Inductive Learning

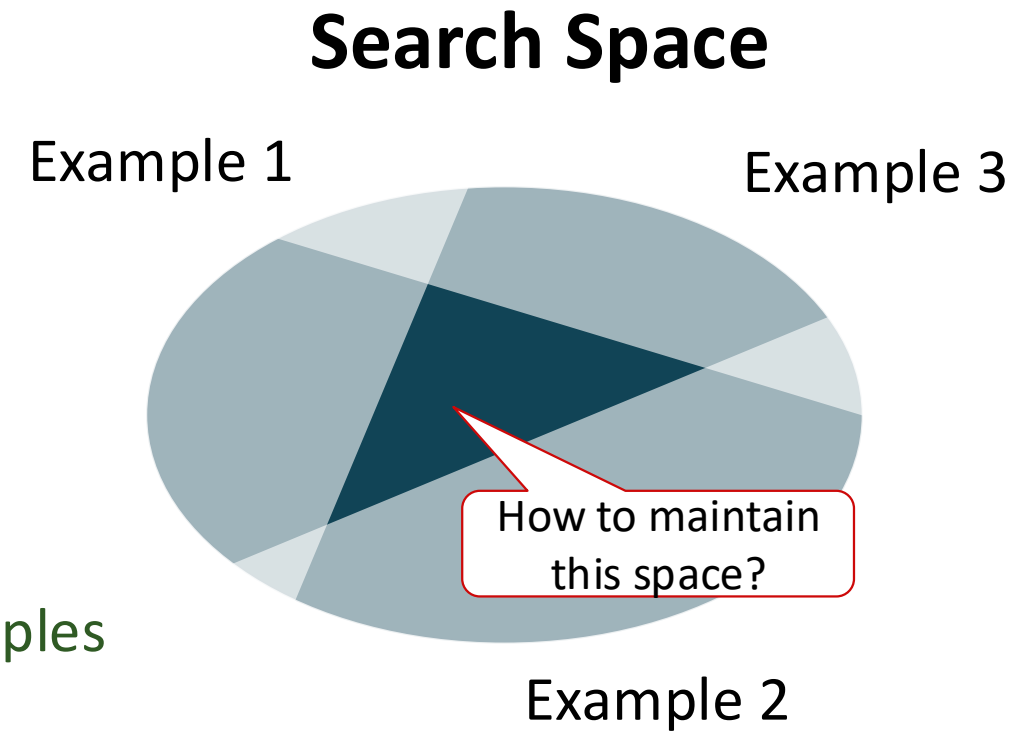
Is Sketch all we need?

Pros

- Very generic

Cons

- Very generic!
- E.g., not specialized for PBE/PBD
- The user may incrementally add more examples



PBE/PBD vs. inductive learning

1, 2, 4, 8, ??

1, 2, 4, 7, ??

PBE/PBD are examples of inductive learning

Most machine learning falls into this category as well

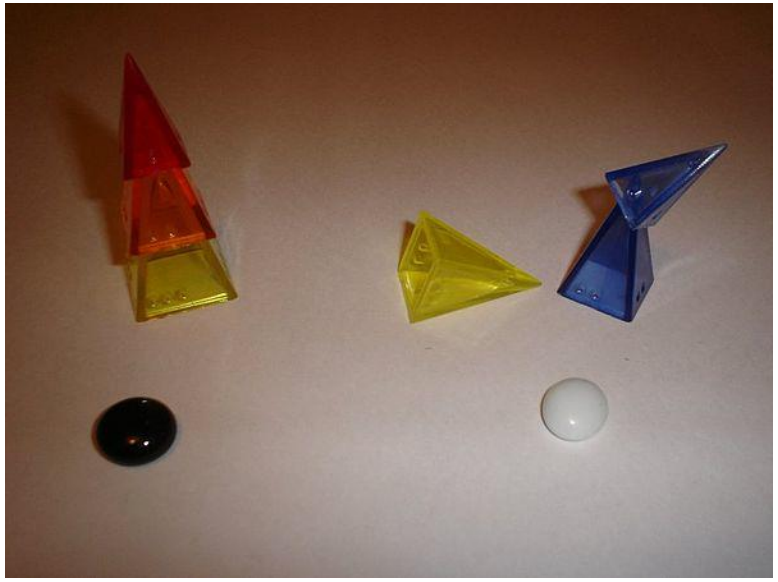
A little bit of history: who was the first?

Learning Structural Descriptions from Examples

Patrick Winston [1970]

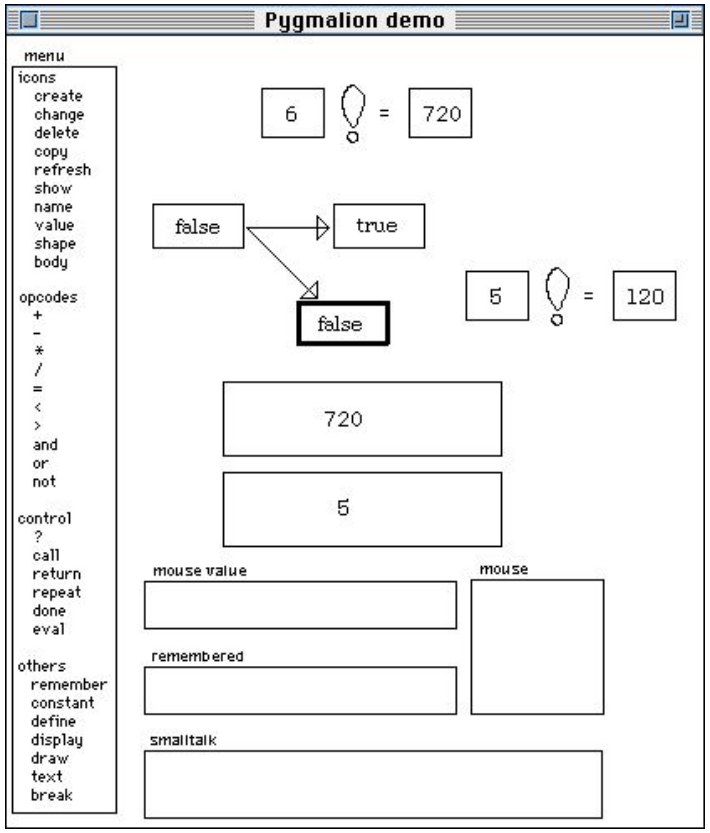
Explored the question of generalizing from a set of observations

Similar to the Zendo game:



Pygmalion [Smith 75]

- Real programming by demonstration
- http://acypher.com/wwid/Chapters/01Pygmalion.html



A little bit of history: inductive learning

Ad-hoc approaches → General inductive learning techniques

- Version-space generalization
- Inductive logic programming

Programming by Demonstration: An Inductive Learning Formulation*

Tessa A. Lau and **Daniel S. Weld**
Department of Computer Science and Engineering
University of Washington
Seattle, WA 98195-2350
October 7, 1998
{tlau, weld}@cs.washington.edu

ABSTRACT
Although Programming by Demonstration (PBD) has

- Applications that support macros allow users to record a fixed sequence of actions and later replay this

Tessa Lau

Version Space Generalization

Goto message #1, whose sender is “Dan” Copy its subject to the clipboard Paste from the clipboard into “File1”
Goto message #2, whose sender is “Tessa” Copy its subject to the clipboard Paste from the clipboard into “File1”

Act1	Arg1	Sender	Act2	Arg2	Act3	Arg3
Goto	1	Dan	Copy	Subj	Paste	File1
Goto	2	Tessa	Copy	Subj	Paste	File1

Act1	Arg1	Sender	Act2	Arg2	Act3	Arg3
Goto	?	?	Copy	Subj	Paste	File1

Inductive Logic Programming

Goto message #1, whose sender is “Dan” Copy its subject to the clipboard Paste from the clipboard into “File1”
Goto message #2, whose sender is “Tessa” Copy its subject to the clipboard Paste from the clipboard into “File1”

Action	Command	Arg	Message	Text	File	Sender	CurrentMesg	PrevCommand
a1	goto	arg1	mesg1			dan	null	null
a2	copy	arg2		subject			mesg1	goto
a3	paste	arg3			file1		mesg1	copy
a4	goto	arg4	mesg2			tessa	mesg1	paste
a5	copy	arg2		subject			mesg2	goto
a6	paste	arg3			file1		mesg2	copy
a7	goto	arg6	mesg3			oren	mesg2	paste
a8	copy	arg2		subject			mesg3	goto
a9	paste	arg3			file1		mesg3	copy

```
command(A,goto) :- prevcommand(A,paste).
command(A,copy) :- prevcommand(A,goto).
command(A,paste) :- prevcommand(A,copy).
command(A,goto) :- prevcommand(A,null).

argument(A,R):-prevcommand(A,goto), textofarg(R,T).
argument(A,R):-prevcommand(A,copy), fileofarg(R,F).
argument(A,R):-currentmesg(A,M), prevcommand(A,
              paste), mesgofarg(R,N), inseq(M,N).
argument(A,R):-currentmesg(A,null),
              mesgofarg(R,mesg1).
```

Version Space Generalization

Slides by Armando Solar-Lezama

Version Space Formulation

Hypothesis space H

- Space of possible functions $In \rightarrow Out$

Version Space $VS_{H,D} \subseteq H$

- H is the original hypothesis space
- D is a set of examples i_j, o_j
- $h \in VS_{H,D} \Leftrightarrow \forall\, i, o \in D \;\; h(i) = \mathrm{o}$

Hypothesis space provides *restriction bias*

- Defines what functions one is allowed to consider
- *Preference bias* needs to be provided independently

Partial Ordering of Hypotheses

Partial order $h_1 \sqsubseteq h_2$

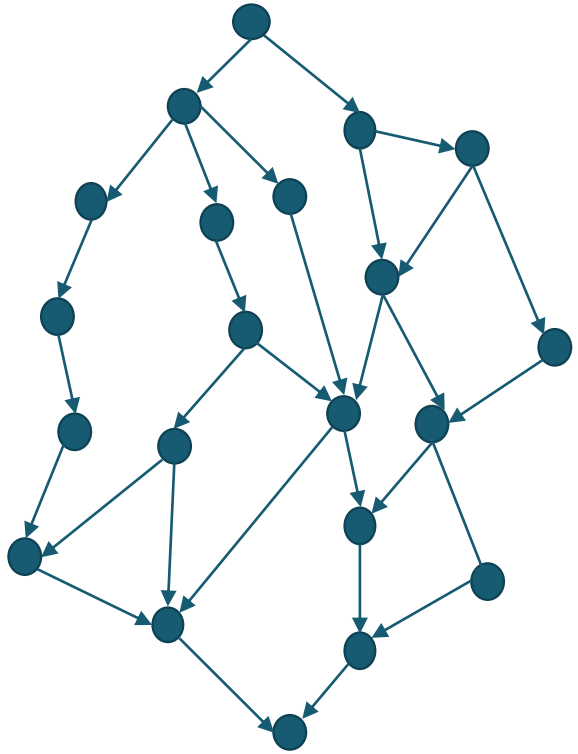
- h_2 is "better" than h_1

Ex: For boolean hypothesis

- "better" == more general
- $h_1 \sqsubseteq h_2 \Leftrightarrow (h_1 \Rightarrow h_2)$

For booleans, VS forms a lattice

- $h_1, h_2 \in VS \Rightarrow h_1 \sqcap h_2 = h_1 \wedge h_2 \in VS$



Most specific hypothesis that satisfies the observations

Partial Order

Set P

Partial order $\leq$ such that $\forall x, y, z \in P$

- $x \leq x$ (reflexive)
- $x \leq y$ and $y \leq x$ implies $x = y$ (asymmetric)
- $x \leq y$ and $y \leq z$ implies $x \leq z$ (transitive)

Can use partial order to define

- Upper and lower bounds
- Least upper bound
- Greatest lower bound

Upper Bound

If $S \subseteq P$ then

- $x \in P$ is an upper bound of S if $\forall y \in S. y \leq x$
- $x \in P$ is the least upper bound of S if
  - x is an upper bound of S, and
  - $x \leq y$ for all upper bounds y of S
- $\sqcup$ - join, least upper bound, lub, supremum, sup
  - $\sqcup S$ is the least upper bound of S
  - $x \sqcup y$ is the least upper bound of $\{x, y\}$
- Often written as $\sqcup$ as well

Lower Bound

If $S \subseteq P$ then

- $x \in P$ is a lower bound of S if $\forall y \in S. x \leq y$
- $x \in P$ is the greatest lower bound of S if
  - x is a lower bound of S, and
  - $y \leq x$ for all lower bounds y of S
- $\sqcap$ - meet, greatest lower bound, glb, infimum, inf
  - $\sqcap S$ is the greatest lower bound of S
  - $x \sqcap y$ is the greatest lower bound of $\{x, y\}$
- Often written as $\sqcap$ as well

Lattice

If $x \sqcap y$ and $x \sqcup y$ exist for all $x, y \in P$
then P is a lattice

If $\sqcap S$ and $\sqcup S$ exist for all $S \subseteq P$
then P is a complete lattice

All finite lattices are complete

Example of a lattice that is not complete

- Integers $\mathbb{Z}$
- For any $x, y \in \mathbb{Z}$, $x \sqcup y = \max(x, y)$, $x \sqcap y = \min(x, y)$
- But $\sqcup \mathbb{Z}$ and $\sqcap \mathbb{Z}$ do not exist
- $\mathbb{Z} \cup \{+\infty, -\infty\}$ is a complete lattice

How to (succinctly) represent a version space?

Boundary set representable

You can represent a VS by the pair (G,S) where

- G is most general hypothesis (i.e. $\top$)
- S is the most specific (i.e. $\bot$)

Applies in general when hypothesis space is partially ordered and version space is a lattice

Update

$U(VS, d) = \{p \in VS \mid p(i) = o \text{ where } d = (i, o)\}$

- Subset of a version space satisfying a new example d

Ex: For boolean HS

- $VS = (G, S)$
- If $d = (i, \text{true})$

  $U(VS, d) = (G, S \vee \lambda x.\text{if } x = i \text{ then true else false})$
- If $d = (i, \text{false})$

  $U(VS, d) = (G \wedge \lambda x.\text{if } x = i \text{ then false else true}, S)$

Example: FindSuffix

Programming by Demonstration Using Version Space Algebra

TESSA LAU tessalau@us.ibm.com
IBM T.J. Watson Research, P.O. Box 704, Yorktown Heights, NY 10598, USA

STEVEN A. WOLFMAN wolf@cs.washington.edu
PEDRO DOMINGOS pedrod@cs.washington.edu
DANIEL S. WELD weld@cs.washington.edu
Department of Computer Science & Engineering, University of Washington, Seattle, WA 98195, USA

FS_T: move to the position right before the next occurrence of T.

We shall go on to the end. We |shall fight in France, we |shall fight on the seas and oceans, we shall fight with growing confidence and growing strength in the air,…

$FS_{""}$

$FS_{"s"}$

$FS_{"sha"}$

$FS_{"shall"}$

$FS_{"shall\ fight"}$

$FS_{"shall\ fight\ on"}$

$FS_{"shall\ fight\ on\ the\ seas\ and\ oceans,we\ shall\ fight\ldots"}$

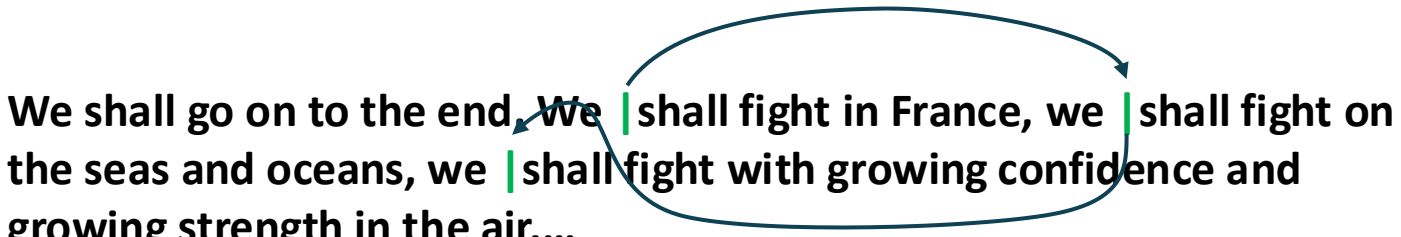
Example: FindSuffix

Programming by Demonstration Using Version Space Algebra

TESSA LAU tessalau@us.ibm.com
IBM T.J. Watson Research, P.O. Box 704, Yorktown Heights, NY 10598, USA

STEVEN A. WOLFMAN wolf@cs.washington.edu
PEDRO DOMINGOS pedrod@cs.washington.edu
DANIEL S. WELD weld@cs.washington.edu
Department of Computer Science & Engineering, University of Washington, Seattle, WA 98195, USA

FS_T: move to the position right before the next occurrence of T.



$FS_{""}$

$FS_{"s"}$

$FS_{"sha"}$

$FS_{"shall"}$

$FS_{"shall\ fight"}$

$FS_{"shall\ fight\ on"}$

$FS_{"shall\ fight\ on\ the\ seas\ and\ oceans,we\ shall\ fight\ldots"}$

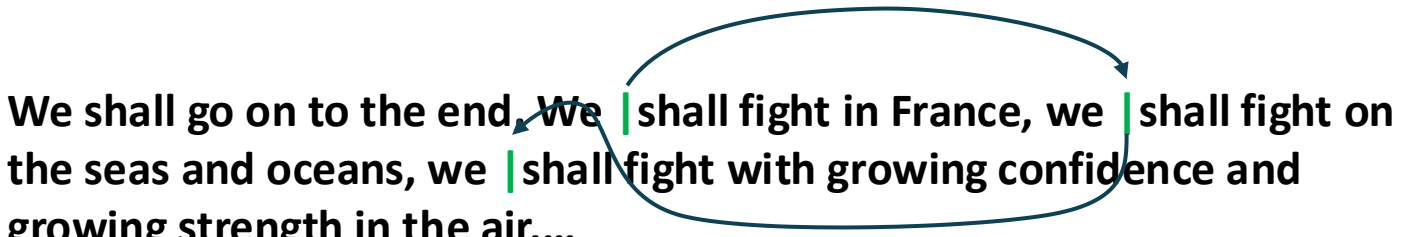
Example: FindSuffix

Programming by Demonstration Using Version Space Algebra

TESSA LAU tessalau@us.ibm.com
IBM T.J. Watson Research, P.O. Box 704, Yorktown Heights, NY 10598, USA

STEVEN A. WOLFMAN wolf@cs.washington.edu
PEDRO DOMINGOS pedrod@cs.washington.edu
DANIEL S. WELD weld@cs.washington.edu
Department of Computer Science & Engineering, University of Washington, Seattle, WA 98195, USA

FS_T: move to the position right before the next occurrence of T.



$FS_{""}$

$FS_{"s"}$

$FS_{"sha"}$

$FS_{"shall"}$

$FS_{"shall\ fight"}$

$FS_{"shall\ fight\ on"}$

$FS_{"shall\ fight\ on\ the\ seas\ and\ oceans,we\ shall\ fight\ldots"}$

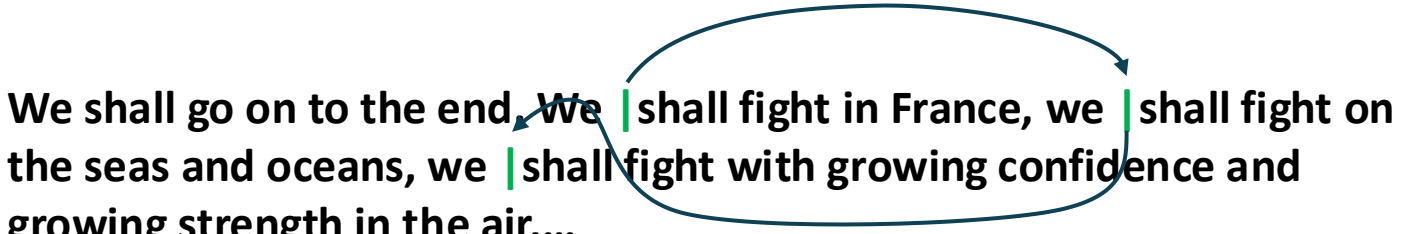
Example: FindSuffix

Programming by Demonstration Using Version Space Algebra

TESSA LAU tessalau@us.ibm.com
IBM T.J. Watson Research, P.O. Box 704, Yorktown Heights, NY 10598, USA

STEVEN A. WOLFMAN wolf@cs.washington.edu
PEDRO DOMINGOS pedrod@cs.washington.edu
DANIEL S. WELD weld@cs.washington.edu
Department of Computer Science & Engineering, University of Washington, Seattle, WA 98195, USA

FS_T: move to the position right before the next occurrence of T.



$FS_{""}$

$FS_{"s"}$

$FS_{"sha"}$

$FS_{"shall"}$

$FS_{"shall\ fight"}$

$FS_{"shall\ fight\ on"}$

$FS_{"shall\ fight\ on\ the\ seas\ and\ oceans,we\ shall\ fight…"}$

What if not boundary set representable?

If your hypothesis space is partially ordered and your VS are boundary set representable, you can represent and search very efficiently

If they are not?

Break them down into simpler hypothesis spaces!

Union

Union of BSR version spaces:

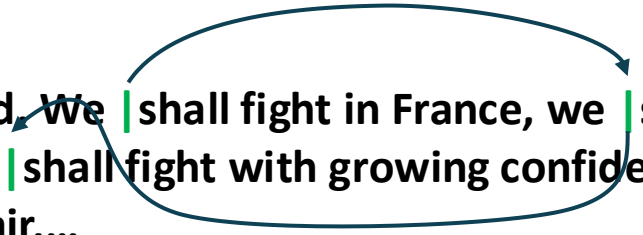
$$VS_{H_1 D} \cup VS_{H_2 D} = VS_{H_1 \cup H_2 D}$$

FindSuffix U FindPrefix

Programming by Demonstration Using Version Space Algebra

TESSA LAU tessalau@us.ibm.com
IBM T.J. Watson Research, P.O. Box 704, Yorktown Heights, NY 10598, USA

STEVEN A. WOLFMAN wolf@cs.washington.edu
PEDRO DOMINGOS pedrod@cs.washington.edu
DANIEL S. WELD weld@cs.washington.edu
Department of Computer Science & Engineering, University of Washington, Seattle, WA 98195, USA



We shall go on to the end. We | shall fight in France, we | shall fight on the seas and oceans, we | shall fight with growing confidence and growing strength in the air,…

FS(“sh”-”shall fight ”)

U

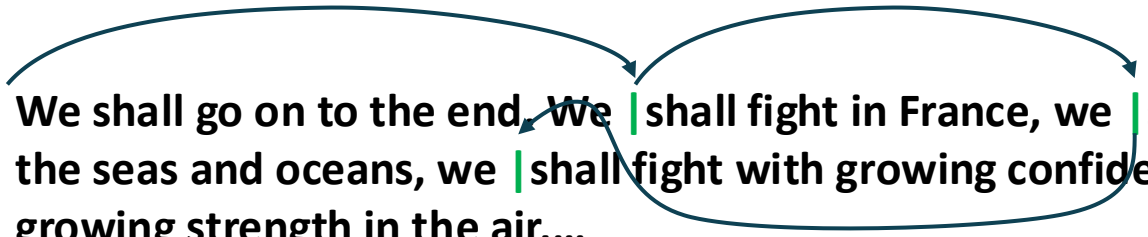
FP(“we “ – “, we”)

FindSuffix U FindPrefix

Programming by Demonstration Using Version Space Algebra

TESSA LAU tessalau@us.ibm.com
IBM T.J. Watson Research, P.O. Box 704, Yorktown Heights, NY 10598, USA

STEVEN A. WOLFMAN wolf@cs.washington.edu
PEDRO DOMINGOS pedrod@cs.washington.edu
DANIEL S. WELD weld@cs.washington.edu
Department of Computer Science & Engineering, University of Washington, Seattle, WA 98195, USA



FS(“sh”-”shall fight ”)

U

Ø

Join

$$VS_{H_1D_1} \bowtie VS_{H_2D_2} = \{\langle h_1, h_2\rangle \mid h_1 \in VS_{H_1D_1}, h_2 \in VS_{H_2D_2}, C(\langle h_1, h_2\rangle, D)\}$$

- Where $D_1 = \{d_1^i\}_{i=0..n}$ and $D_2 = \{d_2^i\}_{i=0..n}$ and $D = \{\langle d_1^i, d_2^i\rangle\}_{i=0..n}$
- $C(\langle h_1, h_2\rangle, D)$ means that $\langle h_1, h_2\rangle$ is consistent with the input-output pairs in D

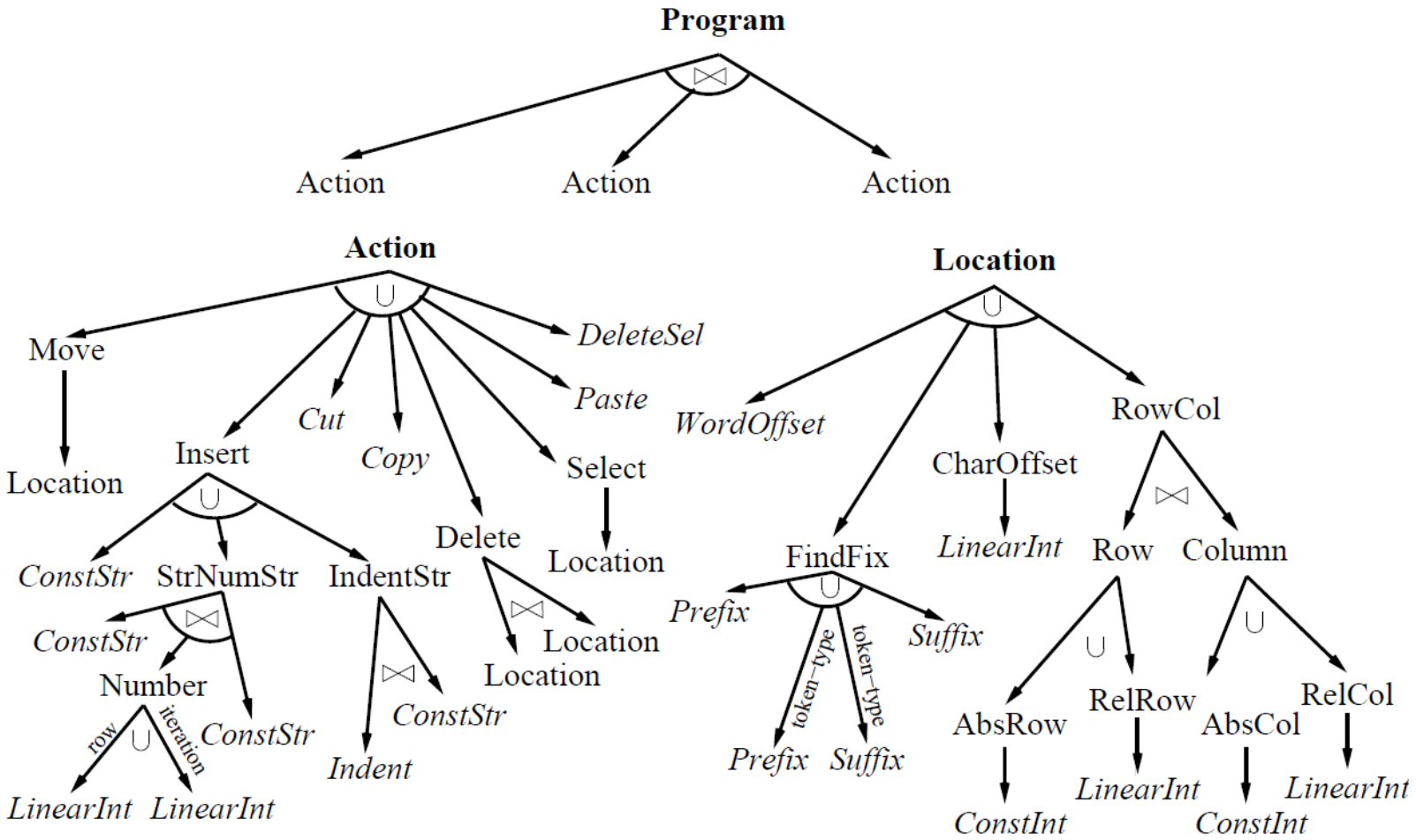
What does $\langle h_1, h_2\rangle$ mean? What about $\langle d_1, d_2\rangle$?

- Pair
- Composition $\langle h_1, h_2\rangle = h_1 \circ h_2$ and $\langle d_1, d_2\rangle = (d_1.in, d_2.out)$

Independent join: C is unnecessary

- It's a property of $\langle .,. \rangle$
- True for pair, not for composition

SMARTedit version space



Before FlashFill

- Detect failure and fail gracefully
- Make it easy to correct the system
- Encourage trust by presenting a model users can understand
- Enable partial automation

Why PBD systems fail: Lessons learned for usable AI

Tessa Lau
IBM Almaden Research Center
650 Harry Road
San Jose, CA 95120 USA
tessalau@us.ibm.com

Abstract
Programming by demonstration systems have long attempted to make it possible for people to program computers without writing code. However, while these systems have resulted in many publications in AI venues, none of the technologies have yet achieved widespread adoption. Usability remains a critical barrier to their success. Based on lessons learned from three different programming by demonstration systems, we present a a set of guidelines to consider when designing usable AI-based systems.

