Energy Analysis, Debugging, and Verification
(with the CiaoPP system)

Manuel Hermenegildo [1,2]   P. López-García[1,3]

R. Haemmerlé[1]   M. Klemen[1]   U. Liqat[1]   J. Morales[1]

[1]IMDEA Software Institute

[2]Technical University of Madrid (UPM)

[3]Spanish Research Council (CSIC)

In collaboration with:

[4]U. Bristol, XMOS, Roskilde U.

K. Georgiou[4], N. Grech[4], S. Kerrison[4], and K. Eder[4]

EACO Workshop, Bristol, September 10-11, 2014

Analysis/Debugging/Verification of Resources

Automatically infer upper/lower bounds on the usage that a program makes of a general notion of various (*user-definable*) resources.

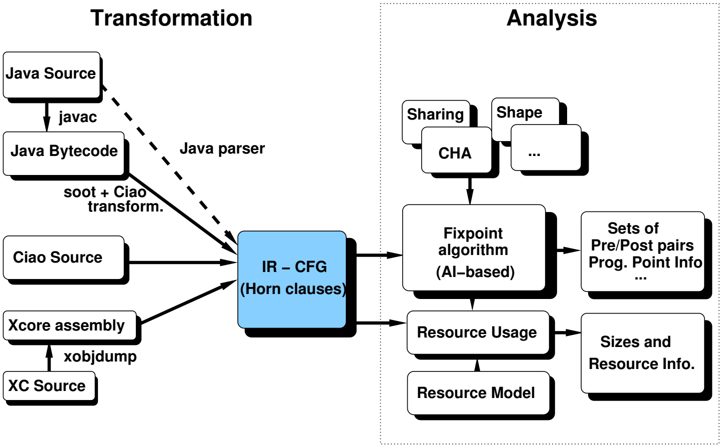
- Memory, execution time, execution steps, data sizes, ...
- Bits sent or received over a socket, SMSs sent or received, accesses to a database, calls to a procedure, files left open, money spent, ...
- **Energy consumed**, . . .

- Key observations about resource consumption:
  - Imprecision → infer **bounds** that are safe and as accurate as possible.
  - Dependent on input data metrics → infer the bounds **as functions**. (Difference with WCET and related methods.)
- Developed methodology and tool (CiaoPP) for such analysis where:
  1. Programmer defines the resource consumption for basic elements (e.g., instructions, bytecodes, libraries, ...).
  2. System infers resource usage bound functions for rest of program.
- Applications: performance debugging and verification, resource-oriented optimization, *granularity control in parallelism*, . . .

[DLH90, DLGHL94, DLGHL97, NMLGH07, MLGCH08, NMLH08, NMLH09, LGDB10, SLBH13, LKSGL13, SLH14]

Intermediate Representation:

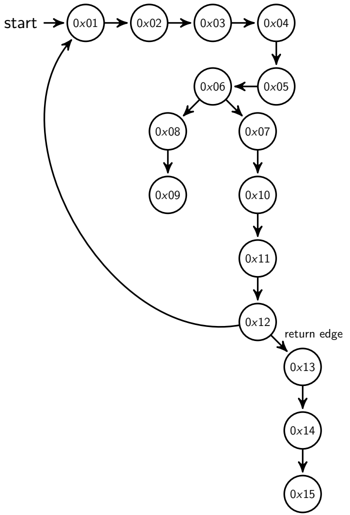
[MLNH07]



- Allows supporting multiple languages.
- Block-based CFG. Each block represented as a *Horn clause*.
- Used for all analyses: aliasing, CHA/shape/types, data sizes, resources, etc.
- Analysis is *parametric* w.r.t. languages, abstractions, resources, etc.

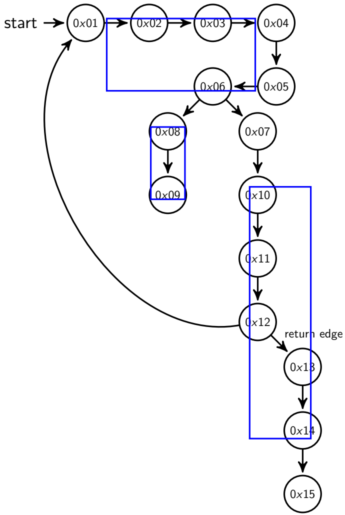
Xcore Example: Control Flow Graph (CFG)

```
<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
```



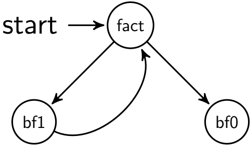
Xcore Example: Block Representation

```
<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
```



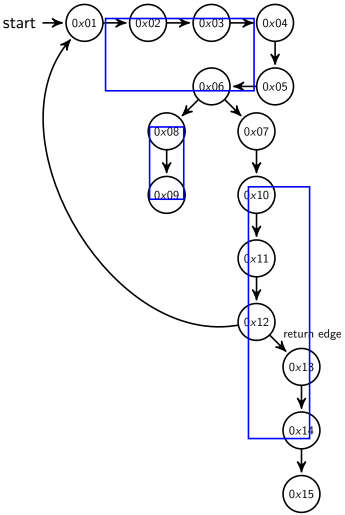
Xcore Example: Block Representation

```
 fact :-
0x01: entsp(0x2)
0x02: stw(r0, sp[0x1])
0x03: ldw(r1, sp[0x1])
0x04: ldc(r0, 0x0)
0x05: lss(r0, r0, r1)
0x06: bf(r0, 0x1 <0x08>)
 branch(bf0, bf1)

 bf1 :-
0x07: bu(0x2 <0x10>)
0x10: ldw(r0, sp[0x1])
0x11: sub(r0, r0, 0x1)
0x12: bl(-0xc <fact>)
 call(fact)
0x13: ldw(r1, sp[0x1])
0x14: mul(r0, r1, r0)
0x15: retsp(0x2)

 bf0 :-
0x08: mkmsk(r0, 0x1)
0x09: retsp(0x2)
```



Block Control Flow Graph

Xcore Example: Horn Clause IR

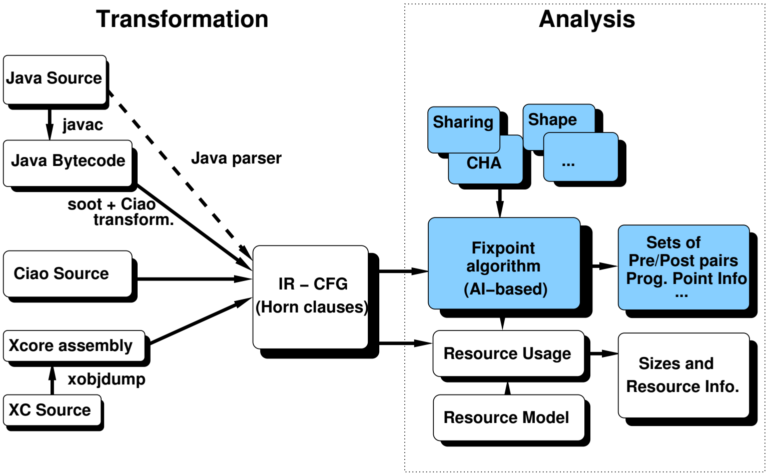



```
:- entry fact/2 : int * var.
fact(R0,R0_3):-
  entsp(_),
  stw(R0,Sp0x1),
  ldw(R1,Sp0x1),
  ldc(R0_1,0x0),
  lss(R0_2,R0_1,R1),
  bf(R0_2,_),
  bf01(R0_2,Sp0x1,R0_3,R1_1).

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(_).
```

