EBMT Based on Finite Automata State Transfer Generation

Feiliang Ren
renfeiliang@gmail.com

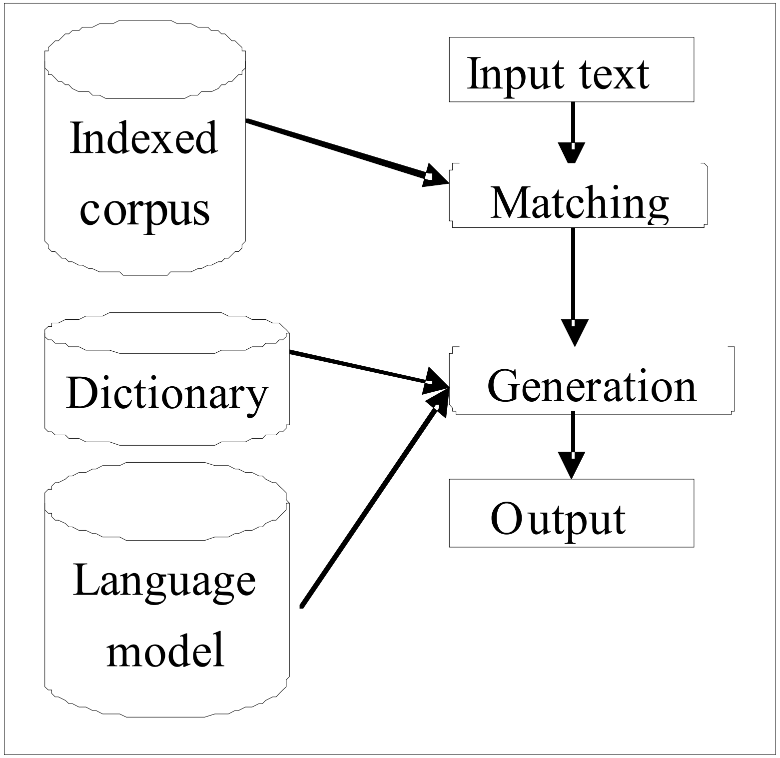
Contents

Introduction

- EBMT ： a method of translation by the principle of analogy
- Three basic modules
  - Matching module
  - Alignment module
  - Recombination module
- The last two modules can be regarded as a translation generation module.
  - Semantic-based generation approach
    - Obtains an appropriate translation fragment for each part of the input sentence.
    - Final translation is generated by recombining the translation fragments in some order.
    - Shortcoming: doesn’t take into account the fluency between the translation fragments
  - Statistical approach
    - Selects translation fragments with a statistical model
    - Can improve the fluency between the translation fragments by using n-gram co-occurrence statistics.
    - Shortcoming: doesn’t take into account the semantic relation between the example and the input sentence
  - Method based on tree string correspondence (TSC) and statistical generation
    - Can solve the shortcomings of the above generation approaches ；
    - But: depends on the tree parser so much that if the parser doesn’t work well, it is impossible to generate a proper translation result.

System Structure of Our CJ EBMT System

- Our generation method
  - Uses the target sentence of the selected example to generate the translation of the input sentence.
  - Generate the translation in a finite automata state transfer manner.



Structure of our EBMT System

Generation Based on Finite Automata State Transfer

- Matching: select translation examples for the input sentence
  - Method: a combined method based on substantive word matching and stop word matching

$$\text{WordSim(A,B)}=2\cdot\frac{SameWC(A,B)}{len(A)+len(B)}$$

$$\text{StopWord\_Sim(A,B)}=\exp((abs(StopWord(A)-StopWord(B))\times\beta)$$

$$final\_Sim(A,B)=\frac{WordSim(A,B)}{StopWord\_Sim(A,B)}$$

- Generation
  - Step 1、 Build links from the fragments in the input sentence to the fragments in the target sentence of the selected example
  - Step 2、 Assign states to each of these links
  - Step 3、 Construct a finite automaton and generate the translation result in an automaton state transfer manner

