

Lecture 14

September 6, 2016

Evaluation: Precision and Recall



Reminder: start the recording

Some material from Will Lewis

Announcements

- Project 4 Solution
<http://courses.washington.edu/ling473/project-4.html>
- Writing Assignment
 - Due tonight at 11:45 p.m.
- Project 5: Naïve Bayesian Classifier
 - Due Thursday at 11:45 p.m.
 - Solution will be posted on the course website
- Self-study project
 - Calculate edit distance between two documents
 - Solution posted next week

Evaluation

- Contemporary research in computational linguistics is unacceptable if it is not accompanied by principled evaluation

“An important recent development in NLP has been the use of much more rigorous standards for the evaluation of NLP systems”

Manning and Schutze

- Quantitative measurement of results is what differentiates our field from armchair theorizing
- It is one of the cross-cutting pillars of the CLMS curriculum to emphasize the critical importance of evaluation at all stages of research
- To be published, all research must:
 - establish a baseline, and
 - quantitatively show that it improves on the baseline

Basic Evaluation

- “How well does the system work?”
- Possible domains for evaluation
 - Processing time of the system
 - Space usage of the system
 - Human satisfaction
 - Correctness of results



example

- You are building a system which automatically provides short, human-readable summaries of a set of documents on a given topic
- Your system picks sentences from the documents based on word co-occurrences, and presents these sentences as the summary
- We want to evaluate the “quality” of the results
- One choice in such a system is whether you should use **stemming** when determining the word co-occurrences
- Let’s briefly examine **stemming**

Stemming

- Morphological suffixes used can make our data more sparse
- In content-analysis tasks (IE, IR, summarization), we may only care about ‘stems,’ because they carry the “content” of the lemma

example:

He doesn't like to **shop**.

She **shops** at the mall.

I went **shopping** last week.

Ben **shopped** until he dropped.

Bill is quite and avid **shopper**.

Porter stemmer

- One well known stemming algorithm for English is the Porter stemmer
 - M. F. Porter. 1980. *An algorithm for suffix stripping*. Program, 14(3):130–137.
- It is a heuristic which contains lots of code like this:

```
}  
else if ((ends("ed") || ends("ing")) && vowelinstem())  
{  
    k = j;  
    if (ends("at"))  
        setto("ate");  
    else if (ends("bl"))  
        setto("ble");  
    else if (ends("iz"))  
        setto("ize");  
    else if (doublec(k))  
    {  
        k--;  
        int ch = b[k];  
        if (ch == 'l' || ch == 's' || ch == 'z')  
            k++;  
    }  
    else if (m() == 1 && cvc(k)) setto("e");  
}
```

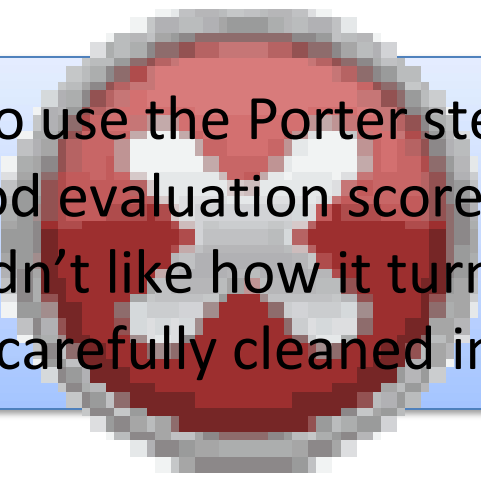
Porter stemmer

Two medical experts testifying Wednesday in the doping trial of a former East German sports doctor said the female swimmers they examined showed health damage linked to performance-enhancing drugs, including liver damage and excessive facial hair.

two medic expert testifi wednesdai in the dope trial of a former east german sport doctor said the femal swimmer thei examin show health damag link to performance-enhanc drug includ liver damag and excess facial hair

Back to our example

- Looking at the output of the Porter stemmer makes linguists cringe
- For the document summarizer system:



“We elected not to use the Porter stemmer because we were getting good evaluation scores with our system already, and we didn’t like how it turns the linguistic data that we had carefully cleaned into gibberish.”

Evaluate!

		ROUGE-1			ROUGE-2			ROUGE-SU4		
		min	max	avg	min	max	avg	min	max	avg
0	Baseline: sqrt(c) dampening; +stopwords +contractions	0.34597	0.36072	0.35246	0.0635	0.06641	0.06477	0.11843	0.12377	0.12077
1	No Quotes post-svd	0.3578	0.35836	0.35802	0.06763	0.06776	0.06769	0.12328	0.12347	0.12335
2	Quotes, no fluff- pre SVD	0.35316	0.35101	0.35203	0.06536	0.06496	0.06515	0.12141	0.12065	0.12101
3	New stopwords, 250-word-cramming	0.35034	0.34729	0.34878	0.06125	0.06071	0.06097	0.11692	0.1159	0.1164
4	250-word cramming off; no title boost	0.3502	0.34764	0.34888	0.06125	0.06078	0.06101	0.11691	0.11605	0.11647
5	experiments 1 + 4	0.35913	0.35751	0.35827	0.06646	0.06618	0.06631	0.12327	0.1227	0.12297
6	5 + simple stemming	0.36809	0.3653	0.36664	0.07163	0.0711	0.07135	0.12814	0.12716	0.12763
7	6 + banning quote sentences prior to SVD	0.366	0.36361	0.36475	0.07195	0.07151	0.07172	0.12792	0.12709	0.12749
8	5 + stem "-ing"	0.36841	0.36553	0.36692	0.07254	0.07197	0.07224	0.12901	0.12799	0.12848
9	8 + double-boost title words	0.36391	0.36079	0.36229	0.07126	0.07066	0.07095	0.12618	0.12506	0.1256
10	no title boost at all	0.36361	0.36108	0.3623	0.06979	0.06934	0.06956	0.12515	0.12429	0.12471
11	8 + single-boost, pre-svd word-based PTB POS tag filters	0.36682	0.36383	0.36527	0.07181	0.07123	0.07151	0.12817	0.12711	0.12762
12	checked in version as of 23:59 9/9/2010	0.36858	0.36573	0.3671	0.07265	0.07209	0.07236	0.12899	0.12797	0.12846
13	svd k=150, a popular value in the literature	0.37395	0.37132	0.37259	0.07683	0.0763	0.07656	0.13253	0.13157	0.13203
14	minor bug: wasn't stemming title word boost	0.37526	0.37239	0.37378	0.07781	0.07723	0.07751	0.13389	0.13285	0.13335
15	Porter stem instead of simple	0.37591	0.3736	0.37471	0.07875	0.07828	0.0785	0.13467	0.13381	0.13422



Now who's cringing?

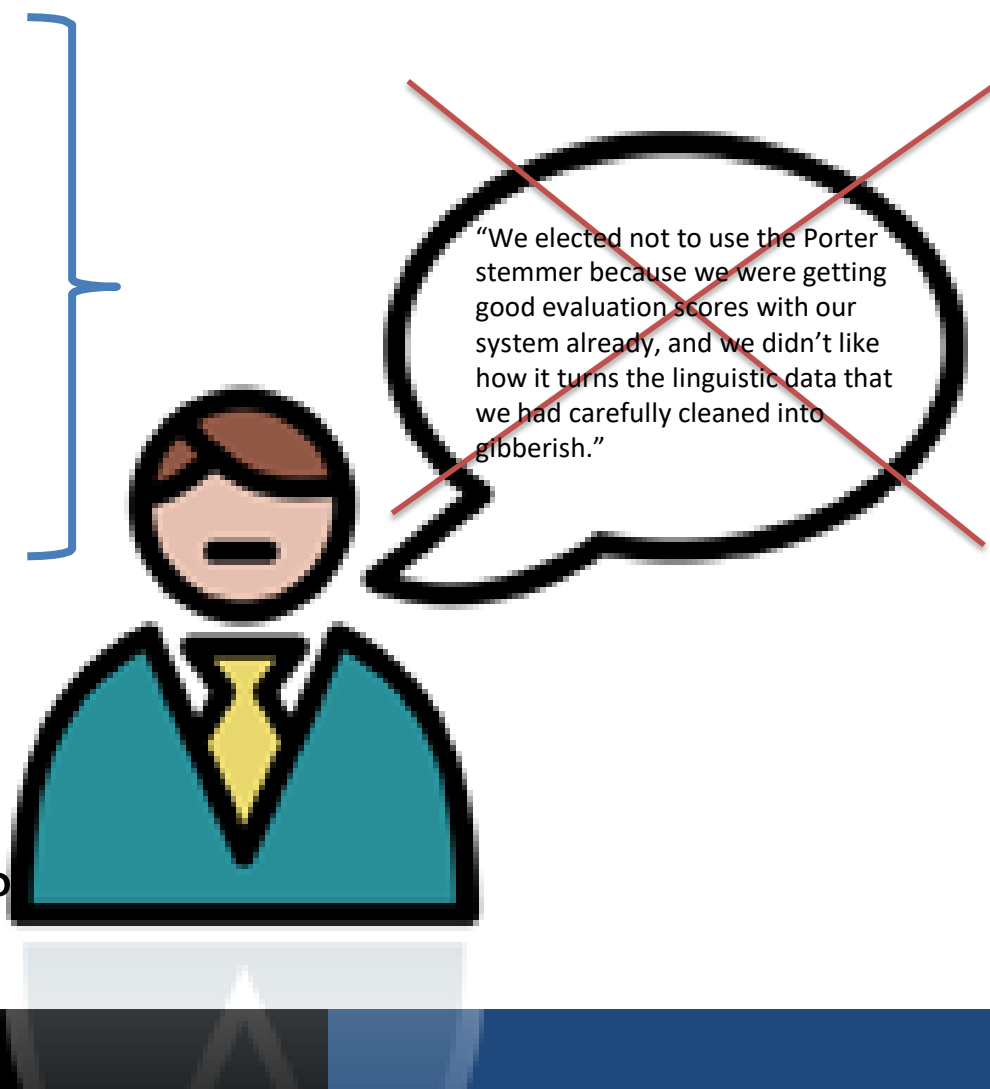
- Although the crude approach of the Porter stemmer seems intuitively ugly, *it improves our evaluation metric*
- As a computational linguist, you must remain objective
- If you have a legitimate linguistic intuition in mind, test it!
- Evaluate, evaluate, evaluate



~~"We elected not to use the Porter stemmer because we were getting good evaluation scores with our system already, and we didn't like how it turns the linguistic data that we had carefully cleaned into gibberish."~~

Now who's cringing?

- As a professional computational linguist, it is this type of statement that should immediately set off your alarm bells
- Always ask:
 - What is the **BASELINE**?
 - What is the **RESULT**?
 - How was it **EVALUATED**?



Stemming and Performance

- Does stemming help in IR, IE, and document summarization?
- Harman 1991 indicated that it hurt as much as it helped
 - D. Harman (1991) How effective is suffixing. In *Journal of the American Society for Information Science*. 42(7) 7-15.
- Krovetz 1993 shows that it does help
 - R. Krovetz (1993) Viewing morphology as an inference process. In *Proc. 16th ACM SIGR R&D IR* 191-202
 - Porter-like algorithms work well with smaller documents
 - Krovetz proposes that stemming loses information
 - Derivational morphemes tell us something that helps identify word senses; stemming them loses this

Evaluating a stemmer

- In the summarization example, the stemmer is a small part of the whole system
- We used an end-to-end measurement (ROUGE scores) to evaluate the impact of using stemming
- How would you evaluate the “performance” of stemming by itself?

“Correct” stemming

- This would be difficult, because there’s no “correct” stemming of a word
- The best stemmer is a hash function that conflates all words with the same linguistic stem: the actual stemmed token doesn’t matter

Stemmer #7

eat	04xBrLt
ate	04xBrLt
eating	04xBrLt
eats	04xBrLt

Better
than...

