Pattern Languages

Seminar Algorithmic Learning Theory, SS 2015

Michael Krause

RWTH Aachen

07.05.2015

What are pattern languages?

- Type of formal languages
- Introduced by Dana Angluin in 1980

Why do they interest us?

Gold '67: *Language Identification in the Limit*

- Learning from positive and negative data more powerful than from positive data only

Why do they interest us?

Gold '67: *Language Identification in the Limit*

- Learning from positive and negative data more powerful than from positive data only

Angluin '80:
Inductive Inference of Formal Languages from Positive Data

- *inductive inference* - generalizing rules from examples

Why do they interest us?

Gold '67: *Language Identification in the Limit*

- Learning from positive and negative data more powerful than from positive data only

Angluin '80:
Inductive Inference of Formal Languages from Positive Data

- *inductive inference* - generalizing rules from examples

Finding Patterns Common to a Set of Strings

- Pattern languages

Why do they interest us?

Gold '67: *Language Identification in the Limit*

- Learning from positive and negative data more powerful than from positive data only

Angluin '80:
Inductive Inference of Formal Languages from Positive Data

- *inductive inference* - generalizing rules from examples

Finding Patterns Common to a Set of Strings

- Pattern languages
  - can be learned from positive data

Why do they interest us?

Gold '67: *Language Identification in the Limit*

- Learning from positive and negative data more powerful than from positive data only

Angluin '80:
Inductive Inference of Formal Languages from Positive Data

- *inductive inference* - generalizing rules from examples

Finding Patterns Common to a Set of Strings

- Pattern languages
  - can be learned from positive data
  - are a natural model for inductive inference

Example

- Let $p = x1y2x$ a pattern

Example

- Let $p = x1y2x$ a pattern
- By substituting

$$x := 10$$

$$y := 3$$

we get:

$$1013210$$

Example

- Let $p = x1y2x$ a pattern

- By substituting

$$x := 10$$

$$y := 3$$

we get:

$$1013210$$

- By substituting

$$x := 0x$$

$$y := z3$$

we get:

$$0x1z320x$$

Example

- Let $p = x1y2x$ a pattern

- By substituting

$$x := 10$$

$$y := 3$$

we get:

$$1013210$$

- By substituting

$$x := 0x$$

$$y := z3$$

we get:

$$0x1z320x$$

- By substituting

$$x := y$$

$$y := x$$

we get:

$$y1x2y$$

Example

- Let $p = x1y2x$ a pattern

- By substituting

$$x := 10$$

$$y := 3$$

we get:

$$1013210$$

- By substituting

$$x := 0x$$

$$y := z3$$

we get:

$$0x1z320x$$

- By substituting

$$x := y$$

$$y := x$$

we get:

$$y1x2y$$

- Many more substitutions possible!

Formal definition

- A pattern is any finite string of constants and variables

Formal definition

- A pattern is any finite string of constants and variables
- Σ: finite alphabet of constants, for us: $\Sigma = \{0, 1\}$

Formal definition

- A pattern is any finite string of constants and variables
- Σ: finite alphabet of constants, for us: $\Sigma = \{0, 1\}$
- X: set of variables disjoint from Σ, for us: $X = \{x_1, x_2, \dots\}$

Formal definition

- A pattern is any finite string of constants and variables
- Σ: finite alphabet of constants, for us: $\Sigma = \{0, 1\}$
- X: set of variables disjoint from Σ, for us: $X = \{x_1, x_2, \dots\}$
- A substitution replaces symbols in a pattern so that

Formal definition

- A pattern is any finite string of constants and variables
- Σ: finite alphabet of constants, for us: $\Sigma = \{0, 1\}$
- X: set of variables disjoint from Σ, for us: $X = \{x_1, x_2, \dots\}$
- A substitution replaces symbols in a pattern so that
  - constants remain the same

Formal definition

- A pattern is any finite string of constants and variables
- Σ: finite alphabet of constants, for us: $\Sigma = \{0, 1\}$
- X: set of variables disjoint from Σ, for us: $X = \{x_1, x_2, \dots\}$
- A substitution replaces symbols in a pattern so that
  - constants remain the same
  - variables are mapped to any *non-null* string

Formal definition

- A pattern is any finite string of constants and variables
- Σ: finite alphabet of constants, for us: $\Sigma = \{0, 1\}$
- X: set of variables disjoint from Σ, for us: $X = \{x_1, x_2, \dots\}$
- A substitution replaces symbols in a pattern so that
  - constants remain the same
  - variables are mapped to any *non-null* string
- The language of a pattern is the set of all strings of constants we get through substitutions

Our main question

- We call a set of strings of constants a sample
  e.g: $S = \{101, 10010, 0110011\}$

Our main question

- We call a set of strings of constants a sample
  e.g: $S = \{101, 10010, 0110011\}$
- Given a sample S, which pattern generates every string in S?

