

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ



دانشگاه صنعتی امیرکبیر

(پلی تکنیک تهران)

دانشکده مهندسی کامپیوتر و فن آوری اطلاعات

پایان نامه کارشناسی

گرایش نرم افزار

عنوان

طراحی و پیاده سازی ابزاری به منظور اِعمال خط مشی امنیتی

عدم تداخل مبتنی بر روش بازنویسی برنامه

نگارش

سید محمد مهدی احمدپناه

استاد راهنما

دکتر مهران سلیمان فلاح

شهریور ۱۳۹۴

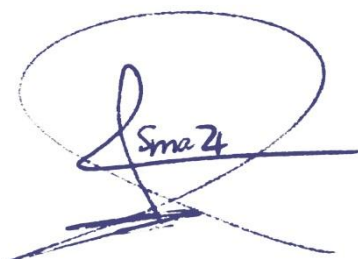
اینجانب سید محمدمهدی احمدپناه متعهد می‌شوم که مطالب مندرج در این پایان نامه حاصل کار پژوهشی اینجانب تحت نظارت و راهنمایی اساتید دانشگاه صنعتی امیرکبیر بوده و به دستاوردهای دیگران که در این پژوهش از آنها استفاده شده است، مطابق مقررات و روال متعارف ارجاع و در فهرست منابع و مآخذ ذکر گردیده است. این پایان نامه قبلاً برای احراز هیچ مدرک هم‌سطح یا بالاتر ارائه نگردیده است.

در صورت اثبات تخلف در هر زمان، مدرک تحصیلی صادر شده توسط دانشگاه از درجه اعتبار ساقط بوده و دانشگاه حق پیگیری قانونی خواهد داشت.

کلیه نتایج و حقوق حاصل از این پایان نامه متعلق به دانشگاه صنعتی امیرکبیر می‌باشد. هرگونه استفاده از نتایج علمی و عملی، واگذاری اطلاعات به دیگران یا چاپ و تکثیر، نسخه‌برداری، ترجمه و اقتباس از این پایان نامه بدون موافقت کتبی دانشگاه صنعتی امیرکبیر ممنوع است. نقل مطالب با ذکر 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 پاک 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 می کند، به رسم ادب دست به قلم برده ام، 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 غفلت های گذشته اند و در تمام عرصه های زندگی یار و یاورم بوده اند؛

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 مهندس بهمن پوروطن که زحمت داوری این پایان نامه را متقبل شدند؛

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

و در پایان، از دوستان عزیزم، محمد پزشکی، بهنام ستارزاده، علی قنبری، مسعود غفاری نیا، احسان عدالت، حمیدرضا رضانی، ریحانه شاه محمدی و پرهام الوانی که در طول انجام پروژه از نظرات و راهنمایی هایشان استفاده کردم؛

کمال تشکر و قدردانی را دارم.

چکیده

خط مشی امنیتی، تعریفی از امن بودن یک سیستم یا برنامه را ارائه می‌دهد که در آن اعمال مجاز و غیرمجاز برای آن سیستم یا برنامه مشخص می‌شود. خط مشی امنیتی مورد نظر در این پروژه، عدم تداخل است. به زبان ساده، خط مشی عدم تداخل بیان می‌کند که یک مشاهده‌گر سطح پایین که فقط به برنامه و مقادیر عمومی زمان اجرا 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، اعمال 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 اعمال 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 دیگر 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 تحلیل گره 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

فصل اول مقدمه	۱
فصل دوم خط مشی امنیتی عدم تداخل و اِعمال آن	۵
۱.۲ خط مشی امنیتی عدم تداخل	۶
۲.۲ مروری بر کارهای گذشته	۸
۳.۲ اِعمال خط مشی عدم تداخل	۹
فصل سوم زبان WL و گراف وابستگی برنامه	۱۳
۱.۳ توصیف زبان برنامه‌نویسی WL	۱۴
۱.۱.۳ نحو زبان	۱۴
۲.۱.۳ معناشناخت زبان	۱۵
۲.۳ گراف وابستگی برنامه	۱۷
۱.۲.۳ وابستگی‌های کنترلی و داده‌ای	۱۷
۲.۲.۳ نحوه تولید گراف وابستگی برنامه	۲۰
فصل چهارم الگوریتم بازنویسی برنامه	۲۷
۱.۴ بازنویسی برای حالت غیرحساس به پیشرفت	۲۹
۲.۴ بازنویسی برای حالت حساس به پیشرفت	۳۳
فصل پنجم پیاده‌سازی و ایجاد رابط کاربری	۳۹
۱.۵ تحلیل و طراحی نرم‌افزار	۴۰
۲.۵ شرح کلی مراحل پیاده‌سازی و ابزارهای مورد استفاده	۴۵
۳.۵ ایجاد رابط کاربری گرافیکی	۴۸
۴.۵ راستی‌آزمایی و آزمون	۵۰
فصل ششم جمع‌بندی و کارهای آینده	۶۱
منابع و مراجع	۶۳
پیوست	۶۵

شکل ۱ - ساختار نحوی زبان برنامه‌نویسی WL	۱۵
شکل ۲ - نمونه کد مبدأ به زبان WL و گراف وابستگی برنامه مربوط به آن	۱۹
شکل ۳ - نمودار کلی نحوه تولید گراف وابستگی برنامه از روی کد مبدأ برنامه [۱۵]	۲۰
شکل ۴ - نحوه تولید زیرگراف بلوک پایه و اتصال به یکدیگر [۱۶]	۲۱
شکل ۵ - نحوه تولید زیرگراف گزاره‌های شرطی - حالت اول [۱۶]	۲۱
شکل ۶ - نحوه تولید زیرگراف گزاره‌های شرطی - حالت دوم [۱۶]	۲۱
شکل ۷ - نحوه تولید زیرگراف گزاره‌های حلقه while [۱۶]	۲۲
شکل ۸ - محاسبه گره‌های غلبه‌کننده برای هر گره [۱۷]	۲۳
شکل ۹ - محاسبه گره‌های پس‌غلبه‌کننده مرزی برای هر گره [۱۷]	۲۵
شکل ۱۰ - الگوریتم کلی بازنویسی برای اعمال خط مشی عدم تداخل [۵]	۲۸
شکل ۱۱ - الگوریتم بازنویسی عدم تداخل حالت غیرحساس به پیشرفت که برنامه M و گراف وابستگی برنامه مربوط به آن G را می‌گیرد [۵]	۳۱
شکل ۱۲ - (الف) نمونه کد به زبان WL (ب) برنامه بازنویسی‌شده برنامه الف در حالت غیرحساس به پیشرفت (پ) گراف وابستگی برنامه الف	۳۲
شکل ۱۳ - (الف) نمونه برنامه به زبان WL (ب) برنامه بازنویسی‌شده الف برای حالت غیرحساس به پیشرفت، که حالت حساس به پیشرفت را برآورده نمی‌کند	۳۳
شکل ۱۴ - برنامه‌ای که حلقه موجود در آن در حالتی که $h1 \geq 11$ or $11 < 0$ باشد، خاتمه خواهد یافت.....	۳۵
شکل ۱۵ - الگوریتم بازنویسی عدم تداخل حالت حساس به پیشرفت که برنامه M و گراف وابستگی برنامه مربوط به آن G را می‌گیرد [۵]	۳۷
شکل ۱۶ - کد مبدأ بازنویسی‌شده توسط الگوریتم حالت حساس به پیشرفت برای برنامه شکل ۱۴	۳۸
شکل ۱۷ - مدل فرآیندی آبشاری [۱۹]	۴۰
شکل ۱۸ - نمودار درخواست سیستم نرم‌افزار پروژه	۴۲
شکل ۱۹ - نمودارهای فعالیت نرم‌افزار پروژه	۴۳
شکل ۲۰ - نمودار کلاس نرم‌افزار پروژه (بدون ذکر فیلدها و متدها)	۴۴
شکل ۲۱ - نمای کلی رابط کاربری گرافیکی نرم‌افزار	۴۹
شکل ۲۲ - نمونه‌ای از اجرای برنامه در رابط کاربری گرافیکی نرم‌افزار	۵۰
شکل ۲۳ - (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	۵۱
شکل ۲۴ - (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	۵۱

- شکل ۲۵- (الف) برنامه مورد آزمون با نام 05if2.wl (ب) برنامه بازنویسی شده برای حالت غیرحساس به پیشرفت مربوط به برنامه الف (پ) گراف وابستگی برنامه مربوط به برنامه الف ۵۲
- شکل ۲۶- (الف) برنامه مورد آزمون با نام 07ifelseadvanced.wl (ب) برنامه بازنویسی شده برای حالت غیرحساس به پیشرفت مربوط به برنامه الف (پ) گراف وابستگی برنامه مربوط به برنامه الف ۵۴
- شکل ۲۷- (الف) برنامه مورد آزمون با نام 11whilewhileconcat.wl (ب) برنامه بازنویسی شده برای حالت غیرحساس به پیشرفت مربوط به برنامه الف (پ) برنامه بازنویسی شده برای حالت حساس به پیشرفت مربوط به برنامه الف (ت) گراف وابستگی برنامه مربوط به برنامه الف ۵۶
- شکل ۲۸- (الف) برنامه مورد آزمون با نام 17whilewhilenested.wl (ب) برنامه بازنویسی شده برای حالت غیرحساس به پیشرفت مربوط به برنامه الف (پ) برنامه بازنویسی شده برای حالت حساس به پیشرفت مربوط به برنامه الف (ت) گراف وابستگی برنامه مربوط به برنامه الف ۵۸

صفحه

فهرست جدول‌ها

جدول ۱ - نمونه ورودی‌ها و خروجی‌ها برای برنامه به زبان C متناظر با برنامه شکل ۲۵- (الف).....	۵۳
جدول ۲ - نمونه ورودی‌ها و خروجی‌ها برای برنامه به زبان C متناظر با برنامه شکل ۲۵- (ب).....	۵۳
جدول ۳ - نمونه ورودی‌ها و خروجی‌ها برای برنامه به زبان C متناظر با برنامه شکل ۲۶- (الف).....	۵۵
جدول ۴ - نمونه ورودی‌ها و خروجی‌ها برای برنامه به زبان C متناظر با برنامه شکل ۲۶- (ب).....	۵۵
جدول ۵ - نمونه ورودی‌ها و خروجی‌ها برای برنامه به زبان C متناظر با برنامه شکل ۲۷- (الف).....	۵۷
جدول ۶ - نمونه ورودی‌ها و خروجی‌ها برای برنامه به زبان C متناظر با برنامه شکل ۲۷- (ب).....	۵۷
جدول ۷ - نمونه ورودی‌ها و خروجی‌ها برای برنامه به زبان C متناظر با برنامه شکل ۲۷- (پ).....	۵۷
جدول ۸ - نمونه ورودی‌ها و خروجی‌ها برای برنامه به زبان C متناظر با برنامه شکل ۲۸- (الف).....	۵۹
جدول ۹ - نمونه ورودی‌ها و خروجی‌ها برای برنامه به زبان C متناظر با برنامه شکل ۲۸ شکل ۲۷- (ب).....	۵۹
جدول ۱۰ - نمونه ورودی‌ها و خروجی‌ها برای برنامه به زبان C متناظر با برنامه شکل ۲۸ شکل ۲۷- (پ).....	۵۹

فصل اول

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 عادی سیستم‌ها نیز حائز اهمیت است.

