The CIAO Multi-Dialect Compiler and System: A Demo and a Status Report

M. Hermenegildo† F. Bueno† D. Cabeza† M. Carro†
M. García de la Banda‡ P. López† G. Puebla†

Extended Abstract

This paper presents a brief overview of the CIAO system, the capabilities of its compiler and the techniques used for supporting them. It essentially provides an overview of current work at the UPM CLIP group, performed in collaboration with the U. of Arizona, K. U. Leuven, Melbourne U., Monash U., and New Mexico State U. More details and references on this project can be found in the full version of the paper [Her].

CIAO is a compiler, run-time, and program development system which supports several programming paradigms and execution models in a unified framework. It was initially conceived as a compiler writer's workbench, i.e., a testbed for advanced compilation techniques, but has since proven also quite useful for application development. Programming styles supported include those of Prolog/Logic Programming, CLP, and CC. In the belief that network applications are a good target for computational logic systems, the styles above are combined in CIAO with distributed and network-wide execution capabilities. Such capabilities allow both the transparent execution of parallel and concurrent code written for a multiprocessor in a distributed environment [CH95] and the straightforward generation of WWW-based applications [CH96a, CH96b, CHV96]. CIAO also supports several computation rules, including standard left-to-right SLD resolution and the determinate-first principle (as in the Andorra model [SCWY90, dMSC93]).

The implementation of CIAO is based on the observation that, under certain assumptions, a large number of currently proposed LP, CLP, and CC programming and execution models can be explained and implemented through the application of only a few basic principles and constructs, and that there is much in common at the abstract machine level among such models [HtCg93, HtCg94]. Thus, CIAO programs are compiled and optimized via source to source transformations into a kernel language (which is also the native CIAO language). We believe medium to high performance implementations can be obtained in

*This work has been funded in part by ESPRIT projects PRINCE, PARFORCE, and ACCLAIM, and CICYT project IPL-D.

†CS Dept., Technical U. of Madrid. {herme,bueno,dcabeza,mcarro,plg,german}@fi.upm.es

‡Dept. of CS, Monash University, Clayton 3168, Australia. mbanda@bruce.cs.monash.oz.au

this way, with the advantages that most analysis phases can be performed at the kernel language level and the same analyzer can be shared among several execution models. Also, optimizations can be performed via source to source transformations, and the low-level machinery can be kept minimal. Optimizations in CIAO include multivariant (abstract) specialization and automatic program parallelization [CC94]. The kernel language in turn is directly supported by a comparatively simple, generic abstract machine including parallelism, concurrency, and distributed execution capabilities (and which is essentially an extension of the &-Prolog abstract machine [Her86, HG91]). It includes native support for *attributed variables* [Hol92, Hui90, Neu90], which are used in the implementation of constraint solvers (as in Eclipse [Eur93] and SICStus 3 [Swe95]) and for communication among concurrent tasks [HCC95].[1] In addition, the CIAO compiler can also support a significant part of the system's functionality by compilation into certain Prolog dialects (those which include coroutining and attributed variables, such as Eclipse or SICStus 3). Thus, the CIAO system can also be seen as a library or front end for current Prolog systems.

