40 Years of Adventure in Abstract Interpretation of Logic Programs

Manuel Hermenegildo

with lots of help over the years from (among others):

D. Cabeza, E. Albert, P. Arenas, F. Bueno Carrillo, M. Carro, A. Casas, M. Codish, J. Correas, S.K. Debray, M. García de la Banda, S. Genaim, N.-W. Lin, P. López-García, K. Marriott, M. Marron, E. Mera, J. Morales, K. Muthukumar, M. Méndez-Lojo, J. Navas, P. Pietrzak, G. Puebla, F. Rossi, P. Stuckey, C. Vaucheret, R. Warren, D. Zanardini

CS Dept., T.U. Madrid, Spain
CS Dept., U. of Texas at Austin, USA
CS Dept., Complutense U. Madrid, Spain
CS and EECE Depts., U. of New Mexico, USA
Microel. and Computer Tech. Corp. (MCC), USA
IMDEA SW Development Technology Institute, Spain

~~40~~ 25 Years of Adventure in Abstract Interpretation of Logic (and Imperative) Programs

Manuel Hermenegildo

with lots of help over the years from (among others):

D. Cabeza, E. Albert, P. Arenas, F. Bueno Carrillo, M. Carro, A. Casas, M. Codish, J. Correas, S.K. Debray, M. García de la Banda, S. Genaim, N.-W. Lin, P. López-García, K. Marriott, M. Marron, E. Mera, J. Morales, K. Muthukumar, M. Méndez-Lojo, J. Navas, P. Pietrzak, G. Puebla, F. Rossi, P. Stuckey, C. Vaucheret, R. Warren, D. Zanardini

CS Dept., T.U. Madrid, Spain
CS Dept., U. of Texas at Austin, USA
CS Dept., Complutense U. Madrid, Spain
CS and EECE Depts., U. of New Mexico, USA
Microel. and Computer Tech. Corp. (MCC), USA
IMDEA SW Development Technology Institute, Spain

~~40~~ 25 Years of Adventure in Abstract Interpretation of Logic (and Imperative) Programs

Or: *From Parallelism to Program Development and Back*

Manuel Hermenegildo

with lots of help over the years from (among others):

D. Cabeza, E. Albert, P. Arenas, F. Bueno Carrillo, M. Carro, A. Casas, M. Codish, J. Correas, S.K. Debray, M. García de la Banda, S. Genaim, N.-W. Lin, P. López-García, K. Marriott, M. Marron, E. Mera, J. Morales, K. Muthukumar, M. Méndez-Lojo, J. Navas, P. Pietrzak, G. Puebla, F. Rossi, P. Stuckey, C. Vaucheret, R. Warren, D. Zanardini

CS Dept., T.U. Madrid, Spain
CS Dept., U. of Texas at Austin, USA
CS Dept., Complutense U. Madrid, Spain
CS and EECE Depts., U. of New Mexico, USA
Microel. and Computer Tech. Corp. (MCC), USA
IMDEA SW Development Technology Institute, Spain

The Paper

Non-Strict Independence-Based
Program Parallelization
Using Sharing and Freeness Information

Daniel Cabeza and Manuel Hermenegildo

- Abstract Interpretation, Logic Programming, Parallelism.
- Done within the ParForCE project, time of great Pisa-Madrid collaboration.
- An attempt to complete some missing parts on this (now old, but relevant) topic.

Provide context: what happened before and after.
~

The Original Challenge / Early Steps: Parallelization and Abs. Int.

- The original problem: LP parallelization (circa 1983 in US: UT Austin, MCC):
  - Parallel abstract machine, &-Prolog [ICLP'86, ICLP'87, ...] $\Rightarrow$ real speedups!
  - Detecting dependencies (pointer "sharing") among proc. calls, statements, etc.

  Traditional approach (ad-hoc dataflow analysis [Chang, Despain, Degroot '85]): correctness? — we wanted something *rigorous and more powerful*.

The Original Challenge / Early Steps: Parallelization and Abs. Int.

- The original problem: LP parallelization (circa 1983 in US: UT Austin, MCC):
  - Parallel abstract machine, &-Prolog [ICLP'86, ICLP'87, ...] ⇒ real speedups!
  - Detecting dependencies (pointer "sharing") among proc. calls, statements, etc.