Our main question

- We call a set of strings of constants a sample
  e.g: $S = \{101, 10010, 0110011\}$
- Given a sample S, which pattern generates every string in S?
  - $p = x$ works, but here $q = x0x$ is more precise

Our main question

- We call a set of strings of constants a sample
  e.g: $S = \{101, 10010, 0110011\}$
- Given a sample S, which pattern generates every string in S?
  - $p = x$ works, but here $q = x0x$ is more precise
- We call a pattern p descriptive of S iff

Our main question

- We call a set of strings of constants a sample
  e.g: $S = \{101, 10010, 0110011\}$
- Given a sample S, which pattern generates every string in S?
  - $p = x$ works, but here $q = x0x$ is more precise
- We call a pattern p descriptive of S iff
  - it generates every string in S

Our main question

- We call a set of strings of constants a sample
  e.g: $S = \{101, 10010, 0110011\}$
- Given a sample S, which pattern generates every string in S?
  - $p = x$ works, but here $q = x0x$ is more precise
- We call a pattern p descriptive of S iff
  - it generates every string in S
  - no other pattern q generates S so that the language of q is a strict subset of the language of p

Our main question

- We call a set of strings of constants a sample
  e.g: $S = \{101, 10010, 0110011\}$
- Given a sample S, which pattern generates every string in S?
  - $p = x$ works, but here $q = x0x$ is more precise
- We call a pattern p descriptive of S iff
  - it generates every string in S
  - no other pattern q generates S so that the language of q is a strict subset of the language of p
- Given a sample S, which pattern is descriptive of S?

Repetition: Gold's model

- Objects: formal languages

Repetition: Gold's model

- Objects: formal languages
- Presentation: sequence of strings from a language, where each string appears at least once (a text)

Repetition: Gold's model

- Objects: formal languages
- Presentation: sequence of strings from a language, where each string appears at least once (a text)
- The learner outputs hypotheses after receiving a string

Repetition: Gold's model

- Objects: formal languages
- Presentation: sequence of strings from a language, where each string appears at least once (a text)
- The learner outputs hypotheses after receiving a string
- The learner learns the language, if, after some finite amount of time, the hypotheses are correct and remain the same

In our case

Assuming a learner is presented with a text $s_1, s_2, s_3, \ldots$ of some pattern language

In our case

Assuming a learner is presented with a text $s_1, s_2, s_3, \ldots$ of some pattern language

- The hypothesis space is the set of all patterns

In our case

Assuming a learner is presented with a text $s_1, s_2, s_3, \ldots$ of some pattern language

- The hypothesis space is the set of all patterns
- The hypotheses are patterns descriptive of the strings seen so far

In our case

Assuming a learner is presented with a text $s_1, s_2, s_3, \ldots$ of some pattern language

- The hypothesis space is the set of all patterns
- The hypotheses are patterns descriptive of the strings seen so far

Assuming there exists an algorithm to find descriptive patterns

In our case

Assuming a learner is presented with a text $s_1, s_2, s_3, \ldots$ of some pattern language

- The hypothesis space is the set of all patterns
- The hypotheses are patterns descriptive of the strings seen so far

Assuming there exists an algorithm to find descriptive patterns

- Then paper by Angluin shows:
  Pattern languages can be learned in the limit from positive data

Our first attempt

Let S be a sample

Our first attempt

Let S be a sample

- Enumerate all patterns of shorter or equal length of the shortest string in S

Our first attempt

Let S be a sample

- Enumerate all patterns of shorter or equal length of the shortest string in S
- Test for each pattern if its language contains S

Our first attempt

Let S be a sample

- Enumerate all patterns of shorter or equal length of the shortest string in S
- Test for each pattern if its language contains S
- From all patterns that pass the test:
  Select one which is minimal with regards to inclusion

Our first attempt

Let S be a sample

- Enumerate all patterns of shorter or equal length of the shortest string in S $\rightarrow$ exponential growth
- Test for each pattern if its language contains S
- From all patterns that pass the test:
  Select one which is minimal with regards to inclusion

Our first attempt

Let S be a sample

- Enumerate all patterns of shorter or equal length of the shortest string in S $\rightarrow$ exponential growth

Theorem (3.6, Angluin)

The membership problem for pattern languages is NP-complete

- Test for each pattern if its language contains S
- From all patterns that pass the test:
  Select one which is minimal with regards to inclusion

Our first attempt

Let S be a sample

- Enumerate all patterns of shorter or equal length of the shortest string in S $\rightarrow$ exponential growth

Theorem (3.6, Angluin)

The membership problem for pattern languages is NP-complete

- Test for each pattern if its language contains S $\rightarrow$ NP-complete
- From all patterns that pass the test:
  Select one which is minimal with regards to inclusion

Our first attempt

Let S be a sample

- Enumerate all patterns of shorter or equal length of the shortest string in S $\rightarrow$ exponential growth

Theorem (3.6, Angluin)

The membership problem for pattern languages is NP-complete

- Test for each pattern if its language contains S $\rightarrow$ NP-complete

Theorem (5.1, Jiang et al.)

