJP'07, Bytecode'07, Bytecode'09]

Transformation Example - Java Bytecode



Fact.factorial(Ret,This,N)

Builtin.eq(void, N,0)
Builtin.asg(Ret,1)

Fact.factorial(Ret,This,N)

@Resource("STEPS")

Builtin.ne_int(void,N,0)
Builtin.sub(I1,N,1)
Fact.factorial(I2, This, I1)
Builtin.mul(I3,N,I2)
Builtin.asg_int(Ret,I3)

→

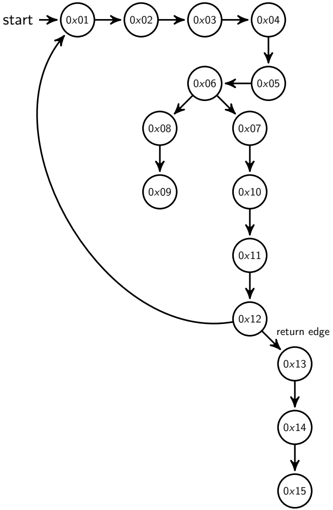
```
:- entry 'Fact.factorial'/3:var*atm*num.

'Fact.factorial'(Ret, This, N):-
        eq_int(void,N,int,0,int),
        asg_int(Ret,int,1,int).

'Fact.factorial'(Ret, This, N):-
        ne_int(void,N,int,0,int),
        sub(I1,int,N,int,1,int),
        Fact.factorial(I2,This,I1),
        mul(I3, int,N,int,I2,int),
        asg_int(Ret,int,I3,int).
```

[LOPSTR'07, ICLP'88, Bytecode'09, VMCAI'08, FTfJP'07, Bytecode'07, Bytecode'09, NASA FM'08]

Transformation example - binaries

Xcore ISA Example: Control Flow Graph (CFG)

```
<fact>:
0x01: entsp (u6)    0x2
0x02: stw   (ru6)   r0, sp[0x1]
0x03: ldw   (ru6)   r1, sp[0x1]
0x04: ldc   (ru6)   r0, 0x0
0x05: lss   (3r)    r0, r0, r1
0x06: bf    (ru6)   r0, 0x1 <0x08>
0x07: bu    (u6)    0x2 <0x10>
0x08: mkmsk (rus)   r0, 0x1
0x09: retsp (u6)    0x2
0x10: ldw   (ru6)   r0, sp[0x1]
0x11: sub   (2rus)  r0, r0, 0x1
0x12: bl    (u10)   -0xc <fact>
0x13: ldw   (ru6)   r1, sp[0x1]
0x14: mul   (l3r)   r0, r1, r0
0x15: retsp (u6)    0x2
```

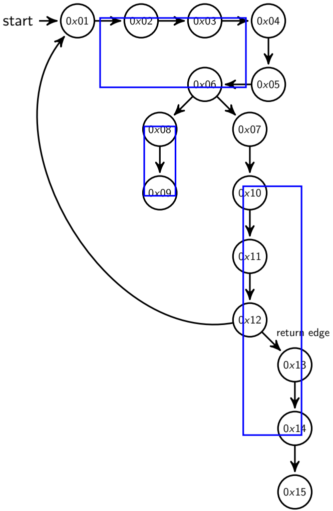


[LOPSTR'07, ICLP'88, TPLP'18, FOPARA'15, HIP3ES'15, LOPSTR'13]

Transformation example - binaries

Xcore ISA Example: Block Representation

```
<fact>
0x01: entsp (u6)    0x2
0x02: stw  (ru6)    r0, sp[0x1]
0x03: ldw  (ru6)    r1, sp[0x1]
0x04: ldc  (ru6)    r0, 0x0
0x05: lss  (3r)     r0, r0, r1
0x06: bf  (ru6)     r0, 0x1 <0x08>

0x07: bu  (u6)      0x2 <0x10>
0x10: ldw  (ru6)    r0, sp[0x1]
0x11: sub  (2rus)   r0, r0, 0x1
0x12: bl  (u10)     -0xc <fact>
0x13: ldw  (ru6)    r1, sp[0x1]
0x14: mul  (l3r)    r0, r1, r0
0x15: retsp (u6)    0x2

0x08: mkmsk  (rus)  r0, 0x1
0x09: retsp (u6)    0x2
```

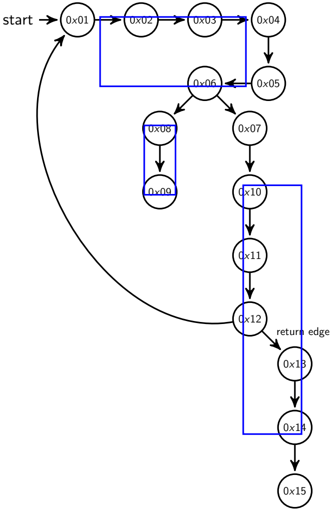


[LOPSTR'07, ICLP'88, TPLP'18, FOPARA'15, HIP3ES'15, LOPSTR'13]

Transformation example - binaries

Xcore ISA Example: Constrained Horn Clauses IR

```
:- entry fact/2.
fact(R0,R0_3):-
      entsp(_),
      stw(R0,Sp0x1),
      ldw(R1,Sp0x1),
      ldc(R0_1,0x0),
      lss(R0_2,R0_1,R1),
      bf(R0_2,_),
      bf01(R0_2,Sp0x1,R0_3,R1_1).

bf01(1,Sp0x1,R0_4,R1):-
      bu(_),
      ldw(R0_1,Sp0x1),
      sub(R0_2,R0_1,0x1),
      bl(_),
      fact(R0_2,R0_3),
      ldw(R1,Sp0x1),
      mul(R0_4,R1,R0_3),
      retsp(_).

bf01(0,Sp0x1,R0,R1):-
      mkmsk(R0,0x1),
      retsp(_).
```



[LOPSTR'07, ICLP'88, TPLP'18, FOPARA'15, HIP3ES'15, LOPSTR'13]

Generating the Intermediate Representation

- Typical tasks:
  - Generation of block-based CFG.
  - SSA transformation (e.g., splitting of input/output param).
  - Conversion of loops into recursions among blocks.
  - Branching, cases, dynamic dispatch → blocks w/same signature.
- Some specifics for Java bytecode:
  - Elimination of stack variables.
  - Conversion to three-address statements.
  - Explicit representation of this and ret as extra block parameters.
- Some specifics for binaries:
  - Control flow graph is constructed from ISA or LLVM IR representation.
  - Inferring block parameters.
  - Resolving branching to predicates with multiple clauses.
- Can be done via:
  - **partial evaluation of an interpreter** (implementing the semantics of the low-level code) w.r.t. the concrete low-level program or
  - **directly** (cf. Futamura projections).

[LOPSTR'07, ICLP'88, NASA FM'08, VMCAI'08, Bytecode'09, FOPARA'15, LOPSTR'13, HIP3ES'15, TPLP'18]

User-Definable Resources: Some Results (Java Bytecode)

Program	Resource(s)	t	Resource Usage Func. (*) / Metric	
BST	Heap usage	367	$O(2^n)$	$n \equiv$ tree depth
CellPhone	SMS monetary cost	386	$O(n^2)$	$n \equiv$ packets length
Client	Bytes received and	527	$O(n)$	$n \equiv$ stream length
	Bandwidth required		$O(1)$	—
Dhrystone	Energy consumption	759	$O(n)$	$n \equiv$ int value
Divbytwo	Stack usage	219	$O(log_2(n))$	$n \equiv$ int value
Files	Files left open and	649	$O(n)$	$n \equiv$ number of files
	Data stored		$O(n \times m)$	$m \equiv$ stream length
Join	DB accesses	460	$O(n \times m)$	$n, m \equiv$ table records
Screen	Screen width	536	$O(n)$	$n \equiv$ stream length

- Different cost/complexity functions, resources, size metrics, types of loops/recursion, etc.

(*) We represent just order for brevity (but full upper and lower bounds inferred).

[Bytecode'09, VMCAI'08, LOPSTR'07, FTfJP'07]

Time Bound Function Analysis

Cost model: cost assertions with the (WAM) bytecode instruction execution costs:

Examples

```
:- trust pred unify_variable(A, B) : int(A), int(B)
   + ( cost(ub, exectime, 667.07),
       cost(lb, exectime, 667.07) ).

:- trust pred unify_variable(A, B) : var(A), gnd(B)
   + ( cost(ub, exectime, 233.3),
       cost(lb, exectime, 233.3) ).

:- trust pred unify_variable(A, B): list(A), list(B)
   + cost(ub, exectime, 271.58+284.34*length(A) ).
...
```

- Automatically generated in a one-time profiling phase (regression).
- Made available to static cost analyzer, which uses it to infer timing bound functions for any program.

[PPDP'08, PADL'07, ICLP'07]

Observed and Estimated Execution Time (Intel)

Pr.	Cost.	Intel (μs)				
No.	App.	Est.	Prf.	Obs.	D. %	Pr.D. %
1	E	110	110	113	-2.4	-2.4
2	E	69	69	71	-2.3	-2.3
3	E	1525	1525	1576	-3.3	-3.3
4	E	1501	1501	1589	-5.7	-5.7
5	E	2569	2569	2638	-2.7	-2.7
6	E	1875	1875	2027	-7.8	-7.8
7	E	1868	1868	1931	-3.3	-3.3
8	L	43	68	81	-67.2	-17.8
	U	3414	3569	3640	-6.4	-2.0
9	L	54	79	91	-54.6	-14.8
	U	3414	3694	4011	-16.2	-8.2
10	L	135	142	124	8.6	13.7
	U	7922	2937	2858	120.6	2.7
11	L	216	138	111	72.3	22.5
	U	226	216	162	34.0	29.5

[PPDP'08, PADL'07, ICLP'07]

Energy Consumption Analysis – Approach

Requires low-level modeling – approach: [NASA FM'08]

- Specialize our parametric resource analysis with instruction-level models:
  - Provide energy and data size assertions for each individual instruction. (Energy and data sizes can be constants or *functions*.)
- CiaoPP then generates statically safe upper- and lower-bound energy consumption functions.

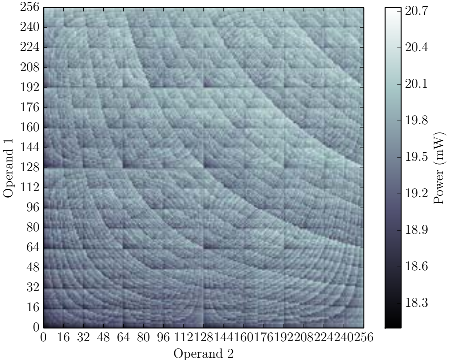
⇒ Addressed recently:

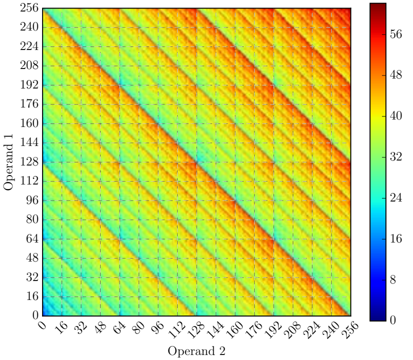
- Analysis of (embedded) programs written in XC, on XMOS processors.
- Using more sophisticated *ISA- and LLVM-level energy models* for XMOS XS1 (Bristol & XMOS).
- Comparing to measured energy consumption.

[NASA FM'08, LOPSTR'13, HIP3ES'15, FOPARA'15, HIP3ES'16, TPLP'18]

Low-level ISA Characterization – Operand Size

Obtaining the Cost Model: Energy Consumption/Instruction; Operand Size.

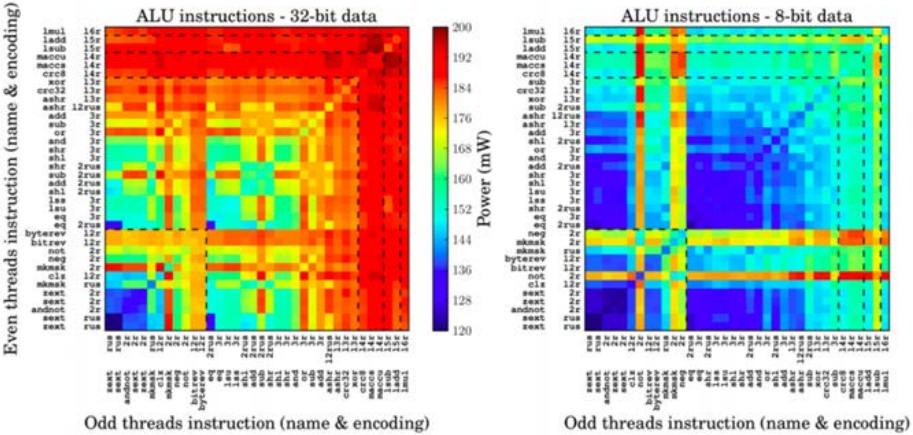




Eder, Kerrison – Bristol U / XMOS.

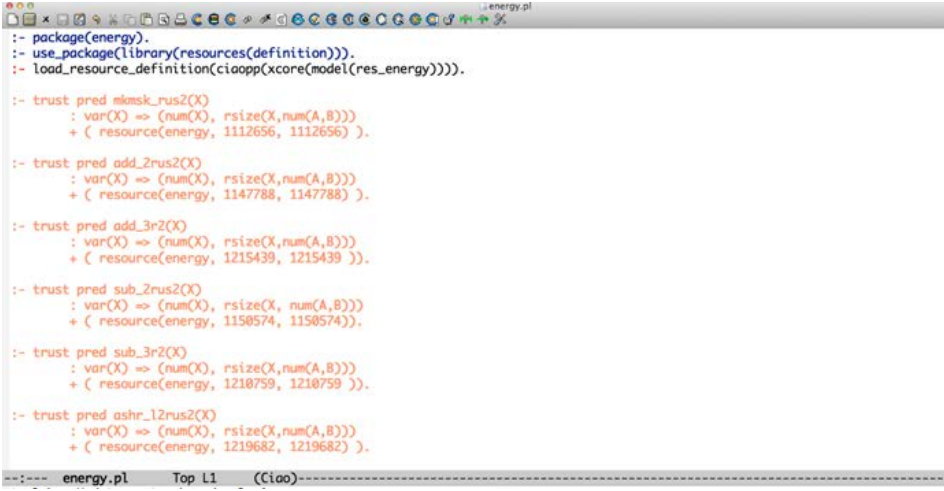
Low-level ISA Characterization – Interference

Obtaining the Cost Model: Energy Consumption/Instruction; Interference.



