Analysis and Transformation of File processing Programs

M. Raveendra Kumar

raveendra.kumar@tcs.com

Guide

Raghavan Komondoor

raghavan@csa.iisc.ernet.in

Indian Institute of Science, Bangalore, India

File Processing Programs - Context

- File processing programs process data from ***sequential files***.
- Need for tool support in analysis and transformation tasks such as:
  - Bug detection, program understanding.
  - Service extraction, batch to online conversion etc.
- Key challenges.
  - Lack of abstractions and modularity.
  - Large size and evolved over a period.
- **Our research goal** is to extend low-level building blocks of tools to file processing programs.
  - Program specialization, slicing, and Symbolic execution.

Summary of Key contributions

1. Novel static analysis approach for
   - Program specialization
   - File format conformance checking

   [In Int. Conf. on Software Maintenance and Evolution (ICSME), 2015.]

2. Approximate inter-procedural (static) analysis using Prefix call strings
   - Improves scalability
   - Maintains precision at application level modules

   [In Int. Conf. on Software Analysis, Evolution, and Reengineering (SANER), 2015.]

3. Automated crash testing to detect buffer overflow errors.
   - Generates test cases that can expose buffer overflow vulnerabilities.

Novel static analysis approach for

1. Program specialization and
2. File format conformance checking

Describing Files- Input file format specification

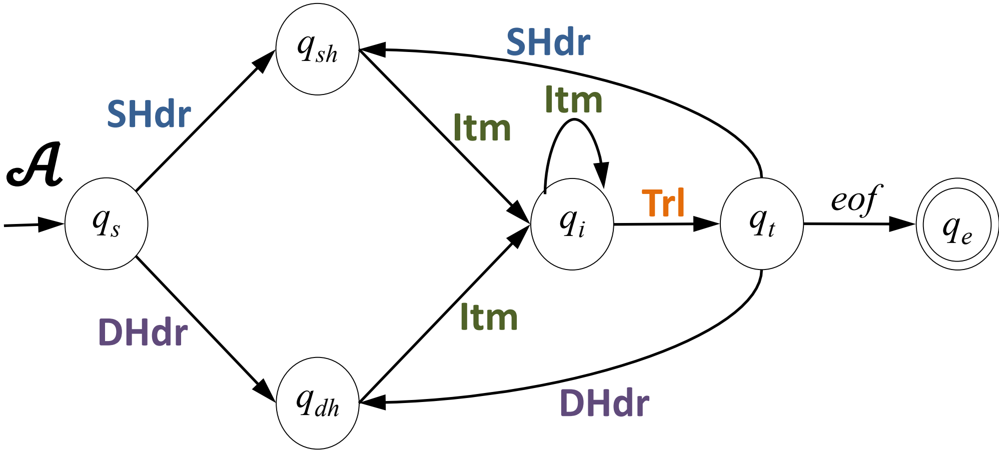
Input file

typ: HDR	pyr: 10205	amt: 9000	src: SAME
ITM	rcv: 10201	amt: 3000	
ITM	10103	4000	
ITM	18888	2000	
TRL			
HDR	20221	4000	DIFF
ITM	10234	4000	
TRL			

Record types

SHdr
Itm
Itm
Itm
Trl
DHdr
Itm
Trl
eof

File format automaton

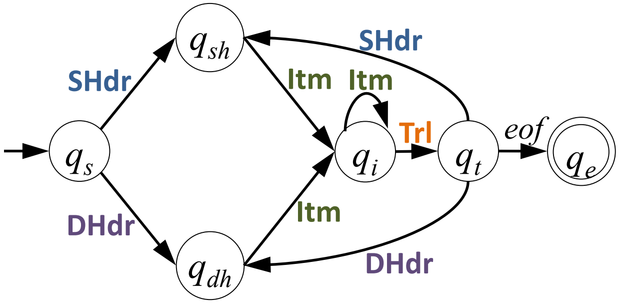


A file *f* ***conforms*** to a file format automaton, *iff* the sequence of types of the records in *f* takes the file format automaton from ***start state*** to some ***final state***.

