Learning to Synthesize

Yingfei Xiong

Peking University

Program Estimation

- Problem:
  - Given a context c, find program $s = \mathrm{argmax}_s\, P(s \mid c)$?
  - A sub problem of program synthesis
- Applications:
  - Learning by examples
    - Context = input/output examples
  - Code completion
    - Context = partial code
  - Code generation from natural language description
    - Context = natural language description
  - Test-based program repair
    - Context = buggy program & tests
  - Test Generation
    - Context = program under test, P = bug-detecting capability

Challenges

- How to estimate the conditional probability $P(Prog \mid Context)$?
  - Should be consistent with other constraints, e.g., $P(invalid \mid Context) = 0$

- How to find program s such that $P(\, s \mid context \,)$ is the largest?
  - The space of program is huge

Learning to synthesis (L2S)

- A general framework to address program estimation

- Combining four tools
  - **Syntax**: defining a search problem
  - **Constraints**: pruning off invalid choices in each step
  - **Machine-learned models**: estimating the probabilities of choices in each step
  - **Search algorithms**: solving the search problem

Example – Repairing incorrect conditions

- Condition bugs are common

```
  hours = convert(value);
+ if (hours > 12)
+   throw new ArithmeticException();
```

Missing boundary checks

```
-  if (hours >= 24)
+ if (hours > 24)
      withinOneDay=true;
```

Conditions too weak or too strong

- Existing work can pinpoint incorrect condition
- Can we generate a correct condition to replace the incorrect one?

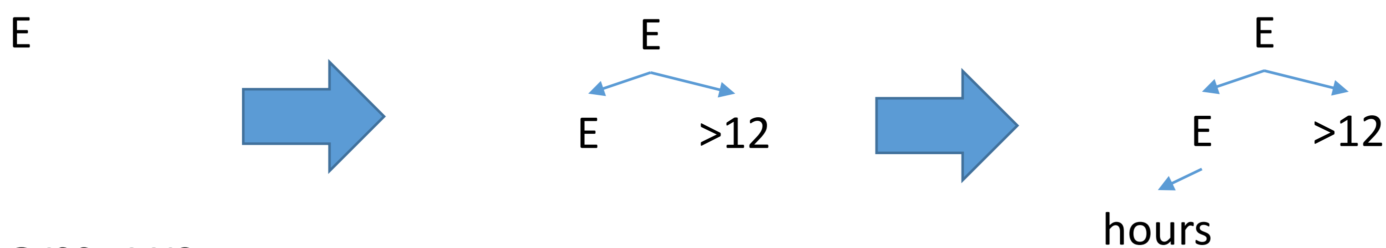
Syntax

- $\mathrm{E} \rightarrow \mathrm{E}$ “>12”
  | E “>0”
  | E “+” E
  | *“hours”*
  | *“value”*
  | …

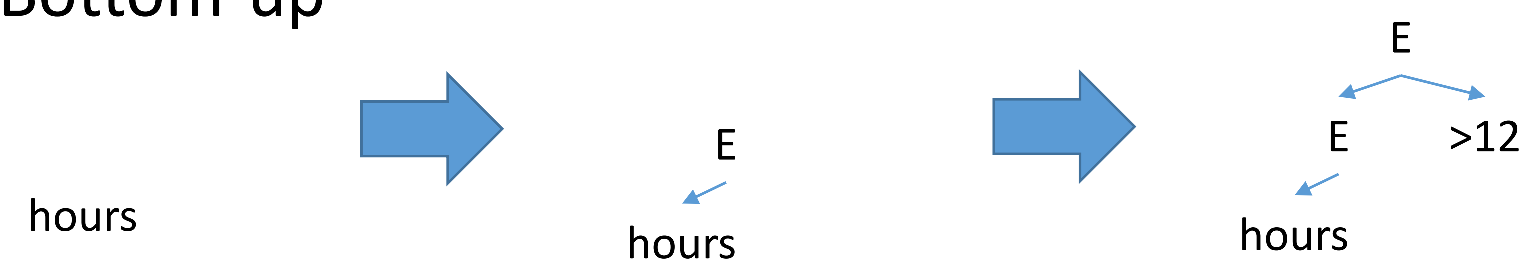
- The syntax defines the search space of conditional expressions

Search Order

- A program may be completed in different orders
  - hours>12
- Top-down



- Bottom-up



The order may greatly affect the performance of L2S.