The inclusion problem for arbitrary pattern languages is undecidable

- From all patterns that pass the test:
  Select one which is minimal with regards to inclusion

Our first attempt

Let S be a sample

- Enumerate all patterns of shorter or equal length of the shortest string in S $\rightarrow$ exponential growth

Theorem (3.6, Angluin)

The membership problem for pattern languages is NP-complete

- Test for each pattern if its language contains S $\rightarrow$ NP-complete

Theorem (5.1, Jiang et al.)

The inclusion problem for arbitrary pattern languages is undecidable

- From all patterns that pass the test:
  Select one which is minimal with regards to inclusion $\rightarrow$ ☹!

Our second attempt

Let S be a sample

- Enumerate all patterns of shorter or equal length of the shortest string in S $\rightarrow$ exponential growth
- Test for each pattern if its language contains S $\rightarrow$ NP-complete

Our second attempt

Let S be a sample

- Enumerate all patterns of shorter or equal length of the shortest string in S $\rightarrow$ exponential growth
- Test for each pattern if its language contains S $\rightarrow$ NP-complete

Corollary (3.4, Angluin)

Let p, q be patterns with the same length.
Then the language of q includes the language of p iff there is a substitution from q to p

Our second attempt

Let S be a sample

- Enumerate all patterns of shorter or equal length of the shortest string in S $\rightarrow$ exponential growth
- Test for each pattern if its language contains S $\rightarrow$ NP-complete

Corollary (3.4, Angluin)

Let p, q be patterns with the same length.
Then the language of q includes the language of p iff there is a substitution from q to p

- From all patterns that pass the test select the longest

Our second attempt

Let S be a sample

- Enumerate all patterns of shorter or equal length of the shortest string in S $\rightarrow$ exponential growth
- Test for each pattern if its language contains S $\rightarrow$ NP-complete

Corollary (3.4, Angluin)

Let p, q be patterns with the same length.
Then the language of q includes the language of p iff there is a substitution from q to p

- From all patterns that pass the test select the longest
- From the resulting set of patterns, output any which cannot be gained by substituting from another

Our second attempt

Let S be a sample

- Enumerate all patterns of shorter or equal length of the shortest string in S $\rightarrow$ exponential growth
- Test for each pattern if its language contains S $\rightarrow$ NP-complete

Corollary (3.4, Angluin)

Let p, q be patterns with the same length.
Then the language of q includes the language of p iff there is a substitution from q to p

- From all patterns that pass the test select the longest
- From the resulting set of patterns, output any which cannot be gained by substituting from another $\rightarrow$ NP-complete

Results so far

Let S be a sample

Theorem (4.2)

If $P \neq NP$ then there is no polynomial-time algorithm to find a pattern of maximum possible length descriptive of S

Results so far

Let S be a sample

Theorem (4.2)

If $P \neq NP$ then there is no polynomial-time algorithm to find a pattern of maximum possible length descriptive of S

- We may still solve this efficiently in special cases!

Comparison to other language types

- The pattern language $L(xx)$ is not context-free

Comparison to other language types

- The pattern language $L(xx)$ is not context-free
- The regular language $L(0|1) = \{0, 1\}$ is not a pattern language

Comparison to other language types

- The pattern language $L(xx)$ is not context-free
- The regular language $L(0|1) = \{0, 1\}$ is not a pattern language

Theorem (3.4, Jiang)

Every pattern language is context-sensitive

Comparison to other language types

- The pattern language $L(xx)$ is not context-free
- The regular language $L(0|1) = \{0, 1\}$ is not a pattern language

Theorem (3.4, Jiang)

Every pattern language is context-sensitive

Language	Membership	Emptiness	Equivalence	Inclusion
Context-sens.	D	U	U	U
Context-free	D	D	U	U
Regular	D	D	D	D
Pattern lang.	D	D	D	U

Table: D=decidable, U=undecidable

Overview

1. Introduce necessary conditions for one-variable patterns that could generate a string

Overview

1. Introduce necessary conditions for one-variable patterns that could generate a string
2. Bound the number of one-variable patterns that could generate every string in a sample

Overview

1. Introduce necessary conditions for one-variable patterns that could generate a string
2. Bound the number of one-variable patterns that could generate every string in a sample
3. Construct automata that recognize exactly these patterns

Overview

1. Introduce necessary conditions for one-variable patterns that could generate a string
2. Bound the number of one-variable patterns that could generate every string in a sample
3. Construct automata that recognize exactly these patterns
4. Finally, select a specific automaton that recognizes descriptive one-variable patterns

Feasible triples

Let p be a one-variable pattern and s a string of constants

Feasible triples

Let p be a one-variable pattern and s a string of constants

- We define a mapping $\tau(p) = (i, j, k)$ where

Feasible triples

Let p be a one-variable pattern and s a string of constants

- We define a mapping $\tau(p) = (i, j, k)$ where
  - i is the number of constants in p

