

Transformation-Based Learning

Torbjörn Lager

Tagging and Learning

- Distinguish tagging from learning
 - Tagging: the process of applying 'linguistic knowledge' to untagged text in order to produce tagged text
 - Learning: the process of inducing 'linguistic knowledge' from (manually tagged) text
- In this part of the course
 - What to learn? Transformation-rules
 - How to learn? Transformation-based learning

Transformation-Based Tagging: Small Part-of-Speech Tagging Example

lexicon

data:NN
decided:VB
her:PN
she:PN N
table:NN VB
to:TO

rules

```
pos:NN>VB <- pos:TO@[-1] o
pos:VB>NN <- pos:DT@[-1] o
....
```

input

She decided to table her data
NP VB TO NN PN NN

Transformation-Based Tagging

- Start with baseline tagger.
- Apply each transformation rule to whole corpus, one at a time.
- Later transformation rules can undo some of the previous transformation rules' work.
- Transformations have the format
If <context> then change tag <x> to <y>.

Rules for Part-of-Speech Tagging

```
pos:add T <- pos:C@[0] & {lex(C,T)} o
pos:'NN'>'VB' <- pos:'TO'@[-1] o
pos:'VBP'>'VB' <- pos:'MD'@[-1,-2,-3] o
pos:'NN'>'VB' <- pos:'MD'@[-1,-2] o
pos:'VB'>'NN' <- pos:'DT'@[-1,-2] o
pos:'VBD'>'VBN' <- pos:'VBZ'@[-1,-2,-3] o
pos:'VBN'>'VBD' <- pos:'PRP'@[-1] o
pos:'POS'>'VBZ' <- pos:'PRP'@[-1] o
pos:'VB'>'VBP' <- pos:'NNS'@[-1] o
pos:'IN'>'RB' <- wd:as@[0] & wd:as@[2] o
pos:'IN'>'WDT' <- pos:'VB'@[1,2] o
pos:'VB'>'VBP' <- pos:'PRP'@[-1] o
pos:'IN'>'WDT' <- pos:'VBZ'@[1] o
...
```

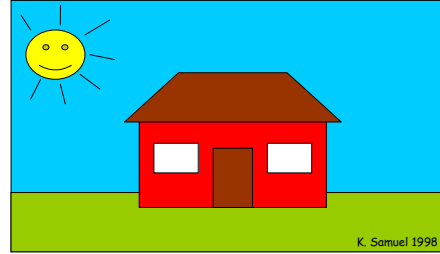
Transformation Rules a'la Eric Brill

```
NN VB PREVTAG TO
VB VBP PREVTAG PRP
VBD VBN PREVIOR2TAG VBD
VBN VBD PREVTAG PRP
NN VB PREVIOR2TAG MD
VB VBP PREVTAG NNS
VB NN PREVIOR2TAG DT
VBN VBD PREVTAG NNP
VBD VBN PREVIOR2OR3TAG VBZ
IN DT PREVTAG IN
VBP VB PREVIOR2OR3TAG MD
IN RB MDANDAPT SB SB
VBD VBN PREVIOR2TAG VB
RB JJ NEXTTAG NN
VBP VB PREVIOR2OR3TAG TO
POS VBZ PREVTAG PRP
NN VBP PREVTAG PRP
DT PDT NEXTTAG DT
...
```

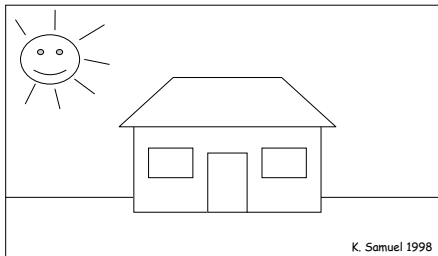
Approaches to Tagging (Joakim's classification)

- HMM tagging = The bold approach: 'Use all the information you have and guess'
- CG tagging = The cautious approach: 'Don't guess, just eliminate the impossible!'
- TB tagging = The whimsical approach: 'Guess first, then change your mind if necessary!'

