Program Slicing

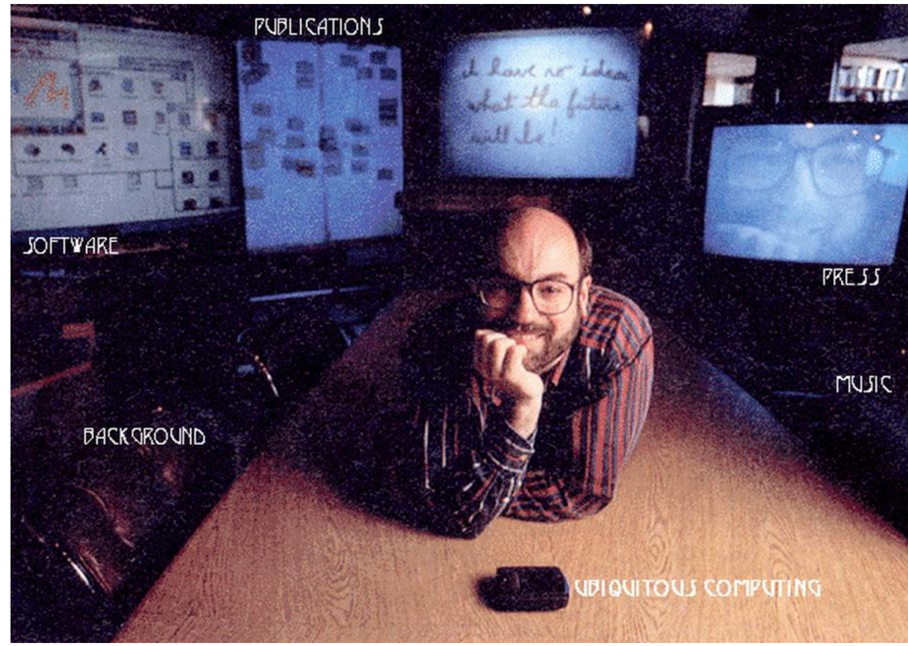
Author: Mark Weiser

Published in TSE, 1984

Presented by Peeratham (Karn) Techapalokul
10/13/2015

About Mark Weiser

- a chief scientist at Xerox PARC
- Widely considered to be the father of ubiquitous computing



“My research interests are garbage collection, operating systems, user interfaces, and ubiquitous computing. I used to work on software engineering and program slicing, but not much any more.” –Weiser

Outline

- Definitions
- Finding slice (manually)
- Applications of Program slicing
- Finding slice using data flow analysis (Intraprocedural slicing)
- Interprocedural slicing
- Testing the slicer on student compiler programs

What is Program Slicing?

- A **program slice S** is a ***reduced, executable program*** obtained from a program P by removing statements, such that the program slice S replicates part of the behavior of program P .

- **Program slicing** is the computation of a set of program statements (**the program slice)** that can possibly affects the values at some points of interest (**slicing criterion**)

Applications of program slicing:

- **Debugging**
- **Parallelization**
- **Software maintenance**
- **Testing**
- **Reverse engineering**
- **Compiler tuning**
- **Security**

Finding Slices

```
(1)  read (n);
(2)  i := 1;
(3)  sum := 0;
(4)  product := 1;
(5)  while i <= n do
     begin
(6)          sum := sum + i;
(7)          product := product * i;
(8)          i := i + 1
     end;
(9)  write(sum);
(10) write(product)
```

Slicing criterion C = < statement, variables >

< 10 , {product}>

Finding Slices

(1) read (n);
(2) i := 1;
(3) ~~sum := 0;~~
(4) product := 1;
(5) while i <= n do
begin
(6) ~~sum := sum + i;~~
(7) product := product * i;
(8) i := i + 1
end;
(9) ~~write(sum);~~
(10) write(<u>product</u>)

Slice on criterion < 10 , {product}>

```
read (n);
i := 1;
product := 1;
while i <= n do
begin
    product := product * i;
    i := i + 1
end;
write(product)
```