Feasible triples

Let p be a one-variable pattern and s a string of constants

- We define a mapping $\tau(p) = (i, j, k)$ where
  - i is the number of constants in p
  - j is the number of occurences of x in p

Feasible triples

Let p be a one-variable pattern and s a string of constants

- We define a mapping $\tau(p) = (i, j, k)$ where
  - i is the number of constants in p
  - j is the number of occurences of x in p
  - k is the position of the first occurence of x in p

Feasible triples

Let p be a one-variable pattern and s a string of constants

- We define a mapping $\tau(p) = (i, j, k)$ where
  - i is the number of constants in p
  - j is the number of occurences of x in p
  - k is the position of the first occurence of x in p
- A pattern p can only generate s, if $\tau(p)$ is feasible for s

Feasible triples

Let p be a one-variable pattern and s a string of constants

- We define a mapping $\tau(p) = (i, j, k)$ where
  - i is the number of constants in p
  - j is the number of occurences of x in p
  - k is the position of the first occurence of x in p
- A pattern p can only generate s, if $\tau(p)$ is feasible for s

Let $S = \{s_1, \ldots, s_m\}$ a sample

Feasible triples

Let p be a one-variable pattern and s a string of constants

- We define a mapping $\tau(p) = (i, j, k)$ where
  - i is the number of constants in p
  - j is the number of occurences of x in p
  - k is the position of the first occurence of x in p
- A pattern p can only generate s, if $\tau(p)$ is feasible for s

Let $S = \{s_1, \ldots, s_m\}$ a sample

- Let F be the set of all triples feasible for every string in S

Feasible triples

Let p be a one-variable pattern and s a string of constants

- We define a mapping $\tau(p) = (i, j, k)$ where
  - i is the number of constants in p
  - j is the number of occurences of x in p
  - k is the position of the first occurence of x in p
- A pattern p can only generate s, if $\tau(p)$ is feasible for s

Let $S = \{s_1, \ldots, s_m\}$ a sample

- Let F be the set of all triples feasible for every string in S
- We can bound $|F| = \mathcal{O}(l^2 \log l)$ where l is the length of the shortest string in S

Anguin's algorithm for finding descriptive one-variable patterns

Let S be a sample

Anguin's algorithm for finding descriptive one-variable patterns

Let S be a sample

- Construct F by enumerating all feasible triples

Anguin's algorithm for finding descriptive one-variable patterns

Let S be a sample

- Construct F by enumerating all feasible triples
- For each triple $f \in F$

Anguin's algorithm for finding descriptive one-variable patterns

Let S be a sample

- Construct F by enumerating all feasible triples
- For each triple $f \in F$
  - For each string $s \in S$

Anguin's algorithm for finding descriptive one-variable patterns

Let S be a sample

- Construct F by enumerating all feasible triples
- For each triple $f \in F$
  - For each string $s \in S$
    - Construct automaton which recognizes patterns p that
      - fulfill $\tau(p) = f$ - generate s

Anguin's algorithm for finding descriptive one-variable patterns

Let S be a sample

- Construct F by enumerating all feasible triples
- For each triple $f \in F$
  - For each string $s \in S$
    - Construct automaton which recognizes patterns p that
      - fulfill $\tau(p) = f$ - generate s
  - Intersect these automata

Anguin's algorithm for finding descriptive one-variable patterns

Let S be a sample

- Construct F by enumerating all feasible triples
- For each triple $f \in F$
  - For each string $s \in S$
    - Construct automaton which recognizes patterns p that
      - fulfill $\tau(p) = f$ - generate s
  - Intersect these automata
- From the resulting set of automata: discard those whose language is empty

Anguin's algorithm for finding descriptive one-variable patterns

Let S be a sample

- Construct F by enumerating all feasible triples
- For each triple $f \in F$
  - For each string $s \in S$
    - Construct automaton which recognizes patterns p that
      - fulfill $\tau(p) = f$ - generate s
  - Intersect these automata
- From the resulting set of automata: discard those whose language is empty

Lemma (6.3)

Any pattern accepted by an automaton built from a triple that maximizes $i + j$ is descriptive of S among one variable patterns

Example

Let $S = \{s_1, s_2, s_3\}$ a sample with

$$s_1 = 1101011, s_2 = 10011, s_3 = 11111$$

- We construct F through enumeration
  We get:
  $F = \{(1,1,k),(1,2,k),(2,1,k),(3,1,k),(3,2,k),(4,1,k)\}$
  $1 \leq k \leq i+1$

Example

Let $S = \{s_1, s_2, s_3\}$ a sample with

$$s_1 = 1101011, s_2 = 10011, s_3 = 11111$$

- We construct F through enumeration
  We get:
  $F = \{(1, 1, k), (1, 2, k), (2, 1, k), (3, 1, k), (3, 2, k), (4, 1, k)\}$
  $1 \leq k \leq i + 1$
- We construct three automata per triple in F

