srcSlice: Very efficient and scalable forward static slicing

Hakam W. Alomari, Michael L. Collard, Jonathan I.Maletic, Nouh Alhindawi, Omar Meqdadi
Journal of Software : Evolution and Process, Volume 26, Issue 11, November 2014.

Presented by
Siddarth Udayakumar

Program Slicing

- Computing program statements.
- Set of statements – program slice.
- Values affected at some point – slicing criterion.
- Debugging, used for model checking, test coverage.
- Originally defined by Mark Weiser.
- Different types of slicing.

Introduction

- Program Slicing – Understand and detect impact of changes
- Variable and location of that variable
- Slice – parts of program affected by variable. Maintenance tasks.
- Slice S(A, 10) – Affecting A in 10.
- Weiser– executable program that preserved behavior of original.
- Based on Program Dependence graph.

Introduction (Contd)

- Eliminate time and Effort to build PDG
- Combination of text approach and lightweight static analysis.
- Slicing – srcML (Source code Markup Language).
- Fast and Scalable but less accurate.
- Efficient with respect to Computational time
- Implemented approach using srcSlice tool.

Forward Slicing

Program	Forward Slice

n = 0;
product = 1;
sum = 1;
scanf("%d",&x) ;
while (x >= 0){
sum = sum + x;
product = product * x ;
n = n + 1;
scanf("%d",&x);
}
average = (sum - 1) / n ;
printf("The total is %d\n",sum) ;
printf("The product is %d\n",product) ;
printf("The average is %d\n",average) ;
	n = 0;
	product = 1;
	sum = 0;
	scanf("%d",&x) ;
	while (x >= 0) {
	sum = sum + x; /* AFFECTED */
	product = product * x ;
	n = n + 1;
	scanf("%d",&x);
	}
	Average = (sum - 1) / n ; /* AFFECTED */
	printf("The total is %d\n",sum) ; /*
	AFFECTED */
	printf("The product is %d\n",product) ;
	printf("The average is %d\n",average) ; /*
	AFFECTED */

Forward slice from point P includes all points in forward control flow affected by computation at P.

Lightweight Forward Static Slicing

- Forward, Static, non executable and inter-procedural program slice for each variable.
- PDG is not computed for entire program, no precise reference location for slicing criterion.
- Forward Slice – Not executable but Binkley et al – Complement executable forward slice.
- Complement forward slice – execution not affected by slicing criterion.
- Approach – follows original convention but extension is used for executable slices.

Lightweight Forward Static Slicing (Contd..)

- Approach relies on XML representation of Source – srcML.
- Augments source code with abstract syntactic information.
- Syntactic info – id program dependencies.
- srcML format – toolkits like src2srcml, scrcml2src.
- Lightweight fact extraction, source code transform, pattern matching.

Extended Decomposition Slicing Criterion

- Slicing criterion – file name, function name, variable name.
- S(f, m, v). Decomposition slice - Gallagher et al.
- Union of collection of slices of individual statements of a variable.
- Adapt it to forward static slicing.
- Slice at statements that define a variable.
- Slice obtained from slicing algorithm – D Slice computed.

Slice Profile and System Dictionary Construction

- Algorithm does not rely on pre computed data and control dependencies.
- Required dependencies for slicing calculated as and when needed.
- Slice Profile – statements from all slices for slicing variable V.
- System dictionary – (f, m, v).

(a)

```
1.  main(){
2.      int sum = 0;
3.      int i = 1;
4.      while (i<=10){
5.          sum = sum + i;
6.          i++;
7.      }
8.      cout<<sum;
9.      cout<<i;
10. }
```

(b) *Slice Profile(sum)= @index(1), slines={2,5,8},*

Slice Profile(i)= @index(2), slines={3,4,5,6,9}, dvars={sum}

Slice Profile Structure for Dictionary

- file, function, and variable names;
- @index, an index of each variable in the order that it was declared in the function;
- slines, a list of lines that comprise the slice;
- cfunctions, a list of functions called using the slicing variable;
- dvariables, a list of variables that are data dependent on the slice variable;
- pointers, a list of aliases of the slicing variable; and
- controledges, a list of all possible control-flow edges of the slicing variable.

