

دانشگاه صنعتي امیرکبیر  
(پلی تکنیک تهران)

دانشكده مهندسی کامپیوتر و فن­­آوری ­اطلاعات

پایان‌نامه کارشناسی

گرایش نرم­افزار

عنوان

طراحی و پیاده‌سازی ابزاری به منظور اِعمال خط مشی امنیتیِ عدم تداخل مبتنی بر روش بازنویسی برنامه

نگارش

سید محمدمهدی احمدپناه

استاد راهنما

دکتر مهران سلیمان ‌فلاح

شهریور 1394

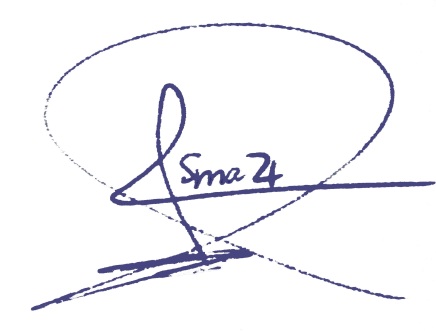
اينجانب سید محمدمهدی احمدپناه متعهد مي‌شوم كه مطالب مندرج در اين پايان نامه حاصل كار پژوهشي اينجانب تحت نظارت و راهنمايي اساتيد دانشگاه صنعتي اميركبير بوده و به دستاوردهاي ديگران كه در اين پژوهش از آنها استفاده شده است، مطابق مقررات و روال متعارف ارجاع و در فهرست منابع و مآخذ ذكر گرديده است. اين پایان نامه قبلاً براي احراز هيچ مدرك هم‌سطح يا بالاتر ارائه نگرديده است.

در صورت اثبات تخلف در هر زمان، مدرك تحصيلي صادر شده توسط دانشگاه از درجه اعتبار ساقط بوده و دانشگاه حق پيگيري قانوني خواهد داشت.

كليه نتايج و حقوق حاصل از اين پایان نامه متعلق به دانشگاه صنعتي اميركبير مي‌باشد. هرگونه استفاده از نتايج علمي و عملي، واگذاري اطلاعات به ديگران يا چاپ و تكثير، نسخه‌برداري، ترجمه و اقتباس از اين پایان نامه بدون موافقت كتبي دانشگاه صنعتي اميركبير ممنوع است.   
نقل مطالب با ذكر مآخذ بلامانع است.

سید محمدمهدی احمدپناه

امضا



تقدیم به پدرم

کوهی استوار و حامی من در سراسر زندگی 

تقدیم به مادرم

سنگ صبوری که الفبای زندگی به من آموخت

تقدیم به خواهر و برادرم

همراهان همیشگی و پشتوانه‌های زندگیم

**تقدیر و تشکر:**

سپاس خدای را که سخنوران، در ستودن او بمانند و شمارندگان، شمردن نعمت های او ندانند و کوشندگان، حق او را گزاردن نتوانند. سلام و درود بر پیامبر رحمت و خاندان پاك او، طاهران معصوم، هم آنان که وجودمان وام‌دار وجودشان است.

بدون شک جایگاه و منزلت معلم، بالاتر از آن است که در مقام قدردانی از زحمات بی شائبه‌ او، با زبان قاصر و دست ناتوان، چیزی بنگارم. اما از آنجا که تجلیل از معلم، سپاس از انسانی است که هدف آفرینش را تأمین می‌کند، به رسم ادب دست به قلم برده‌ام، باشد که این خردترین بخشی از زحمات آنان را سپاس گوید.

از پدر و مادر مهربانم، این دو معلم بزرگوار که همواره بر کوتاهی من، قلم عفو کشیده و کریمانه از کنار غفلت‌های گذشته‌اند و در تمام عرصه‌های زندگی یار و یاورم بوده‌اند؛

از استاد فرزانه و دلسوز، جناب آقای دکتر مهران سلیمان فلاح که در کمال سعه صدر، با حسن خلق و فروتنی، از هیچ کمکی در این عرصه بر من دریغ نداشتند؛

از اساتید محترم، جناب آقای دکتر محمدرضا رزازی و جناب آقای دکتر بهمن پوروطن که زحمت داوری این پایان‌نامه را متقبل شدند؛

از جناب آقای افشین لامعی که از آغاز پروژه در درک بهتر مفاهیم و بخش‌های مختلف مرا راهنمایی کردند؛

و در پایان، از دوستان عزیزم، محمد پزشکی، بهنام ستارزاده، علی قنبری، مسعود غفاری‌نیا، احسان عدالت، حمیدرضا رمضانی، ریحانه شاه‌محمدی و پرهام الوانی که در طول انجام پروژه از نظرات و راهنمایی‌هایشان استفاده کردم؛

کمال تشکر و قدردانی را دارم.

چكيده

خط مشیِ امنیتی، تعریفی از امن بودنِ یک سیستم یا برنامه را ارائه می‌دهد که در آن اَعمال مجاز و غیرمجاز برای آن سیستم یا برنامه مشخص می‌شود. خط مشی امنیتیِ مورد نظر در این پروژه، *عدمِ تداخل* است. به زبان ساده، خط مشی عدم تداخل بیان می‌کند که یک مشاهده‌گر سطح پایین که فقط به برنامه و مقادیر عمومیِ زمان‌اجرا دسترسی دارد، نتواند درکی نسبت به ورودی‌های سطح بالا یا خصوصی پیدا کند. نکته مهم این است که خط مشی عدم تداخل را نمی‌توان به صورت مجموعه‌ای از اجراها بیان کرد؛ بلکه می‌توان آن را با مجموعه‌ای از مجموعه اجراها بیان کرد. از این رو، اِعمال این خط مشی با محدودیت‌ها و دشواری‌هایی مواجه است. در این پروژه، با تقسیم‌بندی این خط مشی به دو حالت غیرحساس و حساس به پیشرفت، سعی در بیان دقیق‌تر و اِعمال آن داریم.

هدف از انجام این پروژه، طراحی و پیاده‌سازی ابزاری است که بتواند با استفاده از بازنویسی برنامه، خط مشی عدم تداخل را روی آن اعمال کند. در این روش، گراف وابستگی برنامه و کد مبدأ، ورودی‌های الگوریتم بازنویسی خواهند بود و نتیجه اجرای آن، تغییر و جایگزینی دستوراتی است که با پیمایش در گراف وابستگی برنامه، تهدیدی برای جریان اطلاعات از سطوح بالا به سطوح پایین امنیت هستند. برخلاف مکانیزم‌های دیگرِ اعمال خط مشی‌ها، با بهره‌گیری از روش بازنویسی برنامه جلوی اجرای برنامه‌های مغایر با خط مشی، چه قبل یا چه در زمان ‌اجرا، گرفته نمی‌شود؛ بلکه آن‌ها بازنویسی می‌شوند و برنامه‌های ناامن به برنامه‌های امن تبدیل می‌شوند. صحت و شفافیت الگوریتم مورد استفاده، که مهم‌ترین عوامل مقایسه روش‌های بازنویسی به شمار می‌روند، اثبات شده است.

در این پروژه، ابتدا با استفاده از تحلیل‌گر لغوی و نحوی، کدهای مبدأ ورودی بررسی می‌شوند و پس از تولید گراف وابستگی برنامه و نمایش گرافیکی آن، الگوریتم‌های بازنویسی اجرا می‌شوند و کدهای خروجی، هم به زبان WL و هم به زبان C، به کاربر نمایش داده می‌شود.

واژه‌های کلیدی:

امنیت جریان اطلاعات؛ خط مشی عدم تداخل؛ گراف وابستگی برنامه؛ بازنویسی برنامه

|  |  |
| --- | --- |
| فهرست عناوین | صفحه |

[1 ‌ فصل اول مقدمه 1](#_Toc432185259)

[2 فصل دوم خط مشی امنیتی عدم تداخل و اِعمال آن 5](#_Toc432185260)

[2‌.1‌ خط مشی امنیتی عدم تداخل 6](#_Toc432185261)

[2‌.2‌ مروری بر کارهای گذشته 8](#_Toc432185262)

[2‌.3‌ اِعمال خط مشیِ عدم تداخل 9](#_Toc432185263)

[3 فصل سوم زبان WL و گراف وابستگی برنامه 13](#_Toc432185264)

[3‌.1‌ توصیف زبان برنامه‌نویسی WL 14](#_Toc432185265)

[3‌.1‌.1‌ ساختار نحوی و گرامر زبان 14](#_Toc432185266)

[3‌.1‌.2‌ معناشناخت زبان 16](#_Toc432185267)

[3‌.2‌ گراف وابستگی برنامه 17](#_Toc432185268)

[3‌.2‌.1‌ وابستگی‌های کنترلی و داده‌ای 17](#_Toc432185269)

[3‌.2‌.2‌ نحوه تولید گراف وابستگی برنامه 20](#_Toc432185270)

[4 فصل چهارم الگوریتم بازنویسی برنامه 27](#_Toc432185271)

[4‌.1‌ بازنویسی برای حالت غیرحساس به پیشرفت 29](#_Toc432185272)

[4‌.2‌ بازنویسی برای حالت حساس به پیشرفت 33](#_Toc432185273)

[5 فصل پنجم پیاده‌سازی و ایجاد رابط کاربری 39](#_Toc432185274)

[5‌.1‌ تحلیل و طراحی نرم‌افزار 40](#_Toc432185275)

[5‌.2‌ شرح کلی مراحل پیاده‌سازی و ابزارهای مورد استفاده 44](#_Toc432185276)

[5‌.3‌ ایجاد رابط کاربری گرافیکی 46](#_Toc432185277)

[5‌.4‌ راستی‌آزمایی و آزمون 48](#_Toc432185278)

[6 فصل ششم جمع‌بندی و کارهای آینده 59](#_Toc432185279)

[منابع و مراجع 61](#_Toc432185280)

[پیوست الف Error! Bookmark not defined.](#_Toc432185281)

[پیوست ب 62](#_Toc432185282)

|  |  |
| --- | --- |
| فهرست اشكال | صفحه |

[شکل 1 - ساختار نحوی زبان برنامه‌نویسی WL 15](file:///D:\University\8th%20Term\Software%20BSc.%20Project_Fallah%202015\BScProject\Reports\Thesis-v2.2.docx#_Toc432193707)

[شکل 2 - نمونه کد مبدأ به زبان WL و گراف وابستگی برنامه مربوط به آن 19](file:///D:\University\8th%20Term\Software%20BSc.%20Project_Fallah%202015\BScProject\Reports\Thesis-v2.2.docx#_Toc432193708)

[شکل 3 - نمودار کلی نحوه تولید گراف وابستگی برنامه از روی کد مبدأ برنامه [15] 20](file:///D:\University\8th%20Term\Software%20BSc.%20Project_Fallah%202015\BScProject\Reports\Thesis-v2.2.docx#_Toc432193709)

[شکل 4 - نحوه تولید زیرگراف بلوک پایه و اتصال به یکدیگر [16] 21](file:///D:\University\8th%20Term\Software%20BSc.%20Project_Fallah%202015\BScProject\Reports\Thesis-v2.2.docx#_Toc432193710)

[شکل 5 - نحوه تولید زیرگراف گزارههای شرطی – حالت اول [16] 21](file:///D:\University\8th%20Term\Software%20BSc.%20Project_Fallah%202015\BScProject\Reports\Thesis-v2.2.docx#_Toc432193711)

[شکل 6 - نحوه تولید زیرگراف گزاره‌های شرطی - حالت دوم [16] 21](file:///D:\University\8th%20Term\Software%20BSc.%20Project_Fallah%202015\BScProject\Reports\Thesis-v2.2.docx#_Toc432193712)

[شکل 7 - نحوه تولید زیرگراف گزاره‌های حلقه while [16] 22](file:///D:\University\8th%20Term\Software%20BSc.%20Project_Fallah%202015\BScProject\Reports\Thesis-v2.2.docx#_Toc432193713)

[شکل 8 - محاسبه گره‌های غلبه‌کننده برای هر گره [17] 23](file:///D:\University\8th%20Term\Software%20BSc.%20Project_Fallah%202015\BScProject\Reports\Thesis-v2.2.docx#_Toc432193714)

[شکل 9- محاسبه گره‌‌های پس‌غلبه‌کننده مرزی برای هر گره [17] 25](file:///D:\University\8th%20Term\Software%20BSc.%20Project_Fallah%202015\BScProject\Reports\Thesis-v2.2.docx#_Toc432193715)

[شکل 10 - الگوریتم کلی بازنویسی برای اعمال خط مشی عدم تداخل [5] 28](file:///D:\University\8th%20Term\Software%20BSc.%20Project_Fallah%202015\BScProject\Reports\Thesis-v2.2.docx#_Toc432193716)

[شکل 11 - الگوریتم بازنویسی عدم تداخل حالت غیرحساس به پیشرفت که برنامه M و گراف وابستگی برنامه مربوط به آنِ G را می‌گیرد [2] 31](file:///D:\University\8th%20Term\Software%20BSc.%20Project_Fallah%202015\BScProject\Reports\Thesis-v2.2.docx#_Toc432193717)

[شکل 12 – (الف) نمونه کد به زبان WL (ب) برنامه بازنویسی‌شده برنامه الف در حالت غیرحساس به پیشرفت (پ) گراف وابستگی برنامه الف 32](file:///D:\University\8th%20Term\Software%20BSc.%20Project_Fallah%202015\BScProject\Reports\Thesis-v2.2.docx#_Toc432193718)

[شکل 13 - (الف) نمونه برنامه به زبان WL (ب) برنامه بازنویسی‌شده الف برای حالت غیرحساس به پیشرفت، که حالت حساس به پیشرفت را برآورده نمی‌کند 33](file:///D:\University\8th%20Term\Software%20BSc.%20Project_Fallah%202015\BScProject\Reports\Thesis-v2.2.docx#_Toc432193719)

[شکل 14 – برنامه‌ای که حلقه موجود در آن در حالتی که h1 >= l1 or l1 < 0 باشد، خاتمه خواهد یافت 35](file:///D:\University\8th%20Term\Software%20BSc.%20Project_Fallah%202015\BScProject\Reports\Thesis-v2.2.docx#_Toc432193720)

[شکل 15 - الگوریتم بازنویسی عدم تداخل حالت حساس به پیشرفت که برنامه M و گراف وابستگی برنامه مربوط به آنِ G را می‌گیرد [2] 37](file:///D:\University\8th%20Term\Software%20BSc.%20Project_Fallah%202015\BScProject\Reports\Thesis-v2.2.docx#_Toc432193721)

[شکل 16 - کد مبدأ بازنویسی‌شده توسط الگوریتم حالت حساس به پیشرفت برای برنامه شکل 14 38](file:///D:\University\8th%20Term\Software%20BSc.%20Project_Fallah%202015\BScProject\Reports\Thesis-v2.2.docx#_Toc432193722)

[شکل 17 - مدل فرآیندی آبشاری 40](file:///D:\University\8th%20Term\Software%20BSc.%20Project_Fallah%202015\BScProject\Reports\Thesis-v2.2.docx#_Toc432193723)

[شکل 18 - نمودار موردکاربرد نرم‌افزار پروژه 41](file:///D:\University\8th%20Term\Software%20BSc.%20Project_Fallah%202015\BScProject\Reports\Thesis-v2.2.docx#_Toc432193724)

[شکل 19 - نمودارهای فعالیت نرم‌افزار پروژه 42](file:///D:\University\8th%20Term\Software%20BSc.%20Project_Fallah%202015\BScProject\Reports\Thesis-v2.2.docx#_Toc432193725)

[شکل 20 - نمودار کلاس نرم‌افزار پروژه (بدون ذکر فیلدها و متدها) 43](file:///D:\University\8th%20Term\Software%20BSc.%20Project_Fallah%202015\BScProject\Reports\Thesis-v2.2.docx#_Toc432193726)

[شکل 21 - نمای کلی رابط کاربری گرافیکی نرم‌افزار 47](file:///D:\University\8th%20Term\Software%20BSc.%20Project_Fallah%202015\BScProject\Reports\Thesis-v2.2.docx#_Toc432193727)

[شکل 22 – نمونه‌ای از اجرای برنامه در رابط کاربری گرافیکی نرم‌افزار 48](file:///D:\University\8th%20Term\Software%20BSc.%20Project_Fallah%202015\BScProject\Reports\Thesis-v2.2.docx#_Toc432193728)

[شکل 23 – (الف) برنامه مورد آزمون با نام 02basic.wl (ب) گراف وابستگی برنامه مربوط به برنامه الف (پ) برنامه بازنویسی‌شده برای حالت غیرحساس به پیشرفت مربوط به برنامه الف 49](file:///D:\University\8th%20Term\Software%20BSc.%20Project_Fallah%202015\BScProject\Reports\Thesis-v2.2.docx#_Toc432193729)

[شکل 24- (الف) برنامه مورد آزمون با نام 03assign.wl (ب) گراف وابستگی برنامه مربوط به برنامه الف (پ) برنامه بازنویسی‌شده برای حالت غیرحساس به پیشرفت مربوط به برنامه الف 49](file:///D:\University\8th%20Term\Software%20BSc.%20Project_Fallah%202015\BScProject\Reports\Thesis-v2.2.docx#_Toc432193730)

[شکل 25- (الف) برنامه مورد آزمون با نام 05if2.wl (ب) برنامه بازنویسی‌شده برای حالت غیرحساس به پیشرفت مربوط به برنامه الف (پ) گراف وابستگی برنامه مربوط به برنامه الف 50](file:///D:\University\8th%20Term\Software%20BSc.%20Project_Fallah%202015\BScProject\Reports\Thesis-v2.2.docx#_Toc432193731)

[شکل 26- (الف) برنامه مورد آزمون با نام 07ifelseadvanced.wl (ب) برنامه بازنویسی‌شده برای حالت غیرحساس به پیشرفت مربوط به برنامه الف (پ) گراف وابستگی برنامه مربوط به برنامه الف 52](file:///D:\University\8th%20Term\Software%20BSc.%20Project_Fallah%202015\BScProject\Reports\Thesis-v2.2.docx#_Toc432193732)

[شکل 27 - (الف) برنامه مورد آزمون با نام 11whilewhileconcat.wl (ب) برنامه بازنویسی‌شده برای حالت غیرحساس به پیشرفت مربوط به برنامه الف (پ) برنامه بازنویسی‌شده برای حالت حساس به پیشرفت مربوط به برنامه الف (ت) گراف وابستگی برنامه مربوط به برنامه الف 54](file:///D:\University\8th%20Term\Software%20BSc.%20Project_Fallah%202015\BScProject\Reports\Thesis-v2.2.docx#_Toc432193733)

[شکل 28 - (الف) برنامه مورد آزمون با نام 17whilewhilenested.wl (ب) برنامه بازنویسی‌شده برای حالت غیرحساس به پیشرفت مربوط به برنامه الف (پ) برنامه بازنویسی‌شده برای حالت حساس به پیشرفت مربوط به برنامه الف (ت) گراف وابستگی برنامه مربوط به برنامه الف 56](file:///D:\University\8th%20Term\Software%20BSc.%20Project_Fallah%202015\BScProject\Reports\Thesis-v2.2.docx#_Toc432193734)

|  |  |
| --- | --- |
| فهرست جداول | صفحه |

[جدول 1 – نمونه ورودی‌ها و خروجی‌ها برای برنامه به زبان C متناظر با برنامه شکل 26-(الف) 51](#_Toc432154137)

[جدول 2 - نمونه ورودی‌ها و خروجی‌ها برای برنامه به زبان C متناظر با برنامه شکل 26-(ب) 51](#_Toc432154138)

[جدول 3 - نمونه ورودی‌ها و خروجی‌ها برای برنامه به زبان C متناظر با برنامه شکل 27-(الف) 53](#_Toc432154139)

[جدول 4 - نمونه ورودی‌ها و خروجی‌ها برای برنامه به زبان C متناظر با برنامه شکل 27-(ب) 53](#_Toc432154140)

[جدول 5 - نمونه ورودی‌ها و خروجی‌ها برای برنامه به زبان C متناظر با برنامه شکل 28-(الف) 55](#_Toc432154141)

[جدول 6 - نمونه ورودی‌ها و خروجی‌ها برای برنامه به زبان C متناظر با برنامه شکل 28-(ب) 55](#_Toc432154142)

[جدول 7 - نمونه ورودی‌ها و خروجی‌ها برای برنامه به زبان C متناظر با برنامه شکل 28-(پ) 55](#_Toc432154143)

[جدول 8 - نمونه ورودی‌ها و خروجی‌ها برای برنامه به زبان C متناظر با برنامه شکل 29-(الف) 57](#_Toc432154144)

[جدول 9 - نمونه ورودی‌ها و خروجی‌ها برای برنامه به زبان C متناظر با برنامه شکل 29-(ب) 57](#_Toc432154145)

[جدول 10 - نمونه ورودی‌ها و خروجی‌ها برای برنامه به زبان C متناظر با برنامه شکل 29-(پ) 57](#_Toc432154146)

# ‌ فصل اول مقدمه

**مقدمه**

با گسترش روزافزون سیستم‌های کامپیوتری، امنیت ذخیره‌سازی و انتقال اطلاعات بیش از پیش اهمیت پیدا کرده است. امنیت اطلاعات در جنبه‌های گوناگونی نظیر امنیت شبکه‌های کامپیوتری، امنیت پایگاه داده، امنیت برنامه‌های کاربردی و غیره مورد توجه پژوهشگران این رشته است. در گذشته، مسائل امنیتی بیشتر مورد توجه مراکز نظامی و سیاسی بوده است، اما اکنون برای مردم و کاربران عادیِ سیستم‌ها نیز حائز اهمیت است.

یکی از زمینه‌های مطرح در امنیت اطلاعات و ارتباطات، امنیتِ برنامه‌های کاربردی و به تبع آن، امنیتِ زبان‌های برنامه‌نویسی یا امنیتِ زبان‌مبنا[[1]](#footnote-1) می‌باشد. امنیت زبان‌مبنا را می‌توان مجموعه‌ای از تکنیک‌های مبتنی بر نظریه زبان‌های برنامه‌سازی و پیاده‌سازی آن‌ها شامل معناشناخت[[2]](#footnote-2)، نوع‌ها[[3]](#footnote-3)، بهینه‌سازی و راستی‌آزمایی[[4]](#footnote-4) برای به‌کارگیری در مسائل امنیتی تعریف کرد. [1] تلاش این حوزه بر آن است که برنامه‌های کاربردیِ تولید شده توسط برنامه‌نویسان و توسعه‌دهندگان، با توجه به رویکردهای مختلف امنیتی، قابل اعتماد و اطمینان باشند. به همین دلیل، طراحی و توسعه زبان‌های برنامه‌نویسی امن یا ایجاد ابزارهای کمکی بر روی زبان‌های برنامه‌نویسی موجود باعث می‌شود تا توسعه‌دهندگان نرم‌افزار، کمتر درگیر مسائل امنیتی برنامه‌های خود شده و به کمک این ابزارها، با تلاش کمتری به تولید برنامه‌های امن بپردازند، که این خود موجب کاهش هزینه‌های تولید و توسعه نرم‌افزارها می‌شود.

روش‌های مختلفی برای تولید ابزارهای مرتبط با زبان‌های برنامه‌نویسی با رویکرد برآورده کردن نیازها و خط مشی‌های امنیتی وجود دارد که به طور کلی می‌توان به دو دسته روش‌های تحلیل ایستا[[5]](#footnote-5) یا زمان‌کامپایل[[6]](#footnote-6) و تحلیل پویا[[7]](#footnote-7) یا زمان‌اجرا[[8]](#footnote-8) دسته‌بندی کرد. تحلیل جریان داده[[9]](#footnote-9)، تحلیل نوع‌مبنا[[10]](#footnote-10)، وارسی مدل[[11]](#footnote-11) و تفسیر انتزاعی[[12]](#footnote-12) نمونه‌هایی از انواع تحلیل‌های ایستا هستند. انواع مختلف ناظران اجرا[[13]](#footnote-13) نیز در تحلیل‌های زمان‌اجرا کاربرد دارند. البته نمی‌توان مرز دقیقی بین دو روش تحلیل ایستا و پویا مشخص کرد و هر کدام از این روش‌ها نقاط قوت و ضعف مربوط به خود را دارند که بنا به کاربرد، استفاده از هر یک از آن‌ها متفاوت خواهد بود. شایان ذکر است که تعریف و مشخص کردن دقیق خط مشی[[14]](#footnote-14) امنیتی، یکی از چالش‌های پیشِ روی متخصصان این حوزه می‌باشد. چنان‌که نحوه و رویکرد اِعمال آن نیازمندی امنیتی، وابستگی زیادی به تعریف ارائه شده خواهد داشت.

هدف از این پروژه، تولید ابزاری برای تشخیص برقرار بودن خط مشی عدم تداخل در کد مبدأ ورودی است که در صورت تشخیص نقض این خط مشی، با بهره‌گیری از روش بازنویسی برنامه، کد مبدأ به نحوی اصلاح ‌شود تا این نیازمندی برآورده گردد. در این‌جا، خط مشی امنیتیِ عدم تداخل[[15]](#footnote-15) به عنوان نیازمندی امنیتی در نظر گرفته می‌شود و برای اِعمال این خط مشی در برنامه‌ها، از یکی از روش‌های تحلیل ایستا؛ یعنی روش بازنویسی برنامه[[16]](#footnote-16) استفاده می‌شود که در ادامه این پایان نامه، به شرح و توضیح آن‌ها می‌پردازیم.

در فصل دوم، به توضیح خط مشی امنیتی عدم تداخل و تعریف آن پرداخته خواهد شد و پس از آن، مکانیزم‌های اعمال آن و به ویژه، روش بازنویسی برنامه شرح داده خواهد شد. فصل سوم حاوی توصیف زبان مدل مطرح شده، گراف وابستگی برنامه[[17]](#footnote-17) و کاربرد آن در پروژه می‌باشد. فصل چهارم به توضیح الگوریتم مورد نظر برای بازنویسی کد مبدأ[[18]](#footnote-18) در دو حالت خط مشی عدم تداخل؛ یعنی حالت غیرحساس و حساس به پیشرفت تخصیص یافته است و فصل پنجم به فرآیند پیاده‌سازی و تولید ابزار می‌پردازیم. در نهایت، فصل ششم دربرگیرنده جمع‌بندی و کارهای پیشنهادی آینده پروژه خواهد بود.

# فصل دوم خط مشی امنیتی عدم تداخل و اِعمال آن

**خط مشی امنیتی عدم تداخل و اعمال آن**

## خط مشی امنیتی عدم تداخل

به طور کلی، خط‌ مشیِ امنیتی، امن‌بودنِ یک سیستم یا برنامه را تعریف می‌کند که در آن اَعمال مجاز و غیرمجاز در آن سیستم یا برنامه مشخص می‌شود. خط مشی امنیتی، قیودِ روی توابع و جریان‌‌های بین آن‌ها را مشخص می‌کند؛ مثل قیود دسترسی‌ بر روی برنامه‌ها و سطوح دسترسیِ داده‌های بین کاربران که مانع از بروز مشکلات امنیتی از طریق سیستم‌های خارجی و نفوذگران شود.

از منظر دیگری، یک خط مشی امنیتی را می‌توان به‌ عنوان یک زیرمجموعه از مجموعه‌ توانیِ همه اجراها تعریف کرد که هر اجرا یک دنباله دلخواه از حالت‌[[19]](#footnote-19)ها است. همچنین، می‌‌توان آن را به ‌عنوان مجموعه‌ برنامه‌هایی در نظر گرفت که آن خط مشی را برآورده می‌کنند. از این حیث، می‌توان   
خط مشی‌های امنیتی را به دو دسته تقسیم کرد. بعضی از خط مشی‌های امنیتی، *خاصیت*[[20]](#footnote-20) هستند؛ زیرا قابل دسته‌بندی و تشخیص توسط مجموعه اجراهای جداگانه می‌باشند؛ یعنی می‌توان آن‌ها را به صورت مجموعه‌ای از اجراها بیان کرد. از این نوع خط مشی‌ها می‌توان به خط‌ مشی‌های کنترل دسترسی اشاره کرد. برخی از نیازمندی‌های مهم امنیتی، خاصیت نیستند. یک نمونه مهم از این‌گونه نیازمندی‌ها، خط مشیِ امنیتی عدم تداخل[[21]](#footnote-21) است. [2] عدم تداخل گوگن-مِسِگر [3] مثالی از خط مشی‌هایی است که نمی‌‌توان آن‌ها را در قالب خاصیت بیان کرد و اصطلاحاًبه آن‌ها *فوق خاصیت[[22]](#footnote-22)* گفته می‌شود [4]. نکته حائز اهمیت این است که روش اِعمال خاصیت‌ها با نحوه اعمال خط مشی‌هایی که خاصیت نیستند، متفاوت است.