Example

Let $S = \{s_1, s_2, s_3\}$ a sample with

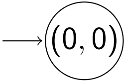
$$s_1 = 1101011, s_2 = 10011, s_3 = 11111$$

- We construct F through enumeration
  We get:
  $F = \{(1, 1, k), (1, 2, k), (2, 1, k), (3, 1, k), (3, 2, k), (4, 1, k)\}$
  $1 \leq k \leq i + 1$
- We construct three automata per triple in F
- In this example we do this for: $(3, 2, 2) \in F$

Triple: $(3, 2, 2)$, String: $s_1 = 1101011$

Triple: $(3, 2, 2)$, String: $s_1 = 1101011$
Substring starts at position 2, length: $(|s_1| - 3)/2 = 2$

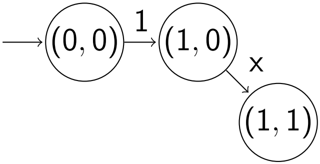
Triple: $(3, 2, 2)$, String: $s_1 = 1101011$
Substring starts at position 2, length: $(|s_1| - 3)/2 = 2$
Substring: $x = 10$



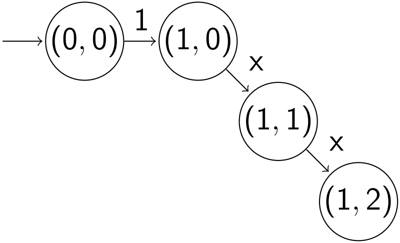
Triple: $(3, 2, 2)$, String: $s_1 = 1101011$
Substring starts at position 2, length: $(|s_1| - 3)/2 = 2$
Substring: $x = 10$

$$\begin{tikzcd} {} \arrow[r] & (0,0) \arrow[r, "1"] & (1,0) \end{tikzcd}$$

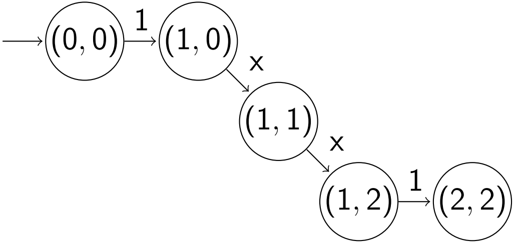
Triple: $(3, 2, 2)$, String: $s_1 = 1101011$
Substring starts at position 2, length: $(|s_1| - 3)/2 = 2$
Substring: $x = 10$



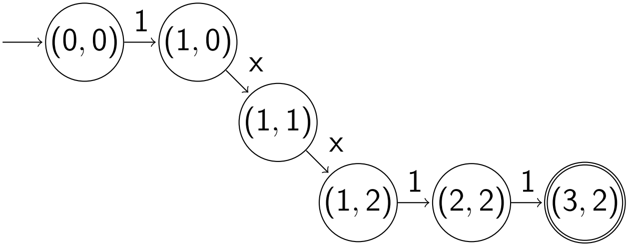
Triple: $(3, 2, 2)$, String: $s_1 = 1101011$
Substring starts at position 2, length: $(|s_1| - 3)/2 = 2$
Substring: $x = 10$



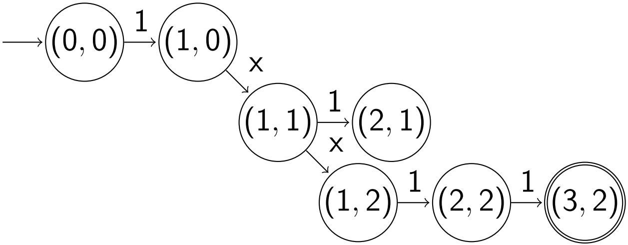
Triple: $(3, 2, 2)$, String: $s_1 = 1101011$
Substring starts at position 2, length: $(|s_1| - 3)/2 = 2$
Substring: $x = 10$



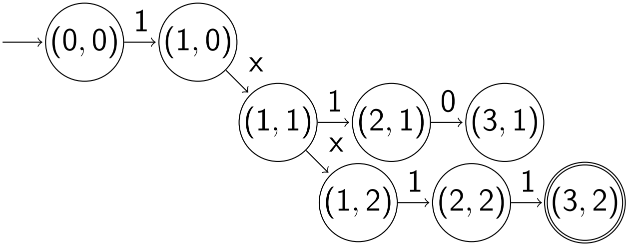
Triple: $(3, 2, 2)$, String: $s_1 = 1101011$
Substring starts at position 2, length: $(|s_1| - 3)/2 = 2$
Substring: $x = 10$



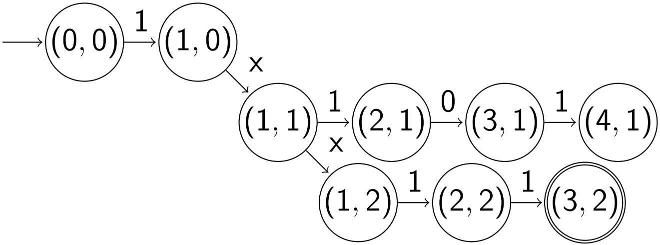
Triple: $(3,2,2)$, String: $s_1 = 1101011$
Substring starts at position 2, length: $(|s_1| - 3)/2 = 2$
Substring: $x = 10$