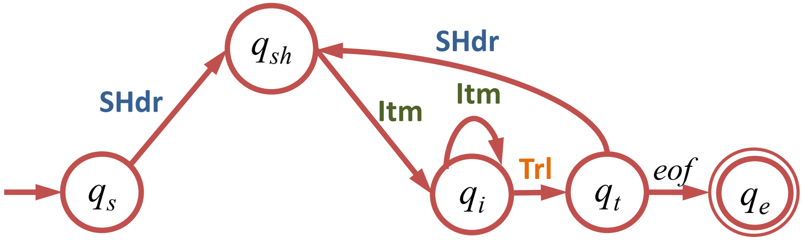
Program Specialization Problem

- What portion of a program is relevant to a ***functionality***?

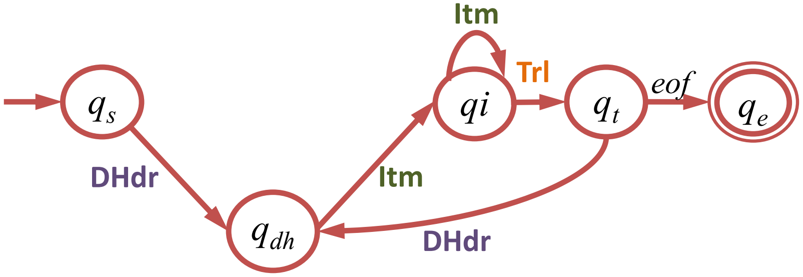
A ***specialization automaton*** is a sub-automaton of a file format automaton that accepts a subset of files



File format specification for **same bank** functionality

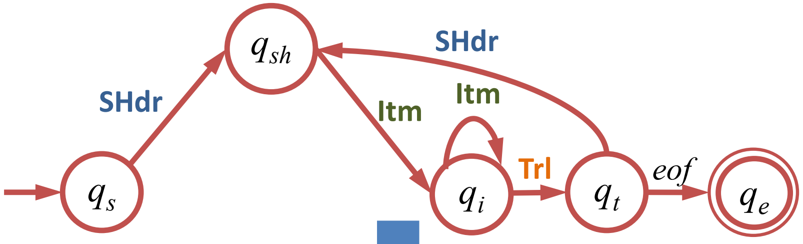


File format specification for **diff bank** functionality



Key observation : In file processing programs, different restricted input files trigger different functionality in a program. Therefore, each specialization automaton represents different functionality.

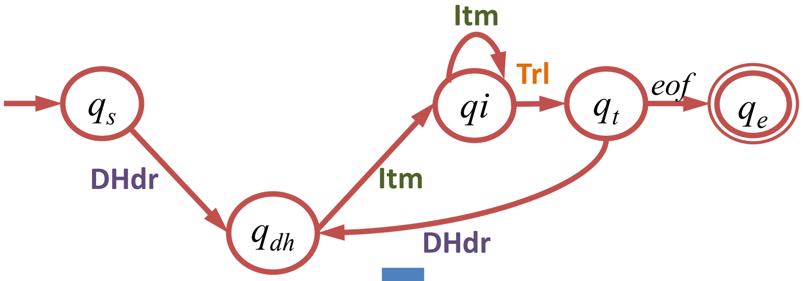
Program Specialization



```
/1/void processrec(rec[] inp){
/2/ int A[5], B[5],rec r = inp[0];
/3/ int iA =0, iB=0, i = 0;
/4/ while(r != Eof){
/5/   if(r.typ == ‘HDR’ )
/6/    if(r.src == ‘SAME’)
/7/       A[iA++] = r.Amt;
/8/    else if(r.src == ‘DIFF’)
/9/      B[iB++] = r.Amt;
/10/  else if
/11/      . . . .
/12/   i++;
/13/ r = inp[i];
/14/ }
/15/}
```

