GandALF: synthesis tools

Jan Otop

May 24, 2018

Plan for today

- Sketching programs (CEGIS)
- Synthesis for Concurrency (Liss)
- Learning string transformations from examples (Flash Fill)

SKETCH: Framework

- Idea: write an algorithm and let synthesizer work out the details.
- Sketch: a C-style language with holes ?? and derived constructs — incomplete programs.
- The synthesizer finds integer constants to fill in holes.
- Specification: assertions and complete functions.

Armando Solar-Lezama

`https://bitbucket.org/gatoatigrado/sketch-frontend/`

SKETCH: Examples

SKETCH: CEGIS loop

Counter-example guided inductive synthesis:
Return a solution $\sigma : holes \rightarrow \mathbb{Z}$.

Start with a random input in_0.

1. Construct the set of all solutions Φ consistent with $in_0, \ldots, in_{i-1}$.
2. Verify whether Φ is consistent and satisfies the spec.
3. If not, pick a violating input as in_i and go to 1.

SKETCH: SAT Encoding

- A solution $\sigma : holes \to \mathbb{Z}$ is a bit-vector.
- The set of all solutions Φ represented through arithmetic constraints (with comparisons and Boolean connectives).
- Constraints (algorithmically) closed under intersections, complements and unions.
- Constraints translated to a DAG and then to a CNF formula.

Concurrency Repair: Framework

- Write a program assuming non-preemptive scheduler.
- The synthesizer introduces locks to ensure correctness.

Authors: Pavol Černý, Edmund M. Clarke, Thomas A. Henzinger, Arjun Radhakrishna, Leonid Ryzhyk, Roopsha Samanta and Thorsten Tarrach
`https://github.com/thorstent/Liss`

Concurrency Repair: Example

```
void open dev() {
1: while (*) {
2: if (open==0) {
3: power up();
4: }
5: open=open+1;
6: yield; } }
```

```
void close dev() {
7: while (*) {
8: if (open>0) {
9: open=open-1;
10: if (open==0) {
11: power down();
12: } }
13: yield; } }
```

Concurrency Repair: Observational equivalence

- Threads can be scheduled in an arbitrary way.
- Observational equivalence: only operations on global variables matter.
- Abstraction: traces of events the form $\langle tid, op, v, l \rangle$.
- $\langle tid, op, v, l \rangle$ and $\langle tid, op, v, l \rangle$ can be swapped if $tid \neq tid'$ and $(v \neq v')$ or $op1, op_2 \neq write$.
- I pairs of swappable events.
- $\mathcal{L}_{pree}, \mathcal{L}_{non-pree}$ — language of all traces under non-preemptive scheduler (resp., preemptive).

Theorem (Observational safety)

A program is correct under a preemptive scheduler iff $\mathcal{L}_{pree}/I \subseteq \mathcal{L}_{non-pree}/I$.

Concurrency Repair: Example

```
void open dev abs() {
1: while (*) {
2: (A) r open;
if (*) {
3: (B) w dev;
4: }
5: (C) r open;
(D) w open;
6: yield; } }
```

```
void close dev abs() {
7: while (*) {
8: (E) r open;
if (*) {
9: (F) r open;
(G) w open;
10: (H) r open;
if (*) {
11: (I) w dev;
12: } }
13: yield; } }
```

Concurrency Repair: Checking inclusion modulo independence

Problem: $\mathcal{L}(\mathcal{M}_1)/I \subseteq \mathcal{L}(\mathcal{M}_2)/I$

Given $\mathcal{M}_1, \mathcal{M}_2$ over Σ and $I \subseteq \Sigma \times \Sigma$, decide whether for all $w \in \mathcal{L}(\mathcal{M}_1)$ there is $w'\mathcal{L}(\mathcal{M}_2)$ such that $w \sim_I w'$.

Concurrency Repair: Checking inclusion modulo independence

Problem: $\mathcal{L}(\mathcal{M}_1)/I \subseteq \mathcal{L}(\mathcal{M}_2)/I$

Given $\mathcal{M}_1, \mathcal{M}_2$ over Σ and $I \subseteq \Sigma \times \Sigma$, decide whether for all $w \in \mathcal{L}(\mathcal{M}_1)$ there is $w' \mathcal{L}(\mathcal{M}_2)$ such that $w \sim_I w'$.

- Problem: language inclusion modulo independence is undecidable.
- Solution: parametrized language inclusion (restrain independence).
- Make $\sim_I^k$ computable by an automaton.

String Processing: Framework

- A user presents examples of string transformations.
- The synthesizer tries to come up with a transformation.
- Specification by example.

Author: Sumit Gulwani

`https://www.microsoft.com/en-us/research/project/flash-fill-excel-feature-office-2013/`

String Processing: Examples

String Processing: Language

String expr P	$:=$	$\texttt{Switch}((b_1, e_1), \cdot\cdot, (b_n, e_n))$
Bool b	$:=$	$d_1 \vee \cdot\cdot \vee d_n$
Conjunct d	$:=$	$\pi_1 \wedge \cdot\cdot \wedge \pi_n$
Predicate π	$:=$	$\texttt{Match}(v_i, r, k) \mid \neg\, \texttt{Match}(v_i, r, k)$
Trace expr e	$:=$	$\texttt{Concatenate}(f_1, \cdot\cdot, f_n)$
Atomic expr f	$:=$	$\texttt{SubStr}(v_i, p_1, p_2)$
	$\mid$	$\texttt{ConstStr}(s)$
	$\mid$	$\texttt{Loop}(\lambda w : e)$
Position p	$:=$	$\texttt{CPos}(k) \mid \texttt{Pos}(r_1, r_2, c)$
Integer expr c	$:=$	$k \mid k_1 w + k_2$
Regular Expression r	$:=$	$\texttt{TokenSeq}(T_1, \cdot\cdot, T_m)$
Token T	$:=$	$C+ \quad \mid \quad [\neg C]+$
	$\mid$	SpecialToken

A table from “Automating String Processing in Spreadsheets Using Input-Output Examples” by Sumit Gulwani

String Processing: Synthesis algorithm

Input: A set of pairs $(\sigma_1, s_1), \ldots, (\sigma_k, s_k)$.

Output: A string program P consistent with the input.

Algorithm:

Step 1: For each i generate **all** trace expressions E_i that map σ_i to s_i.

Step 2: Greedy join find clusters of E_i's with non-empty intersection.

Step 3: Find Boolean classifier for each cluster.

The output expression is a Switch over all clusters.

String Processing: Synthesis of trace expressions

Given σ and s: return DAG representing all trace expressions that map σ to s.

1. The nodes of the DAG are positions of s.
2. 0 is the source and $|s|$ is the sink.
3. Each edge (i, j) is labeled by the set of atomic expressions generating $s[i, j]$.
4. Atomic expressions are substring, loop or a constant expression.
5. The number of (simple) atomic expressions is quadratic in σ.