  Traditional approach (ad-hoc dataflow analysis [Chang, Despain, Degroot '85]): correctness? — we wanted something *rigorous and more powerful*.
- Our first "obsession:" correctness/practicality –speedups possible? AI practical? Built system [Warren, Debray, Herme. "MA3 system" ICSLP'88]
- Bruynooghe's framework [87-91]: multivariance + genericity of framework –parametric on the domains– (but no tabling or efficient fixpoint).
- The fixpoint algorithm and PLAI analyzer: efficient, context sensitive, multivariant, parametric analysis! [Muthukumar, Herme. NACLP'89]
  - Tabling, multiple call–success pairs Dependency tracking Interprocedural, dealing with mutual recursion, etc. (project/extend).
- Many useful extensions (1990 on, Spain, mostly at UPM; also UNM): Incremental framework; CLP; concurrent programs; Extension to Java/Java bytecode

Some Achievements in Parallelization

- Fully automatic parallelizers for logic programs
  – arguably the most powerful parallelizers for "irregular" applications.
  [Herme. and Warren, CAN'87, Bueno, G. de la Banda, and Herme. ICLP'94, TOPLAS'99]
  - Parallelization using non-strict independence [Cabeza and Herme. SAS'94]
  - Parallelization of constraint programs [G. de la Banda and Herme. PLILP'96]

  Perhaps the first fully implemented, practical application of AI?
- Prompted considerable *abstract domain development*:
  - Set sharing. [Jacobs-Langen NACLP 89, Muthukumar and Herme. NACLP'89 (abstr. unif.)]
  - Set sharing and freeness. [Muthukumar and Herme. ICLP'91]
  - Def (propositional horn clauses) [de la Banda/Herme. WSA'92, Sondergaard, Marriott]
  - Combinations with depth-k, shape analysis (regular types), etc.
  - *Set sharing for Java* [Méndez-Lojo, Herme. VMCAI'08]
- (C)LP/Prolog/Ciao very useful: allow studying challenging problems (pointers, irregular computations, irregular data, dynamic heap, dynamic dispatch, etc.) in much better context.

Some Achievements in Parallelization

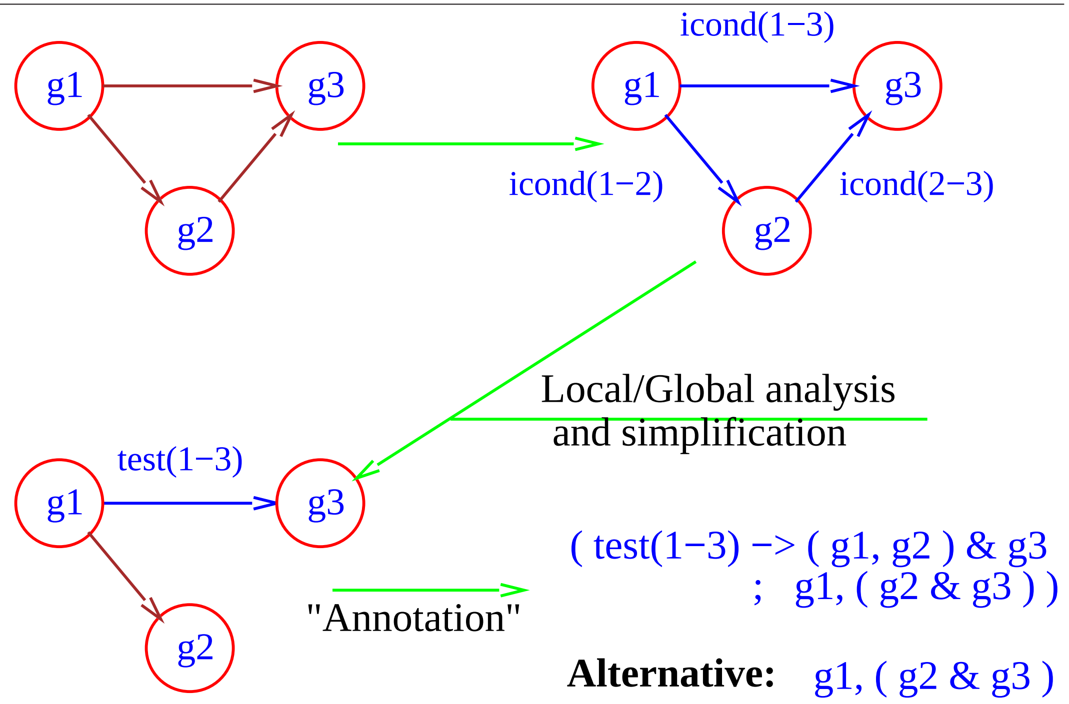
- Fully automatic parallelizers for logic programs
  – arguably the most powerful parallelizers for “irregular” applications.
  [Herme. and Warren, CAN’87, Bueno, G. de la Banda, and Herme. ICLP’94, TOPLAS’99]
  - Parallelization using non-strict independence [Cabeza and Herme. SAS’94]
  - Parallelization of constraint programs [G. de la Banda and Herme. PLILP’96]