FlashFill

Slides by Gulwani, Singh, and Solar-Lezama

Core language

Trace expr e	:=	`Concatenate`($f_1, \ldots, f_n$) | f
Atomic expr f	:=	`ConstStr`(s) | `SubStr`(v_i, p_1, p_2) | Loop($\lambda w : e$)
Position p	:=	`CPos`(k) | Pos(r_1, r_2, c)
Integer expr c	:=	$k \mid k_1 w + k_2$
Regular expr r	:=	`TokenSeq`($T_1, \ldots, T_n$) | T | ε

Additional shorthand

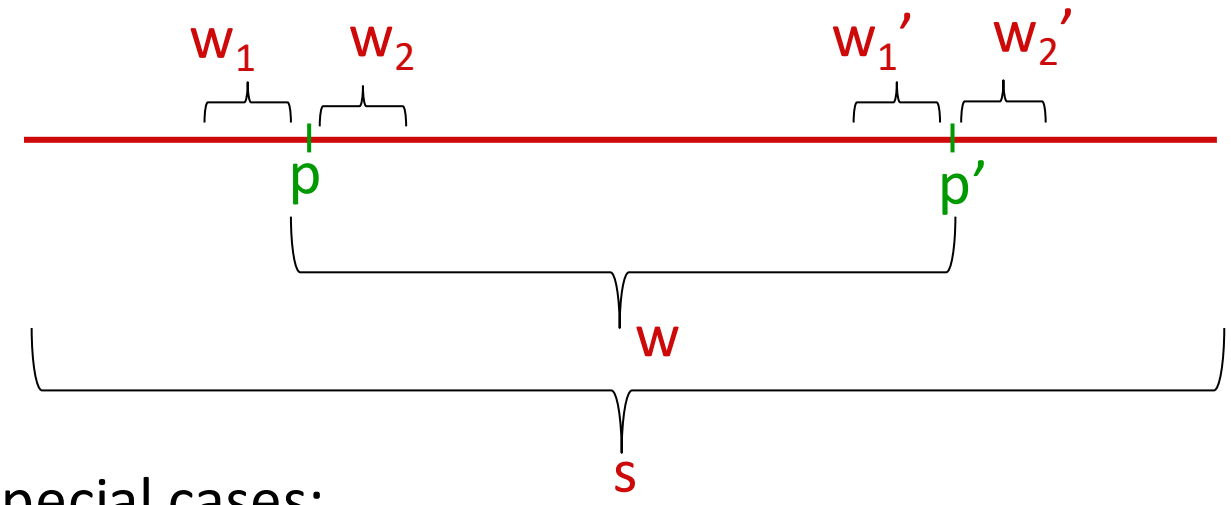
$$SubStr2(v_i, r, k) = SubsStr(v_i, Pos(\epsilon, r, k), Pos(r, \epsilon, k))$$

- k^{th} occurrence of regular expression r in v_i

Substring Operator

Let w = SubString(s, p, p')

where p = Pos(r_1, r_2, k) and p' = Pos(r_1', r_2', k')



r_1 matches w_1
r_2 matches w_2

r_1' matches w_1'
r_2' matches w_2'

Two special cases:

- $r_1 = r_2' = \epsilon$: This describes the substring
- $r_2 = r_1' = \epsilon$: This describes boundaries around the substring

The general case allows for the combination of the two and is thus a very powerful operator!

Additional Control Structure

$$
\begin{array}{rcl}
\text{String program } P & := & \texttt{Switch } ((\mathbf{b}_1, e_1), \ldots, (b_n, e_n)) \mid e \\
\text{Boolean condition b} & := & d_1 \vee \ldots \vee d_n \\
\text{Conjunction d} & := & \pi_1 \wedge \ldots \wedge \pi_n \\
\text{Predicate } \pi & := & \texttt{Match}(v_i, r, k) \mid \neg\, \texttt{Match}(v_i, r, k)
\end{array}
$$

Syntactic String Transformations: Example

Input v_1	Output
(425)-706-7709	425-706-7709
510.220.5586	510-220-5586
235 7654	425-235-7654
745-8139	425-745-8139

$Switch((b_1, e_1), (b_2, e_2))$, where

$$b_1 \equiv Match(v_1, NumTok, 3), \qquad b_2 \equiv \neg Match(v_1, NumTok, 3),$$

$$e_1 \equiv Concatenate\begin{pmatrix} SubStr2(v_1, NumTok, 1), \\ ConstStr(" - "), \\ SubStr2(v_1, NumTok, 2), \\ ConstStr(" - "), \\ SubStr2(v_1, NumTok, 3) \end{pmatrix} \quad e_2 \equiv Concatenate\begin{pmatrix} \text{ConstStr("425–")}, \\ ConstStr(" - "), \\ SubStr2(v_1, NumTok, 1), \\ ConstStr(" - "), \\ SubStr2(v_1, NumTok, 2) \end{pmatrix}$$

How does it work

Reuse ideas from version space algebra

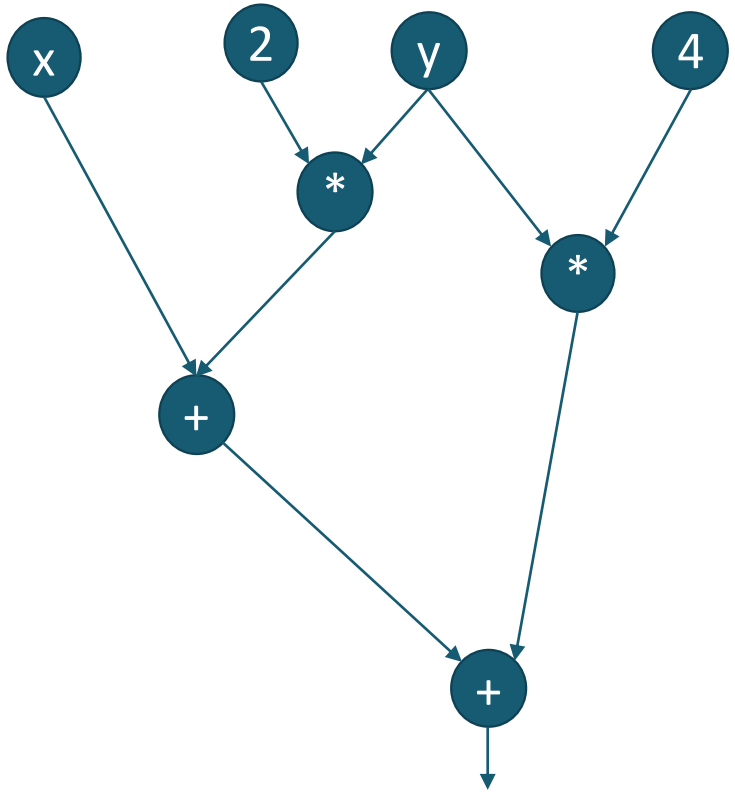
- Start with simple version spaces
- Define combinators to construct complex version spaces from simple ones

New twist:

- Move beyond simple lattice representations

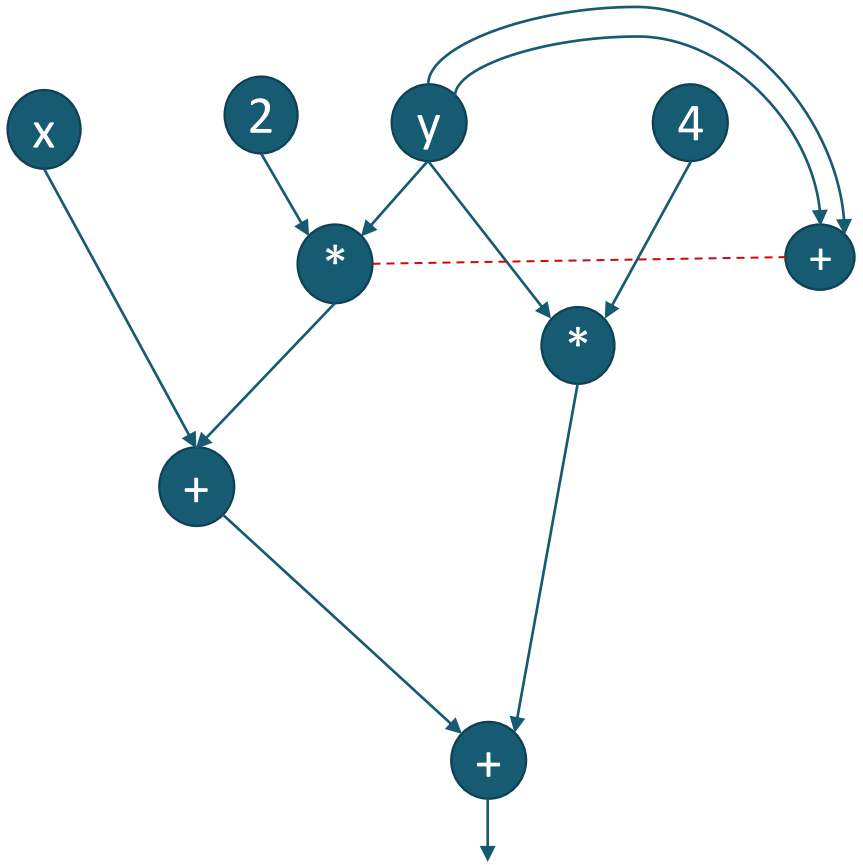
E-Graph

(x+2*y)+4*y



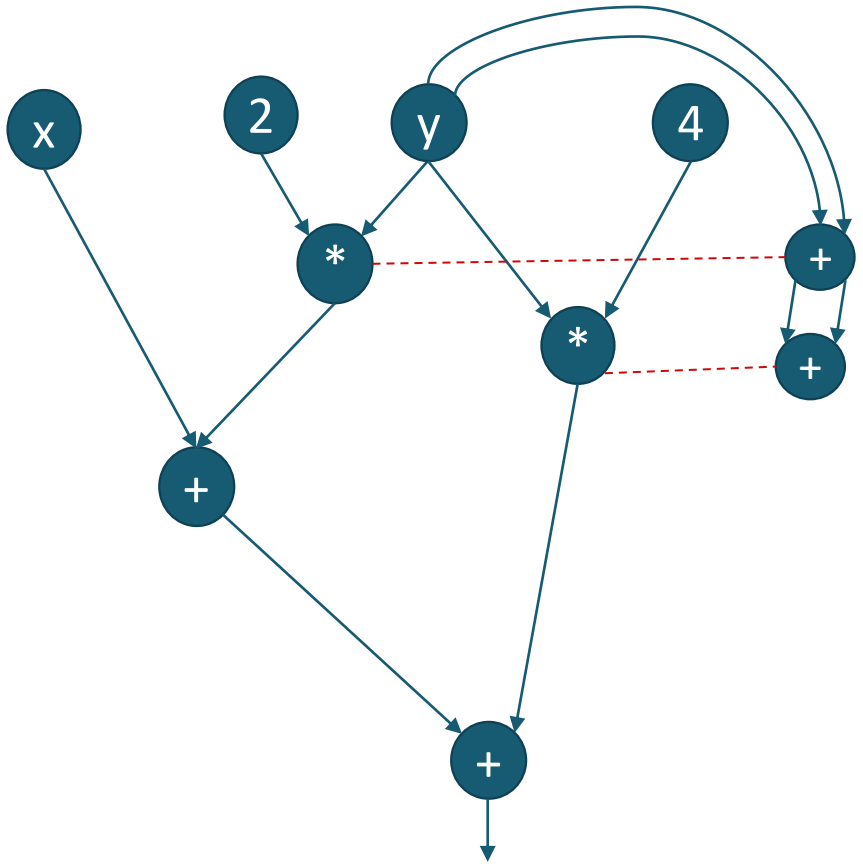
E-Graph

(x+2*y)+4*y



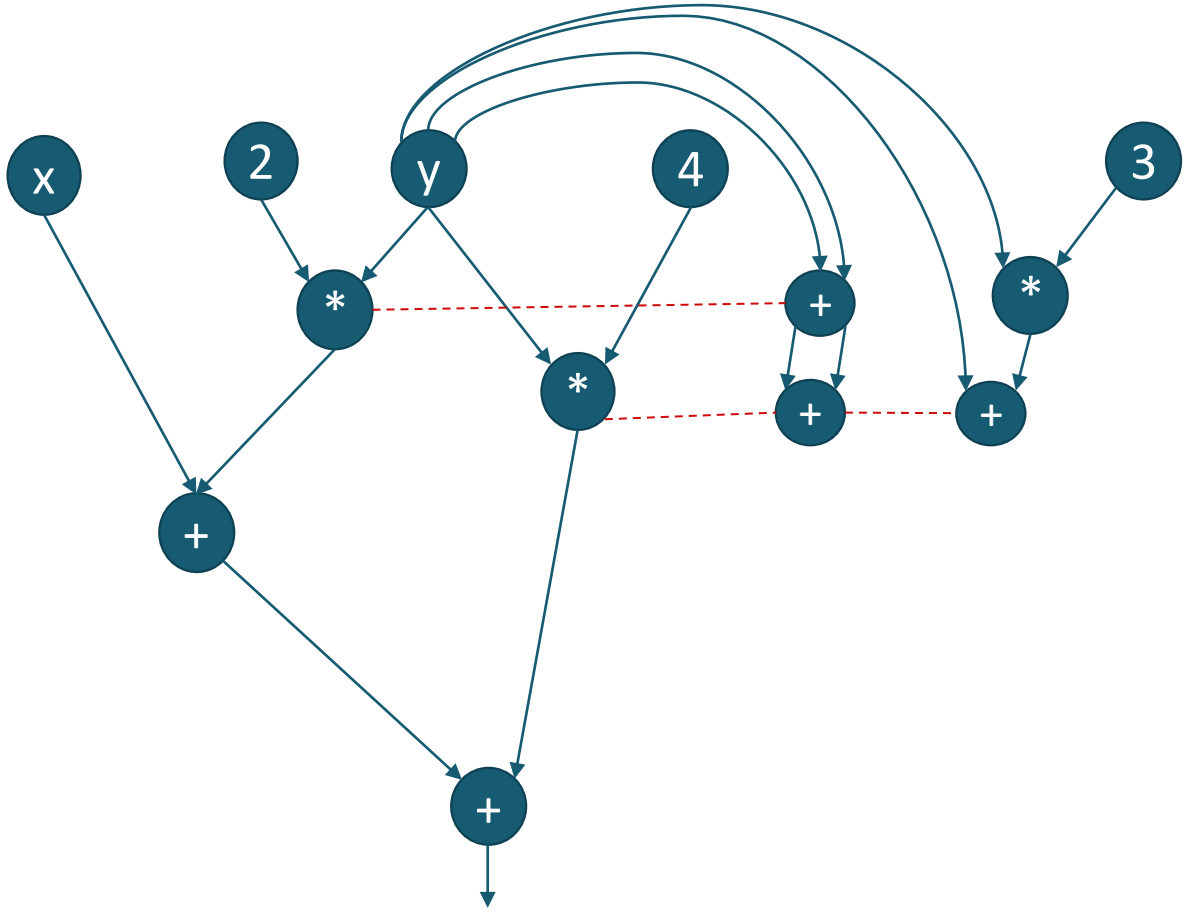
E-Graph

(x+2*y)+4*y



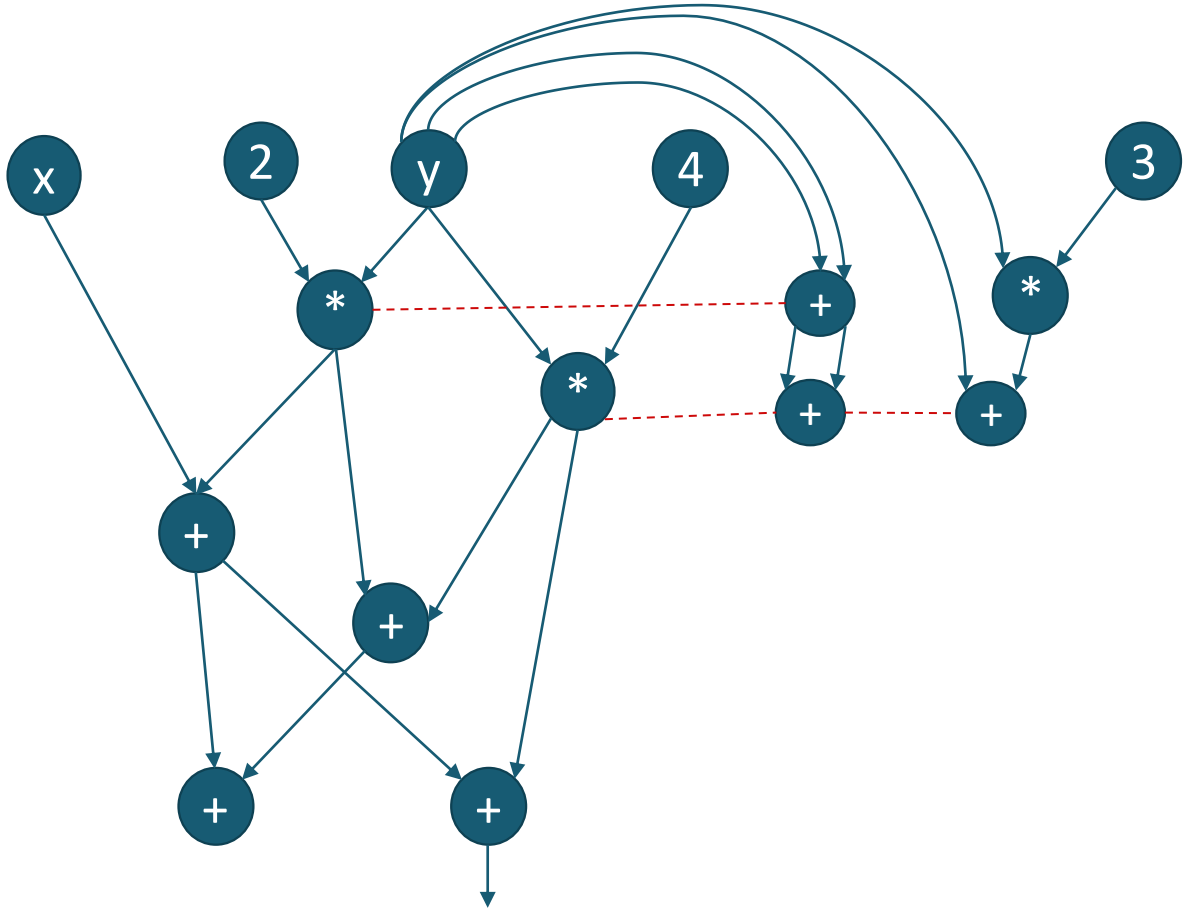
E-Graph

(x+2*y)+4*y



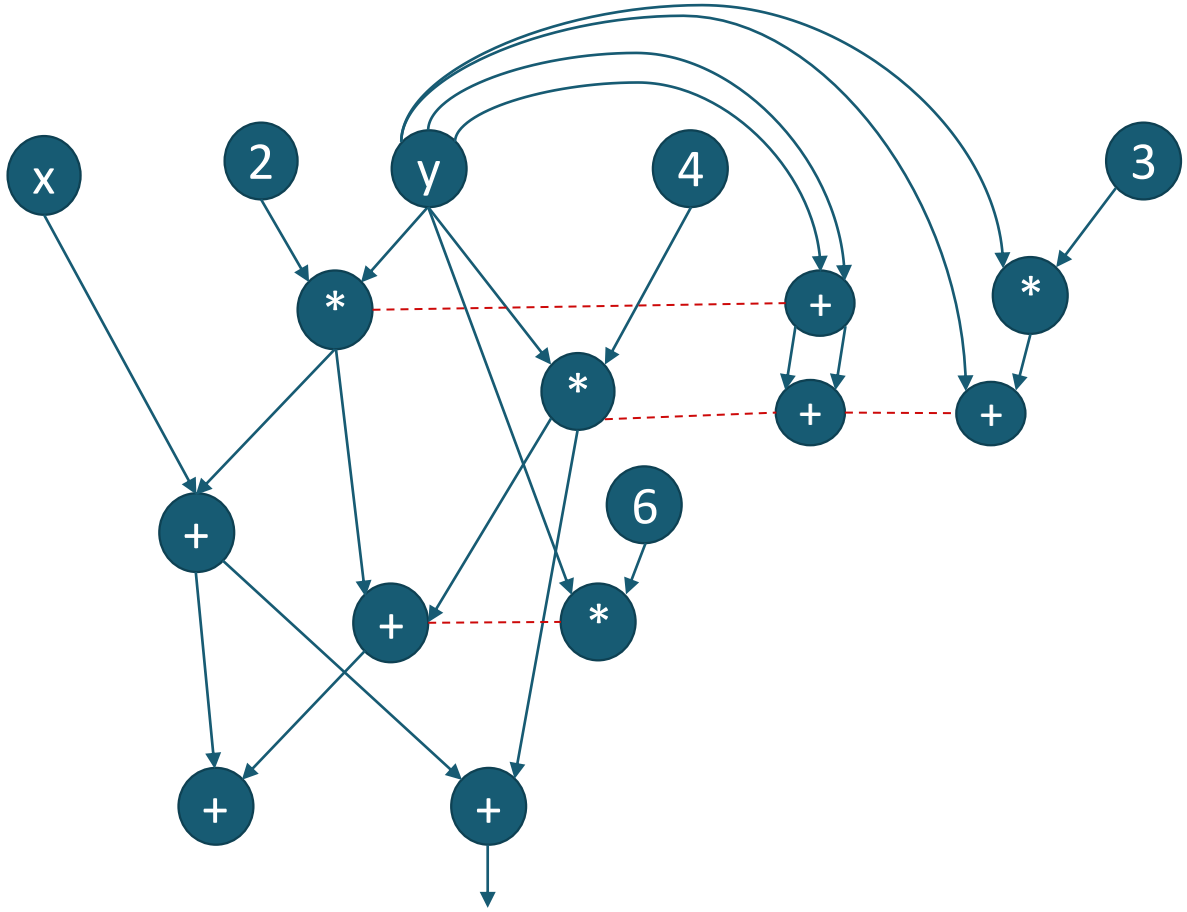
E-Graph

(x+2*y)+4*y



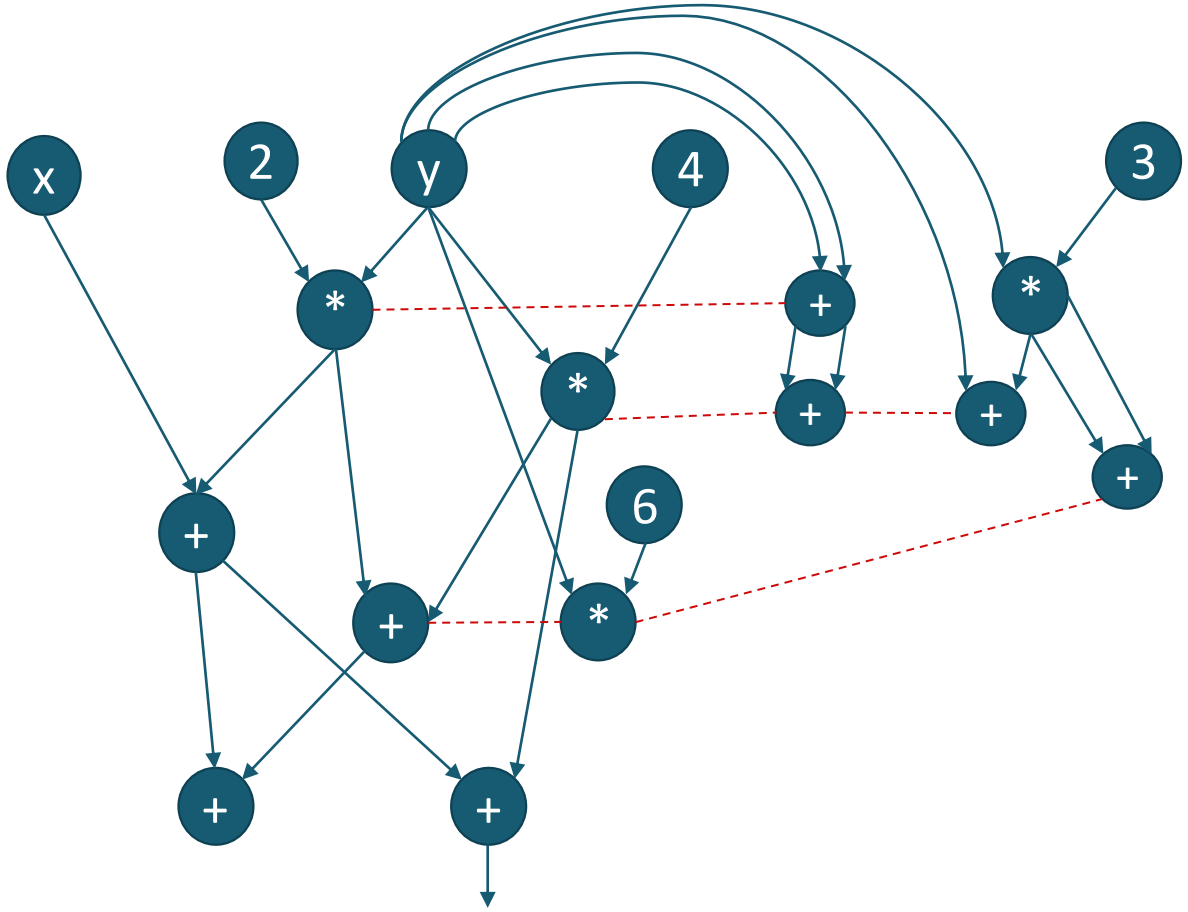
E-Graph

(x+2*y)+4*y



E-Graph

(x+2*y)+4*y



Guarded Expressions

D is a set of examples i_j, o_j H corresponds to trace expressions

Compute $S_j = VS_{(i_j o_j)H}$ and $S = VS_{DH} = \sqcap S_j$

Solution $= S \cup Switch\left(\left(p_i, VS_{D_i H}\right)\right)$

- Either you can find a solution to D in H, or you can split D and find solutions for subsets $D_i \subset D$
- Partition into subsets D_i must satisfy the following properties:
  - $VS_{D_i H} \neq \emptyset$
  - Partition is minimial
  - You can learn predicates p_i to create the partitions

Learning Trace Expressions

Full Name	Title Name
Rob Miller	Dr. Rob
Saman Amarasinghe	
Sumit Gulwani	
Armando Solar-Lezama	
Martin Rinard	

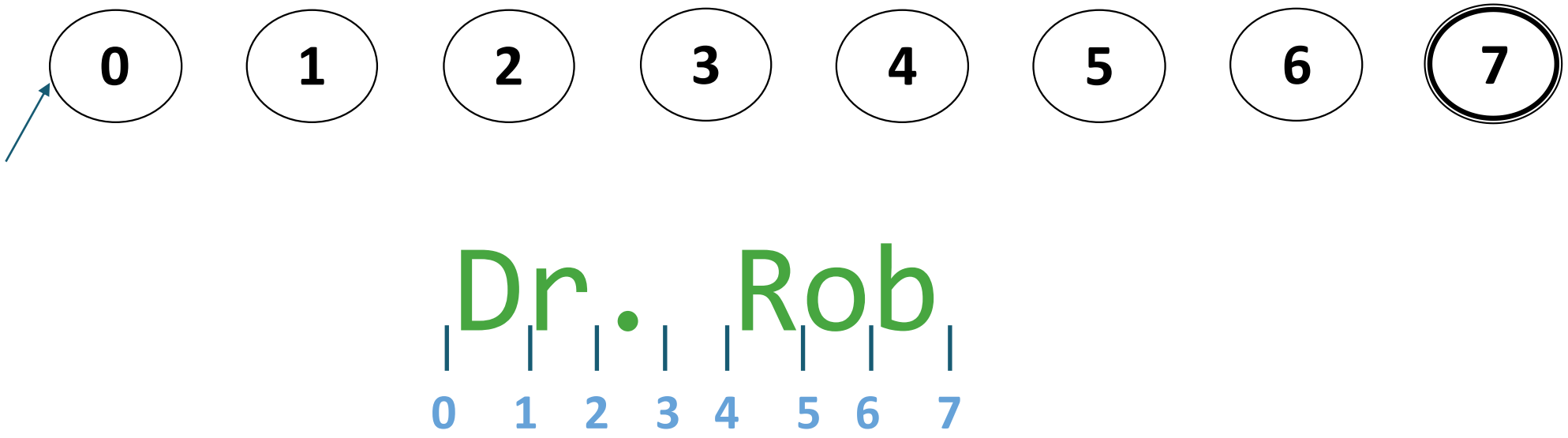
Learning Trace Expressions

Top level is always concatenation

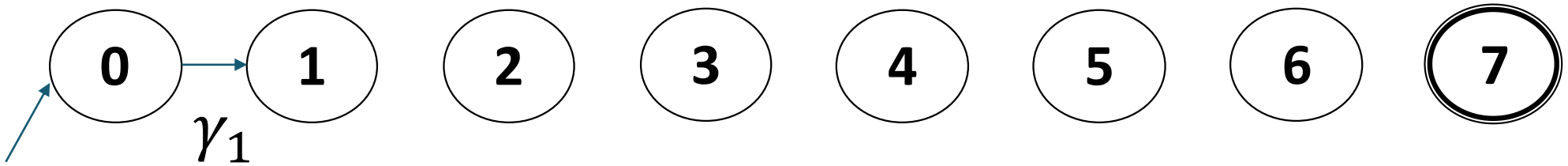
Concat Expression (Associative)

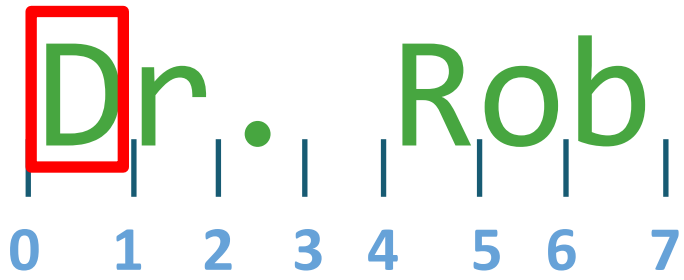
Rob Miller → Dr. Rob

Concat Expression (Associative)

