
An Overview of The Ciao Multiparadigm Language and Program Development Environment and its Design Philosophy

Manuel Hermenegildo^{1,3,4}

(with F. Bueno,¹ G. Puebla,¹ M. Carro,¹ Daniel Cabeza,¹ P. López-García,¹ E. Albert,²
J. Navas,³ M. Méndez,³ A. Casas,³ J. Correas,² J. Morales,² E. Mera,¹ D. Trallero,¹
C. Ochoa,¹ P. Pietrzak,¹ P. Arenas,² S. Genaim¹)

The Ciao Development Team

¹*Fac. Informática, U. Politécnica de Madrid*

²*Fac. Informática, U. Complutense de Madrid*

³*CS and EECE Depts., U. of New Mexico, USA*

⁴*IMDEA SW Development Technology Institute*

MPOOL – July 31, 2007

Modular, Evolvable Design

- Ciao is built in layers over a small, pure (LP-based) *kernel*.
 - Allows designing *syntactic and semantic extensions* in a simple, flexible, and scalable way.
 - Also, building small, fast executables and embeddability (non-needed parts of the language and libraries are not included).
 - Fundamental enabler –its module/class system:
 - Designed from the ground up to be *extensible* and *analysis-friendly*.
 - Most language features (loops, conditionals, functions, ...) are not built-in, but rather in libraries.
 - Language extension libraries (“packages”) affect only the modules in which they are loaded (e.g., operators and expansions are local to modules, etc.).
- Allows modular program development, separate/incremental compilation.
- Allows extensive global analysis for detecting errors and optimizing code.
- Support for programming “in the large.”

Multiparadigm

- *Logic programming:*
 - Certainly ISO-Prolog (one of the best Prologs!) –but *via a library*; and also:
 - Pure LP.
 - Various comp. rules: breadth-first, iterative-deepening, Andorra, *tabling*, etc.
 - *Functional programming:*
 - Function definitions and function calls and functional syntax for predicates.
 - *Higher-order* and *lazyness* for functions and predicates.
 - *Constraint programming:* clpr, clpq, fd, Leuven CHR.
 - *Objects:* a naturally embedded notion of classes and objects.
 - *Concurrency, parallelism, distributed execution.*
 - *Imperative features:* mutable data struct., assignment, loops, cases, etc.
 - *Assertion language*, consistent across paradigms; with many uses!
- + many other packages: types, records, DCGs, application-specific languages, ...

Advanced Development Environment

- *Emacs* and *eclipse* versions.
- Top-level, source debugger, standalone optimizing compiler, script interpreter, ...
- Autodocumenter, large set of libraries, ...
- Preprocessor (*ciaopp*):
 - Input: program, optionally w/assertions declaring properties such as types, modes, det, nf, sizes, cost, resources, etc.
 - Output: *error/warning messages + transformed program*, with
 - Results of static checking of assertions / error detection / verification. (and certificates for Abstraction Carrying Code).
 - Assertion run-time checking code.
 - High-level optimizations (specialization, slicing, parallelization).
 - Results of analysis (as assertions): used for low-level optimizations.
 - Technology: modular polyvariant abstract interpretation/specialization.

Verification and Diagnosis via Abstract Interpretation [AADEBUG'97]

- Program verification/diagnosis: compare $\llbracket P \rrbracket$ with intended semantics \mathcal{I} e.g.:

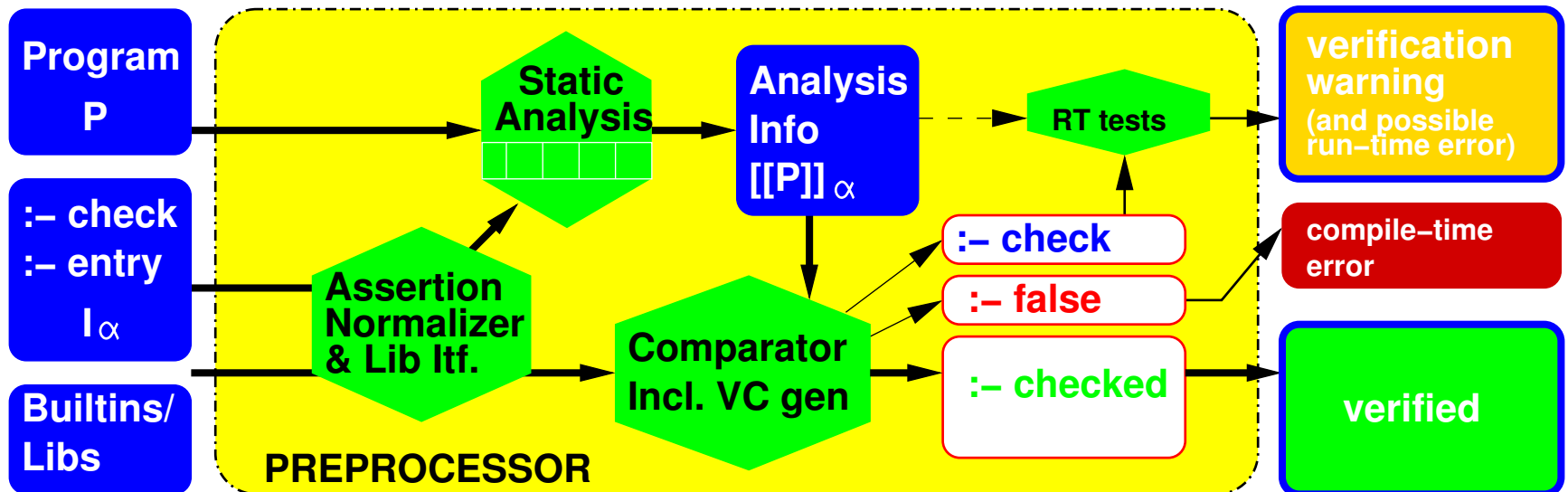
P is <i>partially correct</i> w.r.t. \mathcal{I} iff	$\llbracket P \rrbracket \subseteq \mathcal{I}$
P is <i>complete</i> w.r.t. \mathcal{I} iff	$\mathcal{I} \subseteq \llbracket P \rrbracket$
P is <i>incorrect</i> w.r.t. \mathcal{I} iff	$\llbracket P \rrbracket \not\subseteq \mathcal{I}$
P is <i>incomplete</i> w.r.t. \mathcal{I} iff	$\mathcal{I} \not\subseteq \llbracket P \rrbracket$

- Usually, only partial descriptions of \mathcal{I} are available, typically as *assertions*.
- Problem*: difficulty in computing $\llbracket P \rrbracket \rightarrow$ use *abstract interpretation* to compute a *safe approximation* $\llbracket P \rrbracket_{\alpha^+}$
 $\llbracket P \rrbracket_{\alpha^+}$ indicates $\llbracket P \rrbracket_{\alpha} \supseteq \alpha(\llbracket P \rrbracket)$.

Property	Definition	Sufficient condition
P is partially correct w.r.t. \mathcal{I}_{α} if	$\alpha(\llbracket P \rrbracket) \subseteq \mathcal{I}_{\alpha}$	$\llbracket P \rrbracket_{\alpha^+} \subseteq \mathcal{I}_{\alpha}$ if
P is complete w.r.t. \mathcal{I}_{α} if	$\mathcal{I}_{\alpha} \subseteq \alpha(\llbracket P \rrbracket)$	$\mathcal{I}_{\alpha} \subseteq \llbracket P \rrbracket_{\alpha^+}$
P is incorrect w.r.t. \mathcal{I}_{α} if	$\alpha(\llbracket P \rrbracket) \not\subseteq \mathcal{I}_{\alpha}$	$\llbracket P \rrbracket_{\alpha^+} \not\subseteq \mathcal{I}_{\alpha}$, 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}_{α} if	$\mathcal{I}_{\alpha} \not\subseteq \alpha(\llbracket P \rrbracket)$	$\mathcal{I}_{\alpha} \not\subseteq \llbracket P \rrbracket_{\alpha^+}$

- Specially attractive if compiler computes (most of) $\llbracket P \rrbracket_{\alpha^+}$ anyway.

