한국정보과학회
프로그래밍언어연구회
겨울학교 2008

String Analysis

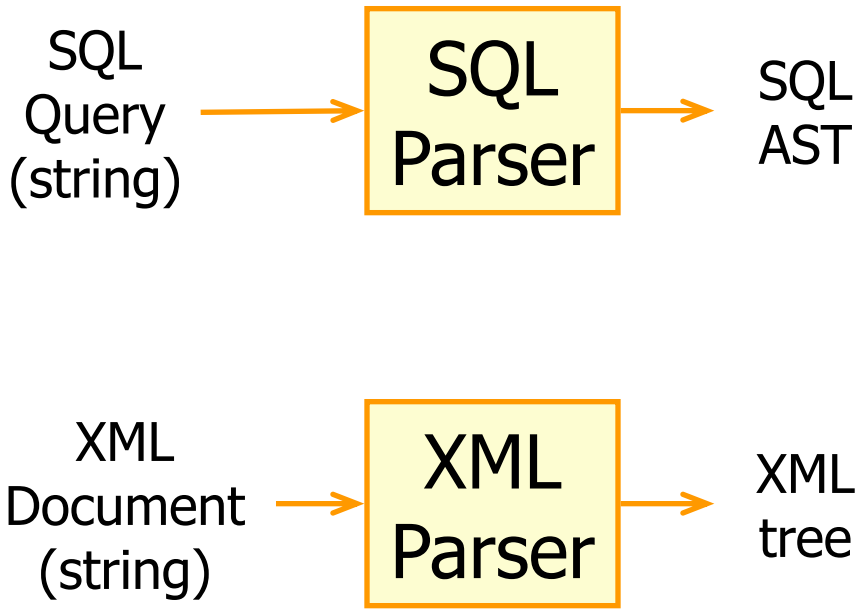
2008.01.31

한양대학교 안산캠퍼스 컴퓨터공학과
프로그래밍언어연구실
도경구

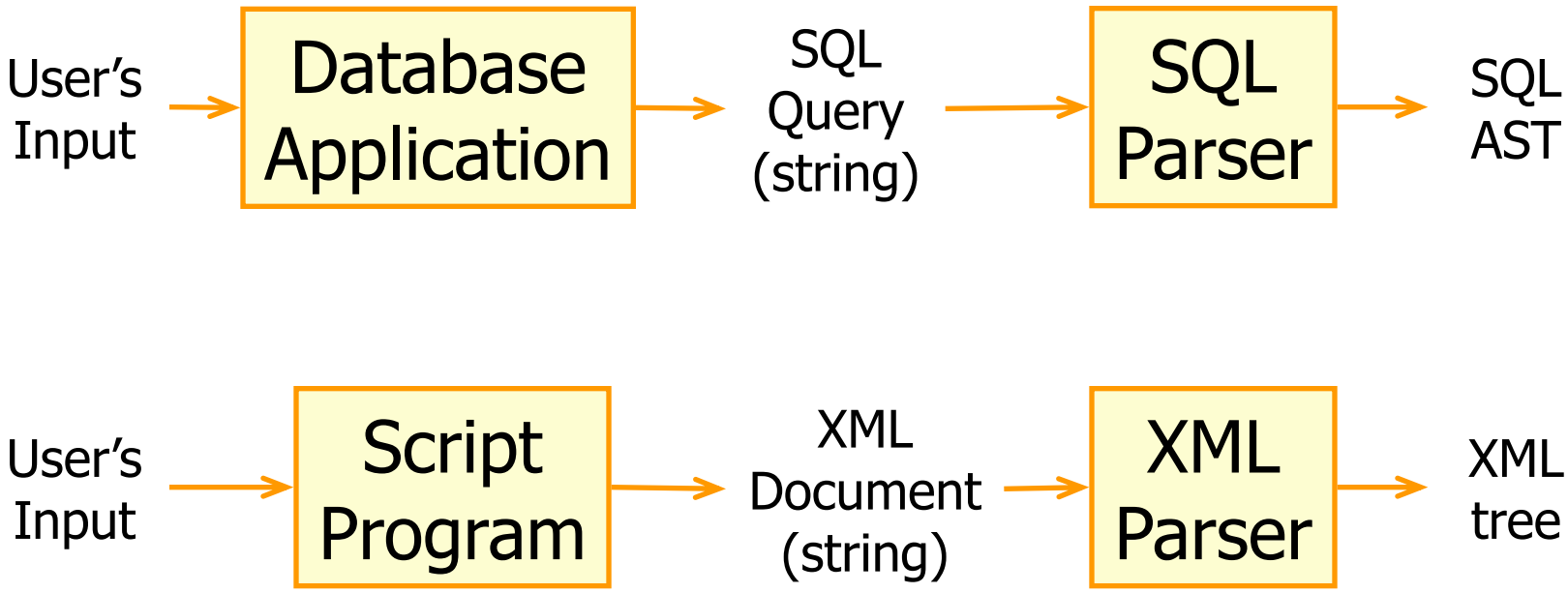
String

- A Sequence of Characters
- Examples
  - A program
  - A HTML document
  - An XML document
  - A collection of formatted data
  - A SQL command
  - etc.