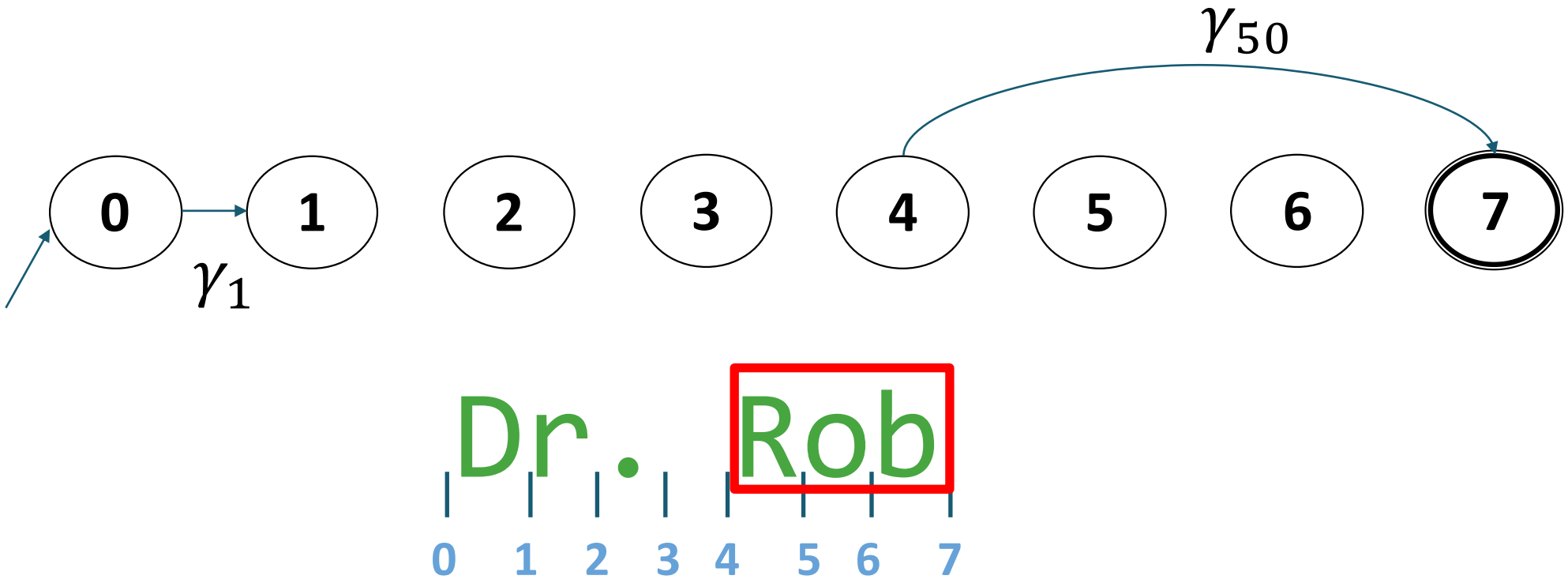


Concat Expression (Associative)

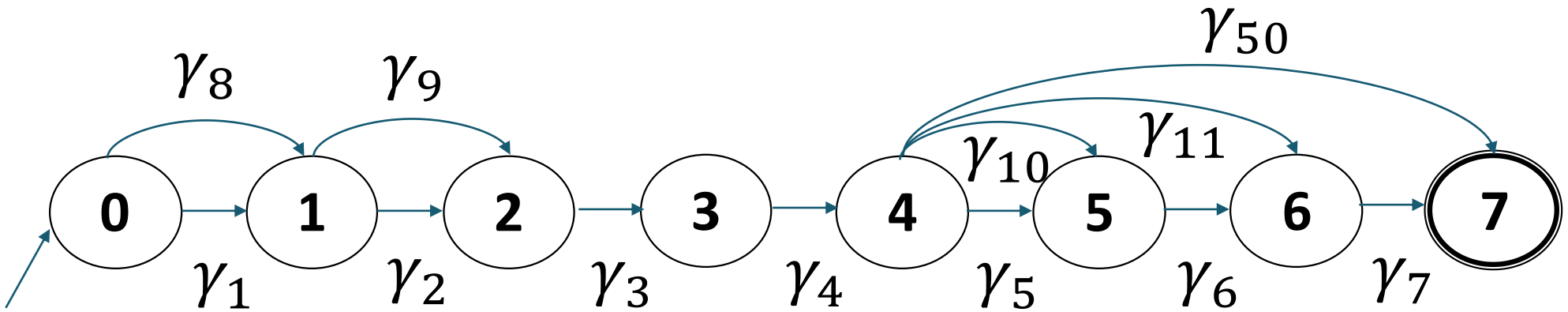




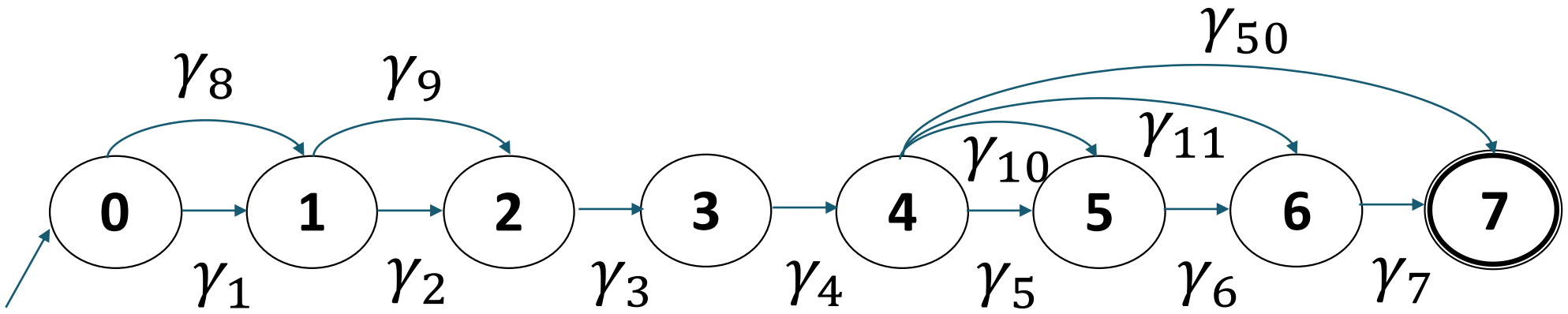
Concat Expression (Associative)



Concat Expression (Associative)



DAG-based Sharing



Any Path is a valid program

$\gamma_1 \cdot \gamma_2 \cdot \gamma_3 \cdot \gamma_4 \cdot \gamma_5 \cdot \gamma_6 \cdot \gamma_7$

$\gamma_8 \cdot \gamma_2 \cdot \gamma_3 \cdot \gamma_4 \cdot \gamma_5 \cdot \gamma_6 \cdot \gamma_7$

$\gamma_8 \cdot \gamma_2 \cdot \gamma_3 \cdot \gamma_4 \cdot \gamma_{50}$

Exponential Number of paths

Learning atomic expressions

Can only be constants, substring or loop expressions

Constants are trivial to learn

Substring can be factored!

Substring Expression

$145.67 → 145.67

1 7

Substr(left, right)

1 7

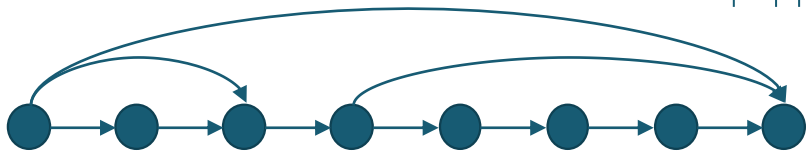
{ dollar, ϵ, 1
dollar, ϵ, -1
ϵ , decimal, -1
ϵ , number, 1
.....
constant 1 }

{ ϵ , End of line, 1
alphanumeric, ϵ, −1
alphanumeric, ϵ, 2
decimal, ϵ, −1
.....
constant 7 }

Intersection

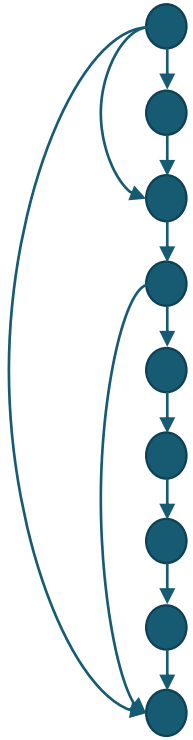
Armando Solar

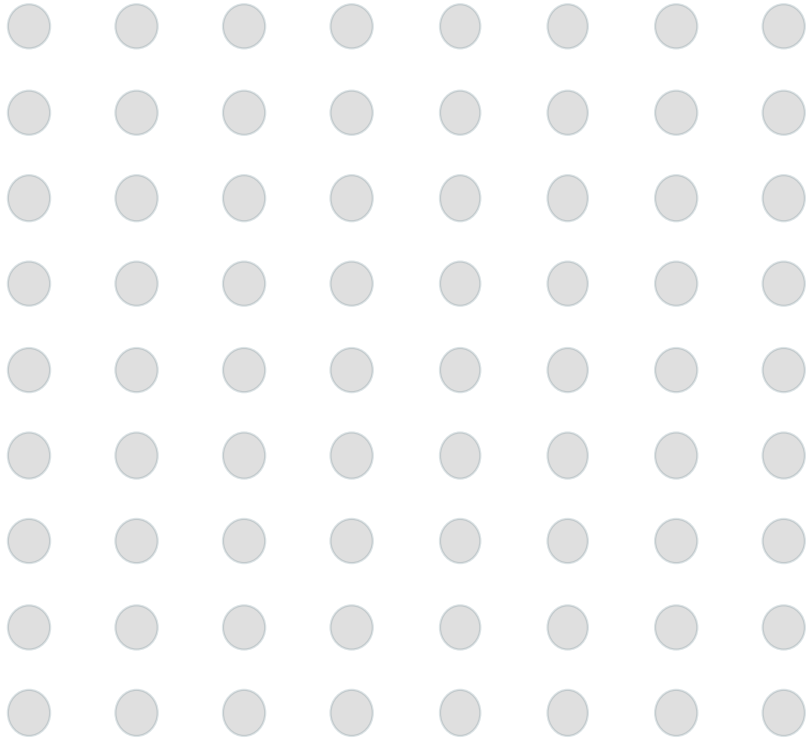
A. Solar



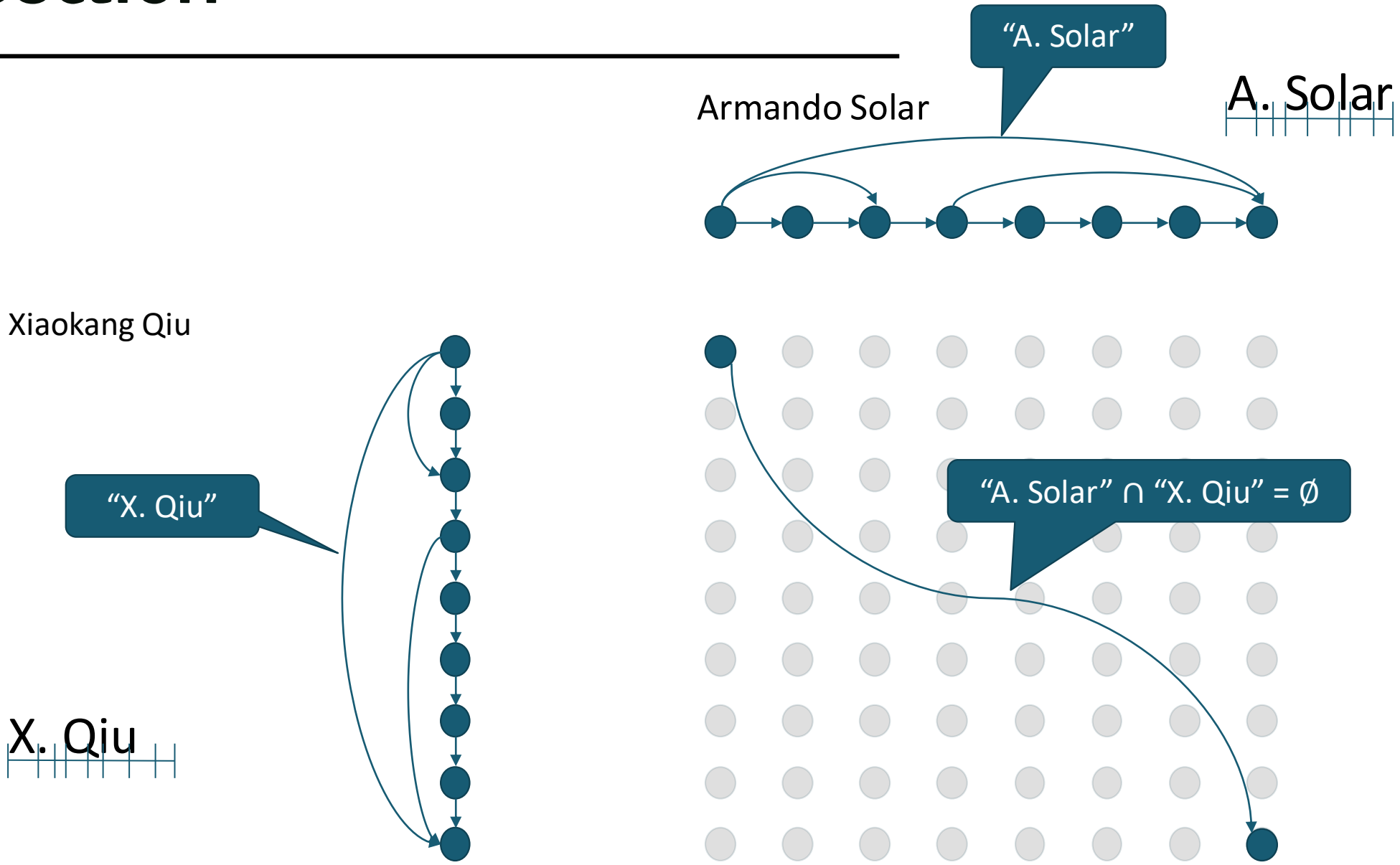
Xiaokang Qiu

X. Qiu

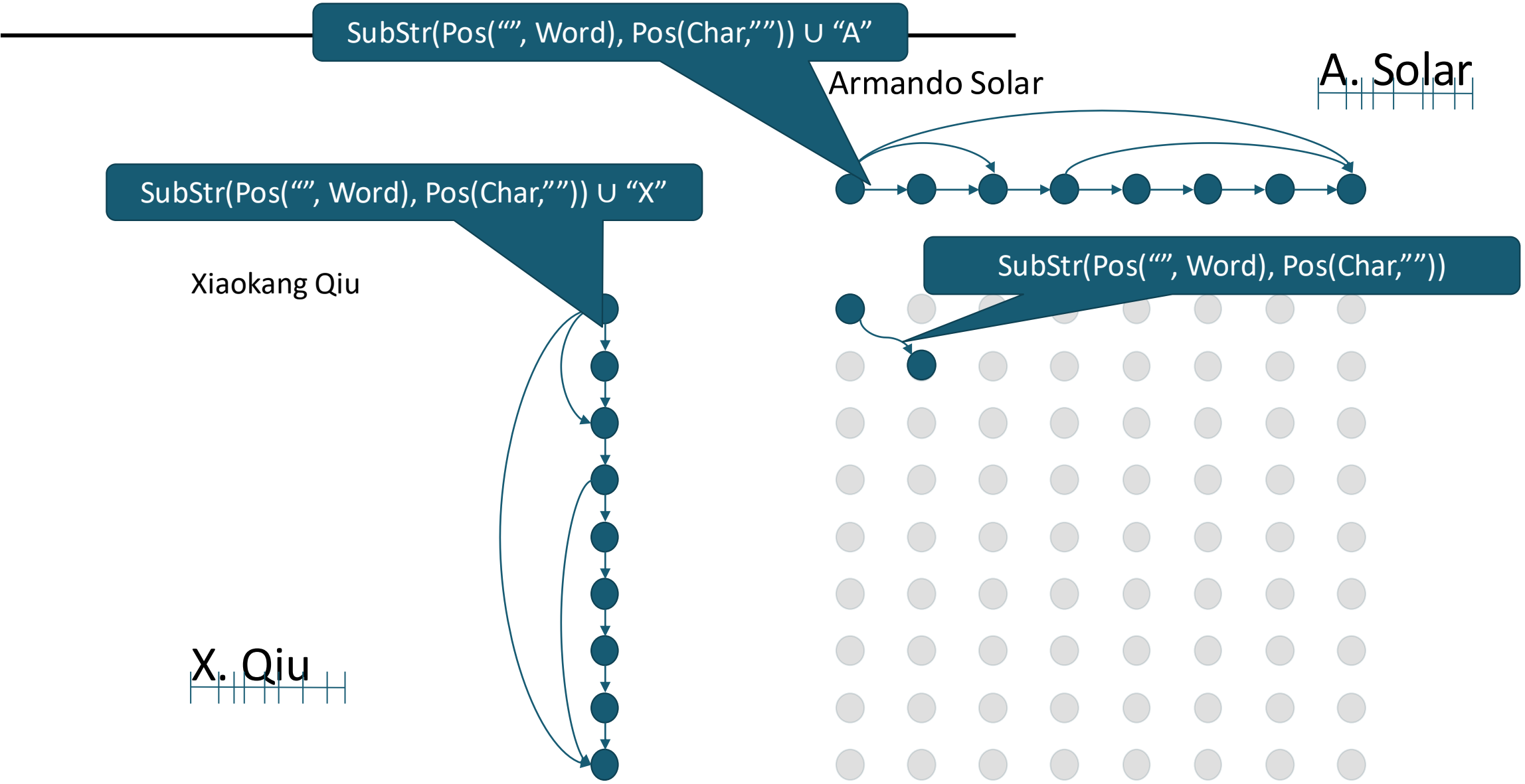




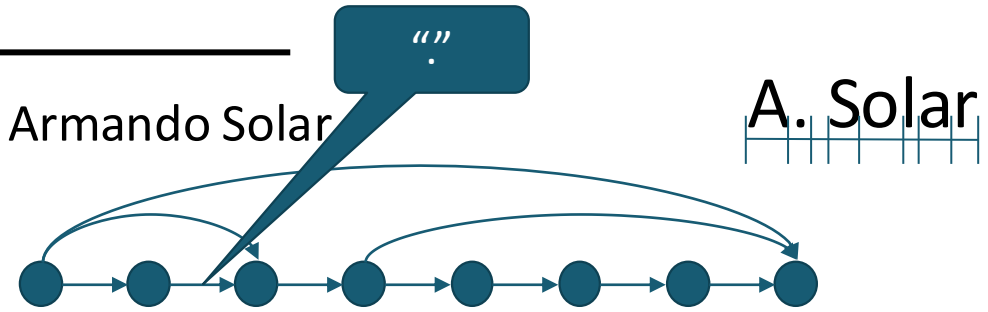
Intersection



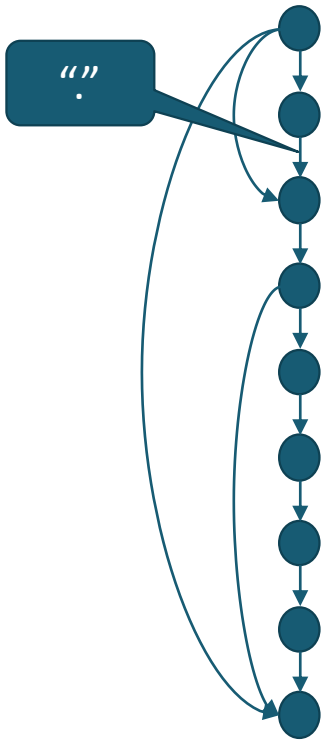
Intersection



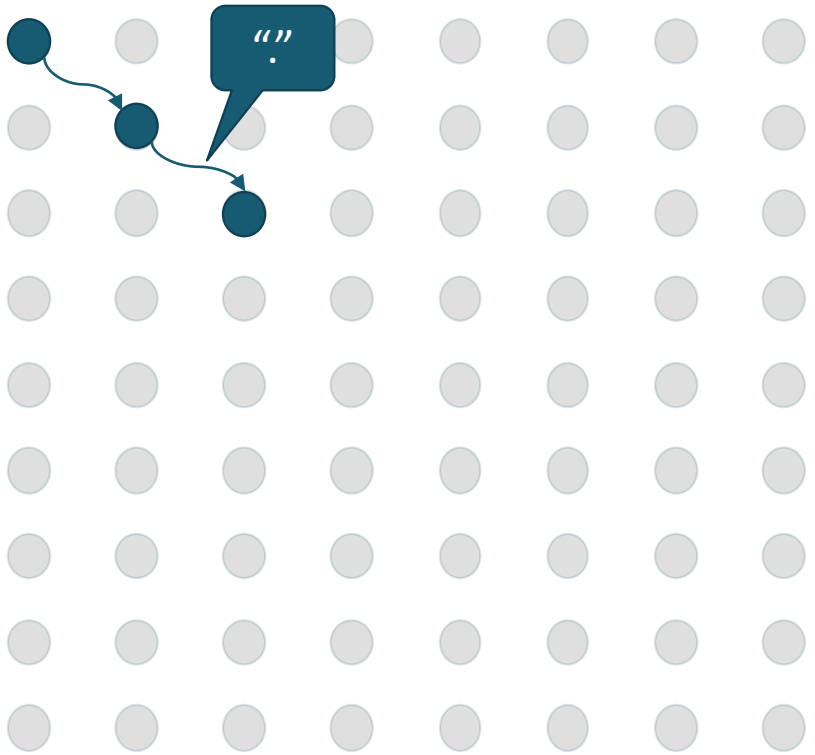
Intersection



X. Qiu



X. Qiu



Ranking

Prefer shorter programs.

- Fewer number of conditionals.
- Shorter string expression, regular expressions.

Prefer programs with fewer constants.

Strategies

Baseline: Pick any minimal sized program using minimal number of constants.

Manual: Break conflicts using a weighted score of program features.

Machine Learning: Weights are learned from training data.

Automata-theoretic inductive learning

Slides by Paul Krogmeier

Version spaces beyond lattices?

Automata are a natural choice

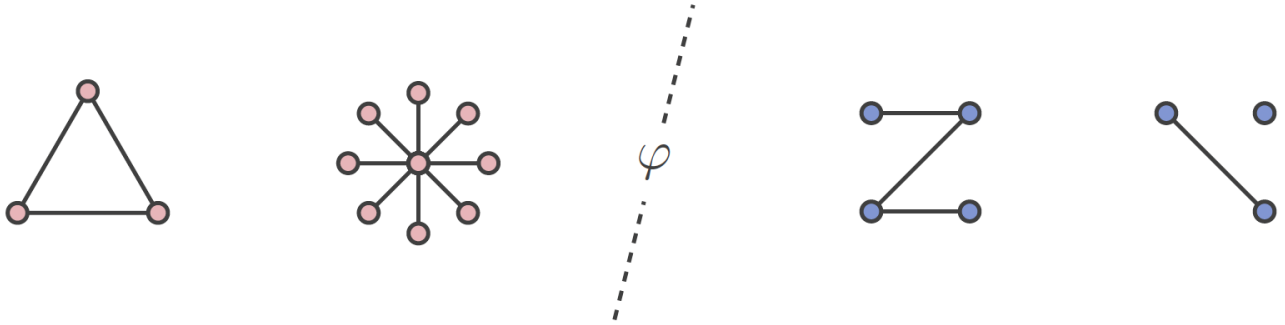
- A classical formalism for languages (sets of strings/trees)
- A lot of nice properties
  - Unique minimized representation
  - Closed under union, intersection, and complementation

Logic learning

Learning Formulas in Finite Variable Logics

PAUL KROGMEIER, University of Illinois, Urbana-Champaign, USA
P. MADHUSUDAN, University of Illinois, Urbana-Champaign, USA

What: learning logical formulas and terms from **structures**

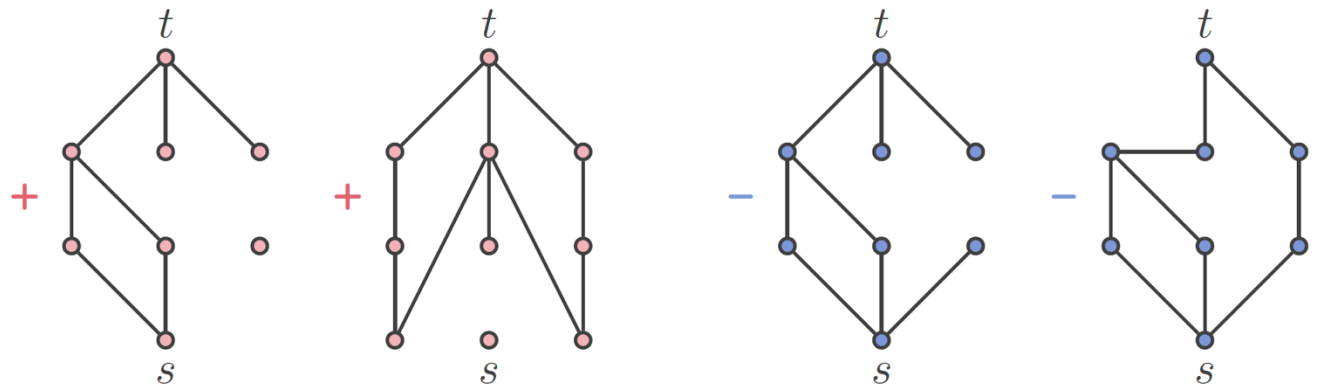


Why:

1. Want machine learning artifacts that we can interpret
2. First-order structure is common

   program states
   algebraic datatypes
   relational databases
   visual scenes

Examples: first-order logic (FO)



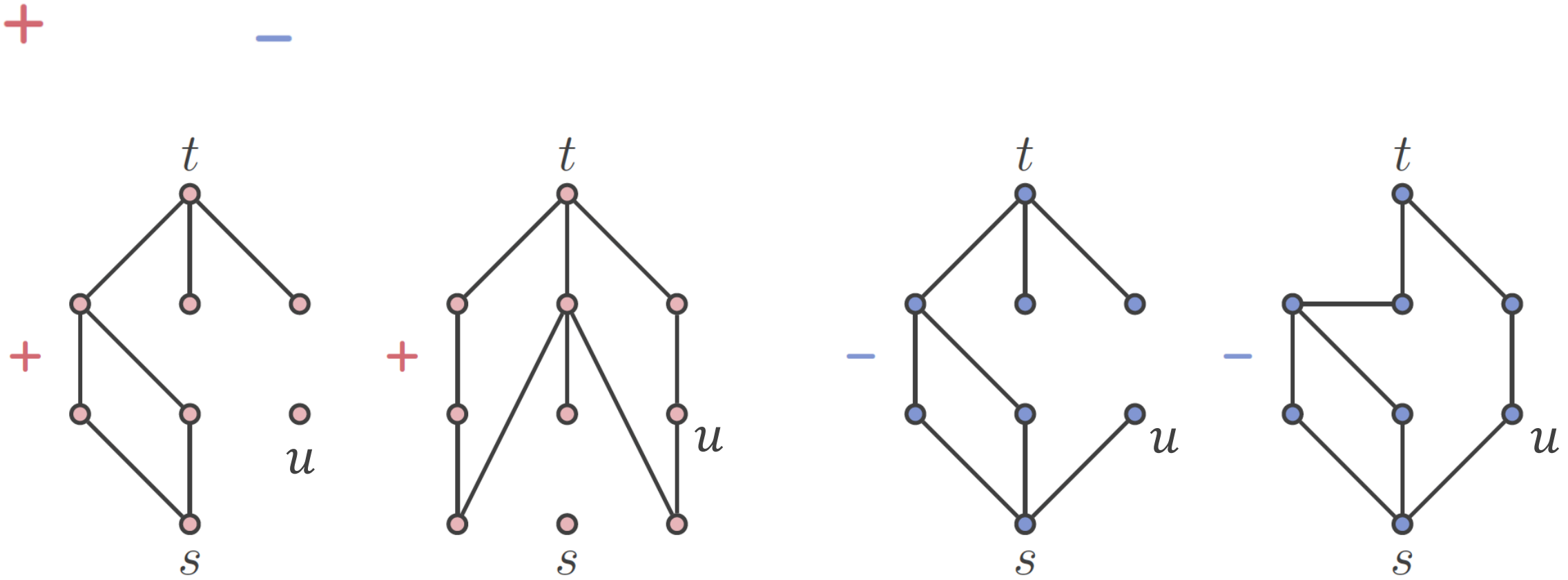
$$\varphi ::= E(term, term) \mid \varphi \wedge \varphi \mid \varphi \rightarrow \varphi \mid \exists x.\varphi \mid \forall x.\varphi$$

$$term ::= s \mid t \mid x \in V$$

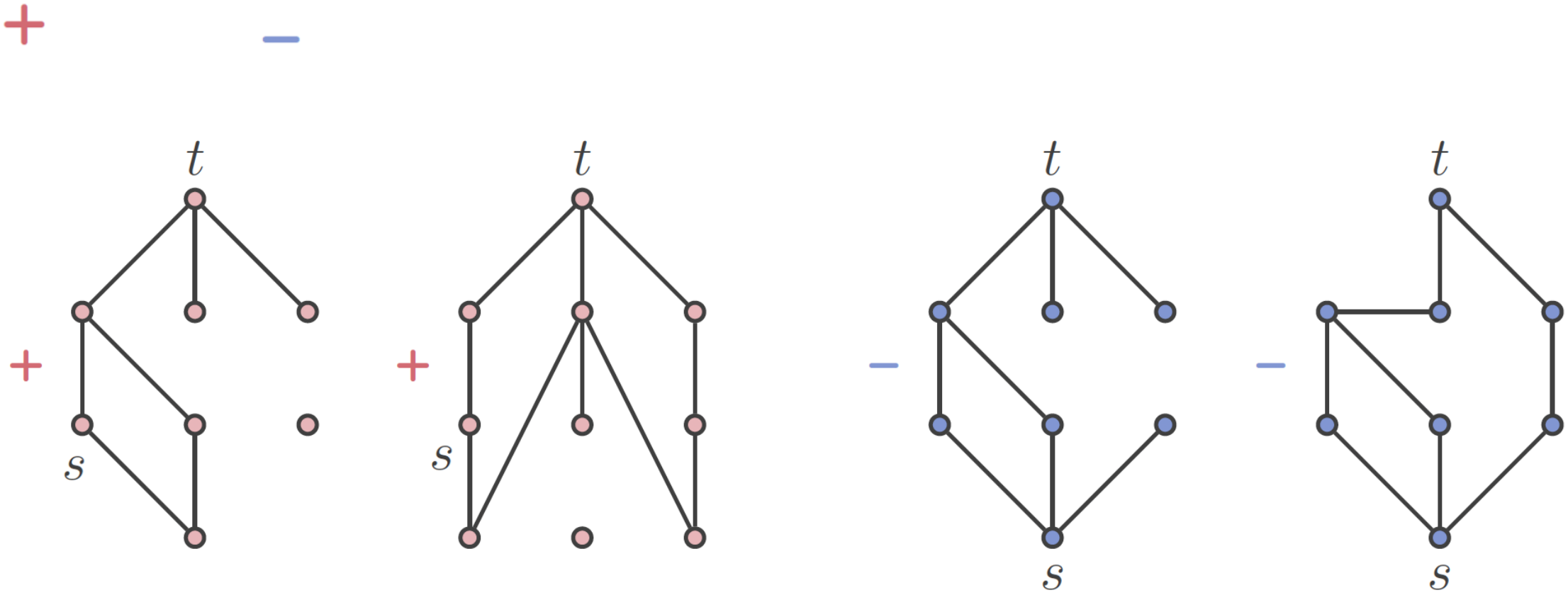
"Nodes adjacent to s can get to t in two edges"

