An Eager SMT Solver for Quantifier-Free Algebraic Data Type Queries

Amar Shah

Algebraic Datatypes Background

Algebraic Data Types (ADTs) are an exciting construct found in modern programming languages. Consider the following declaration:

```
(declare-datatype List (
    (Nil)
    (Cons (Head Int) (Tail List))))
```

This theory gives us three types of functions:

- Constructors like `Cons` and `Nil` which are used to build terms
- Selectors like `Head` and `Tail` which are used to deconstruct terms that are created by constructors
- Testers like `is-Nil` and `is-Cons` which tell us whether a given term was created by a constructor

ADT objects must be finite (i.e. well-founded). Given the earlier declaration, an example query in the theory **ADT**:

```
(and (not (= x Nil)) (not (= y Nil)))
(and (= x (Tail y)) (= y (Tail x)))
```

This is clearly **UNSAT** because it creates an infinite cycle:

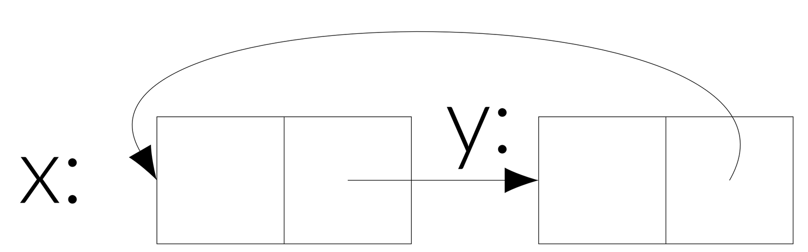

Figure 1. The lists x and y from the query will be infinite (and thus our model is not well-founded)

Reduction from ADT to UF

We propose a satisfiability modulo theory (SMT) solver for ADT queries. Our solver is eager: it reduces quantifier-free ADT queries to quantifier-free Uninterpreted Functions (**UF**) queries.

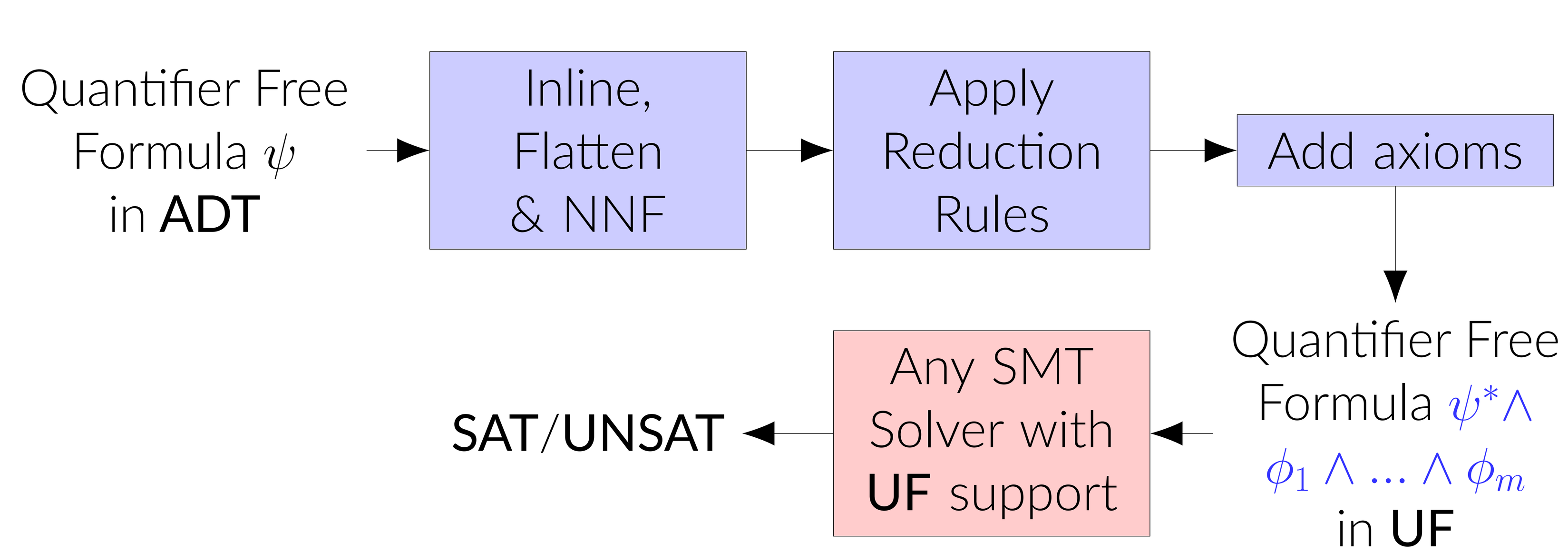

Figure 2. Our Reduction Process. We preprocess and apply rules/axioms to get a reduced formula in **UF** that can be solved by most out of the box SMT solvers

Using our reduction, we have equipped (almost) every solver with ability to solve ADT queries.

