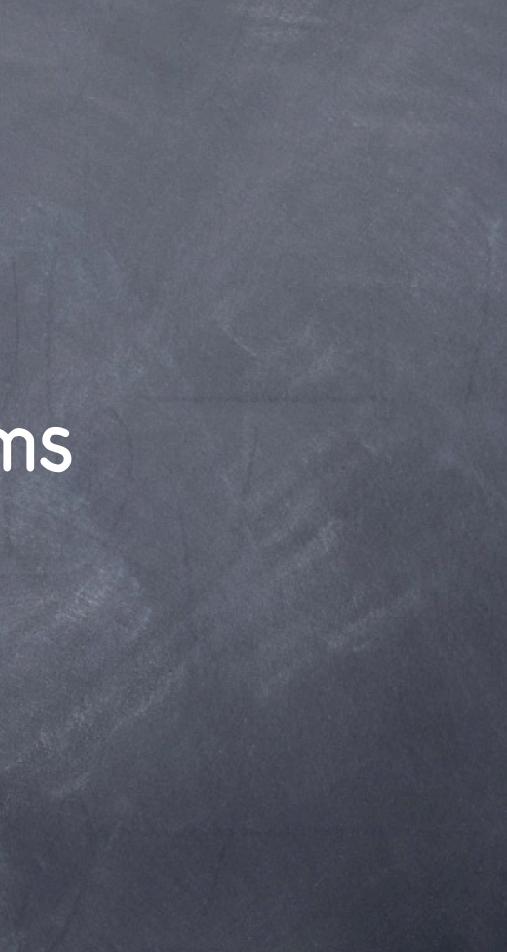
# Undecidable Problems

Andreas Klappenecker



# Post's Correspondence Problem

Given: A finite alphabet A, a finite set of pairs (x,y) of strings over the alphabet A.

Goal: Find a string over the alphabet A that can be composed in two different ways:

- by concatenating strings  $x_1x_2...x_n$  from the first components

- by concatenating strings  $y_1y_2...y_n$  from the second components

of a sequence  $(x_1, y_1)$ ,  $(x_2, y_2)$ , ...,  $(x_n, y_n)$  of the given pairs.

# PCP Example 1

Given: Alphabet  $A=\{a,b\}, P = \{(bab, a), (ab, abb), (a, ba)\}$ Solution: abbaba  $x_2 x_1 x_3 = ab || bab || a$  $y_2 y_1 y_3 = abb || a || ba$ Important: You need to select a sequence of pairs from P Projecting on first components must be the same as projecting on the second components. Reordering is not allowed.

## PCP Exercise

Given: Set of pairs P = { (1, 111), (10111,10), (10,0) } over A={0,1} Find a solution to Post's correspondence problem.

### Solution

Given: Set of pairs P = { (1, 111), (10111,10), (10,0) } over A={0,1} Find a solution to Post's correspondence problem. Solution: (2,1,1,3)  $x_2 x_1 x_1 x_3 = 10111 \parallel 1 \parallel 1 \parallel 1 \parallel 10 = 101111110$  $y_2 y_1 y_1 y_3 = 10 || 111 || 111 || 0 = 10111110$ 

# PCP Example

The Post's correspondence problem with  $P = \{ (001,0), (01,011), (01,101), (10,001) \} \text{ over } A = \{0,1\}$ has a solution, but the smallest requires n=66 words!

### Main Result

Theorem: The Post's correspondence problem is undecidable when the alphabet has at least two elements.

Idea of the proof: Reduce the halting problem onto the Post's correspondence problem. This is often done via an intermediate step, where a RAM machine with a single register is used.