  Perhaps the first fully implemented, practical application of AI?
- Prompted considerable *abstract domain development*:
  - Set sharing. [Jacobs-Langen NACLP 89, Muthukumar and Herme. NACLP’89 (abstr. unif.)]
  - Set sharing and freeness. [Muthukumar and Herme. ICLP’91]
  - Def (propositional horn clauses) [de la Banda/Herme. WSA’92, Sondergaard, Marriott]
  - Combinations with depth-k, shape analysis (regular types), etc.
  - *Set sharing for Java* [Méndez-Lojo, Herme. VMCAI’08]
- (C)LP/Prolog/Ciao very useful: allow studying challenging problems (pointers, irregular computations, irregular data, dynamic heap, dynamic dispatch, etc.) in much better context.
- **ParForCE and other projects –much collaboration with Giorgio / Pisa!**

Parallelization Process

- Conditional dependency graph (of some segment, e.g., a clause):
  - vertices are possible tasks (statements, calls,...),
  - edges=possible dependency (labels=conditions needed for independence).
- Local or global analysis used to reduce/remove checks in the edges.

```
foo(...)  :-
     g1(...),
     g2(...),
     g3(...).
```

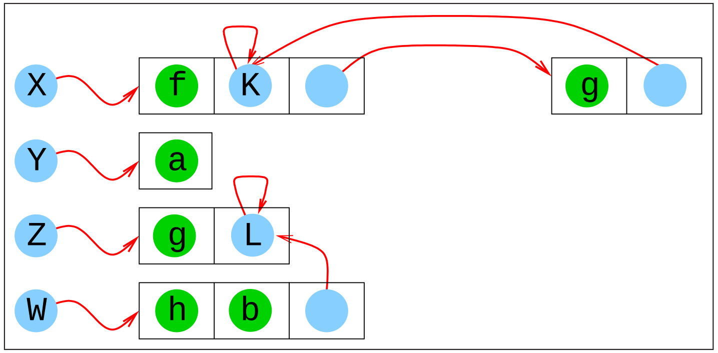


A Priori Independence: Strict Independence

- Correctness and efficiency (search space preservation) guaranteed for `p & q` if no shared variables.

- The “pointers” view:
  correctness and efficiency guaranteed if there are no “pointers” between `p` and `q`.

```
main :- X=f(K,g(K)), Y=a,
        Z=g(L), W=h(b,L),
        --------------------->
        p(X,Y),
        q(Y,Z),
        r(W).
```



`p` and `q` are strictly independent, but `q` and `r` are not.

Independence – Non-Strict Independence

- Pure goals: only one thread “touches” each shared variable. Example:

```
main :- t(X,Y), p(X), q(Y).
t(X,Y) :- Y = f(X).
```

- Impure goals: only rightmost “touches” each shared variable. Example:

