KIDS: A Semi-Automatic Program Development System

Douglas R. Smith, Kestrel Institute, TSE '90

CS294-2: Software Synthesis
Presented by Gilad Arnold

Been there, did that...

Bunch of specification languages

- recursive functions
- SETL
- OO + sets
- (*else?*)

declarative (?), expressive, ...

Wealth of transformation techniques

- simplification
- (un)folding
- finite differencing
- (*else?*)

correctness preserving, useful, complementary, ...

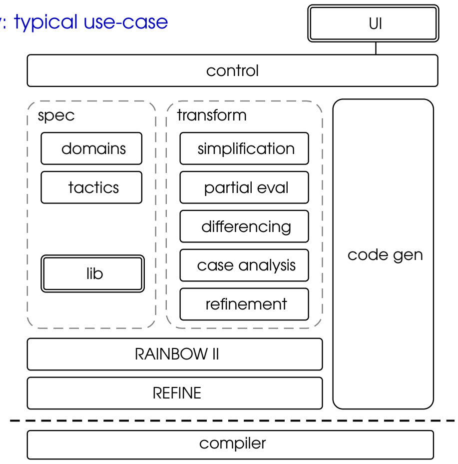
So what's the problem?

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 automagically
- for the remainder...
  - exhaustive library (theories)
  - cookbook (design tactics)

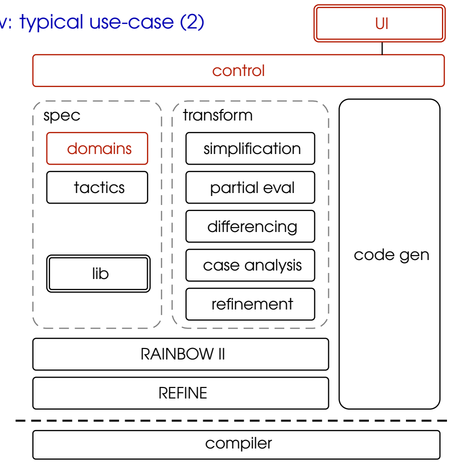
KIDS: software synthesis applied

Functional view: typical use-case



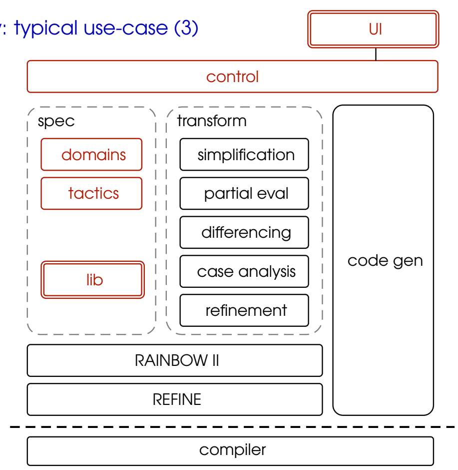
Functional view: typical use-case (2)

① spec



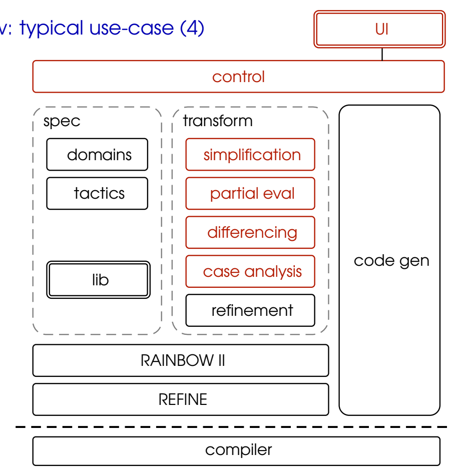
Functional view: typical use-case (3)