One of the design decisions in CIAO is to provide *explicit control capabilities in the kernel language*. This allows the system to perform optimizations such as parallelization [BGH94] (task creation based on dependencies), partitioning and schedule analysis (task coalescence based on dependencies), and granularity control (task coalescence based on task size considerations) as source to source transformations. Such control is provided first by separate *sequentiality* ("`,`"), *concurrency* ("`&/1`"), and *parallelism* ("`&/2`", "`&>/2`", "`<&/1`") operators. The *parallel operators* allow indicating points where parallelism can be exploited. Their behavior is otherwise equivalent to that of the sequential operator (full backtracking is supported). These operators essentially assume independence among goals and communication of bindings is therefore not guaranteed until the join. No variable locking is performed. The *concurrency operator* allows concurrent programming in the style of CC languages (also, the parallelism in such concurrent execution may be exploited if resources are available). Backtracking is limited to allow a relatively straightforward implementation. Variable communication (and locking) is performed. In addition, a *fair concurrency operator* is provided ("`&&/1`") which explicitly requests the (efficient) association of computational resources (e.g., an operating system thread) to a goal. This allows implementing a fair source language that compiles efficiently into this and the above operators (perhaps via an analysis which can determine the program points where fairness is really needed – to ensure, for example, termination). Also, *explicit synchronization* is provided in the kernel language by means of "`wait/1`" and "`ask/1`" operators (the latter as in concurrent constraint programming [JM94, Tic95], the former as in &-Prolog), augmented with some meta-tests on the variables (the latter, using the attributted variables approach of Holzbaur [Hol90]). Finally, a *placement operator* ("`@`") allows control of task placement in distributed execution. Other primitives for controlling distributed execution deal with teams of workers, and with using active modules or active objects.

The *CIAO compiler* first performs a translation of the input source into the kernel

[1] While the abstract machine currently supports only ("dependent" and "independent") and-parallelism, its combination with or-parallelism is planned by applying the techniques developed in ACE and similar models [GSCYH91, GC92, GHPC94].

language, followed by (incremental) global analysis and optimization phases. The resulting program is finally compiled into abstract machine code (including byte-code files, standalone executables, and incore compilation). Alternatively, the compiler can produce output which can run on a Prolog system supporting delay and attributed variables, as mentioned before. It is also possible to obtain the results of each of the intermediate compilation phases. This allows visualizing and affecting the transformation, analysis, parallelization, and optimization steps. Because of the source to source nature of the compiler, this output is always a (possibly annotated) kernel CIAO program. Also, (parallel) execution traces can be generated which can be analyzed by several graphical performance analysis tools.

The compiler currently supports the CIAO kernel language (backwards compatible with Prolog, but extended with the specific CIAO primitives mentioned before), languages based on the basic Andorra model, and basic CC languages. Program transformations bridge the semantic gaps between the different programming paradigms supported. The methods used for translating programs based on the (Basic) Andorra model to CIAO are described in [BDGH95]. The methods used for translating CC languages are an extension of those of [DGB94, Deb93] and are described in [BH95b]. Also, for each of these languages the system currently supports the constraint domains of Prolog, CLP(R), and CLP(Q). Finally, there is support for programming in the functional style, both in terms of syntax (allowing functional notation and nesting of calls via a designated output argument in relations) and of improved support at the abstract machine for meta-programming (higher-order).

The CIAO compiler includes both local and global analysis of programs. Global analysis is performed in the context of abstract interpretation [CC77, Deb92, CC92]. The underlying framework of analysis is that of PLAI [HWD92, MH90, MH92], which implements a generic (goal-dependent and goal-independent) top-down driven abstract interpreter. The genericity of the framework allows plugging in different abstract domains, provided suitable interfacing functions are defined. PLAI also incorporates incremental analysis [HMPS95] in order to deal with large programs and is capable of analyzing full languages (in particular, full standard Prolog [BCHP96, CRH94]). A modification of the PLAI framework capable of analyzing dynamically scheduled programs is also included in order to support the concurrent models. Note that, thanks to the transformational approach, the same framework allows analyzing programs with delays [Col82, Car87] and CC languages with angelic nondeterminism.[2] Initial studies showed that accurate analysis in such programs is possible [MGH94], although this technique involves relatively large cost in analysis time. The analysis integrated into the CIAO compiler, developed in collaboration with Monash and Melbourne U., uses a novel method based on approximating the delayed atoms by a closure operator, which improves efficiency without significant loss of accuracy [GMS95].

