Synthesizing String Transformations from Input-Output Examples

Xiao Jia

2013 May 29

Motivation

From:

```
Ey = Sy + Ey;
Ez = Sz + Ez;
Eyw = Syw + Eyw;
Ef = Sf + Ef;
Efw = Sfw * Efw;
```

To:

```
Ey[i] += Sy;
Ez[i] += Sz;
Eyw[i] += Syw;
Ef[i] += Sf;
Efw[i] *= Sfw;
```

Motivation

From:

```
Ey = Sy + Ey;
Ez = Sz + Ez;
Eyw = Syw + Eyw;
Ef = Sf + Ef;
Efw = Sfw * Efw;
```

To:

```
Ey[i] += Sy;
Ez[i] += Sz;
Eyw[i] += Syw;
Ef[i] += Sf;
Efw[i] *= Sfw;
```

Choice 1: Manually edit the 5 lines

Choice 2: Write a specific program to transform the 5 lines

Choice 3: /usr/bin/sed

Motivation

Choice 1: Manually edit the 5 lines …

… boring and time-consuming

Choice 2: Write a specific program …

… not reusable

… debugging takes longer time

Choice 3: /usr/bin/sed …

… require expertise

Motivation

Choice 4: Programming by Example (PbE)

```
Ey = Sy + Ey;
Ez = Sz + Ez;
Eyw = Syw + Eyw;
Ef = Sf + Ef;
Efw = Sfw * Efw;
```

Motivation

Choice 4: Programming by Example (PbE)

```
Ey = Sy + Ey;
Ez = Sz + Ez;
Eyw = Syw + Eyw;
Ef = Sf + Ef;
Efw = Sfw * Efw;
```

Motivation

Choice 4: Programming by Example (PbE)

```
Ey[i] += Sy;
Ez = Sz + Ez;
Eyw = Syw + Eyw;
Ef = Sf + Ef;
Efw = Sfw * Efw;
```

Motivation

Choice 4: Programming by Example (PbE)

```
Ey[i] += Sy;
Ez = Sz + Ez;
Eyw = Syw + Eyw;
Ef = Sf + Ef;
Efw = Sfw * Efw;
```

→

```
Ey = Sy + Ey;
Ey[i] += Sy;
```

Motivation

Choice 4: Programming by Example (PbE)

```
Ey[i] += Sy;
Ez = Sz + Ez;
Eyw = Syw + Eyw;
Ef = Sf + Ef;
Efw = Sfw * Efw;
```

```
Ey = Sy + Ey;
Ey[i] += Sy;
```

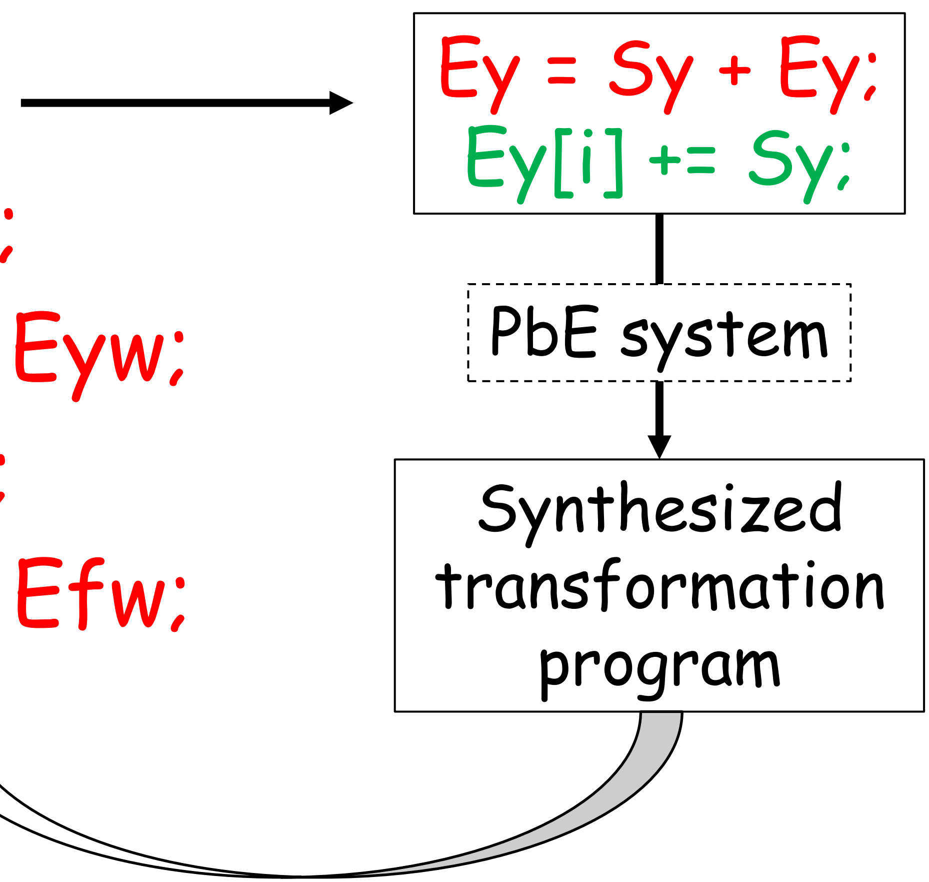
PbE system

Synthesized transformation program

Motivation

Choice 4: Programming by Example (PbE)

```
Ey[i] += Sy;
Ez = Sz + Ez;
Eyw = Syw + Eyw;
Ef = Sf + Ef;
Efw = Sfw * Efw;
```



Motivation

Choice 4: Programming by Example (PbE)

```
Ey[i] += Sy;
Ez[i] += Sz;
Eyw[i] += Syw;
Ef[i] += Sf;
Efw[i] *= Sfw;
```

generalize to new operators
not shown in the example

```
Ey = Sy + Ey;
Ey[i] += Sy;
```

PbE system

Synthesized transformation program

Programming by Example

- Input:
  user intent expressed in the form of input-output examples

- Output:
  a program which is <u>consistent</u> with the given IO examples

PbE v.s. PbD

- PbE: Programming by Example
  - user only provides <u>final</u> state (the output)
- PbD: Programming by Demonstration
  - user provides <u>intermediate</u> states (e.g. traces)

Ey = Sy + Ey;	Ey[i] += Sy + Ey;	Ey[i] += Syy;
Ey[= Sy + Ey;	Ey[i] += Sy+ Ey;	Ey[i] += Sy;
Ey[i = Sy + Ey;	Ey[i] += Sy Ey;	
Ey[i] = Sy + Ey;	Ey[i] += SyEy;	

PbE v.s. PbD

- Traces provide more information of user intent than just input-outputs, but <u>it's very hard to make use of the extra information</u>

- For a given pair of input and output, there are infinite number of consistent traces, most of which are <u>noises</u>

PbE: Previous Approaches

- Gap programs (POPL 1984)
- Substring extraction (POPL 2011)
- Guided search of function compositions (ICML 2013)

Gap Programs

Robert P. Nix, Editing by example.