Verification/Diagnosis Framework – Architecture in Ciao



- I_α (partial specification) provided via an *assertion language*. Assertions state properties at calls/success and of (parts of) the execution.
- Predefined properties + others user-defined (and written in source language). Types, modes, pointer sharing, cost, sizes, termination, determinacy, non-fail, ...

Assertion Language: Properties

```

:- regtype list/1.
list([]).
list(_|Y) :- list(Y).
-----
:- prop sorted/1.
sorted([]).
sorted(_).
sorted([X,Y|Z]) :- X>Y, sorted([Y|Z]).
| :- regtype list/1.
| list := [] | [_|~list].
| -----
| :- regtype color/1.
| color := green | blue | red.
| -----
| :- regtype peano_int/1.
| peano_int := 0 | s(~peano_int).

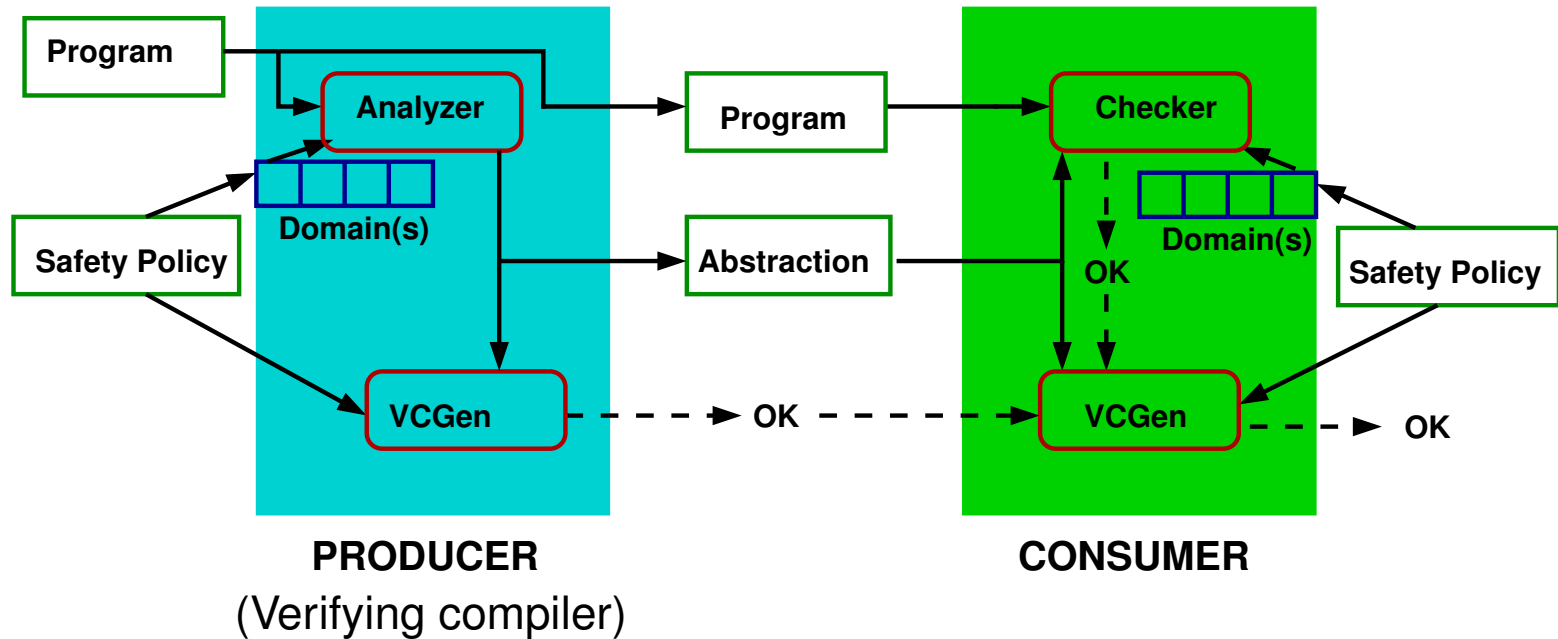
```

- Arbitrary predicates in restricted logic (a subset of Ciao).
- Some conditions on them: termination, no instantiation, ...
- Many predefined in system libs, some of them “native” to an analyzer.
- Can also be user-defined.
- Should be visible in the module and “runnable:” they will be used also as run-time tests! (but the property may be an approximation itself).
- *Types* are a special case of property (e.g., regtypes).
- But also, e.g., argument sizes, instantiation states, sizes, cost, ...