The CIAO analyzer currently includes several domains, some of which have been implemented by other users of the PLAI system, notably the K. U. Leuven, Monash U., and Melbourne U. For the analysis of (classical) logic programs a number of traditional domains capturing information on variable groundness, freeness, sharing, and linearity are included. These include the set sharing *Sh* [JL89, MH89], set sharing and freeness *Sh+Fr* [MH91], and

[2]This is a kind of nondeterminism which does not give rise to an arbitrary choice when applying a search rule.

pair sharing *ASub* [Son86] domains. Combinations of *Sh* and *Sh+Fr* with *ASub* are also supported, resulting in the *Sh+ASub* and *Sh+Fr+ASub* domains. The combination is done in such a way that the original domains and operations of the analyzer over them are re-used, instead of redefining the domains for the combination [CC79, CMB+95]. Three other domains, modified versions of *Path sharing*, *Aeqns* (abstract equations), and *Types* have recently also been incorporated (the latter is needed in granularity control and also allows interactive type checking and generation). Additional domains are supported for CLP. The abstract domain *Def* [GH93, Gar94] determines whether program variables are definite, i.e. constrained to a unique value. In doing this it keeps track of *definite* dependencies among variables. The abstract domain *Fr* [DJBC93, DJ94, Dum94] determines which variables act as *degrees of freedom* with respect to the satisfiability of the constraint store in which they occur. In doing this it keeps track of *possible* dependencies among variables. A combined domain *Fd* which infers both definiteness and freeness is also integrated. Finally, a preliminary version of the domain *LSign* [MS94] is also supported.

The compile-time *parallelization* module currently aims at uncovering goal-level, restricted (i.e., fork and join), independent and-parallelism (independence has the very desirable properties of correct and efficient execution w.r.t. standard sequential execution of Prolog or CLP). In the context of LP, parallelization is performed based on the well-understood concepts of *strict* and *non-strict* independence [HR95], using the information provided by the abstract domains. Also, appropriate parallelizers using recently proposed definitions of independence for CLP [GHM93, Gar94] (and for constraint programming with dynamic scheduling) have been implemented in order to parallelize CLP and CC programs [GHM95, GBH96], making use of the information from the CLP analyses.

The optimization phase in the compiler is strongly based on (abstract) program specialization. Literals and predicates which are proved to always succeed, fail, be reducible to a few unifications, or lead to error are simplified or eliminated. The same principle applies at lower levels. This can speed up the program at run-time (by eliminating static code such as run-time tests that are always known to succeed), and also be useful to detect programming errors at compile-time. At user request the compiler can also specialize the program using the versions generated during analysis [PH95a]. This may involve generating different versions of a predicate for different *abstract call patterns*, thus increasing the program size. In order to keep the size of the specialized program as reduced as possible, the number of versions of each predicate is minimized to those which are found useful for performing optimizations. The optimization module also supports optimization in the context of dynamic scheduling [PH95b]. The optimizations then include simplification and elimination of suspension conditions and elimination of concurrency primitives (sequentialization).

As well as handling sequential code, the optimization module of the CIAO compiler contains what we believe is the first automatic optimizer for languages with dynamic scheduling [PH95b]. The potential benefits of the optimization of this type of programs were already shown in [MGH94], but they can now be obtained automatically. These kinds of optimizations include simplification and elimination of suspension conditions and elimination of concurrency primitives (sequentialization).

A further step in the compiler provides granularity control for parallel execution using the techniques described in [DLH90, Tic88, LHD94, DGHL94, LH95]. The compiler estimates the granularity of parallel tasks, i.e. the work available under them, by generating expressions that are upper and lower bounds for the computation time of parallel tasks as a function of the size of task input data. These functions are used at run-time to perform granularity control by comparing cost bounds to suitable thresholds. The preliminary results obtained are quite encouraging, although there remains much work to be done in this very important area.

The back end of the compiler takes the result of the previous program transformations and generates a number of final output formats. Normally, the result of the compiler is intended for the CIAO/&-Prolog abstract machine. Output possibilities are then byte-code ("`.ql`") files, stand-alone executables, and incore compilation (when the compiler is running inside the system rather than as a stand-alone application). As mentioned before, and as an alternative output, most of the capability of the system can also be handled by any Prolog which supports delay declarations and attributed variables. Alternatively, also AKL [JH91] can be used as a target, using the techniques described in [BH95a].