"Classic" String Analysis = Parsing



Automatically Generated String



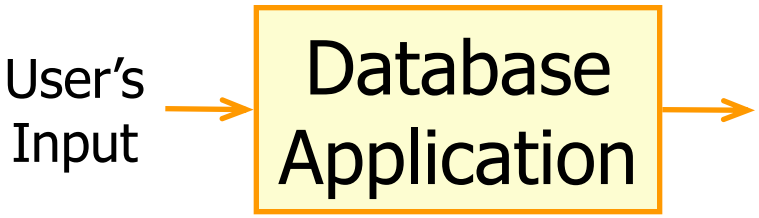
Example: DB Program

```
public void printAddresses(String id) throws SQLException {
  Connection con = DriverManager.getConnection("students.db");
  String q = "SELECT * FROM address";
  if (id != 0) q = q + "WHERE studentid=" + id;
  ResultSet rs = con.createStatement().executeQuery(q);
  while (rs.next()) { System.out.println.getString("addr")); }
```

taken from Christensen/Moeller/Schwartzbach's SAS2003 paper "Precise analysis of string expression" with minor modification

- This program parses OK.
- But,
- Does the generated SQL query parse OK?
- Which database table does this program access?
- Is this program vulnerable to SQL injection attack?

String-Generating Programs



- Does this DB application always generate grammatically correct SQL queries?
- Which DB tables and fields this application access and update?
- Is the DB application free from SQL injection attack?

- Does this script always generate valid XML documents?
- Is the script free from command injection attack?

What to Know and How to Know?

We would like to know:

- whether or not SQL queries generated by the application are always grammatically correct ⇒ syntax analysis
- DB tables and fields this application access and update ⇒ impact analysis
- whether or not the application is not vulnerable to SQL injection attack ⇒ security vulnerability analysis

What to Know and How to Know?

We would like to know:

- whether or not XML documents generated by this script are always valid ⇒ syntax analysis
- whether or not this script program is vulnerable to command injection attack ⇒ security vulnerability analysis

"Static" String Analysis

- Approximates the value of string expression with a grammar.
- History
  - XDuce: A statically typed XML processing language [Hosoya/Pierce, WebDB 2000]
  - Christensen/Moeller/Schwartzbach's Java String Analyzer [SAS 2003]
  - Minamide's PHP String Analyzer [WWW 2005]
  - Choi/Lee/Kim/Doh's abstract-interpretation approach [APLAS 2006]
  - Doh/Kim/Schmidt's abstract parsing [unpublished]

XDuce

- a domain-specific language for XML transformation
- extends ML's type system with regular expression types for describing the structure of XML documents
- Its sound type system guarantees the validity of dynamically generated documents.
- Example: [taken from Hosoya/Pierce's ACM TOIT 2004 paper]
  - Given an address book XML document,
  - create a telephone book XML document by extracting just the entries with telephone numbers.

XDuce Example

the type definitions for input documents

```
type Addrbook = addrbook[Person*]
type Person   = person[Name,Email*,Tel?]
type Name     = name[String]
type Email    = email[String]
type Tel      = tel[String]
```

the type definitions for output documents

```
type TelBook   = telbook[TelPerson*]
type TelPerson = person[Name,Tel]
```

```
let val doc = load_xml("mybook.xml")
let val valid_doc = validate doc with Addrbook
let val out_doc = match valid_doc with
                    addrbook[val persons as Person*] ->
                      telbook[make_tel_book(persons)]
save_xml("output.xml")(out_doc)
```

```
fun make_tel_book (val ps as Person*) : TelPerson* =
  match ps with
      person[name[val n as String], Email*, tel[val t as String]],
      val rest as Person*
        -> person[name[n], tel[n]], make_tel_book(rest)
    | person[name[val n as String], Email*], val rest as Person*
        -> make_tel_book(rest)
    | () -> ()
```

"Static" String Analysis

- Approximates the value of string expression with a grammar.
- History
  - XDuce: A statically typed XML processing language [Hosoya/Pierce, WebDB 2000]
  - Christensen/Moeller/Schwartzbach's Java String Analyzer [SAS 2003]
  - Minamide's PHP String Analyzer [WWW 2005]
  - Choi/Lee/Kim/Doh's abstract-interpretation approach [APLAS 2006]
  - Doh/Kim/Schmidt's abstract parsing [unpublished]

Example: String Analysis

```
x = "a";
while <cond> do
  x = "0" + x;
  x = x + "1";
print x;
```

Hot Spot

The possible values of string variable, x, at the hot spot are expected as follows:

$$\{ 0^n a 1^n \mid n \geq 0 \}$$

Can we obtain the sound approximation of the string values by static analysis of the program?