A gap program: pattern => replacement

Both patterns and replacements are lists of constants and gaps

Gap Programs

Nix's EBE system synthesizes patterns by calculating the longest common subsequence

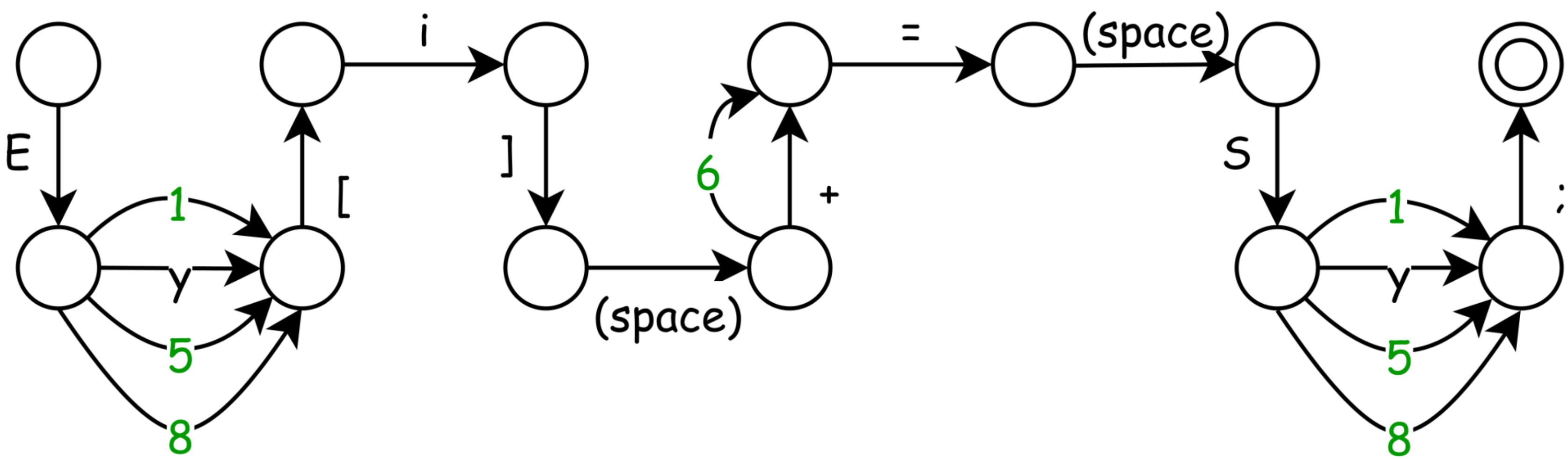
lcs(
```
Ey = Sy + Ey;
Ez = Sz + Ez;
Eyw = Syw + Eyw;
Ef = Sf + Ef;
Efw = Sfw * Efw;
```
) = "E = S E;"

then the inputs are parsed into a table

1	2	3	4	5	6	7	8
y	ε	ε	ε	y	+	ε	y
z	ε	ε	ε	z	+	ε	z
yw	ε	ε	ε	yw	+	ε	yw
f	ε	ε	ε	f	+	ε	f
fw	ε	ε	ε	fw	*	ε	fw

Gap Programs

Input parses are used to construct replacement automata



1	2	3	4	5	6	7	8
y	ε	ε	ε	y	+	ε	y

The final replacement automaton is the intersection of the automata constructed from the parses of all examples

Substring Extraction

The problem of generating the output string can be decoupled into independent sub-problems of generating different parts of the output string.

In a sub-problem, some substrings of the input are extracted and combined in a new way to generate the output.

Substring Extraction

Gap programs locate input substrings according to characters in the LCS

Observation: *A* substring is determined by two positions ➔ quadratic number of sub-problems

Gap patterns can be improved by locating positions using regular expressions

More than substring extraction

Sumit Gulwani, Automating String Processing in Spreadsheets Using Input-Output Examples, POPL 2011.

Conditionals & Loops

However, program sizes should be less than 20~30, or it will be very very slow.

Search Function Compositions

Treat transformation as the composition of arbitrary functions

Input		Output
Anthony Hopkins Al Pacino Tom Hanks Tom Hanks Nicolas Cage	→	Anthony Hopkins (1) Al Pacino (1) Tom Hanks (2) Nicolas Cage (1)

```
output = join(dedup(concatLists(x, " ",
                     concatLists("(", count(x, x), ")"))),
              "\n")
where
              x = split(input, "\n")
```

Guided Search of Compositions

Certain textual features can help **bias** the search process by providing clues about which functions may be relevant:

- there are duplicate lines in the input but not output, suggesting that dedup may be useful
- there are many more spaces in the output than the input, suggesting that " " may be useful
- …

Anthony Hopkins
Al Pacino
Tom Hanks
Tom Hanks
Nicolas Cage

→

Anthony Hopkins (1)
Al Pacino (1)
Tom Hanks (2)
Nicolas Cage (1)

PbE: Previous Approaches

Gap programs (POPL 1984)

➔ separate parsing and output generation

Substring extraction (POPL 2011)

➔ decide positions by using RE's

Guided search of function compositions (ICML 2013)

➔ allow arbitrary transformation functions

PbE: Previous Approaches

Gap programs (POPL 1984)

➔ very limited expressibility; fails in most test cases

Substring extraction (POPL 2011)

➔ very very slow; does <u>not</u> work well in Excel 2013

Guided search of function compositions (ICML 2013)

➔ unable to handle infinite streams; requires <u>knowledge</u>

Our Approach

Finite-state transducers …

There are two types of finite-state machines:

1. Finite-state automata/acceptors (only an input tape; accept or not)
2. Finite-state transducers (an input tape and an output tape)

Our Approach

Finite-state transducers …

… extended with <u>registers</u> …

Think about multi-stack pushdown automata, or multi-tape Turing machines

Registers are <u>infinite</u> buffers for different parts/substrings of the input