Two tools have been developed to complement the environment and help during performance debugging. VisAndOr [CGH93] is a tool for visualizing parallel executions. It supports both conjunctive and disjunctive execution graphs (or- and and-parallelism). It is currently in use by several other researchers in the field and it is distributed with other parallel systems such as Muse [AK90]. IDRA [FCH96] is a simulator which can quite accurately compute ideal speedups from traces from a sequential execution of a parallelized program. It allows evaluating the performance of the parallel run-time system independently of the quality of the parallelization performed by the compiler, by comparing the obtained speedups with those predicted by IDRA for the given parallelized program.

Current versions of different parts of the &-Prolog and CIAO systems are publicly available for experimentation (please contact the authors; further information can be obtained from `http://www.clip.dia.fi.upm.es/`).

References

[AK90] K. A. M. Ali and R. Karlsson. The Muse Or-Parallel Prolog Model and its Performance. In *1990 North American Conference on Logic Programming*, pages 757–776. MIT Press, October 1990.

[BCHP96] F. Bueno, D. Cabeza, M. Hermenegildo, and G. Puebla. Data–flow Analysis of Standard Prolog Programs. In *European Symposium on Programming*, number 1058 in LNCS, pages 108–124, Sweden, April 1996. Springer-Verlag.

[BDGH95] F. Bueno, S. Debray, M. García de la Banda, and M. Hermenegildo. Transformation-based Implementation and Optimization of Programs Exploiting the Basic Andorra Model. Technical Report CLIP11/95.0, Facultad de Informática, UPM, May 1995.

[BGH94] F. Bueno, M. García de la Banda, and M. Hermenegildo. Effectiveness of Global Analysis in Strict Independence-Based Automatic Program Parallelization. In *International Symposium on Logic Programming*, pages 320–336. MIT Press, November 1994.

[BH95a] F. Bueno and M. Hermenegildo. An Automatic Translation Scheme from CLP to AKL. Technical Report CLIP7/95.0, Facultad de Informática, UPM, June 1995.

[BH95b] F. Bueno and M. Hermenegildo. Compiling Concurrency into a Sequential Logic Language. Technical Report CLIP15/95.0, Facultad de Informática, UPM, June 1995.

[Car87] M. Carlsson. Freeze, Indexing, and Other Implementation Issues in the Wam. In *Fourth International Conference on Logic Programming*, pages 40–58. University of Melbourne, MIT Press, May 1987.

[CC77] P. Cousot and R. Cousot. Abstract Interpretation: a Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *Fourth ACM Symposium on Principles of Programming Languages*, pages 238–252, 1977.

[CC79] P. Cousot and R. Cousot. Systematic Design of Program Analysis Frameworks. In *Sixth ACM Symposium on Principles of Programming Languages*, pages 269–282, San Antonio, Texas, 1979.

[CC92] P. Cousot and R. Cousot. Abstract Interpretation and Application to Logic Programs. *Journal of Logic Programming*, 13(2 and 3):103–179, July 1992.

[CC94] J. Chassin and P. Codognet. Parallel Logic Programming Systems. *Computing Surveys*, 26(3):295–336, September 1994.

[CGH93] M. Carro, L. Gómez, and M. Hermenegildo. Some Paradigms for Visualizing Parallel Execution of Logic Programs. In *1993 International Conference on Logic Programming*, pages 184–201. MIT Press, June 1993.

[CH95] D. Cabeza and M. 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.clip.dia.fi.upm.es/`.

[CH96a] D. Cabeza and M. Hermenegildo. Implementing Distributed Concurrent Constraint Execution in the CIAO System. In *Proc. of the 1996 APPIA-GULP-PRODE'96 Joint conference on Declarative Programming*, San Sebastian, Spain, July 1996. U. of the Basque Country. Available from `http://www.clip.dia.fi.upm.es/`.

