Unions of Slices are not Slices

Andrea De Lucia
Research Centre on
Software Technology,
University of Sannio, Italy
delucia@unisannio.it

Mark Harman & **Robert Hierons**
Department of Information
Systems and Computing,
Brunel University, UK
mark.harman@brunel.ac.uk

Jens Krinke
Lehrstuhl Softwaresysteme,
Universität Passau,
Germany
krinke@fmi.uni-passau.de

Abstract

Many approaches to slicing rely upon the 'fact' that the union of two static slices is a valid slice. It is known that static slices constructed using program dependence graph algorithms are valid slices [19]. However, this is not true for other forms of slicing. For example, it has been established that the union of two dynamic slices is not necessarily a valid dynamic slice [8]. In this paper this result is extended to show that the union of two static slices is not necessarily a valid slice, based on Weiser's definition of a (static) slice. We also analyse the properties that make the union of different forms of slices a valid slice.

1. Introduction

Many approaches to slicing rely upon the ability to union the results of two or more slices to produce a slice which agglomerates the results of each individual slice. For example, in decomposition slicing [7], the decomposition slice (and its complement) can be expressed as a union of static slices, each of which shares the same variable, but differs upon the point for which it is constructed. The parallel algorithm of Danicic et al. [5] explicitly relies upon union of slicing. Other, well-known algorithms for static slicing, such as Weiser's [23] and the HRB inter-procedural algorithm [11] also, implicitly rely upon the 'fact'[1] that the union of two slices on two different criteria is a slice on the union of the two criteria (which we call the 'distributive law' in this paper). This has only been proved for program dependence graph based slicing of programs without procedures [19].

Also, in the application of slicing algorithms to software engineering problems, the approaches often rely, either explicitly or implicitly upon the belief that the union of two slices is a valid slice. For example, Canfora et al. [4] use the union of slices to identify reusable functions and in the work of Bieman and Ott [2] and Ott and Thuss [16, 17], slicing is used to assess the functional cohesion in a function. The approach is essentially to define metrics which derive their cohesion score from the level of overlap between the 'important' slices of the function. This overlap represents the portion of the function which contributes to all the important variables and thus can be thought of as the cohesive part. Of course, this overlap is an intersection, but other metrics, such as coverage [16] rely upon union, as does work on coupling [9], which is derived from and inspired by the original work on cohesion measurement.

The reason all these approaches safely use the union of slices is due to the fact that the algorithm used to build a slice preserves control and data flow dependences of the original program (most of them are indeed based on the Program Dependence Graph, PDG [6]). Indeed, according to Horwitz *et al.*, two PDG slices of the same program can be seen as two non-interfering versions of the program which can be safely integrated [10]. However, despite these important results, we show that the union of static slices is *not necessarily* a valid slice, based on Weiser's definition of slice.

Unioning of slices is not merely relied upon in static slicing, it is also used in the construction of dynamic slices and in the approximation of the more precise 'realizable' static slice, expressed as a union of dynamic slices [1]. However, although PDG based algorithms enable the valid union of two static slices of a program, unioning is not valid for other forms of slicing. For example, it has been established that the union of two dynamic slices is not necessarily a valid dynamic slice [8]. In this paper we also identify further conditions for the validity of unioning other forms of slices, such as dynamic and conditioned slices.

The remainder of the paper is organized as follows. In Section 2 we show that according to Weiser's definition, the union of static slices is not a valid slice, while Section 3 discusses the condition for the validity of unioning static slices. Section 4 discusses validity conditions for unioning other forms of slices, while concluding remarks and future

[1] which this paper demonstrates is questionable.

work directions are outlined in Section 5.

2. Unioning slices

Weiser has formally defined a slice as any subset of a program, which preserves a specific behavior in respect to a criterion. The criterion, also called the *slicing criterion*, is a pair $c = (s, V)$ consisting of a statement s and a subset V of the variables of the analyzed program.

Definition 1 (Weiser-style Slice) *A slice $Slice(c)$ of a program P on a slicing criterion c is any executable program P', where*

1. *P' is obtained by deleting zero or more statements from P,*
2. *whenever P halts on a given input I, P' will halt for that input, and*
3. *P' will compute the same values as P for the variables of V on input I.*

The most trivial (but irrelevant) slice of a program P is always the program P itself. Slices of interest are as small as possible, hopefully minimal.

In this section we show that the union of two static slices is not a static slice. This is the case, both for two slices constructed for different criteria and also for two different valid slices, constructed for the same criterion, each of which satisfies Weiser's definition. For the sake of simplicity, we use the end of the program as the statement part of slicing criteria. This does not affect the validity of our considerations.

