Advanced Software Security

4. Search Space Pruning

Kihong Heo

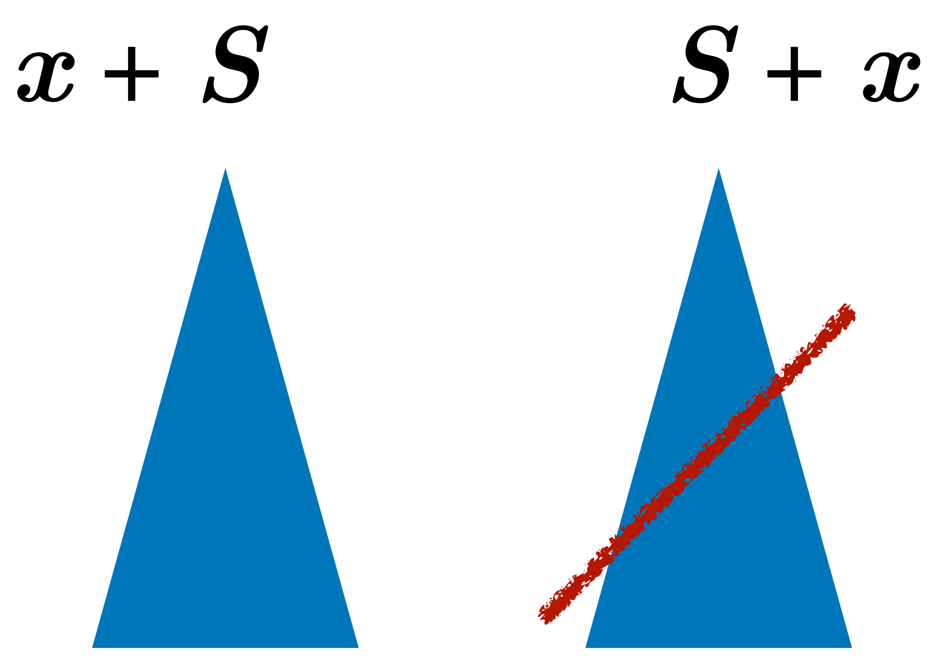
KAIST

Search Space Pruning

- Problem: too huge search space
- How to prune the search space for program synthesis?
- Common strategies:
    - Equivalence reduction = discard **redundant** candidates
    - Top-down propagation = discard **unpromising** candidates

Equivalence Reduction

- Discard **redundant** subprograms as early as possible
- How to implement this idea for top-down and bottom-up searches?



Eq. Reduction for Top-down

- If two non-ground programs will be expand the same set of programs, discard one of them
- What candidates can be discarded by the following programs in the queue?

$\texttt{if}\ (\mathbf{true})\ y\ S$ $\quad\quad$ $\texttt{1 * (}S\ \texttt{+}\ S\texttt{)}$ $\quad\quad$ $\texttt{sort(}S\texttt{)}$

Top-down + Eq. Reduction

```
top-down(G = <Σ, N, R, S>, ϕ):
  Q := {S}
  while Q != {}:
    p := dequeue(Q)
    if ground(p) ∧ ϕ(p): return p
    P’ := unroll(G, p)
    forall p’ ∈ P’:
+     if not equiv(p, p’):
        enqueue(Q, p’)
```

```
unroll(G = <Σ, N, R, S>, p):
  Q’ := {}
  A := left-most non-terminal in p
  forall (A → B) in R:
    p’ := p[B/A]
    Q’ := Q’ ∪ {p’}
  return Q’
```

Eq. Reduction for Bottom-up

- How to apply equivalence reduction for bottom-up search?
- Problem: candidates are all ground but might not be whole
- Solution for PBE: **observation equivalence**
  - We only care of equivalence on the given inputs
- E.g., the programs below are observationally equivalent modulo the inputs (1,0) and (2,1)

x $\qquad\qquad$ $\texttt{if}\ (x \leq y)\ \ y\ \ x$

Example

Specification

Find a function $f(x)$ where $f(1) = 3$

Grammar

$$S \rightarrow x \mid 1 \mid -S \mid S + S \mid S \times S$$

Enumeration

iter 1	x	~~1~~		
iter 2	$-x$	~~-1~~	~~$1 + x$~~	~~$1 + 1$~~
	$x + x$	~~$x + 1$~~	~~$x \times x$~~	…
iter 3	$x + x + x$			