Step 1 for Generation: Building Links

- Link ： a link from a fragment in one sentence S_1 to a fragment in another sentence S_2 is defined as a 3-tuple (Sf_i, Tf_j , t).
  - Sf_i: a fragment in S_1
  - Tf_i: a fragment in S_2
  - t: link type, we define four link types: *I, R, D, N*, which mean *inserting, replacing, deleting and outputting directly* respectively
- Build links from the fragments in the input sentence ***S*** to the fragments in the target sentence ***B*** of the selected example ***(A, B)***
  - First: Build links from ***S****'s* fragments to ***A****'s* fragments using a revised edit distance algorithm (will be shown in the next slide). Its result is denoted as *LinkSet(S→A)*.
  - Second: Build links from *S's* fragments to *B's* fragments (denoted as *LinkSet(S→B)*) according to following rules.
    - (a) For a link in *LinkSet(S→A)*, if neither its source fragment nor its target fragment is null, replace its target fragment with this target fragment's corresponding aligned fragment in *B*, and add this new link to *LinkSet(S→B)*.
    - (b) For a link in *LinkSet(S→A)* whose target fragment is null, add it to *LinkSet(S→B)* directly.
    - (c) For those fragments in *B* that have not been linked, build links for each of them by assigning a null source fragment and a *D* link type to them respectively, and add these links to *LinkSet(S→B)*.
    - (d) Reorder the items of *LinkSet(S→B)* in their target fragments' order in sentence *B*

Step 1 for Generation: Building Links

- The algorithm for building links from ***S*** *'s* fragments to ***A*** *'s* fragments is shown as followings.

```
m=length(S1), n=length(S2)
d[0][0] =0; tags[0][0] = 0;
for i=1 to m
   d[i][0]=q+d[i-1][0]; tags[i][0]='D'
for j=1 to n
   d[0][j]=r+d[0][j-1]; tags[0][j]='I'
for i=1 to m
  for j=1 to n
    p = computeCost(S1[i-1],S2[j-1]);
    a = d[i-1][j-1] + p;
    b=d[i-1][j] + q;
    c=d[i][j-1] + r;
    d[i][j] = min(a,b,c);
    if(min==a and p==0)
        tags[i][j] = 'N';
    else if (min==a)
        tags[i][j] = 'R';
    else if (min==b)
```

Revised Edit Distance Algorithm

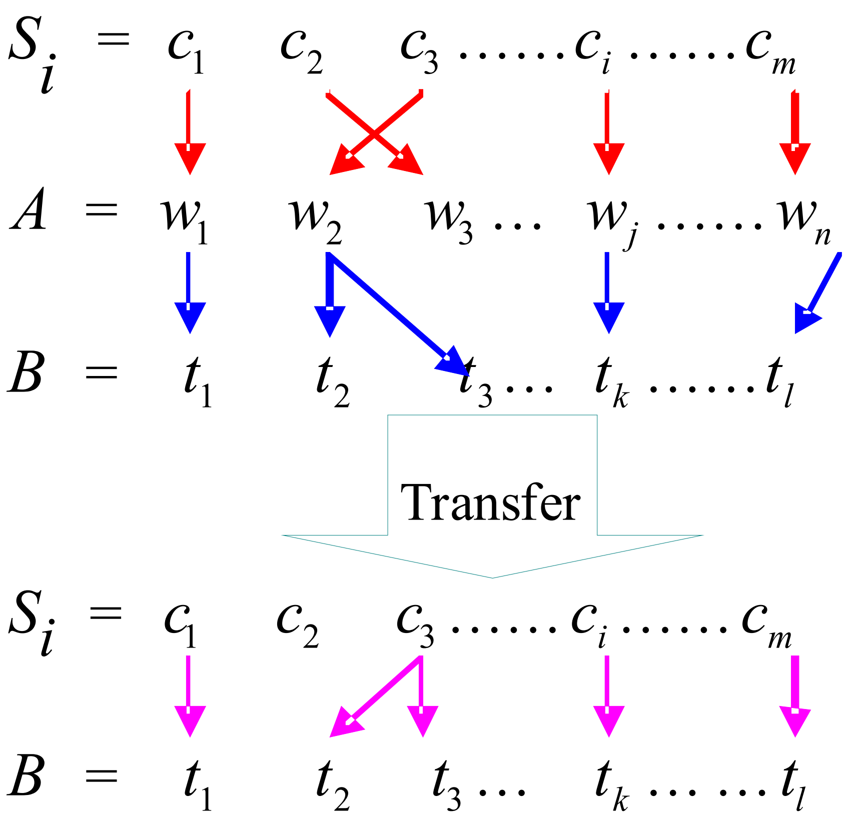
- ***computeCost*** is a function to compute two fragments' linking cost based on their lexical forms and their head words' POSs.
  - If two fragments' lexical forms are the same and their head words' POSs are the same too, this cost is zero;
  - if two fragments' lexical forms are the same but their head words' POSs are different, this cost is 0.2;
  - otherwise, this value is assigned by human's experiences according to the two fragments' head words' POSs as shown in the following table