Eder, Kerrison – Bristol U / XMOS.

Energy Model, Expressed in the Ciao Assertion Language

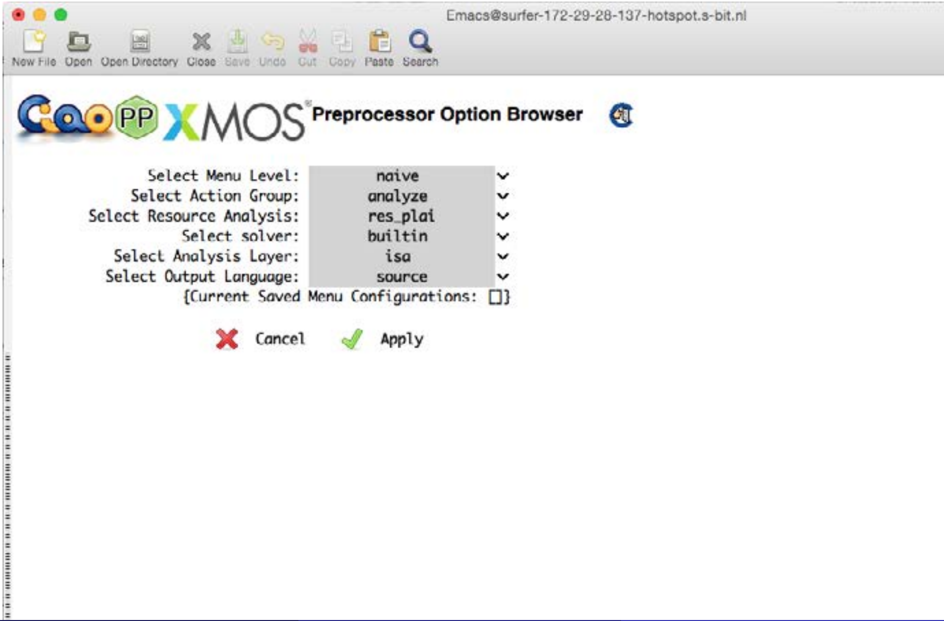


```
:- package(energy).
:- use_package(library(resources(definition))).
:- load_resource_definition(ciaopp(xcore(model(res_energy)))).

:- trust pred mkmsk_rus2(X)
        : var(X) => (num(X), rsize(X,num(A,B)))
        + ( resource(energy, 1112656, 1112656) ).

:- trust pred add_2rus2(X)
        : var(X) => (num(X), rsize(X,num(A,B)))
        + ( resource(energy, 1147788, 1147788) ).

:- trust pred add_3r2(X)
        : var(X) => (num(X), rsize(X,num(A,B)))
        + ( resource(energy, 1215439, 1215439 )).

:- trust pred sub_2rus2(X)
        : var(X) => (num(X), rsize(X, num(A,B)))
        + ( resource(energy, 1150574, 1150574)).

:- trust pred sub_3r2(X)
        : var(X) => (num(X), rsize(X,num(A,B)))
        + ( resource(energy, 1210759, 1210759 )).

:- trust pred ashr_l2rus2(X)
        : var(X) => (num(X), rsize(X,num(A,B)))
        + ( resource(energy, 1219682, 1219682) ).

--:---  energy.pl      Top L1     (Ciao)----------------------------------------
```

Very simple model depicted (constant cost) but real models can include:

- Data properties: operand sizes or other (e.g., number of 1's, bits changing, ...).
- External parameters (voltage, clock, ...).
- List of previous instructions, pipeline state, cache state, etc.

CiaoPP Menu


Emacs@surfer-172-29-28-137-hotspot.s-bit.nl
New File
Open
Open Directory
Close
Save
Undo
Cut
Copy
Paste
Search
Preprocessor Option Browser

Select Menu Level: naive

Select Action Group: analyze

Select Resource Analysis: res_plai

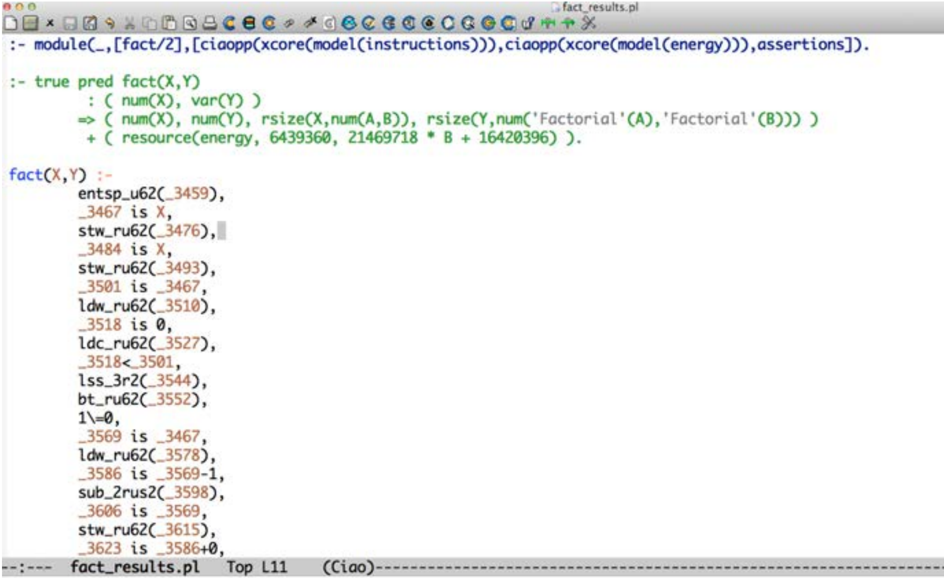
Select solver: builtin

Select Analysis Layer: isa

Select Output Language: source

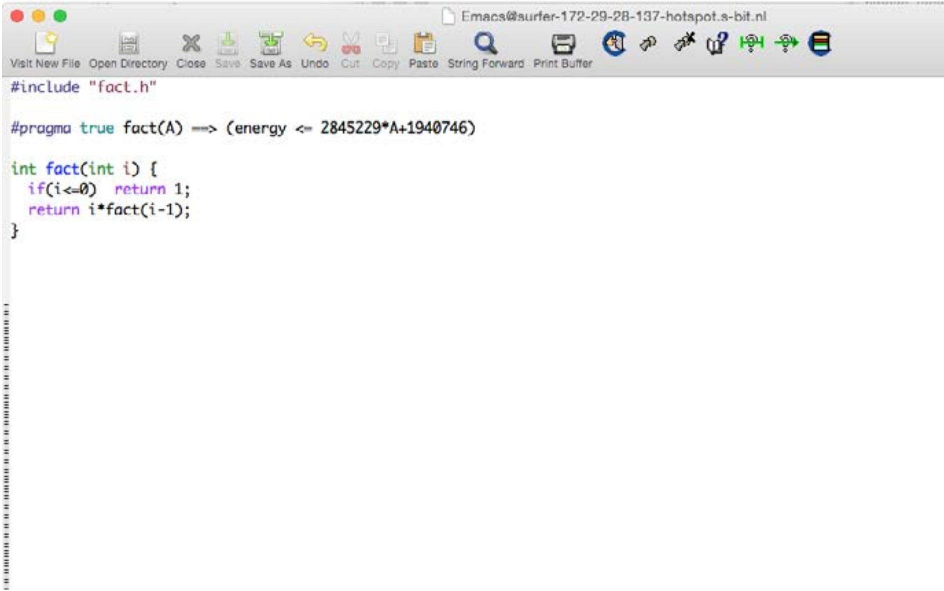
{Current Saved Menu Configurations: □}

Analysis Results



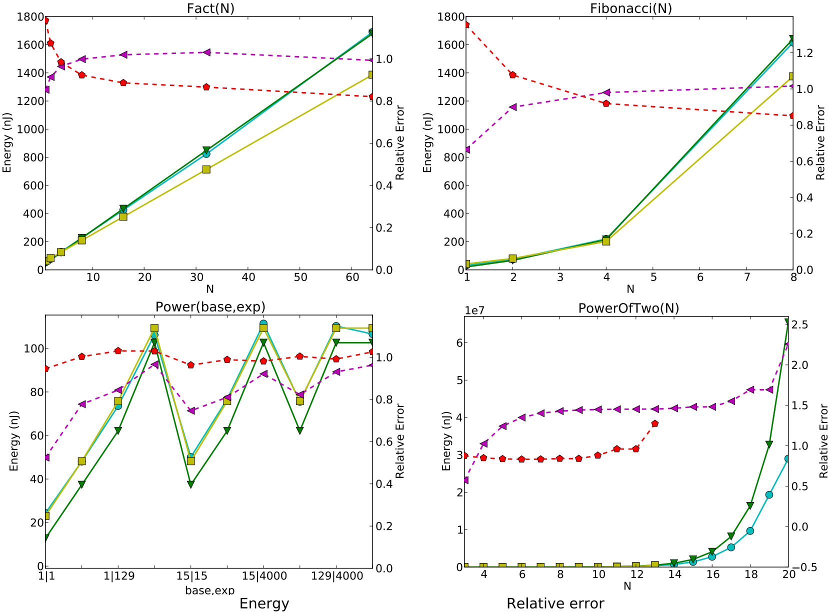
```
:- module(_,[fact/2],[ciaopp(xcore(model(instructions))),ciaopp(xcore(model(energy))),assertions]).

:- true pred fact(X,Y)
         : ( num(X), var(Y) )
        => ( num(X), num(Y), rsize(X,num(A,B)), rsize(Y,num('Factorial'(A),'Factorial'(B))) )
         + ( resource(energy, 6439360, 21469718 * B + 16420396) ).

fact(X,Y) :-
        entsp_u62(_3459),
        _3467 is X,
        stw_ru62(_3476),
        _3484 is X,
        stw_ru62(_3493),
        _3501 is _3467,
        ldw_ru62(_3510),
        _3518 is 0,
        ldc_ru62(_3527),
        _3518<_3501,
        lss_3r2(_3544),
        bt_ru62(_3552),
        1\=0,
        _3569 is _3467,
        ldw_ru62(_3578),
        _3586 is _3569-1,
        sub_2rus2(_3598),
        _3606 is _3569,
        stw_ru62(_3615),
        _3623 is _3586+0,
--:---  fact_results.pl    Top L11     (Ciao)-----------------------------------------
```

Analysis Output



```
#include "fact.h"

#pragma true fact(A) ==> (energy <= 2845229*A+1940746)

int fact(int i) {
  if(i<=0)  return 1;
  return i*fact(i-1);
}
```

Some Results [LOPSTR'13]



XC Analysis Results (FIR Filter, LLVM IR Level) [FOPARA'15]

```
#pragma true fir(xn, coeffs, state, N) :
             (3347178*N + 13967829 <= energy &&
              energy <= 3347178*N + 14417829)
```

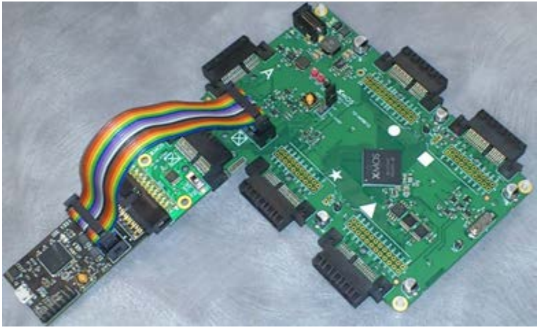
```
int fir(int xn, int coeffs[], int state[], int ELEMENTS)
{
  unsigned int ynl; int ynh;
  ynl = (1<<23); ynh = 0;
  for(int j=ELEMENTS-1; j!=0; j--) {
      state[j] = state[j-1];
      {ynh, ynl} = macs(coeffs[j], state[j], ynh, ynl);
  }
  state[0] = xn;
  {ynh, ynl} = macs(coeffs[0], xn, ynh, ynl);
  if (sext(ynh,24) == ynh) {
      ynh = (ynh << 8) | (((unsigned) ynl) >> 24);}
  else if (ynh < 0) { ynh = 0x80000000; }
  else { ynh = 0x7fffffff; }
  return ynh;
}
```

XC Analysis Results (FIR Filter, LLVM IR Level) [FOPARA'15]

```
#pragma true fir(xn, coeffs, state, N) :
              (3347178*N + 13967829 <= energy &&
               energy <= 3347178*N + 14417829)

int fir(int xn, int coeffs[], int state[], int ELEMENTS)
{
  unsigned int ynl; int ynh;
  ynl = (1<<23); ynh = 0;
  for(int j=ELEMENTS-1; j!=0; j--) {
      state[j] = state[j-1];
      {ynh, ynl} = macs(coeffs[j], state[j], ynh, ynl);
  }
  state[0] = xn;
  {ynh, ynl} = macs(coeffs[0], xn, ynh, ynl);
  if (sext(ynh,24) == ynh) {
      ynh = (ynh << 8) | (((unsigned) ynl) >> 24);}
  else if (ynh < 0) { ynh = 0x80000000; }
  else { ynh = 0x7fffffff; }
  return ynh;
}
```

Measuring Power Consumption on the Hardware

- XMOS XTAG3 measurement circuit.
- Plugs into XMOS XS1 board.



We compare these HW measurements with:

- Static Resource Analysis (SRA).
- Instruction Set Simulation (ISS).