Assertion Language: *Pred* Assertions

- `:- pred PredPattern [: Pre] [=> Post] [+ Comp].`
 - *Closed* on calls: cover all uses of a predicate (they imply a calls assertion).
 - *Assertion status*: check, true/false, trust, checked.
- Some examples, and some syntactic sugar:
 - `:- pred qsort(X,Y) => sorted(Y).`
 - `:- pred qsort(X,Y) : list(int) * var => sorted(Y) + (is_det,not_fails).`
`:- pred qsort(X,Y) : var * list(int)) => ground(X) + not_fails.`
 - `:- pred foo(X,Y) : ground * var => (ground(Y), X>Y) + det.`
`:- pred foo(X,Y) : var * ground => (ground(X), X>Y).`
 - `:- trust pred is/2 => num * numexpr.`
 - `:- modedef +X : nonvar(X).`
 - `:- pred sortints(+L,-SL) :: list(int) * list(int) => sorted(SL)`
`# "@var{SL} has same elements as @var{L}."`

The Abstraction Carrying Code (ACC) Scheme [COCV04,LPAR04]



$$\llbracket P \rrbracket_{\alpha} = \text{Analysis} = \text{lfp}(\text{analysis_step})$$

$$\text{Certificate} \subset \llbracket P \rrbracket_{\alpha}$$

$$\text{Checker} = \text{analysis_step}$$

- Scheme incorporated in CiaoPP, with domains: types, modes, data structure shape (including pointer sharing), bounds on data structure sizes, determinacy, termination, non-failure, bounds on resource consumption (time or space cost), ...

Discussion: Comparison with “classical” Types

- Allows going well beyond the “straight-jacket” of classical type systems:

“Traditional” Types	CiaoPP Assertion-based Model [AADEBUG’97]
“Property” language limited by decidability	Much more general property language
May need to limit programming language	No need to limit programming language
“Untypable” programs rejected	Run-time checks introduced
(Almost) Decidable	Decidable and Undecidable (and Approximated)
Expressed in a special language	(Expressed in the source language –for LP)
Types must be defined	Types can be defied or inferred
Assertions are only of type “check”	“check”, “trust”, ...
Type signatures and assertions different	Type signatures <i>are</i> assertions

...without giving up much (types are included as just another kind of property).

- Some key issues:

Approximation

Abstract Interpretation

Suitable assertion language

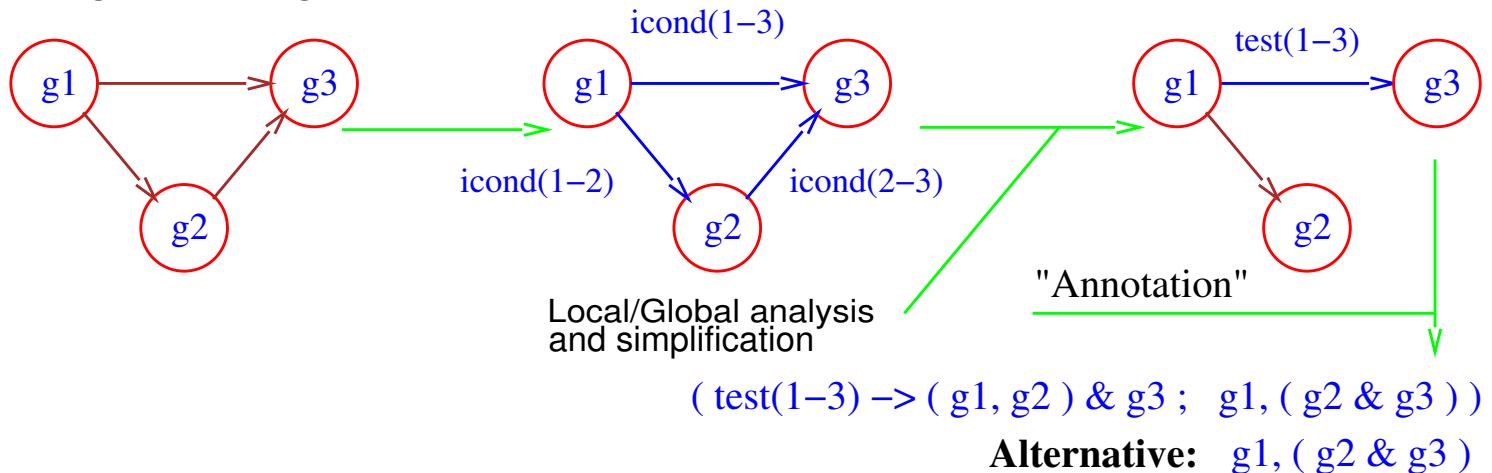
Powerful abstract domains

- Worst best when properties and assertions can be expressed in the source language (i.e., the source language supports *predicates* and *constraints*).

Automatic Program Parallelization

- Parallelization process [4] starts with dependency graph:
 - edges exist if there can be a dependency,
 - conditions label edges if the dependency can be removed.
- Global analysis: reduce number of checks in conditions (also to true and false).
- Annotation: encoding of parallelism in the target parallel language:

$g_1(\dots), g_2(\dots), g_3(\dots)$



- Granularity control: based on cost / size analysis.

Automatic Program Parallelization (Contd.)

- *Example:*

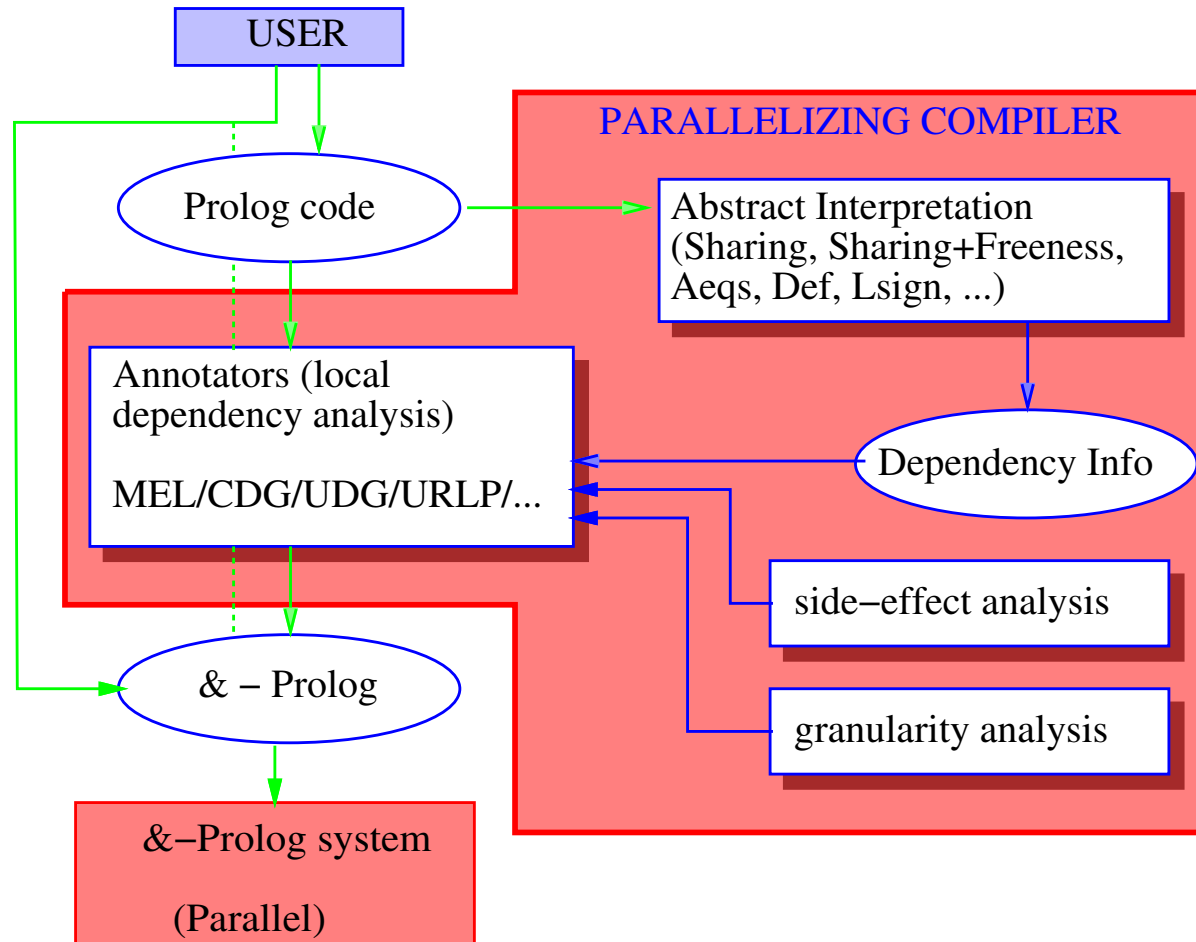
```
qs([X|L],R) :- part(L,X,L1,L2),
               qs(L2,R2), qs(L1,R1),
               app(R1,[X|R2],R).
```