Porter Stemmer

eat	eat
ate	at
eating	eat
eats	eat

Therefore

- Results are sensitive to the specific application.
- Rule of thumb: when there's an implementation decision to be made:
 - Evaluate the alternatives
 - Document your results and choice
- Some choices may require going back and re-evaluating earlier decisions

Accuracy

- In order to evaluate your system, you need to know what is correct or desired
- A set of data which is labeled with the correct or desired result is called a **gold standard**
- If the system you are evaluating is a function that maps one input to one output, then you can evaluate **accuracy**
 - Correct: matches the gold standard
 - Incorrect: otherwise

$$\text{aaaaaaaaaaaaaaaa} = \frac{\text{nnaannnnnnnaa aaccdaaannaacc}}{\text{nnaannnnnnnaa ccoo daannrraarrccrr}}$$

Accuracy: example

- With your Naïve Bayesian classifier for language identification, say we are only interested in the single best language it selects for a given input sentence

“ مساء الخير ” → Arabic ✓

“Bon soir!” → French ✓

“Good evening!” → Spanish ✗

Accuracy:
.67
67%

Error

- You can also measure **error**. This is the proportion of items that you got wrong

$$nnaaaaccaa = \frac{nnaannnnnnnaa\ wwaaccnnww}{nnaannnnnnnaa\ ccoo\ aannrraarrccrr}$$

“مساء الخير” → Arabic ✓

“Bon soir!” → French ✓

“Good evening!” → Spanish ✗

Error:
.33
33%

Evaluating classifiers

- Many NLP problems involve classifying things
- For these problems, there is a more nuanced way to evaluate performance
- Also, note that many NLP problems can be re-stated as classification problems
- Before we talk about evaluating classification results, let's see how to re-state a problem as a classification problem

Example: Thai sentence-breaking

- In Thai, there is no end-of sentence punctuation character (such as the period)
- A space character ' ' always appears between sentences
- *BUT*, the space character is also used in other ways, within a sentence
- The problem: break Thai text into sentences

Sentence-breaking as classification

- Here is some Thai text that we would like to break into one or more sentences

ผมขอหนัง ขำ มาทกเลมทพอจะหาขอได้ ้ ทมรปประกอบก็ขอ
สอ ขน กระทบ
ถอขนมาบนหองในโรงแรมทผมพัก ั้ง พายเนอหมและโดนัทอีก
นอกจากนย ขอ
หลายสบอน ผมกนพายกบโดนัทแลว นั้งทอ อยบนเตยง อานหนังสอ
ดหย
การตนเลมแลวเลมเลา ในทสดผมก็ร้สวาวความงวงเหงารายกาจคอยๆแอบคบ
ูก ก
คลานเขามาในตว ผมจงเออมหยบหนัง ลอนดอนไทมร ายสปดาร์ แลว เป็ด
สอ
ทนา บทบรรณาธการคา งไวตรงหนา

- How can we treat this as a classification problem?



Sentence-breaking as classification

For each space character in the text: classify it as either sentence-breaking (sb) or non-sentence-breaking (nsb)

ผมขอหนึ่ง ขำ มาทกลेमทพอจะหาข้อได้ █ กระทบบั๊บท ประกอบก็ขอ
สอ ขน มรป
█ ถอนมาบนห้องในโรงแรมผมพัก █ นอกจากนัยัง พายเนอหมและโดนัท
ช่อ
อกหลายสบน █ ผมกนพายกบโดนัทแลว นั้งทอ อยบนเตยง █ ่อ่านหนึ่งสอ
ดหย
การตุนเล่มแล เล่มเล่า █ ผมก็ร้สวความงวเหงารายกาจคอยๆแอบคบ
ว ในทสด ู ก
คลานเขามาในตัว █ ผมจงเออมหยบหนึ่ง ลอนดอนไทมร ายสปดาห █ แลว เป็ด
สอ

Reading the literature

- Refereed journal papers will have Results and Evaluation sections
- Evaluation is often shown in a table versus previous work
- If skimming papers, skip to the results and evaluation section for a summary of the work

Our results are consistent with recent work using the Winnow algorithm, which itself compares favorably with the probabilistic POS trigram approach. Both of these studies use evaluation metrics, attributed to Black and Taylor (1997), which aim to more usefully measure sentence-breaker utility. Accordingly, the following definitions are used in Table 2:

$$\text{space-correct} = \frac{(\# \text{correct sb} + \# \text{correct nsb})}{\text{total \# of space tokens}}$$
$$\text{false break} = \frac{\# \text{sb false positives}}{\text{total \# of space tokens}}$$

It was generally possible to reconstruct precision and recall figures from these published results¹ and we present a comprehensive table of results. Reconstructed values are marked with a dagger and the optimal result in each category is marked in boldface.

	Mittrapiyanuruk et al.	Charoenpornasawat et al.	Our result
method	POS Trigram	Winnow	MaxEnt
#sb in reference	10528	1086 [†]	2133
#space tokens	33141	3801	7227
nsb-precision	90.27 [†]	91.48 [†]	93.18
nsb-recall	87.18 [†]	97.56[†]	94.41
sb-precision	74.35 [†]	92.69[†]	86.21
sb-recall	79.82	77.27	83.50
"space-correct"	85.26	89.13	91.19
"false-break"	8.75	1.74	3.94

Table 2. Evaluation of Thai Sentence Breakers against ORCHID

chine translation service. In this section, we provide a brief overview of this large-scale SMT system, focusing on Thai-specific integration issues.

4.1 Overview

Like many multilingual SMT systems, our system is based on hybrid generative/discriminative models. Given a sequence of foreign words, f , its best translation is the sequence of target words, e , that maximizes

$$e^* = \operatorname{argmax}_e p(e|f) = \operatorname{argmax}_e p(f|e)p(e)$$
$$= \operatorname{argmax}_e \{ \log p(f|e) + \log p(e) \}$$

where the translation model $p(f|e)$ is computed on dozens to hundreds of features. The target language model (LM), $p(e)$, is represented by a smoothed n-grams (Chen 1996) and sometimes more than one LM is adopted in practice. To achieve the best performance, the log likelihoods evaluated by these features/models are linearly combined. After $p(f|e)$ and $p(e)$ are trained, the combination weights λ_i are tuned on a held-out dataset to optimize an objective function, which we set to be the BLEU score (Papineni et al. 2002):

$$\{\lambda_i\} = \max_{\{\lambda_i\}} \text{BLEU}(\{e^*\}, \{r\})$$

$$e^* = \operatorname{argmax}_e \left\{ \sum_i \lambda_i \log p_i(f|e) + \sum_j \lambda_j \log p_j(e) \right\}$$

where $\{r\}$ is the set of gold translations for the given input source sentences. To learn λ_i we use the algorithm described by Och (2003), where the decoder output at any point is approximated

Evaluating Classifiers

- Classifiers divide items into different categories
- Or, they “label” items
- Or, they put them into different sets
- For each label/category/set, you can say:
 - There are items which are selected
 - There are items which are not selected
- The gold standard tells you which items should have been selected, i.e. “correct”

Precision and Recall

- Precision and Recall are **set-based** measures
- They evaluate the quality of some set membership, based on a reference set membership

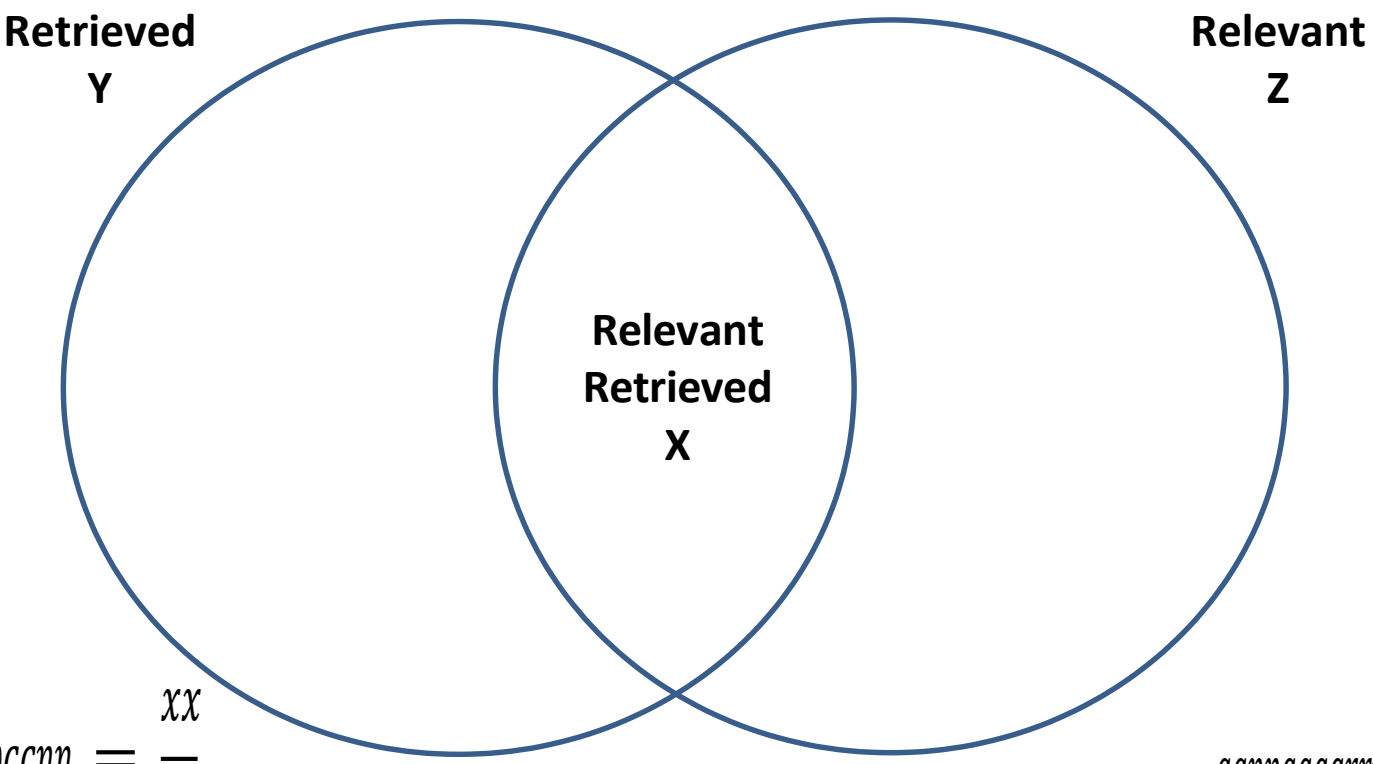
Precision and Recall

Precision: what proportion of the *selected* items are in the reference set?

Recall: how many *items from the reference set* got selected?

