درک آزمون جهش در پایتون با یک پیاده‌سازی عملی

شقایق شهبازی[۱]، مریم سادات صفوی[۲] و ریحانه خرمیان[۳]

[۱] دانشجو کارشناسی، مهندسی کامپیوتر، دانشگاه اصفهان
[۲] دانشجو کارشناسی، مهندسی کامپیوتر، دانشگاه اصفهان
[۳] دانشجو کارشناسی، مهندسی کامپیوتر، دانشگاه اصفهان

چکیده: در این مقاله، یک روش نوین برای اجرای آزمایش جهش در پروژه‌های به زبان پایتون ارائه شده است. از مفهوم درخت نحوی انتزاعی به همراه جایگذاری عملگرهای ریاضی در کدها استفاده شده است. هدف این پژوهش ارتقاء کیفیت آزمایش‌ها و افزایش امنیت و پایداری پروژه‌ها با بهبود فرآیند آزمایش جهش در پروژه‌های به زبان پایتون بوده است. با جایگذاری عملگرهای ریاضی در درخت نحوی انتزاعی، تغییرات کوچکی در کد ایجاد شده و آزمایش‌های اعمال شده بر روی آن باید این تغییرات را تشخیص دهند. نتایج نشان می‌دهند که این روش می‌تواند به بهبود پوشش کد آزمایش‌ها و شناسایی خطاهای ناشناخته کمک کند.
این مقاله از مزایای استفاده از درخت نحوی انتزاعی در پایتون برای ایجاد آزمایش جهش برخوردار است و به توسعه‌دهندگان این امکان را می‌دهد تا به صورت دقیق‌تر و جامع‌تری آزمایش‌های خود را ارزیابی کنند. از اهمیت این روش در بهبود اطمینان از کیفیت کد، پیشگیری از اشکالات ناخواسته و کاهش هزینه‌های آزمایش در فرآیند توسعه نرم‌افزار سخن گفته شده است.

کلمات کلیدی: درخت نحوی انتزاعی، آزمایش جهش، جهش‌های کشته شده.

۱ مقدمه

توسعه نرم‌افزارها یک فرآیند پیچیده و حساس است که نیازمند تضمین کیفیت بالا و عملکرد صحیح می‌باشد. یکی از ابزارهای مؤثر برای افزایش اعتماد به کیفیت نرم‌افزار، آزمون جهش[۱] می‌باشد. آزمون جهش یک روش آزمون فراگیر است که با هدف ارزیابی توان آزمایش‌های واحد در تشخیص تغییرات کوچک در کد منبع، ایجاد تغییرات (جهش) مصنوعی در کد و سپس اجرای آزمایش‌ها با این تغییرات انجام می‌دهد. این روش به ارزیابی قدرت آزمایش‌ها در شناسایی خطاهای پنهان کمک می‌کند و در نهایت، باعث بهبود کیفیت کلی نرم‌افزار می‌شود.

در این مقاله، به بررسی یک ابزار اجرای آزمایش جهش بر اساس کد پایتون می‌پردازیم. از آنجایی که پایتون به عنوان یک زبان برنامه‌نویسی پرکاربرد و انعطاف‌پذیر در صنعت نرم‌افزار شناخته می‌شود، ارائه یک ابزار جامع و کارآمد برای آزمایش جهش در این زبان امری اساسی به نظر می‌رسد. این ابزار با استفاده از تحلیل ساختار درخت نحوی انتزاعی[۲] و ایجاد جهش‌های متنوع در کد، امکان ارزیابی کارآیی آزمایش‌ها در شناسایی تغییرات را فراهم می‌سازد.

در ادامه، به تفصیل به معرفی کد اجرای آزمایش جهش بر عملگرهای ریاضی بر اساس پایتون پرداخته و نتایج به دست آمده از این فرآیند را، از جمله امتیاز جهش، تجزیه و تحلیل خواهیم کرد.

۲ راهکار به کار برده شده

در این بخش، روش‌ها و ابزارهای مورد استفاده برای انجام آزمایش جهش در پروژه‌های پایتون به طور دقیق توضیح داده شده است. استفاده از تحلیل ساختار درخت نحوی انتزاعی به منظور تشخیص الگوهای کد و ایجاد جهش‌ها بر روی عملگرهای ریاضی از جمله مواردی است که در این بخش مورد بررسی قرار گرفته است.

