KIDS: A Semi-Automatic Program Development System
Douglas R. Smith, Kestrel Institute, TSE '90

CS294-2: Software Synthesis
Presented by Gilad Arnold

Bunch of specification languages
- recursive functions
- SETL
- OOP + sets
- (else?)

Wealth of transformation techniques
- simplification
- (un)folding
- finite differencing
- (else?)

The missing link
- give me one tool
- all-in-one synthesizer/optimizer
- unified language for spec + transformations (deduction)
- IDE: expression highlighting, menus, …
- does (almost) everything automatically
- for the remainder…
- exhaustive library (theories)
- cookbook (design tactics)
The Basics

13

Refine: underlying knowledge-based environment
- data management / representation
- also defines high-level language: sets, sequences, FOL...

Rainbow II: deductive inference engine

Find some(T)(A \implies (S(x_1, \ldots, x_m) \rightarrow T(x_1, \ldots, x_n)))

Directed: \iff \geq \leq

Extensive rewrite rules library

"Optimality" = semantic distance + complexity heuristic

Specification

\begin{align*}
F(x : D) : & \text{ set}(R) \\
\text{where } I(x) & \rightarrow \text{ interface} \\
\text{returns } & \text{ implementation}
\end{align*}

Real-world example: k-queens

Input: \(k \in \mathbb{N}\)

Output: set of sequences \([3, 1, 4, 2] = [Q, Q, Q, Q]\)

How to express constraints on valid outputs?

1. Unique columns (trivial)
2. Unique rows expressed using bijection
   \begin{align*}
   \text{injective}(M : \text{seq}(\text{integer}), S : \text{set}(\text{integer})) : & \text{ boolean} \\
   = & \text{range}(M) \subseteq S \land \forall i \neq j \in \text{domain}(M), M(i) \neq M(j) \\
   \text{bijective}(M : \text{seq}(\text{integer}), S : \text{set}(\text{integer})) : & \text{ boolean} \\
   = & \text{injective}(M, S) \land \text{range}(M) = S
   \end{align*}
3. Unique diagonals expressed using diffs/sums
   \begin{align*}
   \text{ntqpud}(S : \text{set}(\text{integer})) : & \text{ boolean} \\
   = & \forall i \neq j \in \text{domain}(S), S(i) - i \neq S(j) - j
   \end{align*}

Enrich the theory

Distributive laws are good practice

\begin{align*}
\forall W_1, W_2, S. \text{injective}\left(\text{concat}(W_1, W_2), S\right) \\
= & \text{injective}(W_1, S) \land \text{injective}(W_2, S) \land \text{range}(W_1) \cap \text{range}(W_2) = \emptyset
\end{align*}

Formal specification

\begin{align*}
\text{function } \text{Queens}(k : \text{integer}) : & \text{ set}(\text{seq}(\text{integer})) \\
\text{where } & 1 \leq k \\
\text{returns } & \{ \text{assign } (\text{bijective}\text{assign}, \{1, 2, \ldots, k\}) \\
& \land \text{ntqpud(assign)} \\
& \land \text{ntqpdd(assign)} \}
\end{align*}

Step 1: Describe the problem

Step 2: Construct an algorithm

What’s missing...
Identify your problem

1. what kind of algorithm can be used?
   - our case: global search
   - partition (abstract) search space
   - solution extracted/collected bottom-up
   - pruning (necessary filter)
   - theorem: consistent specification of global search algorithm can be obtained from its theory

2. what form does solution take?
   - (aka: type of output)
   - our case: integer sequences of bounded length

How is that helpful?

Our case: \( m \)-bounded sequences \( \leq k \)-queens

\[
\begin{array}{c}
\vdots \\
[2, 1] \\
[2, 2] \\
\vdots \\
[1] \\
\vdots \\
\end{array}
\]

- given \( (S, m) \), enumerate all sequences of length \( \leq m \) over set \( S \)
- to verify:
  - \( \text{seq}(\text{integer}) = \text{seq}(n) \land \\
  \forall k : \text{integer}, \exists S : \text{set}(\text{integer}), m : \text{integer} : \text{assign} : \text{seq}(\text{integer}), \\
  \text{bijective}(\text{assign}, \{1 \ldots k\}) \land \text{ntqgd}(\text{assign}) \land \text{ntqgd}(\text{assign}) \\
  \implies \text{range}(\text{assign}) \subseteq S \land \text{length}(\text{assign}) \leq m \\
\]
- yields substitution \( \{n \mapsto \text{integer}, S \mapsto \{1 \ldots k\}, m \mapsto k\} \)

Our case: \( k \)-bounded sequences \( \leq k \)-queens

\[
\begin{array}{c}
\vdots \\
[2, 1] \\
[2, 2] \\
\vdots \\
[1] \\
\vdots \\
\end{array}
\]

- obtained a global search theory for our problem!
- notice a deficiency?
- infer necessary condition:
  - find some \( \Phi \)

\[
1 \leq k \implies ((\text{assign} \forall \text{assign} = \text{concat}(\text{prefix}, r) \\
\land \text{bijective}(\text{assign}, \{1 \ldots k\}) \\
\land \text{ntqgd}(\text{assign}) \land \text{ntqgd}(\text{assign})) \\
\implies \Phi(k, \text{prefix}))
\]

Our case: \( k \)-bounded sequences \( \leq k \)-queens (3)

\[
\begin{array}{c}
\vdots \\
[2, 1] \\
[2, 2] \\
\vdots \\
[1] \\
\vdots \\
\end{array}
\]

- obtained filter predicate:
  \( \Phi(k, \text{prefix}) = \text{ntqgd}(\text{prefix}) \land \text{ntqgd}(\text{prefix}) \\
\land \text{injective}(\text{prefix}, \{1 \ldots k\}) \\
\]
- prunes away infeasible prefixes
- ... but how to find strongest precondition?