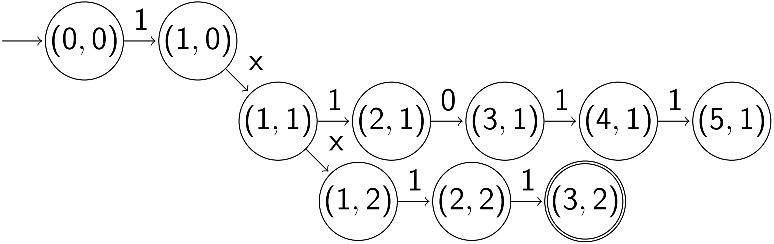
Triple: $(3, 2, 2)$, String: $s_1 = 1101011$
Substring starts at position 2, length: $(|s_1| - 3)/2 = 2$
Substring: $x = 10$



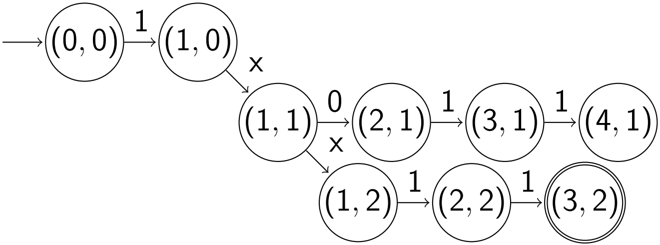
Triple: $(3, 2, 2)$, String: $s_1 = 1101011$
Substring starts at position 2, length: $(|s_1| - 3)/2 = 2$
Substring: $x = 10$



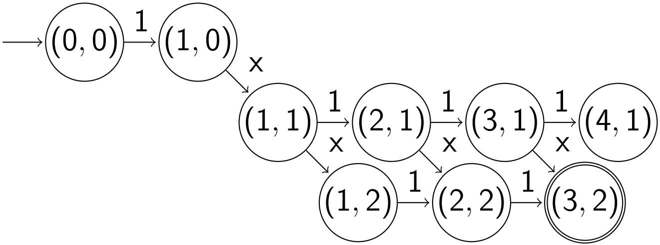
Triple: $(3, 2, 2)$, String: $s_1 = 1101011$
Substring starts at position 2, length: $(|s_1| - 3)/2 = 2$
Substring: $x = 10$



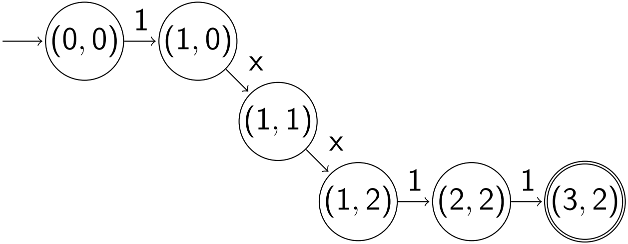
Triple: $(3, 2, 2)$, String: $s_2 = 10011$,
Substring length: $(|s_2| - 3)/2 = 1$
Substring: $x = 0$



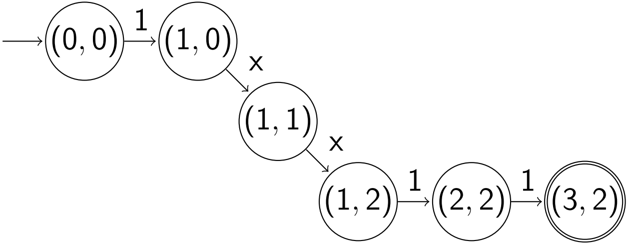
Triple: $(3, 2, 2)$, String: $s_3 = 11111$,
Substring length: $(|s_3| - 3)/2 = 1$
Substring: $x = 1$



Intersection of all three automata:



Intersection of all three automata:



Clearly the automaton recognizes the language $\{1xx11\}$

Recall:
$S = \{s_1, s_2, s_3\}$ with $s_1 = 1101011, s_2 = 10011, s_3 = 11111,$
$F = \{(1, 1, k), (1, 2, k), (2, 1, k), (3, 1, k), (3, 2, k), (4, 1, k)\},$
$1 \leq k \leq i + 1$

Example automata for $(3, 2, 2) \in F$

Recall:
$S = \{s_1, s_2, s_3\}$ with $s_1 = 1101011, s_2 = 10011, s_3 = 11111$,
$F = \{(1,1,k),(1,2,k),(2,1,k),(3,1,k),(3,2,k),(4,1,k)\}$,
$1 \leq k \leq i+1$

