Application for a Computing Time Project on the RWTH Compute Cluster

New project proposal for project "pmGenerator"

Period: 01.04.2023 – 31.03.2024

An exhaustive generator to find shortest known condensed detachment proofs and verify minimality, focussing on popular theorems from Principia Mathematica and Metamath's mmsolitaire project

Samiro Discher

March 21, 2023

Principal investigator: Thomas Noll*
Technical contact & project contributor: Samiro Discher*

* Lehrstuhl für Informatik 2 (Softwaremodellierung und Verifikation)

Abstract

Utilization of a proof generator with shared memory parallelization making heavy use of Intel's oneTBB library, and distributed memory parallelization via MPI for a computing-intensive filtering method.

Given a deductive system with a finite amount of axioms and rules, one can assign a unique symbol to each of those primitives in order to represent proofs in so-called *condensed detachment* notation. These strings are a very concise, yet complete specification of proofs, i.e. every theorem has a condensed detachment proof. Essentially, they are formulas in prefix notation, where rules in $\mathcal{R}$ of arity $n : \mathcal{R} \to \mathbb{N}$ take subproofs as arguments, with axioms in $\mathcal{A}$ being primitives. For example `DD211` for $n(\mathtt{D}) = 2$ and $\mathtt{1}, \mathtt{2} \in \mathcal{A}$ means axioms `2` and `1` serve as respective inputs for rule `D` to result in a formula which then serves as a first input for rule `D` with axiom number 1 as second input. The evaluation requires unification of formulas so that all inputs are specified to match the pattern as required by the rule. This way, the most general possible conclusion is evaluated. If unification fails, the given string is not a proof. This concept was developed by the Irish logician Carew Meredith in the 1950s.

Formulas within the deductive system's language are not explicitly part of a condensed detachment proof, but must be stated separately. Take for instance a system with rule $(R1)\ (\vdash \psi, \vdash \psi \to \varphi) \Rightarrow\ \vdash \varphi$, and $(A1)\ \psi \to (\varphi \to \psi)$, $(A2)\ (\psi \to (\varphi \to \chi)) \to ((\psi \to \varphi) \to (\psi \to \chi))$ as axiom schemas, then with $\mathtt{D} := (R1)$, $\mathtt{1} := (A1)$ and $\mathtt{2} := (A2)$, the proof `DD211` means:

1. $\psi \to (\varphi \to \psi)$	$(A1)$
2. $\psi \to ((\varphi \to \psi) \to \psi)$	$(A1)$
3. $(\psi \to ((\varphi \to \psi) \to \psi)) \to ((\psi \to (\varphi \to \psi)) \to (\psi \to \psi))$	$(A2)$
4. $(\psi \to (\varphi \to \psi)) \to (\psi \to \psi)$	$(R1) : 2, 3$
5. $\psi \to \psi$	$(R1) : 1, 4$

Since formulas blow up in size for increasing lengths of proofs, using condensed detachment clearly leads to significant savings in data to be processed. While a conventional proof may require fewer steps due to multiple formulas referencing the same line, unfolding and reducing a condensed detachment proof in this way can be easily done.

Conclusions are formulas that are part of the deductive system and important to the computation. To further save memory and for performance reasons, they are stored and processed in a Polish notation variant, where variable names are $0, 1, 2, \ldots$ based on order of occurrence. Consecutive variable names are separated by dots, so they can be

identified also for more than 10 variables. For the previous example, this leads to $(A1)$, $(A2)$ and $\psi \to \psi$ being represented by `C0C1.0`, `CC0C1.2CC0.1C0.2` and `C0.0`, respectively. Condensed detachment proofs have the additional advantage that finding subproofs only requires a substring search, so longer proofs can be shortened by replacing subproofs with conclusions for which shorter proofs are known, by using search and replace on their strings. Consequently, having minimal subproofs at hand helps with shortening larger proofs, which can be of interest for example in the field of proof complexity. The tool *pmGenerator* generates exhaustive files for minimal proofs of increasing lengths to reduce proofs based on a popular calculus by Polish logician Jan Łukasiewicz. Metamath, a project aiming to advance computer-verified formalization of mathematics, has a database which includes, but is not limited to, the shortest known proofs of theorems from Principia Mathematica. This database, named *pmproofs.txt*, can be parsed and composed to improved versions, using pmGenerator. While the tool currently only supports its fixed set of primitives, it can be extended to support arbitrary rules and axioms in order to explore different proof systems as well. I am already working on such an implementation as part of another tool, which might become part of this computing time project in the future.

An idea behind efficiently generating minimal proofs of length $n+2$ for known minimal proofs up to length n is to implement a pushdown automaton for a context-free grammar with start symbol S, nonterminals $\{S, A\} \cup \{N_x \,|\, x > 0 \text{ odd, and } x \leq n\}$, and production rules

$S \to N_1 \mid \ldots \mid N_n \mid A$ (S produces a superset of all representative proofs.)
$A \to$ `proofLengthCombinations`(n)
$N_1 \to \{p \,|\, p \text{ is representative proof of length } 1\}$
$\vdots$
$N_n \to \{p \,|\, p \text{ is representative proof of length } n\}$,

where A produces all proofs of at least length $n+2$, using combinations of $\{N_1, \ldots, N_n, A\}$, e.g. for $n = 3$:
$A \to \mathtt{D}N_1N_3 \mid \mathtt{D}N_3N_1 \mid \mathtt{D}N_3N_3 \mid \mathtt{D}N_1A \mid \mathtt{D}AN_1 \mid \mathtt{D}N_3A \mid \mathtt{D}AN_3 \mid \mathtt{D}AA$
By starting the automaton with stack $[A]$ and also limiting all outcomes to lengths of at most $n + 2$, this leads to efficient iteration of only candidates of length $n + 2$. Invalid candidates are skipped after resulting in parse errors, valid minimal ones are inserted into a hash map with conclusions as keys. Filtering also based on whether a conclusion has a schema that is provable in at most as many steps, is an extensive computation due to lack of known heuristics and high number of checks, but it helps with dampening the data explosion.