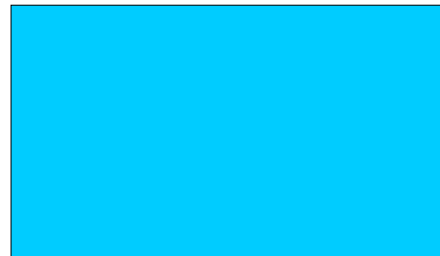
Transformation-Based Painting



Transformation-Based Painting



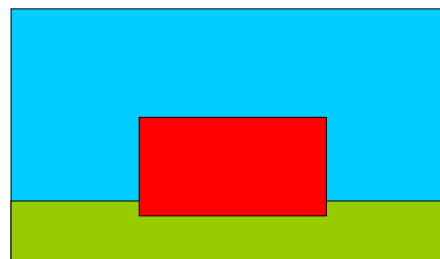
Transformation-Based Painting



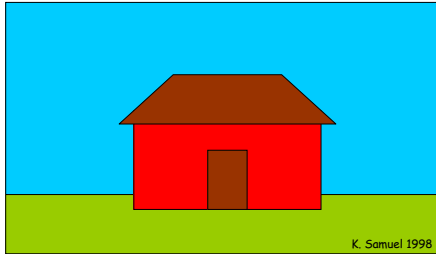
Transformation-Based Painting



Transformation-Based Painting

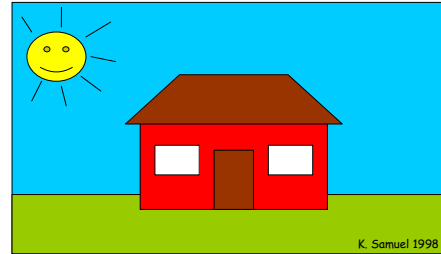


Transformation-Based Painting



K. Samuel 1998

Transformation-Based Painting



K. Samuel 1998

Advantages of TB Tagging

- Transformation rules are intelligible
- Transformation rules can be created/edited manually
- Transformation rule sequences are compact
- Sequences of transformation rules have a declarative, logical semantics
- TB taggers are simple to implement
- Transformation-based taggers can be extremely fast (but then implementation is more complex)

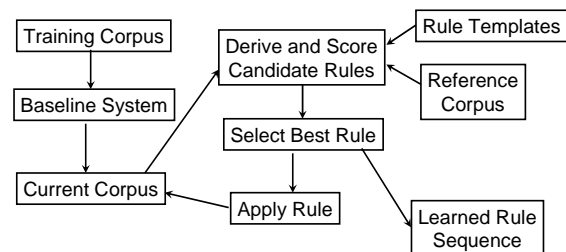
Transformation-Based Learning

- Invented by Eric Brill in the beginning of the nineties
- TBL has been used to learn rules for many natural language processing tasks, such as
 - part-of-speech tagging (Brill 1992)
 - pp-attachment disambiguation (Brill & Resnik 1994)
 - text chunking (Ramshaw & Marcus 1995)
 - spelling correction (Mangu & Brill 1997)
 - dialogue act tagging (Samuel et al. 1998)
 - ellipsis resolution (Hardt 1998)
- See also the Transformation-Based Learning Bibliography at <http://www.ling.gu.se/~lager/Mutbl/bibliography.html>

The μ -TBL System

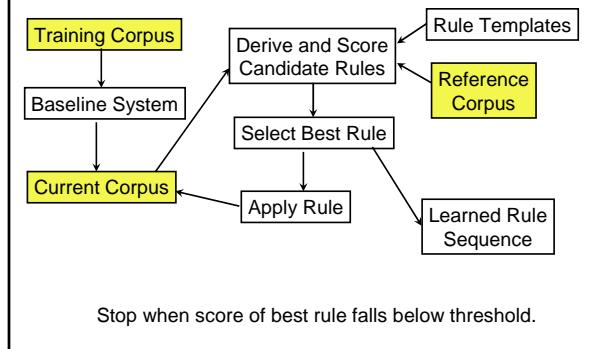
- Logic programming tools for TBL
- Features
 - General – supports four types of rules
 - Transparent – rules have a declarative, logical semantics and the system is derived from there
 - Extensible – templates and 'pluggable' algorithms
 - Efficient – the μ -TBL system is an order of magnitude faster than Brill's learner
 - Interactive – Prolog is an interactive language and this is something that the μ -TBL system inherits
 - Small – yes, but μ -TBL Lite is even smaller!

Learning Transformation Rules



Stop when score of best rule falls below threshold.

Learning Transformation Rules



Various Corpora

- Training corpus
w0 w1 w2 w3 w4 w5 w6 w7 w8 w9 w10
- Current corpus (CC 1)
dt vb nn dt vb kn dt vb ab dt vb
- Reference corpus
dt nn vb dt nn kn dt jj kn dt nn

Representing Data in the μ -TBL System

- Assuming the part of speech tagging task, corpus data can be represented by means of three kinds of clauses:

- `wd(P, W)` is true iff the word `W` is at position `P` in the corpus
- `tag(P, A)` is true iff the word at position `P` in the corpus is tagged `A`
- `tag(A, B, P)` is true iff the word at position `P` in the corpus is tagged `A` and the correct tag for the word at `P` is `B`

- For example, corresponding to the sentence "The can rusted" we have:

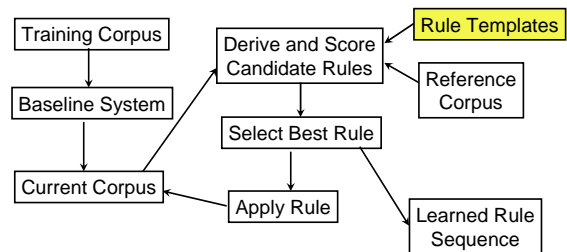
```

:- dynamic wd/2, tag/2, tag/3.
:- set data_size=3.
  
```

```

wd(1, 'The').      tag(1, det).      tag(det, det, 1).
wd(2, can).        tag(2, aux).      tag(aux, n, 2).
wd(3, rusted).    tag(3, v).        tag(v, v, 3).
  
```

Learning Transformation Rules



Rule Templates

- In TBL, only rules that are instances of *templates* can be learned.