Key idea: solution by reduction

1. select theory which solves an enumeration of the output type
   - many provided by library
2. find substitution which completely reduces your problem to it
   - verify: \( \forall x : D, \exists y : D : R(x, y) \implies O(x, y) \)
   - instantiate a deductive inference task
   - theorem: global search theory for problem \( A \) can be obtained from that of \( B \), given \( B \subseteq A \)
3. derive necessary filter + create global search algorithm
   - find parameterized necessary condition:
     \( \exists z : R, \text{Satisfies}(x, r) \land O(x, z) \implies \Phi(x, r) \)
   - again, another inference task
   - correctness guaranteed by theorem (previous slide)

What have we got so far?

- domain (global search) theory for our problem
- correct algorithm to solve it!
  - w/ some heuristic to prune unnecessary branching
- very high-level, evidently unknowledgeable
- therefore quite inefficient...
STEP 3: IMPROVE THE ALGORITHM

Ci-simplification: extreme rewriting

- Distributivity:

\[
\begin{align*}
\text{injective}([], [1 \ldots k]) \\
\land \text{ntqpd}([]) \land \text{ntqdd}([]) \\
\text{then } \text{Queens}_{gs}(k, []) \\
\text{else } \}
\end{align*}
\]

- And others:

\[
\begin{align*}
\{ \text{assign } | \text{ntqdd}(\text{assign}) \\
\land \text{ntqpd}(\text{assign}) \\
\land \text{injective}(\text{assign}, [1 \ldots k]) \\
\land \text{assign } = \text{prex} \}
\end{align*}
\]

Some distributive rules introduce cross-dependencies

CD-simplification: extreme rewriting with contexts

- Idea: assume all preceding predicates, then simplify

- Formed as an inference task:

\[
\begin{align*}
\text{find some (simp) } \\
\text{(ntqpd}(\text{prefix}) \\
\land \text{injective}(\text{prefix}, [1 \ldots k]) \\
\land \text{length}(\text{prefix}) \leq k \\
\land \text{range}(\text{prefix}) \subseteq [1 \ldots k] \\
\land \lambda i \leq k \\
\implies (\text{ntqpd}(\text{prefix}) \land \text{ntqpd}(\text{prefix}) \land \text{bijective}(\text{prefix}, [1 \ldots k]))
\end{align*}
\]

- Input assumptions

- Resulting expression: \([1 \ldots k] \subseteq \text{range}(\text{prefix})\)

Partial evaluation: unfolding with substitution

- More opportunities for simplification

- Otherwise nothing interesting

Finite differencing: incrementalize repeated computations

- Same as last week’s paper

- Just slightly different

- Phased approach: first abstraction, then simplification!

- Works across functions

- Reuses common pool of laws

Case analysis: even further simplification

- Idea: replace if with \( \text{if } P \text{ then } e \text{ else } e \), then CD-simplify

- Useful (eg) when joining complementing sets

STEP 4: CONCRETIZE

Code after simplification: as clear as can be

Function Queens(k)

Where...

Returns...

= Queens_{gs}(k, [])

Function Queens_{gs}(k, prefix)

Where...

Returns...

= \{(prefix | \{1 \ldots k\} \subseteq \text{range}(prefix)) \\
\| \bigcup \{ \text{Queens}_{gs}(k, \text{append}(prefix, i)) | \\
i \not\in \text{range}(prefix) \land i \in \{1 \ldots k\} \\
\land \text{length}(prefix) < k \\
\land \text{cross_ntqpd}(prefix, [i]) \\
\land \text{cross_ntqdd}(prefix, [i]) \}

Step 4: concretize
Data type refinement

➜ no single “standard” implementation covers all use cases
➜ idea: extract partial schedule, then find the right data structure
➜ some caveats
  ➜ may require deriving upper/lower bounds on set cardinality
  ➜ may require restricting the spec with fixed bounds

Compilation

➜ via Common Lisp

Aftermath

➜ transformations cut processing time by cubic factor
➜ actual benchmark drops from 1 hours to 1 second
➜ takes 16 high-level informed decisions
➜ conjecture: could be reduced to zero
➜ possibly applying CI-simplify exhaustively?

Discussion

➜ adding distributive (and other) laws to high-level theories
➜ how critical are they in practice?
➜ how applicable are reductions to arbitrary problem types?
➜ are they really easier than straightforward coding?
➜ (your question here)

Thank you!