Taller de Investigación I
Work Memory

Signature: Non clasic execution models
(Modelos de ejecución no clásicos)

Title: Development of a Prolog Profiler.

Director: Dr. Francisco Bueno

Student: Edison Mera Menéndez

Madrid, September 2004.

Contents

1 Introduction

This work is part of the requirements to obtain the necessary credits for the Phd in Artificial Intelligence and Computer Science.

1.1 Work description

In this paper we developed a tool called profiler [Deb83], and is used to measure the performance of the pieces of a program. It has been done to running the programs in a modified semantic, wich give us the execution time or the count that a given event occurs in the program.

However, make a correct profiler in a logic language is not trival, due to is necessary to consider properties that are exclusives to it, such as the fail or backtracking.

The language used to implement it is Ciao Prolog[1]. Thanks to the characteristics of Ciao, such as the modularity and the automatic code transformation through language semantic packages, the profiler have new properties not availables in other profilers, for example the combination of time profiler with count profiler, and the fact that it consist in two part, as we will show in details after.

One of the two parts of the profiler is implemented at low level in the WAM, and count global events, i. e., it not depend on the context in that such event occurs, for example the total count of calls to a predicate, and the other part is implemented at high level using semantic packages, and it counts 'local events', that depends on the context of such event occurs, for example the number of time that inside a predicate is called other predicate.

The advantage of have implemented the profiler using semantic package is the modularity, and then we can specify what modules requires to be instrumented for profiling.

1.2 Goals of this work

The goal of this work is to develop a profiler that can collect useful information to optimize prolog aplications. Also that profiler has been designed to be used in the future with tools of static analysis, such as the abstract interpretation or the specialization techniques. In such cases, the profiler will be useful to show the advantage that these techniques gives to the system.

[1]Ciao Prolog is available in http://www.clip.dia.fi.upm.es/Software

2 Prolog Profilers

The implementation of prolog profilers has already considered by many authors [DN00, Deb83]. These works has demostrated the issues that appears, due to the control flow in prolog is more complex than imperative languages.

In prolog there are 4 counters: The number of calls, fails, exits and redos [Mat94]. Also the flow control present complex things to be considered such as the backtracking or the choice points.

In general, the information that can be obtained with the profiler is of two kinds [Mat94]:

- Execution time: Is the total amount of time spent in execute a part of code.
- Execution count: Is the total count of a given part of code has been executed.

Note that the second one does not give directly time information, but is near related with the first one, and have the advantage that it don't depends of the architecture in that the program is being executed.

2.1 Time Profiling

Is clear that we could be more interesting in the execution time, because is the only one that can give us true information about what is happening. However, there are many problems that must be solved so that the given invormation be relevant.

In our work, the firs problem that must address is that in almost all operating systems, the system clock have a poor resolution, near to 0.01 seconds, and then many event times will be measured as spent 0 seconds, obviously that is absurd. Also we should consider that the time measurement functions normally are implemented at high level, and introduces an important overhead.

After the observations above, we propose that for a profiler give us useful information, it must fit the next requirements:

- Be independent of the plattform speed. That is, the objective of the benchmark is to measure the time of execution of any piece of software independently of the plattform used to do the proof.
- Do use of the high capacity and speed of modern computers, by this way, at most power computing, one could espect better results, and for

the same benckmark, the improvement of the hardware platform must be reflected in a improvement of the performance of the profiling tool.

For the first requirement, the measurement unit suggested is one CPU clock cycle. These measure is relatively independent to the hardware, and for estimate the seconds used in execute some task, we must divide the number of CPU cycles by the velocity in Hertzios of the computer. For example, if some task takes 1.000.000.000 of cpu cycles, then the taken time in a 2GHz modern computer is 0.5 seconds.

The second requirement, is solved fortunately with the same approach. A improvement in the hardware will be reflected in a better performance of the profiling tool. In the example above, if we use a 4GHz computer, then the task will take 0.25 seconds.

But there are some issues: modern computers could execute more than 1 instruction by clock cycle, some laptops could slow down your cpu clock to improve power managment, and there are a few of optimizations such as the unsorted excecution, that are the benchmarks imprecise, and for that, make completely hardware independent test is a challenge task.

Now, the pentium II and highter processors have an instruction RDTSC, that returns the number of cpu clock since the last power on of the computer. These information could be used to do the time measurement in the benchmarks.

However, to introduce this functionality to measure the time is not trivial, because of it we want to obtain correct times of each system event we must add it in critical parts of the operating system. Fortunately, these task has already been done [Ras00], and there are a patch kernel that incorporates these changes in the operating system.

This library is not essential to let the profiler work, but is recommended to use it to improve the precision in the measurements.

2.2 Count Profiling

The direct count of events are nearly related with the execution time, and is possible to estimate the total time in a proof if the time of a given event is known. That profiling have advantages in opposite to the time profiling, because it is plattform independent.

2.3 Cost Center Models

As already we say after, the flow control in Prolog is more complex than other languages. We are taken the “box model” of Lawrence Byrd, and done

some modifications to get the counts.

Say p a predicate marked to profiling. We call that predicate a cost center. We define the next:

- Call count: Number of times the control reach to p from the left.
- Fail count: Number of times the control do "backtracking" at the left of p.
- Exit count: Number of times the control reach the right of p.
- Redo count: Number of times the control come from the right of p.

The word cost center was used in [RGM98], in reference to profiling in functional languages like Haskel, and we have adopt the same word for Prolog. However is important to have in mind that Haskel is different to prolog and don't have the same sense.

However, a profiler that only shows that kind of information is not very useful, there are many information that leave hidden, for example the cause of the spend time in the predicates. To work around that issue, We register inside a cost center the next things:

- The predicate list called inside the cost center.
- Calls: The number of calls done to these predicates.
- Skips: The number of skips over a node, if it hava a cut (!).
- NSkips: The number of nodes that has been cutted with the cut (!) operation.
- Cuts: The number of effective cuts that has effect and that has been done in a node scope. We refer to effective cuts in the sense that it has dropped point choices.
- SCuts: The number of cuts without effects and that has been done in a node scope.
- Retrys: The number of times a choice point is re tryed. Is not the same as redo.

Note that the total amount of NSkips is equal to SCuts.

With these definitios, we can build a profiler that give us more useful information and also let us to detect the bottle necks of our program in a more eficient way.

This type of profiler is called detailed profiler, and we call flat profiler at the total amount of all counters without consider the const centers.

Finally, the resumed profiler is the result of show only the count events for each cost center, calculating the totals without have in mind the predicate names called.

All program hava at least a const center, that represents the overall system. In our profiler, that cost center appears with the name "Beyond Cost Centers".

3 System Function

As already tell briefly, the profiler have two parts, one do the instrumentation of a module marked for profiling and is a semantic package, and other is implemented at low level in the WAM.

To use the profiler in a given module, we must use the "profiler" package as follows:

```
:- module(_, _, [profiler]).
```

or

```
:- use_package(profiler).
```

If we don't specify more things, all the predicates implemented inside the module will be instrumented for profiling, but the next directives let us to change these behavior:

```
:- profile F1/N1, ..., Fi/Ni.
```

Specify that only the predicates `F1/N1`, ..., `Fi/Ni` will be instrumented.

```
:- noprofile F1/N1, ..., Fi/Ni.
```

Opossite to the prior, it specify that all the predicates, except the `F1/N1`, ..., `Fi/Ni` will be instrumented.

Let us to show an example:

Suppose we have the module `qsort1.pl`, if we like to mark this module for profiling, we must include the package "profiler", as follows:

3.0.1 File `qsort1.pl`

```
:- module(qsort1, [qsort/2], [profiler]).

:- profile append/3.

qsort([],[]).
qsort([X|L],R) :-
split(L, X, L1, L2),
qsort(L1, R1),
qsort(L2, R2),
append(R1, [X|R2], R).

split([],_,[],[]).
split([X|L],Y,[X|L1],L2) :-
X =< Y, !,
split(L,Y,L1,L2).
split([X|L],Y,L1,[X|L2]) :-
split(L,Y,L1,L2).

append([], L, L).
append([E|Es], L, [E|R]) :- append(Es, L, R).
```

3.1 Code Instrumentation

When a module has been marked to profiling, this is transformed in other equivalent but that have predicates that intercept the four events that let to enter or exit from a cost center.

In the example above, suppose that we want to instrument the predicate `append/3`, then we must add the next directive to the module:

```
:- profile append/3.
```

After the expansion the code will looks as follow:

```
:- multifile cost_center/2,
:- data cost_center/2,
:- discontiguous cost_center/2

...

cost_center('qsort1:append',3).
append(E,L,R) :-
          profile__hook_cc_call(Prev_cc),
          profile__hook_cc_fail(Prev_cc),
          'prof$append'(E,L,R),
          profile__hook_cc_exit(Prev_cc,Active_cc),
          profile__hook_cc_redo(Active_cc).
```

```
'prof$append'([],L,L).
'prof$append'([E|Es], L, [E|R]) :-
          'prof$append'(Es, L, R).
```

The multifile predicate `cost_center/2` is used to know the predicates that has been marked as cost centers.

As we can see, the `append/3` predicate has been renamed to `'prof$append'/3`. And to avoid recursive calls inside the same cost center also the call to `append/3` is changed by `'prof$append'/3`. This avoid the destruction of the last call optimization over the predicate itself.

The original predicate is replaced with another that uses the next auxiliar predicates or hooks:

- `profile__hook_cc_call(Prev_cc)`: Used to detect that a cost center is being called, increment the call counter and unifies `Prev_cc` with a pointer to the previous cost center.

- `profile__hook_cc_fail(Prev_cc)`: Specifies `Prev_cc` as current cost center, and increment the fail counter in the cost center that is being leaved. To let it work correctly is necessary that the first time not fail, and that is done to introducing a choice point as we can see next:

  ```
  profile__hook_cc_fail(_).
  profile__hook_cc_fail(Prev_cc) :-
          profile__hook_cc_fail_(Prev_cc).
  ```

  `profile__hook_cc_exit(Prev_cc,Active_cc)`: Specifies `Prev_cc` as current cost center, unifies `Active_cc` with the previous cost center, and increment the exit counter.

- `profile__hook_cc_redo(Active_cc)`: Specifies `Active_cc` as current cost center and increment the redo counter of the current cost center. As occurs with the fail hook, is necessary to introduce a choice point:

  ```
  profile__hook_cc_redo(_).
  profile__hook_cc_redo(Active_cc) :-
          profile__hook_cc_redo_(Active_cc).
  ```

The variables `Prev_cc` and `Active_cc` let us to build a cost center stack, that is necessary to know the cost center from the execution comes and if the control leave a cost center, the previous cost center must be restored.

That way to implement the cost center stack is very clear, and it works in a consistent way with the cut operation (!). If a cut is done, then the choice points erased warranty the free in the stack of the cost centers saved in the variables `Prev_cc` and `Active_cc`.

3.2 Implementation of the profiler inside the engine

We have resumed how the instrumentation is done, now we will see the part of the profiler that is implemented inside the engine. The advantage of embed the profiler in the engine is that we can obtain a real idea that what happen when a probram is executed. In the case of ciao prolog, there are many code expansion and sintac sugar that could leave hidden in a high level profiler, and also its could have a poor performance.

When we configuring the Ciao Prolog system, at the installation time we must do that `DEBUG_LEVEL` have the value `profile` or `profile-debug` if alwo we need debug information. Thes values controls the compilation for profiling of the Ciao Prolog engine. These steps will embed the next hook functions in the correct places, and are declared in the file `engine initial.c`:

```
#if defined(PROFILE)
BOOL profile = FALSE;         /* profile program execution */
BOOL prof_include_time = FALSE; /* include time in profile */
void (*profile__hook_fail)(struct worker *w) = NULL;
void (*profile__hook_retry)(struct worker *w,
        int count_retry) = NULL;
void (*profile__hook_cut)(struct worker *w) = NULL;
void (*profile__hook_proceed)(void) = NULL;
void (*profile__hook_neck_proceed)(void) = NULL;
void (*profile__hook_call)(struct worker *w,
        struct definition *functor) = NULL;
#endif
```

The definitions of these variables are the next:

- `profile`: Flag indicating if we are doing profiling.
- `profile_include_time`: Flag indicating if we are doing time profiling.
- `profile__hook_fail`: Hook placed at fail time.
- `profile__hook_retry`: Hook placed at retry time.

- `profile__hook_cut`: Hook placed at cut time.
- `profile__hook_proceed`: This hook is only present to trace the execution.
- `profile__hook_neck_proceed`: At this moment is only present to trace the execution.
- `profile__hook_call`: Hook placed in a predicate call.

These functions are not implemented inside the engine folder, they are in a separated module that redefines these hooks to non empty functions, and these are implemented in `library(profiler)`. When the module `profiler_rt.pl` is used, in the initialization predicates, the empty hooks are set to concrete values of functions defineds here, as we can see in the `profiler__init()` function:

```
void profile__init(void)
{

...

  profile__hook_fail = profile__hook_fail_;
  profile__hook_retry = profile__hook_retry_;
  profile__hook_cut = profile__hook_cut_;
  profile__hook_proceed = profile__hook_proceed_;
  profile__hook_neck_proceed = profile__hook_neck_proceed_;
  profile__hook_call = profile__hook_call_;
}
```

Using this technique the engine is separated of the profiler and is more easy to maintaing and improve the code at future.

With the profiler, and because is a related tool, also has been implemented a tracer [DN94, Duc97]. This tracer is only to satisfy the requirements of the profiler and to debug it, have in mind that already have a tracer in the debuger.

3.3 Cost center Implementation

In the paragraphs above we mentioned what the cost centers are, but We don't have mention about how it has been implemented. In this section will be briefly to describe the more relevants aspects of that.

A cost center is a predicate marqued for profiling, and it contains information about the four counters of the box model (calls, redos, exits, fails), also have the total time spent by the cost center given. Also it carry a database with the predicates executed inside, each predicate is related with its low level counters, as are calls, retrys, nskips, cuts, scuts, and the time spent in execute itself (time_spent).

The implementation of the database in the predicate has been done using hash tables, these hash tables was implementeds by Bob Jenkins [Jen96], are public domain, and was adapted to let to work with Prolog in the profiler.

4 Using the profiler

To use the predicates that do the profiling, is necessary to include the module `library('profiler/profiler_utils')`, and optionally if we chan do concrete profiling, the high resolution timer module `library(hrtimer)`.

```
?- use_module(library('profiler/profiler_utils')).

yes
?- use_module(library(hrtimer)).

yes
?-
```

4.1 Example

Now we will describe an example to see how this tool can be used. Consider the `qsort1.pl` module listed above.

Suppose that we want to do profiling over `qsort1.pl`.

Once the package profiler has been included in such module, is necessary to load it in ciao:

```
?- use_module(qsort1).

yes
?-
```

We use the predicate `profile/1` to cumulate profiling information, where the first argument is the goal that we want to evaluate:

```
?- profile(qsort([1,3,9,4,7,5],X)).
{NOTE: Goal has success}

...

X = [1,3,4,5,7,9] ?

yes
?-
```

We are obvied the details that what happen in each cost center because the report is too long, however in the apendix is possible to look some examples that includes all the information.

To restart the profiler erasing the cumulated information we used the predicate `profile_reset/0`.

In the next tables we show a summary with the results of the profiler when the cost center is `append/3`.

These tests has been done in the next PC: Acer Travelmate C302XMi, Pentium M 1.6GHz, 512MB RAM memory, 60GB Hard disk and operating system Linux Mandrake 10.0 with kernel 2.4.27.