Generating the Intermediate Representation

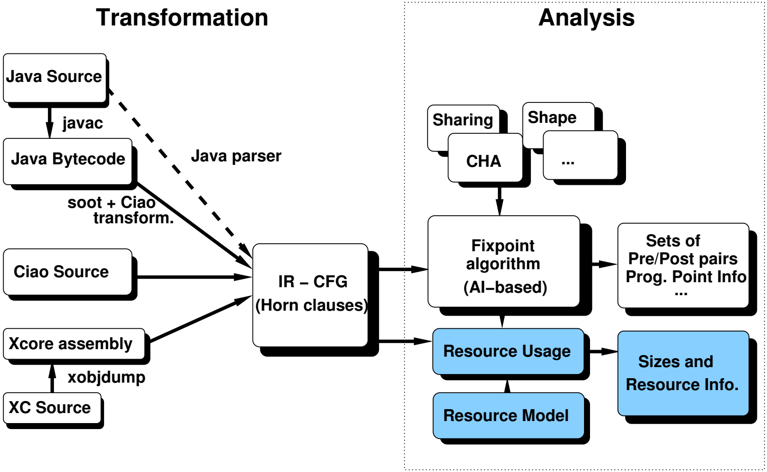
- Specifics for Java:
  - Control flow graph construction from bytecode representation.
  - Elimination of stack variables.
  - Conversion to three-address statements.
  - Explicit representation of this and ret as extra block parameters.
- Specifics for XC:
  - Control flow graph construction from ISA (or LLVM IR) representation.
  - Resolving branching to predicates with multiple clauses.
  - Inferring block parameters.
- Some common 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.
  - Representation as Horn clauses.
- Can be done directly or via partial evaluation of an interpreter (implementing the semantics of the low level code) w.r.t. the concrete low-level program.

Analysis: CiaoPP Parametric AI Framework



[MH92, BGH99, PH96, HPMS00, NMLH07] [MGH94, BCHP96, PH00, BdlBH+01, PCPH06, PCPH08]
[MH89, MH91, DLGH97, VB02, BLGH04, LGBH05, NBH06, MSHK07]
[MLH08, MKSH08, MMLH+08, MHKS08, MKH09, LGBH10, MLLH08]

Resource Analysis



[DLH90, LGHD94, LGHD96, DLGHL94, DLGHL97, NMLGH07, MLNH07, MLGCH08, NMLH08]

[NMLH09, LGDB10, SLBH13, LKSGL13, SLH14]

Resource Analysis

- The objective of the resource analysis is to obtain for each predicate/block *resource usage functions*:

  Monotonic arithmetic functions that return lower/upper bounds on the resource usage of the predicate/block given the sizes of its input data.

Example output

```
true nrev(x) :  list(x) → resource(energy, 0, 1+length(x)**2)
```

- x points to a list → number of execution steps $\leq 1 + \mathit{length}(A)^2$

- To define a new resource and its analysis:
  - User just needs to provide (via assertions) the resource usage functions for relevant primitive operations.
- The system provides libraries of predefined resources.

Overview of the Analysis ("classical" approach)

1. Supporting analyses:
   - Class hierarchy analysis simplifies CFG and improves overall precision. Types/shapes for size metrics.
   - Sharing analysis for correctness (conservative: only when there is no sharing among data structures).
   - *Non-failure* (no exceptions) inferred for non-trivial lower bounds.
   - *Determinacy* (mutual exclusion) to obtain tighter bounds.
2. Set up recurrence equations representing the size of each output argument as a function of the input data sizes.
   - Data dependency graphs determine *relative* sizes of variable contents.
   - Size measures are derived from inferred shape (type) information.
3. Compute upper bounds to the solutions of these recurrence equations to obtain output argument sizes as functions of input sizes.
   - We have an internal recurrence solver, and the system also interfaces with tools like Mathematica, Parma, PUBS, Matlab, etc.

Overview of the Analysis ("classical" approach)

1. Supporting analyses:
   - Class hierarchy analysis simplifies CFG and improves overall precision. Types/shapes for size metrics.
   - Sharing analysis for correctness (conservative: only when there is no sharing among data structures).
   - *Non-failure* (no exceptions) inferred for non-trivial lower bounds.
   - *Determinacy* (mutual exclusion) to obtain tighter bounds.
2. Set up recurrence equations representing the size of each output argument as a function of the input data sizes.
   - Data dependency graphs determine *relative* sizes of variable contents.
   - Size measures are derived from inferred shape (type) information.
3. Compute upper bounds to the solutions of these recurrence equations to obtain output argument sizes as functions of input sizes.
   - We have an internal recurrence solver, and the system also interfaces with tools like Mathematica, Parma, PUBS, Matlab, etc.

Overview of the Analysis ("classical" approach)

1. Supporting analyses:
   - Class hierarchy analysis simplifies CFG and improves overall precision. Types/shapes for size metrics.
   - Sharing analysis for correctness (conservative: only when there is no sharing among data structures).
   - *Non-failure* (no exceptions) inferred for non-trivial lower bounds.
   - *Determinacy* (mutual exclusion) to obtain tighter bounds.
2. Set up recurrence equations representing the size of each output argument as a function of the input data sizes.
   - Data dependency graphs determine *relative* sizes of variable contents.
   - Size measures are derived from inferred shape (type) information.
3. Compute upper bounds to the solutions of these recurrence equations to obtain output argument sizes as functions of input sizes.
   - We have an internal recurrence solver, and the system also interfaces with tools like Mathematica, Parma, PUBS, Matlab, etc.

Overview of the Analysis ("classical" approach)

1. Supporting analyses:
   - Class hierarchy analysis simplifies CFG and improves overall precision. Types/shapes for size metrics.
   - Sharing analysis for correctness (conservative: only when there is no sharing among data structures).
   - *Non-failure* (no exceptions) inferred for non-trivial lower bounds.
   - *Determinacy* (mutual exclusion) to obtain tighter bounds.
2. Set up recurrence equations representing the size of each output argument as a function of the input data sizes.
   - Data dependency graphs determine *relative* sizes of variable contents.
   - Size measures are derived from inferred shape (type) information.
3. Compute upper bounds to the solutions of these recurrence equations to obtain output argument sizes as functions of input sizes.
   - We have an internal recurrence solver, and the system also interfaces with tools like Mathematica, Parma, PUBS, Matlab, etc.
4. E.g.:

```
true conc(x,y) : list * list
               => ( size_lb(ret, length(x)+length(y) ),
                    size_ub(ret, length(x)+length(y) ) )
```

Overview of the Analysis ("classical" approach)

1. Supporting analyses:
   - Class hierarchy analysis simplifies CFG and improves overall precision. Types/shapes for size metrics.
   - Sharing analysis for correctness (conservative: only when there is no sharing among data structures).
   - *Non-failure* (no exceptions) inferred for non-trivial lower bounds.
   - *Determinacy* (mutual exclusion) to obtain tighter bounds.
2. Set up recurrence equations representing the size of each output argument as a function of the input data sizes.
   - Data dependency graphs determine *relative* sizes of variable contents.
   - Size measures are derived from inferred shape (type) information.
3. Compute upper bounds to the solutions of these recurrence equations to obtain output argument sizes as functions of input sizes.
   - We have an internal recurrence solver, and the system also interfaces with tools like Mathematica, Parma, PUBS, Matlab, etc.