Linking Cost for Two Fragments

$PosPair(c_i, c_j)$	w_i
(noun, noun)	0.5
(noun, auxiliary)	0.8
(noun, adjective)	0.85
…	…

Step 1 for Generation: Building Links

- The whole process of this step can be shown in the following figure

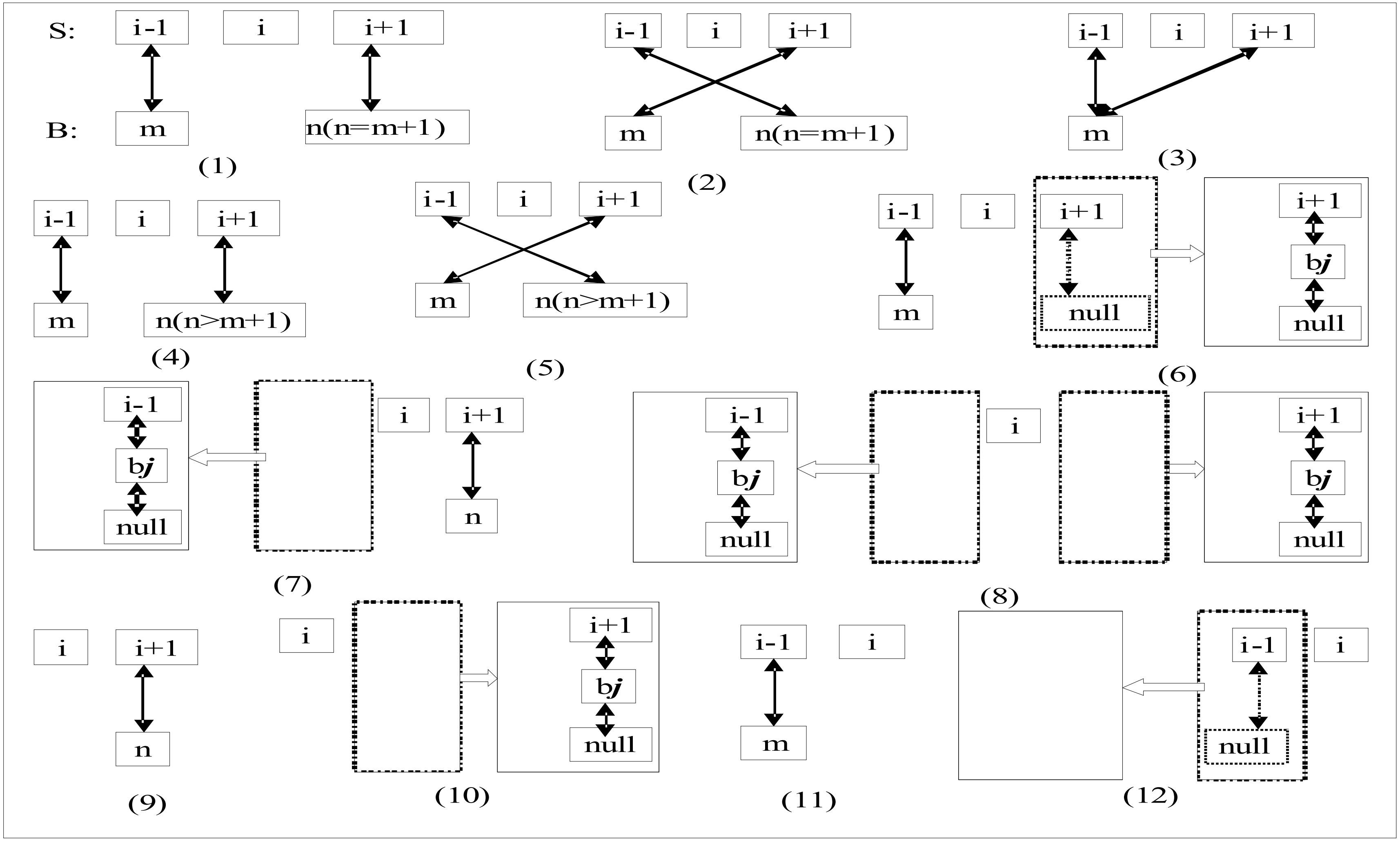


Step 2 for generation: States Assignment

- States for Non-*I* Type's Links
  - If its link type is *R*, a state named *S_R* is assigned
  - If its link type is *D*, a state named *S_D* is assigned;
  - If its link type is *N*, a state named *S_N* is assigned.
- States for *I* Type's Links
  - Consider context of current *I*-type link's pre- and post- links
  - Consider link shapes
  - Define 12 basic link shapes and 3 extended link shapes for *I*-type link, and map each of these link shapes to an *I*-type link's state.

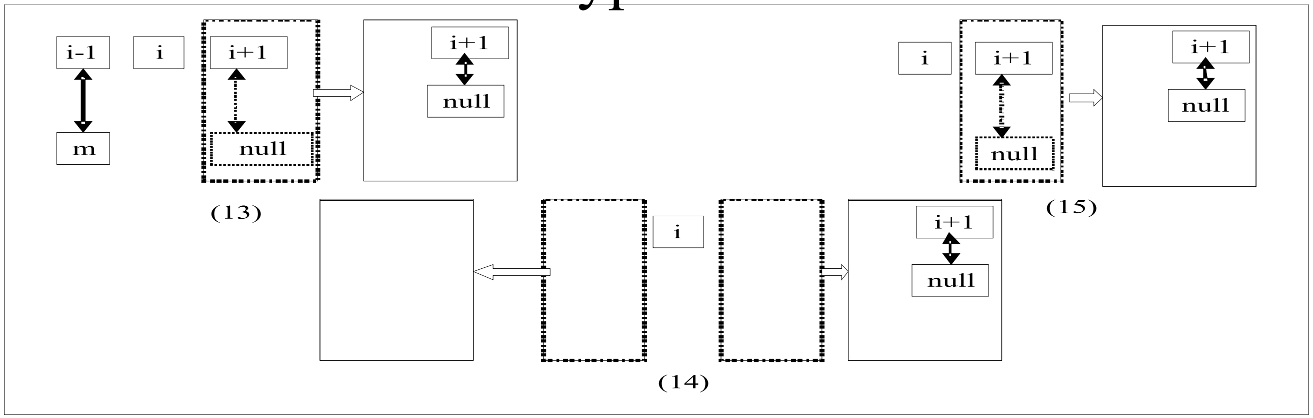
Step 2 for generation: States Assignment

➢Basic States for *I*-type's Link



Step 2 for generation: States Assignment

- Extended States for *I*-type's Link



- Extended states can be converted into basic states
  - For state 13, move rightward until find a non-*I* type's link, if this link's target fragment is null, convert it to state 6; otherwise, convert it to a state among state 1 to state 5 according to the link shapes of fragment *i*-1's link and the new found link; if can't find a non-*I* type's link in current link's right side, convert it to state 11.
  - For state 14, move rightward until find a non-*I* type's link, if this link's target fragment is null, convert it to state 8, otherwise, convert it to state 7; if can't find a non-*I* type's link in current link's right side, convert it to state 12.
  - For state 15, move rightward until find a non-*I* type's link, if this link's target fragment is null, convert it to state 10, otherwise, convert it to state 9; if can't find a non-*I* type's link in current link's right side, move leftward until find a non-*I* type's link (this link will be found always) and convert it to state 11.