A less frequently used evaluation measure is **fallout**, (or collateral damage) the proportion of *items from the reference set* that were selected

Precision / Recall



$$ppaannaaprrppccnn = \frac{xx}{aa}$$

$$aannaaaarrrr = \frac{xx}{zz}$$

$$\Omega = aa \cup zz$$

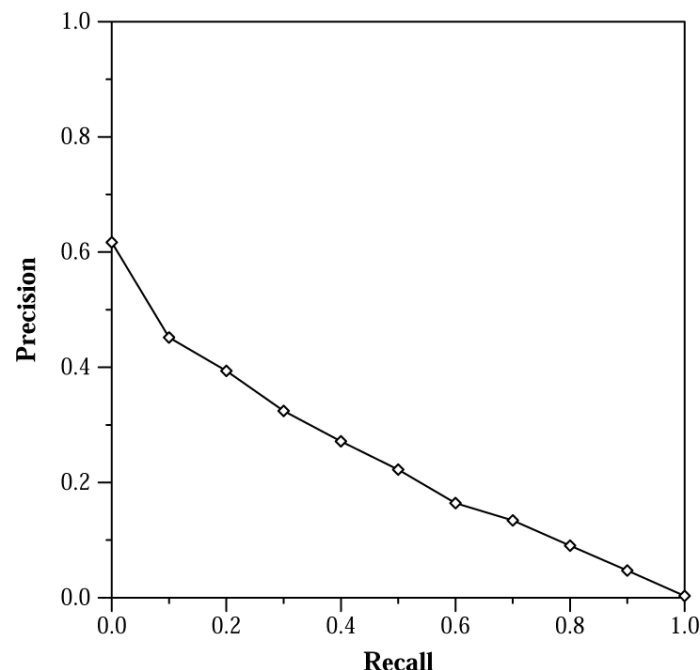
Precision-recall trade-off

- Usually, precision and recall can be traded for each other by changing your system parameters

The higher the proportion of correct items you require in the selected set (high precision), the fewer of the total correct items you will select (low recall)

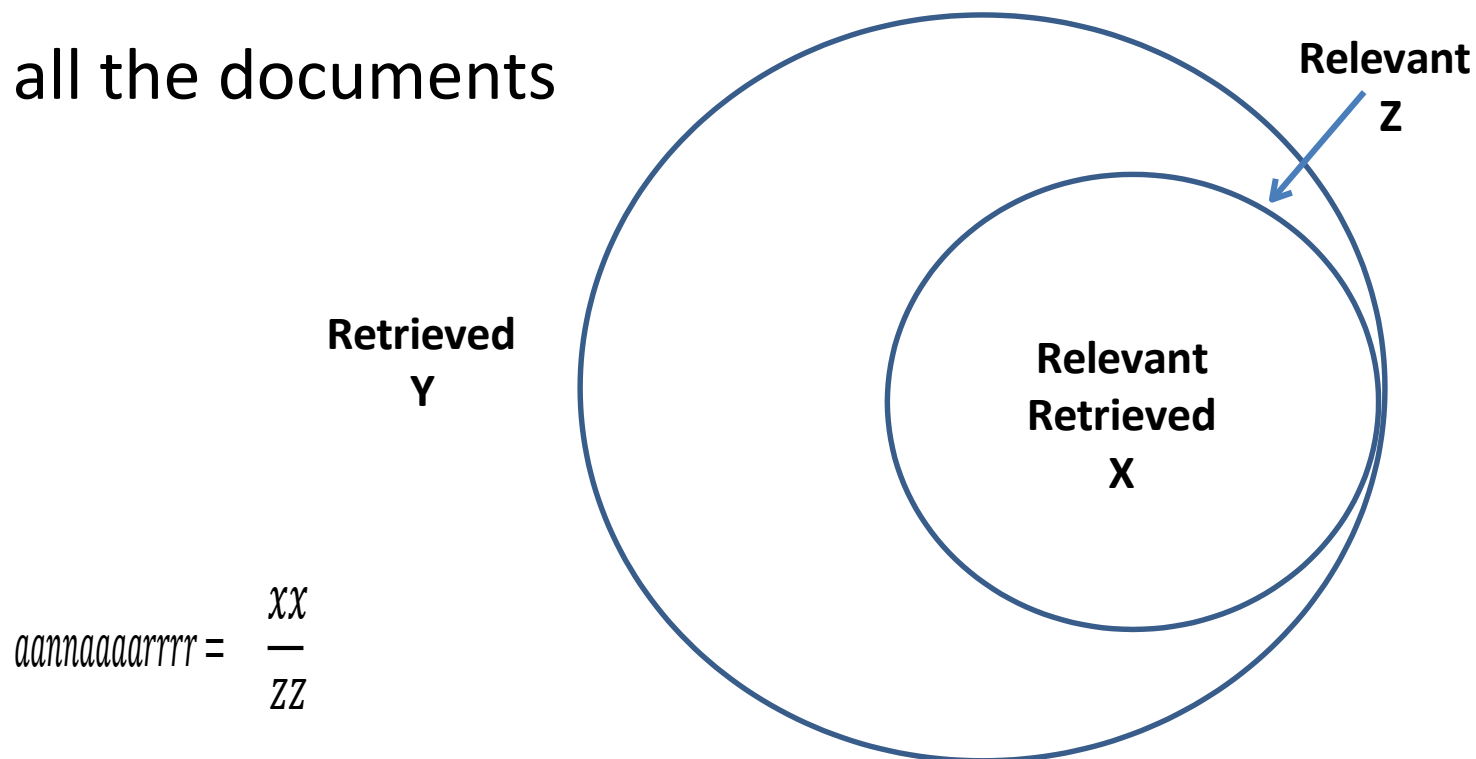
If you can tolerate a higher proportion of incorrect items in the selected set (low precision), you will capture more of the total correct items (high recall)

Recall-Precision Curve



Precision-recall trade-off

- It's easy to get recall of 1.0. Why?
- Return all the documents



Summary

		Gold standard	
		X	Y
Your Result	X	true positive tp	false positive fp type I error
	Y	false negative fn type II error	true negative tn

$$\text{aaaaaaaaaaaaaaaa} = \frac{ccpp + ccnn}{ccpp + ccnn + oopp + oonn}$$

$$\text{ppaannaapprrppccnn} = \frac{ccpp}{ccpp + oopp}$$

$$\text{nnaaaaccaa} = \frac{oopp + oonn}{ccpp + ccnn + oopp + oonn}$$

$$\text{aannaaaarrrr} = \frac{ccpp}{ccpp + oonn}$$

$$\text{ooaarrrrccaacc} = \frac{oopp}{oopp + ccnn}$$

Example

- Tokenizing task

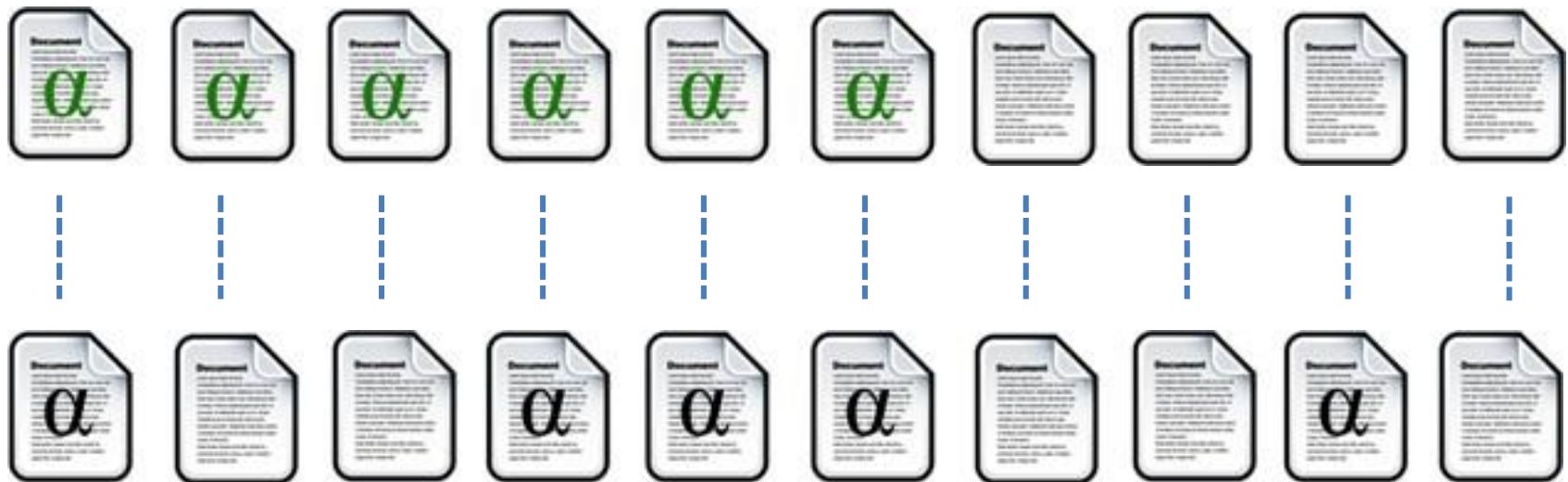
	Precision	Recall	F-Measure
Tokenizer 1	0.800	0.889	0.844
Tokenizer 2	0.962	0.926	0.944
Tokenizer 3	0.929	0.963	0.946
Tokenizer 4	1.000	1.000	1.000

Baseline	Tokenizer 1	Tokenizer 2	Tokenizer 3	Tokenizer 4
After	After	After	After	After
coming	coming	coming	coming	coming
close	close	close	close	close
to	to	to	to	to
After	After	After	After	After
partial	partial	partial	partial	partial
settlement	settlement	settlement	settlement	settlement
a	a	a	a	a
year	year	year	year	year
ago	ago	ago	ago	ago
,				,
shareholders	shareholders	shareholders	shareholders	shareholders
who	who	who	who	who
filed	filed	filed	filed	filed
civil	civil	civil	civil	civil
suits	suits	suits	suits	suits
against	against	against	against	against
Ivan	Ivan	Ivan	Ivan	Ivan
F.	F	F.	F.	F.
Boesky		Boesky	Boesky	Boesky
&	Boesky	&	&	&
Co.	&	Co.	Co	Co.
L.P.	Co	L.P.		L.P.
Drexel		Drexel's	L.P.	Drexel
's	L.P.	plaintiffs	Drexel	's
plaintiffs	's	'	's	plaintiffs
'	Drexel's		plaintiffs	'
	plaintiffs			
	'			

In-class quiz #1



This is the gold standard for biomedical documents which mention the IL-2R α -promoter. The result of our classifier is shown below.



What is the accuracy? .70

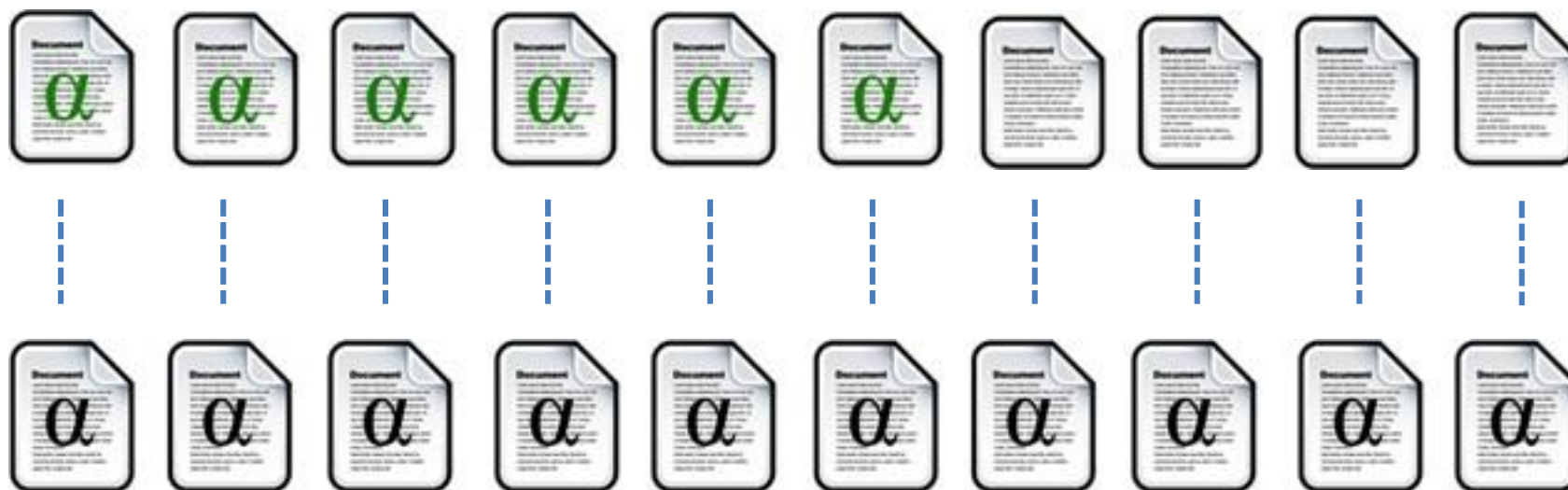
What is the precision? .80

What is the recall? .66

In-class quiz #2



This is the gold standard for biomedical documents which mention the IL-2R α -promoter. The result of our classifier is shown below.