Example automata for $(3,2,2) \in F$

- Clearly $3+2$ maximizes $i+j$ in F

Recall:
$S = \{s_1, s_2, s_3\}$ with $s_1 = 1101011, s_2 = 10011, s_3 = 11111$,
$F = \{(1,1,k),(1,2,k),(2,1,k),(3,1,k),(3,2,k),(4,1,k)\}$,
$1 \leq k \leq i+1$

Example automata for $(3,2,2) \in F$

- Clearly $3+2$ maximizes $i+j$ in F
- The language recognized by the automaton for $(3,2,2) \in F$ is $\{1xx11\} \neq \emptyset$

Recall:
$S = \{s_1, s_2, s_3\}$ with $s_1 = 1101011, s_2 = 10011, s_3 = 11111,$
$F = \{(1,1,k),(1,2,k),(2,1,k),(3,1,k),(3,2,k),(4,1,k)\},$
$1 \leq k \leq i+1$

Example automata for $(3,2,2) \in F$

- Clearly $3+2$ maximizes $i+j$ in F
- The language recognized by the automaton for $(3,2,2) \in F$ is $\{1xx11\} \neq \emptyset$
- Thus, $1xx11$ is descriptive of S among one-variable patterns

Summary

Summary

Let S be a sample

- Construct F by enumerating all feasible triples

Summary

Let S be a sample

- Construct F by enumerating all feasible triples
- For each triple $f \in F$

Summary

Let S be a sample

- Construct F by enumerating all feasible triples
- For each triple $f \in F$
    - For each string $s \in S$ construct automaton

Summary

Let S be a sample

- Construct F by enumerating all feasible triples
- For each triple $f \in F$
  - For each string $s \in S$ construct automaton
  - Intersect these automata

Summary

Let S be a sample

- Construct F by enumerating all feasible triples
- For each triple $f \in F$
  - For each string $s \in S$ construct automaton
  - Intersect these automata
- Discard automata whose language is empty

Summary

Let S be a sample

- Construct F by enumerating all feasible triples
- For each triple $f \in F$
  - For each string $s \in S$ construct automaton
  - Intersect these automata
- Discard automata whose language is empty
- Choose any pattern recognized by an automaton that was built from a triple maximizing $i + j$

Summary

Let S be a sample

- Construct F by enumerating all feasible triples
- For each triple $f \in F$
  - For each string $s \in S$ construct automaton
  - Intersect these automata
- Discard automata whose language is empty
- Choose any pattern recognized by an automaton that was built from a triple maximizing $i + j$

- We can bound the number of feasible triples and construct the automata in time polynomial in their sizes

Summary

Let S be a sample

- Construct F by enumerating all feasible triples
- For each triple $f \in F$
  - For each string $s \in S$ construct automaton
  - Intersect these automata
- Discard automata whose language is empty
- Choose any pattern recognized by an automaton that was built from a triple maximizing $i + j$

- We can bound the number of feasible triples and construct the automata in time polynomial in their sizes
- The algorithm runs in time polynomial in the length of the input

What if we allowed wrong results?

- Paper by Steffen Lange and Rolf Wiehagen published in 1991
  Polynomial-time Inference of Arbitrary Pattern Languages

What if we allowed wrong results?

- Paper by Steffen Lange and Rolf Wiehagen published in 1991 *Polynomial-time Inference of Arbitrary Pattern Languages*
- Presents an algorithm that identifies any pattern language in the limit

What if we allowed wrong results?

- Paper by Steffen Lange and Rolf Wiehagen published in 1991
  Polynomial-time Inference of Arbitrary Pattern Languages
- Presents an algorithm that identifies any pattern language in the limit
- Each hypothesis is found in polynomial time

Lange and Wiehagen's algorithm

Idea:

- Only look at strings of minimal length (discard the others)

Lange and Wiehagen's algorithm

Idea:

- Only look at strings of minimal length (discard the others)
- Output pattern descriptive of strings of minimal length

Lange and Wiehagen's algorithm

Idea:

- Only look at strings of minimal length (discard the others)
- Output pattern descriptive of strings of minimal length

Result:

Lange and Wiehagen's algorithm

Idea:

- Only look at strings of minimal length (discard the others)
- Output pattern descriptive of strings of minimal length

Result:

- Will identify pattern language in the limit

Lange and Wiehagen's algorithm

Idea:

- Only look at strings of minimal length (discard the others)
- Output pattern descriptive of strings of minimal length

Result:

- Will identify pattern language in the limit
- Polynomial run time - finding descriptive patterns of the same length is easy

Lange and Wiehagen's algorithm

Idea:

- Only look at strings of minimal length (discard the others)
- Output pattern descriptive of strings of minimal length

Result:

- Will identify pattern language in the limit
- Polynomial run time - finding descriptive patterns of the same length is easy
- Algorithm will sometimes output wrong hypotheses

- Possible extensions of pattern languages

- Possible extensions of pattern languages
  - In extended pattern languages, empty substitutions are allowed

- Possible extensions of pattern languages
  - In extended pattern languages, empty substitutions are allowed
  - A regular pattern contains each variable at most once