Criterion Definition

- Forward Decomposition Slice

$$ds(f, m, v) = \bigcup_{s \in \{s_1, \ldots, s_k\}} sfs_{(v,s)}(p).$$

- General Forward Decomposition Slice

$$mds(f, m) = \bigcup_{i=1}^{d} ds(f, m, v_i).$$

- Decomposition Slice - special case for simultaneous slicing
  - variation of original definition by Weiser

Algorithm Overview

- Slice Profiles for all variables are computed.
- After above step, system dictionary is used to take into account variables, function calls, to generate final slices.
- Algorithm – computation of forward decomposition slice.
- Given Below are the main algorithms used in this paper:
  - ComputeSliceProfile – Used for intra-procedural slicing
  - ComputeInterProcedural – Inter-procedural slicing.
  - ComputeControlPaths – generate control flow paths for slices.

Computing Control Flow Paths

- Facilitates handling of unstructured control flow.
- Our approach – no control flow information stored during slice production.
- Storage causes computational overhead so used when required.
- Generate CF paths -extending ComputeSliceProfile – sequence info between lines.
- Two control flow edges for each block
  - (predecessor, true successor)
  - (predecessor, false successor)

Control Flow Paths

(a)

```
1.  main(){
2.      int sum = 0;
3.      int i = 1;
4.      while (i<=10){
5.          sum = sum + i;
6.          i++;
7.      }
8.      cout<<sum;
9.      cout<<i;
10. }
```

(b)

Slice Profile(sum)= @index(1), slines={2,5,8},

Slice Profile(i)= @index(2), slines={3,4,5,6,9}, dvars={sum}

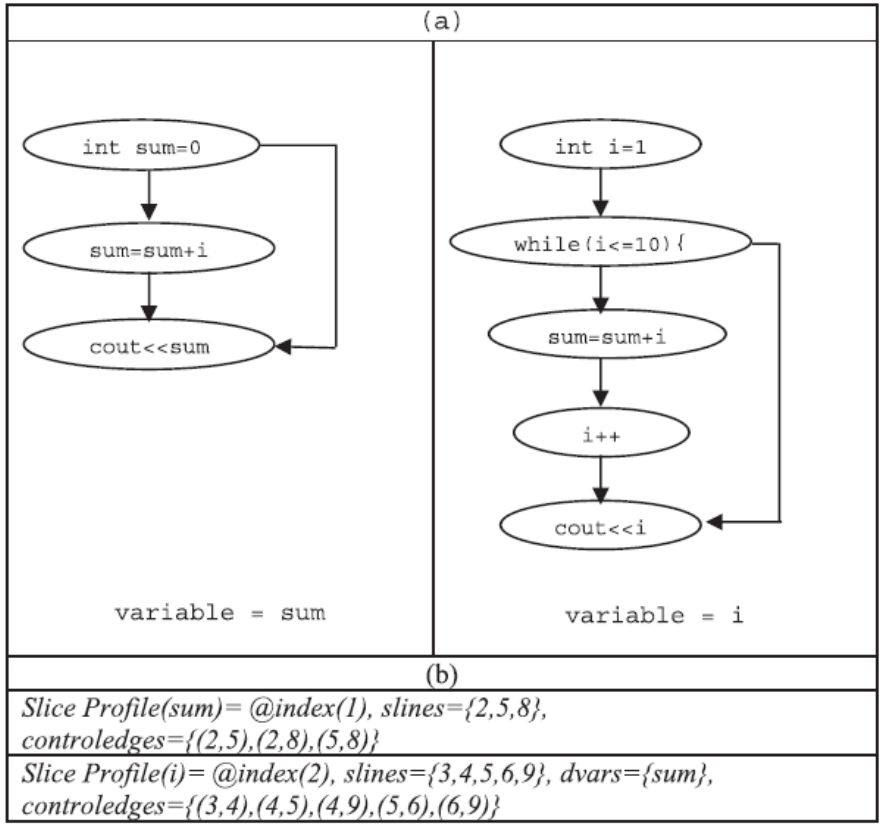

Figure 5. Control-flow path computation for the code in Figure 2. (a) Control-flow edges (b) system dictionary with slice profiles including control-flow edges shown in (a).