Our Approach

Finite-state transducers …

… extended with registers …

… whose transitions include …

(r1,r2) where r1 and r2 are restricted RE's

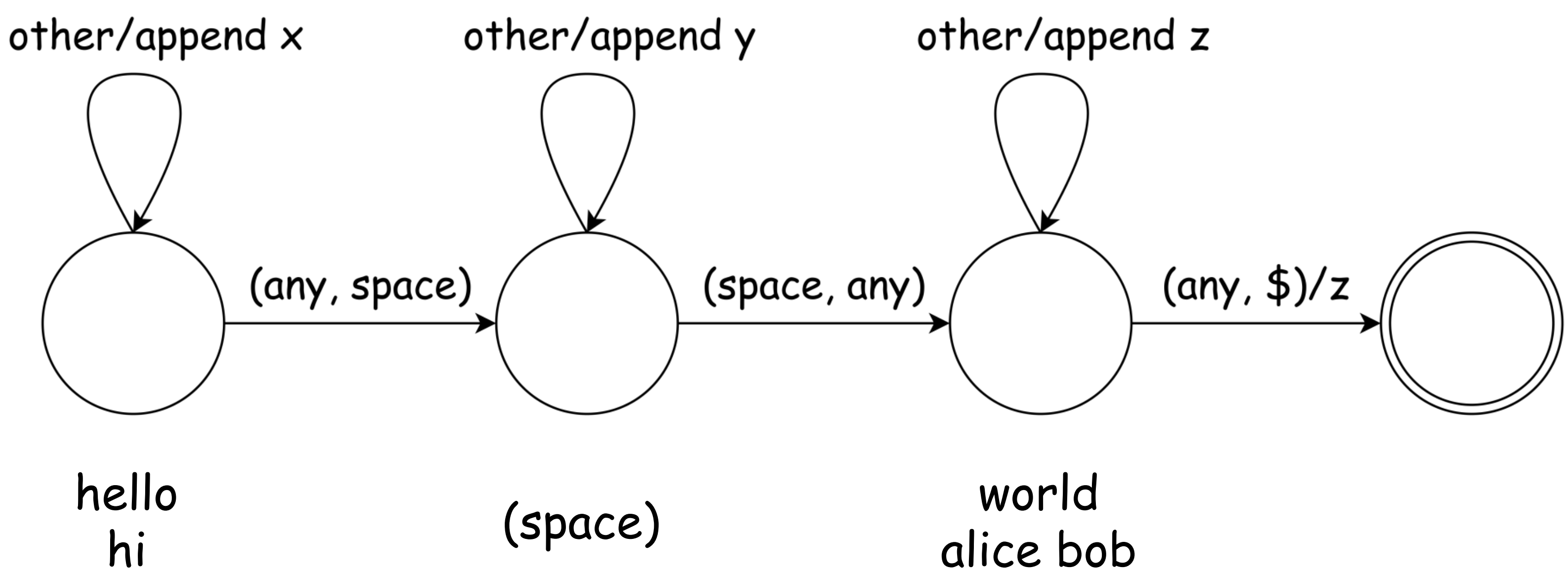
. pdf

• (lowercase letters)…

Transducer Example

hello world$ ➔ world

hi alice bob$ ➔ alice bob



Our Approach

Finite-state transducers …

… extended with registers …

… whose transitions include …

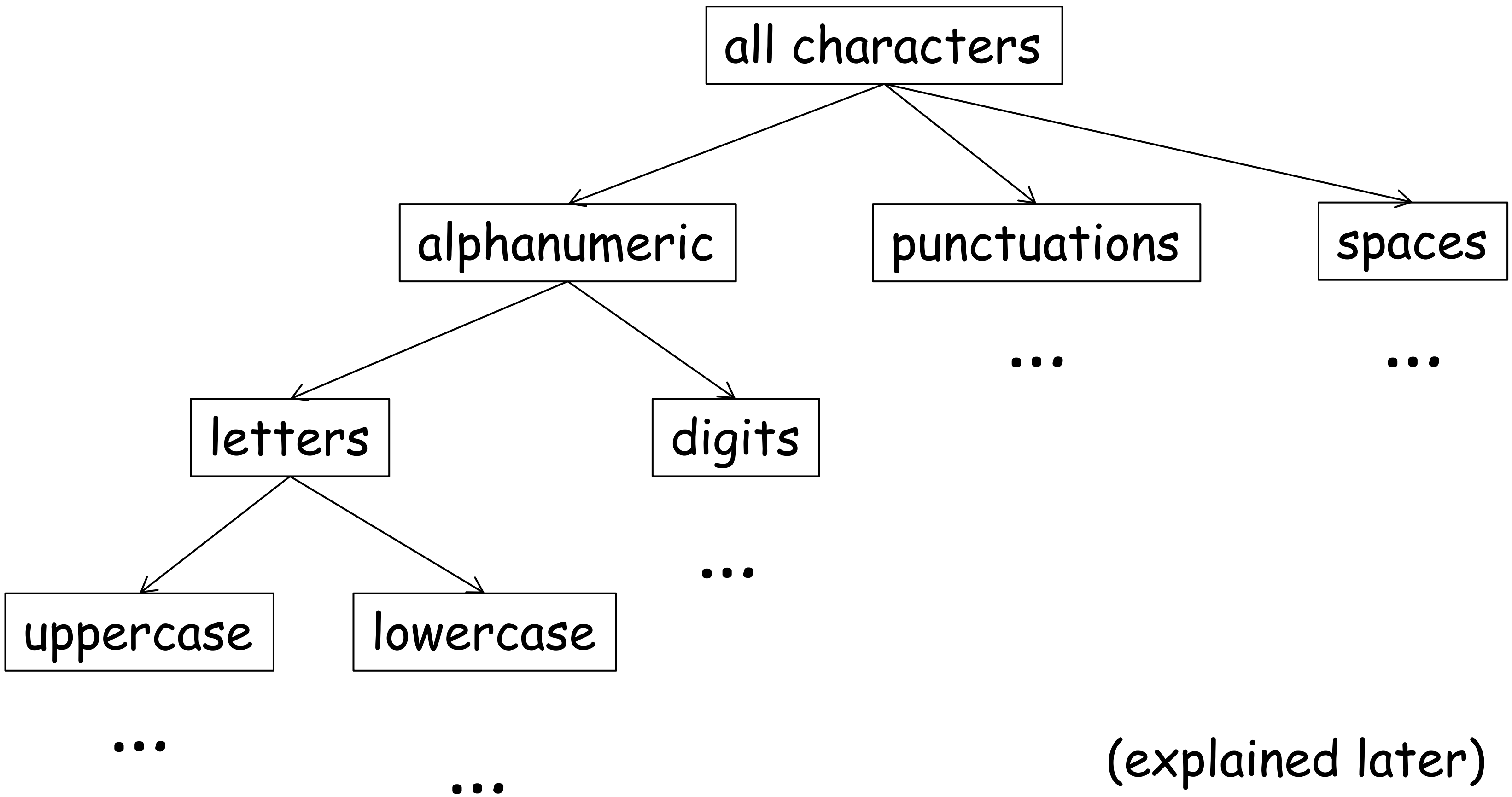
(r1,r2) where r1 and r2 are restricted RE's

we want to locate the position by generalizing character conditions on both sides

