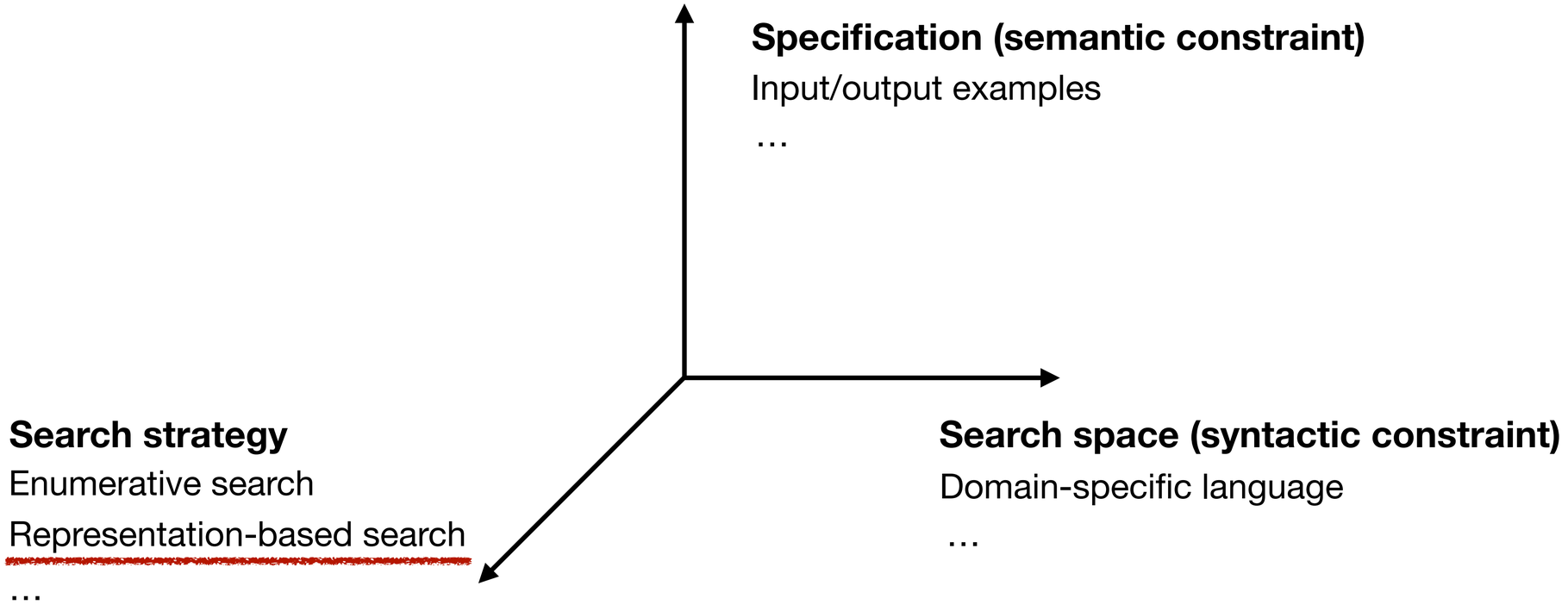
Advanced Software Security

6. Representation-based Search

Kihong Heo

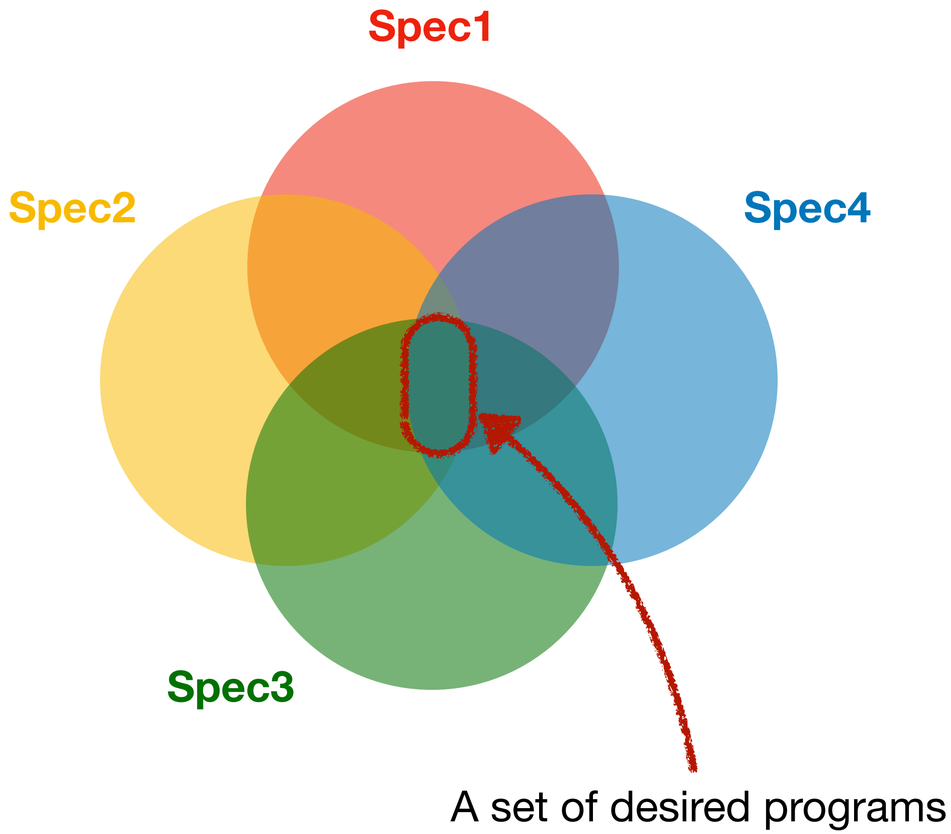
KAIST

Dimensions in Program Synthesis



Goal: Finding a Set of Programs

- So far: search for a single solution
  - Enumerate one-by-one
- This lecture: search for a set of solutions
  - Return multiple results then rank them
  - Space-efficient search