Consider the example in Figure 1. In this example, two slices are constructed for the final value of the variable x. That is, each slice is constructed by statement deletion and each must preserve the final value of the variable x. Of course, a *particular* algorithm for slicing would only produce a single (unique) slice for a single criterion. However, Weiser's definition allows for many possible valid slices. In this example, both slices are minimal (minimal slices are not unique [22]). Observe that the union of the two slices is not a slice on x.

This result is interesting from a theoretical point of view, but is less important practically speaking, because, any deterministic algorithm[2] would only construct a single slice for a single criterion and therefore there would be no practical ramifications from this observation.

However, by a similar argument, it can be shown that the union of two static slices for two different criteria is also not a static slice for the union of the two criteria; slicing is not distributive. More formally, the law

$$Slice(P, V \cup W) = Slice(P, V) \cup Slice(P, W)$$

[2]All the currently published algorithms for static slicing [20] *are* deterministic.

	P	P_1	P_2	$P_3 = P_1 \cup P_2$
1	x = 2;	x = 2;	x = 2;	x = 2;
2	x = x + 1;	x = x + 1;		x = x + 1;
3	y = x;			
4	x = 2;			
5	x = x + 1;		x = x + 1;	x = x + 1;
6	y = x;	y = x;	y = x;	y = x;

Figure 1. Two slices with the same criterion.

	P	P_1	P_2	$P_3 = P_1 \cup P_2$
1	x = 2;	x = 2;	x = 2;	x = 2;
2	x = x + 1;	x = x + 1;		x = x + 1;
3	y = x;	y = x;		y = x;
4	x = 2;			
5	x = x + 1;		x = x + 1;	x = x + 1;
6	z = x;		z = x;	z = x;

Figure 2. Two slices with two criteria.

does not hold, according to Weiser's definition. This is surprising, because this law is widely believed in the 'folklore' of slicing and is implicitly and explicitly relied upon in many approaches to slicing.

Consider the example in Figure 2. In this example, two minimal slices are constructed for two different variables, y and z, but the union of these two slices is *not* a slice on $\{y, z\}$.

3. Unioning static slices

The problem embodied by these examples is that the constructed slices do not take into account the data dependences. That is, in both examples, statement 5 is data dependent on statement 4 and not on statement 2. The data flow from statement 2 is *killed* before it reaches statement 5, but the killing statement is not included in either slice. Therefore a dependence is inserted into the unioned code, which is not present in the original program.

Fortunately, all the approaches that make use of union of static slices rely on slicing algorithms that do preserve a subset of the direct data and control dependence relations of the original program. For example, in Figure 3 both slices on variables y and z are dependence preserving and the resulting union is a valid slice.

Reps and Yang have proved that a slice in a procedureless program computed by a dependence graph based algorithm is a valid slice [19]. Also, according to Horwitz *et al.* two program dependence graph based slices of the same program can be seen as two non-interfering versions of the program and then can be safely integrated [10]. Kumar and

	P	P_1	P_2	$P_3 = P_1 \cup P_2$
1	x = 2;	x = 2;		x = 2;
2	x = x + 1;	x = x + 1;		x = x + 1;
3	y = x;	y = x;		y = x;
4	x = 2;		x = 2;	x = 2;
5	x = x + 1;		x = x + 1;	x = x + 1;
6	z = x;		z = x;	z = x;

Figure 3. Preserving slices for two criteria.

	P	P_1	P_2	$P_3 = P_1 \cup P_2$
1	read (n);			
2	x = 1;	x = 1;		x = 1;
3	y = 2;		y = 2;	y = 2;
4	if (n == 1)			
5	y = 1;			
6	if (y == 2)		if (y == 2)	if (y == 2)
7	x = 2;		x = 2;	x = 2;

Figure 4. Dynamic slices for two criteria.

Horwitz [14] present a modified definition of a slice based on *semantic effects* (similar to the *semantic dependence* of [18]), which basically inverts the definition of a slice: A statement x of program P has a semantic effect on a statement y, iff a program P' exists, created by modifying or removing x from P, and some input I exists such that P and P' halt on I and produce different values for some variables used at y. This definition does not allow 'problematic' slices like in Figure 1 or 2. However, such a slice may be a superset of a Weiser's slice because a statement like "`x = x;`" has a semantic effect according to their definition.