4.1.1 Flat profiler Information

Calls	Time(ms)	Type	Spec
21 - 32.31%	0.014701 - 44.29%	Emul	qsort1:split/4
13 - 20.00%	0.004890 - 14.73%	Emul	qsort1:qsort/2
10 - 15.38%	0.003044 - 9.17%	Emul	qsort1:prof$append/3
3 - 4.62%	0.002187 - 6.59%	Built	hiord_rt:call/1
3 - 4.62%	0.001682 - 5.07%	Emul	basiccontrol:metacall/3
1 - 1.54%	0.000837 - 2.52%	Emul	internals:$$$25/4
1 - 1.54%	0.000797 - 2.40%	Emul	internals:rt_module_exp/6
1 - 1.54%	0.000759 - 2.29%	Built	internals:$current_instance/5
1 - 1.54%	0.000667 - 2.01%	Emul	exceptions:$$$0/3
1 - 1.54%	0.000574 - 1.73%	C	internals:$current_clauses/2
1 - 1.54%	0.000509 - 1.53%	C	internals:$erase/1
1 - 1.54%	0.000492 - 1.48%	Emul	exceptions:$$$1/2
2 - 3.08%	0.000445 - 1.34%	Emul	basiccontrol:metacall2/2
1 - 1.54%	0.000381 - 1.15%	Emul	data_facts:$$$0/1
1 - 1.54%	0.000316 - 0.95%	Emul	exceptions:retract_catching/3
1 - 1.54%	0.000281 - 0.85%	C	basiccontrol:$metachoice/1
1 - 1.54%	0.000251 - 0.76%	Emul	data_facts:retract_fact_nb/1
1 - 1.54%	0.000214 - 0.64%	C	internals:$unlock_predicate/1
1 - 1.54%	0.000167 - 0.50%	Emul	data_facts:this_is_true/1
65 - 100%	0.033193 - 100%		Total

The flat profiler give us the detail of all the predicates that has been executed, including those that are internals to the system.

4.1.2 Overhead

Calls	Time(ms)	Spec
6 - 20%	0.003217 - 29.20%	profiler_rt:profile__hook_cc_call/1
6 - 20%	0.002334 - 21.19%	profiler_rt:profile__hook_cc_fail/1
6 - 20%	0.001989 - 18.05%	profiler_rt:profile__hook_cc_exit/2
6 - 20%	0.001772 - 16.09%	profiler_rt:profile__hook_cc_redo/1
6 - 20%	0.001705 - 15.48%	qsort1:append/3
30 - 100%	0.011017 - 100%	Total

These predicates are news and appears in the code instrumentation. To minimize the hidden efects the system is able of estimate the amount of time spent in do profiling. In this example we have the next overhead:

0.102829 ms	69.93%	Do profiling
0.011017 ms	7.49%	Running instrumentation code
0.113845 ms	77.43%	Total Overhead

4.1.3 Detailed information of the profiler

The profiler can show the result obtained for each cost center. In our example there are only two cost centers, and the summary of the detailed profiling is the next:

Calls	Time(ms)	Spec
55 - 84.62%	0.030149 - 90.83%	Beyond Cost Centers
10 - 15.38%	0.003044 - 9.17%	qsort1:append/3
65 - 100%	0.033193 - 100%	Total

If we leave that all the predicates in qsort be instrumented, we will obtain the next detailed profiling:

Calls	Time(ms)	Type	Spec
21 - 32.31%	0.015240 - 45.41%	Emul	qsort1:split/4
21 - 32.31%	0.010685 - 31.84%		Beyond Cost Centers
13 - 20.00%	0.004650 - 13.86%	Emul	qsort1:qsort/2
10 - 15.38%	0.002989 - 8.90%	Emul	qsort1:append/3
65 - 100%	0.033564 - 100%		Total

Now, to see how the profiler can be used to help us to improve a program, we will consider other qsort implementation, that we will place in the module `qsort2.pl`:

4.1.4 File `qsort2.pl`

```
:- module(qsort2, [qsort/2], [profiler]).

qsort(A, B) :-
qsort_(A, B, []).

qsort_([], R, R).
qsort_([X|L], R, R0) :-
split(L, X, L1, L2),
qsort_(L2, R1, R0),
qsort_(L1, R, [X|R1]).

split([],_,[],[]).
split([X|L],Y,[X|L1],L2) :-
X =< Y, !,
split(L,Y,L1,L2).
split([X|L],Y,L1,[X|L2]) :-
split(L,Y,L1,L2).
```

Calls	Time(ms)	Type	Spec
21 - 37.50%	0.011393 - 40.00%	Emul	qsort2:split/4
21 - 37.50%	0.011342 - 39.82%		Beyond Cost Centers
13 - 23.21%	0.004923 - 17.29%	Emul	qsort2:qsort_/3
1 - 1.79%	0.000822 - 2.89%	Emul	qsort2:qsort/2
56 - 100%	0.028480 - 100%		Total

If we compare this table with the previous example We can say that eliminating `append/3` the number of calls has been reduced, and in effect, the spent time.

In these examples, there is no need to include information about the amount of fails or redos, due to qsort don't have fails. But exists other

examples on with these kind of information could be useful. A very simple example is to compare two call modes to a predicate, is show as follows:

Suppose whe have the next module:

```
:- module(person1,_).

person(edison,   19).
person(fernando, 17).
person(david,    14).
person(julio,    15).
person(cesar,    18).
```

If we do the next query:

```
?- profile_reset,profile(person(cesar, X)).
```

The answer will be found directly, but if We do the next:

```
?- profile_reset,profile((person(A, X),A=cesar)).
```

The system will redo 4 times before to reach a success.

5 Conclussions and Future Work

Whe have say the advantages that the Ciao Prolog Profiler can give us, also se developed a basic example to know how to use that. However, this profiler can be improved. A mayor integration with prolog is desirable. Specifically the `profile_dump/0` predicate that shows the current status must be migrated to prolog.

Other pending task is to build examples about the use of the static analysis techniques of programs in which the profiler be used to compare results.

The profiler also can be used to guide the automatic program optimization, used in conjunction with static analysis techniques in the system bottle necks detecteds with it.

References

[Deb83] S. K. Debray. Profiling prolog programs. *Software Practice and Experience*, 18(9):821–839, 1983.

[DN94] M. Ducassé and J. Noyé. Logic programming environments: Dynamic program analysis and debugging. *Journal of Logic Programming*, 19,20:351–384, 1994.

[DN00] M. Ducassé and J. Noyé. Tracing prolog programs by source instrumentation is efficient enough. *Journal of Logic Programming*, 43:157–172, 2000.

[Duc97] M. Ducassé. Optium: An extendable trace analyser for prolog. Technical report rr-3257, IRISA/INSA, 1997.

[Jen96] Bob Jenkins. An in-memory hash table. Available at `http://burtleburtle.net/bob/hash/index.html` Public Domain, 1996.

[Mat94] A. B. Matos. A Matrix Model for the flow of Control in Prolog Programs with Applications to Profiling. *Software Practice and Experience*, 24(8):729–746, August 1994.

[Ras00] Nick Rasmussen. High resolution timing for the linux kernel release 0.6.1. Available at `http://www.cs.wisc.edu/paradyn/libhrtime/` Public Domain, 2000. This work was done for the Paradyn Parallel Performance Tools project at the University of Wisconsin `http://www.cs.wisc.edu/paradyn/`.

[RGM98] S. A. Jarvis R. G. Morgan. Profiling large-scale lazy functional programs. *Journal of Functional Programing*, 8(3):201–237, May 1998.

A Source code of the Profiler

Note that inside the engine only are the hooks that after are defined to concrete functions that are implemented in separated files in `C` language.

Next we will show the code that implements the `profiler` package, and also the `hrtimer` module that have the high resolution time implementation and the `hashtable` module that implements the hash tables useds to hold the cost centers.

A.1 `profiler` Package

A.1.1 File `profiler.pl`

```
% The Profile package

:- discontiguous cost_center/2.
:- multifile cost_center/2.
%:- data cost_center/2.

:- load_compilation_module(library('profiler/profiler_tr')).

:- use_module(library('profiler/profiler_rt')).

:- add_sentence_trans(profiler_def/3).

:- op(1150, fx, [profile,noprofile]).
```

A.1.2 File `profiler_rt.pl`

```
:- module(profiler_rt,[
profile__hook_cc_call/1,
profile__hook_cc_exit/2,
profile__hook_cc_redo/1,
profile__hook_cc_fail/1,
profile_init/0
], [foreign_interface]).


:- use_module(library(hashtable)).

%:- initialization(set_option(walltime)).
:- initialization(profile_init).

:- foreign_inline("
#define PROFILE
#include \"ciao_prolog.h\"
#include \"datadefs.h\"
#include \"support.h\"
#include \"predtyp.h\"
#include \"profile_defs.h\"
#include \"profiler.h\"

").

:- use_foreign_source(library(profiler)).
```

```
:- true pred profile_init + (native(prolog_profile_init)).

:- foreign_inline(profile_init/0,
"
BOOL prolog_profile_init(Arg)
Argdecl;
{
  profile__init();
  profile__hook_cc_call  =
    GET_DEFINITION(\"profiler_rt:profile__hook_cc_call\", 1);
  profile__hook_cc_redo  =
    GET_DEFINITION(\"profiler_rt:profile__hook_cc_redo\", 1);
  profile__hook_cc_redo_ =
    GET_DEFINITION(\"profiler_rt:profile__hook_cc_redo_\",1);
  profile__hook_cc_exit  =
    GET_DEFINITION(\"profiler_rt:profile__hook_cc_exit\", 2);
  profile__hook_cc_fail  =
    GET_DEFINITION(\"profiler_rt:profile__hook_cc_fail\", 1);
  profile__hook_cc_fail_ =
    GET_DEFINITION(\"profiler_rt:profile__hook_cc_fail_\",1);
  return TRUE;
}
").

profile__hook_cc_fail(_).
profile__hook_cc_fail(Prev_cc) :-
profile__hook_cc_fail_(Prev_cc).

profile__hook_cc_redo(_).
profile__hook_cc_redo(Active_cc) :-
profile__hook_cc_redo_(Active_cc).

:- true pred profile__hook_cc_call(go(Prev_cc)) :: int +
   (native(prolog_profile__hook_cc_call)).

:- foreign_inline(profile__hook_cc_call/2,
"BOOL prolog_profile__hook_cc_call(Arg)
    Argdecl;
{
  if(profile) {
    /* This will cause not count recursive calls */
    if(prev_cc!=active_cc)
      active_cc->calls++;
    ShowFuncPoint(\"cc-call\",0,0, active_cc->functor);
    return cunify(Arg,PointerToTerm(prev_cc),X(0));
  }
  else
    return TRUE;
}
").

:- true pred profile__hook_cc_redo_(in(Active_cc)) :: int +
   (native(prolog_profile__hook_cc_redo)).

:- foreign_inline(profile__hook_cc_redo_/1,
"BOOL prolog_profile__hook_cc_redo(Arg)
    Argdecl;
{
  if(profile) {
    struct cost_center_item *cci;
    REGISTER struct cost_center *cc=active_cc;
    DEREF(X(0),X(0));
```

```
    cc = active_cc;
    active_cc = (struct cost_center *)TermToPointer(X(0));
    /* This will cause not count recursive calls */
    if(cc!=active_cc)
      active_cc->redos++;
    cci = get_cc_item(active_cc->cc_item_table,
      profile__hook_cc_redo);
    node_stack[node_stack_point].last_cci = cci;
    last_cci = cci;
    ShowFuncPoint(\"cc-redo\",0,0, active_cc->functor);
  }
  return FALSE;
}
").

:- true pred profile__hook_cc_exit(in(Prev_cc), go(Active_cc)) :: int
   * int + (native(prolog_profile__hook_cc_exit)).

:- foreign_inline(profile__hook_cc_exit/1,
"BOOL prolog_profile__hook_cc_exit(Arg)
    Argdecl;
{
  if(profile) {
    REGISTER struct cost_center *cc=active_cc;
    DEREF(X(0),X(0));
    ShowFuncPoint(\"cc-exit\",0,0, cc->functor);
    active_cc = (struct cost_center *)TermToPointer(X(0));
    /* This will cause not count recursive calls */
    if(cc!=active_cc)
      cc->exits++;
    return cunify(Arg,PointerToTerm(cc),X(1));
  }
  else
    return TRUE;
}
"
).

:- true pred profile__hook_cc_fail_(in(Prev_cc)) :: int +
   (native(prolog_profile__hook_cc_fail)).

:- foreign_inline(profile__hook_cc_fail_/1,
"BOOL prolog_profile__hook_cc_fail(Arg)
    Argdecl;
{
  if(profile) {
    REGISTER struct cost_center *cc=active_cc;
    DEREF(X(0),X(0));
    ShowFuncPoint(\"cc-fail\",0,0, cc->functor);
    active_cc = (struct cost_center *)TermToPointer(X(0));
    /* This will cause not count recursive calls */
    if(cc!=active_cc)
      cc->fails++;
  }
  return FALSE;
}
").
```

A.1.3 File profiler_tr.pl

```
:- module(profiler_tr,[profiler_def/3],[assertions]).
```

```
:- use_module(library(lists),[append/3,length/2]).
:- use_module(library('profiler/profiler_utils'),
   [profile_reset/0]).

% Note: You must not use the use_module directive here with
% a module suitable to be profiled.

%:- use_module(library(patterns)).

:- data instrumented/3.
:- data profile/3.
:- data noprofile/3.

% do_profile(F,N,M) :-
%  (   profile(PatternF,PatternN,M),
%      match_pattern_pred(PatternF,F),
%      match_pattern_pred(PatternN,N)
%  ;
%      \+ profile(_,_,_)
%  ),
%  \+ (
%      noprofile(NoPatternF,NoPatternN,M),
%      match_pattern_pred(NoPatternF,F),
%      match_pattern_pred(NoPatternN,N)
%  ).

do_profile(F,N,M) :-
(   profile(F,N,M)
;
    \+ profile(_,_,_)
),
\+ (
    noprofile(F,N,M)
).

ipred_recursivity(OrigFunctor,NewFunctor,Arity,Pred,NewPred) :-
(
    functor(Pred,OrigFunctor,Arity),
    Pred =.. [_|Args] ->
    NewPred =.. [NewFunctor|Args]
;
    NewPred = Pred
).

avoid_recursivity(OrigFunctor, NewFunctor, Arity,
(Pred,Body), (NewPred,NewBody)) :-
ipred_recursivity(OrigFunctor,NewFunctor,Arity,Pred,NewPred),
avoid_recursivity(OrigFunctor,NewFunctor,Arity,Body,NewBody).
avoid_recursivity(OrigFunctor,NewFunctor,Arity,Pred,NewPred) :-
ipred_recursivity(OrigFunctor,NewFunctor,Arity,Pred,NewPred).

profiler_def(A,B,M) :-
profiler_def_(A,B,M).

% profiler_def_(0,Clause,_M) :-
%  Clause = .

profiler_def_(end_of_file, end_of_file, M) :-
retractall_fact(instrumented(_,_,M)),
retractall_fact(profile(_,_,M)).

profiler_def_((:- profile(F/N)), [], M) :-
```

```
assertz_fact(profile(F,N,M)).
profiler_def_((:- profile((A,B))), [], M) :-
profiler_def((:- profile(A)), [], M),
profiler_def((:- profile(B)), [], M).

profiler_def_((:- noprofile(F/N)), [], M) :-
assertz_fact(noprofile(F,N,M)).
profiler_def_((:- noprofile((A,B))), [], M) :-
profiler_def((:- noprofile(A)), [], M),
profiler_def((:- noprofile(B)), [], M).

profiler_def_(OrigClause, Clauses, M) :-
(   OrigClause = (Head :- Body) ->
    true
;
    (   OrigClause = (:- _) ->
fail
    ;
Head = OrigClause,
Body = true
    )
),
functor(Head,F,N),
atom(F),
(   do_profile(F,N,M) ->
    Head =.. [_|Args],
    atom_concat('prof$',F, ProfFunctor),
    ProfHead =.. [ProfFunctor|Args],
    (   avoid_recursivity(F,ProfFunctor,N,Body,NewBody) -> true
    ;
display('problems in avoid_recursivity\n')
    ),
    (   instrumented(F,N,M) ->
Clauses = [ ( ProfHead :- NewBody ) ]
    ;
length(DummyArgs,N),
DummyHead =.. [F|DummyArgs],
DummyBody =.. [ProfFunctor|DummyArgs],
atom_concat(M,':',P0), atom_concat(P0,F,P),
Clauses = [ ( cost_center(P,N) ),
            ( DummyHead :-
      profile__hook_cc_call(Prev_cc),
      profile__hook_cc_fail(Prev_cc),
      DummyBody,
      profile__hook_cc_exit(Prev_cc,Active_cc),
      profile__hook_cc_redo(Active_cc) ),
    ( ProfHead :- NewBody ) ],
assertz_fact(instrumented(F,N,M))
    ),
;
    Clauses=OrigClause
).

:- comment(version_maintenance,dir('../../version')).

:- comment(version(1*11+236,2004/06/08,13:05*05+'CEST'), "Added a
   predicate called avoid_recursion, to optimize the instrumentation
   of code in recursive calls.  (Edison Mera)").
```