Pointer Analysis

- Keep track of aliases in the slice profile.
- Slice of aliased variables VP is added to slice of variable V for final computation.
- Direct pointer aliases and call by references are supported.

Implementation.

- Implemented approach as srcSlice tool. C++ code. SrcML as input.
- src2srcML is applied on program to convert to srcML.
- Source code – unprocessed. Conversion is fast (28KLOC/s).
- srcSlice run on input using Qt XML Parser.
- Pull XML processor produces the srcML in node order as per source code.
- Pull approach – memory conservative and straightforward.

Implementation (Contd)

- srcSlice – slice profile and System dictionary as outputs.
- Change sets – expressed as set of lines.
- Computation of slice based on lines – line slicing.
- Slice profile only for variable of interest.
- Line slicing granularity.
  - Prune all lines prior to specified variable.
  - No pruning at all.
- System dictionary = O(cn). Complete closure in constant time.

Limitations

- No analysis of dynamic binding through virtual functions.
- Does not fully support all overloading situations.
- Support for unstructured control flow is not completely accurate.
- Exceptions are not supported.
- No full type resolution support w.r.t pointer analysis. Complex cases of pointers are not supported.

Comparative study

- Approach assessed by comparing srcSlice with CodeSurfer.
- CodeSurfer – commercial based slicing tool for C & C++ programs.
- Slices measured by accuracy and size.
- To prove, considered approach is highly scalable and efficient compared to existing one.
- CodeSurfer – Gold standard.
- Feature and performance benchmarks.

CodeSurfer

- CodeSurfer – generates CFG for each source file.
- SDG is constructed for system as whole.
- Slicing – graph reachability problem.
- Dependencies – control dependence, edge dependence or both.
- Different settings for precision and build time.
- Highest setting – more precise results

Limitations of CodeSurfer

- Incomplete source code can’t be sliced.
- Programs > 200 KLOC not sliced because of free license.

  • Get a list of all program points in the PDG.
  • Perform slicing using the nodes from the list.
  • Calculate the number of program points in the slice.
  • Output the size of the slice into a specified file.
  • Repeat from steps 2 to 4 until end of list.

Evaluation Criteria

- Evaluated on execution time & memory requirements.
- Accuracy of slices.
- Slice size and quality.
- Slice size – measure by SLOC.
- Safe slice, minimal slice.
- Intersection of slices returned by both tools.
- srcSlice slice ~= Intersected Slice.

Study Results

- Feature Benchmark – accuracy calculation
- Performance Benchmark – run time performance

- RQ1 : Does srcSlice produce Accurate slices
- RQ2 : is SrcSlice highly efficient and scalable?

- Tools run on same machine (Standard PC with 4GB of RAM).

Feature Benchmark

- Tools ran on set of small programs.
- Included various situations such as function calls, global and local variables, pointer casting, external libraries.
- Statistics :
  - Size of program in LOC.
  - Size of program as both file and function.
  - Slice size relative to LOC.

- **Information_flow:** pointers, pointer casting, double pointers, data and control dependencies, global variables, function indirection.
- **Sum and Wc:** external libraries.
- **Pointer:** pointer flow.
- **Callofcall:** nested function calls.
- **Testcases:** functions calls, local and global variables, call by reference, calling built-in/library functions, dependence flow.

Feature Benchmark

Table I. Feature benchmark results and comparison of *CodeSurfer* and *srcSlice*.

