From Natural Language Specifications to Program Input Parsers

Tao Lei,

Fan Long, Regina Barzilay, Martin Rinard

CSAIL, MIT

Translating Natural Language to Input Parser

Input Specification:

Defines the format of input data

- The input starts with a line containing two integers n and r.

- This is followed by n lines, each containing two integers xi, yi, giving the coordinates of the polygon vertices.

Two Input Examples:

```
3 6
0 4
0 0
5 1
```

```
4 10
-8 2
8 14
0 14
0 6
```

Input Parser:

Part of a program that reads and stores data

```
int n, r, x[], y[];

Scanner scanner = new
   Scanner(new File("input.txt"));

n = scanner.nextInt();
r = scanner.nextInt();

x = new int[n];
y = new int[n];
for (int i = 0; i < n; i++) {
   x[i] = scanner.nextInt();
   y[i] = scanner.nextInt();
}
```

Translating Natural Language to Input Parser

Input Specification:

Defines the format of input data

- The input starts with a line containing two integers n and r.

- This is followed by n lines, each containing two integers xi, yi, giving the coordinates of the polygon vertices.

Two Input Examples:

3 6	4 10

Input Parser:

Part of a program that reads and stores data

```
int n, r, x[], y[];

Scanner scanner = new
   Scanner(new File(“input.txt”));

n = scanner.nextInt();
r = scanner.nextInt();

x = new int[n];
y = new int[n];
for (int i = 0; i < n; i++) {
   x[i] = scanner.nextInt();
   y[i] = scanner.nextInt();
}
```

Goal: generating input parser by reading natural language

Motivation

- Reading and processing data is a common task
- Writing input parsers is mechanical, tedious and time-consuming

MST dependency data format

John	ate	an	apple
NN	VB	DT	NN
SUBJ	ROOT	MOD	OBJ
2	0	4	2

The	do
DT	NN
MOD	SU
2	3

POS tagger data format

This	DT
is	VBZ
a	DT
short	JJ
	NN

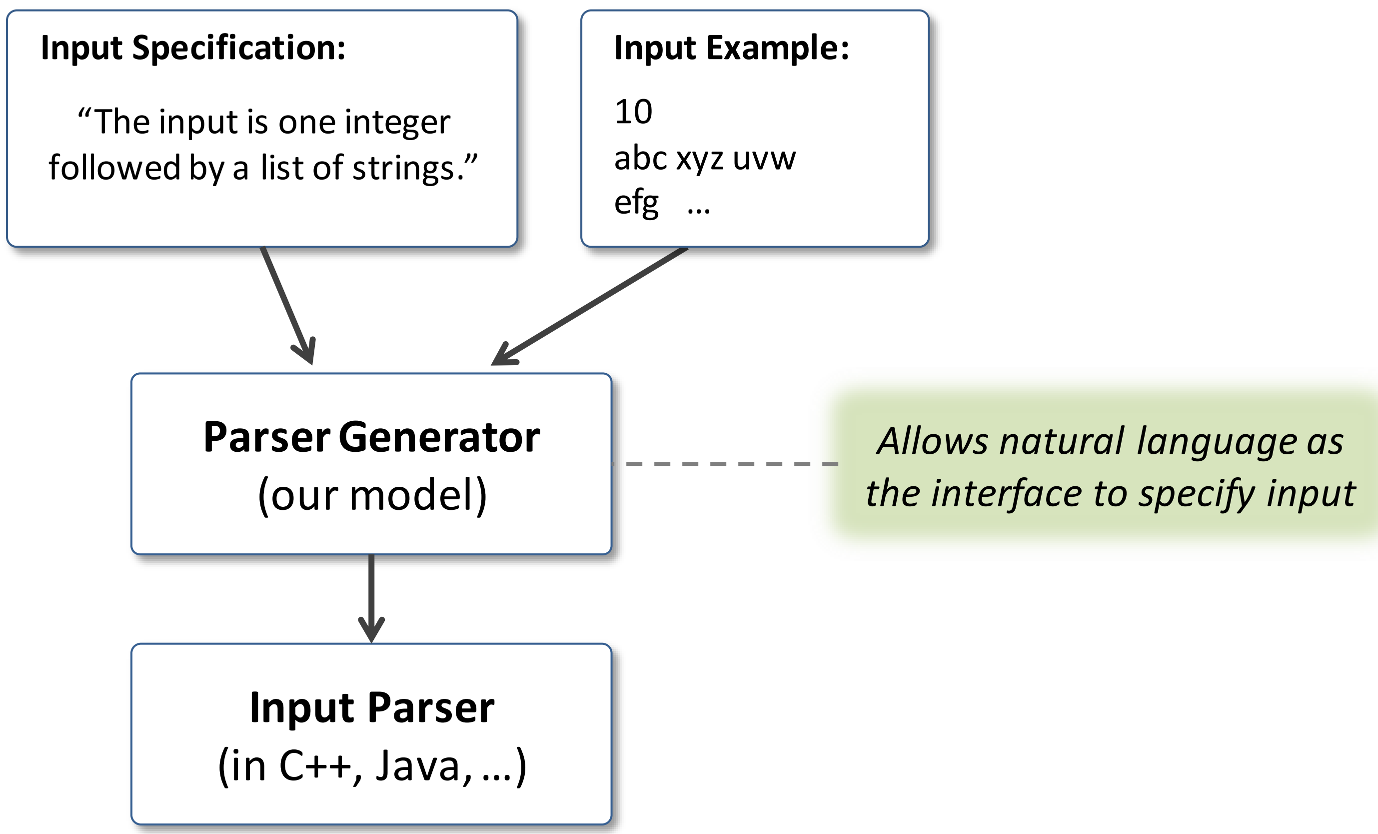
CONLL dependency data format

1	Cathy	Cathy	N	N	…	2	su
2	zag	zie	V	V	…	0	ROOT
3	hen	hen	Pron	Pron	…	2	obj1
4	wild	wild	Adj	Adj	…	5	mod
5	zwaaien	zwaai	N	N	…	2	vc
6	.	.	Punc	Punc	…	5	punct