Annotations

- Introduce annotations to symbols
  - E^D indicates E can be expanded downward
  - E^U indicates E can be expanded upward
  - E^{UD} indicates E can be expanded in both directions

From Grammar to Rewriting Rules

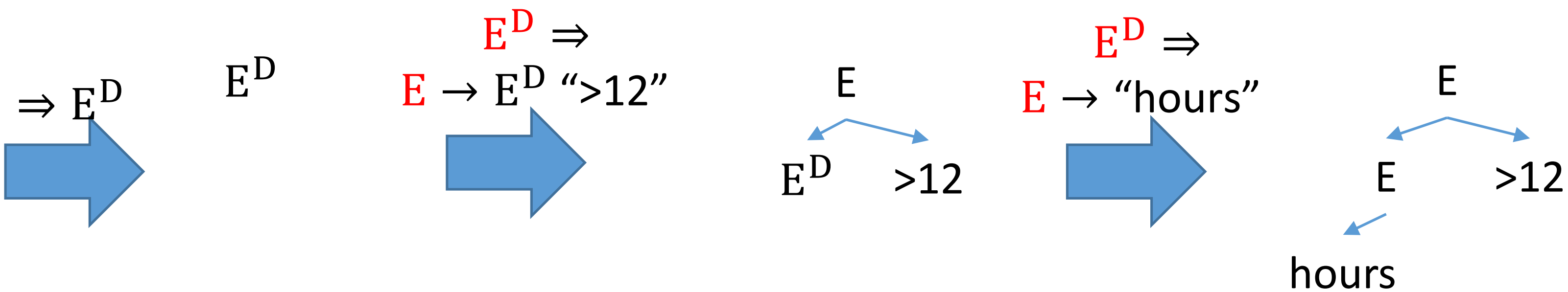
Grammar	Top-down Rules	Bottom-up Rules
$E \rightarrow E$ "+" E	$E^D \Rightarrow E \rightarrow E^D$ "+" E^D	$E^U \Rightarrow E^U \rightarrow E$ "+" E^D $E^U \Rightarrow E^U \rightarrow E^D$ "+" E
$E \rightarrow E$ ">12"	$E^D \Rightarrow E \rightarrow E^D$ ">12"	$E^U \Rightarrow E^U \rightarrow E$ ">12"
$E \rightarrow$ "hours"	$E^D \Rightarrow E \rightarrow$ "hours"	"hours"$^D \Rightarrow E^D \rightarrow$ "hours"

Creation Rules	
$\Rightarrow E^D$	// starting from the root
$\Rightarrow E^{DU}$	// starting from a middle node
$\Rightarrow$ "hours"^{U}	// starting from a leaf