- For example, the rules

```

tag:'VB'>'NN' <- tag:'DT'@[-1].
tag:'NN'>'VB' <- tag:'DT'@[-1].
  
```

- are instances of the template

```

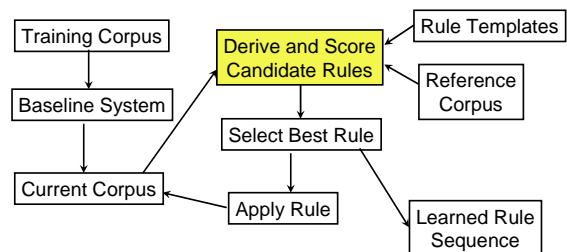
tag:A>B <- tag:C@[-1].
  
```

- Alternative syntax using anonymous variables

```

tag:~_ <- tag:~@[-1].
  
```

Learning Transformation Rules



Score, Accuracy and Thresholds

- The *score* of a rule is the number of its positive matches minus the number of its negative instances:

$$\text{score}(R) = |\text{pos}(R)| - |\text{neg}(R)|$$
- The *accuracy* of a rule is its number of positive matches divided by the total number of matches of the rule:

$$\text{accuracy}(R) = \frac{|\text{pos}(R)|}{|\text{pos}(R)| + |\text{neg}(R)|}$$
- The *score threshold* and the *accuracy threshold* are the lowest score and the lowest accuracy, respectively, that the highest scoring rule must have in order to be considered.
- In *ordinary* TBL, we work with an accuracy threshold < 0.5 .

Derive and Score Candidate Rule 1

```
Template = tag: >_ <- tag: @[-1]
R1 = tag: vb > nn <- tag: dt @[-1]
```

CC i	dt	vb	nn	dt	vb	kn	dt	vb	ab	dt	vb
CC i+1	dt	nn	nn	dt	nn	kn	dt	nn	ab	dt	nn

Ref. C	dt	nn	vb	dt	nn	kn	dt	jj	kn	dt	nn
--------	----	----	----	----	----	----	----	----	----	----	----

```
pos(R1) = 3
neg(R1) = 1
score(R1) = pos(R1) - neg(R1) = 3 - 1 = 2
```

Derive and Score Candidate Rule 2

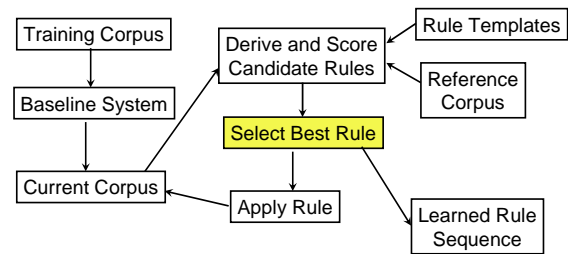
```
Template = tag: >_ <- tag: @[-1]
R2 = tag: nn > vb <- tag: vb @[-1]
```

CC i	dt	vb	nn	dt	vb	kn	dt	vb	ab	dt	vb
CC i+1	dt	vb	vb	dt	vb	kn	dt	vb	ab	dt	vb

Ref. C	dt	nn	vb	dt	nn	kn	dt	nn	kn	dt	nn
--------	----	----	----	----	----	----	----	----	----	----	----

```
pos(R2) = 1
neg(R2) = 0
score(R2) = pos(R2) - neg(R2) = 1 - 0 = 1
```

Learning Transformation Rules



Stop when score of best rule falls below threshold.

Select Best Rule

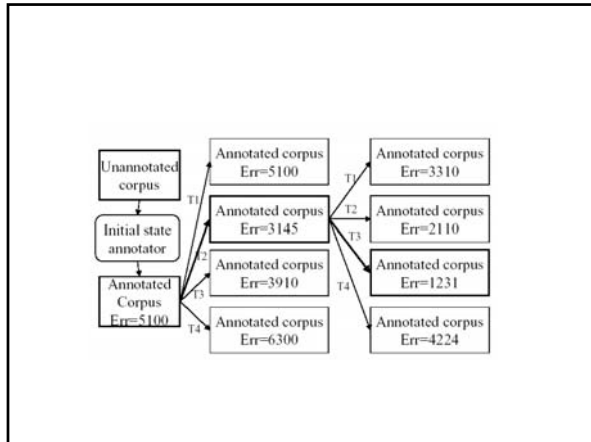
- Current ranking of rule candidates

```
R1 = tag: vb > nn <- tag: dt @[-1]   Score = 2
R2 = tag: nn > vb <- tag: vb @[-1]   Score = 1
...
```

- If score threshold ≤ 2 then select R1, else if score threshold > 2 , terminate.

Optimizations

- Reduce some of the naive generate-and-test behaviour: We only need to generate candidate rules that have at least one match in the training data.
- Incremental evaluation: Keep track of the leading rule candidate. If the number of positive matches of a rule is less than the score for the leading rule, we don't need to count the negative matches, and in any case we may not need to count *all* the negative matches.