Solution: $\forall x.\ (E(s,x) \rightarrow \exists y.\ (E(x,y) \wedge E(y,t)))$

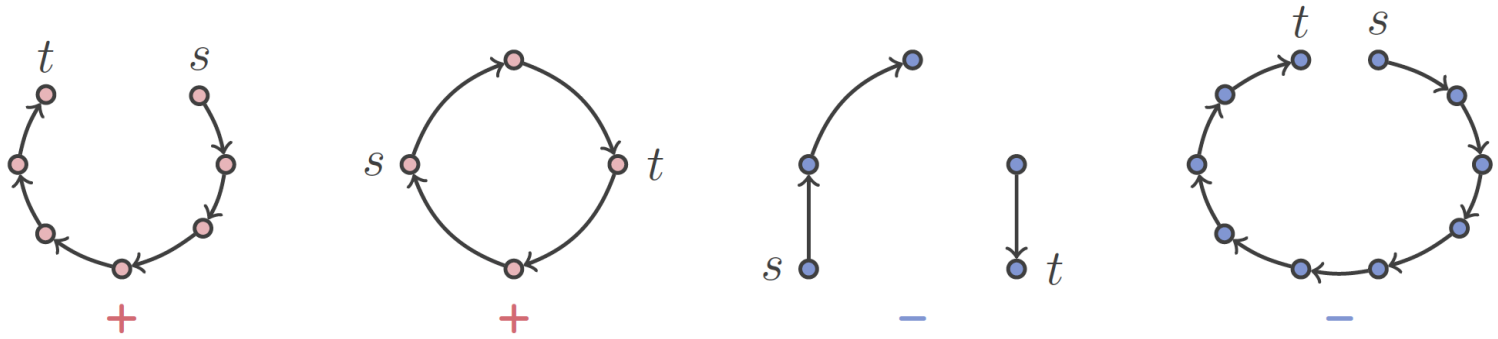
Examples: first-order logic (FO)



Examples: first-order logic (FO)



Examples: first-order logic (FO)

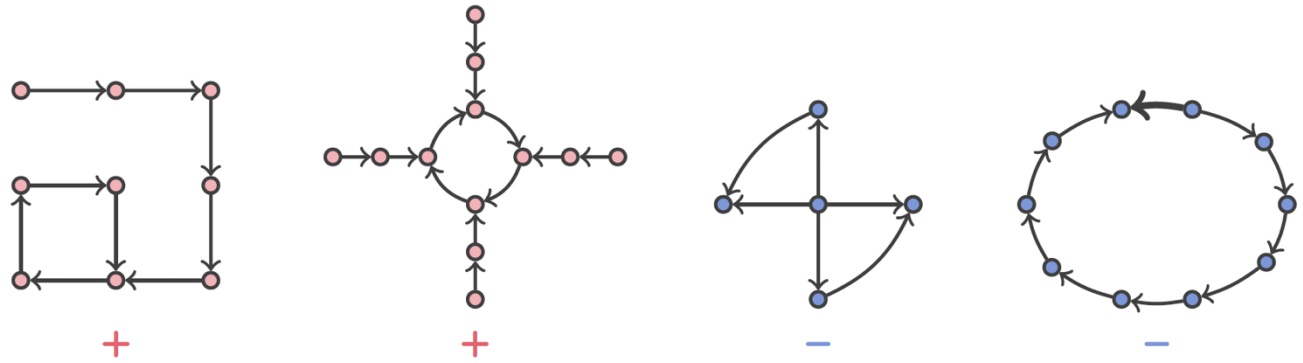


"Node s reaches node t with a small number of edges"

Solution: $\bigvee_{i=1}^{7} path_i(s,t)$

$$path_1(x,y) \leftrightarrow E(x,y)$$
$$path_{i+1}(x,y) \leftrightarrow \exists z.\,(E(x,z) \wedge \exists x.\,(x = z \wedge path_i(x,y))) \quad i > 0$$

Examples: FO with Recursive Definitions



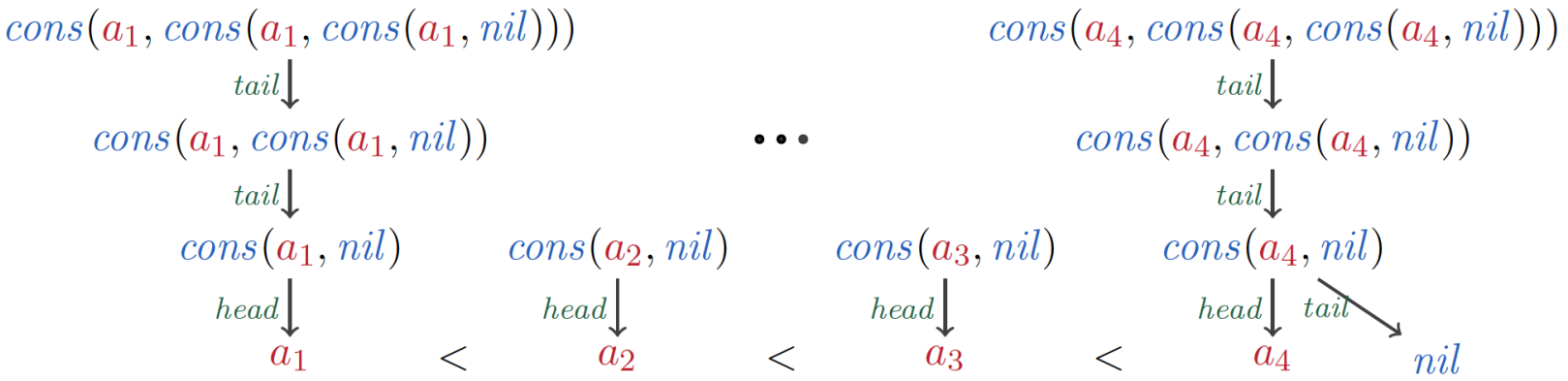
"All nodes can reach a cycle"

Solution:

let $reach(x, y) =_{\mathsf{lfp}} E(x, y) \vee \exists z.\ (E(x, z) \wedge reach(z, y))$
in $\forall x.\ \exists y.\ (reach(x, y) \wedge reach(y, y))$

Examples: Term Synthesis

Lists of length ≤ 3 over an ordered domain $\{a_i\}$
Signature: $\{\, cons, nil, head, tail \,\}$



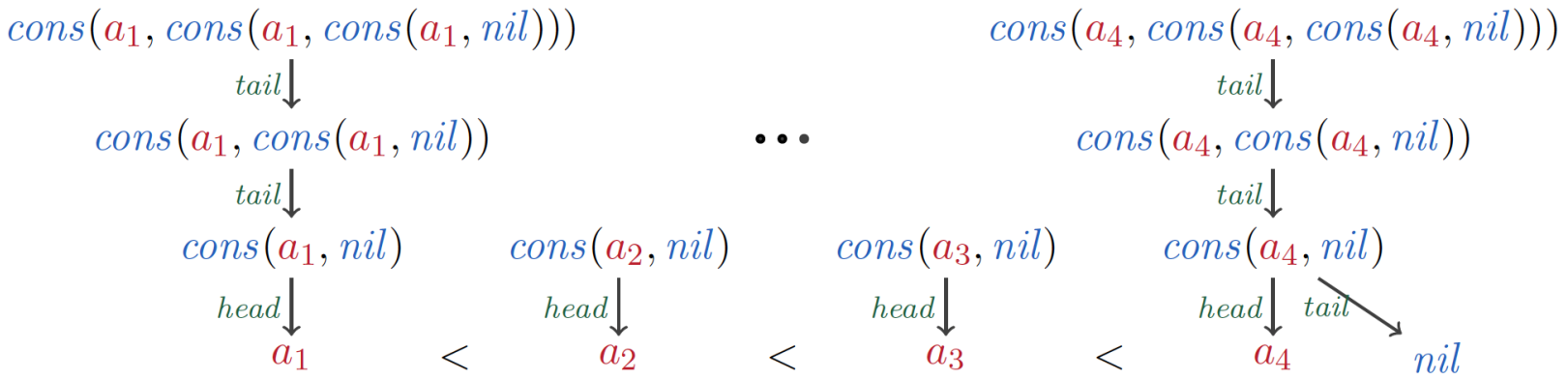
Given constants in_1, in_2, and out interpreted as:

$$[\![in_1]\!] = cons(a_4, nil)$$
$$[\![in_2]\!] = cons(a_2, cons(a_3, nil))$$
$$[\![out]\!] = cons(a_2, cons(a_3, cons(a_4, nil)))$$

Synthesize a term t such that $[\![t]\!] = [\![out]\!]$

Examples: Term Synthesis

Lists of length ≤ 3 over an ordered domain $\{a_i\}$
Signature: $\{\ cons, nil, head, tail\ \}$



Solution: let $merge(x, y) =_{\mathsf{lfp}}$ $\mathsf{ite}(x = nil, y, \mathsf{ite}(y = nil, x,$
$\mathsf{ite}(head(y) > head(x),$
$cons(head(x), merge(tail(x), y)),$
$cons(head(y), merge(x, tail(y)))))$

in $merge(in_1, in_2)$

Separation Problem

Given:

- logic $\mathcal{L}$
- positive and negative structures $\{P_i\}$ and $\{N_j\}$
- grammar G for $\mathcal{L}$

Synthesize:

$\varphi \in L(G)$ such that each $P_i \models \varphi$ and each $N_j \not\models \varphi$

or say unrealizable

Query Problem

Given:

- logic $\mathcal{L}$
- pairs $\{\langle A_i, Ans_i \rangle\}$, with $Ans_i \subseteq A_i^r$
- grammar G for $\mathcal{L}$

Synthesize:

$\varphi(x_1, \ldots, x_r) \in L(G)$, each $Ans_i = \{\overline{a} \in A_i^r : A_i \models \varphi(\overline{a})\}$

or say unrealizable

Term Synthesis Problem

Given:

- logic $\mathcal{L}$
- structures $\{A_j\}$ interpreting constants $\{in_r\}$ and out
- grammar G for $\mathcal{L}$

Synthesize:

$t \in L(G)$ such that each $A_j \models t = out$

or say unrealizable

Main Results

Each problem is **decidable** for

- First-order logic with k variables (FO^k)
- FO^k with nested definitions
- FO^k with recursive relations and functions
- Finite-variable higher-order logics

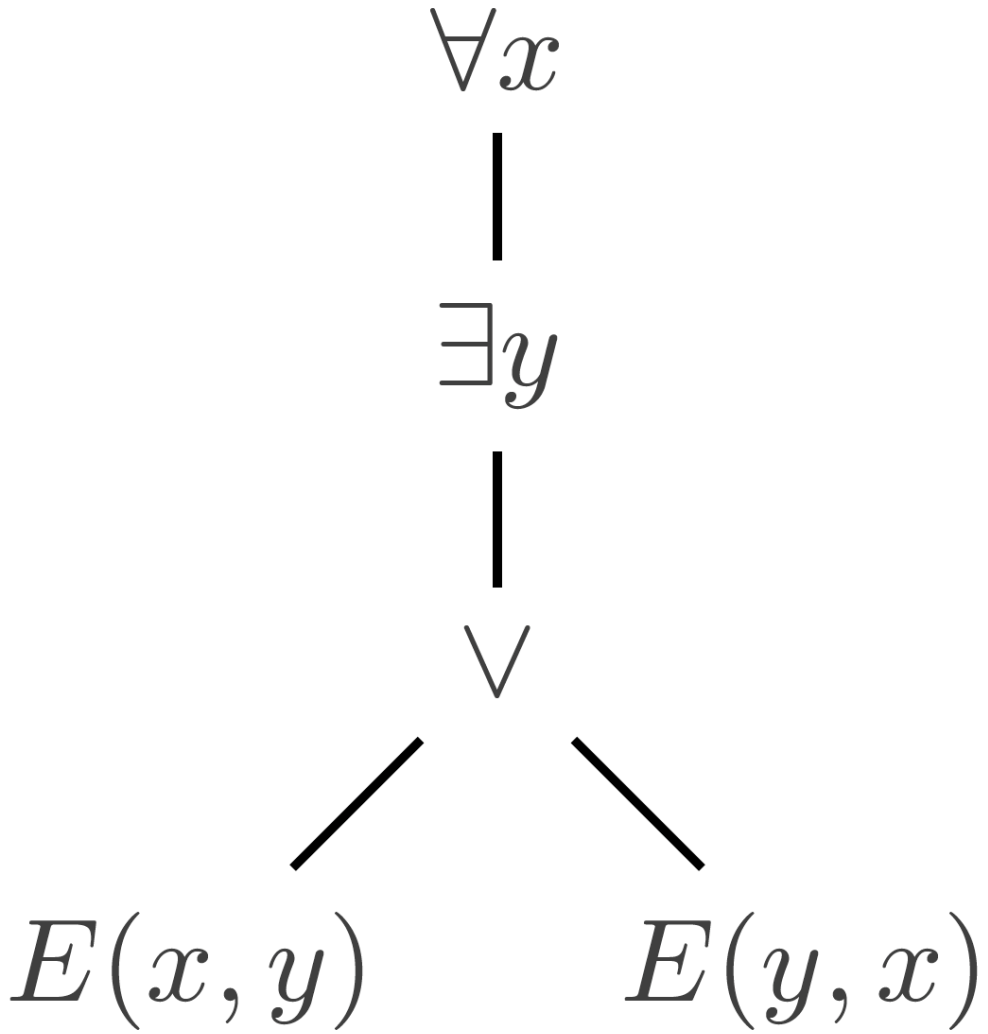
`EXPTIME`-complete for first-order logics

$\exp(m)$ $\exp(n)$ $2\exp(k)$

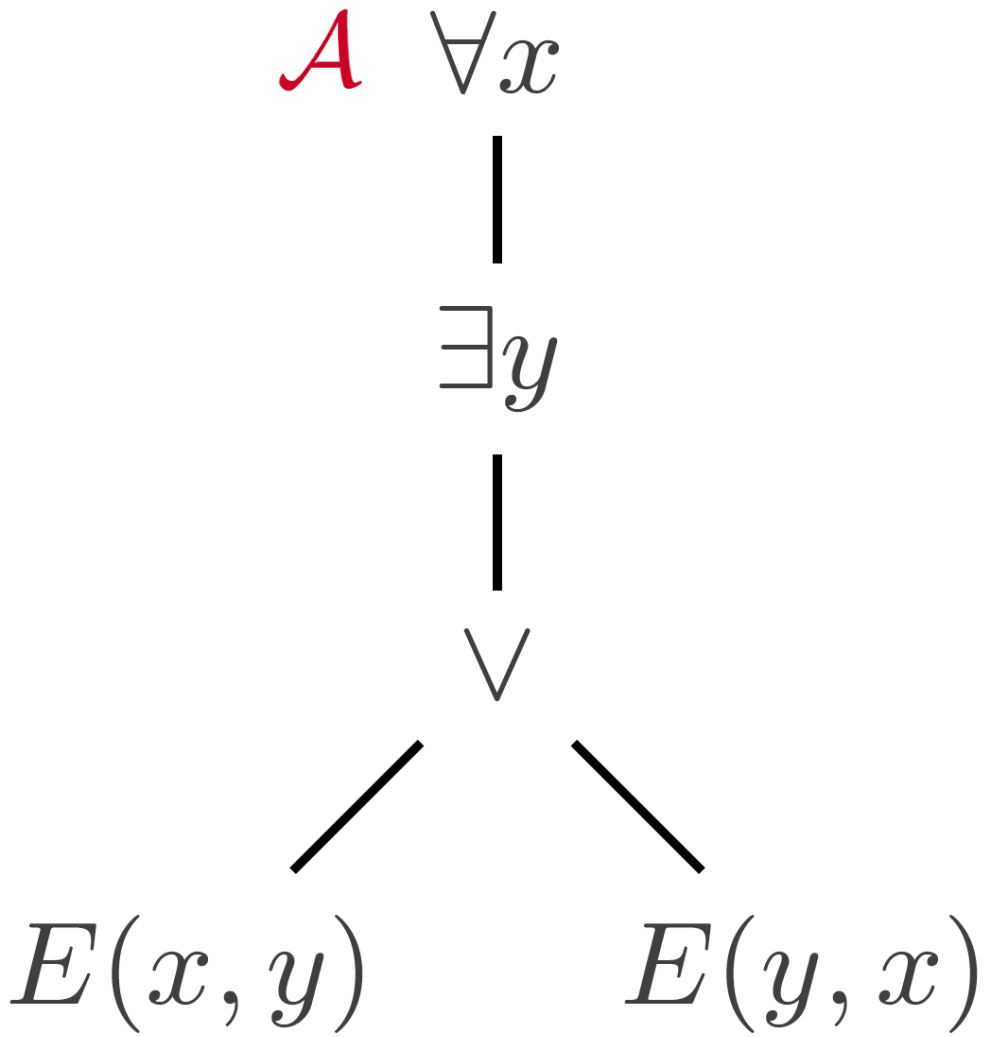
Can handle mutual recursion and background knowledge
(e.g. Datalog) (e.g. ILP)

Uniform decision procedures using **tree automata**

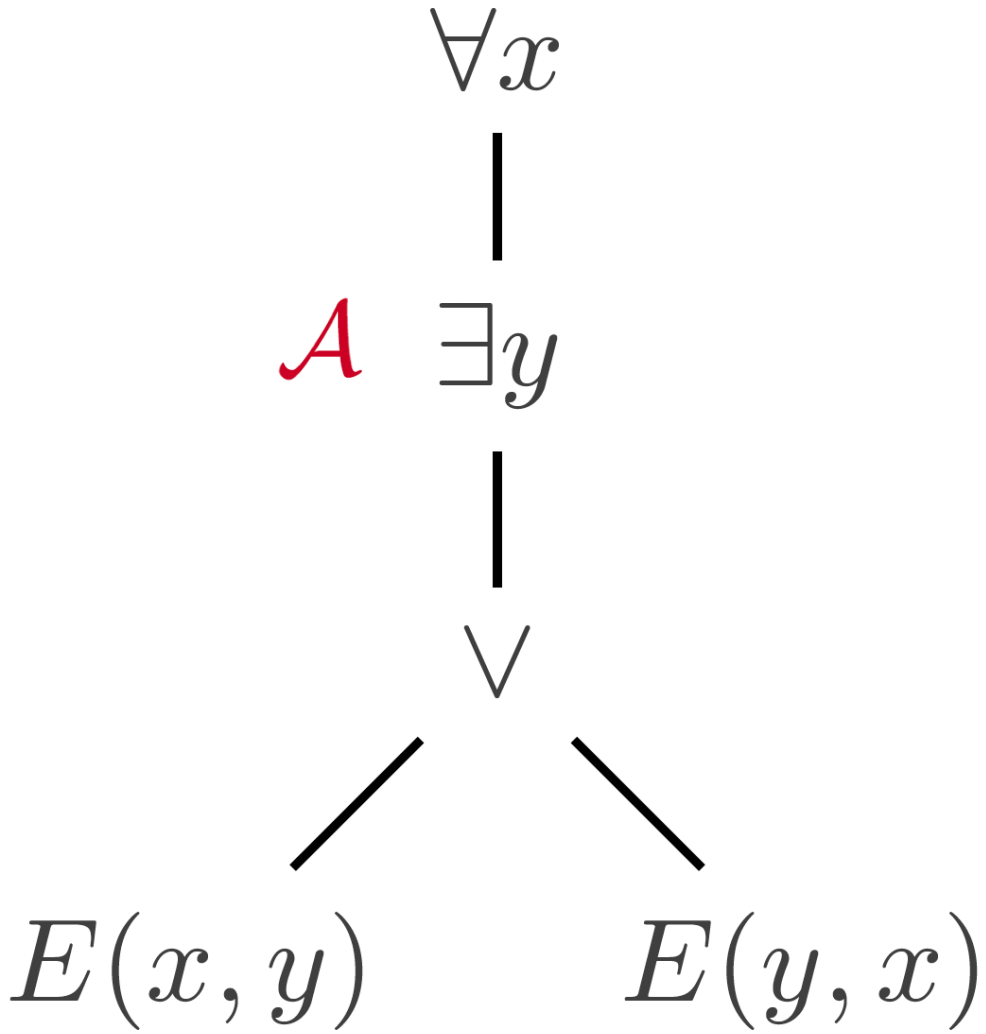
Evaluating Syntax Trees



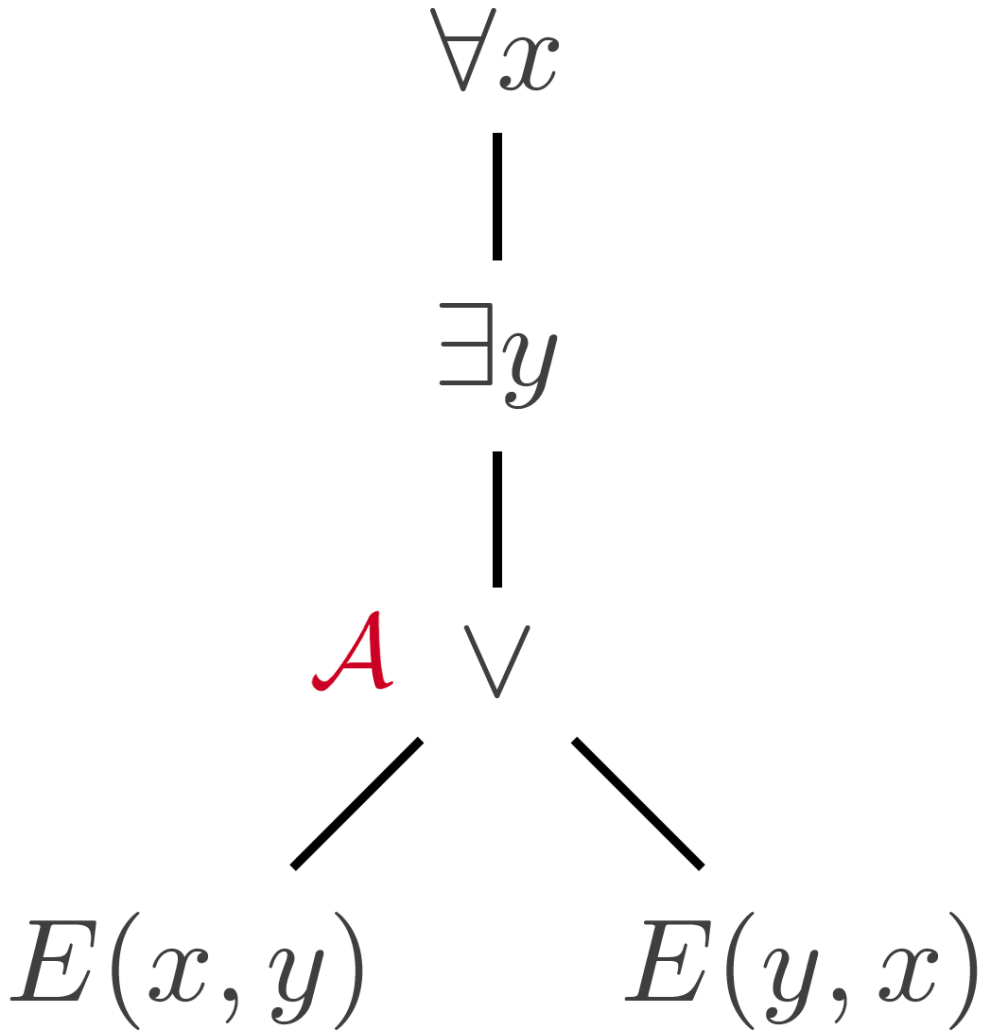
Evaluating Syntax Trees



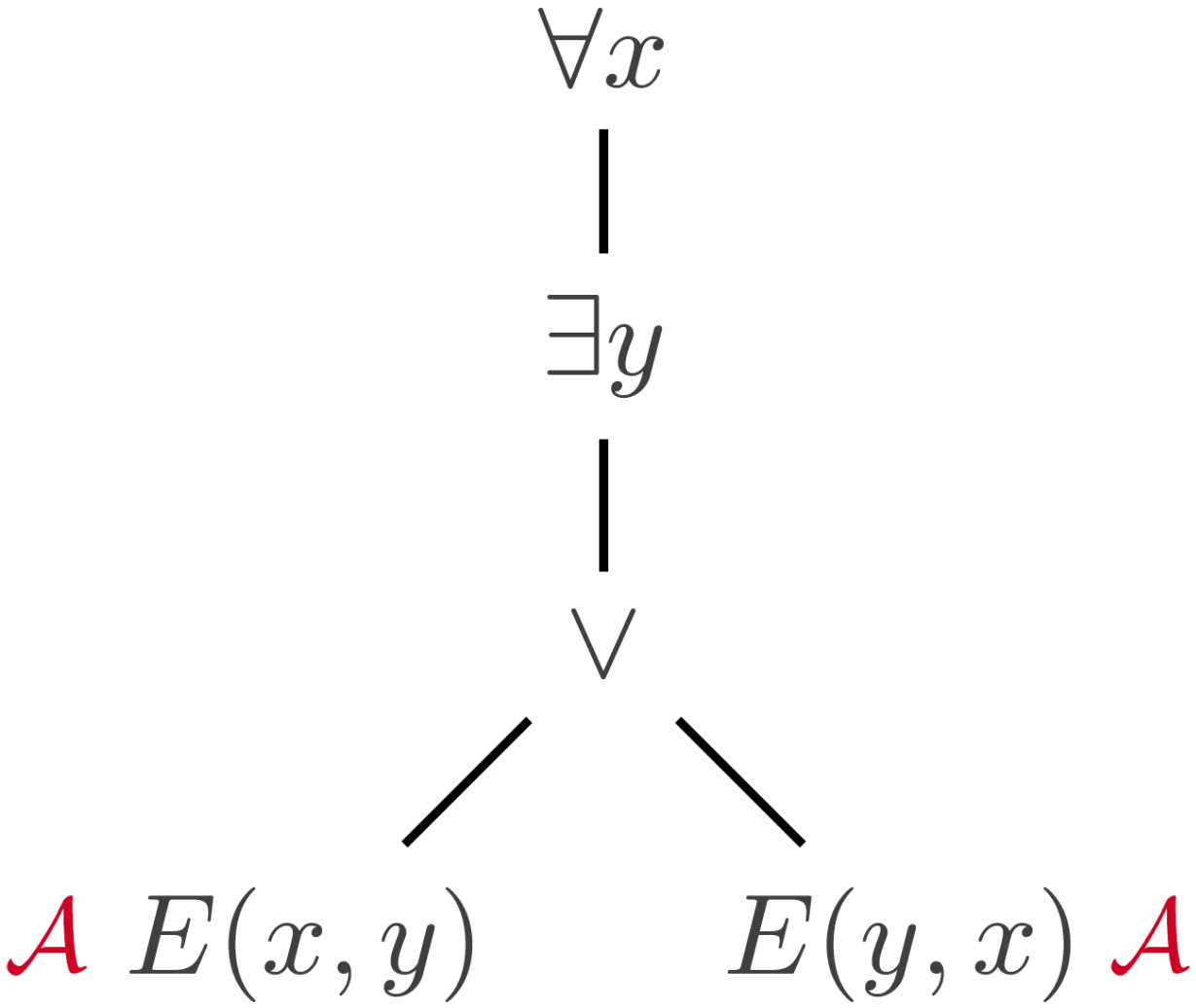
Evaluating Syntax Trees



Evaluating Syntax Trees



Evaluating Syntax Trees



Tree Automata as Version Space Algebra

1. For each positive P define automaton $\mathcal{A}(P)$ so that
   $$P \models \varphi \Leftrightarrow \varphi \in L(\mathcal{A}(P))$$
2. For each negative N define $\mathcal{A}(N)$ so that
   $$N \not\models \varphi \Leftrightarrow \varphi \in L(\mathcal{A}(N))$$
3. Define $\mathcal{A}(G)$ so that (G must be a tree grammar)
   $$\varphi \in G \Leftrightarrow \varphi \in L(\mathcal{A}(G))$$
4. Define product $\mathcal{A} :=$
   $$\mathcal{A}(P_1) \times \cdots \times \mathcal{A}(P_p) \times \mathcal{A}(N_1) \times \cdots \times \mathcal{A}(N_n) \times \mathcal{A}(G)$$
5. Check whether $L(\mathcal{A})$ is empty

Evaluating Trees on Fixed Structures

Consider FO^k with variables $V = \{x_1 \dots x_k\}$

Fix a structure M

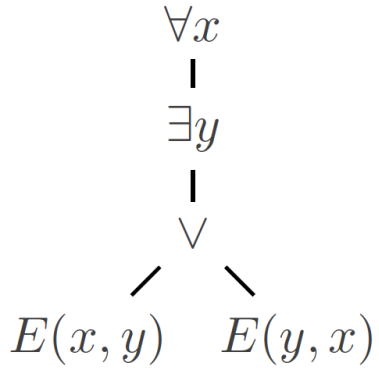
Define automaton $\mathcal{A}$ that accepts $\varphi \in \mathsf{FO}^k$ precisely when $M \models \varphi$

