# Deductive Synthesis

## Programs vs. Proofs

## Closing Arguments

---

### A bullet we’ve missed

- **Constraint-based synthesis**
  - How to solve constraints about infinitely many inputs? CEGIS
  - How to encode semantics of looping / recursive programs?
  - Bounded reasoning
  - Unbounded / deductive reasoning

Enumerative (and deductive) synthesis
  - How to use deductive reasoning to guide the search?

---

### Verification

```daffny
method SumMax (a: array<int>) returns (sum: int, max: int)
{requires a != null;
  ensures sum <= a.Length * max;
  {sum, max := 0, 0;
   var i := 0;
   while i < a.Length
     {if max < a[i] { max := a[i]; }
      sum := sum + a[i];
      i := i + 1;
     }
  }
}
```

- Dafny
- AutoProof
- VCC
- Verifast

---

### Invariant inference

```daffny
method SumMax (a: array<int>) returns (sum: int, max: int)
{requires a != null;
  ensures sum <= a.Length * max;
  {sum, max := 0, 0;
   var i := 0;
   while i < a.Length
     {invariant ??
      decreases ??;
      {if max < a[i] { max := a[i]; }
       sum := sum + a[i];
       i := i + 1;
     }
   }
}
```

- BLAST
- Astrée
- FB Infer
- LiquidHaskell

---
Program synthesis

```
method maxes (a: array<int>) returns (max: int, sum: int)

// Updates a <= max.
// Precondition: a.length <= max;
{i = 0; sum = 0;}
while (i < a.length)
  {assert i <= a.length};
  sum := sum + a[i];
  {i++; sum <= i * max;}
assert i < a.length;
{i = i + 1; sum <= i * max;}
returns (max, sum);}
```

Program logic

```

from Verification to Synthesis

Program logic

∀ x. Q(x)

SAT

UNSAT

Inference

Program logic

∃ i. Q(i, x) unknown formulas for invariants

Synthesis

Program logic

∀ i. Q(\mathcal{P}, x) unknown formulas for invariants and commands

on the bright side: not much harder than inference!

Step 1: eliminate loops

Viewed as a Transition System

```
{i = 0; max := 0; l := 0; while (i < a.length)
  {assert i <= a.length as sum <= i * max};
  sum := sum + a[i];
  {i++; sum <= i * max;}
assert i < a.length;
{i = i + 1; sum <= i * max;}
returns (max, sum);}
```

```
{i = 0; max := 0; l := 0; while (i < a.length)
  {assert i <= a.length as sum <= i * max};
  sum := sum + a[i];
  {i++; sum <= i * max;}
assert i < a.length;
{i = i + 1; sum <= i * max;}
returns (max, sum);}
```

Viewed as a Transition System

```
{i = 0; max := 0; l := 0; while (i < a.length)
  {assert i <= a.length as sum <= i * max};
  sum := sum + a[i];
  {i++; sum <= i * max;}
assert i < a.length;
{i = i + 1; sum <= i * max;}
returns (max, sum);}
```