A.1.4 File `profiler.c`

```
#if !defined(PROFILE)
#define PROFILE

#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "datadefs.h"
#include "initial.h"
#include "predtyp.h"
#include "profile_defs.h"
#include "timing_defs.h"
#include "alloc_defs.h"
#include "support.h"
#include "support_defs.h"
#include "debug.h"
#include "builtin_defs.h"
#include "profiler.h"

extern BOOL prof_include_time;

static int compare_calls(const void *arg1, const void *arg2);
static int compare_clicks(const void *arg1, const void *arg2);
static int compare_cci_calls(const void *arg1, const void *arg2);
static int compare_cci_clicks(const void *arg1, const void *arg2);
static int compare_ccc_calls(const void *arg1, const void *arg2);
static int compare_ccc_clicks(const void *arg1, const void *arg2);

struct cost_center_clip {
  struct cost_center *cc;
  long int realsize;
  long int realsize_oh;
  struct profile_currents total;
  struct profile_currents overhead;
};

struct ht_tab *cost_center_table = NULL;
struct cost_center *active_cc = NULL;
struct cost_center *default_cc = NULL;
struct cost_center *prev_cc = NULL;
struct cost_center_item *last_cci = NULL;

struct definition *last_called_predicate = NULL;
struct definition *profile__hook_cc_call = NULL;
struct definition *profile__hook_cc_redo = NULL;
struct definition *profile__hook_cc_redo_ = NULL;
struct definition *profile__hook_cc_exit = NULL;
struct definition *profile__hook_cc_fail = NULL;
struct definition *profile__hook_cc_fail_ = NULL;
int node_stack_point = 0;
int node_stack_size = CALL_STACK_INITIAL_SIZE;
struct node_stack_item *node_stack = NULL;
ENG_LINT click_last_addition = 0;
ENG_LINT click_ini_profiling = 0;
ENG_LINT click_start = 0;
# if defined(PROFILE__PROFTIME)
ENG_LINT click_profiling = 0;  /* time (in clicks) doing profiling (overhead) */
ENG_LINT click_end_profiling = 0;
# endif

/* #if defined(PROFILE__TRACER) */
```

```
int profile_trace = 0;
/* #endif */

# if !defined(PROFILE__USE_TIMESTAMP)
ENG_LINT (**profclick)(void) = &userclick;
ENG_LINT *profclockfreq = &stats.userclockfreq;
# endif

/* run time hooks */

void profile__hook_fail_(struct worker *w) {
  if(profile) {
    PROFILE__TIME_INI;
    if(node_stack_point >= 0) {
      struct cost_center_item *cce=NULL;   /* EMM */
      unsigned int choice, prev_choice, curr_choice;
      /* now redo from the node_stack up to the first actual choice point */
      ComputeA(w->local_top,w->node);
      /* choice = (TAGGED *)w->node->local_top - w->stack_start; */
      choice = ChoiceToInt(w->node);
      prev_choice = node_stack[node_stack_point].choice;
      node_stack_point--;
      while(((int)(node_stack[node_stack_point].choice)>=choice)&&(node_stack_point>=0))
        {
          curr_choice = node_stack[node_stack_point].choice;
          cce = node_stack[node_stack_point].first_cci;
          printf("%d\n", profile_trace);
          ShowFuncPoint(" unkn", node_stack_point, choice,
            node_stack[node_stack_point].last_cci->functor);
          if(prev_choice >= curr_choice) {
            ShowFuncPoint("+skip", node_stack_point, choice,
              node_stack[node_stack_point].last_cci->functor);
            cce->functor->prof.skips++;
            cce->prof.skips++;
          } else {
            ShowFuncPoint("+cuts", node_stack_point, choice,
              node_stack[node_stack_point].last_cci->functor);
          }
          prev_choice = curr_choice;
          node_stack[node_stack_point].first_cci=NULL;
          node_stack[node_stack_point].last_cci=NULL;
          node_stack_point--;
        }
    }
    PROFILE__TIME_END;
  };
}

void profile__hook_retry_(Arg, count_retry)
     int count_retry;
     Argdecl;
{
  if(profile) {
    PROFILE__TIME_INI;
    if(node_stack_point>=0) {
      struct cost_center_item *cci=NULL;
      cci = node_stack[node_stack_point].last_cci;
      if(cci!=NULL) {
        ShowFuncPoint("retry", node_stack_point,
          node_stack[node_stack_point].choice, cci->functor);
        if(count_retry) { /* w->node->next_alt */
          ShowClauseNumber;
```

```
          cci->functor->prof.retrys++;
          cci->prof.retrys++;
        }
        else
          ShowNoMoreAlts;
      }
    }
    PROFILE__TIME_END;
  }
}

void profile__hook_cut_(struct worker *w) {
  if(profile) {
    PROFILE__TIME_INI;
    if(node_stack_point >= 0) {
      struct cost_center_item *first_cci=NULL, *last_cci;   /* EMM */
      unsigned int choice, prev_choice, curr_choice, skips=0;
      choice = ChoiceToInt(w->node);
      last_cci = node_stack[node_stack_point].last_cci;
      prev_choice = node_stack[node_stack_point].choice;
      while(((int)(node_stack[node_stack_point].choice)>=choice)&&(node_stack_point>=0))
        {
          curr_choice = node_stack[node_stack_point].choice;
          first_cci = node_stack[node_stack_point].first_cci;
          if(prev_choice > curr_choice) {
            ShowFuncPoint("+skip", node_stack_point, curr_choice,
                          node_stack[node_stack_point].last_cci->functor);
            first_cci->functor->prof.skips++;
            first_cci->prof.skips++;
            skips++;
          } else if (prev_choice < curr_choice) {
            ShowFuncPoint("+warning: undetected cutn", node_stack_point,
                          curr_choice, node_stack[node_stack_point].last_cci->functor);
          }
          prev_choice = curr_choice;
          node_stack[node_stack_point].first_cci=NULL;
          node_stack[node_stack_point].last_cci=NULL;
          node_stack_point--;
        }
      if(node_stack_point >= 0) {
        last_cci = node_stack[node_stack_point].last_cci;
        ShowFuncPoint("+cut level", node_stack_point, choice, last_cci->functor);
        ShowSkipNodes(skips);
        last_cci->functor->prof.nskips+=skips;
        last_cci->prof.nskips+=skips;
        if(skips)
          last_cci->functor->prof.scuts++;
        last_cci->prof.scuts++;
      } else {
        last_cci->functor->prof.cuts++;
        last_cci->prof.cuts++;
      }
    }
    PROFILE__TIME_END;
  }
}

void profile__hook_proceed_(void)
{
  if(profile) {
    struct cost_center_item *cce=node_stack[node_stack_point].last_cci;
    if(cce)
```

```
      ShowFuncPoint("+proc", node_stack_point,
        node_stack[node_stack_point].choice, cce->functor);
  }
}

void profile__hook_neck_proceed_(void)
{
  if(profile) {
    struct cost_center_item *cce=node_stack[node_stack_point].last_cci;
    if(cce)
      ShowFuncPoint("+nprc", node_stack_point,
        node_stack[node_stack_point].choice, cce->functor);
  }
}

void profile__hook_call_(w,functor)
     struct worker *w;
     struct definition *functor;
{
  if(profile) {
    PROFILE__TIME_INI;
    {
      struct node_stack_item *csn;
      unsigned int choice;
      ENG_LINT profdiff;
      if(functor==profile__hook_cc_call) {
        prev_cc = active_cc;
        active_cc = get_cost_center(cost_center_table, last_called_predicate);
      }
      {
        ComputeA(w->local_top,w->node);
        choice=ChoiceToInt(w->node);
        ShowFuncPoint("+call", node_stack_point, choice, functor);
        if (prof_include_time) {
          profdiff = click_ini_profiling - click_last_addition;
          if(last_called_predicate!=NULL) {
            last_called_predicate->prof.time_spent += profdiff;
            last_cci->prof.time_spent += profdiff;
          }
          else {
            click_start += profdiff;
          }
        }
        if(last_called_predicate!=NULL) {
          last_called_predicate->prof.calls++;
            last_cci->prof.calls++;
}
        if((node_stack_point<0)||node_stack[node_stack_point].choice!=choice) {
          node_stack_point++;
          if(node_stack_point>=node_stack_size) {
            struct node_stack_item *new_node_stack;
            int new_node_stack_size = 2 * node_stack_size;
            printf("{WARNING: Increasing node stack to %d.}\n",
              new_node_stack_size);
            new_node_stack = (struct node_stack_item *)checkalloc(new_node_stack_size
              * sizeof(struct node_stack_item));
            memcpy(new_node_stack, node_stack, node_stack_size
              * sizeof(struct node_stack_item));
            checkdealloc((TAGGED *)node_stack, node_stack_size
              * sizeof(struct node_stack_item));
            node_stack = new_node_stack;
            node_stack_size = new_node_stack_size;
```

```
          }
          csn=&node_stack[node_stack_point];
          csn->first_cci = get_cc_item(active_cc->cc_item_table,functor);
          csn->choice=choice;
          if(functor!=profile__hook_cc_redo_) {
            csn->last_cci = csn->first_cci;
            last_cci = csn->last_cci;
          }
        }
        else {
          csn=&node_stack[node_stack_point];
          if(functor!=profile__hook_cc_redo_) {
            csn->last_cci = get_cc_item(active_cc->cc_item_table,functor);
            last_cci = csn->last_cci;
          }
        }
        last_called_predicate = functor;
      }
      click_last_addition = click_ini_profiling;
    }
    PROFILE__TIME_END;
  }
}

void profile__init(void)
{
  if(cost_center_table) {
    ht_destroy(cost_center_table);
  }
  cost_center_table = ht_create(8);
  default_cc = add_cost_center(cost_center_table, NULL);
  active_cc = default_cc;
  node_stack_point = -1; /* empty stack */
  if(node_stack) {
    checkdealloc((TAGGED *)node_stack,node_stack_size * sizeof(struct node_stack_item));
  }
  node_stack_size = CALL_STACK_INITIAL_SIZE;
  node_stack = (struct node_stack_item *)checkalloc(node_stack_size
    * sizeof(struct node_stack_item));
  node_stack[0].first_cci=NULL;
  profile__hook_fail = profile__hook_fail_;
  profile__hook_retry = profile__hook_retry_;
  profile__hook_cut = profile__hook_cut_;
  profile__hook_proceed = profile__hook_proceed_;
  profile__hook_neck_proceed = profile__hook_neck_proceed_;
  profile__hook_call = profile__hook_call_;
}

struct cost_center *add_cost_center(cct, functor)
     struct ht_tab * cct;
     struct definition *functor;
{
  struct cost_center *r;
  r = (struct cost_center *)checkalloc(sizeof(struct cost_center));
  r->functor=functor;
  r->calls=r->redos=r->fails=r->exits=0;
  r->time_spent=0;
  r->cc_item_table=ht_create(8);
  ht_add(cct,(ub1 *)(&(r->functor)),sizeof(functor),r);
  return r;
}
```

```
struct cost_center *get_cost_center(cct, functor)
     struct ht_tab *cct;
     struct definition *functor;
{
  if(ht_find(cct,(ub1 *)(&functor),sizeof(functor)))
    return (struct cost_center *)ht_stuff(cct);
  else
    return add_cost_center(cct, functor);
}

struct cost_center_item *add_cc_item(ht,functor)
     struct ht_tab *ht;
     struct definition *functor;
{
  struct cost_center_item *e;
  e = (struct cost_center_item *)checkalloc(sizeof(struct cost_center_item));
  e->functor = functor;
  PROFILE__RESET_PROF(e->prof);
  ht_add(ht,(ub1 *)(&(e->functor)),sizeof(functor),(void *)e);
  return e;
}

struct cost_center_item *get_cc_item(ht,functor)
     struct ht_tab *ht;
     struct definition *functor;
{
  if(ht_find(ht,(ub1 *)(&functor),sizeof(functor)))
    return (struct cost_center_item *)ht_stuff(ht);
  else
    return add_cc_item(ht,functor);
}

void empty_hashtable(tab,size)
     struct ht_tab * tab;
{
  while(ht_first(tab)) {
    checkdealloc((TAGGED *)ht_stuff(tab),size);
    ht_del(tab);
  }
}

void empty_cost_center_table(cct)
     struct ht_tab * cct;
{
  struct cost_center *r;
  while(ht_first(cct)) {
    r = (struct cost_center *)ht_stuff(cct);
    empty_hashtable(r->cc_item_table, sizeof(struct cost_center_item));
    ht_destroy(r->cc_item_table);
    checkdealloc((TAGGED *)r,sizeof(struct cost_center));
    ht_del(cct);
  }
}

void reset_cc_item_table(cc_item_table)
     struct ht_tab * cc_item_table;
{
  struct cost_center_item *r;
  if(ht_first(cc_item_table)) do {
    r = (struct cost_center_item *)ht_stuff(cc_item_table);
    PROFILE__RESET_PROF(r->prof);
  }
```

```
  while(ht_next(cc_item_table));
}

void reset_cost_center_table(cct)
     struct ht_tab *cct;
{
  struct cost_center *r;
   if(ht_first(cct)) do {
    r = (struct cost_center *)ht_stuff(cct);
    r->calls=r->redos=r->fails=r->exits=0;
    r->time_spent=0;
    reset_cc_item_table(r->cc_item_table);
  }
  while (ht_next(cct));
}

void show_func_point(label, call_point, choice, functor)
  char *label;
  int call_point;
  unsigned int choice;
  struct definition *functor;
{
  printf("%s %d-%d ", label, call_point, choice);
  if(functor) {
    printf("%-7s ", predicate_type(functor->predtyp));
    if(IsString(functor->printname))
      printf("%s/", GetString(functor->printname));
    else
      printf("*** Unknown term ***/");
    printf("%d\n", functor->arity);
  }
  else
    printf("*** Null functor ***\n");
}

BOOL functor_have_overhead(cct, functor)
     struct ht_tab * cct;
     struct definition *functor;
{
  return
    functor==profile__hook_cc_call ||
    functor==profile__hook_cc_redo ||
    functor==profile__hook_cc_redo_ ||
    functor==profile__hook_cc_exit ||
    functor==profile__hook_cc_fail ||
    functor==profile__hook_cc_fail_ ||
    ht_exists(cct, (ub1 *)(&functor),sizeof(functor));
}

void profile_ccc_dump(ccc)
     struct cost_center_clip *ccc;
{
  struct ht_tab *t;
  struct cost_center_item **cci_table, **cci_table_oh;
  struct cost_center_item *e;
  int (*compare_cci)(const void *, const void *);
  long int i, j;
  if(ccc->cc->functor) {
    printf("Cost Center     %s/%d\n", GetString(ccc->cc->functor->printname),
      ccc->cc->functor->arity);
  }
  else
```

```
  printf("Beyond Cost Centers \n");
printf("        ================================================================================================\n");
printf("        Calls=%-7ld Redos=%-7ld Exits=%-7ld Fails=%-7ld\n", ccc->cc->calls,
       ccc->cc->redos,ccc->cc->exits,ccc->cc->fails);
printf("        ================================================================================================\n");
printf("        Calls           Skips   NSkips  Cuts    SCuts   Retrys  Time(ms) -rough         Type    Spec\n");
printf("        =============== ======= ======= ======= ======= ======= ======================= ====    ====\n");
t=ccc->cc->cc_item_table;
/* first calculate the totals */
/* Make a table with room for them */
cci_table = (struct cost_center_item **)checkalloc(ccc->realsize
  * sizeof(struct cost_center_item *));
cci_table_oh = (struct cost_center_item **)checkalloc((ccc->realsize_oh)
  * sizeof(struct cost_center_item *));
i = 0;
j = 0;
if(ht_first(t)) do {
  e = (struct cost_center_item *)ht_stuff(t);
  if(e->prof.calls) {
    if(functor_have_overhead(cost_center_table, e->functor))
      cci_table_oh[j++] = e;
    else
      cci_table[i++] = e;
  }
} while(ht_next(t));
if(prof_include_time)
  compare_cci = compare_cci_clicks;
else
  compare_cci = compare_cci_calls;
qsort(cci_table, ccc->realsize, sizeof(struct cost_center_item *), compare_cci);
qsort(cci_table_oh, ccc->realsize_oh, sizeof(struct cost_center_item *), compare_cci);
i = ccc->realsize;
while(i>0){
  e = cci_table[--i];
  printf("        %7ld %6.2f%% %7ld %7ld %7ld %7ld %7ld %12f (%6.2f%%)  %s  %s/%d\n",
         e->prof.calls,
         (((double)e->prof.calls)*100.0)/(double)ccc->total.calls,
         e->prof.skips,
         e->prof.nskips,
         e->prof.cuts,
         e->prof.scuts,
         e->prof.retrys,
         (((double)e->prof.time_spent)/PROFILE__GET_FREQ)*1.0e3,
         (((double)e->prof.time_spent)*100.0)/(double)ccc->total.time_spent,
         predicate_type(e->functor->predtyp),
         GetString(e->functor->printname),
         e->functor->arity);
}
printf("        =============== ======= ======= ======= ======= ======= ======================= ====    ====\n");
printf("        %7ld %6.2f%% %7ld %7ld %7ld %7ld %7ld %12f (%6.2f%%)          Total\n",
       ccc->total.calls, 100.0, ccc->total.skips, ccc->total.nskips,
       ccc->total.cuts, ccc->total.scuts, ccc->total.retrys,
       (double)ccc->total.time_spent/PROFILE__GET_FREQ*1.0e3, 100.0);
printf("        %ld predicates called\n\n", ccc->realsize);
j = ccc->realsize_oh;
if(j)
{
  printf("Overhead:\n");
  printf("        Calls           Skips   NSkips  Cuts    SCuts   Retrys  Time(ms) -rough         Type    Spec\n");
  printf("        =============== ======= ======= ======= ======= ======= ======================= ====    ====\n");
  while(j>0){
    e = cci_table_oh[--j];
```

```
      printf("        %7ld %6.2f%% %7ld %7ld %7ld %7ld %7ld %12f (%6.2f%%)  %s  %s/%d\n",
             e->prof.calls,
             (((double)e->prof.calls)*100.0)/(double)ccc->overhead.calls,
             e->prof.skips,
             e->prof.nskips,
             e->prof.cuts,
             e->prof.scuts,
             e->prof.retrys,
             (((double)e->prof.time_spent)/PROFILE__GET_FREQ)*1.0e3,
             (((double)e->prof.time_spent)*100.0)/(double)ccc->overhead.time_spent,
             predicate_type(e->functor->predtyp),
             GetString(e->functor->printname),
             e->functor->arity);
    }
    printf("        =============== ======= ======= ======= ======= ======= ====================== ====    ====\n");
    printf("        %7ld %6.2f%% %7ld %7ld %7ld %7ld %7ld %12f (%6.2f%%)          Total\n",
           ccc->overhead.calls, 100.0, ccc->overhead.skips, ccc->overhead.nskips,
           ccc->overhead.cuts, ccc->overhead.scuts, ccc->overhead.retrys,
           (double)ccc->overhead.time_spent/PROFILE__GET_FREQ*1.0e3, 100.0);
    printf("        %ld instrumentation predicates called\n", ccc->realsize_oh);
    printf("        %.2f%% executing instrumentation code\n\n",
           ((double)ccc->overhead.time_spent)/(ccc->total.time_spent
           + ccc->overhead.time_spent)*100.0);
  }
  else
    printf("{NOTE: Don't have overhead caused by instrumentation predicates}\n");
  checkdealloc((TAGGED *)cci_table_oh, ccc->realsize_oh*sizeof(struct cost_center_item *));
  checkdealloc((TAGGED *)cci_table, ccc->realsize*sizeof(struct cost_center_item *));
}

static int compare_cci_calls(arg1, arg2)
     const void *arg1, *arg2;
{
  struct cost_center_item **cce1, **cce2;

  cce1 = (struct cost_center_item **)arg1;
  cce2 = (struct cost_center_item **)arg2;

  if ((*cce1)->prof.calls > (*cce2)->prof.calls)
    return (1);
  if ((*cce1)->prof.calls < (*cce2)->prof.calls)
    return (-1);
  return (0);
}

static int compare_cci_clicks(arg1, arg2)
     const void *arg1, *arg2;
{
  struct cost_center_item **cce1, **cce2;

  cce1 = (struct cost_center_item **)arg1;
  cce2 = (struct cost_center_item **)arg2;

  if ((*cce1)->prof.time_spent > (*cce2)->prof.time_spent)
    return (1);
  if ((*cce1)->prof.time_spent < (*cce2)->prof.time_spent)
    return (-1);
  return (0);
}

void profile_detail_dump(void)
{
```

```
struct cost_center_item *e;
struct cost_center_clip *ccc_table, *c;
struct cost_center *cc;
struct ht_tab *t;
int (*compare_ccc)(const void *, const void *);
long int cc_calls=0,
  cc_redos=0,
  cc_exits=0,
  cc_fails=0;
long int i, realsize=0;
struct profile_currents total;

PROFILE__RESET_PROF(total);
ccc_table = (struct cost_center_clip *)checkalloc(ht_count(cost_center_table)
  * sizeof(struct cost_center_clip));
if(ht_first(cost_center_table)) do {
  cc = (struct cost_center *)ht_stuff(cost_center_table);
  t=cc->cc_item_table;
  /* first calculate the totals */
  ccc_table[realsize].realsize=0;
  ccc_table[realsize].total.skips=0;
  ccc_table[realsize].total.nskips=0;
  ccc_table[realsize].total.cuts=0;
  ccc_table[realsize].total.scuts=0;
  ccc_table[realsize].total.retrys=0;
  ccc_table[realsize].total.calls=0;
  ccc_table[realsize].total.time_spent=0;

  ccc_table[realsize].realsize_oh=0;
  ccc_table[realsize].overhead.skips=0;
  ccc_table[realsize].overhead.nskips=0;
  ccc_table[realsize].overhead.cuts=0;
  ccc_table[realsize].overhead.scuts=0;
  ccc_table[realsize].overhead.retrys=0;
  ccc_table[realsize].overhead.calls=0;
  ccc_table[realsize].overhead.time_spent=0;

  if(ht_first(t)) do {
    e = (struct cost_center_item *)ht_stuff(t);
    if(e->prof.calls) {
      if(functor_have_overhead(cost_center_table, e->functor)) {
        ccc_table[realsize].realsize_oh++;
        ccc_table[realsize].overhead.calls += e->prof.calls;
        ccc_table[realsize].overhead.skips += e->prof.skips;
        ccc_table[realsize].overhead.nskips += e->prof.nskips;
        ccc_table[realsize].overhead.cuts += e->prof.cuts;
        ccc_table[realsize].overhead.scuts += e->prof.scuts;
        ccc_table[realsize].overhead.retrys += e->prof.retrys;
        ccc_table[realsize].overhead.time_spent += e->prof.time_spent;
      }
      else
      {
        ccc_table[realsize].realsize++;
        ccc_table[realsize].total.calls += e->prof.calls;
        ccc_table[realsize].total.skips += e->prof.skips;
        ccc_table[realsize].total.nskips += e->prof.nskips;
        ccc_table[realsize].total.cuts += e->prof.cuts;
        ccc_table[realsize].total.scuts += e->prof.scuts;
        ccc_table[realsize].total.retrys += e->prof.retrys;
        ccc_table[realsize].total.time_spent += e->prof.time_spent;
      }
    }
```

```
  } while(ht_next(t));
  if(ccc_table[realsize].total.calls+ccc_table[realsize].overhead.calls != 0) {
    ccc_table[realsize].cc = cc;
    realsize++;
  }
} while(ht_next(cost_center_table));

if(realsize<=1) {
  printf("{NOTE: No cost centers has been specified}\n");
}
else {
  if(prof_include_time)
    compare_ccc = compare_ccc_clicks;
  else
    compare_ccc = compare_ccc_calls;
  qsort(ccc_table, realsize, sizeof(struct cost_center_clip), compare_ccc);
  i = realsize;
  while(i>0) {
    --i;
    profile_ccc_dump(&ccc_table[i]);
    total.calls += ccc_table[i].total.calls;
    total.skips += ccc_table[i].total.skips;
    total.nskips += ccc_table[i].total.nskips;
    total.cuts += ccc_table[i].total.cuts;
    total.scuts += ccc_table[i].total.scuts;
    total.retrys += ccc_table[i].total.retrys;
    total.time_spent += ccc_table[i].total.time_spent;
    cc_calls += ccc_table[i].cc->calls;
    cc_redos += ccc_table[i].cc->redos;
    cc_exits += ccc_table[i].cc->exits;
    cc_fails += ccc_table[i].cc->fails;
  }
  printf("Detailed profiling resume:\n");
  printf("================================================================================\n");
  printf("Calls=%-7ld Redos=%-7ld Exits=%-7ld Fails=%-7ld\n",
     cc_calls, cc_redos, cc_exits, cc_fails);
  printf("================================================================================\n");
  printf("Calls           Skips   NSkips  Cuts    SCuts   Retrys  ");
  if (prof_include_time) printf("Time(ms) -rough         ");
  printf("Type    Spec\n");
  printf("=============== ======= ======= ======= ======= ======= ");
  if (prof_include_time) printf("======================== ");
  printf("====    ====\n");
  i = realsize;
  while(i>0){
    c = &ccc_table[--i];
    printf("%7ld %6.2f%% %7ld %7ld %7ld %7ld %7ld ",
           c->total.calls,
           (((double)c->total.calls)*100.0)/(double)total.calls,
           c->total.skips,
           c->total.nskips,
           c->total.cuts,
           c->total.scuts,
           c->total.retrys);
    if (prof_include_time)
      printf("%12f (%6.2f%%)  ",
        (((double)c->total.time_spent)/PROFILE__GET_FREQ)*1.0e3,
        (((double)c->total.time_spent)*100.0)/(double)total.time_spent);
    if (c->cc->functor)
      printf("%s  %s/%d\n",
             predicate_type(c->cc->functor->predtyp),
             GetString(c->cc->functor->printname),
```

```
              c->cc->functor->arity);
      else
        printf("        Beyond Cost Centers\n");
    }
    printf("=============== ======= ======= ======= ======= ======= ");
    if (prof_include_time) printf("======================= ");
    printf("====    ====\n");
    printf("%7ld %6.2f%% %7ld %7ld %7ld %7ld %7ld ",
      total.calls, 100.0, total.skips, total.nskips, total.cuts, total.scuts, total.retrys);
    if (prof_include_time)
      printf("%12f (%6.2f%%)  ", (double)total.time_spent/PROFILE__GET_FREQ*1.0e3, 100.0);
    printf("        Total\n");
    printf("%ld cost centers called\n\n", realsize);
  }
  checkdealloc((TAGGED *)ccc_table, ht_count(cost_center_table)
    * sizeof(struct cost_center_clip));
}

static int compare_ccc_calls(arg1, arg2)
     const void *arg1, *arg2;
{
  struct cost_center_clip *ccc1, *ccc2;

  ccc1 = (struct cost_center_clip *)arg1;
  ccc2 = (struct cost_center_clip *)arg2;

  if ((ccc1)->total.calls > (ccc2)->total.calls)
    return (1);
  if ((ccc1)->total.calls < (ccc2)->total.calls)
    return (-1);
  return (0);
}

static int compare_ccc_clicks(arg1, arg2)
     const void *arg1, *arg2;
{
  struct cost_center_clip *ccc1, *ccc2;

  ccc1 = (struct cost_center_clip *)arg1;
  ccc2 = (struct cost_center_clip *)arg2;

  if ((ccc1)->total.time_spent > (ccc2)->total.time_spent)
    return (1);
  if ((ccc1)->total.time_spent < (ccc2)->total.time_spent)
    return (-1);
  return (0);
}

void profile_flat_dump(void)
{
  struct sw_on_key *table = (struct sw_on_key *)*predicates_location;
  struct sw_on_key_node *keyval;
  int j = SwitchSize(*predicates_location);
  long int i, realsize = 0, realsize_oh = 0;
  struct profile_currents total, overhead;
  struct definition **pred_table, **pred_table_oh, *d;
  int (*compare_func)(const void *, const void *);

  PROFILE__RESET_PROF(total);
  PROFILE__RESET_PROF(overhead);
  PROFILE__TIME_FLUSH;
  printf("\nPlease keep in mind the next definitions: \n");
```

```
  printf("Calls=  Number of times a predicate is called.\n");
  printf("Skips=  Number of skips over a node in case it be cutted.\n");
  printf("NSkips= Number of nodes that have been cutteds with the cut.\n");
  printf("Cuts=   Number of efective cuts that are done in the scope of a node.\n");
  printf("SCuts=  Number of cuts that don't have effects in the scope of a node.\n");
  printf("Retrys= Number of times a choice point is retried (It is not equal to redo).\n");
  printf("Total NSkips = Total SCuts.\n\n");
  printf("Flat profile information:\n\n");
  for (--j; j>=0; --j) {           /* Find how many preds. we have called */
    keyval = &table->tab.asnode[j];
    if ((d = keyval->value.def) &&
        d -> predtyp != ENTER_UNDEFINED  &&
        d -> prof.calls)
    {
      if(functor_have_overhead(cost_center_table, d)) {
        realsize_oh++;
        overhead.calls += d->prof.calls;
        overhead.skips += d->prof.skips;
        overhead.nskips += d->prof.nskips;
        overhead.cuts += d->prof.cuts;
        overhead.scuts += d->prof.scuts;
        overhead.retrys += d->prof.retrys;
        overhead.time_spent += d->prof.time_spent;
      }
      else {
        realsize++;
        total.calls += d->prof.calls;
        total.skips += d->prof.skips;
        total.nskips += d->prof.nskips;
        total.cuts += d->prof.cuts;
        total.scuts += d->prof.scuts;
        total.retrys += d->prof.retrys;
        /* if we need to verify the time conservation law: (available
           only when time profiling is on) */
        total.time_spent += d->prof.time_spent;
      }
    }
  }
  /* using the time conservation law, we can obtain the tot_clicks:
     (valid also when time profiling is off)*/
#if defined(PROFILE__PROFTIME)
  /*   tot_clicks = click_last_addition - click_start - click_profiling + 1; */
  if(click_profiling + total.time_spent + overhead.time_spent
      != click_last_addition - click_start)
    printf("Verification Error (good if equal) %lld = %lld\n",
           click_profiling + total.time_spent + overhead.time_spent,
           click_last_addition - click_start);
#else
  /*   tot_clicks = click_last_addition - click_start + 1; */
  if(total.time_spent + overhead.time_spent
      != click_last_addition - click_start)
    printf("Verification Error (good if equal) %lld = %lld\n",
           total.time_spent + overhead.time_spent,
           click_last_addition - click_start);
#endif

  pred_table =                          /* Make a table with room for them */
    (struct definition **)checkalloc(realsize*sizeof(struct definition *));
  pred_table_oh =
    (struct definition **)checkalloc(realsize_oh*sizeof(struct definition *));

  j = SwitchSize(*predicates_location);
```

```
realsize = 0;
realsize_oh = 0;
for (--j; j>=0; --j) {
  keyval = &table->tab.asnode[j];
  if ((d  = keyval->value.def) &&
      d -> predtyp != ENTER_UNDEFINED &&
      d -> prof.calls) {
    if(functor_have_overhead(cost_center_table, d)) {
      pred_table_oh[realsize_oh++] = d;
    }
    else
      pred_table[realsize++] = d;
  }
}

if(prof_include_time)
  compare_func = compare_clicks;
else
  compare_func = compare_calls;
qsort(pred_table, realsize, sizeof(struct definition *), compare_func);
qsort(pred_table_oh, realsize_oh, sizeof(struct definition *), compare_func);
printf("Calls           Skips   NSkips  Cuts    SCuts   Retrys  Time(ms) -rough         Type    Spec\n");
printf("=============== ======= ======= ======= ======= ======= ====================== ====    ====\n");
i = realsize;
while(i>0){
  d = pred_table[--i];
  printf("%7ld %6.2f%% %7ld %7ld %7ld %7ld %7ld %12f (%6.2f%%) %s  %s/%d\n",
         d->prof.calls,
         (((double)d->prof.calls)*100.0)/(double)total.calls,
         d->prof.skips,
         d->prof.nskips,
         d->prof.cuts,
         d->prof.scuts,
         d->prof.retrys,
         (((double)d->prof.time_spent)/PROFILE__GET_FREQ)*1.0e3,
         (((double)d->prof.time_spent)*100.0)/(double)total.time_spent,
         predicate_type(d->predtyp),
         GetString(d->printname),
         d->arity);
}
printf("=============== ======= ======= ======= ======= ======= ====================== ====    ====\n");
printf("%7ld %6.2f%% %7ld %7ld %7ld %7ld %7ld %12f (%6.2f%%)         Total\n",
  total.calls, 100.0, total.skips, total.nskips, total.cuts, total.scuts,
  total.retrys, (double)total.time_spent/PROFILE__GET_FREQ*1.0e3, 100.0);
printf("%ld predicates called\n\n", realsize);

i = realsize_oh;
if(i) {
  printf("Overhead:\n");
  printf("Calls           Skips   NSkips  Cuts    SCuts   Retrys  Time(ms) -rough         Type    Spec\n");
  printf("=============== ======= ======= ======= ======= ======= ====================== ====    ====\n");
  while(i>0){
    d = pred_table_oh[--i];
    printf("%7ld %6.2f%% %7ld %7ld %7ld %7ld %7ld %12f (%6.2f%%) %s  %s/%d\n",
           d->prof.calls,
           (((double)d->prof.calls)*100.0)/(double)overhead.calls,
           d->prof.skips,
           d->prof.nskips,
           d->prof.cuts,
           d->prof.scuts,
           d->prof.retrys,
           (((double)d->prof.time_spent)/PROFILE__GET_FREQ)*1.0e3,
```

```
            (((double)d->prof.time_spent)*100.0)/(double)overhead.time_spent,
            predicate_type(d->predtyp),
            GetString(d->printname),
            d->arity);
    }
    printf("=============== ======= ======= ======= ======= ======= ============            ====    ====\n");
    printf("%7ld %6.2f%% %7ld %7ld %7ld %7ld %7ld %12f (%6.2f%%)         Total\n",
      overhead.calls, 100.0, overhead.skips, overhead.nskips, overhead.cuts, overhead.scuts,
      overhead.retrys, (double)overhead.time_spent/PROFILE__GET_FREQ*1.0e3, 100.0);
    printf("%ld instrumentation predicates called\n", realsize_oh);
#if defined(PROFILE__PROFTIME)
    printf("%8f ms (%6.2f%%) doing profiling\n",
      ((double)click_profiling/PROFILE__GET_FREQ)*1.0e3,
      ((double)click_profiling)/(click_profiling + total.time_spent
      + overhead.time_spent)*100.0);
    printf("%8f ms (%6.2f%%) executing instrumentation code\n====================\n",
      ((double)overhead.time_spent/PROFILE__GET_FREQ)*1.0e3,
      ((double)overhead.time_spent)/(click_profiling+total.time_spent
      + overhead.time_spent)*100.0);
#endif
  }
  else
    printf("{NOTE: Don't have overhead caused by instrumentation predicates}\n");
#if defined(PROFILE__PROFTIME)
  printf("%8f ms (%6.2f%%) Total overhead\n\n",
         (((double)click_profiling+overhead.time_spent)/PROFILE__GET_FREQ)*1.0e3,
         ((double)click_profiling+overhead.time_spent)/(click_profiling
         + total.time_spent + overhead.time_spent)*100.0);
#endif
  checkdealloc((TAGGED *)pred_table_oh, realsize_oh*sizeof(struct definition *));
  checkdealloc((TAGGED *)pred_table, realsize*sizeof(struct definition *));
}

static int compare_calls(arg1, arg2)
     const void *arg1, *arg2;
{
  struct definition **pred1, **pred2;

  pred1 = (struct definition **)arg1;
  pred2 = (struct definition **)arg2;

  if ((*pred1)->prof.calls > (*pred2)->prof.calls)
    return (1);
  if ((*pred1)->prof.calls < (*pred2)->prof.calls)
    return (-1);
  return (0);
}

static int compare_clicks(arg1, arg2)
     const void *arg1, *arg2;
{
  struct definition **pred1, **pred2;

  pred1 = (struct definition **)arg1;
  pred2 = (struct definition **)arg2;

  if ((*pred1)->prof.time_spent > (*pred2)->prof.time_spent)
    return (1);
  if ((*pred1)->prof.time_spent < (*pred2)->prof.time_spent)
    return (-1);
  return (0);
}
```

```
char * predicate_type(int t)
{
  switch (t) {
  case ENTER_COMPACTCODE:
  case ENTER_COMPACTCODE_INDEXED:
  case ENTER_PROFILEDCODE:
  case ENTER_PROFILEDCODE_INDEXED: return "Emul  " ;
  case ENTER_FASTCODE:
  case ENTER_FASTCODE_INDEXED:     return "Fast  " ;
  case ENTER_UNDEFINED:            return "Undef " ;
  case ENTER_C:                    return "C     " ;
  case ENTER_INTERPRETED:          return "Interp" ;
  case BUILTIN_ABORT:
  case BUILTIN_APPLY:
  case BUILTIN_CALL:
  case BUILTIN_SYSCALL:
  case BUILTIN_NODEBUGCALL:
  case BUILTIN_TRUE:
  case BUILTIN_FAIL:
  case BUILTIN_CURRENT_INSTANCE:
  case BUILTIN_RESTORE:
  case BUILTIN_COMPILE_TERM:
  case BUILTIN_GELER:
  case BUILTIN_INSTANCE:
  case BUILTIN_DIF:               return "Built " ;
  default:                        return "Other " ;
  }
}

void profile_dump(void)
{
  profile_flat_dump();
  profile_detail_dump();
}

#endif
```