[CH96b] Daniel Cabeza and Manuel Hermenegildo. *html.pl: An HTML Package for (C)LP systems.* Spain, March 1996. Available from `http://www.clip.dia.fi.upm.es/miscdocs/`.

[CHV96] D. Cabeza, M. Hermenegildo, and S. Varma. The PiLLoW/CIAO Library for INTERNET/WWW Programming using Computational Logic Systems. In *Proceedings of the 1st Workshop on Logic Programming Tools for INTERNET Applications*, JICSLP"96, Bonn, September 1996. Available from `http://clement.info.umoncton.ca/~lpnet`.

[CMB+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.

[Col82] A. Colmerauer et al. *Prolog II: Reference Manual and Theoretical Model.* Groupe D'intelligence Artificielle, Faculté Des Sciences De Luminy, Marseille, 1982.

[CRH94] B. Le Charlier, S. Rossi, and P. Van Hentenryck. An Abstract Interpretation Framework Which Accurately Handles Prolog Search–Rule and the Cut. In *International Symposium on Logic Programming*, pages 157–171. MIT Press, November 1994.

[Deb92] S. Debray, editor. *Journal of Logic Programming, Special Issue: Abstract Interpretation*, volume 13(1–2). North-Holland, July 1992.

[Deb93] S. K. Debray. Implementing logic programming systems: The quiche-eating approach. In *ICLP '93 Workshop on Practical Implementations and Systems Experience in Logic Programming*, Budapest, Hungary, June 1993.

[DGB94] S. Debray, D. Gudeman, and P. Bigot. Detection and Optimization of Suspension-free Logic Programs. In *1994 International Symposium on Logic Programming*, pages 487–501. MIT Press, November 1994.

[DGHL94] S.K. Debray, P. López García, M. Hermenegildo, and N.W. Lin. Estimating the Computational Cost of Logic Programs. In Springer-Verlag, editor, *Static Analysis Symposium, SAS'94*, number 864 in LNCS, pages 255–265, Namur, Belgium, September 1994.

[DJ94] V. Dumortier and G. Janssens. Towards a practical full mode inference system for CLP(H,N). In Pascal Van Hentenryck, editor, *Proceedings of the 11th International Conference on Logic Programming*, pages 569–583, Italy, June 1994. MIT Press.

[DJBC93] V. Dumortier, G. Janssens, M. Bruynooghe, and M. Codish. Freeness analysis in the presence of numerical constraints. In David S. Warren, editor, *Proceedings of the 10th International Conference on Logic Programming*, pages 100–115, Budapest, Hungary, June 1993. MIT Press.

[DLH90] 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.

[dMSC93] Vítor Manuel de Morais Santos Costa. *Compile-Time Analysis for the Parallel Execution of Logic Programs in Andorra-I.* PhD thesis, University of Bristol, August 1993.

[Dum94] Veroniek Dumortier. *Freeness and Related Analyses of Constraint Logic Programs using Abstract Interpretation.* PhD thesis, K.U.Leuven, Dept. of Computer Science, October 1994.

[Eur93] European Computer Research Center. *Eclipse User's Guide*, 1993.

[FCH96] M.J. Fernández, M. Carro, and M. Hermenegildo. IDRA (IDeal Resource Allocation): Computing Ideal Speedups in Parallel Logic Programming. In *Proceedings of EuroPar'96*, LNCS. Springer-Verlag, August 1996.

[Gar94] María José García de la Banda García. *Independence, Global Analysis, and Parallelism in Dynamically Scheduled Constraint Logic Programming.* PhD thesis, Universidad Politécnica de Madrid (UPM), July 1994.

[GBH96] M. García de la Banda, F. Bueno, and M. Hermenegildo. Towards independent And-Parallelism in CLP. In *International Symposium on Programming Language Implementation and Logic Programming, PLILP'96*, LNCS. Springer Verlag, September 1996.

