



ТЕСТВАНЕ НА УПРАВЛЯВАЩИЯ ПОТОК, ДАННОВИТЕ ЗАВИСИМОСТИ И ВЗАИМОДЕЙСТВИЯТА

доц. д-р Десислава Петрова-Антонова

Съдържание

- ❖ Основни типове взаимодействия при изпълнение
- ❖ Тестване на управляващия поток
- ❖ Анализ на данновите зависимости
- ❖ Тестване на данновия поток

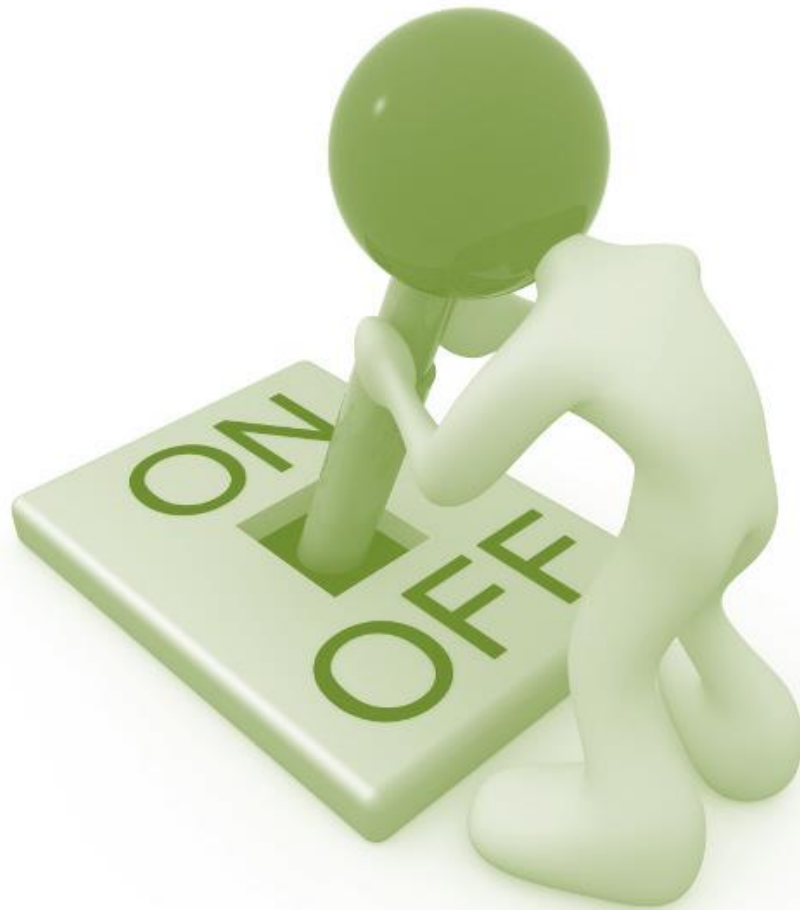
Софтуерна система

- Свързани функции или операции
- Компоненти
- Модули
- Подсистеми



ТЕСТВАНЕ НА УПРАВЛЯВАЩИЯ ПОТОК

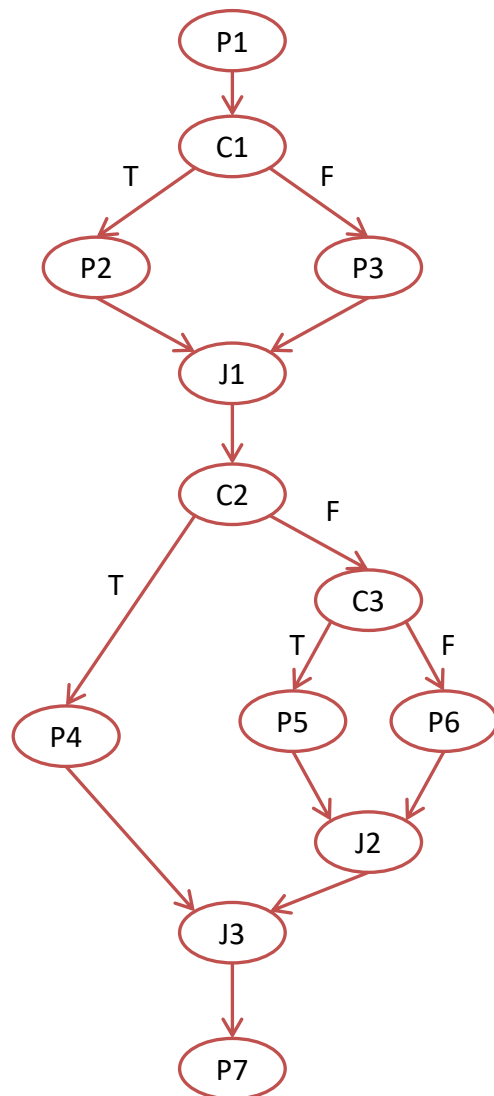
Тестване на управляващия поток: базова концепция



Използва граф на управляващия поток и се цели **пълно покритие на изпълнимите пътища**

При тестването с краен автомат се цели **покритие на състоянията** или **покритие на преходите**

Граф на управляващия поток: структура



❖ Възли

- Програмни единици за обработка на информация (White box) или работен товар (Black box)

❖ Дъги

- Взаимовръзка от типа “е следван от”

❖ Начални и крайни възли

- Възли, в които изпълнението на програмата съответно започва или приключва

❖ Изходна дъга

- Дъга, която започва от определен възел

❖ Входна дъга

- Дъга, която влиза в определен възел
- При наличие на няколко входни дъги за възел изпълнението преминава по една от тях

❖ Възли за взимане на решение

- Възел, който се асоциира с множество изходни дъги (C1÷C3)