Bottom-up + Eq. Reduction

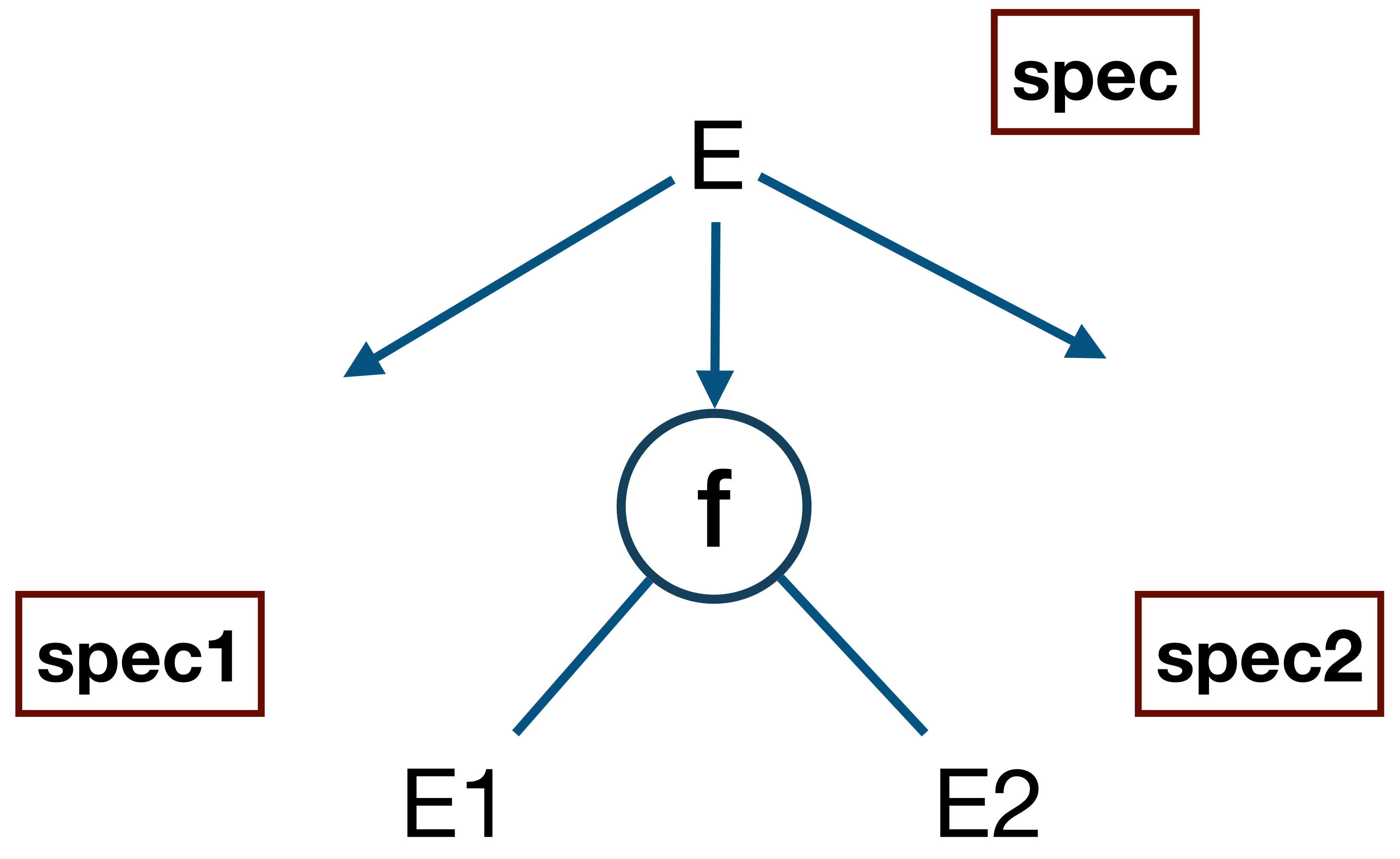
```
bottom-up(G = <Σ, N, R, S>, ϕ):
  Q := set of all terminals in G
  while true:
    forall p in Q:
      if ϕ(p): return p
    Q += grow(G, Q)

grow(G, Q):
  Q’ := {}
  forall (A -> B) in G:
    Q’ += { B[C → p] | p ∈ Q, C →* p }
+ Q’ := { p’∈ Q’ | forall p ∈ Q. ¬equiv(p, p’) }
  return Q’
```

Search Space Pruning

- Problem: too huge search space
- How to prune the search space for program synthesis?
- Common strategies:
  - Equivalence reduction
  - Top-down propagation

Top-down Propagation

- Discard **unpromising** subprograms as early as possible
- Given a spec and a production, infer specs for subprograms (divide-and-conquer)
  - When `f<E₁, E₂, …, Eₙ>` (In) = Out where `Eᵢ` is a subprogram
  - What is the spec for each `Eᵢ`?
  - If any `Eᵢ` is undesirable, discard `f`