	Size			Slicing criterion		*CodeSurfer*				*srcSlice*			
Program	LOC	Files	Functions	Method	Variable	Slices taken	Slicing time	Slice size	%	Slices taken	Slicing time	Slice size	%
Information _flow	112	1	12	Main	hi	1	1.481	32	28.6	1	0.978	27	24.1
				All criterions		149		66	58.9	22		48	42.9
Sum	21	1	2	Main	Sum	1	0.989	6	28.6	1	0.531	4	19.0
				All criterions		26		14	66.7	2		8	38.1
Wc	39	1	3	line_char_count	eof_flag	1	1.212	16	41.0	1	0.362	10	25.6
				Scan_line	i	1		7	17.9	1		4	10.3
				All criterions		46		24	61.5	9		19	48.7
Pointer	36	1	5	Main	var1	1	1.519	11	30.6	1	0.358	15	41.7
				All criterions		37		25	69.4	8		17	47.2
Testcases	114	1	14	Main	var1	1	7.662	50	43.9	1	0.641	44	38.6
				All criterions		156		79	69.3	24		56	49.1
Callofcall	24	1	3	Main	var1	1	2.921	4	16.7	1	0.411	4	16.7
				All criterions		23		13	54.2	7		10	41.7
Total	346	6	39			444	15.78	347		79	3.28	266	
Average	57.7	1	6.5			34.2	2.63	26.7	45.2	6.1	0.55	20.5	34.1

The slicing time measured in seconds and includes converting to srcML, slice size measured in number of statements. The percentage (%) columns are the slice size relative to LOC (lines of code).

Performance Benchmark

- Statistics :
  - Size of program in LOC.
  - Size of program as both file and function.
  - Slice size relative to LOC.
- Slices – number of forward slices for all possible criterions.
- ~1810 KLOC of C/C++ code from 20 open source projects.

Performance Benchmark

Table II. Performance benchmark results and comparison of *CodeSurfer* and *srcSlice*.

	Size			*CodeSurfer*					*srcSlice*				
Program	LOC	Files	Functions	Slices taken	Time highest	Time Super-Lite	Slice size	%	Slices taken	Time highest	Time Super-Lite	Slice size	%
ed-1.2	3087	10	126	4438	21	5	1782	57.7	420	3	1	1136	36.8
ed-1.6	3260	10	128	4527	19	5	1863	57.1	442	3	1	1199	36.8
which-2.20	3586	14	51	1429	15	9	736	20.5	196	2	1	754	21.0
wdiff-0.5	3874	13	56	1097	11	4	652	16.8	154	2	1	581	15.0
barcode-0.98	5205	18	74	4590	29	9	2177	41.8	392	3	1	1728	33.2
acct-6.5	8749	27	127	4983	47	11	2510	28.7	635	5	1	1767	20.2
enscript-1.4.0	18,162	52	180	9456	71	15	5916	32.6	920	7	2	3472	19.1
make-3.82	36,397	58	474	17,012	807	27	9446	26.0	2914	143	3	10,598	29.1
libkate 0.3.8	53,441	111	2210	28,017	109	33	12,846	24.0	4993	17	6	12,622	23.6
enscript-1.6.5.2	56,491	107	488	20,234	184	35	12,907	22.8	2580	90	5	10,635	18.8
enscript-1.6.5	56,494	107	488	20,252	184	35	12,913	22.9	2579	85	5	10,636	18.8
enscript-1.6.5.1	56,494	107	488	20,252	185	35	12,913	22.9	2579	85	5	10,636	18.8
a2ps-4.10.4	57,052	188	1104	24,493	393	116	14,249	25.0	3844	61	6	10,756	18.9
findutils-4.4.2	72,384	314	1141	23,641	215	94	13,689	18.9	5657	79	9	16,764	23.2
radius-1.0	82,029	196	1719	38,487	335	148	19,218	23.4	6997	170	11	21,905	26.7
dico-2.2	119,592	332	2504	52,297	1763	246	28,639	23.9	10,451	471	16	29,418	24.6
cvs-1.12.10	144,278	340	2027	74,328	286,328	371	40,869	28.3	9779	192	18	34,955	24.2
Clanlib -0.8.1	181,064	1137	5624	68,716	280,914	344	45,447	25.1	13,514	58	21	30,488	16.8
HippoDraw-1.21.3	248,592	1385	8627	—	—	377	—	—	12,400	66	29	25,976	10.5
Quantlib 0.9.7	599,802	3389	15,696	—	—	384	—	—	27,348	168	99	53,476	8.9
Total	1,810,033	7915	43,332	418,249			238,772		69,046			210,050	
Average	90,502	396	2167	23,236	31,757	115	13,265	28.8	3836	86	12	11,055	22.4