. pdf

. (lowercase letters)…

Character Generalization Hierarchy

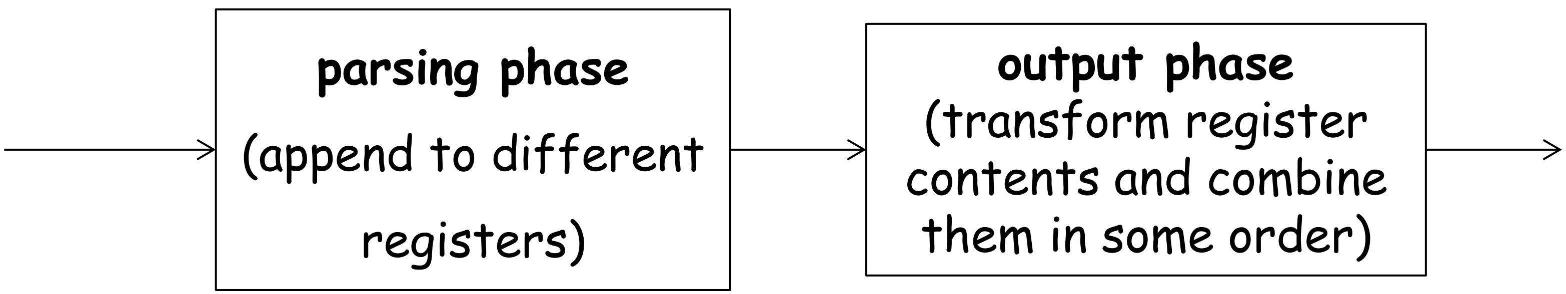


(explained later)

Our Approach

We synthesize transducers which …

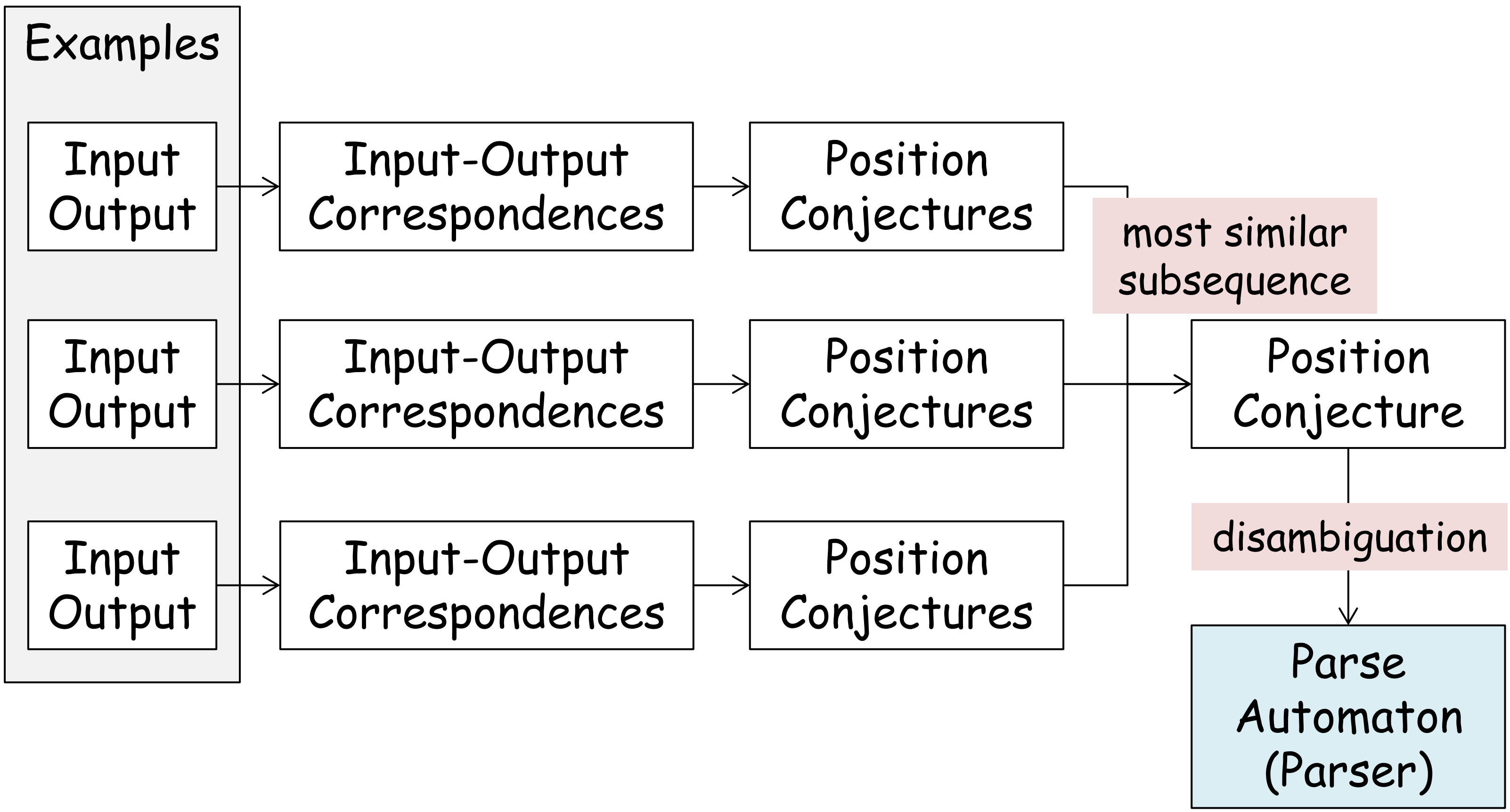
… separates parsing and output generation



