Анализ кода и информационная безопасность

Лекция 08

МГУ / ВМК / СП

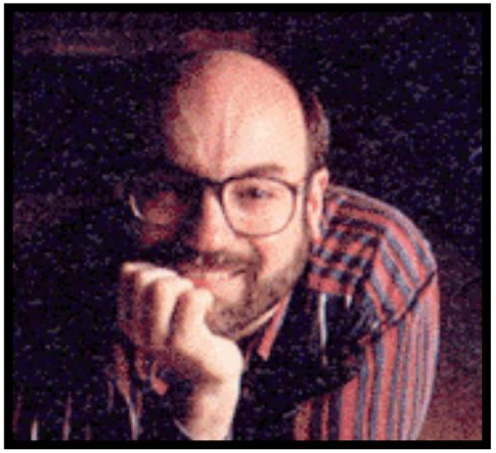
0801

ПРОГРАММНЫЙ СЛАЙСИНГ. СТАТИЧЕСКИЙ, ДИНАМИЧЕСКИЙ, УСЛОВНЫЙ СЛАЙСИНГ.

Анализ ошибки при отладке программе

1. **Обнаружение факта наличия ошибки.** Падение программы или некорректное поведение на некоторых фиксированных входных данных.
2. **Отладка программы**. Обнаружение точки в программе, где с некоторыми данными осуществляется недопустимая операция.
3. **Выявление источника ошибки.** Поиск инструкций в программе, в которой переменная приобрела значение, впоследствии приведшее к ошибке.

Возникновения идеи слайсинга



Марк Вейзер наблюдал за процессом отладки программ опытными программистами

Пришёл к выводу что все они используют слайсинг в процессе отладки

Описал эту технику в своей диссертации в 1979 году и опубликовал ряд статей в 1980-1981.

Впоследствии предложенная им идея вылилась в большое семейство алгоритмов. Исходный подход классифицируется как: обратный исполнимый статический слайсинг

Идея слайсинга (по Вейзеру)

Слайс программы для инструкции n и переменной x – множество инструкций программы, которые **могут** повлиять на значение x перед инструкцией n.

Инструкция **может** повлиять на значение переменных точке n потому что:

- Определяет будет ли вообще выполняться (зависимость по управлению)
- Инструкция определяет значение переменной, которая используется в n (зависимость по данным)

Слайсинг – алгоритм, описывающий построение слайса

Так как предположений относительно входных данных не делается – получаемый слайс **статический**

Виды зависимостей

Зависимость по данным

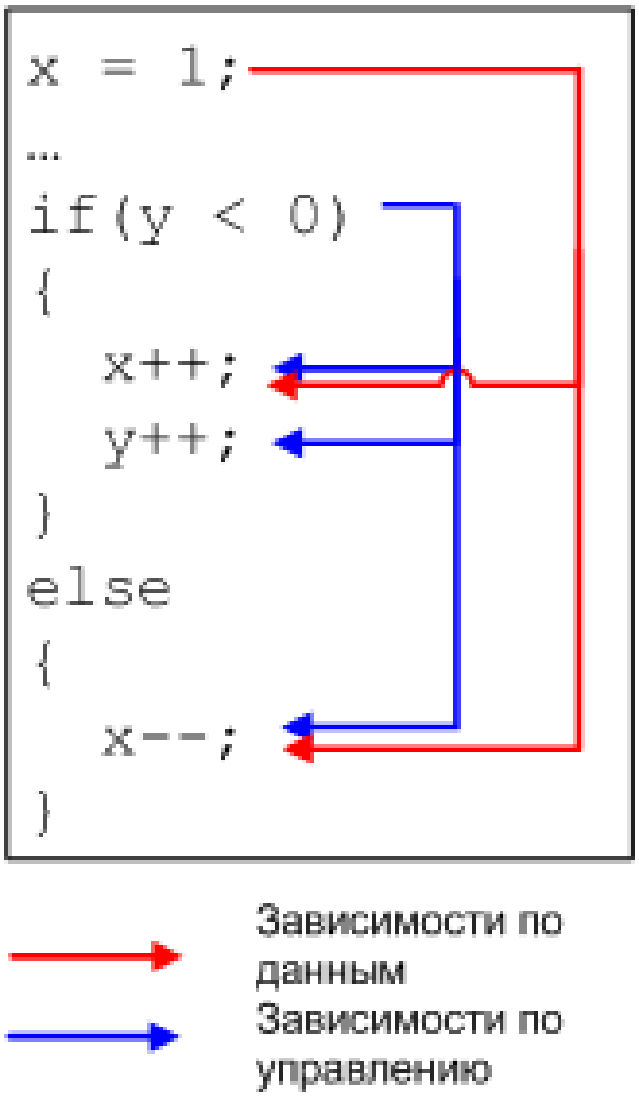
Определение переменной v в выражении s1 достигает использования переменной v в выражении s2.

