آزمون نرم افزار – فصل۵ بخش ۵–۲

آزمایش مبتنی بر نحو برای گرامرهای مبتنی بر برنامه

صدیقه خوشنویس
دانشگاه آزاد اسلامی – واحد شهرقدس

آزمایش مبتنی بر نحو برای برنامه ها

- آزمایش مبتنی بر نحو از ابتدا برای برنامه ها طراحی شد و اغلب برای برنامه ها به کار می رود.

- معیارهای BNF بیشتر برای آزمایش کامپایلرها استفاده می شود.
  - آزمایش کامپایلرها بسیار پیچیده است.
  - معیارهای BNF می تواند به تولید برنامه برای آزمایش همه امکانات زبان برنامه نویسی کمک کند.
  - این یک کاربرد خیلی خاص است که درباره اش در این درس وارد جزئیات بیشتری نمی شویم.

- معیارهای آزمایش جهش بیشتر برای آزمایش واحد (unit testing) و آزمایش یکپارچه سازی (integration testing) کلاسها به کار می رود.

گرامرهای مبتنی بر برنامه

- کاربرد اولیه و وسیع ترین کاربرد از آزمایش مبتنی بر نحو، تغییر دادن برنامه ها است.
- اپراتورهای جهش، یک رشته پایه (که همان برنامه تحت تست است) را تغییر می دهند تا برنامه های جهشگر را ایجاد کنند.
- برنامه های جهشگر باید به درستی کامپایل شوند (رشته معتبر باشند).
- جهشگرها آزمون نیستند، ولی برای یافتن آزمونها مورد استفاده قرار می گیرند.
- وقتی جهشگرها تعریف شدند، باید آزمونهایی را بیابیم که موجب خرابی در اجرای برنامه شود.
- و این یعنی کشتن جهشگرها.

کشتن جهشگرها

اگر $m \in M$ جهشگری برای یک برنامه P (به عنوان رشته پایه)، و t یک آزمون باشد، می‌گوییم t، m را می کُشد اگر و فقط اگر خروجی t روی P با خروجی آن روی m متفاوت باشد.

- اگر اپراتورهای جهش به خوبی طراحی شده باشند، آزمونهای به دست آمده بسیار قوی خواهند بود.
- برای زبانهای مختلف و اهداف متفاوت، باید اپراتورهای مختلفی را تعریف کرد.
- تا وقتی که همه جهشگرها کشته شوند، آزمونگر به آزمایش ادامه می دهد.
  - جهشگر مرده (dead mutant): جهشگری که یک آزمون آن را کشته است.
  - جهشگر مرده به دنیا آمده (stillborn mutant): جهشگری که از نظر نحوی صحیح نیست.
  - جهشگر بدیهی (trivial mutant): تقریباً هر آزمونی می تواند آن را بکشد.
  - جهشگر معادل (equivalent mutant): هیچ آزمونی نمی تواند آن را بکشد (معادل برنامه اصلی است).

گرامرهای مبتنی بر برنامه

متد اولیه

```
int Min (int A, int B)
{
        int minVal;
        minVal = A;
        if (B < A)
        {
            minVal = B;
        }
        return (minVal);
} // end Min
```

متد با جاسازی چند جهشگر

```
int Min (int A, int B)
{
        int minVal;
        minVal = A;
Δ 1     minVal = B;
        if (B < A)
Δ 2     if (B > A)
Δ 3     if (B < minVal)
        {
            minVal = B;
Δ 4         Bomb ();
Δ 5         minVal = A;
Δ 6         minVal = failOnZero (B);
        }
        return (minVal);
} // end Min
```

جایگذاری یک متغیر به جای دیگری

تغییر عملگر

ایجاد خرابی آنی در زمان اجرا به محض رسیدن به این دستور

ایجاد خرابی آنی در زمان اجرا به شرط آن که B==0 شود. در غیر این صورت کاری نمی کند.

۶ جهشگر که هر یک، نشان دهنده برنامه مجزایی است.

