DRYADSYNTH : A Concolic SyGuS Solver

Kangjing Huang Xiaokang Qiu Yanjun Wang
School of Electrical and Computer Engineering
Purdue University
West Lafayette, Indiana 47907–2035
Email: {kangjinghuang,xkqiu,wang3204}@purdue.edu

***Abstract*—this paper presents DRYADSYNTH , a concolic SyGuS solver. The synthesis algorithm is CEGIS-based and combines enumerative search and symbolic search.**

DRYADSYNTH is a Syntax-Guided Synthesis (SyGuS) synthesizer that combines explicit search and symbolic search. The solver currently supports synthesis for Conditional Linear Integer Arithmetic (CLIA) and Invariants (INV). For both the two tracks, a conditional arithmetic function or predicate is represented using a decision-tree data structure.

Decision Tree Representation

To represent a n-ary function $f(x_1, \ldots, x_n)$ in LIA, we first normalize to the *decision tree normal form* described in Figure 1. It is not hard to see that every LIA function can be converted to this normal form. The proof relies on the fact that every atomic LIA equation or inequation can be rewritten to the form of $e \geq 0$, where e is a linear expression. Then the normal form expression can be represented as a binary tree in which every node contains $n+1$ integer fields $c_0, \ldots, c_n$, representing the expression $c_0 + \sum_{1 \leq i \leq n} c_i \cdot x_i$. Each decision node (non-leaf node) tests whether the associated expression is nonnegative and proceeds to the "true" child or "false" child. Each leaf node determines the value of the function using its associated expression. For example, the binary max function

$$max2(x_1, x_2) \stackrel{def}{=} \text{ if } x_1 \geq x_2 \text{ then } x_1 \text{ else } x_2$$

can be represented as the tree shown in Figure 2.

Representing invariants is similar. The only difference is that the leaf node will return true or false, depending on whether the associated expression is nonnegative or not.

We assume that the decision-tree is a full binary tree – if it is not full, one can extend the tree with extra nodes with a tautology condition, e.g., $1 \geq 0$. This allows us to reduce the SyGuS synthesis problem to the problem of searching for:

a) the height of the decision tree;
b) the field values $c_0, \ldots, c_n$ for each node.

Concolic Search

Overall, DRYADSYNTH implements the standard CEGIS framework [2]. The verifier is quite straightforward: let the synthesis specification be $\varphi(f, \vec{x})$, then whenever the synthesizer proposes a candidate solution f_0, the specification $\forall \vec{x}.\varphi(f_0, \vec{x})$ is encoded as a SMT query and solved by Z3 [1], a state-of-the-art SMT solver. If the verification fails, the counterexample (assignments to x_1 through x_n) will be added to the accumulated set of counterexamples.

$$\begin{array}{rrcl} \text{Int Const: } & c_0, c_1, c_2 \ldots & & \\ \text{Expr: } & e, e_1, e_2 & ::= & c_0 + \sum\limits_{1 \leq i \leq n} c_i \cdot x_i \\ \text{Atom Cond: } & \alpha & ::= & e \geq 0 \\ \text{Cond Expr: } & E, E_1, E_2 & ::= & \alpha \mid \text{if } \alpha \text{ then } E_1 \text{ else } E_2 \\ \text{Cond: } & \varphi & ::= & \alpha \mid \text{if } \alpha \text{ then } \varphi_1 \text{ else } \varphi_2 \end{array}$$

Fig. 1. Decision Tree Normal Form

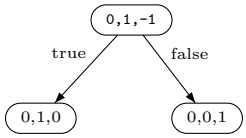

Fig. 2. Representation of the $max2$ function

The synthesizer in the CEGIS framework adopts an enumeration strategy: it enumerates all possible heights of the decision tree, starting from 1. In each iteration of the CEGIS loop, the synthesizer attempts to synthesize a solution with the current height h. If there is no solution, the synthesizer increases the height to $h+1$ and retry.

For each synthesis attempt, the height h is fixed and a set of counterexamples Γ is available. The synthesis task is to find all field values of the full decision tree of height h, representing a function f_0, such that for any counterexample $\vec{d}$ in Γ, the synthesis specification is valid: $\bigwedge\limits_{\vec{d} \in \Gamma} \varphi(f_0, \vec{d})$.

DRYADSYNTH symbolically solves the synthesis task by encoding it to a SMT query. Notice that the full decision tree of height h consists of $2^h - 1$ nodes. We first encode the location of each node to a distinct integer between 0 and $2^h - 2$, and represent the field c_i of node j as c_i^j. Then each occurrence of $f(\vec{d})$ in the synthesis condition with concrete $\vec{d}$ can be encoded using these field variables. For example, if an assignment $(x_1 = 1, x_2 = 2)$ is a counterexample in Γ and the current height is 2, then the expression $f(1, -2)$ can be

represented as a LIA expression

$$\text{if } c_0^0 + c_1^0 - 2c_2^0 \geq 0 \text{ then } c_0^1 + c_1^1 - 2c_2^1 \text{ else } c_0^2 + c_1^2 - 2c_2^2$$

Parallelization and Optimization

While the naive enumeration algorithm incrementally searches all possible height of the decision tree, starting from 1, the algorithm can be naturally parallelized to leverage the multi-cores, if available. If there are n cores available on the machine, the parallel version DRYADSYNTH solver runs the same CEGIS algorithm with n different heights on the n cores independently, i.e., each maintains a separate set of counterexamples. The algorithm starts with the n smallest heights, $\{1, \ldots, n\}$. It also maintains a variable p as the next height to search, starting from $n + 1$. Whenever a core concludes that there is no solution at the current height, it starts a new CEGIS loop at height p, and the value of p gets increased. The whole algorithm stops whenever a core finds a solution.

We observed several things from the synthesis results for SyGuS benchmarks:

1) most of the c_i's are very small. In fact, a lot of benchmarks don't need any coefficient beyond 0, 1, or -1.
2) when a benchmark's solution contains large coefficients, there is usually a large coefficient in the specification as well.
3) if the search space to restricted to solutions with small coefficients only, the performance can be significantly improved.

These observations allow us to optimize the synthesis algorithm by leveraging a multi-stage strategy. For each height h, we split the CEGIS loop into three stages:

1) in each synthesis iteration, we impose extra constraint $c_i^j \leq 1 \wedge c_i^j \geq -1$ for every i and j. Once the synthesis query can't find any solution, move to the next stage –
2) in each synthesis iteration, we impose extra constraint $c_i^j \leq D \wedge c_i^j \geq -D$ where D is a bound such that all coefficients in $\varphi(f, \vec{x})$. Once the synthesis query can't find any solution, move to the next stage –
3) resume the regular synthesis setting, i.e., remove all constraints on coefficients.

We also implemented several simplification strategies. For example, a variable x is irrelevant in an invariant synthesis problem if: a) x does not appear in the pre function or x' does not appear in the $trans$ function; and b) x does not appear in the $post$ function.

REFERENCES

[1] L. de Moura and N. Bjørner, *Z3: An Efficient SMT Solver*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 337–340.

[2] A. Solar-Lezama, L. Tancau, R. Bodik, S. Seshia, and V. Saraswat, "Combinatorial sketching for finite programs," in *ASPLOS'06*. ACM, 2006, pp. 404–415.