States of $\mathcal{A}$ are the finitely-many variable assignments $\gamma : V \to M$

More specifically: $L(\mathcal{A}, \gamma) = \{\varphi \in \mathsf{FO}^k \ : \ M, \gamma \models \varphi\}$

Alternating Tree Automata

Automaton reads a labeled tree (a syntax tree)
Sends "copies" of itself to subtrees
Each copy must accept its subtree
Choices constrained by transition function



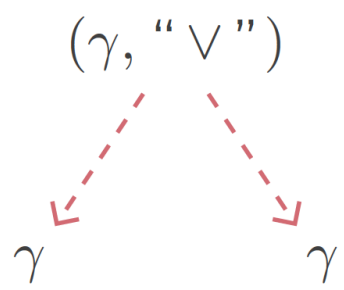
Transitions are propositional formulas, like

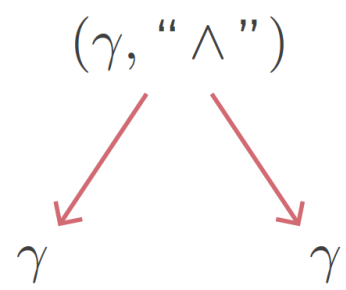
"$(q, \text{leftChild}) \vee (q', \text{leftChild}) \wedge (q', \text{rightChild})$"

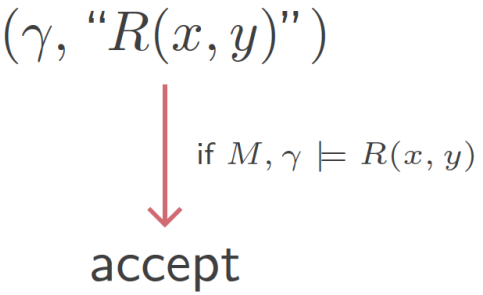
Key idea: alternation neatly expresses FO semantics

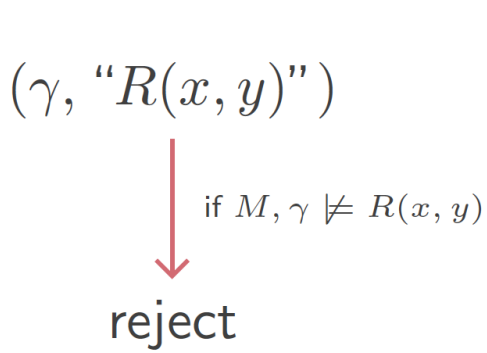
Evaluating Formulas

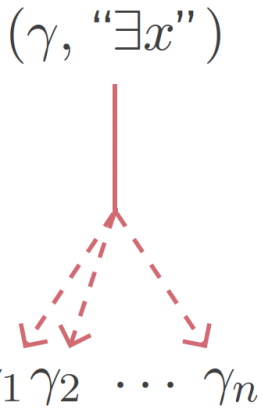
M is a fixed structure
γ is a variable assignment
Goal: define $\mathcal{A}$ so $L(\mathcal{A}, \gamma) = \{\varphi \in \mathsf{FO}^k : M, \gamma \models \varphi\}$

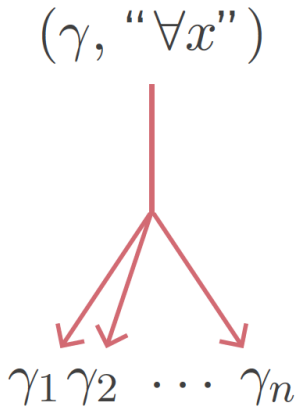


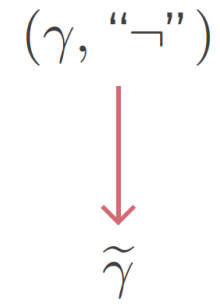






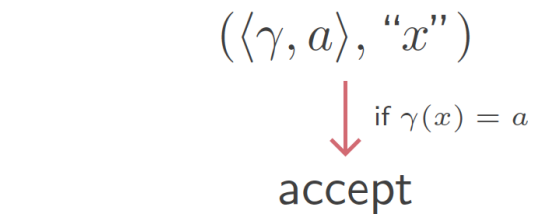


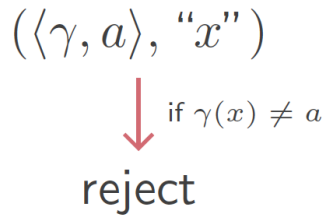


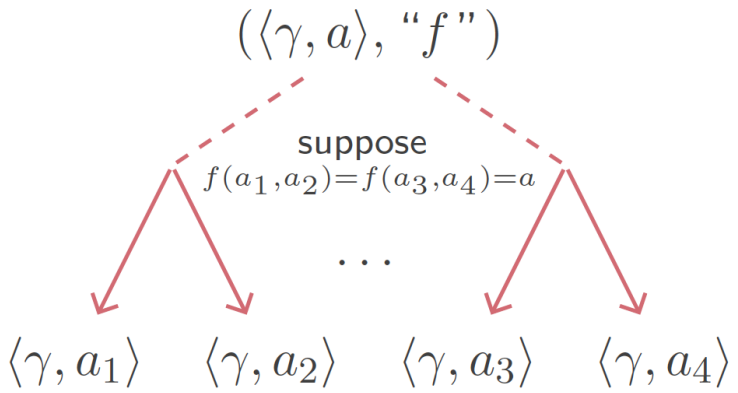


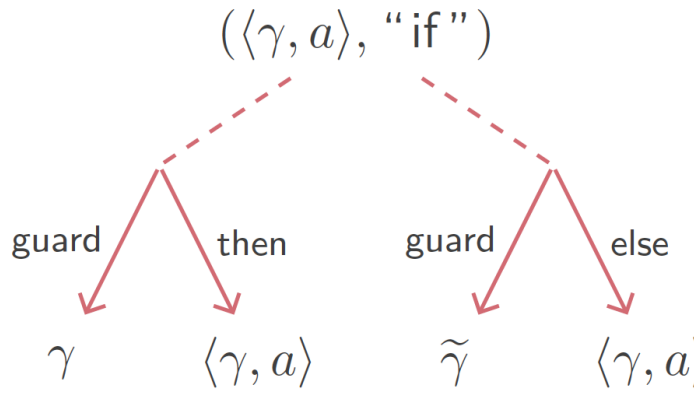
Evaluating Terms

M is a fixed structure
γ is a variable assignment
a is a target domain element
Goal: $L(\mathcal{A}, \langle \gamma, a \rangle) = \{t \in \text{terms}(\text{FO}^k) \ : \ M, \gamma \models t = a\}$









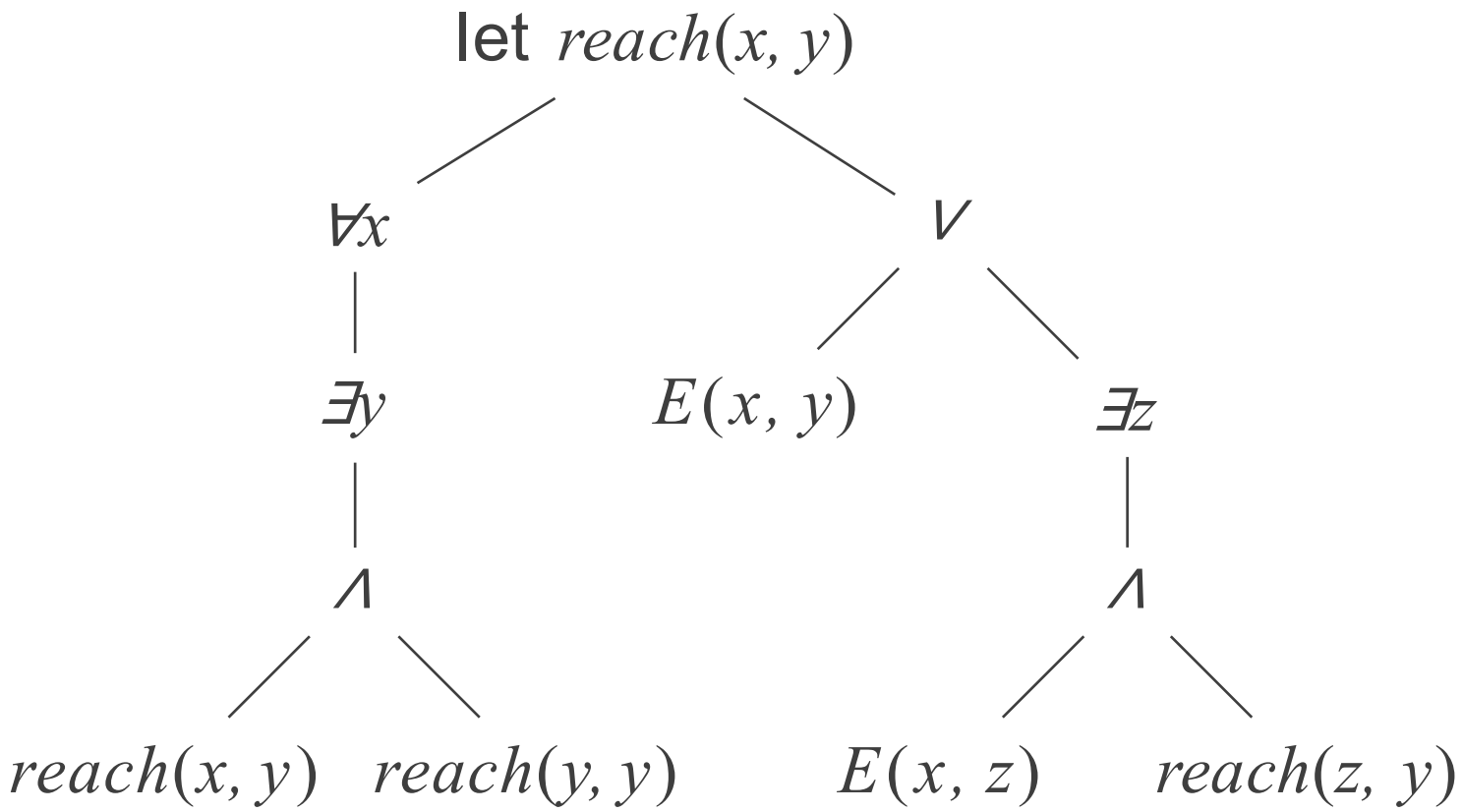
Evaluating Recursive Definitions

let $reach(x, y) =_{\mathsf{lfp}} E(x, y) \vee \exists z.\ (E(x, z) \wedge reach(z, y))$
in $\forall x.\ \exists y.\ (reach(x, y) \wedge reach(y, y))$

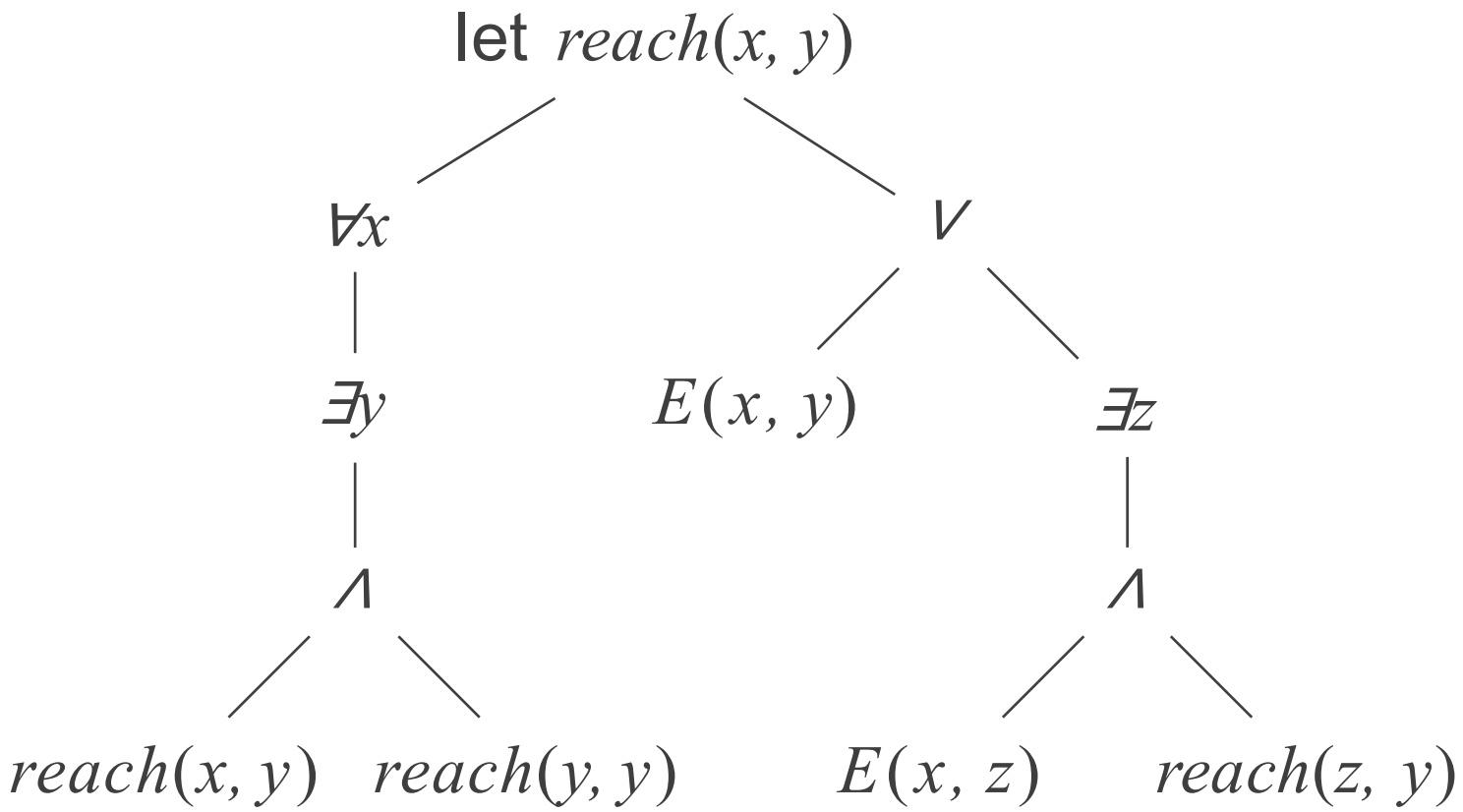
The relation $reach$ has least fixed point semantics