Representation-based Search

- Idea:
  - Build a data structure that concisely represents a set of programs
  - Extract solutions from that data structure
- Two well-known methods
  - Version space algebra (VSA)
  - Finite tree automata (FTA)

Version Space

- Hypothesis: a function that takes an input and an output
- Hypothesis space *H*: a set of all hypotheses (i.e. programs)
- Version space $VS_{H,D} \subseteq H$: a set of programs that satisfy the examples in the given dataset
  - $D = \{(in_i, out_i)\}_i$: a set of input-output examples
  - $h \in VS_{H,D} \iff \forall i, o \in D \,.\, h(i) = o$

Version Space Algebra

- A set of operations to manipulate and compose version space
- Operations on version spaces:
  - learn(i, o): construct a version space of functions consistent with (i, o)
  - $VS_1 \cap VS_2$, $VS_1 \cup VS_2$: intersection and union of two version spaces
  - pick VS: pick a function from version space VS
- Synthesis idea: use of compact symbolic representation for the version spaces

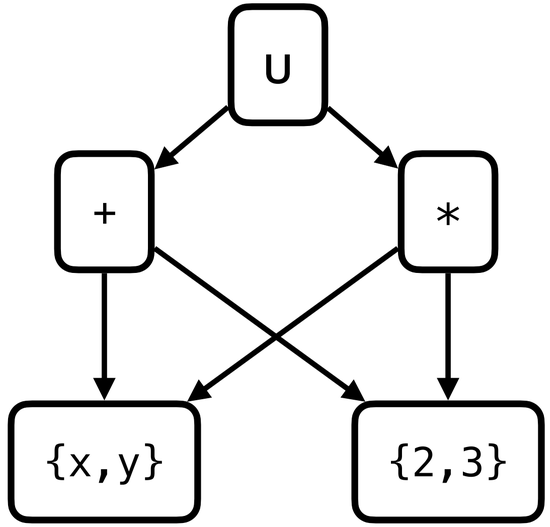
Syntax of VSA

- Grammar of VSA

$$\widetilde{P} ::= \{P_1, \ldots, P_k\} \mid \mathbf{U}(\widetilde{P}_1, \ldots, \widetilde{P}_k) \mid F_{\bowtie}(\widetilde{P}_1, \ldots, \widetilde{P}_k)$$

- Example: {x+2, x+3, y+2, y+3, x∗2, x∗3, y∗2, y∗3}

$$\mathbf{U}(+_{\bowtie}(\{x,y\}, \{2,3\}), *_{\bowtie}(\{x,y\}, \{2,3\}))$$

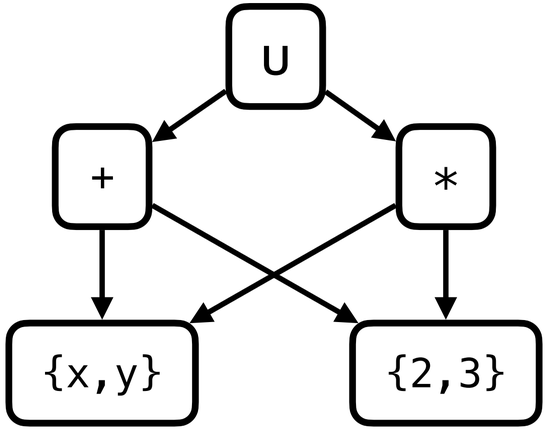


Semantics of VSA

- A program P is an element of a VSA

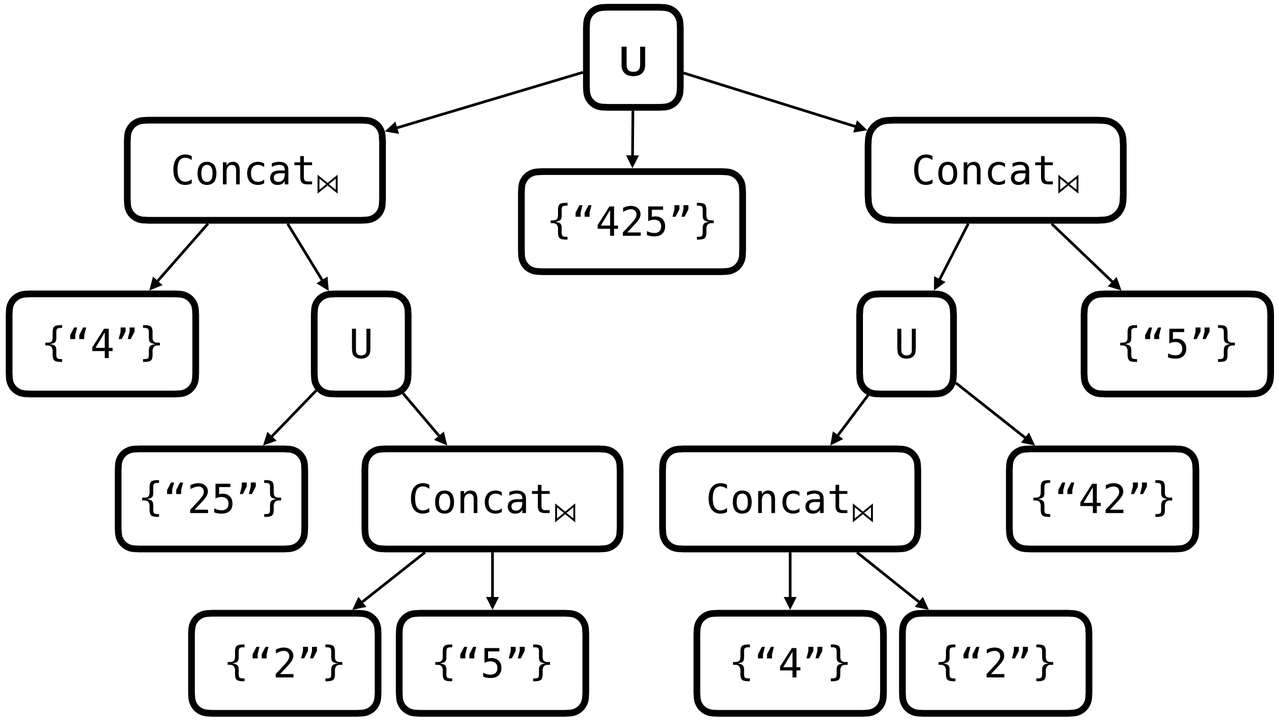
$$
\begin{aligned}
&P \in \{P_1, \ldots, P_k\} && \exists j.P = P_j \\
&P \in \mathbf{U}(\widetilde{P}_1, \ldots, \widetilde{P}_k) && \exists j.P = \widetilde{P}_j \\
&P \in F_{\bowtie}(\widetilde{P}_1, \ldots, \widetilde{P}_k) \quad P = F(P_1, \ldots, P_k) \wedge \forall j.P_j \in \widetilde{P}_j
\end{aligned}
$$