خط مشی عدم تداخل بیان می‌کند که یک مشاهده‌گرِ[[23]](#footnote-23) سطحِ پایین که فقط به برنامه و مقادیر عمومیِ زمانِ اجرا دسترسی دارد، نتواند ورودی‌های سطح بالا یا خصوصیِ برنامه را بفهمد. به عبارت دیگر، در هر دو اجرا از برنامه‌ که ورودی‌های عمومی یکسان هستند، خروجی‌های عمومی نیز باید یکی باشند. این بدان معناست که طبق این خط مشی، تغییرات ورودی‌های سطح بالا، نباید برای مشاهده‌گر سطح پایین قابل تشخیص و درک باشد.

نکته مهم این است که خط مشی عدم تداخل، یک خاصیت نیست؛ زیرا توسط اجراهای جداگانه که این خط مشی را برآورده می‌کند، قابل تعریف نیست. این نکته باعث ایجاد محدودیت‌هایی برای اِعمال این خط مشی در برنامه‌ها می‌شود.

خط مشی عدم تداخل را می‌توان به دو دسته *حساس به پیشرفت[[24]](#footnote-24)* و *غیر حساس به پیشرفت[[25]](#footnote-25)* تقسیم کرد. [5] در عدم تداخلِ غیر حساس به پیشرفت، مشاهده‌گرِ سطح پایین تنها می‌تواند خروجی‌های میانیِ سطح پایین برنامه را ببیند. این در حالیست ‌که یک مشاهده‌گرِ سطح پایین در   
عدم تداخلِ حساس به پیشرفت، علاوه بر دسترسی‌های قبلی، به *وضعیت پیشرفتِ[[26]](#footnote-26)* برنامه نیز دسترسی دارد. این باعث می‌شود تا بتواند تفاوت بین واگرایی[[27]](#footnote-27) برنامه با موقعیتی که برنامه خاتمه[[28]](#footnote-28) می‌یابد یا در حال محاسبه مقادیر قابل‌مشاهده‌ بعدی است را تمیز دهد.

با تعاریف بالا، سیستم یا برنامه‌ای که خروجی‌های سطح پایین آن از ورودی‌های سطح بالا تأثیر نگیرد، خط مشی عدم تداخل را برآورده می‌کند. حال باید توجه داشت که ممکن است جریان اطلاعاتِ[[29]](#footnote-29) از سطح بالا به پایین، صریح[[30]](#footnote-30) یا ضمنی[[31]](#footnote-31) باشد. انتساب[[32]](#footnote-32) یک مقدار سطح بالا به یک متغیر سطح پایین، نمونه‌ای از جریان اطلاعات صریح است. همچنین، جریانِ از بالا به پایین در زمانی‌که مقدار یک متغیر سطح پایین یا زمان‌بندی و رفتار خاتمه برنامه، به اجرای دستور حاوی یک مقدار سطح بالا وابسته باشد، می‌تواند نمونه‌ای از جریان اطلاعات ضمنی باشد.

در این سیستم، توانایی‌های کاربرانِ سطح پایین بیانگر مدل مهاجم[[33]](#footnote-33) خواهد بود. با توجه به این توانایی‌ها، روش‌های مختلفی برای جریان اطلاعات از بالا به پایین وجود خواهد داشت. یک مکانیزم اعمال خط مشی، باید همه انواع مختلف جریان‌های غیرمجاز ناشی از مدل مهاجم را در نظر بگیرد.

## مروری بر کارهای گذشته

وُلپانو و اسمیت [2] نشان می‌دهند که در صورتی که یک برنامه در نوع‌سامانه[[34]](#footnote-34) پیشنهادی آن‌ها خوش‌نوع[[35]](#footnote-35) شناخته شود، خاصیت امنیتی عدم تداخل را برآورده خواهد کرد. در این کار که جزو کارهای اولیه این حوزه محسوب می‌شود، الگوریتمی برای استنتاج و ساده‌سازی نوع‌های اولیه نیز ارائه شده است. باید در نظر داشتکه به طور کلی، ذات نوع‌سامانه‌ها محافظه‌کارانه[[36]](#footnote-36) است.

وِنکَتَکریشنَن و همکارانش [6] یک روش تبدیل برنامه‌ی ترکیبی برای اِعمال عدم تداخل ارائه داده‌اند که از روش‌های زمان‌اجرا برای صحت و از روش‌های تحلیل ایستا برای کامل بودنِ روش خود استفاده کرده‌اند. برنامه تغییر داده‌ شده، سطوح امپنیتی انتساب[[37]](#footnote-37) را پیگیری می‌کند و زمانی‌که یک جریان غیرمجاز در حال وقوع باشد، خاتمه می‌یابد. این روش، تنها در صورت‌بندی‌هایی قابل استفاده است که از عدم تداخلِ بدون توجه به رفتار خاتمه‌ی برنامه‌ها مطرح می‌شود و جریان‌های ضمنی که بدون انتساب ایجاد می‌شوند را تشخیص نمی‌دهد.

مَگَزینیوس و همکارانش [7] یک چارچوب برای ناظر‌های امنیتی پویای درون‌برنامه‌ای ساخته‌اند. این روش، عدم تداخلِ غیر حساس به خاتمه را تضمین می‌کند و قابل به‌کارگیری در زبان‌های پرل[[38]](#footnote-38) و جاوا اسکریپت[[39]](#footnote-39) است که از ارزیابی پویای کد پشتیبانی می‌کنند. ضمناً در این روش لازم است که تغییردهنده‌ی برنامه در زمانِ اجرا در دسترس باشد که یک ناظرِ مناسب بتواند در کدی که به صورت پویا تولید می‌شود، ورود کند.

لِگوئِرنیک و همکارانش [8] یک ماشین[[40]](#footnote-40) طراحی کرده‌اند که رخداد‌های انتزاعی[[41]](#footnote-41) در زمان اجرا را دریافت می‌کند و اجرا توسط بعضی از اطلاعات ایستا ویرایش می‌شود. اشکال این روش آن‌جاست که اجازه وقوع کانال‌های خاتمه را نیز می‌دهد.

این موضوع اثبات‌ شده است که هیچ روش کاملاً پویایی برای اِعمال عدم تداخل حساس به جریان وجود ندارد. [9] حساس به جریان به این معنا که ترتیب گزاره‌های برنامه حائز اهمیت است و برای هر گزاره باید یک تحلیل جداگانه محاسبه شود. این موضوع باعث می‌شود که پروژه‌هایی که محدودیت‌های نَحوی[[42]](#footnote-42) بر روی کد دارند، از اطلاعات ایستا در ناظری بر اجراهای چندگانه‌ی برنامه‌ها استفاده کنند.

## اِعمال خط مشیِ عدم تداخل

همان‌طور که در بالا آمده است، برای اعمال خط مشی عدم تداخل روش‌ها و مکانیزم‌های گوناگونی وجود دارد. اما باید توجه داشت که به طور کلی، مسئله تشخیص برنامه‌هایی که عدم تداخل را برآورده می‌کنند، تصمیم‌ناپذیر[[43]](#footnote-43) است. [2] پس در حالت کلی، عدم تداخل توسط روش‌های ایستا قابل اعمال نیست؛ به همین دلیل است که نوع‌سامانه‌های ارائه‌شده برای این مسئله، محافظه‌کارانه هستند و ممکن است بعضی برنامه‌های امن را نیز نپذیرند. از طرفی، این مسئله مکمل شمارش‌پذیر بازگشتی[[44]](#footnote-44) نیز نیست. [10] بنابراین، توسط ناظر‌های اجرا که نقضِ خط مشی عدم تداخل در یک برنامه‌ی در حال اجرا را بررسی می‌کنند، قابل اعمال نیست.

یکی از روش‌های مورد استفاده برای اعمال خط مشی‌هایی که خاصیت نیستند، روش بازنویسی برنامه می‌باشد. بازنویسی برنامه مکانیزمی است که یک برنامه داده شده را به برنامه‌ای تبدیل می‌کند که ویژگی‌های مورد نظر را برآورده می‌کند. این روش اخیراً به عنوان مکانیزمی برای اعمال خط مشی‌های امنیتی پیشنهاد شده است. [1] لذا می‌توان از روش بازنویسی برنامه برای اعمال خاصیت‌های امنیتی گوناگونی استفاده کرد. از طرف دیگر، علی‌رغم تلاش‌های بسیاری که در این زمینه صورت گرفته است، جنبه‌های مهمی از سرشت‌نمایی[[45]](#footnote-45) صوری بازنویسی برنامه کماکان از جمله مباحث باز این حوزه به شمار می‌رود.

در روش بازنویسی برنامه، جلوی اجرای یک برنامه مغایر با خط مشی، قبل یا در هنگام اجرا، گرفته نمی‌شود؛ بلکه آن‌ها با تغییراتی متناسب با نیاز امنیتی خواسته شده بازنویسی می‌شوند و برنامه‌هایی امن تولید خواهند شد. این در حالیست که اکثر مکانیزم‌های اعمال خط مشی‌های امنیتی، اجازه اجرای برنامه ناقض خط مشی را نمی‌دهند. در تحلیل‌های ایستا، برنامه قبل از اجرا بررسی می‌شود و در صورتی که خط مشی مورد نظر را برآورده نسازد، به طور کلی برنامه به مرحله اجرا نمی‌رسد. در تحلیل‌های پویا، در هر زمانی از اجرای برنامه، در صورت تشخیص مغایرت برنامه با خط مشی، اجرای برنامه متوقف خواهد شد. حال آن‌که در روش بازنویسی برنامه، تحلیل و تغییر برنامه قبل از اجرا انجام می‌شود اما خط مشی امنیتی مورد نظر، در زمان اجرا اعمال می‌شود. همچنین، می‌توان نشان داد که بازنویسی برنامه از نظارت اجرا و تحلیل ایستا قدرت بیشتری دارد. [5] از این رو، می‌توان روش بازنویسی برنامه را روشی بین روش‌های ایستا و روش‌های پویا دانست.

از طرفی، بازنویسیِ برنامه تغییراتی به ذاتِ عدم تداخل وارد نمی‌کند؛ بلکه برنامه‌ای جدید تولید می‌کند هم عدم تداخل را برآورده می‌کند و هم حداقل تغییرات ممکن نسبت به برنامه اصلی دارد.‌ به این ترتیب، برتری این روش بر مکانیزم‌های ایستا و نظارتی مشخص خواهد شد.

یک بازنویس برنامه باید با توجه به خط مشی امنیتی مورد نظر، صحیح[[46]](#footnote-46) و شفاف[[47]](#footnote-47) باشد. صحت به این معنا که کد تولید شده توسط آن، خط مشی را برآورده کند و شفاف بودن به معنای این‌که معناشناخت و رفتارهای مناسب برنامه، فارغ از امن بودن یا نبودن آن، حفظ بماند. به بیان ساده‌تر، شفاف بودن یک بازنویس بدین معناست که تا حد امکان، مجموعه اجراهای ممکنِ برنامه‌ تبدیل‌شده، خواه امن یا ناامن، مشابه برنامه‌ ورودی باشد. البته تا زمانی‌که صحت و شفافیت دچار خدشه نشود، بازنویس برنامه می‌‌تواند هر تغییری را به کد داده شده اعمال کند. بدیهی است که یک بازنویس حتماً باید صحیح باشد تا نیاز امنیتی مورد نظر را برآورده سازد، اما شفافیتِ بیشتر در الگوریتم بازنویسی، باعث برتری آن روش بر دیگر بازنویس‌های مشابه خواهد بود.

روش بازنویسی برنامه مورد استفاده در این پروژه، از گرافِ وابستگیِ برنامه [11] استفاده می‌کند که پیشتر اثبات شده است که در تشخیص جریان‌های اطلاعاتی احتمالی، دست‌کم به قدرت نوع‌سامانه‌های امنیتی است. [12] برخلاف مکانیزم‌های کنترل جریان اطلاعات مطرح شده، بازنویس مورد استفاده در این‌ پروژه، رفتارهای معتبر برنامه‌ها را نگه می‌دارد؛ بدین معنا که آن دسته از اجراهای برنامه که به نقض امنیتی منجر نمی‌شوند، تغییری نخواهند کرد. این به دلیل رویکرد تعریف و اعمال شفافیت -چالش‌برانگیزترین نیازمندی برای یک مکانیزم اعمال کارا- است. در این روش، جریان‌های غیرمجاز صریح و ضمنیِ کد داده شده، با اصلاح کد برطرف می‌شود. این اصلاح همان‌قدر کاراست که در صورت وقوع آن‌ها در زمان اجرا، جلوی آن‌ها گرفته می‌شود. همچنین در روش مورد استفاده، از شرط‌های مسیر[[48]](#footnote-48) به منظور بهبود تشخیص شروط مورد نیاز برای برقراری جریان غیرمجاز استفاده می‌شود. با این کار، رفتارهای معتبر بیشتری از برنامه داده شده حفظ خواهد شد. تعریف مفهوم شفافیت به گونه‌ای است که مجموعه اجراهای ممکن برنامه تبدیل‌شده، به شباهت مجموعه اجراهای ممکن برنامه ورودی باشد. می‌توان اثبات کرد که روش مورد استفاده در این پروژه صحیح و شفاف است [5]، پس قطعاً برنامه‌هایی که توسط آن تولید می‌شوند، امن خواهند بود. ضمناً برای اثبات شفافیت الگوریتم، از بیان دقیق و صوری مفهوم اِعمال اصلاحی[[49]](#footnote-49) و رابطه پیش‌ترتیبی[[50]](#footnote-50) استفاده شده است.

توضیحات بیشتر و جزئیات الگوریتم بازنویسی برنامه مورد استفاده در این پروژه، در فصل چهارم به تفصیل آمده است.

# فصل سوم زبان WL و گراف وابستگی برنامه

**زبان WL و گراف وابستگی برنامه**

## توصیف زبان برنامه‌نویسی WL

برای تعریف دقیق و صوری خط مشیِ عدم تداخل و روش اعمال بازنویسی برنامه، به تعیین یک زبان برنامه‌نویسی نیاز داریم. از همین رو، زبان مدلی به نام While Language یا به اختصار WL در نظر گرفته می‌شود. [5] در این پروژه، این زبان برنامه‌نویسی بازطراحی و پیاده‌سازی شده است و برنامه‌های نوشته‌شده به این زبان، از طریق ابزار پیاده‌سازی شده، به زبان برنامه‌نویسی C قابل تبدیل می‌باشند. برای این کار، از ابزارهای jflex [13] و bison [14] استفاده شده که در فصل پنجم توضیحات کاملی در این خصوص ارائه می‌گردد. در این فصل به معرفی و توصیف زبان؛ یعنی ساختار نحوی و معناشناخت می‌پردازیم.

### ساختار نحوی و گرامر زبان

در شکل 1، ساختار نحوی تجریدی[[51]](#footnote-51) زبانِ WL آمده است. نماد‌های پررنگ‌شده به معنای پایانه‌های زبان هستند و سایر نمادها نشانگر غیرپایانه‌های این زبان می‌باشند.

یک عبارت که در گرامر زبان با نماد exp مشخص شده است، می‌تواند یک عدد صحیح ثابت، مقدار منطقی بولی[[52]](#footnote-52)، یک متغیر عددی و یا یک عملیات یا ارتباط بین یک یا چند عبارت دیگر باشد.

مجموعه دستورات، شامل NOP برای دستور *عملیات هیچ[[53]](#footnote-53)*، x = exp برای انتساب یک عبارت به یک متغیر، inL varlist، inH varlist، outL x و outH x برای ورودی گرفتن و خروجی دادن هستند که پسوندهای L و H، به ترتیب، به معنای سطح امنیتی پایین[[54]](#footnote-54) یا بالا[[55]](#footnote-55) در دستورات ورودی و خروجی می‌باشند. دستور outL BOT یا outH BOT، دستورات خروجی منحصر به فردی هستند که مقدار ثابت BOT یا ⊥ را به عنوان خروجی برای مشاهده‌گران سطح پایین یا بالا فراهم می‌کنند که این مقدار از همه مقادیر ثابت موجود در زبان متمایز است. دستورات شرطی if-then و if-then-else و دستور ایجاد حلقه while از دیگر دستورات این زبان به شمار می‌روند.

program ::= **program** **;** clist

clist ::= c | clist **;** c

exp ::= b | n | x | exp **==** exp | exp **<** exp | exp **<=** exp | exp **>=** exp | exp **>** exp

| exp **+** exp | exp **-** exp | exp **or** exp | exp **and** exp | **!** exp

c ::= **NOP** | x **=** exp | **inL** varlist | **inH** varlist | **outL** x | **outH** x | **outL** **BOT** | **outH** **BOT**

|**if** exp **then** clist **endif** | **if** exp **then** clist **else** clist **endif** | **while** exp **do** clist **done**

varlist ::= x | x **,** varlist

b ::= **true** | **false** | **TRUE** | **FALSE**

n ::= integer\_number

x ::= identifier

شکل 1 - ساختار نحوی زبان برنامه‌نویسی WL

قاعده clist ::= clist; c برای ایجاد دنباله دستورات است که در آن‌ها ممکن است دستورات شرطی یا حلقه وجود داشته باشد. همان‌طور که در نام‌گذاری این زبان نیز مشهود است، مهم‌ترین ساختار کنترلی زبان WL، ساختار while است که مشابه ساختارهای موجود در زبان‌های برنامه‌نویسی رایج، امکان ایجاد حلقه تکرار را برای برنامه‌نویس فراهم می‌آورد.

لازم به ذکر است که گرامر فوق، با حفظ بخش‌های اصلی زبان، تغییراتی نسبت به ساختار نحوی موجود در مقاله [5] یافته است.

### معناشناخت زبان

یک برنامه به زبان WL، با عبارت program; آغاز می‌شود و پس از آن، دستورات گرفتن ورودی از کاربر که پیشوند in را دارند، خواهد آمد. مقادیری که در ابتدای برنامه از کاربر گرفته می‌شود، در متغیرهای تعریف‌شده ذخیره می‌شود و با توجه به دستورات بعدی برنامه، مقادیر آن‌ها تغییر می‌کند. در این زبان، این فرض صورت گرفته است که فقط در ابتدای کد مبدأ برنامه، دستورات گرفتن ورودی‌های مورد نظر نوشته خواهد شد؛ گرچه دستورات خروجی در هر نقطه‌ای از برنامه می‌توانند وجود داشته باشند. نکته مهم در خصوص زبان برنامه‌نویسی WL این است که در برنامه‌های نوشته‌شده به این زبان، مقادیر در هر زمان دلخواه در طول زمان‌اجرا می‌توانند به عنوان خروجی نمایش داده شوند. این در حالیست که در بیشتر کارهای انجام شده مرتبط با امنیت جریان اطلاعات، استفاده از دستورات نمایش خروجی فقط در حالت‌های نهایی امکان‌پذیر است.

عملگرهای and و or، همان عملگرهای دوعملوندیِ منطقیِ عطف و فصل هستند. عملگر تک‌عمل‌وندی !، برای نقیض یک عبارت منطقی استفاده می‌شود. پایانه‌ true یا TRUE برای عبارت منطقی همیشه درست و false یا FALSE برای عبارت منطقی همیشه نادرست لحاظ شده است. منظور از integer\_number، همان اعداد صحیح و identifier همان شناسه‌های مورد استفاده برای نام متغیرها است که با یک حرف انگلیسی آغاز و پس از آن تعداد محدودی عدد یا حرف می‌آید.

منظور از دستور NOP، همان دستور ; در زبان C می‌باشد که بدون انجام عملیاتی، تأخیری در اجرای برنامه ایجاد می‌کند. برای دستورات شرطی، مشابه زبان‌های برنامه‌نویسی رایج، در صورت برقراری عبارت شرطی، دستورات پس از پایانه then تا else اجرا می‌شوند و در صورت عدم برقراری عبارت شرطی، اگر پایانه else وجود داشته باشد، دستورات پس از پایانه else تا endif اجرا خواهند شد. در دستور while، تا زمانی که عبارت شرطی برقرار باشد، بدنه حلقه؛ یعنی دستورات پس از پایانه do تا done اجرا می‌شوند.

با استفاده از نماد «;» می‌توان دنباله دستورات را ایجاد کرد، با این توضیح که گزاره پایانی هر دنباله از دستورات، نیازی به این نماد ندارد.

با مشاهده مثال‌های موجود در پایان‌نامه، می‌توان درک بهتری نسبت به معناشناخت این زبان پیدا کرد. همچنین، معناشناخت صوری این زبان نیز در مقاله [5] موجود است.

## گراف وابستگی برنامه

بازنویس‌های برنامه مورد استفاده در این پروژه، از گراف‌های وابستگی برنامه بهره می‌برند. در این فصل به معرفی گراف وابستگی برنامه، نحوه تولید آن و کاربرد آن در الگوریتم بازنویسی خواهیم پرداخت.

برای هر مکانیزمِ اعمال خط مشی عدم تداخل، به ماشینی برای تشخیص جریان‌های اطلاعات ممکنِ از ورودی‌های سطح بالا به خروجی‌های سطح پایین نیاز است. گراف‌های وابستگی برنامه یا به اختصار PDG[[56]](#footnote-56)ها می‌توانند این امکان را برای ما فراهم کنند. گراف وابستگی برنامه، برنامه را به شکل یک گراف جهت‌دار نمایش می‌دهد که در آن، گره‌ها بیانگر عبارت‌ها یا گزاره[[57]](#footnote-57)‌های برنامه و یال‌ها بیانگر وابستگی‌های کنترلی یا داده‌ای بین گره‌ها هستند. گراف وابستگی برنامه تمامی وابستگی‌های بین گزاره‌های آن برنامه را منعکس می‌کند.

پایه اصلی تولید گراف وابستگی برنامه، گراف جریان کنترل[[58]](#footnote-58) یا به اختصار CFG است. گراف جریان کنترل، ترتیب دنباله اجرای گزاره‌ها را بیان می‌کند. این گراف، یک گراف جهت‌دار است که گره‌ها در آن نمایانگر گزاره‌های برنامه و یال‌ها بیانگر جریان‌های کنترلی بین گره‌ها هستند. در گراف جریان کنترل، دو گره مشخص به نام‌های شروع و پایان در نظر گرفته می‌شود که نقاط ورود و خروج برنامه را تعیین می‌کنند. با به دست آوردن وابستگی‌های کنترلی و داده‌ای، گراف جریان کنترل به گراف وابستگی برنامه تبدیل خواهد شد.

### وابستگی‌های کنترلی و داده‌ای

در گراف وابستگی برنامه، یک یال وابستگی داده‌ای از گره X به گره Y، که با نمایش داده می‌شود، به این معناست که گره Y دارای متغیری است که در گره X منتسب شده است. همچنین، یک یال وابستگی کنترلی از X به Y، که با نمایش داده می‌شود، به معنای این است که اجرای گزاره Y، توسط مقدار محاسبه‌شده در گزاره X کنترل می‌شود. ضمناً می‌توان یک مسیر[[59]](#footnote-59) از X به Y در گراف وابستگی برنامه را به شکل علامت‌گذاری کرد. هر مسیر مشخص‌کننده یک وابستگی کنترلی یا داده‌ای است که بستگی به نوعِ آخرین یال آن مسیر دارد. در ضمن، مسیر ساخته‌شده به واسطه اضافه کردن یال -که به این معناست که نوع وابستگی آن اهمیتی ندارد- به مسیر ، به شکل نمایش داده می‌شود. مسیری مانند در گراف وابستگی برنامه بیانگر این است که ممکن است جریانی از X به Y وجود داشته باشد.

می‌توان چنین تعریف کرد که اگر مقدار محاسبه‌شده در Y یا صرفِ اجرای Y به مقدار محاسبه‌شده در X بستگی داشته باشد، آنگاه گوییم جریانی از X به Y در گراف وابستگی برنامه مربوط به برنامه M وجود دارد. باید توجه داشت که اگر جریانی از X به Y برقرار باشد، آنگاه یک مسیر از X به Y در گراف وابستگی برنامه وجود خواهد داشت؛ در حالی‌که عکس آن لزوماً صحیح نیست.

همان‌طور که در قبل مطرح شد، می‌توان جریانِ از X به Y بر روی مسیر را به دو نوع صریح و ضمنی دسته‌بندی کرد. یک جریان صریح زمانی برقرار است که مقدار محاسبه‌شده در X، مستقیماً به گره Y منتقل شود. این جریان می‌تواند ناشی از زنجیره انتساب‌های روی آن مسیر باشد. همچنین، یک جریان ضمنی زمانی برقرار خواهد بود که مقدار محاسبه‌شده در Y، به اجرا شدن یا نشدنِ یک گزاره خاص در مسیر وابسته باشد و اجرای آن گزاره، توسط مقدار محاسبه‌شده در X کنترل شود.

پس با تعاریف فوق می‌توان چنین گفت که مسیر روی گراف وابستگی برنامه تعیین‌کننده یک جریان صریح از X به Y است اگر همه یال‌های موجود در مسیر، از نوع وابستگی داده‌ای باشند. در غیر این صورت، آن مسیر به یک جریان ضمنی دلالت خواهد داشت.

program;

inH h1, h2;

inL l1;

while h1 > l1 do

l1 = l1 + 1;

while h2 > 0 do

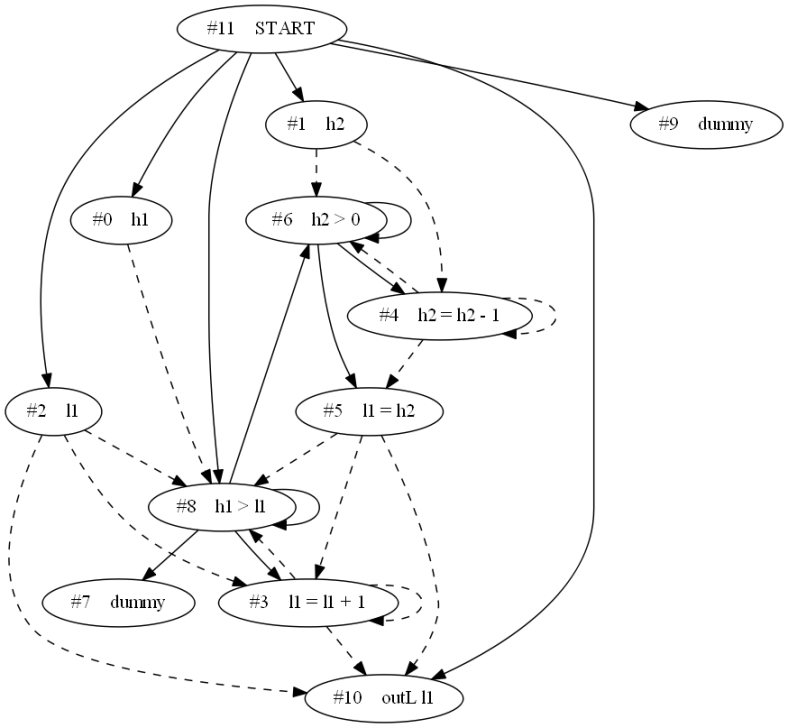
h2 = h2 - 1;

l1 = h2

done

done;

outL l1



شکل 2 - نمونه کد مبدأ به زبان WL و گراف وابستگی برنامه مربوط به آن

همان‌طور که در شکل 2 مشاهده می‌شود، یال‌های با خطوط ساده نماد وابستگی‌های کنترلی و یال‌های خط‌چین نمایانگر وابستگی‌های داده‌ای هستند.

در ادامه این فصل به توضیح نحوه ساختن گراف وابستگی برنامه از روی کد مبدأ برای زبان برنامه‌نویسی WL می‌پردازیم و در فصل بعدی، نحوه استفاده از این گراف در الگوریتم بازنویسی برنامه را به تفصیل شرح خواهیم داد.