A.1.5 File `profiler.h`

```
#include "../hashtable/hashtab.h"

/* #define PROFILE__TRACER */
#define PROFILE__PROFTIME

extern BOOL prolog_profile_dump(Argdecl);

extern void profile_dump(void);

extern char * predicate_type(int t);
extern void show_func_point(char *label, int call_point, unsigned int choice,
  struct definition *functor);
extern void profile__init(void);

extern struct cost_center_item *get_cc_item(struct ht_tab *ht,
  struct definition *functor);

struct node_stack_item {
  struct cost_center_item *first_cci;
  struct cost_center_item *last_cci;
  unsigned int choice;
```

```
};

struct cost_center {
  struct definition *functor;
  unsigned long int calls;
  unsigned long int redos;
  unsigned long int exits;
  unsigned long int fails;
  ENG_LINT time_spent;
  ht_tab * cc_item_table;
};

struct cost_center_item {
  struct definition *functor;
  struct profile_currents prof;
};

extern struct cost_center *get_cost_center(struct ht_tab *cct, struct definition *functor);
extern struct cost_center *add_cost_center(struct ht_tab *cct, struct definition *functor);

extern struct ht_tab *cost_center_table;
extern struct cost_center *active_cc, *default_cc, *prev_cc;
extern struct cost_center_item *last_cci;
extern struct cost_center **cc_stack;
extern struct definition *last_called_predicate;

extern struct definition *profile__hook_cc_call;
extern struct definition *profile__hook_cc_redo;
extern struct definition *profile__hook_cc_redo_;
extern struct definition *profile__hook_cc_exit;
extern struct definition *profile__hook_cc_fail;
extern struct definition *profile__hook_cc_fail_;
extern int node_stack_point;
extern int node_stack_size;
extern struct node_stack_item *node_stack;
extern ENG_LINT click_last_addition;
extern ENG_LINT click_ini_profiling;
extern ENG_LINT click_start;
#if defined(PROFILE__PROFTIME)
extern ENG_LINT click_profiling;
extern ENG_LINT click_end_profiling;
#endif


/* #if defined(PROFILE__TRACER) */
int profile_trace;
#define ShowFuncPoint(label,call_point,choice,functor) \
  {if(profile_trace) show_func_point(label,call_point,choice,functor);}
#define ShowSkipNodes(skips) \
  {if(profile_trace) printf("cut skipped %d nodes \n", skips);}
#define ShowClauseNumber \
  {if(profile_trace) printf("\tclause %d \n", w->node->next_alt->number);}
#define ShowNoMoreAlts \
  {if(profile_trace) printf("No more alts\n");}

/* #else */
/*   #define ShowFuncPoint(label,call_point,choice,functor) */
/*   #define ShowSkipNodes(skips) */
/*   #define ShowClauseNumber */
/*   #define ShowNoMoreAlts */
/* #endif */
```

```
#define GET_DEFINITION(NAME, ARITY) \
  insert_definition(predicates_location,init_atom_check((NAME)),(ARITY),TRUE)
/*
example:

struct definition *address_mypred;
address_mypred = GET_DEFINITION("profilermodule:mypred", 3);

 */


#define PROFILE__TIME_INI \
if(prof_include_time) { \
  REGISTER ENG_LINT profclick0 = PROFILE__GET_CLICK(); \
  PROFILE__TIME_CARRY \
  click_ini_profiling = profclick0; \
}

#if defined(PROFILE__PROFTIME)

# define PROFILE__TIME_END \
if(prof_include_time) { \
  click_end_profiling = PROFILE__GET_CLICK(); \
}

# define PROFILE__TIME_CARRY \
{ \
  REGISTER ENG_LINT profdiff0 = click_end_profiling - click_ini_profiling; \
  click_profiling += profdiff0; \
  click_last_addition += profdiff0; \
}

# define PROFILE__TIME_FLUSH \
{ \
  PROFILE__TIME_CARRY \
  click_ini_profiling = click_end_profiling; \
}

#else

# define PROFILE__TIME_END
# define PROFILE__TIME_CARRY
# define PROFILE__TIME_FLUSH \
{ \
  PROFILE__TIME_CARRY \
  click_ini_profiling = 0; \
}

#endif

/* #define PROFILE__TIME_PROFILING(Code) \ */
/* { \ */
/*   if(profile) { \ */
/*     PROFILE__TIME_INI \ */
/*     Code; \ */
/*     PROFILE__TIME_END \ */
/*   } \ */
/* } */
```