معیارهای پوشش مبتنی بر نحو

<u>پوشش جهش (Mutation Coverage) (MC):</u>
برای هر جهشگر $m \in M$، TR دقیقاً دارای یک نیازمندی است: کشتن m.

- **یادآوری مدل RIP از فصل ۱:**
  - **دسترس پذیری (Reachability): آزمون موجب می شود که عبارت دارای نقص، <u>مورد دسترسی</u> قرار گیرد (در جهش: عبارت جهش یافته).**
  - **آلوده سازی (Infection): آزمون موجب می شود که عبارت دارای نقص منجر به یک <u>حالت اشتباه</u> شود.**
  - **انتشار (Propagation): <u>حالت اشتباه به خروجی</u> منتشر شده و منجر به مشاهده خروجی اشتباه می‌شود.**
- **مدل RIP منجر به دو گونه پوشش جهش می شود: کشتن قوی جهشگرها و کشتن ضعیف جهشگرها**

معیارهای پوشش مبتنی بر نحو...

- ۱) کشتن قوی جهشگرها
- اگر $m \in M$ جهشگری برای برنامه P، و t یک آزمون باشد، می‌گوییم t، m را به طور قوی می‌کُشد اگر و فقط اگر خروجی t روی P با خروجی آن روی m متفاوت باشد.

- ۲) کشتن ضعیف جهشگرها
- اگر $m \in M$ جهشگری برای برنامه P باشد که محل l از آن برنامه را تغییر داده باشد، و t یک آزمون باشد، می‌گوییم t، m را به طور ضعیف می کُشد اگر و فقط اگر حالت اجرای P بلافاصله بعد از l با استفاده از t متفاوت با حالت اجرای m بلافاصله بعد از l با استفاده از t باشد.

  – در واقع، کشتن ضعیف، دسترس پذیری و آلوده سازی را برآورده می کند ولی انتشار را خیر.

جهش ضعیف

پوشش جهش ضعیف (Weak Mutation Coverage) (WMC):
برای هر جهشگر $m \in M$، TR دقیقاً دارای یک نیازمندی است: کشتن m به طور ضعیف.

- جهش ضعیف، از آن جهت به این شکل نامگذاری شده که کشتن جهشگرها در آن آسانتر است و به تحلیل کمتری هم نیاز دارد.
- برخی جهشگرها را می توان به طور ضعیف کشت، ولی نمی توان به طور قوی کشت (دارای خاصیت انتشار نیستند)!
- در عمل، تفاوت آنها بسیار کم است.

مثال از جهش (قوی)

- جهشگر ۱ در مثال Min() به این شکل است:

```
     minVal = A;
Δ 1  minVal = B;
     if (B < A)
        minVal = B;
```

- مشخصات کامل آزمونی که بتواند جهشگر ۱ را بکشد به شرح زیر است:

- ۱) شرط دسترس پذیری: true
  - (همیشه برقرار است؛ همواره به این عبارت می رسیم)
- ۲) شرط آلوده سازی: $A \neq B$
  - (اگر A و B برابر باشند، جهشگر، معادل برنامه اصلی است و هرگز کشته نمی شود)
- ۳) شرط انتشار: $(B < A) = false$ (اجرای برنامه از انتساب بعد از دستور if بپرد)
  - (اگر false نباشد، باز هم آنچه به خروجی می رسد، معادل برنامه اصلی خواهد بود)

- مشخصات کامل آزمون:

$$true \wedge (A \neq B) \wedge ((B < A) = false)$$
$$\equiv (A \neq B) \wedge (B \geq A)$$
$$\equiv (B > A)$$

بنا بر این آزمونی مانند (A = 5, B = 7) جهشگر ۱ را «قویاً» خواهد کشت.