…

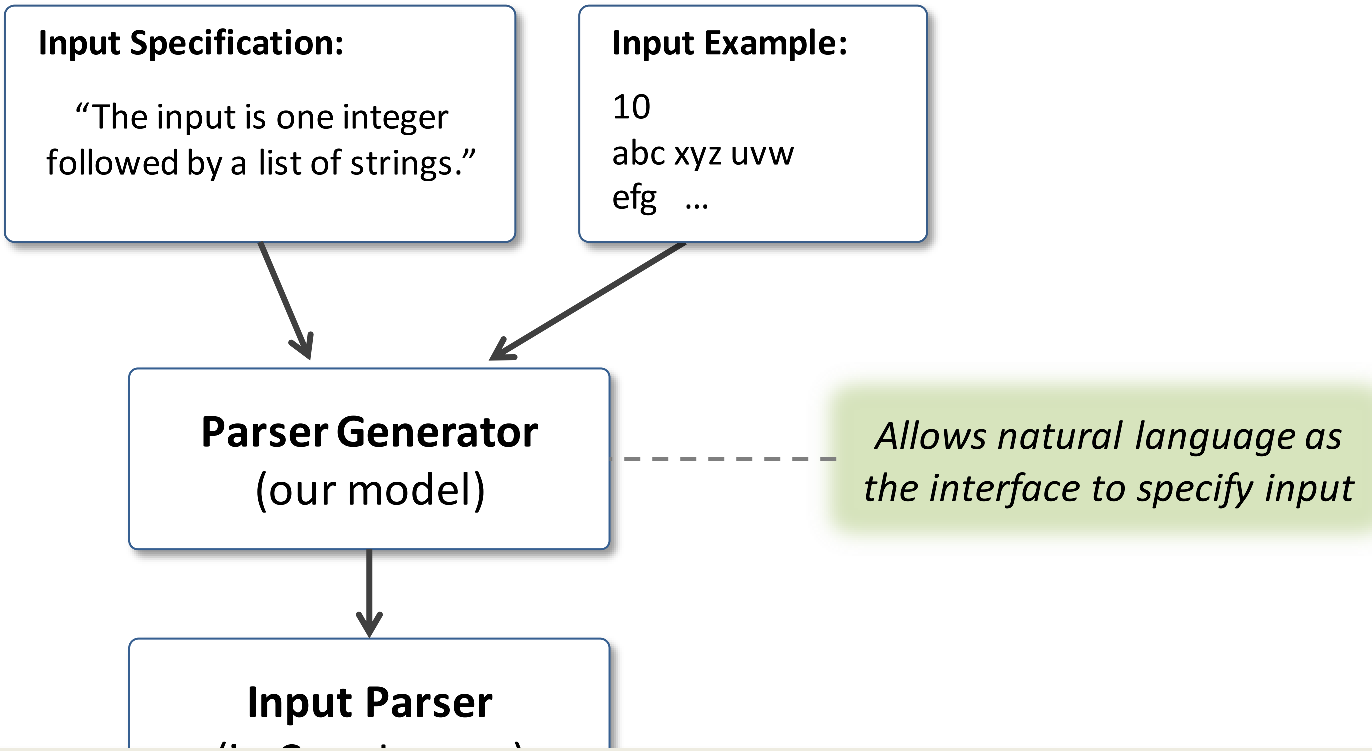
Motivation

- Reading and processing data is a common task
- Writing input parsers is mechanical, tedious and time-consuming



Motivation

- Reading and processing data is a common task
- Writing input parsers is mechanical, tedious and time-consuming



Advantage: reducing programming effort and the chance of making code mistakes

How to Translate NL to Input Parser?

- Need an abstraction that connects NL and input parser

Input Specification:

The input consists of multiple sentences.

- The first line of each sentence is the list of words in the sentence;
- The second line of each sentence contains the POS tokens;
- The third line are dependency labels;
- The last line are integers representing the positions of each word's parent.

?

Input Parser:

```
sentence = [ ];
with open("input.txt") as fin:
    line = fin.readline().strip();
    while line:
        if line != "":
            word = line.split();
            pos = fin.readline().split();
            label = fin.readline().split();
            parent = fin.readline().split();
            parent = [ int(x) for x in parent ];

            sentence.append( (word, pos,
                              label, parent) );
        line = fin.readline().strip();
```

How to Translate NL to Input Parser?

- Need an abstraction that connects NL and input parser

Input Example:

John	ate	an	apple
NN	VB	DT	NN
SUBJ	ROOT	MOD	OBJ
2	0	4	2

The	dog	barks
DT	NN	VB
MOD	SUBJ	ROOT
2	3	0

...

How to Translate NL to Input Parser?

- Need an abstraction that connects NL and input parser

Input Example:

John	ate	an	apple
NN	VB	DT	NN
SUBJ	ROOT	MOD	OBJ
2	0	4	2
The	dog	barks	
DT	NN	VB	
MOD	SUBJ	ROOT	
2	3	0	

...

Input

How to Translate NL to Input Parser?

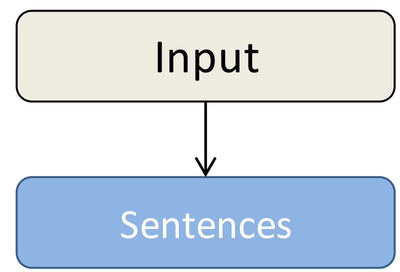
- Need an abstraction that connects NL and input parser

Input Example:

John	ate	an	apple
NN	VB	DT	NN
SUBJ	ROOT	MOD	OBJ
2	0	4	2

The	dog	barks
DT	NN	VB
MOD	SUBJ	ROOT
2	3	0

...



How to Translate NL to Input Parser?

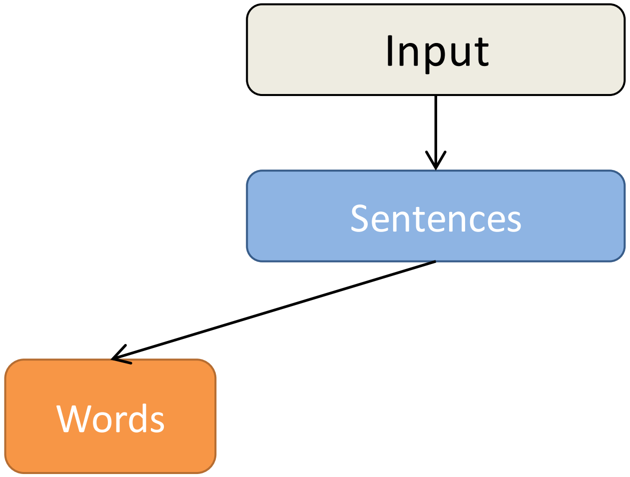
- Need an abstraction that connects NL and input parser

Input Example:

John	ate	an	apple
NN	VB	DT	NN
SUBJ	ROOT	MOD	OBJ
2	0	4	2

The	dog	barks
DT	NN	VB
MOD	SUBJ	ROOT
2	3	0

...



How to Translate NL to Input Parser?