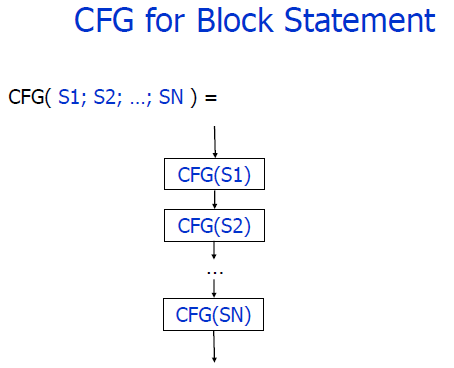
### نحوه تولید گراف وابستگی برنامه

برای ساخت گراف وابستگی برنامه، مطابق با شکل 3، گراف‌های مورد نیاز برای تحلیل ساخته می‌شود.



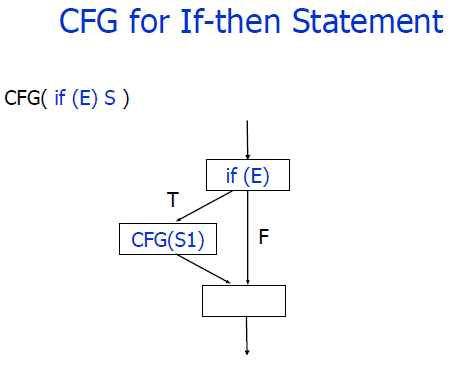
شکل 3 - نمودار کلی نحوه تولید گراف وابستگی برنامه از روی کد مبدأ برنامه [15]

ابتدا از روی کد مبدأ، گراف جریان کنترل یا CFG به دست می‌آید. نحوه تولید این گراف به این نحو است که گزاره‌ها و عبارت‌های برنامه، به عنوان یک گره در گراف در نظر گرفته می‌شود. در این گراف، مفهومی به نام بلوک پایه[[60]](#footnote-60) مطرح می‌شود. هر بلوک پایه، شامل تعدادی گره است که تنها یک گره ورودی و یک گره خروجی در آن وجود دارد. به این منظور، برای گزاره‌‌های ساده که اجرای آن‌ها مشروط نیست، به صورت دنباله پشت سرِ همی از گره‌ها در نظر گرفته می‌شود. پس برای این‌گونه گزاره‌ها و عبارت‌ها، تنها ساختن یک گره جدید و متصل کردن آن‌ها به گراف کفایت می‌کند. شکل 4، نحوه تولید زیرگراف بلوک‌های پایه را نمایش می‌دهد. با همین روش، بلوک‌های پایه با یکدیگر ادغام می‌شوند و در نهایت، گراف جریان کنترل نهایی تولید خواهد شد.

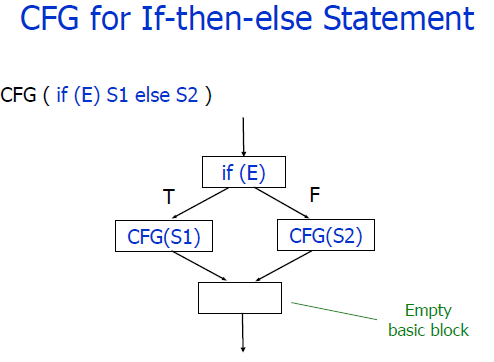


شکل 4 - نحوه تولید زیرگراف بلوک پایه و اتصال به یکدیگر [16]

برای گزاره‌های شرطی، دو حالت ممکن زیر وجود دارد:



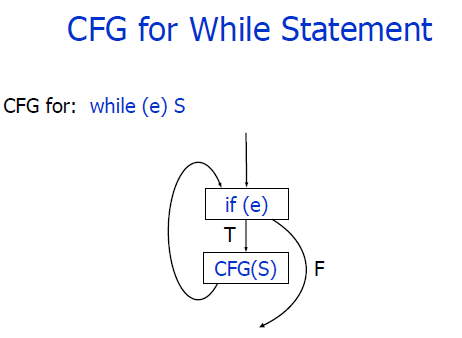
شکل 5 - نحوه تولید زیرگراف گزاره­های شرطی – حالت اول [16]



شکل 6 - نحوه تولید زیرگراف گزاره‌های شرطی - حالت دوم [16]

مطابق با شکل 5 و شکل 6، یک گره برای عبارت شرطی تولید می‌شود. گزاره‌های مربوط به برقراری عبارت شرطی در یک بلوک پایه و گزاره‌های متعلق به حالت عدم برقراری شرط، در بلوک پایه‌ دیگری در نظر گرفته می‌شود و این دو بلوک پایه، به گره مربوط به عبارت شرطی متصل خواهند شد. پس از این کار، برای گره پایانی این بلوک پایه، یک گره مجازی[[61]](#footnote-61) ایجاد می‌شود. وجود این گره برای این است که این زیرگراف تنها یک مجرای خروجی داشته باشد.

ساختار دیگری که در این زبان وجود دارد، ساختار حلقه یا همان while است. برای تولید زیرگراف جریان کنترل مربوط به این عنصر زبان، مطابق با شکل 7 عمل می‌شود.



شکل 7 - نحوه تولید زیرگراف گزاره‌های حلقه while [16]

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

حال به ازای هر قاعده موجود در زبان، مطابق با توضیحات بالا، زیرگراف‌های کنترل جریان در هنگام تولید درخت تجزیه[[62]](#footnote-62) ساخته می‌شوند و با اتصال آن‌ها به یکدیگر، گراف کنترل جریان برنامه به دست خواهد آمد.

مطابق با شکل 3، برای تولید گراف وابستگی کنترل یا به اختصار CDG[[63]](#footnote-63)، به درخت غلبه رو به جلو[[64]](#footnote-64) یا درخت پس‌غلبه[[65]](#footnote-65) نیاز خواهد بود. برای تولید این درخت، الگوریتم ساخت درخت غلبه[[66]](#footnote-66) بر روی معکوسِ گراف جریان کنترل؛ یعنی همان گره‌ها ولی با جهت یال‌های معکوس شده، اعمال می‌گردد.

برای این کار، ابتدا منظور از غلبه کردن دو گره را بیان می‌کنیم. گره M بر گره N غلبه می‌کند، اگر و تنها اگر همه مسیرهای با شروع از از گره آغازین تا گره N، حتماً و الزاماً از گره M بگذرند. همچنین، گره M بر گره N اکیداً غلبه[[67]](#footnote-67) می‌کند، اگر و تنها اگر بر آن گره غلبه کند و M، همان گره N نباشد. واضح است که یک گره در گراف جریان کنترل می‌تواند چندین غلبه‌کننده[[68]](#footnote-68) داشته باشد، اما برای تولید درخت غلبه، نزدیک‌ترین غلبه‌کننده یا غلبه‌کننده بدون واسطه[[69]](#footnote-69) اهمیت دارد. با استفاده از شبه‌کد زیر می‌توان غلبه‌کنند‌ه‌های یک گره در گراف جریان کنترل را به دست آورد.

Compute Dominators(){

For (each n ∈ NodeSet)

Dom(n) = NodeSet

WorkList = {StartNode}

While (WorkList ɸ ≠) {

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}

}

}

}

شکل 8 - محاسبه گره‌های غلبه‌کننده برای هر گره [17]

سپس با توجه به مجموعه غلبه‌کنندگانِ به دست آمده از الگوریتم بالا و مقایسه با مجموعه غلبه‌کنندگانِ سایر گره‌ها، می‌توان گره غلبه‌کننده بی‌درنگ را یافت و درخت غلبه را تشکیل داد اما در این‌جا، تولید درخت پس‌غلبه[[70]](#footnote-70) مورد نظر است. به این صورت که، گره Z، گره Y را پس‌غلبه می‌کند، اگر و تنها اگر همه مسیرهای از Y تا گره پایانی، حتماً و الزاماً از Z عبور کنند. حال، در صورتی که این الگوریتم برای معکوس گراف جریان کنترل اعمال شود، درخت پس‌غلبه تولید می‌شود.

سپس برای ساخت گراف جریان کنترل، مرزهای پس‌غلبه[[71]](#footnote-71) یا اختصاراً PDF، مورد نیاز است. مرز پس‌غلبه گره X، مجموعه‌ گره‌هایی هستند که توسط X اکیداً پس‌غلبه نمی‌شوند اما گره‌های مابَعدی[[72]](#footnote-72) دارند که توسط X پس‌غلبه می‌شوند. تعریف دقیق‌تر 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[[73]](#footnote-73) است که به گره X منجر می‌شوند.

با استفاده از قضیه زیر، می‌توان وابستگی‌های کنترلی برنامه را برای هر گره موجود در گراف جریان کنترل، به دست آورد:

**قضیه** – گره y به مجموعه PDF(X) تعلق دارد، اگر و تنها اگر X به Y وابستگی کنترلی داشته باشد. [17]

حال با استفاده از الگوریتم شکل 9، می‌توان مجموعه مرزهای پس‌غلبه هر گره را به دست آورد که بیانگر وابستگی‌های کنترلی نیز خواهند بود.

For each x in the bottom-up traversal of the postdominator tree do

PDF(X) = ɸ

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}

شکل 9- محاسبه گره‌‌های پس‌غلبه‌کننده مرزی برای هر گره [17]

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 تولید شده است.

اکنون نوبت تولید گراف وابستگی داده‌ای یا به اختصار [[74]](#footnote-74)DDG است. وابستگی‌های داده‌ای مختلفی وجود دارد اما برای این 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 باشد. پس با توجه به همین تعریف و مطابق با قواعد زبان، در گره مربوط به هر گزاره، متغیری که به آن مقداری نسبت داده شده یا استفاده شده است، نگه‌داری می‌شود. حال برای به دست آوردن وابستگی‌های داده‌ای، در صورتی که در یک گره از متغیری استفاده شود که در گره دیگری مقداردهی شده است، یک وابستگی داده‌ای لحاظ می‌شود. برای افزایش دقت 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 گراف جریان کنترل امکان‌پذیر است.

پس از این مرحله، گراف وابستگی داده‌ای برنامه نیز آماده است. بدین ترتیب گراف‌های وابستگی کنترلی و داده‌ای از روی کد مبدأ ساخته شده‌اند. با ترکیب این دو گراف که دارای گره‌های یکسان هستند، گراف وابستگی برنامه تولید خواهد شد.

در فصل بعدی، نحوه بازنویسی برنامه‌ها با استفاده از گراف وابستگی برنامه تولید شده تا این مرحله بیان می‌شود.

# فصل چهارم الگوریتم بازنویسی برنامه

**الگوریتم بازنویسی برنامه**

در این فصل ابتدا الگوریتم کلی مورد استفاده برای اعمال خط مشی عدم تداخل بیان می‌شود. سپس به الگوریتم دقیق مربوط به حالت غیرحساس به پیشرفت یا به اختصار PINI[[75]](#footnote-75) و حالت حساس به پیشرفت یا PSNI[[76]](#footnote-76) خواهیم پرداخت.

با داشتن گراف وابستگی برنامه‌ کد مبدأ ورودی، می‌توان وابستگی‌های موجود بین گزاره‌های مختلف برنامه را بررسی کرد. می‌توان تابع affect را تابعی در نظر گرفت که یک عبارت یا گزاره از یک برنامه، یا گره مربوط به آن در گراف وابستگی برنامه را به عنوان ورودی می‌گیرد وگزاره‌ها و عباراتی که به گزاره یا عبارت گرفته‌شده وابستگی دارند را به عنوان خروجی برمی‌گرداند. به بیانی دیگر، تابع affect بر روی گره داده‌شده X از گراف وابستگی برنامه اجرا می‌شود و همه گره‌های مانند Y که یک مسیر از X به آن وجود دارد را برمی‌گرداند. الگوریتم کلی بازنویسی برای خط مشی عدم تداخل را می‌توان براساس تابع پیشنهادشده ارائه کرد. تعبیر رویداد قابل‌مشاهده برای کاربر سطح پایین در این پروژه، مطابق با بیان صوری مطرح شده در مقاله [5] است.

**foreach** statement X producing a high input event h­in **do**

**foreach** statement Y producing a low observable event eL **do**

**if**  **then**

transform Y into Yˊ such that in the new program

end

end

end

شکل 10 - الگوریتم کلی بازنویسی برای اعمال خط مشی عدم تداخل [5]

همان‌طور که گفته شد، یک گراف وابستگی برنامه که بیانی ایستا از وابستگی‌های برنامه است، جریان‌های غیرمجاز ممکن را در خود دارد؛ گرچه ممکن است این جریان‌ها در همه اجراهای برنامه رخ ندهند. بنابراین در الگوریتم شکل 10، باید شرط‌هایی که تعیین‌کننده وقوع جریان غیرمجاز احتمالی هستند را لحاظ کرد.

## بازنویسی برای حالت غیرحساس به پیشرفت

ایده اصلی استفاده شده این است که دستورهای outLای که از ورودی‌های سطح بالا متأثر شد‌ه‌اند، با گزاره‌های outL ⊥ یا NOP جایگزین شوند. چنین تغییراتی که مستقل از اطلاعات زمان‌اجرا هستند، ممکن است بیش از حد مورد نیاز و کمی سخت‌گیرانه باشد. به خاطر دسترسی برنامه‌ها به اطلاعات زمان‌اجرا، می‌توان از این اطلاعات در برنامه بازنویسی‌شده استفاده کرد. به همین منظور، از گونه‌ای از شرط‌های مسیر[[77]](#footnote-77) استفاده شده است. در مقاله [5]، شرط مسیرِ p(X,Y) روی متغیرهای برنامه تعریف می‌شود و همان شرط‌هایی هستند که باعث می‌شوند تا جریان واقعاً رخ بدهد. به این معنا که شرط‌های مسیر باید برقرار باشند تا جریان مربوط به آن مسیر در اجرا نیز روی دهد. این تعریف از شرط‌های مسیر می‌تواند نشان دهد که آیا یک مسیر در زمان‌اجرا واقعاً پیمایش می‌شود یا خیر. همین نکته برای تشخیص جریان‌های صریح نیز کاربرد خواهد داشت؛ در حالی‌که برای جریان‌های ضمنی ممکن است جریان در زمان‌اجرا به وقوع بپیوندد، حتی اگر مسیر مربوط به آن به طور کامل پیمایش نشده باشد. این مورد زمانی اتفاق می‌افتد که یک گره روی مسیرِ با یال وارد شونده[[78]](#footnote-78) وابستگی کنترلی، به خاطر مقدار عبارت کنترلی اجرا نشود. پس اجرای همه نود‌های روی مسیرِ تعیین‌کننده یک جریان ضمنی، برای وقوع آن جریان الزامی نیست. به همین ترتیب، جریان مربوط به مسیر روی گراف وابستگی برنامه مربوط به یک برنامه به زبان WL فقط زمانی رخ می‌دهد که همه گره‌های آن مسیر که یالِ وارد شونده وابستگی داده‌ای دارند، اجرا شوند. می‌توان چنین گفت که همه گره‌های میانی روی مسیرِ بیان‌کننده یک جریانِ صریح، باید در زمان‌اجرا پیمایش شوند. همچنین، یک گره میانی روی مسیر مشخص‌کننده یک جریان ضمنی، فقط باید زمانی پیمایش شود که یال واردشونده به آن، از نوع وابستگی داده‌ای باشد. همان‌طور که در ادامه مطرح خواهد شد، بازنویس‌های استفاده شده در این پروژه بررسی می‌کنند که آیا همه گره‌های میانی با یال وارد شونده وابستگی داده‌ای بر روی مسیری که به دستورات outL l ختم می‌شوند، در طول برنامه اجرا ‌شوند یا خیر. اگر مسیر چنین باشد، outL ⊥ یا NOP به جای آن دستور اجرا خواهد شد، وگرنه خود دستور outL l به اجرا در می‌آید.

برنامه‌های نوشته‌شده به زبان WL حاوی شرط‌های مسیر ساده هستند که از شرط‌های اجرای[[79]](#footnote-79) گره‌ها به دست می‌آیند. به طور کلی، شرط اجرا برای گره X به کمک پیمایش معکوس[[80]](#footnote-80) با شروع از گره X تا گره آغازین، از طریق یال‌های وابستگی‌های کنترلی روی مسیر حاصل می‌شود. شرط اجرا، یک عبارت منطقی بولی است که برقرار خواهد بود، اگر و تنها اگر گزاره X اجرا شود. به همین شیوه، شرط مسیر p(X,Y) برای ، ترکیب عطفیِ شرط‌های اجرای گره‌های روی آن مسیر تعریف می‌شود. شرط‌های مسیر را می‌توان بر اساس ترکیب عطفیِ گره‌های روی مسیر با یک یال وارد شونده وابستگی داده‌ای تعریف کرد. اگر چنین گره‌ای نباشد، شرط مسیر همواره درست محسوب می‌شود.

با توجه به قاعده‌های زبان و استفاده مناسب‌تر از شرط‌های مسیر، در برنامه‌ها تنها انتساب‌های یگانه ایستا[[81]](#footnote-81) مجاز است. به معنای آن‌که برنامه‌ها حاوی انتساب‌های چندگانه برای یک متغیر نخواهد بود. البته این برنامه‌ها به برنامه‌های صرفاً حاوی انتساب‌های یگانه قابل تبدیل هستند. از طرفی، چنین فرض می‌شود که هیچ وابستگی داده‌ایِ حلقه نقلی[[82]](#footnote-82) در مسیرهای از ورودی‌های سطح بالا به خروجی‌های سطح پایین برنامه وجود ندارد. [5] پس در برنامه‌های ورودی باید به این محدودیت‌ها نیز توجه داشت.

در الگوریتم شکل 11، بازنویس مربوط به اعمال خط مشی عدم تداخل در حالت غیرحساس به پیشرفت، کد مبدأ برنامه M و گراف وابستگی برنامه G مربوط به آن را به عنوان ورودی می‌گیرد و کد مبدأ Mˊ را که عدم تداخل غیرحساس به پیشرفت را برآورده می‌کند، به عنوان خروجی برمی‌گرداند. به بیان دیگر، باید دنباله خروجی‌های برنامه بازنویسی‌شده برای هر دو اجرای دلخواه از آن برنامه که ورودی‌های سطح پایین یکسان دارند، نسبت به حالت غیرحساس به پیشرفت معادل باشند. در ادامه، نمونه‌ای از نحوه عملکرد این الگوریتم را بر روی یک برنامه داده شده به زبان WL در شکل 12 آورده شده است.

RWPINI (M, G):

Initialize to the set of all paths in the PDG *G* of *M* where *P* is the node representing a high input and is the node representing outL l for some l;

if then

return M;

end

create a copy of M, name it Mˊ, and change it as follows:

determine the type of flow indicated by each path ;

foreach do:

Generate the path condition of as the conjunction of the execution conditions of node N satisfying if there are such nodes on the path and true otherwise;

end

**foreach** node n on G representing outL l for some l **do**

let c be the disjunction of the path conditions of all which terminate at n;

**if** all paths terminating at n indicate an explicit flow **then**

replace outL l with the statement “if c then outL ⊥ else outL l endif”;

else

replace outL l with the statement “if c then NOP else outL l endif”;

end

end

return Mˊ;

شکل 11 - الگوریتم بازنویسی عدم تداخل حالت غیرحساس به پیشرفت که برنامه M و گراف وابستگی برنامه مربوط به آنِ G را می‌گیرد [5]

program;

inL l1, l2;

inH h1;

if l1 == 0 then

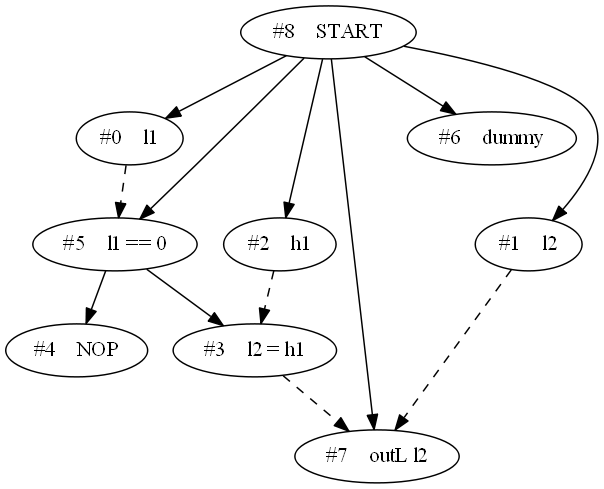
l2 = h1

else

NOP

endif;

outL l2



program;

inL l1, l2;

inH h1;

if l1 == 0 then

l2 = h1

else

NOP

endif;

if (l1 == 0) then

outL BOT

else

outL l2

endif

(ب) (الف)

(پ)

شکل 12 – (الف) نمونه کد به زبان WL (ب) برنامه بازنویسی‌شده برنامه الف در حالت غیرحساس به پیشرفت   
(پ) گراف وابستگی برنامه الف

## بازنویسی برای حالت حساس به پیشرفت

حالت حساس به پیشرفت نسبت به حالت غیرحساس به پیشرفت محدودیت و قیود بیشتری روی رفتار مشاهده‌گر سطح پایین اعمال می‌کند. می‌توان با بررسی مثالی بیشتر به آن پرداخت. در برنامه شکل 13، الگوریتم بازنویسی برای حالت غیرحساس به پیشرفت دستور outL l1 در خط ششم را با دستور NOP جایگزین می‌کند. گرچه نتیجه بازنویسی حالت غیرحساس به پیشرفت را برآورده می‌کند، اما با توجه به حالت حساس به پیشرفت، برنامه‌ای ناامن تلقی می‌شود؛ چرا که واگرایی حلقه while موجود در برنامه ممکن است متناسب با مقدار سطح بالای h1 باشد. به عبارت دیگر، یک مشاهده‌گر سطح پایین می‌تواند با بررسی روند پیشرفت برنامه، مقدار h1 را استنباط کند.

program;

inH h1;

inL l1;

if h1 == l1 then

while true do

outL l1

done

else

NOP

endif;

outL l1

program;

inH h1;

inL l1;

if h1 == l1 then

while true do

if true then

NOP

else

outL l1

endif

done

else

NOP

endif;

outL l1

(ب) (الف)

شکل 13 - (الف) نمونه برنامه به زبان WL (ب) برنامه بازنویسی‌شده الف برای حالت غیرحساس به پیشرفت، که حالت حساس به پیشرفت را برآورده نمی‌کند

بنابراین، برای اِعمال حالت حساس به پیشرفت، باید مطمئن شد که نحوه پیشرفت برنامه نیز هیچ اطلاعات سطح بالایی را افشا نمی‌کند. پس برنامه باید با شروع از حالت‌های آغازینِ معادل از نظر مشاهده‌گرِ سطح پایین، یا همواره خاتمه یابد یا همواره واگرا باشد. گرچه ابزارها و روش‌هایی برای دسته‌های خاصی از برنامه‌ها وجود دارد که بررسی شود که آیا یک برنامه خاتمه می‌یابد یا خیر، اما این مسئله در حالت کلی تصمیم‌ناپذیر است. شاید به همین خاطر است که راه‌حل‌های ارائه شده برای حالت حساس به پیشرفت بسیار محافظه‌کارانه است و هر برنامه‌ای که یک حلقه وابسته به مقدار سطح بالا باشد، پذیرفته نمی‌شود. منظور از حلقه وابسته به مقدار سطح بالا، یعنی حلقه‌ای که اجرای بدنه‌ آن یا تعداد تکرارهای اجرای آن به یک مقدار سطح بالا وابسته باشد. به عنوان نمونه‌ای از تلاش‌های صورت گرفته در این زمینه، مور و همکارانش [18] یک نوع‌سامانه به همراه یک مکانیزم زمان‌اجرا به نام پیش‌گویی خاتمه[[83]](#footnote-83) ارائه کرده‌اند تا حلقه‌هایی که وضعیت پیشرفت آن‌ها فقط به مقادیر سطح پایین وابسته است، تشخیص داده شوند. گرچه چنین مکانیزمی در مقایسه با راه‌حل‌های ایستا از دقت بالاتری برخوردار است، اما هزینه سربار اضافه زمان‌اجرا را در پی خواهد داشت. از طرفی، اگر مکانیزم پیش‌گویی نتواند وضعیت پیشرفت حلقه را پیش‌بینی کند، برنامه در هنگام اجرا، گیر خواهد کرد.

بازنویس مورد استفاده در این پروژه، برنامه‌ها را به نحوی تغییر می‌دهد که وضعیت پیشرفت برنامه بازنویسی‌شده به مقادیر سطح بالا وابستگی نداشته باشد. باید توجه داشت که به دلیل اهمیت صحتِ الگوریتم بازنویسی، در برنامه‌ای که ممکن است هنگام اجرا مقادیر سطح بالا از طریق وضعیت پیشرفت برنامه نشت پیدا کنند، معناشناخت برنامه دچار تغییراتی شود. همان‌طور که در فصل قبل مطرح شد، در زبان برنامه‌نویسی WL، عنصر while تنها ساختاری است که می‌تواند باعث واگرایی برنامه‌ها شود. پس ابزار یا تابعی برای تحلیل حلقه‌های while نیاز خواهیم داشت. در الگوریتم بازنویسی مورد استفاده چنین فرض می‌شود که یک تحلیل‌گر حلقه[[84]](#footnote-84) وجود دارد که می‌تواند به طور ایستا با گرفتن کد حلقه، آن را تحلیل و ارزیابی کند. این الگوریتم این تضمین را می‌دهد که برنامه بازنویسی‌شده برای حالت‌های آغازینِ معادل از نظر مشاهده‌گر سطح پایین، یا همواره خاتمه می‌یابد یا همواره واگرا می‌شود [5] که البته با این فرض که یک تحلیل‌گر حلقه مناسب و کارا وجود دارد.

تحلیل‌گر حلقه مورد نظر در این الگوریتم چنین در نظر گرفته می‌شود که کد یک حلقه را می‌گیرد و یک عبارت منطقی بولی را به عنوان نتیجه تحلیل برمی‌گرداند. این عبارت منطقی برای حالت‌هایی درست خواهد بود که اجرای آن حلقه قطعاً خاتمه می‌یابد. بدین معنا که اگر حلقه همواره خاتمه ‌یابد، تابع تحلیل‌گر حلقه عبارت همواره درست یا True را برمی‌گرداند و عبارت همواره نادرست یا False را برمی‌گرداند، اگر حلقه در همه حالت‌ها واگرا باشند. به عنوان مثال، این تابع تحلیل‌گر، برای حلقه موجود در برنامه شکل 14، عبارت h1 >= l1 or l1 < 0 را به عنوان نتیجه تحلیل برمی‌گرداند.

program;

inH h1;

inL l1;

while h1 < l1 do

NOP;

h1 = h1 - l1

done;

outL l1

شکل 14 – برنامه‌ای که حلقه موجود در آن در حالتی که h1 >= l1 or l1 < 0 باشد، خاتمه خواهد یافت

الگوریتم بازنویسی مورد استفاده وابستگی زیادی به وجود یک تحلیل‌گر حلقه قدرتمند دارد، ابزاری که بتواند بسیاری از حلقه‌ها را با موفقیت تحلیل کند. تولید چنین ابزاری کار دشواری است که در فصل آینده، به نحوه پیاده‌سازی آن در این پروژه خواهیم پرداخت. با این حال و با وجود تلاش‌هایی که در این زمینه صورت گرفته است، باز هم ممکن است حلقه‌هایی در برنامه‌ها وجود داشته باشند که تحلیل‌گر مورد استفاده، از تحلیل آن‌ها عاجز باشد. در این‌جا فرض می‌شود که چنین برنامه‌هایی برای ورودی الگوریتم بازنویسی در نظر گرفته نمی‌شود.