یکی از زمینه‌های مطرح در امنیت اطلاعات و ارتباطات، امنیت برنامه‌های کاربردی و به تبع آن، امنیت زبان‌های برنامه‌نویسی یا امنیت زبان مبنا^۱ می‌باشد. امنیت زبان مبنا را می‌توان مجموعه‌ای از تکنیک‌های مبتنی بر نظریه زبان‌های برنامه‌سازی و پیاده‌سازی آن‌ها شامل معناشناخت^۲، نوع‌ها^۳، بهینه‌سازی و راستی‌آزمایی^۴ برای به‌کارگیری در مسائل امنیتی تعریف کرد. [۱] تلاش این حوزه بر آن است که برنامه‌های کاربردی 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 خط‌مشی‌های امنیتی وجود دارد که به طور کلی می‌توان به دو دسته روش‌های تحلیل ایستا^۵ یا زمان‌کامپایل^۶ و تحلیل پویا^۷ یا زمان‌اجرا^۸ دسته‌بندی کرد. تحلیل جریان داده^۹، تحلیل نوع مبنا^{۱۰}، واریسی

¹ Language-based Security

² Semantics

³ Types

⁴ Verification

⁵ Static Analysis

⁶ Compile Time

⁷ Dynamic Analysis

⁸ Run-time

مدل^{۱۱} و تفسیر انتزاعی^{۱۲} نمونه‌هایی از انواع تحلیل‌های ایستا هستند. انواع مختلف ناظران اجرا^{۱۳} 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^{۱۴} امنیتی، یکی از چالش‌های پیش روی متخصصان این حوزه می‌باشد. چنان‌که نحوه و رویکرد اعمال 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 امنیتی عدم تداخل^{۱۵} به عنوان نیازمندی امنیتی در نظر گرفته می‌شود و برای اعمال این خط مشی در برنامه‌ها، از یکی از روش‌های تحلیل ایستا؛ یعنی روش بازنویسی برنامه^{۱۶} استفاده می‌شود که در ادامه این پایان‌نامه، 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^{۱۷} و کاربرد آن در پروژه می‌باشد. فصل چهارم به توضیح الگوریتم مورد نظر برای بازنویسی کد مبدأ^{۱۸} در دو حالت خط مشی عدم تداخل؛ یعنی حالت

⁹ Data-Flow Analysis

¹⁰ Type-based Analysis

¹¹ Model Checking

¹² Abstract Interpretation

¹³ Execution Monitors

¹⁴ Policy

¹⁵ Noninterference

¹⁶ Program Rewriting

¹⁷ Program Dependence Graph

¹⁸ Source Code

غیر حساس 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

۱.۲ خط مشی امنیتی عدم تداخل

به طور کلی، خط مشی امنیتی، امن بودن یک سیستم یا برنامه را تعریف می‌کند که در آن اعمال مجاز و غیرمجاز در آن سیستم یا برنامه مشخص می‌شود. خط مشی امنیتی، قیود روی توابع و جریان‌های بین آن‌ها را مشخص می‌کند؛ مثل قیود دسترسی بر روی برنامه‌ها و سطوح دسترسی 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 توانی همه اجراها تعریف کرد که هر اجرا یک دنباله دلخواه از حالت^{۱۹}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^{۲۰} 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 این گونه نیازمندی‌ها، خط مشی امنیتی عدم تداخل^{۲۱} است. [۲] عدم تداخل گوگن-میسگر [۳] مثالی از خط مشی‌هایی است که نمی‌توان آن‌ها را در قالب خاصیت بیان کرد و اصطلاحاً به آن‌ها فوق خاصیت^{۲۲} گفته می‌شود [۴]. نکته حائز اهمیت این است که روش اعمال خاصیت‌ها با نحوه اعمال خط مشی‌هایی که خاصیت نیستند، متفاوت است.

خط مشی عدم تداخل بیان می‌کند که یک مشاهده‌گر^{۲۳} سطح پایین که فقط به برنامه و مقادیر عمومی زمان اجرا دسترسی دارد، نتواند ورودی‌های سطح بالا یا خصوصی برنامه را بفهمد. به عبارت

¹⁹ State

²⁰ Property

²¹ Noninterference Security Policy

²² Hyperproperty

²³ Observer

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 اعمال این خط مشی در برنامه‌ها می‌شود.

خط مشی عدم تداخل را می‌توان به دو دسته حساس به پیشرفت^{۲۴} و غیر حساس به پیشرفت^{۲۵} تقسیم کرد. [۵] در عدم تداخل غیر حساس به پیشرفت، مشاهده‌گر سطح پایین تنها می‌تواند خروجی‌های میانی سطح پایین برنامه را ببیند. این در حالیست که یک مشاهده‌گر سطح پایین در عدم تداخل حساس به پیشرفت، علاوه بر دسترسی‌های قبلی، به وضعیت پیشرفت^{۲۶} برنامه نیز دسترسی دارد. این باعث می‌شود تا بتواند تفاوت بین واگرایی^{۲۷} برنامه با موقعیتی که برنامه خاتمه^{۲۸} می‌یابد یا در حال محاسبه مقادیر قابل مشاهده 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 اطلاعات^{۲۹} از سطح بالا به پایین، صریح^{۳۰} یا ضمنی^{۳۱} باشد. انتساب^{۳۲} یک مقدار سطح بالا به یک متغیر سطح پایین، نمونه‌ای از جریان اطلاعات صریح است. همچنین، جریان از بالا به پایین در زمانی که مقدار یک متغیر

²⁴ Progress-Sensitive

²⁵ Progress-Insensitive

²⁶ Progress Status

²⁷ Divergence

²⁸ Terminate

²⁹ Information Flow

³⁰ Explicit

³¹ Implicit

³² Assign

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 کاربران سطح پایین بیانگر مدل مهاجم^{۳۳} 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.

۲.۲ مروری بر کارهای گذشته

وُلپانو و اسمیت [۲] نشان می‌دهند که در صورتی که یک برنامه در نوع سامانه^{۳۴} پیشنهادی آن‌ها خوش‌نوع^{۳۵} 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 داشته‌که به طور کلی، ذات نوع سامانه‌ها محافظه‌کارانه^{۳۶} است.

وَنگَتَکْرِشَن و همکارانش [۶] یک روش تبدیل برنامه‌ی ترکیبی برای اعمال عدم تداخل ارائه داده‌اند که از روش‌های زمان‌اجرا برای صحت و از روش‌های تحلیل ایستا برای کامل بودن روش خود استفاده کرده‌اند. برنامه تغییر داده شده، سطوح امنیتی انتساب^{۳۷} را پیگیری می‌کند و زمانی که یک جریان غیرمجاز در حال وقوع باشد، خاتمه می‌یابد. این روش، تنها در صورت‌بندی‌هایی قابل استفاده است که از عدم تداخل بدون توجه به رفتار خاتمه‌ی برنامه‌ها مطرح می‌شود و جریان‌های ضمنی که بدون انتساب ایجاد می‌شوند را تشخیص نمی‌دهد.

³³ Attacker

³⁴ Type System

³⁵ Well-typed

³⁶ Conservative

³⁷ Assignment

مگزینوس و همکارانش [۷] یک چارچوب برای ناظرهای امنیتی پویای درون برنامه‌ای ساخته‌اند. این روش، عدم تداخل غیر حساس به خاتمه را تضمین می‌کند و قابل به‌کارگیری در زبان‌های پزل^{۳۸} و جاوا اسکریپت^{۳۹} است که از ارزیابی پویای کد پشتیبانی می‌کنند. ضمناً در این روش لازم است که تغییردهنده‌ی برنامه در زمان اجرا در دسترس باشد که یک ناظر مناسب بتواند در کدی که به صورت پویا تولید می‌شود، ورود کند.

لگوژنیک و همکارانش [۸] یک ماشین^{۴۰} طراحی کرده‌اند که رخدادهای انتزاعی^{۴۱} در زمان اجرا را دریافت می‌کند و اجرا توسط بعضی از اطلاعات ایستا ویرایش می‌شود. اشکال این روش آن جاست که اجازه وقوع کانال‌های خاتمه را نیز می‌دهد.

این موضوع اثبات شده است که هیچ روش کاملاً پویایی برای اعمال عدم تداخل حساس به جریان وجود ندارد. [۹] 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 نحوی^{۴۲} بر روی کد دارند، از اطلاعات ایستا در ناظری بر اجراهای چندگانه‌ی برنامه‌ها استفاده کنند.

۳.۲ اعمال خط مشی 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^{۴۳} است. [۲] پس در حالت کلی، عدم تداخل توسط روش‌های ایستا قابل

³⁸ Perl

³⁹ JavaScript

⁴⁰ Automaton

⁴¹ Abstract Events

⁴² Syntactic

⁴³ Undecidable

اعمال نیست؛ به همین دلیل است که نوع سامانه‌های ارائه شده برای این مسئله، محافظه کارانه هستند و ممکن است بعضی برنامه‌های امن را نیز نپذیرند. از طرفی، این مسئله مکمل شمارش پذیر بازگشتی^{۴۴} نیز نیست. [۱۰] بنابراین، توسط ناظرهای اجرا که نقض 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. [۱] 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^{۴۵} 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. [۵] از این رو، می‌توان روش بازنویسی برنامه را روشی بین روش‌های ایستا و روش‌های پویا دانست.

⁴⁴ Co-recursively Enumerable

⁴⁵ Characterization

از طرفی، بازنویسی برنامه تغییراتی به ذات 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^{۴۶} و شفاف^{۴۷} باشد. صحت به این معنا که کد تولید شده توسط آن، خط مشی را برآورده کند و شفاف بودن به معنای این که معناساخت 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 ممکن برنامه تبدیل شده، خواه امن یا ناامن، مشابه برنامه ورودی باشد. البته تا زمانی که 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 شفافیت بیشتر در الگوریتم بازنویسی، باعث برتری آن روش بر دیگر بازنویس‌های مشابه خواهد بود.