Example: String Manipulation

Specification

Find a function $f(x)$ where $f(\text{"SA"}) = \text{"USA"} \land f(\text{"AE"}) = \text{"UAE"}$

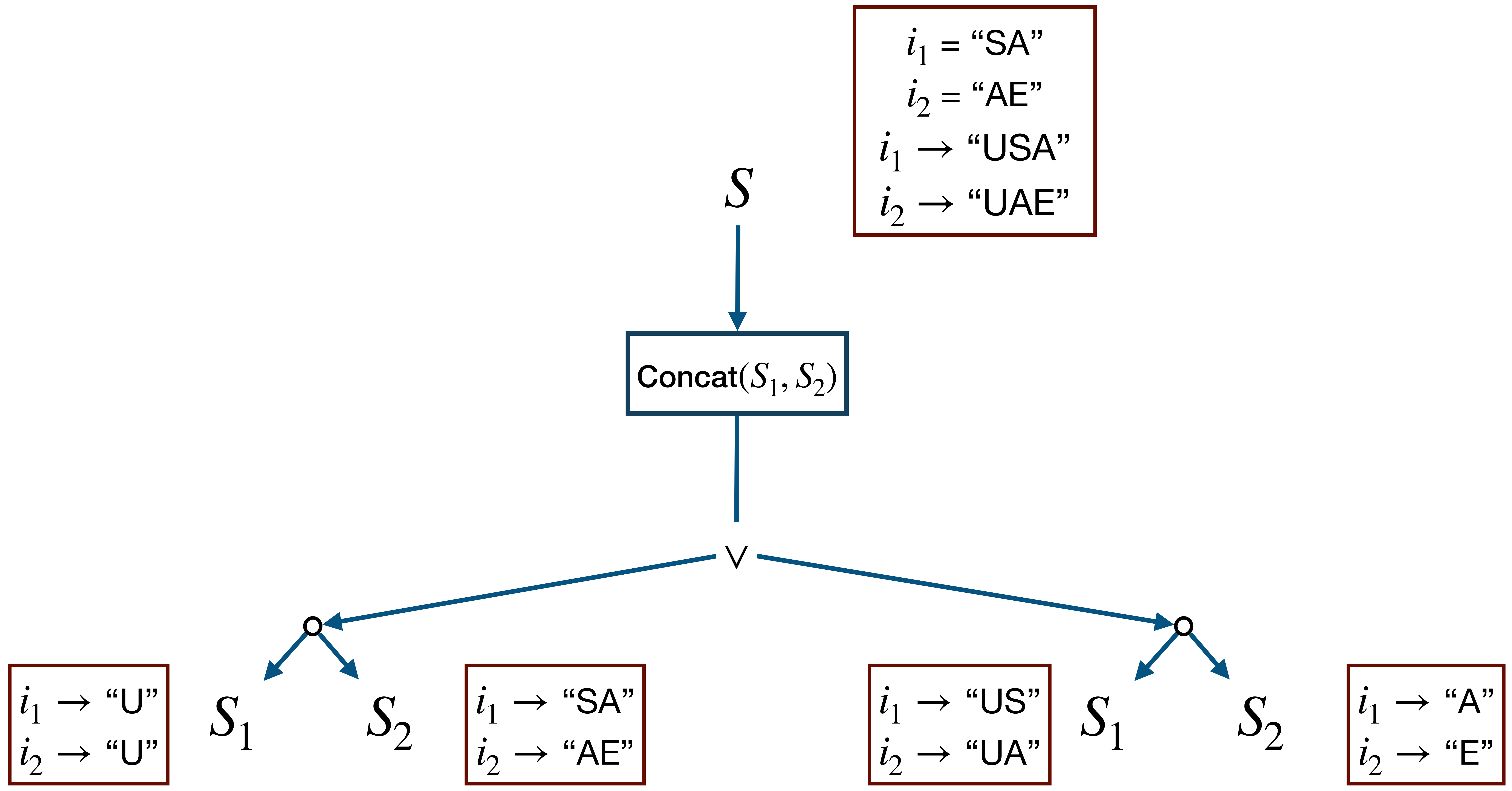
Grammar

$$S \to ConstStr \mid x \mid \texttt{concat}(S, S)$$

Examples

$$\texttt{concat}\ (\text{“}U\text{”}, \text{“}SA\text{”}) = \text{“}USA\text{”}$$