پس با توضیحات فوق، بازنویس مورد استفاده در این پروژه، مسیرهای روی گراف وابستگی برنامه با شروع از متغیرهای سطح بالا تا عبارت شرطی حلقه‌ها[[85]](#footnote-85) را می‌پیماید و با استفاده از نتیجه تحلیل‌گر حلقه، به بازنویسی کد مبدأ برنامه می‌پردازد. به این صورت که اگر نتیجه تحلیل‌گر حلقه برای یک حلقه داده شده، عبارت همواره درست باشد، حلقه را بدون تغییر رها می‌کند و مشابه این رفتار را برای یک حلقه همواره واگرا نیز خواهد داشت، به شرطی که هیچ مسیری از نوع وابستگی کنترلی از مقادیر سطح بالا به عبارت شرطی حلقه وجود نداشته باشد. در واقع، حلقه‌ای که همواره واگراست، ممکن است باعث افشای اطلاعات سطح بالا شود، اگر آن حلقه توسط یک عبارت کنترل شود که به ورودی‌های سطح بالا وابسته است. اگر چنین باشد، آن حلقه با یک ساختار if-then با همان عبارت شرطی حلقه و همان بدنه حلقه جایگزین می‌شود. در شرایطی که تحلیل‌گر حلقه عبارتی غیر از همواره درست یا همواره نادرست را برگرداند، بازنویس اجرای آن حلقه را به همان عبارتِ برگردانده شده مشروط می‌کند. پس به این ترتیب، کد برنامه جدید قطعاً خاتمه می‌یابد.

الگوریتم بازنویسی مطرح‌شده در شکل 15، کد مبدأ برنامه M و گراف وابستگی برنامه متناظر آن را می‌گیرد و کد Mˊ بازنویسی‌شده را برمی‌گرداند که خط مشی عدم تداخل را در حالت حساس به پیشرفت برآورده می‌کند. منظور از تابع LoopAnalyzer همان ابزار تحلیل‌گر حلقه‌ای است که در بالا توضیح داده شده بود. این بازنویس، ابتدا بازنویس مربوط به حالت غیرحساس به پیشرفت را فراخوانی می‌کند. نتیجه این کار، برنامه‌ای خواهد بود که اگر M، حلقه‌های وابسته به مقادیر سطح بالا نداشته باشد، در حالت حساس به پیشرفت نیز پذیرفته می‌شود. در غیر این صورت، حلقه‌هایی که در مسیرهای به شکل هستند، که در آن h یک ورودی سطح بالا و یک مسیر منتهی به یک عبارت شرطی حلقه است، ممکن است بازنویسی شوند. البته ممکن است این‌گونه مسیرها حاوی گره‌های میانی باشند که خود بیانگر عبارت‌ شرطی حلقه‌های دیگری هستند. توابع *guard*(n)، *body*(n) و *loop*(n) به ترتیب مقدار عبارت شرطی حلقه، بدنه حلقه و کل حلقه موجود در Mˊ متناظر با گره n در گراف وابستگی برنامه را برمی‌گردانند.

RWPSNI(M, G):

Initialize *D* to the set of all paths in *G* where is a path terminating at a loop guard and *P* is the node representing a high input;

*Mˊ* = *RWPINI*(M, G);

if then

return Mˊ;

end

*H* = max {*height*(n) | n is a node on G}, where *height* is a function that returns the height of a given node on the tree obtained by removing data dependence edges from G;

Change Mˊ as follows:

**for** *h* = *H* to 1 **do**

**foreach** node n with *height(n)* = *h* representing a loop on some path **do**

r = LoopAnalyzer(loop(n));

**if** r = False **then**

**if** appears on at least one path **do**

replace *loop*(n) with the statement “if *guard*(n) then *body*(n) endif”;

end

else

if r ≠ True then

replace *loop*(n) with the statement “if r then *loop*(n) endif”;

end

end

end

h = h – 1;

end

return Mˊ;

شکل 15 - الگوریتم بازنویسی عدم تداخل حالت حساس به پیشرفت که برنامه M و گراف وابستگی برنامه مربوط به آنِ G را می‌گیرد [5]

همان‌طور که مشاهده می‌شود، بازنویس مطرح شده برای حالت حساس به پیشرفت ممکن است یک حلقه وابسته به مقادیر سطح بالا را به یک گزاره شرطی با همان بدنه حلقه جایگزین کند که این باعث می‌شود تا بدنه حلقه تنها یک بار در برنامه بازنویسی‌شده اجرا شود. گرچه راه‌کارهای دیگری مثل تغییر عبارت شرطی حلقه برای این‌که فقط به مقدار متناهی حلقه اجرا شود نیز وجود دارد، اما باید آن راهبردها را از نظر شفافیت با روش مورد استفاده در این‌جا بررسی کرد. ضمناً باید دقت داشت که ابتدا باید حلقه‌های تودرتو[[86]](#footnote-86) و سپس حلقه بیرونی تحلیل ‌شوند. زیرا تأثیر رفتار نسخه بعد از بازنویسی آن‌ها ممکن است با نسخه قبل از بازنویسی متفاوت باشد. به همین منظور، الگوریتم مطرح‌شده ارتفاع گره‌های بیانگرِ حلقه در درختی که با حذف یال‌های وابستگی داده‌ای از گراف وابستگی برنامه به دست‌ آمده است را ملاک عمل قرار می‌دهد. در شکل 16، نمونه کد برنامه تبدیل شده توسط این الگوریتم برای کد مبدأ برنامه شکل 14 را مشاهده می‌شود.

اثبات صحت و شفافیت الگوریتم‌های استفاده‌شده در این پروژه، در مقاله [5] قابل مشاهده است.

شکل 16 - کد مبدأ بازنویسی‌شده توسط الگوریتم حالت حساس به پیشرفت برای برنامه شکل 14

program;

inH h1;

inL l1;

if l1 <= h1 or l1 < 0 then

while h1 < l1 do

NOP;

h1 = h1 - l1

done

endif;

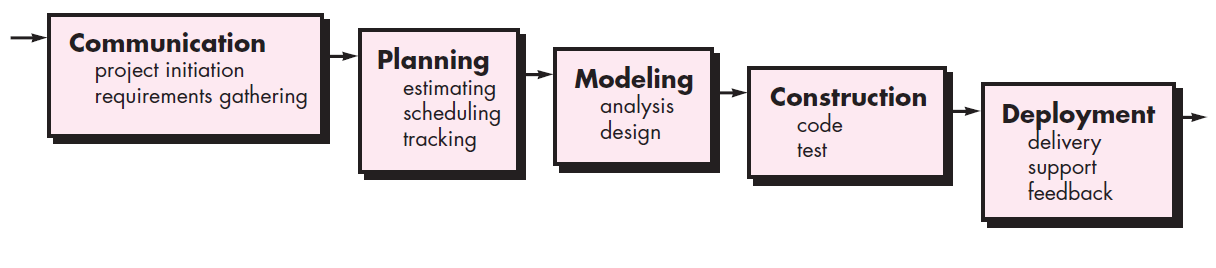
outL l1

# فصل پنجم پیاده‌سازی و ایجاد رابط کاربری

**پیاده‌سازی و ایجاد رابط کاربری**

## تحلیل و طراحی نرم‌افزار

با توجه به مشخص و ثابت بودن نیازهای این نرم­افزار در همان ابتدای تعریف پروژه، می‌توان از مدل فرآیندی آبشاری[[87]](#footnote-87) یا چرخه حیات کلاسیک[[88]](#footnote-88) استفاده کرد. هم‌چنین، تحلیل و طراحی این نرم‌افزار با رویکرد شی‌گرایی[[89]](#footnote-89) انجام شده است.

این مدل فرآیندی شامل پنج مرحله ارتباط[[90]](#footnote-90)، برنامه‌ریزی[[91]](#footnote-91)، مدل‌سازی[[92]](#footnote-92)، ساخت[[93]](#footnote-93) و استقرار[[94]](#footnote-94) است.

شکل 17 - مدل فرآیندی آبشاری [19]

دلایل انتخاب این مدل فرآیندی، علاوه بر ثابت و مشخص بودن نیازهای پروژه در ابتدای امر عبارتند از:

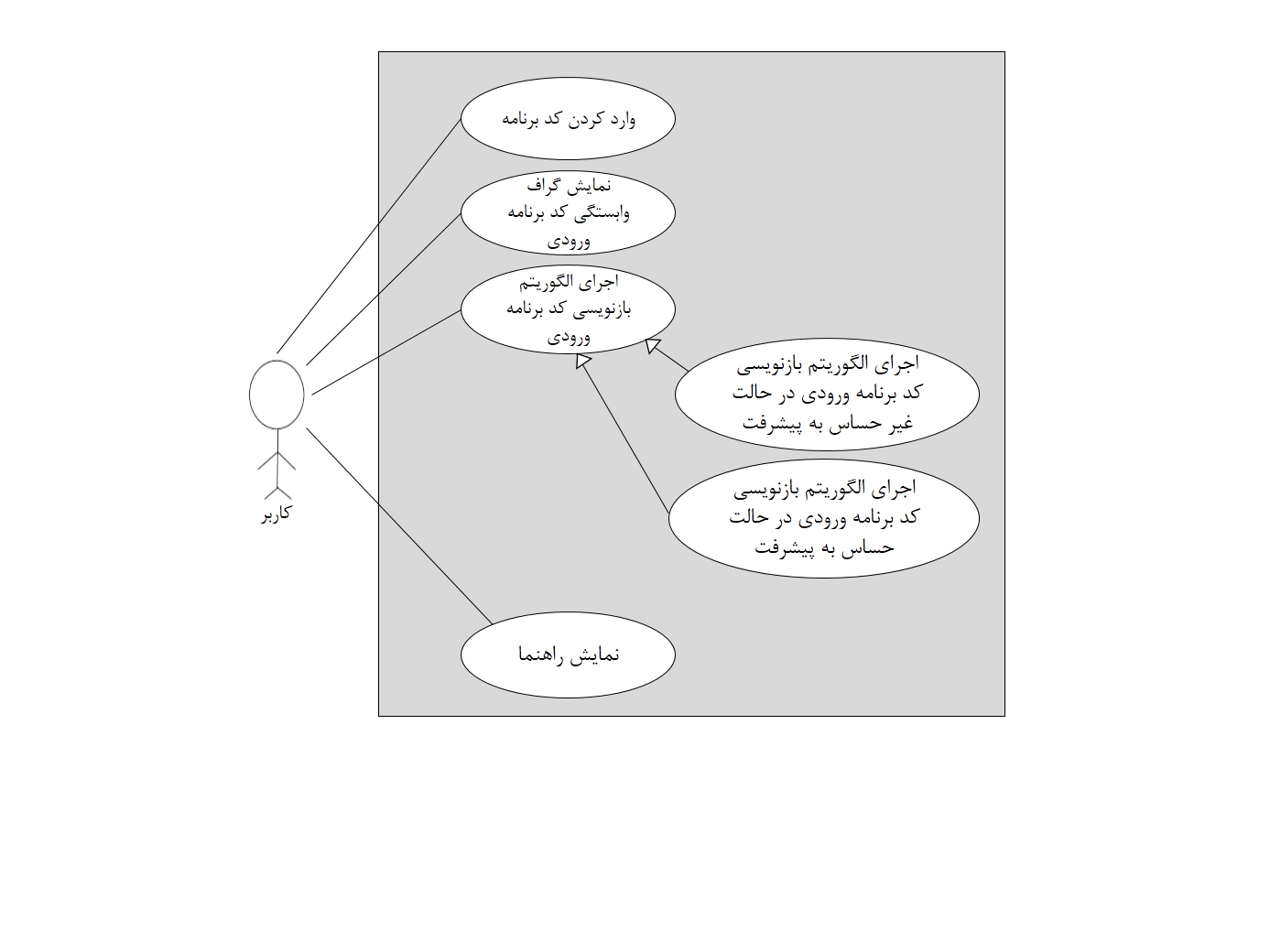
* فهم این مدل نسبت به مدل‌های فرآیندی دیگر ساده‌تر است.
* از حیث تولید مستندات، شرایط بهتر و آسان‌تری دارد.
* مراحل به سادگی قابل بررسی و کنترل هستند.
* هر مرحله، تحویل‌دادنی‌ها و روش مرور مشخصی دارند.
* مراحل هم‌پوشانی ندارند و با پیمایش یکباره مدل فرآیندی، نرم‌افزار به طور کامل تولید شده است.
* تعداد نیازهای پروژه متناسب با این مدل فرآیندی می‌باشد.

مدل فرآیندی آبشاری یک روش خطیِ ترتیبی و روش‌مند برای توسعه نرم‌افزار محسوب می‌شود که با مشخص کردن نیازمندی‌ها آغاز می‌شود و با گذر از مراحل برنامه‌ریزی، مدل‌سازی، ساخت و استقرار به پایان می‌رسد. این مدل فرآیندی زمانی به کار بسته می‌شود که نیازمندی‌های پروژه، همانند این پروژه، کاملاً خوش‌تعریف، پایدار، مشخص و بدون ابهام باشند. به همین دلیل، میزان ریسک انجام این مدل فرآیندی قابل قبول خواهد بود.

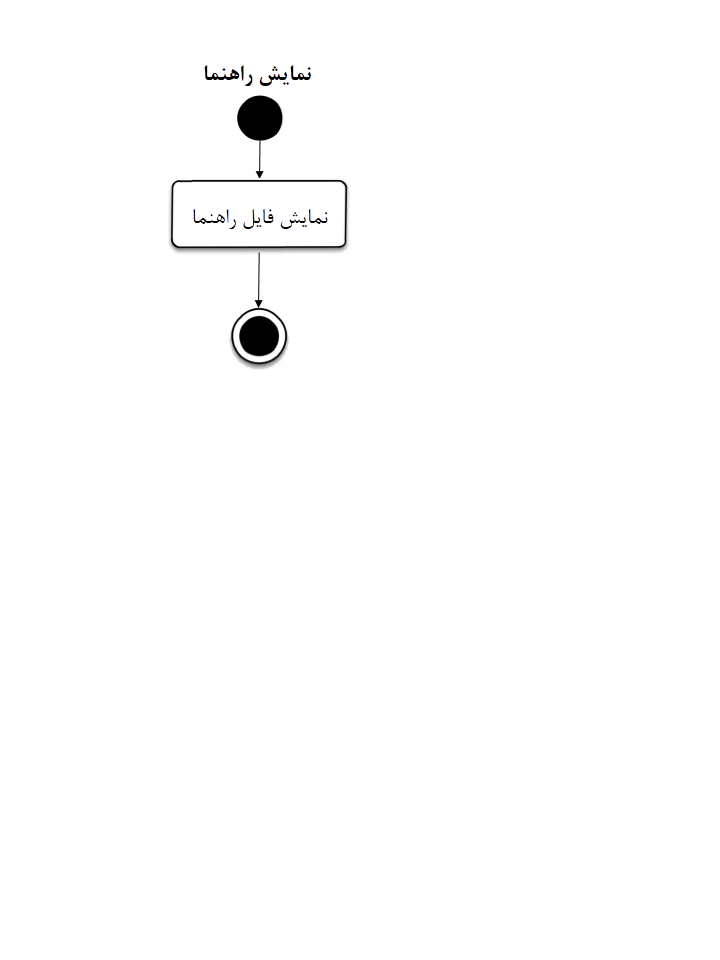
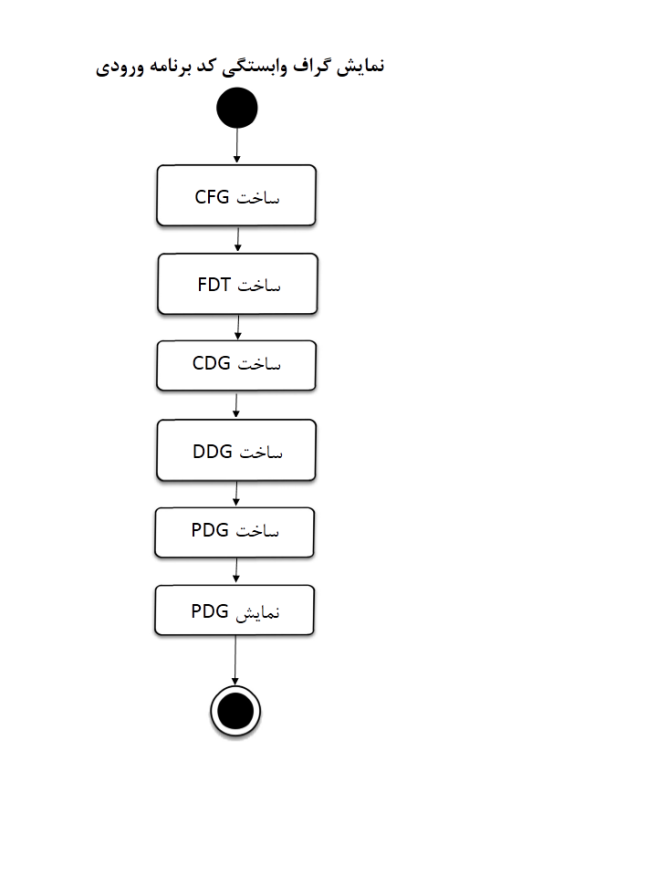
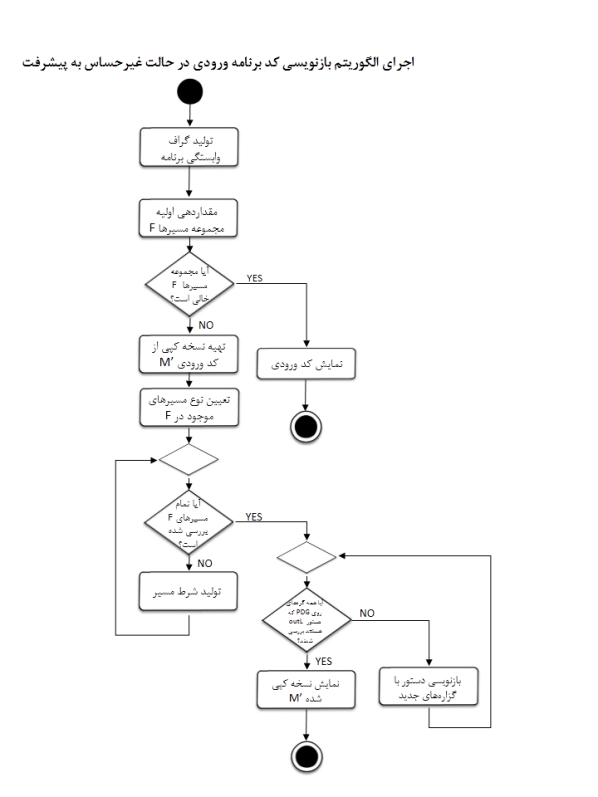
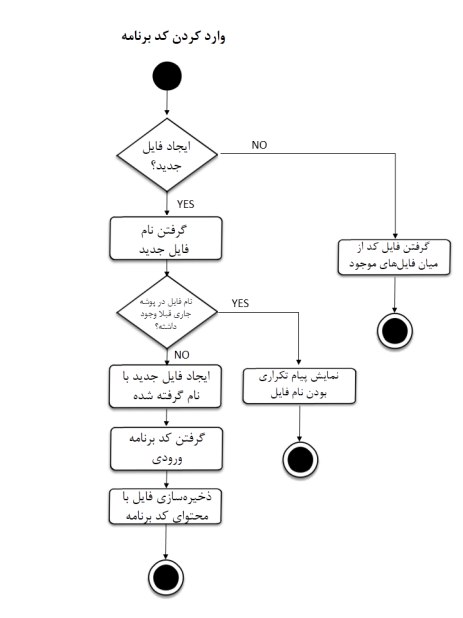
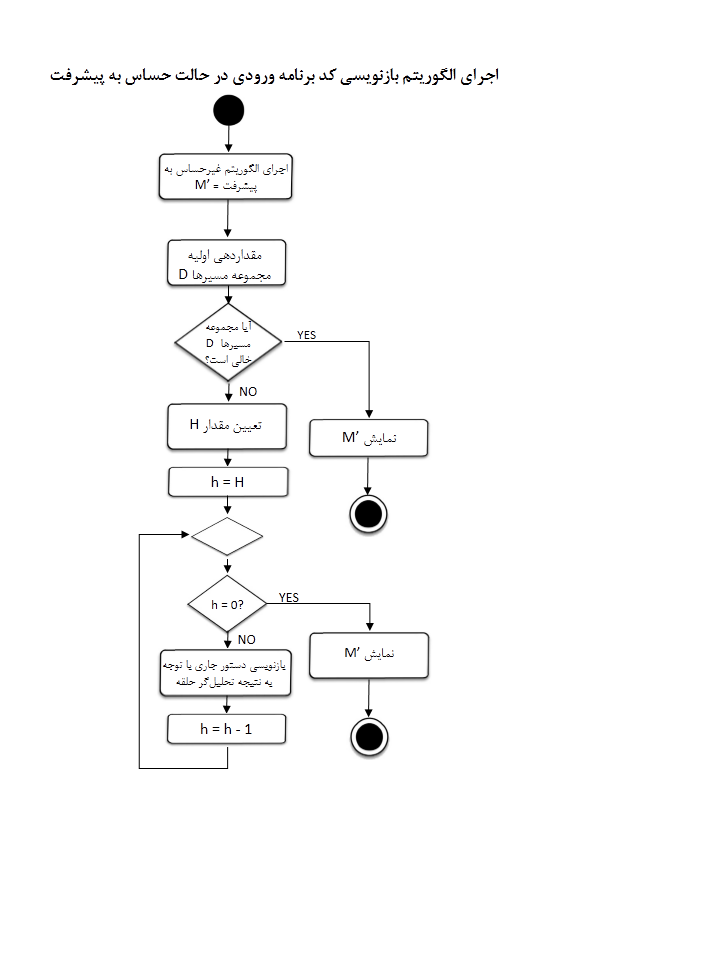
گام اول به تعریف و جمع‌آوری نیازمندی‌های پروژه اختصاص پیدا می‌کند که در این پروژه، نیازسنجی، برقراری ارتباط برای درک نیازها و مطالعه مستندات لازم انجام شده است.

پس از درک کامل نیازها، گام برنامه‌ریزی انجام می‌شود. در این مرحله، تخمین‌ها و برنامه‌ریزی‌های زمانی برآورد می‌شود. این تخمین‌ها و برنامه‌ریزی‌ها، شامل برآورد زمانی، هزینه‌، نیروی انسانی و سایر بخش‌هاست. برای این نرم‌افزار، برنامه‌ریزی زمانی اولیه‌ای تخمین زده شد و قسمت‌های مختلف پروژه به بخش‌هایی برای تحلیل، طراحی و پیاده‌سازی تقسیم‌بندی شد.

قدم بعدی، مدل‌سازی یا همان تحلیل و طراحی نرم‌افزار خواهد بود. در این قسمت، با توجه به نیازمندی‌های پروژه، تحلیل‌های مربوط صورت می‌گیرد و مستندات و نمودارهای تحلیل و طراحی تولید می‌شوند. مهم‌ترین نمودار در مرحله تحلیل، نمودار موردکاربرد[[95]](#footnote-95) است که با توجه به نیازمندی‌ها و موردکاربردهای به دست آمده از تعریف پروژه ترسیم می‌شود. این نمودار مبنای تحلیل‌های بعدی خواهد بود. در ادامه نمودارهای موردکاربرد و فعالیت[[96]](#footnote-96) به عنوان بخشی از قسمت تحلیل آمده است. برای ترسیم این نمودارها، از زبان UML[[97]](#footnote-97) استفاده شده است.

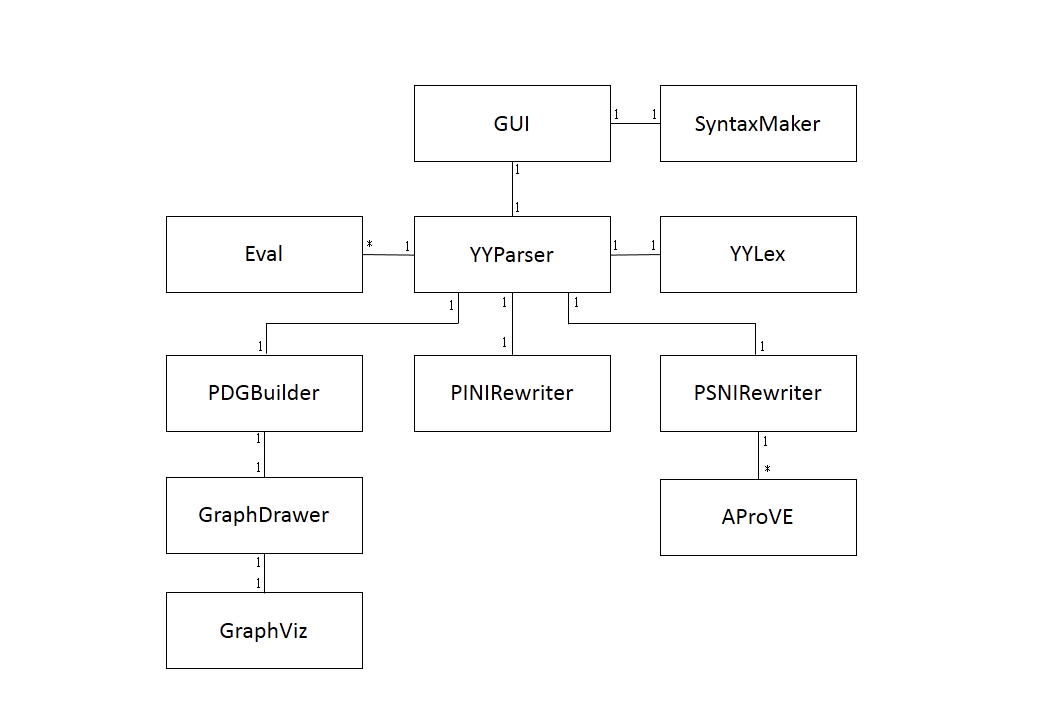


شکل 18 - نمودار موردکاربرد نرم‌افزار پروژه



شکل 19 - نمودارهای فعالیت نرم‌افزار پروژه

در مرحله طراحی، مهم‌ترین نمودار که آینه تمام‌نمای معماری نرم‌افزار نیز به شمار می‌رود، نمودار کلاس[[98]](#footnote-98) است. این نمودار، کلاس‌های مورد استفاده در نرم‌افزار و نحوه ارتباط بین آن‌ها را مشخص می‌کند. در واقع این نمودار، مرز بین تحلیل و طراحی است و از این نمودار به عنوان مبنای نمودارها و طراحی‌های نرم‌افزار می‌توان نام برد. در شکل 20، نمودارِ کلاس نرم‌افزار این پروژه را مشاهده می‌کنید. در این شکل، جزئیات فیلدها و متدهای هر کلاس‌ آورده نشده و تنها به نام کلاس‌ها و ارتباط بین آن‌ها اکتفا شده است.



شکل 20 - نمودار کلاس نرم‌افزار پروژه (بدون ذکر فیلدها و متدها)

پس از این مراحل، گام بعدی پیاده‌سازی و آزمون نرم‌افزار خواهد بود که در قسمت‌های بعدی به تفصیل به آن‌ها پرداخته می‌شود. پیش از ورود به مرحله ساخت، بخش‌های مختلف پیاده‌سازی پروژه به فازهایی دسته‌بندی شدند. این فازبندی به تقسیم‌شدن مسئله پیاده‌سازی به مسائل کوچک‌تر کمک بسیاری می‌کند. از طرف دیگر، بعد از پیاده‌سازی کامل هر فاز، نرم‌افزار آزمون می‌شود. با این روش، پس از بررسی موارد آزمون برای هر فاز، اشکال‌زدایی صورت می‌گیرد و سپس با کسب اطمینان از صحت برنامه تا آن فاز، پیاده‌سازی فاز بعدی آغاز می‌شود. فازهای تعیین‌شده برای پیاده‌سازی این ابزار بدین شرح می‌باشند:

- فاز اول: پیاده‌سازی تحلیل‌گر لغوی و نحوی زبان برنامه‌نویسی WL

- فاز دوم: پیاده‌سازی گراف‌های جریان کنترل، غلبه رو به جلو، وابستگی داده‌ای، وابستگی کنترلی و در نهایت گراف وابستگی برنامه

- فاز سوم: پیاده‌سازی بازنویس برنامه برای حالت غیرحساس به پیشرفت

- فاز چهارم: پیاده‌سازی تحلیل‌گر حلقه و بازنویس برنامه برای حالت حساس به پیشرفت

شایان ذکر است که طبق اصول مهندسی نرم‌افزار، در هر یک از فازهای فوق، ابتدا به جستجوی ابزارهای مشابه یا کمکیِ موجود پرداخته می‌شد و با بهره‌گیری از آن‌ها، پیاده‌سازی صورت می‌گرفت. ضمناً گزارش مراحل انجام هر یک از فازهای فوق در پوشه پروژه[[99]](#footnote-99) موجود می‌باشد.

