Lecture 15

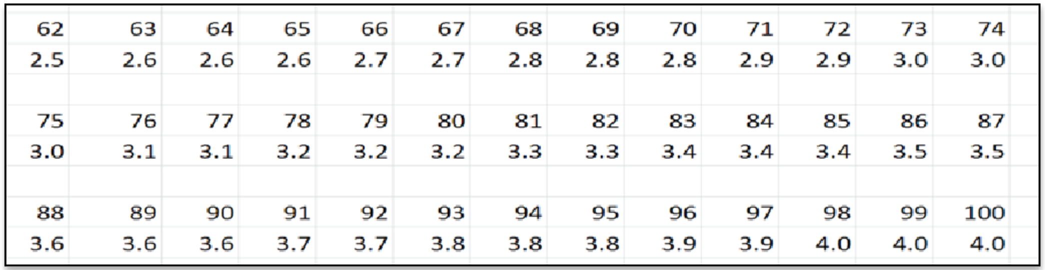
September 8, 2016

Review

● Reminder: start the recording

Announcements

- Writing assignment
  - Everyone did a great job, I am enjoying reading your papers
- Final grade conversion
  - class participation 5%
  - projects 65%
  - assignments 30%
  - % / 25, rounded up:



62	63	64	65	66	67	68	69	70	71	72	73	74
2.5	2.6	2.6	2.6	2.7	2.7	2.8	2.8	2.8	2.9	2.9	3.0	3.0
75	76	77	78	79	80	81	82	83	84	85	86	87
3.0	3.1	3.1	3.2	3.2	3.2	3.3	3.3	3.4	3.4	3.4	3.5	3.5
88	89	90	91	92	93	94	95	96	97	98	99	100
3.6	3.6	3.6	3.7	3.7	3.8	3.8	3.8	3.9	3.9	4.0	4.0	4.0

Ling 473
Review and Summary

Congratulations, you made it!

Linguistic Motivations

- Nearly all research areas in linguistics can benefit from computational techniques:
  - Phonetics
  - Phonology
  - Morphology
  - Syntax
  - Semantics
  - Pragmatics
  - Discourse Analysis
  - Information Structure
  - Language Typology

Breaking things down: Constituents

- Noun phrases (NPs)

(DET NN)	*The ostrich*
(NNP)	*Kim*
(NN NN)	*container ship*
(DET JJ NN)	*A purple lawnmower*
(DET JJ NN)	*That darn cat*

- Verb phrases (VPs)

(VB)	tango
(VBD NP NP)	gave the dog a bone
(VBD NP PP)	gave a bone to the dog

Syntax

The set of rules governing permissible constructions in a language.

- Syntax constrains the ways in which words may be combined to form constituents and sentences.
- Syntax forms one part of the description, or *grammar*, of a language.

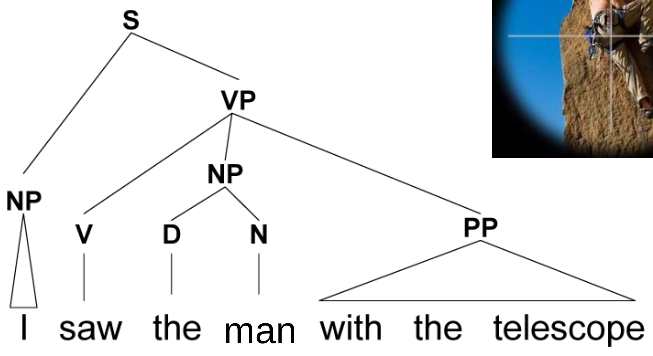
Language can be ambiguous

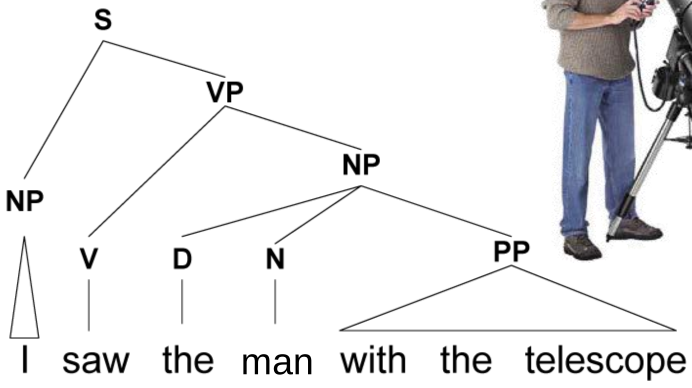
?

?

Syntactic Constituents

Constituents help us characterize and talk about some of the ambiguities in language





Trees: Dan Jinguji

Corpus linguistics

- The study of language as expressed in samples (corpora) or “real world” text
- A variant of rule-based NLP where the form, substance, and quantity of “rules” are generated automatically according to the maximization of an empirical objective function

- i.e. Penn TreeBank

Sets and Counting

- When you have a corpus, you have a large number of observations
- Different ways of counting them becomes important
- *the main thing:* Probability always ultimately comes down to counting

Tallies

```
"a man gave sandy a pomelo"

(a, 2)
(man, 1)
(gave, 1)
(sandy, 1)
(pomelo, 1)
```

Combinatorics

{ a b c }

- Permutation: how many different orderings?

  (a b c) (a c b) (b a c) (b c a) (c a b) (c b a) $n!$

- Combination: how many different subsets (i.e. of 2)?

  { a b } { a c } { b c } $\binom{n}{k}$

  allowing repetition in the output

  { a a } { a b } { a c } { b b } { b c } { c c } multiset coefficient $\binom{n+k-1}{k}$

- Variations: how many different ordered subsets (i.e. of 2)?

  (a b) (a c) (b a) (b c) (c a) (c b) $\frac{n!}{(n-k)!}$

  allowing repetition in the output

  (a a) (a b) (a c) (b a) (b b) (b c) (c a) (c b) (c c) n^k

Linux and Cluster Computing

- unix
  - command line “shell”
  - pipes, redirection
  - console input and output
  - this is adequate for text and corpus analysis
- condor cluster
  - don’t overload the head nodes when you have long-running jobs

Basic Regular Expressions

^	matches the start of a line
$	matches the end of a line
.	matches any one character (except newline)
[*xyz*]	matches any one character from the set
[^*pdq*]	matches any one character not in the set
|	accepts either its left or its right side
\	escape to specify special characters

anything else: must match exactly

Events

- Event: a composition of outcomes
- Independent vs. Mutually exclusive

Probability

- How likely is the observation of some event?