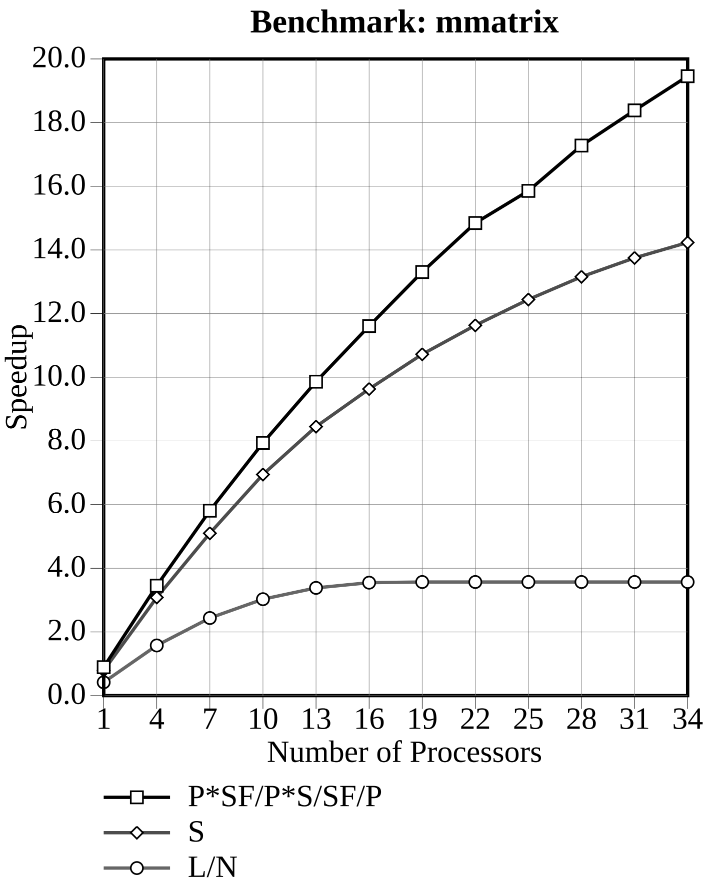
```
main :- t(X,Y), p(X), q(Y).
t(X,Y) :- Y = a.   p(X) :- var(X), ..., X=b, ...
```

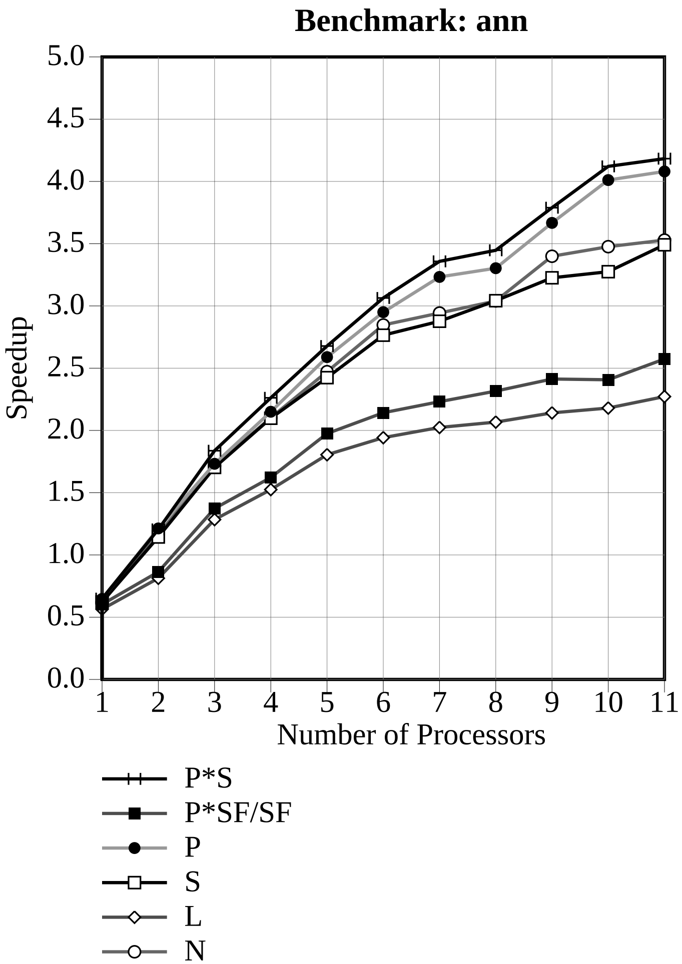
- Very important in programs using “incomplete structures.”

```
flatten(Xs,Ys) :- flatten(Xs,Ys,[]).
flatten([], Xs, Xs).
flatten([X|Xs],Ys,Zs)  :- flatten(X,Ys,Ys1), flatten(Xs,Ys1,Zs).
flatten(X, [X|Xs], Xs) :- atomic(X), X \== [].
```

- Another example: qsort with difference lists.
- **Paper:** NSI from sharing+freeness; new run-time tests (`allvars`, `sharedvars`).