[1] Mutation Testing

[2] Abstract Syntax Tree (AST)

۲-۱ تحلیل ساختار درخت نحوی انتزاعی

برای ایجاد تغییرات و جهش‌ها بر مبنای عملگرهای ریاضی در کد پایتون، از قابلیت تحلیل ساختار درخت نحوی انتزاعی بهره گرفته شده است. این تحلیل به امکان تفکیک اجزاء کد و شناسایی نقاط تغییرات احتمالی کمک می‌کند. در نتیجه، نودها و عملگرهای ریاضی مشخص شده و جهش‌های متنوع ایجاد می‌شوند.

۲-۲ ساخت جهش‌ها

با توجه به نوع عملگرهای ریاضی موجود در کد که شامل ۱۲ عملگرهای مختلف است، جهش‌های مختلفی ایجاد می‌شود. به عنوان مثال، برای یک عملگر جمع، تغییرات متناظر با این عملگر انجام می‌شود. این تغییرات باعث ایجاد تغییرات کوچک و مخفی در کد می‌شوند که به منظور ارزیابی صحت تست‌ها بسیار مفید هستند. در ادامه لیست عملگرهای پوشش داده شده در این کد آورده شده است.

جدول ۱: عملگرهای ریاضی تحت پوشش

نماد	نام
+	Add
−	Sub
*	Mult
/	Div
//	Div Floor
%	Mod
**	Pow
>>	Shift Right
<<	Shift Left
|	OR Bit
&	AND Bit
^	XOR Bit

۲-۳ اجرا و ارزیابی آزمون جهش

بعد از ایجاد جهش‌ها، تست‌های واحد پروژه با این تغییرات اجرا می‌شوند. از ابزارهای تست جهش، مانند PyTest برای اجرا دقیق تست‌ها و تشخیص آنکه آیا تست‌ها با جهش‌ها برخورد کرده‌اند یا خیر، استفاده می‌شود. این اطلاعات ما را قادر می‌سازد تا اثربخشی تست‌ها در شناسایی تغییرات در کد را ارزیابی کنیم.

۲-۴ محاسبه امتیاز و ارزیابی جهش انجام شده

پس از اجرای تست‌های از پیش تعیین شده بر روی کدهای جهش یافته، با بررسی آن که جهش انجام شده، یک جهش کشته شده است یا زنده، اقدام به تحلیل آزمون جهش خود از طریق محاسبه امتیاز جهش خواهیم کرد. امتیاز جهش یک معیار ارزیابی است که به ما کمک می‌کند تا دقت تاثیر و کیفیت تست جهش را اندازه‌گیری کنیم. این معیار نشان دهنده نسبت تعداد جهش‌های کشته شده به کل تعداد جهش‌های ایجاد شده در کد می‌باشد. به عبارت دیگر، امتیاز جهش نشان‌دهنده قدرت تست در شناسایی تغییرات مخرب در کد است. فرمول محاسبه این امتیاز به این صورت است:

$$\text{Score Mutation} = \left(\frac{\text{Mutations Killed}}{\text{Mutations Total}}\right) \times 100 \quad (۱)$$

از موارد اهمیت محاسبه این امتیاز می‌توان به موارد زیر اشاره کرد:

- ارزیابی کیفیت تست: امتیاز جهش می‌تواند نشان‌دهنده این باشد که تست‌ها چه میزان مؤثر در شناسایی تغییرات در کد هستند.
- اعمال بهبود: با تغییر در کد و افزایش امتیاز جهش، توسعه‌دهندگان می‌توانند به سمت بهبودی مستمر در تست‌ها هدایت شوند.
- مقایسه پروژه‌ها: امتیاز جهش اجازه می‌دهد تا پروژه‌ها در مقایسه با یکدیگر ارزیابی شوند و توانایی تست در هر پروژه مقایسه گردد.

بنابراین با توجه به اینکه امتیاز جهش نشان‌دهنده اثربخشی تست جهش در شناسایی خطاهاست، این معیار یک ابزار قدرتمند برای بهبود تست‌ها و تضمین کیفیت نرم‌افزار می‌باشد.

