Software Synthesis and Automated Reasoning
Who

• Instructor
  ‣ Shachar Itzhaky  
    shachari@cs.technion.ac.il

• TA
  ‣ Dor Cohen  
    dorafifitcohen@gmail.com
Structure of Lectures

Traditional Classroom

Blah blah boring stuff

Flipped Classroom

Semi-Flipped Classroom

How did the pig’s paw-prints get on the ceiling?
Structure of Lectures

• 50 min lecture + 50 min class assignments

  ‣ Work in pairs (recommended)
  ‣ Have a laptop
  ‣ **Pre-install** required software
    ○ Either use supplied VM (easy) or install locally (harder)
  ‣ Not part of final grade
Structure of Grade

Homework Assignments
- O(4)
- Programming problems
- Small

Final Project
- Application of Sketch
  (with the help of TA)
  Medium

40%  60%
Reading List

• Textbooks
  ▶ “Formal Reasoning About Programs” / Chlipala
  ▶ “The Formal Semantics of Programming Languages — an introduction” / Winskel

• Papers
  ▶ (for specific topics during 2nd half of term)
Automated Formal Reasoning

• What is reasoning?

Every person has secrets.

Angelina Jolie is a person.

Angelina Jolie has secrets.
Automated Formal Reasoning

• What is **formal reasoning**?

Every student in my class gets an A.

What we need is a **formalism**.
Reasoning: Formalism

• Zermelo’s Theorem
  ‣ In Chess, either White can force a win, or Black can force a win, or both sides can force a draw.

Despite a growing interest in the history of game theory, (e.g., Aumann (1989a), Dimand and Dimand (1996, 1997), Kuhn (1997), Leonard (1995) and Weintraub (1992)), confusion, at least in the English language literature, as to the contribution made by Zermelo and some of the other early game theorists seems to prevail. This problem may be due in part to a language barrier. Many of the early papers in game theory were not written in English and have not been translated. For example, to the best of our knowledge, there is no English version of Zermelo (1913). The same holds for the lesser known but related work by König (1927)\textsuperscript{2}. A second paper related to that of Zermelo, Kalmár (1928/29)\textsuperscript{3}, has recently been translated (see Dimand and
Reasoning: First-order Logic

• In every (non-empty) bar, there is one person such that if he (she) drinks, then everybody drinks.

\[ \exists x. \, \text{drinks}(x) \rightarrow \forall y. \, \text{drinks}(y) \]

• In every (non-empty) set of natural numbers there is a minimal element.

\[ \exists n \in S. \, \forall m \in S. \, n \leq m \]
Reasoning: Second-order Logic

• Every real number is the supremum (least upper bound) of some set of rational numbers.

\[ \forall r \in \mathbb{R}. \exists S \subseteq \mathbb{Q}. \; r = \sup S \]

*Where*(definition of sup)

\[ \forall S. \; \forall x \in \mathbb{Q}. \; x \in S \to x \leq \sup S \]

\[ \land \]

\[ \forall y \in \mathbb{R}. \; (\forall x \in \mathbb{Q}. \; x \in S \to x \leq y) \to \sup S \leq y \]
Reasoning: Proofs

• What is a proof?

A proof is a description that we can read and be convinced, beyond any doubt, that a statement is correct.

A proof is a document whose consistency can be checked mechanically, (via an algorithm) to validate its conclusion.
Reasoning: Proofs

- Modus Ponens

\[ \vdash A \quad \vdash A \rightarrow B \]

\[ \vdash B \quad \text{(MP)} \]

- Proof calculi:
  - Hilbert, Natural Deduction, Sequent Calculus, ...

A proof is a mathematical object.
Automated Reasoning

Proof Checker
validates logical consequence
(small trusted core)

Proof Assistant
automates tedious mechanics
of writing proofs

Theorem Prover
proof search
exhaustive (heuristic)

Solver
constructs a model for a
formula, or declares none
Reasoning About Programs

- Euclid’s algorithm

```python
# a,b > 0
while a != b:
    if a < b:
        b = b - a
    else:
        a = a - b

output a = gcd(input a, input b)
```
Programming Language Semantics

• WHILE language
  
  ‣ Syntax:

  $$S \rightarrow x := E \mid S ; S \mid \text{skip} \mid \text{if } E \text{ then } S \text{ else } S \mid \text{while } E \text{ do } S$$

  $$E \rightarrow x \mid \# \mid E \diamond E$$

  $$\diamond \in \{+, -, *, /, =, \neq, <, >, \leq, \geq\}$$
Semantics: Expressions