What is the accuracy? .60

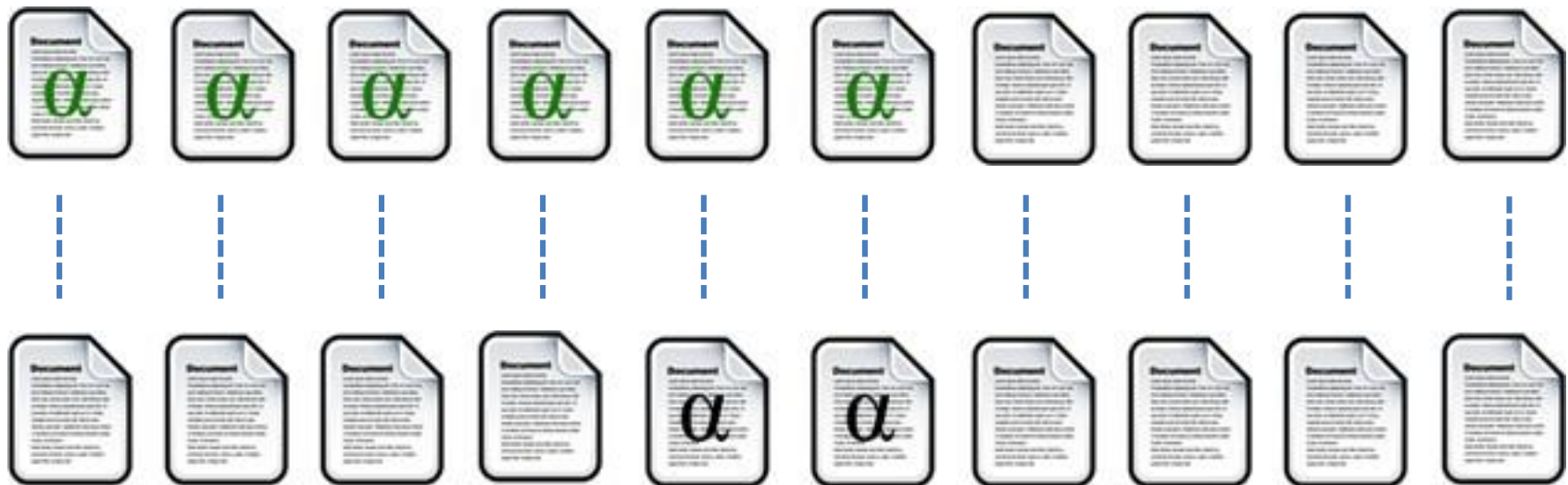
What is the precision? .60

What is the recall? 1.0

In-class quiz #3



This is the gold standard for biomedical documents which mention the IL-2R α -promoter. The result of our classifier is shown below.

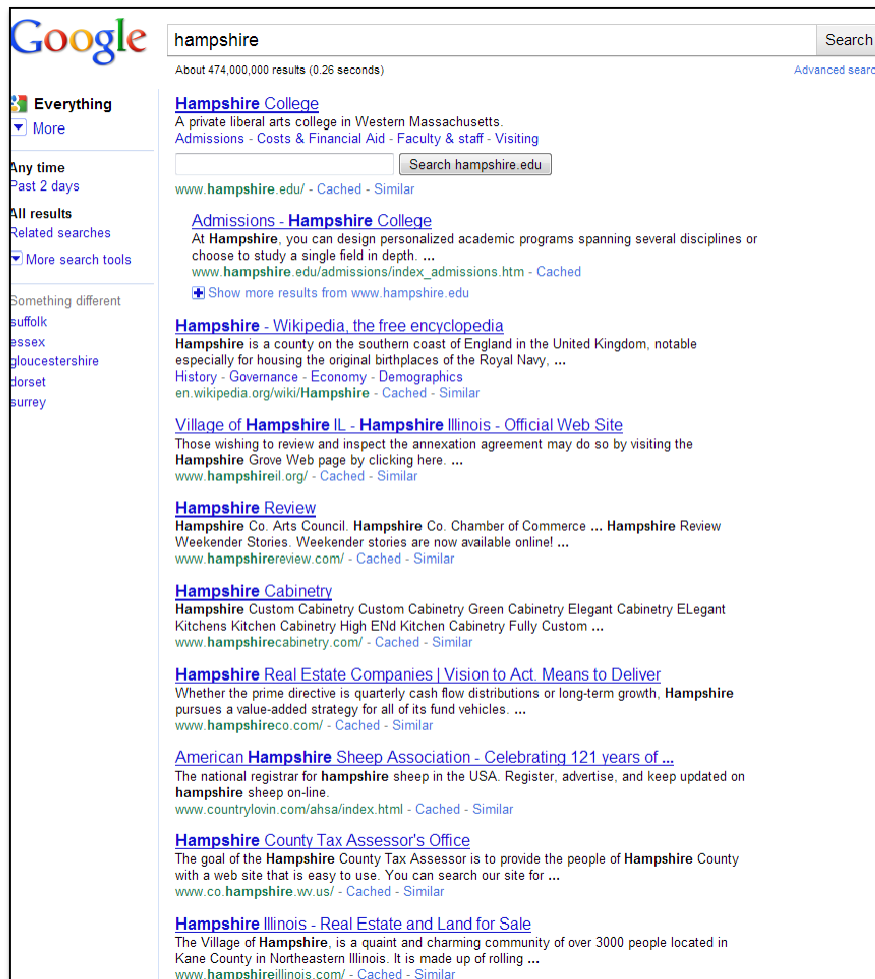


What is the accuracy? .60

What is the precision? 1.0

What is the recall? .33

Why do precision/recall matter



- Take the first page of results
- Precision: how many of these are relevant?
- Recall: how many of the relevant results are included?

Information Retrieval

- A retrieval engine must generally have high recall to be useful
- Better quality retrieval means increasing precision without sacrificing recall
- Good recall but poor precision means relevant hits will be lost in the noise of irrelevant results
- Precision/recall are set-based measures; they don't take result ranking into account

Chunking

- We just looked at P/R for information retrieval
- We talked about P/R for general classification
- P/R is appropriate for any set-oriented task
- “**Chunking**” is another shallow NLP task that can be evaluated with P/R
- Chunking: Assign some additional structure over POS tagging, without the expense of full parsing

Pla, F., Molina, A., and Prieto, N. 2000. Tagging and chunking with bigrams. In *Proceedings of the 18th Conference on Computational Linguistics - Volume 2* (Saarbrücken, Germany, July 31 - August 04, 2000). International Conference On Computational Linguistics. Association for Computational Linguistics, Morristown, NJ, 614-620.

- P/R Evaluation of chunking: See J&M Section 13.5.3

F-measure

- based on van Rijsbergen (1979):

$$F = \frac{2PR}{P + R}$$

- This is a simple average; P and R are each weighted by .5
- Other versions of F scores use different weights for P versus R

Planning for evaluation

- Design evaluation strategy at the start of your research
 - What will be measured?
 - What is the baseline?
 - What is the gold standard (reference)?
 - What is the measurement heuristic?

Measure/metric/indicator

- a measure:
 - a figure, extent, or amount obtained by measuring
- a metric (distance)
 - implies a 'cline'
 - the degree to which a system possesses a given attribute, or
 - a combination of two or more measures
- an indicator
 - the amount of deviation from a baseline state

Standardized evaluation

- When evaluating complex systems, it's helpful to use the same measure that was used to measure comparable (competitive) systems
- Machine translation:
 - BLEU, NIST, METEOR
- Document summarization:
 - ROUGE-1, ROUGE-2, ROUGE-SU4, Pyramid

Constrained Optimization

- Many NLP problems search a vast problem space
 - parsing, aggregated classification
- Argmin, Argmax
- “Hill climbing”
- Discrete case: Integer programming
 - Simplex method
- Continuous: Lagrange Multipliers

<http://courses.washington.edu/ling473/lagrange-constraint/>

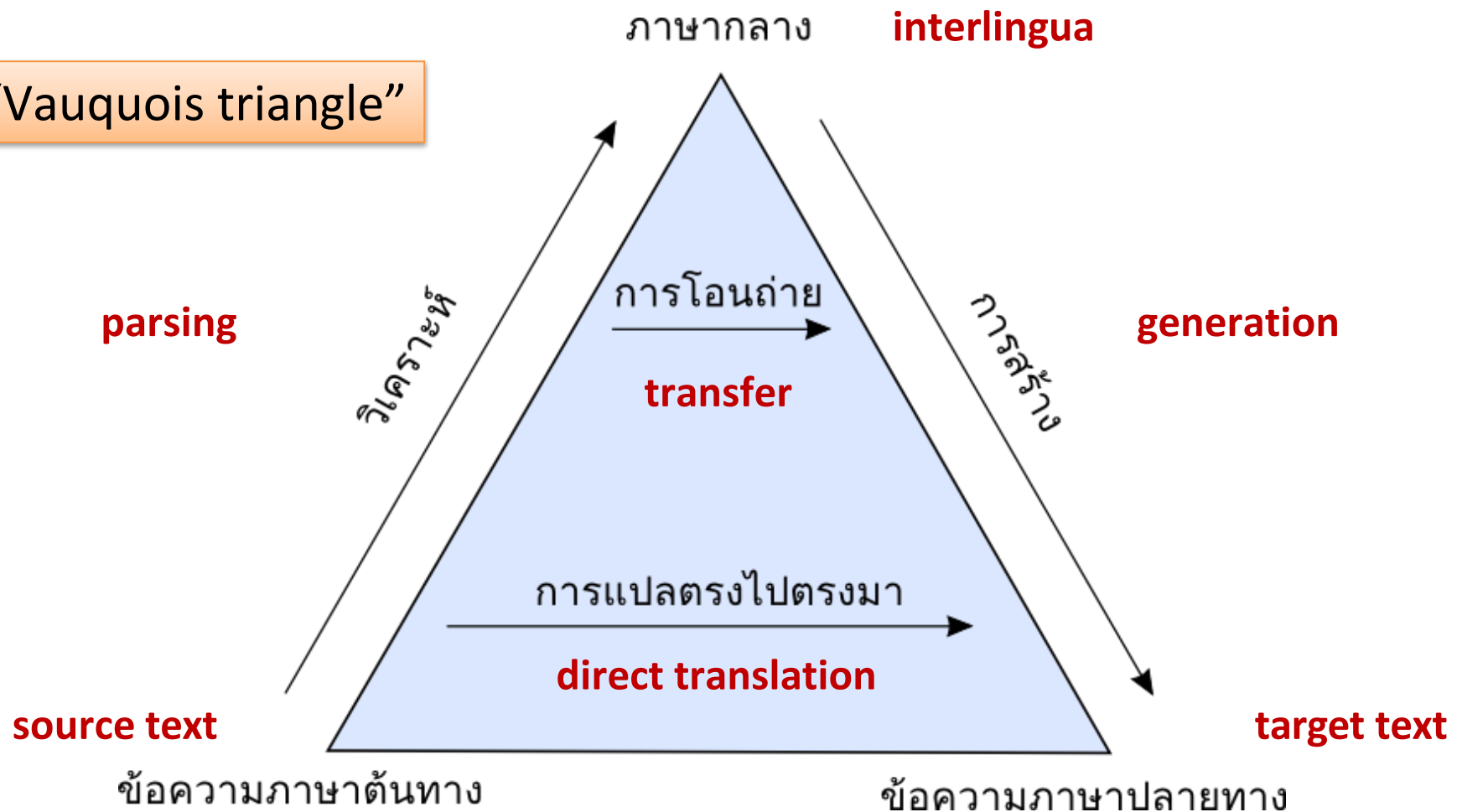
Feature structure parsing

Deep vs. shallow

- Analytical methods:
 - “deep,” rule-based
 - linguistically informed/motivated
 - traditionally rules have been hand-crafted
 - but see Poon and Domingos 2009
<http://www.cs.washington.edu/homes/pedrod/papers/emnlp09.pdf>
 - system development feedback: direct
- Statistical methods
 - “shallow,” automatically-extracted
 - development feedback from results: indirect