- Joint probability: both events happen in the same trial

Definition of Probability

- Let P be a function that satisfies the following:

  $P(\Omega) = 1$

  all possible outcomes are accounted for

  $\forall A \subseteq \Omega : P(A) \geq 0$

  probabilities are non-negative real numbers

  $\forall \{ A, B \} \subseteq \Omega, A \cap B = \emptyset : P(A \cup B) = P(A) + P(B)$

  for any pair of events that are mutually exclusive, the union of their probabilities is the sum of their probabilities

Conditional Probability

$$PP(AA|BB) = \frac{PP(AA \cap BB)}{PP(BB)}$$

$$PP(AA \cap BB) = PP(AA|BB)PP(BB)$$

$$PP(AA \cap BB) = PP(AA|BB)PP(BB)$$

joint probability = conditional probability × marginal probability
(or "prior" probability)

Facts about probability

- $P(A^C) = 1 - P(A)$
- $P(A \cup B) = P(A) + P(B) - P(A \cap B)$
- If $P(A \cap B) = P(A)\, P(B)$, then A and B are called independent events
- Otherwise

$$P(A \cap B) = P(A|B)P(B)$$

Random Variables

- In order to:
  - generalize about events;
  - allow for the variability of stochastic trials; and
  - map outcomes to empirical measurement values

  …we use random variables

A random variable is a function that maps a probability space Ω to the set of real numbers ℝ

$$X : \Omega \to$$

- Discrete vs. Continuous random variables

Bayes Theorem

Rev. Thomas Bayes (1701-1761)

Conditional probability of B given A "likelihood"

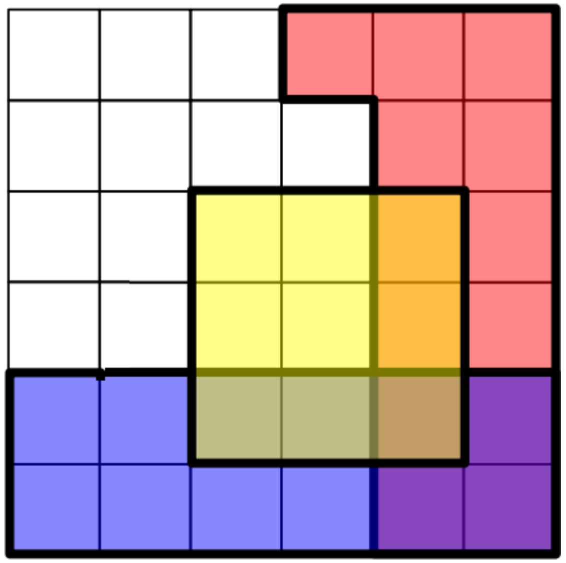
marginal or prior probability of A

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Conditional probability of A given B "posterior" probability

marginal or prior probability of B

Conditional independence



$$P(R) = \frac{13}{36}$$

$$P(B) = \frac{12}{36} = \frac{1}{3}$$

$$P(R \cap B) = \frac{4}{36} \neq P(R)P(B) = \frac{13}{108}$$

R and B are dependent

$$P(R|Y) = \frac{3}{9} = \frac{1}{3}$$

$$P(B|Y) = \frac{3}{9} = \frac{1}{3}$$

$$P(R \cap B|Y) = \frac{1}{9}$$

R and B can be *conditionally* independent given Y, even if they are dependent in the absence of information about Y

Probability distributions

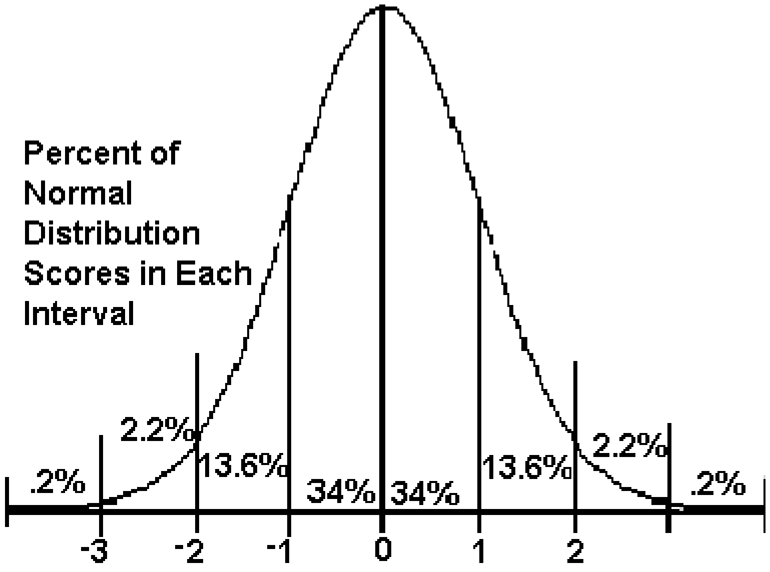
- Probability Density Function (PDF)
- Probability Mass Function (PMF)
- A random variable's probability distribution encapsulates both:
  - a characteristic type of "spread" or "shape" (distribution)
    - uniform
    - normal
    - etc.
  - the scaling and normalization factors that map between probabilities [0.0, 1.0] and the range of measurement values

Probability distributions

- Discrete distributions
  - Uniform
  - Bernoulli
  - Binomial
  - Geometric
  - Poisson
- Continuous distributions
  - Uniform
  - Normal

Normal Distribution

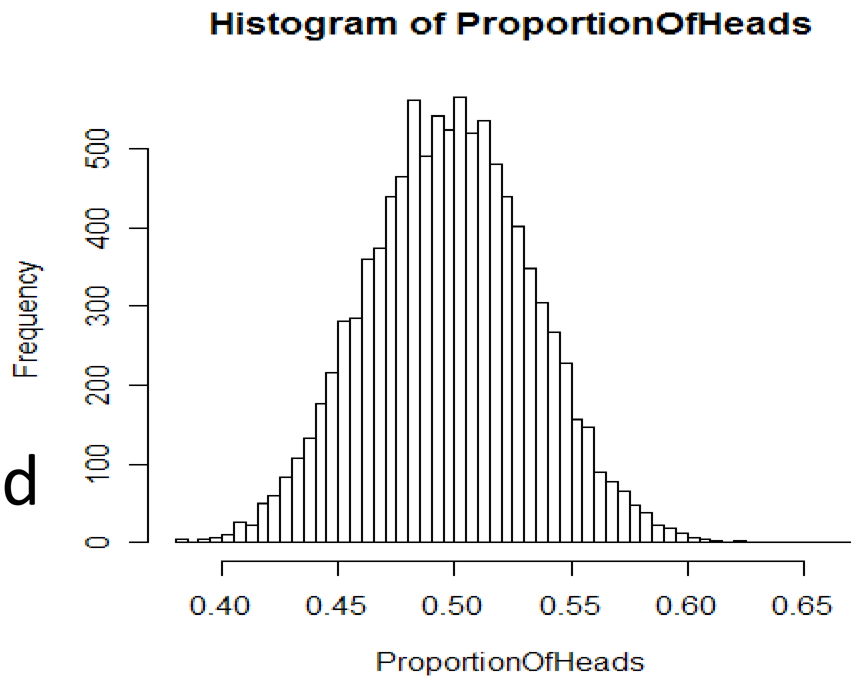
- aka Gaussian distribution
- Parameters:
  - μ
  - σ^2
- $f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$