• $\sigma : \text{Var} \rightarrow \mathbb{Z}$
  ‣ A state of the program
    (also called a store)

• $\Sigma$ — the set of all such states

• $\llbracket e \rrbracket : \Sigma \rightarrow \mathbb{Z}$
  ‣ $\llbracket e \rrbracket_\sigma$ is the value of $e$ when interpreted in
    state $\sigma$
Semantics: Expressions

Base cases

\[ \llbracket x \rrbracket \sigma = \sigma x \] \hspace{1cm} (x \in \text{Var})

\[ \llbracket n \rrbracket \sigma = n \] \hspace{1cm} (n \in \mathbb{Z})

Recursive case

\[ \llbracket e_1 \odot e_2 \rrbracket \sigma = \llbracket e_1 \rrbracket \sigma \odot \llbracket e_2 \rrbracket \sigma \]
Structural Operational Semantics

Small-step Semantics

• Define a new relation:

\[ \sigma, c \rightarrow \sigma', c' \]

- running statement \( c \) in state \( \sigma \) results in a new state \( \sigma' \) and a new (“remaining”) command \( c' \).
  - \( c' \) serves as program counter

• An execution is a trace:

\[ \langle \sigma, c \rangle \rightarrow \langle \sigma_1, c_1 \rangle \rightarrow \langle \sigma_2, c_2 \rangle \rightarrow \cdots \rightarrow \langle \sigma_n, c_n \rangle \]
Structural Operational Semantics

- Rule-based definition of $\rightarrow$:

\[
\begin{align*}
\sigma, x := e & \rightarrow \sigma[x \mapsto \llbracket e \rrbracket\sigma], \text{skip} \\
\sigma, c_1 & \rightarrow \sigma', c_1' \\
\sigma, c_1; c_2 & \rightarrow \sigma', c_1'; c_2 \\
\sigma, \text{skip}; c_2 & \rightarrow \sigma, c_2
\end{align*}
\]
Structural Operational Semantics

\[
\begin{align*}
\text{if } e \text{ then } c_1 \text{ else } c_2 & \rightarrow \sigma, c_1 \\
\text{if } e \text{ then } c_1 \text{ else } c_2 & \rightarrow \sigma, c_2 \\
\text{while } e \text{ do } c & \rightarrow \sigma, \text{while } e \text{ do } c \\
\end{align*}
\]
Properties of a Semantics

• **Theorem.** (*determinism*)
  If $\sigma, c \rightarrow^* \sigma'_1, \text{skip}$ and $\sigma, c \rightarrow^* \sigma'_2, \text{skip}$ then $\sigma'_1 = \sigma'_2$.

• **Proof.** By induction on the length of the derivation.
  Notice that for a given $c$, at most one derivation rule may match.
Semantic Properties: euclid

- **Proposition**: the following program terminates (subject to the input condition).

```plaintext
# a,b > 0
while a ≠ b do
  if a < b then
    b := b - a
  else
    a := a - b
```

\[(\sigma_a > 0 \land \sigma_b > 0) \Rightarrow \exists \sigma'. \sigma, \text{euclid} \rightarrow^* \sigma', \text{skip}\]
Semantic Properties: euclid

\[(\sigma_a > 0 \land \sigma_b > 0) \Rightarrow \exists \sigma'. \sigma, \text{euclid} \rightarrow^* \sigma', \text{skip}\]

- **Proof**: by well-founded induction on states by the order relation \(\prec\) defined as —

\[\sigma_1 < \sigma_2 \triangleq \sigma_1 a \leq \sigma_2 a \land \sigma_1 b \leq \sigma_2 b \land \]
\[\sigma_1 a < \sigma_2 a \lor \sigma_1 b < \sigma_2 b\]

# \(a, b > 0\)
while \(a \neq b\) do
  if \(a < b\) then
    \[b := b - a\]
  else
    \[a := a - b\]
Variants of Operations Semantics

• Big-step semantics: $\sigma, c \Downarrow \sigma'$
  ‣ Also called “natural semantics”

• Small-step semantics for expressions
  ‣ When is it useful..?
Now, for the good stuff
What is Software Synthesis?

