Slicing

Rupesh Nasre.

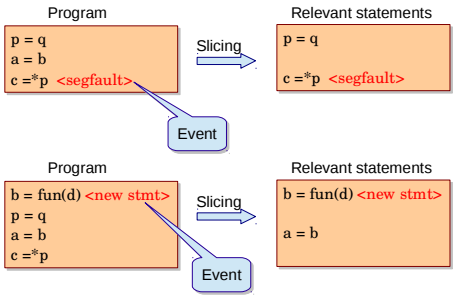
CS6843 Program Analysis
IIT Madras
Jan 2016

Outline

- Introduction and applications
- Application time
  - static
  - dynamic
- Direction
  - forward
  - backward

Definition

- A slice is a subset of program statements (possibly) relevant to an event of interest.



Applications

- Program understanding / debugging
- Program restructuring
- Program differencing
- Test coverage
- Model checking

Program with Multiple Functionality

Line + Character counting

```
extern void scanLine(FILE *f, bool *eof, int *nread);
void lineCharCount(FILE *f) {
    int nlines = 0;
    int nchars = 0;
    int nread;
    bool eof = false;

    do {
        scanLine(f, &eof, &nread);
        ++nlines;
        nchars += nread;
    } while (!eof);

    printf("nlines = %d\n", nlines);
    printf("nchars = %d\n", nchars);
}
```


Program with Single Functionality

Line + Character counting

```
extern void scanLine(FILE *f, bool *eof,
                     int *nread);
void lineCharCount(FILE *f) {
    int nlines = 0;
    int nchars = 0;
    int nread;
    bool eof = false;

    do {
        scanLine(f, &eof, &nread);
        ++nlines;
        nchars += nread;
    } while (!eof);

    printf("nlines = %d\n", nlines);
    printf("nchars = %d\n", nchars);
}
```

Line + Character counting

```
extern void scanLine(FILE *f, bool *eof,
                     int *nread);
void lineCharCount(FILE *f) {
    int nlines = 0;
    int nchars = 0;
    int nread;
    bool eof = false;

    do {
        scanLine(f, &eof, &nread);
        ++nlines;
        nchars += nread;
    } while (!eof);

    printf("nlines = %d\n", nlines);
    printf("nchars = %d\n", nchars);
}
```

Program with Single Functionality

Line + Character counting

```
extern void scanLine2(FILE *f, bool *eof,
                      int *nread);
void lineCharCount(FILE *f) {
    int nlines = 0;
    int nchars = 0;
    int nread;
    bool eof = false;

    do {
        scanLine2(f, &eof, &nread);
        ++nlines;
        nchars += nread;
    } while (!eof);

    printf("nlines = %d\n", nlines);
    printf("nchars = %d\n", nchars);
}
```

Line + Character counting

```
extern void scanLine(FILE *f, bool *eof,
                     int *nread);
void lineCharCount(FILE *f) {
    int nlines = 0;
    int nchars = 0;
    int nread;
    bool eof = false;

    do {
        scanLine(f, &eof, &nread);
        ++nlines;
        nchars += nread;
    } while (!eof);

    printf("nlines = %d\n", nlines);
    printf("nchars = %d\n", nchars);
}
```


Slicing Types

- Forward
  - The forward slice at program point p is the program subset that may be affected by p
- Backward
  - The backward slice at program point p is the program subset that may affect p
- Chop
  - The chop between program points p and q is the program subset that may be affected by p and that may affect q

Forward Slice

```
void main() {
    int sum = 0;
    int i = 0;

    while (i < 11) {
        sum += i;
        ++i;
    }
    printf("sum = %d\n", sum);
    printf("i = %d\n", i);
}
```

Slicing criteria

Forward Slice

```
void main() {
    int sum = 0;
    int i = 0;

    while (i < 11) {
        sum += i;
        ++i;
    }
    printf("sum = %d\n", sum);
    printf("i = %d\n", i);
}
```

Slicing criteria

Backward Slice

```
void main() {
    int sum = 0;
    int i = 0;

    while (i < 11) {
        sum += i;
        ++i;
    }
    printf("sum = %d\n", sum);
    printf("i = %d\n", i);
}
```