The slicing time is measured in seconds and includes the conversion to srcML. The slice size is measured in number of statements. The percentage (%) columns are the slice size relative to LOC (lines of code). The times for *CodeSurfer* are given for both the Highest and Super-lite settings. At the highest setting, *CodeSurfer* was unable to slice programs in our study over 200 KLOC. All the *CodeSurfer* and *srcSlice* results, regarding slice taken and slice size, are from using the Highest setting because the Super-lite setting gives poor accuracy. The last two systems are not included in the totals and averages for *srcSlice* or *CodeSurfer*.

Slice Intersection Comparison

- Intersected slice – measure of quality.
- Results ~= intersected slice, more accuracy.
- Codesurfer – Higher Safety Margin.
- srcSlice results consistently closer to intersection slices.

Slice Intersection Comparison

Table III. Comparison of *CodeSurfer* and *srcSlice* to the intersected slice over 13 files from enscript-1.6.5.

	Size		Slice size						Intersection		
			CodeSurfer			*srcSlice*					
enscript-1.6.5 File name	LOC	SLOC	LOC	%	Safety margin	LOC	%	Safety margin	LOC	*CodeSurfer* intersection %	*srcSlice* intersection %
src/psgen.c	2860	1993	1351	67.8	1.75	863	43.3	1.12	771	57.1	89.3
src/util.c	2156	1623	1227	75.6	1.48	853	52.6	1.03	827	67.4	97.0
src/main.c	2660	1406	1178	83.8	1.59	768	54.6	1.04	739	62.7	96.2
src/mkafmmap.c	250	153	92	60.1	2.04	45	29.4	1.00	45	48.9	100.0
afmlib/strhash.c	386	268	145	54.1	1.36	145	54.1	1.36	107	73.8	73.8
afmlib/afmparse.c	1017	759	636	83.8	2.05	313	41.2	1.01	310	48.7	99.0
states/ex.c	2378	1536	813	52.9	3.35	279	18.2	1.15	243	29.9	87.1
states/gram.c	2408	1607	433	26.9	2.41	301	18.7	1.67	180	41.6	59.8
afmlib/afm.c	824	590	468	79.3	1.31	357	60.5	1.00	357	76.3	100.0
afmlib/afmtest.c	184	113	67	59.3	1.60	42	37.2	1.00	42	62.7	100.0
afmlib/deffont.c	379	323	311	96.3	8.18	38	11.8	1.00	38	12.2	100.0
afmlib/e_88594.c	284	261	190	72.8		0	0.0		0	0.0	0.0
afmlib/e_mac.c	284	261	219	83.9		0	0.0		0	0.0	0.0
Total	16,070	10,893	7130			4004			3659		
Average	1236	838	548	69.0	2	308	32.4	1	281	52.8	91.1
Min	184	113	67	26.9	1.31	0	11.8	1	0	12.2	59.8
Max	2860	1993	1351	96.3	8.18	863	60.5	1.67	827	76.3	100

The percentage (%) in the *CodeSurfer* and *srcSlice* columns (under slice size) is the slice size relative to the individual lines of code (LOC). The intersection percentage (%) is the slice size for each tool relative to the intersection.