4. Unioning other forms of slices

In static slicing, the union of two dependence-preserving slices constructed for different criteria can only augment the slice of either. That is, the union of the two slices is simply an unnecessarily large slice of each of the contributing slicing criteria. By contrast, the union of two dynamic slices [13], even where they are dependence-preserving, can interfere, to alter the semantics of one of the two slices. Consider the example in Figure 4: the slice P_1 is constructed with respect to variable x and input $n = 1$, while slice P_2 is constructed with respect to variable x and input $n = 2$. The union of the two slice P_3 fails to preserve the semantics of the program with respect to input $n = 1$ and then cannot be considered a valid dynamic slice for this input. Hall [8] noticed this[3] and proposed a method called simultaneous dynamic slicing to compute dynamic slices that simultaneously preserve the semantics of the original program for all the different input of the contributing slices.

Indeed, the problem with the union of dynamic slices is that they belong to different execution traces (one for each input) and definitions on an execution trace might kill definitions of a different execution trace. For example, in Figure 4 the definition of x at statement 7 in the union slice kills the definition of x at statement 2 on any input, while in the original program this does not happen for input $n = 1$.

It is worth noting that the problem derives from the fact that there is a dynamic dependence between statement 5 and statement 6 on the execution trace for input $n = 1$ and not on the execution trace for input $n = 2$. As statement 6 only affects the computation of x on the execution trace for input $n = 2$, neither slice P_1, nor slice P_2 include statement 5. In this way, in the union slice the data dependence between statements 5 and 6 on the execution trace for input $n = 1$ is lost, thus resulting in an erroneous result for the variable x. It is also important to remark that statement 5 transitively depends on statement 1 that defines the value of the input variable n. Such dependences are lost in current dynamic slicing algorithms whenever the dependent statement does not affect the computation of the variable of interest and then is lost in the union of the dynamic slices too.

The same problem is likely to affect other forms of slicing, such as quasi-static slicing [21] and conditioned slicing [3], where slices are constructed with respect to a subset of the execution traces[4].

Therefore, to build valid unions for forms of slices computed with respect to subsets of execution traces, we need to consider other properties than just preserving program dependences. It is likely that the union is a valid slice if the two slices are constructed with respect to the same subset of execution traces. For example, we should get valid union slices of two dependence preserving dynamic slices constructed with respect to two different variables but the same input, or of two conditioned slices constructed with respect to the same condition on the input variables.

5. Conclusions/future work

This short paper raises some questions about set operations on slicing, in particular various forms of union of slices. It has shown that the union of two static slices is not (necessarily) a valid static slice. This complements the work by Hall [8], on simultaneous slicing which shows that the union of dynamic slices is not a dynamic slice.

[3]The contribution of our example, is that it is a much simpler demonstration of the problem than the example used by Hall.

[4]In quasi-static slicing the subset of execution traces is identified by assigning a value to a subset of the input variables, while in conditioned slicing a condition on the input variables is used.

Despite our results showing that the union of static slices is not necessarily a valid static slice, current PDG based static slicing algorithms (which rely upon the union of slices) are correct. It is not the same for other forms of slices computed with respect to a subset of the execution traces. For example, the union of two dynamic slices is not necessarily a valid dynamic slice, even if the slices preserve the program dependences. An approach to get valid union slices is to propose algorithms that take into account simultaneously the execution traces of the slicing criteria, as in the simultaneous dynamic slicing algorithm proposed by Hall [8].

It is worth noting that the goal of some authors [1] is to union dynamic slices to achieve an approximation of a static slice, rather than a slice that preserves the semantics of the original program on the different program input used to build the single dynamic slices. Although, the proposed approach is interesting from a performance point of view (the union slice can be constructed in a much faster and cheaper way than a static slice), more experimental work is required to see the effects of using union slices that do not preserve the semantics of the original program in software maintenance tasks, such as program comprehension.

The considerations expressed for the union of dynamic slices also applies to other forms of slices, such as conditioned slices and quasi-static slices. For example, it is likely that a conditioned slice computed on the disjunction of the conditions of two conditioned slicing criteria is a valid slices, but the union of the corresponding conditioned slices is not necessarily a valid one. Therefore, it remains a problem for future work to *demonstrate* that the particular algorithmic manner in which existing slicing algorithms union slices, leads to unions which turn out always to be valid slices, themselves.

The paper also suggests some other questions about set operations on slices, which the authors would like to encourage the slicing community to consider, for example:

1. Is the union of two dataflow minimal slices [15, 22] a dataflow minimal slice and, if not, is it even a slice?

2. Do results for backward slicing also apply to forward [11] slicing?

3. Is the union of chops [12] defined by $Chop(s_1,t_1) \cup Chop(s_2,t_1) \cup Chop(s_1,t_2) \cup Chop(s_2,t_2)$ a valid chop on the union of the chopping criteria $Chop(s_1 \cup s_2, t_1 \cup t_2)$?