Accuracy vs. HW measurements (ISA and LLVM IR) [FOPARA'15]

Program	Error vs. HW		ISA/LLVMIR
	isa	llvmir	
fact(N)	2.86%	4.50%	0.94
fibonacci(N)	5.41%	11.94%	0.92
sqr(N)	1.49%	9.31%	0.91
power_of_two(N)	4.26%	11.15%	0.93
Average	**3.50%**	**9.20%**	**0.92**
reverse(N,M)	N/A	2.18%	N/A
concat(N,M)	N/A	8.71%	N/A
mat_mult(N,M)	N/A	1.47%	N/A
sum_facts(N,M)	N/A	2.42%	N/A
fir(N)	N/A	0.63%	N/A
biquad(N)	N/A	2.34%	N/A
Average	**N/A**	**3.0%**	**N/A**
Gobal Avg.	**3.50%**	**5.48%**	**0.92**

Accuracy vs. HW measurements (ISA and LLVM IR) [FOPARA'15]

Program	Error vs. HW		ISA/LLVMIR
	isa	llvmir	
fact(N)	2.86%	4.50%	0.94
fibonacci(N)	5.41%	11.94%	0.92
sqr(N)	1.49%	9.31%	0.91
power_of_two(N)	4.26%	11.15%	0.93
biquad(N)	N/A	2.34%	N/A
Average	**N/A**	**3.0%**	**N/A**
Gobal Avg.	**3.50%**	**5.48%**	**0.92**

- ISA analysis estimations are reasonably accurate.
- ISA estimations are more accurate than LLVM estimations.
- LLVM estimations are close to ISA estimations.
- Some programs cannot be analysed at the ISA level but can be analyzed at the LLVM level.

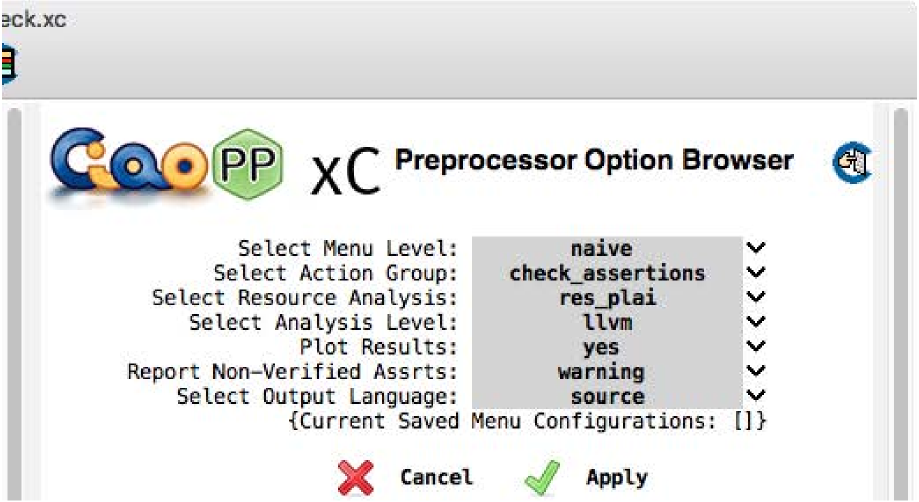
XC Program (FIR Filter) w/Energy Specification [HIP3ES'15, TPLP'18]

```
#pragma check fir(xn, coeffs, state, N) :
          (1 <= N) ==> (energy <= 416079189)

#pragma true fir(xn, coeffs, state, N) :
                (3347178*N + 13967829 <= energy &&
                 energy <= 3347178*N + 14417829)

#pragma checked fir(xn, coeffs, state, N) :
                (1 <= N && N <= 120) ==> (energy <= 416079189)

#pragma false fir(xn, coeffs, state, N) :
                (121 <= N) ==> (energy <= 416079189)

int fir(int xn, int coeffs[], int state[], int ELEMENTS)
{
  unsigned int ynl; int ynh;
  ynl = (1<<23); ynh = 0;
  for(int j=ELEMENTS-1; j!=0; j--) {
      state[j] = state[j-1];
      {ynh, ynl} = macs(coeffs[j], state[j], ynh, ynl);
  }
  state[0] = xn;
  {ynh, ynl} = macs(coeffs[0], xn, ynh, ynl);
  if (sext(ynh,24) == ynh) {
      ynh = (ynh << 8) | (((unsigned) ynl) >> 24);}
  else if (ynh < 0) { ynh = 0x80000000; }
  else { ynh = 0x7fffffff; }
  return ynh;
}
```

XC Program (FIR Filter) w/Energy Specification [HIP3ES'15, TPLP'18]

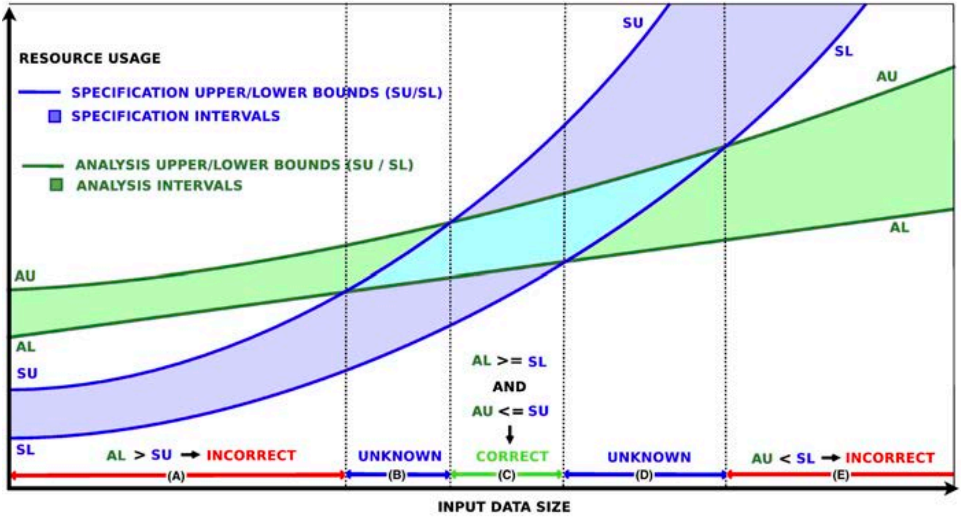
```
#pragma check fir(xn, coeffs, state, N) :
          (1 <= N) ==> (energy <= 416079189)

#pragma true fir(xn, coeffs, state, N) :
             (3347178*N + 13967829 <= energy &&
              energy <= 3347178*N + 14417829)

#pragma checked fir(xn, coeffs, state, N) :
             (1 <= N && N <= 120) ==> (energy <= 416079189)

#pragma false fir(xn, coeffs, state, N) :
             (121 <= N) ==> (energy <= 416079189)

int fir(int xn, int coeffs[], int state[], int ELEMENTS)
{
  unsigned int ynl; int ynh;
  ynl = (1<<23); ynh = 0;
  for(int j=ELEMENTS-1; j!=0; j--) {
      state[j] = state[j-1];
      {ynh, ynl} = macs(coeffs[j], state[j], ynh, ynl);
  }
  state[0] = xn;
  {ynh, ynl} = macs(coeffs[0], xn, ynh, ynl);
  if (sext(ynh,24) == ynh) {
      ynh = (ynh << 8) | (((unsigned) ynl) >> 24);}
  else if (ynh < 0) { ynh = 0x80000000; }
  else { ynh = 0x7fffffff; }
  return ynh;
}
```

CiaoPP Menu



```
         Select Menu Level:      naive
       Select Action Group:   check_assertions
  Select Resource Analysis:     res_plai
     Select Analysis Level:       llvm
             Plot Results:         yes
Report Non-Verified Assrts:     warning
    Select Output Language:      source
          {Current Saved Menu Configurations: []}
```

Resource Usage Verification



[TPLP'18, FOPARA'12, ICLP'10] [AADEBUG'97, LNCS'99, PBH00a, SCP'05, ICLP'09]

Outline

Part I The Ciao language

Part II The Ciao assertions model

Part III Using CiaoPP as a multi-language analyzer/verifier

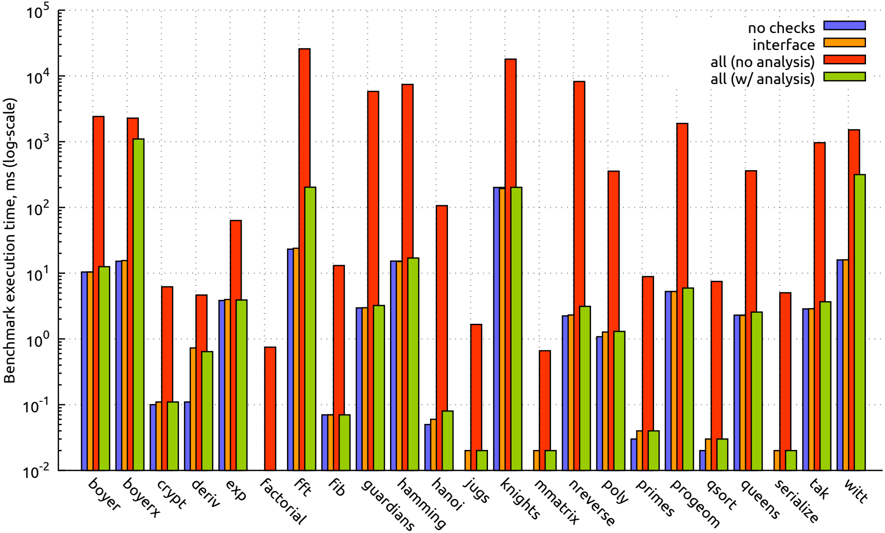
Part IV Conclusion and some recent work

Dynamic Checking of Assertions: Optimizations

- Checking complex properties (like shape of terms) may incur significant performance penalty and can often even change program complexity.
- Same problem observed in the context of gradual typing for functional programs.
- Contributed several solutions:
  - Simplify assertions (based on static analysis) to remove checks.
  - Cache checks at run time (exploit immutable data and monotonous updates).
  - Restrict symbol visibility (when possible) to enforce stronger data invariants.

[PPDP'18, PADL'18, PPDP'16, TPLP'15]

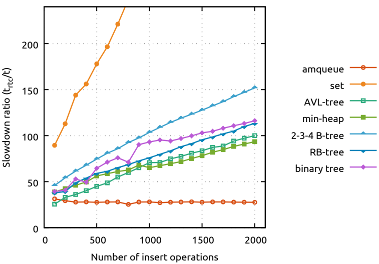
Dynamic Checking of Assertions: Optimizations



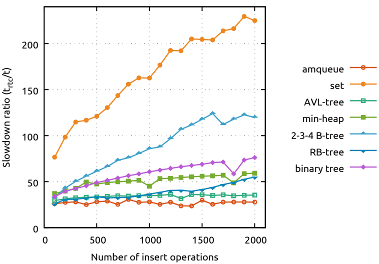
[PPDP'18]

Dynamic Checking of Assertions: Optimizations

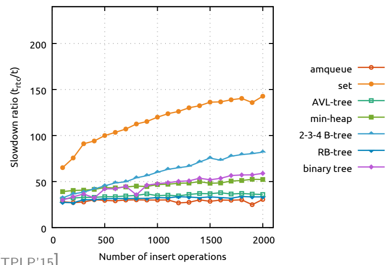
cache size = 32, check depth limit = 2



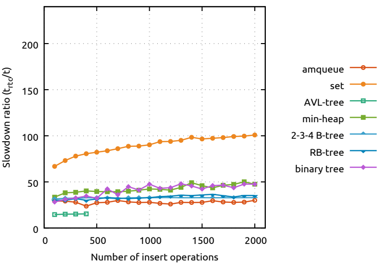
cache size = 64, check depth limit = 2



cache size = 128, check depth limit = 2



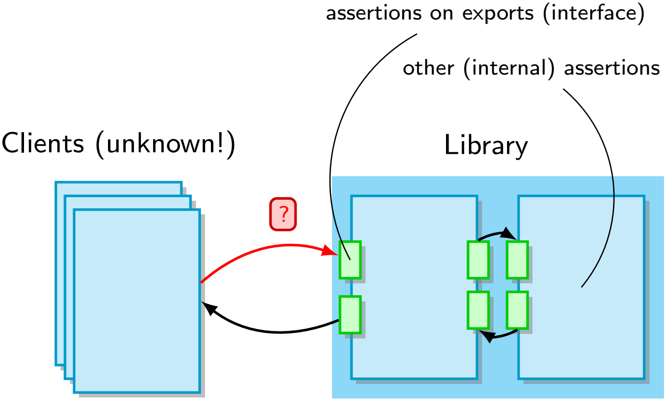
cache size = 256, check depth limit = 2



[TPLP'15]

Dynamic Checking of Assertions: Optimizations

Shallow run-time checking



assertions on exports (interface)

other (internal) assertions

Clients (unknown!)

Library

```
:- hide e/0. :- hide t/3. % hidden functors
tree(e).     tree(t(L,X,R)) :- tree(L), int(X), tree(R).
% (auto-computed) =======>
tree#(e).    tree#(t(_,_,_)). % shallow property
```

[PADL'18]

More expressive higher-order assertions [?]

More powerful description of meta-arguments – we introduce:

- anonymous assertions.
- predprops (predicate properties).

Can apply all the assertion to meta-arguments.

```
:- comparator(Cmp) {
  :- pred Cmp(Res,M,N) : num(M), num(N)
                      => between(-1,1,Res). }.

:- pred my_min(X,Y,Cmp,Min) : comparator(Cmp).

my_min(X,Y,P,Min) :-
    P(R,X,Y)●, (R < 0 -> Min = X ; Min = Y).

test_min :- my_min(4,2,compA,2).

compI( 0,A,B) :- A = B.    compA(=,A,B) :- A = B.
compI(-1,A,B) :- A < B.    compA(<,A,B) :- A < B.
compI( 1,A,B) :- A > B.    compA(>,A,B) :- A > B.
```

[PPDP'14]

Semantic Search

Main idea:

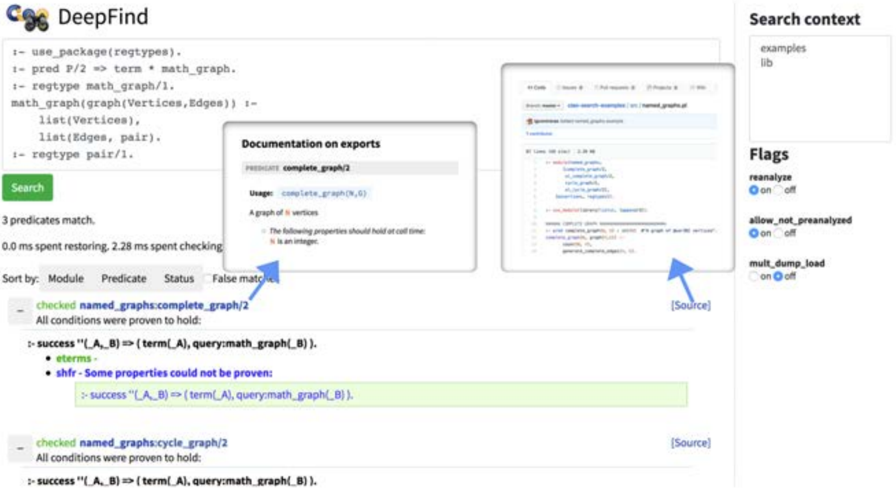
1. A **set of modules** is specified within which code is to be found.
2. A **static pre-analysis** made to infer semantic properties in one or more abstract domains.
   - E.g.: shapes/types, variable sharing, inst. modes, polyhedra, ...
3. User specifies **semantic properties** in a query, using a new kind of assertions that we call **query assertions**:
   - Example: `:- pred P(X,Y) : list(X) => sorted(Y).`
4. System looks for predicates that meet those properties, by comparing the query to the inferred information.

