Content-based encoding of mathematical and code libraries

Josef Urban
Institute for Computing and Information Sciences
Radboud University, Nijmegen

Abstract

This is a proposal for content-based canonical naming of mathematical objects aimed at semantic machine processing, and an initial investigation of how useful it can be, how similar it is to other approaches, what disadvantages and limitations it has, and how it could be extended.

1 Motivation

There are interesting naming problems when dealing with a large formally encoded mathematical library that is changing, in particular by parallel (nonlinear) editing. Some of the problems might appear also in large code libraries, either via correspondences like Curry-Howard and Prolog-like interpretations of mathematics, or just by the fact that code can have (implicit, explicit, chosen) mathematical semantics attached to it, like formal mathematics, and the structure of code libraries is similar to formal mathematical ones. Some of the problems (for example renaming of concepts) might appear also in not completely formally specified mathematical repositories/wikis, or this might also be relevant to semi-formal wikis where at least some part of articles has some semantic encoding. The problems are following:

Renaming: Mathematical objects might change their name, or have parallel names. Bolzano-Weierstrass theorem is often known just as Weierstrass theorem, Jaśkowski natural deduction [Jas34] as Fitch-style [Pel99], Jarník's algorithm [Jar30] as Prim's algorithm [Pri57]. Similarly for Solomonoff vs. Kolmogorov vs. Chaitin complexity vs. algorithmic entropy. Composition of two relations might be called `compose(R,S)`, or `rel_compose(R,S)`, or just `R*S`. Still, these names refer to the same mathematical content: The theorems have the same wording, and the functors have the same definition.

Moving: An item might be (as a result of wiki-like refactoring) moved to a different place in the library, which in many naming schemes can result in some kind of renaming. For example, a fully qualified name in the formal CoRN library [CFGW04] based on the Coq system could change from CoRN.algebra.Basics.iterateN to CoRN.utilities.iterateN . In the formal Mizar Mathematical library [1], moving the fiftieth theorem in article `CARD_1` [Ban90b] to the first position in article `CARD_2` [Ban90a] would change its fully qualified name from `CARD_1:50` to `CARD_2:1`. Similarly for fully qualified names of functions in many programming languages. Any old term/formula/proof referring to the old name would later have to refer to the new name, making them look different.

Merging: Sometimes two or more objects might be merged into a single one. For example it might be recognized that Prim's description is essentially the same as Jarník's, and the two rewritten into a single unified form. Function composition might be recognized to be a special case of relation composition, and the two definitions merged into one.

[1] `http://mizar.org/`

Note that whole hierarchies of such parallel notations and objects might be developed, possibly even within one sufficiently large library that has no tools for detecting such terminological parallelism. For example, it is quite conceivable that a large amount of parallelism exists when developing concepts related to functions and relations, like domain, range, inverse, transitive closure, etc. Similarly for multiplicative vs. additive groups/monoids, arithmetical operations/theorems on complex vs. real vs. rational vs. integer vs. natural numbers (when viewed as restrictions), leading to parallely derived "different" versions of operations like power and modulo and parallel theorems about them using recursively different terminology.

How do we detect such parallelism and deal with it? Are we sentenced to re-invent pyramids of "more convenient" or "more fitting" or "more general" or "more politically correct" names of concepts over and over? And how do we find in the 1930's vocabulary used in the developments by Jaśkowski and Jarník that they are isomorphic to other developments from 1950's? Is there a matching/unification/subsumption/resolution/automated-reasoning algorithm that would (recursively) unfold the different concept names to some basic common language layer (set or type theory for example) without a significant performance penalty, allowing us to have easy search and inference tools for such heterogeneous libraries?

2 Content-based naming

It seems that the best the programming languages and formal libraries came up so far are module-based names, possibly enhanced by mangling the parameters and their types. The Mizar solution just follows the frequent mathematical practice of numbering definitions and theorems. Thus, the object naming typically depends either on human imagination, on the placement of the object at a particular place, or on both. None of these two guarantee stability with respect to the problems mentioned above.