Program Specializer Tool

```
/1/void processrec(rec[] inp){
/2/ int A[5], B[5],rec r = inp[0];
/3/ int iA =0, iB=0, i = 0;
/4/ while(r != Eof){
/7/       A[iA++] = r.Amt;
/12/   i++;
/13/ r = inp[i];
/14/ }
/15/}
```



```
/1/void processrec(rec[] inp){
/2/ int A[5], B[5],rec r = inp[0];
/3/ int iA =0, iB=0, i = 0;
/4/ while(r != Eof){
/5/   if(r.typ == ‘HDR’)
/6/    if(r.src == ‘SAME’)
/7/       A[iA++] = r.Amt;
/8/    else if(r.src == ‘DIFF’)
/9/      B[iB++] = r.Amt;
/10/  else if
/11/      . . . .
/12/   i++;
/13/ r = inp[i];
/14/ }
/15/}
```

Program Specializer Tool

```
/1/void processrec(rec[] inp){
/2/ int A[5], B[5],rec r = inp[0];
/3/ int iA =0, iB=0, i = 0;
/4/ while(r != Eof){
/9/      B[iB++] = r.Amt;
/10/  else if
/11/      . . . .
/12/   i++;
/13/ r = inp[i];
/14/ }
/15/}
```

Key Results – Program specialization

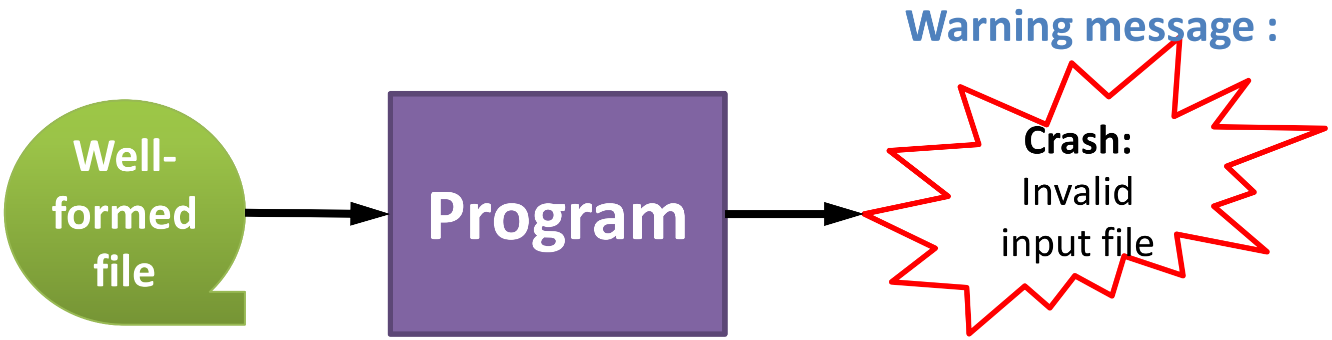
S.NO	Program	# in Main loop	Functionality query	% of lines in specialized program
1	ACCTRAN	43	Deposit	30%
			Withdraw	81%
2	PROG1	410	Edit	23.4%
			Update	33.7%
3	PROG2	236	Form	34.3%
			Telex	33.9%
			Modified	34.4%
4	PROG3	454	TranCopy-1	16.1%
			TranCopy-7	4.6%
5	PROG4	5692	NormalAccounts	52.7%
			SpecialAccounts	92.3%

