Adding a positive/negative cache to the term comparison routine

Enrico Tassi

INRIA Saclay

14 Feb 2013 — Paris

Roadmap

The problem

The idea

Work in progress

Some results

Long story short

Computation in CIC (reduction):

- plays an important role in CIC
- optimized for > 10 years (call by need, *_compute)

Type checking compares terms up-to reduction (conversion):

big scale (e.g. reflection)

small scale (e.g. regular type checking)

As of today:

- the big case is mainly reduction, we are good!
- very few optimizations for the "small" one...

The current algorithm

Conversion test (`reduction.ml`):

```
let rec are_conv t1' t2' =
  let t1 = whd_nodelta t1' and t2 = whd_nodelta t2' in
  if are_App_with_same_head t1 t2 then
    try List.forall2 are_conv (args t1) (args t2)
    with Err -> try are_conv t1 (unfold_head t2)
    with NotUnfold -> are_conv (unfold_head t1) t2
  else ...
```

Call by need (as in `closure.ml`):

- employs mutable terms (`fconstr`/`fterm`)
- weak head normal forms are "cached" in place

So what?

What's wrong with this approach:

- when a term has variables its reducts are usually bigger
- hence sharing reductions means sharing expansions
- comparing expanded terms is *way* more costly

Not so artificial example:

- testing: `proj_op S1 0 10!` $\equiv$ `proj_op S2 10 9!`
  may mean: `0 + 10!` $\equiv$ `10 * 9!`
  as well as: `0 + 10!` $\equiv$ `10 + 9!`

Pitfalls

We do use HO constructs and *today* we have big terms even while proving:

- HO may trick conversion to normalize sub-terms first
- every comparison that follows works with bigger terms (imperative updates of the CBN machine)

Work around

In the Mathematical Components library:

- locking of some transparent definitions
- inline the definition of some constants (projection) to force eager head reduction
- ...

In the (old) Univalent Foundations library:

- patch `closure.ml` to implement call by name

Roadmap

The problem

The idea

Work in progress

Some results

Congruence closure

Used when terms have to be tested for equality

- using transitivity ($a = b \to c = b \to a = c$)
- using congruence ($a = b \to f\ a = f\ b$)

In the context of SMT solvers, ML type checking, etc.

With a bit of sugar it resembles conversion:

- $a \equiv c$ if $a \rhd b$ and $c \rhd b$
- $f\ a \equiv f\ b$ if $a \equiv b$ (otherwise we unfold f and continue)

Admits efficient implementation (based on union-find).

A negative/positive cache

Union-find requires:

- fast syntactic comparison of terms (usually an int)
- fast associative map over terms (array: `int -> int`)

Could be read as: fast hash tables on "terms", that requires:

- fast comparison
- fast hash calculation

Roadmap

The problem

The idea

Work in progress

Some results

My approach

Steps:

1. dump all conversion problems (OK)
2. hashconsing for term and red machine (ok)
3. reduction machine on new data structures (ok)
4. union-find cache (almost)
5. testing (little)
6. tune conversion strategy (todo)

Where: `github.com/gares/coq/` branch `speedup/tcomp`

Opportunistic hashconsing

Fast hash tables means:

- fast comparison (hashconsing)
- fast hashing (cache the hash value)

The idea

- non-leaf nodes have an extra `int` field
- `mkBla` sets it to 0 (invalid hash value)
- internalizing a term updates the hash value
- invariant: non zero hash field means already canonical

Advantage: easy/progressive integration

New conversion test

Conversion as in `conversion.ml`:

```
let rec are_conv t1' t2' =
 let t1 = intern (whd_nodelta t1') in UF.union t1 t1';
 let t2 = intern (whd_nodelta t2') in UF.union t2 t2';
 match UF.same t1 t2 with
 | `No -> raise Err
 | `Yes -> ()
 | `Maybe ->
     if are_App_with_same_head t1 t2 then
       try List.forall2 are_conv (args t1) (args t2);
           UF.union t1 t2
       with Err -> ...
     else
       ... UF.partition t1 t2; raise Err ...
```