CFG of the example program

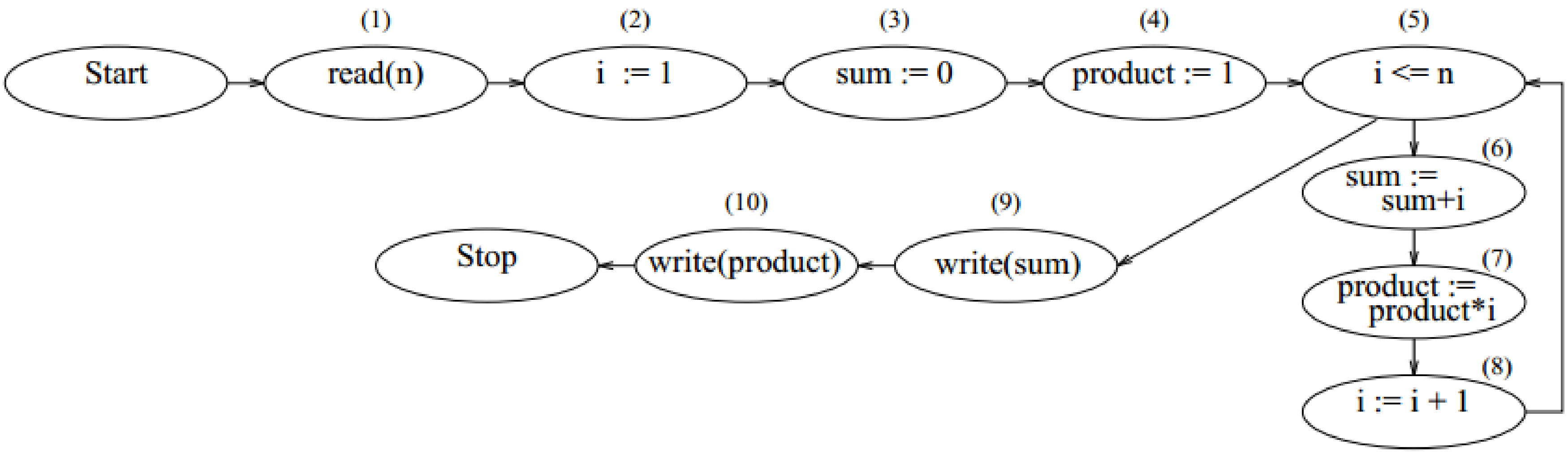


Figure 3: CFG of the example program of Figure 1 **(a)**.

Overview: finding program slices

Two **types of iteration:**

1. Tracing transitive data dependences
    - determine directly relevant variables $\boldsymbol{R_C^0}$
    - derive $\boldsymbol{S_C^0}$ from $\boldsymbol{R_C^0}$

2. Tracing control dependences (dealing with branch statement)

- INFL (b) : set of statements control dependent on b

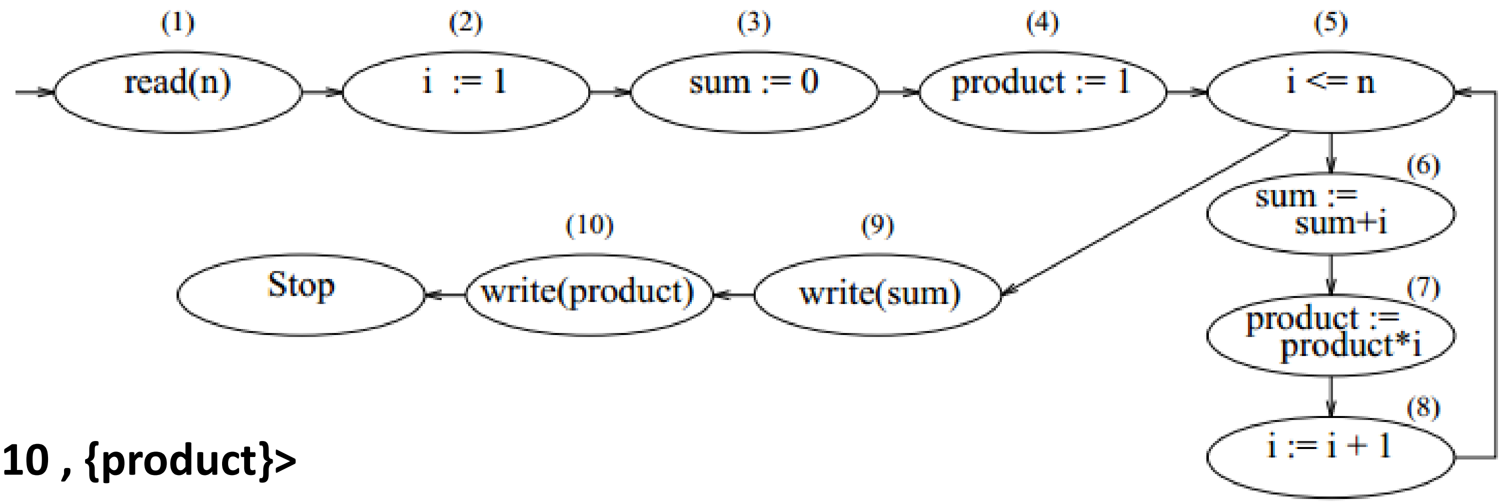
```
1  READ (X)
2  IF X < 1
3              THEN Z := 1
4              ELSE  Z := 2
5  WRITE (Z).
```