Зависимость по управлению

Условное выражение s1 с определяет, будет или нет выполнено выражение s2

Иногда выделяют слайсинг, учитывающий только один вид зависимостей:

- Слайсинг по данным
- Слайсинг по управлению

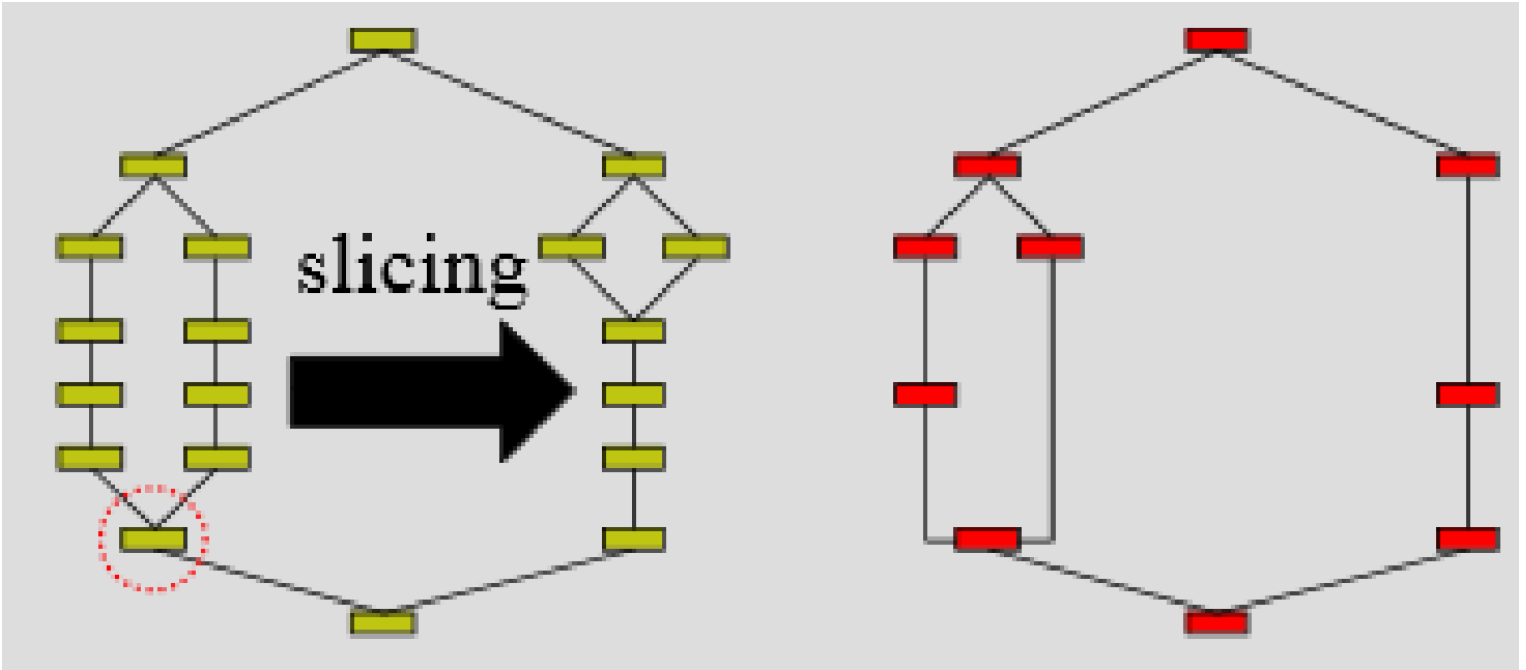


Ограничения слайсинга

Задача построения **минимального статического** слайса в общем случае неразрешима

Тривиальный слайс = «вся программа»

На практике требуется построить приемлемое приближение, т.е. уменьшить количество кода, который необходимо анализировать.