Might be annotated in &-Prolog (or Ciao Prolog), using local analysis, as:

```
qs([X|L],R) :-
    part(L,X,L1,L2),
    ( indep(L1,L2) ->
      qs(L2,R2) & qs(L1,R1)
    ;   qs(L2,R2) , qs(L1,R1) ),
    app(R1,[X|R2],R).
```

Global analysis would eliminate the `indep(L1,L2)` check.

&-Prolog/Ciao parallelizer overview



Granularity Control

- Do not schedule tasks for parallel execution if they are too small.
- Cannot be done well completely at compile-time: work done by a call often depends on the size of its input:
 - $q([], [])$.
 - $q([X|RX], [X1|RX1]) :- X1 \text{ is } X + 1, \quad q(RX, RX1)$.
- Approach [3]:
 - generate at compile-time *functions* (to be evaluated at run-time) that efficiently approximate task size (upper and lower bounds),
 - transform programs to carry out run-time granularity control.
 - Note: size computations can be done on-the-fly.
- Example (with q above):
 - ..., $q(X, Y) \ \& \ r(X)$, ...
 - Cost = $2 * \text{length}(X) + 1$ (cost function $2 * n + 1$). Assuming *threshold* is 4 units:
 - ..., $\text{length}(X, LX)$, Cost is $LX * 2 + 1$, (Cost > 4 \rightarrow $q(X, Y) \ \& \ r(Z)$
; $q(X, y), \ r(X)$), ...

Granularity Control System Output

```

g_qsort([], []).
g_qsort([First|L1], L2) :-
    partition3o4o(First, L1, Ls, Lg, Size_Ls, Size_Lg),
    Size_Ls > 20 ->
        (Size_Lg > 20 -> g_qsort(Ls, Ls2) & g_qsort(Lg, Lg2);
         g_qsort(Ls, Ls2), s_qsort(Lg, Lg2));
    (Size_Lg > 20 -> s_qsort(Ls, Ls2), g_qsort(Lg, Lg2);
     s_qsort(Ls, Ls2), s_qsort(Lg, Lg2)),
    append(Ls2, [First|Lg2], L2).

```

```

partition3o4o(F, [], [], [], 0, 0).
partition3o4o(F, [X|Y], [X|Y1], Y2, SL, SG) :-
    X =< F, partition3o4o(F, Y, Y1, Y2, SL1, SG), SL is SL1 + 1.
partition3o4o(F, [X|Y], Y1, [X|Y2], SL, SG) :-
    X > F, partition3o4o(F, Y, Y1, Y2, SL, SG1), xSG is SG1 + 1.

```

- Note: when term sizes are compared directly with a threshold: not necessary to traverse all the terms involved, only to the point at which threshold is reached.