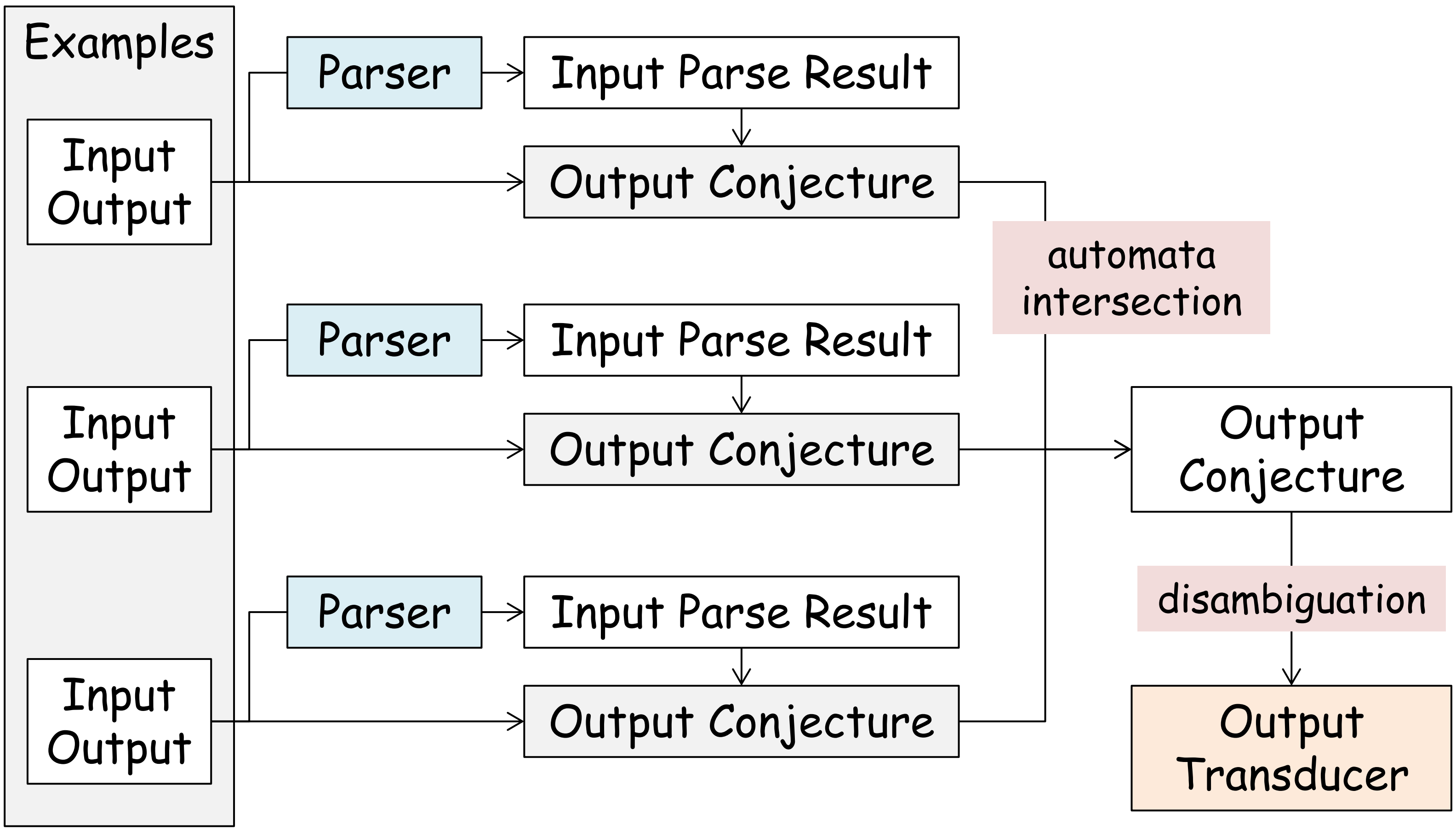
… decides positions by using RE's

… allows <u>arbitrary</u> transformations
(explained later)

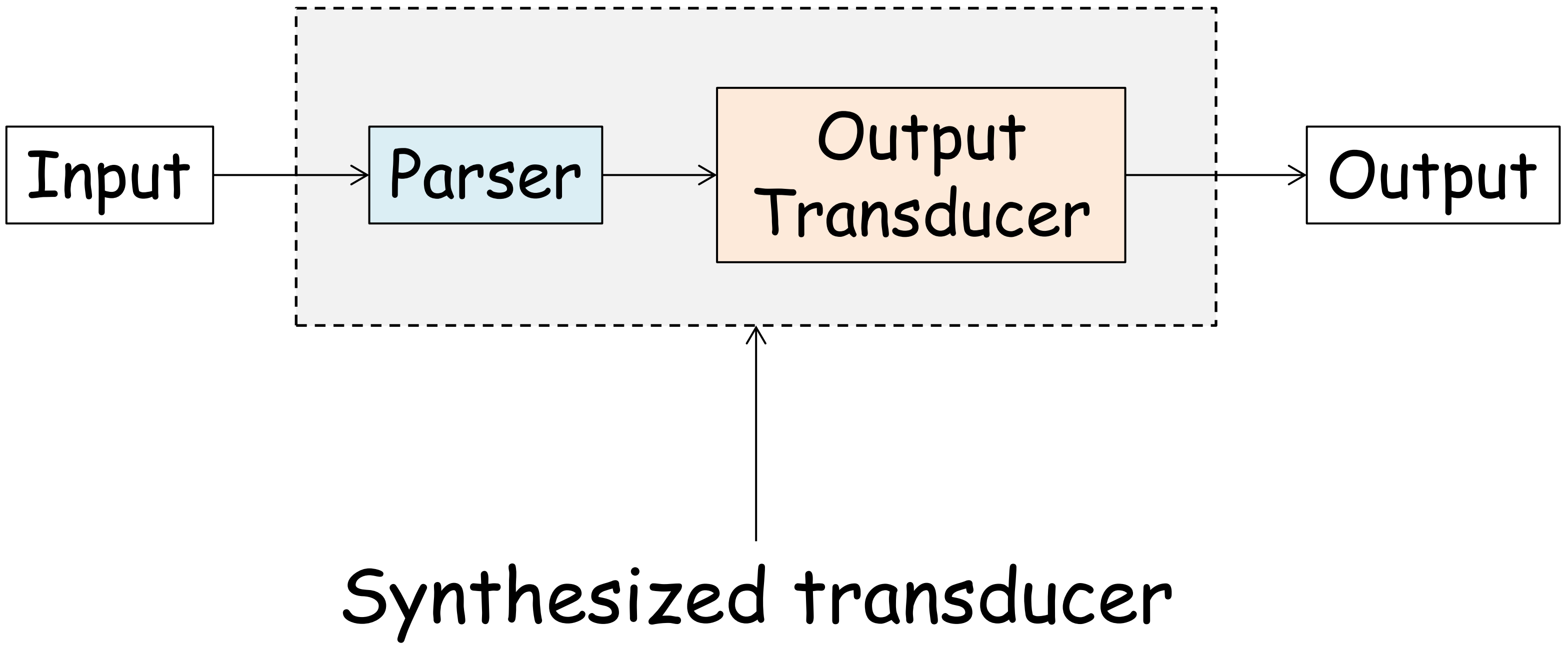
Data Structures (I)



Data Structures (II)



Data Structures (III)



Why Transducers?