- Possible extensions of pattern languages
  - In extended pattern languages, empty substitutions are allowed
  - A regular pattern contains each variable at most once

Language	Membership	Equivalence	Inclusion
Standard	NP	P	U
Regular	P	P	P
Extended	NP	Open	U
Extended Regular	P	P	P

Table: U=undecidable

- Possible extensions of pattern languages
  - In extended pattern languages, empty substitutions are allowed
  - A regular pattern contains each variable at most once

Language	Membership	Equivalence	Inclusion
Standard	NP	P	U
Regular	P	P	P
Extended	NP	Open	U
Extended Regular	P	P	P

Table: U=undecidable

- Polynomial *update* time does not guarantee good *learning* time

- Possible extensions of pattern languages
  - In extended pattern languages, empty substitutions are allowed
  - A regular pattern contains each variable at most once

Language	Membership	Equivalence	Inclusion
Standard	NP	P	U
Regular	P	P	P
Extended	NP	Open	U
Extended Regular	P	P	P

Table: U=undecidable

- Polynomial *update* time does not guarantee good *learning* time
- One variable patterns can be learned very efficiently - will be covered in next talk!

- Shinohara '82: Data entry systems

- Shinohara '82: Data entry systems
- Nix '83: Automatic text editing by examples

- Shinohara '82: Data entry systems
- Nix '83: Automatic text editing by examples
- Arimura '94: Finding patterns in amino acid sequences

- Shinohara '82: Data entry systems
- Nix '83: Automatic text editing by examples
- Arimura '94: Finding patterns in amino acid sequences

- Much work done in related fields!

Summary

- Pattern languages: model for inductive inference

Summary

- Pattern languages: model for inductive inference
- Finding descriptive patterns: generally not efficiently possible

Summary

- Pattern languages: model for inductive inference
- Finding descriptive patterns: generally not efficiently possible
- Special case: polynomial-time algorithm for one-variable patterns

Summary

- Pattern languages: model for inductive inference
- Finding descriptive patterns: generally not efficiently possible
- Special case: polynomial-time algorithm for one-variable patterns
- Lange/Wiehagen algorithm: inconsistent algorithm turns out to be very effective

References I

- Dana Angluin.
  Finding patterns common to a set of strings.
  Journal of Computer and System Sciences, 21(1):46 – 62, 1980.
- Dana Angluin.
  Inductive inference of formal languages from positive data.
  Information and Control, 45(2):117 – 135, 1980.
- Hiroki Arimura, Ryoichi Fujino, Takeshi Shinohara, and Setsuo Arikawa.
  Protein motif discovery from positive examples by minimal multiple generalization over regular patterns.
  Genome Informatics, 5:39–48, 1994.

References II

- Thomas Erlebach, Peter Rossmanith, Hans Stadtherr, Agelika Steger, and Thomas Zeugmann.
  Learning one-variable pattern languages very efficiently on average, in parallel, and by asking queries.
  Theor. Comput. Sci., 261(1):119–156, June 2001.
- Dominik D. Freydenberger and Daniel Reidenbach.
  Bad news on decision problems for patterns.
  Information and Computation, 208(1):83 – 96, 2010.
- E. Mark Gold.
  Language identification in the limit.
  Information and Control, 10(5):447–474, 1967.

References III

- Tao Jiang, Ming Li, Bala Ravikumar, and Kenneth W. Regan.
  Formal grammars and languages.
  In Mikhail J. Atallah and Marina Blanton, editors, *Algorithms and Theory of Computation Handbook*, pages 20–20. Chapman & Hall/CRC, 2010.
- Tao Jiang, Arto Salomaa, Kai Salomaa, and Sheng Yu.
  Inclusion is undecidable for pattern languages.
  In Andrzej Lingas, Rolf Karlsson, and Svante Carlsson, editors, *Automata, Languages and Programming*, volume 700 of *Lecture Notes in Computer Science*, pages 301–312. Springer Berlin Heidelberg, 1993.

References IV

- Steffen Lange and Rolf Wiehagen.
  Polynomial-time inference of arbitrary pattern languages.
  New Generation Computing, 8(4):361–370, 1991.
- Yen Kaow Ng and Takeshi Shinohara.
  Developments from enquiries into the learnability of the pattern languages from positive data.
  Theoretical Computer Science, 397(1):150–165, 2008.
- Takeshi Shinohara.
  Polynomial time inference of pattern languages and its applications.
  In *Proceedings of the 7th IBM Symposium on Mathematical Foundations of Computer Science*, pages 191–209, 1982.

References V

- Takeshi Shinohara and Setsuo Arikawa.
  Pattern inference.
  In *Algorithmic Learning for Knowledge-Based Systems*, pages 259–291. Springer, 1995.
- Thomas Zeugmann.
  Lange and wiehagen's pattern language learning algorithm: An average-case analysis with respect to its total learning time.
  Annals of Mathematics and Artificial Intelligence, 23(1-2):117–145, January 1998.