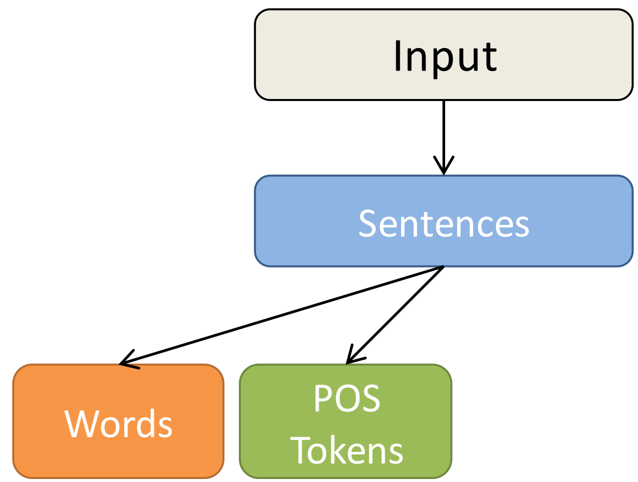
- Need an abstraction that connects NL and input parser

Input Example:

John	ate	an	apple
NN	VB	DT	NN
SUBJ	ROOT	MOD	OBJ
2	0	4	2

The	dog	barks
DT	NN	VB
MOD	SUBJ	ROOT
2	3	0

...



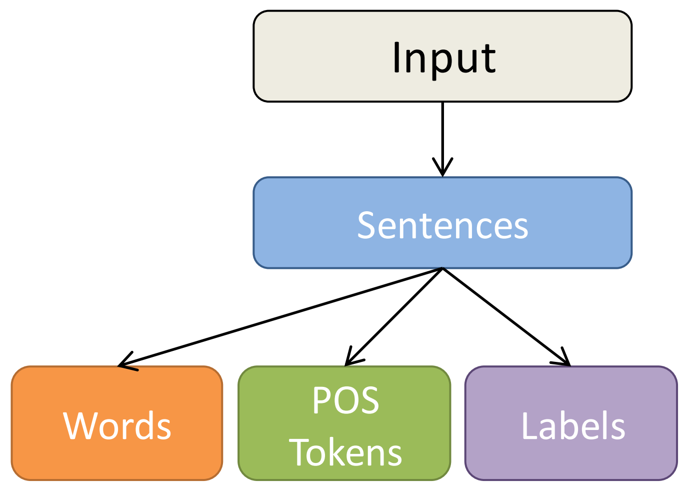
How to Translate NL to Input Parser?

- Need an abstraction that connects NL and input parser

Input Example:

John	ate	an	apple
NN	VB	DT	NN
SUBJ	ROOT	MOD	OBJ
2	0	4	2
The	dog	barks	
DT	NN	VB	
MOD	SUBJ	ROOT	
2	3	0	

...



How to Translate NL to Input Parser?

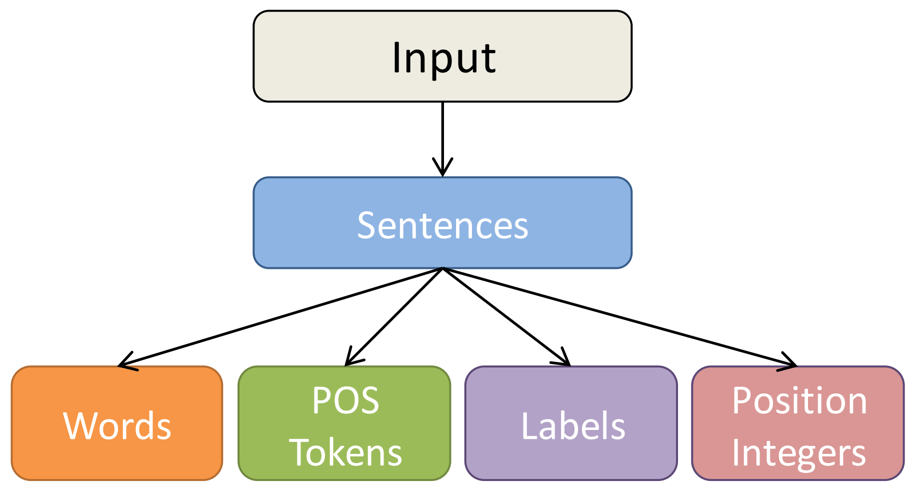
- Need an abstraction that connects NL and input parser

Input Example:

John	ate	an	apple
NN	VB	DT	NN
SUBJ	ROOT	MOD	OBJ
2	0	4	2

The	dog	barks
DT	NN	VB
MOD	SUBJ	ROOT
2	3	0

...



How to Translate NL to Input Parser?

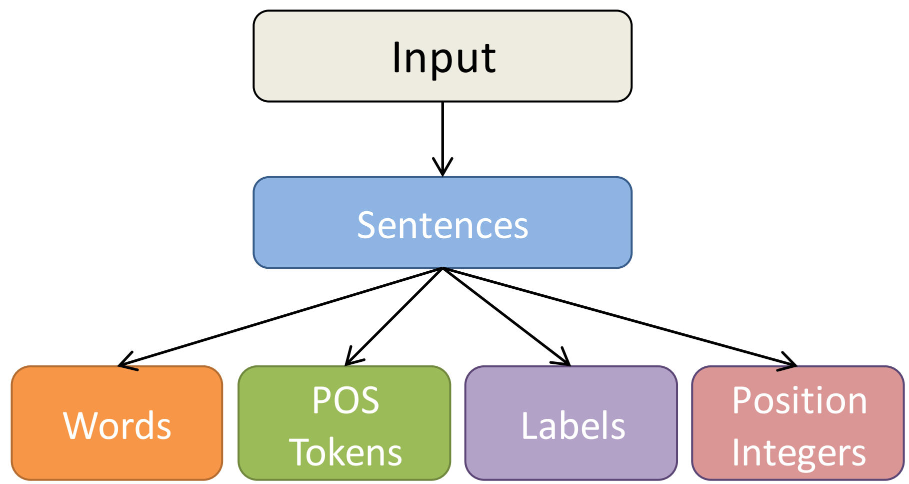
- Need an abstraction that connects NL and input parser
- **Specification tree** of nested input formats

Input Example:

John	ate	an	apple
NN	VB	DT	NN
SUBJ	ROOT	MOD	OBJ
2	0	4	2

The	dog	barks
DT	NN	VB
MOD	SUBJ	ROOT
2	3	0

...



Specification Tree

How to Translate NL to Input Parser?

- Need an abstraction that connects NL and input parser
- **Specification tree** of nested input formats

Input Specification

↓

Specification Tree

↓

Input Parser

- The input parser is **deterministically generated** from the **specification tree**.

How to Translate NL to Input Parser?