Slice Intersection Comparison

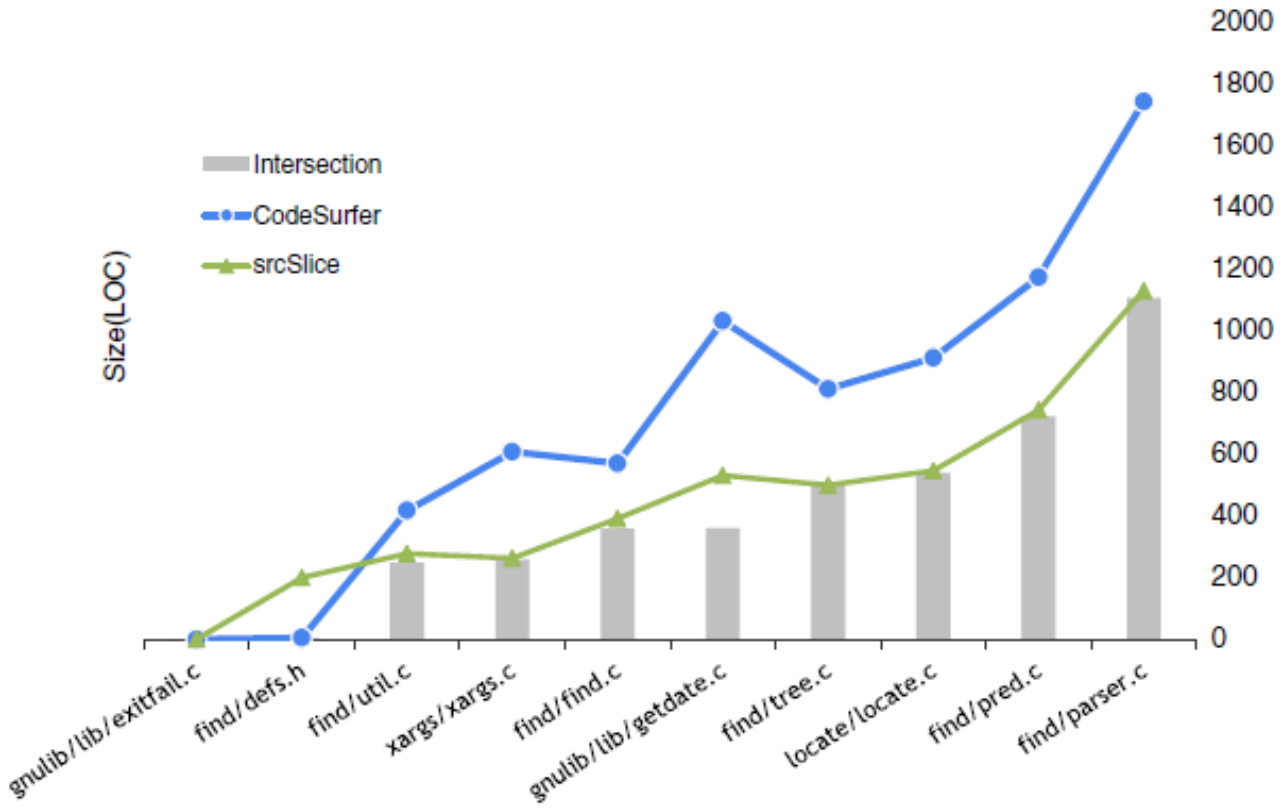

Figure 8. Comparison of *CodeSurfer*, *srcSlice*, and *slice intersection* over 10 files from the program findutils-4.4.2 ordered by intersection size. Except for a single file, *srcSlice* was much closer to the slice intersection.

SrcSlice Scalability

- srcSlice over Linux Kernel
- Slicing criterion – (F, M, V).
- 748s. 1, 334, 504 slice profiles. ~4MLOC
- 974 versions of Linux kernels sliced.
- Slice profile for each individual variable. O/P -> System dictionary