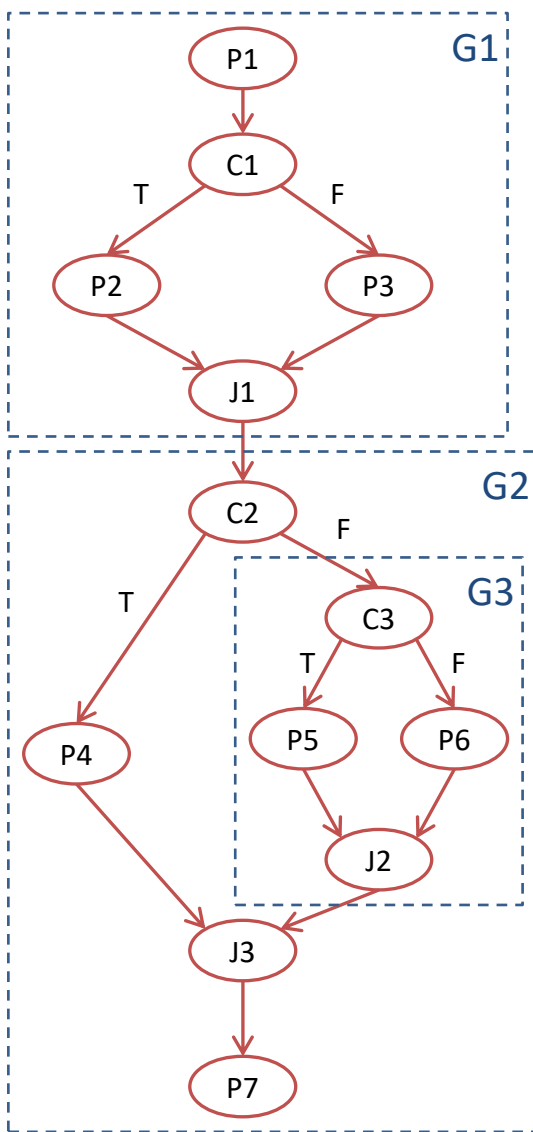
❖ Свързващи възли

- Възел, асоцииран с множество входни дъги (J1÷J3)

❖ Обработващи възли

- Възел, който извършва вътрешна или външна обработка и не е възел за взимане на решение или свързващ възел (P1÷P7)

Граф на управляващия поток: понятия и тестване



❖ Път

- Завършен път от начален до краен възел, преминаващ през множество дъги и междинни възли

❖ Сегмент

- Част от завършен път, при която първият и последният възел може съответно да не са начален и краен възел ($G1 \div G3$)

❖ Цикъл

- Многократно посещение на възли в път или сегмент

❖ Особености

- Ако се допуска обработка във всички възли, то някои от тях могат да се обединят (J1 и C2; J2, J3 и P7)

Базова идея



Последователност на тестване

Създаване и верифициране на граф
(базира се на блок схеми, програмен код,
документи от фазата на проектиране)

Дефиниране и избор на пътища с цел
покрытие на определени тестови
сценарии

Определяне на входни стойности с цел
изпълнение на пътищата

Изготвяне на план за проверка на
резултата

Конструиране на модел при структурно тестване

$$ax^2 + bx + c = 0$$

L1: input(a, b, c)

L2: $d \leftarrow b * b - 4 * a * c$

L3: if (d > 0) then

L4: $r \leftarrow 2$

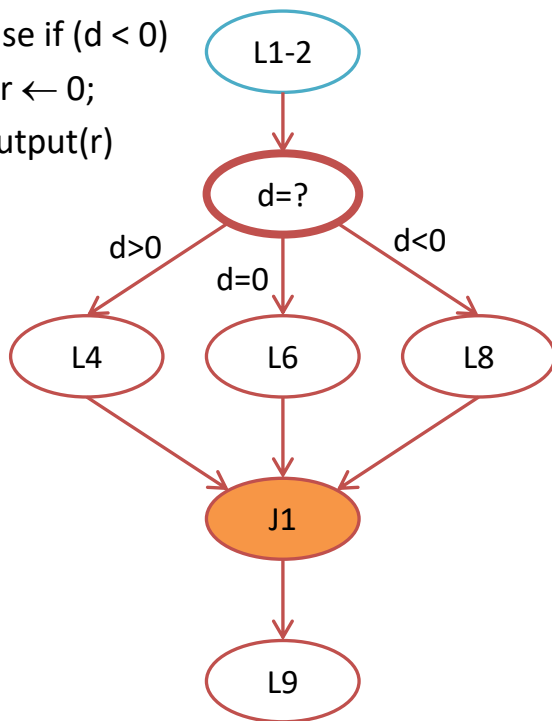
L5: else if (d = 0) then

L6: $r \leftarrow 1$

L7: else if (d < 0)

L8: $r \leftarrow 0$;

L9: output(r)



❖ Последователност на конструиране на граф на управляващия поток

- Асоцииране на обработващите възли с изразите за **присвояване, извикване на процедури или функции**
- Асоцииране на възлите за взимане на решение с изразите за **условен преход** “if-then-else” или “if-then”, или **множествено разклонение** “switch-case”
- Създаване на специален тип възли за разклонение и асоциирането им с **изразите за цикъл**
- Асоцииране на началния и крайния възел на графа с **първия и последния израз в програмата**

❖ Проблеми

- Създаване на граф с прекалено голям брой възли
 - ✓ Обединяване на последователни обработващи възли
- Трудности при представяне на оператора “goto”

Конструиране на модел при функционално тестване

- ❖ Използване на потокови диаграми и случаи на употреба, описани в продуктовата спецификация
- ❖ Извличане на информация от продуктовата спецификация при отсъствие на потокови диаграми:
 - Асоцииране на обработващите възли с **действия**
 - Асоцииране на разклоняващите възли с **условия и взимане на решения**
 - Асоцииране на началните и крайните възли съответно с **първия и последния елемент в спецификацията**
- ❖ Проверка на конструирания модел
 - Липсващи или излишни възли и дъги, анализ на достижимостта