روش بازنویسی برنامه مورد استفاده در این پروژه، از گراف وابستگی برنامه [۱۱] استفاده می‌کند که پیشتر اثبات شده است که در تشخیص جریان‌های اطلاعاتی احتمالی، دست‌کم به قدرت نوع سامانه‌های امنیتی است. [۱۲] 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 -چالش برانگیزترین نیازمندی برای یک مکانیزم اعمال کارا- است. در این روش، جریان‌های غیرمجاز صریح و ضمنی کد داده شده، با اصلاح کد برطرف می‌شود. این اصلاح همان قدر کاراست که در صورت وقوع آن‌ها در زمان اجرا، جلوی آن‌ها گرفته می‌شود. همچنین در روش مورد استفاده، از شرط‌های مسیر^{۴۸} 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

⁴⁶ Sound

⁴⁷ Transparent

⁴⁸ Path Conditions

به گونه‌ای است که مجموعه اجراهای ممکن برنامه تبدیل شده، به شباهت مجموعه اجراهای ممکن برنامه ورودی باشد. می‌توان اثبات کرد که روش مورد استفاده در این پروژه صحیح و شفاف است [۵]، پس قطعاً برنامه‌هایی که توسط آن تولید می‌شوند، امن خواهند بود. ضمناً برای اثبات شفافیت الگوریتم، از بیان دقیق و صوری مفهوم اعمال اصلاحی^{۴۹} و رابطه پیش‌ترتیبی^{۵۰} استفاده شده است.

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

⁴⁹ Corrective Enforcement

⁵⁰ Preorder

فصل سوم

زبان WL و گراف وابستگی برنامه

۱.۳ توصیف زبان برنامه‌نویسی WL

برای تعریف دقیق و صوری خط مشی 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. [۵] 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 [۱۳] و bison [۱۴] 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 معناساخت می‌پردازیم.

۱.۱.۳ نحو زبان

در شکل ۱، ساختار نحوی تجربیدی^{۵۱} زبان 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^{۵۲}، یک متغیر عددی و یا یک عملیات یا ارتباط بین یک یا چند عبارت دیگر باشد.

مجموعه دستورات، شامل NOP برای دستور عملیات هیچ^{۵۳}، $x = \text{exp}$ برای انتساب یک عبارت به یک متغیر، $\text{inL varlist, inH varlist}$ و outL x و outH x برای ورودی گرفتن و خروجی دادن هستند که پسوندهای L و H، به ترتیب، به معنای سطح امنیتی پایین^{۵۴} یا بالا^{۵۵} در دستورات ورودی و خروجی می‌باشند. دستور outL BOT یا outH BOT، دستورات خروجی منحصر به فردی هستند که مقدار ثابت

⁵¹ Abstract Syntax

⁵² Boolean

⁵³ No Operation

⁵⁴ Low

⁵⁵ High

BOT یا \perp 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 ۱ - 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 [۵] یافته است.

۲.۱.۳ 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 تعریف شده ذخیره می شود و با توجه به دستورات بعدی برنامه، مقادیر آنها تغییر می کند. در این زبان، این فرض صورت گرفته است که فقط در ابتدای کد مبدأ برنامه، دستورات گرفتن ورودی های مورد نظر نوشته خواهد شد؛ گرچه دستورات خروجی در هر نقطه ای از برنامه می توانند وجود داشته باشند. نکته مهم در خصوص زبان برنامه نویسی WL این است که در برنامه های نوشته شده به این زبان، مقادیر در هر زمان دلخواه در طول زمان اجرا می توانند 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 حالت های نهایی امکان پذیر است.

عملگرهای and و or، همان عملگرهای دو عملوندی منطقی عطف و فصل هستند. عملگر تک عملوندی !، برای نقیض یک عبارت منطقی استفاده می شود. 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 شناسه های مورد استفاده برای نام متغیرها است که با یک حرف انگلیسی آغاز و پس از آن تعداد محدودی عدد یا حرف می آید.

منظور از دستور NOP، همان دستور ; در زبان C می باشد که بدون انجام عملیاتی، تأخیری در اجرای برنامه ایجاد می کند. برای دستورات شرطی، مشابه زبان های برنامه نویسی رایج، در صورت برقراری عبارت شرطی، دستورات پس از پایانه then تا else اجرا می شوند 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 می شوند.

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

با مشاهده مثال های موجود در پایان نامه، می توان درک بهتری نسبت به معناساخت این زبان پیدا کرد. همچنین، معناساخت صوری این زبان نیز در مقاله [۵] موجود است.

۲.۳ 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 مکانیزم اعمال خط مشی عدم تداخل، به ماشینی برای تشخیص جریان‌های اطلاعات ممکن از ورودی‌های سطح بالا به خروجی‌های سطح پایین نیاز است. گراف‌های وابستگی برنامه یا به اختصار ^{۵۶}PDG 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 گزاره‌های ^{۵۷} 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 ^{۵۸} 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.

۱.۲.۳ وابستگی‌های کنترلی و داده‌ای

در گراف وابستگی برنامه، یک یال وابستگی داده‌ای از گره X به گره Y ، که با $X \xrightarrow{d} Y$ 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 $X \xrightarrow{c} Y$ نمایش داده می‌شود، به معنای این است که اجرای

^{۵۶} Program Dependence Graph

^{۵۷} Statement

^{۵۸} Control Flow Graph

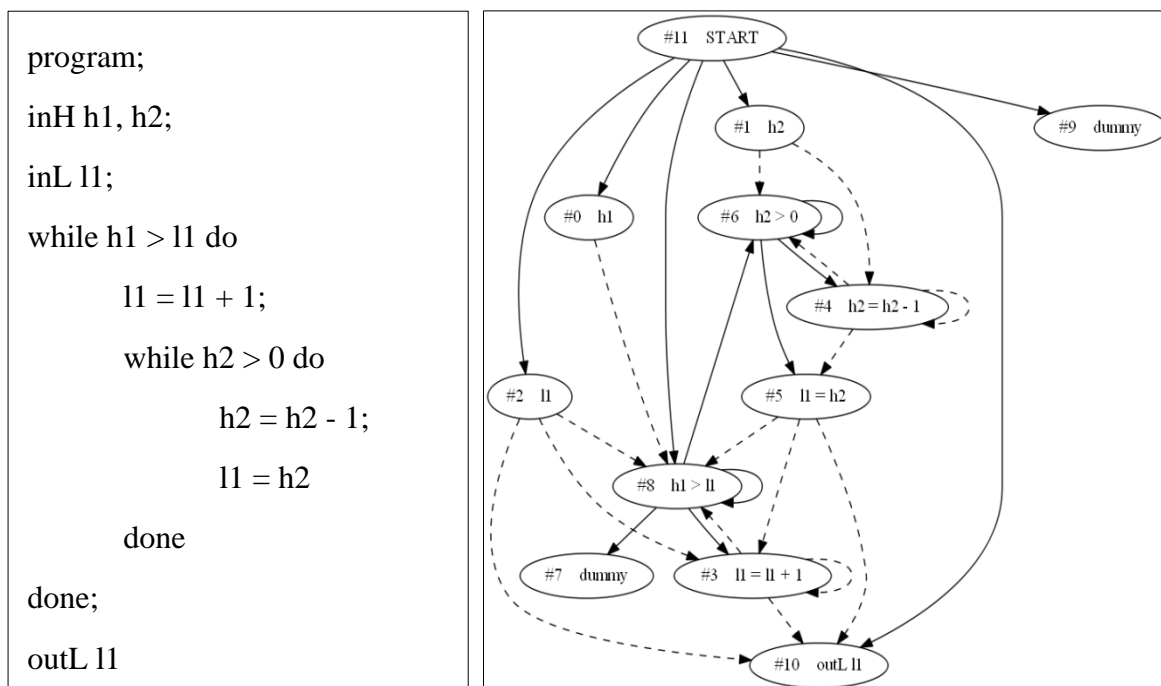
گزاره Y ، توسط مقدار محاسبه شده در گزاره X کنترل می شود. ضمناً می توان یک مسیر^{۵۹} از X به Y در گراف وابستگی برنامه را به شکل $X \rightsquigarrow Y$ علامت گذاری کرد. هر مسیر مشخص کننده یک وابستگی کنترلی یا داده ای است که بستگی به نوع آخرین یال آن مسیر دارد. در ضمن، مسیر ساخته شده به واسطه اضافه کردن یال $X \rightarrow Y$ - که به این معناست که نوع وابستگی آن اهمیتی ندارد - به مسیر $Y \rightsquigarrow Z$ ، به شکل $X \rightarrow Y \rightsquigarrow Z$ نمایش داده می شود. مسیری مانند $X \rightsquigarrow Y$ در گراف وابستگی برنامه بیانگر این است که ممکن است جریانی از X به Y وجود داشته باشد.

می توان چنین تعریف کرد که اگر مقدار محاسبه شده در Y یا صرف اجرای Y به مقدار محاسبه شده 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 حالی که عکس آن لزوماً صحیح نیست.

همان طور که در قبل مطرح شد، می توان جریان از X به Y بر روی مسیر $X \rightsquigarrow Y$ را به دو نوع صریح و ضمنی دسته بندی کرد. یک جریان صریح زمانی برقرار است که مقدار محاسبه شده در X ، مستقیماً به گره Y منتقل شود. این جریان می تواند ناشی از زنجیره انتساب های روی آن مسیر باشد. همچنین، یک جریان ضمنی زمانی برقرار خواهد بود که مقدار محاسبه شده در Y ، به اجرا شدن یا نشدن یک گزاره خاص در مسیر $X \rightsquigarrow Y$ وابسته باشد و اجرای آن گزاره، توسط مقدار محاسبه شده در X کنترل شود.

پس با تعاریف فوق می توان چنین گفت که مسیر $X \rightsquigarrow Y$ روی گراف وابستگی برنامه تعیین کننده یک جریان صریح از X به Y است اگر همه یال های موجود در مسیر، از نوع وابستگی داده ای باشند. در غیر این صورت، آن مسیر به یک جریان ضمنی دلالت خواهد داشت.

⁵⁹ Path



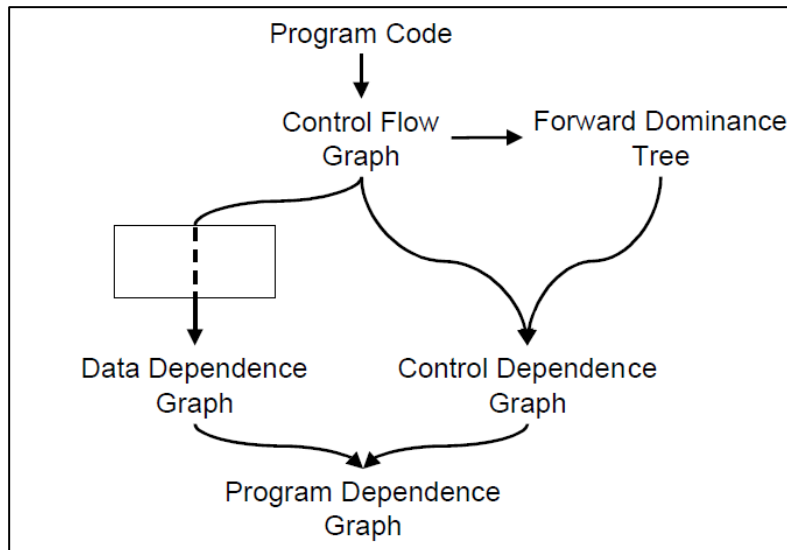
شکل ۲ - نمونه کد مبدأ به زبان WL و گراف وابستگی برنامه مربوط به آن

همان طور که در شکل ۲ مشاهده می شود، یال های با خطوط ساده نماد وابستگی های کنترلی و یال های خط چین نمایانگر وابستگی های داده ای هستند.