Входные данные и результат алгоритма слайсинга

Входные данные: программа P и критерий слайсинга – пара <n,V>, где

- n – точка в программе
- V – множество переменных

Результат: слайс S(n,V, P)**,** программа, содержащая все выражения программы P, влияющие на вычисление значений переменных из множества V в точке n.

Свойства слайса по определению

1. Слайс S(n, V, P) должен получаться из программы P путём удаления части выражений P.
2. Слайс S(n, V , P) должен быть синтаксически корректным (исполнимый)
3. Для всех входных данных программы P, значения переменных из множества V при выполнении P и S(n, V , P) перед выражением n должны быть одинаковы (обратный)

Общая схема вычисления обратного слайса

1. Проход по рёбрам графа потока управления в обратном направлении
2. Начало в точке n (критерий C = <n,V>)
3. Просмотр инструкций, которые могли быть выполнены перед n
4. Добавление в слайс тех инструкций которые повлияли на значения V или факт выполнения n
5. При добавлении новой инструкции требуется учесть транзитивные зависимости, т.е. зависимости добавляемой инструкции

Пример слайса

Исходная программа

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

Критерии слайсинга:

<10, product>

Пример слайса(2)

Исходная программа

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

Критерии слайсинга:

<10, product>

Вычисление слайсов

Исходная программа

```
1 begin
2 read(x,y)
3 total := 0.0
4 sum := 0.0
5 if x <= 1
6   then
7     sum := y
8   else
9     begin
10      read(z)
11      total := x*y
12    end
13  write(total, sum)
14  end.
```

Критерии слайсинга:

1. Критерий слайса: <13, z>

2. Критерий слайса: <11, x>

3. Критерий слайса : <13, total>

Исполнимый слайсинг

- Если выполняется свойство 2 слайса из определения Вейзера:

  Слайс S(n, V , P) должен быть синтаксически корректным

  слайсинг называется исполнимым
- При этом получаемый слайс является корректной исполняемой программой и может быть проверен на выполнение свойства 3 буквально – перебором входных данных, выполнении исходной программы и слайса до точки n и сопоставлением значений из V

Свойства статического слайсинга

1. Не делается предположений о входных данных
2. Слайсы содержат все выражения которые как-либо могут повлиять на переменные при любом возможном выполнении
3. Доступна только статическая информация (проблемы с алиасами)
4. Получаемые слайсы достаточно не точны и имеют относительно большой размер
5. Задача построения минимального слайса алгоритмически неразрешима - существующие реализации алгоритмов могут вычислять слайса в некотором приближении.
6. Результаты зачастую бесполезны

Более точные виды слайсинга

- **Динамический**
  - Фиксируются входные данные, на которых строится
  - Вводится понятие истории выполнения – последовательность выполнявшихся инструкций
  - Критерий слайсинга: <I, n, V>, где I – набор входных данных программы, а n – точка в «истории выполнения программы».
- **Условный**
  - На входные данные накладывается усовие c.
  - Критерий слайсинга: <c, n, V>, где c – условие на входные данные.

Пример динамического слайса

Исходная программа

```
1.  read (n)
2.  for I := 1 to n do
3.     a := 2
4.     if c1 then
5.        if c2 then
6.           a := 4
7.        else
8.           a := 6
9.     z := a
10. write (z)
```

Критерии слайсинга:

n = 2

c1,c2 – на первой итерации оба true, на второй - оба false

История выполнения:

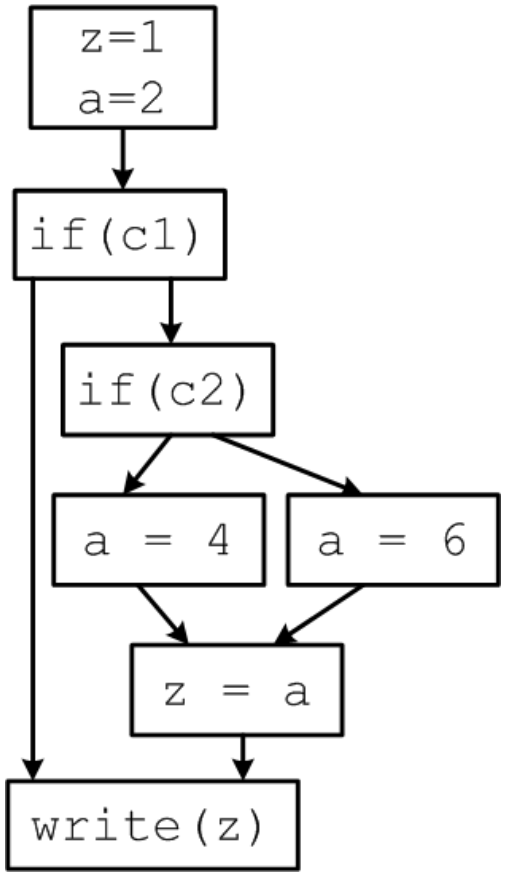
$1^1, 2^1, 3^1, 4^1, 9^1, 2^2, 3^2, 4^2, 5^1, 6^1, 9^2, 2^3, 10^1>$

Особенности динамического и условного слайсинга

- **Динамический**
  - Имеют точный алгоритм построения.
  - Имеют относительной маленький размер.
  - Вычисляются реальные, а не возможные зависимости.
  - Известны значения всех переменных – нет проблем с алиасами.
- **Условный**
  - Условия на входные данные обычно связаны с предикатами путей в программе
  - Получаемый слайс по размеру занимает промежуточное положение между динамическим и статическим слайсингом

Сравнение вариантов слайсинга

```
1. z = 1;
2. a = 2;
3. if (c1)
4.   if (c2)
5.     a = 4;
6.   else
7.     a = 6;
8.   z = a;
9. write (z);
```



Условия

- Все слайсы будем строить от точки 9 по значению z

Сравнение слайсов

Статический слайс

Критерий <9, z>

```
1. z = 1;
2. a = 2;
3. if (c1)
4.   if (c2)
5.     a = 4;
6.   else
7.     a = 6;
8.   z = a;
9. write (z);
```

Условный слайс

Критерий

< c1- любое, c2 – true, 9, z>

```
z = 1;
a = 2;
if (c1)
  if (c2)
    a = 4;
  else
    a = 6;
  z = a;
write (z);
```

Динамический слайс

c1, c2 – false, I=1, 2, 3, 9

Критерий <I, 9, z>

```
z = 1;
a = 2;
if (c1)
  if (c2)
    a = 4;
  else
    a = 6;
  z = a;
write (z);
```

Уровни построения слайса

Внутрипроцедурный

- Слайс строится в рамках отдельной функции.
- Вызовы других функций интепретируются
- консервативно:
- могут возвращать любые значения
- могут изменять передаваемые по указателю параметры

Межпроцедурный (предложен Вейзером)

- Может проходить вовнутрь и наружу вызовов
- Фактические параметры заменятся на формальные и наоборот
- Переменные при выходе из области видимости удаляются
- Не контекстно-чувствительный – может быть очень не точным

Методы вычисления слайсинга

- Анализ потоков данных на графе потока
  - Внутрипроцедурный – по графу потока управления (CFG)
  - Межпроцедурный – по межпроцедурному графу потока управления (ICFG)
- Анализ достижимости на графе зависимостей по данным
  - Внутрипроцедурный – граф зависимостей программы (PDG)
  - Межпроцедурный – граф зависимостей системы (SDG)

Уравнения потока данных

- Использовались Вейзером
- Итеративный процесс над графом потока управления
  - На каждой итерации консервативно вычисляется множество «релевантных» переменных для каждой вершины графа, используя достигающие определения
  - Зависимости по управлению явно не используются
  - Переменные предикатов управления (if,while,…) являются «косвенно релевантными», если любая инструкция из области, которая зависит от них по управлению «релевантна»
- Процесс начинается с точки из критерия слайсинга
- Заканчивается пи достижении фиксированной точки – на очередной итерации не добавляется новых релевантных инструкций

Уравнения потока данных Определения.