Semantic transfer machine translation

“Vauquois triangle”



HPSG

- Highly consistent and powerful formalism
- Monostratal, declarative, non-derivational, lexicalist, constraint-based
- Has been studied for many different languages
- Psycholinguistic evidence

HPSG foundations: Typed Feature Structures

- Typed Feature Structures (Carpenter 1992)
- High expressive power
- Parsing complexity: exponential (to the input length)
- Tractable with efficient parsing algorithms
- Efficiency can be improved with a well designed grammar

Feature Structures In Unification-Based Grammar Development

- A feature structure is a set of attribute-value pairs
 - Each attribute (or feature) is an atomic symbol
 - The value of each attribute can be either atomic, or complex (a feature structure, a list, or a set)

CATEGORY	<i>noun-phrase</i>				
AGREEMENT	<table><tr><td>PERSON</td><td><i>3rd</i></td></tr><tr><td>NUMBER</td><td><i>sing</i></td></tr></table>	PERSON	<i>3rd</i>	NUMBER	<i>sing</i>
PERSON	<i>3rd</i>				
NUMBER	<i>sing</i>				

Typed Feature Structure (TFS)

- A typed feature structure is composed of two parts a type
- A (possibly empty) set of attribute-value pairs with each value being a TFS

Properties of TFSes

- **Finiteness**
a typed feature structure has a finite number of nodes
- **Unique root and connectedness**
a typed feature structure has a unique root node; apart from the root, all nodes have at least one parent
- **No cycles**
no node has an arc that points back to the root node or to another node that intervenes between the node itself and the root
- **Unique features**
no node has two features with the same name and different values
- **Typing**
each node has single type which is defined in the hierarchy

Type hierarchy

- In the DELPH-IN joint reference formalism:
 - A unique most general type: *top* T
 - Each non-top type has one or more parent type(s)
 - Two types are compatible if they share at least one offspring type
 - Each non-top type is associated with optional constraints
 - Constraints specified in ancestor types are monotonically inherited
 - Constraints (either inherited, or newly introduced) must be compatible

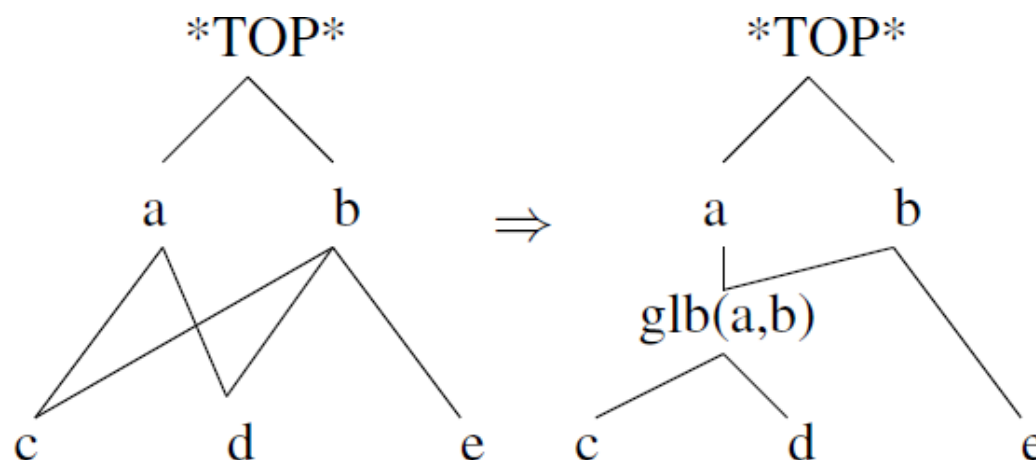
Unification

The unification result on two TFSes TFS_a and TFS_b is:

- \perp , if either one of the following:
 - type *aa* and *nn* are incompatible
 - unification of values for attribute *X* in TFS_a and TFS_b returns \perp
- a new TFS, with:
 - the most general shared subtype of *aa* and *nn*
 - a set of attribute-value pairs being the results of unifications on sub-TFSes of TFS_a and TFS_b

GLB Types

- In case of multiple inheritance, two types can have more than one shared subtype that neither is more general than the others
- Non-deterministic unification results
- Type hierarchy can be automatically modified to avoid this



Semantics desiderata

- For each sentence admitted by the grammar, we want to produce a meaning representation suitable for applying rules of inference.

“This fierce dog chased that angry cat.”

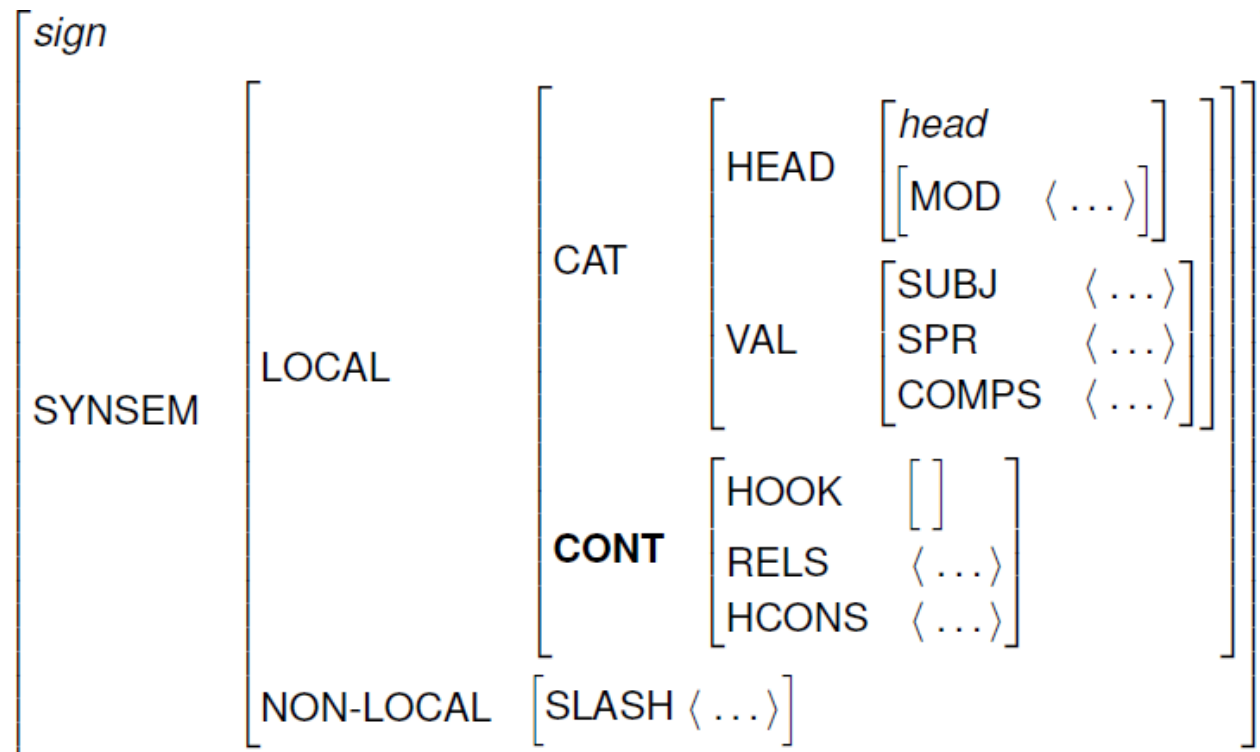
$\text{this}(xx) \wedge \text{fierce}(xx) \wedge \text{dog}(xx) \wedge$
 $\text{chased}(nn, xx, aa) \wedge$
 $\text{that}(aa) \wedge \text{angry}(aa) \wedge \text{cat}(aa)$

Semantics desiderata

- Compositionality
 - The meaning of a phrase is composed of the meanings of its parts.
- Existing machinery
 - Unification is the only mechanism we use for constructing semantics in the grammar.

Semantics in feature structures

- Semantic content in the CONT attribute of every word and phrase



Semantics formalism: MRS

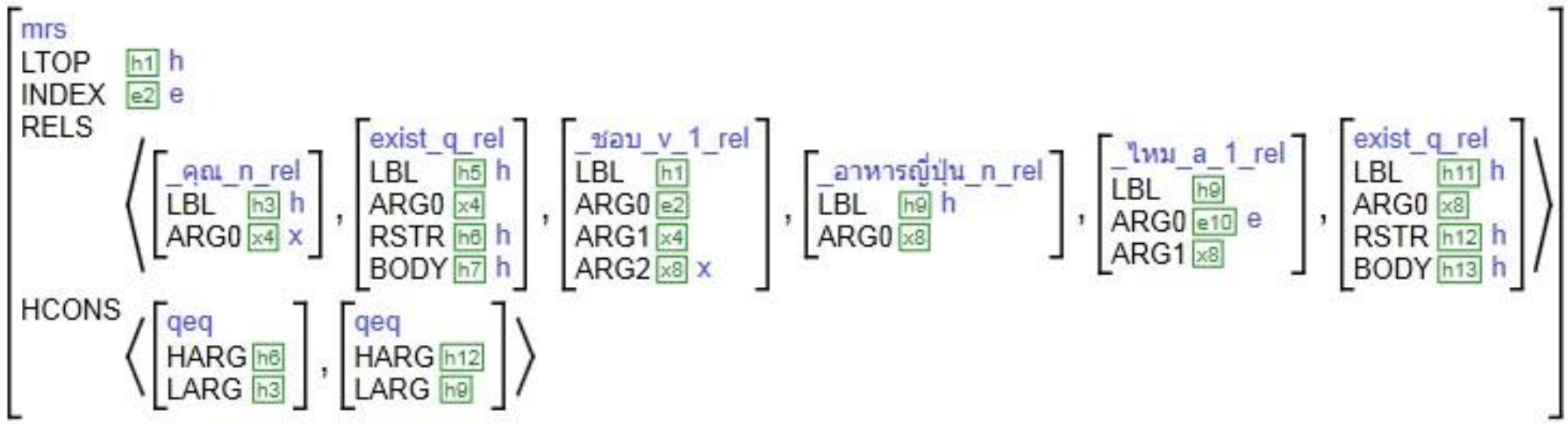
- Minimal Recursion Semantics

Copestake, A., Flickinger, D., Pollard, C. J., and Sag, I. A. (2005).
Minimal recursion semantics: an introduction. Research on Language
and Computation, 3(4):281–332.

- Used across DELPH-IN projects
- The value of CONT for a sentence is essentially a list of relations in the attribute RELS, with the arguments in those relations appropriately linked:
 - Semantic relations are introduced by lexical entries
 - Relations are appended when words are combined with other words or phrases.