- Need an abstraction that connects NL and input parser
- **Specification tree** of nested input formats

Input Specification

↓

Specification Tree

↓

Input Parser

- The input parser is **deterministically generated** from the specification tree.

Focus: translating input specifications into specification trees

How to Translate NL to Specification Tree?

Input Specification

↓

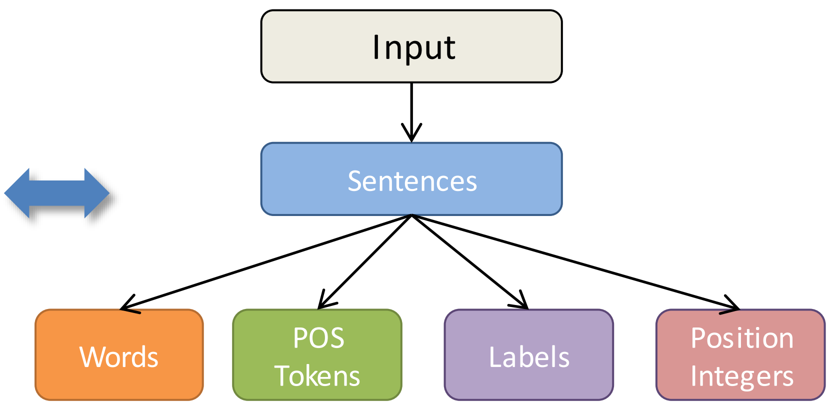
Specification Tree

- **Specification tree** is a ***dependency tree*** *over noun phrases* in the NL specification.

Input Specification:

The input consists of multiple sentences.

- The first line of each parse is the list of words in the sentence;
- The second line of each parse contains the POS tokens;
- The third line are dependency labels;
- The last line are integers representing the positions of each word's parent.



Task: translation as an NLP problem

Learning Scenario

Training Data

N input specifications
$\boldsymbol{w} = \{w^1, \dots, w^N\}$

Input

The input consists of a single test case. A test case consists of two lines. The first line contains an integer n indicating the number of molecule types. The second line contains n eight-character strings, each describing a single type of molecule, separated by single spaces. Each string consists of four two-character connector labels

some input examples
for each specification

Input Example:

3
A+00A+A+ 00B+D+A- B-C+00C+

No human annotation

Output

specification trees
$\boldsymbol{t} = \{t^1, \dots, t^N\}$

corresponding input parsers

$$\boldsymbol{t} \sim P(\boldsymbol{t} | \boldsymbol{w})$$

Learning Scenario

Training Data

N input specifications
$\boldsymbol{w} = \{w^1, \dots, w^N\}$

Input

The input consists of a single test case. A test case consists of two lines. The first line contains an integer n indicating the number of molecule types. The second line contains n eight-character strings, each describing a single type of molecule, separated by single spaces. Each string consists of four two-character connector labels

some input examples
for each specification

Input Example:

3
A+00A+A+ 00B+D+A- B-C+00C+

No human annotation

Idea: learning from **feedback** -- testing input parser on input examples

Key Intuitions

a **correct** tree should read all input examples successfully

- Necessary but NOT sufficient condition
- **False-positive** parsers

5
-8
8
0
0
-8

a list of integers?

a list of integer pairs?

⋮

a list of strings?

Input Example

Possible Interpretations

Many input parsers can read the same input

Key Intuitions

a **correct** tree should read all input examples successfully

the **correct** trees should share common features

X contains Y

X starts with Y

The input contains an integer

Test case contains several strings

Each line starts with two numbers

the input → an integer

test case → several strings

Patterns

Example Sentences

Tree Structures

Bayesian Generative Model

$$P(\theta) \cdot \prod_i P(t^i) \cdot P(w^i | t^i; \theta)$$

(i) Generating Parameters

$$\theta_{\cdot} \sim Dirichlet(\boldsymbol{\alpha})$$

(ii) Generating Specification Trees

$$P(t^i) \propto \begin{cases} 1 & \text{parser of } t^i \text{ read input examples successfully} \\ \epsilon & \text{otherwise} \end{cases}$$

(iii) Generating Feature Observations

$$P(w^i | t^i; \theta) = \prod_{f \in \phi(w^i, t^i)} \theta_f$$

$\phi(w^i, t^i)$: set of features over (w^i, t^i)

a **correct** tree should read all input examples successfully

the **correct** trees should share common features

Idea: encode both intuitions in our model

Inference: Gibbs Sampling

$$\boldsymbol{t} \sim P(\boldsymbol{t}|\boldsymbol{w}) = \int_{\theta} P(\boldsymbol{t}, \theta|\boldsymbol{w})$$

$$t^1 \longrightarrow t^2 \longrightarrow \cdots \longrightarrow \boldsymbol{t^i} \longrightarrow \cdots \longrightarrow t^N$$

update specification tree t^i
for the i-th input specification

Sample from conditional probability:

$$t^i \sim P\left(t^i|\boldsymbol{w}, \boldsymbol{t}^{-i}\right)$$

Intractable

Inference: Gibbs Sampling

$$\boldsymbol{t} \sim P(\boldsymbol{t}|\boldsymbol{w}) = \int_{\theta} P(\boldsymbol{t},\theta|\boldsymbol{w})$$

$$t^1 \longrightarrow t^2 \longrightarrow \cdots \longrightarrow \boldsymbol{t^i} \longrightarrow \cdots \longrightarrow t^N$$

update specification tree t^i
for the i-th input specification

(i) Estimate current parameters

$$\theta^* = E\left[\theta|\boldsymbol{w},\boldsymbol{t}^{-i}\right]$$

(ii) Sample a new tree

$$t' \sim Q(t') \propto P\left(w^i|t';\theta^*\right)$$

(iii) Apply *Metropolis-Hastings* rule

$t^i := t'$ with probability:

$$\min\left\{1,\frac{P(t^i)Q(t')}{P(t')Q(t^i)}\right\}$$

Experiments

Input

The input consists of a single test case. A test case consists of two lines. The first line contains an integer n indicating the number of molecule types. The second line contains n eight-character strings, each describing a single type of molecule, separated by single spaces. Each string consists of four two-character connector labels

Domain:

Programming contest (ACM-ICPC)

Training Data:

106 input specifications

100 input examples for each

Text Statistics:

Sentences:	*424*
Vocabulary:	*781*
# of Sent. in Document	*1 ~ 8*
Avg. Sent. Length	*17.3*

relative clauses in sentences

Evaluation Metrics

Recall:

$$\frac{\#\ \text{correct specification trees}}{\#\ \text{input specifications}}$$

Precision:

$$\frac{\#\ \text{correct specification trees}}{\#\ \text{positive specification trees}}$$