5. Use the size information to set up recurrence equations representing the computational cost of each block and compute upper bounds to their solutions to obtain **resource usage functions**.

Some results

Program	Resource	Usage Function	Metrics	Time
`client`	"bits received"	$\lambda x.8 \cdot x$	length	186
`color_map`	"unifications"	39066	size	176
`copy_files`	"files left open"	$\lambda x.x$	length	180
`eight_queen`	"queens movements"	19173961	length	304
`eval_polynom`	"FPU usage"	$\lambda x.2.5x$	length	44
`fib`	"arith. operations"	$\lambda x.2.17 \cdot 1.61^x + 0.82 \cdot (-0.61)^x - 3$	value	116
`grammar`	"phrases"	24	length/size	227
`hanoi`	"disk movements"	$\lambda x.2^x - 1$	value	100
`insert_stores`	"accesses Stores"	$\lambda n, m.n + k$	length	292
	"insertions Stores"	$\lambda n, m.n$		
`perm`	"bytecode instructions"	$\lambda x.(\sum_{i=1}^{x} 18 \cdot x!) + (\sum_{i=1}^{x} 14 \cdot \frac{x!}{i}) + 4 \cdot x!$	length	98
`power_set`	"output elements"	$\lambda x.\frac{1}{2} \cdot 2^{x+1}$	length	119
`qsort`	"lists parallelized"	$\lambda x.4 \cdot 2^x - 2x - 4$	length	144
`send_files`	"bytes read"	$\lambda x, y.x \cdot y$	length/size	179
`subst_exp`	"replacements"	$\lambda x, y.2xy + 2y$	size/length	153
`zebra`	"steps"	30232844295713061	size	292

Some results (Java)

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 complexity functions, resources, types of loops/recursion, etc.

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

Energy Consumption Analysis

Approach: [NMLH08]

- Specialize generic 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.

- Initially applied to Java bytecode: [NMLH08]
  - Java bytecode energy consumption models available for simple processors –upper bound consumption per bytecode in joules:

Opcode	Inst. Cost in μJ	Mem. Cost in μJ	Total Cost in in μJ
`iadd`	.957860	2.273580	3.23144
`isub`	.957360	2.273580	3.230.94
...	...	...	...

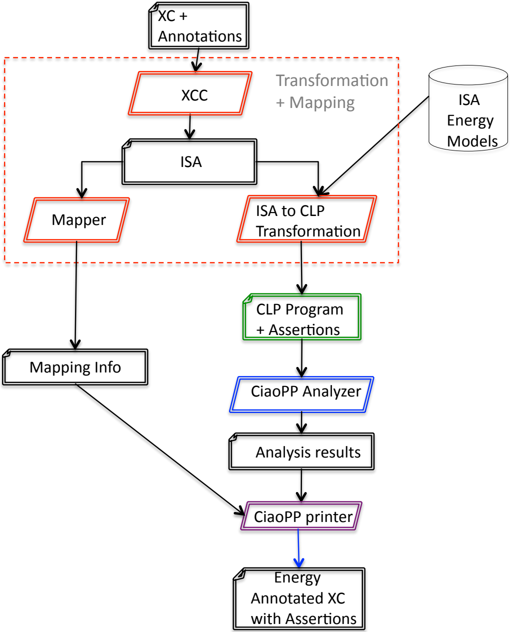
  - Encouraging results: meaningful functions inferred in many cases.
  - But no comparison with actual device consumption.

Energy Consumption Analysis

Approach: [NMLH08]

- Specialize generic 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.

- Much more ambitious (within ENTRA Project):
  - Analysis of (embedded) programs written in XC, on XMOS processors.
  - Use more sophisticated *ISA-level energy models* for XMOS XS1, developed by Bristol & XMOS (based on "Tiwari" model).


XC +
Annotations
XCC
Transformation
+ Mapping
ISA
Energy
Models
ISA
Mapper
ISA to CLP
Transformation
CLP Program
+ Assertions
Mapping Info
CiaoPP Analyzer
Analysis results
CiaoPP printer
Energy
Annotated XC
with Assertions

Example: An XC source, its ISA (left) and its IR (right)

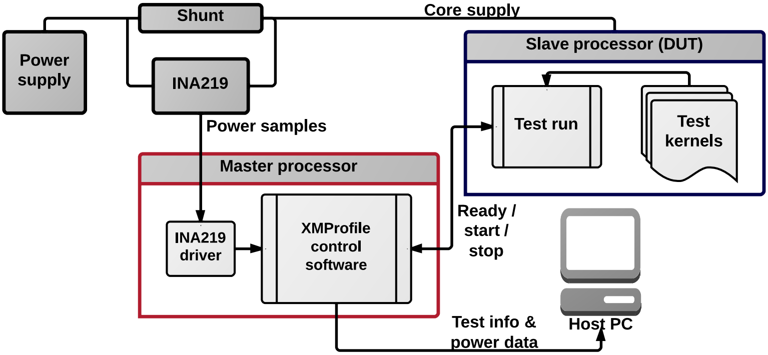
```
int fact(int N) {
  if (N <= 0) return 1;
  return N * fact(N - 1);
}
```

```
1   <fact>:
2   0x01: entsp 0x2
3   0x02: stw   r0, sp[0x1]
4   0x03: ldw   r1, sp[0x1]
5   0x04: ldc   r0, 0x0
6   0x05: lss   r0, r0, r1
7   0x06: bf    r0, <0x08>


10  0x07: bu    <0x10>
11  0x0a: ldw   r0, sp[0x1]
12  0x0b: sub   r0, r0, 0x1
13  0x0c: bl    <fact>

15  0x0d: ldw   r1, sp[0x1]
16  0x0e: mul   r0, r1, r0
17  0x0f: retsp 0x2

19  0x08: mkmsk r0, 0x1
20  0x09: retsp 0x2
```

```
1   fact(R0,R0_3):-
2      entsp(0x2),
3      stw(R0,Sp0x1),
4      ldw(R1,Sp0x1),
5      ldc(R0_1,b0x0),
6      lss(R0_2,bR0_1,R1),
7a     bf(R0_2,0x8),
7b     fact_aux(R0_2,Sp0x1,R0_3,R1_1).
9   fact_aux(1,Sp0x1,R0_4,R1):-
10     bu(0x10),
11     ldw(R0_1,Sp0x1),
12     sub(R0_2,R0_1,0x1),
13a    bl(fact),
13b    fact(R0_2,R0_3),
15     ldw(R1,Sp0x1),
16     mul(R0_4,R1,R0_3),
17     retsp(0x2).
18  fact_aux(0,Sp0x1,R0,R1):-
19     mkmsk(R0,0x1),
20     retsp(0x2).
```