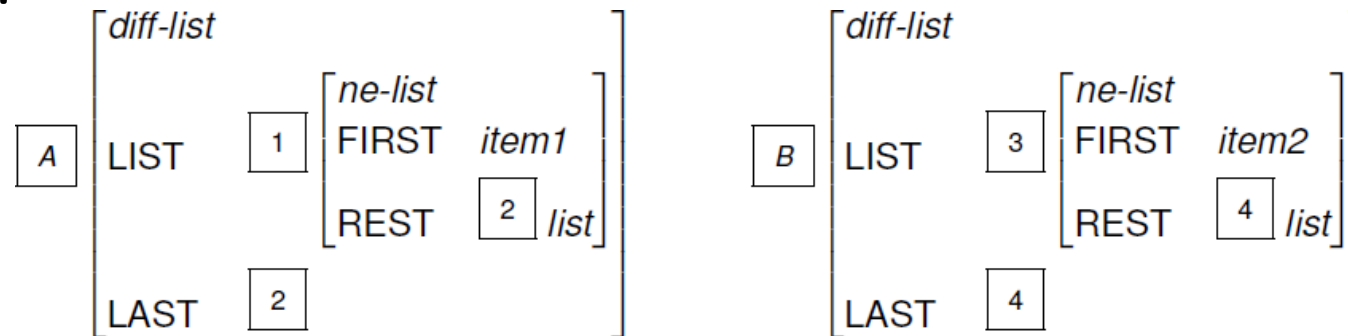
MRS: example

คุณชอบอาหารญี่ปุ่นไหม



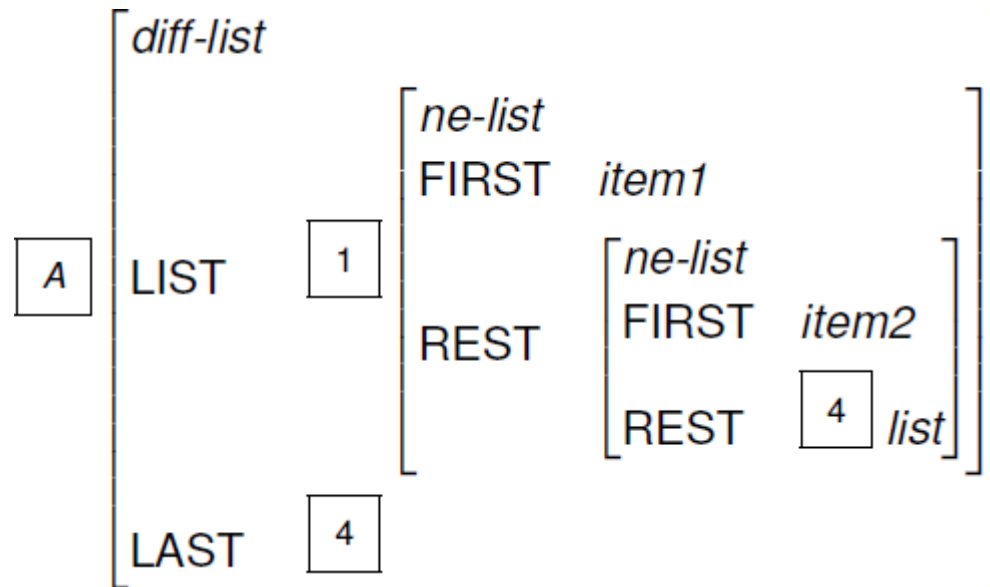
Building semantics with unification

- A *difference list* embeds an open-ended list into a container structure that provides a 'pointer' to the end of the ordinary list.



- Using the LAST pointer of difference list A we can append A and B by
 - unifying the front of B (i.e. the value of its LIST feature) into the tail of A (its LAST value) and
 - using the tail of difference list B as the new tail for the result of the concatenation.

Result of appending the lists



Linking semantic arguments

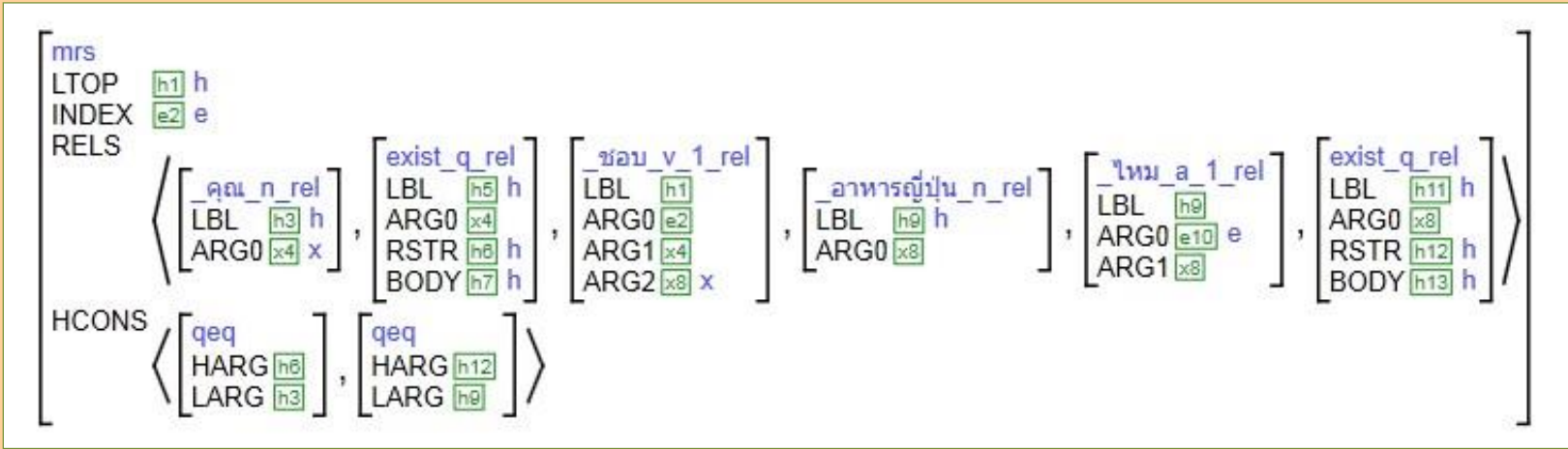
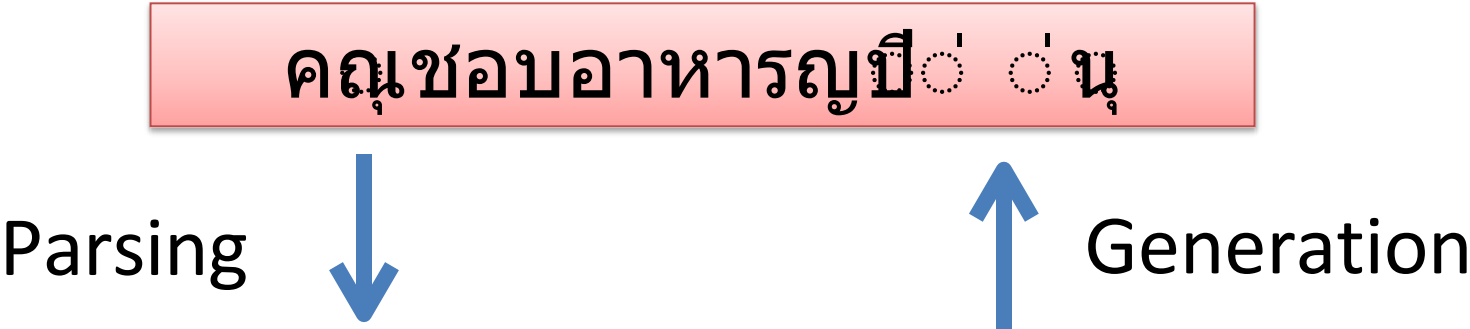
- Each word or phrase also ‘publishes’ an INDEX attribute in CONT.HOOK
- When heads select a complement or specifier, they constrain its INDEX value – an instance variable for nouns, an event variable for verbs.
- Each lexeme also specifies a KEY relation (to allow complex semantics)

Semantics of phrases

- Every phrase identifies its RELS attribute with the concatenation of its daughters' RELS lists
- Every phrase identifies its semantic INDEX value with the INDEX value of one of its daughters (the semantic head).
- Since we unify the *synsem* of a complement or specifier with the constraints in the head-daughter, unification also takes care of semantic linking.
- Head-modifier structures: the modifier constrains the semantic index of the modified head-daughter, and then rules unify the *synsem* of the head-daughter with the MOD value in the modifier.

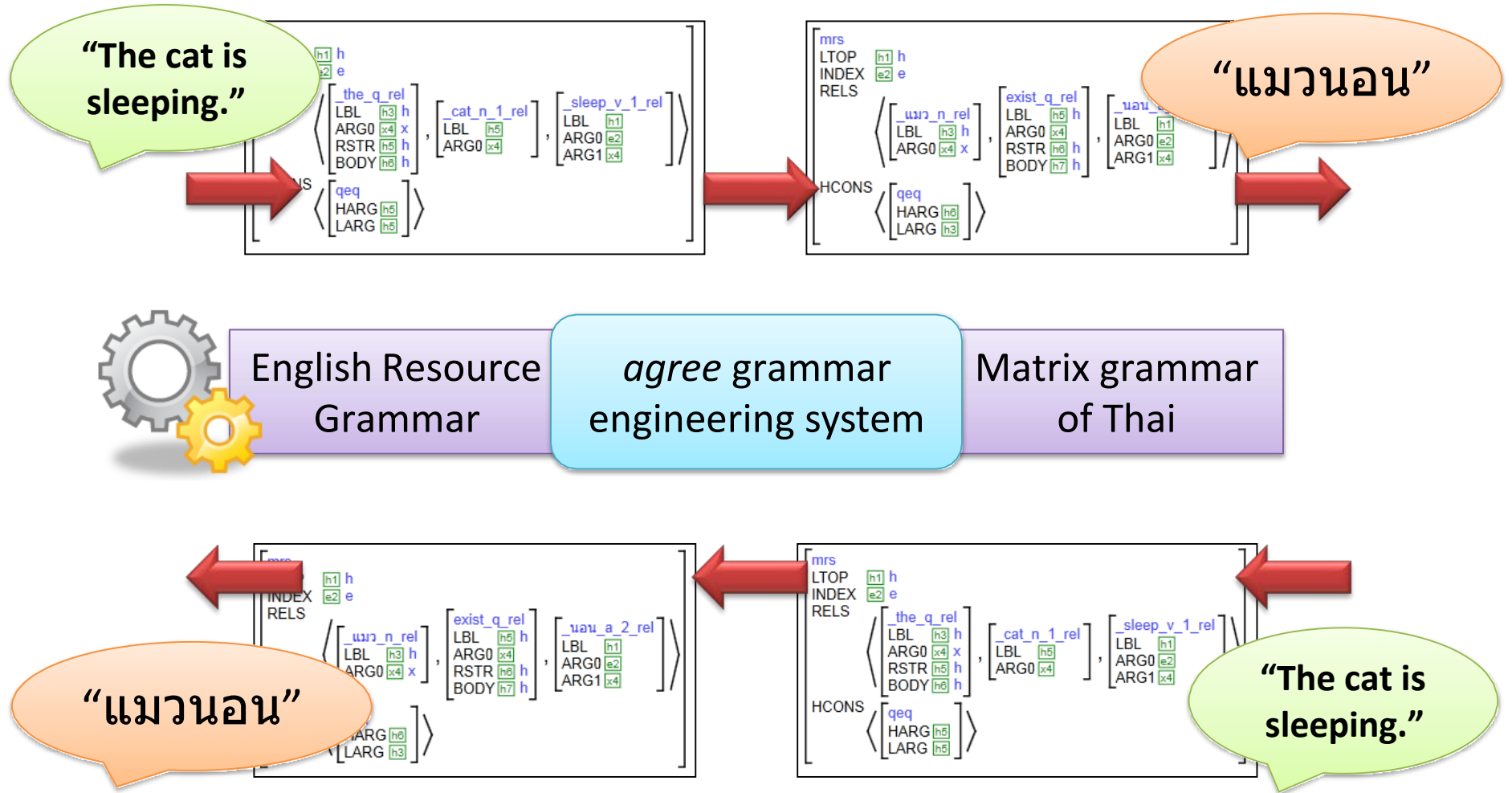
Parsing and Generation

- DELPH-IN computational grammars are bi-directional:



- These grammars are primarily concerned with mapping between syntax and semantics

Proposed “deep” Thai-English system



C# walkthrough for Self-study project

Declaring variables

- Some languages require you to declare the type of a variable before you use it

```
// C, C++, C#  
int v = 5;
```

- Others don't

```
# python  
v = 5
```

Strong (static) v. dynamic typing

- Programming languages can be strongly typed (C#, java) or dynamically typed (Python, Basic, Javascript, Perl)
- Static (strong) type enforcement allows more errors to be caught before running the program, because compilers and editing tools can flag inconsistent usages which are probably programming errors
- In reality, there is a spectrum of type strength. Polymorphism in strongly-typed languages is a controlled form of dynamic typing

Strong typing is often considered a productivity gain, because it uncovers conceptual errors earlier in the development process.