A.2 hrtimer Module

A.2.1 File hrtimer.pl

```
:- module(hrtimer,[],[foreign_interface]).

:- use_module(library('hrtimer/hrtimea')).

:- comment(title,"Redefiner for statistics using hrtimer").

:- comment(author,"Edison Mera").

:- comment(module,"

@section{Implementing correct time benchmarking for profiling tools.}

The aim of this work is to show some techniques that improves the
results of benchmarks, and fit them to the task of profiling.

By now, when I need to proof some computer process, the time
measurement used is the millisecond.  However, the problem of that
approach is that most of the systems are doted with low resolution
calendar-clocks, some C functions such as time, or clock, returns the
time with a resolution of milliseconds, but the right is that the
resolution is about 0.01 seconds, and the number of milliseconds
returned have the last digit unaccurate.  A correct benchmarking for
profiling systems, must fit some requirements, such as:

@begin{enumerate}

@item Be independent of the plattform speed.  That is, the objective
   of the benchmark is to measure the time of execution of any piece
   of software independently of the plattform used to do the proof.

@item Do use of the high capacity and speed of modern computers, by
   this way, at most power computing, one could espect better results,
   and for the same benckmark, the improvement of the hardware
   platform must be reflected in a improvement of the performance of
   the profiling tool.

@end{enumerate}

For the first requirement, the measurement unit suggested is one CPU
clock cycle.  These measure is relatively independent to the hardware,
and for estimate the seconds used in execute some task, we must divide
the number of CPU cycles by the velocity in Hertzios of the computer.
For example, if some task takes 1.000.000.000 of cpu cycles, then the
taken time in a 2GHz modern computer is 0.5 seconds.

The second requirement, is solved fortunately with the same approach.
A improvement in the hardware will be reflected in a better
performance of the profiling tool.  In the example above, if we use a
4GHz computer, then the task will take 0.25 seconds.

But there are some issues: modern computers could execute more than 1
instruction by clock cycle, some laptops could slow down your cpu
clock to improve power managment, and there are a few of optimizations
such as the unsorted excecution, that are the benchmarks imprecise,
and for that, make completely hardware independent test is a challenge
task.

Now, the pentium II and highter processors have an instruction RDTSC,
```

```
that returns the number of cpu clock since the last power on of the
computer.  These information could be used to do the time measurement
in the benchmarks.

The next module, let us to redefine the traditional benchmark time
measurement with this new focus, and proof that this focus could
impact positively the development of any profiling tool for our
ciao/ciaopp system, and by extension, all systems that requires
profiling.

At this moment, this library has been implemented using a third part
software written in C, you can check these softare at:

@uref{http://www.cs.wisc.edu/paradyn/libhrtime/}

").

:- initialization(init_hrtimer(true)).

:- foreign_inline(
"#include \"hrtime.h\"
#include \"datadefs.h\"
#include \"timing_defs.h\"

ENG_LINT internal_userclick_hrtime(void)
{
  hrtime_t r;
  get_hrutime_self(&r);
  return r;
}

ENG_LINT internal_systemclick_hrtime(void)
{
  hrtime_t r;
  get_hrstime_self(&r);
  return r;
}

").

:- true pred init_hrtimer(go(Error)) :: int + (foreign, returns(Error)).
:- foreign_inline(init_hrtimer/1,
"long init_hrtimer(void) {
  long r;
  if(hrtime_is_present()) {
    r = hrtime_init();
    userclick = internal_userclick_hrtime;
    systemclick = internal_systemclick_hrtime;
    stats.userclockfreq = stats.systemclockfreq = stats.wallclockfreq;
    reset_statistics();
    return r;
  }
  else {
    printf(\"{WARNING: The kernel don't support hrtime, nothing has been done}\\n\");
    return 0;
  }
}
").
```