Roadmap

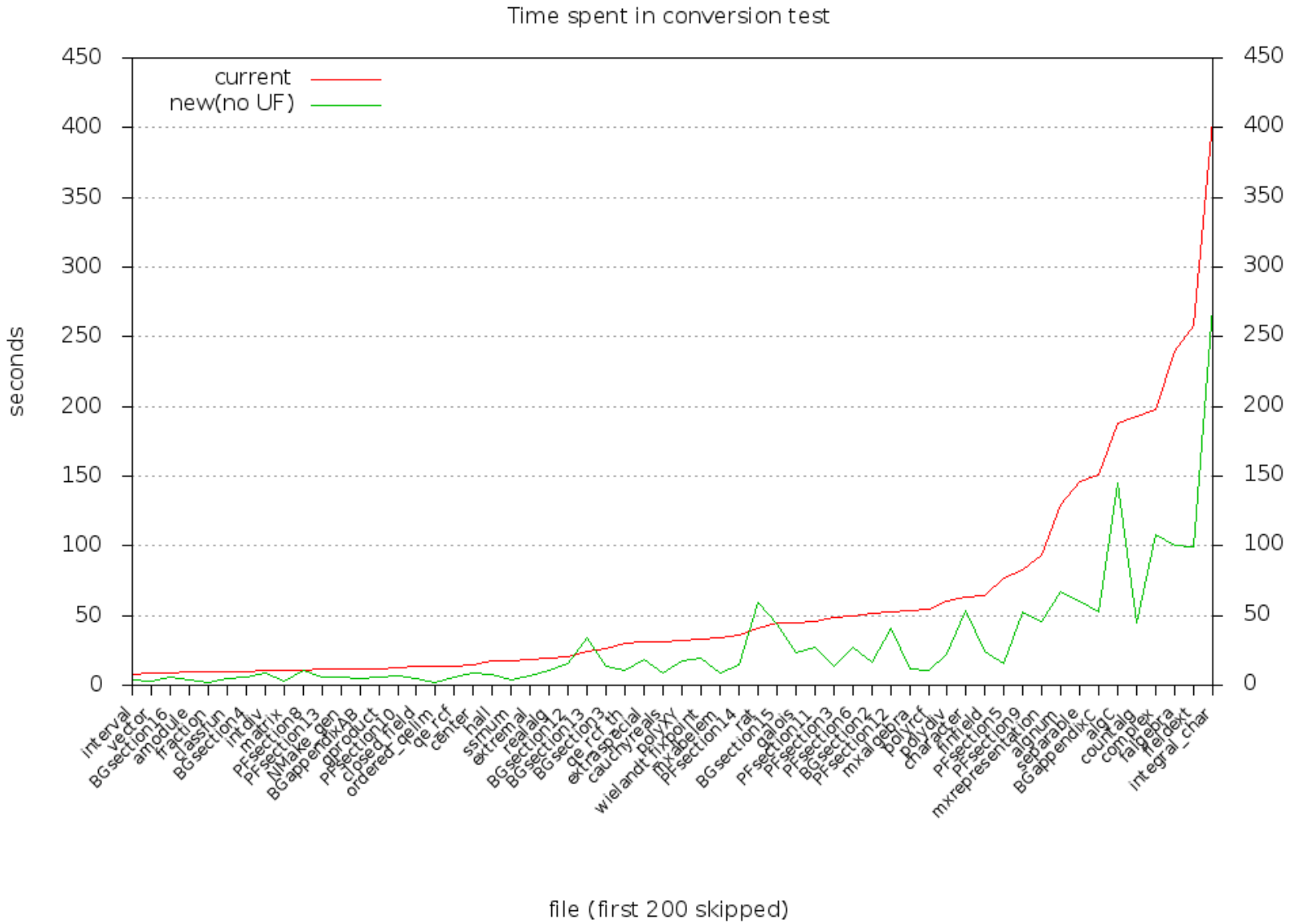
The problem

The idea

Work in progress

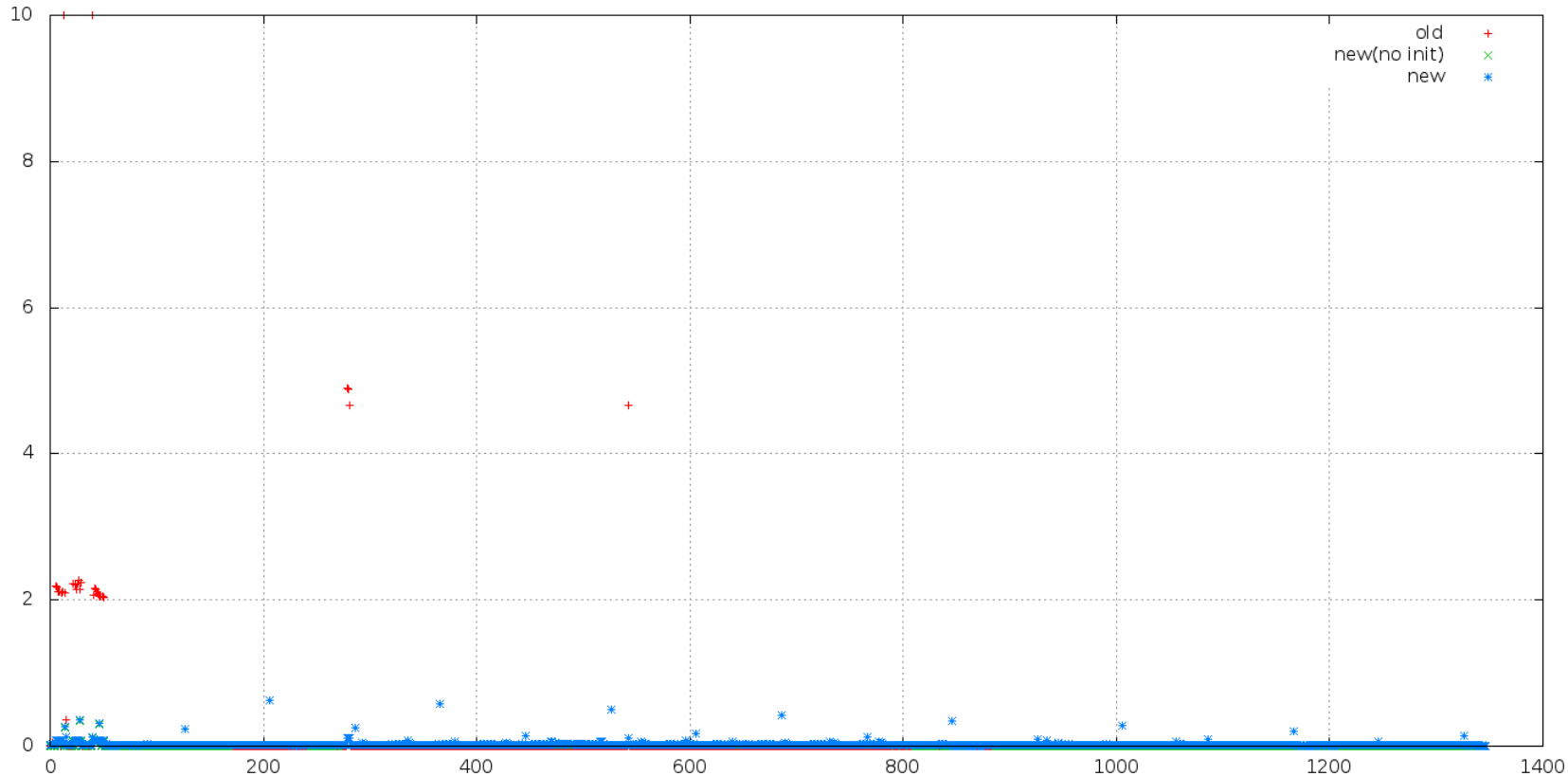
Some results

The conversion machine w.o. the cache