- Example:



Example

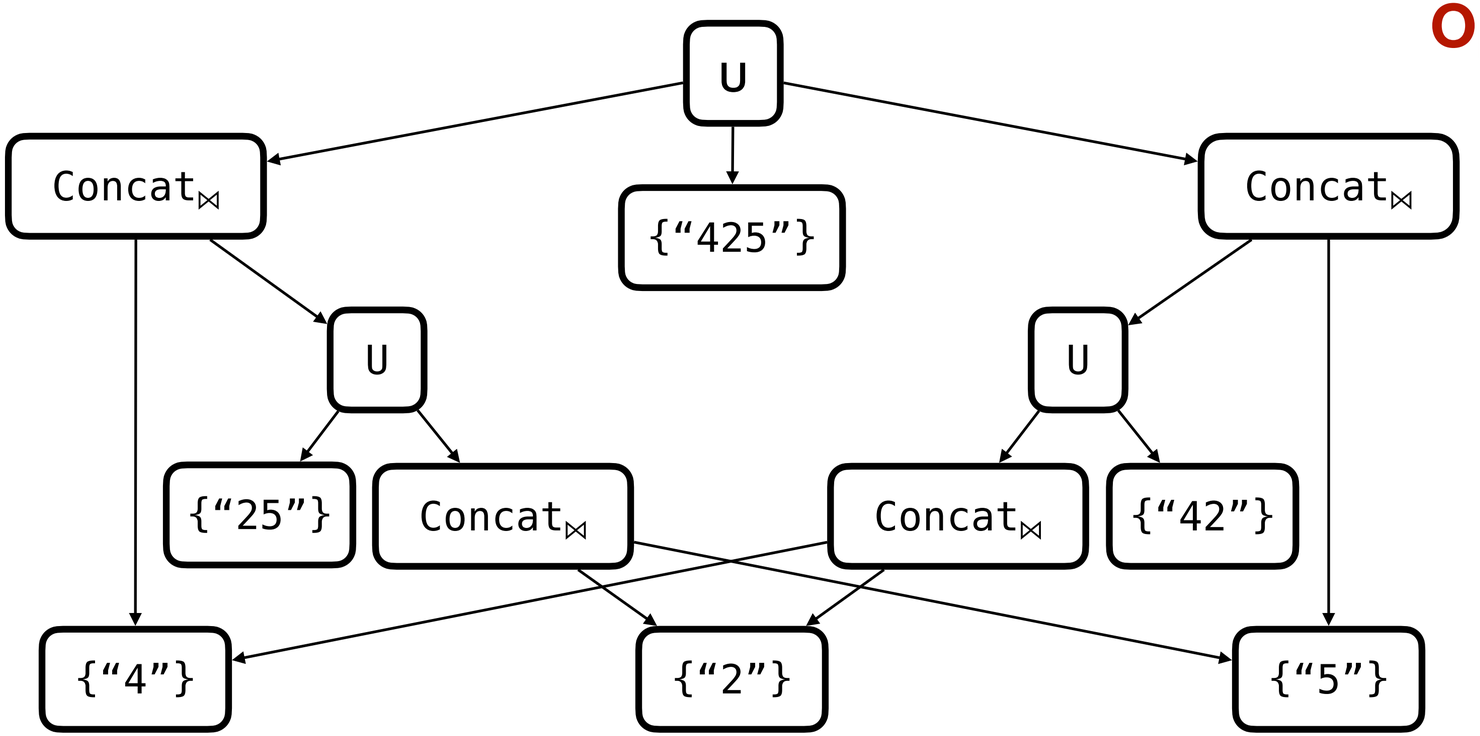
- Grammar $S \rightarrow ConstStr \mid \text{Concat}(S, S)$
- A set of program that returns “425”



Example

- Grammar $S \rightarrow ConstStr \mid \text{Concat}(S, S)$
- A set of program that returns “425”

Optimization!