۲-۵ نمایش نتایج حاصل از اجرا

و در پایان نیز با نمایش نتایج اعمال تست جهش، از جمله نمایش امتیاز جهش، به توسعه‌دهندگان یک دید کلی از موفقیت مجموعه آزمایشی آن‌ها در شناسایی جهش‌ها ارائه خواهد داد. نمره جهش بالاتر بیان‌گر خوب بودن مجموعه تست‌ها است.

۳ نتایج

برای بررسی تمام عملگرهای دودویی بیتی و آزمایش جامع کد پیاده‌سازی شده، از کد نمونه در تصویر۱ که همه عملگرها را پوشش می‌دهد استفاده کردیم. نتایج تست جهش بر روی کد نشان داده شده در تصویر ۱، نشان دهنده امتیاز جهش ۹۷ است که نشان دهنده درجه بالایی از اثربخشی در تشخیص جهش در کد داده شده است.

۳-۱ نقاط قوت مجموعه تست

از مهم‌ترین نقاط قوت مجموعه تست، پوشش جامع عملگر است. مجموعه موارد آزمایشی ارائه شده طیف گسترده ای از عملگرها را شامل می شود، از جمله محاسبات پایه، تقسیم و حالت لبه آن (تقسیم بر صفر)، تقسیم طبقه، باقیمانده، توان، عملیات بیتی (تغییر، OR، AND، XOR) ، اطمینان از بررسی کامل عملکرد ماشین حساب.

۳-۲ رسیدگی به حالت لبه

موارد تست شامل سناریوهایی با موارد لبه بالقوه، مانند تقسیم بر صفر است. این یک رویکرد متفکرانه برای آزمایش را نشان می دهد و اطمینان می دهد که عملکرد ماشین حساب می تواند شرایط ورودی متنوع را مدیریت کند.

۳-۳ تست عملکرد

هر مورد آزمایشی مربوط به یک اپراتور خاص است که صحت عملکرد ماشین حساب را تحت عملیات های مختلف ریاضی تأیید می کند. این تضمین می کند که ماشین حساب برای هر عملیات پشتیبانی شده مطابق انتظار عمل می کند.

۳-۴ زمینه های بالقوه برای بهبود

۳-۴-۱ آزمایش رسیدگی به خطای صریح

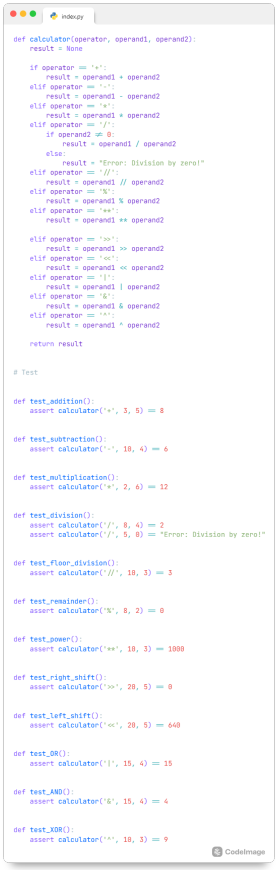
در حالی که تست تقسیم بر صفر وجود دارد، ممکن است آزمایش صریح سناریوهای خطا برای سایر اپراتورها، مانند تلاش برای انجام یک شیفت بیتی با یک عملوند منفی، مفید باشد

۳-۴-۲ تنوع مورد آزمایشی

اگرچه موارد آزمایشی موجود طیف گسترده ای از عملیات را پوشش می دهند، سناریوهای اضافی، به ویژه آنهایی که شامل عبارات پیچیده یا ترکیبی از عملگرها هستند، می توانند جامعیت مجموعه آزمایشی را بیشتر افزایش دهند.