Конструиране на модел при функционално тестване

❖ Продуктово описание на програмата, решаваща уравнението $ax^2 + bx + c = 0$

- За да реши уравнението, потребителят е необходимо да въведе параметри
- Ако $b^2 - 4ac < 0$, то уравнението няма корен и потребителят трябва да бъде информиран
- Ако $b^2 - 4ac = 0$, то коренът на уравнението се изчислява с формулата

$$r = -b/(2a)$$

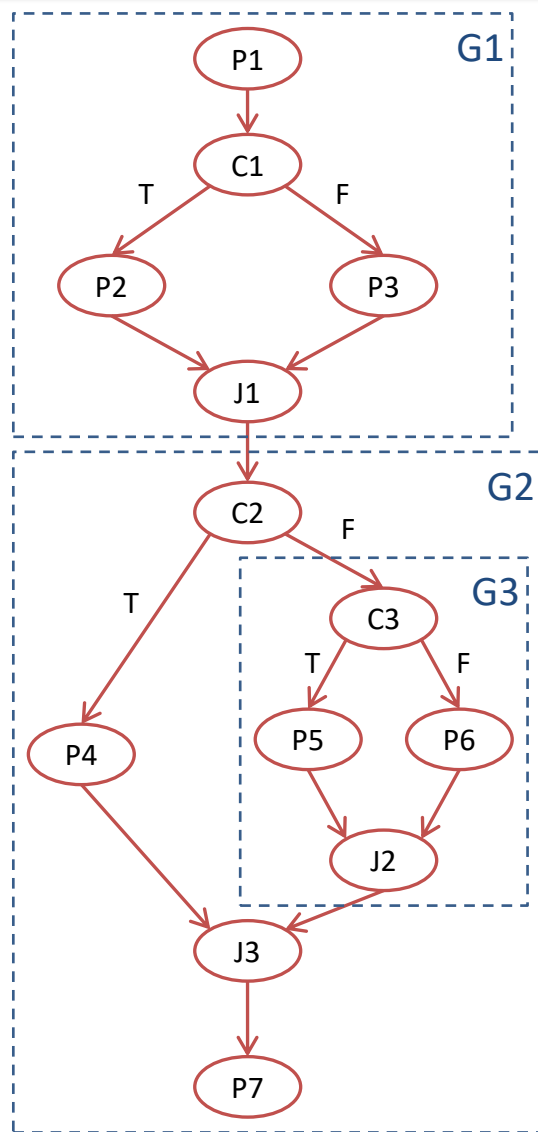
- Ако $b^2 - 4ac > 0$, то корените на уравнението се изчисляват със следната формула

$$r = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

❖ *Графът на управляващия поток е аналогичен на този при пресмятане на броя на корените*

- Фокусът е върху пътищата на изпълнение, а не върху специфичната обработка във възлите

Определяне на път



❖ Базови стъпки за определяне на път

1. Декомпозиция на графа
2. Дефиниция на път отдолу нагоре

❖ Структуриран граф на управляващия поток

- Позволява само последователна конкатенация и влагане с единствени вход и изход

❖ Декомпозиция

- $G = G1 \bullet G2 (-, G3)$; $G3$ е вложен във False разклонението

❖ Брой на пътищата при комбиниране на граф $G1$ с M пътища и граф $G2$ с N пътища

- При последователна конкатенация $G = G1 \bullet G2$ в G са налични $M \times N$ пътища
✓ TT, TF, FT, FF ($2 \times 2 = 4$)
- При влагане $G = G2(G3)$ в G са налични $M + N - 1$ пътища
✓ T, FT, FF ($2 + 2 - 1 = 3$)

❖ Дефиниране на път

- Дефиниране на два пътя в $G3$, съответстващи на $C3=T$ и $C3=F$
- Влагане на пътищата от $G3$ в $G2$ и формиране на 3 пътища: **(1)** $C2=T$ (**T-**); **(2)** $C2=F$ и $C3=T$ (**FT**); и **(3)** $C2=F$ и $C3=F$ (**FF**)
- Конкатениране на $G2(G3)$ с $G1$ и формиране на 6 пътища: **TT-**, **TFT**, **TFF**, **FT-**, **FFT**, **FFF**

❖ Независими условия

- Избор на стойности за променливите, които удовлетворяват специфичните условия за всеки път
 - ✓ При логически променливи в условията инициализацията на пътищата е директна
 - ✓ Пример: $C1 \equiv (x > 0)$, $C2 \equiv (y < 100)$ и $C3 \equiv (z = 10)$ се избират стойности за променливите x , y и z , така че да се удовлетворят условията по всеки път ($x = 1$, $y = 1024$ и $z = 10$ за път TFT)

❖ Зависими условия, определени от споделени логически или числови променливи

- Елиминират се пътищата, които не могат да бъдат инициализирани
 - ✓ При конкатенация на два бинарни подграфа с противоположни условия $C1 = \neg C2$ се елиминират пътищата TT и FF
 - ✓ При конкатенация на два бинарни подграфа с условия $C1 \equiv (x > 0)$ и $C2 \equiv (x < 100)$ се елиминира пътят FF, както следва

$$(C1 = F) \wedge (C2 = F) \equiv \neg (x > 0) \wedge \neg (x < 100) \equiv (x \leq 0) \wedge (x \geq 100) \equiv \emptyset$$

❖ Цикли

- Многократно изпълнение на изрази в програмния код или функции от продуктова спецификация
- Имплицитни (рекурсия)
- Експлицитни (goto)

❖ Тяло на цикъл

- Представя се с възел или вложен граф

❖ Условие за изход от цикъл

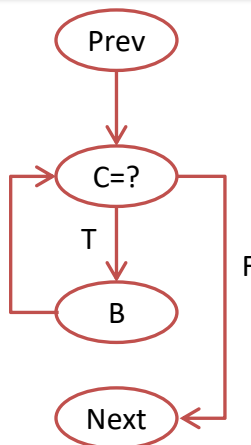
- Представя се с възел, който се асоциира с предикат, определен от управляващи променливи с динамични стойности за взимане на решение

❖ Входен и изходен възел от цикъла

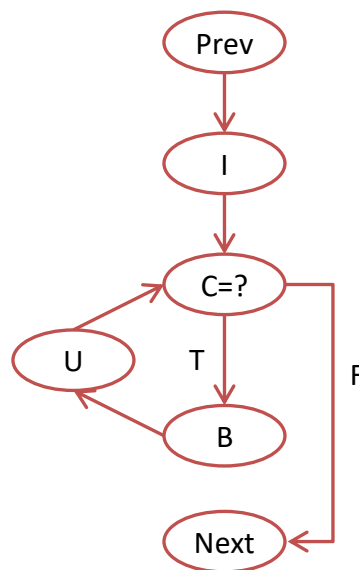
❖ Два или повече цикъла могат да бъдат комбинирани посредством конкатенация и влагане