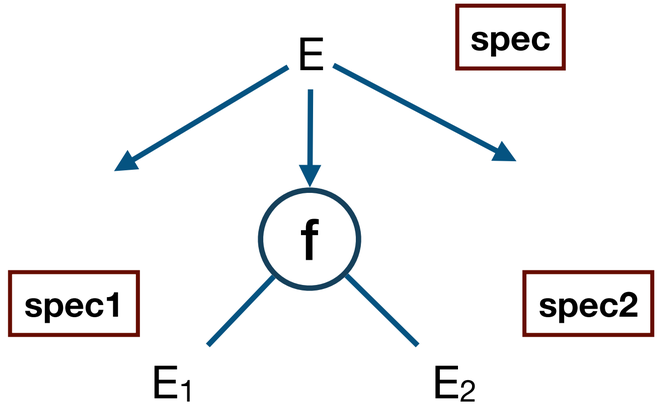


Efficiency

- Represent potentially exponential program sets in polynomial space
  - V(VSA) : # nodes in VSA
  - |VSA| : # programs in VSA
  - V(VSA) = O(log|VSA|)
- E.g., millions of programs => hundreds of nodes

TDP with VSA

- Given a spec and a production, infer specs for subprograms (divide-and-conquer)
  - When $f<E_1, E_2, \ldots, E_n>$ (In)= Out where E_i is a subprogram
  - What is the spec for each E_i?

