1 Appendix

1.1 Grammars

In the following, we show the crci and strpred (extension of strterm where `s` corresponds to `Start` in strterm) grammars.

Listing 1. crci

```
(synth-fun f ((x Bool) (y Bool)
  (z Bool) (w Bool)) Bool (
 (Start Bool ((and d1 d1) (not d1)
  (or d1 d1) (xor d1 d1)))
 (d1 Bool (x (and d2 d2) (not d2)
  (or d2 d2) (xor d2 d2)))
 (d2 Bool (w (and d3 d3) (not d3)
  (or d3 d3) (xor d3 d3)))
 (d3 Bool (y (and d4 d4) (not d4)
  (or d4 d4) (xor d4 d4)))
 (d4 Bool (z))))
```

Listing 2. strpred

```
(synth-fun f ((x String)
  (y String) (z Int))
  Bool (
 (Start Bool (
  true false
  (not Start)
  (= s s)
  (str.prefixof s s)
  (str.suffixof s s)
  (str.contains s s)))
 ...))
```

1.2 Example Output

We provide sample output of CVC4 in its rewrite rule enumeration mode. For each grammar, we give the results of CVC4 with no rewriter (none), its standard rewriter (std), and the rewriter we generated as a result of this work so far (ext). In each case, the output is a list of unoriented pairs indicating possible rewrites. All variables $x, y, z, w, s, t, \ldots$ should be interpreted as universal, in that the rewrite holds for all values of $x, y, z, w, s, t, \ldots$. Intuitively, the output when using none corresponds to a list of rewrites over terms in the grammar if the developer was writing a rewriter from scratch; the output when using std (resp. ext) corresponds to the simplest rewrites, measured in term size, over the terms from the grammar that the given rewriter currently can not infer.

1.3 strterm

Listing 3. none

```
(= (str.at "A" 1) "")
(= (str.at "B" 0) "B")
(= (str.at "B" 1) "")
```

```
(= (str.at "" 0) "")
(= (str.at "" 1) "")
(= (str.at "" z) "")
(= (str.substr x 0 0) "")
(= (str.substr x 0 1) (str.at x 0))
(= (str.substr x 1 0) "")
(= (str.substr x 1 1) (str.at x 1))
```

Listing 4. std

```
(= (str.at "" z) "")
(= (str.substr "A" 1 z) "")
(= (str.substr "A" z z) "")
(= (str.substr "B" 1 z) "")
(= (str.substr "B" z z) "")
(= (str.substr "" 0 z) "")
(= (str.substr "" 1 z) "")
(= (str.substr "" z z) "")
(= (str.replace x x y) y)
(= (str.replace y y "A") (str.replace x x "A"))
```

Listing 5. ext

```
(= (int.to.str (str.indexof "B" "" z))
   (int.to.str (str.indexof "A" "" z)))
(= (str.at x (str.indexof x "" 1)) (str.at x 1))
(= (str.at x (str.indexof x "" z)) (str.at x z))
(= (str.at x (str.indexof "A" x 1)) "")
(= (str.at x (str.indexof "B" "" z))
   (str.at x (str.indexof "A" "" z)))
(= (str.at x (str.indexof "" x 0)) "")
(= (str.at x (str.indexof "" x z)) "")
(= (str.at "A" (- 0 z)) (str.at "A" z))
(= (str.at "A" (- z 1)) (str.at "A" (- 1 z)))
(= (str.at "A" (+ z z)) (str.at "A" z))
```

1.4 strpred

Listing 6. none

```
(= (= x x) true)
(= (str.prefixof x x) true)
(= (str.suffixof x x) true)
(= (str.contains x x) true)
(= (str.suffixof x "A") (str.prefixof x "A"))
(= (str.suffixof x "B") (str.prefixof x "B"))
(= (str.prefixof x "") (= x ""))
(= (str.suffixof x "") (= x ""))
(= (str.contains x "") true)
(= (= y x) (= x y))
```

Listing 7. std

```
(= (str.suffixof x "A") (str.prefixof x "A"))
(= (str.suffixof x "B") (str.prefixof x "B"))
(= (str.prefixof x "") (= x ""))
(= (str.suffixof x "") (= x ""))
(= (str.contains x "") true)
(= (str.contains "A" x) (str.prefixof x "A"))
(= (str.contains "B" x) (str.prefixof x "B"))
(= (str.prefixof "" x) true)
```

```
(= (str.suffixof x (int.to.str 0))
   (str.prefixof x (int.to.str 0)))
(= (str.suffixof x (int.to.str 1))
   (str.prefixof x (int.to.str 1)))
```