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

۲.۲.۳ نحوه تولید گراف وابستگی برنامه

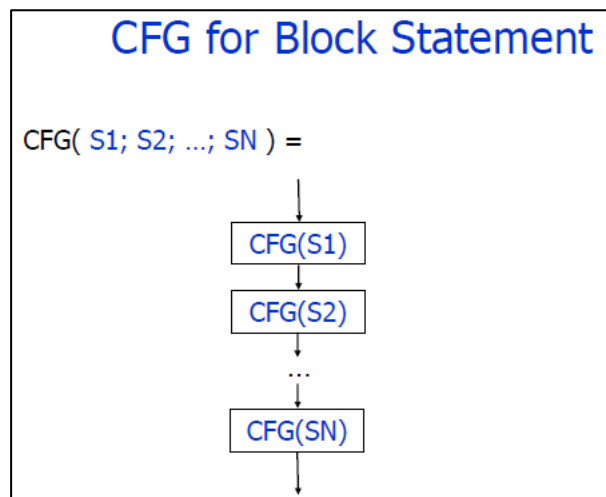
برای ساخت گراف وابستگی برنامه، مطابق با شکل ۳، گراف‌های مورد نیاز برای تحلیل ساخته می‌شود.



شکل ۳ - نمودار کلی نحوه تولید گراف وابستگی برنامه از روی کد مبدأ برنامه [۱۵]

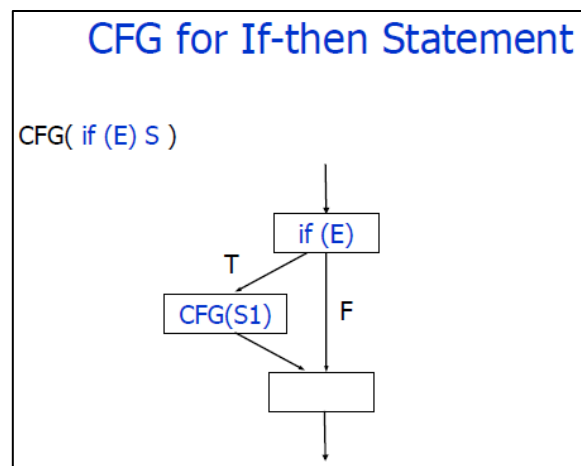
ابتدا از روی کد مبدأ، گراف جریان کنترل یا CFG به دست می‌آید. نحوه تولید این گراف به این نحو است که گزاره‌ها و عبارت‌های برنامه، به عنوان یک گره در گراف در نظر گرفته می‌شود. در این گراف، مفهومی به نام بلوک پایه^{۶۰} مطرح می‌شود. هر بلوک پایه، شامل تعدادی گره است که تنها یک گره ورودی و یک گره خروجی در آن وجود دارد. به این منظور، برای گزاره‌های ساده که اجرای آن‌ها مشروط نیست، به صورت دنباله پشت سر همی از گره‌ها در نظر گرفته می‌شود. پس برای این گونه گزاره‌ها و عبارت‌ها، تنها ساختن یک گره جدید و متصل کردن آن‌ها به گراف کفایت می‌کند. شکل ۴، 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.

⁶⁰ Basic Block

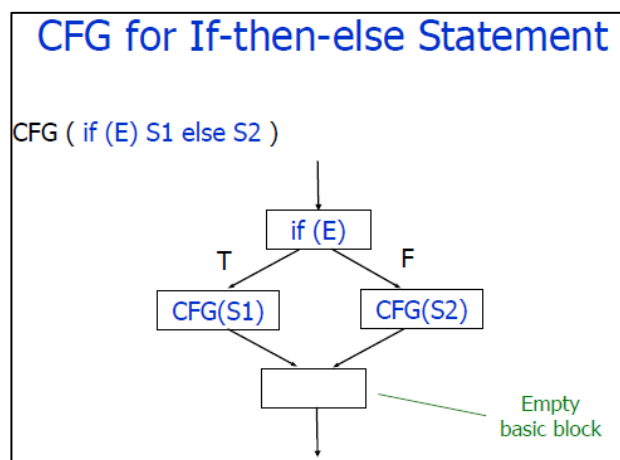


شکل ۴ - نحوه تولید زیرگراف بلوک پایه و اتصال به یکدیگر [۱۶]

برای گزاره‌های شرطی، دو حالت ممکن زیر وجود دارد:



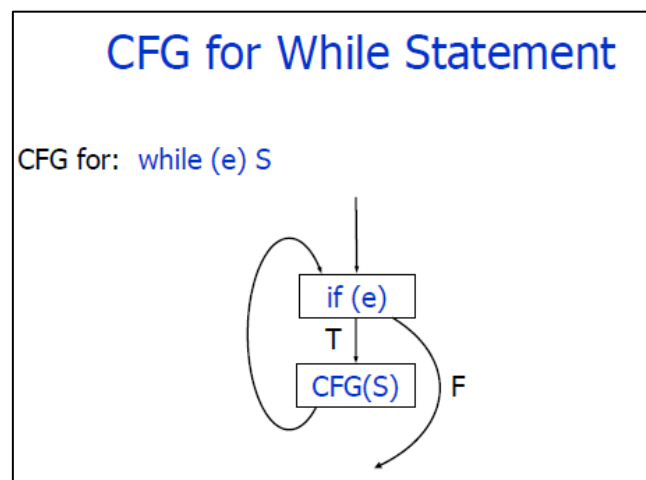
شکل ۵ - نحوه تولید زیرگراف گزاره‌های شرطی - حالت اول [۱۶]



شکل ۶ - نحوه تولید زیرگراف گزاره‌های شرطی - حالت دوم [۱۶]

مطابق با شکل ۵ و شکل ۶، 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^{۶۱} 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 ۷ عمل می‌شود.



شکل ۷ - نحوه تولید زیرگراف گزاره‌های حلقه while [۱۶]

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^{۶۲} ساخته می‌شوند و با اتصال آن‌ها به یکدیگر، گراف کنترل جریان برنامه به دست خواهد آمد.

⁶¹ Dummy

⁶² Parse Tree

مطابق با شکل ۳، برای تولید گراف وابستگی کنترل یا به اختصار CDG^{63} ، به درخت غلبه رو به جلو⁶⁴ یا درخت پس غلبه⁶⁵ نیاز خواهد بود. برای تولید این درخت، الگوریتم ساخت درخت غلبه⁶⁶ بر روی معکوس 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 گره آغازین تا گره N ، حتماً و الزاماً از گره M بگذرند. همچنین، گره M بر گره N اکیداً غلبه⁶⁷ 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 غلبه‌کننده⁶⁸ داشته باشد، اما برای تولید درخت غلبه، نزدیک‌ترین غلبه‌کننده یا غلبه‌کننده بدون واسطه⁶⁹ 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 ≠ ∅) {
    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 ۸ - محاسبه گره‌های غلبه‌کننده برای هر گره [۱۷]

⁶³ Control Dependence Graph

⁶⁴ Forward Dominance Tree

⁶⁵ Post Dominance Tree

⁶⁶ Dominance Tree

⁶⁷ Strictly Dominate

⁶⁸ Dominator

⁶⁹ Immediate dominator

سپس با توجه به مجموعه غلبه‌کنندگان به دست آمده از الگوریتم بالا و مقایسه با مجموعه غلبه‌کنندگان سایر گره‌ها، می‌توان گره غلبه‌کننده بی‌درنگ را یافت و درخت غلبه را تشکیل داد اما در این‌جا، تولید درخت پس‌غلبه^{۷۰} 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 وابستگی کنترلی، مرزهای پس‌غلبه^{۷۱} یا اختصاراً PDF، مورد نیاز است. مرز پس‌غلبه گره X ، مجموعه گره‌هایی هستند که توسط X اکیداً پس‌غلبه نمی‌شوند اما گره‌های مابعدی^{۷۲} دارند که توسط X پس‌غلبه می‌شوند. تعریف دقیق‌تر این گره‌ها بدین شرح است:

$PDF(X) = \{y \mid (\exists z \in Succ(y) \text{ such that } x \text{ post-dominates } z) \text{ and } x \text{ does not strictly post-dominate } y\}$

که این مجموعه بیانگر نزدیک‌ترین نقاط انشعابی^{۷۳} 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. [۱۷]

حال با استفاده از الگوریتم شکل ۹، می‌توان مجموعه مرزهای پس‌غلبه هر گره را به دست آورد که بیانگر وابستگی‌های کنترلی نیز خواهند بود.

⁷⁰ Post-Dominance Tree

⁷¹ Post-Dominance Frontier

⁷² Successor

⁷³ Diverging points

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

$$PDF(X) = \phi$$

Step 1: For each y in Predecessor(X) do

If X is not immediate post-dominator of y then

$$PDF(X) \leftarrow PDF(X) \cup \{y\}$$

Step 2: For each z that x immediately post-dominates, do

For each $y \in PDF(Z)$ do

If X is not immediate post-dominator of y then

$$PDF(X) \leftarrow PDF(X) \cup \{y\}$$

شکل ۹- محاسبه گره‌های پس‌غلبه‌کننده مرزی برای هر گره [۱۷]

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 DDG^{74} 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.

⁷⁴ Data Dependence Graph

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

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 $PINI^{75}$ و حالت حساس به پیشرفت یا $PSNI^{76}$ 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 و گزاره‌ها 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 قابل مشاهده برای کاربر سطح پایین در این پروژه، مطابق با بیان صوری مطرح شده در مقاله [۵] است.

foreach statement X producing a high input event h_{in} **do**

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

if $Y \in affect(X)$ **then**

 transform Y into Y' such that $Y' \notin affect(X)$ in the new program

end

end

end

شکل ۱۰ - الگوریتم کلی بازنویسی برای اعمال خط مشی عدم تداخل [۵]

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

⁷⁵ Progress-Insensitive Non-Interference

⁷⁶ Progress-Sensitive Non-Interference

ندهند. بنابراین در الگوریتم شکل ۱۰، باید شرط‌هایی که تعیین‌کننده وقوع جریان غیرمجاز احتمالی هستند را لحاظ کرد.