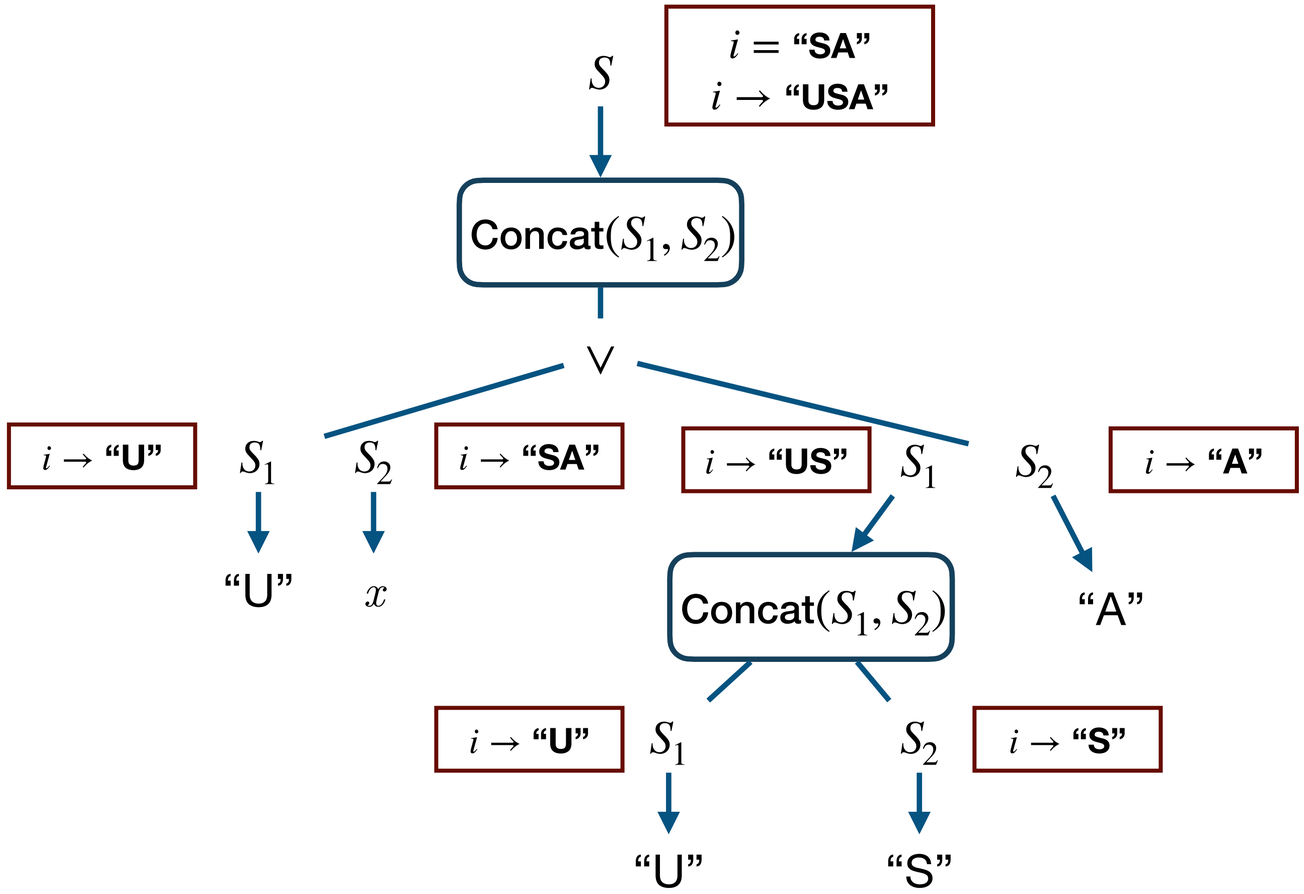


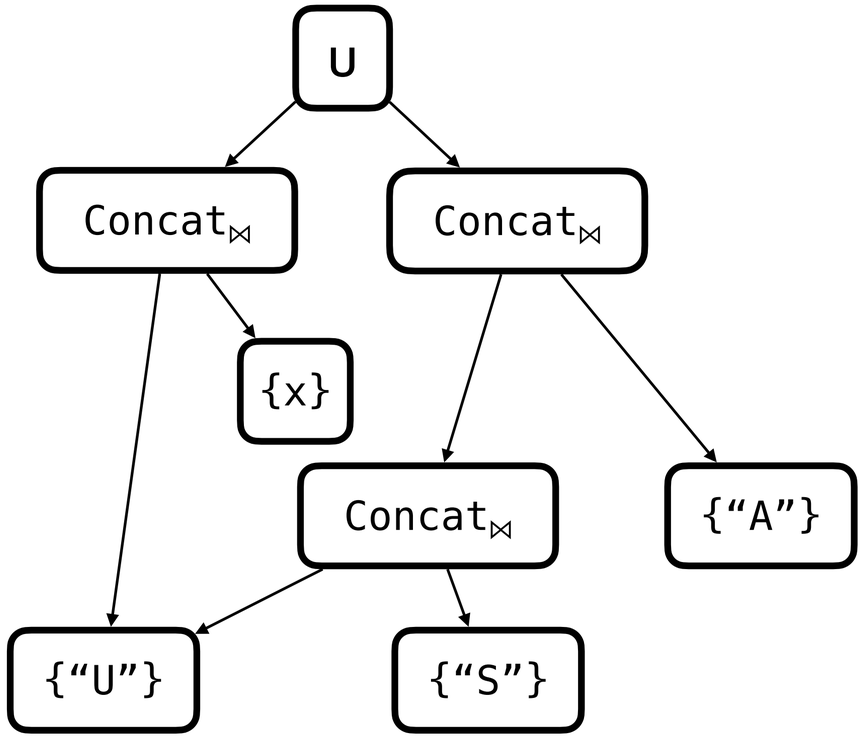
Example

- Grammar: $S \rightarrow ConstStr \mid \text{Concat}(S, S)$
- Specification: $f(\text{“SA”}) = \text{“USA”} \wedge f(\text{“AE”}) = \text{“UAE”}$
- Inverse set:
  - $\text{Concat}^{-1}(\text{“USA”}) = \{(\text{“U”}, \text{“SA”}), (\text{“US”}, \text{“A”})\}$
  - $\text{Concat}^{-1}(\text{“UAE”}) = \{(\text{“U”}, \text{“AE”}), (\text{“UA”}, \text{“E”})\}$