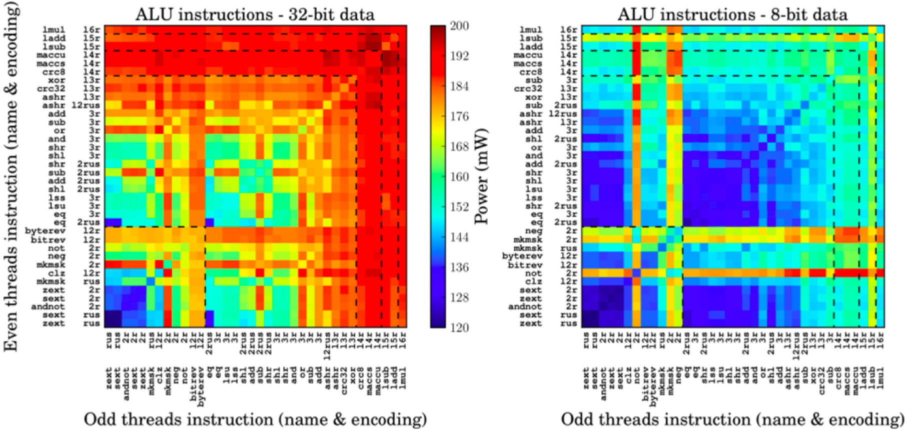
Obtaining the Energy Model for the XS1 Architecture

- The energy model is obtained via profiling (once and for all).
- Gets energy data for each ISA instruction by using a test harness:
  - Slave processor executing test kernels and
  - a master processor collecting power samples.



Low-level ISA characterization

Obtaining the cost model: energy consumption/instructionl; interference.



Bristol U / XMOS.

Energy model, expressed in the Ciao assertion language

energy.pl

```
:- 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)----------------------------------------------
```

XC Source

fact.xc

```
#include "fact.h"

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

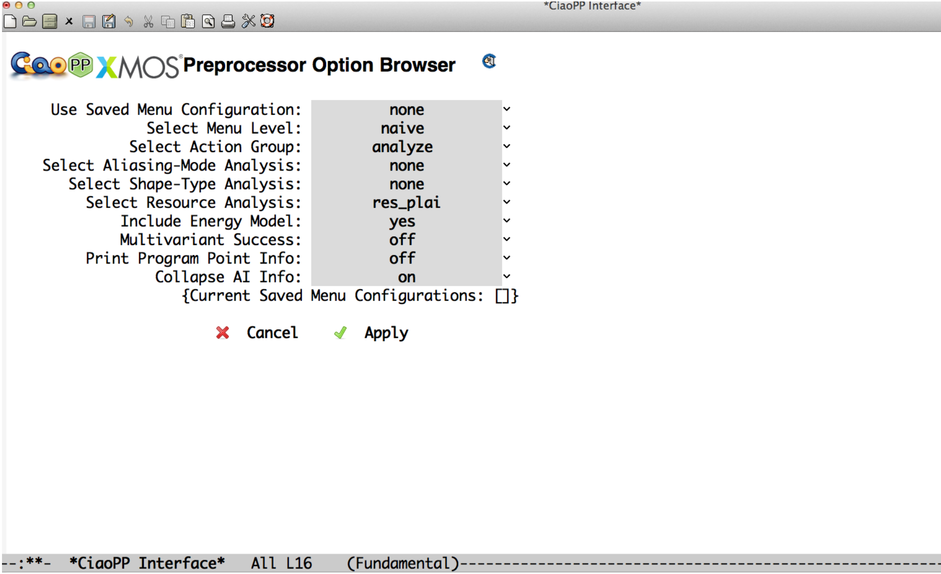
```
--:---  fact.xc       All L10    (C/l Abbrev)------------------------------------------
<nil> <drag-mouse-1> is undefined
```

Assembly Code

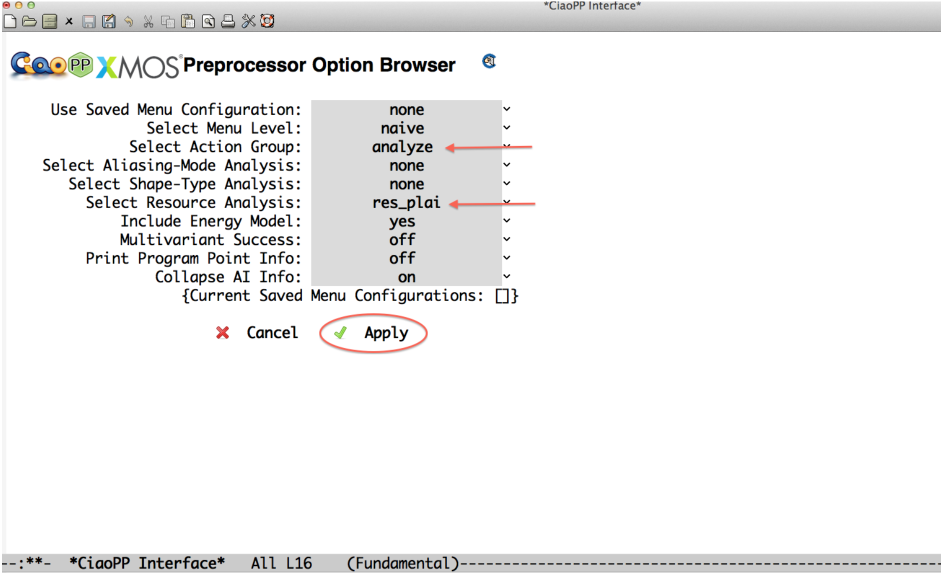
factassembly.pl

```
fact:
        entsp 6
        stw r0, sp[4]
        stw r0, sp[2]
.Lxtalabel0:
        ldw r0, sp[4]
        ldc r1, 0
        lss r0, r1, r0
        bt r0, .LBB0_4
        bu .LBB0_3
.LBB0_3:
        mkmsk r0, 1
        stw r0, sp[3]
        bu .LBB0_5
.LBB0_4:
.Lxtalabel1:
        ldw r0, sp[4]
        sub r1, r0, 1
        stw r0, sp[1]
        mov r0, r1
.Lxta.call_labels0:
        bl fact
        ldw r1, sp[1]
        mul r0, r1, r0
        stw r0, sp[3]
.LBB0_5:
        ldw r0, sp[3]
        retsp 6
--:---  factassembly.pl    Top L3    (Ciao)------------------------------------------
```

CiaoPP Menu


CiaoPP Interface
Preprocessor Option Browser
Use Saved Menu Configuration: none
Select Menu Level: naive
Select Action Group: analyze
Select Aliasing-Mode Analysis: none
Select Shape-Type Analysis: none
Select Resource Analysis: res_plai
Include Energy Model: yes
Multivariant Success: off
Print Program Point Info: off
Collapse AI Info: on
{Current Saved Menu Configurations: []}
Cancel
Apply
--:**- *CiaoPP Interface* All L16 (Fundamental)------------------------------------

Select Resource Analysis



Analysis Results

fact_results.pl