مثال از جهش (ضعیف)

- باز هم برای جهشگر ۱ در مثال Min():

```
     minVal = A;
Δ 1  minVal = B;
     if (B < A)
        minVal = B;
```

- گفتیم مشخصات کامل آزمونی که بتواند جهشگر ۱ را بکشد به شرح زیر است:

- ۱) شرط دسترس پذیری: true
  - (همیشه برقرار است؛ همواره به این عبارت می رسیم)
- ۲) شرط آلوده سازی: $A \neq B$
  - (اگر A و B برابر باشند، جهشگر، معادل برنامه اصلی است و هرگز کشته نمی شود)

- مشخصات کامل آزمون:

$$true \wedge (A \neq B)$$
$$\equiv (A \neq B)$$

بنا بر این آزمونی مانند (A = 2, B =1) (و البته آزمون اسلاید قبل یعنی (A = 5, B = 7)) جهشگر ۱ را «به طور ضعیف» خواهد کشت.

/واضح است که آزمونی که بتواند جهشگری را به طور قوی بکشد، می تواند آن را به طور ضعیف هم بکشد.

مثال از جهشگر معادل

```
     minVal = A;
     if (B < A)
Δ 3  if (B < minVal)
```

- جهشگر ۳ در مثال **Min()** به این شکل است:

- شرط آلوده سازی: **(B < A) != (B < minVal)**

- ولی عبارت قبلی **minVal = A** بود. اگر جایگذاری کنیم خواهیم داشت:

(B < A) != (B < A)

- واضح است که هیچ آزمونی نمی تواند آن را بکشد!

جهش قوی در برابر جهش ضعیف

برنامه تشخیص زوج بودن یک عدد

```
1     boolean isEven (int X)
2     {
3         if (X < 0)
4             X = 0 - X;
Δ 4           X = 0;
5         if (float) (X/2) == ((float) X) / 2.0
6             return (true);
7         else
8             return (false);
9         }
```

دسترس پذیری: $X < 0$

آلودگی: $X \mathrel{!}= 0$

آزمونی مثل ($X = -6$) جهشگر ۴ را با جهش ضعیف خواهد کشت.

انتشار: باید اثر آلودگی را به خروجی برسانیم. اگر خروجی true باشد، با توجه به این که صفر زوج است، خروجی درستی است. پس خروجی باید false باشد یعنی شرط if هم باید false باشد.