Central Limit Theorem

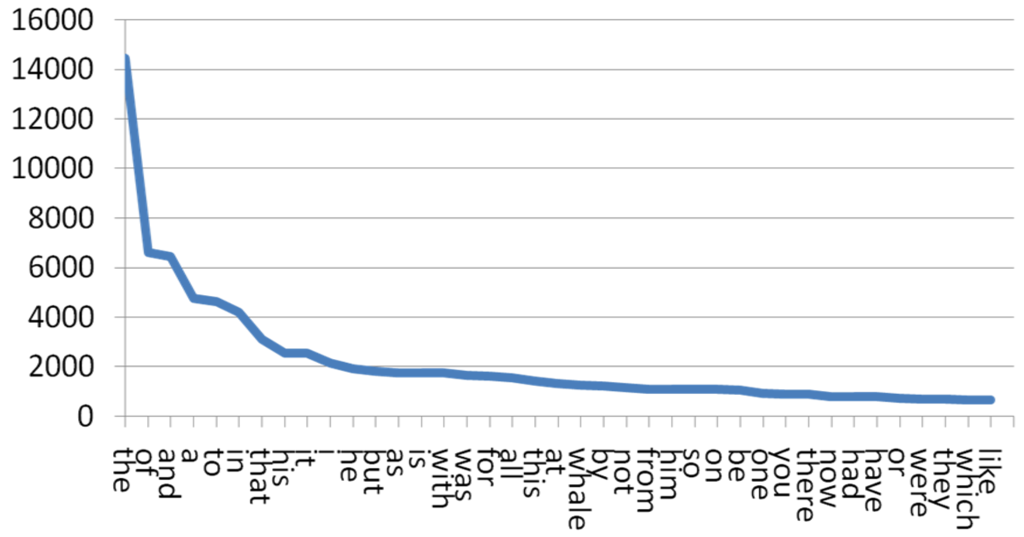
- When a large number of independent random variables is added together, its sum approaches a normal distribution
- Consider a fair coin toss
- $XX = \{ iiii\ sshoooooss\ heeeeeess \}$
- $PP(XX = heeeeeess) = .5$
- Many trials of this r.v.

  will be normally distributed



Zipf’s Law

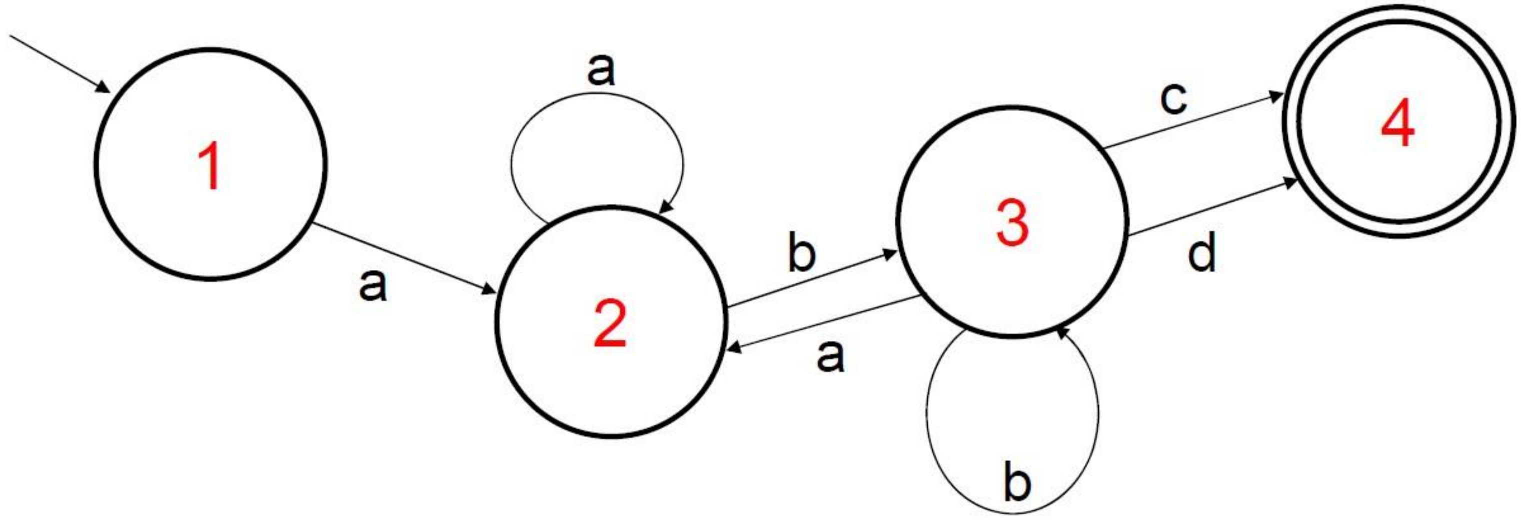
- The frequency of a word in a natural language corpus is inversely proportional to its tally rank
- This follows a geometric distribution



Finite State Machine

- deterministic

  a[ab]*b[cd]



Algorithmic complexity

$00\ kk$	constant	hash table
$00(\log nn)$	logarithmic	binary search
$00(nn)$	linear	naïve search (best and worst case)
$00(nn\ \log nn)$	log-linear	quicksort (best case)
$00(nn^2)$	quadratic	naïve sort (best and worst case)
$00(nn^3)$	cubic	parsing context-free-grammar (worst case)
$00(nn^{kk})$	polynomial	2-SAT (boolean satisfiability)
$00(kk^{nn})$	exponential	traveling salesman DP
$00(nn!)$	factorial	naïve traveling salesman

Using Bayes Theorem for POS tagging

Of course, you have this memorized

$$PP(AA|BB) = \frac{PP(BB|AA)PP(AA)}{PP(BB)}$$

Remember, this was our objective function

$$ii = \operatorname{argmax}_{tt} PP(ii_{ii} | oo_{ii})$$

$$ii = \operatorname{argmax}_{tt} \frac{PP(oo_{ii} | ii_{ii})PP(ii_{ii})}{PP(oo_{ii})}$$