Tracing transitive data dependences (1)

1. Determine ***directly relevant variables*** $\boldsymbol{R_C^0(i)}$ at each node i in the CFG
   - starts with initial values $R_C^0(n)$ = V, $R_C^0(m)$ = ∅ for any node m≠n

$$R_C^0(i) = R_C^0(i) \cup \{\, v \mid v \in R_C^0(j),\ v \notin \mathrm{DEF}(i) \,\}$$
$$\cup \{\, v \mid v \in \mathrm{REF}(i),\ \mathrm{DEF}(i) \cap R_C^0(j) \neq \emptyset \,\}$$

   - requires iteration in the presence of loop



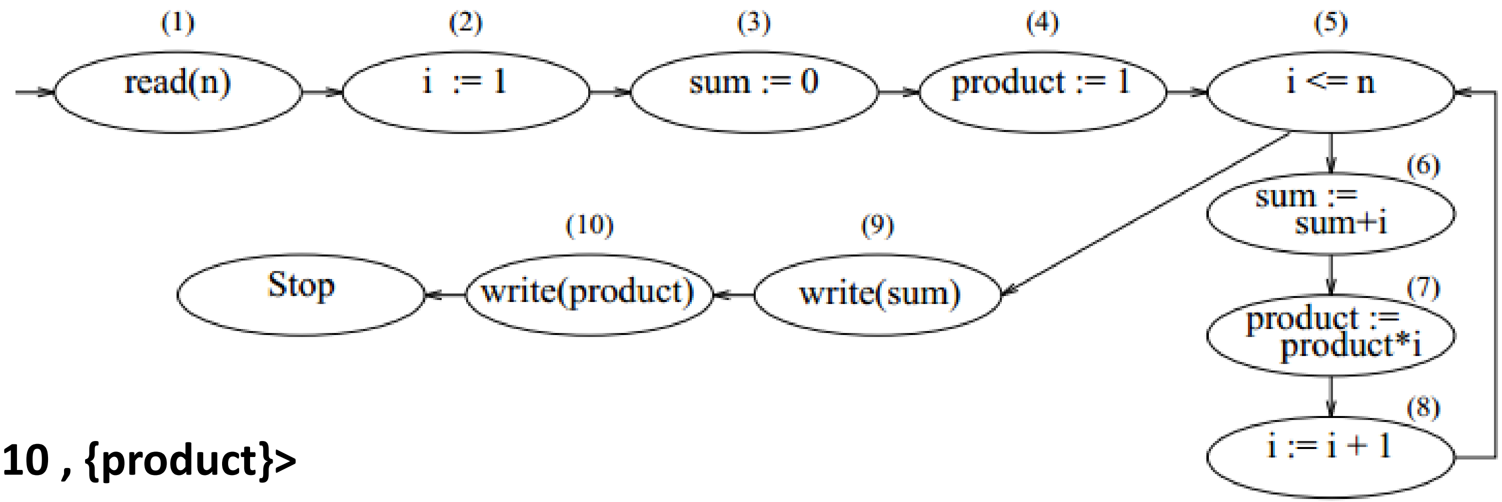
< 10 , {product}>

Node #	Def	Ref	INFL	R_C^0	In S_C^0	In B_C^0	R_C^1	In S_C^1
1	{n}	{}		{}				
2	{i}	{}		{}	/			
3	{sum}	{}		{i}				
4	{product}	{}		{i}	/			
5	{}	{i, n}		{product,i}				
6	{sum}	{sum, i}		{product,i}				
7	{product}	{product, i}		{product,i}	/			
8	{i}	{i}		{product}	/			
9	{}	{sum}		{product}				
10	{}	{product}		{product}				