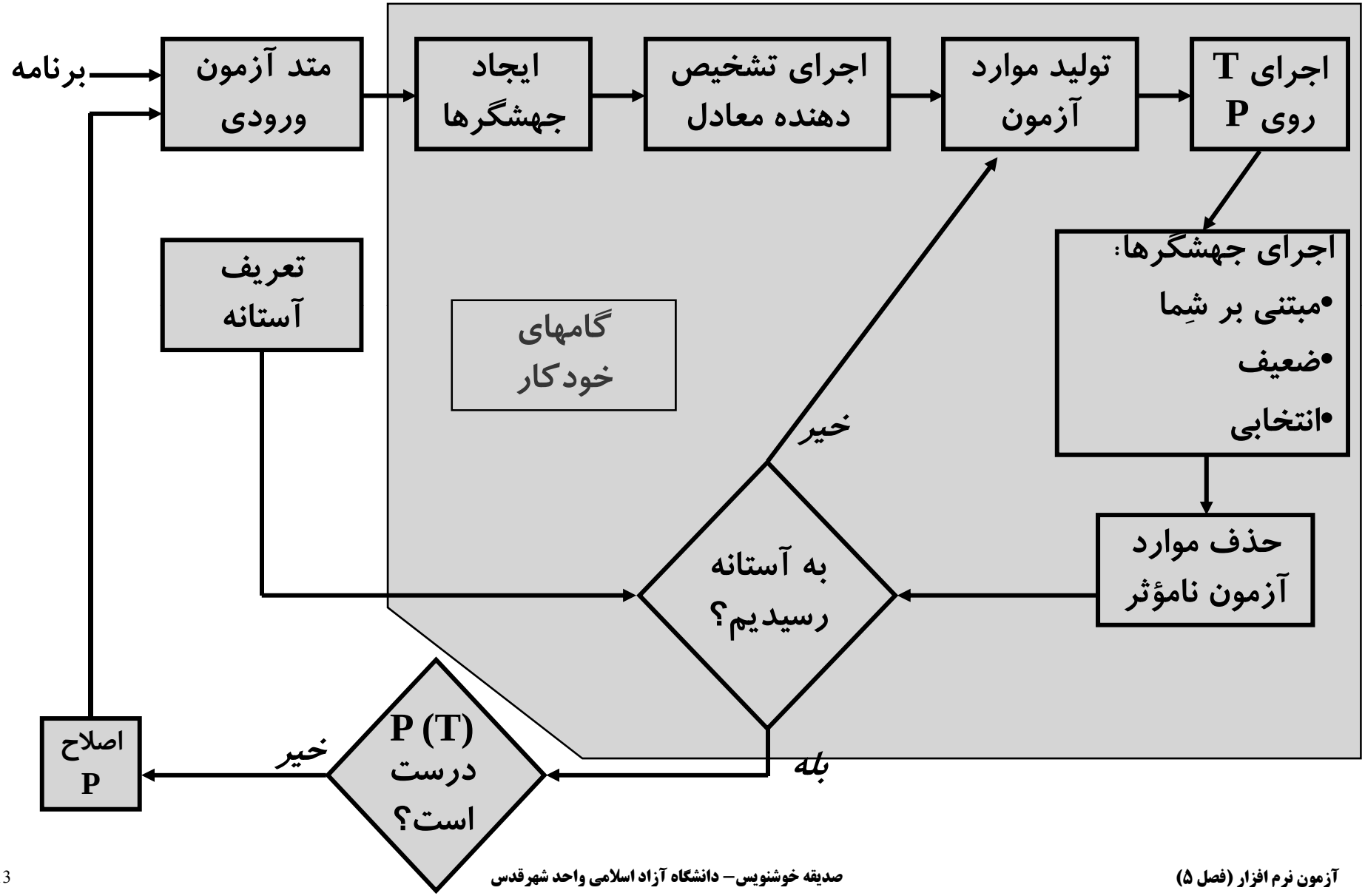
$$((float)\ ((0-X)/2) == ((float)\ 0-X) / 2.0)$$

$$\mathrel{!=}\ ((float)\ (0/2) == ((float)\ 0) / 2.0)$$

یعنی X زوج نباشد.

پس ($X = -6$) جهشگر را با جهش قوی نمی‌کشد.

آزمایش برنامه ها با جهش



چرا جهش کار می کند؟

قضیه بنیادی آزمایش جهش

اگر نرم افزار دارای نقصی باشد، معمولاً مجموعه ای از جهشگرها وجود دارند که تنها با مورد آزمونی می توانند کشته شوند که آن نقص را نیز تشخیص می دهد.

- **این موضوع، مطلق نیست!**
- **جهشگرها آزمونگر را به مجموعه ای از آزمونهای بسیار موثر هدایت می کند.**
- **یک مساله بسیار چالش برانگیز، یافتن یک نقص و مجموعه ای از آزمونهای کافی برای جهش است که آن نقص را پیدا نکنند.**
- **البته همه چیز به اپراتورهای جهش بستگی دارد.**

طراحی اپراتورهای جهش

- در سطح متد، اپراتورهای جهش برای زبانهای مختلف برنامه نویسی شبیه همند.

- این اپراتورها یکی از دو کار زیر را انجام می دهند:
  - تقلید اشتباه های رایج برنامه نویسان (مثلاً تغییر نام متغیر)
  - استفاده از روشهای ابتکاری رایج آزمون (مثلاً کاری کنیم مقدار متغیری 0 شود)

- محققان اپراتورهای زیادی را طراحی کرده اند و سپس به طور تجربی مفیدترین آنها را انتخاب کرده اند.