example

Python:

```
v = 5  
print v  
v = "hello world"  
print v
```

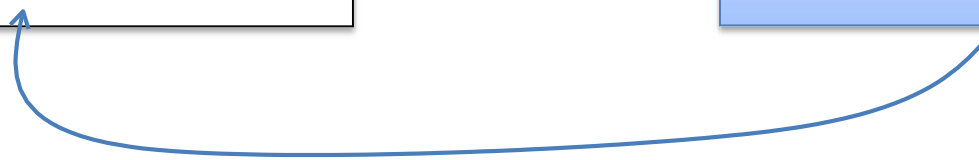


C#:

```
int v = 5;  
v = "hello world";
```



The editor has already
flagged this as an error, as
soon as you wrote it

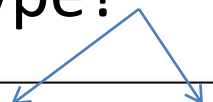


Flexible functions

- For Project 6, we would like a function that operates on a sequence of elements—which could be of any type
- This is no problem in a dynamically typed language, but in C#, would we have to commit to a type?

```
# python
def EditDistance(s,t):
    # ... etc ...
    return 0
```

```
double EditDistance(String s, String t)
{
    // ...
    return 0.0;
}
```



The easy solution is to repeat the entire function, once for each different type you anticipate. Problems:

- You may not anticipate future uses with other types
- The code is duplicated, which invites bugs

```
double EditDistance1(String s, String t)
{
    // ...
    return 0.0;
}

double EditDistance2(String[] s, String[] t)
{
    // ...
    return 0.0;
}
```

Programming with templates

- Fortunately, strongly-typed languages have features that allow for this
 - You can specify exactly which arguments of a function (or parts of an entire class) are type-flexible
 - In C#, you can apply special *constraints* on the allowable types, which *expands* the things you can do with the types in the function
 - The language and environment automatically create instances of the function (or object) upon demand, even for unforeseen types.

Lambda expressions

- A lambda expression is a portable, possibly anonymous (unnamed) snippet of code that you can store, refer to and carry and pass around just like any other data object
- It's an elegant way for callers to customize some aspect of a function's behavior
 - This is exactly what the EditDistance function requires:
 - A way to allow callers to arbitrarily **customize** the substitution cost function

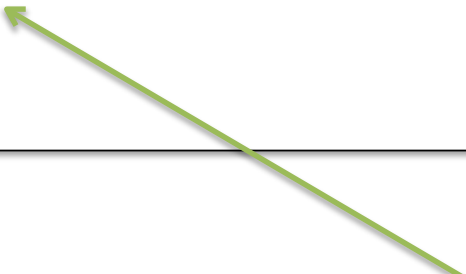
What's the 'type' of a lambda function?

- Like (most) objects in C#, a lambda function must be strongly typed
- This means defining the exact types expected for:
 - One or more input parameters
 - The return value (if any)
- Fortunately, the system libraries provide a template class, so you can specify the parameter types and return value type for any strongly-typed lambda function you need

Func<T1,T2,...,TReturn>

Use this system-defined template class to create lambda functions that have a return value

```
// MyAdd is a lambda function that adds two numbers  
Func<int, int, int> MyAdd = (a1, a2) => a1 + a2;  
// Call it:  
int sum = MyAdd(3, 5);
```



No adding happens here at this point; we're just declaring the function

Action<T1,T2,...>

Action<...> is for lambda functions that do not return a value

```
// This action has no arguments or return value
Action my_beep = () => Console.Beep();

// This one has an argument
Action<int> my_sleep = ms => Thread.Sleep(ms);

// (...later) call them:
my_beep();
my_sleep(400);
```

Two syntaxes

- There are two syntaxes for lambda functions in C#. If it's a short function, you can do it as shown on the previous slides:

```
Func<int, int, int> MyAdd = (a1, a2) => a1 + a2;
```

- If it's longer than one line, you might prefer to write it as shown below instead. If you use this curly brace syntax, you have to use the return keyword.

```
Func<double, double> MyLog = (n) =>
{
    if (n == 0.0)
        throw new InvalidOperationException();
    return Math.Log(n);
};
```


Interfaces

An **interface** is a named set of zero or more function signatures with no implementation(s)

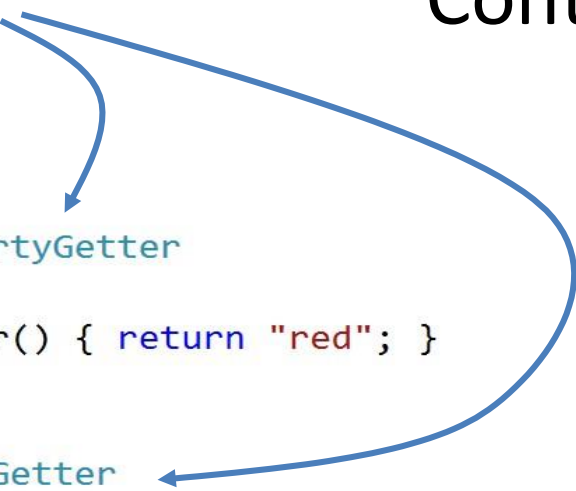
- To implement an interface, a class defines a matching implementation for every function in the interface
- Interfaces are sometimes described as contracts
- You can define and use a reference to an interface just like any other object reference

Contrived Example

```
interface IPropertyGetter
{
    String GetColor();
}

class Strawberry : IPropertyGetter
{
    public String GetColor() { return "red"; }
}

class Ferrari : IPropertyGetter
{
    public String GetColor() { return "yellow"; }
}
```



- This looks like C++ class inheritance
 - yes, but it's more ad-hoc
 - C# classes can have **single inheritance** of other classes, and **multiple inheritance** of interfaces
 - Interfaces can inherit from other interfaces (not shown)

Case Study

interfaces and templates:

`IEnumerable<T>`

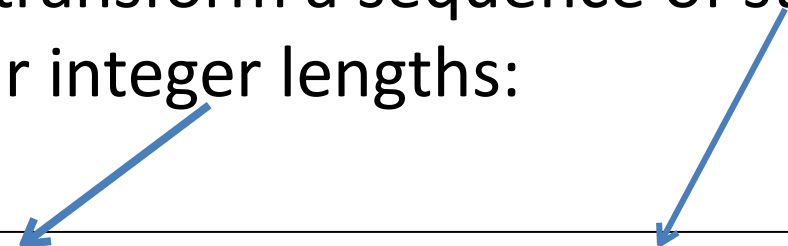
`IEnumerable<T>` is one of many system-defined
interfaces that a class can elect to implement

IEnumerable<T>

- This is one of the simplest interfaces defined in the BCL (base class libraries)
- This interface provides just one thing: a way to iterate over elements of type T
- All of the system arrays, collections, dictionaries, hash sets, etc. implement IEnumerable<T>
 - Implementing IEnumerable<T> on your own classes can be very useful, but you don't need to worry about that
 - For now, what's important is that you get to use it, because it's available on all of the system collections

IEnumerable<T>

- This is an interface (ad hoc grouping of functions) that allows for enumeration of (strongly-typed) objects of type T.
- So we can lazily transform a sequence of strings into a sequence of their integer lengths:



```
IEnumerable<int> MyFunc(IEnumerable<String> seq)
{
    foreach (String s in seq)
        yield return s.Length;
}
```

IEnumerator<T>

- **IEnumerable<T>** has only one function, which allows a caller or caller(s) to obtain an *enumerator* object which is able to iterate over elements
 - The actual enumerator object is an object that implements a different interface, called **IEnumerator<T>**
 - This “factory” design allows a caller to initiate and maintain several simultaneous iterations if needed
 - The enumerator object, **IEnumerator<T>** can only:
 - Get the current element
 - Move to the next element
 - Tell you if you’ve reached the end
 - Note: There’s no count
 - ICollection inherits from IEnumerable to provide this

IEnumerable, yield, and deferred execution

- Before describing the trie data structure, let's look at iterators which enumerate a sequence of elements

Examples in C#. If you use another language, it will be instructive to think about how to adapt the solutions to your language

- Enumeration is obvious when the data is at hand and you want to use it all:

```
String[] data = { "able", "bodied", "cows", "don't", "eat", "fish" };  
  
foreach (String s in data)  
    Console.WriteLine(s);
```

We can pass (a reference to) the array around too, no problem

```
String[] data = { "able", "bodied", "cows", "don't", "eat", "fish" };  
// ...  
ProcessSomeStrings(data);  
// ...  
  
void ProcessSomeStrings(String[] the_strings)  
{  
    foreach (String s in the_strings)  
        Console.WriteLine(s);  
}
```


What if we only want to “process” the four-letter words?

```
String[] data = { "able", "bodied", "cows", "don't", "eat", "fish" };  
// ...
```

```
List<String> filtered = new List<String>();  
foreach (String s in data)  
    if (s.Length == 4)  
        filtered.Add(s);  
ProcessSomeStrings(filtered);  
// ...
```

This doesn't seem very nice. For one thing, we have to use more memory and waste time copying the elements we care about to a new list.

```
void ProcessSomeStrings(List<String> the_strings)  
{  
    foreach (String s in the_strings)  
        Console.WriteLine(s);  
}
```

Is there a way to pass this function enough information to filter the *original list* itself, where it lies?

- Remember the non-filtered example for a second

```
void ProcessSomeStrings(String[] the_strings)
{
    foreach (String s in the_strings)
        Console.WriteLine(s);
}
```

- The processing function doesn't really care about the fact that the data is in an array
- This violates an important programming maxim:

A flexible interface *demands the least* and *provides the most*:

- Inputs* are as general as possible (allowing clients to supply any level of specificity, i.e. be lazy)
- Outputs* are as specific as possible (allowing clients to capitalize on work products, i.e. be lazy).

```
void ProcessSomeStrings(String[] the_strings)
{
    foreach (String s in the_strings)
        Console.WriteLine(s);
}
```

The extra (unused) demands this function is making by asking for String[]:

- That the strings all be in memory at the same time
 - That the strings be randomly accessible by an index
 - That the number of strings be known and fixed before the function starts
-
- To modify this to comply with the maxim, we first ask:
 - Q: What is the absolute minimum that this function actually needs to accomplish it's work?
 - Answer: a way to iterate strings

Interfaces as function arguments

- Using interfaces as function arguments allows you to require the absolute minimum functionality the function actually needs
- In this way, the ad-hoc nature of interfaces allows us to comply with the maxim

```
void ProcessSomeStrings(IEnumerable<String> the_strings)
{
    foreach (String s in the_strings)
        Console.WriteLine(s);
}
```

Now, this function is exposing the **weakest (most general) requirement** possible for the processing it has to do. This provides more flexibility to callers since they can choose whatever level of specificity is convenient. The function can be used in the widest possible variety of situations.

Example

```
String[] d1 = { "able", "bodied", "cows", "don't", "eat", "fish" };  
ProcessSomeStrings(d1);
```


```
List<String> d2 = new List<String> { "clifford", "the", "big", "red", "dog" };  
ProcessSomeStrings(d2);
```

```
HashSet<String> d3 = new HashSet<String> { "these", "must", "be", "distinct" };  
ProcessSomeStrings(d3);
```

```
Dictionary<String,int> d4 =  
    new Dictionary<String, int> { { "the", 334596 }, { "in", 153024 } };  
ProcessSomeStrings(d4.Keys);
```

```
void ProcessSomeStrings(IEnumerable<String> the_strings)  
{  
    foreach (String s in the_strings)  
        Console.WriteLine(s);  
}
```

Python users might not be impressed, but the difference is that this is all 100% strongly typed



Iteration is efficient

- That's cool, `IEnumerable<T>` lets a function **not care** about where a sequence of elements is coming from
 - We don't copy the elements around
 - Iterators let us access elements right from their source
- All of those examples iterate over elements that **already exist** somewhere
- Is there a way to iterate over data that's generated on-the-fly, doesn't exist yet, or is never persisted at all?
- Yes!