srcSlice Scalability

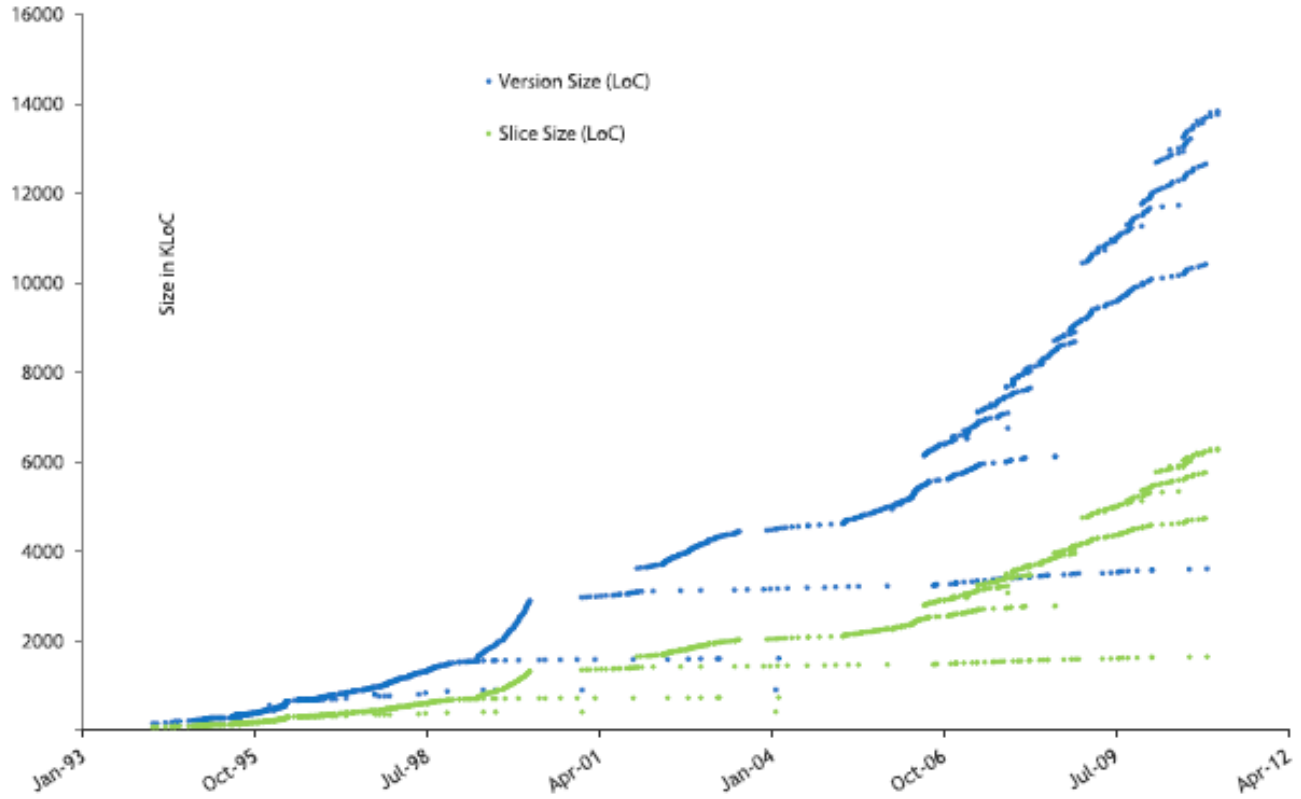

Figure 9. A comparison of the size of the *srcSlice*'s slices to the size of the Linux versions measured in MLOC, the *x*-axis is the version date.

Related Work

- Binkley et al – different implementations
- Harman, Silvia et al – compare slicing techniques
  - Static vs Dynamic
  - Closure vs Executable
  - Inter procedural vs Intra procedural
  - Forward vs Backward
  - Conditioned, Amorphous, Union & Quasi Static.
- Criterion – static, dynamic, hybrid.

Related Work (Contd)

- Approach in paper – does not depend on SDG/PDG.
- No Graph to traverse. No data flow to solve.
- Slice calculated for every local and global variable.
- Scalable in terms of time and program size.
- Approach based on and extension of Gallagher’s decomposition Slice.
- Adaption to forward static slicing.

Conclusion

- Lightweight approach – applicable to both inter procedural and intra procedural slices.
- Cannot be used as a complete alternative for CodeSurfer.
- Used in cases where time and money is of the essence.
- Efficient for large systems' slices in short time.
- Re engineering and refactoring.
- srcSlice offered as part of srcML infrastructure.

Questions

- Have you heard of program slicing earlier? If yes, have you used it in your projects?
- Could backward slicing technique be used in place of forward slicing as per the paper’s approach?

References

- http://www.cc.gatech.edu/~harrold/6340/cs6340_fall2009/Slides/BasicAnalysis6.pdf
- http://www0.cs.ucl.ac.uk/staff/mharman/exe1.html
- https://github.com/srcML/srcSlice
- https://en.wikipedia.org/wiki/Program_slicing
- Papers referenced at the glossary.

Thank you!