```
:- 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 Results

fact_results.pl

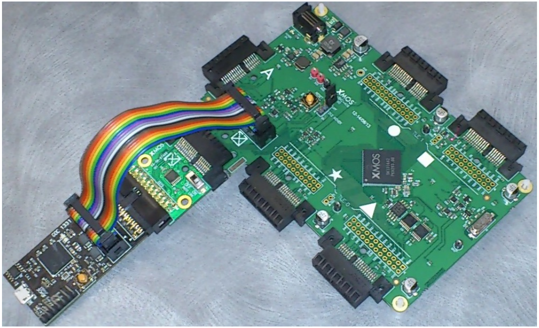
```
:- module(_,[fact/2],[ciaopp(xcore(model(instructions))),ciaopp(xcore(model(energy))),assertions]).

:- 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,
```

Other results

Function name	Description	Energy function
fact(N)	Calculates $N!$	$26.0\ N + 19.4$
fibonacci(N)	Nth Fibonacci no.	$30.1 + 35.6\ \phi^N + 11.0\ (1-\phi)^N$
sqr(N)	Computes N^2	$103.0\ N^2 + 205.8\ N + 188.32$
isprime(N)	Checks if N is prime	$58.6\ N - 35.5$
power(base,exp)	Calculates $base^{exp}$	$6.3\ (\log_2 exp + 1) + 6.5$

Measuring Power Consumption on the Hardware

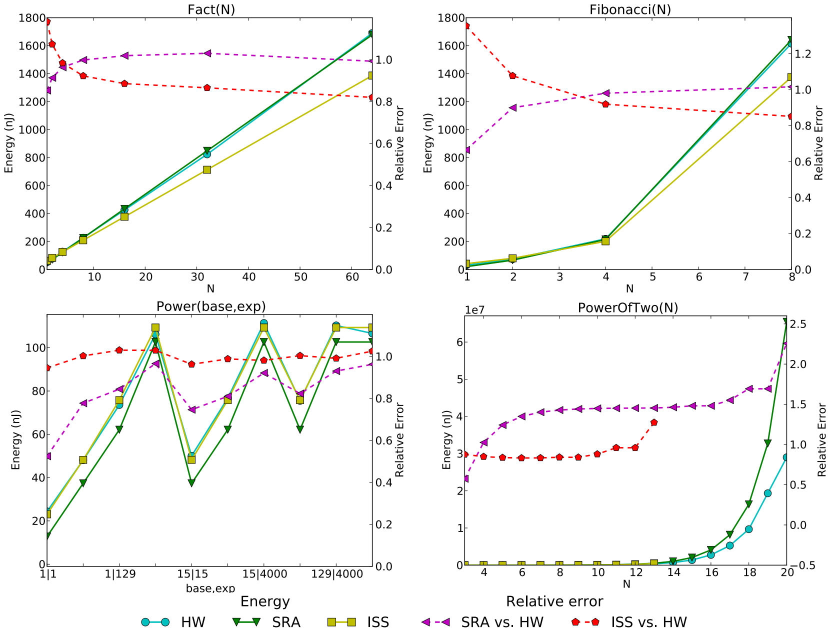
- XMOS Ltd. provides the XTAG3 measurement circuit.
- XTAG3 plugs into the XMOS XS1 board.



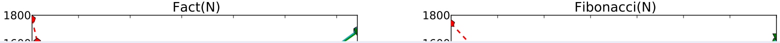
We compare to:

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

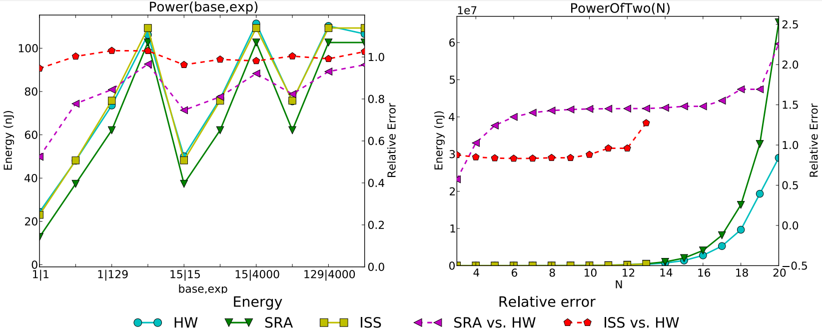
Some Results



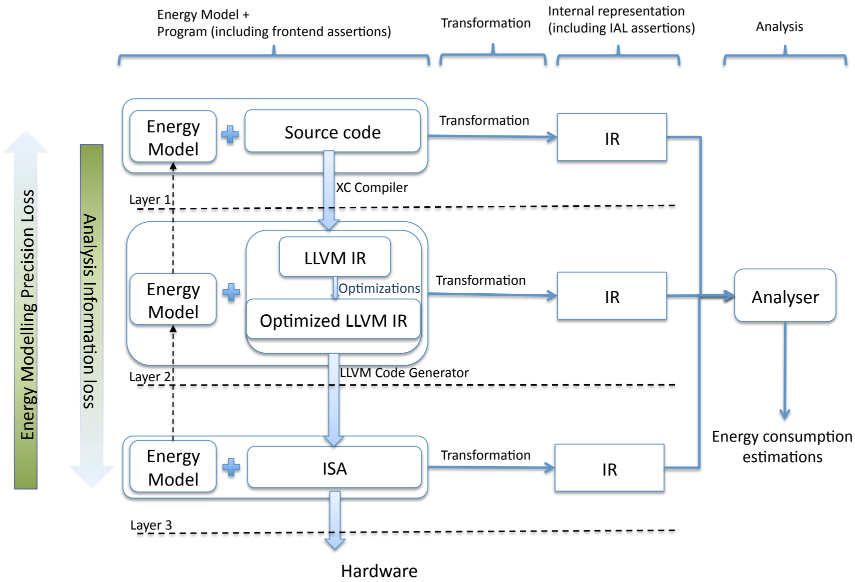
Some Results



- SRA provides results *beyond what is possible with simulation* (as test run-time increases, ISS becomes impractically long).
- SRA shows promising accuracy in comparison with ISS and the HW (at least for the simple cases studied so far).
- Simulation time limits the usefulness of ISS method, whereas equation solving limits SRA.



IR Level Trade-offs



Accuracy of Energy Analyses: LLVM vs. ISA layer

Program	Error vs. HW		ISA/LLVM
	llvm	isa	
fact	4.5%	2.86%	0.94
fibonacci	11.94%	5.41%	0.92
sqr	9.31%	1.49%	0.91
power_of_two	11.15%	4.26%	0.93
reverse	2.18%	N/A	N/A
concat	8.71%	N/A	N/A
mat_mult	1.47%	N/A	N/A
sum_facts	2.42%	N/A	N/A
Average	**6.46%**	**3.50%**	**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 layer but can be analyzed at the LLVM layer.