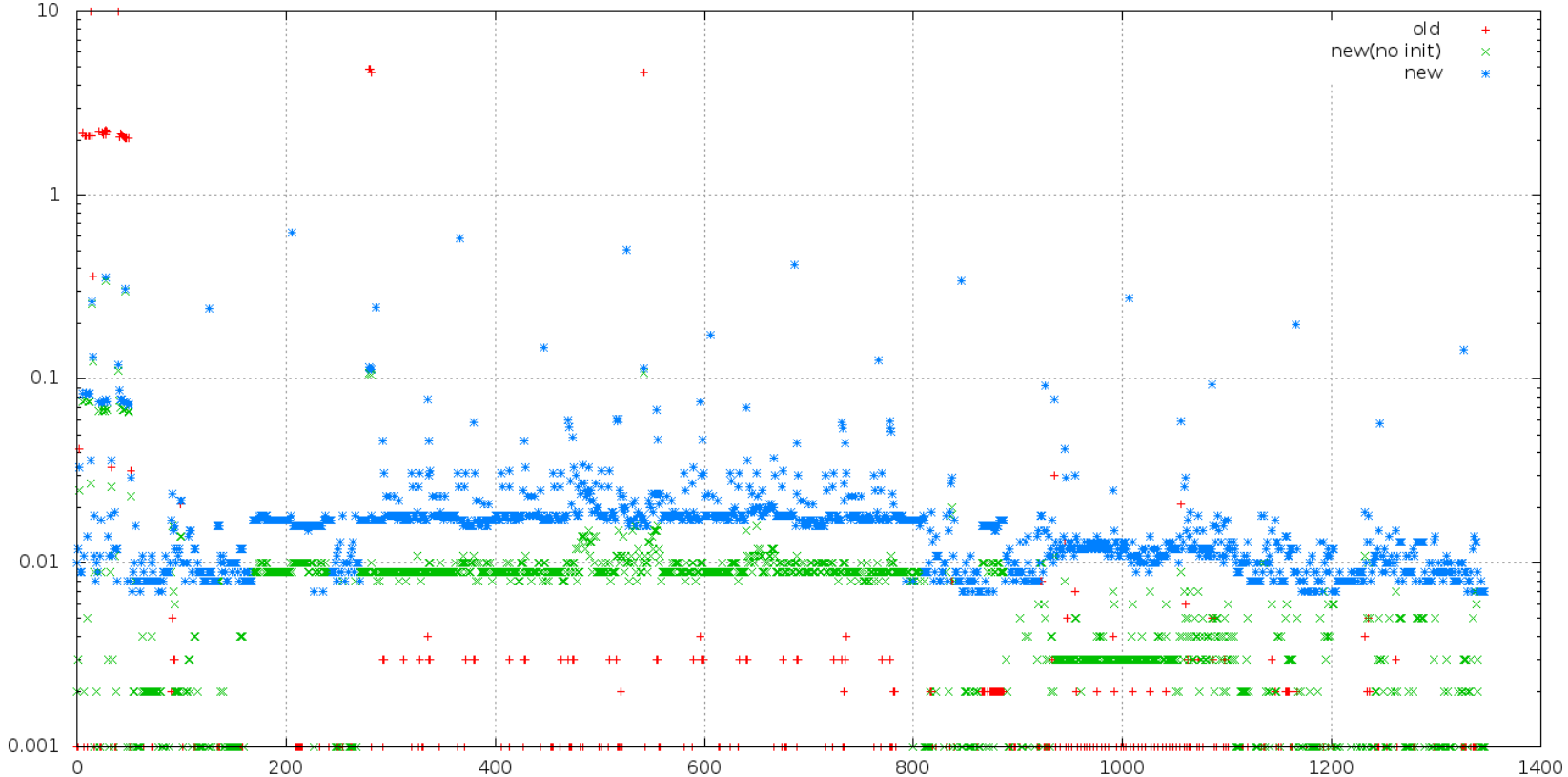


Is there a bench for pure reduction?

The conversion machine on uu0



The conversion machine on uu0 (log scale)



Open questions

- Negative cache (now very naive)
- Better hash tables (faster allocation or faster resizing)
- Better use of opportunistic sharing outside the kernel (unification piggy backs on the kernel)