Tracing transitive data dependences (2)

2. A set of ***directly relevant statements***, $\boldsymbol{S_C^0}$, is derived from $\boldsymbol{R_C^0}$ at each node i in the CFG

$$S_C^0 \quad = \quad \{i \mid (\mathrm{DEF}(i) \cap R_C^0(j)) \neq \emptyset, i \rightarrow_{\mathrm{CFG}} j\}$$



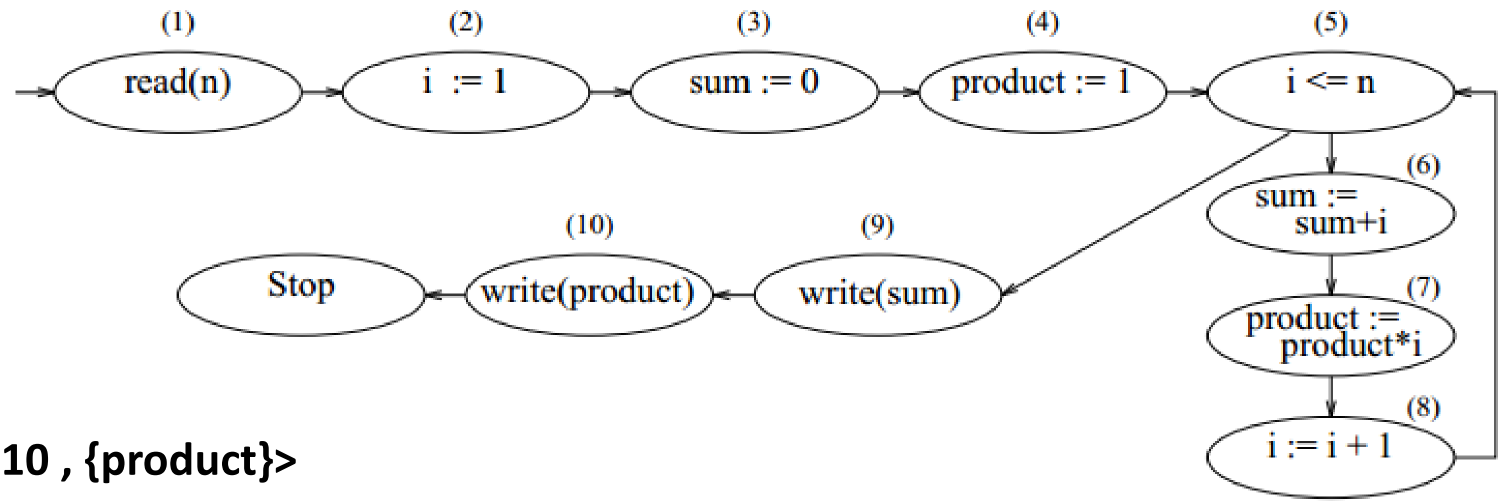
< 10 , {product}>

Node #	Def	Ref	INFL	R_C^0	In S_C^0	In B_C^0	R_C^1	In S_C^1
1	{n}	{}		{}				
2	{i}	{}		{}				
3	{sum}	{}		{i}				
4	{product}	{}		{i}				
5	{}	{i, n}		{product, i}				
6	{sum}	{sum, i}		{product, i}				
7	{product}	{product, i}		{product, i}				
8	{i}	{i}		{product, i}				
9	{}	{sum}		{product}				
10	{}	{product}		{product}				

Tracing control dependences (dealing with branch statement)