This method:

- Ensures that the code found **behaves correctly**.
- Reasons with **relations** between **(user-definable) properties** (implication, abstraction).
- Is **independent from the documentation**.

[ICLP'16]

Semantic Search – Tool snapshot



DeepFind

```
:- use_package(regtypes).
:- pred P/2 => term * math_graph.
:- regtype math_graph/1.
math_graph(graph(Vertices,Edges)) :-
    list(Vertices),
    list(Edges, pair).
:- regtype pair/1.
```

3 predicates match.

0.0 ms spent restoring, 2.28 ms spent checking

Documentation on exports

PREDICATE **complete_graph/2**

Usage: `complete_graph(N,G)`

A graph of N vertices.

- *The following properties should hold at call time:*
  N is an integer.

checked **named_graphs:complete_graph/2** [Source]
All conditions were proven to hold:

:- success "(_A,_B) => (term(_A), query:math_graph(_B)).

- **eterms** -
- **shfr - Some properties could not be proven:**

:- success "(_A,_B) => (term(_A), query:math_graph(_B)).

checked **named_graphs:cycle_graph/2** [Source]
All conditions were proven to hold:

:- success "(_A,_B) => (term(_A), query:math_graph(_B)).

Search context

examples
lib

Flags

reanalyze
● on ○ off

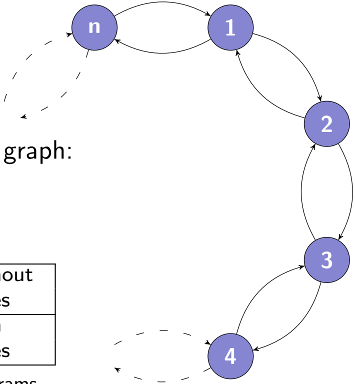
allow_not_preanalyzed
● on ○ off

mult_dump_load
○ on ● off

[ICLP'16]

Tabling and TCLP

- **Tabling**: Detect loops, avoid recomputation.
- **TCLP**: Combine tabling with CLP (Reduce search).



- Example: Bounded-length traversal of cyclic graph: Can I go from **1** to **n** in less than **k** steps?

	LP	Tabling	CLP	TCLP	
Right rec.	✓	✓	✓	✓	Without cycles
Left rec.	×	✓	×	✓	
Right rec.	×	×	✓	✓	With cycles
Left rec.	×	×	×	✓	

Termination properties of equivalent, simple, logically correct programs.

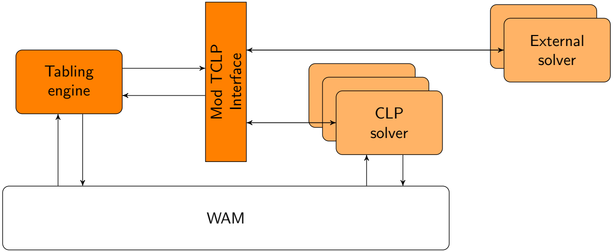
[PPDP'16, PADL'13, FLOPS'12, ICLP'10, ICLP'09, PADL'09, PADL'08]

Modular TCLP

- Different problems require different constraint domains / solvers.
- Connecting constraint solvers and tabling not straightforward.

Ciao provides

- A **simple** interface to perform this connection (Modular TCLP).
- An **answer management strategy** to discard / remove repeated answers.
- The TCLP interfaces for **several constraint domains**, e.g., CLP(Q/R).



Top-down Answer Set Programming – s(CASP)

- Is a top down Constraint Answer Set Programming interpreter.
- Avoids the grounding phase (range of constrained variables may be infinite).
- Each answer provides the mgu of a successful derivation, its justification tree, and the relevant (partial) stable model.
- Retains variables and constraints during the execution and in the model.

s(CASP)

=

ASP

+

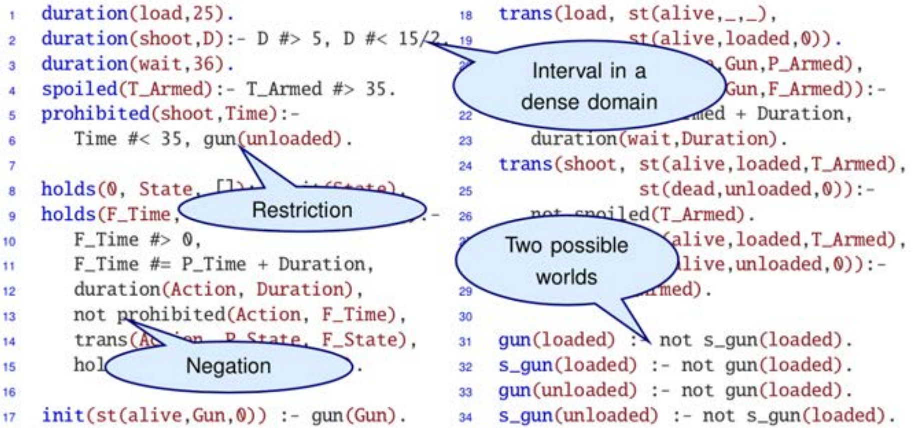
Constraints

-

Grounding

[ICLP'18]

Top-down Answer Set Programming - s(CASP) - Example



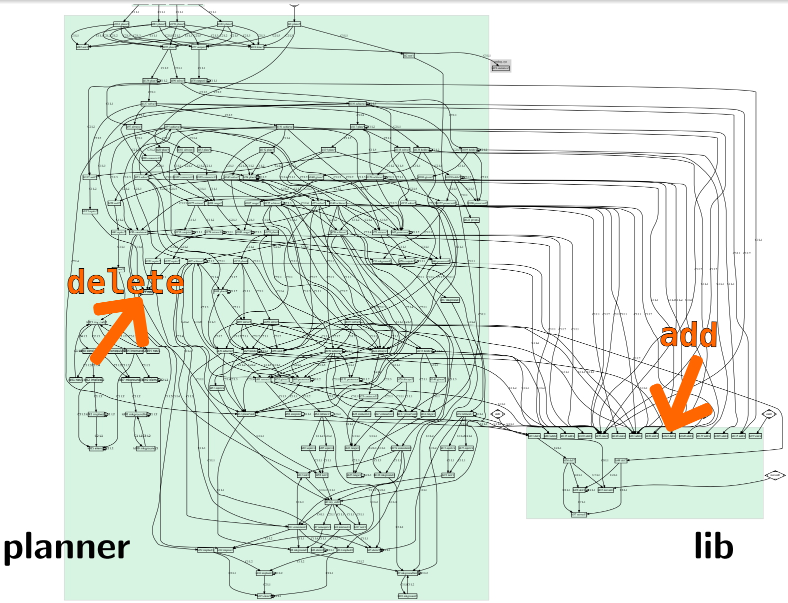
```
duration(load,25).
duration(shoot,D):- D #> 5, D #< 15/2.
duration(wait,36).
spoiled(T_Armed):- T_Armed #> 35.
prohibited(shoot,Time):-
    Time #< 35, gun(unloaded).

holds(0, State, []):-     it(State).
holds(F_Time,                    :-
    F_Time #> 0,
    F_Time #= P_Time + Duration,
    duration(Action, Duration),
    not prohibited(Action, F_Time),
    trans(Action, P_State, F_State),
    hol                    .

init(st(alive,Gun,0)) :- gun(Gun).
```

```
trans(load, st(alive,_,_),
             st(alive,loaded,0)).
                  ,Gun,P_Armed),
                  Gun,F_Armed)):-
                 ed + Duration,
    duration(wait,Duration).
trans(shoot, st(alive,loaded,T_Armed),
              st(dead,unloaded,0)):-
    not spoiled(T_Armed).
               alive,loaded,T_Armed),
               live,unloaded,0)):-
              med).

gun(loaded) :- not s_gun(loaded).
s_gun(loaded) :- not gun(loaded).
gun(unloaded) :- not gun(loaded).
s_gun(unloaded) :- not s_gun(loaded).
```

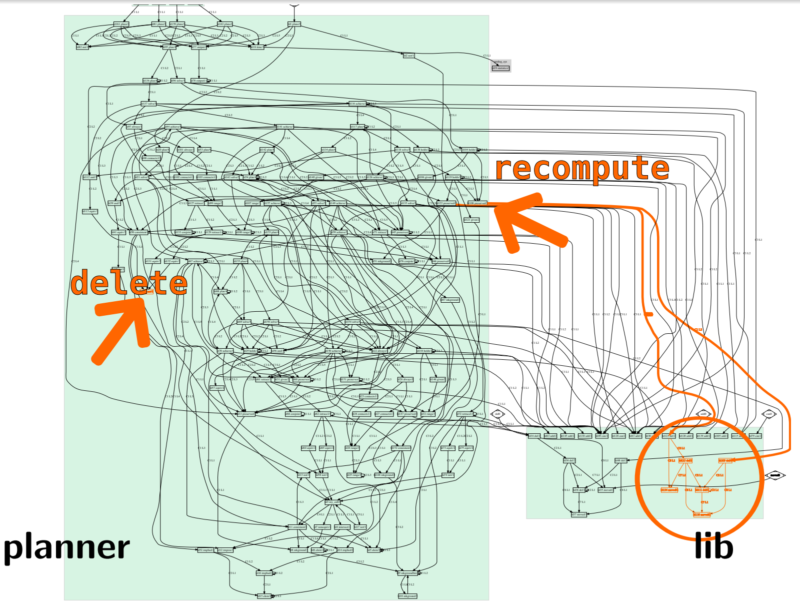
s(CASP) code for the extended and updated Yale Shooting problem.

[ICLP'18]

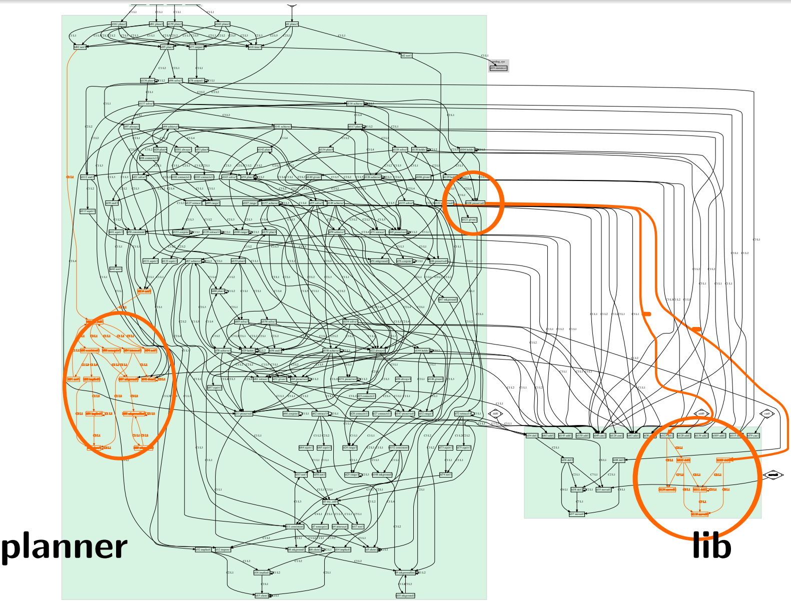
Modular, incremental, context-sensitive analysis



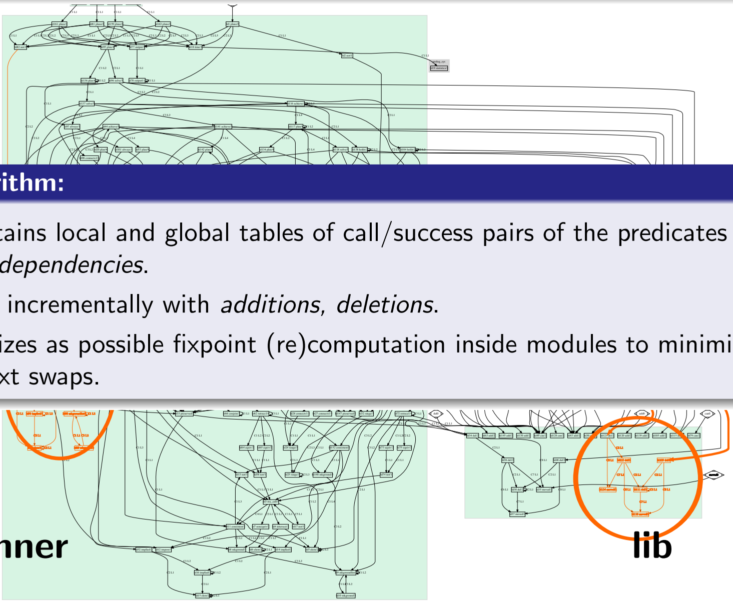
Modular, incremental, context-sensitive analysis



Modular, incremental, context-sensitive analysis



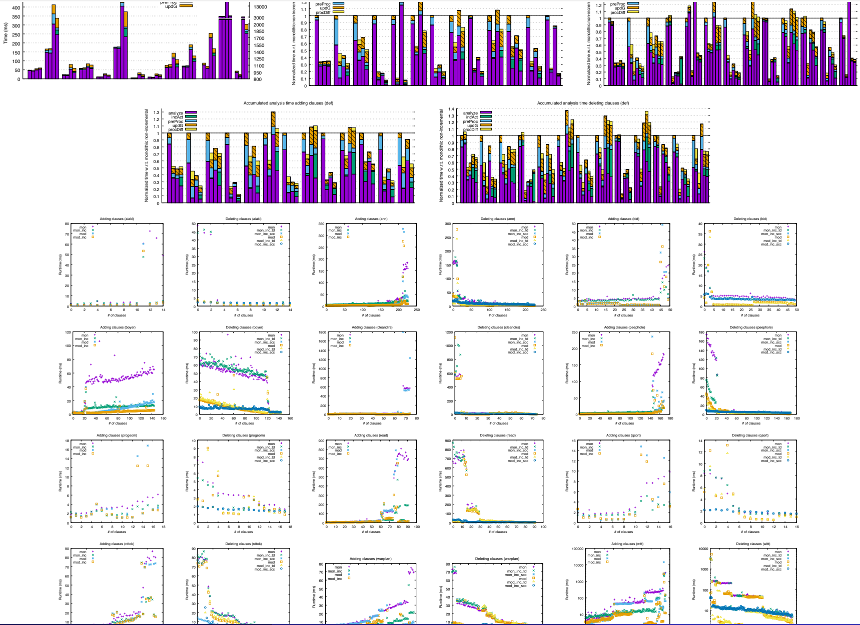
Modular, incremental, context-sensitive analysis



The algorithm:

- Maintains local and global tables of call/success pairs of the predicates *and their dependencies*.
- Deals incrementally with *additions, deletions*.
- Localizes as possible fixpoint (re)computation inside modules to minimize context swaps.