Step 1-1: Learn

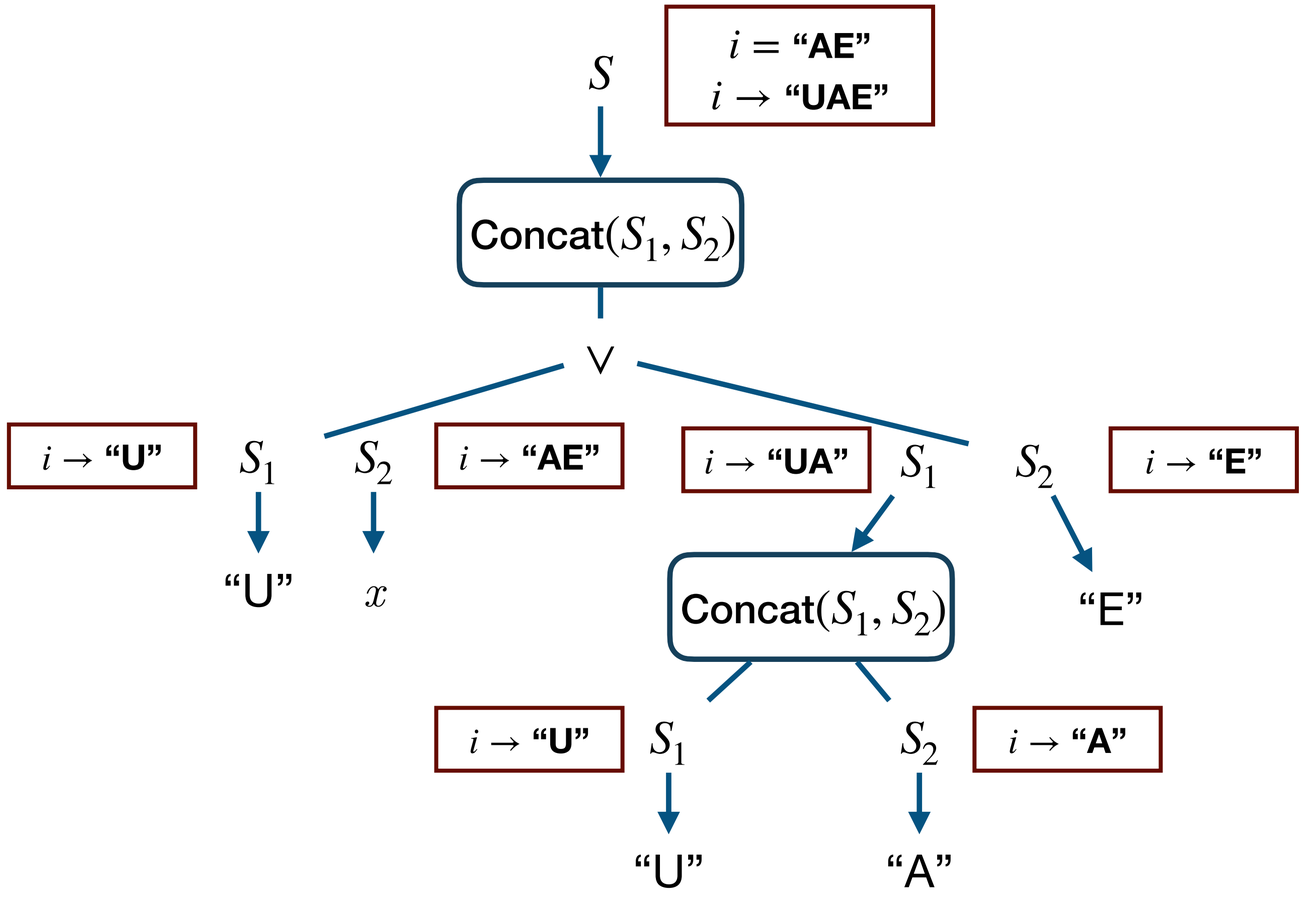
- Top-down propagation with one example

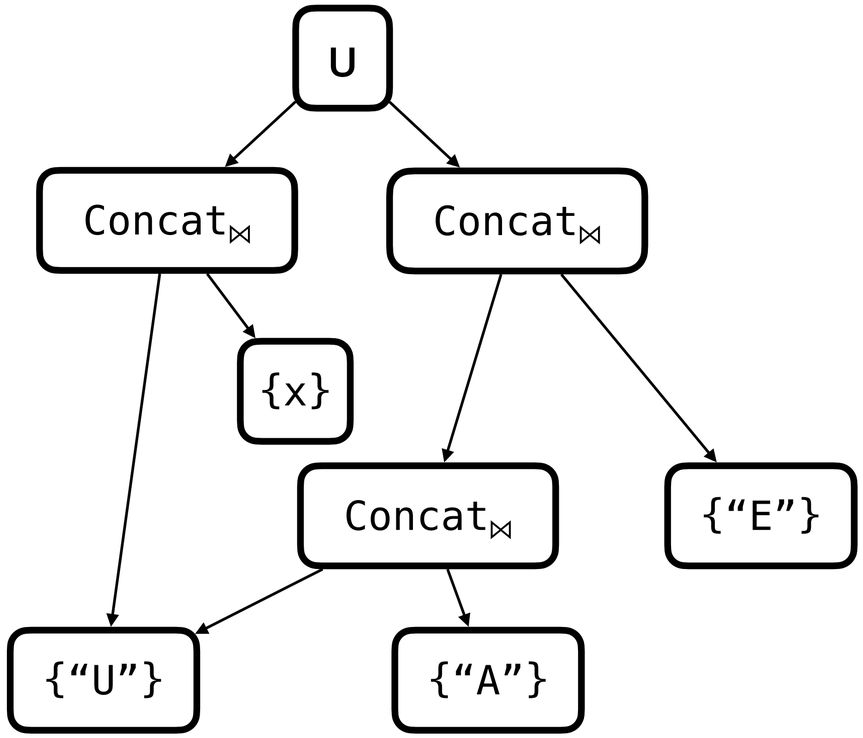




Step 1-2: Learn

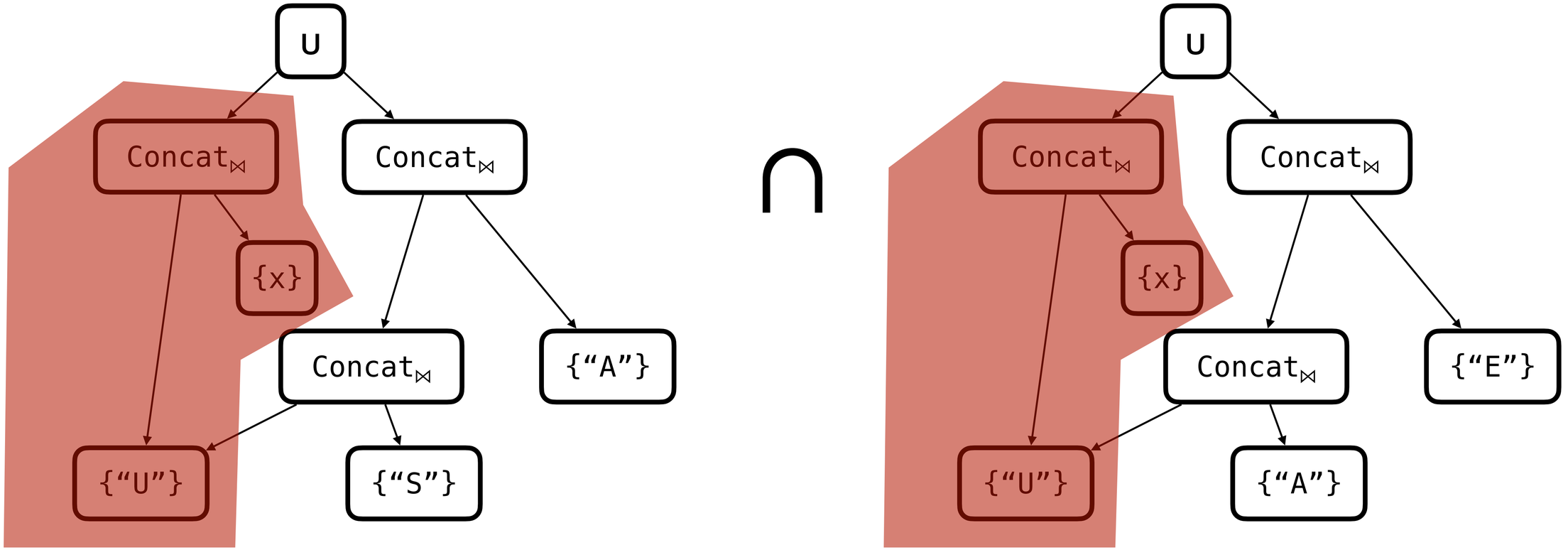
- Top-down propagation with next example