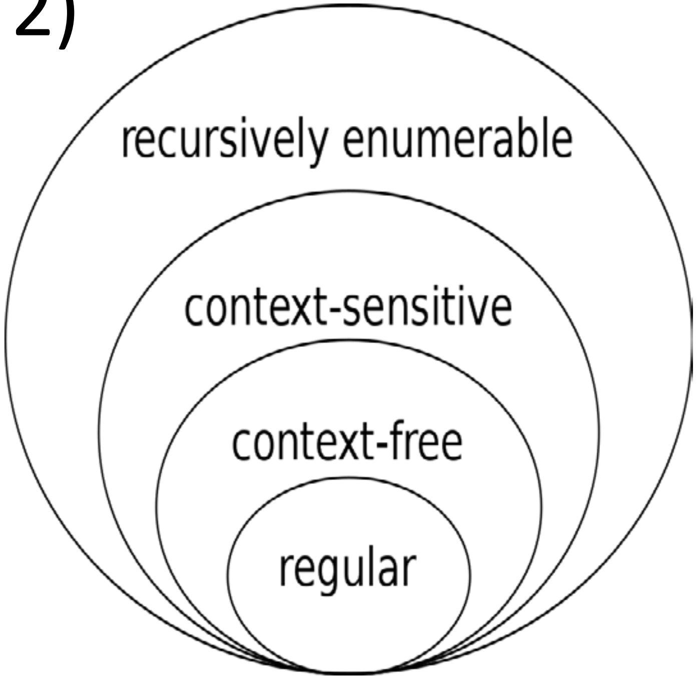
Smoothing

- When modeling a language, we don't want zero values from unseen items to ruin our probability calculations
- Various techniques address this problem:
  - add-one smoothing
  - Good-Turing method
  - Assume unseens have probability of the rarest observation
  - Ideally, smoothing preserves the validity of your probability space

Types of formal grammars

- Unrestricted, recursively enumerable (type 0)
- Context-sensitive (type 1)
- Context-free grammars (type 2)
- Regular grammars (type 3)

The Chomsky hierarchy



PCFG

- Probabilistic context-free grammar
- Adds probabilities to each rule
- Each distinct left-hand-side gets a probability mass 1.0
- Rule weights can be estimated from corpora

Classifiers

- Naïve Bayes
- TBL
- MaxEnt
- kNN
- Perceptron
- SVM
- CRF
- Bayesian networks
- HMM
- …

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/

Clustering

- Automated group of elements according to some feature
- Learn patterns in unlabeled data
- Use a “distance” metric to find similarity among items
- Hierarchical
- Non-hierarchical

Information Theory

- Entropy

  how disordered is a system?

$$H(X) = -\sum_{x} P(x) \log P(x)$$

‘waste’ information == heat

Dynamic programming

- We see this pattern often in computational linguistics
  1. Create a table to hold solutions to the sub-problems
  2. Fill in the table, re-using these previous results

Extending CFG parsing

- Example CFG rule:

$$SS \rightarrow NNPP\ VVPP$$

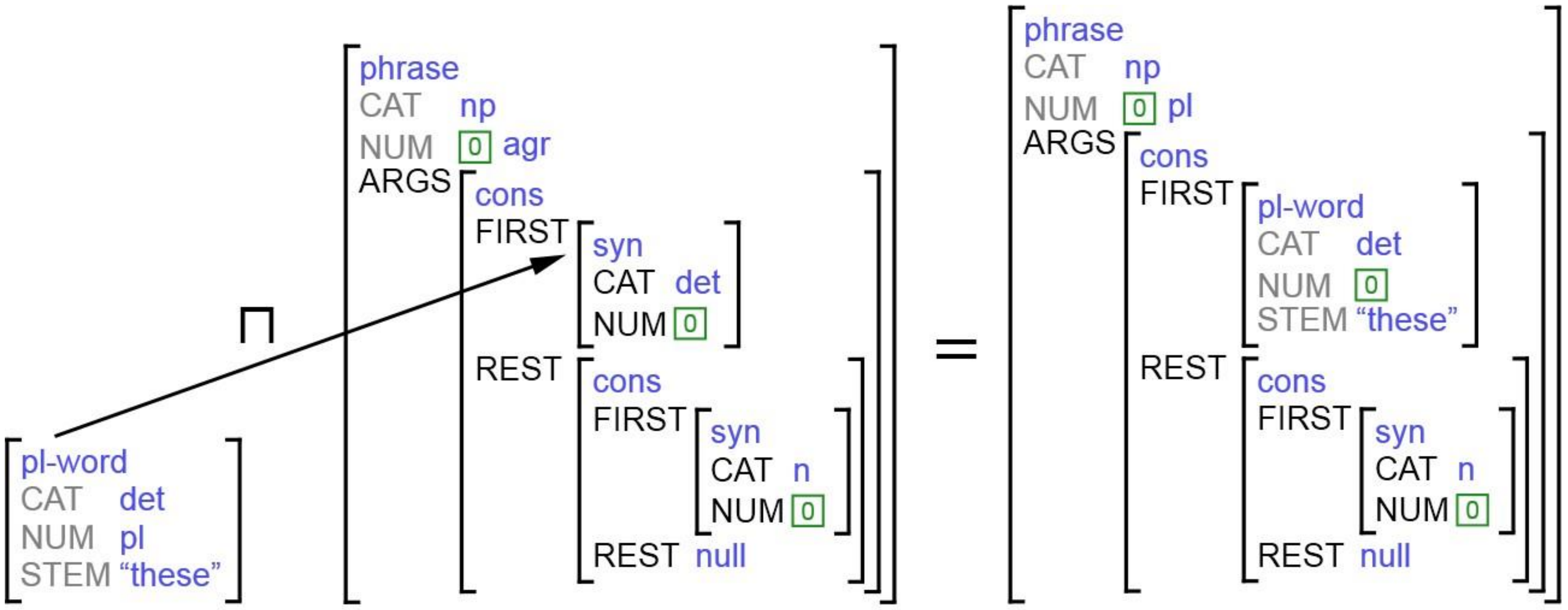
- Satisfiability:
  - Exact match of the entities on the right side of the rule
  - Do we have an NP? Do we have a VP?
  - No → try another rule. Yes →
- Combination:
  - The result of the rule application is:

$$SS$$

Head-Driven Phrase Structure Grammar

- "HPSG," Pollard and Sag, 1994
- Highly consistent and powerful formalism
- Monostratal, declarative, non-derivational, lexicalist, constraint-based
- Has been studied for many different languages
- Psycholinguistic evidence

TFS Unification



Lexical entry

Grammar rule

Partial rule application

Evaluation

		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

$$eepppaappeeppaa = \frac{iipp + iinn}{iipp + iinn + ffpp + ffnn}$$

$$ppppeeppiissiioonn = \frac{iipp}{iipp + ffpp}$$

$$eeppppoopp = \frac{ffpp + ffnn}{iipp + iinn + ffpp + ffnn}$$

$$ppeeppeerrrr = \frac{iipp}{iipp + ffnn}$$

$$ffeerrrrooaaii = \frac{ffpp}{ffpp + iinn}$$

Applications

- Information extraction (IE), Information retrieval (IR)
- Biomedical
- Document summarization
- Question answering (QA)
- Machine translation (MT)
- Human Input methods, Alternative Input Methods
  - speech recognition/synthesis
  - mobile devices
- ASR: Automatic Speech Recognition
- NLG: Natural Language Generation
- Speech Generation/Synthesis
- CALL: Computer-Assisted Language Learning

Thank you!