① spec
② design



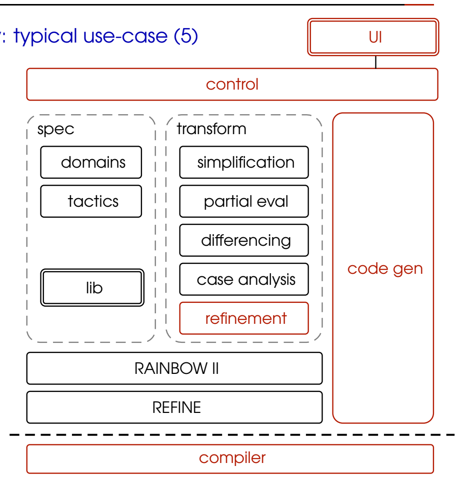
Functional view: typical use-case (4)

① spec
② design
③ transform

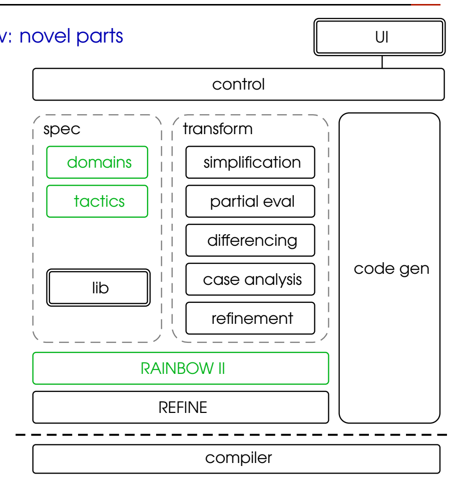


Functional view: typical use-case (5)

① spec
② design
③ transform
④ concretize



Functional view: novel parts



THE BASICS

REFINE: underlying knowledge-based environment

- ➜ data management / representation
- ➜ also defines high-level language: sets, seuqnces, FOL, . . .

RAINBOW II: deductive inference engine

$$\textbf{find } \mathit{some}(T)(A \implies (S(x_1,\ldots,x_m) \longrightarrow T(x_{i_1},\ldots,x_{i_n})))$$

- ➜ directed: $\implies \quad \impliedby \quad = \quad \geq \quad \leq$
- ➜ extensive rewrite rules library
- ➜ "optimality" = semantic distance + complexity heuristic

Specification

$$\left.\begin{array}{l}\textbf{function } F(x:D):\mathit{set}(R)\\ \quad\textbf{where } I(x)\\ \quad\textbf{returns } \{z \mid O(x,z)\}\end{array}\right\}\text{ interface}$$
$$\left.\quad = \mathit{body}\right\}\text{ implementation}$$

STEP 1: DESCRIBE THE PROBLEM

Real-world example: k-queens

- ➜ input: $k \in \mathbb{N}$
- ➜ output: set of sequences $[3,1,4,2] = \begin{array}{|c|c|c|c|}\hline & & Q & \\ \hline Q & & & \\ \hline & & & Q \\ \hline & Q & & \\ \hline\end{array}$
- ➜ how to express constraints on valid outputs?
  - ① unique columns (trivial)
  - ② unique rows expressed using bijection

    $\mathit{injective}(M:\mathit{seq}(\mathit{integer}), S:\mathit{set}(\mathit{integer})):\mathit{boolean}$
    $\quad = \mathit{range}(M) \subseteq S \wedge \forall i \neq j \in \mathit{domain}(M).M(i) \neq M(j))$

    $\mathit{bijective}(M:\mathit{seq}(\mathit{integer}), S:\mathit{set}(\mathit{integer})):\mathit{boolean}$
    $\quad = \mathit{injective}(M,S) \wedge \mathit{range}(M) = S$
  - ③ unique diagonals expressed using diffs/sums

    $\mathit{ntqpud}(S:\mathit{set}(\mathit{integer})):\mathit{boolean}$
    $\quad = \forall i \neq j \in \mathit{domain}(S).S(i) - i \neq S(j) - j$

Enrich the theory

- ➜ distributive laws are good practice

$$\forall W_1, W_2, S.\mathit{injective}(\mathit{concat}(W_1,W_2),S)$$
$$= \mathit{injective}(W_1,S) \wedge \mathit{injective}(W_2,S) \wedge \mathit{range}(W_1) \cap \mathit{range}(W_2) = \emptyset$$