Some Members of The Ciao Forge

- Ciao is really a widely distributed collaborative effort:
 - Directly within the CLIP Group:

M. Hermenegildo, K. Muthukumar, M. García de la Banda, F. Bueno, G. Puebla, M. Carro, D. Cabeza, P. López-G., E. Albert, J. Navas, M. Méndez, A. Casas, J. Correas, J. Morales, E. Mera, D. Trallero, C. Ochoa, P. Pietrzak, P. Arenas, S. Genaim
 - Plus lots of contributors worldwide:

G. Gupta (UT Dallas), E. Pontelli (NM State University), P. Stuckey and M. García de la Banda (Melbourne U.), K. Marriott (Monash U.), M. Bruynooghe, A. Mulkers, G. Janssens, and V. Dumortier (K.U. Leuven), S. Debray (U. of Arizona), J. Maluzynski and W. Drabent, (Linkoping U.), P. Deransart (INRIA), J. Gallagher (Roskilde University), C. Holzbauer (Austrian Research Institute for AI), M. Codish (Beer-Sheva), SICS, ...

Downloading the systems

- Downloading Ciao, CiaoPP, LPdoc, and other CLIP software:
 - Standard distributions:
`http://www.clip.dia.fi.upm.es/Software`
 - Some betas (in testing or completing docs – ask webmaster for info) in:
`http://www.clip.dia.fi.upm.es/Software/Beta`
Mail list stored in
`http://www.clip.dia.fi.upm.es/Mail/ciao-users/`
 - Please contact us for [SVN access](#).
- User's mailing list:
`ciao-users@clip.dia.fi.upm.es`

Subscribe by sending a message with only `subscribe` in the body to
`ciao-users-request@clip.dia.fi.upm.es`

Some Bibliography on the Ciao System

● Basic system manual:

- [1] F. Bueno, D. Cabeza, M. Carro, M. Hermenegildo, P. López-García, and G. Puebla (Eds.). The Ciao System. Ref. Manual (v1.13). Technical report, C. S. School (UPM), 2006. Available at <http://www.ciaohome.org>.
- [2] F. Bueno. The CIAO Multiparadigm Compiler: A User's Manual. Technical Report CLIP8/95.0, Facultad de Informática, UPM, June 1995.

● Overall philosophy:

- [1] The Ciao Development Team. The Ciao Multiparadigm Language and Program Development Environment, November 2006. The ALP Newsletter 19(3). The Association for Logic Programming. Available from <http://www.logicprogramming.org/newsletter/nov06/index.html>.
- [2] 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.
- [3] D. Cabeza. *An Extensible, Global Analysis Friendly Logic Programming System*. PhD thesis, Universidad Politécnica de Madrid (UPM), Facultad Informatica UPM, 28660-Boadilla del Monte, Madrid-Spain, August 2004.
- [4] 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.
- [5] 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.

● Functions, higher order, lazyness:

- [1] 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.
- [2] 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.

● Objects:

- [1] 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.
- [2] 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.

● Auto-documenter:

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

● Abstract machine and low-level optimization:

- [1] 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*, pages 271–281. ACM Press / Sheridan, October 2006.
- [2] 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.
- [3] J.F. Morales, M. Carro, and M. Hermenegildo. Towards Description and Optimization of Abstract Machines in an Extension of Prolog. In Germn Puebla, editor, *Logic-Based Program Synthesis and Transformation (LOPSTR'06)*, number 4407 in LNCS, pages 77–93, May 2007.
- [4] 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*, number 3057 in LNCS, pages 86–103, Heidelberg, Germany, June 2004. Springer-Verlag.

● Automatic parallelization:

- [1] 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.
- [2] 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.
- [3] 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.

- [4] 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.
- [5] 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.
- [6] 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.
- [7] M. Hermenegildo and K. Greene. The &-Prolog System: Exploiting Independent And-Parallelism. *New Generation Computing*, 9(3,4):233–257, 1991.