Example: String Analysis

```
x = "a";
while <cond> do
  x = "0" + x;
  x = x + "1";
print x;
```

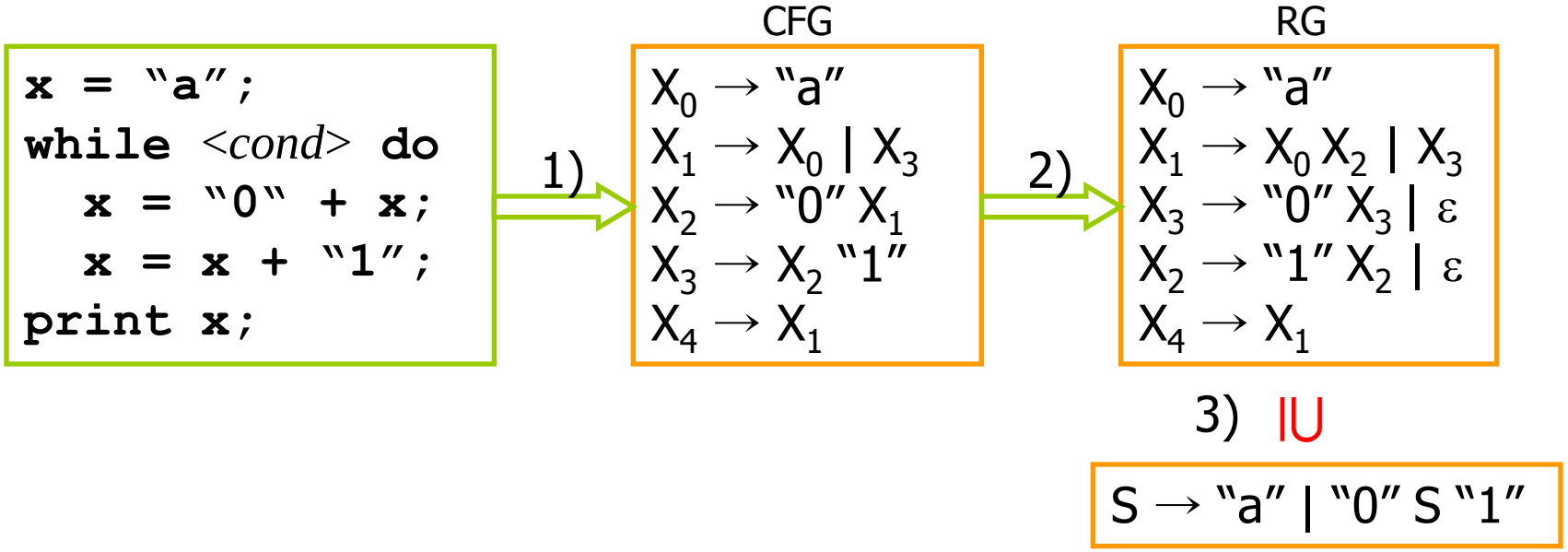
Hot Spot

1) Statically analyze the program and determine a context-free grammar G which represents the values of a variable, **x**, at the hot spot.
2) See if G is equivalent to the reference grammar:

$$S \rightarrow \text{“a”} \mid \text{“0”}\ S\ \text{“1”}$$

Problem: Checking a context-free grammar is included in another context-free grammar is “undecidable”.

C/M/S's Java String Analyzer



1) Extract a context-free grammar from the program.
2) Determine the regular approximation of the extracted grammar.
3) See if the regular grammar includes the reference grammar.