[GC92] G. Gupta and V. Santos Costa. And-Or Parallelism in Full Prolog based on Paged Binding Array. In *Parallel Architectures and Languages Europe '92.* Springer Verlag, June 1992.

[GH93] M. García de la Banda and M. Hermenegildo. A Practical Approach to the Global Analysis of CLP Programs. In Dale Miller, editor, *Proceedings of the 10th International Logic Programming Symposium*, pages 437–455, Vancouver, Canada, October 1993. MIT Press.

[GHM93] M. García de la Banda, M. Hermenegildo, and K. Marriott. Independence in Constraint Logic Programs. In *1993 International Logic Programming Symposium*, pages 130–146. MIT Press, Cambridge, MA, October 1993.

[GHM95] M. García de la Banda, M. Hermenegildo, and K. Marriott. Independence and Search Space Preservation in Dynamically Scheduled Constraint Logic

Languages. Technical Report CLIP10/95.0, Facultad de Informática, UPM, February 1995.

[GHPC94] G. Gupta, M. Hermenegildo, E. Pontelli, and V. Santos Costa. ACE: And/Or-parallel Copying-based Execution of Logic Programs. In *International Conference on Logic Programming*, pages 93–110. MIT Press, June 1994.

[GMS95] M. García de la Banda, K. Marriott, and P. Stuckey. Efficient Analysis of Constraint Logic Programs with Dynamic Scheduling. In *1995 International Logic Programming Symposium*, Portland, Oregon, December 1995. MIT Press, Cambridge, MA.

[GSCYH91] G. Gupta, V. Santos-Costa, R. Yang, and M. Hermenegildo. IDIOM: Integrating Dependent and-, Independent and-, and Or-parallelism. In *1991 International Logic Programming Symposium*, pages 152–166. MIT Press, October 1991.

[HCC95] M. Hermenegildo, D. Cabeza, and M. Carro. Using Attributed Variables in the Implementation of Concurrent and Parallel Logic Programming Systems. In *Proc. of the Twelfth International Conference on Logic Programming*, pages 631–645. MIT Press, June 1995.

[Her] M. Hermenegildo et al. The CIAO Multi-Dialect Compiler and System: An Experimentation Workbench for Future (C)LP Systems. Available from `ftp://www.clip.dia.fi.upm.es/pub/papers`.

[Her86] M. V. 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.

[HG91] M. Hermenegildo and K. Greene. The &-Prolog System: Exploiting Independent And-Parallelism. *New Generation Computing*, 9(3,4):233–257, 1991.

[HMPS95] M. Hermenegildo, K. Marriott, G. Puebla, and P. Stuckey. Incremental Analysis of Logic Programs. In *International Conference on Logic Programming*, pages 797–811. MIT Press, June 1995.

[Hol90] C. Holzbaur. *Specification of Constraint Based Inference Mechanisms through Extended Unification.* PhD thesis, University of Vienna, 1990.

[Hol92] C. Holzbaur. Metastructures vs. Attributed Variables in the Context of Extensible Unification. In *1992 International Symposium on Programming Language Implementation and Logic Programming*, pages 260–268. LNCS631, Springer Verlag, August 1992.

[HR95] 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.

[HtCg93] M. Hermenegildo and the CLIP group. Towards CIAO-Prolog – A Parallel Concurrent Constraint System. In *Proc. of the Compulog Net Area Workshop on Parallelism and Implementation Technologies*. Technical University of Madrid, June 1993.

[HtCg94] 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*, LNCS 874, pages 123–133. Springer-Verlag, May 1994.

[Hui90] S. Le Huitouze. A New Data Structure for Implementing Extensions to Prolog. In P. Deransart and J. Małuszyński, editors, *Proceedings of Programming Language Implementation and Logic Programming*, number 456 in Lecture Notes in Computer Science, pages 136–150. Springer, August 1990.

[HWD92] M. Hermenegildo, R. Warren, and S. Debray. Global Flow Analysis as a Practical Compilation Tool. *Journal of Logic Programming*, 13(4):349–367, August 1992.