F-Score:

$$\frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Baseline Models

No Learning

Does not learn feature parameters; randomly samples the specification tree until successfully reads all input examples

Aggressive (Clarke et al. 2010)

Trains a discriminative structure learner (SVM^{Struct}) using all "positive" specification trees obtained in previous iteration; uses the learner to find the most plausible trees in the next iteration

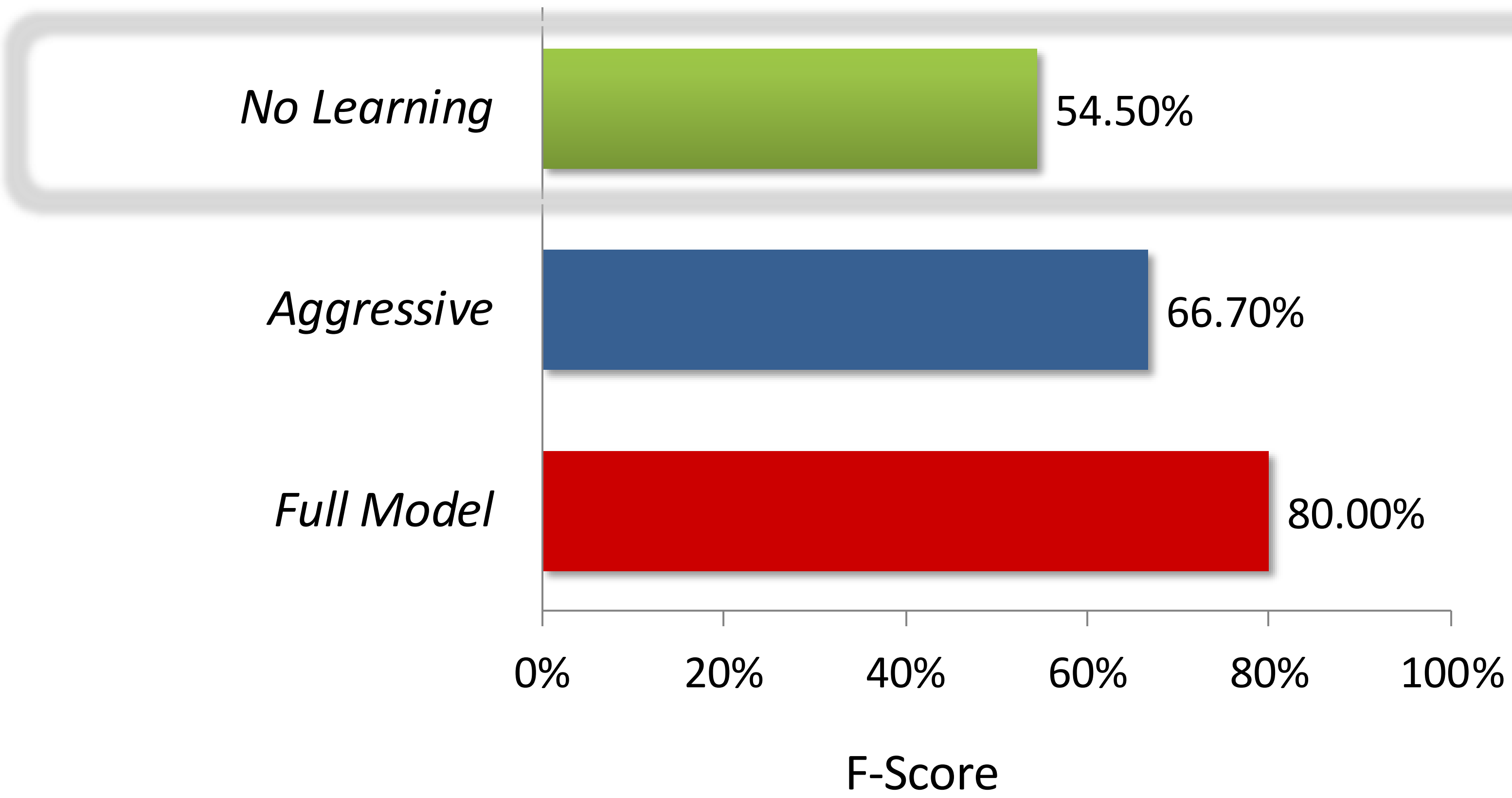
Full Model - Oracle

An "oracle" feedback tells our full model whether the specification tree is correct or not

Aggressive - Oracle

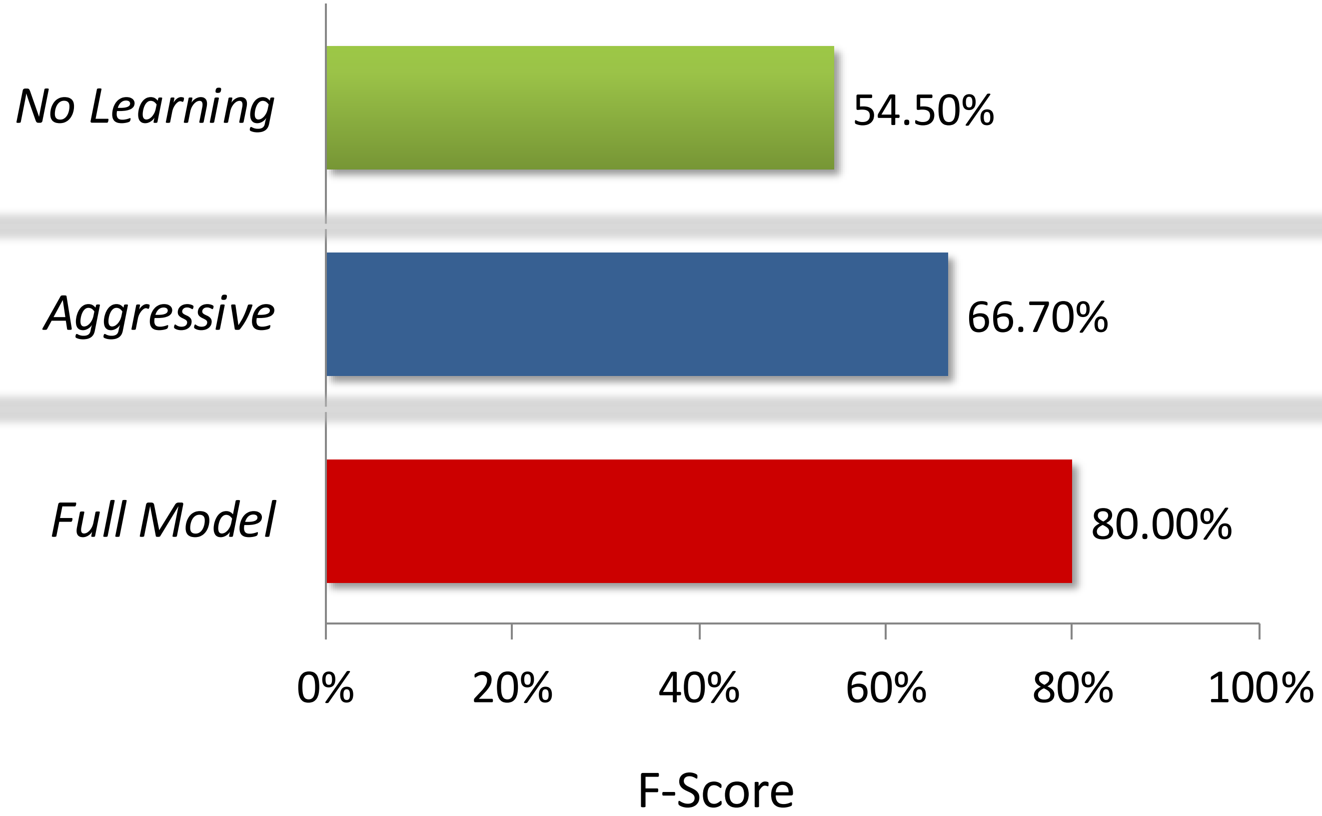
Trains SVM using perfect oracle supervision signal

