A hash algorithm for N3 graphs in CWM

Work in progress

Jesús Arias Fisteus

jfisteus@csail.mit.edu, jaf@it.uc3m.es

Universidad Carlos III de Madrid

Visiting scientist at the Decentralized Information Group at CSAIL–MIT

–

This presentation: *http://www.it.uc3m.es/jaf/mit/20060914/presentation.pdf*

Implementation: *http://www.it.uc3m.es/jaf/mit/20060914/hash-n3.tar.gz*

Edited with emacs + LaTeX + prosper

Goal

- Design a hash algorithm for N3 graphs such that:
  - Equivalent graphs have the same hash value.
  - Non equivalent graphs have (with high probability) different hash value
- For this work graphs are considered equivalent if:
  - Have the same statements, with the same or different order.
  - Have the same variables / blank nodes, with the same or different names.

Edited with emacs + LaTeX + prosper

Operators

- XOR ($\otimes$)
  - Commutative and associative
  - Problem: $a \otimes a = 0$
- Product (modulus N)
  - Commutative and associative
  - If N prime, $\nexists a, b \neq 0 \ / \ ab = 0$.
  - $N = 2^{32} - 5$ is the largest 32-bit prime.
- Product and XOR combined:
  - $(ab) \otimes c \neq (a \otimes c)(b \otimes c)$
  - $(a \otimes b)c \neq (ac) \otimes (bc)$

Edited with emacs + LaTeX+ prosper

Why two different operators

- Associativity and commutativity are not good sometimes:
  - Example: $\{f_1\} \implies \{f_2\}$
    - $hash(f_1) = a \otimes b$
    - $hash(f_2) = d \otimes e$
    - $hash(\implies) = c$
    - $hash(\{f_1\} \implies \{f_2\}) = (a \otimes b) \otimes c \otimes (d \otimes e)$

$$(a \otimes b) \otimes c \otimes (d \otimes e) = (a \otimes e) \otimes c \otimes (d \otimes b)$$

$$(ab) \otimes c \otimes (de) \neq (ae) \otimes c \otimes (db)$$

Edited with emacs + LaTeX+ prosper

Overview of the algorithm

- Recursive (when entering subformulae).
- Combines partial hashes of: formulae, statements (triples), variables, lists, labelled nodes, literals.
- Every statement / formula affects the hash value of the variables that appear in it and viceversa.

Edited with emacs + LaTeX + prosper

Hashing a formula

1. Hash every statement in the formula ($h_{s_1}, h_{s_2}, ..., h_{s_n}$).
2. Take the hash of every varible declared in the formula ($h_{v_1}, h_{v_2}, ..., h_{v_m}$).
3. Combine them: $h = h_{s_1} h_{s_2} ... h_{s_n} h_{v_1} h_{v_2} ... h_{v_m}$.

Edited with emacs + LaTeX + prosper

Hashing a statement (triple)

1. The constants k_s, k_p, k_o are pre–defined.
2. Hash the terms in its subject, predicate and object (h_s, h_p, h_o).
3. Combine them: $h = (h_s k_s) \otimes (h_p k_p) \otimes (h_o k_o)$.

Edited with emacs + LaTeX + prosper

Hashing a term

- Labelled nodes: hash their URI (python's *hash* function).
- Literals: hash them as strings (python's *hash* function).
- Formulae: recursive.
- List: hash its member terms (recursion again).
  - $h = (h_1 \otimes 1)(h_2 \otimes 2)...(h_n \otimes n)$
- Anonymous variables: take their hash in the previuous round (initially a constant, see later).

Edited with emacs + LaTeX+ prosper

Hashing anonymous variables

- For each variable:
  1. Initialize its hash with a constant: universal ($h = k_{v_u}$) or existential ($h = k_{v_e}$).
  2. Recalculate a new hash h' from its previous hash h when it appears in position p (subject, predicate or object) of a statement (hash h_t): $h' = h \otimes (h_t k_p)$.
  3. When the processing of a formula (hash h_f) finishes, if the variable has been used in it or any inner formula and is declared also for the next upper formula, mix their hashes in the upper level: $h'' = h'(h \otimes h_f)$.