Greedy Best-First Search

- $h(n)$ = estimated cost of the cheapest path from the state represented by the node n to a goal state
- Best-first search with h as its evaluation function
- NB: Greedy best-first search is not necessarily optimal

Looking back

- Supervised learning
- TBL is rule-based (since it produces rules)
- TBL is statistics-based (since it counts frequencies etc. in order to produce rules)
- ... just for PoS tagging? No, can do much more with clever problem encoding/representation!

TBL Strong Points

- Conceptually simple
- Simple to implement
- Simple to adapt to different learning problems

Disadvantages of TBL... or not!

- Learning is slow.
 - No, see the work on fn-TBL (Ngai & Florian 2001)
- No probabilities in result.
 - Well, see (Florian, Henderson & Ngai 2000)
- Supervised learning only
 - No, see (Brill 1997). But also (Becker 2000)...
- 'Patching' is simply inelegant...
 - No, see (Lager & Nivre 2001)
 - And (Roche & Shabes 1995)

Unknown-Word Guessing

- Problem
 - Words that don't occur in the lexicon
- TBL solution
 - Learn rules that perform suffix/prefix analysis
- Templates


```

tag:A>B <- suff:C@{0}.
tag:A>B <- pref:C@{0}.
tag:A>B <- suffixlex:C@{0}.
tag:A>B <- prefixlex:C@{0}.
tag:A>B <- charin:C@{0}.

% Auxiliary predicates
suff(P,C) :- wd(P,W), atom_suffix(4,W,C).
pref(P,C) :- wd(P,W), atom_prefix(4,W,C).

:- ['c:/Mutbl/wordlist'].
suffixlex(P,C) :- wd(P,W), atom_suffix(4,W,C,Rest), w(Rest).
prefixlex(P,C) :- wd(P,W), atom_prefix(4,W,C,Rest), w(Rest).

charin(P,C) :- wd(P,W), char_in(W,C).
      
```

Unknown-Word Guessing (cont'd)

- Example rules

```
tag: 'NN' > 'NNS' <-suff:s@[0]
tag: 'NN' > 'VBN' <-suff:ed@[0]
tag: 'NN' > 'VBG' <-suff:ing@[0]
tag: 'NN' > 'JJ' <-suff:al@[0]
tag: 'NNS' > 'JJ' <-suff:us@[0]
...
```

- Performance is so so... (Megyesi 2002)

What is Word Sense Disambiguation?

- Senses of the noun "interest" (according to the LDOCE):
 1. Quality of causing attention to be given
 2. Readiness to give attention
 3. Activity, subject, etc., which one gives time and attention to
 4. Advantage, advancement, or favour
 5. A share (in a company, business, etc.)
 6. Money paid for the use of money
- At the same time, the drop in interest rates since the spring has failed to revive the residential construction industry.
- Cray Research will retain a 10% **interest** in the new company, which will be based in Colorado Springs.
- Although that may sound like an arcane manoeuvre of little **interest** outside Washington, it would set off a political earthquake.

Corpus Data

- Prepared by Rebecca Bruce and Janyce Wiebe (Bruce & Wiebe, 1994).
- More than 2300 sentences extracted from the Penn Treebank
- Occurrences of *interest* and *interests* manually tagged with tags corresponding to the six LDOCE senses.

TBL Experimental Setup

- Known fact
 - The sense of an occurrence of a word can quite successfully be determined from just looking at the two previous words and the two following words (cf. Ide & Véronis 1998).
- Templates
 - In a first experiment on training a sense disambiguation expert, the following set of 7 templates were used:

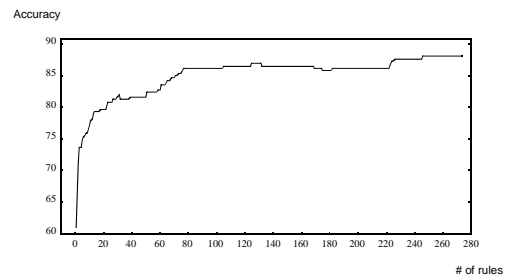
```
sense:A>B <- wd:C@[-1].
sense:A>B <- wd:C@[-1,-2].
sense:A>B <- wd:C@[1].
sense:A>B <- wd:C@[1,2].
sense:A>B <- wd:C@[1] & wd:D@[2].
sense:A>B <- wd:C@[-1] & wd:D@[-2].
sense:A>B <- wd:C@[-1] & wd:D@[1].
```

Rules for Word Sense Disambiguation

- Training resulted in a sequence of 273 rules, the first eight of which are shown below

```
sense:6>1 <- wd:in@[1] o
sense:1>5 <- wd:'%'@[-1,-2] o
sense:6>5 <- wd:short@[-1] o
sense:5>3 <- wd:pursue@[-1,-2] o
sense:5>4 <- wd:of@[1,2] o
sense:6>4 <- wd:public@[-1] o
sense:1>5 <- wd:a@[-1,-2] o
sense:6>4 <- wd:best@[-1,-2] o
...
```

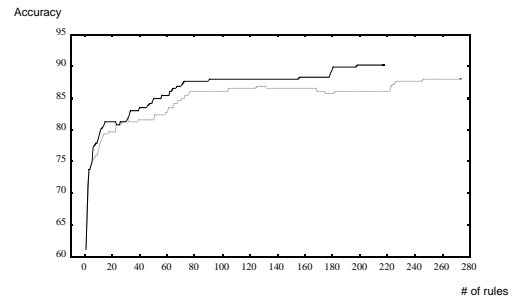
Learning Graph



Results

- Baseline: 60.9% accuracy.
- The accuracy peaks at 88.0%.
- The first rules are the most effective (the curve grows very fast in the beginning). There is no overtraining effect (it does not drop towards the end).