Цикли в граф на управляващия поток

while



for



❖ Специфициране на цикъл “while”

- “while (C) do { B }”

❖ Специфициране на цикъл “for”

- “for (I; C; U) do { B }”

❖ Типове цикли

- Детерминирани

- ✓ “do (n) { B }”, броят на изпълнение на цикъла е известен (цикъл “for”)

- ✓ Пример: обработка на масив

- Недетерминирани

- ✓ Броят на изпълнение на цикъла зависи от удовлетворяването на условие и е неизвестен (цикъл “while”)

- ✓ Пример: функциониране на операционна система

- Типа на цикъла зависи от програмния език и спецификата на операторите за реализацията им

Проблеми при тестване на цикли

❖ Комбиниране на цикли

- Конкатениране на цикли

- ✓ Броят на циклите при конкатенация е произведение от броя на циклите във всеки цикъл

- Влагане на цикли

- ✓ Вътрешния цикъл се изпълнява N на брой пъти за всяка итерация i на външния цикъл

$$\sum_{i=0}^{M-1} N^i = \frac{N^M - 1}{N - 1}$$

❖ *Пълно покритие е недостижимо*

❖ Приложение на граничното тестване

- Проблеми, свързани с долна граница

- ✓ Инициализация на цикъла, обработка при 0 или 1 елемент

- Проблеми, свързани с горна граница

- ✓ Изпълнение на $N \pm 1$ итерации в цикъла

❖ Приложение на тестване с класове на еквивалентност

- Ако тестовите преминават за цикъл с $N/2$ итерации, то те ще преминат и за $N/2+1$ итерации

❖ Тестване на долна граница на цикъла

- Пропускане на цикъл (**bypass**)
 - ✓ Решава проблеми, свързани с инициализация на променливи извън цикъла и промяна на стойностите им в него
- Еднократно изпълнение на цикъл (**once**)
 - ✓ Решава проблеми, свързани с липса на инициализация на променливи, използвани в цикъла
- Двукратно изпълнение на цикъл (**twice**)
 - ✓ Решава проблеми, които пречат на повторното изпълнение на цикъла
- Изпълнение на минимален брой итерации в цикъл
 - ✓ Решаване на \min и $\min \pm 1$ проблеми

Стратегия за тестване на цикли 2/2

- ❖ Тестване на горна граница N на цикъла
 - Изпълнение на тестове за N-1, N и N+1 итерации
- ❖ Тестване на типични случаи (typical)
- ❖ Тестване на конкатенирани цикли
 - bypass, once, twice, typical, max - 1, max, max + 1 (7 тестови сценария)
 - Броят на тестовите сценарии за два конкатенирани цикъла е 49
- ❖ Тестване на вложени цикли
 - Броят на тестовите сценарии при външен цикъл с горна граница N е 7^N
 - Редуциране на тестовите сценарии
 - ✓ Йерархична стратегия за тестване: тестване на вътрешния цикъл и замяна с единичен възел
 - ✓ Случаен избор на тестов сценарий за вътрешния цикъл

Приложимост на тестването на управляващия поток

- ❖ Тестване с граф на управляващия поток vs. тестване с машина на крайните състояния
 - Брой на генерираните тестови сценарии
 - Покритие на проблеми, свързани с динамично взимане на решение и взаимодействие
 - Приложение при системи с взаимосвързани интензивни изчисления
- ❖ Приложение
 - Структурно тестване на малки програми или тестване на отделни програмни единици
 - За големи софтуерни системи се препоръчва създаване на граф с груба гранулярност
 - ✓ Възлите представят главни функции (black-box) или компоненти (white-box)
 - Статистическо тестване, базирано на употреба
 - ✓ Асоцииране на вероятност за изпълнение на отделните пътища (използване на експлицитен оперативен профил на Муса)

ТЕСТВАНЕ НА ПОТОКА ОТ ДАННИ

Даннови зависимости: операции

❖ Анализ на данновите зависимости

- Анализ на взаимовръзката между променливите

❖ Тестване на данновите зависимости

- Верификация на коректното реализиране на взаимовръзките между програмните променливи

❖ Типове използване на променливи или елементи с данни

- P-use: използване в предикат
- C-use: използване за изчисление

❖ Операции върху даннови променливи

- D-operation: дефиниране на данни посредством създаване, инициализиране, присвояване и др.
 - ✓ Деструктивна операция (стойността на променливата се променя)
- U-operation
 - ✓ P-use или C-use
 - ✓ Недеструктивна операция

Даннови зависимости: релации

❖ D-U relation

- Първоначално дефиниране на променлива и последващо използване

❖ D-D relation

- Двукратно дефиниране на променлива, при което първоначалната ѝ стойност се унищожавя
- Възниква при **припускане** на U-операция или при опит за запис на нова стойност от паралелни процеси
 - ✓ Индикация за проблем или неефективност в софтуера

❖ U-U relation

- Двукратно използване на променлива
- Игнорира се при анализ на данновите зависимости, тъй като засяга реализацията на конкретен изпълним път в софтуера

❖ U-D релация

- Използване и последващо дефиниране на променлива
 - ✓ Създава проблеми, ако променливата не е инициализирана

Тестване на данновите зависимости: концепция

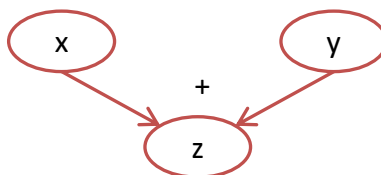
❖ Генериране на тестови сценарии

- Извършва се анализ на данновите зависимости с фокус върху релациите от тип D-U

❖ Граф на данновите зависимости

- Асоцииране на възлите с дефиниции на даннови елементи (променлива, константа, структура)
- Асоцииране на дъгите с релации от тип D-U (“is-used-by”)