IBM, 1957

The FORTRAN Automatic Coding System

J. W. BACKUS†, R. J. BEEBER†, S. BEST‡, R. GOLDBERG†, L. M. HAIBT†,
H. L. HERRICK†, R. A. NELSON†, D. SAYRE†, P. B. SHERIDAN†,
H. STERN†, I. ZILLER†, R. A. HUGHES§, and R. NUTT||

Introduction

THE FORTRAN project was begun in the summer of 1954. Its purpose was to reduce by a large factor the task of preparing scientific problems for IBM's next large computer, the 704. If it were possible for the 704 to code problems for itself and produce as system is now complete. It has two components: the FORTRAN language, in which programs are written, and the translator or executive routine for the 704 which effects the translation of FORTRAN language programs into 704 programs. Descriptions of the FORTRAN language and the translator form the principal
What is Software Synthesis

MIT & NASA, 1957

"Code" (≈165cm)
Excel 2013’s coolest new feature that should have been available years ago
# FlashFill: a feature of Excel 2013

[Gulwani 2011]

<table>
<thead>
<tr>
<th>Column 1</th>
<th>A</th>
<th>Col 2</th>
<th>Col 3</th>
<th>Col 4</th>
<th>Col 5</th>
<th>Col 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ana Trujillo</td>
<td>357 21th Place SE, Redmond, WA, (757) 555-1634, 140-37-6064, 27171</td>
<td>Redmond</td>
<td>WA</td>
<td>(757) 555-1634</td>
<td>140-37-6064</td>
<td>27171</td>
</tr>
<tr>
<td>Antonio Moreno</td>
<td>515 93th Lane, Renton, WA, (411) 555-2786, 562-87-3127, 28581</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thomas Hardy</td>
<td>742 17th Street NE, Seattle, WA, (412) 555-5719, 921-29-4931, 24607</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Christina Berglund</td>
<td>475 22th Lane, Redmond, WA, (443) 555-6774, 844-35-6764, 30146</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hanna Moos</td>
<td>785 45th Street NE, Puyallup, WA, (376) 555-2482, 515-68-1285, 29284</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frédérique Citeaux</td>
<td>308 66th Place, Redmond, WA, (689) 555-2770, 552-23-2608, 21415</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Martin Sommer</td>
<td>887 86th Place, Kent, WA, (715) 555-5450, 870-91-9824, 21536</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Laurence Lebihan</td>
<td>944 13th Street NE, Redmond, WA, (620) 555-2381, 1649-25-5312, 25252</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Elizabeth Lincoln</td>
<td>452 73th Lane NE, Renton, WA, (851) 555-4561, 425-97-6344, 22279</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Victoria Ashworth</td>
<td>463 16th Street, Renton, WA, (696) 555-6044, 690-29-7926, 28322</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Patricio Simpson</td>
<td>630 20th Street, Redmond, WA, (179) 555-3265, 389-78-3236, 24525</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Franklin Chang</td>
<td>683 49th Lane, Seattle, WA, (272) 555-7434, 665-18-6435, 29453</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Yang Wang</td>
<td>944 28th Lane, Redmond, WA, (151) 555-2272, 846-78-4852, 24388</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pedro Afonso</td>
<td>411 70th Place, Kent, WA, (170) 555-2964, 774-36-2298, 29485</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Elizabeth Brown</td>
<td>971 20th Lane, Puyallup, WA, (373) 555-4134, 476-53-7164, 26417</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sven Ottlieb</td>
<td>676 17th Lane NE, Redmond, WA, (828) 555-1593, 548-73-8633, 27440</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Janine Labrune</td>
<td>267 95th Place SE, Seattle, WA, (949) 555-1316, 350-27-8300, 28074</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ann Devon</td>
<td>694 53th Place, Kent, WA, (194) 555-8124, 559-74-4016, 22367</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Roland Mendel</td>
<td>581 12th Street NW, Kent, WA, (103) 555-2146, 303-79-1328, 20518</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Aria Cruz</td>
<td>594 85th Lane, Renton, WA, (431) 555-1376, 329-93-9992, 21498</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Diego Roel</td>
<td>550 22th Lane, Renton, WA, (639) 555-6238, 918-34-5172, 25931</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Martine Rance</td>
<td>688 93th Place NW, Kent, WA, (673) 555-3571, 695-94-3479, 22424</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
FlashFill: a feature of Excel 2013

[Gulwani 2011]
Modern Software Synthesis: Sketch

Problem: isolate the least significant zero bit in a word

- example: 0010 1011 → 0000 0100

Easy to implement with a loop

```
int W = 32;

bit[W] isolate0 (bit[W] x) { // W: word size
    bit[W] ret = 0;
    for (int i = 0; i < W; i++)
        if (!x[i]) { ret[i] = 1; return ret; }
}
```

Can this be done more efficiently with bitwise operations?