How to deal with string operators other than concatenation?

```
x = "a";
while <cond> do
  x = "00" + x;
  x = x + "1";
print replace("00","0",x);
```

1)

$X_0 \rightarrow$ "a"
$X_1 \rightarrow X_0 \mid X_3$
$X_2 \rightarrow$ "00" X_1
$X_3 \rightarrow X_2$ "1"
$X_4 \rightarrow$ replace("00","0",X_1)

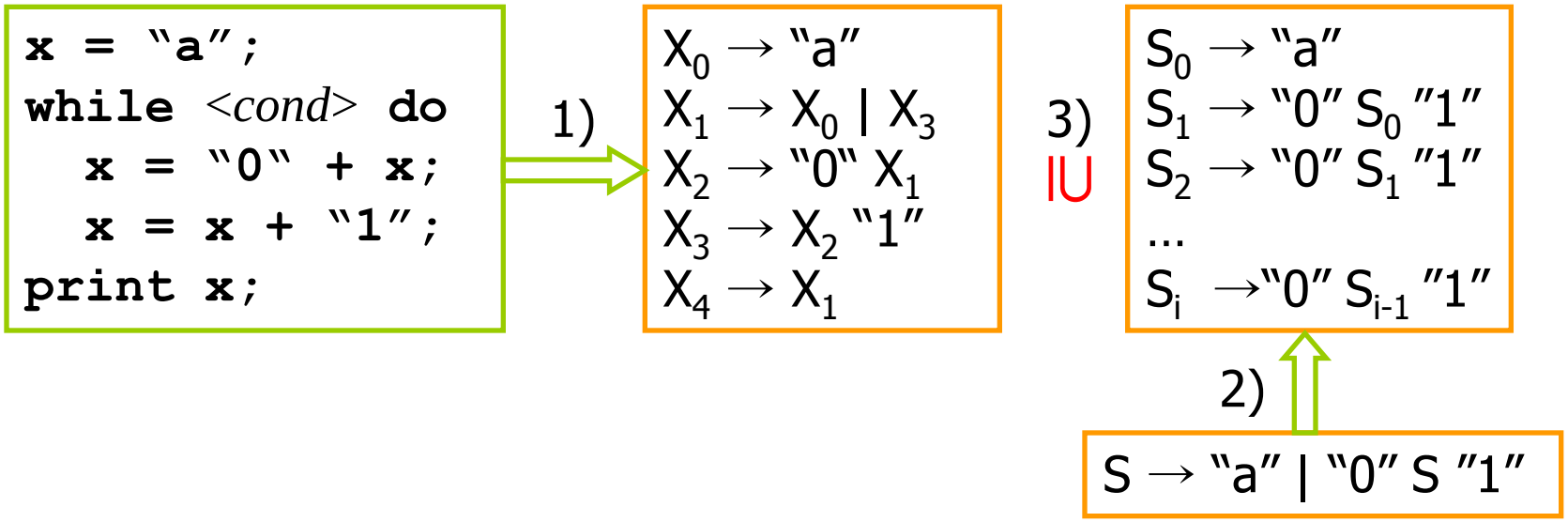
$X_0 \rightarrow$ "a"
$X_1 \rightarrow X_0 \mid X_3$
$X_2 \rightarrow$ "0" X_1
$X_3 \rightarrow X_2$ "1"
$X_4 \rightarrow X_1$

Not clear how to deal with other string operators!

"Static" String Analysis

- Approximates the value of string expression with a grammar.
- History
  - XDuce: A statically typed XML processing language [Hosoya/Pierce, WebDB 2000]
  - Christensen/Moeller/Schwartzbach's Java String Analyzer [SAS 2003]
  - Minamide's PHP String Analyzer [WWW 2005]
  - Choi/Lee/Kim/Doh's abstract-interpretation approach [APLAS 2006]
  - Doh/Kim/Schmidt's abstract parsing [unpublished]

Minamide's PHP String Analyzer



1) Extract a context-free grammar from the program.
2) Transform the reference grammar into a regular grammar by restricting the nesting depth of recursion.
3) See if the context-free grammar includes the reference grammar.

Improvement in dealing with string operators

```
x = "a";
while <cond> do
  x = "00" + x;
  x = x + "1";
print replace("00","0",x);
```

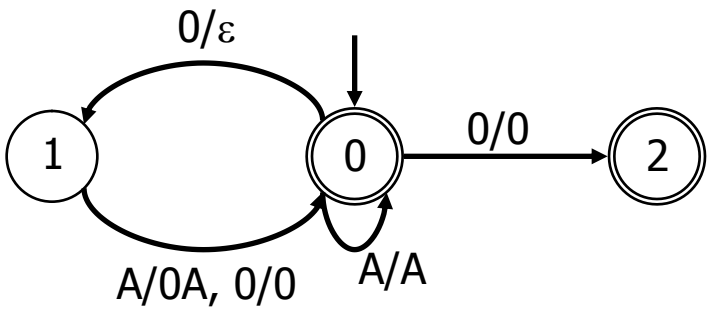
1)

$X_0 \rightarrow$ "a"
$X_1 \rightarrow X_0 \mid X_3$
$X_2 \rightarrow$ "00" X_1
$X_3 \rightarrow X_2$ "1"
$X_4 \rightarrow$ replace("00","0",X_1)

the image under the transducer

$S \rightarrow$ "a" | "0" S "1"