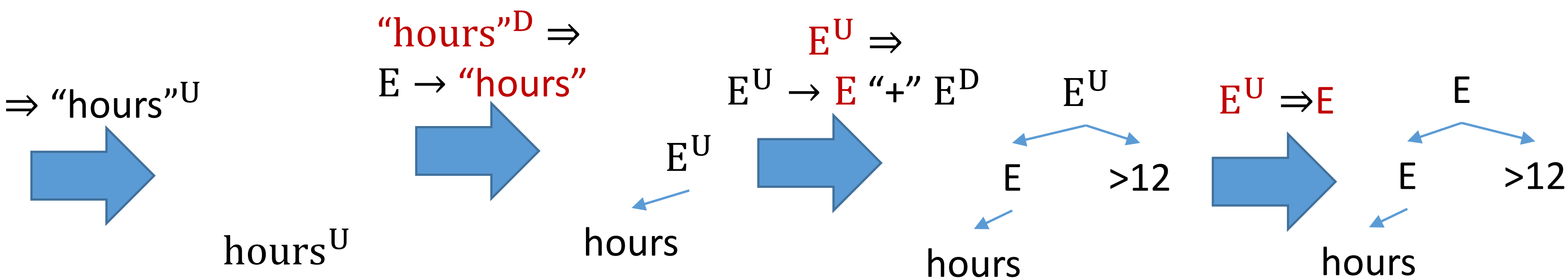
Ending Rule	$E^U \Rightarrow E$

Example

- Top-down



- Bottom-up



Search Problem

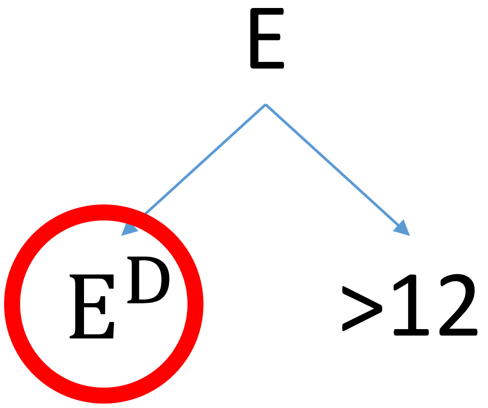
- Each partial AST is a state
  - Start state is an empty AST
  - Ending state is a complete AST without annotations
- Each rewriting rule application gives a successor

- In practice, a subset of rewriting rules is selected to give a more deterministic order, e.g.,
  - Top-down rules + root creation rules
  - Bottom-up rules + leaf creation rules + ending rules

Solving the search problem

- In each step, we need to decide
- (1) which node to expand
- (2) which rule is used to expand

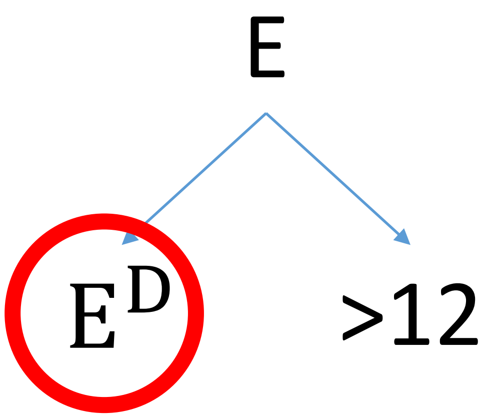
Decisions in Each Step



(1) Choosing a node to expand

A policy is used to pick a node in a fixed order

Decisions in Each Step



$$
\begin{aligned}
E^D \Rightarrow\ & E \rightarrow E^D \text{ “+” } E^D \\
\mid\ & \cancel{E \rightarrow E^D \text{ “>12”}} \\
\mid\ & E \rightarrow \text{“hours”}
\end{aligned}
$$

(2) Pruning off invalid choices

Constraints are generated and satisfiability is checked.