```
{i = 0; max := 0; l := 0; while (i < a.length)
  {assert i <= a.length as sum <= i * max};
  sum := sum + a[i];
  {i++; sum <= i * max;}
assert i < a.length;
{i = i + 1; sum <= i * max;}
returns (max, sum);}
```
VC for Transition Systems

\{p \}
\begin{align*}
\text{init} & \rightarrow \{p\} \\
\text{while } (p) & \rightarrow \{p\} \\
\text{invariant } inv & \rightarrow \{p\} \\
\text{final} & \rightarrow \{p\} \\
\text{post} & \rightarrow \{p\}
\end{align*}

\{init\} \rightarrow \{p\}
\{\text{while } (p)\} \rightarrow \{p\}
\{\text{invariant } inv\} \rightarrow \{p\}
\{\text{final}\} \rightarrow \{p\}
\{\text{post}\} \rightarrow \{p\}

From verification to inference

Inference

\forall y. \forall x. Q(y, x)

Invariant Inference

\{p\} \rightarrow \{p\}
\{\text{init}\} \rightarrow \{p\}
\{\text{while } (p)\} \rightarrow \{p\}
\{\text{invariant } inv\} \rightarrow \{p\}
\{\text{final}\} \rightarrow \{p\}
\{\text{post}\} \rightarrow \{p\}

\forall y. \forall x. Q(y, x)

Horn Constraints

Constraints of this form are called Horn constraints (clauses)
\begin{align*}
\phi & \Rightarrow \Gamma \\
\Gamma \land \psi & \Rightarrow \Gamma \\
\Gamma & \Rightarrow \varnothing
\end{align*}

How do we find \(\Gamma\) ?
- Fix a domain (search space)
- Search for an element of the domain that makes all clauses true
Horn constraints for SumMax

\[ s^I = 0 \land w^I = 0 \land i^I = 0 \Rightarrow \top \]
\[ I \land i < \text{len}(a) \land s^I = \sum(a[i]) \land w^I = \sum(a[i]) \land i^I = i + 1 \Rightarrow I' \]
\[ I \land i < \text{len}(a) \land i^I = i + 1 \Rightarrow I' \]
\[ I \Rightarrow i = \text{len}(a) \lor i < \text{len}(a) \]

(Solution: \( i \leq \text{len}(a) \land i \geq 1 \))

Can we solve this with...
- Enumerative search?
- Sketch?

\[ \exists I. \forall x. Q(I, x) \]
\[ \exists I. \forall x. Q(I', x) \]

Idea: Lattice Search

Domain = all conjunctions of predicates from
\( (p, q, r) = (i \leq \text{len}(a), i < \text{len}(a), i \neq \text{len}(a)) \)

\[ i^I = 0 \land \text{len}(a) > 0 \Rightarrow \top \]
\[ I \land i < \text{len}(a) \land i^I = i + 1 \Rightarrow I' \]
\[ I \Rightarrow i = \text{len}(a) \lor i < \text{len}(a) \]

Example

Domain = all conjunctions of predicates from
\( (p, q, r) = (i \leq \text{len}(a), i < \text{len}(a), i \neq \text{len}(a)) \)

\[ i^I = 0 \land \text{len}(a) > 0 \Rightarrow \top \]
\[ I \land i < \text{len}(a) \land i^I = i + 1 \Rightarrow I' \]
\[ I \Rightarrow i = \text{len}(a) \lor i < \text{len}(a) \]

Example

Domain = all conjunctions of predicates from
\( (p, q, r) = (i \leq \text{len}(a), i < \text{len}(a), i \neq \text{len}(a)) \)

\[ i^I = 0 \land \text{len}(a) > 0 \Rightarrow \top \]
\[ I \land i < \text{len}(a) \land i^I = i + 1 \Rightarrow I' \]
\[ I \Rightarrow i = \text{len}(a) \lor i < \text{len}(a) \]

Example

Domain = all conjunctions of predicates from
\( (p, q, r) = (i \leq \text{len}(a), i < \text{len}(a), i \neq \text{len}(a)) \)

\[ i^I = 0 \land \text{len}(a) > 0 \Rightarrow \top \]
\[ I \land i < \text{len}(a) \land i^I = i + 1 \Rightarrow I' \]
\[ I \Rightarrow i = \text{len}(a) \lor i < \text{len}(a) \]

Example

Domain = all conjunctions of predicates from
\( (p, q, r) = (i \leq \text{len}(a), i < \text{len}(a), i \neq \text{len}(a)) \)

\[ i^I = 0 \land \text{len}(a) > 0 \Rightarrow \top \]
\[ I \land i < \text{len}(a) \land i^I = i + 1 \Rightarrow I' \]
\[ I \Rightarrow i = \text{len}(a) \lor i < \text{len}(a) \]

Example

Domain = all conjunctions of predicates from
\( (p, q, r) = (i \leq \text{len}(a), i < \text{len}(a), i \neq \text{len}(a)) \)

\[ i^I = 0 \land \text{len}(a) > 0 \Rightarrow \top \]
\[ I \land i < \text{len}(a) \land i^I = i + 1 \Rightarrow I' \]
\[ I \Rightarrow i = \text{len}(a) \lor i < \text{len}(a) \]
Least Fixpoint (forward search)

Domain = all conjunctions of predicates from 
\( \{p, q, r\} = \{i \leq \text{len}(a), i \geq \text{len}(a), i \neq \text{len}(a)\} \)

- Find the strongest solution
- Did not have to look at all candidates
- Relies on efficient weakening operation

Greatest Fixpoint (backward search)

- Finds the weakest solution
- Relies on efficient strengthening operation
- Hard to implement

From verification to inference

\[ \forall x. Q(x) \]

From inference to synthesis