Number of lines in specialized program : 43*0.3 = 13 lines

Percentage of lines in specialized programs varies from 4.6% to 92.3%

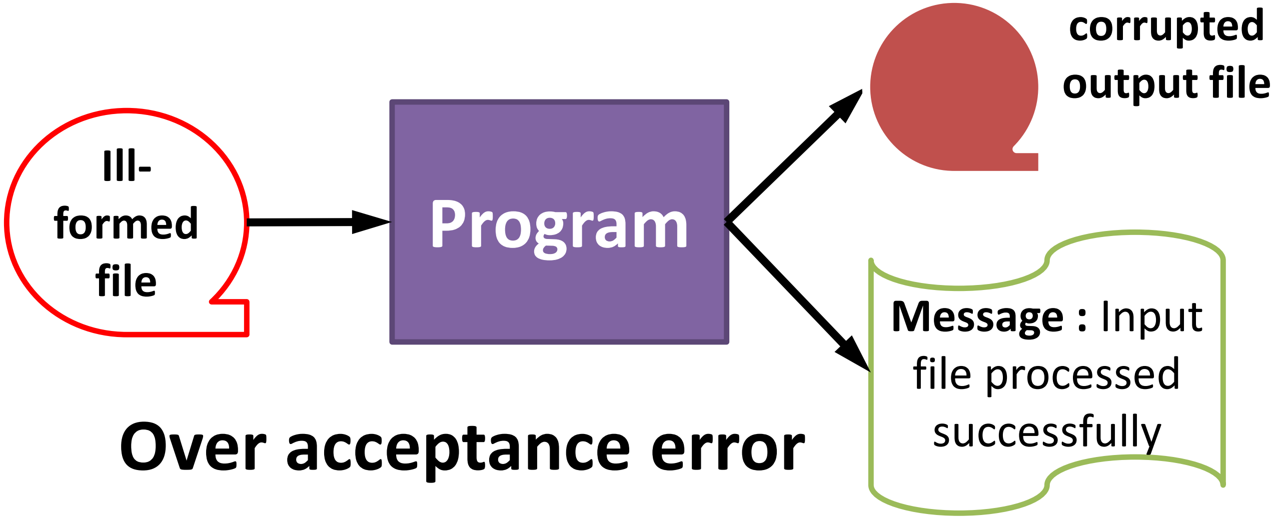
File format conformance

- **Verification Problem :** Does program process file correctly ?
- **Question :** Does program reject any *well-formed* file ?



Under acceptance error

- **Question :** Does program process any *ill-formed* file silently ?



Over acceptance error

Results – File format conformance checking

Program Name	File Conformance warnings	
	Under acceptance	Over acceptance
ACCTRAN	**0**[b]	1[c]
SEQ2000	3[c]	1[c]
DTAP	**0**[b]	**1**[1]
CLIEOPP	**13**[a]	-
PROG1	5	9
PROG2	6	10
PROG3	**0**[b]	1
PROG4	**0**[b]	10*

Our analysis is conservative; i.e., it overstates the number of errors

a) 2 Identified as true positives; i.e. as genuine errors by manual validation (**13**, **1)**.

b) 4 Programs have ***no*** under acceptance errors (**0**).

c) 2 programs Identified as false positives; i.e., not errors (3,1,1).