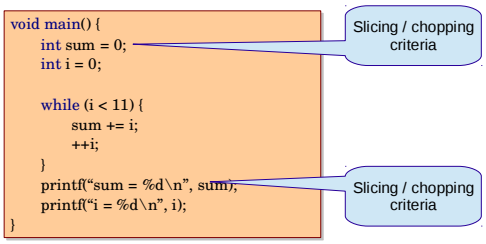
Slicing criteria

Backward Slice

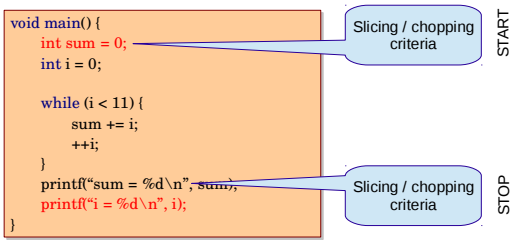
```
void main() {
    int sum = 0;
    int i = 0;

    while (i < 11) {
        sum += i;
        ++i;
    }
    printf("sum = %d\n", sum);
    printf("i = %d\n", i);
}
```

Slicing criteria

Chop



Chop



Slicing Types

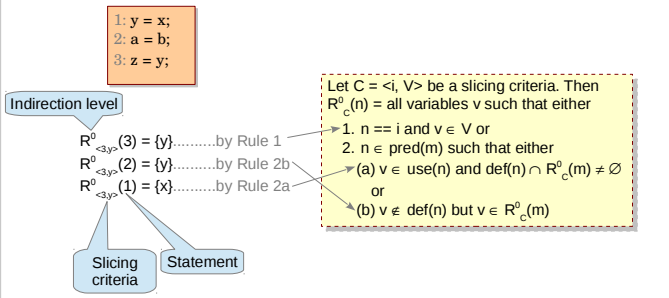
- Static
  - Compile time, without making assumptions about the program inputs
- Dynamic
  - Execution time or compile time, relying on specific test cases

Language features such as functions, unstructured control flow, composite data types, pointers, and concurrency require specific extensions to the slicing algorithms.

Static Slicing

- Whatever we have been looking at so far...
- There could be multiple valid slices for a given criteria, depending upon analysis precision.
- There exists at least one slice for a criteria (the program itself).
- The analysis implicitly assumes that the program halts.
- Finding a statement-minimal slice is impossible (halting problem).

Computing Relevant Variables



2a says that if w is a relevant variable at the node following n and w is defined at n, then w is no longer relevant; instead, all the variables used to define w are now relevant.
2b says that if a relevant variable at the next node is not defined at node n, then it is still relevant at node n.

Computing Relevant Statements

- The relevant statements define relevant variables.

S^0_C = all statements n such that def(n) ∩ $R^0_C(n+1) \neq \emptyset$.

But what about Control Dependence?

- Let's define a set of statements that influence a set of statements
- INFL(b) is the set of statements which directly affect execution of statement b.

B^0_C = ∪ INFL(n) such that n ∈ S^0_C

Branch statements with indirect relevance to a slice

To include ***all*** the indirect influences, the statements with direct influence on B^0_C must now be considered, and then the branch statements influencing those new statements, and so on.

Computing Static Slice

Relevance at level n is defined as below

Next set of relevant variables = current set of relevant variables ∪ relevant variables in current set of relevant branch statements

for all i ≥ 0

$R^{i+1}_C(n) = R^i_C(n) \cup R^0_{BC(b)}(n)$ for $b \in B^i_C$

where BC(b) is a branch statement criterion defined as <b, use(b)>

B^{i+1}_C = ∪ INFL(n) for $n \in S^{i+1}_C$

S^{i+1}_C = all statements n such that $def(n) \cap R^{i+1}_C(n+1) \neq \emptyset$ or $n \in B^i_C$


```
void main() {
 int y, z, ii, n;
 int *a;
 scanf("%d", &n);
 a = (int *)malloc(n);

 for (ii = 0;
      ii < n; ) {
    scanf("%d", a + ii);
    ++ii;
 }
 y = 0;
 z = 1;

 for (ii = 0;
      ii < n;
      ++ii) {
    y += a[ii];
    z *= y;
 }
 printf("%d\n", z);
}
```

Classwork

Find backward slice for the program.
Slicing criteria: **<line number of printf, z>**

Slicing as a DFA

```
1: a = 1;
2: while (f(k)) {
3:    if (g(c)) {
4:          b = a;
5:          x = 2;
6:    } else {
7:          c = b;
8:          y = 3;
      }
9:    k = k + 1;
   }
10: z = x + y;
```

Is statement 1 included in the slice?

Slicing criterion = **<10, z>**

Slicing as a DFA may not be minimal

```
1: a = 1;
2: while (f(k)) {
3:    if (g(c)) {
4:          b = a;
5:          x = 2;
6:    } else {
7:          c = b;
8:          y = 3;
      }
9:    k = k + 1;
   }
10: z = x + y;
```

Is statement 1 included in the slice?

Slicing criterion **= <10, z>**

Statements 1 and 4 would not be part of the slice as there is a guarantee that after if block is executed, else block will definitely be not executed. Therefore, there is no dependence from statement 4 to statement 7.