Formal specification

$$\begin{array}{l}\textbf{function } \mathit{Queens}(k:\mathit{integer}):\mathit{set}(\mathit{seq}(\mathit{integer}))\\ \quad\textbf{where } 1 \leq k\\ \quad\textbf{returns } \{\mathit{assign} \mid \mathit{bijective}(\mathit{assign},\{1\ldots k\})\\ \qquad\qquad\qquad \wedge \mathit{ntqpud}(\mathit{assign})\\ \qquad\qquad\qquad \wedge \mathit{ntqpdd}(\mathit{assign})\}\end{array}$$

What's missing?. . .

STEP 2: CONSTRUCT AN ALGORITHM

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?

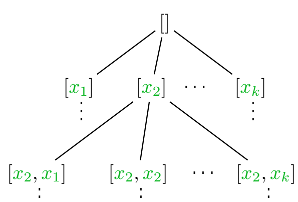
Key idea: solution by reduction

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: $\forall x : D_A . \exists y : D_B . \forall z : R_A . I_A(x) \wedge O_A(x,z) \implies O_B(y,z)$
  - ➜ instantiate a deductive inference task
- ➜ **theorem:** *global search theory for problem A can be obtained from that of B, given $\mathcal{B}_B \leq_\theta \mathcal{B}_A$*

③ derive necessary filter + create global search algorithm
- ➜ find parameterized necessary condition:
  $\exists z : R.Satisfies(z, \hat{r}) \wedge O(x,z) \implies \Phi(x, \hat{r})$
  - ➜ again, another inference task
- ➜ correctness guaranteed by theorem (previous slide)

Our case: m-bounded sequences $\leq$ k-queens

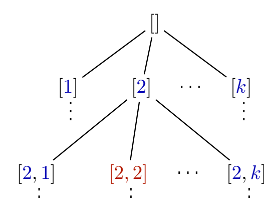


- ➜ given $\langle S, m \rangle$, enumerate all sequences of length $\leq m$ over set S
- ➜ to verify:

$$\begin{aligned}
&seq(integer) = seq(\alpha) \wedge \\
&\forall k : integer . \exists S : set(integer), m : integer . \forall assign : seq(integer). \\
&\quad bijective(assign, \{1 \ldots k\}) \wedge ntqpud(assign) \wedge ntqpdd(assign) \\
&\quad \implies range(assign) \subseteq S \wedge length(assign) \leq m
\end{aligned}$$

- ➜ yields substitution $\{\alpha \mapsto integer, S \mapsto \{1 \ldots k\}, m \mapsto k\}$

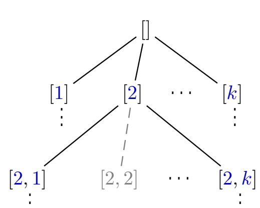
Our case: k-bounded sequences $\leq$ k-queens (2)



- ➜ obtained a global search theory for our problem!
- ➜ notice a deficiency?
- ➜ infer necessary condition:

$$\begin{aligned}
&\textbf{find } some(\Phi) \\
&\quad 1 \leq k \implies ((\exists assign . \exists r . assign = concat(prefix, r) \\
&\qquad\qquad\qquad \wedge bijective(assign, \{1 \ldots k\}) \\
&\qquad\qquad\qquad \wedge ntqpud(assign) \wedge ntqpdd(assign)) \\
&\qquad\qquad \implies \Phi(k, prefix))
\end{aligned}$$

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



- ➜ obtained filter predicate:

$$\begin{aligned}
\Phi(k, prefix) = {}& ntqpud(prefix) \wedge ntqpdd(prefix) \\
&\wedge injective(prefix, \{1 \ldots k\})
\end{aligned}$$

- ➜ prunes away infeasible prefixes
- ➜ . . . but how to find strongest postcondition?