۱.۴ بازنویسی برای حالت غیر حساس به پیشرفت

ایده اصلی استفاده شده این است که دستورهای $outL$ ی که از ورودی‌های سطح بالا متأثر شده‌اند، با گزاره‌های $outL \perp$ 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^{۷۷} استفاده شده است. در مقاله [۵]، شرط مسیر $p(X,Y)$ روی متغیرهای برنامه تعریف می‌شود و همان شرط‌هایی هستند که باعث می‌شوند تا جریان $X \rightsquigarrow Y$ 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 حالی که 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 مسیر با یال وارد شونده^{۷۸} وابستگی کنترلی، به خاطر مقدار عبارت کنترلی اجرا نشود. پس اجرای همه نودهای روی مسیر تعیین‌کننده یک جریان ضمنی، برای وقوع آن جریان الزامی نیست. به همین ترتیب، جریان مربوط به مسیر $inH h \rightsquigarrow outL l$ روی گراف وابستگی برنامه مربوط به یک برنامه به زبان WL فقط زمانی رخ می‌دهد که همه گره‌های آن مسیر که یال وارد شونده وابستگی داده‌ای دارند، اجرا شوند. می‌توان چنین گفت که همه گره‌های میانی روی مسیر بیان‌کننده یک جریان 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

⁷⁷ Path Conditions

⁷⁸ Incoming Edge

استفاده شده در این پروژه بررسی می کنند که آیا همه گره های میانی با یال وارد شونده وابستگی داده ای بر روی مسیری که به دستورات outL 1 ختم می شوند، در طول برنامه اجرا شوند یا خیر. اگر مسیر چنین باشد، outL \perp یا NOP به جای آن دستور اجرا خواهد شد، وگرنه خود دستور outL 1 به اجرا در می آید.

برنامه های نوشته شده به زبان WL حاوی شرط های مسیر ساده هستند که از شرط های اجرای⁷⁹ گره ها به دست می آیند. به طور کلی، شرط اجرا برای گره X به کمک پیمایش معکوس⁸⁰ با شروع از گره X تا گره آغازین، از طریق یال های وابستگی های کنترلی روی مسیر حاصل می شود. 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 $X \rightsquigarrow Y$ ، ترکیب عطفی شرط های اجرای گره های روی آن مسیر تعریف می شود. شرط های مسیر را می توان بر اساس ترکیب عطفی گره های روی مسیر با یک یال وارد شونده وابستگی داده ای تعریف کرد. اگر چنین گره ای نباشد، شرط مسیر همواره درست محسوب می شود.

با توجه به قاعده های زبان و استفاده مناسب تر از شرط های مسیر، در برنامه ها تنها انتساب های یگانه ایستا⁸¹ مجاز است. به معنای آن که برنامه ها حاوی انتساب های چندگانه برای یک متغیر نخواهد بود. البته این برنامه ها به برنامه های صرفاً حاوی انتساب های یگانه قابل تبدیل هستند. از طرفی، چنین فرض می شود که هیچ وابستگی داده ای حلقه نقلی⁸² در مسیرهای از ورودی های سطح بالا به خروجی های سطح پایین برنامه وجود ندارد. [۵] پس در برنامه های ورودی باید به این محدودیت ها نیز توجه داشت.

در الگوریتم شکل ۱۱، 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 می گیرد و کد مبدأ M' را که عدم تداخل غیرحساس به پیشرفت را برآورده می کند، به عنوان خروجی برمی گرداند. به بیان دیگر، باید دنباله خروجی های برنامه بازنویسی شده برای هر دو اجرای دلخواه از آن برنامه که

⁷⁹ Execution Conditions

⁸⁰ Backtracking

⁸¹ Static Single Assignment

⁸² Loop-carried Data Dependency

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 ۱۲ آورده شده است.

$RW_{PINI}(M, G)$:

Initialize F to the set of all paths $Start \hookrightarrow P \rightsquigarrow P'$ in the PDG G of M where P is the node representing a high input and P' is the node representing outL l for some l ;

if $F = \emptyset$ then

 return M ;

end

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

determine the type of flow indicated by each path $f \in F$;

foreach $f \in F$ do:

 Generate the path condition of f as the conjunction of the execution conditions of node N satisfying $f = Start \rightsquigarrow X \xrightarrow{d} N \rightsquigarrow P'$ 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 $f' \in F$ which terminate at n ;

if all paths $f' \in F$ terminating at n indicate an explicit flow **then**

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

 else

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

 end

end

return M' ;

شکل ۱۱ - الگوریتم بازنویسی عدم تداخل حالت غیرحساس به پیشرفت که برنامه M و گراف وابستگی برنامه مربوط به آن G را می‌گیرد [۵]

```

program;
inL l1, l2;
inH h1;
if l1 == 0 then
    l2 = h1
else
    NOP
endif;
outL l2

```

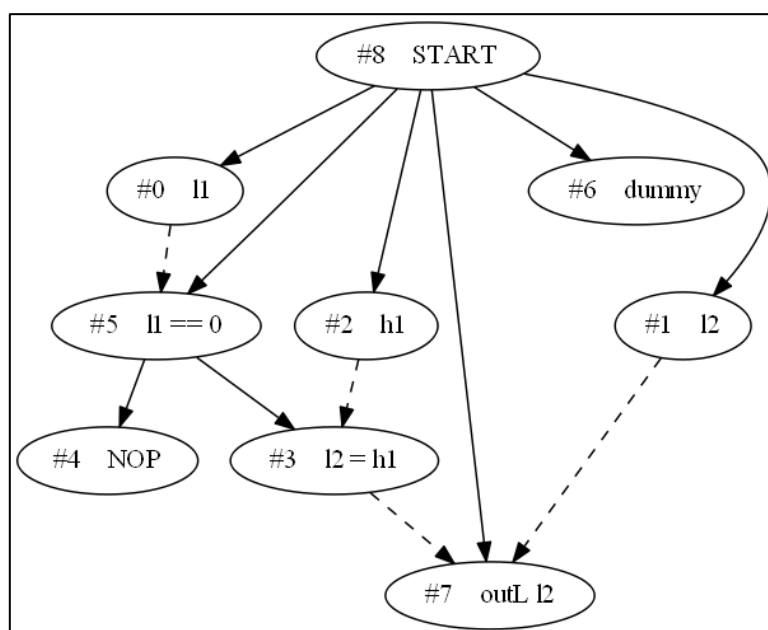
(الف)

```

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)



(پ)

شکل ۱۲ - (الف) نمونه کد به زبان WL (ب) برنامه بازنویسی شده برنامه الف در حالت غیر حساس به پیشرفت (پ) گراف وابستگی برنامه الف

۲.۴ 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 ۱۳، 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
```

(الف)

```
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)

شکل ۱۳ - (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 اعمال 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 آغازین معادل از نظر مشاهده‌گر 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 نوع سامانه به همراه یک مکانیزم زمان‌اجرا به نام پیش‌گویی خاتمه^{۸۳} 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 صحت الگوریتم بازنویسی، در برنامه‌ای که ممکن است هنگام اجرا مقادیر سطح بالا از طریق وضعیت پیشرفت برنامه نشت پیدا کنند، معناساخت 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^{۸۴} 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

⁸³ Termination Oracle

⁸⁴ Loop Analyzer

برای حالت‌های آغازین معادل از نظر مشاهده‌گر سطح پایین، یا همواره خاتمه می‌یابد یا همواره واگرا می‌شود [۵] 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 ۱۴، عبارت $h1 \geq l1 \text{ or } l1 < 0$ را به عنوان نتیجه تحلیل برمی‌گرداند.

```
program;
inH h1;
inL l1;
while h1 < l1 do
    NOP;
    h1 = h1 - l1
done;
outL l1
```

شکل ۱۴ - برنامه‌ای که حلقه موجود در آن در حالتی که $h1 \geq l1 \text{ or } l1 < 0$ 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 این جا 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^{۸۵} 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 عبارت برگردانده شده مشروط می‌کند. پس به این ترتیب، کد برنامه جدید قطعاً خاتمه می‌یابد.

الگوریتم بازنویسی مطرح‌شده در شکل ۱۵، کد مبدأ برنامه M و گراف وابستگی برنامه متناظر آن را می‌گیرد و کد M' 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 $Start \hookrightarrow h \hookrightarrow E^+$ هستند، که در آن h یک ورودی سطح بالا و E^+ یک مسیر منتهی به یک عبارت شرطی حلقه است، ممکن است بازنویسی شوند. البته ممکن است این گونه 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 M' متناظر با گره n در گراف وابستگی برنامه را برمی‌گردانند.

⁸⁵Loop Guards

$RW_{PSNI}(M, G):$

Initialize D to the set of all paths $Start \hookrightarrow P \hookrightarrow E^+$ in G where E^+ is a path terminating at a loop guard and P is the node representing a high input;

$M' = RW_{PIN}(M, G);$

if $D = \emptyset$ then

 return M' ;

end

$H = \max \{ height(n) \mid n \text{ is a node on } G \}$, 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 M' as follows:

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

foreach node n with $height(n) = h$ representing a loop on some path $f \in D$ **do**

$r = \text{LoopAnalyzer}(\text{loop}(n));$

if $r = \text{False}$ **then**

if $X \xrightarrow{c} n$ appears on at least one path $f \in D$ **do**

 replace $\text{loop}(n)$ with the statement “if $\text{guard}(n)$ then $\text{body}(n)$

endif”;

 end

 else

if $r \neq \text{True}$ **then**

 replace $\text{loop}(n)$ with the statement “if r then $\text{loop}(n)$ endif”;

 end

 end

end

$h = h - 1;$

end

return M' ;

شکل ۱۵ - الگوریتم بازنویسی عدم تداخل حالت حساس به پیشرفت که برنامه M و گراف وابستگی برنامه مربوط به آن G را

می‌گیرد [۵]

صفحه ۳۷ از ۸۹

همان طور 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 بازنویسی شده اجرا شود. گرچه راه کارهای دیگری مثل تغییر عبارت شرطی حلقه برای این که فقط به مقدار متناهی حلقه اجرا شود نیز وجود دارد، اما باید آن راهبردها را از نظر شفافیت با روش مورد استفاده در این جا بررسی کرد. ضمناً باید دقت داشت که ابتدا باید حلقه‌های تودرتو^{۸۶} 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 مطرح شده ارتفاع گره‌های بیانگر حلقه در درختی که با حذف یال‌های وابستگی داده‌ای از گراف وابستگی برنامه به دست آمده است را ملاک عمل قرار می‌دهد. در شکل ۱۶، نمونه کد برنامه تبدیل شده توسط این الگوریتم برای کد مبدأ برنامه شکل ۱۴ را مشاهده می‌شود.

اثبات صحت و شفافیت الگوریتم‌های استفاده شده در این پروژه، در مقاله [۵] قابل مشاهده است.