I’ll see you at the weekly Treehouse lecture series and around campus – Enjoy the rest of your summer

Function templating

For students working on the optional Project 6…

Flexible functions

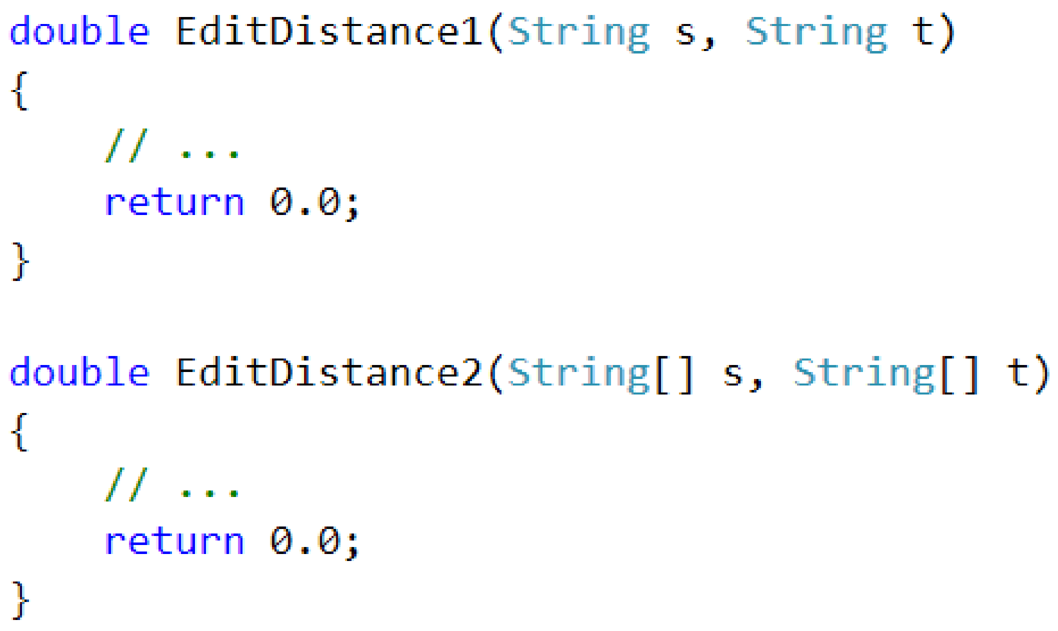
- For Project 6, we would like a function that operates on a sequence of elements—which could be of any type
- This is no problem in a dynamically typed language, but in C#, would we have to commit to a type?

```
# 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

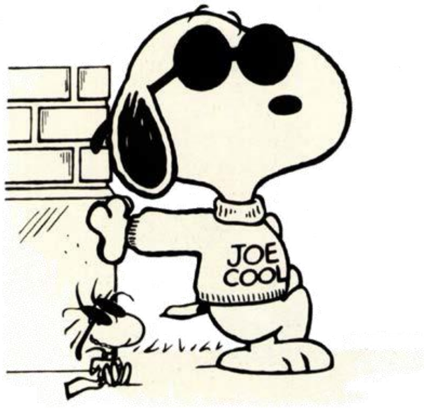
- Fortunately, strongly-typed languages have features that allow for this
  - You can specify exactly which arguments of a function (or parts of an entire class) are type-flexible
  - In C#, you can apply special *constraints* on the allowable types, which *expands* the things you can do with the types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

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;
}
```



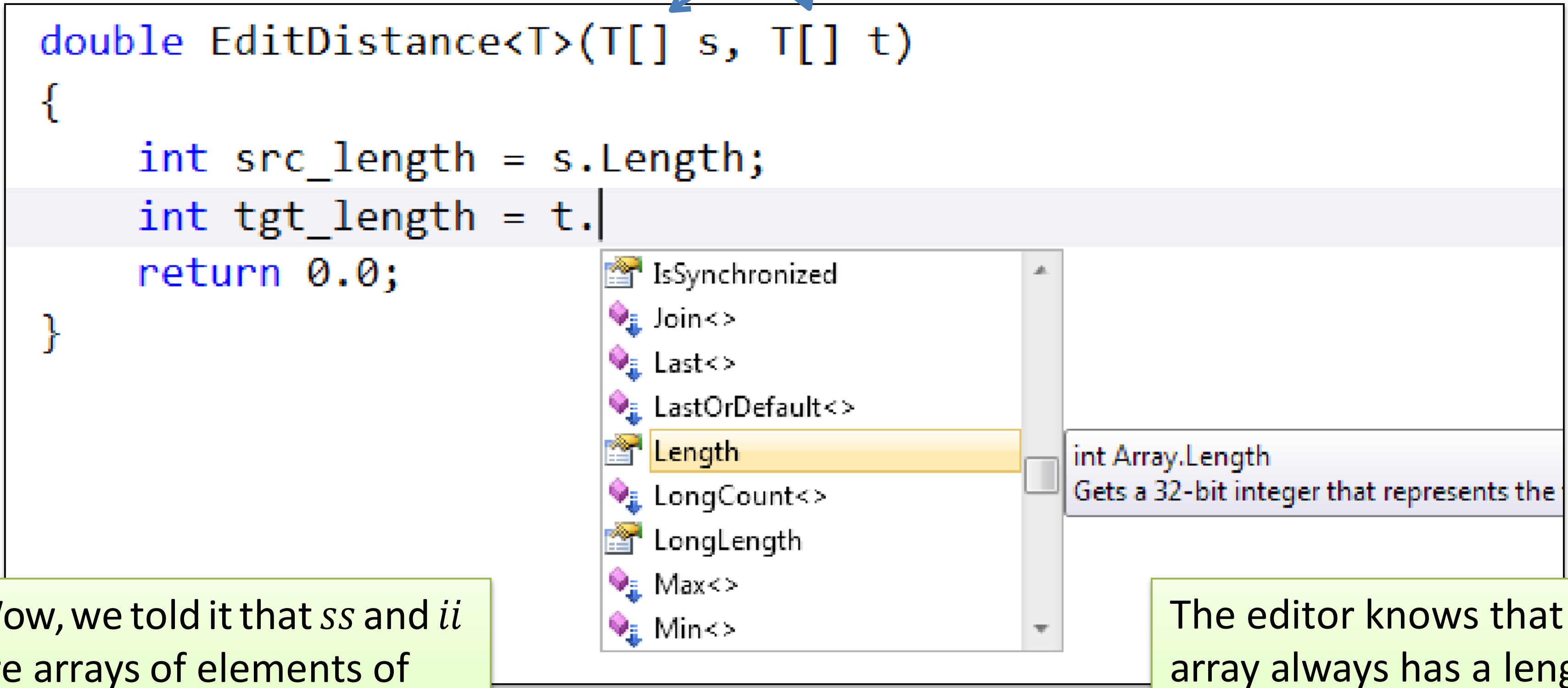
```
double EditDistance<T>(T s, T t)
{
    // ...
    return 0.0;
}
```

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 *s* and *t* are arrays of elements of type T, and strong typing is back!