TDP for String Manipulation



Example: List Manipulation

Specification

Find a function $f(x)$ where f([1; -3; 1; 7]) = [2; -2; 2; 8]

Grammar

$$L \to \texttt{map}\ E\ x \mid \texttt{filter}\ E\ x \mid \texttt{fold}\ E\ E\ x$$

$$E \to \lambda y.\ E \mid E + E \mid E \leq E \mid y \mid 0 \mid 1 \mid \texttt{[]}$$

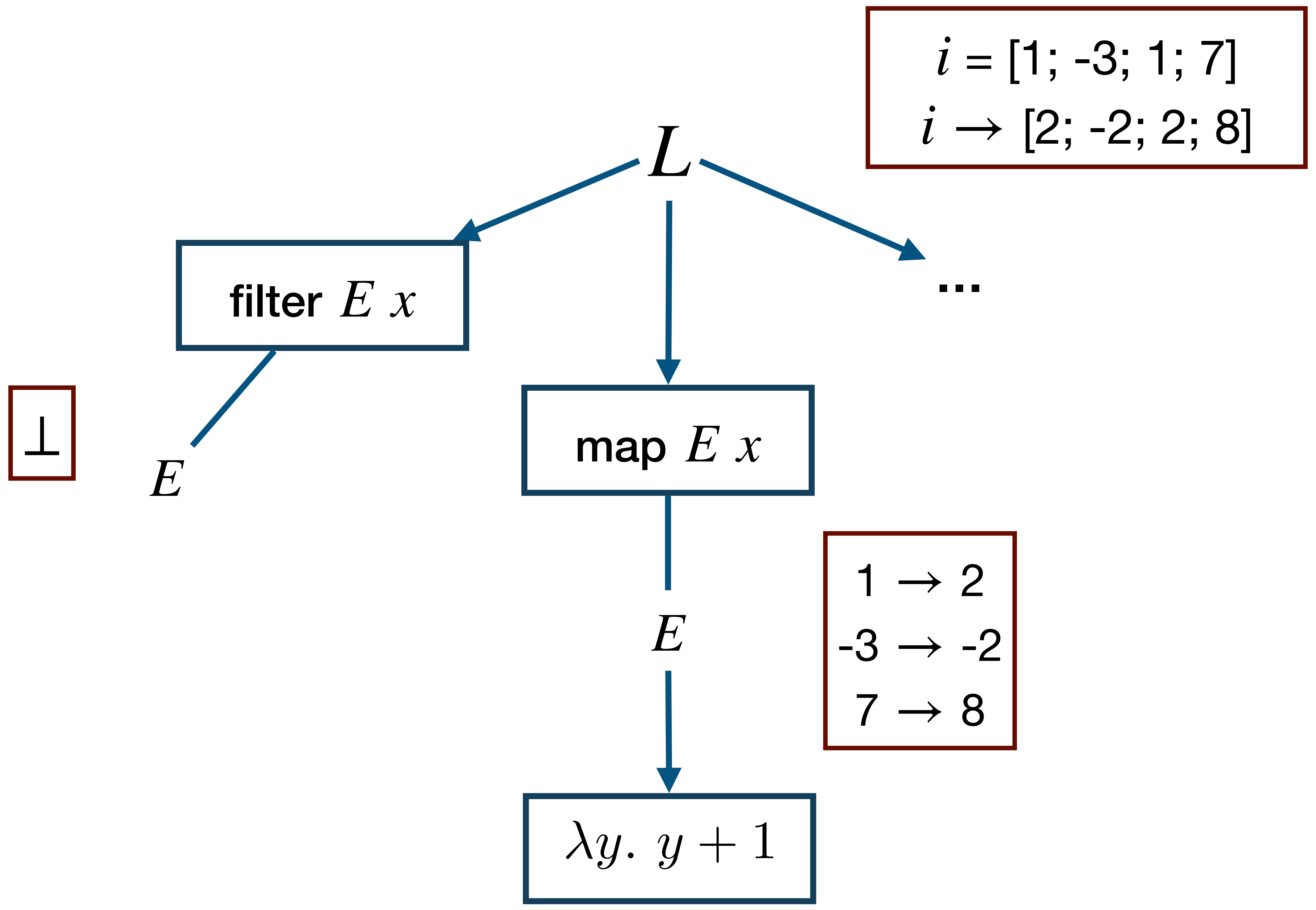
Examples

$$\texttt{map}\ (\lambda y.\ y + 1)\ \texttt{[1; -3; 1; 7]} = \texttt{[2; -2; 2; 8]}$$

$$\texttt{filter}\ (\lambda y.\ y \leq 0)\ \texttt{[1; -3; 1; 7]} = \texttt{[-3]}$$