$i \longrightarrow_{CFG} j$: Из вершины i есть ребро в вершину j

Def(i): Множество переменных изменяемых в вершине i

Ref(i): Множество переменных используемых в вершине i

Локальные

Infl(i): Множество вершин,зависящих от i по управлению

Вычисляются по ГПУ

R^0_C: Релевантные переменные

R^k_C: Косвенно релевантные переменные

S^0_C: Релевантные вершины

S^k_C: Косвенно релевантные вершины

B^k_C: Релевантные вершины ветвлений

Распространяются по ГПУ

Вычисление множеств релевантных инструкций

Итерация 0:

$$R_C^0(i) = V \text{ when } i = p$$

$R_C^0(i)$ (for every i $\rightarrow_{CFG}$ j)

(i) $v \in Ref(i)$ and $Def(i) \cap R_C^0(j) \neq \emptyset$

(ii), $v \notin Def(i)$ and $v \in R_C^0(j)$

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

$$B_C^0 = \{b \mid i \in Infl(b), i \in S_C^0\}$$

Итерация k+1:

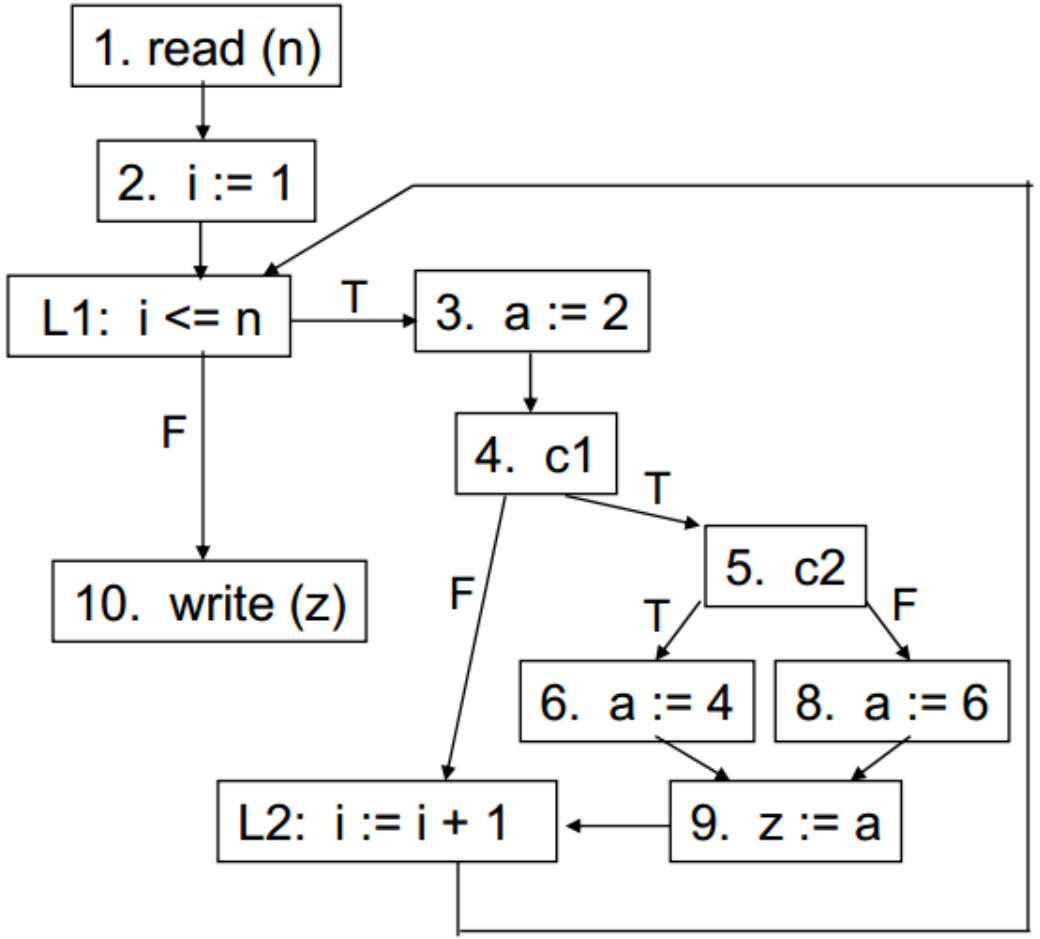
$$R_C^{k+1}(i) = R_C^k(i) \cup \bigcup_{b \in B_C^k} R^0_{(b,Ref(b))}(i)$$

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

$$B_C^{k+1} = \{b \mid i \in Infl(b), i \in S_C^{k+1}\}$$

Достижимость на графах. Граф потока управления.