- Trick: adding 1 to a string of ones turns the next zero to a 1
- i.e. 000111 + 1 = 001000
Sketch: space of possible implementations

/**
 * Generate the set of all bit-vector expressions
 * involving +, &, xor and bitwise negation (~).
 * the bnd param limits the size of the generated expression.
 */

generator bit[W] gen(bit[W] x, int depth) {
    assert depth > 0;
    if (??) return x;
    if (??) return ??;
    if (??) return ~gen(x, depth-1);
    if (??) {
        return { | gen(x, depth-1) (+ | & | ^) gen(x, depth-1) |};
    }
}
Sketch: synthesis goal

```c
int W = 32;

bit[W] isolate0 (bit[W] x) { ... }

generator bit[W] gen(bit[W] x, int depth) {
    assert depth > 0;
    if (??) return x;
    if (??) return ??;
    if (??) return ~gen(x, depth-1);
    if (??) {
        return { | gen(x, depth-1) (+ | & | ^) gen(x, depth-1) |};
    }
}

bit[W] isolate0sk (bit[W] x) implements isolate0 { 
    return gen(x, 3);
}
```
Modern Software Synthesis: Synquid

[Polikarpova et al. 2016]

- Problem: intersection of strictly sorted lists
  - example: [4, 8, 15, 16, 23, 42] [8, 16, 32, 64] → [8, 16]

- Also: we want a guarantee that it’s correct on all inputs — of any size!

- Step 1: define a data type for sorted lists

```haskell
data SList e where
  Nil :: SList e
  Cons :: h:e →
    t:SList {v:e | v > h} →
    SList e
```
Synquid: components and synthesis goal

- **Step 2: define a set of components**
  - Which primitive operations is our function likely to use?
  - Here: \{\text{Nil, Cons, <}\}

- **Step 3: define synthesis goal as a type**

```plaintext
define intersect :: xs:SList e \rightarrow ys:SList e \rightarrow
   \{v:SList e \mid \text{elems } v = \text{elems } xs \cap \text{elems } ys\}
```
Problem: “Find the number of papers in OOPSLA 2010”

Output:

```
SELECT count(Publication.pid)
FROM Publication JOIN Conference
    ON Publication.cid = Conference.cid
WHERE Conference.name = "OOPSLA"
    AND Publication.year = 2010
```
What is Software Synthesis?

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

level of abstraction

???
Python, Haskell, ...
C
assembly
machine code

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

[Gulwani 2010]

Behavioral constraints
how do you tell the system what the program should do?

Structural constraints
what is the space of programs to explore?

Search strategy
How does the system find the program you want?
Behavioral Constraints

• How do you tell the system what the program should do?
  • What is the input language / format?
  • What is the interaction model?
  • What happens when the intent is ambiguous?

Q: What did behavioral constraints look like in FlashFill / Sketch / Synquid / SQLizer?
Behavioral Constraints

For example

- Input/output examples
- Equivalent program
- Formal specifications (pre/post conditions, types, ...)
- Natural language
Structural Constraints

• What is the space of programs to explore?
  • Large enough to contain interesting programs, yet small enough to exclude garbage and enable efficient search
  • Built-in or user defined?
  • Can we extract domain knowledge from existing code?

Q: What did structural constraints look like in FlashFill / Sketch / Synquid / SQLizer?
Structural Constraints

For example

Built-in DSL

User-defined DSL (grammar)
  • + statistical models

User-provided components
  • within straight-line code
  • within recursive functional programs

Languages with synthesis constructs
  • *e.g.* generators in Sketch
Search Strategies

• **Synthesis is search.**
  - Find a program in the space defined by *structural constraints* that satisfies *behavioral constraints*

• **Challenge:** the space is astronomically large
  - The search algorithm is the heart of a synthesis technique

• How does the system find the program you want?
  - How does it know it’s the program you want?
  - How can it leverage structural constraints to guide the search?
  - How can it leverage behavioral constraints to guide the search?
For next week