Overall Performance



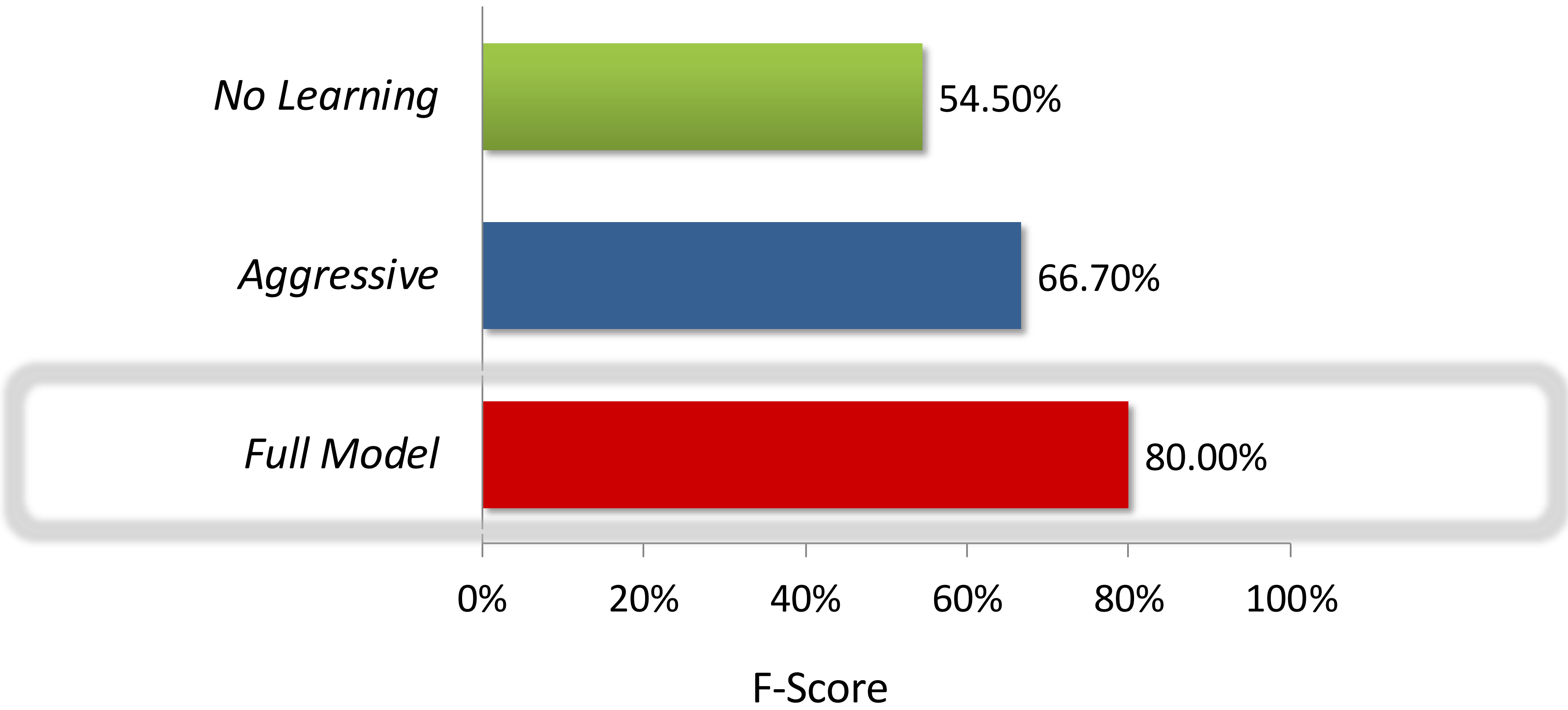
- Search space is exponential, and is large on difficult specifications
- Cannot distinguish between correct parsers and false-positive parsers