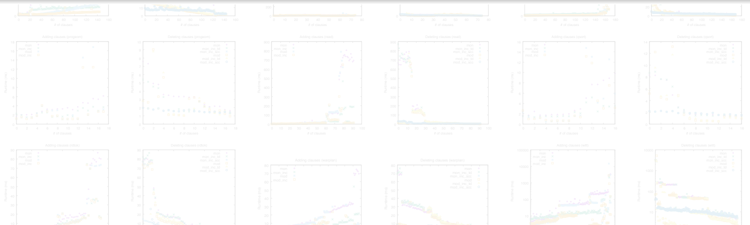
Experimental results



Experimental results

To take home:

- *Modular Incremental analysis works!* – Up to 60× speedup.
- *Modular analysis* from scratch is *improved* (up to 9×).
- Keeping structures for incrementality produces *small overhead*.
- Using the analyzer *interactively* becomes quite feasible, even for complex abstract domains.



Other features and recent developments

- Other interesting Ciao features:
  - *Modular* [LOPSTR'11] and *reversible* [PADL'13] extensions.
  - Compilation to Javascript [ICLP'12].
  - Test generation from properties.
- Other applications of CiaoPP:
  - Java bytecode analysis / verification [VMCAI'08, FTfJP'07, Bytecode'07, Bytecode'09]
  - MiniZinc, Services [Computing'13, MZN'11, ICSOC'11, ICSOC'10]
- Other related work:
  - Shapes, Sharing, and Parallelization in Imperative Languages [ISMM'09, LCPC'08, CC'08, PASTE'08, PASTE'07]

Some other recent work:

- Static performance guarantees for programs with run-time checks [PPDP'18].
- Guiding the analyzer with assertions [LOPSTR-Informal'18].
- Combining the incremental and the modular fixpoints [ICLP'18].

Why is the system called "Ciao"?

- It stems originally from an acronym:
  - *CIAO*: **C**onstraint Programming with **I**ndependent **A**nd + **O**r parallelism.
- But the name also represents the *spirit* of the system:
  - Ciao is an interesting word that means *both Hello* and *Goodbye*.
  - "Ciao Prolog:"
    - is aimed at introducing programmers to Prolog and LP/CLP
      – the "Hello Prolog" part,
    - but it also represents really a new-generation programming language and environment (with FP, HO, assertions, global analysis, objects, ...)
      – the "Goodbye Prolog" part.

http://www.ciao-lang.org

- Ciao, CiaoPP, LPdoc, etc.
- Documentation.
- Mailing lists.
- **Bundles**.
- etc.

Around 1,3 million lines of Ciao/Prolog code (+166K of C, 67K of Java, ...).

Mostly **LGPL** (some packages have some variations).

Continuous integration, etc.

Also on GitHub (including many **bundles**):

https://github.com/ciao-lang

Playground:

http://play.ciao-lang.org

The Current Ciao/CiaoPP Team

Manuel Carro

Manuel Hermenegildo

Pedro López-García

José-Francisco Morales

Joaquín Arias

Ignacio Casso

Isabel García-Contreras

Maximiliano Klemen

IMDEA Software Institute, T.U. Madrid (UPM).
And previously at: U. T. Austin, MCC, U. of New Mexico.

- Other main contributors to Ciao and CiaoPP (incomplete):

Germán Puebla
Mario Méndez-Lojo
Claudio Vaucheret
Pawel Pietrzak
Umer Liqat
Amadeo Casas
Remy Haemmerlé
Christian Holzbaur

Nai-wei Lin
Edison Mera
Saumya Debray
Claudio Ochoa
Nataliia Stulova
Daniel Cabeza
David Trallero
Kim Marriott

Jorge Navas
Francisco Bueno
Jesús Correas
John Gallagher
José M. Gómez
Pablo Chico
Gopal Gupta
Enrico Pontelli

Alejandro Serrano
María G. de la Banda
Elvira Albert
Peter Stuckey
Kalyan Muthukumar
Samir Genaim
Michael Codish
Ángel Pineda

Milestones / tools / timeline (partial)

'83-90	Parallel abstract machines: task stealing, micro tasks, (cactus) stack sets → motivation: auto-parallelization.
'88	**MA3 analyzer**: memo tables (cf. OLDT resolution), practicality established.
'89	**PLAI analyzer**: accelerated "top-down" fixpoint, abstract domains as plugins. Sharing, side-effect analysis → automatic parallelization / real speedups (on shmem).
90's	Incremental analysis, concurrency (dynamic scheduling), automatic domain combinations, scalability, auto-parallelization, extension to constraints.
'93	**GraCoS** (Granularity Control System): fully automatic cost analysis (upper bounds), and automatic parallelization with task granularity control (with optimizations).
mid 90's	**The Ciao/CiaoPP model: Library-based language features,** **Integrated verification/debugging/optimization w/assertions.**
'91-'06	Combined **abstract interpretation and partial evaluation**.
late 90's	**Lower bounds** cost analysis, **divide-and-conquer**. No-fail (no exceptions), determinacy.
'01	**Verification of cost properties**, additional resources, ...
'01-05	Modularity/scalability. Diagnosis (locating origin of assrt. violations). New shape/type domains, widenings. Polyhedra, convex hulls.
'03	**Abstraction carrying code**, reduced certificates.
'04-'07	Verification/debugging/optimization of **user-defined resources**.
'05	**Multi-language support using CLP (CHC) as IR**: Java, C# (shapes, resources, ...).
'06-'08	**Probabilistic** cost, verification of execution **time**, **energy** (Java), heap models, ...
'12-18	(X)C program energy analysis/verification, ISA-level energy models.
'13-18	**Cost analysis as Abstract Interpretation** Sized shapes. LLVM. **Static Profiling**.
'15-17	Optimized dynamic property checking (run-time tests).
'16	Sematic code search, Improved CLP+Tabling.
'18	Top-down **ASP**, fixpoint guidance, analysis scalability, rt check cost assurances, ...

Selected Bibliography: the Ciao Language and its Implementation

All papers available on line at: http://cliplab.org/clippubsbyyear
and http://cliplab.org/clippubsbytopic

Ciao overall language design and philosophy, extensibility

[TPLP'12] M. V. Hermenegildo, F. Bueno, M. Carro, P. López, E. Mera, and J.F. Morales, and G. Puebla.
An Overview of Ciao and its Design Philosophy.
Theory and Practice of Logic Programming 12, 1–2. 2012, Cambridge University Press.
http://arxiv.org/abs/1102.5497

[PADL'13] Z. Drey, J. F. Morales, M. V. Hermenegildo, and M. Carro.
Reversible Language Extensions and their Application in Debugging.
In Kostis Sagonas, editor, *Practical Aspects of Declarative Languages (PADL'13)*, volume 7752 of *LNCS*. Springer, January 2013.

[LOPSTR'11] J. F. Morales, M. V. Hermenegildo, and R. Haemmerlé.
Modular Extensions for Modular (Logic) Languages.
In *Proceedings of the 21th International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR'11)*, volume 7225 of *LNCS*, pages 139–154, Odense, Denmark, July 2011. Springer.

[DCG-PhD'04] D. Cabeza.
An Extensible, Global Analysis Friendly Logic Programming System.
PhD thesis, Universidad Politécnica de Madrid (UPM), August 2004.

[NovaSci'99] M. Hermenegildo, F. Bueno, D. Cabeza, M. Carro, M. García de la Banda, P. López-García, and G. Puebla.
The Ciao Multi-Dialect Compiler and System: An Experimentation Workbench for Future (C)LP Systems.
In *Parallelism and Implementation of Logic and Constraint Logic Programming*, pages 65–85. Nova Science, Commack, NY, USA, April 1999.

[ILPS'95] M. Hermenegildo, F. Bueno, M. García de la Banda, and G. Puebla.
The Ciao Multi-Dialect Compiler and System: An Experimentation Workbench for Future (C)LP Systems.
In *Proceedings of the ILPS'95 Workshop on Visions for the Future of Logic Programming*, Portland, Oregon, USA, December 1995.
Available from http://www.cliplab.org/.

[PPCP'94] M. Hermenegildo and The CLIP Group.
Some Methodological Issues in the Design of Ciao - A Generic, Parallel, Concurrent Constraint System.
In *Principles and Practice of Constraint Programming*, number 874 in LNCS, pages 123–133. Springer-Verlag, May 1994.

Reference manual (latest version)

[CiaoManual'18] F. Bueno, M. Carro, M. Hermenegildo, P. López-García, and J.F. Morales (Eds.).
The Ciao System. Reference Manual (v1.18).
August 2018. Available at http://www.ciao-lang.org.

Module system and compiler

[CL'00] D. Cabeza and M. V. Hermenegildo.
A New Module System for Prolog.
In *International Conference on Computational Logic, CL2000*, number 1861 in LNAI, pages 131–148. Springer-Verlag, July 2000.

[ENTCS'00] D. Cabeza and M. V. Hermenegildo.
The Ciao Modular, Standalone Compiler and Its Generic Program Processing Library.
In *Special Issue on Parallelism and Implementation of (C)LP Systems*, volume 30(3) of *Electronic Notes in Theoretical Computer Science*. Elsevier - North Holland, March 2000.

Auto-documenter

[CL2000] M. Hermenegildo.
A Documentation Generator for (C)LP Systems.
In *Int'l. Conf. on Computational Logic, CL2000*, number 1861 in LNAI, pages 1345–1361. Springer-Verlag, July 2000.

Functions, higher order, lazyness

[PPDP'14] N. Stulova, J. F. Morales, and M. V. Hermenegildo.
Assertion-based Debugging of Higher-Order (C)LP Programs.
In *16th Int'l. ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming (PPDP'14)*, pages 225–235. ACM Press, September 2014.

[FLOPS'06] A. Casas, D. Cabeza, and M. Hermenegildo.
A Syntactic Approach to Combining Functional Notation, Lazy Evaluation and Higher-Order in LP Systems.
In *Eighth International Symposium on Functional and Logic Programming (FLOPS'06)*, Fuji Susono (Japan), April 2006.

[ASIAN'04] D. Cabeza, M. Hermenegildo, and J. Lipton.
Hiord: A Type-Free Higher-Order Logic Programming Language with Predicate Abstraction.
In *Ninth Asian Computing Science Conference (ASIAN'04)*, number 3321 in LNCS, pages 93–108. Springer-Verlag, December 2004.

Top-down Answer Set Programming

[ICLP'18] J. Arias, M. Carro, E. Salazar, K. Marple, and G. Gupta.
Constraint Answer Set Programming without Grounding.
Theory and Practice of Logic Programming, 34th Int'l. Conference on Logic Programming (ICLP'18) Special Issue, 2018.

Tabling

[PPDP'16] J. Arias and M. Carro.
Description and Evaluation of a Generic Design to Integrate CLP and Tabled Execution.
In *18th Int'l. ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming (PPDP'16)*, pages 10–23. ACM Press, September 2016.

[PADL'13] P. Chico de Guzmán, M. Carro, and M. V. Hermenegildo.
Supporting Pruning in Tabled LP.
In *Practical Aspects of Declarative Languages (PADL'13)*, LNCS, Springer, January 2013.

[FLOPS'12] P. Chico de Guzmán, M. Carro, M. V. Hermenegildo, and P. Stuckey.
A General Implementation Framework for Tabled CLP.
In *FLOPS'12*, number 7294 in LNCS, pages 104–119. Springer Verlag, May 2012.

[ICLP'10] P. Chico de Guzmán, M. Carro, and David S. Warren.
Swapping Evaluation: A Memory-Scalable Solution for Answer-On-Demand Tabling.
Theory and Practice of Logic Programming, 26th Int'l. Conference on Logic Programming (ICLP'10) Special Issue, 10 (4–6):401–416, July 2010.

[ICLP'09] P. Chico de Guzmán, M. Carro, and M. V. Hermenegildo.
A Tabling Implementation Based on Variables with Multiple Bindings.
In *International Conference on Logic Programming (ICLP 2009)*, number 5649 in LNCS, pages 190–204. Springer-Verlag, July 2009.

[PADL'09] Pablo Chico de Guzmán Huerta, Manuel Carro, Manuel Hermenegildo.
Towards a Complete Scheme for Tabled Execution Based on Program Transformation.
11th International Symposium on Practical Aspects of Declarative Languages (PADL'09), LNCS, Num. 5418, pages 224-238, Springer-Verlag, January 2009.