Some Speedups (Using Different Abstract Domains)



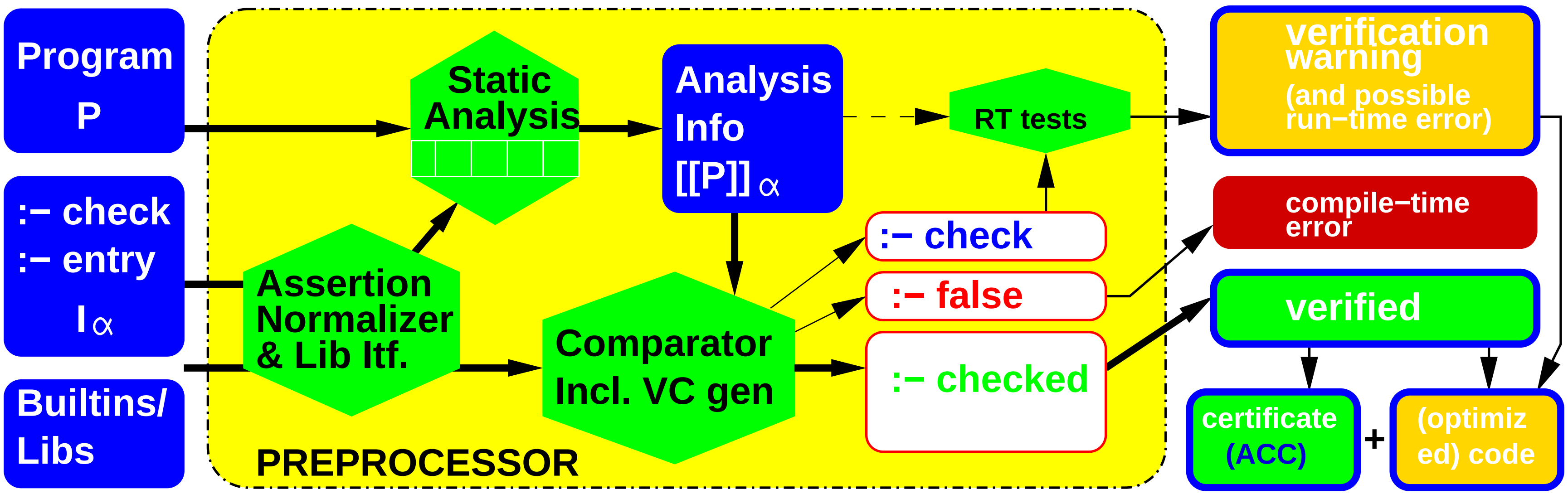


(ann is the parallelizer parallelizing itself;
1-10 proc.: actual speedups on Sequent Symmetry; 10+ simulator projections from execution traces)