A.2.2 File `hrtimea.pl`

```
:- module(hrtimea,[
```

```
% main/0,
hrtime_is_present/1,
get_hrtimef/2,
get_hrvtimef/2,
get_hrutimef/2,
get_hrstimef/2,
get_current_hrtimef/1,

get_hrtime_selff/1,
get_hrvtime_selff/1,
get_hrutime_selff/1,
get_hrstime_selff/1,

hrtime_initl/1,
get_hrtime_structl/3,
free_hrtime_structl/2],[foreign_interface]).

% ============================================================================
:- comment(title, "High Resolution Time Adapter for Prolog.").
:- comment(author, "Edison Mera").
% ============================================================================
% 2004-02-25

:- comment(module,

"This module is an adatper for use in prolog the functions
availables in the library hrtime

This information has been written by Edison Mera, and complements the
official documentation found in the involved software.

@subsection{Installation Instructions for patch the Kernel with the
hrtime library}

To install this patch in the Kernel:

@begin{enumerate}

@item @bf{Uncompress the kernel}.  A recommended place to do that is
      /usr/src.

@item Copy the patch hrtime-0.6.1-2.4.25.patch in the directory
      /usr/src.

@item Now will We suppose that the kernel are in the directory
      /usr/src/linux-2.4.25.  enter into that directory and execute
      the following:

      patch -p1 < ../hrtime-0.6.1-2.4.25.patch

@item Your kernel must be patched now.

@item @bf{Configuring and compiling the Kernel}. More detailed
      information is available in the README file of the kernel, Is
      hightly recommended that you read it before begin.  But in
      addition to that, is hightly recommended that you use the
      original configuration file that comes with your operating
      system.  Normally (RedHat, Mandrake, etc...), It can be found in
      the directory /boot.  And also, you must enable the options
      related with the hrtime library.  In addition to that, to avoid
      problems with the crypto library, you can deactivate It in the
      kernel configuration menu.
```

```
      Resuming, in an ideal case, you must simply apply the next
      commands:

      a) Configuring Kernel:     'make xconfig'.

      b) Making dependencies:    'make dep'.

      c) Compiling the kernel:   'make bzImage'.

      d) Compiling the modules:  'make modules'.

      d) A work around:          'mkdir /lib/modules/2.4.25'.

      e) Installing the kernel:  'make install'.

      f) Installing the modules: 'make modules_install'.

@item Due to some undocumented problem in the kernel, It is possible
      that the file /boot/initrd-2.4.25.img have been created
      incorrectly.  You must recreate that executing:

      mv /boot/initrd-2.4.25.img /boot/initrd-2.4.25.img.old
      mkinitrd /boot/initrd-2.4.25.img 2.4.25

@item First Verify that the file /boot/grub/menu.lst have been
      generated correctly.  Be sure that if something leaves bad, you
      can always back to the previous kernel and solve the problem.

@item Restart the system and cross your fingers.

@item The patch is now ready to be used at Kernel Level.

@end{enumerate}

@subsection{Installation Instructions to install the library that let
us to acces the timming functions.}

Now is necessary to install the library that the access to the
timming functions.

@begin{enumerate}

@item Install the library rpm package:

      rpm -i libhrtime-0.6.1-1.i386.rpm

@item It is possible that the binaries files recently installed needs to
      be recompiled.  To do that, follow the next steps:

      a) Decompress the file libhrtime-0.6.1.tar.gz.

      b) In the subdir src, copy the file Makefile that are in this
         directory.

      c) do make all

      d) as root, do make install

@item If everything is Ok, The system must work now.

@end{enumerate}
```

```
").

:- use_foreign_library(c).
:- use_foreign_source([library('hrtimer/hrtime')]).

%:- use_foreign_library(hrtime).

:- foreign_inline("#include \"hrtime.h\"\n\n").

:- true pred hrtime_is_present(go(Value)) :: int +
   (foreign,returns(Value)).

% the f means that the result is double
:- true pred get_hrtimef(in(Hr), go(Dest)) :: address * num + (foreign).
:- foreign_inline(get_hrtimef/2,
"void get_hrtimef(struct hrtime_struct *hr, double *dest) {
  hrtime_t r;
  get_hrtime(hr, &r);
  *dest = (double)r;
}
").

:- true pred get_hrvtimef(in(Hr), go(Dest)) :: address * num +
   (foreign).

:- foreign_inline(get_hrvtimef/2,
"void get_hrvtimef(struct hrtime_struct *hr, double *dest) {
  hrtime_t r;
  get_hrvtime(hr, &r);
  *dest = (double)r;
}

").

:- true pred get_hrutimef(in(Hr), go(Dest)) :: address * num +
   (foreign).

:- foreign_inline(get_hrutimef/2,
"void get_hrutimef(struct hrtime_struct *hr, double *dest) {
  hrtime_t r;
  get_hrutime(hr, &r);
  *dest = (double)r;
}

").

:- true pred get_hrstimef(in(Hr), go(Dest)) :: address * num +
   (foreign).

:- foreign_inline(get_hrstimef/2,
"void get_hrstimef(struct hrtime_struct *hr, double *dest) {
  hrtime_t r;
  get_hrstime(hr, &r);
  *dest = (double)r;
}

").

:- true pred get_current_hrtimef(go(Dest)) :: num + (foreign).

:- foreign_inline(get_current_hrtimef/1,
```

```
"void get_current_hrtimef(double *dest) {
  hrtime_t r;
  get_current_hrtime(&r);
  *dest = (double)r;
}

").

:- true pred get_hrtime_selff(go(Dest)) :: num + (foreign).

:- foreign_inline(get_hrtime_selff/1,
"void get_hrtime_selff(double *dest) {
  hrtime_t r;
  get_hrtime_self(&r);
  *dest = (double)r;
}

").

:- true pred get_hrvtime_selff(go(Dest)) :: num + (foreign).

:- foreign_inline(get_hrvtime_selff/1,
"void get_hrvtime_selff(double *dest) {
  hrtime_t r;
  get_hrvtime_self(&r);
  *dest = (double)r;
}

").

:- true pred get_hrutime_selff(go(Dest)) :: num + (foreign).

:- foreign_inline(get_hrutime_selff/1,
"void get_hrutime_selff(double *dest) {
  hrtime_t r;
  get_hrutime_self(&r);
  *dest = (double)r;
}

").

:- true pred get_hrstime_selff(go(Dest)) :: num + (foreign).

:- foreign_inline(get_hrstime_selff/1,
"void get_hrstime_selff(double *dest) {
  hrtime_t r;
  get_hrstime_self(&r);
  *dest = (double)r;
}

").

% the l means that the result is long integer

:- true pred hrtime_initl(go(Error)) :: int +
(foreign,returns(Error)).

:- foreign_inline(hrtime_initl/0,
"long hrtime_initl() {
  return hrtime_init();
}
```

```
").

:- true pred get_hrtime_structl(in(Pid), go(Dest), go(Error)) :: int *
   address * int + (foreign,returns(Error)).

:- foreign_inline(get_hrtime_struct/3,
"long get_hrtime_structl(pid_t pid, struct hrtime_struct **dest) {
  return get_hrtime_struct(pid, dest);
}

").

:- true pred free_hrtime_structl(in(Hr), go(Error)) :: address * int +
   (foreign, returns(Error)).

:- foreign_inline(free_hrtime_structl/2,
"long free_hrtime_structl(struct hrtime_struct *hr) {
  return free_hrtime_struct(hr);
}

").

% main :-
%  hrtime_initl(_E1),
%  get_hrtime_structl(0,Ts,_E2),
%  display(Ts),nl,
%  get_hrstimef(Ts,A),
%  display(A),nl,
%  get_hrstimef(Ts,B),
%  display(B),nl,
%  free_hrtime_structl(Ts,_E3).
```

A.3 `hashtable` Module

A.3.1 File `hashtable.pl`

```
:- module(hashtable, _, [assertions, foreign_interface]).

:- use_foreign_source([
library('hashtable/recycle'),
library('hashtable/lookupa'),
library('hashtable/hashtab')]).

:- foreign_inline("#include \"hashtab.h\"\n\n").

:- true pred ht_create(in(LogSize), go(HTab)) :: int * address + (foreign, returns(HTab)).

:- true pred ht_destroy(in(HTab)) :: address + (foreign).

:- true pred ht_count(in(HTab), go(Count)) :: address * int + (foreign(ht_count_),
   returns(Count)).

:- foreign_inline(ht_count/2,
"long ht_count_(ht_tab *t)
{
  return ht_count(t);
}
").

:- true pred ht_key(in(HTab), go(Key)) :: address * address +
```

```
   (foreign(ht_key_), returns(Key)).

:- foreign_inline(ht_key/2,
"void * ht_key_(ht_tab *t)
{
  return ht_key(t);
}
").

:- true pred ht_keyl(in(HTab), go(Keyl)) :: address * address +
   (foreign(ht_keyl_), returns(Keyl)).

:- foreign_inline(hkeyl_/2,
"long ht_keyl_(ht_tab *t)
{
  return ht_keyl(t);
}
").

:- true pred ht_stuff(in(HTab), go(Stuff)) :: address * address +
   (foreign(ht_stuff_), returns(Stuff)).

:- foreign_inline(ht_stuff/2,
"void * ht_stuff_(ht_tab *t)
{
  return ht_stuff(t);
}
").

:- true pred ht_find(in(HTab), in(Key), in(Keyl), go(Find)) :: address
   * int * int * int + (foreign, returns(Find)).

:- true pred ht_add(in(HTab), in(Key), in(Keyl), in(Stuff), go(Add))
   :: address * atm * int * atm * int + (foreign, returns(Add)).

:- true pred ht_add2(in(HTab), in(Key), in(Stuff), go(Add)) :: address
   * atm * atm * int + (foreign, returns(Add)).

:- foreign_inline(ht_add2/4,
"long ht_add2(ht_tab *t, ub1 *key, void *stuff) {
  return ht_add(t, key, strlen(key), stuff);
}
").

:- true pred ht_del(in(HTab), go(Del)) :: address * int + (foreign,
   returns(Del)).

:- true pred ht_first(in(HTab), go(First)) :: address * int + (foreign,
   returns(First)).

:- true pred ht_next(in(HTab), go(Next)) :: address * int + (foreign(ht_next_),
   returns(Next)).

:- foreign_inline(ht_next/2,
"long ht_next_(ht_tab *t)
{
  return ht_next(t);
}
").

:- true pred ht_nbucket(in(HTab), go(NBucket)) :: address * int +
   (foreign, returns(NBucket)).
```

```
:- true pred ht_stat(in(Htab)) :: address + (foreign).

add_rec(_,[]).
add_rec(HashTable,[D|Ds]) :-
ht_add2(HashTable, D, '', _X),
add_rec(HashTable,Ds).

% main :-
%  Dictionary=[one,two,tree,four,five,six,two,four],
%  ht_create(8, HashTable),
%  add_rec(HashTable,Dictionary),
%  ht_destroy(HashTable).
```