Iterating over on-the-fly data

```
IEnumerable<String> GetNewsStories(int desired_count)
{
    for (int i = 0; i < desired_count; i++)
        yield return RealtimeNewswireSource.GetLatestStory();
}
```

see next slide

```
// ...
IEnumerable<String> d5 = GetNewsStories(7);
ProcessSomeStrings(d5);
// ...
```

This is exactly the same as before, but this time there's no "collection" of elements sitting anywhere

```
void ProcessSomeStrings(IEnumerable<String> the_strings)
{
    foreach (String s in the_strings)
        Console.WriteLine(s);
}
```

This function doesn't care. In fact, it can't even tell.

yield keyword

- The **yield** keyword makes it easy to define your own custom iterator functions
- Any function that contains the `yield` keyword becomes special
 - It must be declared as returning an `IEnumerable<T>`
 - Deferred execution means that the function's body is not necessarily invoked when you "call" it
 - It must deliver zero or more elements of type `T` using:
`yield return t;`
 - Sometime later, control may continue immediately after this statement to allow you to yield additional elements
 - It may signal the end of the sequence by using:
`yield break;`

Custom iterator function example

```
IEnumerable<String> GetNewsStories(int desired_count)
{
    for (int i = 0; i < desired_count; i++)
        yield return RealtimeNewswireSource.GetLatestStory();
}
```

code from this custom iterator function is *not* executed at this point.

```
// ...
IEnumerable<String> d5 = GetNewsStories(7);
ProcessSomeStrings(d5);
// ...
```

d5 refers to an iterator that “knows how” to get a certain sequence of strings when asked

```
void ProcessSomeStrings(IEnumerable<String> the_strings)
{
    foreach (String s in the_strings)
        Console.WriteLine(s);
}
```

This finally demands the strings, causing our custom iterator function to execute—interleaved with this loop!

...end of the case study

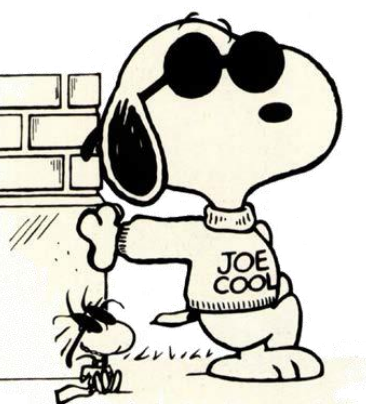
now back to Self study project...

Function templates

- You can use templates to allow your strongly-typed functions to be flexible.

```
double EditDistance2(String[] s, String[] t)
{
    // ...
    return 0.0;
}
```

```
double EditDistance1(String s, String t)
{
    // ...
    return 0.0;
}
```




```
double EditDistance<T>(T s, T t)
{
    // ...
    return 0.0;
}
```

T can be any
identifier name. It's
convention to use
upper case letters

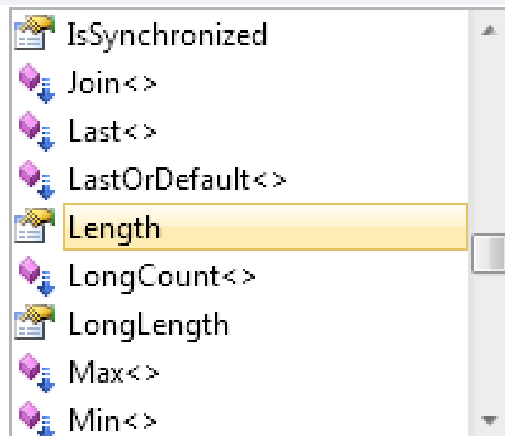
Useful?

- Oops, it's going to be hard to use **s** and **t** for anything in your function body, though because:
 - the compiler can't infer much about it, except that...
 - ...it inherits from type "object"
 - The language is still strongly typed, so inside your function you won't be allowed to do anything that is more specific than what you can do with "object"
 - Actually, you could cast them at runtime to some specific type, but this means you've lost your type safety and you're vulnerable to runtime errors (like a dynamically typed language)

We know that our function always deals with sequences. Let's declare that.



```
double EditDistance<T>(T[] s, T[] t)
{
    int src_length = s.Length;
    int tgt_length = t.
    return 0.0;
}
```



IsSynchronized
Join<>
Last<>
LastOrDefault<>
Length
LongCount<>
LongLength
Max<>
Min<>

int Array.Length
Gets a 32-bit integer that represents the

Wow, we told it that *rr* and *cc* are arrays of elements of type *T*, and strong typing is back!

The editor knows that an array always has a length property.

A note for C++ users

- Templates are a first-class feature of the mono/CLR runtime *environment*, not the C# language per se.
- They are not fully resolved at compile-time like C++ templates
- In certain cases, a new version of your template function or class can be generated by the runtime environment, specialized for a type that may not have even existed when you wrote and compiled your program
 - This happens without needing the source code to your program, or re-compiling from the source code

Calling the template function

- Now we can call the function with an array of any type. The compiler figures out what type T is automatically

```
double d_norm;

// call edit distance on arrays of strings
String[] t1 = { "my", "friend", "al" };
String[] t2 = { "myopia", "fries", "alfredo" };
d_norm = EditDistance(t1, t2);

// call edit distance on arrays of characters
String s = "abc";
String t = "cde";
d_norm = EditDistance(s.ToCharArray(), t.ToCharArray());
```

Template amok?

- As it currently stands, we can also call our function with an array of any other type of element(s)

```
class MyClass { };

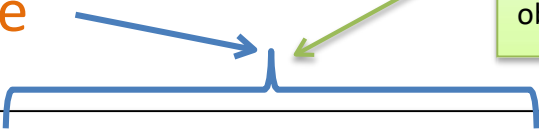
static void foo()
{
    // call edit distance on arrays of MyClass
    MyClass[] mc1 = { };
    MyClass[] mc2 = { };
    double d_norm = EditDistance(mc1, mc2);
}
```

- We may or may not want this. Rather than forbid specific types, we can declare the minimum set of constraints that EditDistance(...) actually needs in order to operate.

Template constraints

- You can restrict T in various ways to allow you to do more with objects of type T in the template function
- For the EditDistance function, it is useful to require that objects of type T be **equatable**

“T must be a type that implements the function `Equals<T>(T t)` which tests the equality of two objects of type T.”



```
double EditDistance<T>(T[] s, T[] t) where T : IEquatable<T>
{
    if (s[0].Equals(t[0]))
        s[0].
    return 0.
}
```

Equals
bool IEquatable<T>.Equals(T other) (+ 1 overload(s))
Indicates whether the current object is equal to another object of the same type.

GetHashCode
GetType
ToString

- After adding the constraint, the editor (and compiler) immediately know(s) that elements of *rr* and *cc* can be tested for equality

Can't you just use == ?

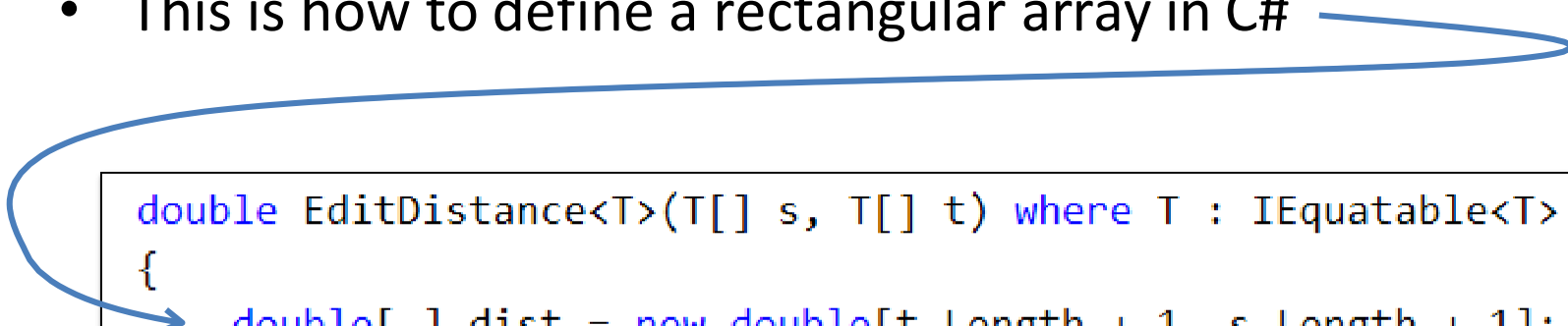
- On the previous slide, why couldn't we just have compared objects of type `T` using `'=='` ?
- Because C# supports user-defined **value types**, which do not automatically implement the `==` operator
 - That's because value types usually prefer to provide **bitwise comparison** semantics, as opposed to **reference equality**
- Since we didn't constrain our template function to *exclude* value types (by saying where `T : class`), we can't use that operator

Some details

- By default, *reference types* support *reference equality* via the `==` operator.
- However, `String` (for example) is one *reference type* which provides *value comparison semantics* instead.
- This is what you'd want and expect: `"Felicia" == "Felicia"` should be true even if the strings happen to be allocated in different places.

Jagged v. rectangular arrays

- Many languages support jagged versus rectangular arrays
- For project 6, you should use a rectangular array, if available
- This is how to define a rectangular array in C#




```
double EditDistance<T>(T[] s, T[] t) where T : IEquatable<T>
{
    double[,] dist = new double[t.Length + 1, s.Length + 1];
    // ...
    return 0.0;
}
```

Implementing the adjustable substitution cost function

- Lastly, the EditDistance function needs to use a different substitution cost function depending on whether you're doing the outer calculation (between the two texts) or inner calculation (between two lines of text)
- If you've created duplicate versions of the function (not using a template), then you can just hard-code the appropriate cost function

However, if you liked the template idea so far and now you have a nice, type-agile function, I'm sure you wouldn't want to ruin it like this:

```
double EditDistance<T>(T[] s, T[] t) where T : IEquatable<T>
{
    int i = 0, j = 0;
    double t_sub = 0.0;
    // ...
    if (s is String[])
        t_sub += EditDistance((s[j] as String).ToCharArray(),
                               (t[i] as String).ToCharArray());
    else
        t_sub += 2.0;
    // ...
    return 0.0;
}
```



Instead, you'd like to do something like this:

```
double EditDistance<T>(T[] s, T[] t) where T : IEquatable<T>
{
    int i = 0, j = 0;
    double t_sub = 0.0;
    // ...
    t_sub += subst_cost_func(s[j], t[i]);
    // ...
    return 0.0;
}
```

Hmm, how do we declare this function in terms of T, though?

This is a great place to use a lambda function

Self-study project

- Here are the two different substitution cost functions that we'd like to “pass in” to our EditDistance function

This one is for comparing strings, by character

```
Func<Char, Char, double> func1 = (s, t) => 2.0;
```

This one is for comparing entire texts, by line

```
Func<String, String, double> func2 = (s, t) =>
{
    return 0.5 + EditDistance(s.ToCharArray(),
                              t.ToCharArray());
};
```

Putting it all together

Now you're ready to add another parameter to the EditDistance function: the substitution cost function—a lambda function—that the caller will pass into the function, in order to customize its behavior

```
double EditDistance<T>(T[] s, T[] t,  
    Func<T, T, double> subst_cost_func) where T : IEquatable<T>  
{  
    int i = 0, j = 0;  
    double t_sub = 0.0;  
    //...  
    t_sub += subst_cost_func(s[j], t[i]);  
    //...  
    return 0.0;  
}
```



The grand finale

Now it all pays off: here's how to nest the calls to your type-agile template function, passing in the two different lambda functions, and getting the final result!

```
String[] text1, text2;  
// ...  
double d_norm = EditDistance(text1, text2, (s1, s2) =>  
    {  
        return 0.5 + EditDistance(  
            s1.ToCharArray(),  
            s2.ToCharArray(),  
            (c1, c2) => 2.0);  
    });
```

The compiler is being really smart here for you, inferring the types for the arguments of the lambda function based on the *element type* of whatever *array type* is passed in for the first arguments. This saves you from having to explicitly specify types when you use a template.