به نام خدا

پروژه کارشناسی

گزارش فاز دوم

سید محمد مهدی احمدپناه

---

در فاز قبل، از روی سورس کد ورودی، گراف‌های وابستگی برنامه به دست آمد. در این فاز، با استفاده از سورس کد ورودی و گراف وابستگی به دست آمده از آن، بازنویس حالت غیرحساس به پیشرفت[1] را پیاده‌سازی می‌کنیم.

برای این کار، ابتدا همه مسیرهای با شروع از گره آغازین[2] که بعد از آن گره‌ای مانند P در PDG، که آن گره بیانگر inH است، موجود است و مسیری از آن به یک گره بیانگر outL وجود دارد را در مجموعه‌ای به نام F در نظر می‌گیریم. سپس نوع جریانی را که هر کدام از مسیرهای موجود در F بیان می‌کنند را محاسبه می‌کنیم. به این ترتیب که اگر در مسیر، همه یال‌ها از نوع وابستگی داده‌ای باشند، آن مسیر یک جریان صریح[3] را بیان می‌کند؛ و در غیر این صورت، آن مسیر دارای جریان ضمنی[4] است (پیاده‌سازی این موارد، در متد initializeF در کلاس PINIRewriter آمده است).

حال بررسی می‌شود که اگر مجموعه F عضوی نداشته باشد؛ یعنی هیچ جریانی، چه ضمنی و چه صریح، در کد برنامه ورودی وجود ندارد، پس نیازی به بازنویسی کد برنامه نیست.

پس با فرض وجود مسیرهایی در مجموعه F، کارهای زیر انجام می‌شود:

۱- برای هر مسیری به نام f موجود در مجموعه مسیرهای F، شرط مسیر۵ را محاسبه می‌کنیم. برای این کار، ابتدا همه گره‌هایی مانند N که در طول مسیر f هستند و از گره دیگری مانند X به N، وابستگی داده‌ای وجود دارد را به دست می‌آوریم. سپس شرط اجرای۶ آن گره (گره N) را با پیمایش رو به عقب روی یال‌های وابستگی کنترلی PDG و and کردن شروط به دست می‌آوریم (متد findExecutionCondition). اینک نوبت آن است که شرط اجرای همه گره‌های Nای که در مسیر f وجود دارد را با یکدیگر and کنیم. حاصل این عمل، عبارتی خواهد شد که شرط مسیر f نام‌گذاری می‌شود که روشن است که شرط مسیر تعریف شده، با

[1] Progress Insensitive
[2] START Node
[3] Explicit
[4] Implicit
[5] Path Condition
[6] Execution Condition

شرط مسیری که ممکن است به سادگی از روی کد برنامه مشاهده شود و شرط اجرای آن مسیر باشد، متفاوت است. تنها در صورتی که برای یک مسیر، هیچ گره‌ای با مشخصات گره نمونه N وجود نداشته باشد، مقدار همواره درست[7] به عنوان شرط مسیر در نظر گرفته می‌شود (متغیر executionConditionsForThisPath و اضافه کردن شرط مسیرها به آرایه pathConditions)

۲- برای هر گره مانند n در PDG که بیانگر outL است، کارهای زیر را انجام می‌دهیم (دقت شود که ممکن است یک متغیر در کد برنامه ورودی مانند l1، چندین بار out شود، برای هر کدام از این دستورات در برنامه، باید جداگانه موارد زیر صورت پذیرد):

عبارتی به نام c در نظر می‌گیریم که حاصل or کردن همه شرط مسیرهایی – که در مرحله ۱ محاسبه شد – است که گره پایانی آن مسیر، گره n باشد. این عبارت، همان عبارت شرطی‌ای خواهد بود که اجرا یا عدم اجرای دستور outL را تعیین خواهد کرد.

در مرحله پایانی، اگر همه مسیرهایی که با n خاتمه می‌یابند، از نوع صریح باشند، دستور outL l با عبارت if c then outL BOT else outL l endif; جایگزین خواهد شد و اگر حداقل یکی از مسیرهای مورد نظر، ضمنی باشد، عبارت if c then NOP else outL l endif جای دستور خروج داده درون متغیر l را خواهد گرفت.

(همه موارد بالا در متد rewriter در کلاس PINIRewriter پیاده‌سازی شده است)

در پایان، پس از اِعمال رویّه بالا، کد بازنویسی‌شده در یک فایل متنی جداگانه، در اختیار کاربر برنامه قرار می‌گیرد.