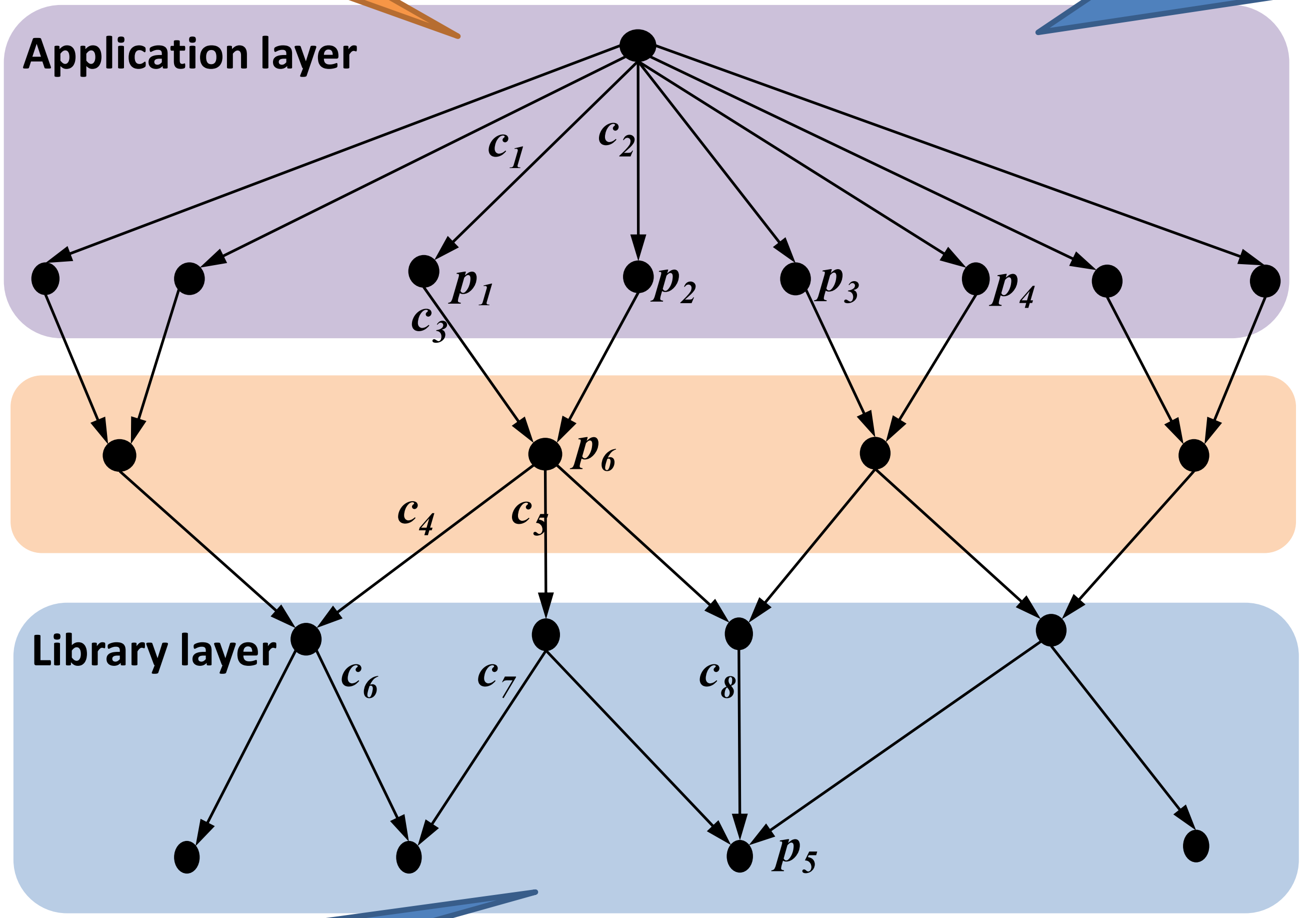
Approximate inter-procedural (static) analysis using Prefix call strings