```

program;
inH h1;
inL l1;
    if l1 <= h1 or l1 < 0 then
        while h1 < l1 do
            NOP;
            h1 = h1 - l1
        done
    endif;
outL l1

```

شکل ۱۶ - کد مبدأ بازنویسی شده توسط الگوریتم حالت حساس به پیشرفت برای برنامه شکل ۱۴

⁸⁶ Nested Loops

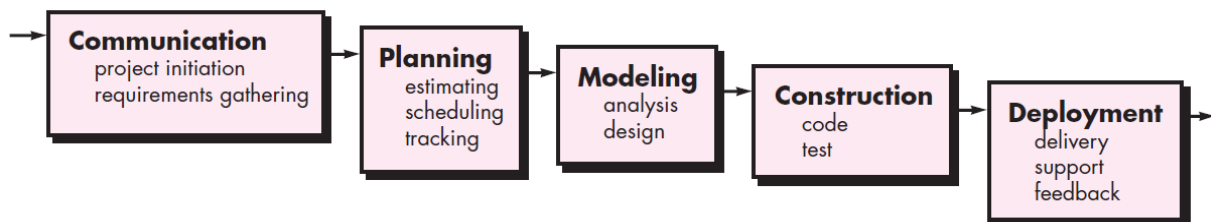
فصل پنجم

پیاده‌سازی و ایجاد رابط کاربری

۱.۵ تحلیل و طراحی نرم‌افزار

با توجه به مشخص و ثابت بودن نیازهای این نرم‌افزار در همان ابتدای تعریف پروژه، می‌توان از مدل فرآیندی آبشاری^{۸۷} یا چرخه حیات کلاسیک^{۸۸} استفاده کرد. همچنین، تحلیل و طراحی این نرم‌افزار با رویکرد شی‌گرایی^{۸۹} انجام شده است.

این مدل فرآیندی شامل پنج مرحله ارتباط^{۹۰}، برنامه‌ریزی^{۹۱}، مدل‌سازی^{۹۲}، ساخت^{۹۳} و استقرار^{۹۴} است.



شکل ۱۷ - مدل فرآیندی آبشاری [۱۹]

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.

⁸⁷ Waterfall Process Model

⁸⁸ Classic Life Cycle

⁸⁹ Object-Oriented

⁹⁰ Communication

⁹¹ Planning

⁹² Modeling

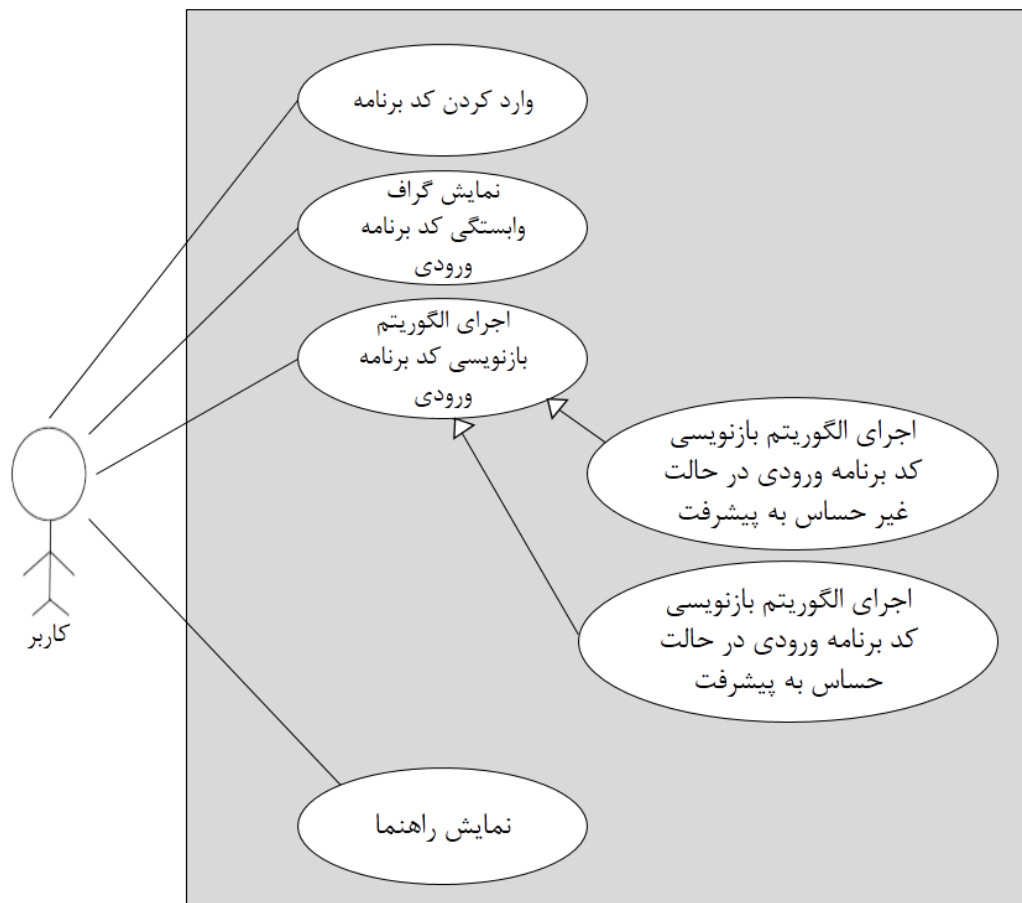
⁹³ Construction

⁹⁴ Deployment

- 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 آشنایی 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 درخواست سیستم^{۹۵} است که با توجه به نیازمندی‌ها و درخواست‌های سیستم به دست آمده از تعریف پروژه ترسیم می‌شود. این نمودار مبنای تحلیل‌های بعدی

^{۹۵} Use Case Diagram

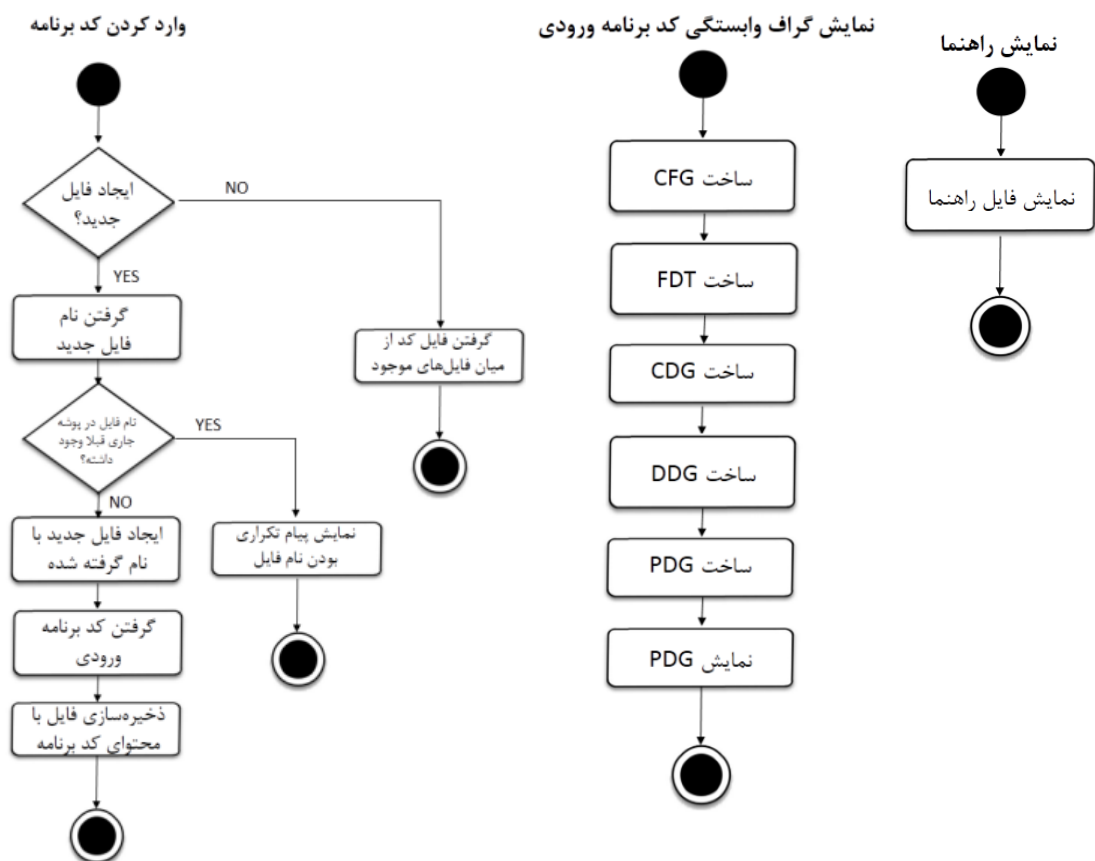
خواهد بود. در ادامه نمودارهای درخواست سیستم و فعالیت^{۹۶} به عنوان بخشی از قسمت تحلیل آمده است. برای ترسیم این نمودارها، از زبان UML^{۹۷} استفاده شده است.



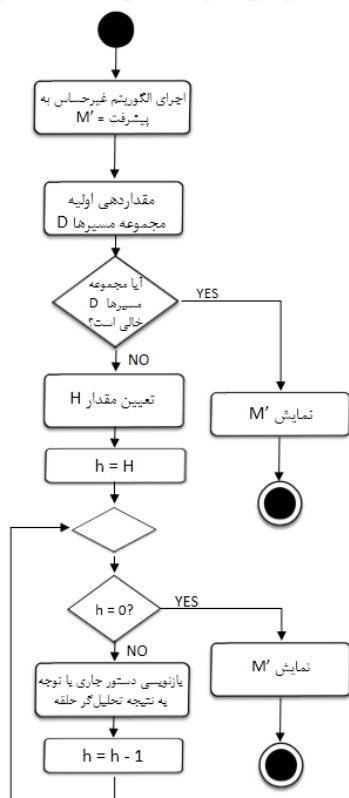
شکل ۱۸ - نمودار درخواست سیستم نرم‌افزار پروژه

^{۹۶} Activity Diagram

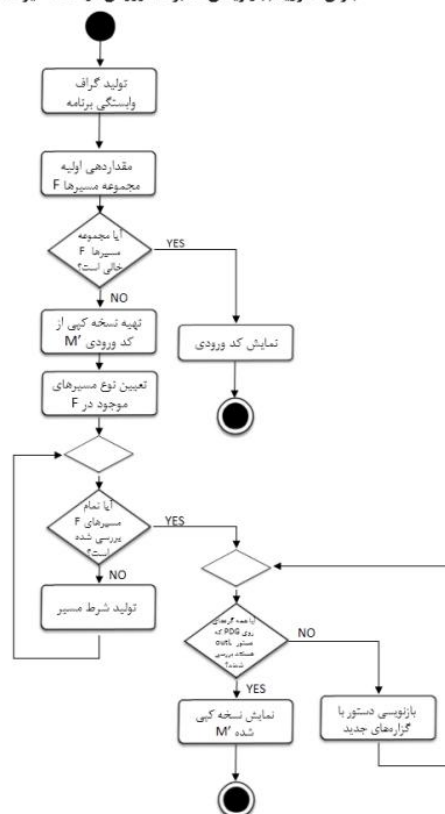
^{۹۷} Unified Modeling Language



اجرای الگوریتم بازنویسی کد برنامه ورودی در حالت حساس به پیشرفت

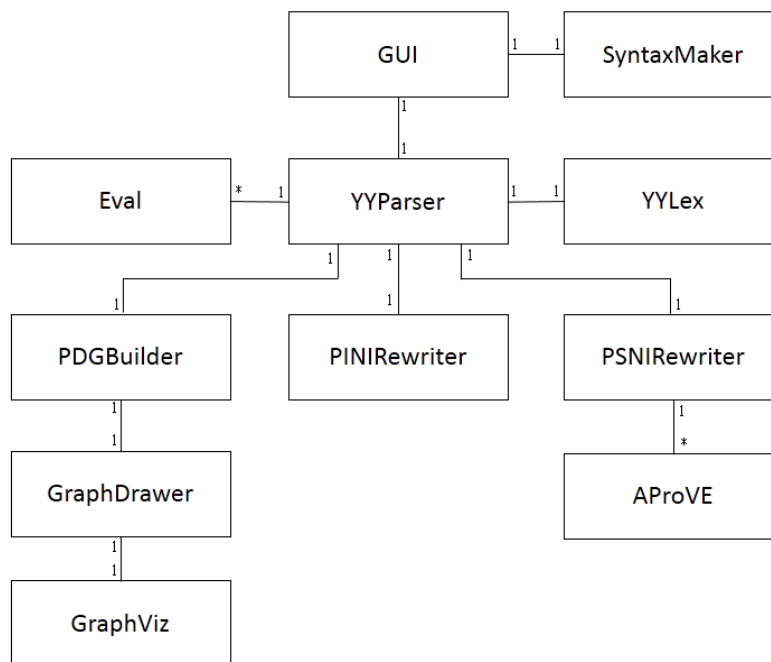


اجرای الگوریتم بازنویسی کد برنامه ورودی در حالت غیرحساس به پیشرفت



شکل ۱۹ - نمودارهای فعالیت نرم‌افزار پروژه

در مرحله طراحی، مهم‌ترین نمودار که آینه تمام‌نمای معماری نرم‌افزار نیز به شمار می‌رود، نمودار کلاس^{۹۸} 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 ۲۰، نمودار 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 ۲۰ - 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

^{۹۸} Class Diagram

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 کمکی موجود پرداخته می‌شد و با بهره‌گیری از آن‌ها، پیاده‌سازی صورت می‌گرفت. ضمناً گزارش مراحل انجام هر یک از فازهای فوق در پوشه پروژه^{۹۹} موجود می‌باشد.

۲.۵ 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^{۱۰۰} و تحلیل‌گر نحوی^{۱۰۱}، فراهم شدند. در تحلیل‌گر لغوی، تمامی کلمات کلیدی و عناصر مختلف زبان به صورت نماد^{۱۰۲}‌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

^{۹۹} <https://github.com/smahmadpanah/BScProject>

^{۱۰۰} Lexer

^{۱۰۱} Parser

^{۱۰۲} Token

پیاده‌سازی می‌شود. در این مرحله، خطاهای نحوی^{۱۰۳} 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^{۱۰۴} 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 گرافیکی گراف وابستگی برنامه، از ابزار قدرت‌مند GraphViz [۲۰] 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