References

[1] Á. Beszédes, C. Faragó, Z. M. Szabó, J. Csirik, and T. Gyimóthy. Union slices for program maintenance. In *International Conference on Software Maintenance (ICSM'02)*, pages 12–21, 2002.

[2] J. M. Bieman and L. M. Ott. Measuring functional cohesion. *IEEE Transactions on Software Engineering*, 20(8):644–657, Aug. 1994.

[3] G. Canfora, A. Cimitile, and A. De Lucia. Conditioned program slicing. In M. Harman and K. Gallagher, editors, *Information and Software Technology Special Issue on Program Slicing*, volume 40, pages 595–607. Elsevier Science B. V., 1998.

[4] G. Canfora, A. Cimitile, A. D. Lucia, and G. A. D. Lucca. Slicing large programs to isolate reusable functions. In *EUROMICRO Conference*, pages 140–147, 1994.

[5] S. Danicic, M. Harman, and Y. Sivagurunathan. A parallel algorithm for static program slicing. *Information Processing Letters*, 56(6):307–313, Dec. 1995.

[6] J. Ferrante, K. J. Ottenstein, and J. D. Warren. The program dependence graph and its use in optimization. *ACM Transactions on Programming Languages and Systems*, 9(3):319–349, July 1987.

[7] K. B. Gallagher and J. R. Lyle. Using program slicing in software maintenance. *IEEE Transactions on Software Engineering*, 17(8):751–761, Aug. 1991.

[8] R. J. Hall. Automatic extraction of executable program subsets by simultaneous dynamic program slicing. *Automated Software Engineering*, 2(1):33–53, Mar. 1995.

[9] M. Harman, M. Okunlawon, B. Sivagurunathan, and S. Danicic. Slice-based measurement of coupling. In R. Harrison, editor, 19^{th} *ICSE, Workshop on Process Modelling and Empirical Studies of Software Evolution*, Boston, Massachusetts, USA, May 1997.

[10] S. Horwitz, J. Prins, and T. Reps. Integrating non–interfering versions of programs. *ACM Transactions on Programming Languages and Systems*, 11(3):345–387, July 1989.

[11] S. Horwitz, T. Reps, and D. W. Binkley. Interprocedural slicing using dependence graphs. *ACM Transactions on Programming Languages and Systems*, 12(1):26–61, 1990.

[12] D. Jackson and E. J. Rollins. A new model of program dependences for reverse engineering. In *Proceedings of the ACM SIGSOFT '94 Symposium on the Foundations of Software Engineering*, pages 2–10, Dec. 1994.

[13] B. Korel and J. Laski. Dynamic program slicing. *Information Processing Letters*, 29(3):155–163, Oct. 1988.

[14] S. Kumar and S. Horwitz. Better slicing of programs with jumps and switches. In *Proceedings of FASE 2002: Fundamental Approaches to Software Engineering*, volume 2306 of *LNCS*, pages 96–112. Springer, 2002.

[15] M. R. Laurence, S. Danicic, M. Harman, R. M. Hierons, and J. Howroyd. Equivalence of conservative, linear, free program schemas is decidable. *Theoretical Computer Science*. to appear.

[16] L. M. Ott and J. J. Thuss. The relationship between slices and module cohesion. In *Proceedings of the* 11^{th} *ACM conference on Software Engineering*, pages 198–204, May 1989.

[17] L. M. Ott and J. J. Thuss. Slice based metrics for estimating cohesion. In *Proceedings of the IEEE-CS International Metrics Symposium*, pages 71–81, Baltimore, Maryland, USA, May 1993. IEEE Computer Society Press, Los Alamitos, California, USA.

[18] A. Podgurski and L. A. Clarke. A formal model of program dependences and its implications for software testing, debugging and maintenance. *IEEE Trans. Softw. Eng.*, 16(9):965–979, Sept. 1990.

[19] T. Reps and W. Yang. The semantics of program slicing. Technical Report Technical Report 777, University of Wisconsin, 1988.

[20] F. Tip. A survey of program slicing techniques. *Journal of Programming Languages*, 3(3):121–189, Sept. 1995.

[21] G. A. Venkatesh. The semantic approach to program slicing. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 26–28, Toronto, Canada, June 1991. Proceedings in *SIGPLAN Notices*, 26(6), pp.107–119, 1991.

[22] M. Weiser. *Program slices: Formal, psychological, and practical investigations of an automatic program abstraction method*. PhD thesis, University of Michigan, Ann Arbor, MI, 1979.

[23] M. Weiser. Program slicing. *IEEE Transactions on Software Engineering*, 10(4):352–357, 1984.