Powerful expressivity

- Finite-state transducers extended with finite number of infinite buffers/registers are Turing-complete
- However our synthesizer is far weaker than that

Further optimization

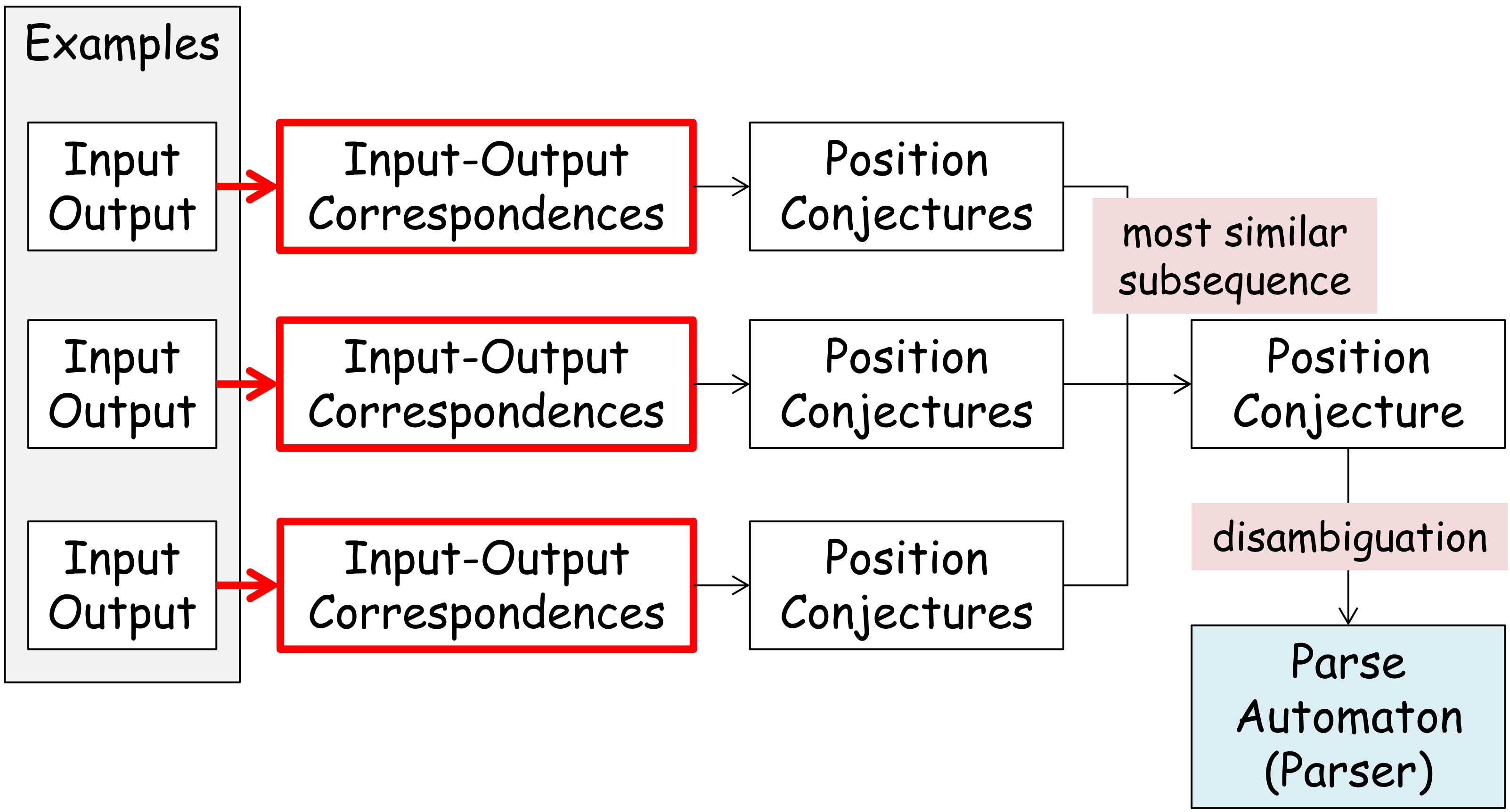
- optional parts in the input
- loops
- …

In the next few slides …

CAUTION

IMPORTANT ALGORITHMS AND TECHNICAL DETAILS (MATHEMATICS INVOLVED)

Data Structures (I)



Finding Input-Output Correspondences

Our synthesizer allows the following transformation:

A substring of the output can be the result of applying a function of arity k to k <u>disjoint</u> substrings of the input.

Example: <u>2</u>+<u>1</u> ➔ <u>3</u> (k=2, f=$\lambda x. \lambda y. x + y$)

Common unary functions:

- monthify: 6 ➔ June
- demonthify: June ➔ 6
- ordinalize: 1 ➔ 1st
- deordinalize: 1st ➔ 1
- weekday: 24 June 2010 ➔ Thursday
- uppercase: hEllo ➔ HELLO
- lowercase: HellO ➔ hello
- capitalize: hello ➔ Hello

weekday(d,m,y)

interesting fact

```
val t = [0,3,2,5,0,3,5,1,4,6,2,4]
val y = if m<3 then y-1 else y
val w = ( y + y div 4
        - y div 100
        + y div 400
        + List.nth(t,m-1)
        + d ) mod 7
```