Edited with emacs + LaTeX+ prosper

Example on hashing

```
{?x test:partOf ?y. ?z test:includes ?y}
=> {?x test:partOf ?z}
```

`?x test:partOf ?y`	h_1	$(k_{v_u}k_s) \otimes (h_{partof}k_p) \otimes (k_{v_u}k_o)$
`?z test:includes ?y`	h_2	$(k_{v_u}k_s) \otimes (h_{includes}k_p) \otimes (k_{v_u}k_o)$
`?x test:partOf ?z`	h_3	$(k_{v_u}k_s) \otimes (h_{partof}k_p) \otimes (k_{v_u}k_o)$
`{?x test:partOf ?y...}`	h_{f_1}	$h_1 h_2$
`{?x test:partOf ?z}`	h_{f_2}	h_3
`?x`	h_x	$k_{v_u}((h_1k_s) \otimes h_{f_1})((h_3k_s) \otimes h_{f_2})$
`?y`	h_y	$k_{v_u}((h_1k_o) \otimes (h_2k_o) \otimes h_{f_1})$
`?z`	h_z	$k_{v_u}((h_2k_s) \otimes h_{f_1})((h_3k_o) \otimes h_{f_2})$

Example on hashing (cntd.)

```
{?x test:partOf ?y. ?z test:includes ?y}
=> {?x test:partOf ?z}
```

$$h = ((h_{f_1} k_s) \otimes (h_{implies} k_p) \otimes (h_{f_2} k_o)) h_x h_y h_z$$

Edited with emacs + LaTeX + prosper

Conclusions on hashing

- Efficient algorithm.
- Seems to work well for comparing / indexing N3 formulae:
  - Independent of the ordering of statements.
  - Independent of the name of variables.
  - Low probability of collision at formula level.

Edited with emacs + LATEX+ prosper

Canonicalization

- The canonicalization system has to decide:
  - A canonical ordering for statements in the same formula.
  - A canonical ordering for variables in the same formula.
  - A canonical name for variables.
- Solution using the hash algorithm:
  - The hash of statements defines their ordering.
  - The hash of variables defines their ordering.
  - The ordering of variables defines their name.

Edited with emacs + LaTeX+ prosper

Drawbacks

- The canonical order is based on the hash value of statements / variables:
  - If two statements in the same formula have the same hash, two different orderings are possible.
  - If two variables have the same hash, two different naming relations are possible.
- Conclusion: collisions at statement / variable level can provoke failures in canonicalization.

Edited with emacs + LaTeX + prosper

Solution

- Run the hash algorithm three times:
  - Initially the hash of variables is constant in the first step.
  - In every step:
    - The hash of statements is computed from the hash of variables in the previous level.
    - The hash of variables is computed from the hash of statements in the same level.

$$V_0 \longrightarrow \overbrace{S_1 \longrightarrow V_1}^{step_1} \longrightarrow \overbrace{S_2 \longrightarrow V_2}^{step_2} \longrightarrow \overbrace{S_3 \longrightarrow V_3}^{step_3}$$

Edited with emacs + LaTeX+ prosper

Other problems and fixes

- Variables defined locally in two or more formulae that are exactly equal will collide.
  - Solution: combine the hash of every variable with the hash of every parent formula of the formula in which the variable is declared.
  - $h'_v = h_v \otimes (h_{f_1} h_{f_2} ... h_{f_n})$
- Variables declared but not used have a fixed hash value and therefore all of them collide.
  - Solution: remove such variables from the canonicalized formula.

Edited with emacs + LaTeX + prosper

Implementation

- Features:
  - Loads documents using the CWM parser.
  - Calculates the hash value of the loaded formula.
  - Canonicalizes the loaded formula.
  - Writes the canonicalized formula.

Edited with emacs + LATEX + prosper

Implementation (cntd.)

- Limitations:
  - The output is written only for testing purposes, doesn't use CWM code for pretty–printing.
  - Problems found in the parser:
    - Recognises as *Fragment* variables defined with `@forSome`.
    - Recognises as *Fragment* variables defined with `this log:forAll`.
    - Sometimes fails recognising variables when they have the same name but are declared inside different overlapping formulae.

Edited with emacs + LaTeX+ prosper

Test and results

- Tested with all the N3 files under `2000/10/swap`:
  - Total files: 889.
  - Files with parse errors: about 20 / 30?
  - Files with canonicalization collisions: 19.
- Conclusion:
  - It works with a reasonable percentage of files.
  - But more work investigating the causes of existing collisions might improve the algorithm.

Edited with emacs + LaTeX + prosper

Time for discussion. . .

Edited with emacs + LaTeX + prosper