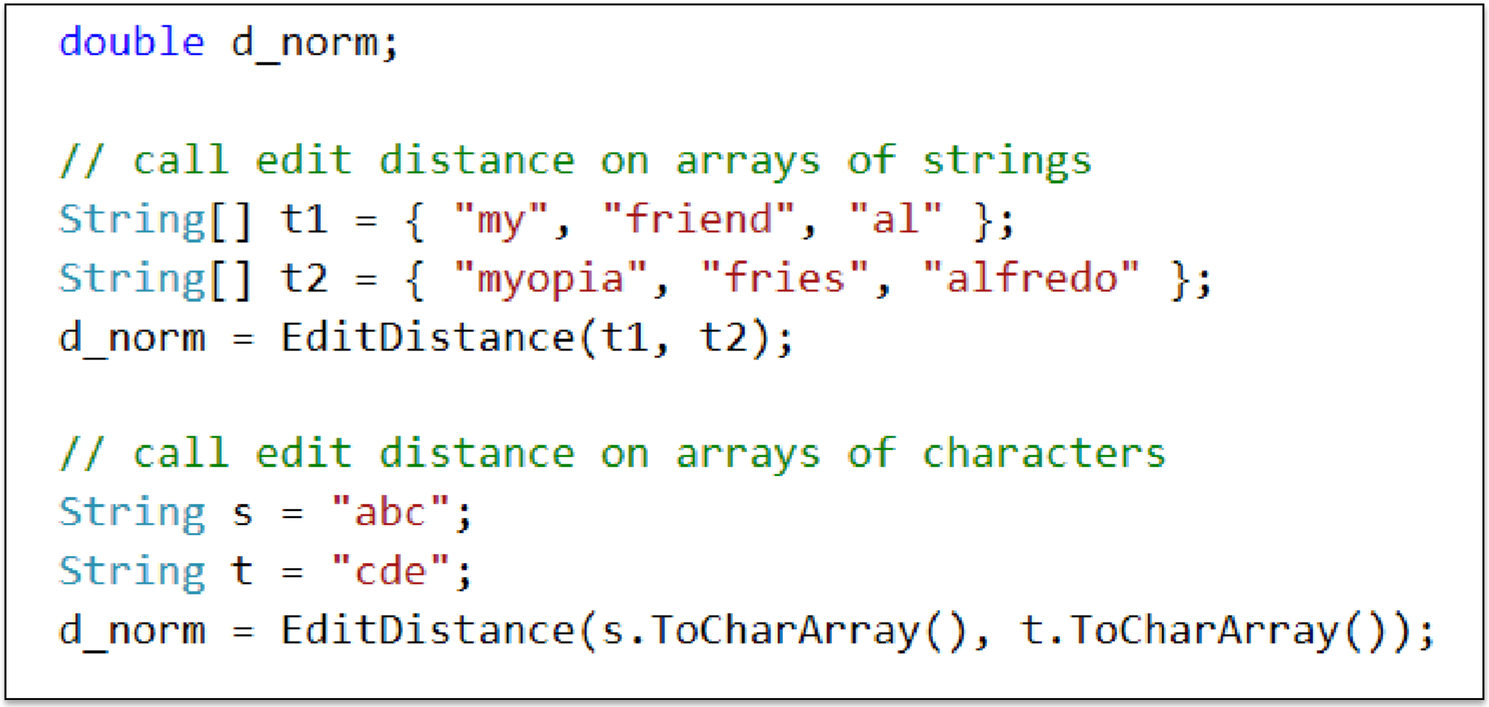
The editor knows that an array always has a length property.

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

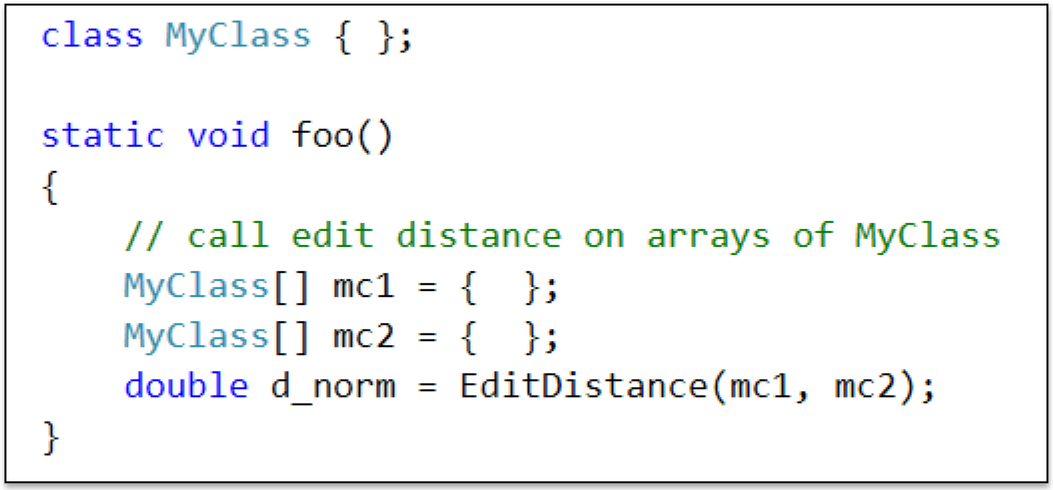
- Now we can call the function with an array of any type. The compiler figures out what type T is automatically



```
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 a an array of any other type of element(s)



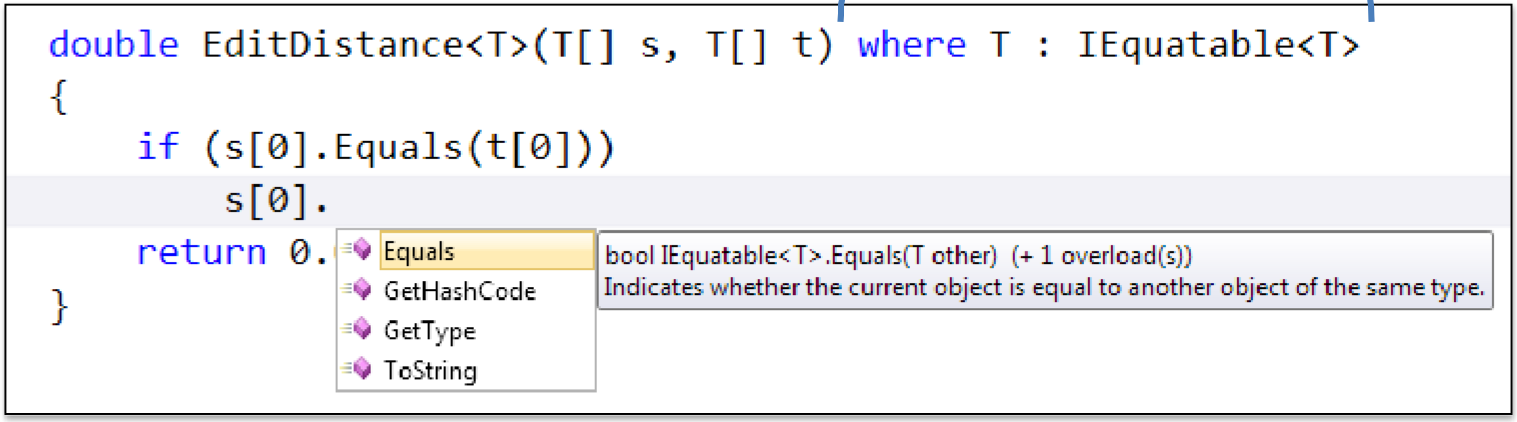
```
class MyClass { };