\[ \exists I. \forall x. Q(I, x) \]
Synthesizing a Loop

\[
\begin{align*}
\text{(pre)} & & (\text{post}) \\
\text{while} & (\text{?}) & \text{invariant} & (\text{?}) & \{ & (\text{?}) \\
\{ & (\text{?}) & \}
\end{align*}
\]

\[
\begin{align*}
\{p_r e\} & & \{p_o s t\} \\
\exists S_i, G_i, I. & & \forall x. & & Q(P, I, x)
\end{align*}
\]

Synthesis constraints

Similar to Horn constraints but not quite

\[
\begin{align*}
I & \land G_i & \land S_i & \land \psi & \Rightarrow I' \\
I & \land G_i & \land S_i & \Rightarrow \omega \\
T & \Rightarrow G_i \lor G_j \\
\end{align*}
\]

Domain for \(G\): like in inference
Domain for \(S\):
  * conjunction of equalities, one per variable

Solving synthesis constraints

Can we solve this with...
  * Enumerative search?
  * Sure [slow]
  * Sketch?
    * Yee!
    * Look we made an unbounded synthesizer out of Sketch!
  * Lattice search?
    * Great for \(G\), not so great for \(S\) (why?)

The takeaway

We can reason about unbounded loops using loop invariants
  * Hoare logic soundly translates a program with a loop (and invariant) into three straight-line programs
We can synthesize a program with a loop by synthesizing those three straight-line programs (and the invariant)!
  * Can use existing synthesis techniques
Powerful idea: to synthesize a provably correct program, look for the program and its proof together

Deductive Synthesis

for real
Deductive reasoning for synthesis

Main idea: Look for the proof to find the program
- The space of valid program derivations is smaller than the space of all programs
- The result is provably correct!

Applications:
- Constraint-based search: use loop invariants to encode the space of correct looping programs
- Enumerative search: prune unverifiable candidates early
- Deductive search: search in the space of provably correct implementations

Deductive Synthesis: Principle

Problem definition:
- Find \( x \) such that \( Q(x,a) \) whenever \( P(a) \)

Using semantic-preserving transformations, gradually refine the problem above into:
- Find \( T \) such that \( \top \) whenever \( P(a) \)
  - where \( T \) is a term that does not mention \( x \)

Find \( x \) such that \( x + x = 4a \)

Find \( x \) such that \( 2x = 4a \)

Find \( 2y \) such that \( 4y = 4a \)

Find \( 2y \) such that \( y = a \)

Find \( 2a \) such that \( \top \)

Deductive synthesis: challenges

Define a set of transformation rules that is sound
- A solution to the transformed problem is a solution to the original problem
  - and “complete”
    - All programs we care about can be derived

In most situations, multiple rules apply to a problem
- Need a search strategy!

Synthesis as Theorem Proving

Idea: extract the program from a constructive proof of

\[
\forall a. \exists x. P(a) \rightarrow Q(a,x)
\]

- There is no need to invent any new reasoning or inference: reuse an existing theorem prover / proof assistant
- ...but record the proof steps and augment them with program extraction rules.
- Reuse any automation that exists in the prover!

Synthesis as Theorem Proving

Intuition (by example).

Axioms
1. \( \text{head}(\text{::} x) = x \)
2. \( \text{tail}(\text{::} x) = xx \)

Prove \( \exists l. \text{head}(l) = 5 \land \text{tail}(l) = [] \)
Synthesis as Theorem Proving

Sequent:

<table>
<thead>
<tr>
<th>assertions</th>
<th>goals</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_i(a, x)$</td>
<td>$G_i(a, x)$</td>
<td>$t_i(a, x)$</td>
</tr>
</tbody>
</table>

Meaning: if $\forall x \ A_i(a, x)$ holds, then $t_i$ holds
- and the corresponding $t_i$ is an acceptable solution

---

Synthesis as Theorem Proving

Synthesis problem: “Find $x$ such that $P$ whenever $Q$”

<table>
<thead>
<tr>
<th>assertions</th>
<th>goals</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P(a)$</td>
<td>$Q(a, x)$</td>
<td>$x$</td>
</tr>
<tr>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
</tr>
</tbody>
</table>

Apply inference rules to add new assertions and goals
- $T$ output
- (where $t$ does not contain $x$)

---

Inference Rules

- Splitting
  - E.g. split assertion $A_1 \land A_2$ into two assertions $A_1$ and $A_2$
- Transformation
  - Apply a rewrite rule $s \rightarrow t$ to a subterm of assertion / goal
  - Apply the unifying substitution of the rewrite to the output!