Step 3 for generation: Translation Generation

- Generation Operation for Non-*I* Type Links' States
  - If a link's state is *S_R*, replace this link's target fragment with its source fragment's translation, and denote this operation as ***O(R)***;
  - If a link's state is *S_D*, delete this link's target fragment, and denotes this operation as ***O(D)***;
  - If a link's state is *S_N*, remain this link's target fragment unchanged, and denote this operation as ***O(N)***.
- Generation Operation for *I* Type Links' States
  - Take its source fragment's pre- and post- fragments into account and judge: whether the fragment combinations *(i-1,i,i+1)*, *(i-1,i)* and *(i,i+1)* are chunks. If they are chunks, look up their corresponding translations in dictionary, otherwise, look up *i's* translation in dictionary (we assume its translation can be found always).
  - According to current *I*-type link's state and the recognized chunk information, we choose one of these chunks as current *I*-type link's new source fragment for later processing, and define 10 possible generation operations

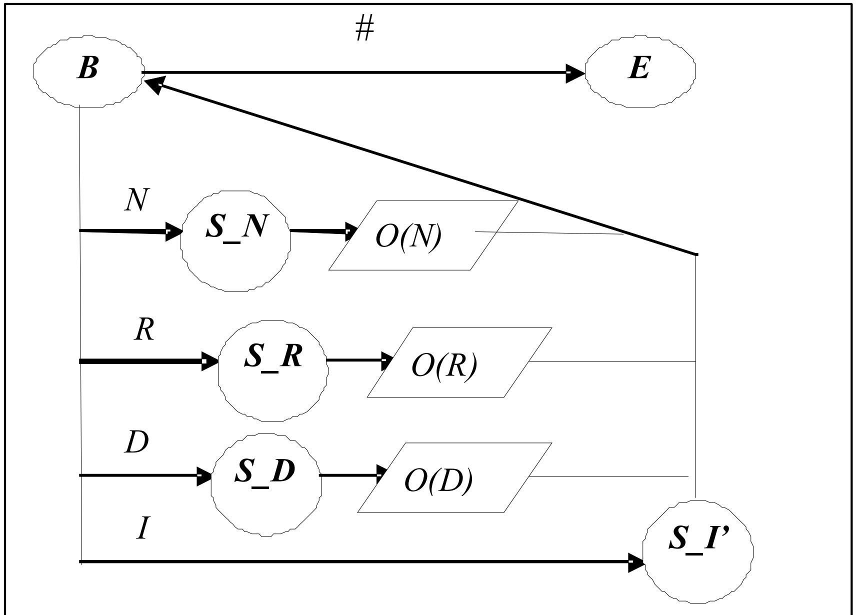
Step 3 for generation: Translation Generation

Generation Operation for *I* Type Links' States

- ***O(0):*** Delete the links that take *B's* fragments among *m+1* to *n* as their target fragments. And for the link that takes *B's* fragment *m* as target fragment, replace *m* with the translation of current *I*-type link's new source fragment.
- ***O(1)***: For the link that takes *B's* fragment *m* as target fragment, replace *m* with the translation of current *I*-type link's new source fragment.
- ***O(2):*** For the link that takes *B's* fragment *n* as target fragment, replace *n* with the translation of current *I*-type link's new source fragment.
- ***O(3):*** For the link that takes *B's* fragment *m* as target fragment, add the translation of current *I*-type link's new source fragment to the end of *m*.
- ***O(4):*** For the link that takes *B's* fragment *n* as target fragment, add the translation of current *I*-type link's new source fragment to the end of *n*.
- ***O(5):*** For the link that takes *B's* fragment *m* as target fragment, replace *m* with the translation of current *I*-type link's new source fragment. And delete the link that takes *B's* fragment *n* as target fragment.
- ***O(6):*** For the link that takes *B's* fragment *n* as target fragment, replace *n* with the translation of current *I*-type link's new source fragment. And delete the link that takes *B's* fragment *m* as target fragment.
- ***O(7):*** For the link that takes *B's* fragment *m* as target fragment, add the translation of current *I*-type link's new source fragment before *m*.
- ***O(8):*** For the link that takes *B's* fragment *n* as target fragment, add the translation of current *I*-type link's new source fragment before *n*.
- ***O(9):*** Do not modify any link's target fragment.