۴ جمع بندی

در این مقاله، به ارائه یک روش جامع و کاربردی برای انجام تست جهش در پروژه‌های پایتون پرداختیم. با استفاده از تحلیل ساختار درخت نحوی انتزاعی و ایجاد جهش‌های مختلف بر اساس عملگرهای ریاضی، توانستیم جهش هایی در کد ایجاد کرده و اثربخشی تست‌ها در شناسایی این تغییرات را ارزیابی نماییم. از ابزارهای تست جهش استفاده شده و نحوه اجرا و تجزیه و تحلیل نتایج تست جهش نیز به‌طور دقیق تشریح شد. این روش نه تنها به توسعه‌دهندگان کمک می‌کند تا تست‌های خود را بهبود دهند بلکه میزان اعتماد به کیفیت نرم‌افزار را افزایش دهند. با افزودن تست جهش به فرایند توسعه نرم‌افزار، توسعه‌دهندگان می‌توانند بهبود مستمری در تست‌ها و کیفیت کلی نرم‌افزار خود را تجربه کنند همچنین امتیاز جهش به آن‌ها امکان می‌دهد تا به دقت میزان توانایی تست در کشتن جهش‌ها و شناسایی تغییرات مخرب را .ارزیابی کنند علاوه بر این راهبرد مطرح شده در این مقاله، برای بهبود و توسعه آن، می‌توان از چند ایده استفاده کرد. اولاً، تحقیقات بیشتر در زمینه افزودن جهش‌های جدید و متنوع به کد می‌تواند به افزایش



```python
def calculator(operator, operand1, operand2):
    result = None

    if operator == '+':
        result = operand1 + operand2
    elif operator == '-':
        result = operand1 - operand2
    elif operator == '*':
        result = operand1 * operand2
    elif operator == '/':
        if operand2 != 0:
            result = operand1 / operand2
        else:
            result = "Error: Division by zero!"
    elif operator == '//':
        result = operand1 // operand2
    elif operator == '%':
        result = operand1 % operand2
    elif operator == '**':
        result = operand1 ** operand2

    elif operator == '>>':
        result = operand1 >> operand2
    elif operator == '<<':
        result = operand1 << operand2
    elif operator == '|':
        result = operand1 | operand2
    elif operator == '&':
        result = operand1 & operand2
    elif operator == '^':
        result = operand1 ^ operand2

    return result


# Test


def test_addition():
    assert calculator('+', 3, 5) == 8


def test_subtraction():
    assert calculator('-', 10, 4) == 6


def test_multiplication():
    assert calculator('*', 2, 6) == 12


def test_division():
    assert calculator('/', 8, 4) == 2
    assert calculator('/', 5, 0) == "Error: Division by zero!"


def test_floor_division():
    assert calculator('//', 10, 3) == 3


def test_remainder():
    assert calculator('%', 8, 2) == 0


def test_power():
    assert calculator('**', 10, 3) == 1000


def test_right_shift():
    assert calculator('>>', 20, 5) == 0


def test_left_shift():
    assert calculator('<<', 20, 5) == 640


def test_OR():
    assert calculator('|', 15, 4) == 15


def test_AND():
    assert calculator('&', 15, 4) == 4


def test_XOR():
    assert calculator('^', 10, 3) == 9
```

شکل ۱: نمونه کد ورودی جهت انجام آزمون جهش

تنوع و اثربخشی تست‌ها کمک کند. همچنین، بهبود ابزارهای تست جهش و افزایش دقت در ارزیابی نتایج تست‌ها از جمله اقداماتی است که می‌تواند به بهبود فرآیند تست جهش کمک کند.

تسلط به تست جهش به توسعه‌دهندگان این امکان را می‌دهد که بهبوداتی مستمر در تست‌ها ایجاد کنند و در نتیجه، اعتماد به کیفیت نرم‌افزار را افزایش دهند. این راهبرد قدرتمند به توسعه‌دهندگان ابزارهایی برای افزایش بهره‌وری تست و بهبود کیفیت کد ارائه می‌دهد. توسعه این ایده‌ها به سمت بهینه‌سازی و ارتقاء ابزارها و روش‌های تست جهش می‌تواند تاثیرگذاری بیشتری در توسعه نرم‌افزارهای پایتون داشته باشد.

مراجع

[۱] B hme، Marcel Gopinath، Rahul Zeller، Andreas Techniques and "Tools Holler، Christian and Fraser، Gordon Tests" Software Generating for

[۲] Based Tree Syntax Abstract An " Liu، Xin Programs" ۲.x Python for Tool Testing Mutation

[۳] https://mutatest.readthedocs.io/en/latest/