Reduction Rules

Once we have ψ in flattened NNF, we can transform it to **UF** by applying the following rules to atomic formulas to make ψ^*:

A. $f(t_1...t_l) = t \implies f(t_1,...t_l) = t \wedge is\text{-}f(t) \wedge \bigwedge_{i=1}^{l} f^i(t) = t_i$

B. $f^j(t) = t_j \implies f^j(t) = t_j \wedge [\exists t_1...t_l[f(t_1,...,t_l) = t \wedge \bigwedge_{k=1}^{l} f^k(t) = t_k]]$

One example from our earlier query is:

$$\texttt{(= x (Tail y))} \implies \texttt{(= x (Tail y))} \wedge [\exists v \texttt{(= y (Cons v x))} \wedge \texttt{(= v (Head y))}]$$

Reduction Axioms

Once we have ψ^* we add additional axioms $\phi_1 \wedge ... \wedge \phi_m$ the first two ensure that testers behave properly. For each term $t : \sigma$, we add:

1. For any tester in $\{is\text{-}f_i\}_{1 \le i \le |C_\sigma|}$, we add:
   $\phi := \bigvee_{i=1}^{|C_\sigma|} [is\text{-}f_i(t) \wedge \bigwedge_{j=1, j \neq i}^{|C_\sigma|} \neg is\text{-}f_j(t)]$
2. For any constant constructor $c : \sigma$, we add: $is\text{-}c(t) \leftrightarrow c = t$

These axioms and rules ensure constructors, selectors, and testers behave well with one another.

Well-Foundedness Axiom

We also need to add another axiom to ensure well-foundedness. Our example from before can be generalized to more variables so the cycles can be made of arbitrary length.

Let k be the number of variables that appear in the input query in flat NNF. Then we add a final axiom:

3. For each $t, s \in T$ where we know that s is a subterm of t up to depth k, we add the axiom $s \neq t$

We have a finite, quantifier-free reduction that can handle well-foundedness!

Soundness and Completeness of Reduction

Theorem 1: Say ψ is an **ADT**-formula that is in flat NNF form. If we define T as above, then $\textbf{ADT} \models \psi \leftrightarrow \textbf{UF} \models \psi^* \wedge \phi_1 \wedge ... \wedge \phi_m$ where we compute ψ^* from ψ using Rules A and B and $\phi_1, ...\phi_m$ using Axioms 1, 2 and 3.

Results on Synthetic Benchmarks

To test runtime we ran three popular SMT solvers *z3*, *CVC5* and *mc2* on 10,000 randomly generated **List** queries:

Table 1. Runtime with and without Reduction (seconds)

(Vars, Asserts)	(2, 4)	(4, 4)	(4, 8)
Z3	392.26	387.40	394.54
Z3 w/ Reduction	237.28	236.63	240.37
CVC5	178.87	**179.06**	**178.68**
CVC5 w/ Reduction	186.08	226.15	230.57
mc2 w/ Reduction	**177.21**	181.51	186.87