- INFL(b) is set of statements that is control dependent on branch statement b
- branching statement b is indirectly relevant to the slice if there is at least one directly relevant statement under its range of influence

$$B_C^k = \{b \mid \exists i \in S_C^k,\ i \in \textsc{Infl}(b)\}$$



< 10 , {product}>

Node #	Def	Ref	INFL	R_C^0	In S_C^0	In B_C^0	R_C^1	In S_C^1
1	{n}	{}		{}				
2	{i}	{}		{}	/			
3	{sum}	{}		{i}				
4	{product}	{}		{i}	/			
5	{}	{i, n}	{6,7,8}	{product, i}		/		
6	{sum}	{sum, i}		{product, i}				
7	{product}	{product, i}		{product, i}	/			
8	{i}	{i}		{product, i}	/			
9	{}	{sum}		{product}				
10	{}	{product}		{product}				

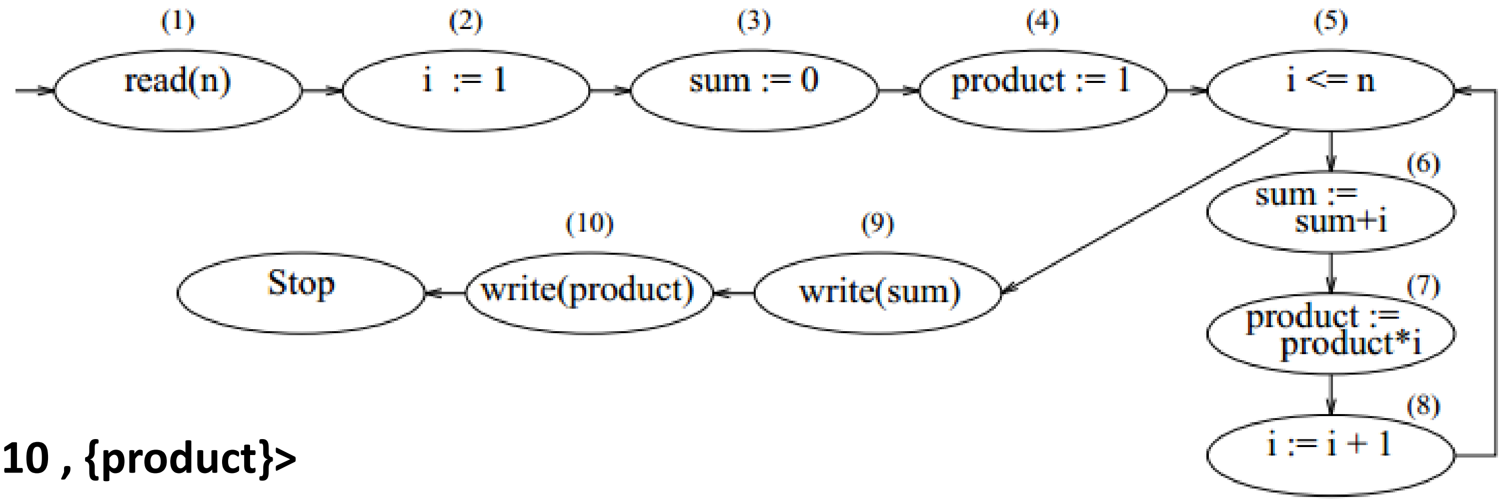
Tracing control dependences (cont'd) (dealing with branch statement)

- Trace relevant variables and statements with direct influence on $\boldsymbol{B_C^0}$

$$R_C^{k+1}(i) = R_C^k(i) \cup \bigcup_{b \in B_C^k} R^0_{(b,\mathrm{REF}(b))}(i)$$

$$S_C^{k+1} = B_C^k \cup \{i \mid \mathrm{DEF}(i) \cap R_C^{k+1}(j) \neq \emptyset, i \rightarrow_{\mathrm{CFG}} j\}$$

- The sets $\boldsymbol{R_C^k}$ and $\boldsymbol{S_C^k}$ are nondecreasing subsets of the program's variables and statements respectively
- The fixpoint of the computation of the $\boldsymbol{S_C^k}$ sets => the desired program slice.



