or even...

clear evidence that MT is used in real life.
Context in Translation

xiaoxin gou
小心 狗 <=》 be aware of dog

fluency problem
(n-gram)

syntax should help...

小学生 => be careful not to VP
小心 NP <=》 be careful of NP
xiaoxin
小心 X <=》 be careful not to X

(SCFG)
How do people translate?

1. understand the source language sentence

2. generate the target language translation

布什 与 沙龙 举行 了 会谈

*Bùshí*  *yu*  *Shalóng*  *juxíng*  *le*  *huìtán*

Bush  *and/with*  Sharon  *hold*  *[past.]*  meeting

“Bush held a meeting with Sharon”
How do compilers translate?

1. parse high-level language program into a syntax tree
2. generate intermediate or machine code accordingly

```
x3 = y + 3;
```

```
LD     R1,  id2
ADDF   R1,  R1, #3.0  // add float
RTOI   R2,  R1        // real to int
ST     id1, R2
```

syntax-directed translation (~1960)
Syntax-Directed Machine Translation

1. parse the source-language sentence into a tree
2. recursively convert it into a target-language sentence

Bush hold and/with Sharon [past.] meeting

(Irons 1961; Lewis, Stearns 1968; Aho, Ullman 1972) ==> (Huang, Knight, Joshi 2006)
Tree-based Translation

- get 1-best parse tree; then convert to English

```
IP
   NP  VPB
      NP  VPB
         CC  NPB
             Bùshí  yǔ  Shālóng

Bush and/with Sharon hold [past.] meeting
```

“Bush held a meeting with Sharon”
Tree-based Translation

• recursive rewrite by pattern-matching

(Huang, Knight, Joshi 2006)
Tree-based Translation

- recursively solve unfinished subproblems

(Huang, Knight, Joshi 2006)
Tree-based Translation

- continue pattern-matching

Bush held with Sharon

NPB huìtán

a meeting

NPB Shālóng

(Huang, Knight, Joshi 2006)
Tree-based Translation

- continue pattern-matching

Bush held a meeting with Sharon

(Galley et al. 2004; Huang, Knight, Joshi 2006)
“Bush held a meeting with Sharon”

“Bush”

“Sharon”

“held a meeting”
Memoization (dynamic programming)

cache the best translation (and its probability) at each node (i.e. subtree)
Algorithm 1 Top-down Memoized Recursion

1: function TRANSLATE($\eta$)
2:     if cache[$\eta$] defined then ♦ this sub-tree visited before?
3:         return cache[$\eta$]
4:     best ← 0
5:     for $r \in \mathcal{R}$ do ♦ try each rule $r$
6:         matched, sublist ← PATTERNMATCH($t(r), \eta$) ♦ tree pattern matching
7:             if matched then ♦ if matched, sublist contains a list of matched subtrees
8:                 prob ← Pr($r$) ♦ the probability of rule $r$
9:             for $\eta_i \in \text{sublist}$ do ♦ recursively solve each sub-problem
10:                $p_i, s_i \leftarrow \text{TRANSLATE}(\eta_i)$
11:                prob ← prob $\cdot$ $p_i$
12:            if prob $>$ best then ♦ plug in the results
13:                best ← prob
14:                str ← [$x_i \mapsto s_i$]$s(r)$ ♦ caching the best solution for future use
15:            cache[$\eta$] ← best, str ♦ returns the best string with its prob.
16: return cache[$\eta$]
English-to-Spanish/French Example

(2) my friend’s black cat

le chat noir de mon ami
el gato negro de mi amigo
[the] [cat] [black] [of] [my] [friend]

Again, assume the parse tree for the English phrase is:

NP
  /      
DP       NP
  /       /
NP       POSJJNN
 /       |
PRPNN'sblackcat
 |
my friend

Then we use a rule to translate the English phrase “my friend”:

(r_1') NP( PRP (my) NN (friend) ) → mon ami mi amigo

and a rule for reordering the adjective (black) with the noun (cat)

(r_2') NP( x0:JJ x1:NN ) → x1 x0

finally, we finish the translation with the following two lexical rules:

(r_3') black → noir negro

(r_4') cat → le chat el gato
Sample Input/Output (online)

input1.txt:

NP(DP(NP(PRP("my") NN("friend")) POS("'s") NP(JJ("black") NN("cat"))))
NP(DP(NP(PRP("my") NN("friend")) POS("'s") NP(JJ("white") NN("cat"))))

rules1.txt:

NP(DP(x0:NP POS("'s") x1:NP) -> x1 "de" x0 ### prob=1.0
NP(PRP("my") NN("friend")) -> "mon" "ami" ### prob=0.51
NP(PRP("my") NN("friend")) -> "mon" "amie" ### prob=0.49
NP(x0:JJ x1:NN) -> x1 x0 ### prob=0.7
NP(x0:JJ x1:NN) -> x0 x1 ### prob=0.3
JJ("black") -> "noir" ### prob=0.6
JJ("black") -> "noire" ### prob=0.4
NN("cat") -> "le" "chat" ### prob=1.0
NP(x0:JJ NN("cat")) -> "le" "chat" x0 ### prob=1
NP(JJ("black") x0:NN) -> x0 "noir" ### prob=0.55
NP(JJ("black") x0:NN) -> x0 "noire" ### prob=0.45

To run the program, type in the terminal

```
cat input1.txt | ./translate.py rules1.txt
```

and you will get the output

```
my friend 's black cat -> le chat noir de mon ami ### prob=0.306
my friend 's white cat -> *** failed ***
```
Derivation and k-best translation

```
cat input1.txt | ./translate.py rules1.txt -d
```

should output the following:

```
my friend 's black cat -> le chat noir de mon ami ### prob=0.306
 NP (DP (x0:NP POS ('s)) x1:NP) -> x1 de x0 ### prob=1.000
  | x0: NP (PRP (my) NN (friend)) -> mon ami ### prob=0.510
  | x1: NP (x0:JJ NN (cat)) -> le chat x0 ### prob=1.000
  | | x0: JJ (black) -> noir ### prob=0.600
  | black cat -> le chat noir ### prob=0.600

my friend 's black cat -> le chat noir de mon ami ### prob=0.306
my friend 's white cat -> *** failed ***
```

```
cat input1.txt | ./translate.py rules1.txt -k 3
```

will output

```
my friend 's black cat -> le chat noir de mon ami ### prob=0.306
my friend 's black cat -> le chat noir de mon amie ### prob=0.294
my friend 's black cat -> le chat noire de mon ami ### prob=0.204
my friend 's white cat -> *** failed ***
```

Note: do not print duplicate translations (say, there might be another way of deriving the best translate