The author is aware of three (related) solutions, each developed in a different context, listed here in the (assumed) historical order:

- Gödel numbering [G̈31]
- Recursive term sharing
- Recursive cryptographic hashing

2.1 Gödel numbering

The famous invention showing that arithmetic is self-referential. Every mathematical object is given a unique (impractically large) natural number as a name, based on an arithmetic function applied to its constituents (contents) already expressed as numbers. This obviously makes the name independent of placement in any particular module, and removes the human naming factor. As long as the wording (contents) is fixed, the name stays the same.

2.2 Recursive term sharing

In some computer implementations that deal with mathematical structures like terms (automated theorem provers (ATPs), Prolog) exhaustive sharing of terms is used to achieve space/time efficiency. In the E ATP system [Sch02], the terms `f(g(a)), g(g(a))` can be printed using the following numeric representation (corresponding to how the system represents them as pointers): `a -> *0, g(*0) -> *1, f(*1) -> *2, g(*1) -> *3` . In this way, the number assigned to a term is unique[2], however,

[2]Obviously, something needs to be said about variables and their treatment, e.g as de Bruijn indices for this purpose.

it depends on the order in which a particular set of terms was presented to the system. If `g(g(a))` was presented before `f(g(a))`, their numbers would be swapped. Because the numbering is contiguous, two such different numberings will be incompatible. So this naming is content-based like the previous one, however only inside one invocation of such a system.

2.3 Recursive cryptographic hashing

While the first scheme leads to impractically large numbers, the second is too fragile for practical purposes. Is there something providing the best of both worlds? An obvious direction in which this leads is minimal perfect hashing functions. However, constructing minimal (or small) perfect hashing function seems to be feasible only for finite sets of objects, possibly extended to uncomplicated denumerable sets of objects. There is no known (to the author) practical way how to have a reasonably well-behaving perfect hashing function for arbitrarily large complicated objects like mathematical formulas, terms, and proofs.

However, there is a practical approximation: cryptographic hash functions. Finding collisions for a cryptographic hash function like SHA-1[3] or SHA256[4] is so far very difficult, and "practically not happening". This has been used for a number of purposes, one of the most recent ones bearing surprising similarity to the Gödel's recursive encoding idea: SHA-1 based naming of files and directories in the Git version control system.[5] To summarize, each file is by Git named as the SHA-1 hash of its contents, and each directory is named by computing the SHA-1 of the file containing the names, permissions, and SHA-1 hashes of its constituent files and subdirectories. This in practice means that Git only stores each duplicated file/directory once, and that some (both file and directory) renamings are very simple to recognize.

3 Content-based naming of formal mathematics

A particular modification of the Git's SHA1-based internal naming scheme for Mizar (and similar formal/code libraries) could be as follows.

1. The initial library items (the set-theoretic equality and membership predicates in case of Mizar) are assigned an SHA1 value (e.g. their SHA1 value as strings, etc.) which serves as their unique name, that does not change between the library versions (all Mizar libraries are build up from the language of set theory).

2. A suitable semantic form is defined for terms, formulas, defined symbols, proofs, and other relevant library items. These all are trees (or DAGs), where the nodes are other items, and possibly some keywords of the language (if we follow Gödel's numbering scheme, we might assign SHA1 values also to the keywords, punctuation, etc.).

3. The semantic form (tree, DAG of items - SHA1 values) is suitably combined to obtain the canonical value for the semantic form. In case of Git, this is for example done by sequential listing of files with their SHA1 value in a file, and computing the SHA1 value of the contents of such file. In our case, the most likely candidate for easy semantic representation is the existing XML format of all Mizar items (similarly for Coq) like definitions and theorems. We could either just compute a SHA1 value of each such item when treated as a file (with other items replaced with their SHA1

[3] `http://tools.ietf.org/html/rfc3174`
[4] `http://csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf`
[5] `http://book.git-scm.com/1_the_git_object_model.html`

value), i.e. treating items as "files" in the Git internal naming algorithm, or we could do (probably more expensive) recursive computation of the value for all subtrees (similar to the shared term representation mentioned above). If such an operation turns out to be expensive, we could still use the fast local serial shared-term numbering to efficiently cache the previously computed SHA1 results for each instance of the library.

4 Proposed use, limitations, and extensions

A straightforward proposal is to compute such content-based names for all items in a large formal library like the MML or CoRN, and see how much naming-based duplication is inside the libraries. Sometimes such results could be surprising, showing that there are some significant redundancies, eg. when developing multiplicative and additive versions of algebraic structures.

