به نام خدا

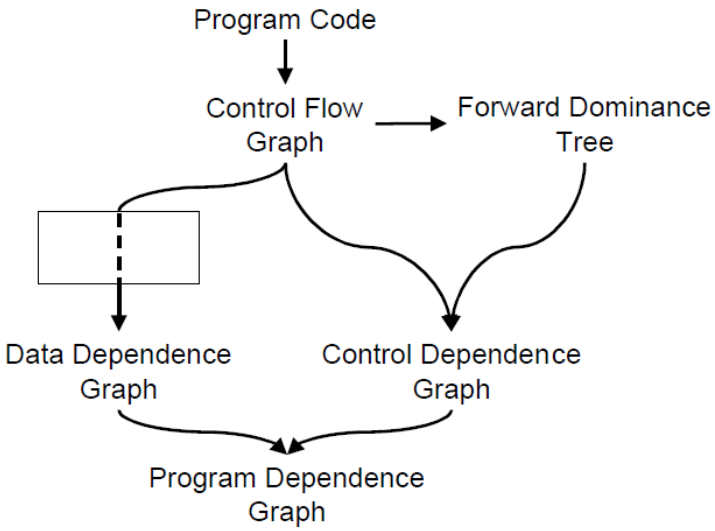
پروژه کارشناسی

گزارش فاز اول

سید محمد مهدی احمدپناه

در فاز قبل، سورس کد ورودی با استفاده از ابزارهای lex و yacc، گرفته می‌شد و در ادامه، در صورت وجود خطای نحوی، به کاربر اطلاع داده می‌شد.

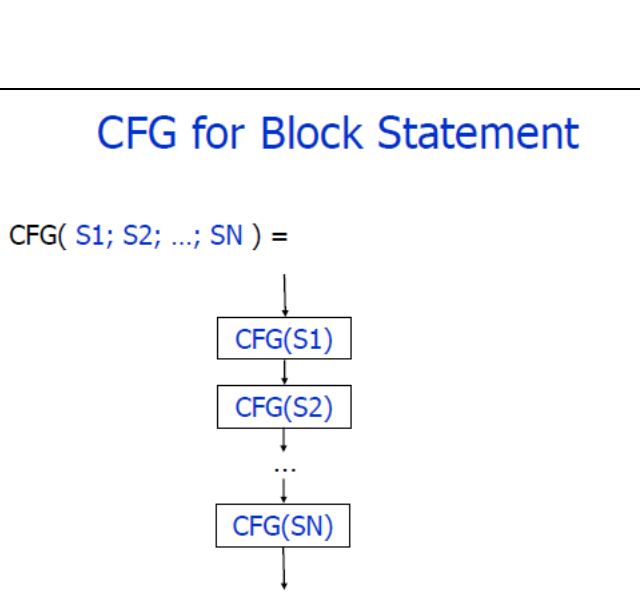
حال در این مرحله، سورس کد ورودی معتبر و مطابق با گرامر زبان، گرفته می‌شود و گراف‌های مورد نیاز برای تحلیل ساخته می‌شود. هدف، تولید گراف وابستگی برنامه است که بدین منظور، مطابق به شکل زیر انجام شد:



شکل ۱- نمودار کلی نحوه به دست آوردن گراف وابستگی برنامه

ابتدا Control Flow Graph یا به اختصار CFG به دست می‌آید. نحوه کار بدین شکل است که گزاره[1]ها و عبارت[2]های برنامه، به عنوان یگ گره[3] در گراف در نظر گرفته می‌شود. هر بلوک پایه[4]، شامل تعدادی گره است. هر بلوک پایه، فقط یک گره ورودی و یک گره خروجی خواهد داشت. برای این منظور، به شکل‌های زیر عمل شد که برای گزاره‌های ساده که اجرای آن‌ها مشروط نیست، به صورت دنباله گره‌ها در نظر گرفته می‌شود.