A transducer for replace("00","0",_)

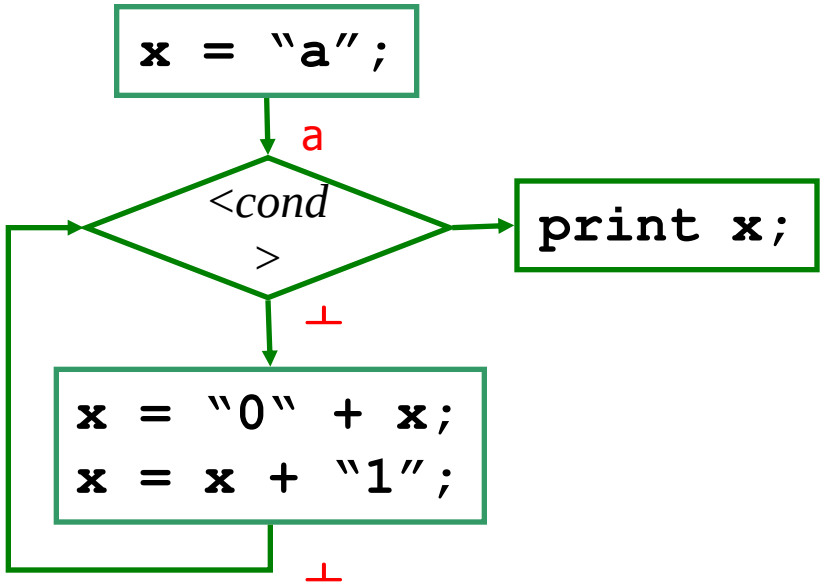


"Static" String Analysis

- Approximates the value of string expression with a grammar.
- History
  - XDuce: A statically typed XML processing language [Hosoya/Pierce, WebDB 2000]
  - Christensen/Moeller/Schwartzbach's Java String Analyzer [SAS 2003]
  - Minamide's PHP String Analyzer [WWW 2005]
  - Choi/Lee/Kim/Doh's abstract-interpretation approach [APLAS 2006]
  - Doh/Kim/Schmidt's abstract parsing [unpublished]

Abstract-Interpretation Approach

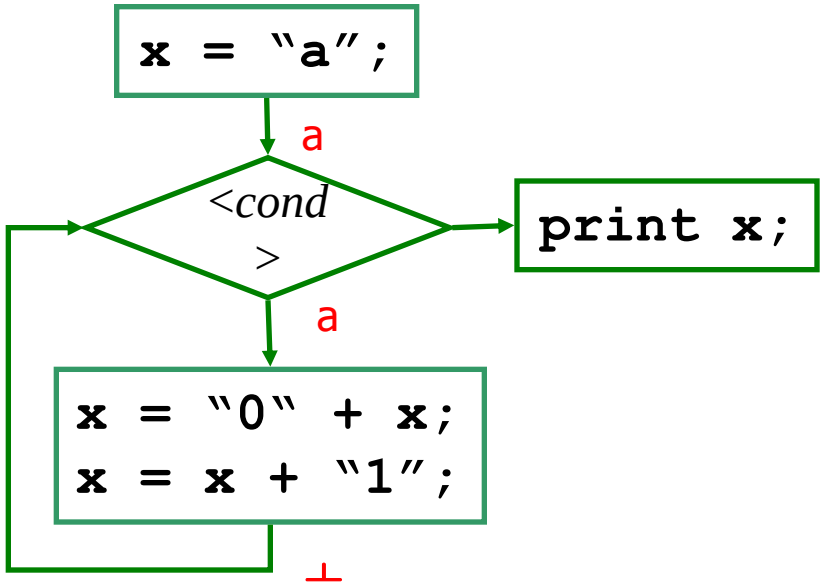
```
x = "a";
while <cond> do
  x = "0" + x;
  x = x + "1";
print x;
```



1) A string value is abstracted as a "regular string" which is the same as a regular expression except that a consecutive repetition is not allowed, i.e., 0*1* $\Rightarrow$ {0,1}*
2) Abstract interpretation of a program on the abstract domain of regular strings.