w=0 for Sunday
w=1 for Monday
…
w=6 for Saturday

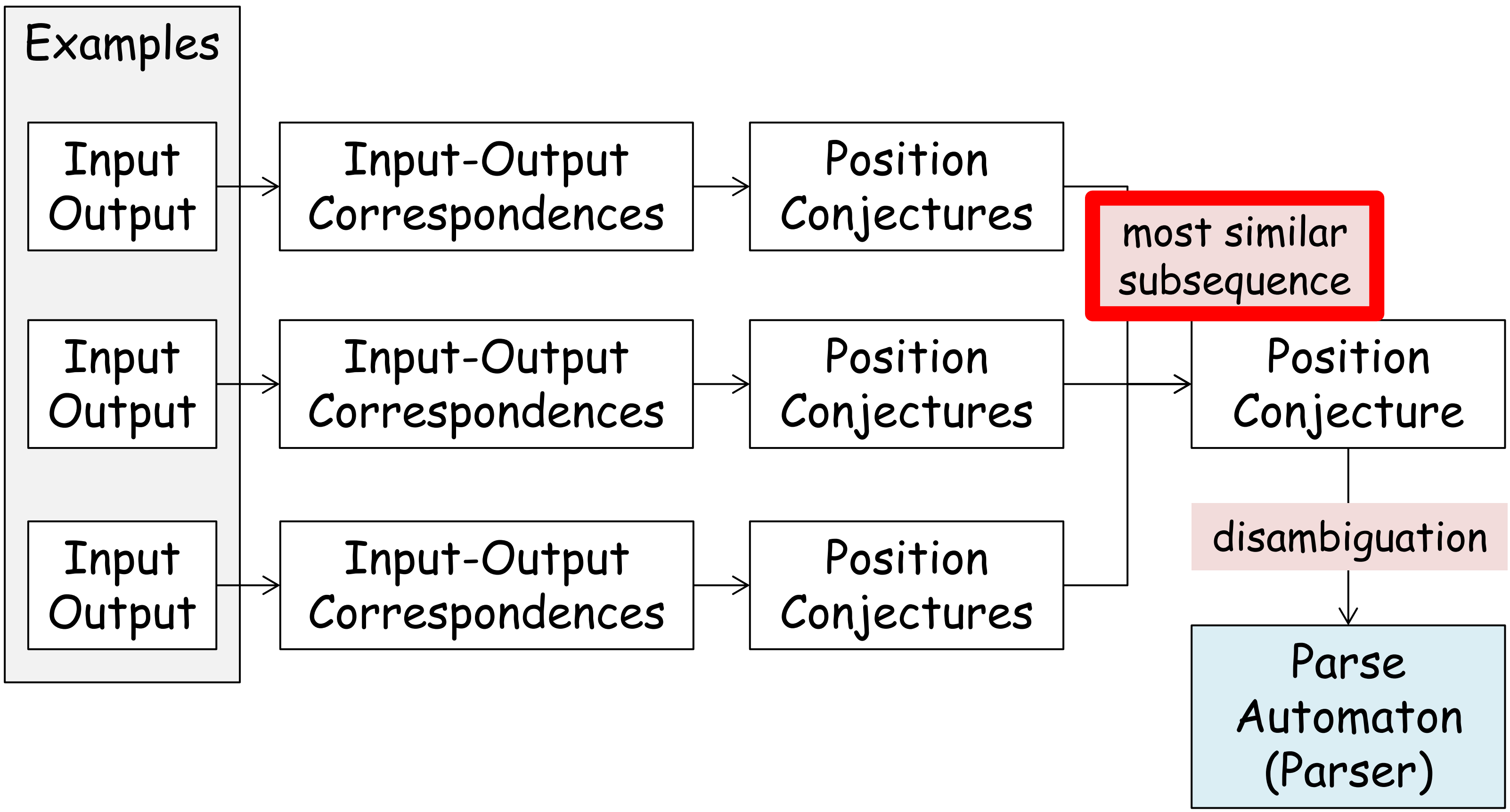
Finding Input-Output Correspondences

Fix k, find k-correspondences:

1. Generate all combinations of k disjoint substrings of the input
2. For a substring of the output, try all k-ary functions on every combination until a result equals that substring

Most trivial transformation: $\lambda x.\, x$

Data Structures (I)



Most Similar Subsequences

What we want is …

… a most similar subsequence (MSS) of sequence x in sequence y, given …

- Σ the input alphabet
- Γ the output alphabet
- $\mathcal{S}: \Sigma \times \Gamma \rightarrow \mathbb{R}$ the similarity function
- $\preccurlyeq$ the subsequence relation
- $|x_1 x_2 \cdots x_n| = n$ the length of a sequence

Most Similar Subsequences

$\mathcal{S}: \Sigma \times \Gamma \to \mathbb{R}$ is the similarity function

We abuse $\mathcal{S}$ for the similarity between sequences defined as

$$\mathcal{S}(a_1 a_2 \cdots a_n, b_1 b_2 \cdots b_n) = \sum_{i=1}^{n} \mathcal{S}(a_i, b_i)$$

Most Similar Subsequences

Definition:

A MSS of $x \in \Sigma^*$ in $y \in \Gamma^*$ is a sequence $z \in \Gamma^{|x|}$ such that …

- $z \preccurlyeq y$
- $\forall w \in \Gamma^{|x|}(w \preccurlyeq y \rightarrow \mathcal{S}(x, w) \leq \mathcal{S}(x, z))$

By definition, there may be multiple MSS'es.

Most Similar Subsequences

$$X = x_1 x_2 \cdots x_m \quad X_i = x_1 x_2 \cdots x_i$$
$$Y = y_1 y_2 \cdots y_n \quad Y_i = y_1 y_2 \cdots y_i$$

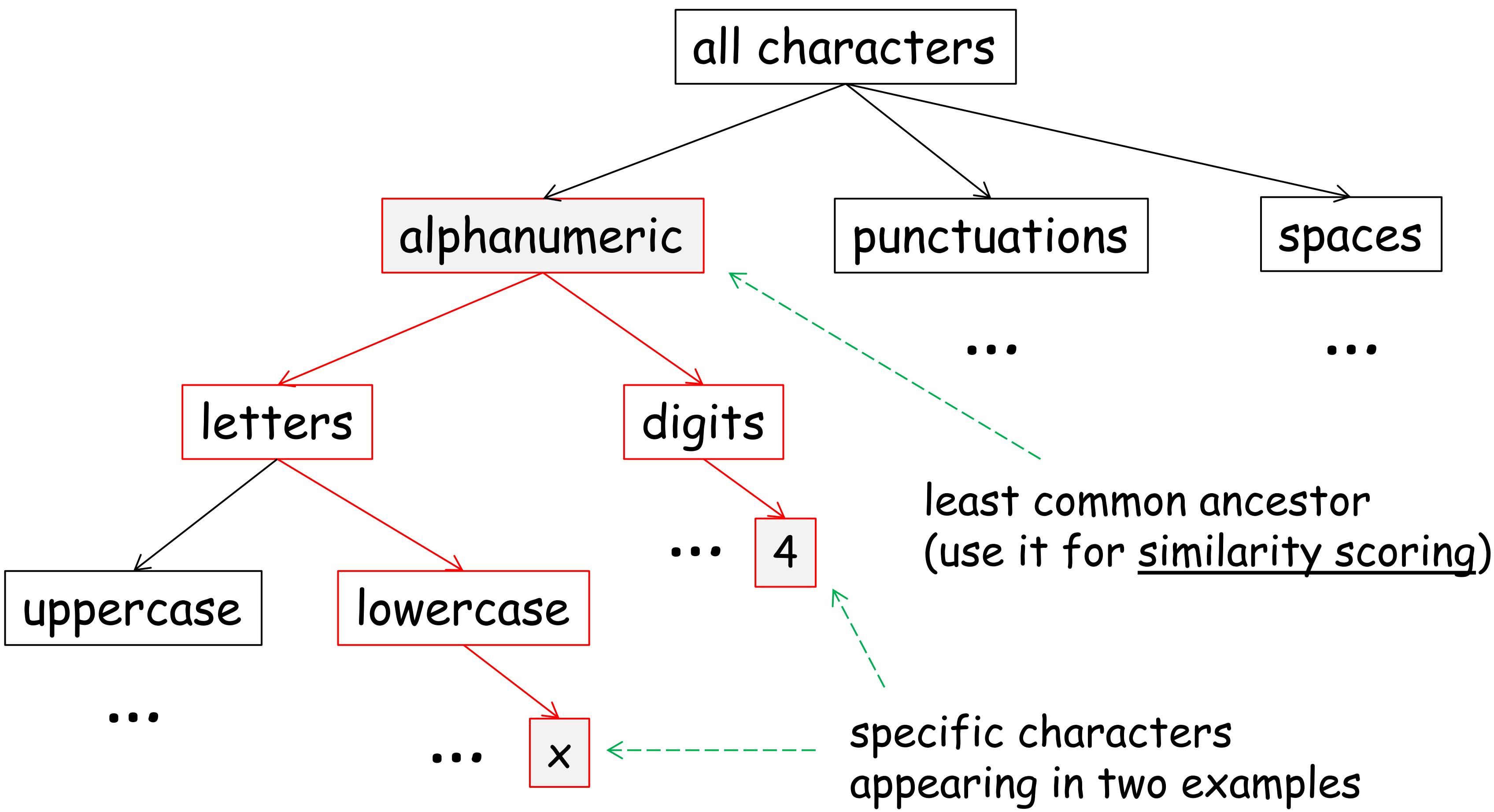
$c[i, j]$ is the length of a MSS of X_i in Y_j