## شرح کلی مراحل پیاده‌سازی و ابزارهای مورد استفاده

در این مرحله، با توجه به تحلیل و طراحی انجام شده، نرم‌افزار پروژه پیاده‌سازی می‌شود. برای این منظور، ابتدا پس از رفع ابهام و بازنویسی گرامر زبان WL مطرح‌شده در مقاله اصلی پروژه، با استفاده از ابزارهای jflex و bison، دو بخش اصلی کامپایلر این زبان؛ یعنی تحلیل‌گر لغوی[[100]](#footnote-100) و تحلیل‌گر نحوی[[101]](#footnote-101)، فراهم شدند. در تحلیل‌گر لغوی، تمامی کلمات کلیدی و عناصر مختلف زبان به صورت نماد[[102]](#footnote-102)هایی در نظر گرفته شدند. سپس این نمادها، به تحلیل‌گر نحوی داده می‌شود و با توجه به قواعد مختلف زبان، رفتار مربوط به هر قاعده در ذیل آن نوشته می‌شود. به این ترتیب اجزای زبان و قواعد گرامر آن پیاده‌سازی می‌شود. در این مرحله، خطاهای نحوی[[103]](#footnote-103) تشخیص داده می‌شود و در صورت بروز آن‌ها، به کاربر گزارش داده می‌شود. البته لازم به یادآوری است که بنا به نیازهای پیاده‌سازی، قسمت‌هایی از تحلیل‌گر نحوی پس از تولید توسط ابزار bison، به صورت دستی تغییر پیدا کرده است، که این امر نیازمند تسلط کافی به جزئیات این کلاس است. در صورت نیاز، نحوه اجرای کدها و تولید آن‌ها توسط ابزارهای ذکر شده، در فایلی به نام README-GuidToRun.txt در پوشه پروژه آمده است. در کد نوشته‌شده برای تحلیل‌گر نحوی، در زمان تشکیل درخت تجزیه و بررسی برنامه داده شده به آن، به طور همزمان گراف جریان کنترل برنامه نیز تولید می‌شود. گراف‌های مورد استفاده در این پروژه، همگی از نوع لیست پیوندی[[104]](#footnote-104) می‌باشند. دلیل استفاده از این ساختمان داده، سهولت در پیمایش، عدم نیاز به دسترسی تصادفی و رعایت حفظ ترتیب گره‌های فرزند و پدر است. در هر گره، اطلاعات مورد نیاز نظیر شماره گره، گزاره، اشاره‌گرهای بعدی و قبلی در گراف‌ها، ارتفاع گره، متغیرهای موجود در گزاره و غیره ذخیره می‌شود.

تا این‌جا، کد برنامه داده شده به برنامه از نظر نحوی بررسی و گراف جریان کنترل ساخته شده است. اکنون با توجه به نوع درخواست کاربر؛ یعنی تولید گراف وابستگی برنامه، بازنویسی در حالت غیرحساس به پیشرفت یا در حالت حساس به پیشرفت، الگوریتم مربوط به هر کدام اجرا می‌شود.

برای نمایش گرافیکیِ گراف وابستگی برنامه، از ابزار قدرت‌مند GraphViz [13] استفاده شده است. به این شکل که کد مربوط به این ابزار به پروژه اضافه شده است و با انجام تنظیمات اولیه، با تولید گراف به زبان dot که توسط این ابزار شناخته شده است، گراف مورد نظر در قالب یک تصویر با فرمت png تولید می‌شود. در هنگام تولید گراف وابستگی برنامه، علاوه بر نمایش گرافیکی آن، گراف‌های وابستگی کنترلی و داده‌ای نیز به طور مجزا ذخیره می‌شوند.

اما علاوه بر ابزارهای ذکر شده در بالا، برای تابع تحلیل‌گر حلقه که در الگوریتم بازنویسی حالت حساس به پیشرفت نقش تأثیرگذاری را ایفا می‌کند، ابزارهای مختلفی بررسی شد. البته هیچ‌کدام از ابزارهای بررسی‌شده، تحلیل کامل مورد نیاز ما برای این تابع را ارائه نکردند و یافتن ابزار مناسب و نزدیک به خواسته ما، کار ساده‌ای نبود. برای این قسمت، مقالات مختلفی مطالعه شد و ابزارهای گوناگونی نظیر DRR، T2، Cooperating T2، Polyrank، Frama-C، Clang، Kittle، rankFinder، PAG، LoopFrog و AProVE [14] نصب و بررسی شدند. در پایان این مرحله و با در نظر گرفتن معیارهای دقت و سرعت بالاتر، راحتی استفاده، نوع چاپ خروجی و میزان شباهت در تحلیل مورد نیاز این نرم‌افزار، از ابزار AProVE استفاده شده است. از آن‌جایی که این ابزار برای زبان C مورد استفاده قرار می‌گیرد، کد حلقه‌ای که به عنوان ورودی به تابع تحلیل‌گر حلقه داده می‌شود، به زبان C تبدیل می‌شود و سپس، برنامه تبدیل شده به زبان C به عنوان ورودی به ابزار تحلیل حلقه AProVE داده می‌شود. این ابزار با توجه به کد ورودی، یکی از سه جواب ممکن Proven، Disproven و Maybe را به پرسش درباره خاتمه آن برنامه می‌دهد. Proven به معنای اثبات خاتمه برنامه و Disproven به معنای اثبات عدم خاتمه برنامه تحت هر شرایطی است. این در حالیست که نتیجه تحلیل Maybe به معنای ناتوانی این ابزار در تحلیل برنامه داده شده تلقی می‌شود. لذاست که نتیجه تحلیل Proven معادل با عبارت همواره درست در تابع تحلیل‌گر حلقه خواهد بود، اما در صورتی‌که پاسخ یکی از حالت‌های Disproven یا Maybe باشد، بهتر است با اجرای الگوریتم‌هایی سعی در تحلیل حلقه داشته باشیم.

در پیاده‌سازی رابط کاربری گرافیکی، ظاهر برنامه متناسب با سیستم‌عامل کاربر است. برای ویرایش‌گر کد مبدأ، به جای استفاده از ویرایش‌گرهای ساده، با تعریف ساختار نحوی زبان و تنظیمات اولیه، از کتابخانه RSyntaxTextArea [15] استفاده شده است که تجربه یک محیط کاربرپسند و حرفه‌ای را در اختیار کاربر می‌گذارد. پیاده‌سازی رابط کاربری در بخش بعدی به طور مفصل شرح داده می‌شود.

شایان ذکر است که به طور کلی، کدهای نوشته شده به زبان WL به زبان‌های سطح بالا و رایج‌تری مثل C قابل تبدیل است. در این نرم‌افزار نیز می‌توان کد ورودی و کدهای بازنویسی‌شده در هر حالت را در قالب برنامه‌هایِ به زبان C نیز مشاهده کرد. برای این کار، به ازای هر قاعده در زبان WL، گزاره متناظر با آن قاعده در زبان C برگردانده می‌شود تا اینکه با اتمام تحلیل نحوی برنامه، کد مبدأ متناظر با آن، در فایل جداگانه‌ای ذخیره می‌شود.

توضیحات جزئیات پیاده‌سازی کلاس‌های مختلف نرم‌افزار و گزارش روند انجام کار، در فایل‌های جداگانه‌ای در پوشه پروژه موجود است. به عنوان نمونه، بخشی از کدهای نوشته‌شده برای پیاده‌سازی در پیوست ارائه شده است.

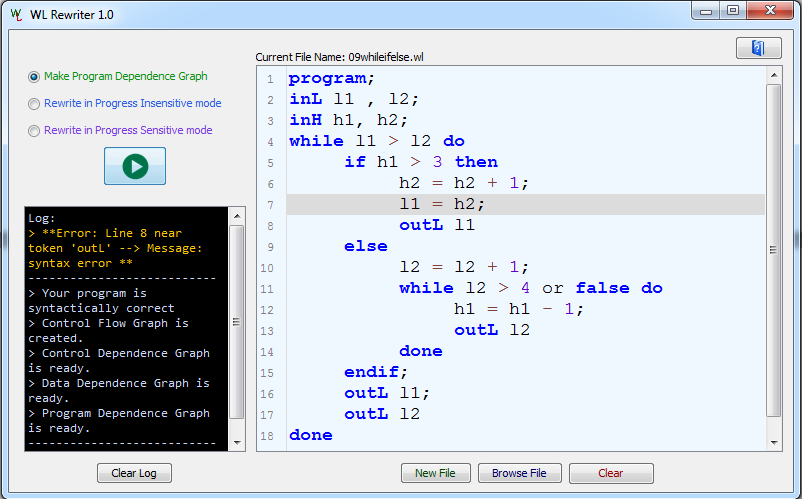
## ایجاد رابط کاربری گرافیکی[[105]](#footnote-105)

اهمیت ظاهر برنامه و صفحاتی که کاربر توسط آن‌ها با سیستم در تعامل است، بر کسی پوشیده نیست. این اهمیت درباره نرم‌افزارهای مورد استفاده توسط کاربران حرفه‌ای رایانه یا همان برنامه‌نویسان که کاربران اصلی این نرم‌افزار هستند، دوچندان می‌شود. چرا که طراح باید پیچیدگی‌ها را در رابط کاربری به حداقل برساند، به نحوی که قابلیت‌های برنامه کاهش نیابد. از این رو، برای طراحی رابط کاربری گرافیکی این برنامه زمان زیادی صرف شده است. ابتدا با مشورت از یکی از اساتید رشته هنر و زیبایی‌شناسی، طراحی گرافیکی کلی صفحه برنامه انجام شد. پس از آن، طرح‌های مختلفی ارائه شد و به عنوان آزمایش، در اختیار تعدادی از برنامه‌نویسان قرار گرفت تا بازخورد آن‌ها نسبت به رابط کاربری این برنامه سنجیده شود. در پایان، با انجام اصلاحات، رابط کاربری گرافیکی برنامه نهایی شد.

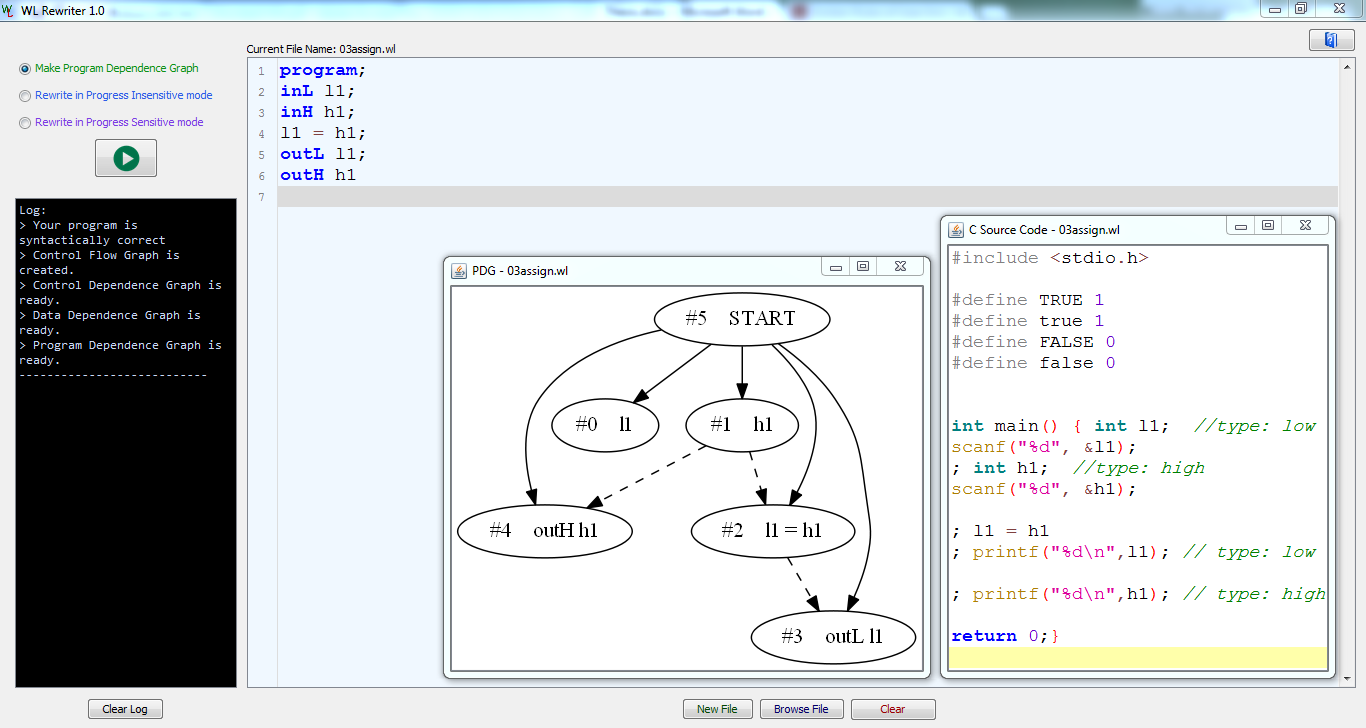
نکات زیر برای طراحی رابط کاربری این برنامه مورد استفاده قرار گرفته شده است:

* گزینه‌ها و دکمه‌های موجود در صفحه باید همگون و با سبک یکسان باشند.
* در هنگام تغییر وضعیت برنامه، باید ظاهر نیز متناسب با آن تغییر یابد. یعنی برنامه متناسب با هر فعالیت، بازخورد مناسبی داشته باشد.
* هر گزینه باید کاملاً واضح و دارای معنای خاص باشد.
* برای همگی فعالیت‌ها، حالت‌های پیش‌فرض در نظر گرفته شود.
* کاربر نیازی به آموزش برای یادگیری کار با رابط کاربری نداشته باشد یا حداقل باشد.
* اجزائی که با یکدیگر مرتبط هستند، در یک گروه‌بندی خاص باشند.
* از رنگ‌ها و سبک‌ها به درستی و با توجه به گروه‌بندی‌ها و معانی رنگ‌ها در ذهن کاربر با توجه به سابقه قبلی آن‌ها استفاده شود.
* برای گزینه‌ها، از میان‌برها و یادمان[[106]](#footnote-106)‌ها استفاده شود.
* برای حذف یا پاک کردن اطلاعات مهم، تأیید مجدد کاربر دریافت شود.
* برای نمایش پیغام‌ها از رنگ‌های متناسب استفاده شود.
* به دلیل استفاده طولانی مدت کاربر از این نرم‌افزار، بهتر است از رنگ‌ها و چینشی استفاده شود که آلودگی بصری برای کاربر را به دنبال نداشته باشد.
* امکان تغییر ابعاد صفحه برای کاربر وجود داشته باشد و ضمناً با تغییر ابعاد پنجره برنامه، چینش اجزا در صفحه منظم باقی بماند.

در این پروژه سعی شده است تا موارد بالا تا حد امکان رعایت شوند و تجربه خوب و لذت‌بخشی را برای کاربر به ارمغان بیاورد. استفاده مناسب از رنگ‌ها، چینش اجزا در صفحه، ایجاد میان‌برهای کاربردی و ویرایشگر کد سفارشی‌شده با ساختار نحوی زبان WL از جمله فعالیت‌های انجام شده است.



شکل 21 - نمای کلی رابط کاربری گرافیکی نرم‌افزار



شکل 22 – نمونه‌ای از اجرای برنامه در رابط کاربری گرافیکی نرم‌افزار

## راستی‌آزمایی و آزمون

همان‌طور که قبلاً ذکر شد، روش مطرح شده و الگوریتم‌های بازنویسی از نظر صحت و شفافیت قابل اثبات هستند؛ اما راستی‌آزمایی و آزمونِ برنامه پیاده‌سازی شده نیز اهمیت دارد. برای این کار، با بهره‌گیری از آزمون دامنه[[107]](#footnote-107) تعدادی مورد آزمون[[108]](#footnote-108) برای بررسی صحت اجرای برنامه پیاده‌سازی شده طراحی شد. روش آزمون دامنه یکی از روش‌های پرکاربرد در آزمون نرم‌افزار به شمار می‌رود. در این روش، تعداد محدودی مورد آزمون که هر یک به عنوان نماینده‌ای از دسته موارد آزمون‌ مشابه هستند، به عنوان ورودی به نرم‌افزار داده می‌شود و خروجی حاصل از پردازش نرم‌افزار بر روی داده ورودی بررسی و راستی‌آزمایی می‌شود. در این پروژه نیز با همین روش، تعداد نزدیک به سی مورد آزمون بررسی شد که هر یک شامل ساختار متفاوتی از عناصر موجود در زبان WL می‌باشند. با توجه به این‌که در حوزه زبان‌های برنامه‌سازی، استقرا از مرسوم‌ترین روش‌های اثبات به شمار می‌رود، سعی شد تا با کمترین تعداد استفاده از عناصر زبان در هر برنامه، نرم‌افزار مورد آزمون و بررسی قرار گیرد و برنامه‌های مشابه یا دارای ساختار مشابه با برنامه‌های مورد آزمون، به استقرا آزمون‌شده بگیریم. از طرفی، در طراحی مواردآزمون سعی شد تا انواع مختلف جریان‌های صریح و ضمنی مدنظر در خط مشی عدم‌تداخل در هر دو حالت حساس و غیرحساس به پیشرفت مورد بررسی قرار بگیرد.

در ادامه نمونه‌هایی از موارد آزمون و برنامه‌های بازنویسی‌شده آن‌ها آورده شده است.

(پ) (ب) (الف)

program;

inL l1;

inH h1;

outL l1;

l1 = h1 + 2;

outH h1

program;

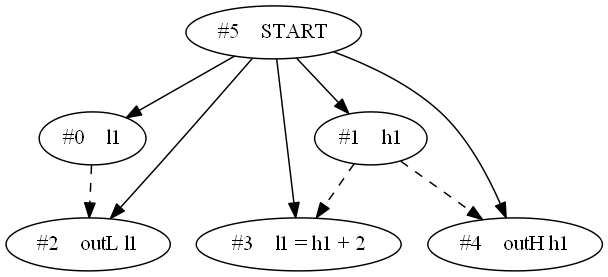
inL l1;

inH h1;

outL l1;

l1 = h1 + 2;

outH h1



شکل 23 – (الف) برنامه مورد آزمون با نام 02basic.wl (ب) گراف وابستگی برنامه مربوط به برنامه الف (پ) برنامه بازنویسی‌شده برای حالت غیرحساس به پیشرفت مربوط به برنامه الف

در شکل 23، مورد آزمونی مشاهده می‌شود که گرچه به صورت صریح مقدار سطح بالا در متغیر سطح پایین l1 قرار می‌گیرد، اما به دلیل این‌که دستور outL l1 بعد از آن دستور نیامده است، پس خط مشی را نقض نمی‌کند. به همین دلیل، برنامه بازنویسی‌شده مربوط به آن نیز تفاوتی با برنامه اولیه ندارد.

(پ) (ب) (الف)

program;

inL l1;

inH h1;

l1 = h1;

outL l1;

outH h1

program;

inL l1;

inH h1;

l1 = h1;

if TRUE then

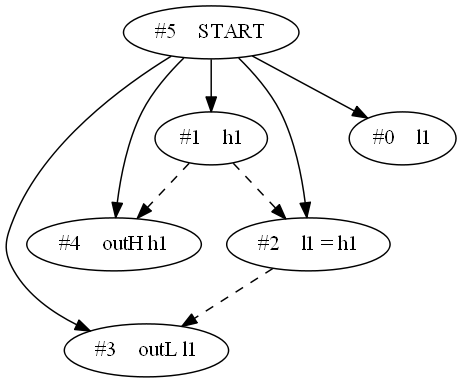
outL BOT

else

outL l1

endif;

outH h1



شکل 24- (الف) برنامه مورد آزمون با نام 03assign.wl (ب) گراف وابستگی برنامه مربوط به برنامه الف (پ) برنامه بازنویسی‌شده برای حالت غیرحساس به پیشرفت مربوط به برنامه الف

در شکل 24، به صورت صریح جریانی وجود دارد که عدم تداخل را نقض می‌کند. به همین دلیل و با توجه به این‌که شرطی برای وقوع مسیر وجود ندارد، پس هیچ‌گاه دستور outL l1 اجرا نخواهد شد.

(پ)

(ب) (الف)

program;

inH h1, h2;

inL l1, l2;

if l1 > 0 then

l1 = l1 - 1;

if l2 == 10 then

if h2 > 9 then

l1 = l1 + 2;

outL l1

else

l2 = 2

endif

else

l2 = h1

endif

endif;

l1 = 2;

outL l1;

outL l2

program;

inH h1, h2;

inL l1, l2;

if l1 > 0 then

l1 = l1 - 1;

if l2 == 10 then

if h2 > 9 then

l1 = l1 + 2;

if ( (l2 == 10) and (l1 > 0) and (h2 > 9) and (l2 == 10) and (l1 > 0) ) or ( (l2 == 10) and (l1 > 0) ) then

NOP

else

outL l1

endif

else

l2 = 2

endif

else

l2 = h1

endif

endif;

l1 = 2;

outL l1;

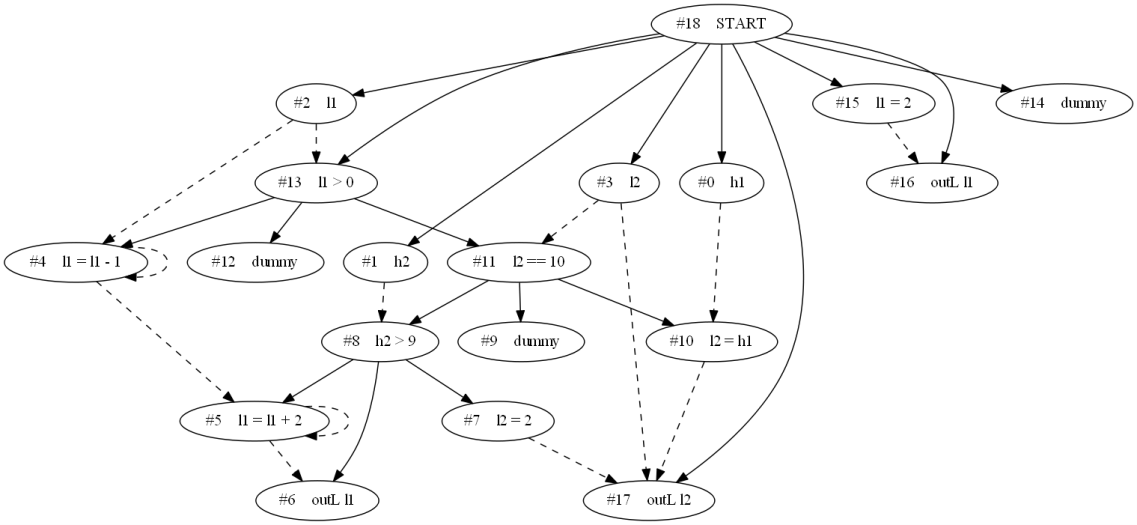
if ( (l2 == 10) and (l1 > 0) ) or ( !(l2 == 10) and (l1 > 0) ) then

NOP

else

outL l2

endif



شکل 25- (الف) برنامه مورد آزمون با نام 05if2.wl (ب) برنامه بازنویسی‌شده برای حالت غیرحساس به پیشرفت مربوط به برنامه الف (پ) گراف وابستگی برنامه مربوط به برنامه الف

در شکل 25، برنامه‌ مورد آزمونی است که برای بررسی دستورات if و else طراحی شده است. مسیرهای مختلفی در این برنامه وجود دارد که ممکن است باعث نقض عدم تداخل شود. در ادامه جدول خروجی‌های برنامه‌های به زبان C متناظر با برنامه‌های الف و ب برای مقادیر ورودی گوناگون آمده است که صحت برنامه بازنویسی‌شده را نشان می‌دهد.

جدول 1 – نمونه ورودی‌ها و خروجی‌ها برای برنامه به زبان C متناظر با برنامه شکل 25-(الف)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Violation** | **outL l2 (line# 19)** | **outL l1 (line# 18)** | **outL l1 (line# 9)** | **h2** | **h1** | **l2** | **l1** |
| **No** | 0 | 2 | - | 1 | 1 | 0 | 0 |
| 0 | 2 | - | 0 | 1 | 0 | 0 |
| **Yes** | 1 | 2 | - | 0 | 1 | 1 | 1 |
| 0 | 2 | - | 0 | 0 | 1 | 1 |
| **Yes** | 10 | 2 | 6 | 10 | 1 | 10 | 5 |
| 2 | 2 | - | 9 | 1 | 10 | 5 |

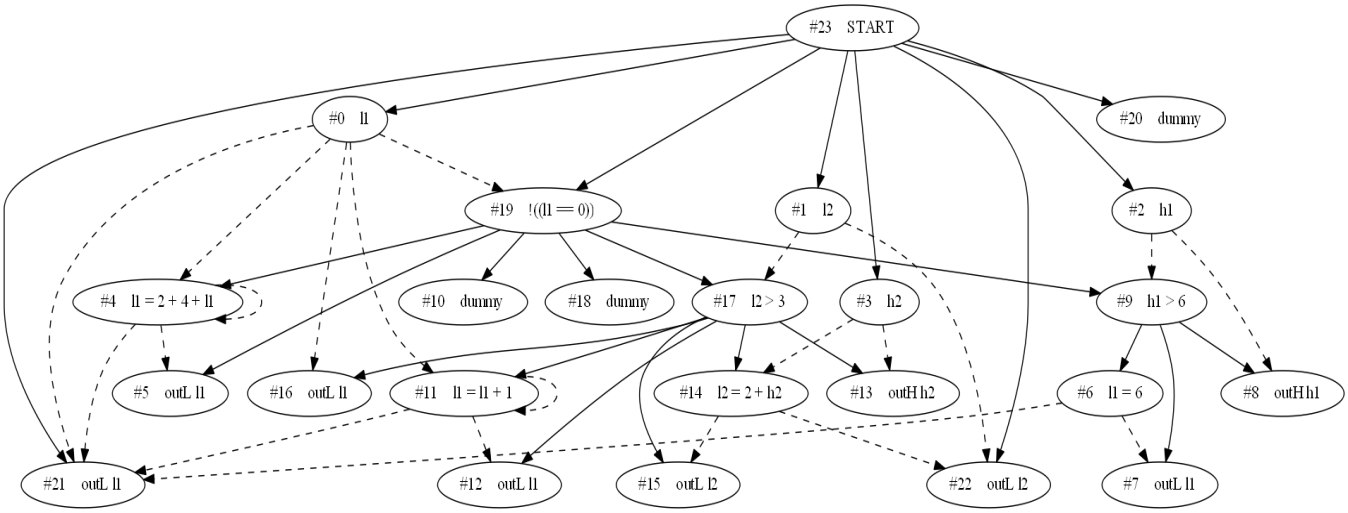
جدول 2 - نمونه ورودی‌ها و خروجی‌ها برای برنامه به زبان C متناظر با برنامه شکل 25-(ب)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Violation** | **outL l2 (line# 19)** | **outL l1 (line# 18)** | **outL l1 (line# 9)** | **h2** | **h1** | **l2** | **l1** |
| **No** | - | 2 | - | 1 | 1 | 0 | 0 |
| - | 2 | - | 0 | 1 | 0 | 0 |
| **No** | - | 2 | - | 0 | 1 | 1 | 1 |
| - | 2 | - | 0 | 0 | 1 | 1 |
| **No** | - | 2 | - | 10 | 1 | 10 | 5 |
| - | 2 | - | 9 | 1 | 10 | 5 |

همان‌طور که در جدول 2 مشاهده می‌شود، برنامه بازنویسی‌شده برخلاف برنامه اولیه، به ازای ورودی‌های مختلف سطح بالا تغییری نمی‌کند و موارد ناقض عدم تداخل اصلاح شده است.