[PADL'08] P. Chico de Guzmán, M. Carro, M. Hermenegildo, Claudio Silva, Ricardo Rocha.
An Improved Continuation Call-Based Implementation of Tabling.
10th International Symposium on Practical Aspects of Declarative Languages (PADL'08), LNCS, Vol. 4902, pages 198-213, Springer-Verlag, January 2008.

Concurrency and Distribution, Objects

[EuroPar'04] M. Hermenegildo, E. Albert, P. López-García, and G. Puebla.
Some Techniques for Automated, Resource-Aware Distributed and Mobile Computing in a Multi-Paradigm Programming System.
In *Proc. of EURO–PAR 2004*, number 3149 in LNCS, pages 21–37. Springer-Verlag, August 2004.

[ICLP'02] A. Pineda and F. Bueno.
The O'Ciao Approach to Object Oriented Logic Programming.
In *Colloquium on Implementation of Constraint and LOgic Programming Systems (ICLP associated workshop)*, Copenhagen, July 2002.

[CICLOPS'02] M. Carro and M. Hermenegildo.
A simple approach to distributed objects in prolog.
In *Colloquium on Implementation of Constraint and LOgic Programming Systems (ICLP associated workshop)*, Copenhagen, July 2002.

[ICLP'99] M. Carro and M. Hermenegildo.
Concurrency in Prolog Using Threads and a Shared Database.
In *1999 International Conference on Logic Programming*, pages 320–334. MIT Press, Cambridge, MA, USA, November 1999.

[ICLP'95] M. Hermenegildo, D. Cabeza, and M. Carro.
Using Attributed Variables in the Implementation of Concurrent and Parallel LP Systems.
In *Proc. of the Twelfth International Conference on Logic Programming*, pages 631–645. MIT Press, June 1995.

[ProDe'96] D. Cabeza and M. V. Hermenegildo.
Implementing Distributed Concurrent Constraint Execution in the CIAO System.
In *Proc. of the AGP'96 Joint conference on Declarative Programming*, pages 67–78, San Sebastian, Spain, July 1996. U. of the Basque Country.
Available from `http://www.cliplab.org/`.

[CLNet'95] D. Cabeza and M. V. Hermenegildo.
Distributed Concurrent Constraint Execution in the CIAO System.
In *Proc. of the 1995 COMPULOG-NET Workshop on Parallelism and Implementation Technologies*, Utrecht, NL, September 1995. U. Utrecht / T.U. Madrid.
Available from `http://www.cliplab.org/`.

[PPCP'94] U. Montanari, F. Rossi, F. Bueno, M. García de la Banda, and M. Hermenegildo.
Towards a Concurrent Semantics-based Analysis of CC and CLP.
In *Principles and Practice of Constraint Programming*, number 874 in LNCS, pages 151–161. Springer-Verlag, May 1994.

[ALP'94] F. Bueno, M. V. Hermenegildo, U. Montanari, and F. Rossi.
From Eventual to Atomic and Locally Atomic CC Programs: A Concurrent Semantics.
In *Fourth International Conference on Algebraic and Logic Programming*, number 850 in LNCS, pages 114–132. Springer-Verlag, September 1994.

Finite domains

[CICLOPS'12] E.J. Gallego-Arias, R. Haemmerlé, M. V. Hermenegildo, and J.F. Morales.
The Ciao CLP(FD) Library: A Modular CLP Extension for Prolog.
In *12th International Colloquium on Implementation of Constraint and LOgic Programming Systems (CICLOPS 2012)*, September 2012.

Other control rules

[TR'95] F. Bueno, S. K. Debray, M. García de la Banda, and M. V. Hermenegildo.
Transformation-based Implementation and Optimization of Programs Exploiting the Basic Andorra Model.
Technical Report CLIP11/95.0, ACCLAIM Deliverable D3.3/4-A1, Facultad de Informática, UPM, May 1995.

Abstract machine and low-level optimization

[TPLP'16] J.F. Morales, M. Carro, and M. V. Hermenegildo
Description and Optimization of Abstract Machines in a Dialect of Prolog,
Theory and Practice of Logic Programming, 16-1, pp. 1–58, January 2016. Cambridge University Press.

[LOPSTR'15] J.F. Morales and M. V. Hermenegildo.
Pre-Indexed Terms for Prolog.
In *Proc. 24th Int'l. Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR'14)*, volume 8981 of *LNCS*, pages 317–331. Springer, 2015.

[ICLP'12] J. F. Morales, R. Haemmerlé, M. Carro, and M. V. Hermenegildo.
Lightweight compilation of (C)LP to JavaScript.
Theory and Practice of Logic Programming, 28th Int'l. Conference on Logic Programming (ICLP'12) Special Issue, 12(4-5):755–773, 2012.

[PPDP'08] J. Morales, M. Carro, M. Hermenegildo.
Comparing Tag Scheme Variations Using an Abstract Machine Generator.
10th Int'l. ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming (PPDP'08), pages 32-43, ACM Press, July 2008.

[LOPSTR'06] J.F. Morales, M. Carro, and M. Hermenegildo.
Towards Description and Optimization of Abstract Machines in an Extension of Prolog.
In *Logic-Based Program Synthesis and Transformation (LOPSTR'06)*, number 4407 in LNCS, pages 77–93, July 2007.

[CASES'06] M. Carro, J. Morales, H.L. Muller, G. Puebla, and M. Hermenegildo.
High-Level Languages for Small Devices: A Case Study.
In Krisztian Flautner and Taewhan Kim, editors, *Compilers, Architecture, and Synthesis for Embedded Systems (CASES)*, pages 271–281. ACM Press / Sheridan, October 2006.

[ICLP'05] J. Morales, M. Carro, G. Puebla, and M. Hermenegildo.
A generator of efficient abstract machine implementations and its application to emulator minimization.
In P. Meseguer and J. Larrosa, editors, *International Conference on Logic Programming*, number 3668 in LNCS, pages 21–36. Springer Verlag, October 2005.

[PADL'04] J. Morales, M. Carro, and M. Hermenegildo.
Improving the Compilation of Prolog to C Using Moded Types and Determinism Information.
In *Proceedings of the Sixth International Symposium on Practical Aspects of Declarative Languages (PADL)*, number 3057 in LNCS, pages 86–103, Heidelberg, Germany, June 2004. Springer-Verlag.

Abstract machine: parallelism and concurrency support

[PADL'12] P. Chico de Guzmán, A. Casas, M. Carro, and M. V. Hermenegildo.
A Segment-Swapping Approach for Executing Trapped Computations.
In Neng-Fa Zhou and Claudio Russo, editors, *PADL'12*, volume 7149 of *LNCS*, pages 138–152. Springer Verlag, January 2012.

[ICLP'11] P. Chico de Guzmán, A. Casas, M. Carro, and M. V. Hermenegildo.
Parallel Backtracking with Answer Memoing for Independent And-Parallelism.
Theory and Practice of Logic Programming, 27th Int'l. Conference on Logic Programming (ICLP'11) Special Issue, 11(4–5):555–574, July 2011.

[ICLP'08] A. Casas, M. Carro, M. Hermenegildo.
A High-Level Implementation of Non-Deterministic, Unrestricted, Independent And-Parallelism.
24th International Conference on Logic Programming (ICLP'08), LNCS, Vol. 5366, pages 651-666, Springer-Verlag, December 2008.

[EuroPar'96] K. Shen and M. Hermenegildo.
Flexible Scheduling for Non-Deterministic, And-parallel Execution of Logic Programs.
In *Proceedings of EuroPar'96*, number 1124 in LNCS, pages 635–640. Springer-Verlag, August 1996.

[NGC'91] M. Hermenegildo and K. Greene.
The &-Prolog System: Exploiting Independent And-Parallelism.
New Generation Computing, 9(3,4):233–257, 1991.

[NGC'89] M. Hermenegildo and E. Tick.
Memory Referencing Characteristics and Caching Performance of AND-Parallel Prolog on Shared-Memory Architectures.
New Generation Computing, 7(1):37–58, October 1989.

[ICPP'88] M. Hermenegildo and E. Tick.
Memory Performance of AND-Parallel Prolog on Shared-Memory Architectures.
In *Proceedings of the 17th International Conference on Parallel Processing*, pages 17–22. IEEE, August 1988.

[ICLP'87] M. Hermenegildo.
Relating Goal Scheduling, Precedence, and Memory Management in AND-Parallel Execution of Logic Programs.
In *Fourth International Conference on Logic Programming*, pages 556–575. University of Melbourne, MIT Press, May 1987.

[ICLP'86] M. Hermenegildo.
An Abstract Machine for Restricted AND-parallel Execution of Logic Programs.
In *Third International Conference on Logic Programming*, number 225 in Lecture Notes in Computer Science, pages 25–40. Imperial College, Springer-Verlag, July 1986.

Selected Bibliography: the Ciao Assertion Model and CiaoPP

The Ciao Assertions Model

[ICLP'09] E. Mera, P. López-García, and M. Hermenegildo.
Integrating Software Testing and Run-Time Checking in an Assertion Verification Framework.
In *25th Intl. Conference on Logic Programming (ICLP'09)*, number 5649 in LNCS, pages 281–295. Springer-Verlag, July 2009.

[LPAR'06] P. Pietrzak, J. Correas, G. Puebla, and M. Hermenegildo.
Context-Sensitive Multivariant Assertion Checking in Modular Programs.
In *LPAR'06*, number 4246 in LNCS, pages 392–406. Springer-Verlag, November 2006.

[SCP'05] M. Hermenegildo, G. Puebla, F. Bueno, and P. López-García.
Integrated Program Debugging, Verification, and Optimization Using Abstract Interpretation
Science of Computer Programming, 58(1–2), 2005.

[SAS'03] M. Hermenegildo, G. Puebla, F. Bueno, and P. López-García.
Program Development Using Abstract Interpretation (and The Ciao System Preprocessor).
In *10th International Static Analysis Symposium (SAS'03)*, number 2694 in LNCS, pages 127–152. Springer-Verlag, June 2003.

[PBH00a] G. Puebla, F. Bueno, and M. Hermenegildo.
A Generic Preprocessor for Program Validation and Debugging.
In P. Deransart, M. Hermenegildo, and J. Maluszynski, editors, *Analysis and Visualization Tools for Constraint Programming*, number 1870 in LNCS, pages 63–107. Springer-Verlag, Sept. 2000.

[LOPSTR'99] G. Puebla, F. Bueno, and M. Hermenegildo.
Combined Static and Dynamic Assertion-Based Debugging of Constraint Logic Programs.
In *Logic-based Program Synthesis and Transformation (LOPSTR'99)*, number 1817 in LNCS, pages 273–292. Springer-Verlag, March 2000.

[ICLP'99] M. V. Hermenegildo, F. Bueno, G. Puebla, P. López-García
Program Analysis, Debugging and Optimization Using the Ciao System Preprocessor
In *Int'l. Conference on Logic Programming*, November 1999, pages 52–66, MIT Press.

[LNCS'99] M. Hermenegildo, G. Puebla, and F. Bueno.
Using Global Analysis, Partial Specifications, and an Extensible Assertion Language for Program Validation and Debugging.
In K. R. Apt, V. Marek, M. Truszczynski, and D. S. Warren, editors, *The Logic Programming Paradigm: a 25–Year Perspective*, pages 161–192. Springer-Verlag, July 1999.

[AADEBUG'97] F. Bueno, P. Deransart, W. Drabent, G. Ferrand, M. Hermenegildo, J. Maluszynski, and G. Puebla.
On the Role of Semantic Approximations in Validation and Diagnosis of Constraint Logic Programs.
In *Proc. of the 3rd. Int'l Workshop on Automated Debugging–AADEBUG'97*, pages 155–170, Linköping, Sweden, May 1997. U. of Linköping Press.

Application to Resources

[TPLP'18] P. Lopez-Garcia, L. Darmawan, M. Klemen, U. Liqat, F. Bueno, and M. V. Hermenegildo.
Interval-based Resource Usage Verification by Translation into Horn Clauses and an Application to Energy Consumption.
Theory and Practice of Logic Programming, Special Issue on Verification, Cambridge U. Press, March 2018.

[FOPARA'12] P. Lopez-Garcia, L. Darmawan, F. Bueno, and M. Hermenegildo
Interval-Based Resource Usage Verification: Formalization and Prototype
In *Foundational and Practical Aspects of Resource Analysis. Second Int'l. Workshop FOPARA 2011, Revised Selected Papers*. Lecture Notes in Computer Science, 2012, 7177, 54–71, Springer.

[ICLP'10] P. López-García, L. Darmawan, and F. Bueno.
A Framework for Verification and Debugging of Resource Usage Properties.
In *Technical Communications of the 26th ICLP. Leibniz Int'l. Proc. in Informatics (LIPIcs)*, Vol. 7, pages 104–113, Dagstuhl, Germany, July 2010.

The Assertion Language

[PPDP'14] N. Stulova, J. F. Morales, M. V. Hermenegildo.
Assertion-based Debugging of Higher-Order (C)LP Programs.
In *16th Int'l. ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming (PPDP'14)*, ACM Press, September 2014.

[ICLP'09] E. Mera, P. López-García, and M. Hermenegildo.
Integrating Software Testing and Run-Time Checking in an Assertion Verification Framework.
In *25th Intl. Conference on Logic Programming (ICLP'09)*, number 5649 in LNCS, pages 281–295. Springer-Verlag, July 2009.

[LNCS'00] G. Puebla, F. Bueno, and M. Hermenegildo.
An Assertion Language for Constraint Logic Programs.
In *Analysis and Visualization Tools for Constraint Programming*, number 1870 in LNCS, pages 23–61. Springer-Verlag, September 2000.

[ILPS-WS'97] G. Puebla, F. Bueno, and M. Hermenegildo.
An Assertion Language for Debugging of Constraint Logic Programs.
In *Proceedings of the ILPS'97 Workshop on Tools and Environments for (Constraint) Logic Programming*, October 1997.

[ESOP'96] F. Bueno, D. Cabeza, M. Hermenegildo, and G. Puebla.
Global Analysis of Standard Prolog Programs.
In *European Symposium on Programming*, number 1058 in LNCS, pages 108–124, Sweden, April 1996. Springer-Verlag.