<u>اپراتور مؤثر جهش</u>

اگر آزمونهایی که برای کشتن یک سری جهشگر که با مجموعه ای از اپراتورهای جهش مثلاً $O = \{o1, o2, \ldots\}$ ایجاد شده اند، بتوانند جهشگرهای ایجاد شده توسط همه اپراتورهای جهش دیگر را هم -با احتمال بالا- بکشند، آنگاه O مجموعه ای مؤثر از اپراتورهای جهش خواهد بود.

اپراتورهای جهش برای جاوا

۱. درج قدر مطلق (*ABS — Absolute Value Insertion*) :

هر عبارت (و زیر عبارت) حسابی را با توابع *abs()*، *negAbs()* و *failOnZero()* تغییر دهید.

متد *failOnZero()* اپراتور جهش مخصوصی است که اگر پارامتر آن صفر باشد، موجب خرابی می شود و در غیر این صورت کاری انجام نمی دهد. این یعنی آزمونگر سعی کند هر متغیر یا عبارت حسابی را با صفر مقدار دهی کند.

۲. جایگزینی عملگر حسابی (*AOR — Arithmetic Operator Replacemen*) :

هر عملگر حسابی +، -، *، / و ٪ هر بار که در برنامه ظاهر شده است با هر یک از عملگرهای حسابی دیگر و به علاوه، با عملگرهای *leftOp()* و *rightOp()* جایگزین می شود.

متد *leftOp()* عملوند سمت چپ یک عبارت حسابی و *rightOp()* عملوند سمت راست آن را بر می گرداند.

۳. جایگزینی عملگر رابطه ای (*ROR — Relational Operator Replacement*) :

هر عملگر رابطه ای ($<, \leq, >, \geq, =, \neq$) هر بار که در برنامه ظاهر شده است با هر یک از عملگرهای رابطه‌ای دیگر و به علاوه، با عملگرهای *falseOp()* و *trueOp()* جایگزین می شود.

متد *falseOp()* مقدار false و *trueOp()* مقدار true را بر می گرداند.

اپراتورهای جهش برای جاوا...

۴. جایگزینی عملگر شرطی (*COR — Conditional Operator Replacement*) :

هر عملگر منطقی (and، or، xor و ...) هر بار که در برنامه ظاهر شده است با هر یک از عملگرهای منطقی دیگر و به علاوه، با عملگرهای *trueOp()*، *falseOp()*، *leftOp()* و *rightOp()* جایگزین می‌شود.

A	B	A|B	A&B	A^B	!A
false	false	false	false	false	true
true	false	true	false	true	false
false	true	true	false	true	true
true	true	true	true	false	false

- **| = OR**
- **& = AND**
- **^ = XOR**
- **! = NOT**
- **|| = OR با محاسبه میانبر**
- **&& =AND با محاسبه میانبر**
- **== = تساوی**
- **!= = نامساوی**

اپراتورهای جهش برای جاوا...

۵. جایگزینی عملگر شیفت (*SOR — Shift Operator Replacement*) :

هر عملگر شیفت (>>، <<، و <<<) هر بار که در برنامه ظاهر شده است با هر یک از عملگرهای شیفت دیگر و به علاوه، با عملگر *leftOp()* جایگزین می‌شود.

مثال: **20>>2 = 5**

20 در سیستم دودویی =

00000000000000000000000000010100

همه بیتها را دو تا به راست شیفت دهید:

00000000000000000000000000000101

که برابر است با ۵ (2*2^2+0*2^1+1*2^0)

شیفت به راست = >>

شیفت به چپ = <<

شیفت به راست بدون علامت = <<<

۶. جایگزینی عملگر منطقی (*LOR — Logical Operator Replacement*) :

هر عملگر منطقی (and بیتی (&)، or بیتی (|)، و xor بیتی (^)) هر بار که در برنامه ظاهر شده است با هر یک از عملگرهای منطقی دیگر و به علاوه، با عملگرهای *leftOp()* و *rightOp()* جایگزین می‌شود.

اپراتورهای جهش برای جاوا...