✓ $z \leftarrow x + y$



❖ Предимство на тестването на данновите зависимости

- Фокусира се върху проверка на взаимовръзките между данните вместо върху последователността на изчисление

✓ $z \leftarrow x + y$

✓ $i \leftarrow i + 1$



Последователност на тестване

!Създаване и верифициране на граф на данните зависимости

Дефиниране и избор на даннови елементи за създаване на тестови сценарии

Инициализиране на входните променливи

Планиране на проверката на резултата

Граф на данните зависимости: елементи

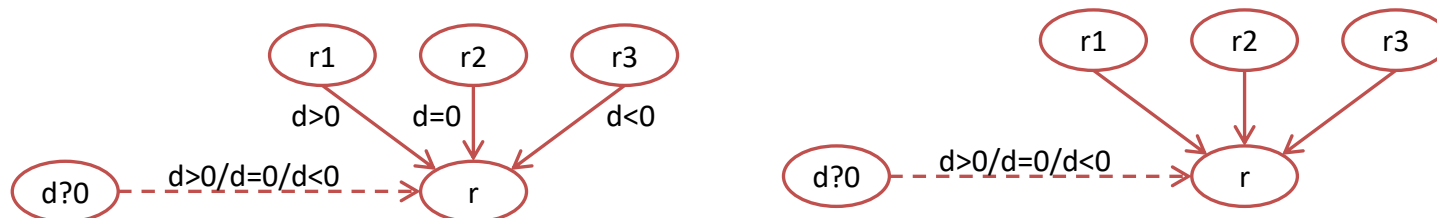
❖ Всеки възел представя дефиниция на даннов елемент x , означена с $D(x)$

- Изходен възел или възел-резултат
 - ✓ Представя изчислителен резултат в програмата
- Входен възел или възел-константа
 - ✓ Представя вход, определен от потребителя или предварително дефинирана константа
- Междинни възли или възли за съхранение
 - ✓ Улесняват изчислителната процедура, правейки по-лесното получаване на резултат от входа на системата

❖ Всички релации в графа са от тип D-U

❖ Селекторен или условен възел

- Представя дефиниция за избор или условие върху определен даннов елемент
 - ✓ Възможните стойности за r за уравнение $ax^2 + bx + c = 0$ се представят с $r1, r2$ и $r3$ и зависят от $d = b^2 - 4ac$

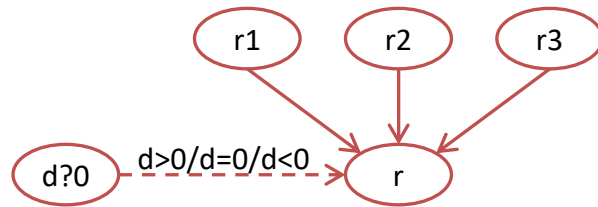


Селекторен възел и характеристики на графа

$(d > 0) \Rightarrow r \leftarrow 2$

$(d = 0) \Rightarrow r \leftarrow 1$

$(d < 0) \Rightarrow r \leftarrow 0$



❖ Типове операции в примера

■ C-use

- ✓ Асоциира се с изчисляване на променливите $r1$, $r2$ и $r3$ и използване на константите 0, 1 и 2

■ P-use

- ✓ Асоциира се с променливата d и константата 0

❖ Характеристики на графа

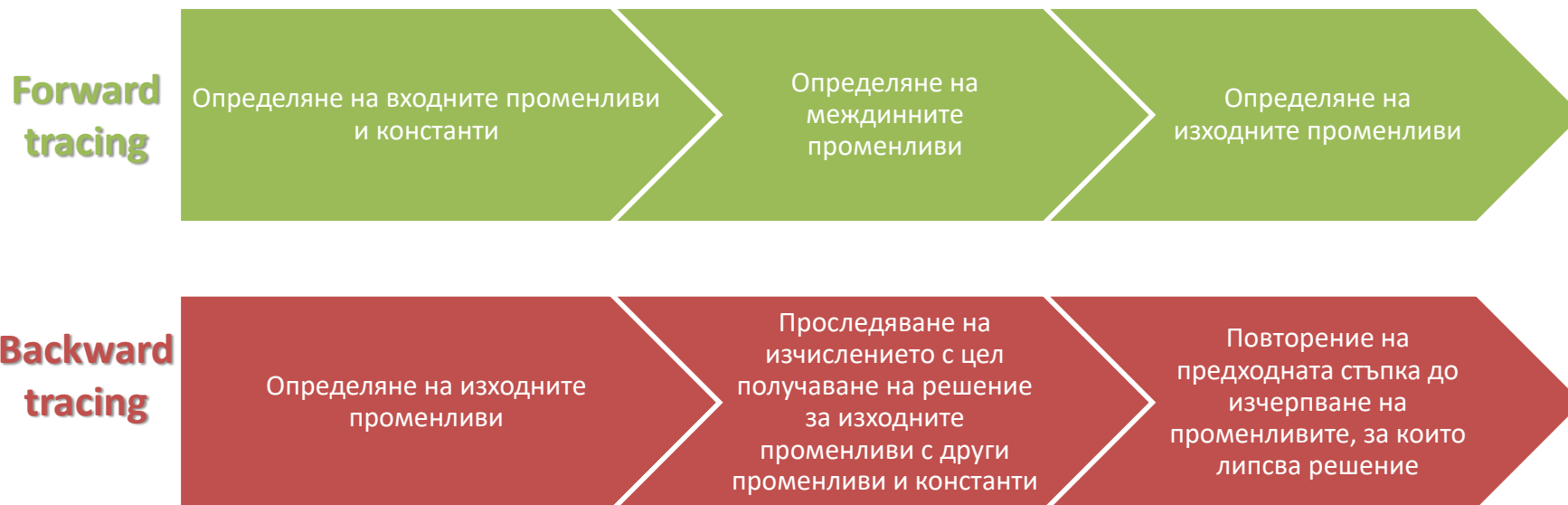
- Наличие на една или малък брой изходни променливи
- Наличие на множество входни променливи и константи
- Наличие на множество входни дъги
- Наличие на “дървовидна” форма на графа