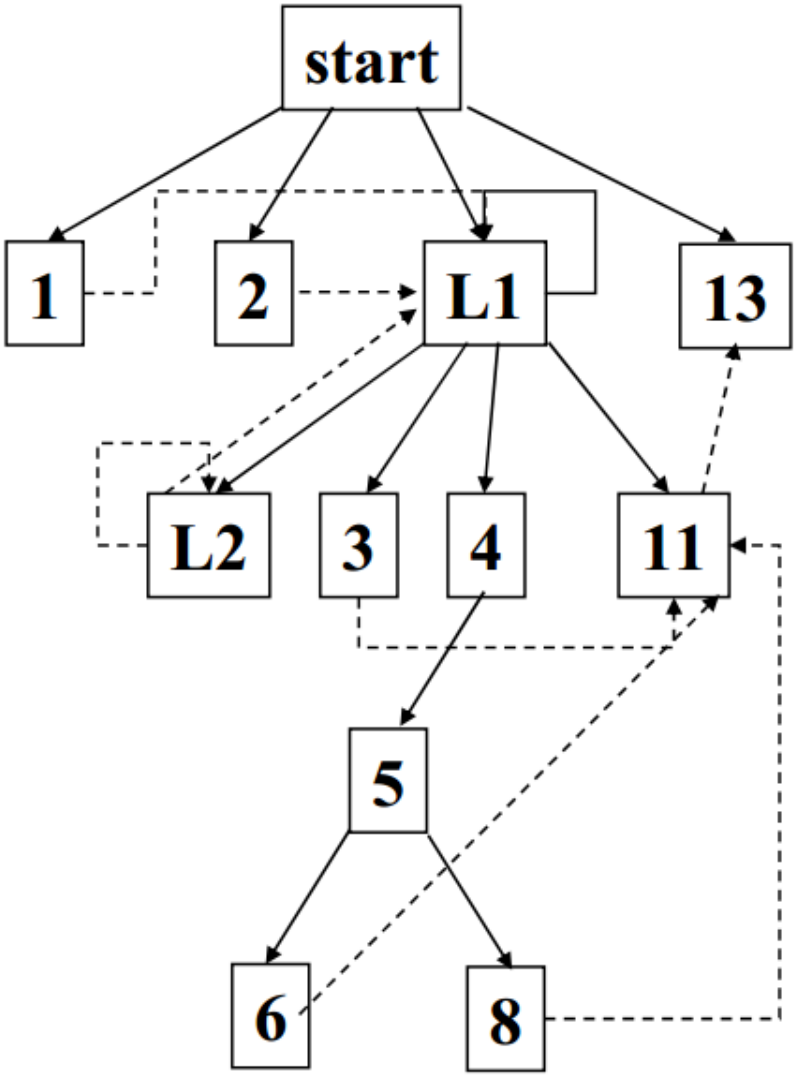
```
1.  read (n)
2.  for I := 1 to n do
3.      a := 2
4.      if c1 then
5.          if c2 then
6.              a := 4
7.          else
8.              a := 6
9.          z := a
10. write (z)
```



Достижимость на графах. Граф зависимостей.

```
1.   read (n)
2.   for I := 1 to n do
3.      a := 2
4.      if c1 then
5.         if c2 then
6.            a := 4
7.         else
8.            a := 6
9.         endif
10.     endif
11.     z := a
12.  endfor
13.  write (z)
```

Слайс: 1, 2, 3, 4, 5, 6, 11, 13



Ограничения алгоритма

- Может применяться только для таких критериев <i,V>, в которых множество переменных V является подмножеством переменных, используемых в точке i (Ref(i))
- Граф зависимостей не определён для программ с неструктурированным потоком управления – например, содержащим выходы из программы и исключения
- Необходимо построить граф зависимостей для всей программы перед началом слайсинга

Применение алгоритма для динамических и условных слайсов

1. Пометить вершины (менее точно) или рёбра которые выполнялись (динамический слайс) или могли выполнится (условный слайс)
2. Выполнить алгоритм для статического слайсинга на уменьшенном графе

Достижимость на графах. Динамический слайсинг.

```
1.   read (n)
2.   for I := 1 to n do
3.     a := 2
4.     if c1 then
5.       if c2 then
6.         a := 4
7.       else
8.         a := 6
9.       endif
10.    endif
11.    z := a
12.  endfor
13.  write (z)
```