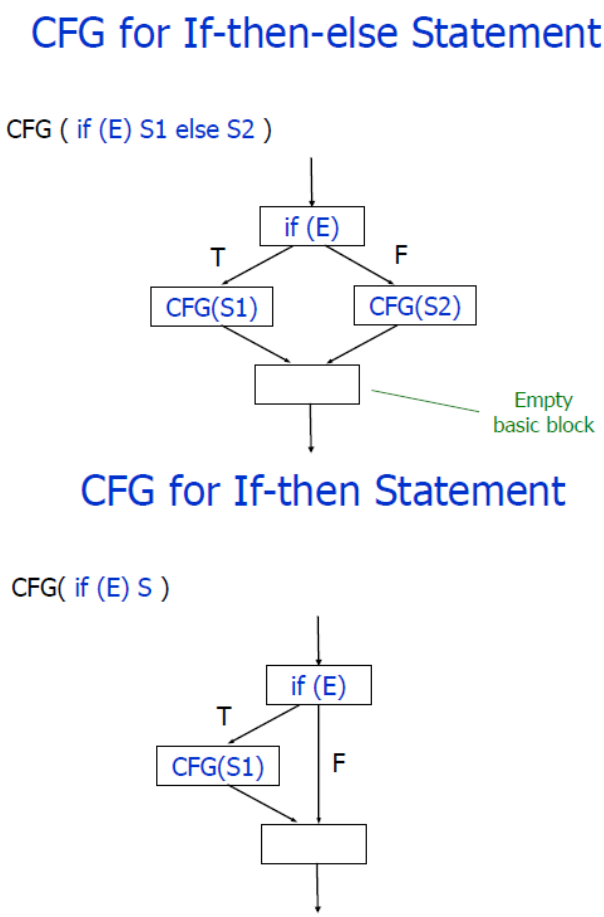
[1] Statement
[2] Expression
[3] Node
[4] Basic Block



شکل ۲ - نحوه تولید زیرگراف بلوک پایه

پس برای این گزاره‌ها و عبارات، تنها ساختن یک گره جدید و متصل کردن آن‌ها به لیست پیوندی گراف کفایت می‌کند.

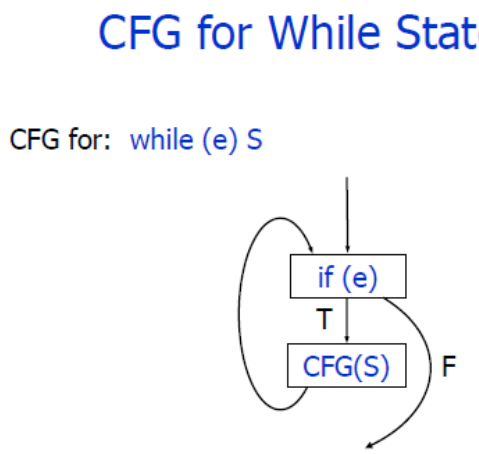
برای گزاره‌های شرطی، دو شکل زیر در نظر گرفته می‌شود:



شکل ۳- نحوه تولید زیرگراف گزاره‌های شرطی

پس برای گره عبارت شرطی، دو پیوند برای برقراری و عدم برقراری شرط در نظر گرفته می‌شود. ضمنا به دلیل اینکه می‌خواهیم قالب کلی زیرگراف کنترل همه گزاره‌ها مشابه باشد، یک گره مجازی[5] در انتهای این گونه زیرگراف‌ها ایجاد می‌کنیم که مطمئن باشیم هر زیرگراف تنها یک مجرای خروجی دارد.

برای گزاره حلقه موجود در زبان، شکل زیر را در نظر می‌گیریم:



شکل ۴ - نحوه تولید زیرگراف گزاره‌های حلقه

مشابه قبل، یک گره مجازی به عنوان گره پایانی زیرگراف ساخته می‌شود.

با توجه به زیرگراف‌های پایه‌ای فوق، در هر کدام از قاعده‌های موجود در گرامر زبان، حالت‌های بالا را ایجاد می‌کنیم و همزمان با ساخته شدن درخت تجزیه[6]، CFG تولید می‌شود.

حال برای تولید Control Dependence Graph یا CDG، نیاز به Forward Dominance Tree یا Post Dominance Tree خواهیم داشت. برای تولید این درخت، الگوریتم ساخت Dominance Tree را بر روی معکوس CFG؛ یعنی همان گره‌ها ولی با جهت یال‌های معکوس شده، اعمال می‌کنیم.