Базова процедура за конструиране на граф

❖ Източници на информация

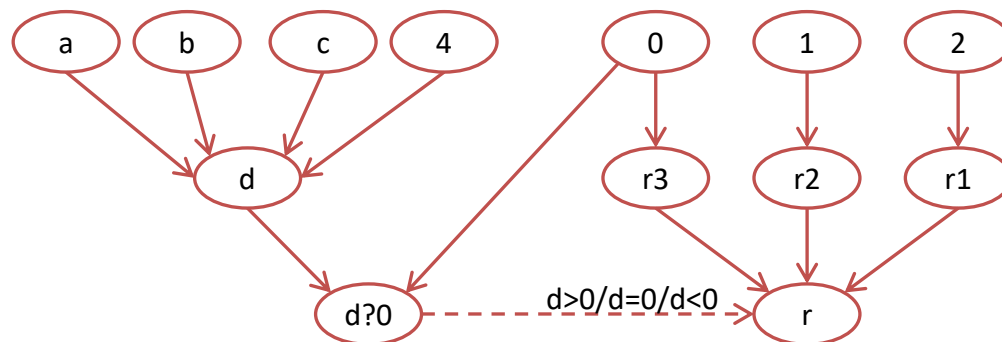
- Програмна реализация на малки компоненти от програмата (white-box)
- Детайлна функционална спецификация (black-box)

❖ Начини за конструиране на граф на данновите зависимости



Конструиране на граф: пример

- ❖ Всички листа на графа са входни променливи или константи
- ❖ Изчисляване на корените на уравнението $ax^2 + bx + c = 0$, $d = b^2 - 4ac$
 - $(d > 0) \Rightarrow r \leftarrow 2$
 - $(d = 0) \Rightarrow r \leftarrow 1$
 - $(d < 0) \Rightarrow r \leftarrow 0$



❖ Приложение

- Идентифициране на променливи в спецификацията или реализацията, които не са включени в граф на данновия поток
 - ✓ Даннови проблеми или загуба на ресурси
- Идентифициране на възли, които не са свързани с път до изходните възли
 - ✓ Загуба на ресурси за конструиране на ненужни пътища

Индиректно конструиране на граф

- ❖ Идентифициране на променливите и константите x_1, x_2, \dots, x_n , използвани за дефиниране на текущ възел y с операция $D(y)$:

$$y \leftarrow (x_1, x_2, \dots, x_n)$$

- ❖ За всеки даннов елемент x_i се извършва връщане към последната му дефиниция посредством идентифициране на двойка D-U за x_i .
- ❖ Ако $D(y)$ не е в разклонение, то се създава дъга между възел x_i и възел y
- ❖ Ако $D(y)$ е в условно разклонение, се изпълнява следното
 - Анализира се ситуация **“blockl; if C then A else B”**
 - Създават се последователни подграфи **“blockl; A”** (y_1) и **“blockl; B”** (y_2) за всяко разклонение
 - Създава се подграф за условен селектор **“blockl; C”**
 - Селекторният възел се използва от y за избор на даннова дефиниция y_1 или y_2
 - ✓ Дефинициите y_1 и y_2 могат да бъдат директно свързани със селекторния възел y
 - ✓ Управляващата входна дъга към y изхожда от селекторния подграф

Примери

❖ Липса на “else” клауза

input (x)

$y \leftarrow x;$

if (x < 0) then $y \leftarrow -x;$ *у има 2 възможни стойности в зависимост от условието $x < 0$!*

output(y)

❖ Използване на няколко променливи при наличие на разклонение

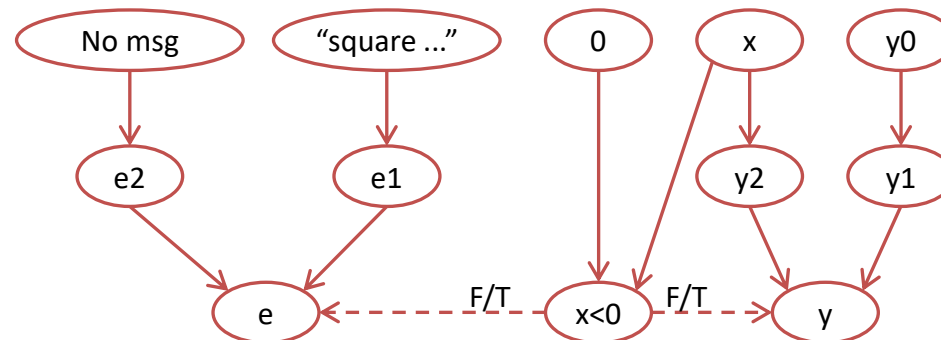
input (x)

if (x < 0) then

exit(“square root undefined for negative numbers”);

else $y \leftarrow \text{sqrt}(x)$

return(y)



Представяне на цикли

❖ Особенности и проблеми при представянето на цикли с граф на данните зависимости

- Наличие на данни, които се споделят между итерациите на цикъла
- Невъзможност да се извърши пълен даннов анализ дори и при цикли с ограничен брой итерации
- Циклите в реалната реализация могат да не съответстват на цикли в концептуалните модели или функционалните спецификации
 - ✓ Пример: сума на елементите в масив