$$c[i,j] = \begin{cases} 0 & \text{if } i = 0 \\ \max(c[i, j-1], c[i-1, j-1] + \mathcal{S}(x_i, y_j)) & \text{if } j \geq i \geq 1 \end{cases}$$

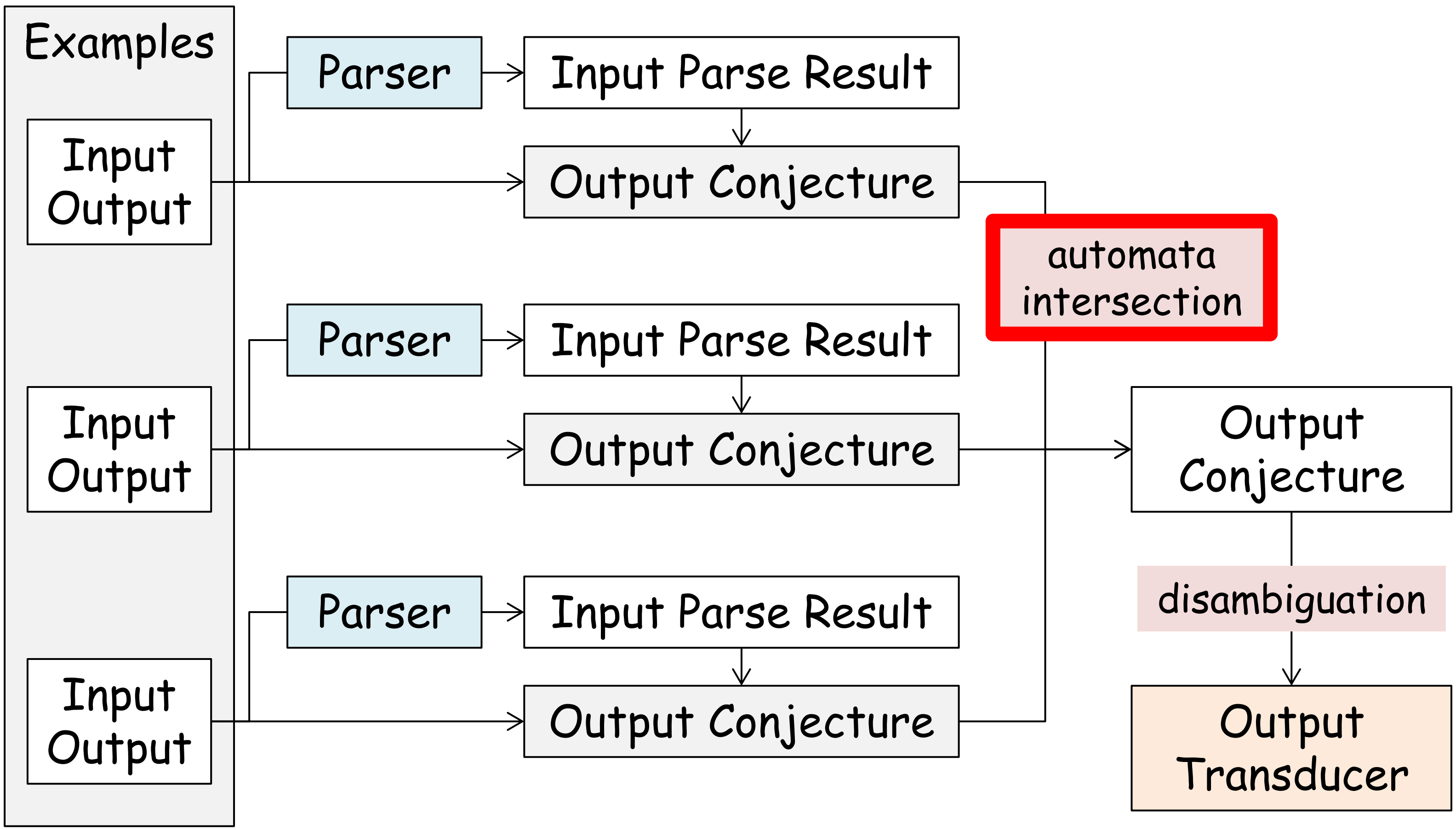
Most Similar Subsequences

	j	0	1	2	3	4	5	6	7
i		y_j	A	B	C	B	D	A	B
0	x_i	0	0						
1	B		$\nwarrow 0$	$\nwarrow 1$					
2	D			$\nwarrow 0$	$\nwarrow 1$				
3	C				$\nwarrow 1$	$\leftarrow 1$			
4	A					$\nwarrow 1$	$\leftarrow 1$		
5	B						$\nwarrow 1$	$\leftarrow 1$	
6	A							$\nwarrow 2$	$\leftarrow 2$

Character Generalization Hierarchy



Data Structures (II)



Automata Intersection

Input automata: A and B

- states of new automaton is the Cartesian product $S_{A\cap B} = S_A \times S_B$
- new transitions:

$$\langle p_1, q_1\rangle \xrightarrow[A\cap B]{\alpha} \langle p_2, q_2\rangle \iff p_1 \xrightarrow[A]{\alpha} p_2 \text{ and } q_1 \xrightarrow[B]{\alpha} q_2$$

Now the automaton is in quadratic size.

Minimize it to get the final result!

Conclusion

- Extended symbolic finite-state transducers are feasible and useful for synthesizing string transformations from input-output examples.

Questions?

* Image from http://resumeconfidence.com/wp-content/uploads/2013/04/3240.thank-you.jpg