۷. جایگزینی عملگر انتساب (*ASR — Assignment Operator Replacement*) :

هر عملگر انتساب (=,+=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=, >>>=) هر بار که در برنامه ظاهر شده است با هر یک از عملگرهای انتساب دیگر جایگزین می‌شود.

x oper= y برابر است با x = x oper y مثلاً : x+=y برابر است با x=x+y

۸. درج عملگر یکانی (*UOI — Unary Operator Insertion*) :

هر عملگر یکانی (تک عملوند) (+ حسابی، - حسابی، ! شرطی، ~ منطقی) قبل از هر عبارت مناسب (از نوع درست) درج می شود.

عملگر یکانی ~ مکمل بیتی می گیرد (تبدیل صفر به یک و برعکس) و فقط روی مقادیر صحیح (int) استفاده می‌شود. همواره این رابطه برقرار است: ~x==(-x)-1

۹. حذف عملگر یکانی (*UOD — Unary Operator Deletion*) :

هر عملگر یکانی (تک عملوند) (+ حسابی، - حسابی، ! شرطی، ~ منطقی) حذف می شود.

اپراتورهای جهش برای جاوا...

۱۰. جایگزینی متغیر عددی (*SVR — Scalar Variable Replacement*) :

هر ارجاع به هر متغیر با همه متغیرهای دیگر، که دارای نوع مناسب بوده و در حیطه فعلی اعلان شده باشند، جایگزین می شود.

۱۱. جایگزینی عبارت بمب (*BSR — Bomb Statement Replacement*) :

هر عبارت با تابع مخصوص *Bomb()* جایگزین می شود.

تابع *Bomb()* به محض اجرا، ایجاد خرابی می کند؛ بنا بر این می توان از آن برای چک کردن رسیدن به هر عبارت استفاده کرد. مثلاً برای عبارتی مثلِ x=a*b; ، آن را به عبارت Bomb(); جهش می دهیم.

خلاصه اپراتورهای جهش

شماره	کد	عملکرد	شماره	کد	عملکرد
۱	**ABS (Absolute) قدر مطلق**	درج عملگر Abs()، negAbs() و failOnZero() برای هر عبارت حسابی	۷	**ASR (Assignment) انتساب**	جایگزینی عملگرهای انتساب +=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=, >>>= با هم
۲	**AOR (Arithmetic) حسابی**	جایگزینی عملگرهای حسابی +، -، *، / و % با هم و با leftOp() و rightOp()	۸	**UOI (Unary Insert) درج یکانی**	درج عملگرهای یکانی (+ حسابی، - حسابی، ! شرطی، ~ منطقی) قبل از هر عبارت مناسب
۳	**ROR (Relational) رابطه ای**	جایگزینی عملگرهای رابطه ای <, ≤, >, ≥, =, ≠ با هم و با trueOp() و falseOp()	۹	**UOD (Unary Delete) حذف یکانی**	حذف هر عملگر یکانی (+ حسابی، - حسابی، ! شرطی، ~ منطقی)
۴	**COR (Conditional) شرطی**	جایگزینی عملگرهای شرطی and، or، xor و ... با هم و با leftOp() و rightOp() و trueOp() و falseOp()	۱۰	**SVR (Variable) متغیر**	جایگزینی هر متغیر با همه متغیرهای مناسب دیگر
۵	**SOR (Shift) شیفت**	جایگزینی عملگرهای شیفت >>، <<، و <<< با هم و با leftOp()	۱۱	**BSR (Bomb) بمب**	جایگزینی هر عبارت با دستور Bomb()
۶	**LOR (Logical) منطقی**	جایگزینی عملگرهای منطقی and بیتی (&)، or بیتی (|)، و xor بیتی (^) با هم و با leftOp() و rightOp()			

رابطه در بر داشتن با سایر معیارها

- جهش قویترین و گرانترین معیار آزمایش محسوب می شود.

- سایر معیارها را در بر دارد (البته با جهش ضعیف)
  - پوشش گره
  - پوشش یال
  - پوشش کلاز
  - پوشش GACC
  - پوشش CACC
  - پوشش همه تعاریف (all-defs) در جریان داده

پایان جلسه نهم