Dynamic Slicing

Slicing criteria
(9^1, x, n=2)

```
1: read(n);
2: i = 1;
3: while (i <= n) {
4:    if (i % 2 == 0)
5:          x = 7;
6:    else
7:          x = 16;
8:    ++i;
   }
9: write(x);
```

```
1: read(n);
2: i = 1;
3: while (i <= n) {
4:    if (i % 2 == 0)
5:          x = 7;
6:    else
7:          ;
8:    ++i;
   }
9: write(x);
```

The else branch may be omitted from the dynamic slice because the assignment of 16 to x in the first iteration is killed by the assignment of 7 in the second iteration.

Static vs. Dynamic Slicing

- Across all inputs
- Less precise
- Source code
- Slicing criteria contains statement and variables.

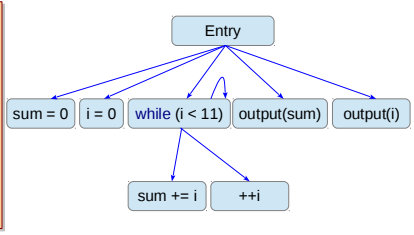
- Specific set of inputs
- More precise
- Source code or trace
- Slicing criteria contains statement instance, variables and inputs.

Computing Slices

- Reachability in a dependence graph
  - PDG (intra-procedural)
  - SDG (inter-procedural)
- Dependence graph
  - Control dependence *(is it the same as a CFG?)*
  - Data dependence

Control Dependence Graph

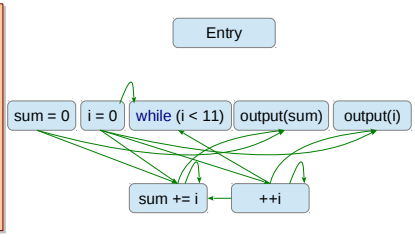
```
void main() {
    int sum = 0;
    int i = 0;

    while (i < 11) {
        sum += i;
        ++i;
    }
    printf("sum = %d\n", sum);
    printf("i = %d\n", i);
}
```



Control dependence

Data Dependence Graph

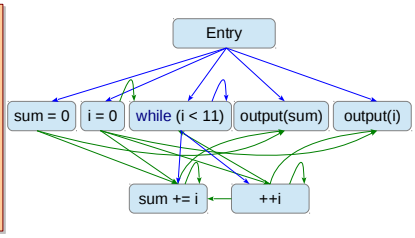
```
void main() {
    int sum = 0;
    int i = 0;

    while (i < 11) {
        sum += i;
        ++i;
    }
    printf("sum = %d\n", sum);
    printf("i = %d\n", i);
}
```



Data dependence

Program Dependence Graph

```
void main() {
    int sum = 0;
    int i = 0;

    while (i < 11) {
        sum += i;
        ++i;
    }
    printf("sum = %d\n", sum);
    printf("i = %d\n", i);
}
```



Control dependence
Data dependence

Syntactically Valid Slices

- Include additional statements or reorder statements to make the slice syntactically valid.
- Operates at the source code level not IR.

```
read(n);
i = 1;
if (p == null)
    x = x + 1;
else
    n = n * 2;
write(n);
```

Slicing

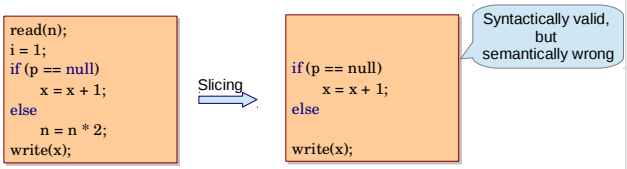
```
read(n);

if (p == null)

else
    n = n * 2;
write(n);
```

Syntax error

Situation is simpler in C due to ; as statement terminator.

Syntactically Valid Slices

- Include additional statements or reorder statements to make the slice syntactically valid.
- Operates at the source code level not IR.



Situation is simpler in C due to ; as statement terminator.

Classwork

```
1  main() {
2  int x = 0;
3  int n = 1;
4  int a[5] = {0, 2};
5  int i = 0;
6  if (n > 0) {
7    for (; i < n; ++i) {
8         a[i] = i * i;
9         if (a[i] < 100) {
10             x += a[i];
11        } else {
12             x = a[i];
13        }
14   }
15 }
16 printf("%d\n", x);
17 }
```

For the given program
- draw control-dependence graph.
- draw data-dependence graph
- find forward slice for <4, a>.
- find backward slice for <16, x>.

Acknowledgments

- Christopher Lewis (upenn)
- Laurie Hendren (mcgill)
- Frank Tip (waterloo)