The Effect of Manual Modification



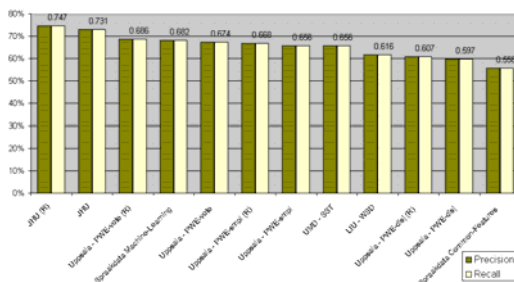
SENSEVAL-2: A Competition between Word Sense Disambiguation Systems

- The purpose of SENSEVAL is to evaluate the strengths and weaknesses of WSD programs with respect to different words, different varieties of language, and different languages.
- 12 languages: Basque, Czech, Chinese, Danish, Dutch, English, Estonian, Italian, Japanese, Korean, Spanish, Swedish
- About 35 teams participated, submitting over 90 systems.

The Swedish Lexical Sample Task

- 40 words: 20 nouns, 15 verbs and 5 adjectives
- Together representing 145 senses
- 8,718 annotated instances were provided as training material
- 1,527 unannotated instances were provided for testing

Swedish Lexical Sample (mixed-grained scoring)



NP Chunking

- Ramshaw and Marcus (1995) represented NP chunking as a tagging task by using three tags I, O, and B (meaning 'Inside', 'Outside' and 'Boundary', respectively).
- That is, instead of using brackets

```
In [NP early trading] in [NP Hong Kong]
[NP Monday], [NP gold] was quoted at
[NP $ 366.50] [NP an ounce] .
```

- they used tags, as follows:

```
In/O early/I trading/I in/O Hong/I Kong/I
Monday/B ,/O gold/I was/O quoted/O at/O $/I
366.50/I an/B ounce/I ./O
```

- Ramshaw and Marcus trained on POS-tagged WSJ-corpus data, using 100 templates sensitive to POS tags, words, and the tags I, O and B, in different combinations.
- They achieved an accuracy of around 92-93 %.

Experiment

- An NP-chunker was trained on 150,000 words of WSJ corpus, which produced 100 rules:

```
tag:i>o <- tag:oa[1] & pos:jj@[0] o
tag:i>b <- tag:i@[-2] & tag:i@[-1] & pos:dt@[0] o
tag:i>b <- tag:i@[-1] & pos:wtd@[0] o
tag:o>i <- tag:oa[-2] & tag:i@[-1] & pos:dt@[-1] o
tag:i>b <- tag:i@[-1] & wd:whos@[0] o
tag:i>b <- tag:i@[-1] & pos:prp@[0] o
tag:o>i <- tag:i@[-1] & pos:cc@[0] & pos:nn@[1] o
tag:o>i <- wd: & @[0] o
tag:i>o <- pos:in@[0] o
tag:o>i <- tag:i@[1] & tag:i@[2] & wd:about@[0] o
tag:o>i <- tag:i@[-1] & pos:cc@[0] & pos:nns@[1] o
tag:o>i <- tag:i@[1] & wd:only@[0] o
tag:h>i <- pos:jj@[0] o
tag:i>b <- tag:i@[-1] & pos:prp@[-1] o
tag:i>o <- pos:vbn@[0] o
tag:o>i <- pos:vbn@[0] & pos:nns@[1] o
tag:o>i <- tag:i@[1] & pos:cc@[1] o
...
```

- Result**

- The rules (prefixed with a suitable default rule) were compiled into a chunker capable of performing shallow parsing/chunking of noun phrases in free text with an accuracy of over 90%.

Training a Dialogue Act Tagger

Torbjörn Lager and Natalia Zinovjeva

Motivation

- Repeat the experiments on dialogue act tagging performed by Ken Samuel and others at the University of Delaware (Samuel, K., Carberry, S. and Vijay-Shanker, K. 1998)
- But with a different corpus – the Maptask Corpus – (Carletta, J., Isard, A., Isard, S., Kowtko, J.C., Doherty-Sneddon, G. and Anderson, A.H. 1997)

The Maptask corpus

- Consists of 128 instructional two-person dialogues. One person – the route giver (g) – gives another person – the route follower (f) – instructions how to navigate a route through a landscape pictured on a map.
- Example

```
Giver: So the start is at the top left-hand side of the page (EXPLAIN)
Follower: Uh-huh (ACKNOWLEDGE)
```
- In the experiment, we used 35 dialogues (9002 utterances) for training, and 5 dialogues (996 utterances) for testing.