Inferring/Reducing Run-time Assertion Checking Overhead

[PPDP'18] M. Klemen, N. Stulova, P. Lopez-Garcia, J. F. Morales, and M. V. Hermenegildo.
Static Performance Guarantees for Programs with Run-time Checks.
In *20th Int'l. ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming (PPDP'18)*. ACM Press, September 2018.

[PADL'18] N. Stulova, J. F. Morales, and M. V. Hermenegildo.
Exploiting Term Hiding to Reduce Run-time Checking Overhead.
20th International Symposium on Practical Aspects of Declarative Languages (PADL 2018), number 10702 in LNCS, pages 99–115. Springer-Verlag, January 2018.

[PPDP'16] N. Stulova, J. F. Morales, and M. V. Hermenegildo.
Reducing the Overhead of Assertion Run-time Checks via static analysis.
In *18th Int'l. ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming (PPDP'16)*, pages 90–103. ACM Press, September 2016.

[TPLP'15] N. Stulova, J. F. Morales, and M. V. Hermenegildo.
Practical Run-time Checking via Unobtrusive Property Caching.
Theory and Practice of Logic Programming, 31st Int'l. Conference on Logic Programming (ICLP'15) Special Issue, 15(04-05):726–741, September 2015.

Semantic Code Search

[ICLP'16] I. Garcia-Contreras, J. F. Morales, and M. V. Hermenegildo.
Semantic Code Browsing.
Theory and Practice of Logic Programming, 32nd Int'l. Conference on Logic Programming (ICLP'16) Special Issue, 16(5-6):721–737, October 2016.

Abstraction-Carrying Code

[TPLP'12] E. Albert, P. Arenas, G. Puebla, and M. Hermenegildo.
Certificate Size Reduction in Abstraction-Carrying Code.
Theory and Practice of Logic Programming, Cambridge U. Press, 2012.

[PPDP'05] M. Hermenegildo, E. Albert, P. López-García, and G. Puebla.
Abstraction Carrying Code and Resource-Awareness.
In *PPDP*. ACM Press, 2005.

[LPAR'04] E. Albert, G. Puebla, and M. Hermenegildo.
Abstraction-Carrying Code.
In *Proc. of LPAR'04*, volume 3452 of *LNAI*. Springer, 2005.

[COCV'04] E. Albert, G. Puebla, and M. Hermenegildo.
An Abstract Interpretation-based Approach to Mobile Code Safety.
In *Proc. of Compiler Optimization meets Compiler Verification (COCV'04)*, Electronic Notes in Theoretical Computer Science 132(1), pages 113–129. Elsevier - North Holland, April 2004.

Selected Bibliography:
Abstract Interpretation in CiaoPP

Basic Analysis Framework (Abstract Interpreter, Fixpoint)

[LOPSTR-Informal'18] I. Garcia-Contreras, J.F. Morales, , and M. V. Hermenegildo.
Multivariant Assertion-based Guidance in Abstract Interpretation.
In *Pre-proceedings of the 28th International Symposium on Logic-based Program Synthesis and Transformation (LOPSTR'18)*, September 2018.

[TOPLAS'00] M. Hermenegildo, G. Puebla, K. Marriott, and P. Stuckey.
Incremental Analysis of Constraint Logic Programs.
ACM Transactions on Programming Languages and Systems, 22(2):187–223, March 2000.

[JLP'97] M. Codish, M. Bruynooghe, M. García de la Banda, and M. V. Hermenegildo.
Exploiting Goal Independence in the Analysis of Logic Programs.
Journal of Logic Programming, 32(3):247–261, September 1997.

[SAS'96] G. Puebla and M. Hermenegildo.
Optimized Algorithms for the Incremental Analysis of Logic Programs.
In *Intl. Static Analysis Symposium (SAS 1996)*, number 1145 in LNCS, pages 270–284. Springer-Verlag, September 1996.

[TOPLAS'96] M. García de la Banda, M. V. Hermenegildo, M. Bruynooghe, V. Dumortier, G. Janssens, and W. Simoens.
Global Analysis of Constraint Logic Programs.
ACM Trans. on Programming Languages and Systems, 18(5):564–615, 1996.

[ESOP'96] F. Bueno, D. Cabeza, M. Hermenegildo, and G. Puebla.
Global Analysis of Standard Prolog Programs.
In *European Symposium on Programming*, number 1058 in LNCS, pages 108–124, Sweden, April 1996. Springer-Verlag.

[TOPLAS'95] M. Codish, A. Mulkers, M. Bruynooghe, M. García de la Banda, and M. Hermenegildo.
Improving Abstract Interpretations by Combining Domains.
ACM Transactions on Programming Languages and Systems, 17(1):28–44, January 1995.

[POPL'94] K. Marriott, M. García de la Banda, and M. Hermenegildo.
Analyzing Logic Programs with Dynamic Scheduling.
In *20th. Annual ACM Conf. on Principles of Programming Languages*, pages 240–254. ACM, January 1994.

[JLP'92] K. Muthukumar and M. Hermenegildo.
Compile-time Derivation of Variable Dependency Using Abstract Interpretation.
Journal of Logic Programming, 13(2/3):315–347, July 1992.

[NACLP'89] K. Muthukumar and M. Hermenegildo.
Determination of Variable Dependence Information at Compile-Time Through Abstract Interpretation.
In *1989 N. American Conf. on Logic Programming*, pages 166–189. MIT Press, October 1989.

[ICLP'88] R. Warren, M. Hermenegildo, and S. K. Debray
On the Practicality of Global Flow Analysis of Logic Programs
In *5th Intl. Conf. and Symp. on Logic Programming*, pp. 684–699, MIT Press, August 1988.

Scalability/Modularity

[ICLP'18] I. Garcia-Contreras, J. F. Morales, and M. V. Hermenegildo.
Towards Incremental and Modular Context-sensitive Analysis.
In *Technical Communications of the 34th International Conference on Logic Programming (ICLP 2018)*, OpenAccess Series in Informatics (OASIcs), July 2018.

[PEPM'08] P. Pietrzak, J. Correas, G. Puebla, and M. Hermenegildo.
A Practical Type Analysis for Verification of Modular Prolog Programs.
In *PEPM'08*, pages 61–70. ACM Press, January 2008.

[LOPSTR'01] F. Bueno, M. García de la Banda, M. Hermenegildo, K. Marriott, G. Puebla, and P. Stuckey.
A Model for Inter-module Analysis and Optimizing Compilation.
In *Logic-based Program Synthesis and Transformation*, number 2042 in LNCS, pages 86–102. Springer-Verlag, March 2001.

[ENTCS'00] G. Puebla and M. Hermenegildo.
Some Issues in Analysis and Specialization of Modular Ciao-Prolog Programs.
In *Special Issue on Optimization and Implementation of Declarative Programming Languages*, volume 30 of *Electronic Notes in Theoretical Computer Science*. Elsevier - North Holland, March 2000.

Abstract Domains: Sharing/Aliasing

[LCPC'08] M. Méndez-Lojo, O. Lhoták, and M. Hermenegildo.
Efficient Set Sharing using ZBDDs.
In *21st Int'l. WS on Languages and Compilers for Parallel Computing (LCPC'08)*, LNCS. Springer-Verlag, August 2008.

[ICLP'08] E. Trias, J. Navas, E. S. Ackley, S. Forrest, and M. V. Hermenegildo.
Negative Ternary Set-Sharing.
In *Int'l. Conference on Logic Programming, ICLP*, LNCS 5366, pages 301–316, Springer-Verlag, December 2008.

[PADL'06] J. Navas, F. Bueno, and M. Hermenegildo.
Efficient top-down set-sharing analysis using cliques.
In *Eight Intl. Symposium on Practical Aspects of Declarative Languages*, number 2819 in LNCS, pages 183–198. Springer-Verlag, January 2006.

[ICLP'91] K. Muthukumar and M. Hermenegildo.
Combined Determination of Sharing and Freeness of Program Variables Through Abstract Interpretation.
International Conference on Logic Programming (ICLP 1991), pages49–63, MIT Press, June 1991.

[NACLP'89] K. Muthukumar and M. Hermenegildo.
Determination of Variable Dependence Information at Compile-Time Through Abstract Interpretation.
In *1989 N. American Conf. on Logic Programming*, pages 166–189. MIT Press, October 1989.

Abstract Domains: Shape/Type Analysis

[ICLP'13] A. Serrano, P. López-Garcia, F. Bueno, and M. Hermenegildo.
Sized Type Analysis of Logic Programs (Technical Communication).
In *Theory and Practice of Logic Programming, 29th Int'l. Conference on Logic Programming (ICLP'13) Special Issue, On-line Supplement*, pages 1–14, Cambridge U. Press, August 2013.

[PEPM'08] P. Pietrzak, J. Correas, G. Puebla, and M. Hermenegildo.
A Practical Type Analysis for Verification of Modular Prolog Programs.
In *PEPM'08*, pages 61–70. ACM Press, January 2008.

[SAS'02] C. Vaucheret and F. Bueno.
More Precise yet Efficient Type Inference for Logic Programs.
In *Intl. Static Analysis Symposium*, volume 2477 of *Lecture Notes in Computer Science*, pages 102–116. Springer-Verlag, September 2002.

Abstract Domains: Non-failure, Determinacy

[NGC'10] P. López-García, F. Bueno, and M. Hermenegildo.
Automatic Inference of Determinacy and Mutual Exclusion for Logic Programs Using Mode and Type Information.
New Generation Computing, 28(2):117–206, 2010.

[LOPSTR'04] P. López-García, F. Bueno, and M. Hermenegildo.
Determinacy Analysis for Logic Programs Using Mode and Type Information.
In *Proceedings of the 14th Intl. Symposium on Logic-based Program Synthesis and Transformation (LOPSTR'04)*, number 3573 in LNCS, pages 19–35. Springer-Verlag, August 2005.

[FLOPS'04] F. Bueno, P. López-García, and M. Hermenegildo.
Multivariant Non-Failure Analysis via Standard Abstract Interpretation.
In *7th Intl. Symposium on Functional and Logic Programming (FLOPS 2004)*, number 2998 in LNCS, pages 100–116, Heidelberg, Germany, April 2004. Springer-Verlag.

[ICLP'97] S.K. Debray, P. López-García, and M. Hermenegildo.
Non-Failure Analysis for Logic Programs.
In *1997 Intl. Conference on Logic Programming*, pages 48–62, Cambridge, MA, June 1997. MIT Press, Cambridge, MA.

Abstract Domains: Analysis of Resources

[ICLP'16] P. Lopez-Garcia, M. Klemen, U. Liqat, and M.V. Hermenegildo.
A General Framework for Static Profiling of Parametric Resource Usage.
Theory and Practice of Logic Programming, 32nd Int'l. Conference on Logic Programming (ICLP'16) Special Issue, 16(5-6):849–865, October 2016.

[FLOPS'16] R. Haemmerlé, P. Lopez-Garcia, U. Liqat, M. Klemen, J. P. Gallagher, and M. V. Hermenegildo.
A Transformational Approach to Parametric Accumulated-cost Static Profiling.
In *FLOPS'16*, volume 9613 of *LNCS*, pages 163–180. Springer, 2016.

[TPLP'14] A. Serrano, P. López-Garcia, and M. Hermenegildo.
Resource Usage Analysis of Logic Programs via Abstract Interpretation Using Sized Types.
In *Theory and Practice of Logic Programming, 30th Int'l. Conference on Logic Programming (ICLP'14) Special Issue*, Vol. 14, Num. 4-5, pages 739-754, Cambridge U. Press, 2014.

[ICLP'13] A. Serrano, P. López-Garcia, F. Bueno, and M. Hermenegildo.
Sized Type Analysis of Logic Programs (Technical Communication).
In *Theory and Practice of Logic Programming, 29th Int'l. Conference on Logic Programming (ICLP'13) Special Issue, On-line Supplement,* pages 1–14, Cambridge U. Press, August 2013.

[PPDP'08] E. Mera, P. López-García, M. Carro, and M. Hermenegildo.
Towards Execution Time Estimation in Abstract Machine-Based Languages.
In *10th Int'l. ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming (PPDP'08)*, pages 174–184. ACM Press, July 2008.

[ICLP'07] J. Navas, E. Mera, P. López-García, and M. Hermenegildo.
User-Definable Resource Bounds Analysis for Logic Programs.
In *23rd International Conference on Logic Programming (ICLP'07)*, LNCS Vol. 4670. Springer, 2007. **ICLP 2017 10-year Test of Time Award.**

[PADL'07] E. Mera, P. López-García, G. Puebla, M. Carro, and M. Hermenegildo.
Combining Static Analysis and Profiling for Estimating Execution Times.
In *Ninth International Symposium on Practical Aspects of Declarative Languages*, number 4354 in LNCS, pages 140–154. Springer-Verlag, January 2007.

[CLEI'06] H. Soza, M. Carro, and P. López-García.
Probabilistic Cost Analysis of Logic Programs: A First Case Study.
In *XXXII Latin-American Conference on Informatics* (CLEI 2006), August 2006.

[ILPS'97] S. K. Debray, P. López-García, M. Hermenegildo, and N.-W. Lin.
Lower Bound Cost Estimation for Logic Programs.
In *1997 Intl. Logic Programming Symp.*, pp. 291–305. MIT Press, Cambridge, MA, October 1997.

[SAS'94] S.K. Debray, P. López-García, M. Hermenegildo, and N.-W. Lin.
Estimating the Computational Cost of Logic Programs.
In *Static Analysis Symposium, SAS'94*, number 864 in LNCS, pages 255–265, Sep 1994. Springer.

[PLDI'90] S. K. Debray, N.-W. Lin, and M. Hermenegildo.
Task Granularity Analysis in Logic Programs.
In *Proc. of the 1990 ACM Conf. on Programming Language Design and Implementation (PLDI)*, pages 174–188. ACM Press, June 1990.

Selected Bibliography: Program Optimization with CiaoPP

Partial evaluation

[TPLP'11] G. Puebla and E. Albert and M. V. Hermenegildo.
Efficient Local Unfolding with Ancestor Stacks.
Theory and Practice of Logic Programming, Cambridge U. Press January 2011.