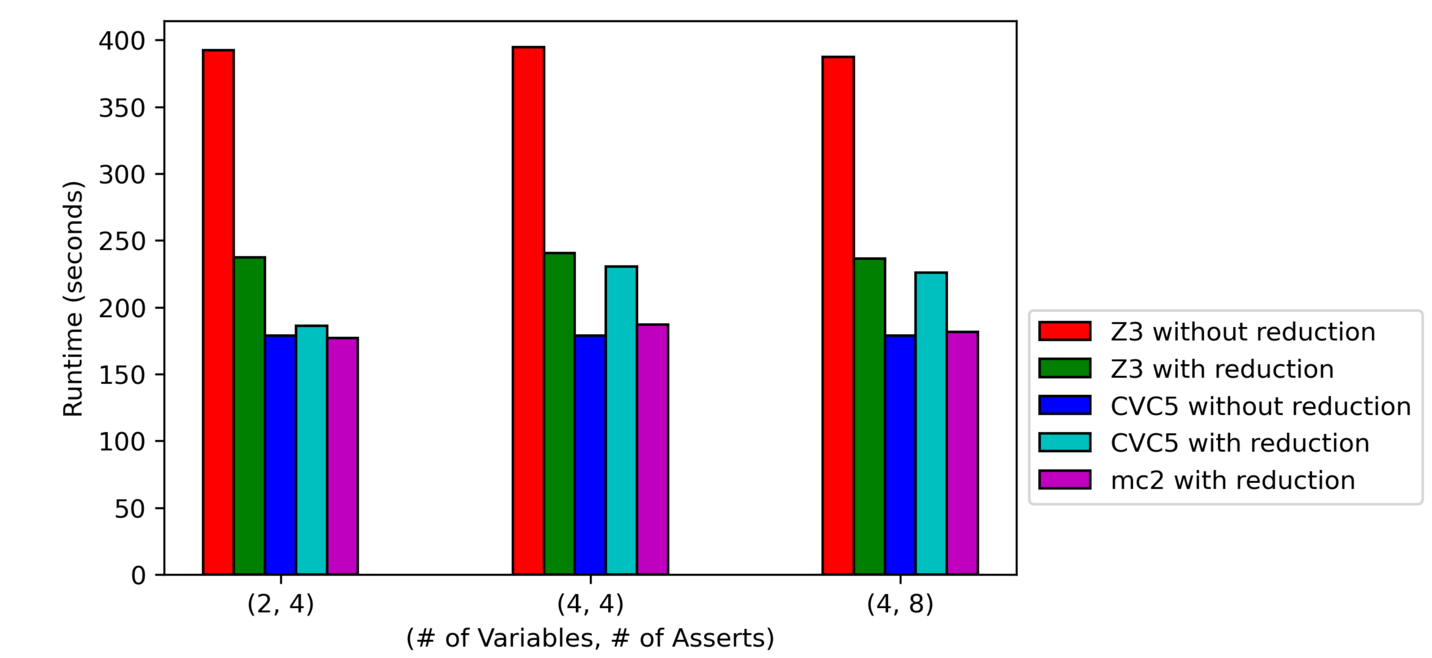

Figure 3. Total time it took each solver to solve 10,000 synthetic queries

mc2 with reduction is competitive with state-of-the-art! Without our reduction mc2 cannot handle Algebraic Data Types.

Results on SMTComp Benchmarks

We tested our reduction on a suite of benchmarks from SMTComp, originally from Bouvier '21.

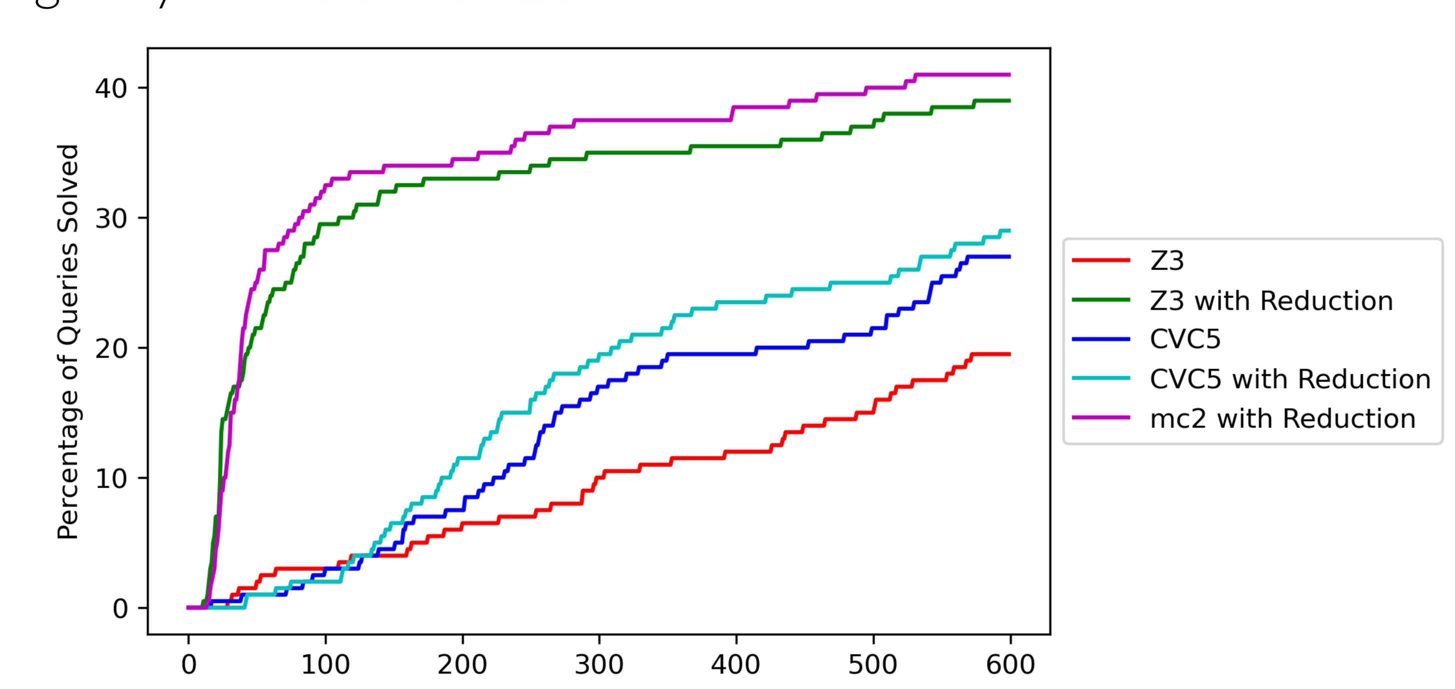

Figure 4. Percentage of 50 queries solved over time

Z3 and mc2 with our reduction are able to beat the state-of-the-art on real world benchmarks!

amarshah1000@berkeley.edu