Входные данные (I):

n = 2,

1 итерация c1 = false, c2 = false

2 итерация c1 = true, c2 = true

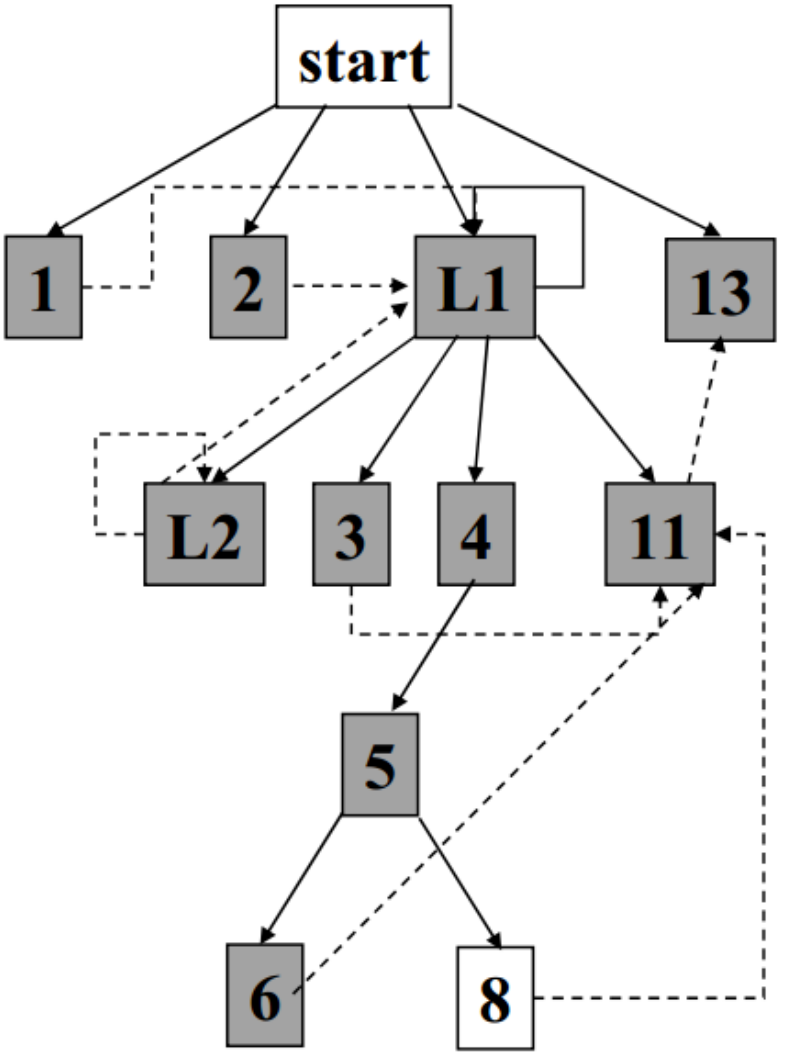
История выполнения:

1, 2, L1, 3, 4, 11, L2, L1, 3, 4, 5, 6, 11, L2, L1,13

Критерий слайсинга <I,13, z>

Достижимость на графах. Динамический слайсинг(2).

```
1.   read (n)
2.   for I := 1 to n do
3.      a := 2
4.      if c1 then
5.         if c2 then
6.            a := 4
7.         else
8.            a := 6
9.         endif
10.     endif
11.     z := a
12.  endfor
13.  write (z)
```



Слайс: 1, 2, 3, 4, 5, 6, 11, 13

Прямой слайсинг

- **Прямой слайс программы с критерием S(n,V) состоит из всех инструкций программы на выполнение которых повлияло значение переменных из V в точке n.**

```
x = 1; /* what happens when this line is changed */
y = 3;
p = x + y ;
z = y -2 ;
if (p==0)
r++ ;
```

```
/* Change to first line will affect */
p = x + y ;
if (p==0)
r++ ;
```

Пример прямого слайса

Исходная программа

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

Критерии слайсинга:

<3, sum>

Вопрос:

Что не так с этим слайсом?

Чоппинг

Пусть есть точка в программе S (источник) и переменные V1 в точке S и точка T (цель) с переменными V2.

Как значения переменных V1 в точке S повлияли на значения V2 в точке T?