Step 3 for generation: Translation Generation

- Based on *LinkSet(S→B)* and the assigned states, we construct an automaton that has a similar form as shown in the following figure

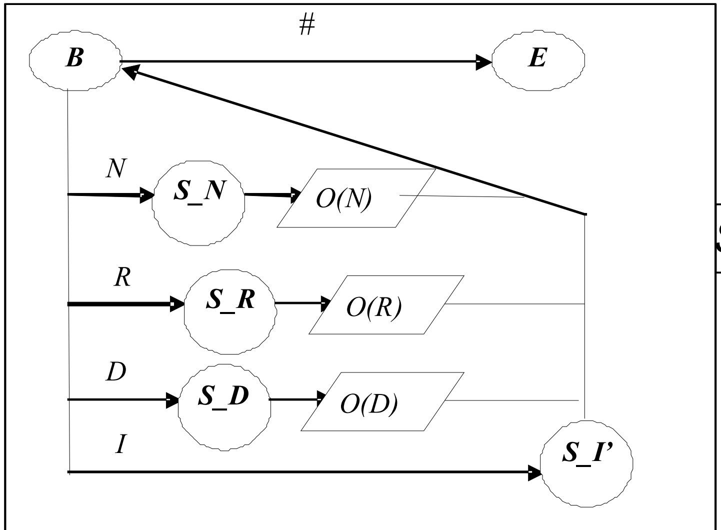


Finite Automation State Transfer Based Generation

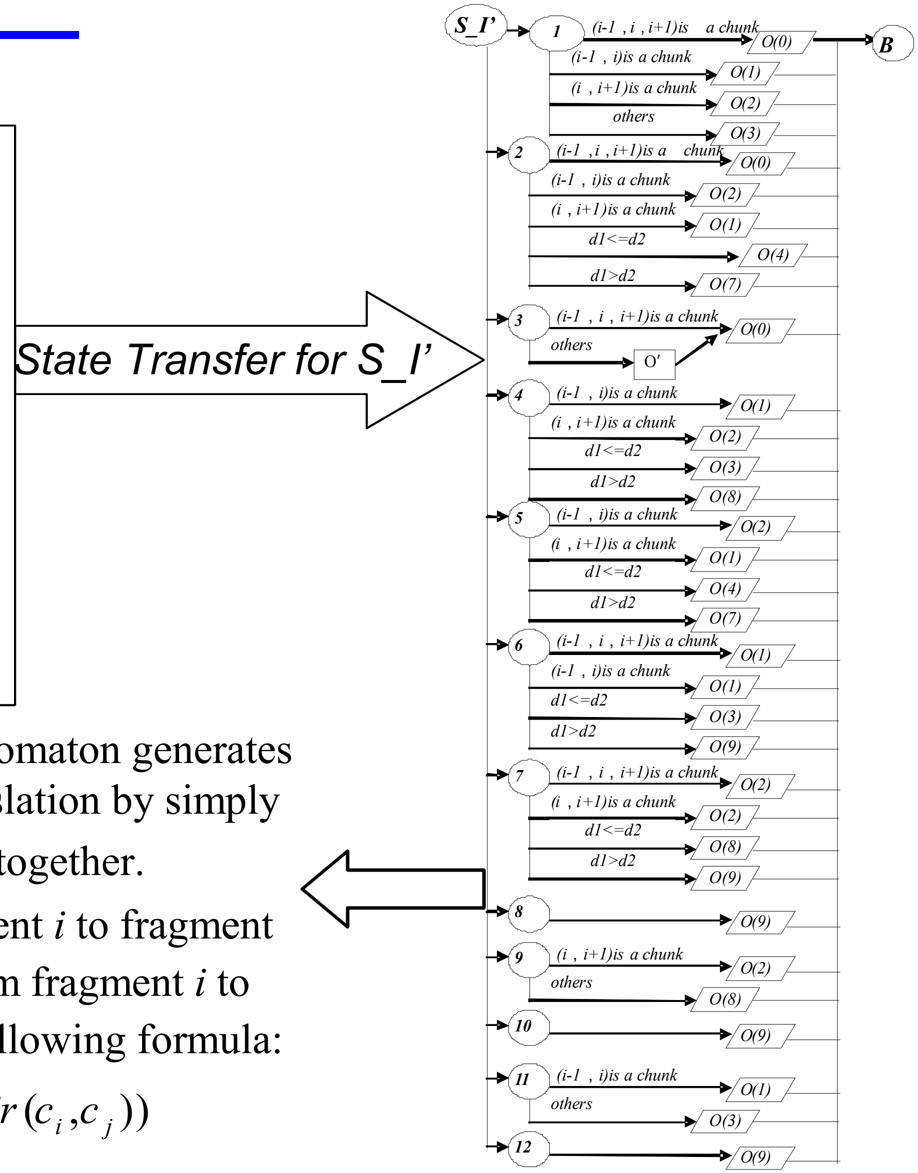
- *B* is a start state
- *E* is an end state
- *{I, R, D, N}* are link types
- *{O(N), O(D), O(R)}* in parallelogram are the operations
- # is a fictitious symbol that indicates the end of the automaton's input
- *{S_R, S_D, S_N}* are states correspond to non-I type's links
- *S_I'* is a state set that corresponds to *I*-type's links

Step 3 for generation: Translation Generation

- State Transfer for *S_I'*



Finite Automation State Transfer Based Generation



- *O'* in the operation of state 3 means the automaton generates the fragment combination *(i-1,i,i+1)'s* translation by simply joining their single fragment's translations together.
- d_1 means the semantic distance from fragment *i* to fragment *i-1*, and d_2 means the semantic distance from fragment *i* to fragment *i+1*, and they are computed as following formula:

$$dist(f_1,f_2) = \sum_{c_i \in f_1} \sum_{c_j \in f_2} w_k(PosPair(c_i,c_j))$$

Step 3 for generation: An Example

- Suppose *S* is "他很爱他的妻子*(He loves his wife very much)*". The selected example *(A,B)* is "(他爱他的妈妈*(He loves his mother)*，彼は、彼の母を愛しています*(He loves his mother)*)".
- After building links, *LinkSet(S→B)* is: (他*(he)*，彼*(he)*,*N*), (null,は*(ha)*,*D*), (很*(very much)* ,null,*I*), (他的 (*his*),彼の*(his)*,*N*), (妻子(wife),母 *(mother)*, *R*), (null,を*(wo)*,*D*) (爱*(loves)*,愛しています*(loves)*,*N*)
- Its corresponding state sequence is: *S_N, S_D, S_I_4(*the forth state in the basic *I*-type's links*),S_N, S_R, S_D, S_N*.
- Construct an automaton, and begin to states transfer and t ranslation generation.
- For the link (他*(he)*,彼*(he)*,*N*), its state is *S_N*. The automaton executes operation *O(N)* and does not mo dify this link's target fragment.
- For the link (null,は*(ha)*,*D*), its state is *S_D*. The automaton executes operation *O(D)* and deletes this link's target fragment.
- For the link (很*(very much)* ,null,*I*), its state is *S_I_4*. If the fragment co mbination *(i-1,i)* "他 很*(he…very much)*" is a chunk and the correspon ding translation is "彼は、とても*(he…very much)* ", the automaton executes operation *O(1)*. It first takes this reco gnized chunk as current link 's new source fra gment. Then it selects the link whose target fra gment is "彼*(he)*", and this link is (他*(he)*,彼*(he)* ,*N*). Thirdly, it replaces the selected link 's target fragment with the tran slation of current *I*-type link's new source fragment. At last the s elected link is changed to (他*(he)*，彼は、とても*(he…very much)* , *N*).
- For the link (他的*(his)*,彼の母*(his)*,*N*), its state is *S_N*. The automaton executes oper ation *O(N)* and does not modify this link's target fra gment.
- For the link (妻子*(wife)*,母*(mother)*,*R*), its state is *S_R*. The automaton executes operation *O(R)* and replaces this link's target fragment with its source fragment 's translation. Finally current link is changed to (妻子*(wife)*,妻*(wife)*,*R*).
- For the link (null,を*(wo)*,*D*), its state is *S_D*. The automaton executes operation *O(D)* and deletes this link' s target fragment.
- For the link (爱*(loves)*,愛しています*(loves)*,*N*), its state is *S_N*. The automaton executes o peration *O(N)* and does not modify this link's target fragment.
- At last, the automaton ends the state tran sfer process and outputs *LinkSet(S→B)'s* modified target fragment s equence "彼は、とても彼の妻愛しています*(he loves his wife very much)*" and takes it as the input sentence 's transl ation.

Experiments

- System Resources
  - Bilingual Corpus: We collect 10083 Chinese-Japanese bilingual sentences from Internet in Olympic domain as examples
  - Bilingual Dictionary: A bilingual dictionary is used to translate the input fragment and to judge whether an input fragment is a chunk.
  - Language Model: We collected an approximate 1,400,000 words' Japanese monolingual corpus and a similar size's Chinese monolingual corpus from Internet, and trained a standard trigram Japanese language model for Chinese-to-Japanese EBMT system and a standard trigram Chinese language model for Japanese-to-Chinese EBMT system respectively.
  - Test Corpus: We collect another 100 bilingual sentences in Olympic domain from Internet as test corpus.
- Experimental Result

Experimental Results for Chinese-to-Japanese EBMT System

Method	NIST	BLEU
Baseline	4.8321	0.4913
Our System	***5.9729***	***0.7705***

Experimental Results for Japanese-to-Chinese EBMT System

Method	NIST	BLEU
Baseline	4.1275	0.4076
Our System	***5.0976***	***0.5908***

Experiments----Some Translation Examples

Input:	我们的足球被对方前锋拦截
Output:	私 たち の サッカー は 相手 の 前鋒 に 阻まれ た
Input:	摔跤强国俄罗斯和日本有很多足球俱乐部
Output:	レスリング の 強国 ロシア と 日本 に は 很多 足球 倶乐部 が ある
Input:	中国运动员孙英杰今年一直主攻马拉松
Output:	中国 の スポーツ 選手 英傑 は 今年 ずっと マラソン を 専攻 として いる

Some Translation Results for Chinese-to-Japanese Translation

Input:	審判員はいささかのためらいもなくペナルティーキックを科した
Output:	裁判 毫不犹豫 地 判罚 点球
Input:	中国のチームにはマンツーマンディフェンス戦術がある
Output:	中国 的 队 有 盯人 战术
Input:	スウェ－デンのチ－ム２０分のペナルティを受けた
Output:	瑞典 队 被 罚 了 ２０ 分

Some Translation Results for Japanese-to-Chinese Translation

Conclusions and Future Work

- Conclusions:
  - The natural of the states are some transfer rules.
  - Our work can work on most of language pairs.
  - It doesn’t need any complicated parsers.
- Future Work
  - Merge syntax analysis into our method
  - Merge probability knowledge into state assignment and generation.

The End

- Thanks!
- If you have any question, please contact me by renfeiliang@gmail.com, or renfeiliang@ise.neu.edu.cn
- Welcome to my website: http://www.nlplab.cn/renfeiliang/