Overall Performance



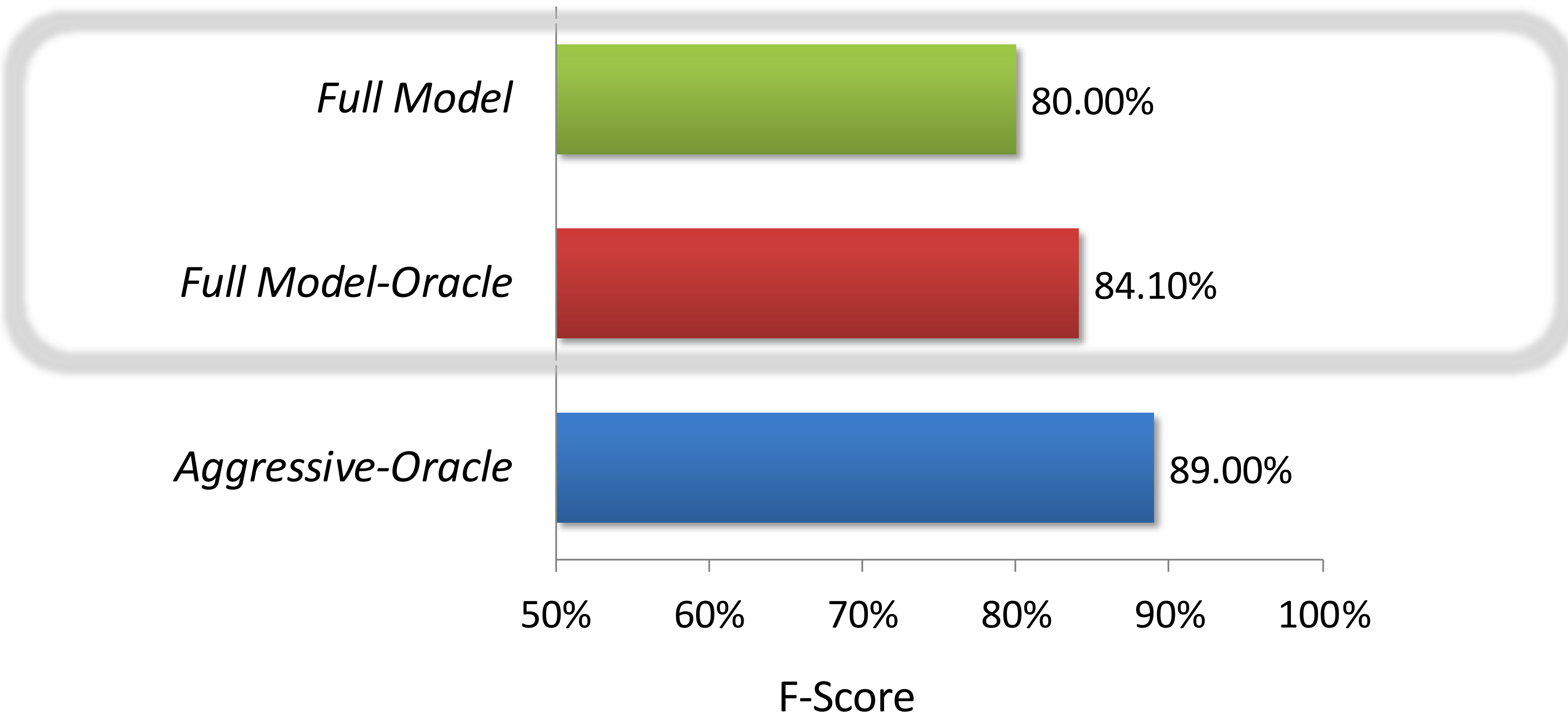
- Using false-positive parsers to train SVM will hurt the performance

Overall Performance

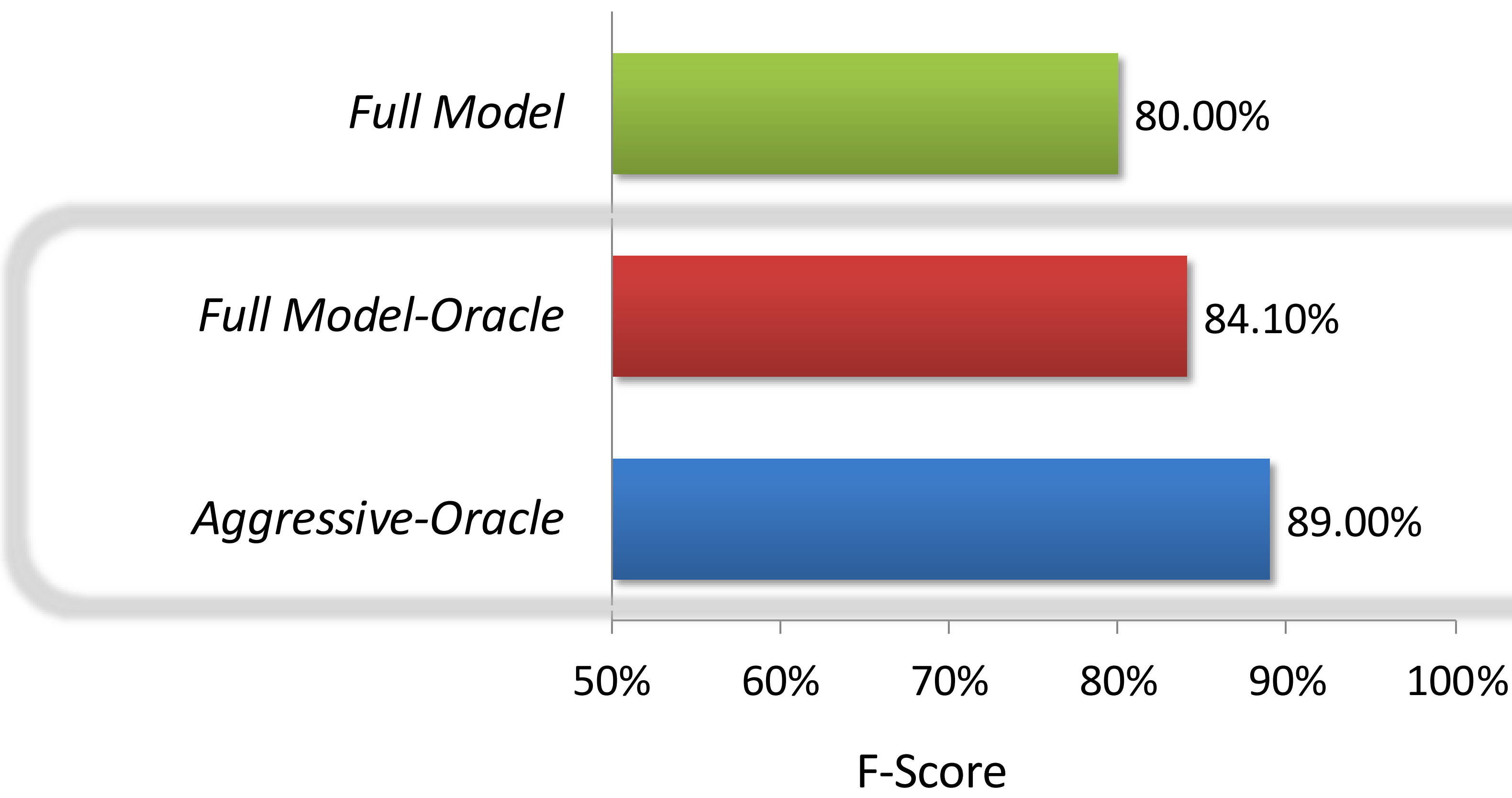


- Learns from feedback and feature observations in a joint, complementary fashion