< 10 , {product}>

Node #	Def	Ref	INFL	R_C^0	In S_C^0	In B_C^0	R_C^1	In S_C^1
1	{n}	{}	{}	{}			{}	/
2	{i}	{}	{}	{}	/		{n}	/
3	{sum}	{}	{}	{i}			{i,n}	
4	{product}	{}	{}	{i}	/		{i, n}	/
5	{}	{i, n}	{6,7,8}	{product, i}		/	{product, i, n}	/
6	{sum}	{sum, i}	{}	{product, i}			{product, i,n}	
7	{product}	{product, i}	{}	{product, i}	/		{product, i,n}	/
8	{i}	{i}	{}	{product, i}	/		{product, i,n}	/
9	{}	{sum}	{}	{product}			{product}	
10	{}	{product}	{}	{product}			{product}	

(1) read (n);
(2) i := 1;
(3) ~~sum := 0;~~
(4) product := 1;
(5) while i <= n do
begin
(6) ~~sum := sum + i;~~
(7) product := product * i;
(8) i := i + 1
end;
(9) ~~write(sum);~~
(10) write(<u>product</u>)

Node #	Def	Ref	INFL	R_C^0	In S_C^0	In B_C^0	R_C^1	In S_C^1
1	{n}	Ø	Ø	Ø			Ø	/
2	{i}	Ø	Ø	Ø	/		{n}	/
3	{sum}	Ø	Ø	{i}			{i, n}	
4	{product}	Ø	Ø	{i}	/		{i, n}	/
5	Ø	{i, n}	{6,7,8}	{product, i}		/	{product, i, n}	/
6	{sum}	{sum, i}	Ø	{product, i}			{product, i, n}	
7	{product}	{product, i}	Ø	{product, i}	/		{product, i, n}	/
8	{i}	{i}	Ø	{product, i}	/		{product, i, n}	/
9	Ø	{sum}	Ø	{product}			{product}	
10	Ø	{product}	Ø	{product}			{product}	

Interprocedural Slicing

- Compute interprocedural summary information for each procedure P
  - MOD(P) = variables that may be modified by P
  - USE(P) = variables that may be used by P
- Generation of new slicing criteria
  Translate **relevant variables R_C** into the scope of new procedure

```
1        READ(A,B)
2        CALL Q(A,B)
3        Z := A + B

         PROCEDURE Q(VAR X,Y : INTEGER)
4        X := 0
5        Y := X + 3
6        RETURN


DOWN(<3,{Z}>) = {<6,{X,Y}>}

UP(<4,{Y}> = {<2,{B}>}
```

Fig. 4. Extending slices to called and calling routines.

P is sliced, P calls Q

generates : <**last statement of Q**, **relevant vars in P in the scope of Q** >

Q is sliced, Q is called by P

generates : **<first statement in P, relevant vars in Q in the scope of P>**

Interprocedural Slicing (cont’d)

- **UP** maps set **C** of slicing criteria in a procedure P to a set of criteria in **procedures that call P**
- **DOWN** maps set **C** of slicing criteria in a procedure P to a set of criteria in **procedures called by P**
- The complete interprocedural slice for a criterion **C** = union of the intraprocedural slices for each criterion in **(UP ∪ DOWN)*({ C })**
- Interprocedurally imprecise because it does not model calling contexts

A Sampling of Slices

- Test the program slicer on 19 student compilers (500 – 900 executable statements long; 20 – 80 subroutines)

- The compilers were sliced at each write statement *i* and a set of output variables *V*

- Slices differed by less than 30 statements were merged into a new slightly large slice

Statistics on slices

TABLE I
STATISTICS ON SLICES

Measure	Mean	Median	Min	Max
Per program measures $N = 19$				
Useless	9.16	6	1	23
Common	14.32	0	0	86
Slices	37.26	32	7	74
Clusters	9.74	7	3	25
Per cluster measures $N = 185$				
Contig	11.78	9.10	0	65.4
% Size	44	40	0	97
% Unique	6	1	0	100
% Overlap	52	51	0	93

Length of contiguous statements in a cluster which were contiguous in original program = 11.78

Low uniqueness of slices reflects high degree of interrelatedness of compiler programs

References:

- Weiser, Mark. "Program slicing." *Proceedings of the 5th international conference on Software engineering*. IEEE Press, 1981.
- http://pubweb.parc.xerox.com/weiser/weiser.html
- Tip, Frank. "A survey of program slicing techniques." *Journal of programming languages* 3.3 (1995): 121-189.
- Mary Jean Harrold's slides, Software Analysis and Testing course: Program Slicing, http://www.cc.gatech.edu/~harrold/6340/cs6340_fall2009/Slides/BasicAnalysis6.pdf