Inter procedural analysis

Users expect precise analysis results in procedures in their application code

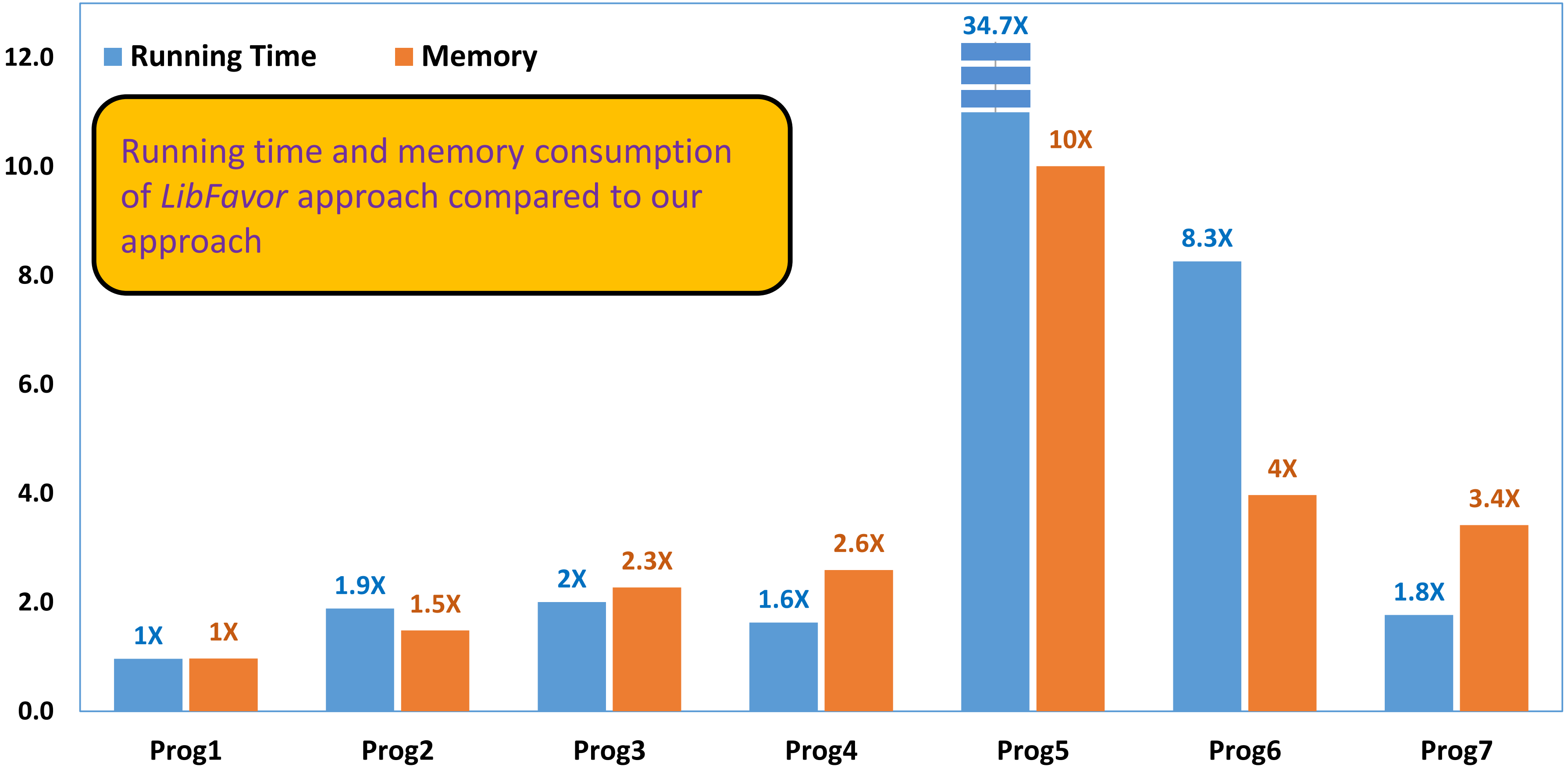
Our approach spends more time in analyzing the procedures in this layer. Therefore more precise in this layer

Precise static analysis is ideal across all parts of the program. But it is expensive and time consuming.



Existing *LibFavor* approach spends more time in analyzing procedures in library layers. Hence precise in this layer.

Key Results – Overall Performance



Running time and memory consumption of *LibFavor* approach compared to our approach

- In ***2*** programs, on an average, *our approach* is **7.5%** more precise than *Libfavor* approach.
- In ***5*** programs, on an average, *LibFavor approach* is **8%** more precise than *our approach*.
- In ***application level*** procedures of all 7 programs, on an average, our approach is 1.3% more precise than LibFavor approach.

Automatic crash testing to detect buffer overflow errors

Problem statement

- Buffer overflow violations are common in programs that read data from files or streams.
  - For instance, certain versions of httpd crash when the URL is more than 2000 characters long.
  - Buffer overflow violations rank 3rd in the list of top 25 most dangerous software errors. (*http://cwe.mitre.org/top25*).

- Our objective is to devise an automated testing tool that tries to generate test inputs that can crash a given program, by analyzing the same program.

