Verified Implementation of an Efficient Term-Rewriting Algorithm for Multiplier Verification on ACL2

Mertcan Temel[1,2]

mert@utexas.edu

[1]University of Texas at Austin, Austin, TX, USA

[2]Intel Corporation, Austin, TX, USA

May 27, 2022 (ACL2 2022)

Introduction

- Integer multipliers have been around a long time but their verification is still hard.
  - An infamous, easy-to-state hardware verification problem
  - Booth Encoding and Wallace tree design algorithms make them harder to verify
  - SAT Solvers, BDDs etc. blow up for even small designs

- My PhD work proposed an efficient, rewrite-based method that is:
  - widely applicable (tested for 250+ benchmarks),
  - scalable (1024x1024-bit multipliers proved in 5 minutes),
  - provably correct (multiplier verification procedure is verified using ACL2)

- Today, I will talk about the implementation details instead of the rewrite algorithm itself.
  - See our CAV20 and FMCAD21 papers for the high-level algorithm

Implementation Goals

- Easily pluggable to different simulators: the DE system, SVL, SVTV ...
- Verification algorithm should be easily modifiable/extensible
- Variations of multipliers (e.g., dot product) can be verified
  - Implementing using a rewriter helped with these three.

- Needs to verify designs very quickly
- Program itself needs to be correct and verified
  - Verifiability and proof-time performance were sometimes at odds with each other.

Example Conjecture to Prove

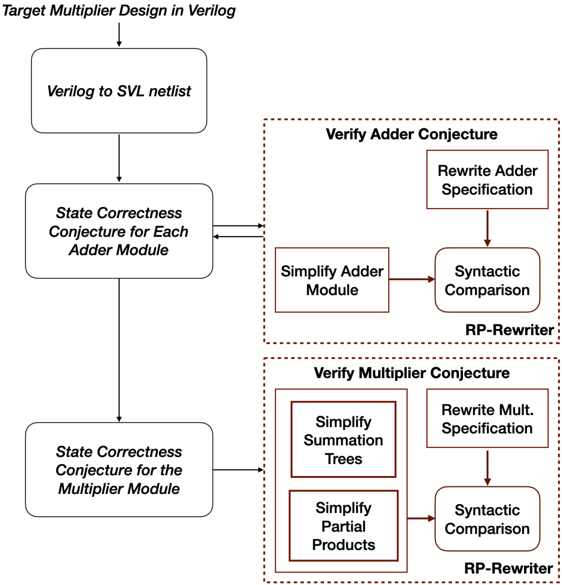
Our goal is to prove such conjectures:

```
(defthm multiplier_is_correct
  (implies (and (integerp a)
                (integerp b))
           (equal (svl-run (list a b) <signed_64x64_mult>)
                  (loghead 128
                           (* (logext 64 a)
                              (logext 64 b))))))
```

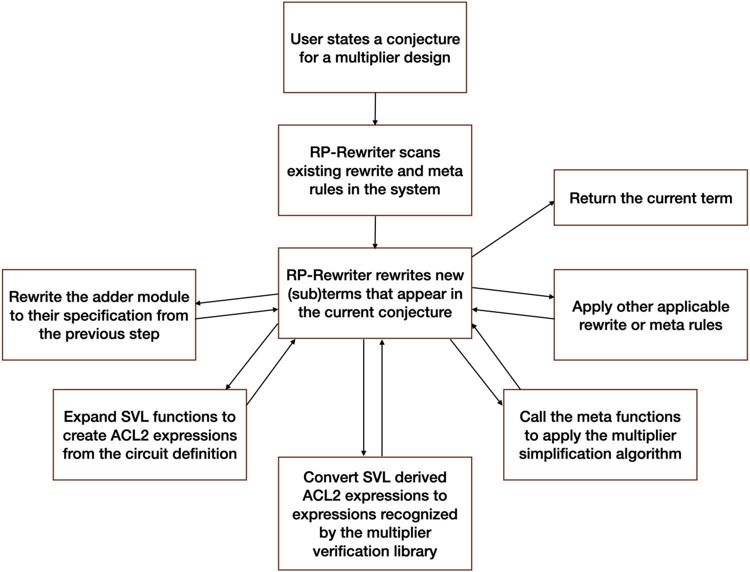
Statement of this conjecture can be changed to, say:

- have a specification for multiply-accumulate, dot-product...
- use a different design simulator.

Verification Flow



Rewriter Flow



User states a conjecture for a multiplier design

RP-Rewriter scans existing rewrite and meta rules in the system

RP-Rewriter rewrites new (sub)terms that appear in the current conjecture

Return the current term

Rewrite the adder module to their specification from the previous step

Apply other applicable rewrite or meta rules

Expand SVL functions to create ACL2 expressions from the circuit definition

Call the meta functions to apply the multiplier simplification algorithm

Convert SVL derived ACL2 expressions to expressions recognized by the multiplier verification library

Challenges

Some of the challenges when developing this method:

- Multipliers are DAG, but rewriter works on terms as trees.
  - Solution: make sure unique expressions are rewritten only once.
- Only unmodifiable linked-lists are allowed.
  - Solution: Meta-functions to reduce rewriting steps
- The simplification algorithm rewrites two-valued functions to high-level arithmetic functions: makes it difficult to detect some terms are still two-valued.
  - Solution: Use RP-Rewriter's side-conditions feature to attach and remember properties.
- Comparison of large terms are expensive.
  - Solution: Calculate and attach hash values to terms as an extra logically redundant argument. E.g., (s args) → (s hash args)

Conclusions

- An efficient method to verify integer multipliers. Compared to the other state-of-the-art tools:
  - It can verify large multipliers in a much shorter amount of time;
  - It has wider applicability;
  - It is used in real-world designs, and;
  - It is verified.
- A simple term-rewriting approach successfully working on a widely-known problem
- An example of how a theorem prover can be used to implement a program competing with other tools