Key idea: Two-way automata can check membership in recursively-defined relations

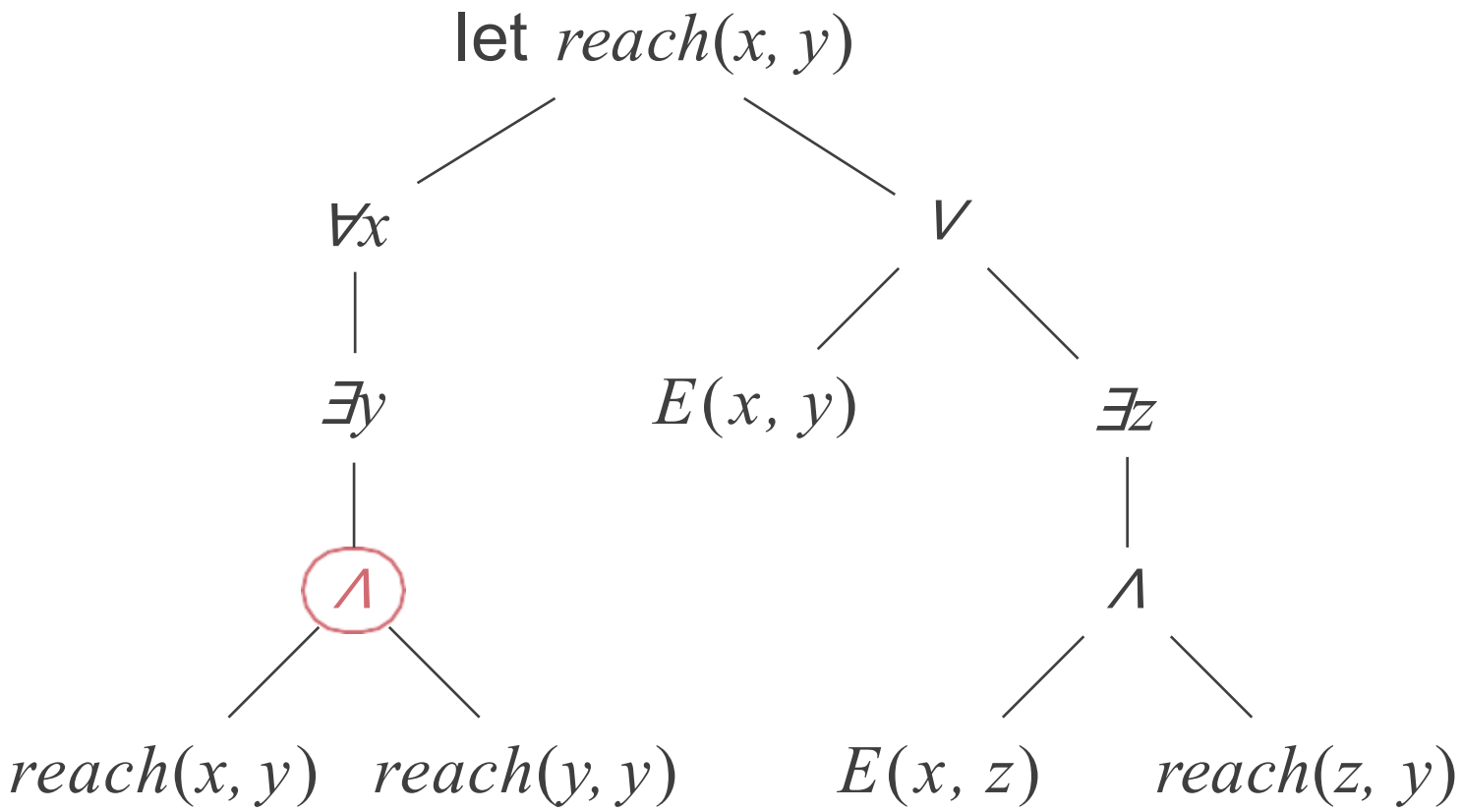
Evaluating Recursive Definitions



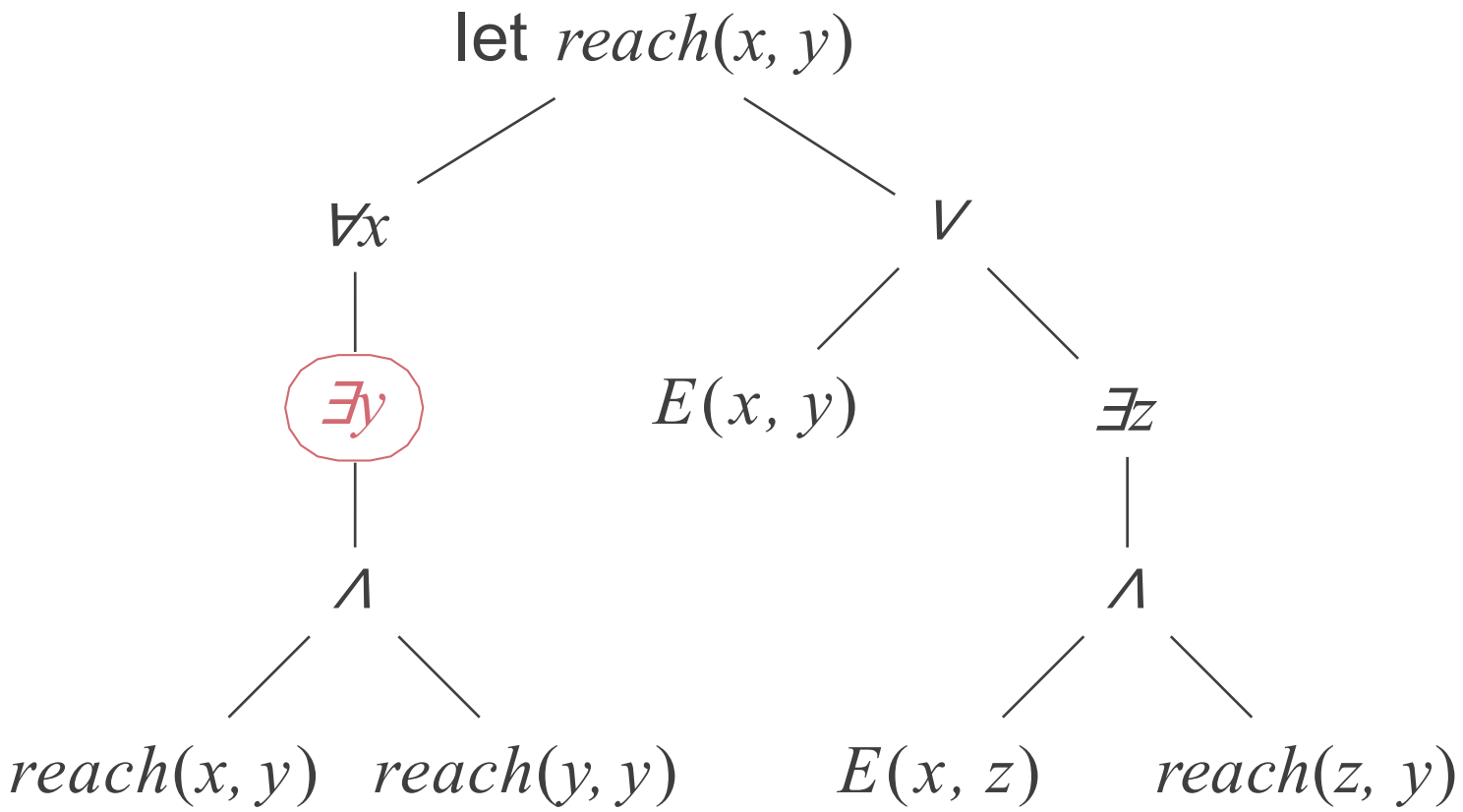
Evaluating Recursive Definitions



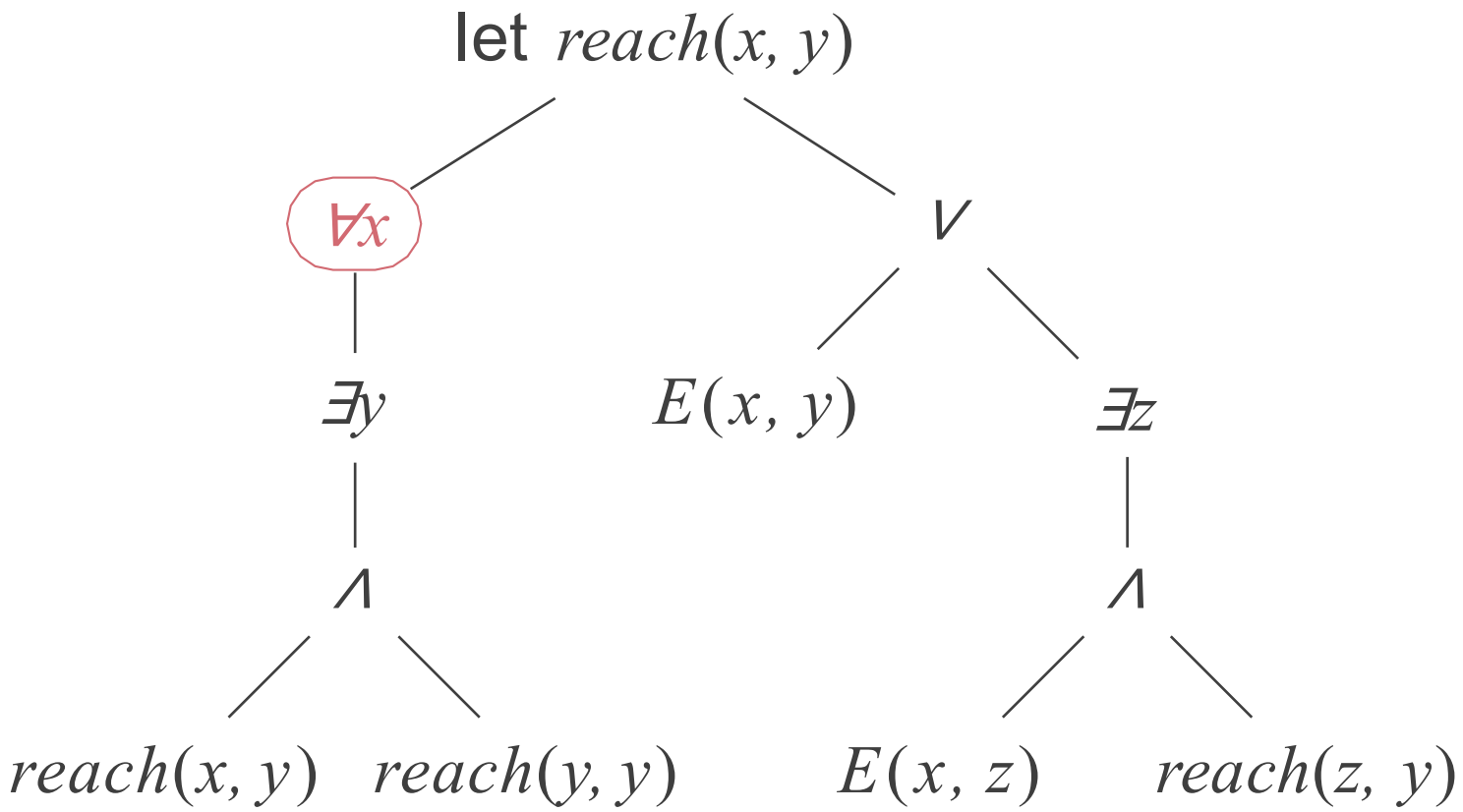
Evaluating Recursive Definitions



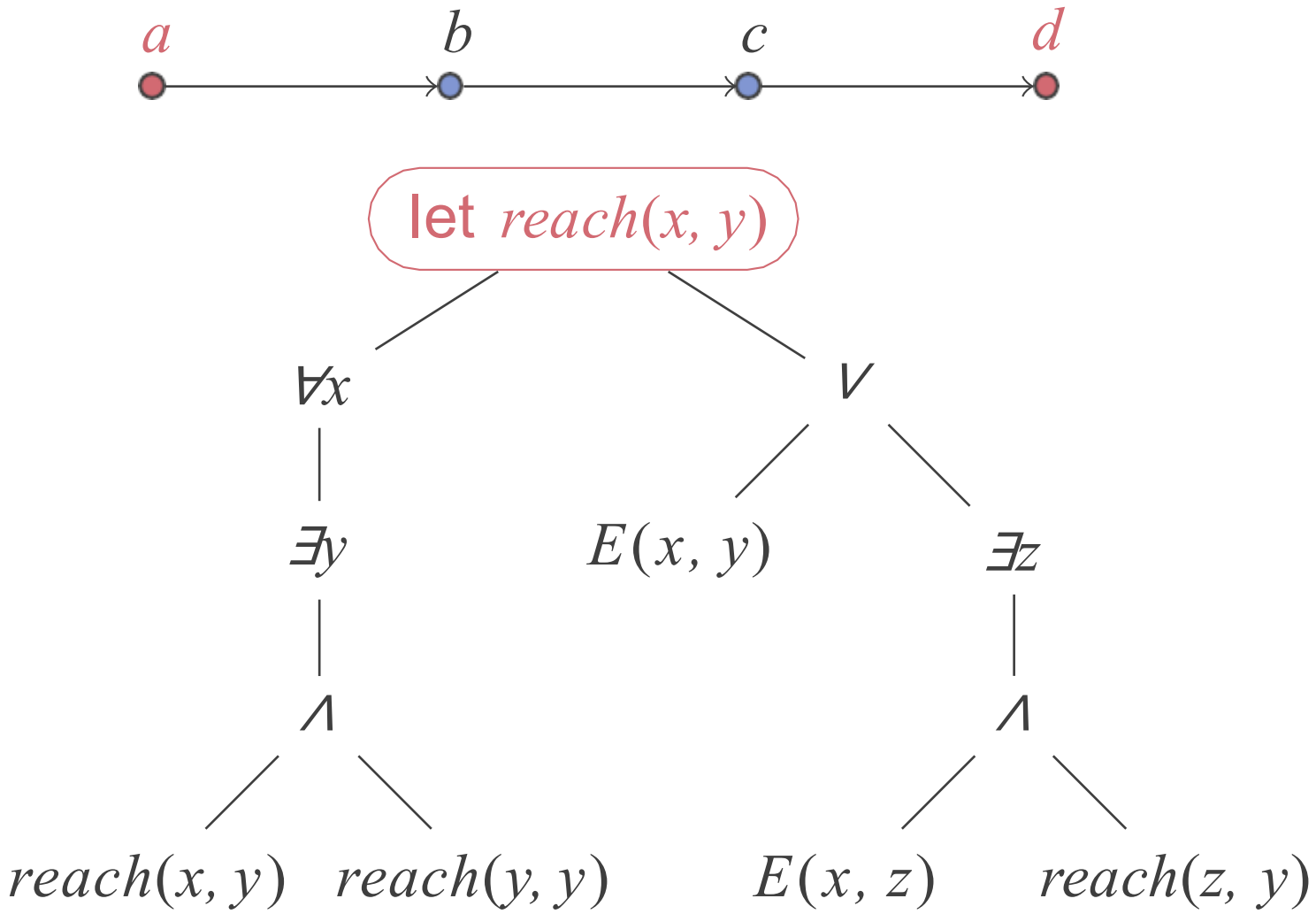
Evaluating Recursive Definitions



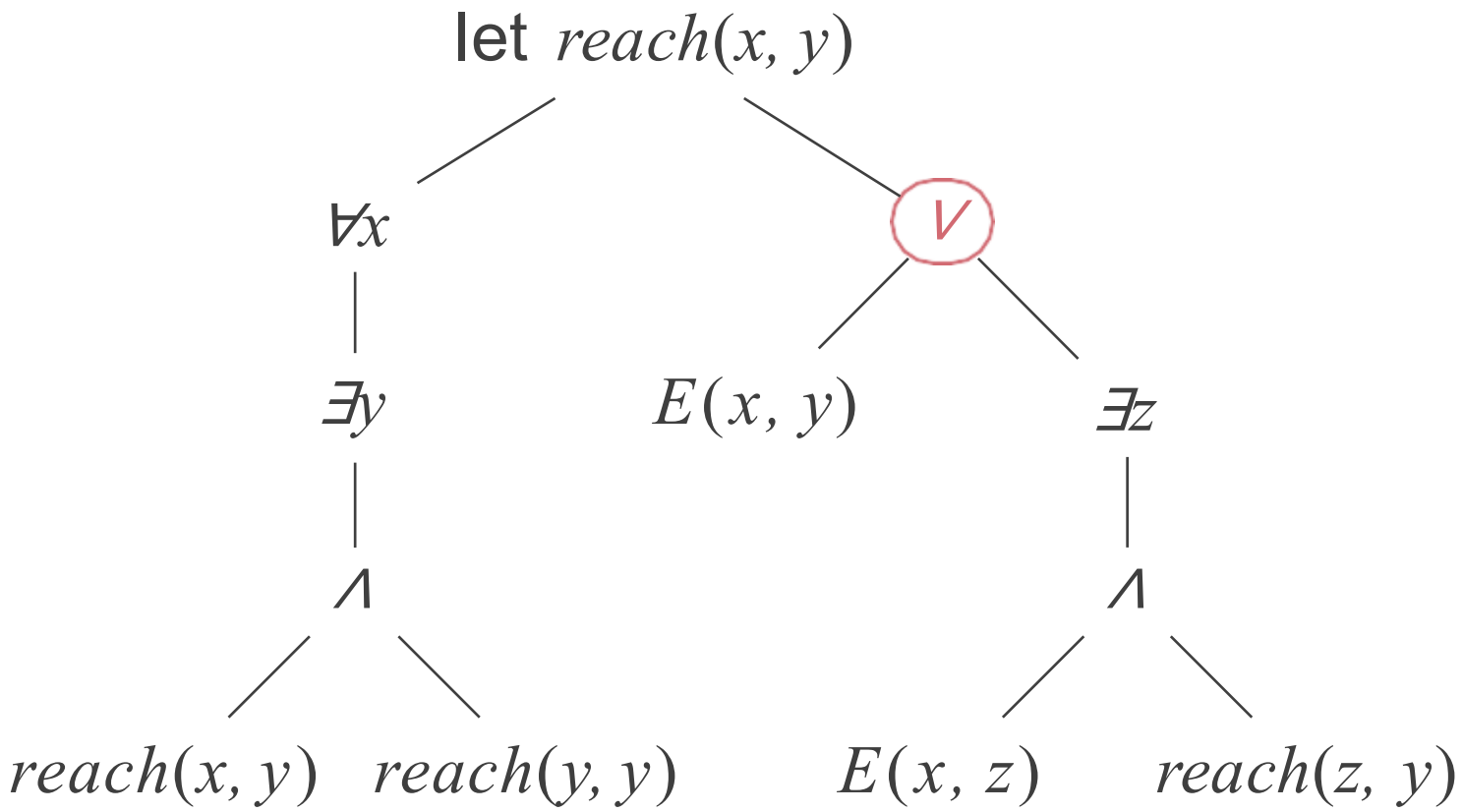
Evaluating Recursive Definitions



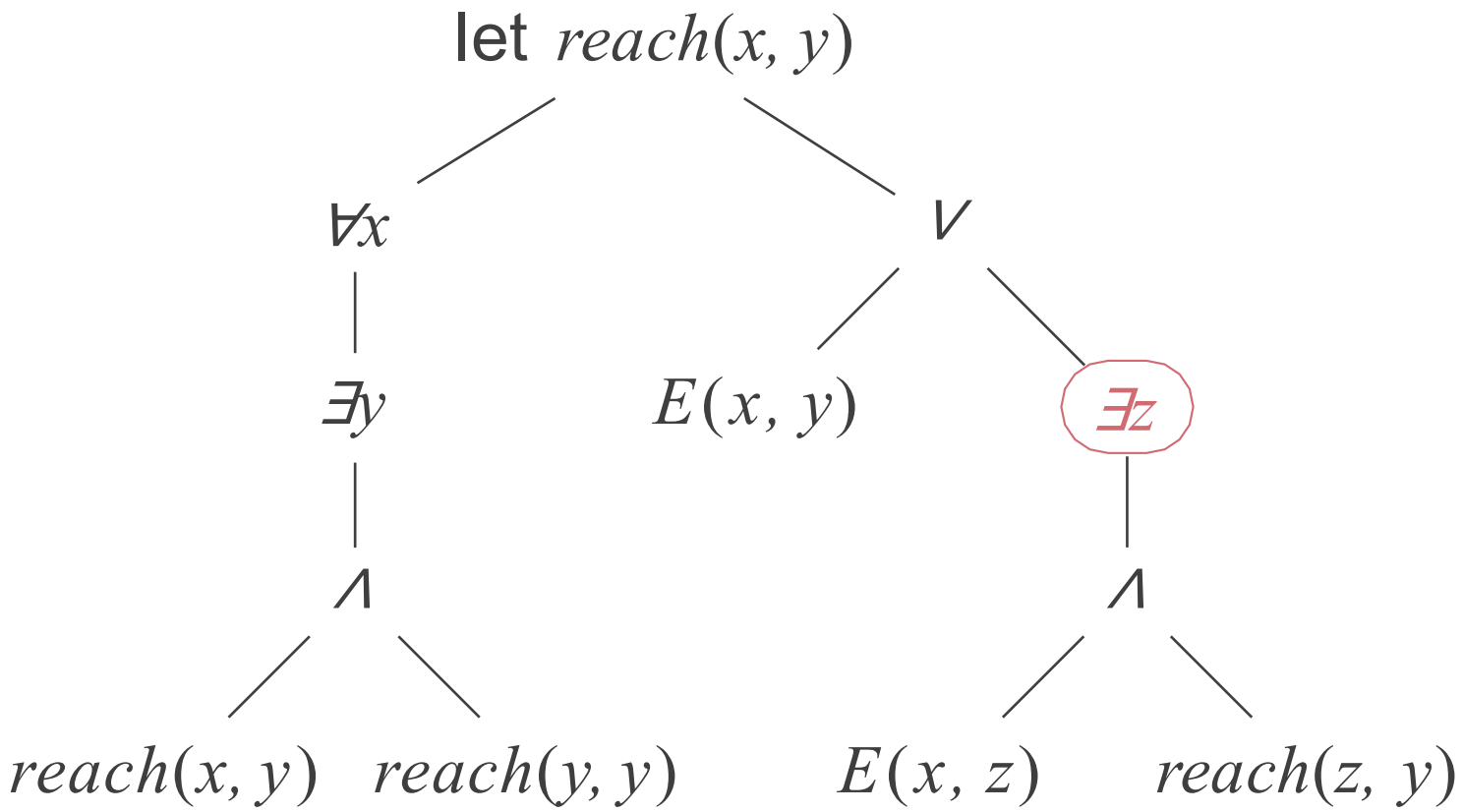
Evaluating Recursive Definitions



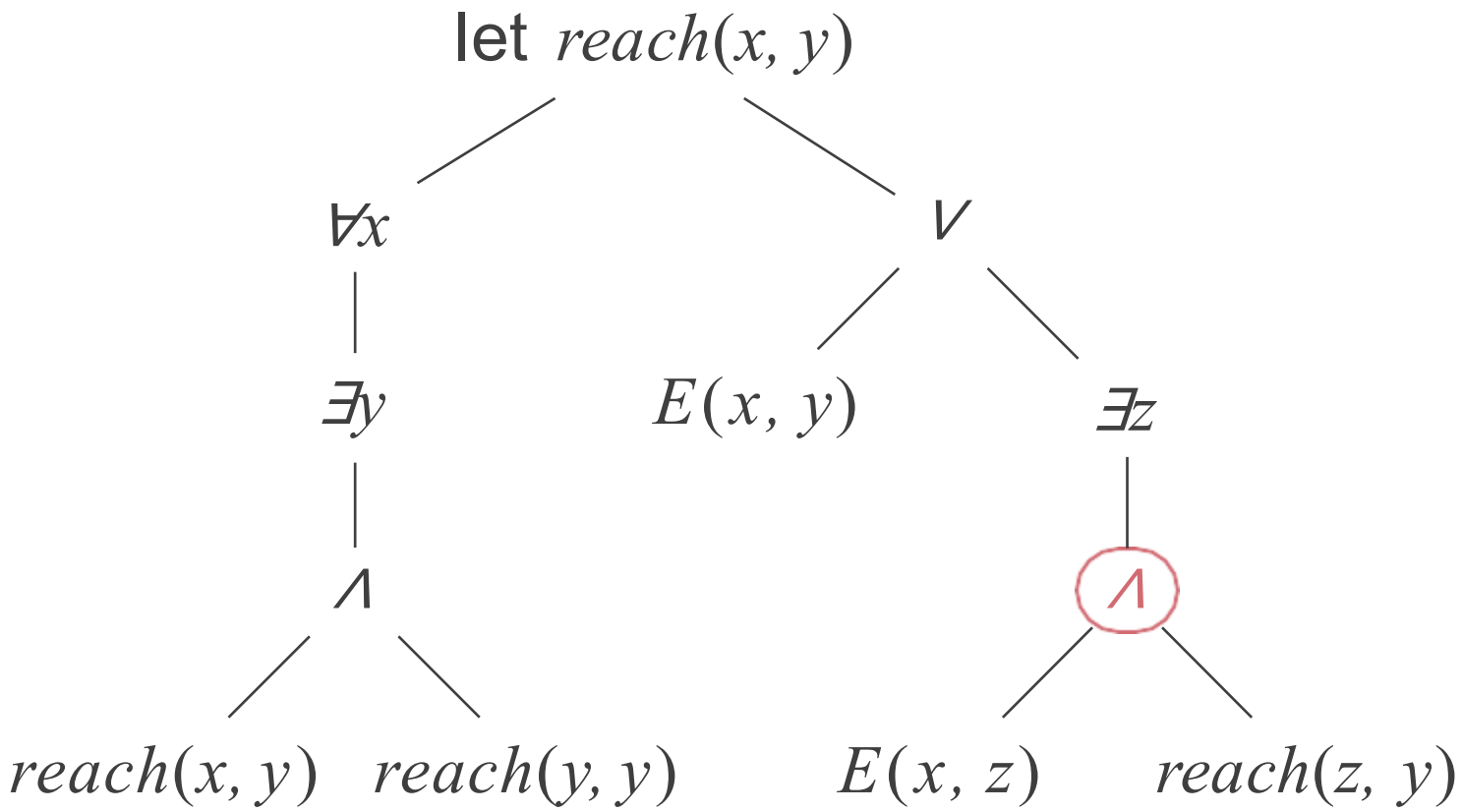
Evaluating Recursive Definitions



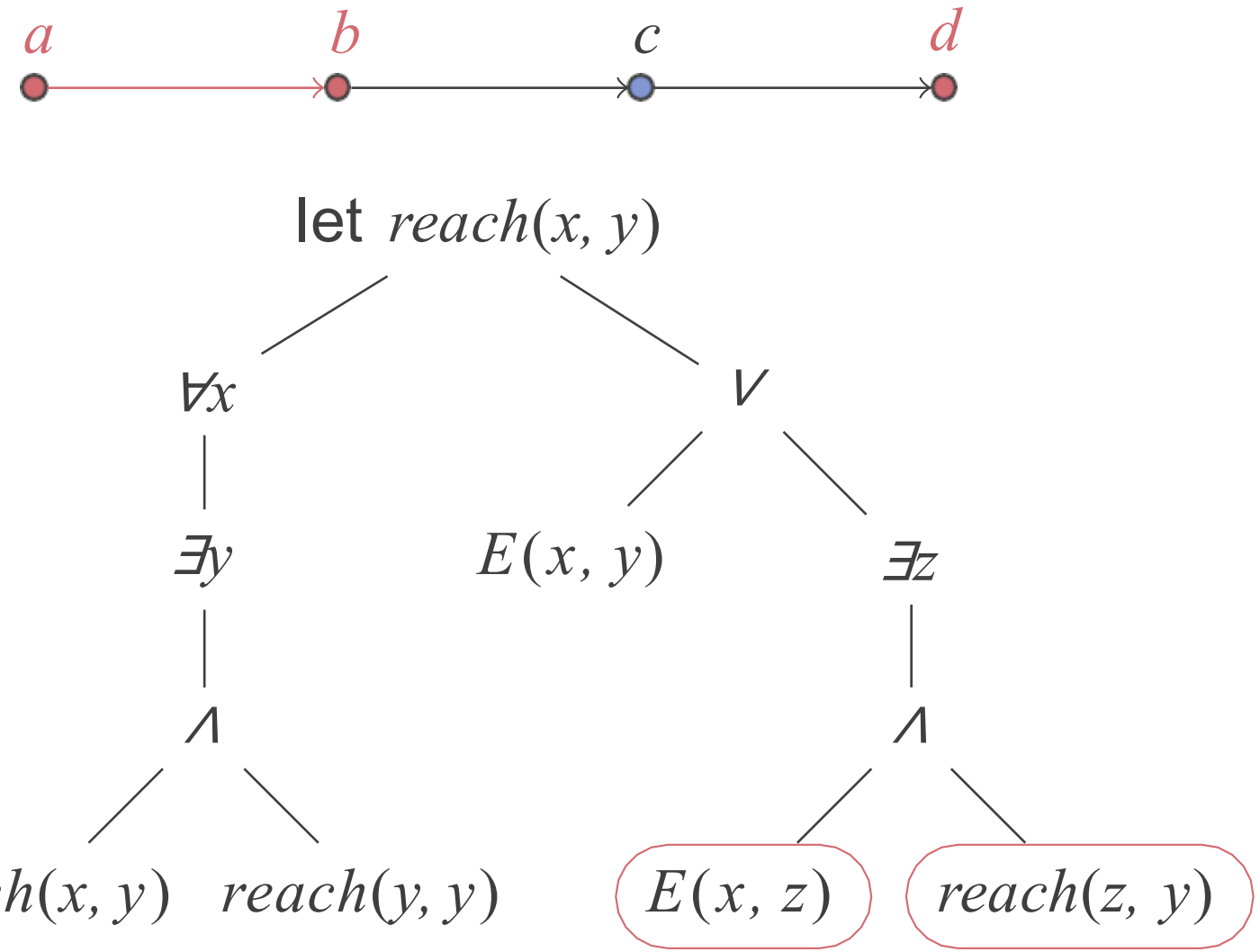
Evaluating Recursive Definitions



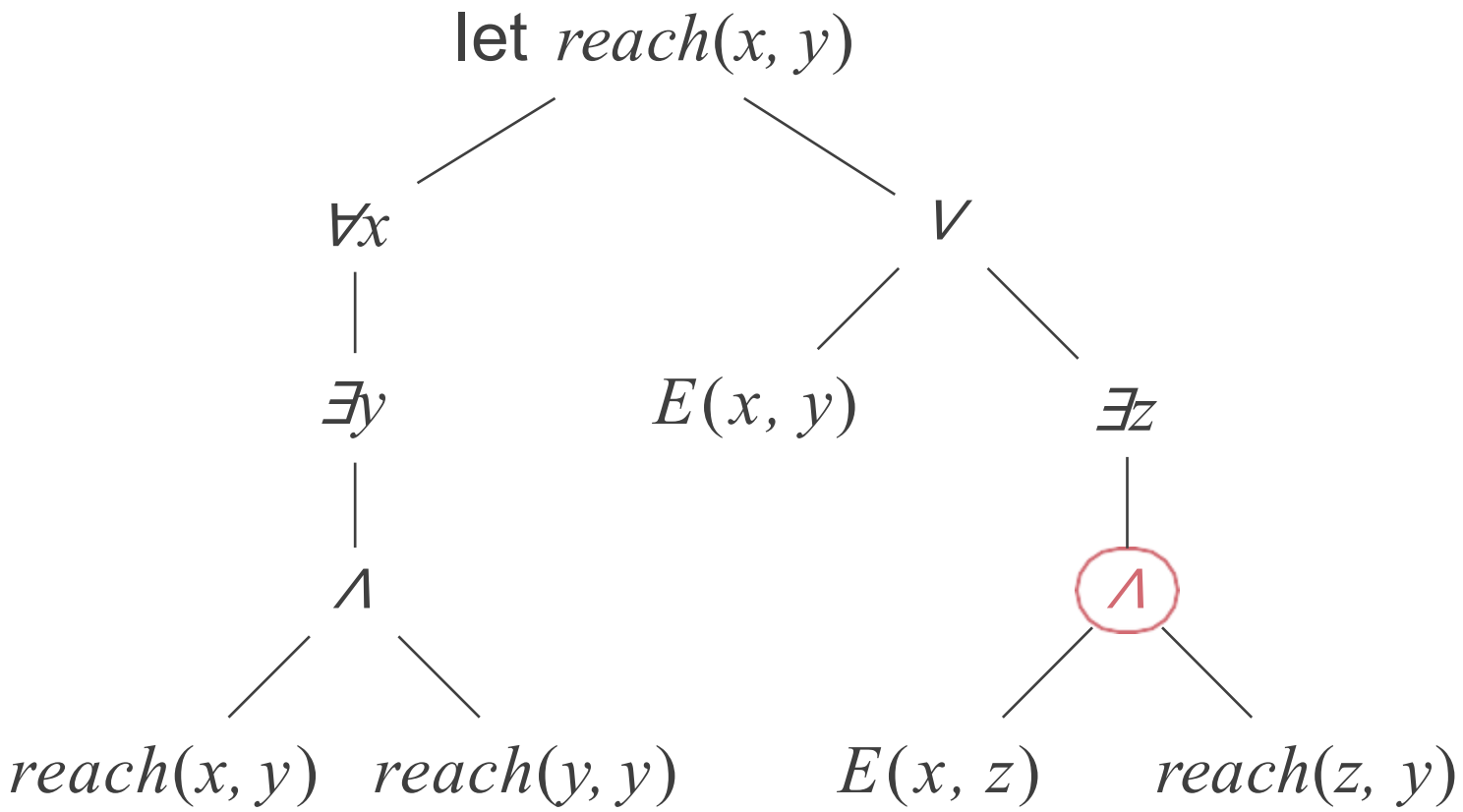
Evaluating Recursive Definitions



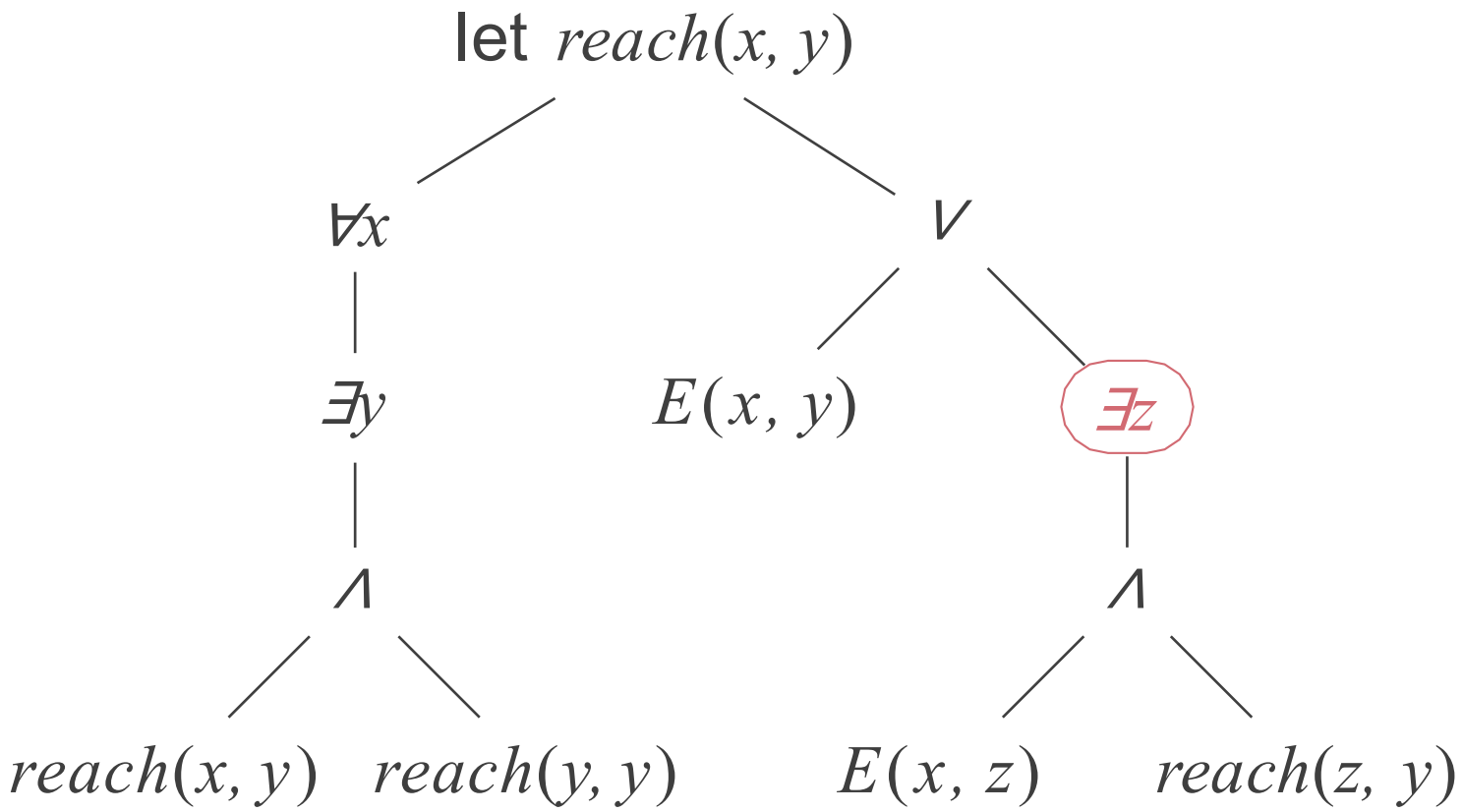
Evaluating Recursive Definitions



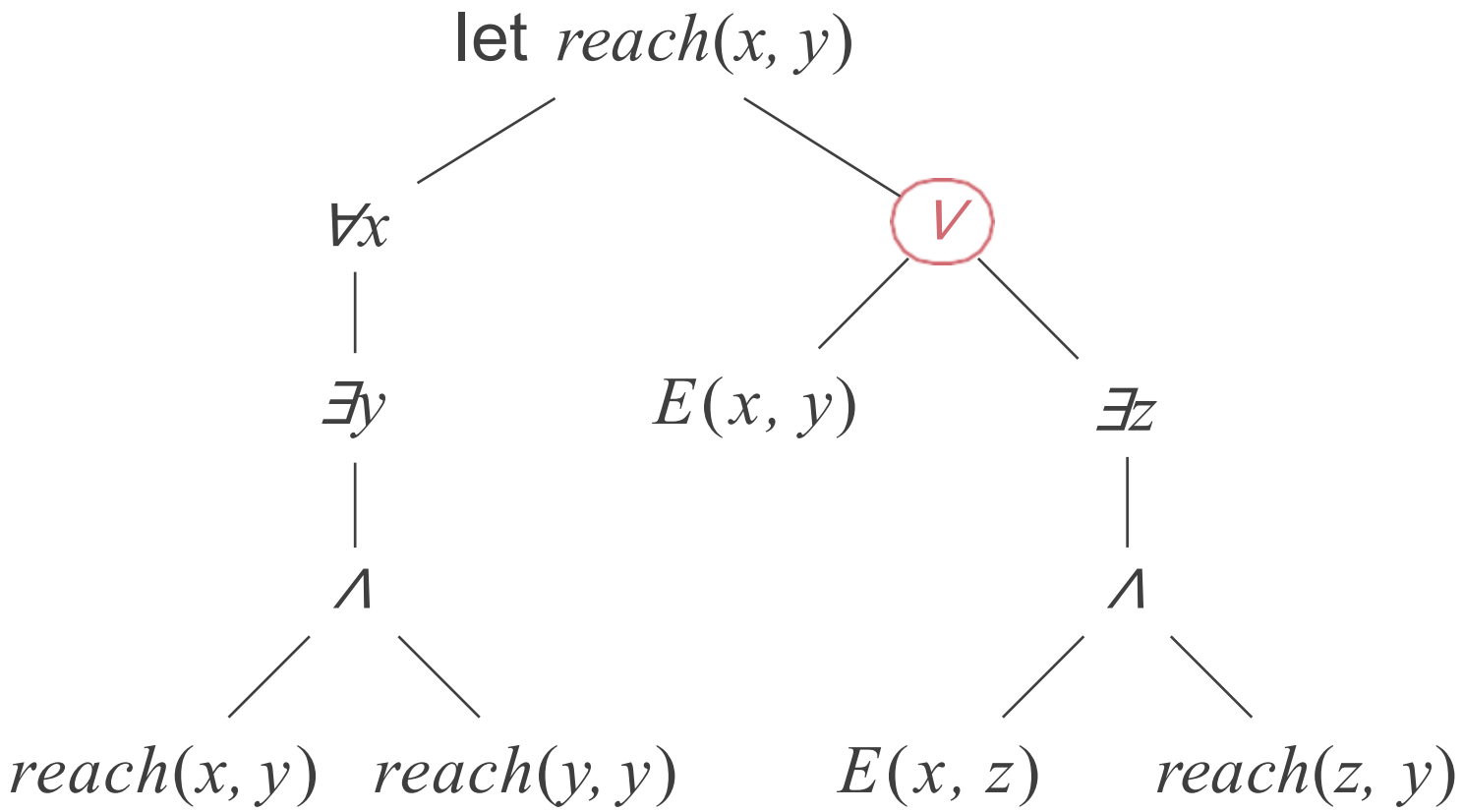
Evaluating Recursive Definitions



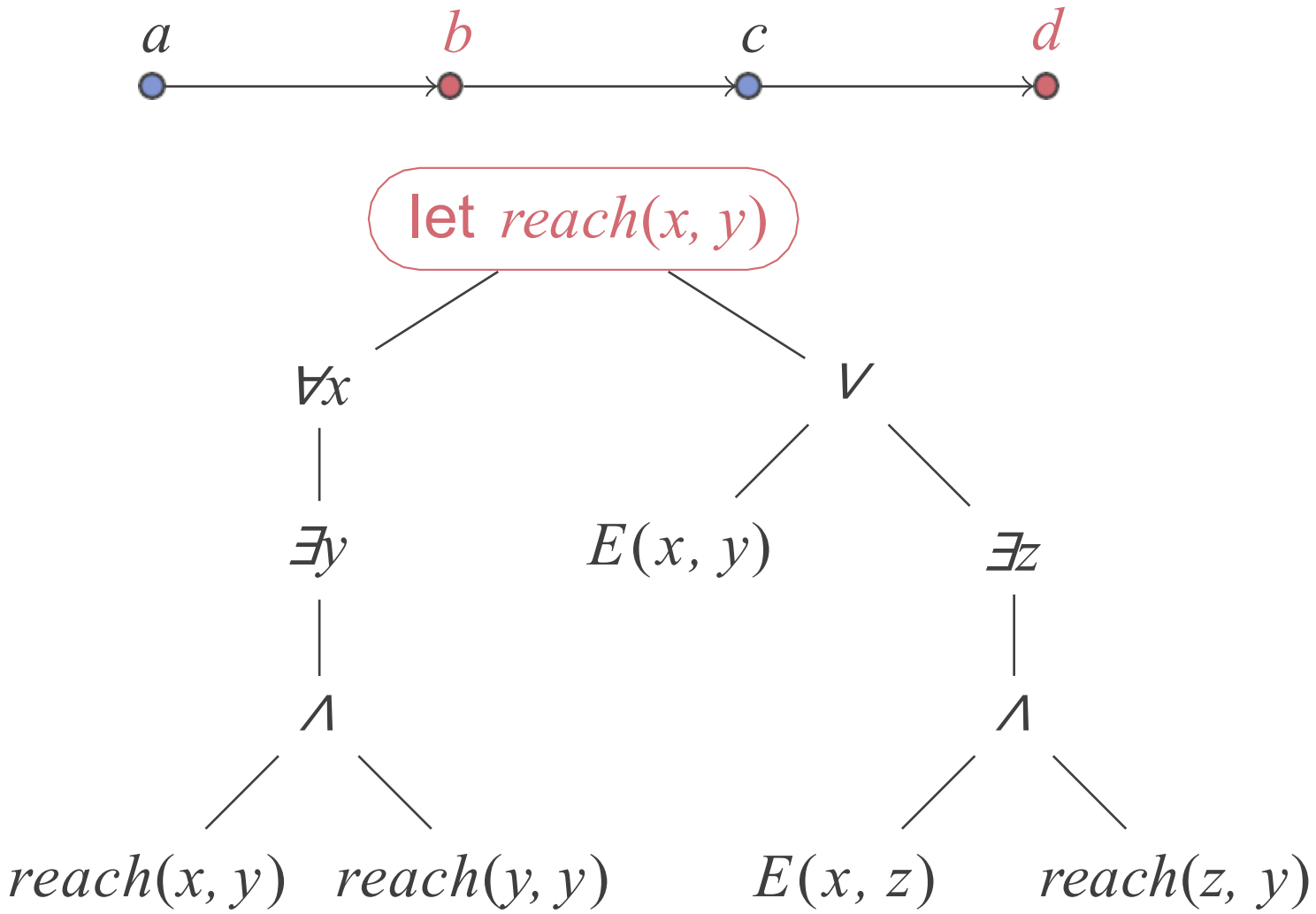
Evaluating Recursive Definitions



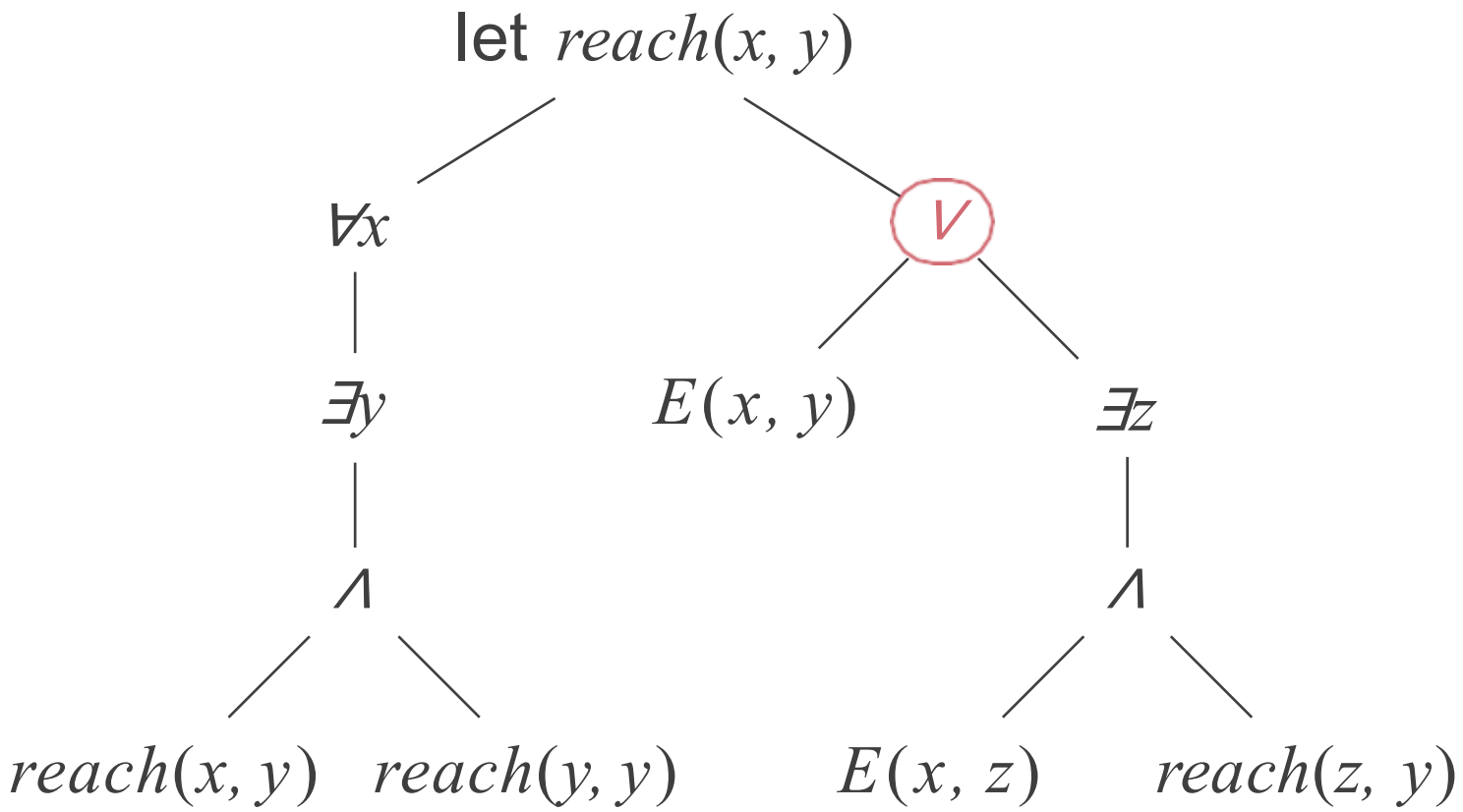
Evaluating Recursive Definitions



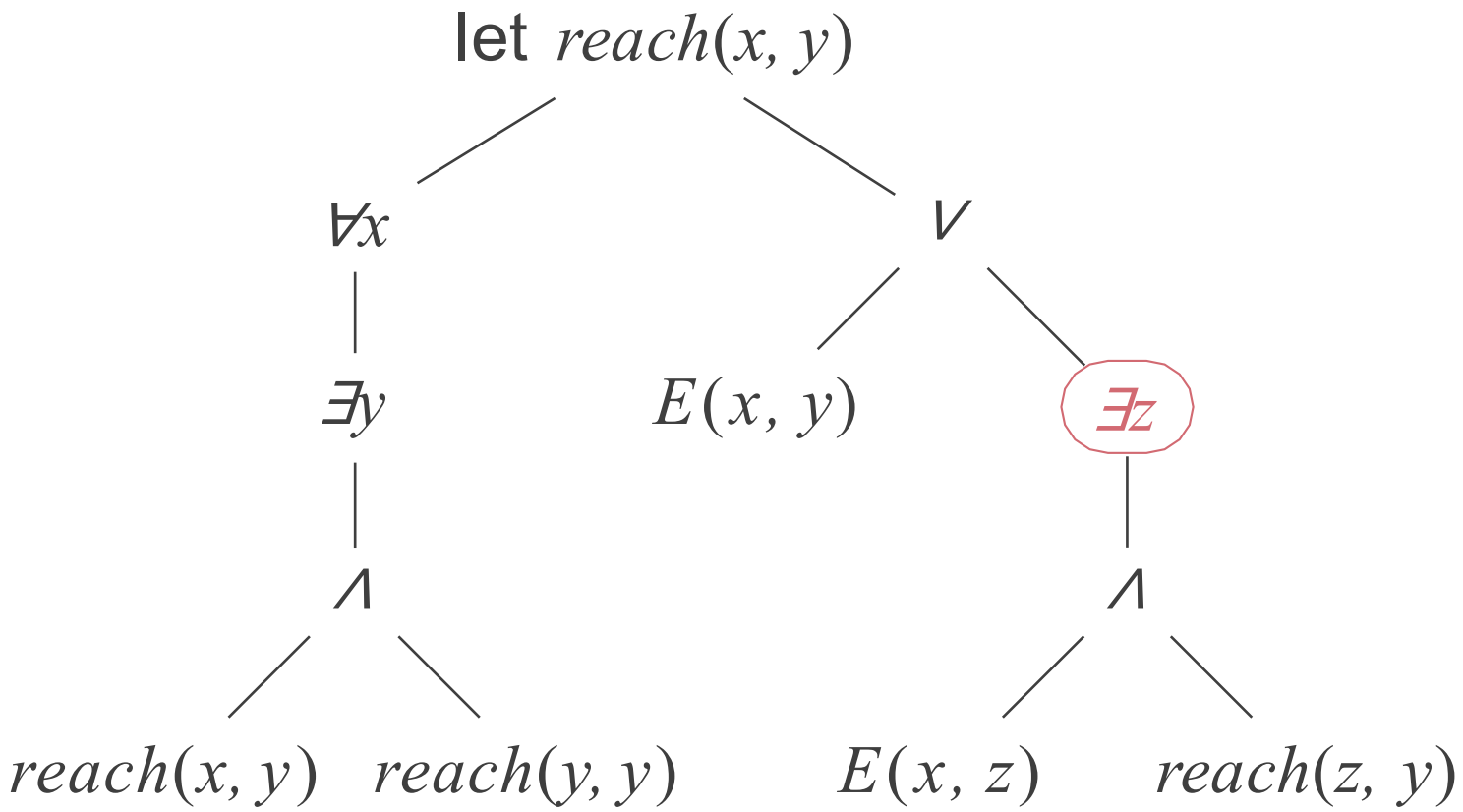
Evaluating Recursive Definitions



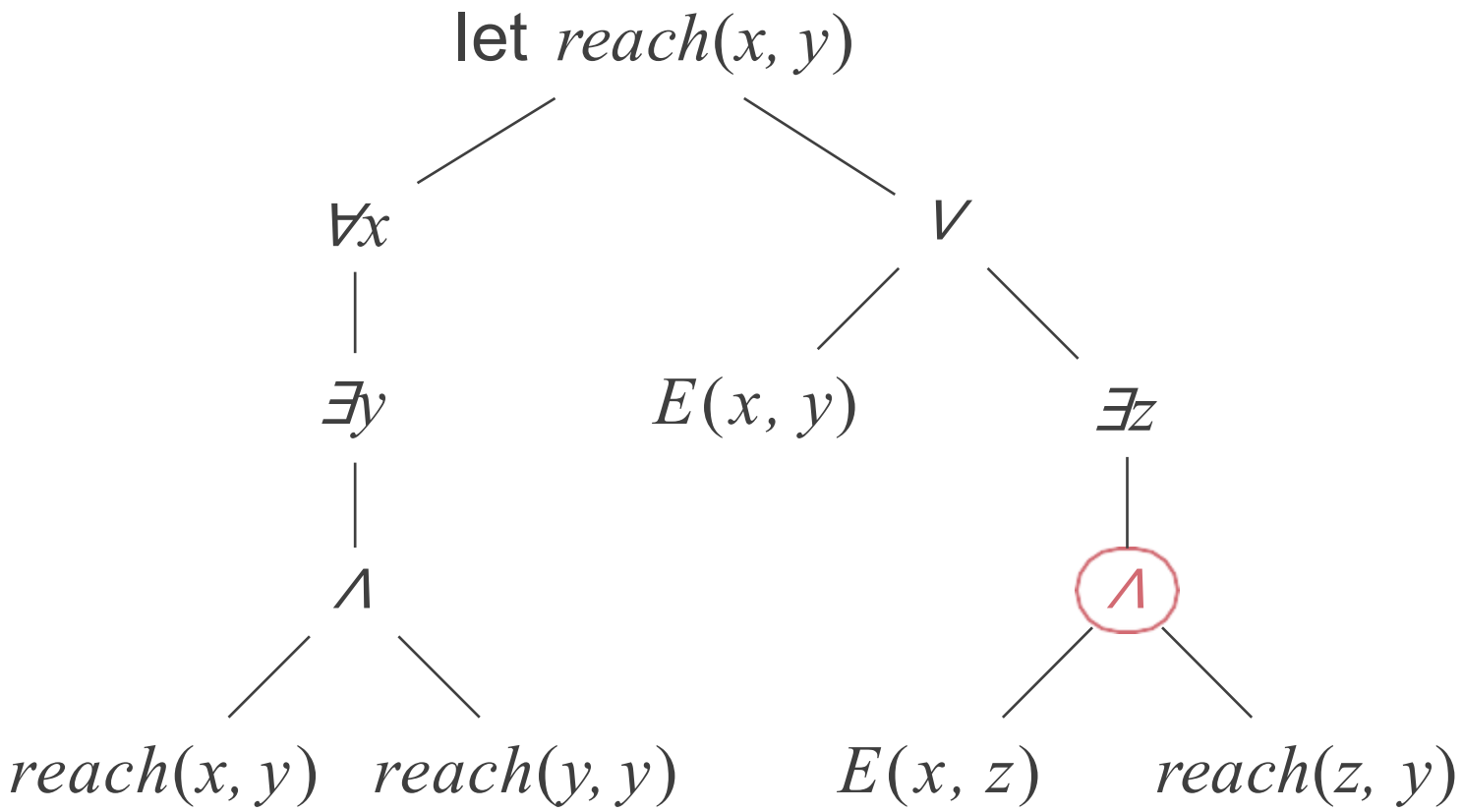
Evaluating Recursive Definitions



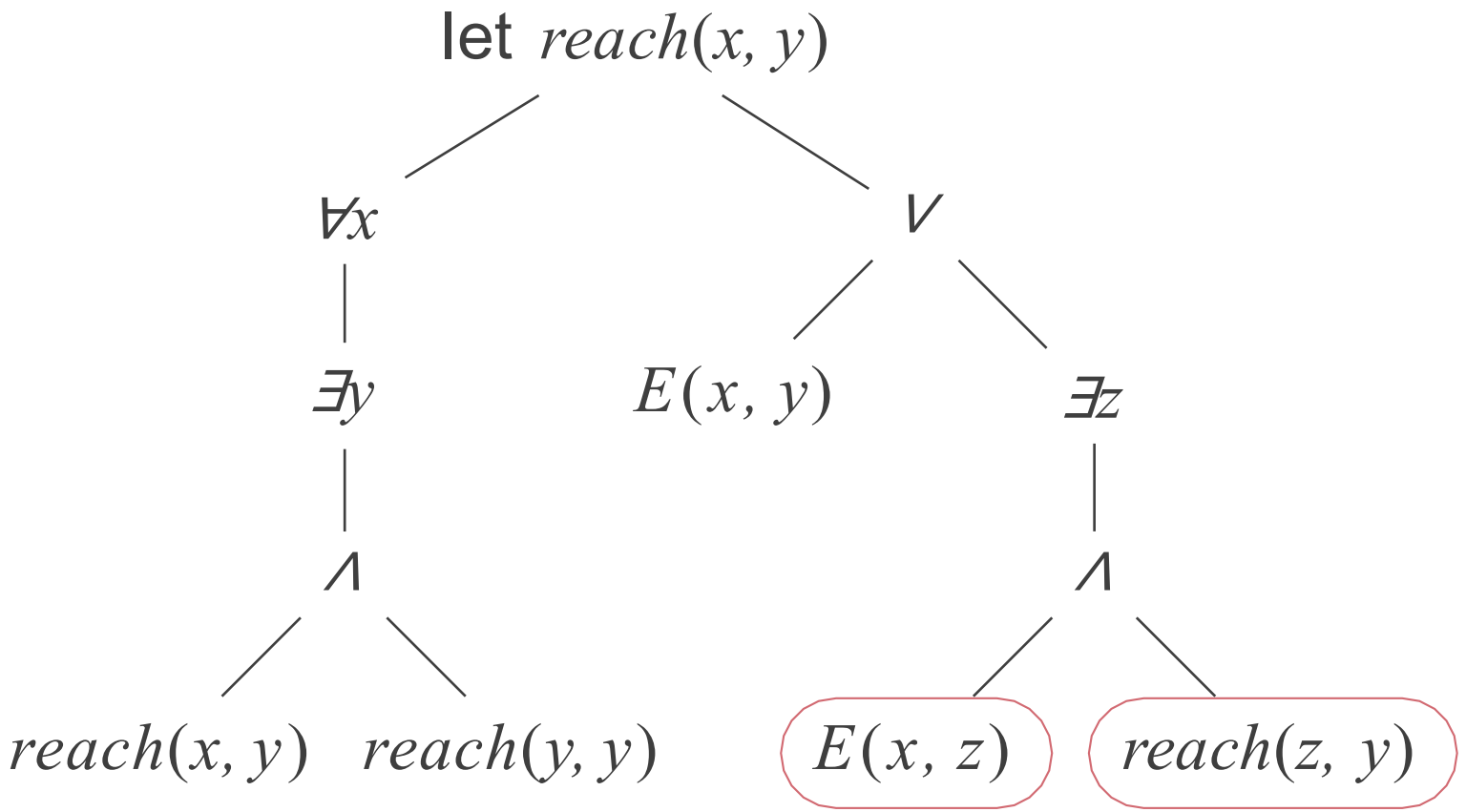
Evaluating Recursive Definitions



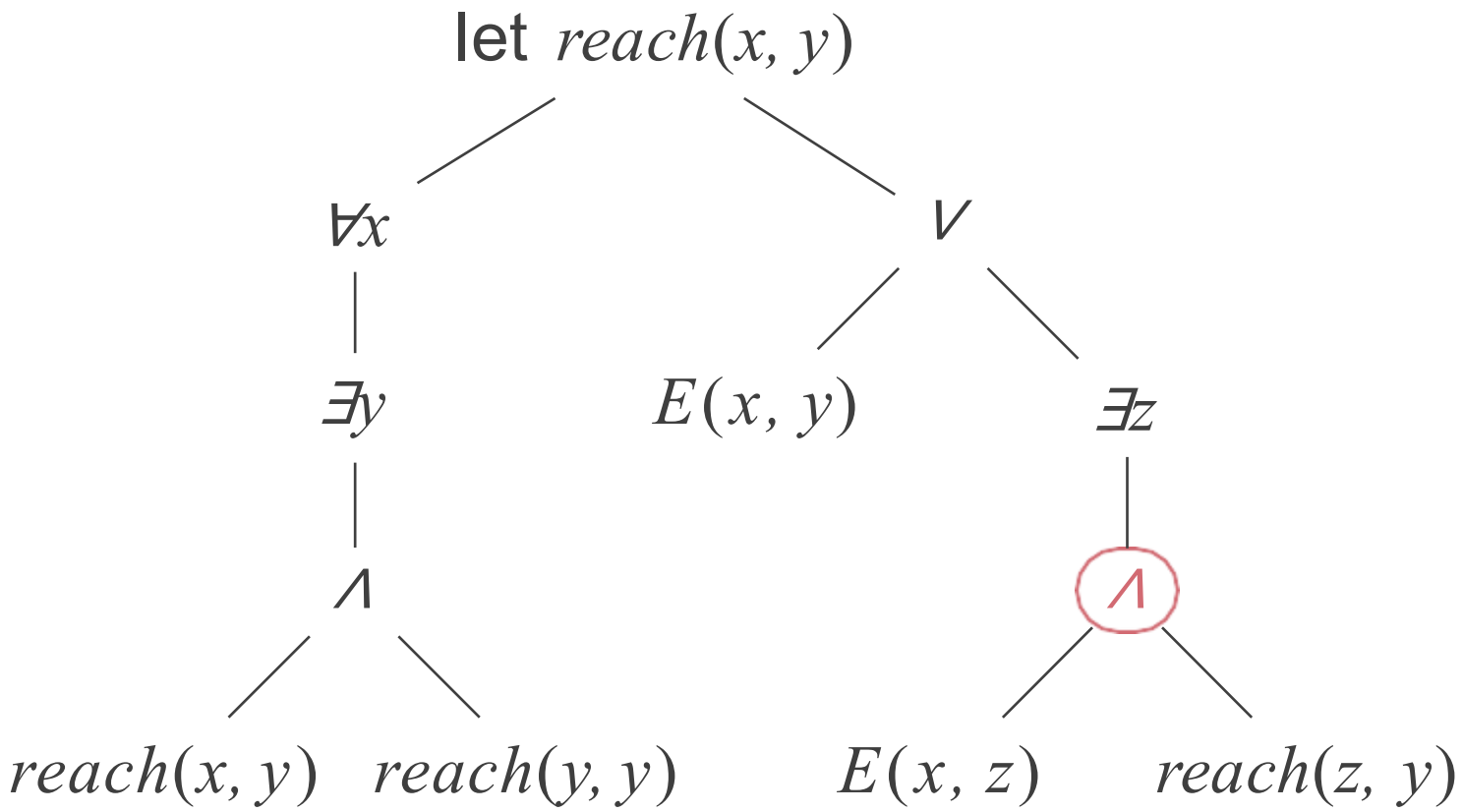
Evaluating Recursive Definitions



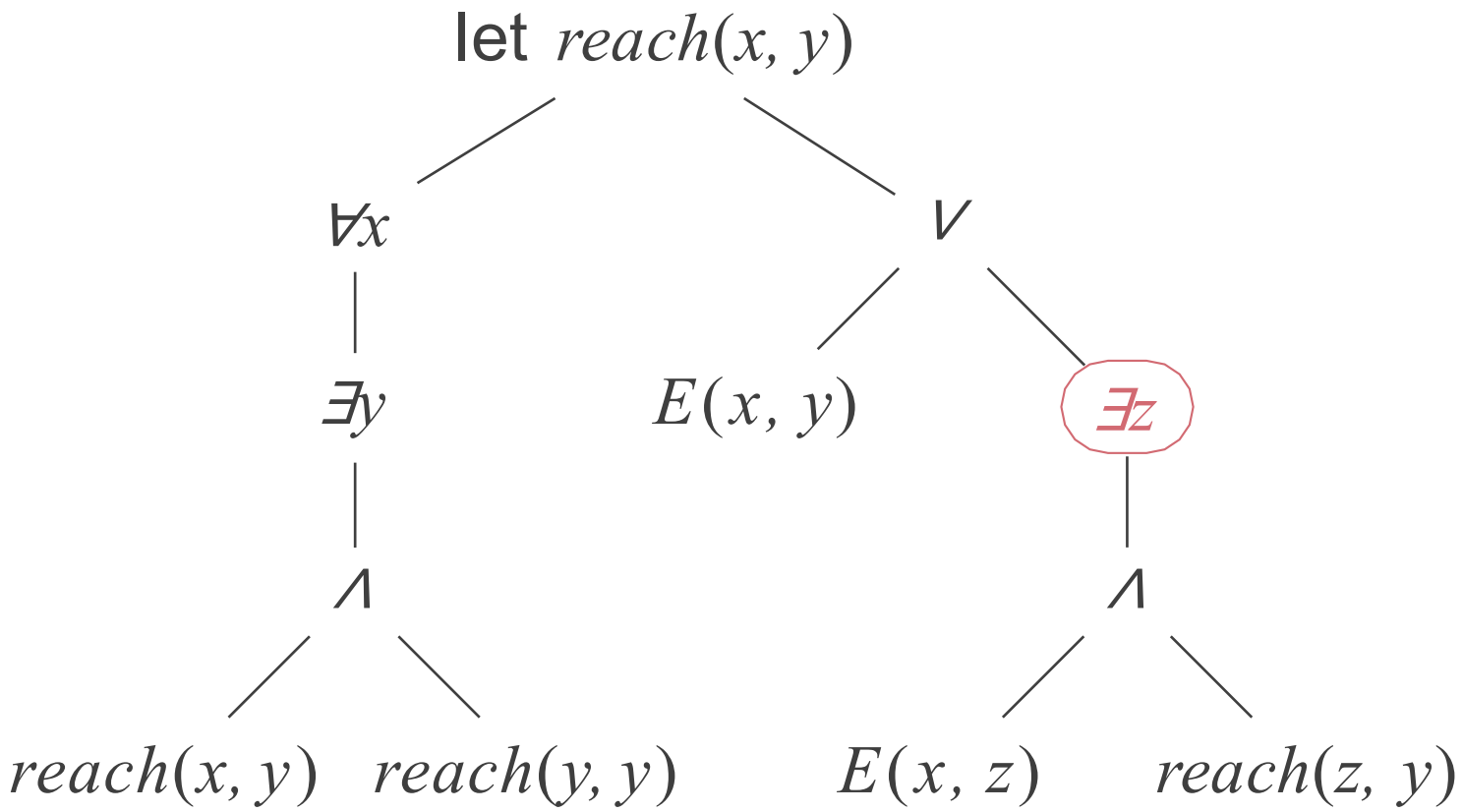
Evaluating Recursive Definitions



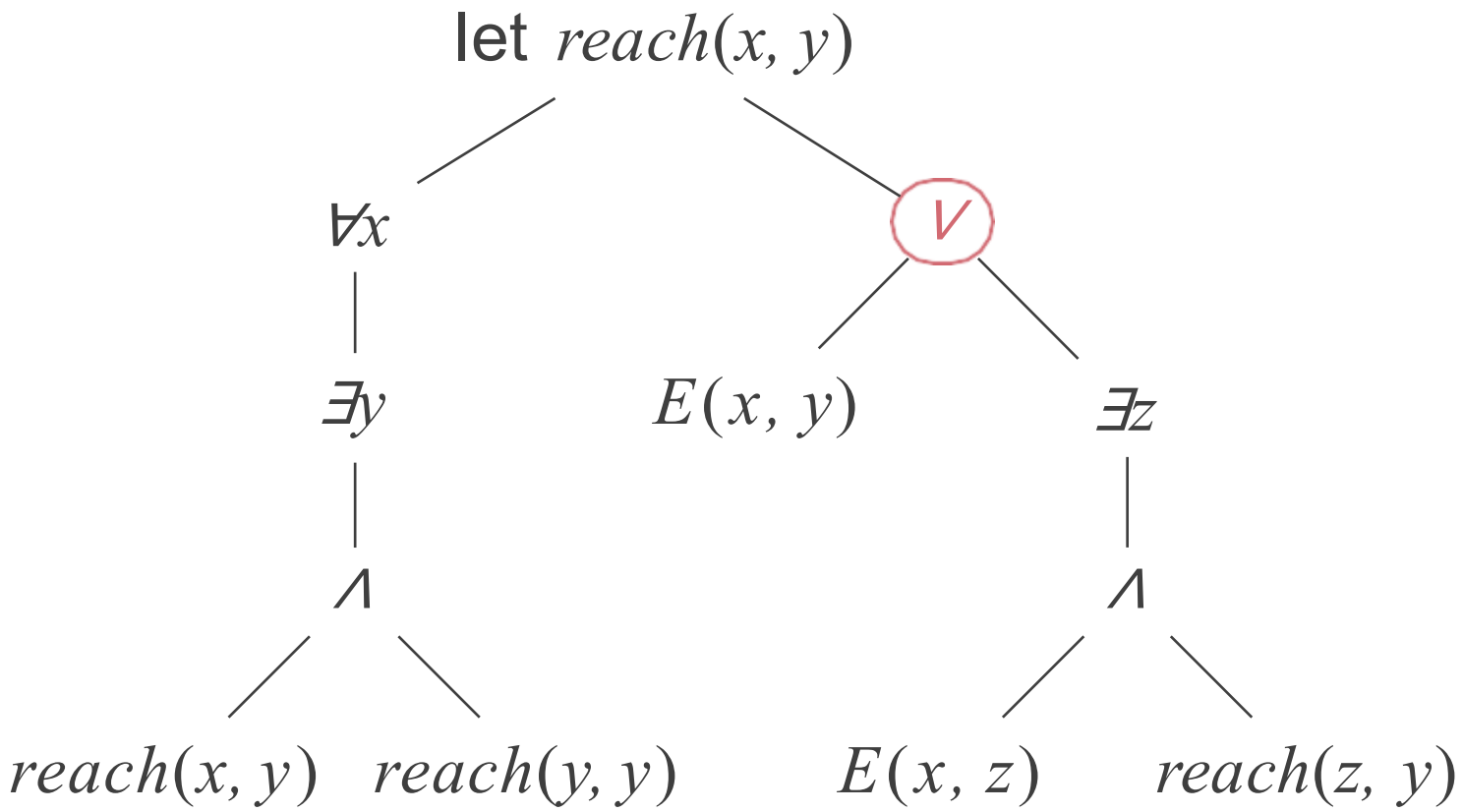
Evaluating Recursive Definitions



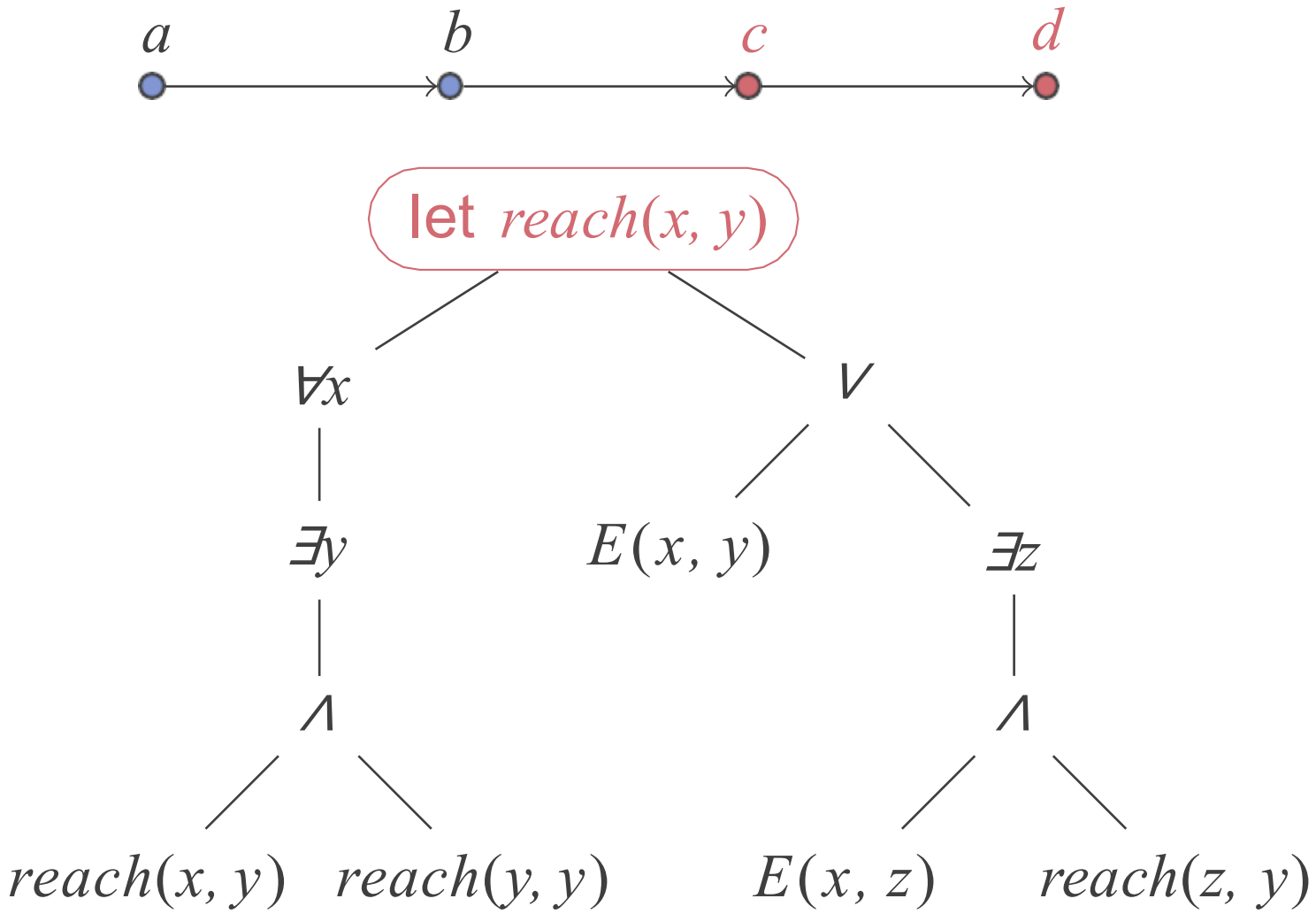
Evaluating Recursive Definitions



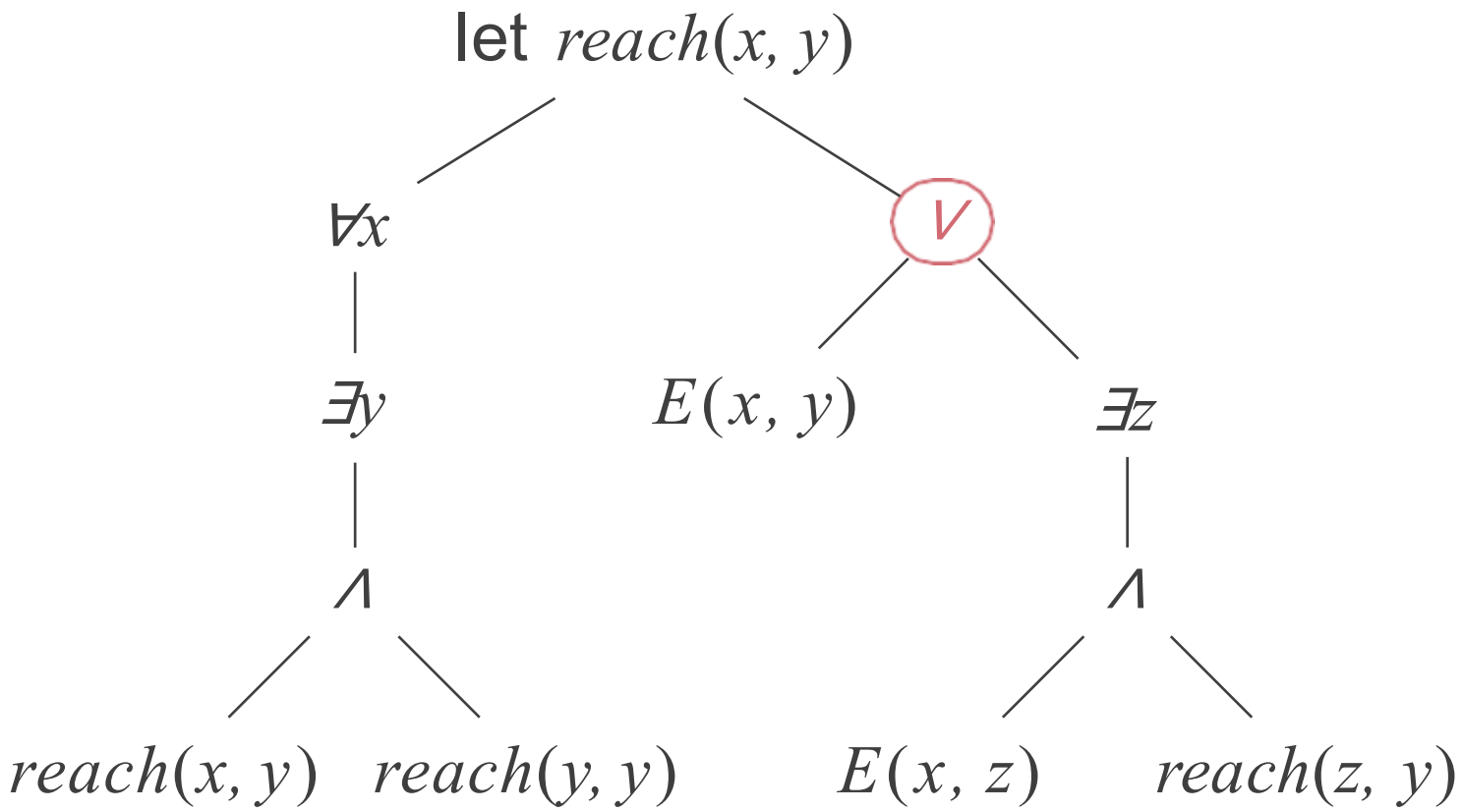
Evaluating Recursive Definitions



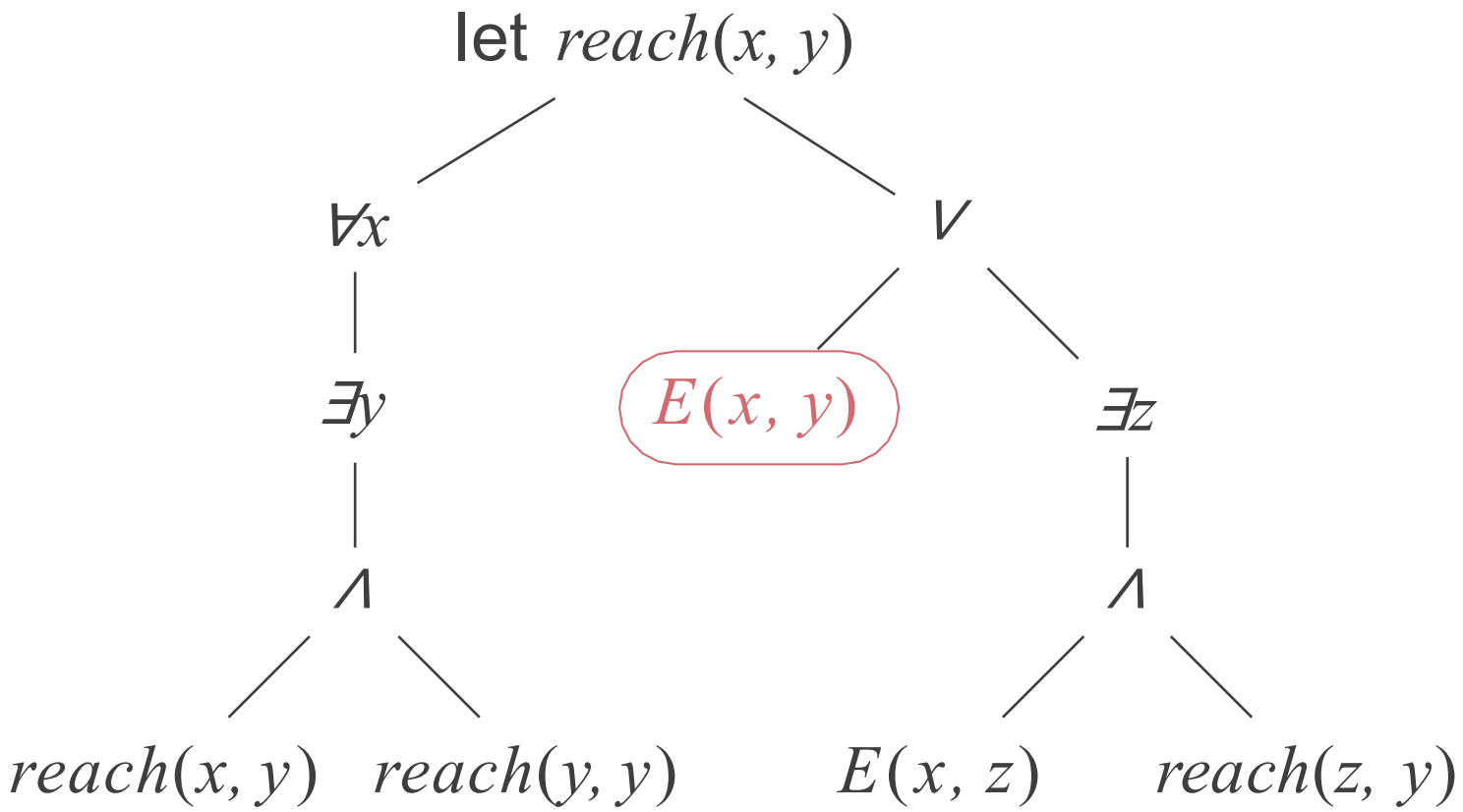
Evaluating Recursive Definitions



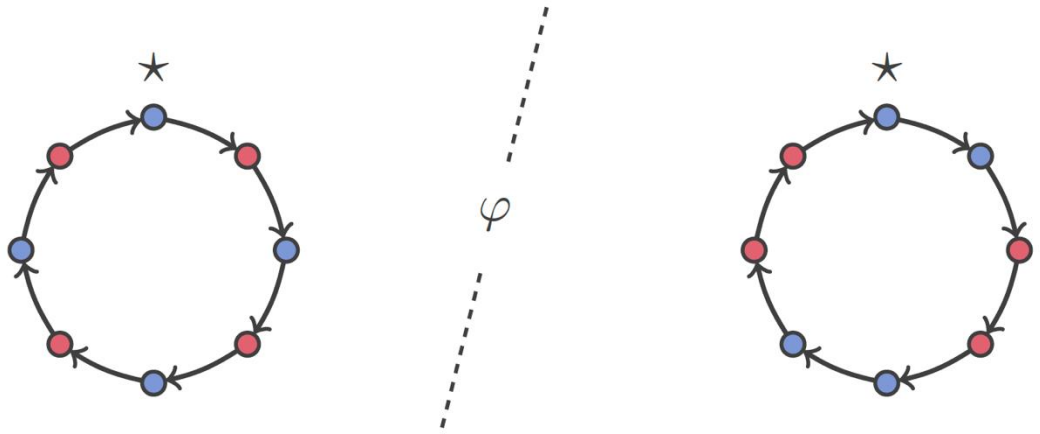
Evaluating Recursive Definitions



Evaluating Recursive Definitions



Mutual Recursion



$$
\begin{aligned}
\text{let } even(y) &= \exists z.\, odd(z) \wedge E(z,y) \vee \star(y) \\
odd(y) &= \exists z.\, even(z) \wedge E(z,y) \\
\text{in } &\forall x.\, (blue(x) \rightarrow even(x)) \wedge (red(x) \rightarrow odd(x))
\end{aligned}
$$

Mutual recursion evaluated similar to recursion

Product of counters (one for each definition in a block)

Languages with Decidable Learning: A Meta-theorem

PAUL KROGMEIER, University of Illinois, Urbana-Champaign, USA
P. MADHUSUDAN, University of Illinois, Urbana-Champaign, USA

Beyond logic learning?

Slides by Paul Krogmeier

$\lambda x.\ \texttt{concat}\ (\texttt{map}\ (\texttt{take}\ 1)\ (\texttt{words}\ x))$

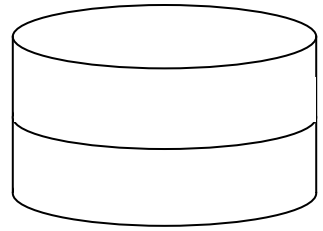
Chenan Wen	CW
Yizhan Qing	YQ
…	…

Program synthesis

$$\exists Q : \mathsf{Quorum}.\forall N_1, N_2 : \mathsf{Node}.$$
$$(\mathsf{leader}(N_1) \wedge \mathsf{member}(N_2, Q)) \rightarrow \mathsf{vote}(N_2, N_1)$$

Invariant inference

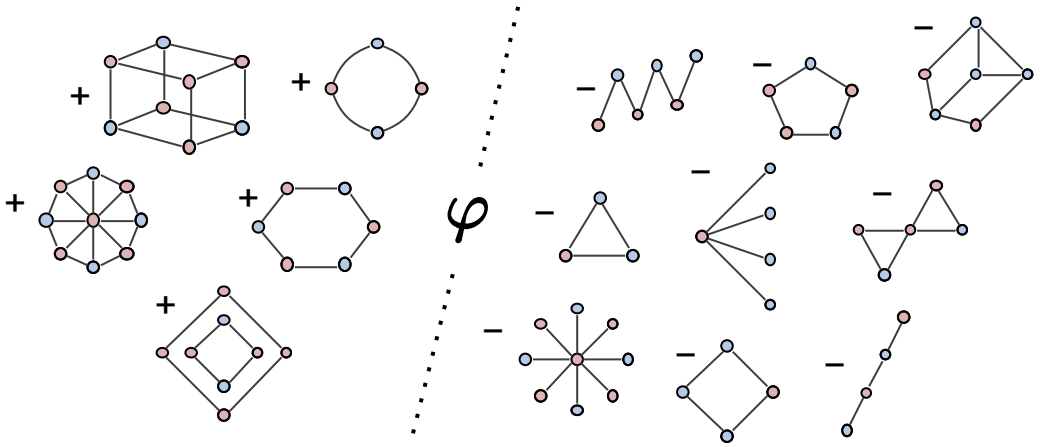
$$+(\texttt{id} = 0, \texttt{name} = \texttt{Alice}, \texttt{age} = 40)$$
$$-(\texttt{id} = 0, \texttt{name} = \texttt{Bob}, \texttt{age} = 19)$$



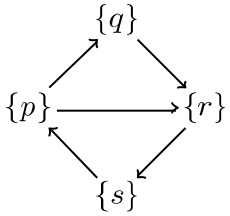
```
SELECT id, name, age