$$S = \sum_{i=1}^n A[i]$$

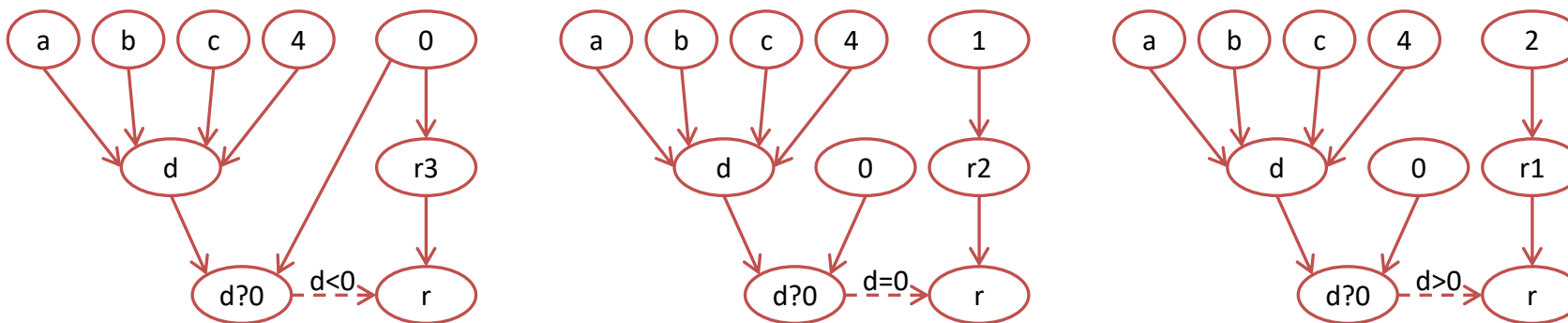
- ✓ Фокусиране върху концептуалните зависимости между S и A, вместо върху индивидуалните елементи A[i]

❖ Възможни решения

- Прилагане на двуфазната стратегия за тестване
 - ✓ Тестване на цикъла с граф на управляващия поток
 - ✓ Замяна на цикъла с единичен изчислителен възел от вида “ $S \leftarrow \text{arraysum}(A)$ ” и изпълнение на тестване с граф на потока от данни
- Представяне на цикъла с един или два вложени “if” оператори
 - ✓ Преобразуване на “while C do B” в “if C then {B; if C then B}”

Тестово покритие на подграфи

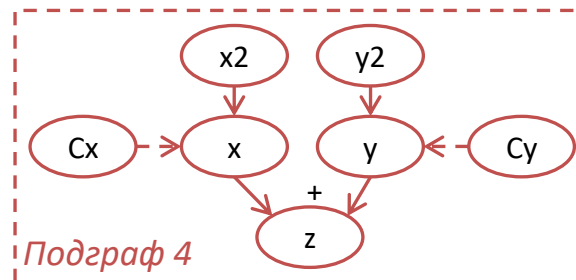
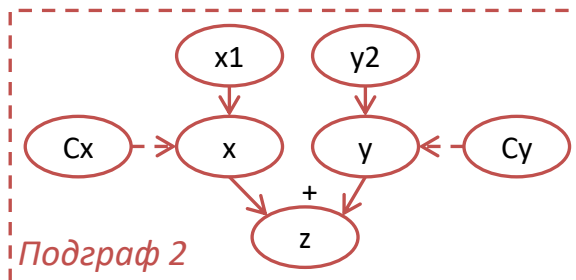
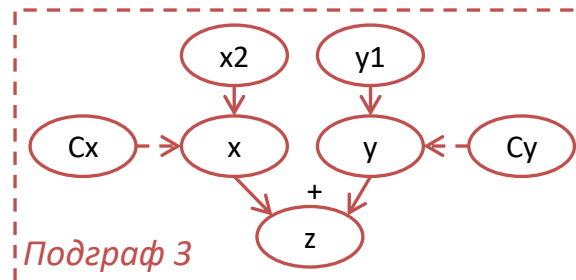
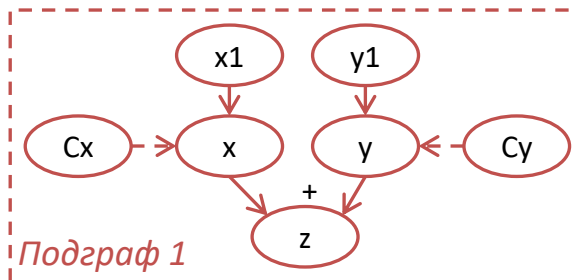
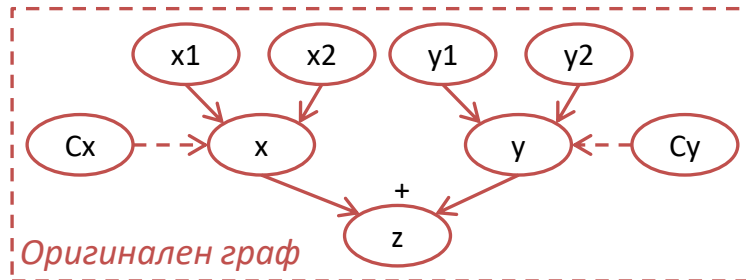
- ❖ Подграф в граф на потока от данни
 - Специфична стойност на изходна променлива от специфични стойности на входни променливи и константи
- ❖ Създаване на един подграф за покритие на всички входни променливи и константи
 - Наличие на една изходна променлива и липса на селекторни възли
- ❖ Създаване на няколко подграфи за всяка входна даннова дъга на селекторния възел
 - Наличие на селекторни възли
- ❖ Формиране на подграфи за селекторния възел при изчисляване на корените на уравнението $ax^2 + bx + c = 0$, $d = b^2 - 4ac$