Abstract-Interpretation Approach

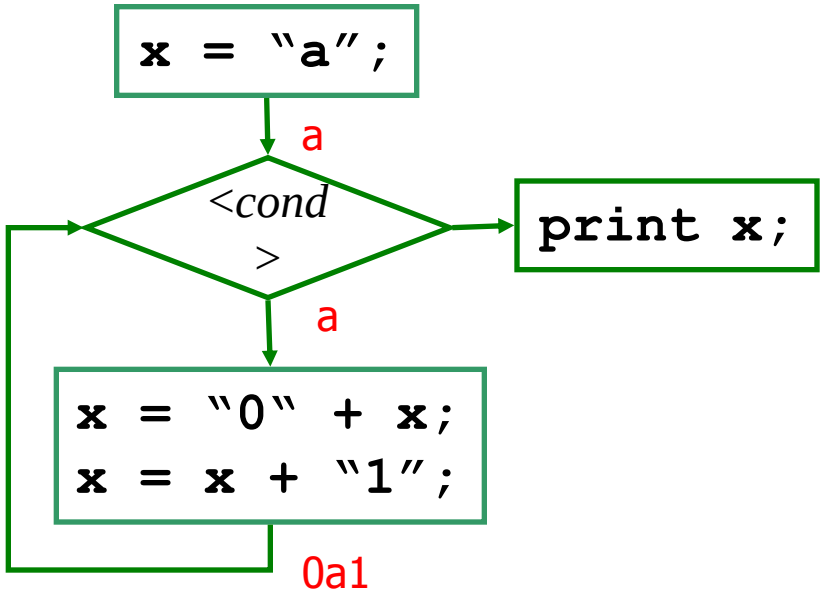
```
x = "a";
while <cond> do
  x = "0" + x;
  x = x + "1";
print x;
```



1) A string value is abstracted as a "regular string" which is the same as a regular expression except that a consecutive repetition is not allowed, i.e., 0*1* $\Rightarrow$ {1,0}*
2) Abstract interpretation of a program on the abstract domain of regular strings.

Abstract-Interpretation Approach

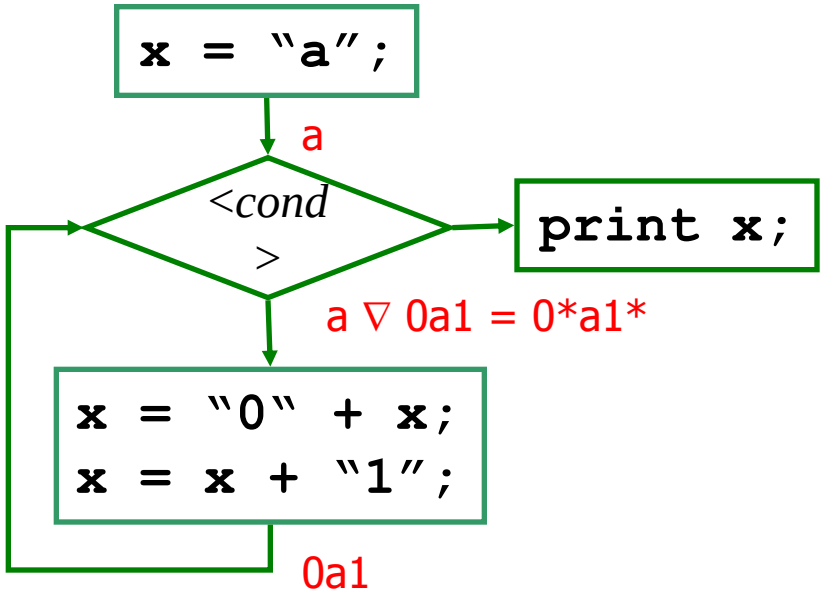
```
x = "a";
while <cond> do
  x = "0" + x;
  x = x + "1";
print x;
```



1) A string value is abstracted as a "regular string" which is the same as a regular expression except that a consecutive repetition is not allowed, i.e., 0*1* $\Rightarrow$ {0,1}*
2) Abstract interpretation of a program on the abstract domain of regular strings.

Abstract-Interpretation Approach

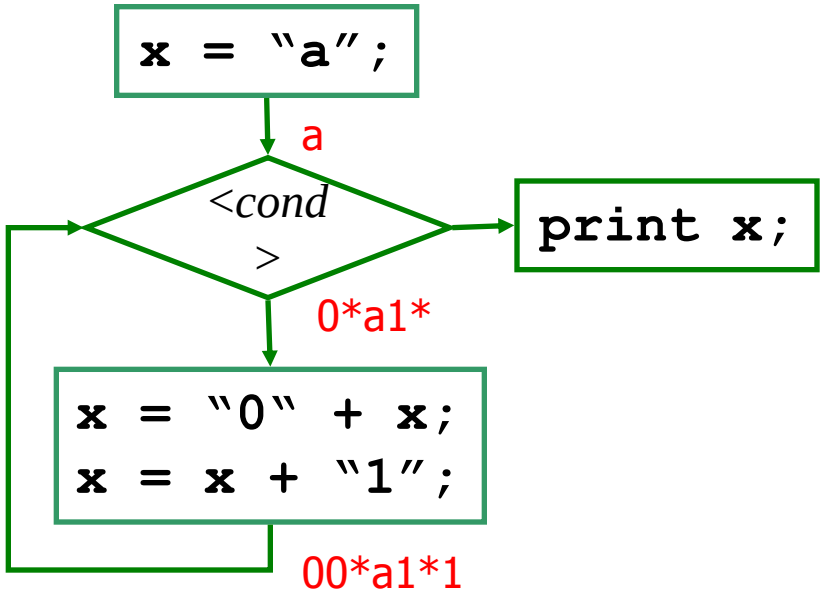
```
x = "a";
while <cond> do
  x = "0" + x;
  x = x + "1";
print x;
```