An illustrative example

contains characters ‘a’ from input *inp*

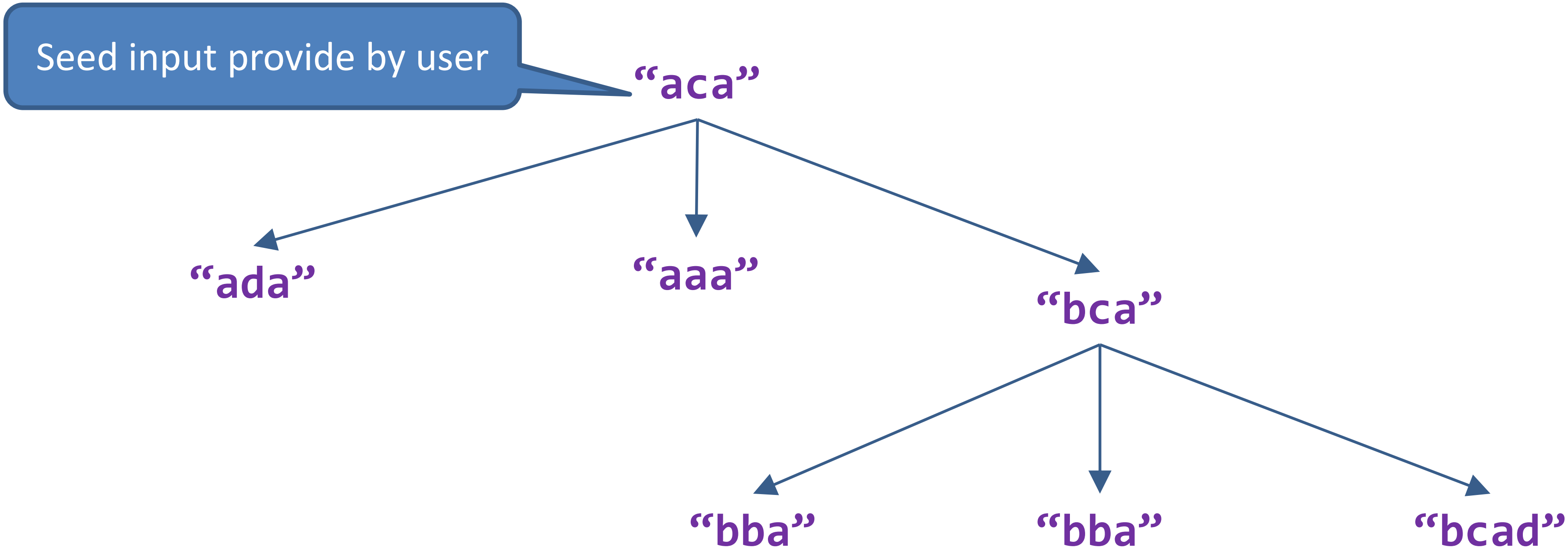
Contains characters ‘b’ from input *inp*

```
void splitstring(char[] inp){
 char A[5], B[5];
 int iA =0, iB=0, i = 0;
 while(inp[i] != ‘\0’){
   if(inp[i] == ‘a’&& iA < 5)
      A[iA++] = inp[i];
   else if(inp[i] == ‘b’)
     B[iB++] = inp[i];
   i++;
  }
}
```

Safe access to this buffer

Unsafe access to this buffer. It can overflow if the input contains more than 5 characters of **‘b’**