[SAS'06] G. Puebla, E. Albert, and M. Hermenegildo.
Abstract Interpretation with Specialized Definitions.
In *The 13th Int'l. Static Analysis Symposium (SAS'06)*, number 4134 in LNCS, pages 107–126. Springer, August 2006.

[PPDP'06] G. Puebla and C. Ochoa.
Poly-Controlled Partial Evaluation.
In *Proc. of 8th ACM-SIGPLAN Int'l. Symposium on Principles and Practice of Declarative Programming (PPDP'06)*, pages 261–271. ACM Press, July 2006.

[LOPSTR'05] C. Ochoa, G. Puebla, and M. Hermenegildo.
Removing Superfluous Versions in Polyvariant Specialization of Prolog Programs.
In *15th Int'l. Symposium on Logic-based Program Synthesis and Transformation (LOPSTR'05)*, number 3901 in LNCS, pages 80–97. Springer-Verlag, April 2006.

[PEPM'03] G. Puebla and M. Hermenegildo.
Abstract Specialization and its Applications.
In *ACM PEPM'03*, pages 29–43. ACM, June 2003. Invited.

[PEPM'99] G. Puebla, M. Hermenegildo, and J. Gallagher.
An Integration of Partial Evaluation in a Generic Abstract Interpretation Framework.
In *ACM WS on Partial Evaluation and Semantics-Based Program Manipulation (PEPM'99)*, number NS-99-1 in BRISC Series, pages 75–85. U. of Aarhus, Denmark, January 1999.

[JLP'99] G. Puebla and M. Hermenegildo.
Abstract Multiple Specialization and its Application to Program Parallelization.
J. of Logic Programming. 41(2&3):279–316, November 1999.

Automatic parallelization

[TCS'09] D. Cabeza, M. Hermenegildo.
Non-Strict Independence-Based Program Parallelization Using Sharing and Freeness Information.
Theoretical Computer Science, Vol. 46, Num. 410, pages 4704-4723, Elsevier Science, October 2009.

[LOPSTR'07] A. Casas, M. Carro, and M. Hermenegildo.
Annotation Algorithms for Unrestricted Independent And-Parallelism in Logic Programs.
In *17th International Symposium on Logic-based Program Synthesis and Transformation (LOPSTR'07)*, The Technical University of Denmark, August 2007. Springer-Verlag.

[EuroPar'07] M. Hermenegildo.
Automatic Parallelization of Irregular and Pointer-Based Computations: Perspectives from Logic and Constraint Programming.
In *Proceedings of EUROPAR'97*, volume 1300 of *LNCS*, pages 31–46. Springer-Verlag, August 1997. Invited.

[TOPLAS'00] M. García de la Banda, M. V. Hermenegildo, and K. Marriott.
Independence in CLP Languages.
ACM TOPLAS, 22(2):269–339, March 2000.

[TOPLAS'99] F. Bueno, M. García de la Banda, and M. Hermenegildo.
Effectiveness of Abstract Interpretation in Automatic Parallelization: A Case Study in Logic Programming.
ACM Transactions on Programming Languages and Systems, 21(2):189–238, March 1999.

[JLP'99] G. Puebla and M. Hermenegildo.
Abstract Multiple Specialization and its Application to Program Parallelization.
J. of Logic Programming. Special Issue on Synthesis, Transformation and Analysis of Logic Programs, 41(2&3):279–316, November 1999.

[JLP'99] K. Muthukumar, F. Bueno, M. García de la Banda, and M. Hermenegildo.
Automatic Compile-time Parallelization of Logic Programs for Restricted, Goal-level, Independent And-parallelism.
Journal of Logic Programming, 38(2):165–218, February 1999.

[PLILP'96] M. García de la Banda, F. Bueno, and M. Hermenegildo.
Towards Independent And-Parallelism in CLP.
In *Programming Languages: Implementation, Logics, and Programs*, number 1140 in LNCS, pages 77–91, Aachen, Germany, September 1996. Springer-Verlag.

[CompLangJ'96] M. Hermenegildo and M. Carro.
Relating Data–Parallelism and (And–) Parallelism in Logic Programs.
The Computer Languages Journal, 22(2/3):143–163, July 1996.

[JLP'95] M. Hermenegildo and F. Rossi.
Strict and Non-Strict Independent And-Parallelism in Logic Programs: Correctness, Efficiency, and Compile-Time Conditions.
Journal of Logic Programming, 22(1):1–45, 1995.

[SAS'94] D. Cabeza and M. Hermenegildo.
Extracting Non-strict Independent And-parallelism Using Sharing and Freeness Information.
In *1994 International Static Analysis Symposium*, number 864 in LNCS, pages 297–313, Namur, Belgium, September 1994. Springer-Verlag.

[ICLP'91] K. Muthukumar and M. Hermenegildo.
Combined Determination of Sharing and Freeness of Program Variables Through Abstract Interpretation.
In *Intl. Conference on Logic Programming (ICLP 1991)*, pages 49–63. MIT Press, June 1991.

[PLILP'91] F. Giannotti and M. Hermenegildo.
A Technique for Recursive Invariance Detection and Selective Program Specialization.
In *Proc. 3rd. Int'l Symposium on Programming Language Implementation and Logic Programming*, number 528 in LNCS, pages 323–335. Springer-Verlag, August 1991.

[NGC'91] M. Hermenegildo and K. Greene.
The &-Prolog System: Exploiting Independent And-Parallelism.
New Generation Computing, 9(3,4):233–257, 1991.

[ICLP'90] M. Hermenegildo and F. Rossi.
Non-Strict Independent And-Parallelism.
In *1990 International Conference on Logic Programming*, pages 237–252. MIT Press, June 1990.

[NACLP'89] M. Hermenegildo and F. Rossi.
On the Correctness and Efficiency of Independent And-Parallelism in Logic Programs.
In *1989 North American Conference on Logic Programming*, pages 369–390. MIT Press, October 1989.

Cost analysis-based granularity control in parallelism

[DAMP'07] M. Hermenegildo, F. Bueno, A. Casas, J. Navas, E. Mera, M. Carro, and P. López-García.
Automatic Granularity-Aware Parallelization of Programs with Predicates, Functions, and Constraints.
In *DAMP'07, ACM SIGPLAN Workshop on Declarative Aspects of Multicore Programming*, January 2007.

[JSC'96] P. López-García, M. Hermenegildo, and S. K. Debray.
A Methodology for Granularity Based Control of Parallelism in Logic Programs.
Journal of Symbolic Computation, Special Issue on Parallel Symbolic Computation, 21(4–6):715–734, 1996.

[ICLP'95] P. López-García and M. Hermenegildo.
Efficient Term Size Computation for Granularity Control.
In *International Conference on Logic Programming*, pages 647–661, Cambridge, MA, June 1995. MIT Press, Cambridge, MA.

[PASCO'94] P. López-García, M. Hermenegildo, and S.K. Debray.
Towards Granularity Based Control of Parallelism in Logic Programs.
In Hoon Hong, editor, *Proc. of First Intl. Symposium on Parallel Symbolic Computation, PASCO'94*, pages 133–144. World Scientific, September 1994.

[PLDI'90] S. K. Debray, N.-W. Lin, and M. Hermenegildo.
Task Granularity Analysis in Logic Programs.
In *Proc. of the 1990 ACM Conf. on Programming Language Design and Implementation*, pages 174–188. ACM Press, June 1990.

Selected Bibliography: Applying CiaoPP to Various Paradigms

Horn Clauses as Intermediate Representation: Basis for Multi-Language Support

[LOPSTR'07] M. Méndez-Lojo, J. Navas, and M. Hermenegildo.
A Flexible (C)LP-Based Approach to the Analysis of Object-Oriented Programs.
In *17th Intl. Symposium on Logic-based Program Synthesis and Transformation (LOPSTR 2007)*, number 4915 in LNCS, pages 154–168. Springer-Verlag, August 2007.

Analysis/Verification of Energy in ISA, LLVM, Java Bytecode

[TPLP'18] P. Lopez-Garcia, L. Darmawan, M. Klemen, U. Liqat, F. Bueno, and M. V. Hermenegildo.
Interval-based Resource Usage Verification by Translation into Horn Clauses and an Application to Energy Consumption.
Theory and Practice of Logic Programming, Special Issue on Computational Logic for Verification, 18:167–223, March 2018. arXiv:1803.04451.

[HIP3ES'16] U. Liqat, Z. Banković, P. Lopez-Garcia, and M. V. Hermenegildo.
Inferring Energy Bounds Statically by Evolutionary Analysis of Basic Blocks.
In *(HIP3ES'16)*, 2016. arXiv:1601.02800.

[FOPARA'15] U. Liqat, K. Georgiou, S. Kerrison, P. Lopez-Garcia, M. V. Hermenegildo, J. P. Gallagher, and K. Eder.
Inferring Parametric Energy Consumption Functions at Different SW Levels: ISA vs. LLVM IR.
In *Foundational and Practical Aspects of Resource Analysis: 4th Intl. Workshop, FOPARA 2015, Revised Selected Papers*, volume 9964 LNCS, pages 81–100. Springer, 2016.

[HIP3ES'15] P. Lopez-Garcia, R. Haemmerlé, M. Klemen, U. Liqat, and M. V. Hermenegildo
Towards Energy Consumption Verification via Static Analysis.
In *HIPEAC Workshop on High Performance Energy Efficient Embedded Systems (HIP3ES 2015)*, Amsterdam.

[LOPSTR'13] U. Liqat, S. Kerrison, A. Serrano, K. Georgiou, P. López-Garcia, N. Grech, M.V. Hermenegildo, and K. Eder.
Energy Consumption Analysis of Programs based on XMOS ISA-Level Models.
In *Pre-proceedings of the 23rd Intl. Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR'13)*, LNCS, Springer, September 2013.

[NASA FM'08] J. Navas, M. Méndez-Lojo, and M. Hermenegildo.
Safe Upper-bounds Inference of Energy Consumption for Java Bytecode Applications.
In *6th NASA Langley Formal Methods Workshop (LFM 08)*, April 2008. Extended Abstract.

Sharing and Parallelization in Imperative Languages

[ISMM'09] M. Marron, D. Kapur, and M. Hermenegildo.
Identification of Logically Related Heap Regions.
In *ISMM'09: Proceedings of the 8th international symposium on Memory management*, New York, NY, USA, June 2009. ACM Press.

[LCPC'08] M. Marron, D. Kapur, D. Stefanovic, and M. Hermenegildo.
Identification of Heap-Carried Data Dependence Via Explicit Store Heap Models.
In *21st Int'l. WS on Languages and Compilers for Parallel Computing (LCPC'08)*, LNCS. Springer-Verlag, August 2008.

[CC'08] M. Marron, M. Hermenegildo, D. Kapur, and D. Stefanovic.
Efficient context-sensitive shape analysis with graph-based heap models.
In Laurie Hendren, editor, *Intl. Conference on Compiler Construction (CC 2008)*, Lecture Notes in Computer Science. Springer, April 2008.

[PASTE'08] M. Marron, M. Méndez-Lojo, M. Hermenegildo, D. Stefanovic, and D. Kapur.
Sharing Analysis of Arrays, Collections, and Recursive Structures.
In *ACM WS on Program Analysis for SW Tools and Engineering (PASTE'08)*. ACM, November 2008.

[PASTE'07] M. Marron, D. Stefanovic, M. Hermenegildo, and D. Kapur.
Heap Analysis in the Presence of Collection Libraries.
In *ACM WS on Program Analysis for SW Tools and Engineering (PASTE'07)*. ACM, June 2007.

Additional applications to Java bytecode

[VMCAI'08] M. Méndez-Lojo, M. Hermenegildo.
Precise Set Sharing Analysis for Java-style Programs.
9th International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI'08), LNCS, Num. 4905, pages 172-187, Springer-Verlag, January 2008.

[FTfJP'07] J. Navas, M. Méndez-Lojo, and M. Hermenegildo.
An Efficient, Context and Path Sensitive Analysis Framework for Java Programs.
In *9th Workshop on Formal Techniques for Java-like Programs FTfJP 2007*, July 2007.

[Bytecode'07] M. Méndez-Lojo, J. Navas, and M. Hermenegildo.
An Efficient, Parametric Fixpoint Algorithm for Analysis of Java Bytecode.
In *ETAPS Workshop on Bytecode Semantics, Verification, Analysis and Transformation (BYTECODE'07)*, Electronic Notes in Theoretical Computer Science. Elsevier - North Holland, March 2007.

[Bytecode'09] J. Navas, M. Méndez-Lojo, and M. Hermenegildo.
User-Definable Resource Usage Bounds Analysis for Java Bytecode.
In *Proceedings of the Workshop on Bytecode Semantics, Verification, Analysis and Transformation (BYTECODE'09)*, volume 253 of *Electronic Notes in Theoretical Computer Science*, pages 6–86. Elsevier - North Holland, March 2009.

Other Applications of CiaoPP: MiniZinc, Services.

[Computing'13] D. Ivanovic, M. Carro, and M. V. Hermenegildo.
A Sharing-Based Approach to Supporting Adaptation in Service Compositions.
Computing, 95(6):453–492, June 2013.

[MZN'11] F. Bueno, M. García de la Banda, M. V. Hermenegildo, P. López-García, E. Mera, and P. J. Stuckey.
Towards Resource Usage Analysis of Minizinc Models.
In *MiniZinc Workshop (MZN'11)*, Perugia, Italy, September 2011.

[ICSOC'11] D. Ivanović, M. Carro, and M. V. Hermenegildo.
Constraint-Based Runtime Prediction of SLA Violations in Service Orchestrations.
In Gerti Kappel, Hamid Motahari, and Zakaria Maamar, editors, *Service-Oriented Computing – ICSOC 2011*, volume 7084 of *LNCS*, pages 62–76. Springer Verlag, December 2011.
Best paper award.

[ICSOC'10] D. Ivanović, M. Carro, and M. V. Hermenegildo.
Automatic Fragment Identification in Workflows Based on Sharing Analysis.
In Mathias Weske, Jian Yang, Paul Maglio, and Marcelo Fantinato, editors, *Service-Oriented Computing – ICSOC 2010*, volume 6470 of *LNCS*, pages 350–364. Springer Verlag, 2010.