The Second Phase: Program Development using AI / Next Gen. LP

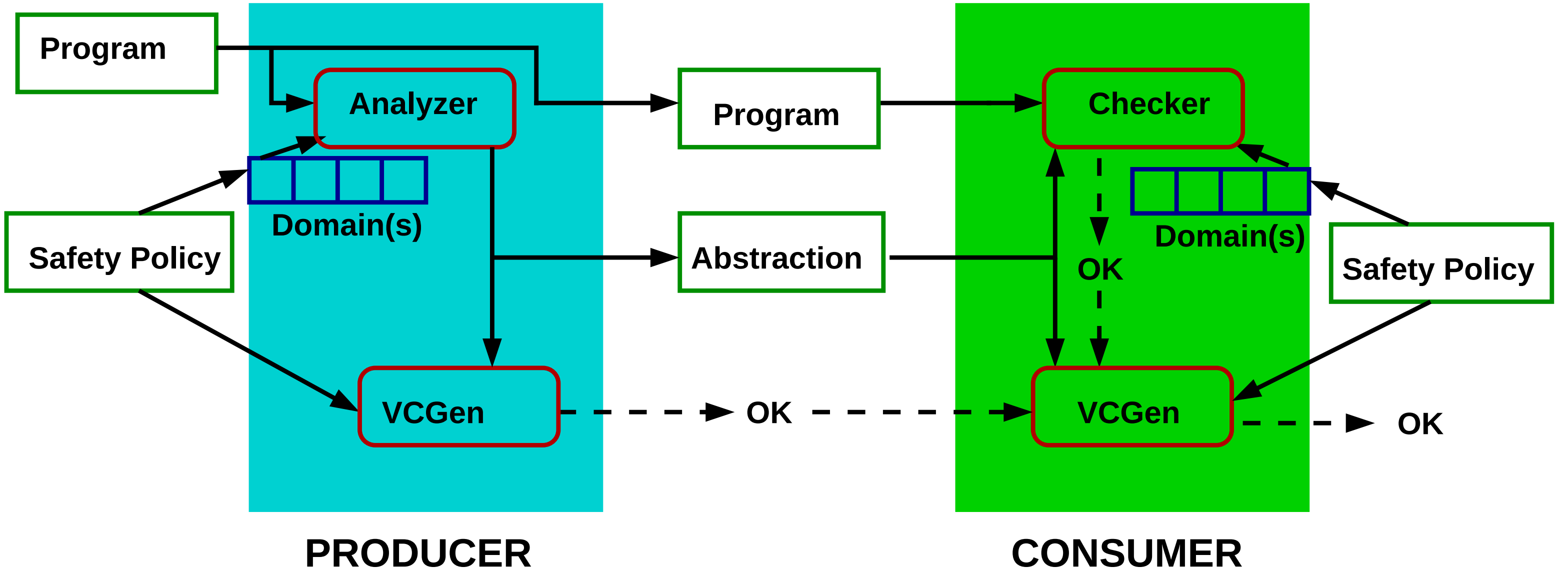
- The next stage:
  - not just optimization (parallelization, abstract partial evaluation, ...), but
  - **program development** fundamentally based on abstract interpretation.
- Great influence of AI and LP in **programming language design**:
  the “Ciao” language and environment design;
  AI-based compiler / env. (assertion based vs. strongly-typed, type systems as domains, multiparadigm –again, Giorgio–, etc.).
- Ideas lurking for a long time, crystallized in first prototypes in 1994-97 [PPCP’94, Ciao system design], and:
  - Overall design [ILPS’97, AADEBUG’97, ICLP’99, LOPSTR’99, SAS’03, SCP’05]
  - Assertion language [LNCS 1870]
  - Modular extensions [Pietrzak, Correas, Puebla, Herme. LPAR’06]
  - Abstract diagnosis [Alpuente, Comini, Escobar, Falaschi, Lucas, LOPSTR’02] [Gallardo, Merino, Pimemtel, SAS’02] [Pietrzak, Herme. ICLP’07]
  - Extensions to Java [Albert, Gómez-Zamalloa, Hubert, Puebla PADL’07]

The CiaoPP Verification/Diagnosis Framework [AADEBUG'97]



- I_α (partial specification) provided via a language of *optional* assertions.
  - State properties at relevant points.
  - Talk about properties which can be predefined or user-defined.
    Types, cost, data sizes, termination, pointer aliasing, no-except, ...
- Implemented for Ciao and (less mature) for Java and Java bytecode.