(پ)

(ب) (الف)



program;

inL l1 , l2;

inH h1 , h2;

if !(l1 == 0) then

l1 = 2 + 4 + l1;

outL l1;

if h1 > 6 then

l1 = 6;

outL l1;

outH h1

endif

else

if l2 > 3 then

l1 = l1 + 1;

outL l1;

outH h2

else

l2 = 2 + h2;

outL l2;

outL l1

endif

endif;

outL l1;

outL l2

program;

inL l1, l2;

inH h1, h2;

if !((l1 == 0)) then

l1 = 2 + 4 + l1;

outL l1;

if h1 > 6 then

l1 = 6;

if ( (!((l1 == 0))) ) or ( (!((l1 == 0))) and (h1 > 6) and (!((l1 == 0))) ) then

NOP

else

outL l1

endif;

outH h1

endif

else

if l2 > 3 then

l1 = l1 + 1;

outL l1;

outH h2

else

l2 = 2 + h2;

if ( !(l2 > 3) and !(!((l1 == 0))) ) then

outL BOT

else

outL l2

endif;

outL l1

endif

endif ;

if ( (!((l1 == 0))) ) then

NOP

else

outL l1

endif;

if ( !(l2 > 3) and !(!((l1 == 0))) ) then

outL BOT

else

outL l2

endif

شکل 26- (الف) برنامه مورد آزمون با نام 07ifelseadvanced.wl (ب) برنامه بازنویسی‌شده برای حالت غیرحساس به پیشرفت مربوط به برنامه الف (پ) گراف وابستگی برنامه مربوط به برنامه الف

در شکل 26، مورد آزمون دیگری بررسی می‌شود که حالت پیشرفته‌تری برای ساختار if و else در کنار هم و تو در تو است. ضمناً در این مورد آزمون خروجی‌های مختلف سطح پایین و بالا در نقاط متفاوتی از برنامه دیده می‌شود. در ادامه به تحلیل این مورد آزمون خواهیم پرداخت.

جدول 3 - نمونه ورودی‌ها و خروجی‌ها برای برنامه به زبان C متناظر با برنامه شکل 26-(الف)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Violation** | **outL l2 (line# 24)** | **outL l1 (line# 23)** | **outL l1 (line# 20)** | **outL l2 (line# 19)** | **outH h2 (line# 16)** | **outL l1 (line# 15)** | **outH h1 (line# 10)** | **outL l1 (line# 9)** | **outL l1 (line# 6)** | **h2** | **h1** | **l2** | **l1** |
| **No** | 4 | 1 | - | - | 0 | 1 | - | - | - | 0 | 0 | 4 | 0 |
| 4 | 1 | - | - | 1 | 1 | - | - | - | 1 | 1 | 4 | 0 |
| **Yes** | 2 | 0 | 0 | 2 | - | - | - | - | - | 0 | 0 | 2 | 0 |
| 3 | 0 | 0 | 3 | - | - | - | - | - | 1 | 1 | 2 | 0 |
| **Yes** | 0 | 6 | - | - | - | - | 7 | 6 | 7 | 1 | 7 | 0 | 1 |
| 0 | 7 | - | - | - | - | - | - | 7 | 1 | 6 | 0 | 1 |

جدول 4 - نمونه ورودی‌ها و خروجی‌ها برای برنامه به زبان C متناظر با برنامه شکل 26-(ب)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Violation** | **outL l2 (line# 24)** | **outL l1 (line# 23)** | **outL l1 (line# 20)** | **outL l2 (line# 19)** | **outH h2 (line# 16)** | **outL l1 (line# 15)** | **outH h1 (line# 10)** | **outL l1 (line# 9)** | **outL l1 (line# 6)** | **h2** | **h1** | **l2** | **l1** |
| **No** | 4 | - | - | - | 0 | 1 | - | - | - | 0 | 0 | 4 | 0 |
| 4 | - | - | - | 1 | 1 | - | - | - | 1 | 1 | 4 | 0 |
| **No** | BOT | 0 | 0 | BOT | - | - | - | - | - | 0 | 0 | 2 | 0 |
| BOT | 0 | 0 | BOT | - | - | - | - | - | 1 | 1 | 2 | 0 |
| **No** | 0 | - | - | - | - | - | 7 | - | 7 | 1 | 7 | 0 | 1 |
| 0 | - | - | - | - | - | - | - | 7 | 1 | 6 | 0 | 1 |

در مورد آزمون شکل 26، با توجه به وجود جریان‌های صریح و ضمنی، برنامه بازنویسی شده است. هما‌ن‌طور که در جدول 4 مشاهده می‌شود، موارد ناقض عدم تداخل برطرف شده است. باید دقت داشت که در این مورد آزمون، خروجی‌های سطح بالایی هم وجود دارند که تفاوت مقادیر خروجی آن‌ها خط مشی مورد نظر را به مخاطره نمی‌اندازد.

تا اینجا موارد آزمون نمونه مطرح شده، به دلیل عدم وجود ساختار حلقه در آن‌ها، در حالت حساس به پیشرفت همان برنامه‌ بازنویسی‌شده در حالت غیرحساس به پیشرفت را خواهند داشت. در ادامه موارد آزمون نمونه مربوط به حالت حساس به پیشرفت آمده است.

(ت)

(پ) (ب) (الف)

program;

inL l1;

inH h1 , h2;

while l1 > 0 do

l1 = h2 + l1

done;

while h1 > l1 do

l1 = l1 + 3;

outL l1

done;

outL l1;

outH h1

program;

inL l1;

inH h1, h2;

while l1 > 0 do

l1 = h2 + l1 done ;

while h1 > l1 do

l1 = l1 + 3;

if TRUE then

NOP

else

outL l1

endif

done ;

if TRUE then

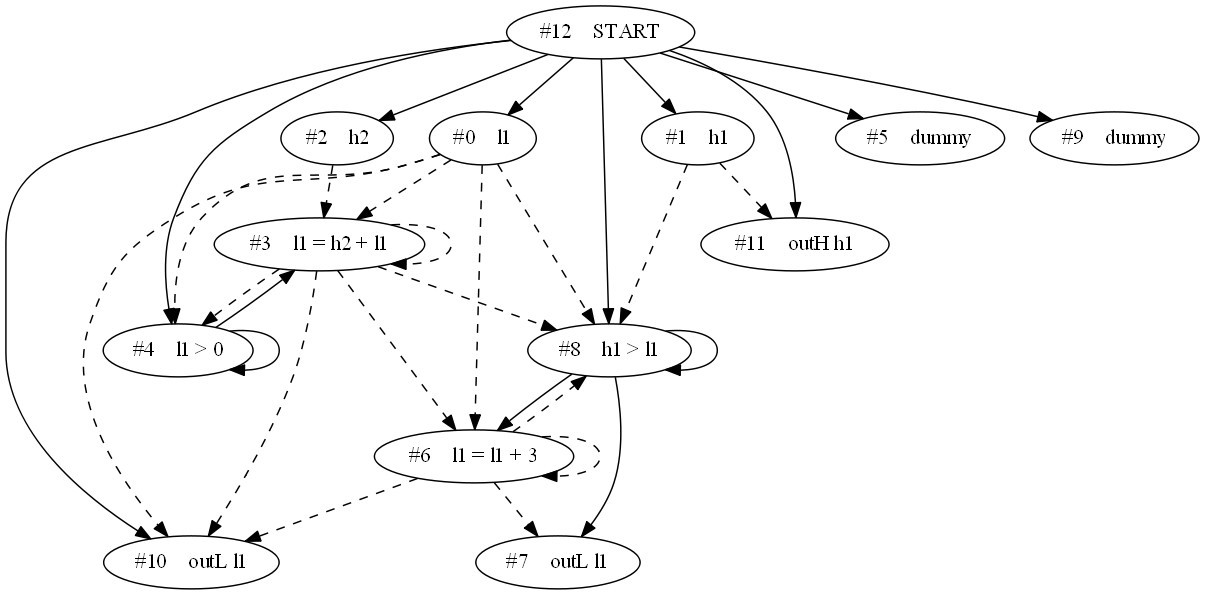
NOP

else

outL l1

endif;

outH h1



program;

inL l1;

inH h1, h2;

if h2 < 0 then

while l1 > 0 do

l1 = h2 + l1

done

endif;

while h1 > l1 do

l1 = l1 + 3;

if TRUE then

NOP

else

outL l1

endif

done ;

if TRUE then

NOP

else

outL l1

endif ;

outH h1

شکل 27 - (الف) برنامه مورد آزمون با نام 11whilewhileconcat.wl (ب) برنامه بازنویسی‌شده برای حالت غیرحساس به پیشرفت مربوط به برنامه الف (پ) برنامه بازنویسی‌شده برای حالت حساس به پیشرفت مربوط به برنامه الف (ت) گراف وابستگی برنامه مربوط به برنامه الف

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Violation** | **outH h1 (line# 12)** | **outL l1 (line# 11)** | **outL l1 (line# 9)** | **h2** | **h1** | **l1** |
| Yes | 1 | 3 | 3 | 0 | 1 | 0 |
| 5 | 6 | 3,6 | 1 | 5 | 0 |
| Yes | 0 | 2 | 2 | -2 | 0 | 1 |
| 5 | 5 | 2,5 | -2 | 5 | 1 |
| Yes | diverge | | | 0 | 1 | 1 |
| 1 | 2 | 2 | -2 | 1 | 1 |

جدول 5 - نمونه ورودی‌ها و خروجی‌ها برای برنامه به زبان C متناظر با برنامه شکل 27-(الف)

**جدول 6 - نمونه ورودی‌ها و خروجی‌ها برای برنامه به زبان C متناظر با برنامه شکل 27-(ب)**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Violation** | **outH h1 (line# 12)** | **outL l1 (line# 11)** | **outL l1 (line# 9)** | **h2** | **h1** | **l1** |
| **No** | 1 | - | - | 0 | 1 | 0 |
| 5 | - | - | 1 | 5 | 0 |
| **No** | 0 | - | - | -2 | 0 | 1 |
| 5 | - | - | -2 | 5 | 1 |
| **No** | diverge | | | 0 | 1 | 1 |
| 1 | - | - | -2 | 1 | 1 |

جدول 7 - نمونه ورودی‌ها و خروجی‌ها برای برنامه به زبان C متناظر با برنامه شکل 27-(پ)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Violation** | **outH h1 (line# 12)** | **outL l1 (line# 11)** | **outL l1 (line# 9)** | **h2** | **h1** | **l1** |
| **No** | 1 | - | - | 0 | 1 | 0 |
| 5 | - | - | 1 | 5 | 0 |
| **No** | 0 | - | - | -2 | 0 | 1 |
| 5 | - | - | -2 | 5 | 1 |
| **No** | 1 | - | - | 0 | 1 | 1 |
| 1 | - | - | -2 | 1 | 1 |

همان‌طور که در جداول فوق آمده است، با بازنویسی برنامه در حالت غیرحساس به پیشرفت، برنامه بازنویسی‌شده در این حالت امن تشخیص داده می‌شود؛ زیرا مشاهده‌گر سطح پایین توانایی مشاهده وضعیت پیشرفت برنامه را ندارد، پس نمی‌تواند تمایزی بین حالت واگرایی یا خاتمه قائل شود. اما واگرا شدن برنامه برای مقدار ورودی 1 برای متغیر سطح پایین l1 و مقادیر ورودی متفاوت برای متغیر سطح بالای h2، باعث می‌شود تا مشاهده‌گر سطح پایین با درک این تفاوت، اطلاعاتی را نسبت به اطلاعات سطح بالا به دست آورد که در برنامه بازنویسی‌شده در حالت حساس به پیشرفت، از این امکان جلوگیری می‌شود. در این مورد آزمون، دو ساختار حلقه در کنار یکدیگر بررسی شد. در مورد آزمون بعدی، دو ساختار حلقه تو در تو را مورد ارزیابی قرار می‌دهیم.

(ت)

(پ) (ب) (الف)

program;

inH h1, h2;

inL l1;

while h1 > l1 do

l1 = l1 + 1;

outL l1;

while h2 < 0 do

h2 = h2 - 1;

outL l1;

l1 = h2

done

done;

outL l1

program;

inH h1, h2;

inL l1;

while h1 > l1 do

l1 = l1 + 1;

if TRUE then

NOP

else

outL l1

endif;

while h2 < 0 do

h2 = h2 - 1;

if TRUE then

NOP

else

outL l1

endif;

l1 = h2

done

done ;

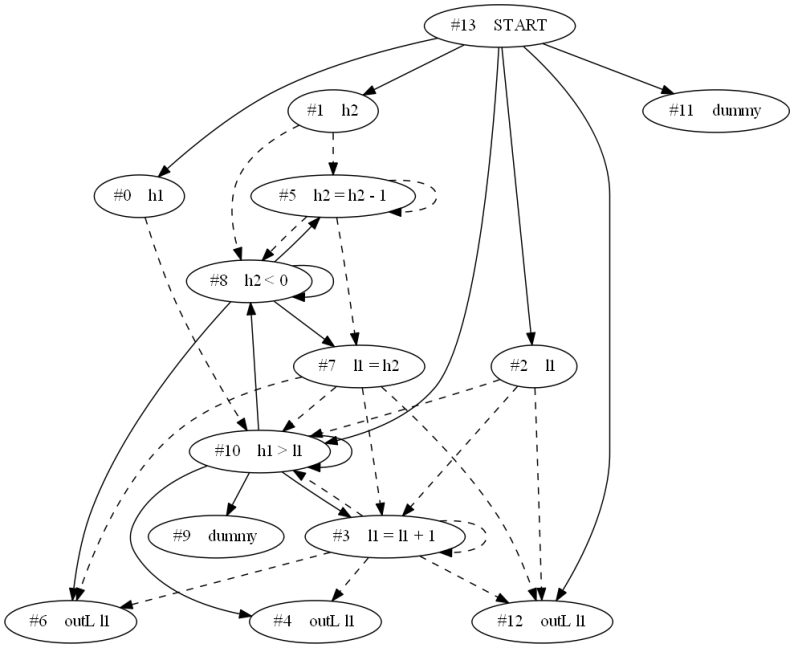
if TRUE then

NOP

else

outL l1

endif



program;

inH h1, h2;

inL l1;

while h1 > l1 do

l1 = l1 + 1;

if TRUE then

NOP

else

outL l1

endif ;

if h2 < 0 then

h2 = h2 - 1;

if TRUE then

NOP

else

outL l1

endif ;

l1 = h2

endif

done ;

if TRUE then

NOP

else

outL l1

endif

شکل 28 - (الف) برنامه مورد آزمون با نام 17whilewhilenested.wl (ب) برنامه بازنویسی‌شده برای حالت غیرحساس به پیشرفت مربوط به برنامه الف (پ) برنامه بازنویسی‌شده برای حالت حساس به پیشرفت مربوط به برنامه الف (ت) گراف وابستگی برنامه مربوط به برنامه الف

**جدول 8 - نمونه ورودی‌ها و خروجی‌ها برای برنامه به زبان C متناظر با برنامه شکل 28-(الف)**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Violation** | **outL l1 (line# 13)** | **outL l1 (line# 9)** | **outL l1 (line# 6)** | **h2** | **h1** | **l1** |
| **Yes** | 2 | - | 2 | 0 | 2 | 1 |
| 1 | - | - | 0 | 0 | 1 |
| **Yes** | 3 | - | 2,3 | 0 | 3 | 1 |
| diverge | | 2 | -1 | 3 | 1 |
| **Yes** | 4 | - | 1,2,3,4 | 2 | 4 | 0 |
| diverge | | 1 | -5 | 4 | 0 |

جدول 9 - نمونه ورودی‌ها و خروجی‌ها برای برنامه به زبان C متناظر با برنامه شکل 28-(ب)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Violation** | **outL l1 (line# 13)** | **outL l1 (line# 9)** | **outL l1 (line# 6)** | **h2** | **h1** | **l1** |
| **No** | - | - | - | 0 | 2 | 1 |
| - | - | - | 0 | 0 | 1 |
| **No** | - | - | - | 0 | 3 | 1 |
| diverge | | | -1 | 3 | 1 |
| **No** | - | - | - | 2 | 4 | 0 |
| diverge | | | -5 | 4 | 0 |

جدول 10 - نمونه ورودی‌ها و خروجی‌ها برای برنامه به زبان C متناظر با برنامه شکل 28-(پ)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Violation** | **outL l1 (line# 13)** | **outL l1 (line# 9)** | **outL l1 (line# 6)** | **h2** | **h1** | **l1** |
| **No** | - | - | - | 0 | 2 | 1 |
| - | - | - | 0 | 0 | 1 |
| **No** | - | - | - | 0 | 3 | 1 |
| - | - | - | -1 | 3 | 1 |
| **No** | - | - | - | 2 | 4 | 0 |
| - | - | - | -5 | 4 | 0 |

همان‌طور که مشاهده می‌شود، در این مورد آزمون موارد ناقض عدم تداخل بسیاری وجود دارد. با بازنویسی برنامه در حالت غیرحساس به پیشرفت، به دلیل تأثیر مقادیر سطح بالا در متغیر l1 در سراسر برنامه، عملاً دستورات نمایش خروجی l1 با دستور NOP جایگزین شده‌اند. باید توجه داشت که حتی در این حالت هم ممکن است در حالت حساس به پیشرفت، خط مشی برآورده نشود. در این نمونه، به دلیل این‌که حلقه درونی همواره واگراست و با توجه به الگوریتم بازنویسی در این حالت حساس به پیشرفت، دستور while با دستور if جایگزین شده است که باعث می‌شود برنامه همواره خاتمه یابد. به این ترتیب، برنامه‌های بازنویسی‌شده نتایج موجود در جدول را منجر می‌شوند.

لازم به یادآوری است که موارد آزمون دیگری برای این پروژه در نظر گرفته شده است که به ذکر تعدادی از آن‌ها بسنده شده است. در طراحی این موارد آزمون سعی شده است تا حالت‌های ممکن در هر دو حالت حساس و غیرحساس به پیشرفت با توجه به ساختارهای مختلف زبان WL در نظر گرفته شود. همه نتایج اجراهای موجود در جداول بالا، از طریق اجرای برنامه‌های به زبان C به دست آمده است که متناظر با هر یک از برنامه‌ها تولید می‌شود.

# فصل ششم ­­جمع‌بندی و کارهای آینده

**جمع‌بندی و کارهای آینده**

در طول فصول گذشته، ابتدا درباره امنیت و خط مشی امنیتی صحبت شد. سپس خط مشی امنیتی عدم تداخل را به عنوان یکی از خطوط مشی که خاصیت نیستند، معرفی کردیم و اشاره‌ای به محدودیت‌هایی که این‌گونه خط مشی‌ها برای اعمال دارند، شد. در ادامه انواع مختلف عدم تداخل؛ یعنی حالت غیرحساس به پیشرفت و حساس به پیشرفت را مطرح کردیم و مشاهده شد که برای اعمال این خط مشی، راه‌های مختلفی وجود دارد که یکی از بهترین مکانیزم‌ها، روش بازنویسی برنامه است. در حالت غیرحساس به پیشرفت، مسیرهایی اهمیت داشت که از مقادیر ورودی سطح بالا آغاز و به دستورات خروجی مقادیر سطح پایین ختم می‌شدند. در حالت حساس به پیشرفت، وضعیت پیشرفت برنامه نیز ممکن بود اطلاعات سطح بالایی را به مشاهده‌گر سطح پایین منتقل کند. از این رو، نگاه ویژه‌ای به ساختار ایجاد واگرایی در برنامه‌ها داشتیم. سپس زبان برنامه‌نویسیِ مدل ارائه‌شده در مقاله اصلی پروژه یا همان WL شرح داده شد. این زبان شامل ساختارهای مختلف و مرسوم زبان‌های برنامه‌نویسی بود که عنصر ایجاد حلقه در آن، ساختار while است. با توجه به تعاریف ارائه شده، الگوریتم‌های بازنویسی برای حالت‌های غیرحساس و حساس به پیشرفت بیان شد و همان‌طور که قبلاً اشاره شده بود، بیان صوری خط مشی‌ها و اثبات صحت و شفافیت الگوریتم‌های بازنویسی، در مقاله اصلی پروژه آمده است. در ادامه به نحوه پیاده‌سازی و ابزارهای مورد استفاده پرداخته شد و با ارائه مورد-آزمون‌های کاربردی، صحت برنامه پیاده‌سازی شده را نشان دادیم.

اما ایده این پروژه یکی از گام‌های ابتدایی و رو به جلویی برای اعمال خط مشی‌های امنیتی با حفظ شفافیت است. می‌توان فکر اصلی استفاده شده در این پروژه را برای زبان‌های برنامه‌نویسی پیشرفته‌تر و رایج‌تر که ساختارهای زبانی پیچیده‌تری دارند به کار بست. زبان‌هایی که از ساختارهای کلاس، شی، چندنخی و سایر ویژگی‌های نوین زبان‌های برنامه‌نویسی امروزی پشتیبانی می‌کنند، می‌توانند به عنوان آینده این پروژه قلمداد کرد. در پیاده‌سازی نیز می‌توان با بهینه‌سازی کد برنامه، به سرعت و استفاده کمتر از حافظه کمک کرد. تابع تحلیل‌گر حلقه نیز به تنهایی می‌تواند موضوع پژوهش جذابی برای علاقمندان این حوزه باشد. از حیث پژوهش‌های نظری نیز می‌توان به دسته‌بندی خط مشی‌های قابل اعمال توسط روش بازنویسی به عنوان یکی از مسائل روز نام برد.

# منابع و مراجع

|  |  |
| --- | --- |
| F.B. Schneider, J.G. Morrisett, and R. Harper, “A Language-Based Approach to Security, in Informatics - 10 Years Back. 10 Years Ahead”, Springer-Verlag Berlin, Heidelberg, 2001, pp. 86-101. | [1] |
| A.Lamei and M. S. Fallah, “Rewriting-Based Enforcement of Noninterference in Programs with Observable Intermediate Values”, To appear in the Journal of Universal Computer Science, Vol. XX, No. XX, Month 20XX, 1-24. | [2] |
| J.A. Goguen and J. Meseguer, Security policies and security models, in Proceedings of IEEE Symposium on Security and Privacy, Vol. 12, IEEE, 1982, pp. 11-18. | [3] |
| A. Russo and A. Sabelfeld, Dynamic vs. Static Flow-Sensitive Security Analysis, in Proceedings of the 2010 23rd IEEE Computer Security Foundations Symposium, CSF '10, IEEE, 2010, pp. 186-199. | [4] |
| J. Ferrante, K.J. Ottenstein, and J.D. Warren, The program dependence graph and its use in optimization, ACM Transactions on Programing Languages and Systems 9 (1987), pp. 319-349. | [5] |
| H. Mantel and H. Sudbrock, Types vs. pdgs in information ow analysis, in Logic-Based Program Synthesis and Transformation, Springer, 2013, pp. 106-121 | [6] |
| “JFlex”, Available: <http://jflex.de/> [Sep. 10, 2015]. | [7] |
| “Bison”, Available: <https://www.gnu.org/software/bison/> [Sep. 10, 2015]. | [8] |
| K. M. Anderson, “Lecture 15: Control Dependence Graphs - CSCI 5828 Slides”, Spring 2000, Available: http://www.cs.colorado.edu/~kena/classes/5828/s00/lectures/lecture15.pdf [Jul. 25 2015]. | [9] |
| T. Teitelbaum, “Introduction to Compilers, Lecture 24: Control Flow Graphs Slides”, 2008. http://www.cs.cornell.edu/courses/cs412/2008sp/lectures/lec24.pdf [Jul. 25 2015]. | [10] |
| “The Program Dependence Graph: Control Flow and Control Dependences – S502 Compilers Slides”. Available: http://pages.cs.wisc.edu/~fischer/cs701.f08/lectures/Lecture19.4up.pdf [Jul. 25 2015]. | [11] |
| S. Moore, A. Askarov, and S. Chong, Precise enforcement of progress-sensitive security, in Proceedings of the 2012 ACM Conference on Computer and Communications Security, CCS '12, ACM, 2012, pp. 881-893. | [12] |
| E. R. Gansner and S. C. North. “An Open Graph Visualization and Its Application to Software Engineering”. Software – Practice and Experience Journal, vol. 30, No. 11, pp. 1203-1233, 2000. Available: [www.graphviz.org](http://www.graphviz.org/) [Aug. 12 2015]. | [13] |
| “AProVE”, Available: http://aprove.informatik.rwth-aachen.de/index.asp?subform=home.html [Aug. 25 2015]. | [14] |
| “RSyntaxTextArea”, Available: http://bobbylight.github.io/RSyntaxTextArea/ [Sep. 04 2015]. | [15] |

# پیوست

**قسمت‌هایی از کدهای پیاده‌سازی پروژه**

برای مشاهده همه فایل‌های مرتبط با پروژه، می‌توانید از آدرس <https://github.com/smahmadpanah/BScProject> استفاده کنید.

**lexer.l:**

(این فایل ورودی ابزار jflex است. پس از تولید فایل Yylex.java، تغییراتی در آن فایل نیز اعمال شده است.)

package wlrewriter;

import java.lang.\*;

%%

%byaccj

LETTER = [a-zA-Z]

DIGIT = [0-9]

NONZERO\_DIGIT = [1-9]

PROGRAM\_KW = (program)

AND\_KW = (and)

OR\_KW = (or)

NEG\_KW = [!]

ASSIGN\_KW = [=]

IF\_KW = (if)

THEN\_KW = (then)

ELSE\_KW = (else)

ENDIF\_KW = (endif)

WHILE\_KW = (while)

DO\_KW = (do)

DONE\_KW = (done)

NOP\_KW = (NOP)

BOT\_KW = (BOT)

INL\_KW = (inL)

INH\_KW = (inH)

OUTL\_KW = (outL)

OUTH\_KW = (outH)

PLUS\_KW =[+]

MINUS\_KW =[-]

LT\_KW =[<]

LE\_KW =(<=)

EQ\_KW =(==)

GT\_KW =[>]

GE\_KW =(>=)

LPAR\_KW = [(]

RPAR\_KW = [)]

INTEGER\_NUMBER ="0"|({NONZERO\_DIGIT}{DIGIT}\*)

BOOL\_CONSTANT ="true"|"false"|"TRUE"|"FALSE"

IDENTIFIER ={LETTER}+|{LETTER}({LETTER}|{DIGIT})\*

LineTerminator = \r|\n|\r\n

%%

{LineTerminator} {yyline++;}

{PROGRAM\_KW} {

return YYParser.PROGRAM\_KW;

}

{AND\_KW} {

return YYParser.AND\_KW;

}

{OR\_KW} {

return YYParser.OR\_KW;

}

{NEG\_KW} {

return YYParser.NEG\_KW;

}

{ASSIGN\_KW} {

return YYParser.ASSIGN\_KW;

}

{IF\_KW} {

return YYParser.IF\_KW;

}

{THEN\_KW} {

return YYParser.THEN\_KW;

}

{ELSE\_KW} {

return YYParser.ELSE\_KW;

}

{ENDIF\_KW} {

return YYParser.ENDIF\_KW;

}

{WHILE\_KW} {

return YYParser.WHILE\_KW;

}

{DO\_KW} {

return YYParser.DO\_KW;

}

{DONE\_KW} {

return YYParser.DONE\_KW;

}

{NOP\_KW} {

return YYParser.NOP\_KW;

}

{BOT\_KW} {

return YYParser.BOT\_KW;

}

{INL\_KW} {

return YYParser.INL\_KW;

}

{INH\_KW} {

return YYParser.INH\_KW;

}

{OUTL\_KW} {

return YYParser.OUTL\_KW;

}

{OUTH\_KW} {

return YYParser.OUTH\_KW;

}

{PLUS\_KW} {

return YYParser.PLUS\_KW;

}

{MINUS\_KW} {

return YYParser.MINUS\_KW;

}

{LT\_KW} {

return YYParser.LT\_KW;

}

{LE\_KW} {

return YYParser.LE\_KW;

}

{EQ\_KW} {

return YYParser.EQ\_KW;

}

{GT\_KW} {

return YYParser.GT\_KW;

}

{GE\_KW} {

return YYParser.GE\_KW;

}

{INTEGER\_NUMBER} {

YYParser.stmt=yytext();

return YYParser.INTEGER\_NUMBER;

}

{BOOL\_CONSTANT} {

String s=yytext();

YYParser.stmt=yytext();

return YYParser.BOOL\_CONSTANT;

}

{IDENTIFIER} {

YYParser.stmt=yytext();

return YYParser.IDENTIFIER;

}

{LPAR\_KW} {

return YYParser.LPAR\_KW;

}

{RPAR\_KW} {

return YYParser.RPAR\_KW;

}

"," {

return ',';

}

";" {

return ';';

}

. {

}

**Parser.y:**

(این فایل ورودی ابزار bison است. پس از تولید فایل YYParser.java، تغییراتی در آن فایل نیز اعمال شده است.)