Listing 8. ext

```
(= (= x (str.at x 1)) (= x ""))
(= (str.suffixof x (str.at y 0))
   (str.prefixof x (str.at y 0)))
(= (str.suffixof x (str.at y 1))
   (str.prefixof x (str.at y 1)))
(= (str.suffixof x (str.at y z))
   (str.prefixof x (str.at y z)))
(= (= x (str.substr x 1 z)) (= x ""))
(= (= x (str.substr x z z)) (= x ""))
(= (str.suffixof x (str.substr "A" 0 z))
   (str.prefixof x (str.substr "A" 0 z)))
(= (str.suffixof x (str.substr "B" 0 z))
   (str.prefixof x (str.substr "B" 0 z)))
(= (str.prefixof x (str.++ "A" x))
   (str.suffixof x (str.++ x "A")))
(= (str.suffixof x (str.++ "A" "A"))
   (str.prefixof x (str.++ "A" "A")))
```

1.5 bvterm$_4$

Listing 9. none

```
(= (bvand s s) s)
(= (bvor s s) s)
(= (bvadd s #b0000) s)
(= (bvmul s #b0000) (bvlshr s s))
(= (bvand s #b0000) (bvlshr s s))
(= (bvlshr s #b0000) s)
(= (bvor s #b0000) s)
(= (bvshl s #b0000) s)
(= (bvadd t s) (bvadd s t))
(= (bvmul t s) (bvmul s t))
```

Listing 10. std

```
(= (bvlshr t t) (bvlshr s s))
(= (bvneg (bvlshr s s)) (bvlshr s s))
(= (bvadd s (bvnot s)) (bvnot (bvlshr s s)))
(= (bvadd s (bvnot #b0000)) (bvnot (bvneg s)))
(= (bvadd s (bvlshr s s)) s)
(= (bvmul s (bvlshr s s)) (bvlshr s s))
(= (bvand s (bvlshr s s)) (bvlshr s s))
(= (bvlshr s (bvlshr s s)) s)
(= (bvor s (bvlshr s s)) s)
(= (bvshl s (bvlshr s s)) s)
```

Listing 11. ext

```
(= (bvshl (bvshl s s) s) (bvshl (bvadd s s) s))
(= (bvlshr s (bvnot (bvneg s))) (bvlshr (bvadd s s) s))
(= (bvadd s (bvnot (bvadd s s))) (bvnot s))
(= (bvshl s (bvneg (bvmul s s)))
   (bvlshr s (bvneg (bvmul s s))))
(= (bvshl s (bvnot (bvmul s s)))
```

```
   (bvlshr s (bvnot (bvmul s s))))
(= (bvlshr s (bvneg (bvshl s s))) (bvlshr s (bvshl s s)))
(= (bvshl s (bvneg (bvshl s s))) (bvlshr s (bvshl s s)))
(= (bvshl s (bvnot (bvshl s s)))
   (bvlshr s (bvnot (bvmul s s))))
(= (bvshl s (bvneg (bvlshr t s)))
   (bvlshr s (bvneg (bvlshr t s))))
(= (bvlshr s (bvnot (bvlshr t s)))
   (bvlshr s (bvnot (bvmul s s))))
```

1.6 crci

Listing 12. none

```
(= (or x x) (and x x))
(= (not (or w w)) (not (and w w)))
(= (and x (or w w)) (and x (and w w)))
(= (or x (or w w)) (or x (and w w)))
(= (xor x (or w w)) (xor x (and w w)))
(= (and x (xor w w)) (xor x x))
(= (or x (xor w w)) (and x x))
(= (xor x (xor w w)) (and x x))
(= (and (not w) x) (and x (not w)))
(= (or (not w) x) (or x (not w)))
```

Listing 13. std

```
(= (or x x) (and x x))
(= (not (or w w)) (not (and w w)))
(= (and x (or w w)) (and x (and w w)))
(= (or x (or w w)) (or x (and w w)))
(= (xor x (or w w)) (xor x (and w w)))
(= (or x (xor w w)) (and x x))
(= (and (not w) x) (and x (not w)))
(= (or (not w) x) (or x (not w)))
(= (xor (not w) x) (xor x (not w)))
(= (or (and w w) x) (or x (and w w)))
```

Listing 14. ext

```
(= (xor (and w (not y)) (not (or y (not z))))
   (and (not (and y y)) (xor w (not (not z)))))
(= (xor (and w (not y)) (not (or y (and z z))))
   (and (not (and y y)) (xor w (not (and z z)))))
(= (xor (and w (not y)) (and w (not (not z))))
   (and (and w w) (not (xor y (and z z)))))
(= (xor (and w (not y)) (or w (not (not z))))
   (xor (not (and y y)) (or w (xor y (not z)))))
(= (xor (and w (not y)) (and w (not (and z z))))
   (and (and w w) (not (xor y (not z)))))
(= (xor (and w (not y)) (or w (not (and z z))))
   (xor (not (and y y)) (or w (xor y (and z z)))))
(= (xor (and w (not y)) (and w (and y (not z))))
   (and (and w w) (not (and y (and z z)))))
(= (xor (and w (not y)) (or w (and y (not z))))
   (and (not (not y)) (or w (not (and z z)))))
(= (or (and w (not y)) (xor w (and y (not z))))
   (xor (not w) (not (and y (not z)))))
(= (xor (and w (not y)) (xor w (and y (not z))))
   (and (not (not y)) (xor w (not (and z z)))))
```

1.7 Challenging Equivalence Checks

In this section, we list the rewrites from the section on "Evaluating SMT Solvers for Verifying Rewrites" that we were not able to prove automatically and that we either verified manually or in a semi-automated fashion. We list them as interesting challenges for tool developers.