Energy consumption verification / debugging

```
check fact(x) :  int(x) + resource(energy, 0, 100)
```

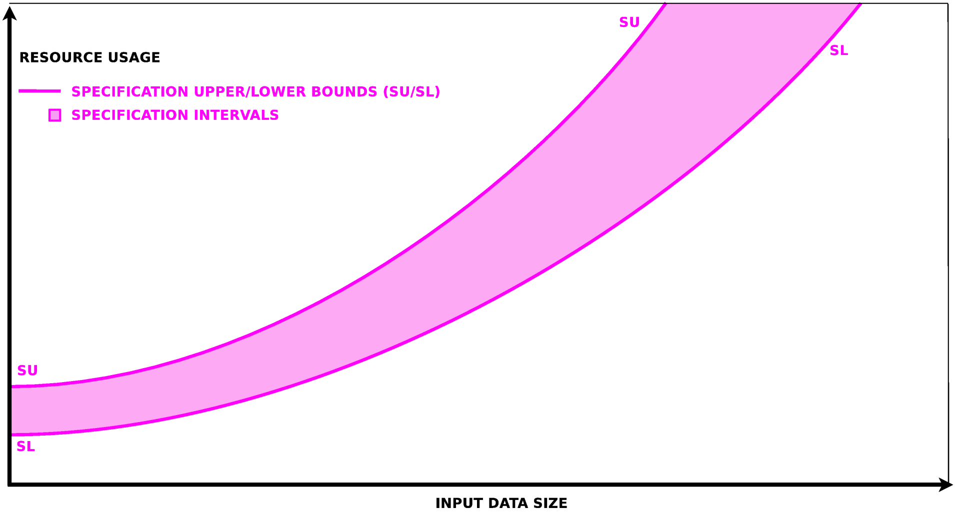
1. Resource analysis infers upper and lower bounds for resource "energy." The analysis results produced are:

```
true fact(x) :  int(x)
             => (int(x), int(ret), rsize(x, num(lx,ux)),
                rsize(ret, num(Factorial(lx),Factorial(ux))))
             + resource(energy, 21 * lx + 16, 21 * ux + 16)
```

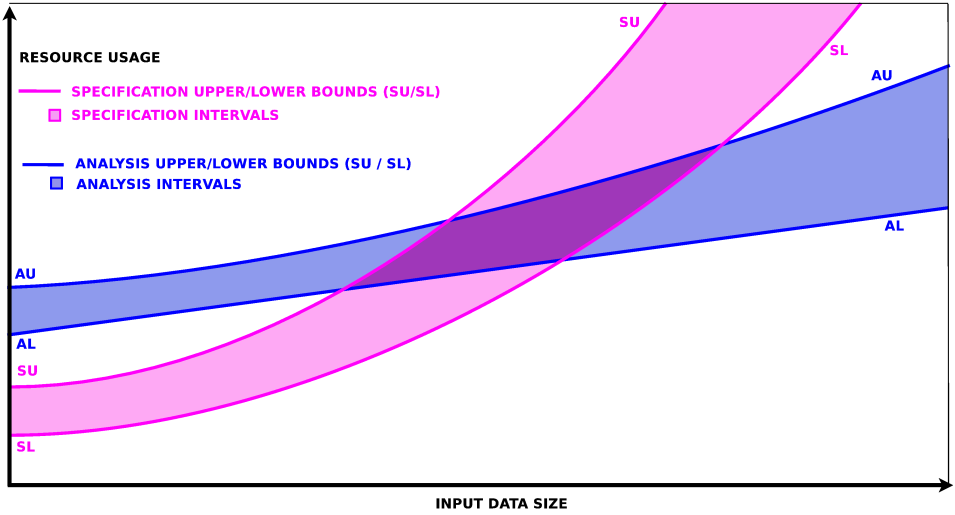
2. Then, the analysis results are compared with the "check" assertion (the specification) and the following assertions are produced:

```
checked fact(x) :  (int(x), intervals(int(x), [i(0,4)]))
                + resource(energy, 0, 100)

false fact(x) :  (int(x), intervals(int(x), [i(5,inf)]))
              + resource(energy, 0, 100)
```

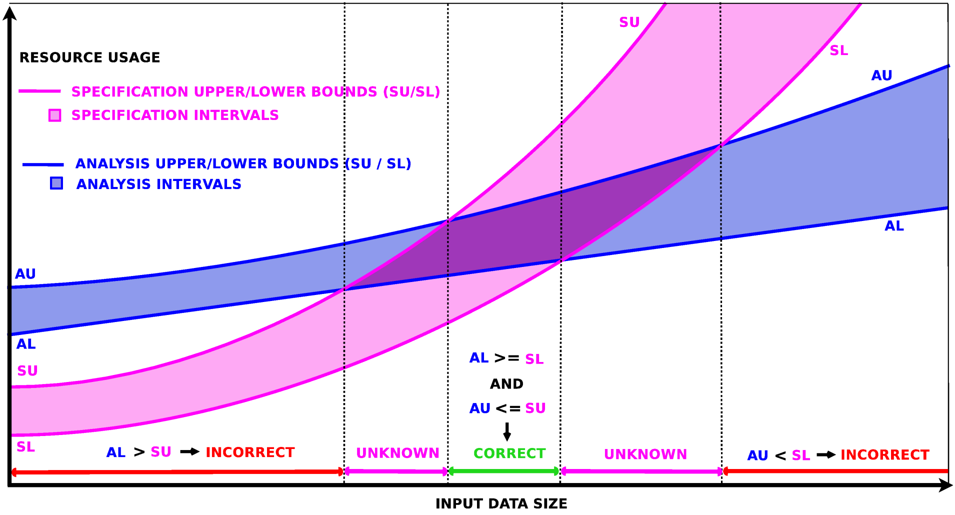
Resource Usage Verification – Function Comparisons



Resource Usage Verification – Function Comparisons



Resource Usage Verification – Function Comparisons



Resource Analysis as an Abstract Interpretation

[SLH14, SLBH13]

- In the classical CiaoPP resource analysis the last steps (setting up and solving recurrences) were not implemented as an abstract domain.
- We have now defined, implemented and integrated the resource analysis as an *abstract domain* (a plugin of the generic fixpoint).
- We get all the good features of the AI framework for free:
  - Multivariance: e.g., separate different call patterns for same block:
    `sort(lst(int),var) ... sort(lst(flt),var) ... sort(var,lst(int))`
  - Easier combination with other domains.
  - Easier integration w/static debugging/verification and rt-checking.
  - Many other engineering advantages.
- New domain for size analysis (*sized types*) that infers bounds on the size of data structures *and substructures*.
  - Size: number of rule applications in type/shape definition.
- Used in the XC energy analysis.