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:

$$\left.\begin{array}{l}\textbf{if } \mathit{injective}([], \{1 \ldots k\}) \\ \quad \wedge \mathit{ntqpud}([]) \wedge \mathit{ntqpdd}([]) \\ \quad \textbf{then } \mathit{Queens\_gs}(k, []) \\ \quad \textbf{else } \emptyset \end{array}\right\} = \mathit{Queens\_gs}(k, [])$$

- and others:

$$\left.\begin{array}{l}\{\mathit{assign} \mid \\ \quad \mathit{ntqpdd}(\mathit{assign}) \\ \quad \wedge \mathit{ntqpud}(\mathit{assign}) \\ \quad \wedge \mathit{bijective}(\mathit{assign}, \{1 \ldots k\}) \\ \quad \wedge \mathit{assign} = \mathit{prefix}\}\end{array}\right\} = \left\{\begin{array}{l}\{\mathit{prefix} \mid \\ \quad \mathit{ntqpdd}(\mathit{prefix}) \\ \quad \wedge \mathit{ntqpud}(\mathit{prefix}) \\ \quad \wedge \mathit{bijective}(\mathit{prefix}, \{1 \ldots k\})\}\end{array}\right.$$

- some distributive rules introduce cross-dependencies

CD-simplification: extreme rewriting with contexts

- idea: assume all preceding predicates, then simplify
- formed as an inference task:

$$\begin{array}{l}\textbf{find } \mathit{some}(\mathit{simp}) \\ \quad \left.\begin{array}{l}(\mathit{ntqpdd}(\mathit{prefix}) \\ \quad \wedge \mathit{ntqpud}(\mathit{prefix}) \\ \quad \wedge \mathit{injective}(\mathit{prefix}, \{1 \ldots k\}) \\ \quad \wedge \mathit{length}(\mathit{prefix}) \leq k \\ \quad \wedge \mathit{range}(\mathit{prefix}) \subseteq \{1 \ldots k\} \\ \quad \wedge 1 \leq k\end{array}\right\} \text{input assumptions} \\ \quad\quad \Longrightarrow \; (\mathit{ntqpdd}(\mathit{prefix}) \wedge \mathit{ntqpud}(\mathit{prefix}) \wedge \mathit{bijective}(\mathit{prefix}, \{1 \ldots k\}))) \\ \quad = \mathit{simp}(\mathit{prefix}, k)\end{array}$$

- resulting expression: $\{1 \ldots k\} \subseteq \mathit{range}(\mathit{prefix})$

Code after simplification: as clear as can be

$$\begin{array}{l}\textbf{function } \mathit{Queens}(k) \\ \quad \textbf{where} \ldots \\ \quad \textbf{returns} \ldots \\ \quad = \mathit{Queens\_gs}(k, []) \\ \textbf{function } \mathit{Queens\_gs}(k, \mathit{prefix}) \\ \quad \textbf{where} \ldots \\ \quad \textbf{returns} \ldots \\ \quad = \{\mathit{prefix} \mid \{1 \ldots k\} \subseteq \mathit{range}(\mathit{prefix})\} \\ \quad\quad \cup \bigcup \{\mathit{Queens\_gs}(k, \mathit{append}(\mathit{prefix}, i)) \mid \\ \quad\quad\quad i \notin \mathit{range}(\mathit{prefix}) \wedge i \in \{1 \ldots k\} \\ \quad\quad\quad \wedge \mathit{length}(\mathit{prefix}) < k \\ \quad\quad\quad \wedge \mathit{cross\_ntqpud}(\mathit{prefix}, [i]) \\ \quad\quad\quad \wedge \mathit{cross\_ntqpdd}(\mathit{prefix}, [i])\}\end{array}$$

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 e with **if** P **then** e **else** e, then CD-simplify
- useful (eg) when joining complementing sets

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?

SHOWTIME

be back with us after this short demo. . .

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!