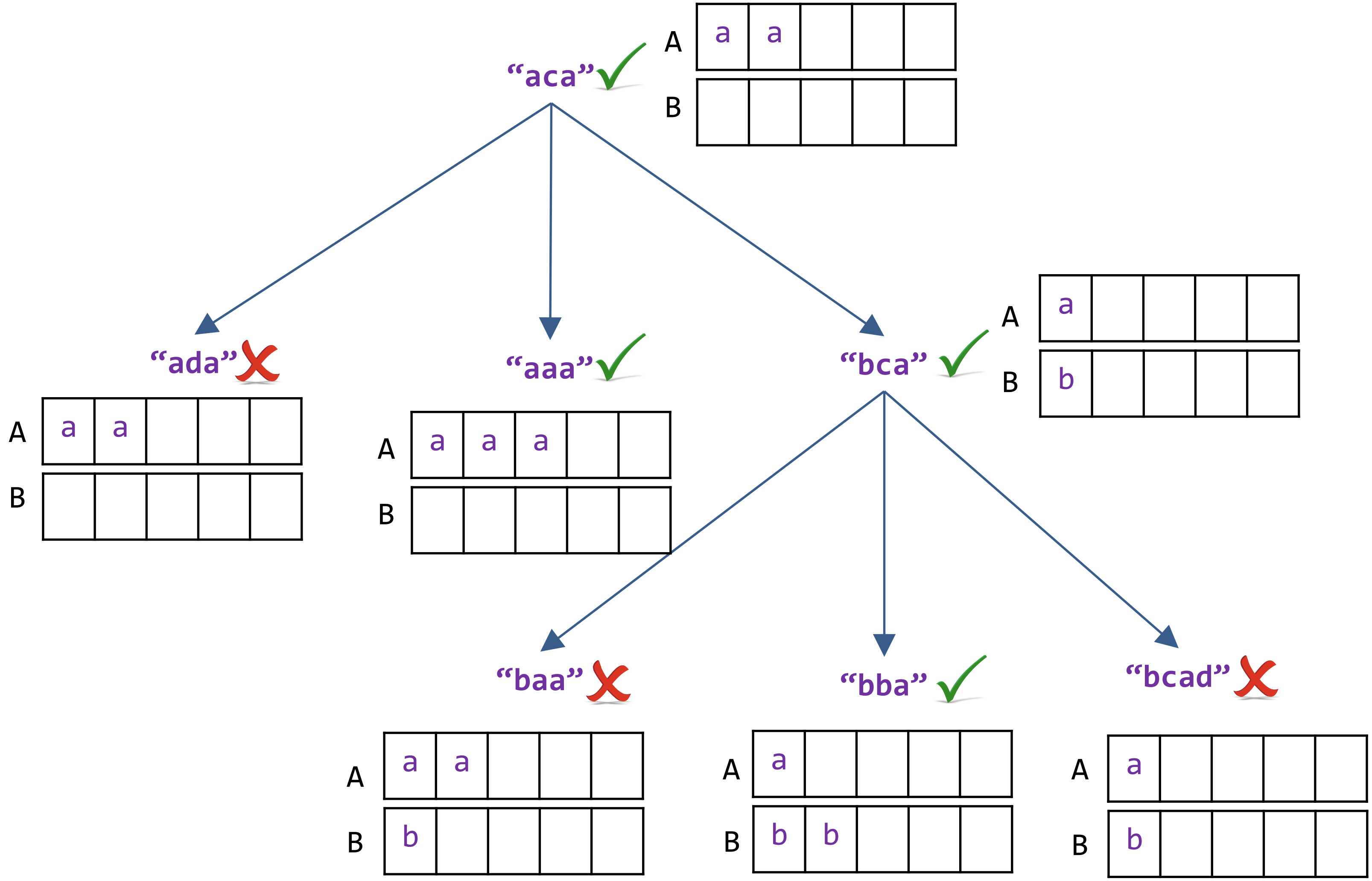
Systematic test case generation



Automatic testing tools can systematically mutate a given seed test input to generate new test inputs. They run the program with each generated test input and decide to keep or discard that test input based on a fitness criteria.

Our approach

- Our approach observes buffers at their access locations in the program, to ***measure extent*** to which they are filled with each muted test input.
- Upon observing this, create more mutant test inputs that ***fill buffers*** to a ***greater*** extent.



Preliminary results on MIT benchmark suite

S.No	MIT benchmark	Program	Potential Buffer overflows	No. Buffer overflows detected	
				Standard fuzzing tool	Our approach
1	sendmail	s1	28	10	23
2	sendmail	s5	3	0	2
3	bind	b4	2	0	2

Summary

- File processing programs play key role in several domains. There is a strong need for tool support for file processing programs to detect bugs and to transform them.
- To this end, our key contribution are :
  - An approach to specialize a file processing program based on an input format specification.
  - An approach to verify file processing programs for absence of file acceptance errors.
  - Improve scalability and maintain the precision of of static analysis by maintaining the analysis information separately at top level procedures in a multi procedural program.
  - Develop an automatic testing tool that can generate crashing inputs for file processing programs.