● Cost analysis and granularity control in parallelism:

- [1] 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.
- [2] 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, Namur, Belgium, September 1994. Springer-Verlag.
- [3] 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.
- [4] 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.
- [5] 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 2007)*, LNCS. Springer-Verlag, September 2007.

● Partial evaluation:

- [1] G. Puebla and M. Hermenegildo. Abstract Specialization and its Applications. In *ACM Partial Evaluation and Semantics based Program Manipulation (PEPM'03)*, pages 29–43. ACM Press, June 2003. Invited.
- [2] E. Albert, G. Puebla, and J. Gallagher. Non-Leftmost Unfolding in Partial Evaluation of Logic Programs with Impure Predicates. In *15th International Symposium on Logic-based Program Synthesis and Transformation (LOPSTR'05)*, number 3901 in LNCS, pages 115–132. Springer-Verlag, April 2006.
- [3] G. Puebla and C. Ochoa. Poly-Controlled Partial Evaluation. In *Proc. of 8th ACM-SIGPLAN International Symposium on Principles and Practice of Declarative Programming (PPDP'06)*, pages 261–271. ACM Press, July 2006.
- [4] G. Puebla, E. Albert, and M. Hermenegildo. Abstract Interpretation with Specialized Definitions. In *The 13th International Static Analysis Symposium (SAS'06)*, number 4134 in LNCS, pages 107–126. Springer, August 2006.
- [5] 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.

● The overall program development framework (CiaoPP):

- [1] 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.
- [2] 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.
- [3] 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.

- [4] 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, September 2000.
- [5] G. Puebla, F. Bueno, and M. Hermenegildo. An Assertion Language for Constraint Logic Programs. In P. Deransart, M. Hermenegildo, and J. Maluszynski, editors, *Analysis and Visualization Tools for Constraint Programming*, number 1870 in LNCS, pages 23–61. Springer-Verlag, September 2000.
- [6] M. Hermenegildo, G. Puebla, F. Bueno, and P. López-García. Abstract Verification and Debugging of Constraint Logic Programs. In *Recent Advances in Constraints*, number 2627 in LNCS, pages 1–14. Springer-Verlag, January 2003.
- [7] M. Hermenegildo, G. Puebla, F. Bueno, and P. López-García. Program Development Using Abstract Interpretation (and The Ciao System Preprocessor). Invited talk. In *10th International Static Analysis Symposium (SAS'03)*, number 2694 in LNCS, pages 127–152. Springer-Verlag, June 2003.

● Abstraction carrying code:

- [1] E. Albert, G. Puebla, and M. Hermenegildo. Abstraction-Carrying Code. In *Proc. of LPAR'04*, number 3452 in LNAI, pages 380–397. Springer-Verlag, 2005.
- [2] 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.
- [3] E. Albert, P. Arenas, G. Puebla, and M. Hermenegildo. Reduced Certificates for Abstraction-Carrying Code. In *22nd International Conference on Logic Programming (ICLP 2006)*, number 4079 in LNCS, pages 163–178. Springer-Verlag, August 2006.
- [4] E. Albert, P. Arenas, and G. Puebla. An Incremental Approach to Abstraction-Carrying Code. In *13th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'06)*, number 4246 in LNCS, pages 377–391. Springer-Verlag, November 2006.

● Scalability, modularity of analysis, debugging, and verification:

- [1] P. Pietrzak, J. Correas, G. Puebla, and M. Hermenegildo. Context-Sensitive Multivariant Assertion Checking in Modular Programs. In *13th International Conference on Logic for Programming Artificial Intelligence and Reasoning (LPAR'06)*, number 4246 in LNCS, pages 392–406. Springer-Verlag, November 2006.
- [2] 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.
- [3] G. Puebla, J. Correas, M. Hermenegildo, F. Bueno, M. García de la Banda, K. Marriott, and P. J. Stuckey. A Generic Framework for Context-Sensitive Analysis of Modular Programs. In M. Bruynooghe and K. Lau, editors, *Program Development in Computational Logic, A Decade of Research Advances in Logic-Based Program Development*, number 3049 in LNCS, pages 234–261. Springer-Verlag, Heidelberg, Germany, August 2004.