1) A string value is abstracted as a "regular string" which is the same as a regular expression except that a consecutive repetition is not allowed, i.e., 0*1* $\Rightarrow$ {0,1}*
2) Abstract interpretation of a program on the abstract domain of regular strings.

Abstract-Interpretation Approach

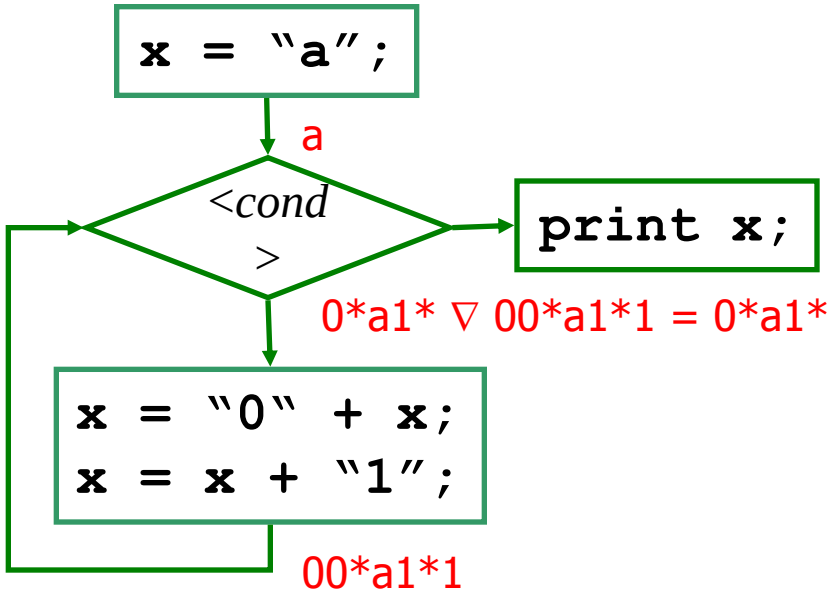
```
x = "a";
while <cond> do
  x = "0" + x;
  x = x + "1";
print x;
```



1) A string value is abstracted as a "regular string" which is the same as a regular expression except that a consecutive repetition is not allowed, i.e., 0*1* $\Rightarrow$ {0,1}*
2) Abstract interpretation of a program on the abstract domain of regular strings.

Abstract-Interpretation Approach

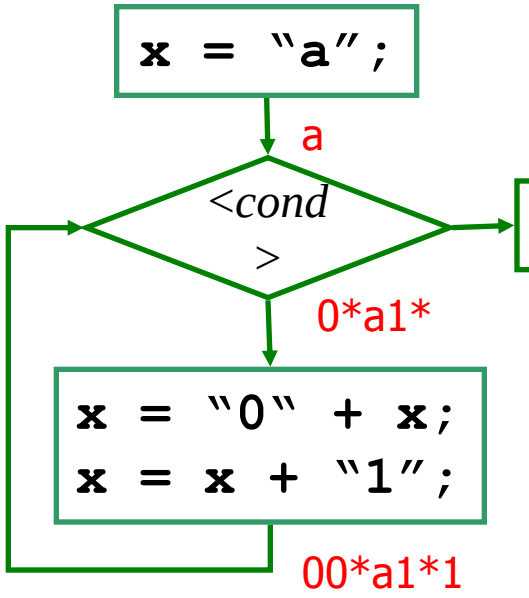
```
x = "a";
while <cond> do
  x = "0" + x;
  x = x + "1";
print x;
```



1) A string value is abstracted as a "regular string" which is the same as a regular expression except that a consecutive repetition is not allowed, i.e., 0*1* $\Rightarrow$ {0,1}*
2) Abstract interpretation of a program on the abstract domain of regular strings.