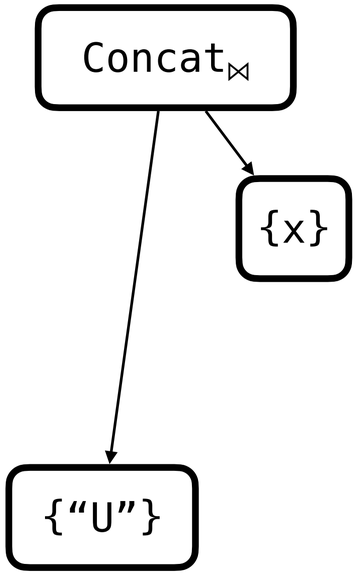
Step 2: Intersection

- Intersection of two version spaces



Step 3: Pick

- Pick a desired program



Concat(**“U”,** x**)**

Pros and Cons

- Pros: efficient
  - Applications: Excel, VSCode, etc
  - See https://www.microsoft.com/en-us/research/group/prose/
- Cons: not always applicable
  - Efficiently computable inverse function
  - Finite inverse set

Representation-based Search

- Idea:
  - Build a data structure that concisely represents a set of programs
  - Extract solutions from that data structure
- Two well-known methods
  - Version space algebra (VSA)
  - Finite tree automata (FTA)

Example

Specification

Find a function $f(x)$ where $f(1) = 9$

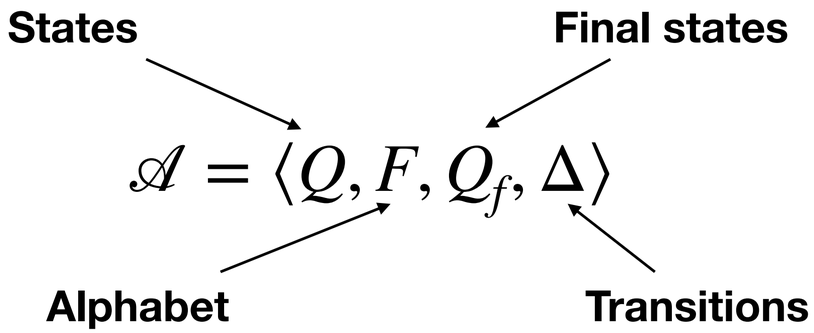
Grammar

$$N \to \texttt{id}(V) \mid N + T \mid N \times T$$
$$T \to 2 \mid 3$$
$$V \to x$$

Example

```
id(x) * 3 * 3
id(x) + 2 + 3 + 3
```

Finite Tree Automata



- Example

Find a function $f(x)$ where $f(1) = 9$

$N \rightarrow \texttt{id}(V) \mid N + T \mid N \times T$

$T \rightarrow 2 \mid 3$

$V \rightarrow x$

$Q = \{N, T, V\} \times \mathbb{N}_{\leq 12}$

$Q_f = \{\langle N, 9 \rangle\}$

$F = \{\texttt{id}, +, \times\}$

$f(q_1, \ldots, q_n) \rightarrow q$

$\Delta = \{\ \texttt{id}(\langle V, 1 \rangle) \rightarrow \langle N, 1 \rangle$

$+(\langle N, 1 \rangle, \langle T, 2 \rangle) \rightarrow \langle N, 3 \rangle$

$\times(\langle N, 1 \rangle, \langle T, 2 \rangle) \rightarrow \langle N, 2 \rangle \quad \ldots\ \}$

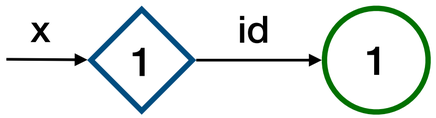
Bottom-up Search with FTA

Specification

Find a function $f(x)$ where $f(1) = 9$

Grammar

- $N \rightarrow \texttt{id}(V) \mid N + T \mid N \times T$
- $T \rightarrow 2 \mid 3$
- $V \rightarrow x$