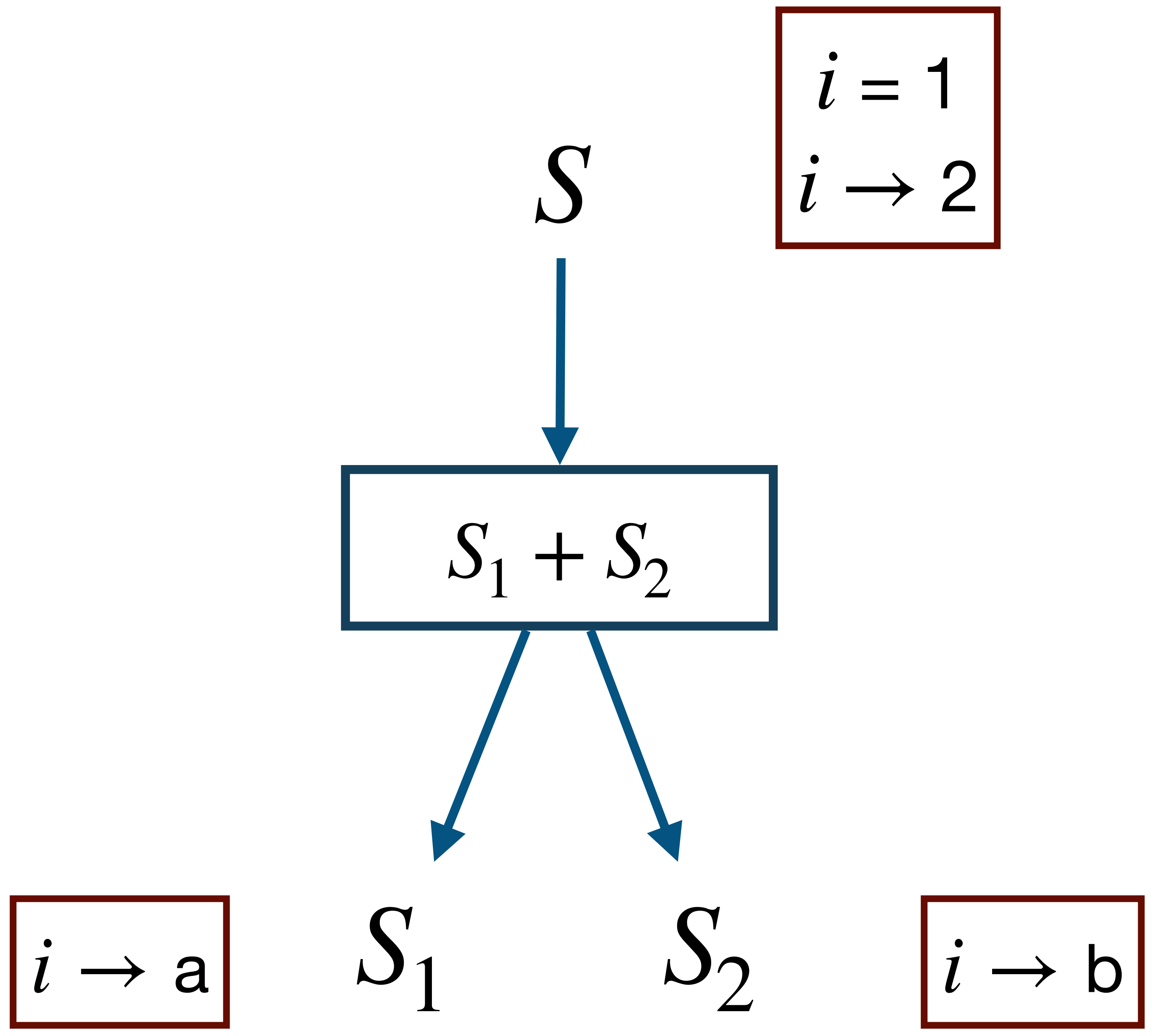
$$\texttt{fold}\ (\lambda y.\lambda z.\ y + z)\ \texttt{0}\ \texttt{[1; -3; 1; 7]} = 6$$

TDP for List Manipulation



Limitation of TDP

- Applicable only when the function is injective so that the inverse exists
- For example, (+) : int -> int -> int



What are possible values of a and b?

Summary

- Challenge: huge search space
- Equivalence reduction = discard **redundant** candidates
    - Applicable to top-down and bottom-up searches
- Top-down propagation = discard **unpromising** candidates
    - Applicable to top-down search when the inverse functions are computable