B Example of running `qsort2.pl` with the profiler.

```
Ciao-Prolog 1.11 #257: mar ago 31 09:53:16 CEST 2004
?- use_module(library(hrtimer)).

yes
?- use_module(library('profiler/profiler_utils')).

yes
?- profile_reset.

yes
?- use_module(qsort2).

yes
?- profile(qsort2:qsort([18,46,83,65,2,32,53,28,85,99,47,28,82,6,11,55,29,39,81, \
   90,37,10,0,66,51,7,21,85,27,31,63,75,4,95,99,11,28,61,74,18, 92,40,53,59,8],X)).
yes
{NOTE: Goal has success}

Please keep in mind the next definitions:
Calls=  Number of times a predicate is called.
Skips=  Number of skips over a node in case it be cutted.
NSkips= Number of nodes that have been cutteds with the cut.
Cuts=   Number of efective cuts that are done in the scope of a node.
SCuts=  Number of cuts that don't have effects in the scope of a node.
Retrys= Number of times a choice point is retried (It is not equal to redo).
Total NSkips = Total SCuts.

Flat profile information:

Calls           Skips   NSkips  Cuts    SCuts   Retrys  Time(ms) -rough         Type    Spec
=============== ======= ======= ======= ======= ======= ====================== ====    ====
    247  68.61%      45       0       0       0       0     0.111193 ( 73.45%) Emul    qsort2:prof$split/4
     91  25.28%      46       0       0       0       0     0.026515 ( 17.51%) Emul    qsort2:prof$qsort_/3
      1   0.28%       0       0       0       0       0     0.001883 (  1.24%) Built   internals:$current_instance/5
      3   0.83%       0       0       1       0       0     0.001755 (  1.16%) Emul    basiccontrol:metacall/3
      1   0.28%       0       0       0       0       0     0.001311 (  0.87%) C       internals:$erase/1
      1   0.28%       0       0       0       0       0     0.001122 (  0.74%) C       internals:$current_clauses/2
      1   0.28%       0       0       1       0       0     0.000993 (  0.66%) Emul    internals:rt_module_exp/6
      3   0.83%       0       0       0       0       0     0.000984 (  0.65%) Built   hiord_rt:call/1
      1   0.28%       0       0       0       0       0     0.000803 (  0.53%) Emul    exceptions:$$$0/3
      1   0.28%       0       0       1       0       0     0.000718 (  0.47%) Emul    internals:$$$25/4
      1   0.28%       0       0       0       0       0     0.000659 (  0.44%) Emul    data_facts:$$$0/1
      1   0.28%       1       0       0       0       0     0.000564 (  0.37%) Emul    qsort2:prof$qsort/2
      1   0.28%       0       0       1       0       0     0.000487 (  0.32%) Emul    exceptions:$$$1/2
      2   0.56%       1       0       0       0       0     0.000486 (  0.32%) Emul    basiccontrol:metacall2/2
      1   0.28%       0       0       0       0       0     0.000461 (  0.30%) Emul    data_facts:retract_fact_nb/1
      1   0.28%       1       0       0       0       0     0.000456 (  0.30%) Emul    exceptions:retract_catching/3
      1   0.28%       0       0       0       0       0     0.000437 (  0.29%) C       internals:$unlock_predicate/1
      1   0.28%       0       0       0       0       0     0.000297 (  0.20%) C       basiccontrol:$metachoice/1
      1   0.28%       0       0       0       0       0     0.000272 (  0.18%) Emul    data_facts:this_is_true/1
=============== ======= ======= ======= ======= ======= ====================== ====    ====
    360 100.00%      94       0       4       0       0     0.151397 (100.00%)         Total
19 predicates called

Overhead:
Calls           Skips   NSkips  Cuts    SCuts   Retrys  Time(ms) -rough         Type    Spec
=============== ======= ======= ======= ======= ======= ====================== ====    ====
     47  20.00%       0       0       0       0       0     0.013683 ( 23.09%) Emul    profiler_rt:profile__hook_cc_fail/1
     47  20.00%       0       0       0       0       0     0.011585 ( 19.55%) Emul    profiler_rt:profile__hook_cc_redo/1
     47  20.00%       0       0       0       0       0     0.011139 ( 18.80%) C       profiler_rt:profile__hook_cc_call/1
     47  20.00%       1       0       0       0       0     0.011037 ( 18.63%) C       profiler_rt:profile__hook_cc_exit/2
     45  19.15%       0       0       0       0       0     0.010415 ( 17.58%) Emul    qsort2:split/4
      1   0.43%       0       0       0       0       0     0.000732 (  1.24%) Emul    qsort2:qsort/2
      1   0.43%       0       0       0       0       0     0.000656 (  1.11%) Emul    qsort2:qsort_/3
=============== ======= ======= ======= ======= ======= ============           ====    ====
    235 100.00%       1       0       0       0       0     0.059247 (100.00%)         Total
7 instrumentation predicates called
0.320942 ms ( 60.37%) doing profiling
0.059247 ms ( 11.15%) executing instrumentation code
====================
0.380190 ms ( 71.52%) Total overhead

Cost Center     qsort2:split/4
        ================================================================================
        Calls=45      Redos=0       Exits=45      Fails=0
        ================================================================================
        Calls           Skips   NSkips  Cuts    SCuts   Retrys  Time(ms) -rough         Type    Spec
        =============== ======= ======= ======= ======= ======= ====================== ====    ====
```

```
          247 100.00%      45       0       0       0       0     0.111193 (100.00%)  Emul    qsort2:prof$split/4
        ============== ======= ======= ======= ======= ======= ====================== ====    ====
          247 100.00%      45       0       0       0       0     0.111193 (100.00%)          Total
        1 predicates called

Overhead:
        Calls           Skips   NSkips  Cuts    SCuts   Retrys  Time(ms) -rough         Type    Spec
        ============== ======= ======= ======= ======= ======= ====================== ====    ====
             45  33.33%       0       0       0      86       0     0.012502 ( 38.73%)  Emul    profiler_rt:profile__hook_cc_fail/1
             45  33.33%       0       0       0       0       0     0.010641 ( 32.96%)  C       profiler_rt:profile__hook_cc_exit/2
             45  33.33%       0       0       0       0       0     0.009139 ( 28.31%)  C       profiler_rt:profile__hook_cc_call/1
        ============== ======= ======= ======= ======= ======= ====================== ====    ====
            135 100.00%       0       0       0      86       0     0.032282 (100.00%)          Total
        3 instrumentation predicates called
        22.50% executing instrumentation code

Cost Center     qsort2:qsort_/3
        ================================================================================================
        Calls=1       Redos=0       Exits=1       Fails=0
        ================================================================================================
        Calls           Skips   NSkips  Cuts    SCuts   Retrys  Time(ms) -rough         Type    Spec
        ============== ======= ======= ======= ======= ======= ====================== ====    ====
             91 100.00%      46       0       0       0       0     0.026515 (100.00%)  Emul    qsort2:prof$qsort_/3
        ============== ======= ======= ======= ======= ======= ====================== ====    ====
             91 100.00%      46       0       0       0       0     0.026515 (100.00%)          Total
        1 predicates called

Overhead:
        Calls           Skips   NSkips  Cuts    SCuts   Retrys  Time(ms) -rough         Type    Spec
        ============== ======= ======= ======= ======= ======= ====================== ====    ====
             45  48.39%       0       0       0       0       0     0.011129 ( 50.08%)  Emul    profiler_rt:profile__hook_cc_redo/1
             45  48.39%       0       0       0       0       0     0.010415 ( 46.87%)  Emul    qsort2:split/4
              1   1.08%       0       0       0       0       0     0.000264 (  1.19%)  Emul    profiler_rt:profile__hook_cc_fail/1
              1   1.08%       0       0       0       0       0     0.000211 (  0.95%)  C       profiler_rt:profile__hook_cc_call/1
              1   1.08%       0       0       0       0       0     0.000202 (  0.91%)  C       profiler_rt:profile__hook_cc_exit/2
        ============== ======= ======= ======= ======= ======= ====================== ====    ====
             93 100.00%       0       0       0       0       0     0.022221 (100.00%)          Total
        5 instrumentation predicates called
        45.59% executing instrumentation code

Beyond Cost Centers
        ================================================================================================
        Calls=0       Redos=0       Exits=0       Fails=0
        ================================================================================================
        Calls           Skips   NSkips  Cuts    SCuts   Retrys  Time(ms) -rough         Type    Spec
        ============== ======= ======= ======= ======= ======= ====================== ====    ====
              1   4.76%       0       0       0       0       0     0.001883 ( 14.35%)  Built   internals:$current_instance/5
              3  14.29%       0       0       1       0       0     0.001755 ( 13.37%)  Emul    basiccontrol:metacall/3
              1   4.76%       0       0       0       0       0     0.001311 (  9.99%)  C       internals:$erase/1
              1   4.76%       0       0       0       0       0     0.001122 (  8.55%)  C       internals:$current_clauses/2
              1   4.76%       0       0       1       0       0     0.000993 (  7.57%)  Emul    internals:rt_module_exp/6
              3  14.29%       0       0       0       0       0     0.000984 (  7.50%)  Built   hiord_rt:call/1
              1   4.76%       0       0       0       0       0     0.000803 (  6.12%)  Emul    exceptions:$$$0/3
              1   4.76%       0       0       1       0       0     0.000718 (  5.47%)  Emul    internals:$$$25/4
              1   4.76%       0       0       0       0       0     0.000659 (  5.02%)  Emul    data_facts:$$$0/1
              1   4.76%       0       0       1       0       0     0.000487 (  3.71%)  Emul    exceptions:$$$1/2
              2   9.52%       1       0       0       0       0     0.000486 (  3.70%)  Emul    basiccontrol:metacall2/2
              1   4.76%       0       0       0       0       0     0.000461 (  3.51%)  Emul    data_facts:retract_fact_nb/1
              1   4.76%       1       0       0       1       0     0.000456 (  3.48%)  Emul    exceptions:retract_catching/3
              1   4.76%       0       0       0       0       0     0.000437 (  3.33%)  C       internals:$unlock_predicate/1
              1   4.76%       0       0       0       0       0     0.000297 (  2.26%)  C       basiccontrol:$metachoice/1
              1   4.76%       0       0       0       0       0     0.000272 (  2.07%)  Emul    data_facts:this_is_true/1
        ============== ======= ======= ======= ======= ======= ====================== ====    ====
             21 100.00%       2       0       4       1       0     0.013124 (100.00%)          Total
        16 predicates called

Overhead:
        Calls           Skips   NSkips  Cuts    SCuts   Retrys  Time(ms) -rough         Type    Spec
        ============== ======= ======= ======= ======= ======= ====================== ====    ====
              1  50.00%       0       0       0       0       0     0.000732 ( 76.90%)  Emul    qsort2:qsort/2
              1  50.00%       0       0       0       0       0     0.000220 ( 23.10%)  Emul    profiler_rt:profile__hook_cc_redo/1
        ============== ======= ======= ======= ======= ======= ====================== ====    ====
              2 100.00%       0       0       0       0       0     0.000952 (100.00%)          Total
        2 instrumentation predicates called
        6.77% executing instrumentation code

Cost Center     qsort2:qsort/2
        ================================================================================================
        Calls=1       Redos=0       Exits=1       Fails=0
        ================================================================================================
        Calls           Skips   NSkips  Cuts    SCuts   Retrys  Time(ms) -rough         Type    Spec
        ============== ======= ======= ======= ======= ======= ====================== ====    ====
              1 100.00%       1       0       0       0       0     0.000564 (100.00%)  Emul    qsort2:prof$qsort/2
```

```
        ============== ======= ======= ======= ======= ======= ====================== ====    ====
              1 100.00%       1       0       0       0       0     0.000564 (100.00%)          Total
        1 predicates called

Overhead:
        Calls           Skips   NSkips  Cuts    SCuts   Retrys  Time(ms) -rough         Type    Spec
        ============== ======= ======= ======= ======= ======= ====================== ====    ====
              1  20.00%       0       0       0       0       0     0.001789 ( 47.17%)  C       profiler_rt:profile__hook_cc_call/1
              1  20.00%       0       0       0       0       0     0.000917 ( 24.18%)  Emul    profiler_rt:profile__hook_cc_fail/1
              1  20.00%       0       0       0       0       0     0.000656 ( 17.31%)  Emul    qsort2:qsort_/3
              1  20.00%       0       0       0       0       0     0.000236 (  6.23%)  Emul    profiler_rt:profile__hook_cc_redo/1
              1  20.00%       1       0       0       0       0     0.000194 (  5.11%)  C       profiler_rt:profile__hook_cc_exit/2
        ============== ======= ======= ======= ======= ======= ====================== ====    ====
              5 100.00%       1       0       0       0       0     0.003792 (100.00%)          Total
        5 instrumentation predicates called
        87.04% executing instrumentation code

Detailed profiling resume:
================================================================================
Calls=47      Redos=0       Exits=47      Fails=0
================================================================================
Calls           Skips   NSkips  Cuts    SCuts   Retrys  Time(ms) -rough         Type    Spec
============== ======= ======= ======= ======= ======= ====================== ====    ====
    247  68.61%      45       0       0       0       0     0.111193 ( 73.45%)  Emul    qsort2:split/4
     91  25.28%      46       0       0       0       0     0.026515 ( 17.51%)  Emul    qsort2:qsort_/3
     21   5.83%       2       0       4       1       0     0.013124 (  8.67%)          Beyond Cost Centers
      1   0.28%       1       0       0       0       0     0.000564 (  0.37%)  Emul    qsort2:qsort/2
============== ======= ======= ======= ======= ======= ====================== ====    ====
    360 100.00%      94       0       4       1       0     0.151397 (100.00%)          Total
4 cost centers called


X = [0,2,4,6,7,8,10,11,11,18,18,21,27,28,28,28,29,31,32,37,39,40,46,47,51,53,53,55,59,61,63,65,66,74,75,81,82,83,85,85,90,92,95,99,99] ?
yes
```

C Example of running `school.pl` with the profiler.

This is the same example that are in [Deb83].

C.1 Source code of the module `school`

C.1.1 File `school.pl`

```
:- module(school,_,[profiler]).

:- use_module(library(aggregates)).
:- use_module(library('profiler/profiler_utils')).

student(john, cs453).
student(john, cs520).
student(john, cs455).
student(john, ma561).
student(tom, cs342).
student(tom, cs453).
student(mary, cs455).
student(mary, cs520).
student(paul, cs520).
student(jane, cs453).
student(jane, ma561).
student(robert, cs342).
student(larry, ma561).
student(larry, cs342).
student(larry, cs455).

teacher(binkley, cs453).
teacher(binkley, cs342).
teacher(opus, cs455).
teacher(dallas, cs520).
teacher(dallas, ma561).

course(cs453, eco103, tue).
course(cs455, gs701, mon).
course(cs455, gs701, wed).
course(cs342, eco103, fri).
course(cs520, gs703, tue).
course(ma561, ma123, mon).

prog(L) :- findall( (S, T, R), p(S, T, R), L).

p(S, T, R) :-
student(S, C1),student(S, C2),
teacher(T, C1), teacher(T, C2),
course(C1, R, _D1), course(C2, R, _D2),
\+(C1 = C2).

%:- data pppppp/2.

%main :-
% measure((assertz_fact(pppppp(x,y)),current_fact(pppppp(X,Y)),retract_fact(pppppp(X,Y))),T),display(T),nl.

main :-
measure(prog(X),T),
```

```
display(X),nl,display('Time='),display(T),display(' ms'),nl.
```