Abstract-Interpretation Approach

```
x = "a";
while <cond> do
  x = "0" + x;
  x = x + "1";
print x;
```



0*a1* ⊔

S → "a" | "0" S "1"

1) A string value is abstracted as a "regular string" which is the same as a regular expression except that a consecutive repetition is not allowed, i.e., 0*1* ⇒ {0,1}*
2) Abstract interpretation of a program on the abstract domain of regular strings.

"Static" String Analysis

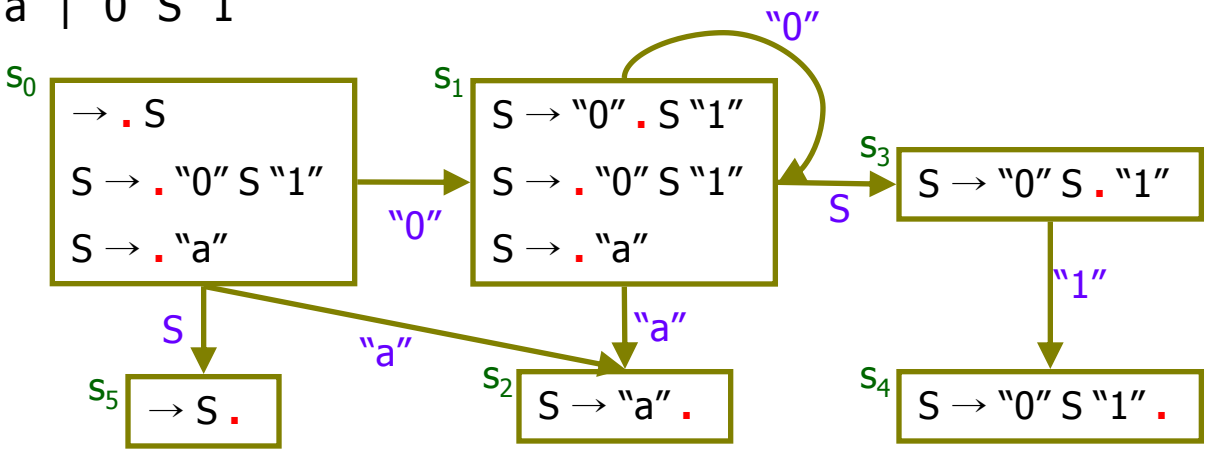
- Approximates the value of string expression with a grammar.
- History
  - XDuce: A statically typed XML processing language [Hosoya/Pierce, WebDB 2000]
  - Christensen/Moeller/Schwartzbach's Java String Analyzer [SAS 2003]
  - Minamide's PHP String Analyzer [WWW 2005]
  - Choi/Lee/Kim/Doh's abstract-interpretation approach [APLAS 2006]
  - Doh/Kim/Schmidt's abstract parsing [unpublished]

Abstract Parsing

- Simultaneous analysis-and-parsing
  - Statically analyze a program that generates strings, and, at the same time, parse the generated strings with the LR(k) reference grammar
- Abstracted parse-stack as the abstract denotations of strings

LR(k) Parsing

Goto Controller for parser built from LR(0)-items for the reference grammar, S → "a" | "0" S "1"



Parse of input sequence, 00a11

parse stack (top at right)
s_0
s_0 s_1
s_0 s_1 s_1
s_0 s_1 s_1 s_2
s_0 s_1 s_1
s_0 s_1 s_1 s_3
s_0 s_1 s_1 s_3 s_4
s_0 s_1
s_0 s_1 s_3
s_0 s_1 s_3 s_4
s_0
s_0 s_5

input sequence (front at left)	
00a11	shift
0a11	shift
a11	shift
11	reduce: S → "a"
S11	shift
11	shift
1	reduce: S → "0" S "1"
S1	shift
1	shift
	reduce: S → "0" S "1"
S	shift
	(done)

Abstract LR(k) Parsing

Given a string-generating program and

a reference grammar for the string variable at hot spot

- Generate data-flow equations from the program.
- Solve the equations in the abstract domain of parse stacks of the reference grammar.

Ongoing Works

- Implementation: Done
- Works fine with string concatenation operators
- But, not with destructive operators
  - replace, substring, etc.

Abstract String Representation

- Regular Grammar
  - Impact Analysis
  - Security Vulnerability Analysis
  - Metrics
- Context-Free Grammar
  - Syntax Analysis
  - Semantics Analysis

Future Directions

- Better Understanding of String Analysis
  - analysis-then-compare vs. analysis-and-parse
  - improve the precision of string analysis
  - understand the abstract semantics of generated strings
- String-Processing Domain-Specific Languages
- Applications
  - Tools for software development
  - Tools for software maintenance
  - Tools for automatic detection of software vulnerabilities

The End