### Context Free Grammars

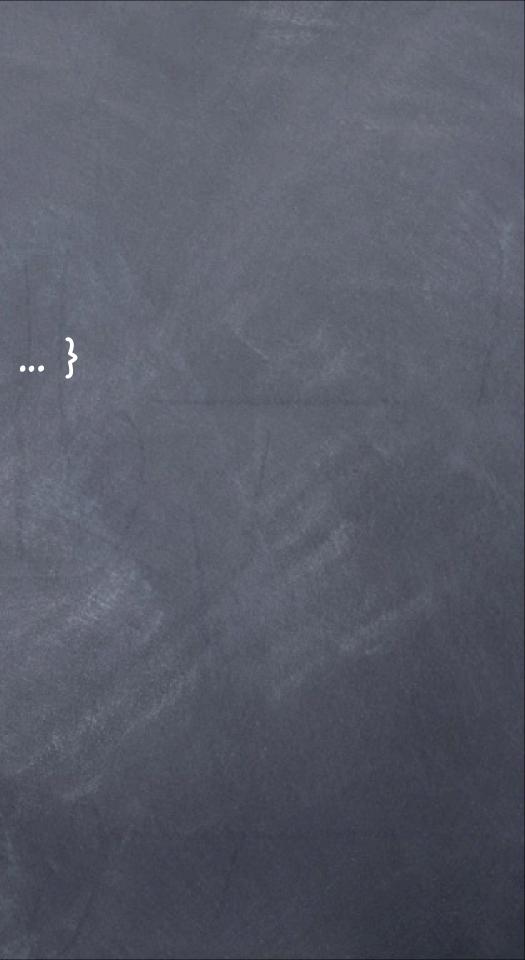
Problem: Is a given context-free grammar G unambiguous?

[A context-free grammar G is unambiguous iff every string s in L(G) has a unique left-most derivation. The reference grammars given for many programming languages are often ambiguous (e.g. dangling else problem). Sometimes formal languages have ambiguous and unambiguous grammars.]

This problem is undecidable. One can reduce the PCP problem to this one.

# Example

The regular language {  $\epsilon$ , a, aa, aaa, aaaa, aaaaa, ... } Ambiguous grammar: A -> aA | Aa |  $\epsilon$ Unambiguous grammar: A -> aA |  $\epsilon$ 



# Example 2

The context free grammar A -> A + A | A - A | a

is ambiguous, since a + a + a has two different left-most derivations.

A -> A + A -> a + A -> a + A + A -> a + a + A -> a + a + a and

A -> A + A -> A + A + A -> ... -> a + a + a (replacing left-most nonterminal A by A+A)

# Example 3 (Dangling Else)

Statement = **if** Condition **then** Statement if Condition then Statement else Statement

The following statement can be parsed in two different ways: if a then if b then s else s2 We can parse it as if a then (if b then s) else s2 or as if a then (if b then s else s2)

This is an example of an ambiguous language.

# Chomsky Hierarchy

The classification of formal grammars by Noam Chomsky imposes restrictions on the production rules u -> v:

(0) no restrictions

(1) no shortening: |u| <= |v|

(2) context free: u is a nonterminal symbol,  $v \neq \epsilon$ 

(3) (right) regular: u is a nonterminal symbol, v is a single terminal symbol, or a nonterminal symbol followed by a terminal symbol, start symbol can produce the empty string.

# Recursive Languages

A formal language is called recursive if and only if there exists a Turing machine such that on input of a finite input string - halts and accept if the string is in the language, - and halts and rejects otherwise.

Recursive languages correspond to decidable problems.

# Examples and Counterexamples

Every context-sensitive grammar is recursive. There exist recursive languages that are not context-sensitive. The language corresponding to the Halting problem is not recursive.

### Recursive Enumerable

The languages that are accepted by a Turing machine are called recursively enumerable languages (or semi-decidable languages).

There exists a TM that accepts yes instances, but might reject or loop forever on input of no instance.

Examples: The language of the Halting Problem, PCP

The type-0 formal languages are precisely the recursively enumerable languages.

### Recursive vs. Recursively Enumerable

Theorem: If a formal language is recursive, then it is recursively enumerable.

Proof. This follows from the definitions.

The converse does not hold. Example: PCP is recursively enumerable, but not recursive (decidable).

# Not Recursively Enumerable Languages

Theorem. There exist formal languages that are not recursively enumerable.

Proof. Let  $S = \{0,1\}^*$  be the set of all finite binary strings. This is a countably infinite set.

Consider the formal language P(S) of all sets of finite binary strings over the alphabet with symbols  $0, 1, \{,\}$ 

This language is uncountable by Cantor's theorem, as |S|<|P(S)|, so there cannot exist a Turing machine accepting P(S).