%{

package wlrewriter;

import java.io.\*;

import java.lang.\*;

import java.util.\*;

import java.awt.Color;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

%}

%type <eval> program exp c clist varlist b n x M N

%token <eval> PROGRAM\_KW AND\_KW OR\_KW NEG\_KW LPAR\_KW RPAR\_KW ASSIGN\_KW IF\_KW THEN\_KW ELSE\_KW ENDIF\_KW WHILE\_KW DO\_KW DONE\_KW NOP\_KW BOT\_KW INL\_KW INH\_KW OUTL\_KW OUTH\_KW PLUS\_KW MINUS\_KW LT\_KW LE\_KW EQ\_KW GT\_KW GE\_KW

%token <eval> INTEGER\_NUMBER

%token <eval> BOOL\_CONSTANT

%token <eval> IDENTIFIER

%code {

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* MAIN \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

static PrintStream writer;

static String stmt;

private int nodeCounter = 0;

private static String sourceCodeFileName;

private String cSourceCodeOfInput="";

public String cSourceCodeForPSNI="";

private int whileID = 0;

public int controlFlag = 0;

public static int selectorPDGorPINIorPSNI; //0: pdg | 1: pini | 2:psni

public ArrayList<Variable> symbolTableOfVariables = new ArrayList<Variable>(); //static bood ghablan

// public static void main(String args[]) throws IOException, FileNotFoundException {

public static void mainMethod(String sFileName, int selector) {

YYParser yyparser;

final Yylex lexer;

selectorPDGorPINIorPSNI = selector;

// System.out.println("Enter the source code file path:");

// Scanner sc = new Scanner(System.in);

// sourceCodeFileName = sc.next();

sourceCodeFileName = sFileName;

// String sourceCodeFileName = "input-while.wl";

try {

writer = new PrintStream(new File("reduction.txt"));

} catch (FileNotFoundException ex) {

System.out.println("File reduction not found.");

GUI.terminal.appendError("File reduction not found.");

}

Yylex yylexTemp = null;

try{

yylexTemp = new Yylex(new InputStreamReader(new FileInputStream(sourceCodeFileName)));

} catch (Exception ex) {

System.err.println("Source code file not found!");

GUI.terminal.appendError("Source code file not found!");

System.exit(0);

}

lexer = yylexTemp;

yyparser = new YYParser(new Lexer() {

@Override

public int yylex() {

int yyl\_return = -1;

try {

yyl\_return = lexer.yylex();

} catch (IOException e) {

System.err.println("IO error : " + e);

GUI.terminal.appendError("IO error : " + e);

}

return yyl\_return;

}

@Override

public void yyerror(String error) {

//System.err.println ("Error : " + error);

System.err.println("\*\*Error: Line " + lexer.getYyline() + " near token '" + lexer.yytext() + "' --> Message: " + error + " \*\*");

GUI.terminal.append("\*\*Error: Line " + lexer.getYyline() + " near token '" + lexer.yytext() + "' --> Message: " + error + " \*\*", Color.orange);

writer.print("\*\*Error: Line " + lexer.getYyline() + " near token '" + lexer.yytext() + "' --> Message: " + error + " \*\*");

}

@Override

public Object getLVal() {

return null;

}

});

try {

yyparser.parse();

} catch (IOException ex) {

System.out.println("parse method is wrong.");

GUI.terminal.appendError("parse method is wrong.");

}

writer.close();

}

public static String getSourceCodeFileName() {

return sourceCodeFileName;

}

/\*-------------------------------------------------------------------------------------------\*/

/\*-------------------------------------------------------------------------------------------\*/

}

%left AND\_KW OR\_KW

%nonassoc q

%right ASSIGN\_KW

%left EQ\_KW

%left LT\_KW GT\_KW

%left LE\_KW GE\_KW

%left PLUS\_KW MINUS\_KW

%left THEN\_KW

%nonassoc p

%nonassoc ELSE\_KW

%%

program : PROGRAM\_KW ';' clist

{

writer.print("\t program -> PROGRAM\_KW ';' clist \n") ;

writer.print("###Hooray! - Your program is syntactically correct### \n");

System.out.println("###Hooray! - Your program is syntactically correct###");

GUI.terminal.append("Your program is syntactically correct");

$$ = new eval();

((eval)$$).stmt += "program; " + ((eval)$3).stmt;

((eval)$$).cSourceCode += "#include <stdio.h> \n \n#define TRUE 1 \n#define true 1 \n"

+ "#define FALSE 0 \n#define false 0 \n\n\n"

+"int main() { " + ((eval)$3).cSourceCode + "return 0;}";

//System.out.print(((eval)$$).cSourceCode);

cSourceCodeForPSNI = ((eval)$$).cSourceCode;

//omit WhileIDs

String pattern = "~WhileID[0-9]+~";

Pattern r = Pattern.compile(pattern);

Matcher m = r.matcher(((eval)$$).cSourceCode);

cSourceCodeOfInput = m.replaceAll("");

pattern = "~ENDWhileID[0-9]+~";

r = Pattern.compile(pattern);

m = r.matcher(cSourceCodeOfInput);

cSourceCodeOfInput = m.replaceAll("");

//System.out.println("\n\n\*\*\*\*\*\*\*\n\n" + cSourceCodeOfInput);

//\*\*\*\*\*\*\*\*\*\*WE CAN USE cSourceCodeOfInput TO MAKE A .c FILE FOR COMPILE IT IN C LANGUAGE.

writer.print(((eval)$$).stmt+ "\n");

//((eval)$$).variables.addAll(((eval)$3).variables);

((eval)$$).node = new Node(nodeCounter++, "START");

//System.out.println(((eval)$3).nodeIdAndStmt);

((eval)$$).nodeIdAndStmt += "#" + ((eval)$$).node.getNodeID() + ":" + "program; \n";

((eval)$$).nodeIdAndStmt += ((eval)$3).nodeIdAndStmt;

((eval)$$).node.setNodeIdAndStmt(((eval)$$).nodeIdAndStmt);

/\*for(String str : ((eval)$$).variables){

for(Variable varvar : symbolTableOfVariables){

if(str.equals(varvar.name)){

((eval)$$).node.addToVariablesOfNode(varvar);

break;

}

}

}

\*/

((eval)$$).list = new MyLinkedList(((eval)$$).node);

((eval)$$).list.merge(((eval)$3).list);

((eval)$$).list.merge(new MyLinkedList(new Node(nodeCounter++, "STOP")));

if(controlFlag==2){

try{

PrintStream writer3 = new PrintStream(new File(sourceCodeFileName+"-PSNI.c"));

writer3.print(cSourceCodeOfInput);

}

catch (Exception e){

System.out.println("ERROR in FILE.");

GUI.terminal.appendError("ERROR in FILE.");

}

}

if(controlFlag==1){

psni.setPSNICFG(((eval)$$).list);

try{

PrintStream writer1 = new PrintStream(new File(sourceCodeFileName+"-PINI.c"));

writer1.print(cSourceCodeOfInput);

}

catch (Exception e){

System.out.println("ERROR in FILE.");

GUI.terminal.appendError("ERROR in FILE.");

}

}

if(controlFlag==0){

System.out.println("Control Flow Graph is created.");

GUI.terminal.append("Control Flow Graph is created.");

try{

PrintStream writer2 = new PrintStream(new File(sourceCodeFileName+".c"));

writer2.print(cSourceCodeOfInput);

}

catch (Exception e){

System.out.println("ERROR in FILE.");

GUI.terminal.appendError("ERROR in FILE.");

}

PDGBuilder pdg = new PDGBuilder(((eval)$$).list); //the CFG is input to build the Forward Dominance Tree and after that, CFG and DDG that make PDG! :)

PINIRewriter pini = null;

if(selectorPDGorPINIorPSNI != 0){ // faghat namayeshe pdg ro nemikhad, ya pini ya psni

pini = new PINIRewriter(pdg.getPDG());

}

if(selectorPDGorPINIorPSNI == 2){

psni = new PSNIRewriter(pini);

}

}

};

clist: c

{

writer.print("\t clist -> c \n") ;

$$=new eval();

((eval)$$).stmt += ((eval)$1).stmt;

((eval)$$).cSourceCode += ((eval)$1).cSourceCode;

writer.print(((eval)$$).stmt+ "\n");

((eval)$$).nodeIdAndStmt += ((eval)$1).nodeIdAndStmt;

//((eval)$$).variables.addAll(((eval)$1).variables);

((eval)$$).list = ((eval)$1).list;

};

| clist ';' M c

{

writer.print("\t clist -> clist ; M c \n") ;

$$=new eval();

((eval)$$).stmt += ((eval)$1).stmt + "; " + ((eval)$4).stmt;

((eval)$$).cSourceCode += ((eval)$1).cSourceCode + "; " + ((eval)$4).cSourceCode;

((eval)$$).cSourceCode += "\n";

writer.print(((eval)$$).stmt+ "\n");

((eval)$$).nodeIdAndStmt += ((eval)$1).nodeIdAndStmt + "; \n" + ((eval)$4).nodeIdAndStmt;

//((eval)$$).variables.addAll(((eval)$1).variables);

//((eval)$$).variables.addAll(((eval)$4).variables);

((eval)$$).list = ((eval)$1).list;

((eval)$$).list.merge(((eval)$4).list);

};

exp : b

{

writer.print("\t exp -> b \n") ;

$$=new eval();

((eval)$$).stmt += ((eval)$1).stmt;

((eval)$$).cSourceCode += ((eval)$1).cSourceCode;

writer.print(((eval)$$).stmt+ "\n");

};

| n

{

writer.print("\t exp -> n \n") ;

$$=new eval();

((eval)$$).stmt += ((eval)$1).stmt;

((eval)$$).cSourceCode += ((eval)$1).cSourceCode;

writer.print(((eval)$$).stmt+ "\n");

};

| x

{

writer.print("\t exp -> x \n") ;

$$=new eval();

((eval)$$).stmt += ((eval)$1).stmt;

((eval)$$).cSourceCode += ((eval)$1).cSourceCode;

((eval)$$).variables.add(((eval)$1).stmt);

writer.print(((eval)$$).stmt+ "\n");

boolean check = false;

for(Variable v : symbolTableOfVariables){

if(((eval)$1).stmt.equals(v.name)){

check = true;

break;

}

}

if(!check){

System.err.println("undefined variable!");

GUI.terminal.append("undefined variable!\n\t"+((eval)$$).stmt, Color.orange);

System.err.println("\t"+((eval)$$).stmt);

//System.exit(0);

return -1;

}

};

| exp EQ\_KW exp

{

writer.print("\t exp -> exp EQ\_KW exp \n") ;

$$=new eval();

((eval)$$).stmt += ((eval)$1).stmt + " == "+ ((eval)$3).stmt;

((eval)$$).cSourceCode += ((eval)$1).cSourceCode + " == "+ ((eval)$3).cSourceCode;

writer.print(((eval)$$).stmt+ "\n");

((eval)$$).variables.addAll(((eval)$1).variables);

((eval)$$).variables.addAll(((eval)$3).variables);

};

| exp LT\_KW exp

{

writer.print("\t exp -> exp LT\_KW exp \n") ;

$$=new eval();

((eval)$$).stmt += ((eval)$1).stmt + " < "+ ((eval)$3).stmt;

((eval)$$).cSourceCode += ((eval)$1).cSourceCode + " < "+ ((eval)$3).cSourceCode;

writer.print(((eval)$$).stmt+ "\n");

((eval)$$).variables.addAll(((eval)$1).variables);

((eval)$$).variables.addAll(((eval)$3).variables);

};

| exp LE\_KW exp

{

writer.print("\t exp -> exp LE\_KW exp \n") ;

$$=new eval();

((eval)$$).stmt += ((eval)$1).stmt + " <= "+ ((eval)$3).stmt;

((eval)$$).cSourceCode += ((eval)$1).cSourceCode + " <= "+ ((eval)$3).cSourceCode;

writer.print(((eval)$$).stmt+ "\n");

((eval)$$).variables.addAll(((eval)$1).variables);

((eval)$$).variables.addAll(((eval)$3).variables);

};

| exp GE\_KW exp

{

writer.print("\t exp -> exp GE\_KW exp \n") ;

$$=new eval();

((eval)$$).stmt += ((eval)$1).stmt + " >= "+ ((eval)$3).stmt;

((eval)$$).cSourceCode += ((eval)$1).cSourceCode + " >= "+ ((eval)$3).cSourceCode;

writer.print(((eval)$$).stmt+ "\n");

((eval)$$).variables.addAll(((eval)$1).variables);

((eval)$$).variables.addAll(((eval)$3).variables);

};

| exp GT\_KW exp

{

writer.print("\t exp -> exp GT\_KW exp \n") ;

$$=new eval();

((eval)$$).stmt += ((eval)$1).stmt + " > "+ ((eval)$3).stmt;

((eval)$$).cSourceCode += ((eval)$1).cSourceCode + " > "+ ((eval)$3).cSourceCode;

writer.print(((eval)$$).stmt+ "\n");

((eval)$$).variables.addAll(((eval)$1).variables);

((eval)$$).variables.addAll(((eval)$3).variables);

};

| exp PLUS\_KW exp

{

writer.print("\t exp -> exp PLUS\_KW exp \n") ;

$$=new eval();

((eval)$$).stmt += ((eval)$1).stmt + " + "+ ((eval)$3).stmt;

((eval)$$).cSourceCode += ((eval)$1).cSourceCode + " + "+ ((eval)$3).cSourceCode;

writer.print(((eval)$$).stmt+ "\n");

((eval)$$).variables.addAll(((eval)$1).variables);

((eval)$$).variables.addAll(((eval)$3).variables);

};

| exp MINUS\_KW exp

{

writer.print("\t exp -> exp MINUS\_KW exp \n") ;

$$=new eval();

((eval)$$).stmt += ((eval)$1).stmt + " - "+ ((eval)$3).stmt;

((eval)$$).cSourceCode += ((eval)$1).cSourceCode + " - "+ ((eval)$3).cSourceCode;

writer.print(((eval)$$).stmt+ "\n");

((eval)$$).variables.addAll(((eval)$1).variables);

((eval)$$).variables.addAll(((eval)$3).variables);

};

| exp AND\_KW M exp

{

writer.print("\t exp -> exp AND\_KW exp \n") ;

$$=new eval();

((eval)$$).stmt += ((eval)$1).stmt + " and "+ ((eval)$4).stmt;

((eval)$$).cSourceCode += ((eval)$1).cSourceCode + " && "+ ((eval)$4).cSourceCode;

writer.print(((eval)$$).stmt+ "\n");

((eval)$$).variables.addAll(((eval)$1).variables);

((eval)$$).variables.addAll(((eval)$4).variables);

};

| exp OR\_KW M exp

{

writer.print("\t exp -> exp OR\_KW exp \n") ;

$$=new eval();

((eval)$$).stmt += ((eval)$1).stmt + " or "+ ((eval)$4).stmt;

((eval)$$).cSourceCode += ((eval)$1).cSourceCode + " || "+ ((eval)$4).cSourceCode;

writer.print(((eval)$$).stmt+ "\n");

((eval)$$).variables.addAll(((eval)$1).variables);

((eval)$$).variables.addAll(((eval)$4).variables);

};

| NEG\_KW exp %prec q

{

writer.print("\t exp -> NEG\_KW exp \n") ;

$$=new eval();

((eval)$$).stmt += "!("+ ((eval)$2).stmt + ")";

((eval)$$).cSourceCode += "!("+ ((eval)$2).cSourceCode + ")";

writer.print(((eval)$$).stmt+ "\n");

((eval)$$).variables.addAll(((eval)$2).variables);

};

| LPAR\_KW exp RPAR\_KW %prec p

{

writer.print("\t exp -> LPAR\_KW exp RPAR\_KW \n") ;

$$=new eval();

((eval)$$).stmt += "("+ ((eval)$2).stmt + ")";

((eval)$$).cSourceCode += "("+ ((eval)$2).cSourceCode + ")";

writer.print(((eval)$$).stmt+ "\n");

((eval)$$).variables.addAll(((eval)$2).variables);

};

c : NOP\_KW

{

writer.print("\t c -> NOP\_KW \n") ;

$$=new eval();

((eval)$$).stmt += "NOP";

((eval)$$).cSourceCode += ";";

writer.print(((eval)$$).stmt+ "\n");

((eval)$$).node = new Node(nodeCounter++, ((eval)$$).stmt);

((eval)$$).nodeIdAndStmt += "#" + ((eval)$$).node.getNodeID() + ":" + ((eval)$$).stmt;

((eval)$$).node.setNodeIdAndStmt(((eval)$$).nodeIdAndStmt);

((eval)$$).list = new MyLinkedList(((eval)$$).node);

};

| x ASSIGN\_KW exp

{

writer.print("\t c -> x ASSIGN\_KW exp \n") ;

$$=new eval();

((eval)$$).stmt += ((eval)$1).stmt + " = " + ((eval)$3).stmt;

((eval)$$).cSourceCode += ((eval)$1).cSourceCode + " = " + ((eval)$3).cSourceCode;

writer.print(((eval)$$).stmt+ "\n");

((eval)$$).variables.addAll(((eval)$3).variables);

boolean check = false;

for(Variable v : symbolTableOfVariables){

if(((eval)$1).stmt.equals(v.name)){

((eval)$$).node = new Node(nodeCounter++, ((eval)$$).stmt);

((eval)$$).nodeIdAndStmt += "#" + ((eval)$$).node.getNodeID() + ":" + ((eval)$1).stmt + " = " + ((eval)$3).stmt;

((eval)$$).node.setNodeIdAndStmt(((eval)$$).nodeIdAndStmt);

for(String str : ((eval)$$).variables){

for(Variable varvar : symbolTableOfVariables){

if(str.equals(varvar.name)){

((eval)$$).node.addToVariablesOfNode(varvar);

break;

}

}

}

for(Variable varvar : symbolTableOfVariables){

if(varvar.name.equals(((eval)$1).stmt)){

((eval)$$).node.setAssignedVariable(varvar);

break;

}

}

((eval)$$).list = new MyLinkedList(((eval)$$).node);

check = true;

break;

}

}

if(!check){

System.err.println("undefined variable can not be assigned:");

GUI.terminal.append("undefined variable can not be assigned:\n\t"+((eval)$$).stmt, Color.orange);

System.err.println("\t"+((eval)$$).stmt);

//System.exit(0);

return -1;

}

((eval)$$).variables.add(((eval)$1).stmt); //not necessary

};

| INL\_KW varlist

{

writer.print("\t c -> INL\_KW varlist \n") ;

$$=new eval();

((eval)$$).stmt += "inL "+((eval)$2).stmt;

//((eval)$$).cSourceCode += "int "+((eval)$2).cSourceCode;

for(String alpha : ((eval)$2).variables){

((eval)$$).cSourceCode += "int " + alpha + "; //type: low \n";

((eval)$$).cSourceCode += "scanf(\"%d\", &" + alpha + ");\n";

}

writer.print(((eval)$$).stmt+ "\n");

((eval)$$).variables.addAll(((eval)$2).variables);

boolean first = true;

for(String i : ((eval)$2).variables){

Variable currentVar = new Variable(i);

for(Variable v : symbolTableOfVariables){

if(v.name.equals(currentVar.name)){

v.type = "low";

currentVar.type = "low";

}

}

((eval)$$).node = new Node(nodeCounter++, currentVar.name);

((eval)$$).node.setAssignedVariable(currentVar);

//((eval)$$).node.addToVariablesOfNode(currentVar);

if(first){

((eval)$$).list = new MyLinkedList(((eval)$$).node);

first = false;

}

else{

((eval)$$).list.merge(new MyLinkedList(((eval)$$).node));

}

}

((eval)$$).nodeIdAndStmt += "#" + ((eval)$$).node.getNodeID() + ":" + ((eval)$$).stmt;

((eval)$$).node.setNodeIdAndStmt(((eval)$$).nodeIdAndStmt);

};

| INH\_KW varlist

{

writer.print("\t c -> INH\_KW varlist \n") ;

$$=new eval();

((eval)$$).stmt += "inH "+((eval)$2).stmt;

//((eval)$$).cSourceCode += "int "+((eval)$2).cSourceCode;

for(String alpha : ((eval)$2).variables){

((eval)$$).cSourceCode += "int " + alpha + "; //type: high \n";

((eval)$$).cSourceCode += "scanf(\"%d\", &" + alpha + ");\n";

}

writer.print(((eval)$$).stmt+ "\n");

((eval)$$).variables.addAll(((eval)$2).variables);

boolean first = true;

for(String i : ((eval)$2).variables){

Variable currentVar = new Variable(i);

for(Variable v : symbolTableOfVariables){

if(v.name.equals(currentVar.name)){

v.type = "high";

currentVar.type = "high";

}

}

((eval)$$).node = new Node(nodeCounter++, currentVar.name);

((eval)$$).node.setAssignedVariable(currentVar);

//((eval)$$).node.addToVariablesOfNode(currentVar);

if(first){

((eval)$$).list = new MyLinkedList(((eval)$$).node);

first = false;

}

else{

((eval)$$).list.merge(new MyLinkedList(((eval)$$).node));

}

}

((eval)$$).nodeIdAndStmt += "#" + ((eval)$$).node.getNodeID() + ":" + ((eval)$$).stmt;

((eval)$$).node.setNodeIdAndStmt(((eval)$$).nodeIdAndStmt);

};

| OUTL\_KW x

{

writer.print("\t c -> OUTL\_KW x \n") ;

$$=new eval();

((eval)$$).stmt += "outL " + ((eval)$2).stmt;

((eval)$$).cSourceCode += "printf(\"%d\\n\","+((eval)$2).cSourceCode+")";

writer.print(((eval)$$).stmt+ "\n");

((eval)$$).variables.add(((eval)$2).stmt);

boolean check = false;

for(Variable v : symbolTableOfVariables){

if(((eval)$2).stmt.equals(v.name) && v.type.equals("low")){

((eval)$$).node = new Node(nodeCounter++, ((eval)$$).stmt);

((eval)$$).nodeIdAndStmt += "#" + ((eval)$$).node.getNodeID() + ":" + ((eval)$$).stmt;

((eval)$$).node.setNodeIdAndStmt(((eval)$$).nodeIdAndStmt);

for(String str : ((eval)$$).variables){

for(Variable varvar : symbolTableOfVariables){

if(str.equals(varvar.name)){

((eval)$$).node.addToVariablesOfNode(varvar);

break;

}

}

}

((eval)$$).list = new MyLinkedList(((eval)$$).node);

check = true;

break;

}

}

if(!check){

System.err.println("undefined variable!");

GUI.terminal.append("undefined variable!\n\t"+((eval)$$).stmt, Color.orange);

System.err.println("\t"+((eval)$$).stmt);

//System.exit(0);

return -1;

}

};

| OUTH\_KW x

{

writer.print("\t c -> OUTH\_KW x \n") ;

$$=new eval();

((eval)$$).stmt += "outH " + ((eval)$2).stmt;

((eval)$$).cSourceCode += "printf(\"%d\\n\","+((eval)$2).cSourceCode+")";

writer.print(((eval)$$).stmt+ "\n");

((eval)$$).variables.add(((eval)$2).stmt);

boolean check = false;

for(Variable v : symbolTableOfVariables){

if(((eval)$2).stmt.equals(v.name) && v.type.equals("high")){

((eval)$$).node = new Node(nodeCounter++, ((eval)$$).stmt);

((eval)$$).nodeIdAndStmt += "#" + ((eval)$$).node.getNodeID() + ":" + ((eval)$$).stmt;

((eval)$$).node.setNodeIdAndStmt(((eval)$$).nodeIdAndStmt);

for(String str : ((eval)$$).variables){

for(Variable varvar : symbolTableOfVariables){

if(str.equals(varvar.name)){

((eval)$$).node.addToVariablesOfNode(varvar);

break;

}

}

}

((eval)$$).list = new MyLinkedList(((eval)$$).node);

check = true;

break;

}

}

if(!check){

System.err.println("undefined variable!");

System.err.println("\t"+((eval)$$).stmt);

GUI.terminal.append("undefined variable!\n\t"+((eval)$$).stmt, Color.orange);

//System.exit(0);

return -1;

}

};

| OUTL\_KW BOT\_KW

{

writer.print("\t c -> OUTL\_KW BOT\_KW \n") ;

$$=new eval();

((eval)$$).stmt += "outL BOT";

((eval)$$).cSourceCode += "printf(\"BOT\\n\")";

writer.print(((eval)$$).stmt+ "\n");

((eval)$$).node = new Node(nodeCounter++, ((eval)$$).stmt);

((eval)$$).nodeIdAndStmt += "#" + ((eval)$$).node.getNodeID() + ":" + ((eval)$$).stmt;

((eval)$$).node.setNodeIdAndStmt(((eval)$$).nodeIdAndStmt);

((eval)$$).list = new MyLinkedList(((eval)$$).node);

};

| OUTH\_KW BOT\_KW

{

writer.print("\t c -> OUTH\_KW BOT\_KW \n") ;

$$=new eval();

((eval)$$).stmt += "outH BOT";

((eval)$$).cSourceCode += "printf(\"BOT\\n\")";

writer.print(((eval)$$).stmt+ "\n");

((eval)$$).node = new Node(nodeCounter++, ((eval)$$).stmt);

((eval)$$).nodeIdAndStmt += "#" + ((eval)$$).node.getNodeID() + ":" + ((eval)$$).stmt;

((eval)$$).node.setNodeIdAndStmt(((eval)$$).nodeIdAndStmt);

((eval)$$).list = new MyLinkedList(((eval)$$).node);

};

| IF\_KW exp THEN\_KW M clist ENDIF\_KW %prec p

{

writer.print("\t c -> IF\_KW exp THEN\_KW M clist ENDIF\_KW \n") ;

$$=new eval();

((eval)$$).stmt += " if " + ((eval)$2).stmt + " then " + ((eval)$5).stmt + " endif";

((eval)$$).cSourceCode += " if (" + ((eval)$2).cSourceCode + ") { " + ((eval)$5).cSourceCode + ";}";

writer.print(((eval)$$).stmt+ "\n");

((eval)$$).variables.addAll(((eval)$2).variables);

//((eval)$$).variables.addAll(((eval)$5).variables);

((eval)$$).node = new Node(nodeCounter++, ((eval)$2).stmt);//condition expression node

((eval)$$).nodeIdAndStmt += " if " + "#" + ((eval)$$).node.getNodeID() + ":" + ((eval)$2).stmt + " then \n" + ((eval)$5).nodeIdAndStmt + " endif";

((eval)$$).node.setNodeIdAndStmt(((eval)$$).nodeIdAndStmt);

for(String str : ((eval)$$).variables){

for(Variable v : symbolTableOfVariables){

if(str.equals(v.name)){

((eval)$$).node.addToVariablesOfNode(v);

break;

}

}

}

((eval)$$).list = new MyLinkedList(((eval)$$).node);

Node dummy = new Node(nodeCounter++, "dummy");//dummy node for last node of if

((eval)$$).list.getLast().setNextPointer2(dummy);//if false

dummy.addPreviousPointer(((eval)$$).list.getLast()); //backward pointer

((eval)$$).list.merge(((eval)$5).list);//if true

((eval)$$).list.merge(new MyLinkedList(dummy));

};

| IF\_KW exp THEN\_KW M clist ELSE\_KW N M clist ENDIF\_KW

{

writer.print("\t c -> IF\_KW exp THEN\_KW M clist ELSE\_KW N M clist ENDIF\_KW \n") ;

$$=new eval();

((eval)$$).stmt += " if " + ((eval)$2).stmt + " then " + ((eval)$5).stmt + " else " + ((eval)$9).stmt + " endif ";

((eval)$$).cSourceCode += " if (" + ((eval)$2).cSourceCode + ") { " + ((eval)$5).cSourceCode + ";} else {" + ((eval)$9).cSourceCode + ";}";

writer.print(((eval)$$).stmt+ "\n");

((eval)$$).variables.addAll(((eval)$2).variables);