The Abstraction Carrying Code (ACC) Scheme

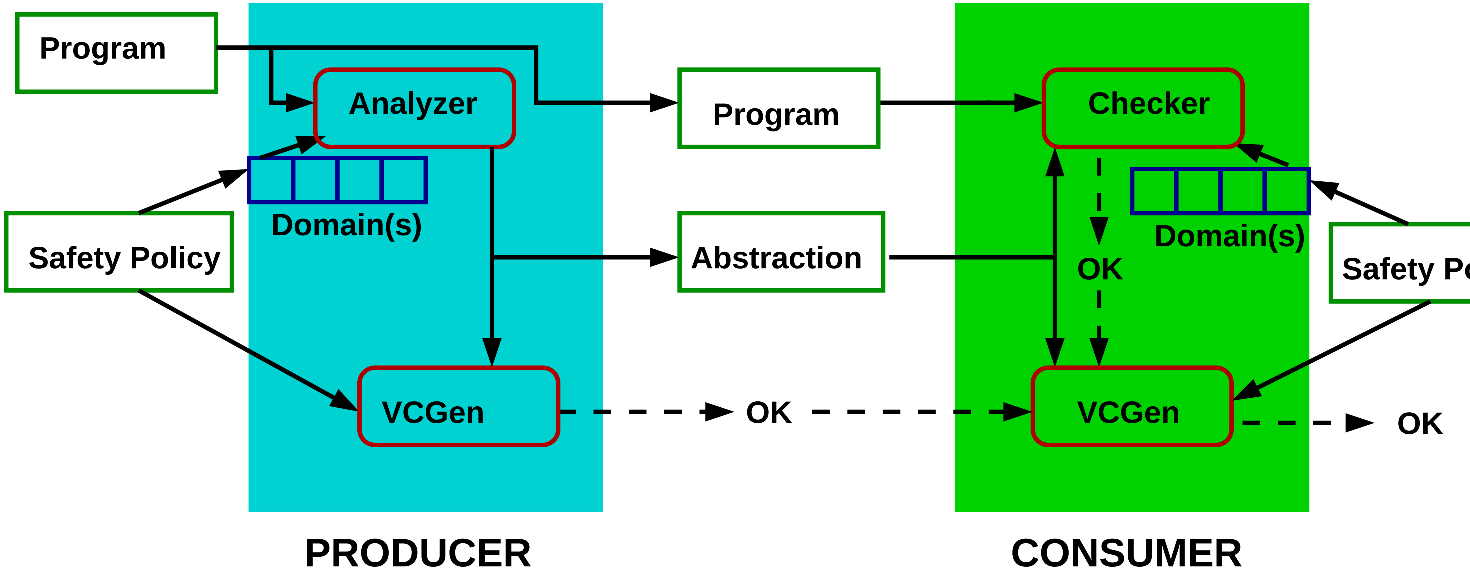


$[\![P]\!]_\alpha = \text{Analysis} = \text{lfp}(analysis\_step)$

$\text{Certificate} \subset [\![P]\!]_\alpha$

$\text{Checker} = analysis\_step$

The Abstraction Carrying Code (ACC) Scheme



$[\![P]\!]_\alpha$ = Analysis = lfp($analysis\_step$) | Certificate $\subset [\![P]\!]_\alpha$ | Checker = $analysis\_step$

- Basic proposal [Albert, Puebla, Herme. COCV'04, LPAR'04, ICLP'04, PPDP'05, NGC'08]
- Incremental version [Albert, Arenas, Puebla LPAR'06]
- Certificate reduction (fixpoint compression) [Albert, Arenas, Puebla, Herme. ICLP'06]
- MOBIUS project [TGC'06].
- Incorporated in `CiaoPP` for types, modes, shapes, sizes, cost, det, termination, non-failure, resources, etc.

Combination with Partial Evaluation; Optimization; Scalability

- Combination with Partial Evaluation:
  - Abstract executability. Multiple abstract specialization [Puebla, Herme. PEPM'95]; application in parallelization [LOPSTR'97, JLP'99]
  - Full integration of Partial Evaluation and AI [Puebla, Albert, Herme. SAS'06]

  → important contribution to the AI-based program development model.
- Application to program optimization (other than parallelization): abstract machines and native compilation [Morales, Carro, Herme. PADL'04, ICLP'05, LOPSTR'06]; embedded systems [Carro, Morales, Muller, Puebla, Herme. CASES'06].
- Modularity, Scalability, Practicality Issues:
  - Efficient, incremental intermodular analysis / fixpoint calculation [Bueno, G. de la Banda, Herme., Marriott, Puebla, Stuckey, Correas LOPSTR'01, LOPSTR'04, LOPSTR'05].
  - Widening of set-sharing [Navas, Bueno, Herme. PADL'06]
  - Dealing with the full ISO-Prolog standard [Bueno, Cabeza, Herme., Puebla ESOP'96]
- Domain combinations, goal-dependent vs. goal indep. analysis (flow sensitivity) [Codish, Mulkers, Bruy., G. de la Banda, Herme. PEPM'93, LPAR'94, TOPLAS'95, JLP'97]