در CFG، گره M بر گره N غلبه[7] می‌کند، اگر و تنها اگر همه مسیرهای از گره شروع تا گره N، حتما و الزاماً از گره M بگذرند و گره M بر گره N اکیداً غلبه می‌کند، اگر و تنها اگر بر آن گره غلبه کند و M، همان گره N نباشد. واضح است که یک گره در CFG می‌تواند چندین غلبه‌کننده[8] داشته باشد، اما برای تولید درخت

[5] dummy
[6] Parse tree
[7] dominate
[8] dominator

غلبه، نزدیک‌ترین غلبه‌کننده که غلبه‌کننده بی‌درنگ[9] اهمیت دارد. با استفاده از شبه‌کد زیر می‌توان غلبه‌کننده‌های یک گره در CFG را به دست آورد:

```
Compute Dominators(){
        For (each n ∈ NodeSet)
                Dom(n) = NodeSet
        WorkList = {StartNode}
        While (WorkList ≠ ϕ) {
                Remove any node Y from WorkList
                New = {Y} ∪ ∩x∈Pred(Y) Dom(X)
                If New ≠ Dom(Y) {
                        Dom(Y) = New
                For (each Z ∈ Succ(Y))
                WorkList = WorkList U {Z}
                }

        }

}
```

الگوریتم ۱ - محاسبه گره‌های غلبه‌کننده برای هر گره

سپس با توجه به مجموعه غلبه‌کنندگان به دست آمده از الگوریتم بالا و مقایسه با مجموعه غلبه‌کنندگان سایر گره‌ها، می‌توان گره غلبه‌کننده بی‌درنگ را یافت و درخت غلبه را تشکیل داد اما برای ما، درخت پس‌غلبه[10] اهمیت دارد.

به این صورت که، گره Z، گره Y را پس‌غلبه می‌کند اگر و تنها اگر همه مسیرهای از Y تا گره پایانی، حتما و الزاماً از Z عبور کنند. حال، در صورتی که این الگوریتم را برای معکوس CFG اعمال کنیم، درخت پس‌غلبه تولید می‌شود.

پس از این، برای ساخت CDG، به مرز پس‌غلبه[11] نیاز داریم. مرز پس‌غلبه گره X، مجموعه‌ای از گره‌هایی هستند که توسط X اکیدا پس‌غلبه نمی‌شوند اما گره‌های ما بَعد[12]ی دارند که توسط X، پس‌غلبه می‌شوند. تعریف ریاضی این گره‌ها بدین شرح است:

PDF(X) = { y | (∃z ∈ Succ(y) such that x post-dominates z) and x does not strictly post-dominate y }

که این مجموعه بیانگر نزدیک‌ترین نقاط انشعابی[13] است که به گره X منجر می‌شوند.

طبق قضیه زیر، می‌توان وابستگی‌های کنترلی برنامه را برای هر گره موجود در CFG، به دست آورد:

[9] Immediate dominator
[10] Post-dominance tree
[11] Post-dominance Frontier
[12] successor
[13] Diverging points

قضیه – گره y تعلق دارد به مجموعه PDF(X) اگر و تنها اگر X به Y وابستگی کنترلی داشته باشد.

حال با استفاده از الگوریتم زیر، می‌توان مجموعه PDF هر گره را به دست آورد که بیانگر وابستگی‌های کنترلی خواهند بود.

```
For each x in the bottom-up traversal of the postdominator tree do
    PDF(X) = ϕ
    Step 1: For each y in Predecessor(X) do
        If X is not immediate post-dominator of y then
            PDF(X) ← PDF(X) ∪ {y}
    Step 2: For each z that x immediately post-dominates, do
        For each y ∈ PDF(Z) do
            If X is not immediate post-dominator of y then
                PDF(X) ← PDF(X) ∪ {y}
```

الگوریتم ۲ - محاسبه گره‌های پس‌غلبه‌کننده مرزی برای هر گره

پس از این محاسبات، وابستگی‌های کنترلی برنامه برای هر گره – گزاره یا عبارت برنامه ورودی – به دست آمده است. در نتیجه، تا این مرحله، گراف وابستگی کنترلی برنامه یا CDG تولید شده است.

حال نوبت تولید Data Dependence Graph یا به اختصار DDG است. وابستگی‌های داده مختلفی وجود دارد اما برای کار ما و طبق مقاله اصلی پروژه، ارتباط بین گره‌هایی که شامل مقداردهی یک متغیر و استفاده از آن متغیر هستند، اهمیت دارد؛ یعنی گره X به گره Y وابستگی داده‌ای دارد اگر و تنها اگر در گره Y متغیری وجود داشته باشد که در گره X مقداردهی شده است. پس با توجه به همین تعریف، مطابق با قواعد، متغیری که به آن مقداری نسبت داده شده یا استفاده شده است، نگه‌داری می‌شوند. حال برای به دست آوردن وابستگی‌های داده‌ای، در صورتی که در یک گره، از متغیری که در گره دیگری مقداردهی شده است، استفاده شود، یک وابستگی داده‌ای در نظر گرفته می‌شود. برای دقیق‌تر بودن و عدم محافظه‌کارانه بودن وابستگی‌ها، تنها نزدیک‌ترین گزاره‌ای که آن متغیر در آن مقداردهی شده است، وابستگی را خواهد داشت؛ و نه همه گزاره‌هایی که آن متغیر را مقداردهی کردند، که این کار با پیمایش CFG امکان‌پذیر است و انجام شده است.