FROM employees
WHERE age >= 21;
```

Learning SQL queries

Example-driven logic learning



Synthesizing axiomatizations



$\Box\alpha \rightarrow \Diamond\alpha$

Synthesizing specifications

…

Algorithmic meta-theorem for symbolic learning problems

Interpreters → Meta-theorem → Algorithms for learning

Learning regular expressions

Find a regular expression that matches words on the left and none on the right.

paul@krogmeier.org

barry@123.edu

barry@self.co.uk

john@self.123

paul@krogmeier

`(LetterDigit+)@(LetterDigit+).(letter+)`

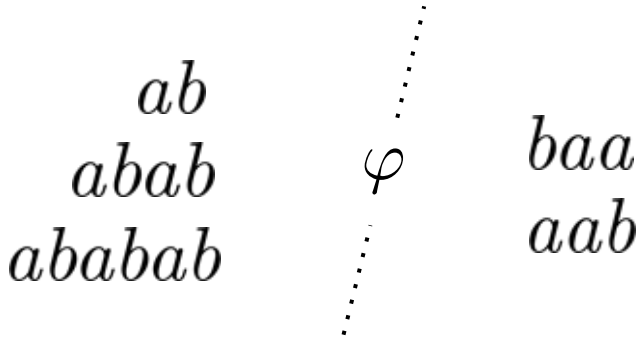
Concept: valid email addresses.

Can we find terminating algorithms?

Infinite number of semantically different expressions

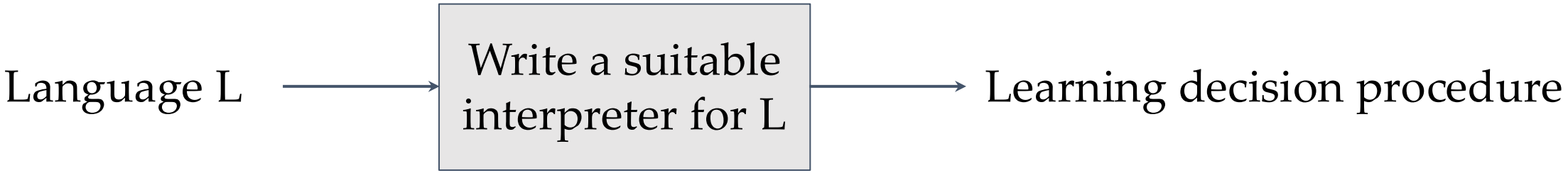
- **Naive enumeration won't work**

Don't overfit!



- Good: $(ab)^*$ Bad: $ab + abab + ababab$
- Grammar is crucial

Meta-theorem



$$\texttt{Interpreter}(\texttt{Structure}, \texttt{Expression}) \in \{\texttt{true}, \texttt{false}\}$$

$$\texttt{FOL}(\texttt{graph}, \varphi) = \texttt{true} \quad \text{iff} \quad \texttt{graph} \models \varphi$$

$$\texttt{Reg}(\texttt{word}, \texttt{regexp}) = \texttt{true} \quad \text{iff} \quad \texttt{word} \in L(\texttt{regexp})$$

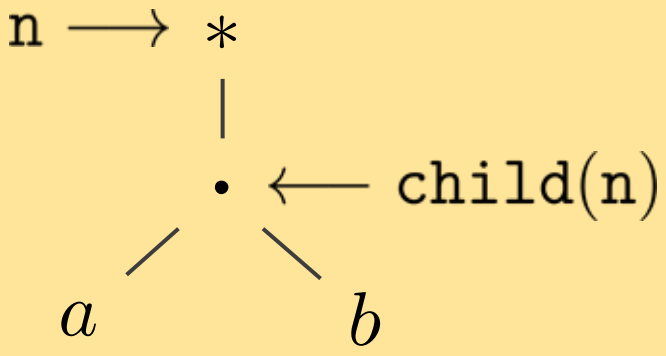
FACET: a programming language for interpreters

$$
\begin{array}{lll}
\mathit{Prog} & ::= & \{\ \mathit{Clause}\ \ldots\ \mathit{Clause}\ \} \\
\mathit{Clause} & ::= & \texttt{P(M, } \sigma\texttt{(}z\texttt{), n)} = \texttt{match label(n) with } \mathit{Cases} \\
\mathit{Cases} & ::= & \alpha_1\texttt{(}z\texttt{)} \rightarrow e_1\ \ldots\ \alpha_n\texttt{(}z\texttt{)} \rightarrow e_n
\end{array}
$$

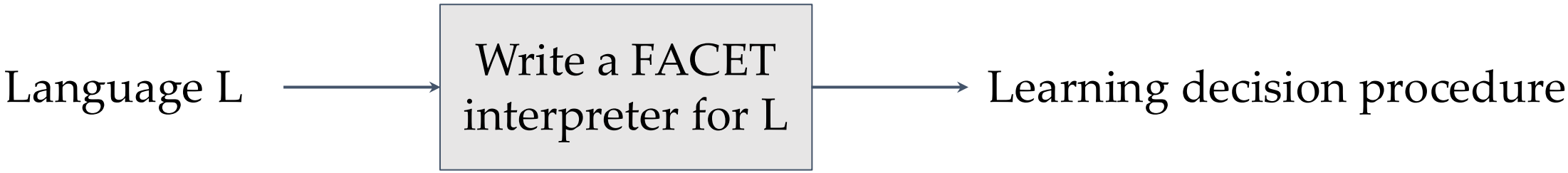
$$
\begin{array}{lll}
e ::= \texttt{True} & \mid \texttt{False} & \mid f\texttt{(}z\texttt{)} \\
\quad \mid e_1 \texttt{ and } e_2 & \mid e_1 \texttt{ or } e_2 & \mid \texttt{P(M, } \sigma\texttt{(}z\texttt{), } \mathit{dir}\texttt{(n))} \\
\quad \mid \texttt{all } (\lambda x.\ e)\ g\texttt{(}z\texttt{)} & \mid \texttt{any } (\lambda x.\ e)\ g\texttt{(}z\texttt{)} & \mid \texttt{if } f\texttt{(}z\texttt{) then } e_1 \texttt{ else } e_2
\end{array}
$$

$\alpha\texttt{(}z\texttt{)} \in \mathit{pat}(\Delta)$ $\quad$ $\sigma\texttt{(}z\texttt{)} \in \mathit{pat}(\mathsf{Asp})$ $\quad$ $f \in B,\ g \in S$ $\quad$ $\mathit{dir} \in \{\texttt{parent,child1,...,childk}\}$

Programs in FACET traverse syntax trees using pointers.



Meta-theorem



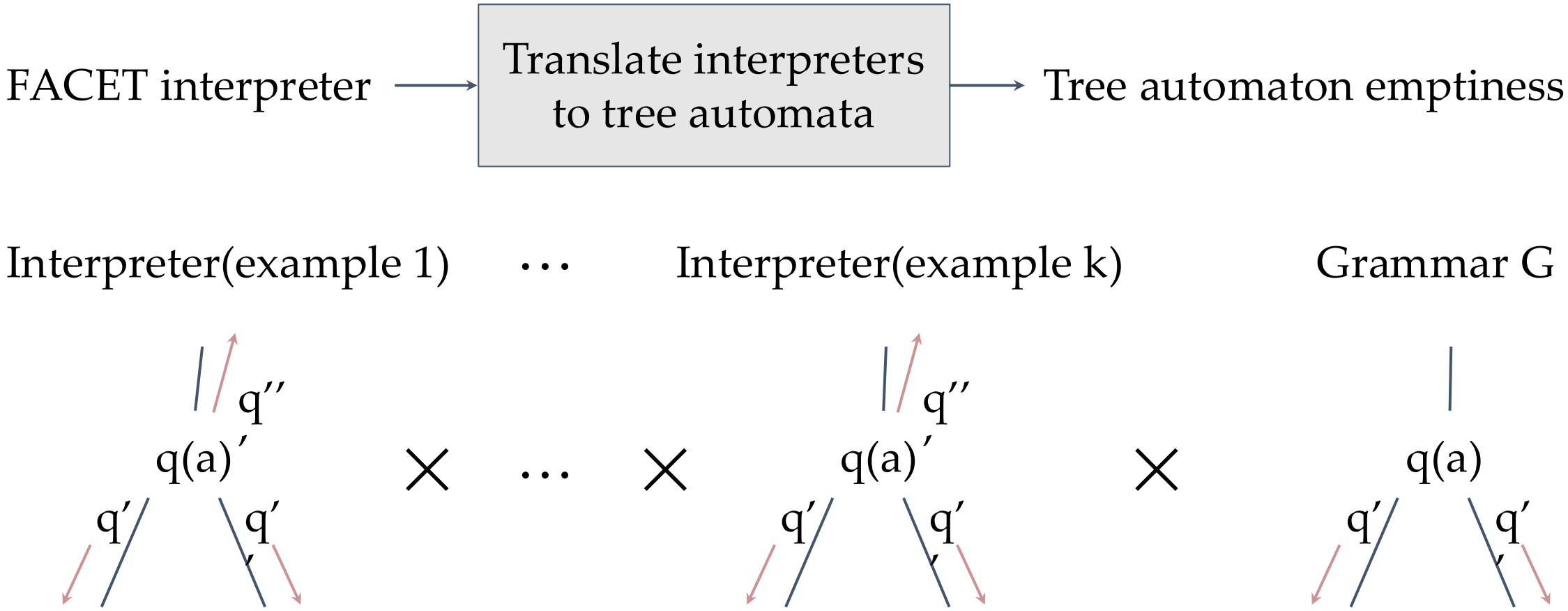
Crucial: interpreter memory is independent of expression size.

$$\texttt{Interpreter}(\texttt{Structure}, \texttt{Memory}, \texttt{Expression}) \in \{\texttt{true}, \texttt{false}\}$$

$$\cap$$

$$|\texttt{Total Memory}| = f(|\texttt{Structure}|)$$

Proof: translate interpreters to tree automata



Example: learning regular expressions

Write a program $\texttt{Reg}(w, e)$ that computes whether w matches e.

$$\texttt{Reg}(w, e) = \texttt{True} \quad \text{iff} \quad w \in L(e)$$

$$\texttt{Reg}(ababab, (ab)^*) = \text{True}$$
$$\texttt{Reg}(aab, (ab)^*) = \text{False}$$