C.2 Run of the module `school`

```
Ciao-Prolog 1.11 #257: mar ago 31 09:53:16 CEST 2004
?- use_module(library(hrtimer)).

yes
?- use_module(library('profiler/profiler_utils')).

yes
?- profile_reset.

yes
?- use_module(school).

yes
?- profile(school:prog(X)).
yes
{NOTE: Goal has success}

Please keep in mind the next definitions:
Calls=  Number of times a predicate is called.
Skips=  Number of skips over a node in case it be cutted.
NSkips= Number of nodes that have been cutteds with the cut.
Cuts=   Number of efective cuts that are done in the scope of a node.
SCuts=  Number of cuts that don't have effects in the scope of a node.
Retrys= Number of times a choice point is retried (It is not equal to redo).
Total NSkips = Total SCuts.

Flat profile information:

Calls           Skips   NSkips  Cuts    SCuts   Retrys  Time(ms) -rough         Type    Spec
============== ======= ======= ======= ======= ======= ====================== ====    ====
     78  33.48%       0       0       0       0      16     0.050747 ( 42.74%) Emul    school:prof$teacher/2
     26  11.16%       0       0       0       0       0     0.013887 ( 11.70%) Emul    school:$$$0/2
     41  17.60%       0       0       0       0      15     0.013081 ( 11.02%) Emul    school:prof$course/3
     16   6.87%       0       0       0       0       5     0.006277 (  5.29%) Emul    school:prof$student/2
      3   1.29%       0       0       0       0       0     0.004405 (  3.71%) Built   internals:$compile_term/2
      4   1.72%       0       0       0       0       0     0.003596 (  3.03%) Built   internals:$current_instance/5
      4   1.72%       0       0       0       0       0     0.003004 (  2.53%) C       internals:$erase/1
      7   3.00%       0       0       0       0       0     0.002525 (  2.13%) C       internals:$current_clauses/2
      3   1.29%       0       0       1       0       0     0.001883 (  1.59%) Emul    internals:$$$25/4
      3   1.29%       0       0       1       0       0     0.001865 (  1.57%) Emul    basiccontrol:metacall/3
      3   1.29%       0       0       0       0       0     0.001740 (  1.47%) C       internals:$inserta/2
      4   1.72%       0       0       0       0       0     0.001581 (  1.33%) C       internals:$unlock_predicate/1
      3   1.29%       0       0       1       0       0     0.001518 (  1.28%) Emul    internals:rt_module_exp/6
      3   1.29%       1       0       0       0       0     0.001491 (  1.26%) Emul    aggregates:list_solutions/3
      4   1.72%       0       0       0       0       0     0.001333 (  1.12%) Emul    data_facts:$$$0/1
      4   1.72%       0       0       0       0       0     0.001056 (  0.89%) Built   hiord_rt:call/1
      3   1.29%       0       0       0       0       0     0.000892 (  0.75%) Emul    data_facts:meta_asserta_fact/1
      1   0.43%       0       0       0       0       8     0.000788 (  0.66%) Emul    aggregates:save_solutions/2
      1   0.43%       1       0       0       0       0     0.000678 (  0.57%) Emul    aggregates:list_solutions/2
      2   0.86%       0       0       0       0       0     0.000675 (  0.57%) Built   basiccontrol:fail/0
      3   1.29%       0       0       0       0       0     0.000666 (  0.56%) Emul    data_facts:retract_fact/1
      1   0.43%       0       0       0       0       0     0.000612 (  0.52%) Emul    aggregates:findall/3
      1   0.43%       0       0       0       0       0     0.000609 (  0.51%) Emul    exceptions:$$$0/3
      4   1.72%       0       0       0       0       0     0.000569 (  0.48%) Emul    data_facts:this_is_true/1
      1   0.43%       0       0       0       0       0     0.000511 (  0.43%) Emul    school:prof$prog/1
      2   0.86%       1       0       0       0       0     0.000486 (  0.41%) Emul    basiccontrol:metacall2/2
      1   0.43%       0       0       0       0       0     0.000476 (  0.40%) Emul    school:prof$p/3
      1   0.43%       0       0       0       0       0     0.000456 (  0.38%) C       basiccontrol:$metachoice/1
      3   1.29%       0       0       0       0       0     0.000427 (  0.36%) Emul    data_facts:asserta_fact/1
      1   0.43%       0       0       1       0       0     0.000356 (  0.30%) Emul    exceptions:$$$1/2
      1   0.43%       1       0       0       0       0     0.000349 (  0.29%) Emul    exceptions:retract_catching/3
      1   0.43%       0       0       0       0       0     0.000199 (  0.17%) Emul    data_facts:retract_fact_nb/1
============== ======= ======= ======= ======= ======= ====================== ====    ====
    233 100.00%       4       0       4       0      44     0.118738 (100.00%)         Total
32 predicates called

Overhead:
Calls           Skips   NSkips  Cuts    SCuts   Retrys  Time(ms) -rough         Type    Spec
============== ======= ======= ======= ======= ======= ====================== ====    ====
    162  15.65%       0       0       0       0       0     0.131208 ( 37.82%) C       profiler_rt:profile__hook_cc_redo_/1
    136  13.14%       0       0       0       0       0     0.049617 ( 14.30%) C       profiler_rt:profile__hook_cc_fail_/1
    163  15.75%       0       0       0       0     161     0.037838 ( 10.91%) Emul    profiler_rt:profile__hook_cc_redo/1
    163  15.75%       1       0       0       0       0     0.035628 ( 10.27%) C       profiler_rt:profile__hook_cc_exit/2
    137  13.24%       0       2       0       2     179     0.032862 (  9.47%) Emul    profiler_rt:profile__hook_cc_fail/1
    137  13.24%       0       0       0       0       0     0.029489 (  8.50%) C       profiler_rt:profile__hook_cc_call/1
     78   7.54%       0       0       0       0       0     0.016131 (  4.65%) Emul    school:teacher/2
```

```
     41   3.96%       0       0       0       0       0     0.009444 (  2.72%) Emul    school:course/3
     16   1.55%       0       0       0       0       0     0.003643 (  1.05%) Emul    school:student/2
      1   0.10%       0       0       0       0       0     0.000667 (  0.19%) Emul    school:prog/1
      1   0.10%       0       0       0       0       0     0.000401 (  0.12%) Emul    school:p/3
============== ======= ======= ======= ======= ======= ============             ====    ====
   1035 100.00%       1       2       0       2     340     0.346927 (100.00%)         Total
11 instrumentation predicates called
0.667214 ms ( 58.90%) doing profiling
0.346927 ms ( 30.62%) executing instrumentation code
====================
1.014141 ms ( 89.52%) Total overhead

Cost Center     school:teacher/2
        ================================================================================================
        Calls=78      Redos=58      Exits=58      Fails=78
        ================================================================================================
        Calls           Skips   NSkips  Cuts    SCuts   Retrys  Time(ms) -rough         Type    Spec
        ============== ======= ======= ======= ======= ======= ====================== ====    ====
             78 100.00%       0       0       0       0      16     0.050747 (100.00%)  Emul    school:prof$teacher/2
        ============== ======= ======= ======= ======= ======= ====================== ====    ====
             78 100.00%       0       0       0       0      16     0.050747 (100.00%)          Total
        1 predicates called

Overhead:
        Calls           Skips   NSkips  Cuts    SCuts   Retrys  Time(ms) -rough         Type    Spec
        ============== ======= ======= ======= ======= ======= ====================== ====    ====
             58  16.57%       0       0       0       0       0     0.059069 ( 44.28%)  Emul    profiler_rt:profile__hook_cc_redo/1
             78  22.29%       0       0       0       0       0     0.028383 ( 21.28%)  C       profiler_rt:profile__hook_cc_fail_/1
             78  22.29%       0       0       0       0      31     0.018180 ( 13.63%)  Emul    profiler_rt:profile__hook_cc_fail/1
             78  22.29%       0       0       0       0       0     0.015522 ( 11.64%)  C       profiler_rt:profile__hook_cc_call/1
             58  16.57%       0       0       0       0       0     0.012230 (  9.17%)  C       profiler_rt:profile__hook_cc_exit/2
        ============== ======= ======= ======= ======= ======= ====================== ====    ====
            350 100.00%       0       0       0       0      31     0.133384 (100.00%)          Total
        5 instrumentation predicates called
        72.44% executing instrumentation code

Cost Center     school:prog/1
        ================================================================================================
        Calls=1       Redos=0       Exits=1       Fails=0
        ================================================================================================
        Calls           Skips   NSkips  Cuts    SCuts   Retrys  Time(ms) -rough         Type    Spec
        ============== ======= ======= ======= ======= ======= ====================== ====    ====
              3   6.00%       0       0       0       0       0     0.004405 ( 17.41%)  Built   internals:$compile_term/2
              3   6.00%       0       0       0       2       0     0.002979 ( 11.77%)  Built   internals:$current_instance/5
              3   6.00%       0       0       0       0       0     0.002486 (  9.83%)  C       internals:$erase/1
              6  12.00%       0       0       0       0       0     0.001805 (  7.13%)  C       internals:$current_clauses/2
              3   6.00%       0       0       0       0       0     0.001740 (  6.88%)  C       internals:$inserta/2
              3   6.00%       1       0       0       0       0     0.001491 (  5.89%)  Emul    aggregates:list_solutions/3
              2   4.00%       0       0       0       0       0     0.001390 (  5.49%)  Emul    internals:$$$25/4
              3   6.00%       0       0       0       0       0     0.001367 (  5.40%)  C       internals:$unlock_predicate/1
              3   6.00%       0       0       0       0       0     0.001029 (  4.07%)  Emul    data_facts:$$$0/1
              3   6.00%       0       0       0       0       0     0.000892 (  3.52%)  Emul    data_facts:meta_asserta_fact/1
              1   2.00%       0       0       0       2       8     0.000788 (  3.11%)  Emul    aggregates:save_solutions/2
              2   4.00%       0       0       0       0       0     0.000736 (  2.91%)  Emul    internals:rt_module_exp/6
              1   2.00%       1       0       0       0       0     0.000678 (  2.68%)  Emul    aggregates:list_solutions/2
              2   4.00%       0       0       0       0       0     0.000675 (  2.67%)  Built   basiccontrol:fail/0
              3   6.00%       0       0       0       0       0     0.000666 (  2.63%)  Emul    data_facts:retract_fact/1
              1   2.00%       0       0       0       0       0     0.000612 (  2.42%)  Emul    aggregates:findall/3
              1   2.00%       0       0       0       0       0     0.000511 (  2.02%)  Emul    school:prof$prog/1
              3   6.00%       0       0       0       0       0     0.000427 (  1.69%)  Emul    data_facts:asserta_fact/1
              3   6.00%       0       0       0       0       0     0.000425 (  1.68%)  Emul    data_facts:this_is_true/1
              1   2.00%       0       0       0       0       0     0.000199 (  0.79%)  Built   hiord_rt:call/1
        ============== ======= ======= ======= ======= ======= ====================== ====    ====
             50 100.00%       2       0       0       4       8     0.025303 (100.00%)          Total
        20 predicates called

Overhead:
        Calls           Skips   NSkips  Cuts    SCuts   Retrys  Time(ms) -rough         Type    Spec
        ============== ======= ======= ======= ======= ======= ====================== ====    ====
              1  16.67%       0       0       0       0       0     0.002215 ( 54.88%)  C       profiler_rt:profile__hook_cc_call/1
              1  16.67%       0       2       0       6      30     0.000631 ( 15.62%)  Emul    profiler_rt:profile__hook_cc_fail/1
              2  33.33%       0       0       0       0       2     0.000524 ( 12.98%)  Emul    profiler_rt:profile__hook_cc_redo/1
              1  16.67%       0       0       0       0       0     0.000401 (  9.94%)  Emul    school:p/3
              1  16.67%       1       0       0       0       0     0.000266 (  6.58%)  C       profiler_rt:profile__hook_cc_exit/2
        ============== ======= ======= ======= ======= ======= ====================== ====    ====
              6 100.00%       1       2       0       6      32     0.004036 (100.00%)          Total
        5 instrumentation predicates called
        13.76% executing instrumentation code

Cost Center     school:p/3
        ================================================================================================
        Calls=1       Redos=2       Exits=2       Fails=1
        ================================================================================================
```

```
        Calls           Skips   NSkips  Cuts    SCuts   Retrys  Time(ms) -rough         Type    Spec
        ============== ======= ======= ======= ======= ======= ====================== ====    ====
             26  96.30%       0       0       0       0       0     0.013887 ( 96.69%)  Emul    school:$$$0/2
              1   3.70%       0       0       0       0       0     0.000476 (  3.31%)  Emul    school:prof$p/3
        ============== ======= ======= ======= ======= ======= ====================== ====    ====
             27 100.00%       0       0       0       0       0     0.014362 (100.00%)          Total
        2 predicates called

Overhead:
        Calls           Skips   NSkips  Cuts    SCuts   Retrys  Time(ms) -rough         Type    Spec
        ============== ======= ======= ======= ======= ======= ====================== ====    ====
            162  53.64%       0       0       0      24     159     0.038256 ( 55.48%)  Emul    profiler_rt:profile__hook_cc_redo/1
             78  25.83%       0       0       0       0       0     0.016131 ( 23.39%)  Emul    school:teacher/2
             41  13.58%       0       0       0       0       0     0.009444 ( 13.70%)  Emul    school:course/3
             16   5.30%       0       0       0       0       0     0.003643 (  5.28%)  Emul    school:student/2
              1   0.33%       0       0       0       0       0     0.000454 (  0.66%)  C       profiler_rt:profile__hook_cc_fail_/1
              2   0.66%       0       0       0       0       0     0.000421 (  0.61%)  C       profiler_rt:profile__hook_cc_exit/2
              1   0.33%       0       0       0       0       8     0.000376 (  0.54%)  Emul    profiler_rt:profile__hook_cc_fail/1
              1   0.33%       0       0       0       0       0     0.000229 (  0.33%)  C       profiler_rt:profile__hook_cc_call/1
        ============== ======= ======= ======= ======= ======= ====================== ====    ====
            302 100.00%       0       0       0      24     167     0.068954 (100.00%)          Total
        8 instrumentation predicates called
        82.76% executing instrumentation code

Cost Center     school:course/3
        ================================================================================================
        Calls=41      Redos=48      Exits=48      Fails=41
        ================================================================================================
        Calls           Skips   NSkips  Cuts    SCuts   Retrys  Time(ms) -rough         Type    Spec
        ============== ======= ======= ======= ======= ======= ====================== ====    ====
             41 100.00%       0       0       0       0      15     0.013081 (100.00%)  Emul    school:prof$course/3
        ============== ======= ======= ======= ======= ======= ====================== ====    ====
             41 100.00%       0       0       0       0      15     0.013081 (100.00%)          Total
        1 predicates called

Overhead:
        Calls           Skips   NSkips  Cuts    SCuts   Retrys  Time(ms) -rough         Type    Spec
        ============== ======= ======= ======= ======= ======= ====================== ====    ====
             48  21.92%       0       0       0       0       0     0.031560 ( 42.01%)  Emul    profiler_rt:profile__hook_cc_redo/1
             41  18.72%       0       0       0       0       0     0.014917 ( 19.86%)  C       profiler_rt:profile__hook_cc_fail_/1
             48  21.92%       0       0       0       0       0     0.010405 ( 13.85%)  C       profiler_rt:profile__hook_cc_exit/2
             41  18.72%       0       0       0       0      59     0.009938 ( 13.23%)  Emul    profiler_rt:profile__hook_cc_fail/1
             41  18.72%       0       0       0       0       0     0.008301 ( 11.05%)  C       profiler_rt:profile__hook_cc_call/1
        ============== ======= ======= ======= ======= ======= ====================== ====    ====
            219 100.00%       0       0       0       0      59     0.075121 (100.00%)          Total
        5 instrumentation predicates called
        85.17% executing instrumentation code

Beyond Cost Centers
        ================================================================================================
        Calls=0       Redos=0       Exits=0       Fails=0
        ================================================================================================
        Calls           Skips   NSkips  Cuts    SCuts   Retrys  Time(ms) -rough         Type    Spec
        ============== ======= ======= ======= ======= ======= ====================== ====    ====
              3  14.29%       0       0       1       0       0     0.001865 ( 20.80%)  Emul    basiccontrol:metacall/3
              3  14.29%       0       0       0       0       0     0.000857 (  9.55%)  Built   hiord_rt:call/1
              1   4.76%       0       0       1       0       0     0.000782 (  8.72%)  Emul    internals:rt_module_exp/6
              1   4.76%       0       0       0       0       0     0.000720 (  8.03%)  C       internals:$current_clauses/2
              1   4.76%       0       0       0       0       0     0.000617 (  6.88%)  Built   internals:$current_instance/5
              1   4.76%       0       0       0       0       0     0.000609 (  6.79%)  Emul    exceptions:$$$0/3
              1   4.76%       0       0       0       0       0     0.000518 (  5.78%)  C       internals:$erase/1
              1   4.76%       0       0       1       0       0     0.000493 (  5.50%)  Emul    internals:$$$25/4
              2   9.52%       1       0       0       0       0     0.000486 (  5.42%)  Emul    basiccontrol:metacall2/2
              1   4.76%       0       0       0       0       0     0.000456 (  5.09%)  C       basiccontrol:$metachoice/1
              1   4.76%       0       0       1       0       0     0.000356 (  3.97%)  Emul    exceptions:$$$1/2
              1   4.76%       1       0       0       1       0     0.000349 (  3.89%)  Emul    exceptions:retract_catching/3
              1   4.76%       0       0       0       0       0     0.000304 (  3.39%)  Emul    data_facts:$$$0/1
              1   4.76%       0       0       0       0       0     0.000213 (  2.38%)  C       internals:$unlock_predicate/1
              1   4.76%       0       0       0       0       0     0.000199 (  2.22%)  Emul    data_facts:retract_fact_nb/1
              1   4.76%       0       0       0       0       0     0.000144 (  1.60%)  Emul    data_facts:this_is_true/1
        ============== ======= ======= ======= ======= ======= ====================== ====    ====
             21 100.00%       2       0       4       1       0     0.008968 (100.00%)          Total
        16 predicates called

Overhead:
        Calls           Skips   NSkips  Cuts    SCuts   Retrys  Time(ms) -rough         Type    Spec
        ============== ======= ======= ======= ======= ======= ====================== ====    ====
              1  50.00%       0       0       0       0       0     0.000667 ( 72.60%)  Emul    school:prog/1
              1  50.00%       0       0       0       0       0     0.000252 ( 27.40%)  Emul    profiler_rt:profile__hook_cc_redo/1
        ============== ======= ======= ======= ======= ======= ====================== ====    ====
              2 100.00%       0       0       0       0       0     0.000919 (100.00%)          Total
        2 instrumentation predicates called
        9.30% executing instrumentation code
```

```
Cost Center     school:student/2
        ================================================================================================
        Calls=16      Redos=54      Exits=54      Fails=16
        ================================================================================================
        Calls           Skips   NSkips  Cuts    SCuts   Retrys  Time(ms) -rough         Type    Spec
        =============== ======= ======= ======= ======= ======= ====================== ====    ====
             16 100.00%       0       0       0       0       5     0.006277 (100.00%)  Emul    school:prof$student/2
        =============== ======= ======= ======= ======= ======= ====================== ====    ====
             16 100.00%       0       0       0       0       5     0.006277 (100.00%)          Total
        1 predicates called

Overhead:
        Calls           Skips   NSkips  Cuts    SCuts   Retrys  Time(ms) -rough         Type    Spec
        =============== ======= ======= ======= ======= ======= ====================== ====    ====
             54  34.62%       0       0       0       0       0     0.039385 ( 61.05%)  Emul    profiler_rt:profile__hook_cc_redo/1
             54  34.62%       0       0       0       0       0     0.012306 ( 19.07%)  C       profiler_rt:profile__hook_cc_exit/2
             16  10.26%       0       0       0       0       0     0.005862 (  9.09%)  C       profiler_rt:profile__hook_cc_fail_/1
             16  10.26%       0       0       0       0      51     0.003737 (  5.79%)  Emul    profiler_rt:profile__hook_cc_fail/1
             16  10.26%       0       0       0       0       0     0.003222 (  4.99%)  C       profiler_rt:profile__hook_cc_call/1
        =============== ======= ======= ======= ======= ======= ====================== ====    ====
            156 100.00%       0       0       0       0      51     0.064513 (100.00%)          Total
        5 instrumentation predicates called
        91.13% executing instrumentation code

Detailed profiling resume:
================================================================================
Calls=137     Redos=162     Exits=163     Fails=136
================================================================================
Calls           Skips   NSkips  Cuts    SCuts   Retrys  Time(ms) -rough         Type    Spec
=============== ======= ======= ======= ======= ======= ====================== ====    ====
     78  33.48%       0       0       0       0      16     0.050747 ( 42.74%)  Emul    school:teacher/2
     50  21.46%       2       0       0       4       8     0.025303 ( 21.31%)  Emul    school:prog/1
     27  11.59%       0       0       0       0       0     0.014362 ( 12.10%)  Emul    school:p/3
     41  17.60%       0       0       0       0      15     0.013081 ( 11.02%)  Emul    school:course/3
     21   9.01%       2       0       4       1       0     0.008968 (  7.55%)          Beyond Cost Centers
     16   6.87%       0       0       0       0       5     0.006277 (  5.29%)  Emul    school:student/2
=============== ======= ======= ======= ======= ======= ====================== ====    ====
    233 100.00%       4       0       4       5      44     0.118738 (100.00%)          Total
6 cost centers called


X = [(tom,binkley,eco103),(tom,binkley,eco103)] ?
yes
```