Another direct use is for tracking the items' histories during wiki-like refactoring: The library is in the case of the formal Mizar/CoRN wiki prototypes held in a Git repository already, and the file/directory-based SHA-1 hashes are very quickly computed each time a commit is made. Given this speed, it is conceivable that the above-explained item-based SHA-1 computation would not be significantly more expensive, and could be made an automated part of each repository commit (a particularly elegant way would be just to replace the Git default SHA-1 application with this one, and let Git use it internally instead). A side-effect of such an enhanced commit would be a mapping of human item names to the content-based ones, and a semantic diff report, saying which items have been moved and which have changed. This kind of experiment is probably readily feasible, for example on the recent one hundred versions of the Mizar library, or on the Coq-contribs repository.

Another immediate use would be for the search and automated reasoning/search tools over the libraries: A query to such tools would always be done in the content-based encoding. Thus, if a new user is (as is quite often the case with large formal/code libraries) ignorant of the particular naming conventions, and partially duplicates the concept hierarchy, this would not be a problem when asking some strong search/inference tools for an advice based on what is already in the library. The automated reasoning/search tools would work on recursive content-based encoding of both the library, and the new user's queries and context, and provide the user also with advice about the relevant library names that should be used.

On the other hand, direct content-based naming is often unwanted. For example, in Wikipedia an article typically keeps its name for long time, even though its content changes. Such a stable name however naturally creates an equivalence class in the semantic space of SHA-1 hashes: some authority claims that a set of SHA-1 hashes have the same semantics according to some (stronger) point of view. Such equivalence classes could be used productively to allow human (or other) influence on the recursive content-based naming, propagating the equivalence classes using congruence-closure algorithms. This could serve as a semantic analogy of text tools like `diff`, which can be instructed to ignore white space changes. The user would in our case specify seeding equivalence pairs, whose (propagated) influence he would like the semantic diff (and strong semantic tools like ATPs) to ignore.

An interesting related problem is providing suitably normalized object representations, before they get consumed by the SHA-1 hash. For example, associative-commutative operators (like plus) are typically normalized into set-like representations (ignoring brackets and order) by ATP systems. This approach complements the previous one: semantic equivalence classes are not specified manually, but by specifying a content-normalization algorithm before the hashing function is applied. For semi-formal wikis, where for example only a small article/section part (say, the theorem) is semantically encoded, such normalizing function might consist just in ignoring the non-semantic parts of the articles/sections.

One limitation is that functions like SHA1 are only "practically" secure, not theoretically. If we are

very unlucky, we could for example infer a new theorem based on a clash between two differently defined concepts. An obvious remedy is to re-check the theorems in a safe encoding.

References

[Ban90a] Grzegorz Bancerek. Cardinal arithmetics. *Formalized Mathematics*, 1(3):543–547, 1990.

[Ban90b] Grzegorz Bancerek. Cardinal numbers. *Formalized Mathematics*, 1(2):377–382, 1990.

[CFGW04] Luís Cruz-Filipe, Herman Geuvers, and Freek Wiedijk. C-corn, the constructive coq repository at nijmegen. In Andrea Asperti, Grzegorz Bancerek, and Andrzej Trybulec, editors, *MKM*, volume 3119 of *Lecture Notes in Computer Science*, pages 88–103. Springer, 2004.

[G$\ddot{3}$1] Kurt Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Monatshefte Fur Mathematik*, 38-38:173–198, 1931.

[Jar30] V. Jarník. O jistém problému minimalním (about a certain minimal problem). *Prace Moravské Prirodovedecké Společnosti*, 6:57–63, 1930.

[Jas34] S. Jaskowski. On the rules of suppositions. *Studia Logica*, 1, 1934.

[Pel99] F. J. Pelletier. A brief history of natural deduction. *History and Philosophy of Logic*, 20:1 – 31, 1999.

[Pri57] R. C. Prim. Shortest connection networks and some generalizations. *Bell Systems Technical Journal*, pages 1389–1401, November 1957.

[Sch02] S. Schulz. E – a brainiac theorem prover. *Journal of AI Communications*, 15(2-3):111–126, 2002.