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.
}
```

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

- After adding the constraint, the editor (and compiler) immediately know(s) that elements of ss and ii 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

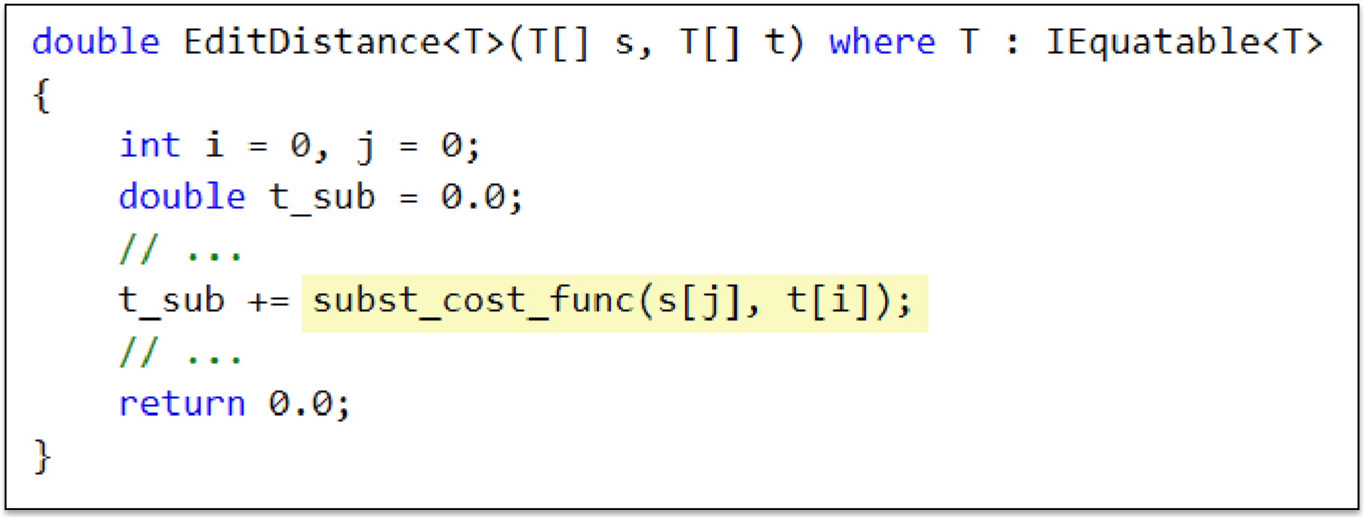
- Lastly, the EditDistance function needs to use a different substitution cost function depending on whether you're doing the outer calculation (between the two texts) or inner calculation (between two lines of text)

- 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

Lambda expressions

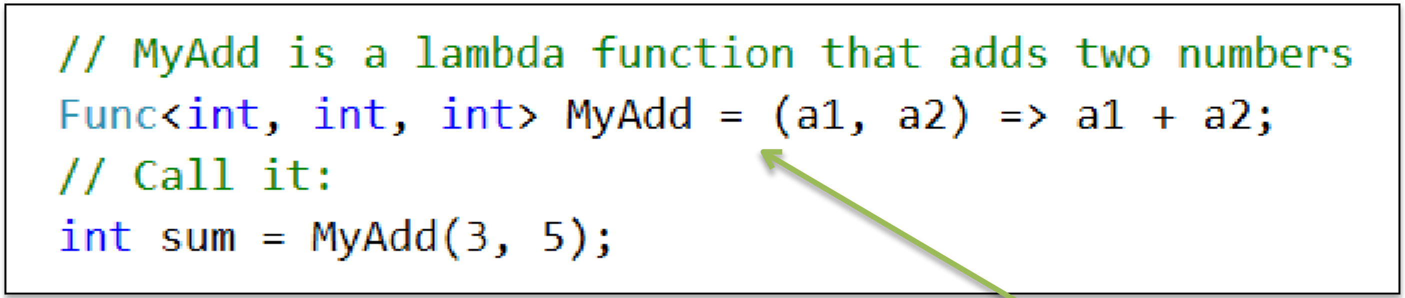
- A lambda expression is a portable, possibly anonymous (unnamed) snippet of code that you can store, refer to and carry and pass around just like any other data object
- It's an elegant way for callers to customize some aspect of a function's behavior
  - This is exactly what the EditDistance function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

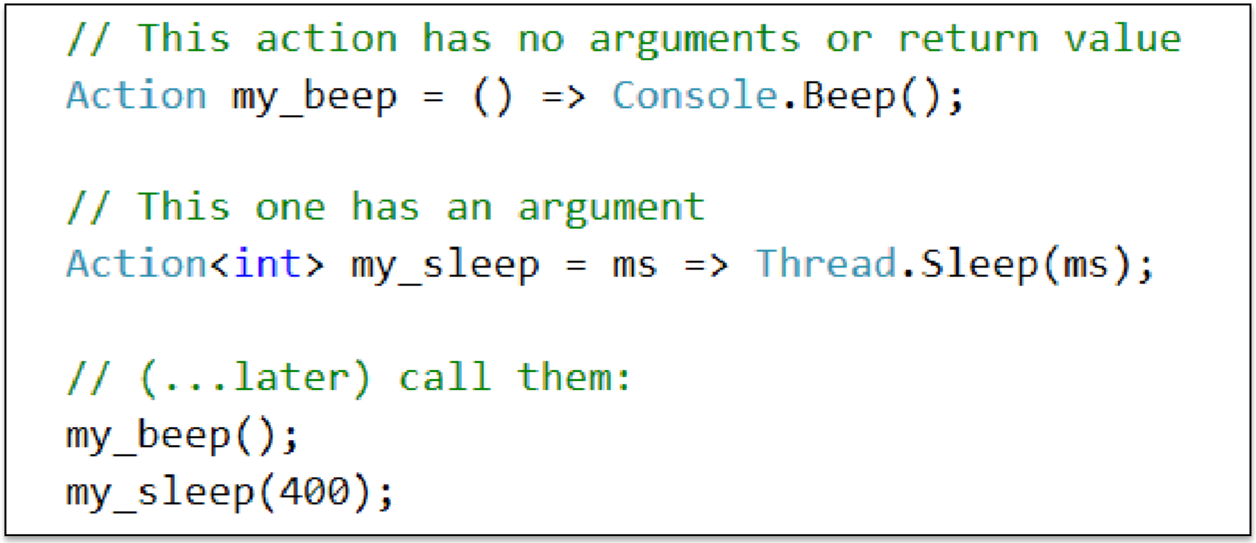