Experimental Results

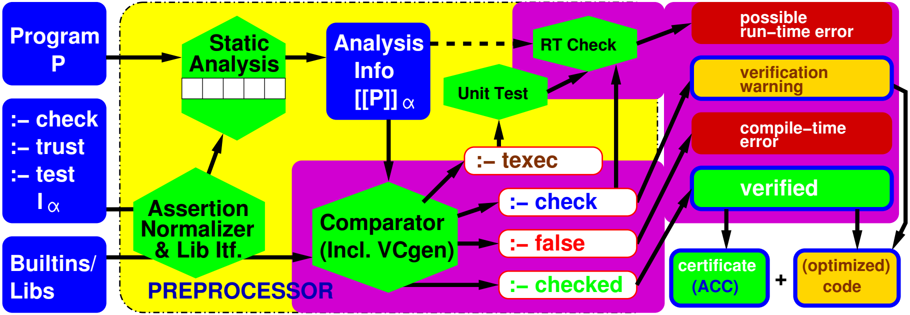
Prog.	Resource An. (LB)			Resource An. (UB)					An. Time (s)	
	New	*Prev.*		*New*	*Prev.*		*RAML*		*New*	*Prev.*
`append`	α	α	$=$	β	β	$=$	β	$=$	1.00	0.53
`appAll`	$a_1 a_2 a_3$	a_1	$+$	$b_1 b_2 b_3$	∞	$+$	$b_1 b_2 b_3$	$=$	2.41	0.67
`coupled`	μ	0	$+$	ν	∞	$+$	ν	$=$	1.37	0.64
`dyade`	$\alpha_1\alpha_2$	$\alpha_1\alpha_2$	$=$	$\beta_1\beta_2$	$\beta_1\beta_2$	$=$	$\beta_1\beta_2$	$=$	1.66	0.62
`erathos`	α	α	$=$	β^2	β^2	$=$	β^2	$=$	2.25	0.77
`fib`	ϕ^μ	ϕ^μ	$=$	ϕ^ν	ϕ^ν	$=$	infeas.	$+$	1.06	0.67
`hanoi`	1	0	$+$	2^ν	∞	$+$	infeas.	$+$	0.82	0.60
`isort`	α^2	α^2	$=$	β^2	β^2	$=$	β^2	$=$	1.68	0.62
`isortl`	a_1^2	a_1^2	$=$	$b_1^2 b_2$	∞	$+$	$b_1^2 b_2$	$=$	2.55	0.67
`lisfact`	$\alpha\gamma$	α	$+$	$\beta\delta$	∞	$+$	unkn.	?	1.39	0.64
`listnum`	μ	μ	$=$	ν	ν	$=$	unkn.	?	1.19	0.58
`minsort`	α^2	α	$+$	β^2	β^2	$=$	β^2	$=$	1.94	0.67
`nub`	a_1	a_1	$=$	$b_1^2 b_2$	∞	$+$	$b_1^2 b_2$	$=$	3.61	0.91
`part`	α	α	$=$	β	β	$=$	β	$=$	1.70	0.65
`zip3`	$\min(\alpha_i)$	0	$+$	$\min(\beta_i)$	∞	$+$	β_3	$+$	2.48	0.57

Some Future Work

- Analysis of multi-threaded applications.
  - We can handle at least 'fork-join' and 'coroutining' in most other analyses, but not (yet) for resources.
- Average cost analysis (based on some earlier work).
- Increase (always?) the classes of programs for which accurate results can be obtained.
- Try richer energy models.
- Improve analysis accuracy at the ISA layer:
  - Propagating high-level program information such as types, loop bounds, and communication into the lower-level representations.
- Explore more analysis/modeling layer choices, e.g.:
  - Try stand-alone models at the LLVM IR layer.
  - Analysis of source code.

Thank you for your attention!

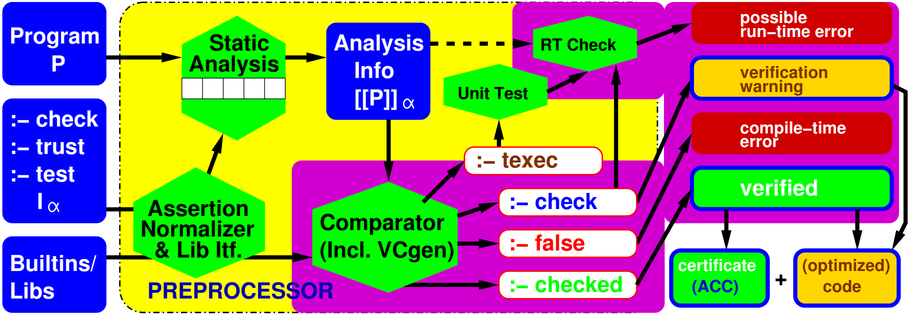
Integrated Static/Dynamic Debugging and Verification



	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	$\mathcal{I}_\alpha \not\leq \alpha(\llbracket P \rrbracket)$	$\mathcal{I}_\alpha \not\leq \llbracket P \rrbracket_{\alpha^+}$

[BDD+97, HPB99, PBH00c, PBH00a, HPBLG03, HALGP05, PCPH06, PCPH08, MLGH09, SMH14]

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.

[BDD+97, HPB99, PBH00c, PBH00a, HPBLG03, HALGP05, PCPH06, PCPH08, MLGH09, SMH14]

References – Analysis and Verification of Resources

[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.

[LGHD94] 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 International Symposium on Parallel Symbolic Computation, PASCO'94*, pages 133–144. World Scientific, September 1994.

[LGHD96] 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.

[DLGHL94] 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.

[DLGHL97] S. K. Debray, P. López-García, M. Hermenegildo, and N.-W. Lin.
Lower Bound Cost Estimation for Logic Programs.
In *1997 International Logic Programming Symposium*, pages 291–305. MIT Press, Cambridge, MA, October 1997.

[NMLGH07] 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)*, volume 4670 of *Lecture Notes in Computer Science*. Springer, 2007.

[MLGCH08] 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.

[NMLH08] J. Navas, M. Méndez-Lojo, and M. Hermenegildo.
Safe Upper-bounds Inference of Energy Consumption for Java Bytecode Applications.
In *The Sixth NASA Langley Formal Methods Workshop (LFM 08)*, April 2008.
Extended Abstract.

[NMLH09] 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.

[LGDB10] P. López-García, L. Darmawan, and F. Bueno.
A Framework for Verification and Debugging of Resource Usage Properties.
In M. Hermenegildo and T. Schaub, editors, *Technical Communications of the 26th Int'l. Conference on Logic Programming (ICLP'10)*, volume 7 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 104–113, Dagstuhl, Germany, July 2010. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[SLBH13] A. Serrano, P. López-Garcia, F. Bueno, M. Hermenegildo.
Sized Type Analysis 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.

[LKSGL13] U. Liqat, S. Kerrison, A. Serrano, K. Georgiou, P. López-Garcia, N. Grech, M.V. Hermenegildo, K. Eder.
Energy Consumption Analysis of Programs based on XMOS ISA-Level Models.
In *Pre-proceedings of the 23rd International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR'13)*, September 2013.

