Team

Richard Li - YL5573

Background

Presburger Arithmetic (PrA) is a weak, completely axiomatisable logic theory first introduced in 1929. Despite including a weak axiom schema of induction, PrA is nevertheless a decidable theory (unlike more typical arithmetics like Peano Arithmetic or ZF(C), both of which are only semi-decidable). Specifically, one can design an algorithm such that, given any sentence, i.e. formula candidate, of finite length, the algorithm outputs correctly whether the sentence is provable from the axioms of PrA.

Typically, theories with induction encounter decidability challenges when universal and existential quantifiers are involved: to prove false an unbounded universal (Π_1) statement, a checking algorithm must enumerate all instances of the bounded variable (of which there are usually infinitely many) that follow from the $\forall$ sign, whereas, if the statement is true, the checking program ends up running forever.

Roughly speaking, PrA does not suffer from such difficulties by way of quantifier reduction: any unbounded universal or existential statement in PrA can be first written into conjunctive normal form (CNF), then unravelled into a long unbounded-quantifier-free (Δ_0) sentence, whose truth value is decidable via mechanical enumeration of all pertinent variables' truth values. The reduction step is implemented via modulo arithmetic.

Project Overview

Parallelise (and further optimise) the Haskell Data.Integer.Presburger library's implementation of Check :: Formula -> Bool

Initial Ideas

1. The library uses Data.IntMap, which can likely be upgraded to Data.IntMap.Strict

2. Since PrA's logical operators are identically recursively axiomatised as the standard, classical sentential logic, theorem-checking is similarly recursively defined - a good place to par away part of the logical connectives onto a separate spread
3. Existential enumerations should also be parallelisable

Forseeable Challenges

1. CNF conversion seems really hard to parallelise, and might take up a sizeable chunk of overhead
2. A really complex unravelled Δ_0 sentence may lead to spark overflow - won't know until early testing with Threadscope

Input Data

I will start by coming up with some arbitrary sentences that are long and complex

Miscellaneous

There is a separate repo that uses a Deterministic Finite Automata approach - worth exploring if time permits.

References

1. https://hackage.haskell.org/package/presburger-0.3/docs/Data-Integer-Presburger.html
2. https://github.com/konn/presburger-dfa
3. https://www.cs.cmu.edu/~emc/spring06/home1_files/Cooper.pdf
4. https://link.springer.com/chapter/10.1007/11532231_20