Inference of Resource-related and other Complex Properties

- Cost, resources (motivated by granularity control –lots of AI!):
  - Upper bounds (execution steps); application to granularity control [Debray, Lin, Herme., PLDI'90].
  - Lower bounds (execution steps) [w/López-García, ILPS'97, SAS'94].
  - Extension to Java bytecode [Albert, Arenas, Genaim, Puebla, Zanardini ESOP'07].
  - Execution times [Mera, López-García, Puebla, Carro, Herme., PADL'07].
  - User-definable resources for LP [Navas, Mera, López-García, Herme., ICLP'07].
  - Heap space analysis (Java bytecode) [Albert, Genaim, Gómez-Zamalloa, ISMM '07].
- Non-failure / no exceptions [Bueno, Debray, López-García, Herme. ICLP'97]; multivariant version [FLOPS'04].
- Determinacy analysis [Bueno, López-García, Herme. LOPSTR'04].
- Heap / shape analysis (C++/Java) [Marron, Herme., Kapur, Stefanovic LCPC'06, PASTE'07, CC'08]; Type domains, type widenings [Vaucheret, Bueno SAS'02].
- Termination analysis (Java) [Albert, Arenas, Codish, Genaim, Puebla, Zanardini WST'07].

Final Thoughts / The Next 25 Years!

- Go mainstream, make everyday tools! (Ciao, COSTA, ...) –coll. w/Industry (e.g., ES_PASS).
  - Further work on scalability, modularity, domains, widenings, ...
  - Emphasis on resource-related properties; specially *user-defined*.
  - Improve diagnosis.
  - Develop ACC further, apply in practice to, e.g., small devices (MOBIUS).
  - Multi-language environments.
- *Back to parallelism! A big push to parallelization, granularity control, ...*
- *Develop new, even more dynamic, multiparadigm, high-level languages, getting correctness and real speed (without user burden), thanks to LP and AI.*

Final Thoughts / The Next 25 Years!

- Go mainstream, make everyday tools! (Ciao, COSTA, ...) –coll. w/Industry (e.g., ES_PASS).
  - Further work on scalability, modularity, domains, widenings, ...
  - Emphasis on resource-related properties; specially *user-defined*.
  - Improve diagnosis.
  - Develop ACC further, apply in practice to, e.g., small devices (MOBIUS).
  - Multi-language environments.
- *Back to parallelism! A big push to parallelization, granularity control, ...*
- *Develop new, even more dynamic, multiparadigm, high-level languages, getting correctness and real speed (without user burden), thanks to LP and AI.*

But, mainly, thanks Giorgio –what an adventure!
Look forward to the next 25 years together!

Conditions for Non-Strict Independence Based on ShFr Info

- We consider the parallelization of pairs of goals.
- Let the situation be: $\{\widehat{\beta}\}\ p\ \{\widehat{\psi}\}\ \ldots\ q$.
  We define:

  $$\mathrm{S}(p) = \{L \in \widehat{\beta}_{\mathrm{SH}} \mid L \cap (\neq \emptyset\}$$
  $$\mathrm{SH} = \mathrm{S}(p) \cap \mathrm{S}(q) = \{L \in \widehat{\beta}_{\mathrm{SH}} \mid L \cap (\neq \emptyset$$
  $$\wedge L \cap (\neq \emptyset\}$$

- Conditions for non-strict independence for p and q:

  C1 $\forall L \in \mathrm{SH}\ L \cap \widehat{\psi}_{\mathrm{FR}} \neq \emptyset$

  C2 $\neg\,(\exists N_1 \ldots N_k \in \mathrm{S}(p)\ \exists L \in \widehat{\psi}_{\mathrm{SH}}$
  $L = \cup_{i=1}^{k} N_i \ \wedge\ N_1, N_2 \in \mathrm{SH}$
  $\wedge \forall i, j\ \ 1 \leq i < j \leq k\ \ N_i \cap N_j \cap \widehat{\beta}_{\mathrm{FR}} = \emptyset)$

- C1: preserves freeness of shared variables.
- C2: preserves independence of shared variables.
- More relaxed conditions if information re. partial answers and purity of goals.

Run-Time Checks for NSI Based on ShFr Info

- Run-time checks can be automatically included to ensure NSI when the previous conditions do not hold.
- The method uses analysis information.
- Possible checks are:
  - `ground(X)`: `X` is ground.
  - `allvars(X,`$\mathcal{F}$`)`: every free variable in `X` is in the list $\mathcal{F}$.
  - `indep(X,Y)`: `X` and `Y` do not share variables.
  - `sharedvars(X,Y,`$\mathcal{F}$`)`: every free variable shared by X and Y is in the list $\mathcal{F}$.
- The method generalizes the techniques previously proposed for detection of SI.
- Even when only SI is present, the tests generated may be better than the traditional tests.