[SLH14] A. Serrano, P. López-Garcia, 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.

References – Overall Model

[BDD+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.

[HPB99] 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.

[PBH00c] 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.

[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, September 2000.

[HPBLG03] 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.

[MLGH09] E. Mera, P. López-García, and M. Hermenegildo.
Integrating Software Testing and Run-Time Checking in an Assertion Verification Framework.
In *25th International Conference on Logic Programming (ICLP'09)*, number 5649 in LNCS, pages 281–295. Springer-Verlag, July 2009.

References – Assertion Language

[PBH97] 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.
Available from `ftp://clip.dia.fi.upm.es/pub/papers/assert_lang_tr_discipldeliv.ps.gz` as technical report CLIP2/97.1.

[PBH00b] 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.

[MLGH09] E. Mera, P. López-García, and M. Hermenegildo.
Integrating Software Testing and Run-Time Checking in an Assertion Verification Framework.
In *25th International Conference on Logic Programming (ICLP'09)*, number 5649 in LNCS, pages 281–295. Springer-Verlag, July 2009.

[SMH14] 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.

References – Intermediate Representation

[MLNH07] M. Méndez-Lojo, J. Navas, and M. Hermenegildo.
A Flexible (C)LP-Based Approach to the Analysis of Object-Oriented Programs.
In *17th International Symposium on Logic-based Program Synthesis and Transformation (LOPSTR 2007)*, number 4915 in LNCS, pages 154–168. Springer-Verlag, August 2007.

References – Abstraction Carrying Code

[APH05] E. Albert, G. Puebla, and M. Hermenegildo.
Abstraction-Carrying Code.
In *Proc. of LPAR'04*, volume 3452 of *LNAI*. Springer, 2005.

[HALGP05] M. Hermenegildo, E. Albert, P. López-García, and G. Puebla.
Abstraction Carrying Code and Resource-Awareness.
In *PPDP*. ACM Press, 2005.

[AAPH06] 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.

References – Fixpoint-based Analyzers (Abstract Interpreters)

[MH92] 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.

[BGH99] 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.

[PH96] G. Puebla and M. Hermenegildo.
Optimized Algorithms for the Incremental Analysis of Logic Programs.
In *International Static Analysis Symposium (SAS 1996)*, number 1145 in LNCS, pages 270–284. Springer-Verlag, September 1996.

[HPMS00] 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.

[NMLH07] 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.

References – Modular Analysis, Analysis of Concurrency

[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.

[BCHP96] 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.

[PH00] 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.

[BdlBH+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.

[PCPH06] 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.

[PCPH08] 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.

References – Domains: Sharing/Aliasing

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

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

[NBH06] J. Navas, F. Bueno, and M. Hermenegildo.
Efficient top-down set-sharing analysis using cliques.
In *Eight International Symposium on Practical Aspects of Declarative Languages*, number 2819 in LNCS, pages 183–198. Springer-Verlag, January 2006.

[MLH08] M. Méndez-Lojo and M. Hermenegildo.
Precise Set Sharing Analysis for Java-style Programs.
In *9th International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI'08)*, number 4905 in LNCS, pages 172–187. Springer-Verlag, January 2008.

[MMLH+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.

[MKSH08] 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.

[MLLH08] 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.

[MKH09] 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.

References – Domains: Shape/Type Analysis

[VB02] C. Vaucheret and F. Bueno.
More Precise yet Efficient Type Inference for Logic Programs.
In *International Static Analysis Symposium*, volume 2477 of *Lecture Notes in Computer Science*, pages 102–116. Springer-Verlag, September 2002.

[MSHK07] M. Marron, D. Stefanovic, M. Hermenegildo, and D. Kapur.
Heap Analysis in the Presence of Collection Libraries.
In *ACM WS on Program Analysis for Software Tools and Engineering (PASTE'07)*. ACM, June 2007.

[MHKS08] M. Marron, M. Hermenegildo, D. Kapur, and D. Stefanovic.
Efficient context-sensitive shape analysis with graph-based heap models.
In Laurie Hendren, editor, *International Conference on Compiler Construction (CC 2008)*, Lecture Notes in Computer Science. Springer, April 2008.

References – Domains: Non-failure, Determinacy

[DLGH97] S.K. Debray, P. López-García, and M. Hermenegildo.
Non-Failure Analysis for Logic Programs.
In *1997 International Conference on Logic Programming*, pages 48–62, Cambridge, MA, June 1997. MIT Press, Cambridge, MA.

[BLGH04] F. Bueno, P. López-García, and M. Hermenegildo.
Multivariant Non-Failure Analysis via Standard Abstract Interpretation.
In *7th International Symposium on Functional and Logic Programming (FLOPS 2004)*, number 2998 in LNCS, pages 100–116, Heidelberg, Germany, April 2004. Springer-Verlag.

[LGBH05] P. López-García, F. Bueno, and M. Hermenegildo.
Determinacy Analysis for Logic Programs Using Mode and Type Information.
In *Proceedings of the 14th International Symposium on Logic-based Program Synthesis and Transformation (LOPSTR'04)*, number 3573 in LNCS, pages 19–35. Springer-Verlag, August 2005.

[LGBH10] 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.

References – Automatic Parallelization, (Abstract) Partial Evaluation, Other Optimizations

[GH91] 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.

[PH97] G. Puebla and M. Hermenegildo.
Abstract Specialization and its Application to Program Parallelization.
In J. Gallagher, editor, *Logic Program Synthesis and Transformation*, number 1207 in LNCS, pages 169–186. Springer-Verlag, 1997.

[MBdlBH99] 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.

[PH99] 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.

[PHG99] G. Puebla, M. Hermenegildo, and J. Gallagher.
An Integration of Partial Evaluation in a Generic Abstract Interpretation Framework.
In O Danvy, editor, *ACM SIGPLAN Workshop on Partial Evaluation and Semantics-Based Program Manipulation (PEPM'99)*, number NS-99-1 in BRISC Series, pages 75–85. University of Aarhus, Denmark, January 1999.

[PH03] 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 talk.

[PAH06] G. Puebla, E. Albert, and M. Hermenegildo.
Abstract Interpretation with Specialized Definitions.
In *SAS'06*, number 4134 in LNCS, pages 107–126. Springer-Verlag, 2006.

[CCH08] A. Casas, M. Carro, and M. Hermenegildo.
A High-Level Implementation of Non-Deterministic, Unrestricted, Independent And-Parallelism.
In M. García de la Banda and E. Pontelli, editors, *24th International Conference on Logic Programming (ICLP'08)*, volume 5366 of *LNCS*, pages 651–666. Springer-Verlag, December 2008.

[MKSH08] 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.

[MCH04] J. Morales, M. Carro, and M. Hermenegildo.
Improving the Compilation of Prolog to C Using Moded Types and Determinism Information.
In *PADL'04*, number 3057 in LNCS, pages 86–103. Springer-Verlag, June 2004.

[CMM+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*, pages 271–281. ACM Press / Sheridan, October 2006.