The Maptask Coding Scheme

- There are 12 dialogue act tags in the Maptask coding scheme.
 - Six initiating moves
 - Five response moves
 - There is also one pre-initiating move – ready
 - which prepares the conversation for a new game to be initiated

Six Initiating Moves

- instruct* – commands the partner to carry out an action
- explain* – states information which has not been elicited by the partner
- check* – requests the partner to confirm information that the checker has some reason to believe, but is not entirely sure about
- align* – checks the attention or agreement of the partner, or his/her readiness for the next move
- query_yn* – asks the partner any question which takes a "yes" or "no" answer and does not fall into the previous two categories
- query_w* – any query which is not covered by the other categories

Five Response Moves

- **acknowledge** – a verbal response which minimally shows that the speaker has heard the move to which it responds
- **reply_y** – any reply to any query with a yes-no surface form which means "yes", however that is expressed
- **reply_n** – a reply to a query with a yes/no surface form which means "no"
- **reply_w** – any reply to any type of query which doesn't simply mean "yes" or "no"
- **clarify** – a repetition of information which the speaker has already stated, often in response to a check move

Example

- Initially, the two turns

Giver: *So the start is at the top left-hand side of the page* (EXPLAIN)
 Follower: *Uh-huh* (ACKNOWLEDGE)

- are represented as follows:

```
s(1495,g)
u(1495,['So',the,start,is,at,the,top,'lefthand',
side,of,the,page]).
da(1495,acknowledge).
da(acknowledge,explain,1495).

s(1496,f)
u(1496,['Uh-huh']).
da(1496,acknowledge).
da(acknowledge,acknowledge,1496).
```

The Idea

- With an eye to a possible application within a dialogue system, we have decided to make template conditions sensitive to features of the current or previous utterances only.
- Conditions for changing the tag of an utterance are sensitive to:
 - the actual words and word combinations used in the utterance,
 - the length of the utterance,
 - the previous dialogue act(s),
 - the speaker's role in the dialogue (giver or follower), and
 - whether the speaker has changed since the previous utterance.

Rule Templates

```
da:A>B <- u_mem:W@[0].
da:A>B <- u_first:W@[0].
da:A>B <- u_last:W@[0].
da:A>B <- u_bigram:W@[0].
da:A>B <- da:C@[-1].
da:A>B <- da:C@[-1,-2].
da:A>B <- da:C@[-1] & da:D@[-2].
da:A>B <- da:C@[-1] & u_mem:W@[0].
da:A>B <- s:C@[0].
da:A>B <- s:C@[0] & da:D@[-1].
da:A>B <- s:C@[0] & u_mem:W@[0].
da:A>B <- s:C@[0] & u_bigram:W@[0].
da:A>B <- s_change:C@[0] & u_mem:W@[0].
da:A>B <- s_change:C@[0] & da:D@[-1].
da:A>B <- u_length:C@[0] & u_mem:W@[0].
da:A>B <- u_length:C@[0] & da:D@[-1].
```

Rules for Speech Act Recognition

```
da:ack>instruct <- s:g@[0] & u_mem:'...@[0] o
da:ack>reply_y <- u_mem:'Yeah@[0] o
da:ack>reply_n <- u_mem:'No@[0] o
da:ack>reply_y <- u_first:'Uh-huh@[0] o
da:ack>query_yn <- u_mem:you@[0] o
da:ack>explain <- u_mem:got@[0] o
da:instruct>align <- u_mem:'See@[0] o
da:instruct>query_yn <- u_bigram:(you,got)@[0] o
da:ack>reply_y <- u_mem:'Yes@[0] o
da:ack>reply_y <- da:align@[-1] o
da:ack>check <- u_first:'So@[0] o
da:ack>align <- s:g@[0] & u_mem:'Okay@[0] o
da:instruct>align <- u_mem:see@[0] o
....
```

Definitions

- Auxiliary predicates:

```
u_mem(P,W) :- u(P,Ws), member(W,Ws).
u_first(P,W) :- u(P,[W]).
u_last(P,W) :- u(P,U), last(W,U).
u_bigram(P,(W1,W2)) :-
  u(P,U), nextto(W1,W2,U).
s_change(P,no) :-
  s(P,A), P1 is P-1, s(P1,A), !.
s_change(P,yes).
u_length(P,Length) :-
  u(P,Ws),
  length(Ws,N),
  ( N == 1 -> Length = '1'
  ; N == 2 -> Length = '2'
  ; N == 3 -> Length = '3'
  ; N > 3, N <= 10 -> Length = '>3'
  ; Length = '>10'
  ).
```

Experimental Results

- Tagging each utterance in the test data with the most common dialogue act in the training data (acknowledge) gave a base-line correctness of 21.6%.
- The learning process resulted in a sequence of 348 rules,
- by means of which we were able to tag the test corpus with an accuracy of 62.1%.