//((eval)$$).variables.addAll(((eval)$5).variables);

//((eval)$$).variables.addAll(((eval)$9).variables);

((eval)$$).node = new Node(nodeCounter++, ((eval)$2).stmt);//condition expression node

((eval)$$).nodeIdAndStmt += " if " + "#" + ((eval)$$).node.getNodeID() + ":" + ((eval)$2).stmt + " then \n" + ((eval)$5).nodeIdAndStmt + " else \n" + ((eval)$9).nodeIdAndStmt + " endif ";

((eval)$$).node.setNodeIdAndStmt(((eval)$$).nodeIdAndStmt);

for(String str : ((eval)$$).variables){

for(Variable v : symbolTableOfVariables){

if(str.equals(v.name)){

((eval)$$).node.addToVariablesOfNode(v);

break;

}

}

}

((eval)$$).list = new MyLinkedList(((eval)$$).node);

Node dummy = new Node(nodeCounter++, "dummy");//dummy node for last node of if

MyLinkedList dummyList = new MyLinkedList(dummy);

((eval)$$).list.getLast().setNextPointer2(((eval)$9).list.getFirst());//if false - else section

((eval)$9).list.getFirst().addPreviousPointer(((eval)$$).list.getLast()); //backward pointer

((eval)$$).list.getNodeSet().addAll(((eval)$9).list.getNodeSet());

((eval)$9).list.merge(dummyList);

((eval)$$).list.merge(((eval)$5).list);//if true

((eval)$$).list.merge(dummyList);

};

| WHILE\_KW exp DO\_KW M clist DONE\_KW

{

writer.print("\t c -> WHILE\_KW exp DO\_KW M clist DONE\_KW \n") ;

$$=new eval();

((eval)$$).stmt += "while " + ((eval)$2).stmt + " do " + ((eval)$5).stmt + " done ";

writer.print(((eval)$$).stmt+ "\n");

((eval)$$).variables.addAll(((eval)$2).variables);

// ((eval)$$).variables.addAll(((eval)$5).variables);

((eval)$$).node = new Node(nodeCounter++, ((eval)$2).stmt);//condition expression node

((eval)$$).node.whileID = whileID++;

((eval)$$).cSourceCode += "~WhileID"+((eval)$$).node.whileID+"~while (" + ((eval)$2).cSourceCode + ") { " + ((eval)$5).cSourceCode + ";\n}"+"~ENDWhileID"+((eval)$$).node.whileID+"~";

((eval)$$).nodeIdAndStmt += "while " + "#" + ((eval)$$).node.getNodeID() + ":" + ((eval)$2).stmt + " do \n" + ((eval)$5).nodeIdAndStmt + " done ";

((eval)$$).node.setNodeIdAndStmt(((eval)$$).nodeIdAndStmt);

for(String str : ((eval)$$).variables){

for(Variable v : symbolTableOfVariables){

if(str.equals(v.name)){

((eval)$$).node.addToVariablesOfNode(v);

break;

}

}

}

((eval)$$).list = new MyLinkedList(((eval)$$).node);

Node dummy = new Node(nodeCounter++, "dummy");//dummy node for last node of if

MyLinkedList dummyList = new MyLinkedList(dummy);

((eval)$$).list.getLast().setNextPointer2(dummy);//while condition false

dummy.addPreviousPointer(((eval)$$).list.getLast()); //backward pointer

((eval)$$).list.getNodeSet().add(dummy);

((eval)$$).list.merge(((eval)$5).list); //while condition true (loop)

((eval)$5).list.getLast().setNextPointer1(((eval)$$).list.getFirst());

((eval)$$).list.getFirst().addPreviousPointer(((eval)$5).list.getLast());

((eval)$$).list.setLast(dummy);

};

varlist : x

{

writer.print("\t varlist -> x \n") ;

$$=new eval();

((eval)$$).stmt += ((eval)$1).stmt;

((eval)$$).cSourceCode += ((eval)$1).cSourceCode;

writer.print(((eval)$$).stmt+ "\n");

Variable tempVar = new Variable(((eval)$1).stmt);

boolean flag = true;

for(int i = 0; i < symbolTableOfVariables.size(); i++){

if(symbolTableOfVariables.get(i).name.equals(tempVar.name)){

flag = false;

break;

}

}

if(flag == true){

symbolTableOfVariables.add(tempVar);

((eval)$$).variables.add(((eval)$1).stmt);

}

else{

System.err.println("The variable " + ((eval)$1).stmt + " is already declared!");

GUI.terminal.append("The variable " + ((eval)$1).stmt + " is already declared!", Color.orange);

//System.exit(0);

return -1;

}

};

| x ',' varlist

{

writer.print("\t varlist -> x , varlist \n") ;

$$=new eval();

((eval)$$).stmt += ((eval)$1).stmt + ", " + ((eval)$3).stmt;

((eval)$$).cSourceCode += ((eval)$1).cSourceCode + ", " + ((eval)$3).cSourceCode;

writer.print(((eval)$$).stmt+ "\n");

Variable tempVar = new Variable(((eval)$1).stmt);

boolean flag = true;

for(int i = 0; i < symbolTableOfVariables.size(); i++){

if(symbolTableOfVariables.get(i).name.equals(tempVar.name)){

flag = false;

break;

}

}

if(flag == true){

symbolTableOfVariables.add(tempVar);

((eval)$$).variables.add(((eval)$1).stmt);

((eval)$$).variables.addAll(((eval)$3).variables);

}

else{

System.err.println("The variable " + ((eval)$1).stmt + " is already declared!");

GUI.terminal.append("The variable " + ((eval)$1).stmt + " is already declared!", Color.orange);

}

};

b : BOOL\_CONSTANT

{

writer.print("\t b -> BOOL\_CONSTANT \n") ;

$$=new eval();

((eval)$$).stmt += this.stmt;

((eval)$$).cSourceCode += ((eval)$$).stmt;

writer.print(((eval)$$).stmt+ "\n");

};

n : INTEGER\_NUMBER

{

writer.print("\t n -> INTEGER\_NUMBER \n") ;

$$=new eval();

((eval)$$).stmt += this.stmt;

((eval)$$).cSourceCode += ((eval)$$).stmt;

writer.print(((eval)$$).stmt+ "\n");

};

x : IDENTIFIER

{

writer.print("\t x -> IDENTIFIER \n") ;

$$=new eval();

((eval)$$).stmt += this.stmt;

((eval)$$).cSourceCode += ((eval)$$).stmt;

writer.print(((eval)$$).stmt+ "\n");

};

M : //lambda

{

};

N : //lambda

{

};

%%

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* eval \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class eval {

public String stmt="";

public String nodeIdAndStmt="";

public String cSourceCode = "";

public HashSet<String> variables = new HashSet<String>();

public Node node;

public MyLinkedList list;

}

**PDGBuilder.java:**

**(**کلاس تولیدکننده گراف وابستگی برنامه)

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package wlrewriter;

import java.util.HashSet;

import java.util.Iterator;

import GraphViz.\*;

/\*\*

\*

\* Forward Dominance Tree and Control Dependence Graph Builder Data Dependence

\* Graph and Program Dependence Graph Builder

\*

\* @param CFG

\*

\* @author Mohammad

\*/

public class PDGBuilder {

private MyLinkedList cfg, FDTree, PDG; //FDTree is equal to PostDomTree

private HashSet<Node> FDTNodes;

private String fileName;

// private ArrayList<DataEdge> dataDeps;//data dependencies - by CFG nodes and we just save the relation between them for data dep graph

public PDGBuilder(MyLinkedList cfg) {

this.cfg = cfg;

fileName = YYParser.getSourceCodeFileName().replace(".wl", "");

this.cfg.getFirst().setNextPointer2(this.cfg.getLast()); //according to the article, it needs to connect START to STOP

//print the cfg for test

// cfg.printNodeSet();

//computing post dominators

computePostDominators();

//print dominators

// printPostDominators();

//draw the FDT (Post Dom Tree) - Find the immediate post dominance

makePostDomTree();

//find the immediate post dom for each node - It must be unique, but I did not take risk and get an array for it

setImmediatePostDoms();

//print the immediate post dom for each node that they are set in the field of each node

// printImmediatePostDoms();

//compute PDF for each node that they will be control dependences

computePDFs();

//print the PDFs for all nodes in FDTNodes

// printPDFs();

//compute Control Dependecies from PDFs

computeControlDep();

//print the Control Dependecies

printControlDeps();

//compute the Data Dependecies from CFG

computeDataDep();

//print the Data Dependecies

printDataDeps();

//merge CDG and DDG together to make PDG

computePDG();

//print and show the PDG

printPDG();

}

private void computePostDominators() {

for (Node n : cfg.getNodeSet()) {

n.setPostDominators(cfg.getNodeSet()); //PostDom(n) = NodeSet

}

HashSet<Node> workList = new HashSet<Node>();

workList.add(cfg.getLast()); //WorkList = {StopNode}

while (!workList.isEmpty()) {

Node y = workList.iterator().next();

workList.remove(y); // Remove any node Y from Worklist

// New = {y} + intersect PostDom(x): x is in succ(y)

HashSet<Node> New = new HashSet<>();

New.add(y);

if (y.succ().iterator().hasNext()) {

HashSet<Node> intersect = new HashSet<>(y.succ().iterator().next().getPostDominators());

for (Node x : y.succ()) {

intersect.retainAll(x.getPostDominators());

}

New.addAll(intersect);

}

if (!New.equals(y.getPostDominators())) {

y.setPostDominators(New); //PostDom(y) = New

workList.addAll(y.pred()); // for (each z in pred(y)) worklist += {z}

}

}

// you can find post dominators for each node in getPostDominators() method

}

private void printPostDominators() {

System.out.println("\*\*\*\* print Post Dominators \*\*\*\*");

for (Node i : cfg.getNodeSet()) {

System.out.print("Post-Dom(" + i.getNodeID() + "): ");

for (Node alpha : i.getPostDominators()) {

System.out.print(alpha.getNodeID() + ", ");

}

System.out.println("");

}

}

private void makePostDomTree() {

FDTNodes = new HashSet<>();

//copy nodes from CFG to FDTNodes

for (Node p : cfg.getNodeSet()) {

Node n = new Node(p.getNodeID(), p.getStatement());

HashSet<Node> tempPostDom = new HashSet<>(p.getPostDominators());

n.setPostDominators(tempPostDom);

//eliminate the node from post-dominators to search its pred

n.getPostDominators().remove(p);

FDTNodes.add(n);

}

HashSet<MyLinkedList> listOfFDTNodes = new HashSet<>();

for (Iterator<Node> it = FDTNodes.iterator(); it.hasNext();) {

Node temp = it.next();

listOfFDTNodes.add(new MyLinkedList(temp));

}

FDTree = new MyLinkedList(null);

for (Iterator<MyLinkedList> it = listOfFDTNodes.iterator(); it.hasNext();) {

MyLinkedList temp = it.next();

Node tempNode = temp.getFirst();

if (tempNode.getStatement().equals("STOP")) {

FDTree.setFirst(tempNode);

}

else {

if (tempNode.getStatement().equals("START")) {

FDTree.setLast(tempNode);

}

for (Iterator<Node> it2 = cfg.getNodeSet().iterator(); it2.hasNext();) {

Node temp2 = it2.next();

if (tempNode.getPostDominators().equals(temp2.getPostDominators())) {

for (Iterator<MyLinkedList> it3 = listOfFDTNodes.iterator(); it3.hasNext();) {

MyLinkedList temp3 = it3.next();

Node tempNode3 = temp3.getFirst();

if (tempNode3.getNodeID() == temp2.getNodeID()) {

mergeLists(temp3, temp);

}

}

}

}

}

}

// System.out.println("Post-Dom Tree is created");

}

private void mergeLists(MyLinkedList first, MyLinkedList second) {

first.getFirst().addNextPointersForPostDomTree(second.getFirst());

second.getFirst().addPreviousPointer(first.getFirst());

}

private void setImmediatePostDoms() {

for (Node n : FDTNodes) {

for (Node i : n.getPostDominators()) {

boolean flag = true;

for (Node j : n.getPostDominators()) {

if (!j.equals(i)) {

if (j.getPostDominators().contains(i)) {

flag = false;

break;

}

}

}

if (flag == true) {

if (n.getImmediatePostDominator() != null) {

System.err.println("Immediate Post Dom for this node is not unique!!");

}

n.setImmediatePostDominator(i);

for (Node k : cfg.getNodeSet()) {

if (k.getNodeID() == n.getNodeID()) {

k.setImmediatePostDominator(i);

break;

}

}

}

}

}

}

private void printImmediatePostDoms() {

System.out.println("\*\*\*\* print Immediate Post Dominators \*\*\*\*");

for (Node n : FDTNodes) {

System.out.println("Immediate Post Dom for Node " + n.getNodeID() + " --> " + n.getStatement() + ":");

if (!n.getStatement().equals("STOP")) { //because of being null for STOP node

Node q = n.getImmediatePostDominator();

System.out.println("\t" + q.getNodeID() + " --> " + q.getStatement());

}

else {

System.out.println("\t");

}

}

}

private void computePDFs() {

for (Node n : FDTNodes) {

if (n.getNextPointersForPostDomTree().isEmpty()) { //the node is leaf of PostDomTree

HashSet<Node> worklist = new HashSet<>();

worklist.add(n);

while (!worklist.isEmpty()) {

Node x = worklist.iterator().next();

worklist.remove(x);

worklist.addAll(x.pred()); //traverse bottom-up the postDomTree

HashSet<Node> currentPDF = new HashSet<>(); //PDF for node x

///local///

for (Node xInCFG : cfg.getNodeSet()) {

if (xInCFG.getNodeID() == x.getNodeID()) { // Node xInCFG is x in CFG - we need pred() in CFG

if (!xInCFG.pred().isEmpty()) {

for (Node y : xInCFG.pred()) {

if (y.getImmediatePostDominator() != null) {

if (y.getImmediatePostDominator().getNodeID() != xInCFG.getNodeID()) {

currentPDF.add(y);

}

}

else {

currentPDF.add(y);

}

}

}

break;

}

}

///up///

for (Node z : FDTNodes) {

if (z.getImmediatePostDominator() != null) {

if (z.getImmediatePostDominator().getNodeID() == x.getNodeID()) {

for (Node y : z.getPDF()) {

if (y.getImmediatePostDominator() != null) {

if (y.getImmediatePostDominator().getNodeID() != x.getNodeID()) {

currentPDF.add(y);

}

}

else {

currentPDF.add(y);

}

}

}

}

}

x.setPDF(currentPDF);

}

}

}

}

private void printPDFs() {

System.out.println("\*\*\*\* print FDTs \*\*\*\*");

for (Node n : FDTNodes) {

System.out.println("PDF(" + n.getNodeID() + ": " + n.getStatement() + ") = {");

for (Node q : n.getPDF()) {

System.out.println("\t" + q.getNodeID() + ": " + q.getStatement() + ", ");

}

System.out.println("}");

}

}

private void computeControlDep() {

for (Node y : FDTNodes) {

for (Node x : FDTNodes) {

for (Iterator<Node> it = y.getPDF().iterator(); it.hasNext();) {

Node w = it.next();

if (w.getNodeID() == x.getNodeID()) {

// mitavanad tekrari bashe [dar sorati ke be khodes nabayad vabastegi dashte bashe, in ja ye if mizarim : if(y.getNodeId()!=x.getNodeId())]

x.getContolDep().add(y);

y.getParentOfControlDep().add(x);

}

}

}

}

}

private void printControlDeps() {

String CDgraph = "";

// System.out.println("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\nprint Control Dependencies ");

for (Node n : FDTNodes) {

// System.out.print("Node -->" + n.getNodeID() + ": " + n.getStatement() + " = {");

for (Node q : n.getContolDep()) {

// System.out.print(q.getNodeID() + ": " + q.getStatement() + " | ");

for (Node u : cfg.getNodeSet()) {

if (u.getNodeID() == q.getNodeID()) {

if (u.getVariablesOfNode().size() > 0) {

for (Variable v : u.getVariablesOfNode()) {

if (v.type.equals("high")) {

CDgraph += " \"" + "#" + q.getNodeID() + " " + q.getStatement() + "\"" + ";\n";

}

}

}

if (u.getAssignedVariable() != null) {

if (u.getAssignedVariable().type.equals("high")) {

CDgraph += " \"" + "#" + q.getNodeID() + " " + q.getStatement() + "\"" + ";\n";

}

}

}

}

CDgraph += "\"" + "#" + n.getNodeID() + " " + n.getStatement() + "\"" + " -> " + "\"" + "#" + q.getNodeID() + " " + q.getStatement() + "\"" + ";\n";

//control dep edges: solid // for data dep edges: " [style=dashed];\n";

}

// System.out.println("}");

}

// System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

GraphDrawer gd = new GraphDrawer();

gd.draw(fileName + "\_CDG.", CDgraph);

System.out.println("Control Dependence Graph is ready.");

GUI.terminal.append("Control Dependence Graph is ready.");

}

private void computeDataDep() {

for (Node n : cfg.getNodeSet()) {

if (n.getAssignedVariable() != null) {

HashSet<Node> dataDepLimit = new HashSet<>();

for (Node zed : cfg.getNodeSet()) {

zed.isVisited = false;

if (zed.getAssignedVariable() != null) {

if (zed.getAssignedVariable().name.equals(n.getAssignedVariable().name) && zed.getNodeID() != n.getNodeID()) {

dataDepLimit.add(zed);

}

}

}

HashSet<Node> worklist = new HashSet<>();

worklist.add(n);

while (!worklist.isEmpty()) {

Node node = worklist.iterator().next();

worklist.remove(node);

node.isVisited = true;

boolean isLimit = false;

for (Node h : dataDepLimit) {

if (h.getNodeID() == node.getNodeID()) {

isLimit = true;

for (Variable varc : node.getVariablesOfNode()) {

if (n.getAssignedVariable().name.equals(varc.name)) {

n.getDataDepsForThisNode().add(node);

node.getParentsOfDataDep().add(n);

break;

}

}

break;

}

}

if (!isLimit) {

for (Variable v : node.getVariablesOfNode()) {

if (v.name.equals(n.getAssignedVariable().name)) {//Is assigned variable used in this node?

n.getDataDepsForThisNode().add(node);

node.getParentsOfDataDep().add(n);

break;

}

}

for (Node w : node.succ()) {

if (!w.isVisited) {

worklist.add(w);

}

}

}

}

}

}

}

private void printDataDeps() {

String DDgraph = "";

for (Node n : cfg.getNodeSet()) {

for (Node q : n.getDataDepsForThisNode()) {

DDgraph += "\"" + "#" + n.getNodeID() + " " + n.getStatement() + "\"" + " -> " + "\"" + "#" + q.getNodeID() + " " + q.getStatement() + "\"" + ";\n";

}

}

GraphDrawer gd = new GraphDrawer();

gd.draw(fileName + "\_DDG.", DDgraph);

System.out.println("Data Dependence Graph is ready.");

GUI.terminal.append("Data Dependence Graph is ready.");

}

private void computePDG() {

PDG = cfg; //PDG is equal to CFG , updating some items in each node:

//Node ID --> FDTNodes

//statement --> FDTNodes

//Control Dep --> FDTNodes

//parentOfContorlDep --> FDTNodes

//dataDepForThisNode --> CFG (PDG)

//parentOfDataDep --> CFG (PDG)

//variablesOfNode --> CFG (PDG)

//assignedVariable --> CFG (PDG)

for (Node nodeInPDG : PDG.getNodeSet()) {

for (Node nodeInFDT : FDTNodes) {

if (nodeInPDG.getNodeID() == nodeInFDT.getNodeID()) {

nodeInPDG.setStatement(nodeInFDT.getStatement());

for (Node temp : nodeInFDT.getContolDep()) {

for (Node temp2 : PDG.getNodeSet()) {

if (temp.getNodeID() == temp2.getNodeID()) {

for (Node temp3 : cfg.getNodeSet()) {

if (temp3.getNodeID() == temp2.getNodeID()) {

// temp2.setNextPointer1(temp3.getNextPointer1());

// temp2.setNextPointer2(temp3.getNextPointer2());

}

}

nodeInPDG.getContolDep().add(temp2);

}

}

}

for (Node temp3 : cfg.getNodeSet()) {

for (Node tempq : nodeInFDT.getParentOfControlDep()) {

if (temp3.getNodeID() == tempq.getNodeID()) {

tempq.setNextPointer1(temp3.getNextPointer1());

tempq.setNextPointer2(temp3.getNextPointer2());

}

}

}

nodeInPDG.setParentOfControlDep(nodeInFDT.getParentOfControlDep());

for (Node temp : nodeInFDT.getDataDepsForThisNode()) {

for (Node temp2 : PDG.getNodeSet()) {

if (temp.getNodeID() == temp2.getNodeID()) {

nodeInPDG.getDataDepsForThisNode().add(temp2);

}

}

}

nodeInPDG.setParentsOfDataDep(nodeInFDT.getParentsOfDataDep());

break;

}

}

}

}

private void printPDG() {

String PDgraph = "";

for (Node n : PDG.getNodeSet()) {

//Control Deps

for (Node w : n.getContolDep()) {

PDgraph += "\"" + "#" + n.getNodeID() + " " + n.getStatement() + "\"" + " -> " + "\"" + "#" + w.getNodeID() + " " + w.getStatement() + "\"" + ";\n";

}

//Data Deps

for (Node q : n.getDataDepsForThisNode()) {

PDgraph += "\"" + "#" + n.getNodeID() + " " + n.getStatement() + "\"" + " -> " + "\"" + "#" + q.getNodeID() + " " + q.getStatement() + "\"" + " [style=dashed];\n";

}

}

GraphDrawer gd = new GraphDrawer();

gd.draw(fileName + "\_PDG.", PDgraph);

System.out.println("Program Dependence Graph is ready.");

GUI.terminal.append("Program Dependence Graph is ready.");

}

public MyLinkedList getPDG() {

return PDG;

}

}



Amirkabir University of Technology   
(Tehran Polytechnic)

Computer and Information Technology Engineering Department

B.Sc. Thesis

Title

A Tool for Rewriting-Based Enforcement of Noninterference in Programs

By

Seyed Mohammad Mehdi Ahmadpanah

Supervisor

Dr. Mehran S. Fallah

September 2015

1. Language-based Security [↑](#footnote-ref-1)
2. Semantics [↑](#footnote-ref-2)
3. Types [↑](#footnote-ref-3)
4. Verification [↑](#footnote-ref-4)
5. Static Analysis [↑](#footnote-ref-5)
6. Compile Time [↑](#footnote-ref-6)
7. Dynamic Analysis [↑](#footnote-ref-7)
8. Run-time [↑](#footnote-ref-8)
9. Data-Flow Analysis [↑](#footnote-ref-9)
10. Type-based Analysis [↑](#footnote-ref-10)
11. Model Checking [↑](#footnote-ref-11)
12. Abstract Interpretation [↑](#footnote-ref-12)
13. Execution Monitors [↑](#footnote-ref-13)
14. Policy [↑](#footnote-ref-14)
15. Noninterference [↑](#footnote-ref-15)
16. Program Rewriting [↑](#footnote-ref-16)
17. Program Dependence Graph [↑](#footnote-ref-17)
18. Source Code [↑](#footnote-ref-18)
19. State [↑](#footnote-ref-19)
20. Property [↑](#footnote-ref-20)
21. Noninterference Security Policy [↑](#footnote-ref-21)
22. Hyperproperty [↑](#footnote-ref-22)
23. Observer [↑](#footnote-ref-23)
24. Progress-Sensitive [↑](#footnote-ref-24)
25. Progress-Insensitive [↑](#footnote-ref-25)
26. Progress Status [↑](#footnote-ref-26)
27. Divergence [↑](#footnote-ref-27)
28. Terminate [↑](#footnote-ref-28)
29. Information Flow [↑](#footnote-ref-29)
30. Explicit [↑](#footnote-ref-30)
31. Implicit [↑](#footnote-ref-31)
32. Assign [↑](#footnote-ref-32)
33. Attacker [↑](#footnote-ref-33)
34. Type System [↑](#footnote-ref-34)
35. Well-typed [↑](#footnote-ref-35)
36. Conservative [↑](#footnote-ref-36)
37. Assignment [↑](#footnote-ref-37)
38. Perl [↑](#footnote-ref-38)
39. JavaScript [↑](#footnote-ref-39)
40. Automaton [↑](#footnote-ref-40)
41. Abstract Events [↑](#footnote-ref-41)
42. Syntactic [↑](#footnote-ref-42)
43. Undecidable [↑](#footnote-ref-43)
44. Co-recursively Enumerable [↑](#footnote-ref-44)
45. Characterization [↑](#footnote-ref-45)
46. Sound [↑](#footnote-ref-46)
47. Transparent [↑](#footnote-ref-47)
48. Path Conditions [↑](#footnote-ref-48)
49. Corrective Enforcement [↑](#footnote-ref-49)
50. Preorder [↑](#footnote-ref-50)
51. Abstract Syntax [↑](#footnote-ref-51)
52. Boolean [↑](#footnote-ref-52)
53. No Operation [↑](#footnote-ref-53)
54. Low [↑](#footnote-ref-54)
55. High [↑](#footnote-ref-55)
56. Program Dependence Graph [↑](#footnote-ref-56)
57. Statement [↑](#footnote-ref-57)
58. Control Flow Graph [↑](#footnote-ref-58)
59. Path [↑](#footnote-ref-59)
60. Basic Block [↑](#footnote-ref-60)
61. Dummy [↑](#footnote-ref-61)
62. Parse Tree [↑](#footnote-ref-62)
63. Control Dependence Graph [↑](#footnote-ref-63)
64. Forward Dominance Tree [↑](#footnote-ref-64)
65. Post Dominance Tree [↑](#footnote-ref-65)
66. Dominance Tree [↑](#footnote-ref-66)
67. Strictly Dominate [↑](#footnote-ref-67)
68. Dominator [↑](#footnote-ref-68)
69. Immediate dominator [↑](#footnote-ref-69)
70. Post-Dominance Tree [↑](#footnote-ref-70)
71. Post-Dominance Frontier [↑](#footnote-ref-71)
72. Successor [↑](#footnote-ref-72)
73. Diverging points [↑](#footnote-ref-73)
74. Data Dependence Graph [↑](#footnote-ref-74)
75. Progress-Insensitive Non-Interference [↑](#footnote-ref-75)
76. Progress-Sensitive Non-Interference [↑](#footnote-ref-76)
77. Path Conditions [↑](#footnote-ref-77)
78. Incoming Edge [↑](#footnote-ref-78)
79. Execution Conditions [↑](#footnote-ref-79)
80. Backtracking [↑](#footnote-ref-80)
81. Static Single Assignment [↑](#footnote-ref-81)
82. Loop-carried Data Dependency [↑](#footnote-ref-82)
83. Termination Oracle [↑](#footnote-ref-83)
84. Loop Analyzer [↑](#footnote-ref-84)
85. Loop Guards [↑](#footnote-ref-85)
86. Nested Loops [↑](#footnote-ref-86)
87. Waterfall Process Model [↑](#footnote-ref-87)
88. Classic Life Cycle [↑](#footnote-ref-88)
89. Object-Oriented [↑](#footnote-ref-89)
90. Communication [↑](#footnote-ref-90)
91. Planning [↑](#footnote-ref-91)
92. Modeling [↑](#footnote-ref-92)
93. Construction [↑](#footnote-ref-93)
94. Deployment [↑](#footnote-ref-94)
95. Use Case Diagram [↑](#footnote-ref-95)
96. Activity Diagram [↑](#footnote-ref-96)
97. Unified Modeling Language [↑](#footnote-ref-97)
98. Class Diagram [↑](#footnote-ref-98)
99. https://github.com/smahmadpanah/BScProject [↑](#footnote-ref-99)
100. Lexer [↑](#footnote-ref-100)
101. Parser [↑](#footnote-ref-101)
102. Token [↑](#footnote-ref-102)
103. Syntax Errors [↑](#footnote-ref-103)
104. Linked List [↑](#footnote-ref-104)
105. Graphical User Interface [↑](#footnote-ref-105)
106. Mnemonic [↑](#footnote-ref-106)
107. Domain Testing [↑](#footnote-ref-107)
108. Test Case [↑](#footnote-ref-108)