[JH91] S. Janson and S. Haridi. Programming Paradigms of the Andorra Kernel Language. In *1991 International Logic Programming Symposium*, pages 167–183. MIT Press, 1991.

[JL89] D. Jacobs and A. Langen. Accurate and Efficient Approximation of Variable Aliasing in Logic Programs. In *1989 North American Conference on Logic Programming*. MIT Press, October 1989.

[JM94] J. Jaffar and M.J. Maher. Constraint Logic Programming: A Survey. *Journal of Logic Programming*, 13/20:503–581, 1994.

[LH95] P. López and M. Hermenegildo. Efficient Term Size Computation for Granularity Control. In *Proc. of the Twelfth International Conference on Logic Programming*, pages 647–661. The MIT Press, June 1995.

[LHD94] P. López García, M. Hermenegildo, and S.K. Debray. Towards Granularity Based Control of Parallelism in Logic Programs. In *Proc. of First International Symposium on Parallel Symbolic Computation, PASCO'94*, pages 133–144. World Scientific Publishing Company, September 1994.

[MGH94] 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.

[MH89] K. Muthukumar and M. Hermenegildo. Determination of Variable Dependence Information at Compile-Time Through Abstract Interpretation. In *1989 North American Conference on Logic Programming*, pages 166–189. MIT Press, October 1989.

[MH90] K. Muthukumar and M. Hermenegildo. Deriving A Fixpoint Computation Algorithm for Top-down Abstract Interpretation of Logic Programs. Technical Report ACT-DC-153-90, Microelectronics and Computer Technology Corporation (MCC), Austin, TX 78759, April 1990.

[MH91] K. Muthukumar and M. Hermenegildo. Combined Determination of Sharing and Freeness of Program Variables Through Abstract Interpretation. In *1991 International Conference on Logic Programming*, pages 49–63. MIT Press, June 1991.

[MH92] K. Muthukumar and M. Hermenegildo. Compile-time Derivation of Variable Dependency Using Abstract Interpretation. *Journal of Logic Programming*, 13(2 and 3):315–347, July 1992.

[MS94] K. Marriott and P. Stuckey. Approximating Interaction Between Linear Arithmetic Constraints. In *1994 International Symposium on Logic Programming*, pages 571–585. MIT Press, 1994.

[Neu90] U. Neumerkel. Extensible Unification by Metastructures. In *Proceeding of the META'90 workshop*, 1990.

[PH95a] G. Puebla and M. Hermenegildo. Implementation of Multiple Specialization in Logic Programs. In *Proc. ACM SIGPLAN Symposium on Partial Evaluation and Semantics Based Program Manipulation*. ACM, June 1995.

[PH95b] G. Puebla and M. Hermenegildo. Specialization and Optimization of Constraint Programs with Dynamic Scheduling. Technical Report CLIP12/95.0, Facultad de Informática, UPM, September 1995. Presented at the 1995 COMPULOG Meeting on Program Development.

[SCWY90] V. Santos-Costa, D.H.D. Warren, and R. Yang. Andorra-I: A Parallel Prolog System that Transparently Exploits both And- and Or-parallelism. In *Proceedings of the 3rd. ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. ACM, April 1990.

[Son86] H. Sondergaard. An application of abstract interpretation of logic programs: occur check reduction. In *European Symposium on Programming, LNCS 123*, pages 327–338. Springer-Verlag, 1986.

[Swe95] Swedish Institute of Computer Science, P.O. Box 1263, S-16313 Spanga, Sweden. *Sicstus Prolog V3.0 User's Manual*, 1995.

[Tic88] E. Tick. Compile-Time Granularity Analysis of Parallel Logic Programming Languages. In *Int. Conf. on FGCS*. Tokyo, November 1988.

[Tic95] E. Tick. The Deevolution of Concurrent Logic Programming Languages. *The Journal of Logic Programming*, 23(1–3):89–125, 1995.