¹⁰³ Syntax Errors

¹⁰⁴ Linked List

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، LoopFrog، PAG و AProVE [۲۱] 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 آن جایی 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 صورتی که 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 [۲۲] 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 برنامه‌هایی 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، راهنمای تصویری استفاده از برنامه 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.

۳.۵ ایجاد رابط کاربری گرافیکی^{۱۰۵}

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.

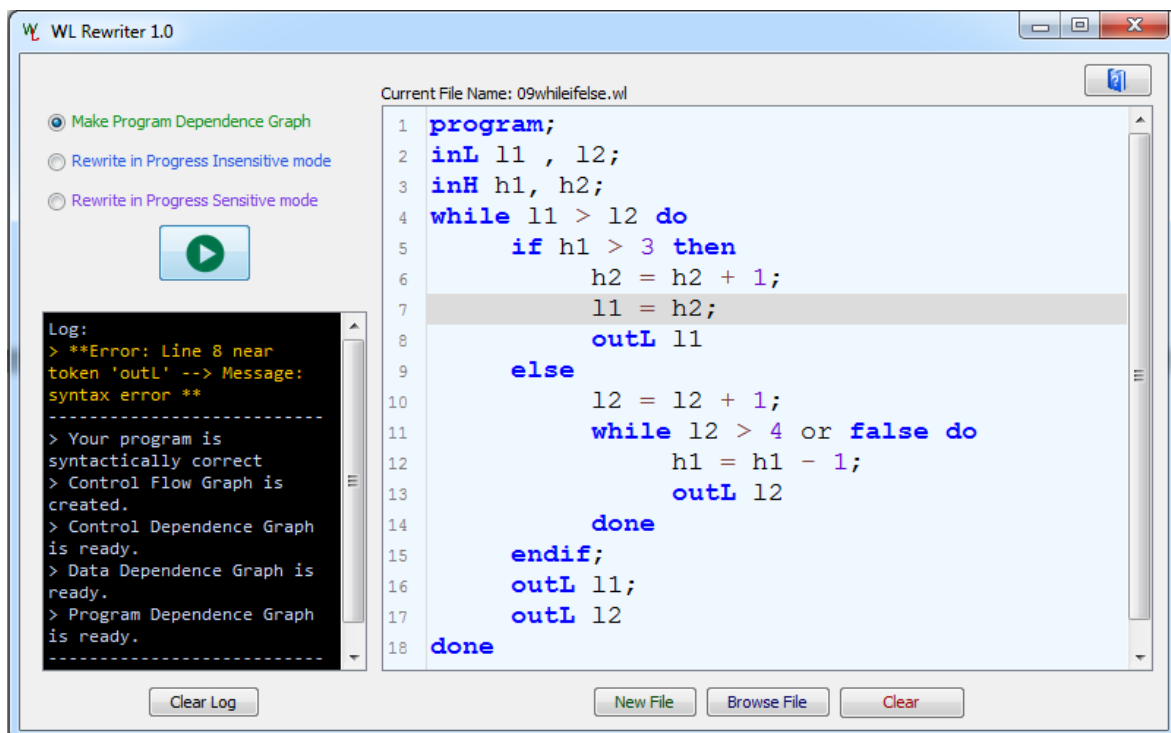
نکات زیر برای طراحی رابط کاربری این برنامه مورد استفاده قرار گرفته شده است:

- 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^{۱۰۶}‌ها استفاده شود.
- برای حذف یا پاک کردن اطلاعات مهم، تأیید مجدد کاربر دریافت شود.
- برای نمایش پیغام‌ها از رنگ‌های متناسب استفاده شود.

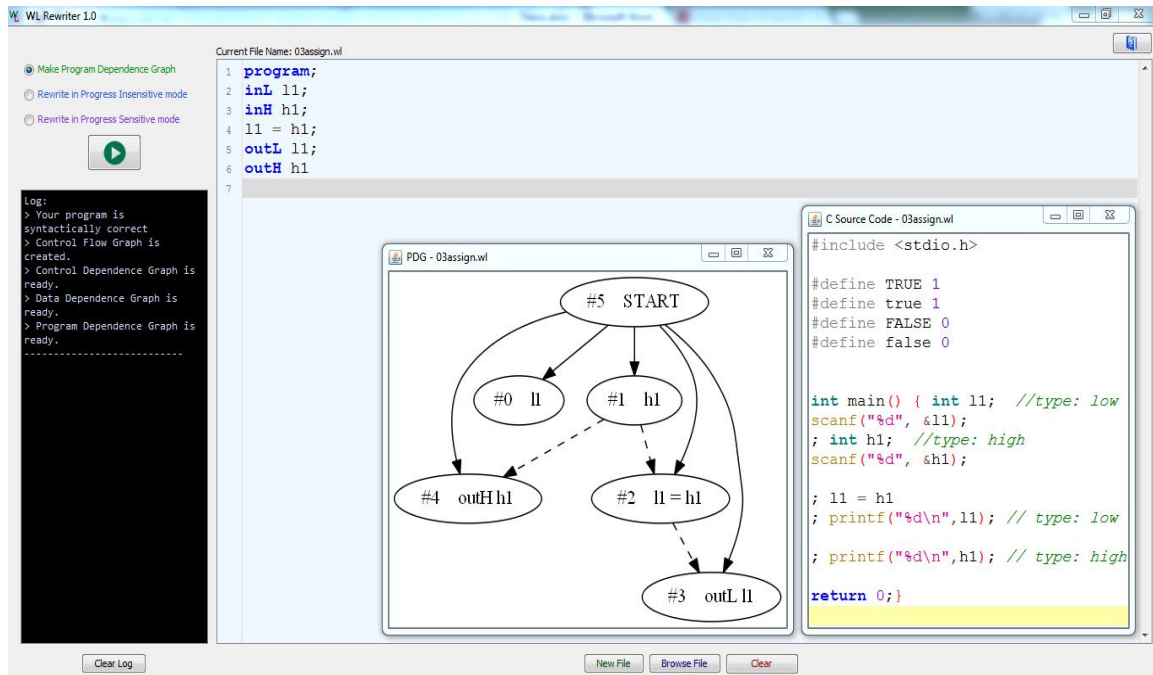
¹⁰⁵ Graphical User Interface

¹⁰⁶ Mnemonic

- 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 ۲۱ - نمای کلی رابط کاربری گرافیکی نرم‌افزار



شکل ۲۲- نمونه‌ای از اجرای برنامه در رابط کاربری گرافیکی نرم‌افزار

۴.۵ راستی‌آزمایی و آزمون

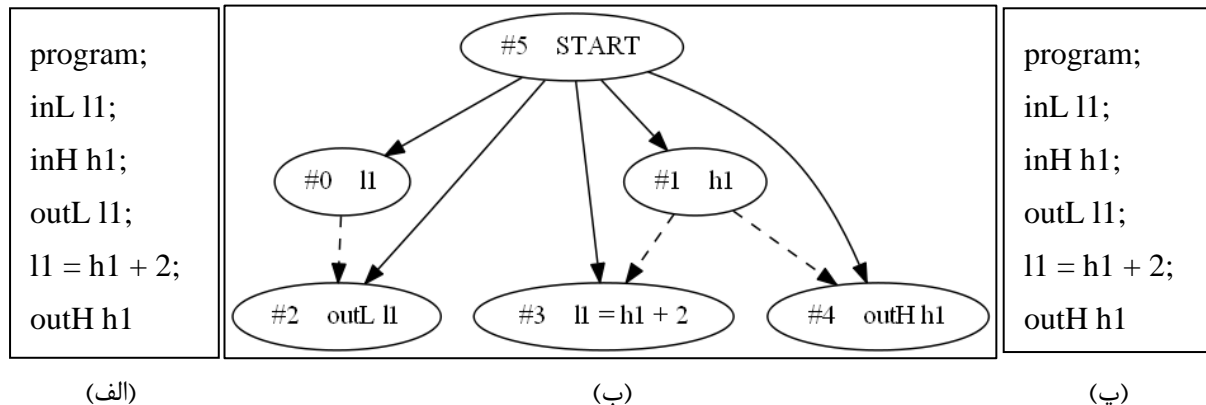
همان‌طور که قبلاً ذکر شد، روش مطرح شده و الگوریتم‌های بازنویسی از نظر صحت و شفافیت قابل اثبات هستند؛ اما راستی‌آزمایی و آزمون برنامه پیاده‌سازی شده نیز اهمیت دارد. برای این کار، با بهره‌گیری از آزمون دامنه^{۱۰۷} تعدادی مورد آزمون^{۱۰۸} برای بررسی صحت اجرای برنامه پیاده‌سازی شده طراحی شد. روش آزمون دامنه 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. به همین منظور سعی شد تا با

¹⁰⁷ Domain Testing

¹⁰⁸ Test Case

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 موارد آزمون 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.

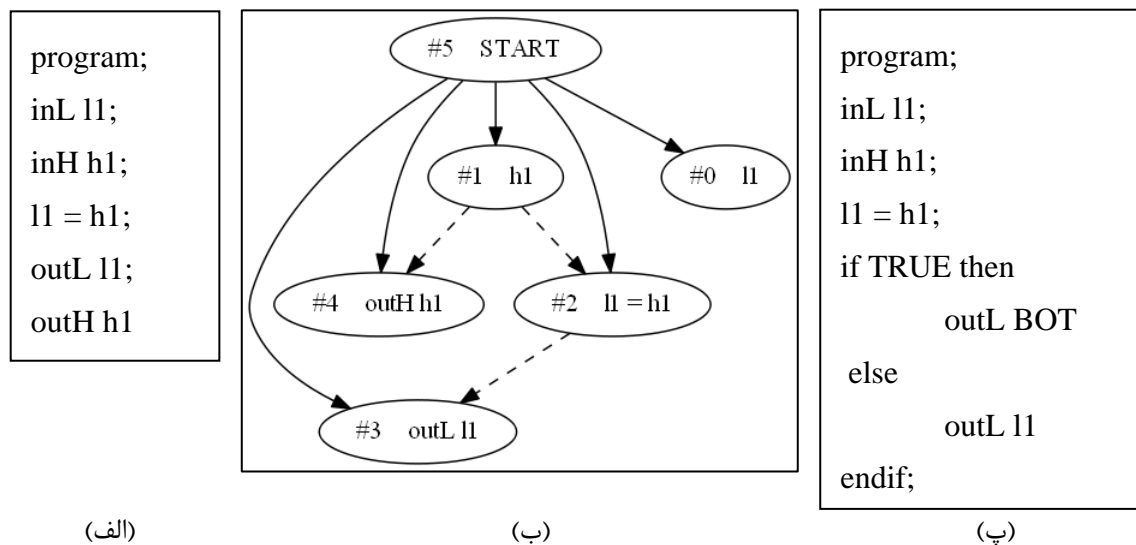
در ادامه نمونه‌هایی از موارد آزمون و برنامه‌های بازنویسی‌شده آن‌ها آورده شده است.



شکل ۲۳- (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 ۲۳، مورد آزمونی مشاهده می‌شود که گرچه به صورت صریح مقدار سطح بالا در متغیر سطح پایین l1 قرار می‌گیرد، اما به دلیل این که 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.



شکل ۲۴- (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 ۲۵، 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 بازنویسی شده را نشان می‌دهد.

جدول ۱ - نمونه ورودی‌ها و خروجی‌ها برای برنامه به زبان C متناظر با برنامه شکل ۲۵-الف)

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

جدول ۲ - نمونه ورودی‌ها و خروجی‌ها برای برنامه به زبان C متناظر با برنامه شکل ۲۵-ب)

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

همان‌طور که در جدول ۲ مشاهده می‌شود، برنامه بازنویسی شده برخلاف برنامه اولیه، به ازای ورودی‌های مختلف سطح بالا تغییری نمی‌کند و موارد ناقض عدم تداخل اصلاح شده است.