Comparison with Oracles

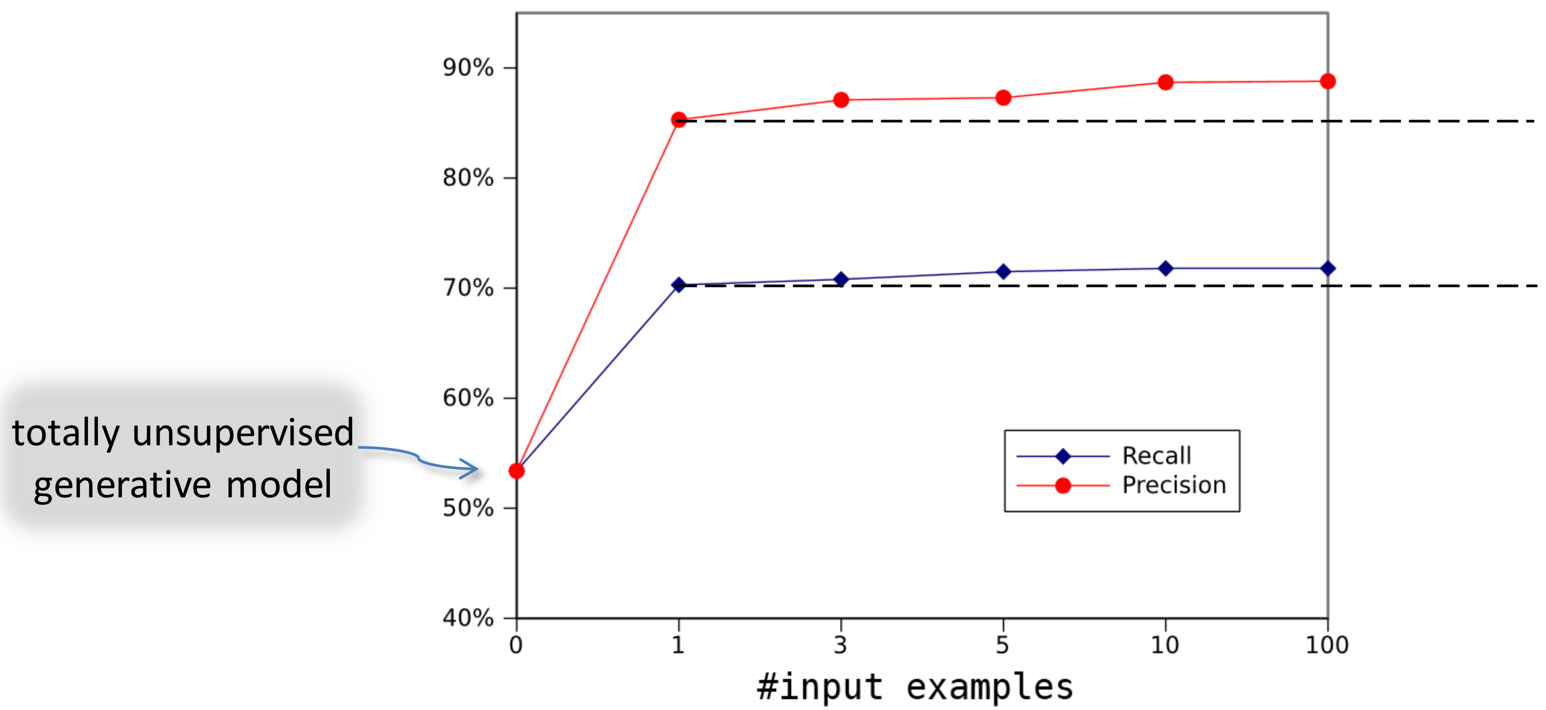


Comparison with Oracles



- Discriminative model is better at learning from strong supervision
- Generative model is itself much more constrained

Learning Curve as a Function of # Input Examples



- May not be possible to obtain so many input examples
- Retains high performance when just **one** example is available

Conclusion

- A new problem in addition to generating database queries or regular expressions from natural language

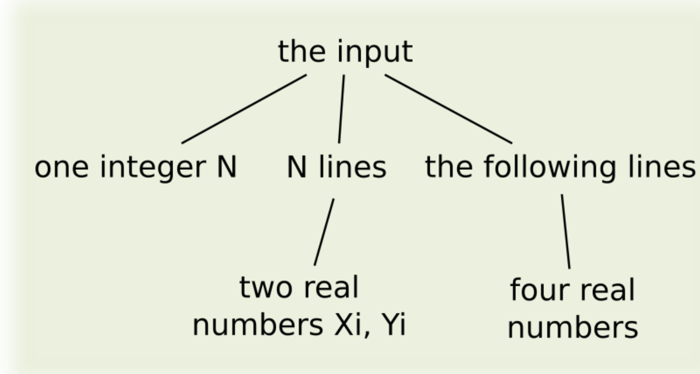
- Our method can learn to ground natural language descriptions of input data formats

Code and data available at:

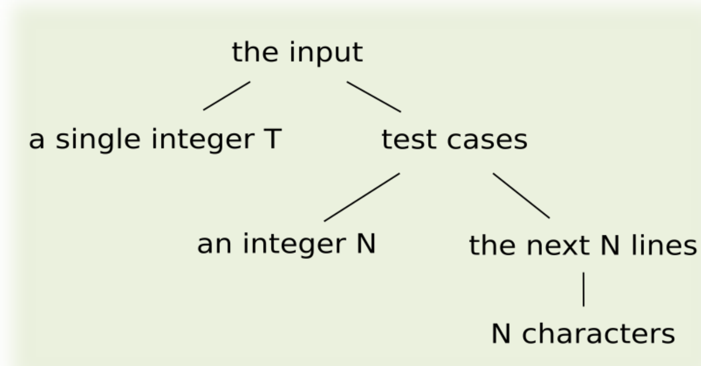
http://groups.csail.mit.edu/rbg/code/nl2p

Model	**Recall**	**Precision**	**F-Score**
No Learning	52.0	57.2	54.5
Aggressive	63.2	70.5	66.7
Full Model	**72.5**	**89.3**	**80.0**
Full Model (Oracle)	72.5	100.0	84.1
Aggressive (Oracle)	80.2	100.0	89.0

Your program is supposed to read the input from the standard input and write its output to the standard output.
The first line of the input contains one integer N. N lines follow, the i-th of them contains two real numbers Xi, Yi separated by a single space - the coordinates of the i-th house. Each of the following lines contains four real numbers separated by a single space. These numbers are the coordinates of two different points (X1, Y1) and (X2, Y2), lying on the highway.



The input contains a single integer T that indicates the number of test cases.
Then follow the T cases. Each test case begins with a line contains an integer N, representing the size of wall. The next N lines represent the original wall. Each line contains N characters. The j-th character of the i-th line figures out the color ...



(a) The input contains several testcases.
Each is specified by two strings S, T of alphanumeric ASCII characters

(b) The next N lines of the input file contain the Cartesian coordinates of watchtowers, one pair of coordinates per line.