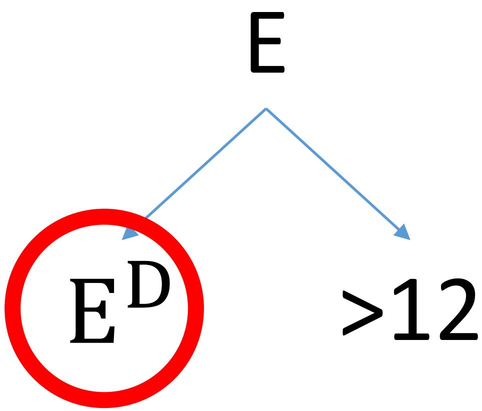
Structural Constraint Generation

- Each AST node defines a set of variables
  - For type constraints, each node n defines a type variable type[n]
- Each grammar rule defines a set of constraints
  - $\mathrm{E}_1 \rightarrow \mathrm{E}_2$ “>12”
    - T[E_1] = Boolean
    - T[E_2] = Int
  - $\mathrm{E}_2 \rightarrow \mathrm{E}_3$ “>12”
    - T[E_2] = Boolean
    - T[E_3] = Int
- The context gives a set of constraints
  - $\mathrm{T}[\mathrm{hours}] = \mathrm{Int}$
  - $\mathrm{T}[\mathrm{E}_{\mathrm{root}}] = \mathrm{Boolean}$
- If a solver returns unsat, drop the current choice

Other constraints

- Semantic constraints
  - The variables are values at each node
  - The constraints are semantics of the operators
- Size constraints
  - The variables are size of the subtree starting from a node
  - The constraints are formulas to calculate the size

Decisions in Each Step



$$E^D \Rightarrow E \rightarrow E^D \text{ "+" } E^D \qquad 0.05$$
$$| \quad \cancel{E \rightarrow E^D \text{ ">12"}}$$
$$| \quad E \rightarrow \text{"hours"} \qquad 0.8$$

(3) Estimating the probabilities of the remaining choices

Training a machine-learned model

Training a model

- The user chooses a machine learning method for each symbol with U or D
- The model is trained on a corpus of programs and their contexts
  - A search process is re-enacted on each program and the choice of rules are stored as the training set
- Local names should be handled with proper embedding

Local Optimal ≠ Global Optimal

E_0

E → E "> 12"	0.3
E → E "> 0"	0.6

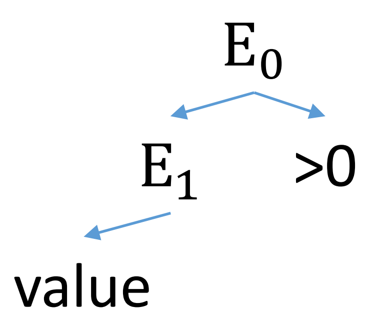
E_1

E → "hours"	0.1
E → "value"	0.2
E → E "+" E	0.05

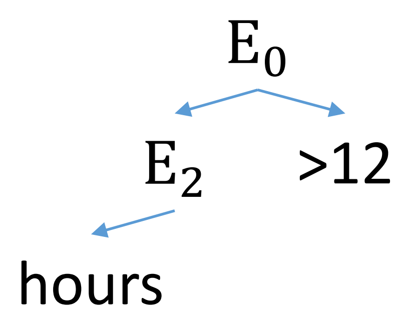
E_2

E → "hours"	0.8
E → "value"	0.1
E → E "+" E	0.05

0.6 * 0.2
= 0.12



0.3 * 0.8
= 0.24



Search Algorithm

- Beam search – a greedy method to solve the search problem
  - Keep 'n' most likely programs and their probabilities
  - Each time, use all valid rule applications to expand the program
  - Keep 'n' most likely expanded programs where their new probability is the highest
    - new probability = old probability $\times$ choice probability

- Other search algorithms may also be used

Preliminary Evaluation

- Predicting conditional expressions in large programs, with surrounding code as context
- A bottom-up order starting from the left most variable
- Use xgboost to train the models and features are manually defined
- Models are trained on the conditional expressions in the same project
- 10-fold cross validation is used

Results

Project	kLOC	Top 10 Precision
Chart-26	146	64.7%
Math-82	104	75.6%
Time-15	81	85.7%
Lang-9	28	85.2%

Summary and Future Work

- Learning to Synthesize: a framework for estimating a program under a context
- Work-in-progress
- Many future directions
  - Applications
    - Program repair, code generation, test generation, fault localization
    - Program verification?
  - Can we automate the choice of rule set?
  - What is the effect of different policies for choosing nodes?
  - Can we use better search algorithms? E.g., MCMC, bayesian optimization