```
// MyAdd is a lambda function that adds two numbers
Func<int, int, int> MyAdd = (a1, a2) => a1 + a2;
// Call it:
int sum = MyAdd(3, 5);
```

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;
```

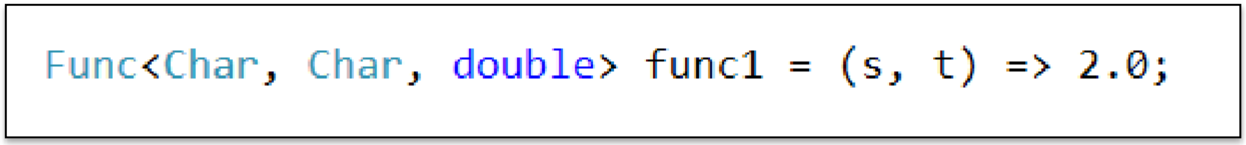
- If it's longer than one line, you might prefer to write it as shown below instead. If you use this curly brace syntax, you have to use the `return` keyword.

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

Once again, back to the project

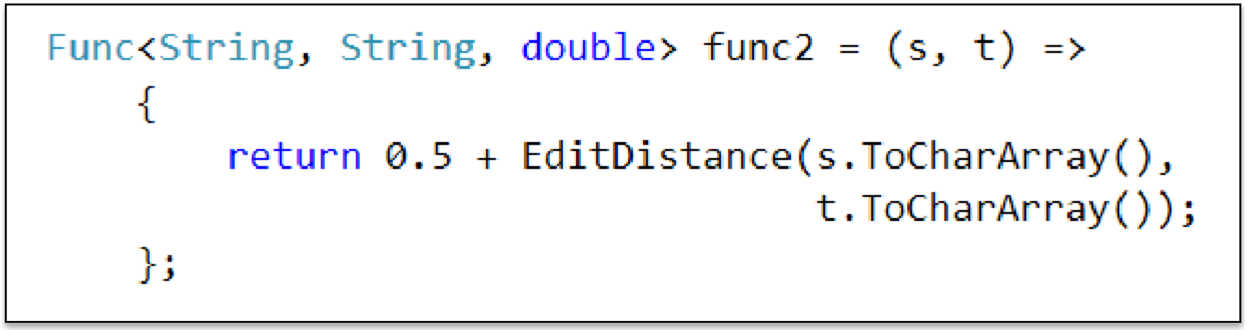
- Here are the two different substitution cost functions that we'd like to "pass in" to our EditDistance function

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

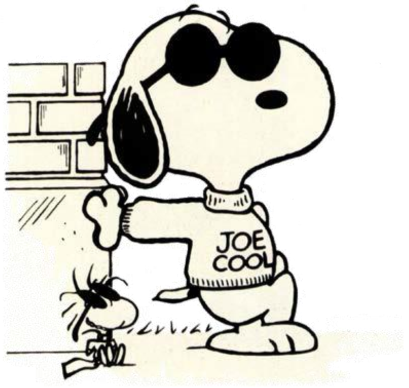


```
Func<String, String, double> func2 = (s, t) =>
    {
        return 0.5 + EditDistance(s.ToCharArray(),
                                  t.ToCharArray());
    };
```

Putting it all together

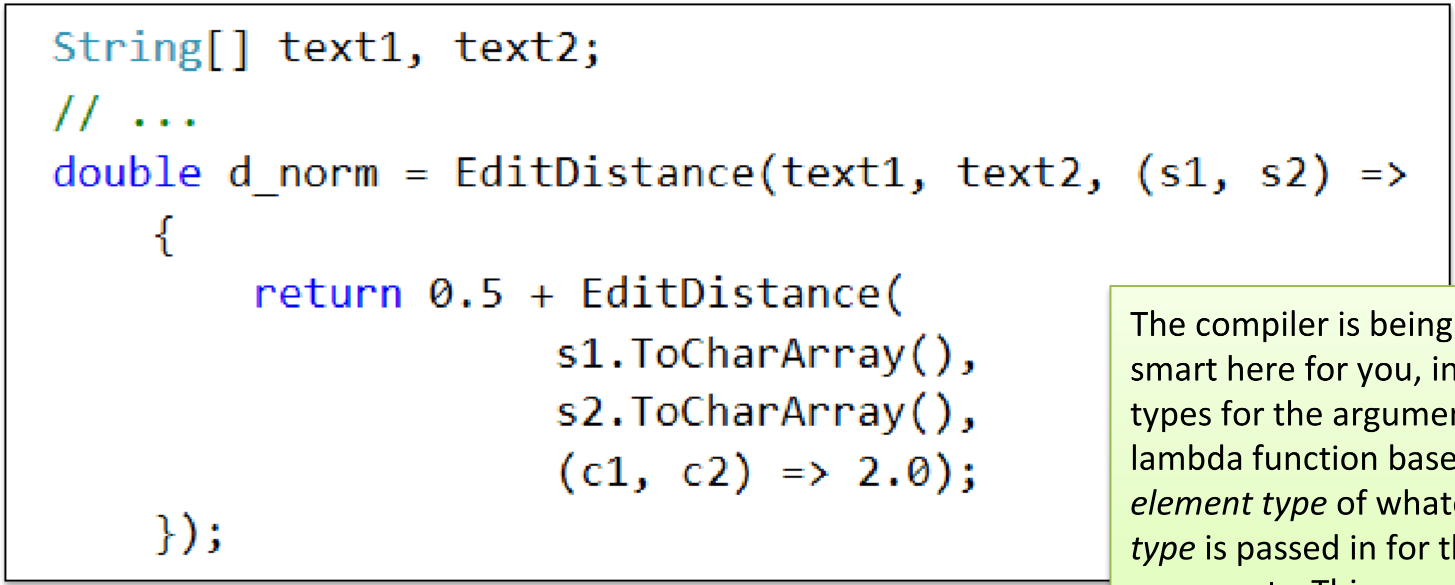
Now you're ready to add another parameter to the EditDistance function: the substitution cost function—a lambda function—that the caller will pass into the function, in order to customize its behavior

```
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.