Комбинация от независими даннови селектори

❖ Изчисляване на израз $z \leftarrow x + y$

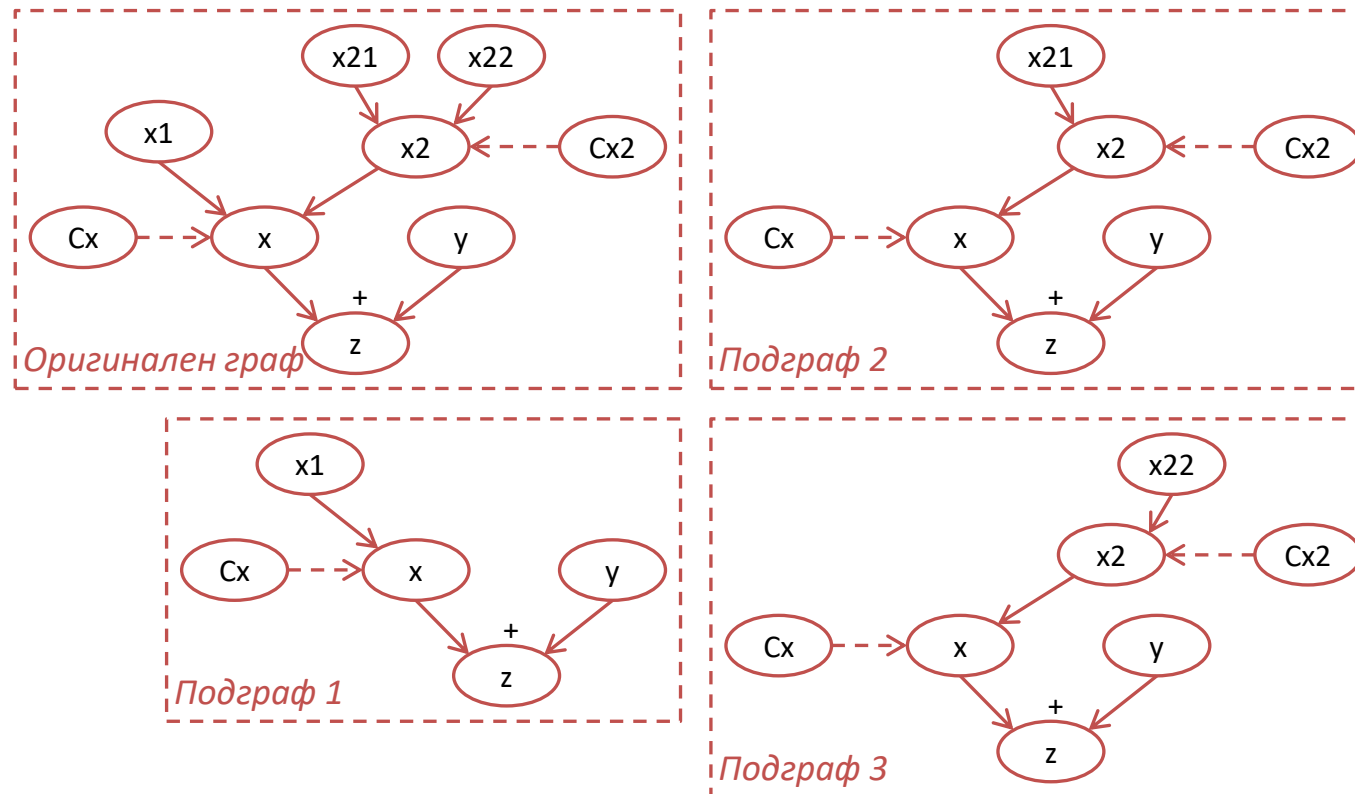
- Кандидат стойности: x_1, x_2, y_1 , и y_2
- Комбиниране на условията за x и y и дефиниране на подграфи



Комбинация от вложени даннови селектори

❖ Изчисляване на израз $z \leftarrow x + y$

- Наличие на селектор за стойности x_1 и x_2
- Наличие на селектор за стойности x_{21} и x_{22}
- Липса на селектор за y



Инициализация на променливи

- ❖ Инициализация на входни променливи и константи, включени в подграфа
 - Ако променливата е включена в предикат, то нейната стойност трябва да осигури подходящо изчисляване на предиката (прилага се стратегия с връщане назад)
 - Ако променливата е включена като даннов вход, то инициализацията е с произволна стойност
- ❖ Инициализация на променливи и константи, които не са включени в подграфа
 - Инициализацията е с произволна стойност, тъй като променливите не влияят на изчислителния резултат

Граф на даннов поток vs. Граф на управляващ поток

Граф на управляващия поток

- Представлява специален тип машина на крайните състояния
- Представя програмния код или потока на изпълнение, асоцииран с последователните изчислителни модели
- Характеризира се с по-малка сложност
- По-малки ограничения при представяне на цикли

Граф на данновия поток

- Различава се значително от машината на крайните състояния
- Представя детайлите на взаимодействието и данните зависимости
- Характеризира се с по-голяма сложност
- По-големи ограничения при представяне на цикли

Приложение на тестването с граф на данновия поток

❖ Типични приложения на двете техники

- Тестване на малки програми
- Тестване на малки модули от големи системи
- Тестване на големи системи с груба гранулярност на представяне на операциите

❖ Йерархична тестова стратегия

- Комбиниране на тестването с управляващ поток и тестването с даннов поток
 - ✓ Използване на CFT за циклите
 - ✓ Изпълнение на CFT и последващо изпълнение на DFT

❖ Структурно тестване

- Фокусиране върху детайлна информация, налична в програмния код (CFG)

❖ Функционално тестване

- Фокусиране върху резултата (DFG)

❖ Статистическо тестване, базирано на употреба

- При извършване на йерархично тестване, по-важните даннови подграфи или тези с по-голяма вероятност за използване се развиват по-детайлно
- Вземане на решение за представяне на циклите

❖ Анализ на паралелни и разпределени системи

- Прихващане на зависимости между различни системни задачи и идентифициране на възможности за извършване на паралелни изчисления

Приложение при тестване на синхронизация

❖ Дефиниране на синхронизация

- Изчисление на задача $y \leftarrow f(x_1, x_2, \dots, x_n)$
- Интерпретиране на като x_i паралелна задача
- Условие за синхронизация
 - ✓ Приключване на всички задачи x_i преди да приключи y

❖ Коректното изпълнение на синхронизацията включва два елемента:

- Получаване на коректен резултат за y или получаване на подходяща y обработка за x_i
- Синхронизация на получаването на резултат от x_i
 - ✓ Последователно или паралелно пристигане във времето

❖ Тестване на синхронизация

- Тества се различен ред на получаване резултат от x_i и проверка за коректен резултат
- Синхронизация на А и В за получаване на С
 - ✓ Липса на изход поради липса на резултат от А и В
 - ✓ Липса на изход поради липса на резултат от А или В (2 тестови случая)
 - ✓ Наличие на изход (3 тестови случая в зависимост реда, в който се получават резултати от А и В)
- Ограничаване на броя на тестовите сценарии посредством създаване на групи от x_i
 - ✓ Тестване на синхронизацията в подгрупите и последващо тълкуване на подгрупите като единични входи