پس از این مرحله، گراف وابستگی داده‌ای برنامه نیز آماده است. تنها کار باقی‌مانده، ترکیب این دو گراف که دارای گره‌های یکسان هستند، خواهد بود تا گراف وابستگی برنامه تولید شود.

برای نمایش گرافیکی این گراف‌ها، از ابزار GraphViz استفاده شده که سورس کد آن نیز به پروژه اضافه شده است و بهترین نحوه نمایش گراف را در یک فایل تصویری، به نمایش درمی‌آورد.

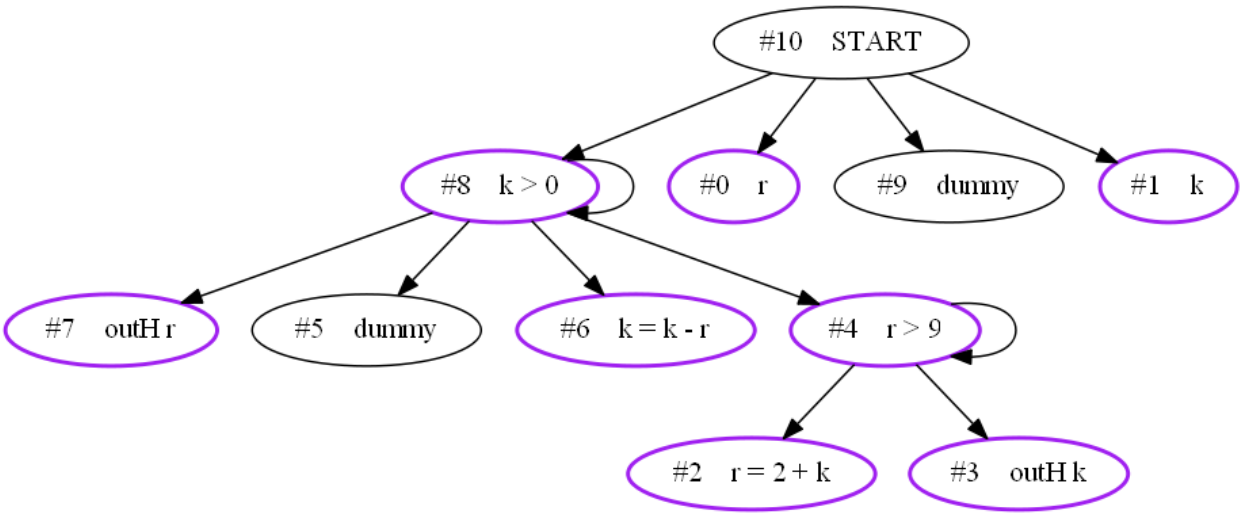
در ادامه نمونه‌ای از یک برنامه مورد آزمون[14] آمده است (سایر موارد آزمون از طریق آدرس پروژه قابل مشاهده است):

فایل سورس کد ورودی:

```
program;
inH k , r;
while k>0 do
        while r>9 do
                r = 2 + k;
                outH k
        done;
        k = k - r;
        outH r
done
```

برنامه ۱ - یکی از موارد آزمون بررسی‌شده

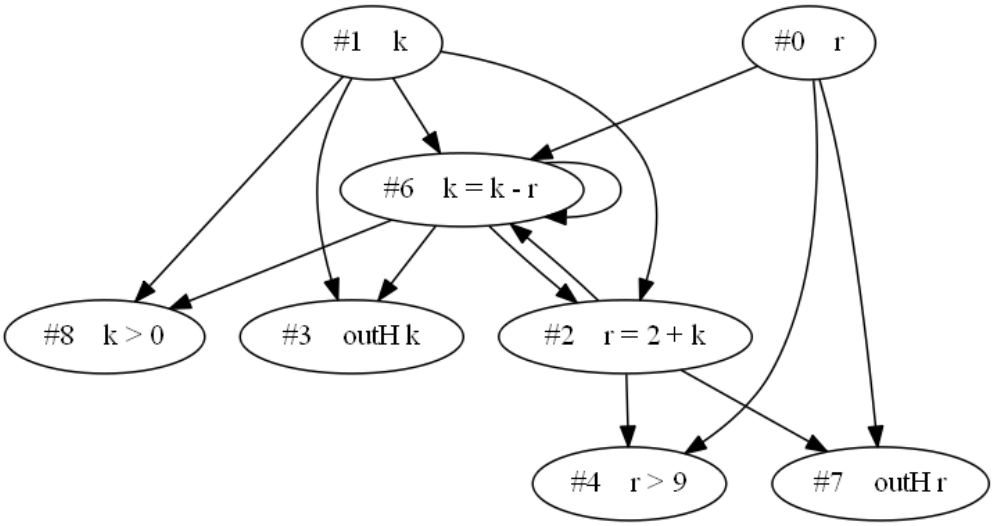
گراف وابستگی کنترل:



شکل ۵ - گراف وابستگی کنترل برای برنامه ۱

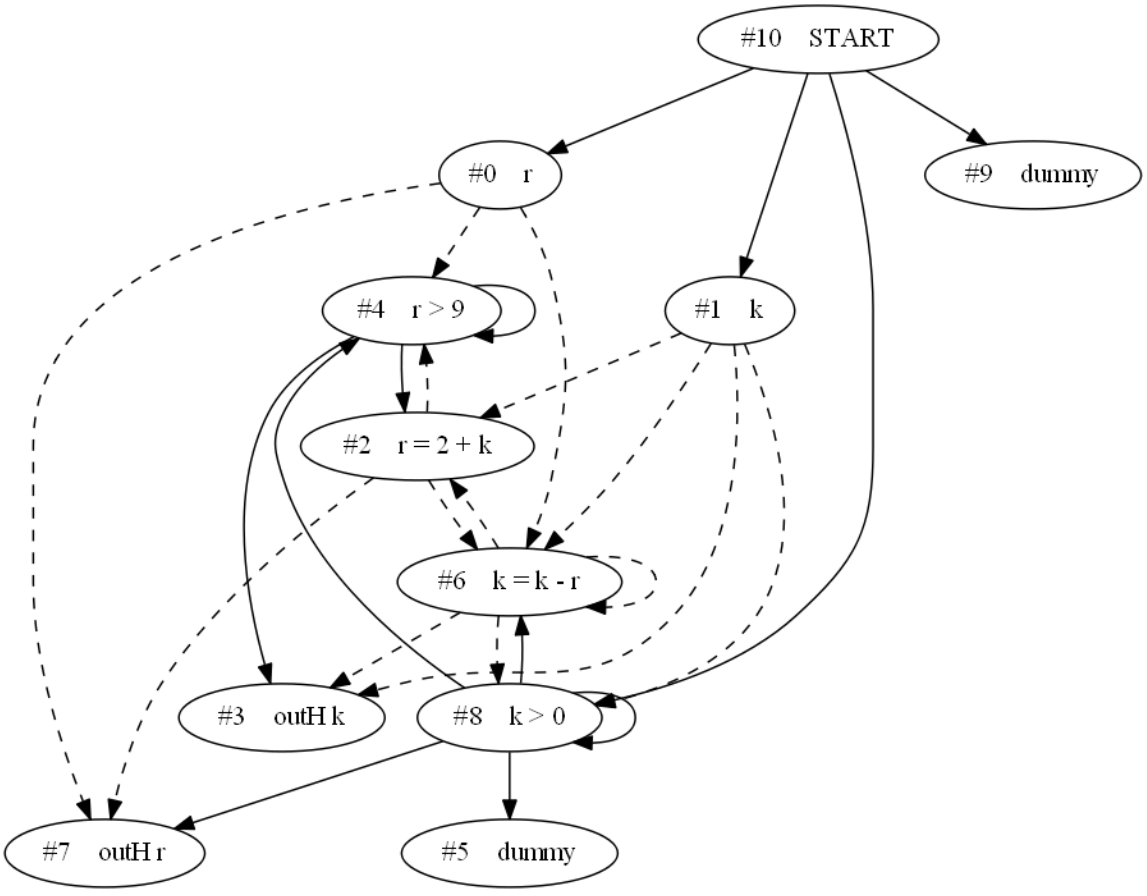
[14] Test case

گراف وابستگی داده:



شکل ۶ - گراف وابستگی داده برای برنامه ۱

گراف وابستگی برنامه:



شکل ۷ - گراف وابستگی برنامه برای برنامه ۱

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.