Чоп – пересечение прямого слайса с критерием <S,V1> и обратного слайса с критерием <T,V2>

Область применения алгоритмов слайсинга

- Отладка
- Понимание программ:
  - Какие части программ действуют друг на друга?
- Тестирование
  - Какие тесты увеличат покрытие кода?
  - Какие регрессионные тесты нужно запускать после изменений?
- Измерение метрик кода:
  - Покрытие кода тестами, связанность компонент в коде
- Рефакторинг
  - Разделение функционально независимых частей кода
- Отслеживание изменений
  - Сравнение версий программ по их слайсам

Области применения прямого и обратного слайсинга

- Обратный слайсинг
  - Изоляция отдельных потоков вычислений в программе
  - Автоматизация распараллеливания
  - Автоматизация совместного использования кода разных версий

- Прямой слайсинг
  - Показывает, как вычисленное значение используется в дальнейшем помогает проверить, что последующий код не нарушает контракт на использование значения
  - Проверяет части кода на которые влияет вносимое изменение для проверки отсутствия неявных и нежелательных эффектов

0802

ВЫЯВЛЕНИЕ ПУТИ РАСПРОСТРАНЕНИЯ ОШИБКИ. СЛАЙСИНГ БИНАРНОГО КОДА.

Путь распространения ошибки

```
int f(bool double){
   char *buf;
   int j;
   int i = 4;
   buf = malloc(10);
   j = i + 6;
   if(!double)
   {
       buf[j] = 0;
   }
   else
   {
       buf[2*j] = 0;
   }
}
```

Поток данных в программе, приведший к ошибке:

- Источник (source) – место инициализации переменных, значения которых привели к ошибке.
- Распространение (propagation) – инструкции, участвовавшие в обработке/передаче значений, которые привели к ошибке
- Место проявления ошибки (sink) – инструкция, приводящая к ошибке

- Цель обнаружения ошибки – исправление
- Цель выявления пути – определить точку программы для внесения исправления

Обнаружение ошибки

Ошибка с точки зрения инструмента обнаружения:

Инструкция в программе и набор переменных, значения или атрибуты которых нарушают условия корректности выполнения этой инструкции. Примеры:

- Инструкция деления, делитель равный 0
- Инструкция доступа к буферу, размер буфера меньше индекса доступа

Подходы к обнаружению:

•Динамический подход (тестирование, отладка, …)

- Конкретные входные данные и путь исполнения, аварийное завершение в точке проявления

•Статический подход (анализ потоков данных)

- Все возможные входные данные, совокупность всех путей выполнения и потоков данных, возможное нарушение условий корректности в точке проявления

•Символический подход (предикаты пути и безопасности)

- Уравнение из условий на входные данные, приводящих к появлению ошибки имеющее решение

Задача выявления пути распространения ошибки

По заданной точке проявления ошибки и множеству переменных, значения которых привели к ошибке найти последовательность инструкций, приведших к ошибке.

Возможные решения:

- Вручную, при динамическом подходе . «Обратная отладка» – отслеживание инструкций и обращений к переменным с помощью точек прерывания.
- Автоматически. Алгоритм слайсинга.

Применение слайсинга для выявления пути ошибки

- В качестве критерия слайсинга C(n, V) используется:
- n – точка обнаружения ошибки
- V - множество переменных, значения которых привели к ошибке

Подход, с помощью которого обнаружена ошибка	**Алгоритм слайсинга для выявления пути ошибки**
Динамический	Обратный динамический
Статический	Обратный статический
Символический	Обратный условный

Слайсинг бинарного кода

Проблемы.

1. Неизвестны переменные и их типы (регистры и буферы памяти)
2. Неизвестны границы и параметры функций.
3. Сложности с построением графа потока управления и графа зависимостей программы.

Решения.

1. Восстановление переменных и их типов
2. Выделение функций в коде
3. Восстановление графов потока управления и графа зависимостей

Литература к лекции

1. M. Weiser., Program Slicing, May 1981
2. F.Tip. A Survey of Program Slicing, 1995
3. Horwitz S., Reps T., and Binkley D. Interprocedural slicing using dependence graphs, 1990.
4. B. Korel, J. Laski. Dynamic program slicing, 1988
5. M.Harman, S.Danicic, Y.Sivagurunathan. Next 700 slicing criteria, 1996