A Closer Look at the Rules

- Rules where 'cue-words' indicate dialogue acts of various kinds:


```
da:ack>reply_n <- u_mem:'No'@[0]
da:ack>reply_y <- u_first:'Uh-huh'@[0]
da:ack>check <- u_first:'So'@[0]
da:query_yn>instruct <- u_mem:go@[0]
```
- The second of these rules is later followed by a rule which reverses that change again if the utterance is preceded by an instruct act:


```
da:reply_y>ack <- da:instruct@[-1] & u_mem:'Uh-huh'@[0]
```
- Other rules capture well-known regularities, e.g. the tendency that questions are often followed by replies


```
da:explain>reply_w <- s_change:yes@[0] & da:query_w@[-1]
```
- or that replies are usually *not* followed by other replies:


```
da:reply_n>ack <- da:reply_n@[-1]
```
- Common word order configurations – captured in bigrams – signal other dialogue acts:


```
da:explain>query_yn <- u_bigram:(do,you)@[0]
```
- Long utterances with pauses in them (transcribed as '...' in the corpus) are usually instructions:


```
da:query_yn>instruct <- u_length:'>10'@[0] & u_mem:...@[0]
```

Learning Constraint Grammar Rules

- **TB tagging**
 - A lexical lookup module assigns *exactly one* tag to each occurrence of a word (usually the most frequent tag for that word type), disregarding context
 - A rule application module then proceeds to *replace* some of the tags with other tags, on the basis of what appears in the local context.
- **CG tagging**
 - A lexical lookup module assigns *sets* of alternative tags to occurrences of words, disregarding context.
 - A rule application module then *removes* tags from such sets, on the basis of what appears in the local context.
 - However, the rule application module adheres to the following principle: don't remove the last remaining tag

CG-Like Rules

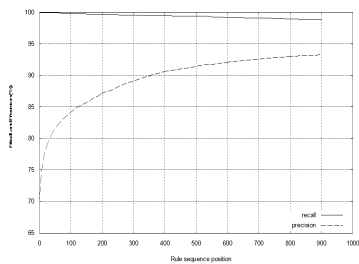
- CG rule sequence


```
pos:add T <- wd:W@[0] & {lexlookup(W,T)} o
pos:red vb <- pos:dt@[-1] o
pos:red mn <- wd:will@[0] & pos:vb@[1] o
...
```
- Selection rules

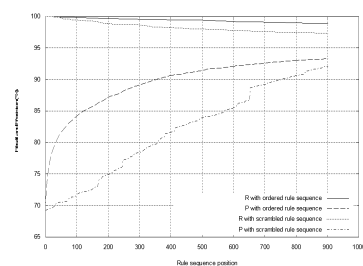

```
pos:sel dt <- wd:the@[0] & pos:nn@[1]
```
- Barrier rules


```
pos:red vb <- unique pos:dt@[-*] barrier [nn]
```

Learning Graph



Order Matters



Using Your Rules

Four Ways to Build a Tagger

- Compile into programming language code
- Transform into FOPL. Then use a general theorem prover.
- Compile into Finite State Transducer
- Incremental tagging

Compiling Rules into Prolog Code

```

tag:vb>nn <- tag:dt@[-1] o
tag:dt>pn <- tag:vb@[1] o
end.

apply_all :-
    apply_one,
    fail.
apply_all.

apply_one :-
    tag(P,vb),
    P1 is P-1,
    tag(P1,dt),
    retract(tag(P,vb)),
    assert(tag(P,nn)).
apply_one :-
    tag(P,dt),
    P1 is P+1, tag(P1,vb),
    retract(tag(P,dt)),
    assert(tag(P,pn)).

:- dynamic tag/2.
wd(1,'The').    tag(1,det).
wd(2,can).     tag(2,aux).
wd(3,rusted).  tag(3,v).
    
```

Compiling Rules into Java Code

```

class ContextRules {
    public String[] apply(String[] tag, String[] wd) {
        int tlength = tag.length;
        String[] tcopy = new String[tlength];
        System.arraycopy(tag,0,tcopy,0,tlength);
        // tag:vb>nn <- tag:dt@[-1]
        for(int i=0; i<tlength; i++)
            if(tag[i].equals("vb"))
                if((i-1) >= 0 && tag[i-1].equals("dt"))
                    tcopy[i] = "nn";
        tag = tcopy;
        // tag:dt>pn <- tag:vb@[1]
        for(int i=0; i<tlength; i++)
            if(tag[i].equals("dt"))
                if((i+1) > tlength && tag[i+1].equals("vb"))
                    tcopy[i] = "pn";
        tag = tcopy;
        return tag;
    }
}
    
```

The Semantics of Transformations

- From rule sequences

```

sense:add 6 <- wd:interest@[0] o
sense:6>1 <- wd:in@[1] o
sense:1>5 <- wd:%@[-1] o
end
    
```

- to first order logic

```

Vp[wd(p,interest) -> sense(p,6)]
Vp,p,s[sense(p,6) ^ p_i=p_i+1 ^ wd(p_i,in) -> sense(p_i,1)]
Vp,p,x[sense(p,x) ^ p_i=p_i+1 ^ ~wd(p_i,in) -> sense(p_i,x)]
Vp,p,s[sense(p,1) ^ p_i=p_i-1 ^ wd(p_i,%) -> sense(p_i,5)]
Vp,p,x[sense(p,x) ^ p_i=p_i-1 ^ ~wd(p_i,%) -> sense(p_i,x)]
Vp,x[sense(p,x) -> sense(p,x)]
    
```