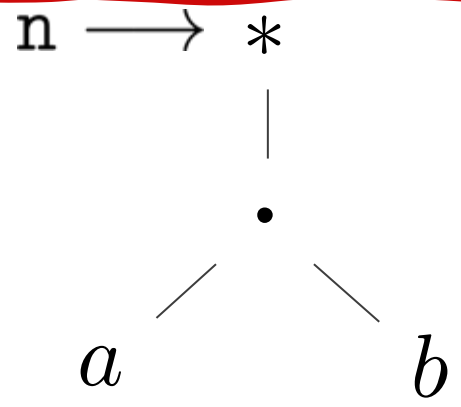
What memory to use?

Remember which **subword** to check. Indicate with $(\texttt{l}, \texttt{r})$.

$$ababab(\texttt{2}, \texttt{5}) = bab$$

Regular expression interpreter

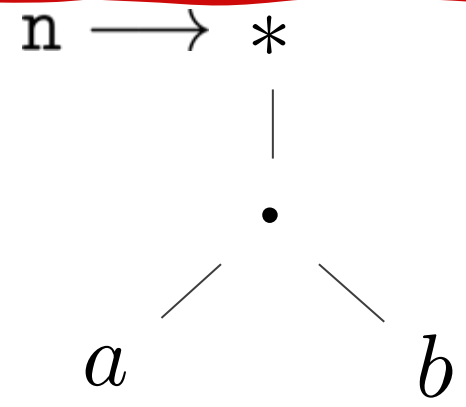
$\texttt{Reg}(\texttt{w}, (\texttt{l}, \texttt{r}), \texttt{n}) = \texttt{True}$ iff $\texttt{w}(\texttt{l}, \texttt{r}) \in L(regexp(\texttt{n}))$




```
Reg(w, (l,r), n) =
 match label(n) with
   *  →  if (l = r) then True
         else
           any (λy. Reg(w, (l,y), child1(n))
                and Reg(w, (y,r), n)) [l+1,r]

   •  → any (λy. Reg(w, (l,y), child1(n))
             and Reg(w, (y,r), child2(n))) [l,r]

   x  → r = l+1 and w(l) = x
```

Regular expression interpreter

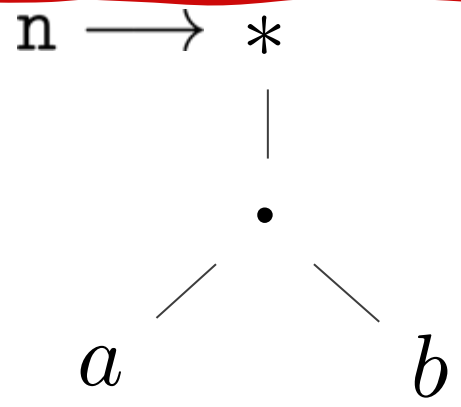
$\texttt{Reg}(\texttt{w}, (\texttt{l}, \texttt{r}), \texttt{n}) = \texttt{True}$ iff $\texttt{w}(\texttt{l}, \texttt{r}) \in L(regexp(\texttt{n}))$




```
Reg(w, (l,r), n) =
 match label(n) with
  *  →  if (l = r) then True
        else
          any (λy. Reg(w, (l,y), child1(n))
               and Reg(w, (y,r), n)) [l+1,r]

  •  → any (λy. Reg(w, (l,y), child1(n))
            and Reg(w, (y,r), child2(n))) [l,r]

  x  → r = l+1 and w(l) = x
```

Regular expression interpreter

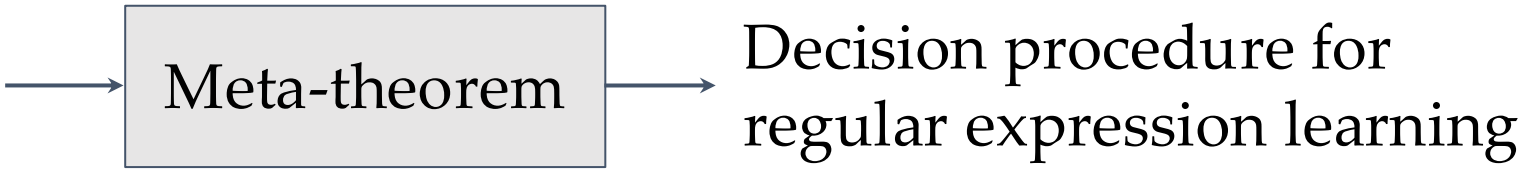
$\texttt{Reg}(\texttt{w}, (\texttt{l}, \texttt{r}), \texttt{n}) = \texttt{True}$ iff $\texttt{w}(\texttt{l}, \texttt{r}) \in L(regexp(\texttt{n}))$




```
Reg(w, (l,r), n) =
 match label(n) with
  *  →  if (l = r) then True
        else
          any (λy. Reg(w, (l,y), child1(n))
               and Reg(w, (y,r), n)) [l+1,r]

  •  → any (λy. Reg(w, (l,y), child1(n))
            and Reg(w, (y,r), child2(n))) [l,r]

  x  → r = l+1 and w(l) = x
```

Bird's eye view

```
Reg(w, (l,r), n) =
 match label(n) with
   *  →  if (l = r) then True
         else
           any (λy. Reg(w, (l,y), child1(n))
                and Reg(w, (y,r), n)) [l+1,r]

   .  → any (λy. Reg(w, (l,y), child1(n))
             and Reg(w, (y,r), child2(n))) [l,r]

   x  → r = l+1 and w(l) = x
```



Decidability results obtained by writing interpreters.

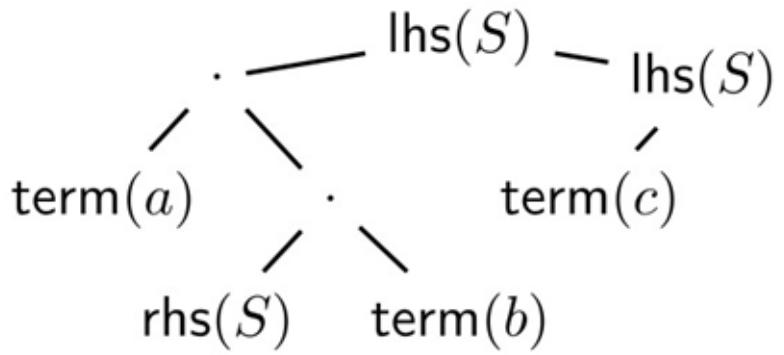
Algorithms based on tree automaton emptiness.

For which languages have we proved decidable learning?

Learning context-free grammars

$aaaacbbbb$

$S \longrightarrow aSb \mid c$



Memory: subwords + counter + parsing information

Learning functional programs

"International Business Machines" $\longrightarrow$ "IBM" $\mathsf{Loop}(\lambda\, w.\, \mathsf{Concat}(\mathsf{SubStr}(in, \mathsf{Upper}, w)))$

Decidable learning in an extension of the DSL for FlashFill (Gulwani 11)

Memory: subwords + bounded counters + finite maps

Results

- Linear temporal logic on infinite periodic words
- Modal logic on Kripke structures
- Computation tree logic on Kripke structures
- First-order queries over rational numbers with order
- Regular expressions on finite words
- Context-free grammars on finite words
- Functional programs on strings (extension of FlashFill's DSL)

Explains recent results on learning in finite-variable logics (POPL 22)