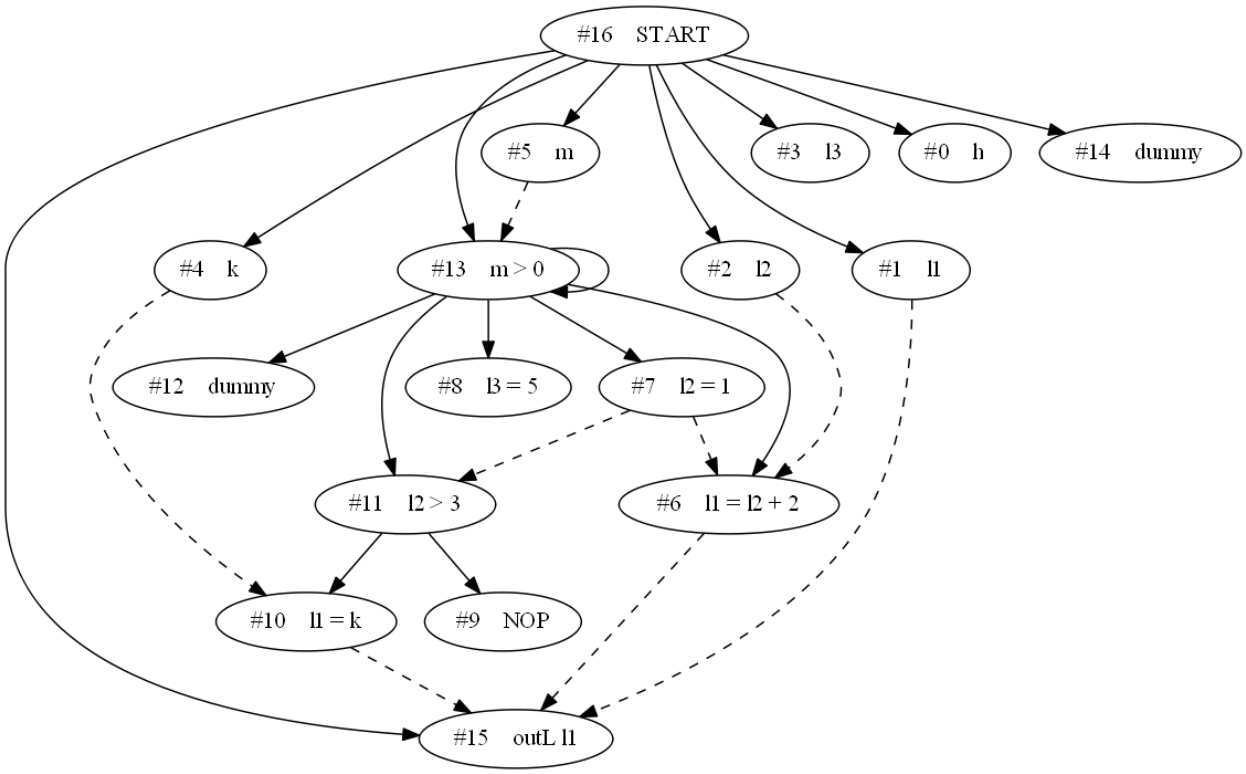
در این فاز، برای پیاده‌سازی الگوریتم بالا، دسترسی به کد برنامه در هر لحظه و هر گره، پیمایش چندین باره PDG و CFG و نیز آزمون و اصلاح کد پیاده‌سازی‌شده برای همه عناصر زبان، باعث ایجاد تغییرات و بهبودهایی در مراحل قبلی پروژه شد که از آوردن جزئیات پیاده‌سازی در این گزارش صرف‌نظر شده است. تنها به عنوان نمونه، عبارت exp ! که عملگر نقیض یک عبارت شرطی می‌باشد، به گرامر زبان افزوده شد.

[7] TRUE

برای مثال، در ادامه یکی از نمونه‌های آزمون آورده شده است:

```
program;
inH h;
inL l1, l2, l3;
inH k , m;
while m > 0 do
        l1 = l2 + 2;
        l2 = 1;
        l3 = 5;
        if l2 > 3 then
                NOP
        else
                l1 = k
        endif
done;
outL l1
```

**برنامه ۱ – سورس کد ورودی با نام input-while.wl**




**شکل ۱ - گراف وابستگی برنامه برای برنامه ۱**

```
PATH 1 : #16 START  ->  #5 m  ->  #13 m > 0  ->  #11 l2 > 3  ->  #10 l1 = k  ->  #15 outL l1
PATH 2 : #16 START  ->  #5 m  ->  #13 m > 0  ->  #6 l1 = l2 + 2  ->  #15 outL l1
PATH 3 : #16 START  ->  #5 m  ->  #13 m > 0  ->  #7 l2 = 1  ->  #11 l2 > 3  ->  #10 l1 = k  ->
#15 outL l1
PATH 4 : *EXPLICIT ==> #16 START  ->  #4 k  ->  #10 l1 = k  ->  #15 outL l1
PATH 5 : #16 START  ->  #5 m  ->  #13 m > 0  ->  #7 l2 = 1  ->  #6 l1 = l2 + 2  ->  #15 outL l1
```

**مسیرهای موجود در مجموعه F برای برنامه ۱**

```
PATH CONDITION 1: TRUE
PATH CONDITION 2: TRUE
PATH CONDITION 3:  (m > 0)
PATH CONDITION 4: !(l2 > 3) and (m > 0)
PATH CONDITION 5:  (m > 0)
```

**شرط مسیرها برای برنامه ۱**

```
program;
inH h;
inL l1, l2, l3;
inH k, m;
while m > 0 do
        l1 = l2 + 2;
        l2 = 1;
        l3 = 5;
         if l2 > 3 then
                NOP
        else
                l1 = k
        endif
done ;
if TRUE then
        NOP
 else
        outL l1
endif
```

**برنامه ۲ – سورس کد بازنویسی‌شده برای برنامه ۱**

در این فاز، موارد آزمون بسیاری بررسی شد و نمونه‌های پیچیده‌تر در لینک پروژه در زیر قابل مشاهده است.

مراحل انجام پروژه از طریق گیت‌هاب به آدرس https://github.com/smahmadpanah/BScProject قابل مشاهده و پیگیری است.

گزارش فازهای بعدی نیز به تدریج ارائه خواهد شد.