Word Sense Disambiguation as Deduction

```

■ wd(1,Sue) wd(2,developed) wd(3,an) wd(4,interest)
  wd(5,in) wd(6,computers) wd(7,and) wd(8,bought)
  wd(9,a) wd(10,20) wd(11,%) wd(12,interest) wd(13,in)
  wd(14,Microsoft)
    
```

- Determining the sense of a particular word occurrence

```

∃x[sense(12,x)]
    
```

- Searching for a word occurrence with a particular sense

```

∃p[sense(p,5)]
    
```

- Sense tagging

```

∃p,x[sense(p,x)]
    
```

Transformation into Prolog

```
sense:6>1 <- wd:in@[1]

∀p0,p1[sense1(p0,6) ∧ p1=p0+1 ∧ wd(p1,in) → sense2(p0,1)]
∀p0,p1,x[sense1(p0,x) ∧ p1=p0+1 ∧ ¬ wd(p1,in) → sense2(p0,x)]

sense2(p0,1) :- sense1(p,6), P1 is P0+1, wd(P1,in).
sense2(p0,S) :- sense1(p0,S), P1 is P0+1, \+ wd(P1,in).
```

Compilation into Efficient Prolog

```
sense(P,S) :- sense3(P,S).
sense3(P0,S) :-
    sense2(P0,S0),
    ( S0==1, P1 is P0-1, wd(P1,'%')
    -> S=5
    ; S=S0
    ).
sense2(P0,S) :-
    sense1(P0,S0),
    ( S0==6, P1 is P0+1, wd(P1,in)
    -> S=1
    ; S=S0
    ).
sense1(P,6) :- wd(P,interest).
```

Using the PWE

- For example, here is how the sense for the word at position 4 is identified

```
| ?- sense(4,S).
S = 1
```

- and here is how we search for occurrences that have the sense 5:

```
| ?- sense(P,5).
P = 12
```

Compile into FST

- Remember the characterization of TBL as 'the whimsical approach': 'Guess first, then change your mind if necessary!'

- Example

```
tag:'NN'>'NNS'<-suff:s@[0]
tag:'NN'>'VBN'<-suff:ed@[0]
tag:'NN'>'VBG'<-suff:ing@[0]
tag:'NN'>'JJ'<-suff:al@[0]
tag:'NNS'>'JJ'<-suff:us@[0]
...
```

- But the 'whimsical' behaviour can be compiled away! Just compile the rule sequence into a deterministic finite-state transducer.

Roche & Schabes

- A naive implementation of a transformation-based tagging require RCn elementary steps to tag an input of n words with R rules requiring at most C tokens of context.
- For each individual rule, the algorithm scans the input from left to the right while attempting to match the rule. This simple algorithm is computationally inefficient for two reasons:
 - the fact that an individual rule is matched at each token of the input, regardless of the fact that some of the current tokens may have been previously examined when matching the same rule at a previous position.
 - the potential interaction between rules. For example, one rule results in a change which is undone by a rule later in the sequence. The algorithm may therefore perform unnecessary computation.
- We present a finite-state tagger inspired by the rule-based tagger which operates in optimal time in the sense that the time to assign tags to a sentence corresponds to the time required to follow a single path in a deterministic finite-state machine. This result is achieved by encoding the application of the rules found in the tagger as a non-deterministic finite-state transducer and then turning it into a deterministic transducer. The resulting deterministic transducer yields a part-of-speech tagger whose speed is dominated by the access time of mass storage devices

Roche & Schabes (cont'd)

- Speed of the different parts of the program

	dictionary lookup	unknown words	contextual
Speed	12,800 w/s	16,600 w/s	125,100 w/s
% of the time	85%	6.5%	8.5%

- Comparison different taggers

	Stochastic Tagger	Rule-Based Tagger	Finite-State Tagger
Speed	1,200 w/s	500 w/s	10,800 w/s
Space	2,158KB	379KB	815KB

Taggers and Incrementality

- **Fact:** Transformation-based taggers perform well and are robust. Furthermore, rule sequences can be automatically learned from tagged corpora. This has made them popular for batch processing of text.
- **Question:** Are they useful also in a dialogue system setting?
- **Answer:** Yes, but only if they can be made to process input in an **incremental** fashion!

Cautious Incrementality

- Word-by-word incrementality
The light |
DT ?
- Trade-off: Incrementality vs accuracy
- Design decision: We don't want to compromise accuracy -> **cautious** incrementality
- "cautious" can mean different things...
 - delay disambiguation decisions
 - entertain representation of uncertainty
- Design decision: We prefer to delay disambiguation decisions

Implementation

- Ideas
 - Dataflow concurrency
 - Each rule runs in a separate thread
 - Gives us incrementality for free!
 - Implementation
 - In the Oz programming language
- <http://www.mozart-oz.org/>

Incremental Tagger Demo