- Resolution
  - Let $A[P], B[P]$ two assertions (both with subformula $P$); add assertion $A[\top] \lor B[\bot]$
- Induction
  - Introduce an induction hypothesis

---

Example: Quotient and Remainder

Specification:

$\text{div}(i, j), \text{rem}(i, j) \in \text{find } (q, r) \text{ s.t. }$

\[
i = q \cdot j + r \land 0 \leq r < j
\]

where $0 \leq i \land 0 < j$

---

Example: base case

<table>
<thead>
<tr>
<th>assertions</th>
<th>goals</th>
<th>outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1. \ 0 \leq i \land 0 &lt; j$</td>
<td>$2. \ i = q \cdot j + r \land 0 \leq r &lt; j$</td>
<td>$x$</td>
</tr>
<tr>
<td>$3. \ i = r$</td>
<td>$4. \ 0 \leq r &lt; j$</td>
<td>$r$</td>
</tr>
</tbody>
</table>

- and-split 1
- lemma 2
- lemma 3
- trans 5
- reflex 8 & 9
- resolve 7 & 3
Example: step case

<table>
<thead>
<tr>
<th>assertions</th>
<th>goals</th>
<th>outputs</th>
<th>dis(𝑖, 𝑗)</th>
<th>rem(𝑖, 𝑗)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 ≤ 𝑖</td>
<td>4. (0 &lt; 𝑗)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. 𝑖 = 𝑖 + 𝑗 + 1 A 0 ≤ 𝑖 + 𝑗</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6. 𝑖 = 𝑖 + 𝑗 + 1 A 0 ≤ 𝑖 + 𝑗</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example: put them together

<table>
<thead>
<tr>
<th>assertions</th>
<th>goals</th>
<th>outputs</th>
<th>dis(𝑖, 𝑗)</th>
<th>rem(𝑖, 𝑗)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8. 1 &lt; 𝑗</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13. ¬(𝑖 &lt; 𝑗)</td>
<td></td>
<td>T</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

If 𝑖 < 𝑗 then if else 𝑑𝑖𝑣(𝑖 − 𝑗, 𝑗) + 1

If 𝑖 < 𝑗 then 𝑖 else rem(𝑖 − 𝑗, 𝑗)

Almost Done

Synthesis!

- Automatic programming?
  - but I have to tell the computer what I want...

Synthesis = an unusually concise / intuitive programming language + a compiler that sometimes doesn’t work 😊

concise

Adjective
Giving a lot of information clearly and in a few words; brief but comprehensive.

Adverb: concisely, short, brief, to the point, pithy, concise, short and sweet, crisp

Origin:
Late 16th century; from French concise or Latin concisus, past participle of concinere ‘cut up, cut down’, from con- ‘completely’ + caedere ‘to cut’.

(Oxford Dictionary)
What went wrong?

Writing specifications is hard!
  - have to think about all the corner cases
  - have to encode hidden assumptions as formulas

Writing specifications for a synthesis tool is even harder!
  - it's the user's responsibility to define the search space
  - too much freedom — divergence (or just very long running times)
  - too restrictive — non-realizability (and that's hard to debug)
  - have to reduce to the logical fragment that the tool supports
What can we do??

- Improved specification language
  - we have seen a lot of repetitive mundane stuff
- Reusable specifications
  - declarative specs are highly composable
- Partial specifications
  - less to write, but heuristics are needed to choose the desired program
- Better algorithms...
  - faster search = less to worry about = easier spec work

Final Project

Logistics

Due: end of August.

- This is an arbitrary deadline. The project itself can be completed within within a few days.
- Still, if you have too many exams / are leaving the country / are joining the military, course staff will be merciful.
- Like homework assignments, projects will be done in pairs.

Project Requirements

**We will build a synthesizer.**

Don’t worry: it’s going to be a rather small synthesizer.

<table>
<thead>
<tr>
<th>Mode of specification</th>
<th>Output language</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRE</td>
<td>You choose!</td>
</tr>
<tr>
<td></td>
<td>• But — must include some kind of list operations</td>
</tr>
<tr>
<td></td>
<td>sorted(input)[-1] - sorted(input)[-2]</td>
</tr>
</tbody>
</table>

We will need Observational Equivalence.

Following this pseudo-code:

- At least one child from the most recent iteration
- Eliminate duplicate modules

\[
\text{true} = \langle \text{if} \rangle = (\text{true} \land \text{true}) \lor (\text{false} \land \text{false})
\]
That something extra

- Symbolic examples
- Synthesis of body of lambdas
- Topmost-level constraint solving
- User-defined sketches
- Condition abduction for corner cases