Listing 15. strterm

```
(= (str.at (int.to.str z) z)
                   (int.to.str (str.indexof x x z)))
(= (str.replace (str.replace x y x) x y)
                   (str.replace x (str.replace x y x) y))
```

Listing 16. strpred

```
(= (str.prefixof x (str.replace y x y))
                   (str.prefixof x y))
(= (str.suffixof x (str.replace y x y))
                   (str.suffixof x y))
(= (str.suffixof "A" (str.replace x "A" ""))
   (str.suffixof "A" (str.replace x "A" "B")))
(= (str.contains "A" (str.replace x "A" ""))
   (str.prefixof x (str.++ "A" "A")))
(= (str.contains "B" (str.replace x "A" ""))
   (str.contains "A" (str.replace x "B" "")))
(= (str.suffixof "B" (str.replace x "B" ""))
   (str.suffixof "B" (str.replace x "B" "A")))
(= (str.contains "B" (str.replace x "B" ""))
   (str.prefixof x (str.++ "B" "B")))
(= (str.prefixof (str.++ "A" "A") x)
   (str.prefixof "A" (str.replace x "A" "")))
(= (str.prefixof (str.++ "B" "B") x)
   (str.prefixof "B" (str.replace x "B" "")))
(= (str.suffixof (str.replace x "A" "") x)
   (str.prefixof x (str.replace x "A" x)))
(= (str.suffixof (str.replace x "B" "") x)
   (str.prefixof x (str.replace x "B" x)))
```

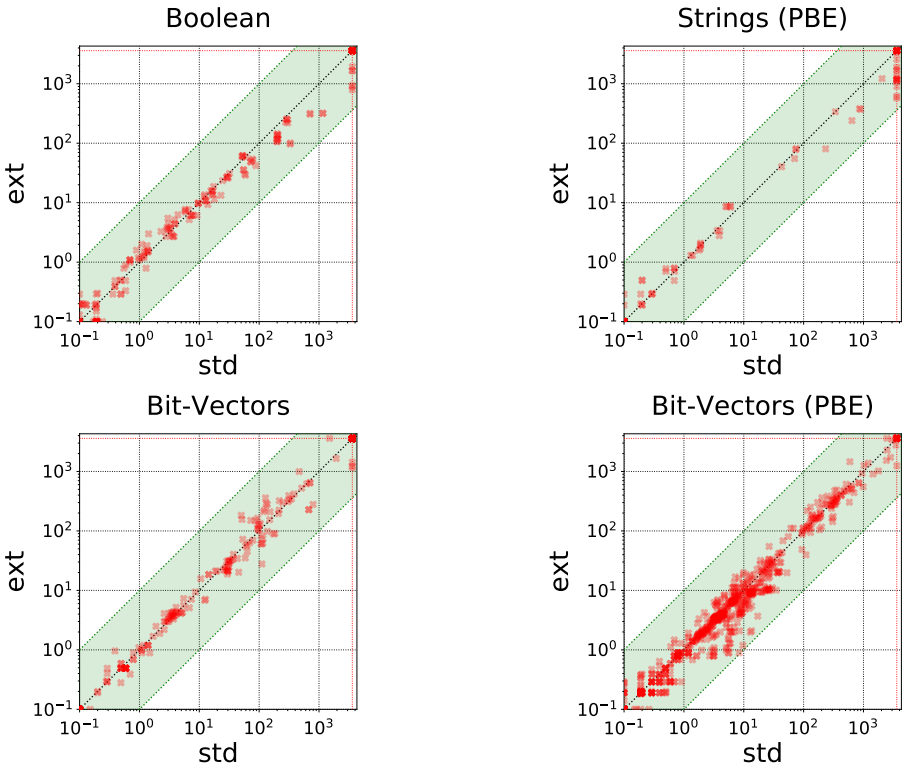
Listing 17. bvterm

```
(= (bvlshr (bvmul s s) (bvshl s s))
                   (bvmul s (bvlshr s (bvshl s s))))
(= (bvlshr (bvmul s s) (bvshl t s))
                   (bvmul s (bvlshr s (bvshl t s))))
```

We solved all but the first two equivalences in strpred using a different configuration of CVC4. We verified the bvterm rewrites semi-automatically by breaking the problem down into smaller chunks and solving those chunks with an SMT solver. We checked the remaining four equivalences in strterm and strpred using pencil-and-paper proofs.

1.8 Scatter Plots Showing Impact of Rewrites on SyGuS Solving

The following scatter plots show the difference between ext and std on SyGuS solving in more detail. They were generated from the same data as the table in Figure 3.


Boolean
Strings (PBE)
Bit-Vectors
Bit-Vectors (PBE)
ext
std
10^{-1}
10^{0}
10^{1}
10^{2}
10^{3}