$\texttt{id}(\langle V,1 \rangle) \rightarrow \langle N,1 \rangle$

Bottom-up Search with FTA

Specification

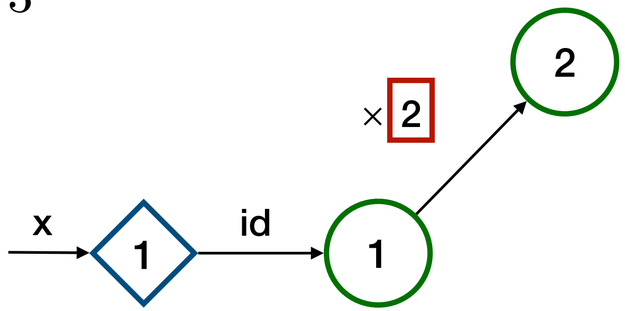
Find a function $f(x)$ where $f(1) = 9$

Grammar

$N \to \texttt{id}(V) \mid N + T \mid N \times T$

$T \to 2 \mid 3$

$V \to x$



$\times(\langle N,\ 1\rangle, \langle T,\ 2\rangle) \to \langle N,\ 2\rangle$

Bottom-up Search with FTA

Specification

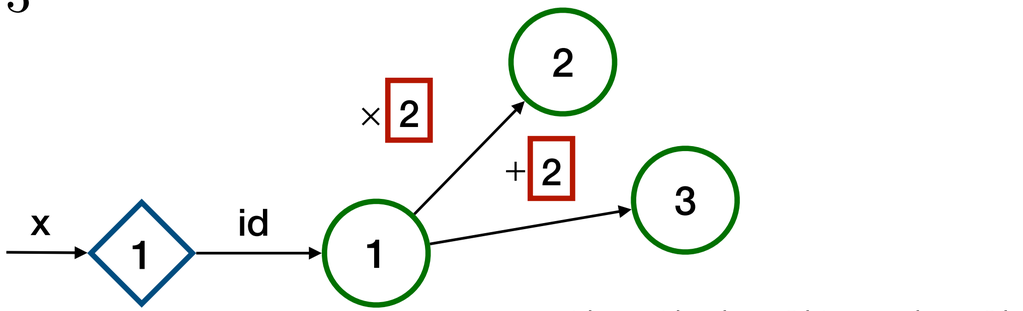
Find a function $f(x)$ where $f(1) = 9$

Grammar

$N \rightarrow \texttt{id}(V) \mid N + T \mid N \times T$

$T \rightarrow 2 \mid 3$

$V \rightarrow x$



$+(\langle N, 1\rangle, \langle T, 2\rangle) \rightarrow \langle N, 3\rangle$

Bottom-up Search with FTA

Specification

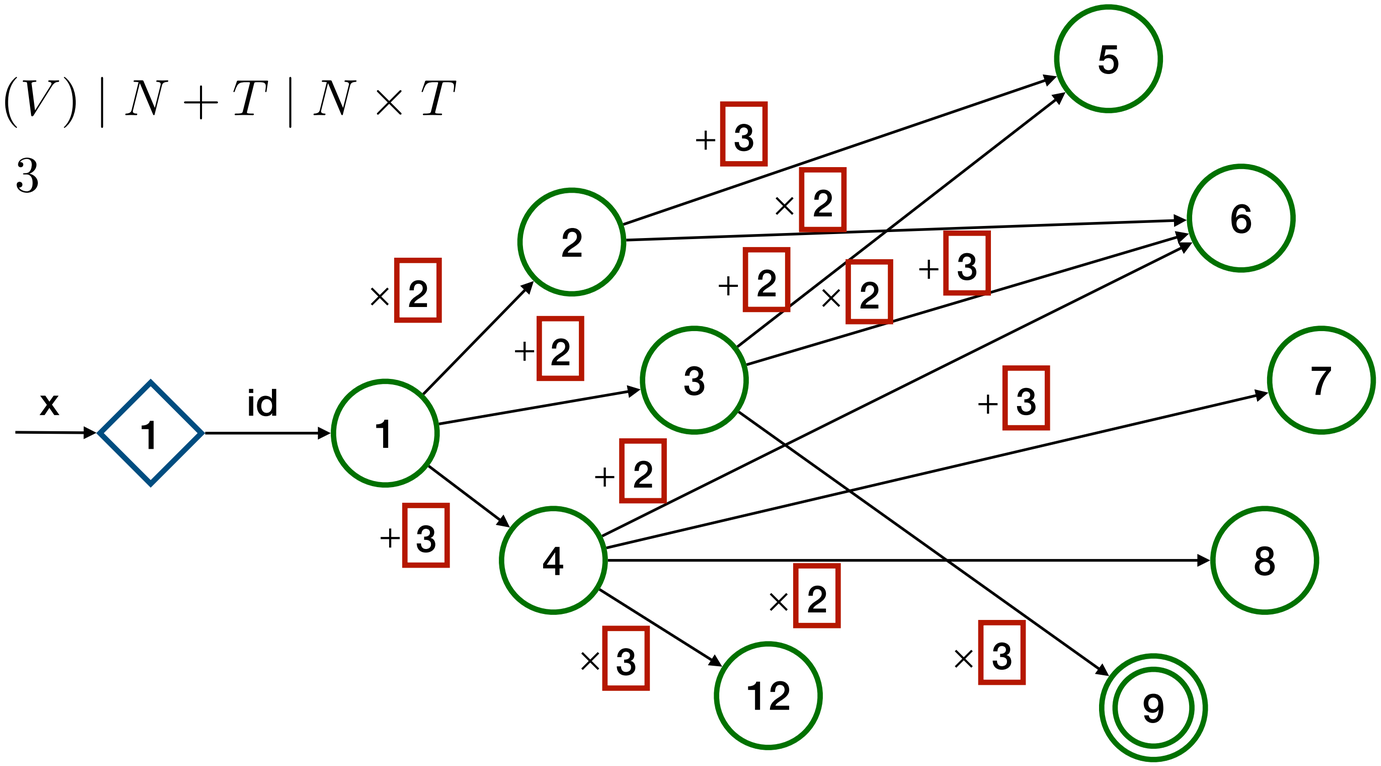
Find a function $f(x)$ where $f(1) = 9$

Grammar

$N \rightarrow \texttt{id}(V) \mid N + T \mid N \times T$

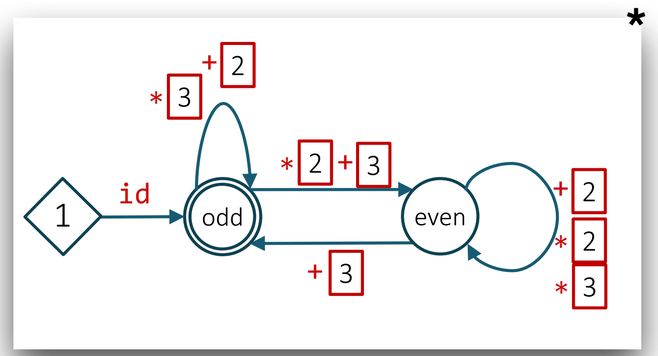
$T \rightarrow 2 \mid 3$

$V \rightarrow x$



Finite Tree Automata

- Space-efficient representation for bottom-up search
- Challenge: still too many states
- Idea: abstraction
  - E.g., {2,4,6,8, …} → even
  - Following the abstract interpretation theory
- See “Program Synthesis using Abstraction Refinement. POPL’18"



*https://github.com/nadia-polikarpova/cse291-program-synthesis/

Summary

- Representation-based search
  - Search with space-efficient data structure
  - Represent multiple programs within a simple representation
- Combination with other search strategies
  - Version space algebra + top-down search (TDP)
  - Finite tree automata + bottom-up search