```

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

```

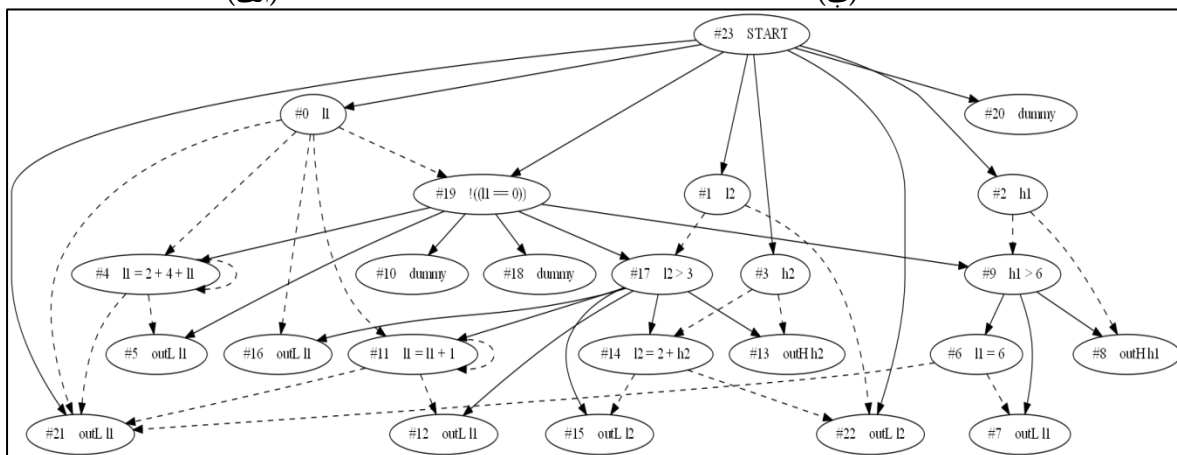
(الف)

```

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;
    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
endif

```

(ب)



(پ)

شکل ۲۶- (الف) برنامه مورد آزمون با نام 07ifelseadvanced.wl (ب) برنامه بازنویسی شده برای حالت غیر حساس به پیشرفت مربوط

به برنامه الف (پ) گراف وابستگی برنامه مربوط به برنامه الف

در شکل ۲۶، 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 ۳ - نمونه ورودی‌ها و خروجی‌ها برای برنامه به زبان C متناظر با برنامه شکل ۲۶- (الف)

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

جدول ۴ - نمونه ورودی‌ها و خروجی‌ها برای برنامه به زبان C متناظر با برنامه شکل ۲۶- (ب)

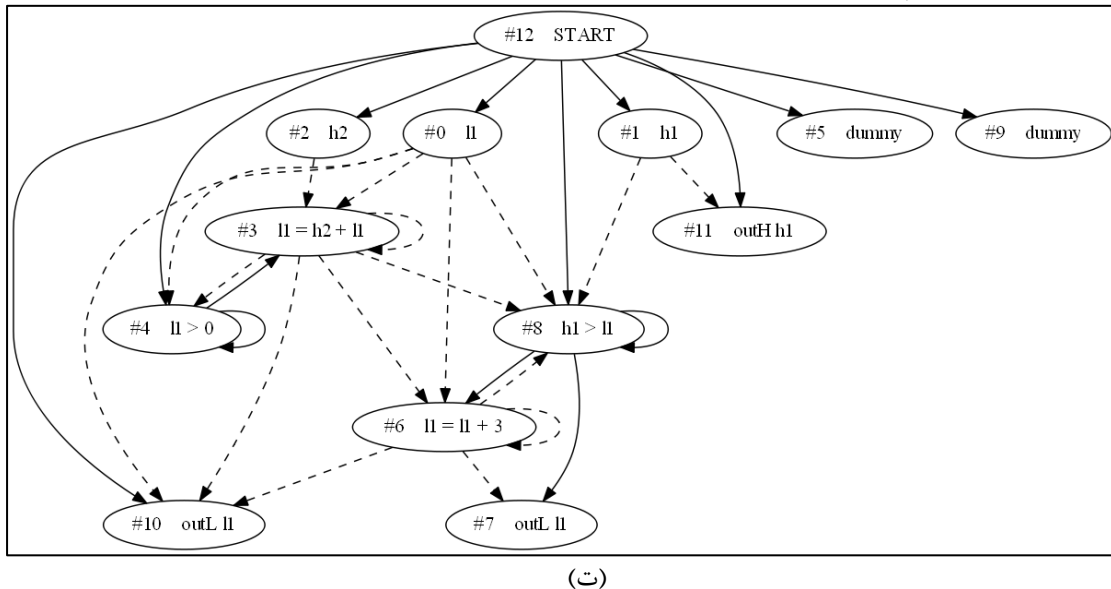
l1	l2	h1	h2	outL l1 (line# 6)	outL l1 (line# 9)	outH h1 (line# 10)	outL l1 (line# 15)	outH h2 (line# 16)	outL l2 (line# 19)	outL l1 (line# 20)	outL l1 (line# 23)	outL l2 (line# 24)	Violation
0	4	0	0	-	-	-	1	0	-	-	-	4	No
0	4	1	1	-	-	-	1	1	-	-	-	4	
0	2	0	0	-	-	-	-	-	BOT	0	0	BOT	No
0	2	1	1	-	-	-	-	-	BOT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

در مورد آزمون شکل ۲۶، با توجه به وجود جریان‌های صریح و ضمنی، برنامه بازنویسی شده است. همان‌طور که در

جدول ۴ 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 بازنویسی شده در حالت غیرحساس به پیشرفت را خواهند داشت. در ادامه موارد آزمون نمونه مربوط به حالت حساس به پیشرفت آمده است.

<pre> 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 </pre>	<pre> 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 </pre>	<pre> 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 </pre>
(الف)	(ب)	(پ)



شکل ۲۷ - (الف) برنامه مورد آزمون با نام `11whilewhileconcat.wl` (ب) برنامه بازنویسی شده برای حالت غیرحساس به پیشرفت مربوط به برنامه

الف (پ) برنامه بازنویسی شده برای حالت حساس به پیشرفت مربوط به برنامه الف (ت) گراف وابستگی برنامه مربوط به برنامه الف

جدول ۵ - نمونه ورودی‌ها و خروجی‌ها برای برنامه به زبان C متناظر با برنامه شکل ۲۷- (الف)

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

جدول ۶ - نمونه ورودی‌ها و خروجی‌ها برای برنامه به زبان C متناظر با برنامه شکل ۲۷- (ب)

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

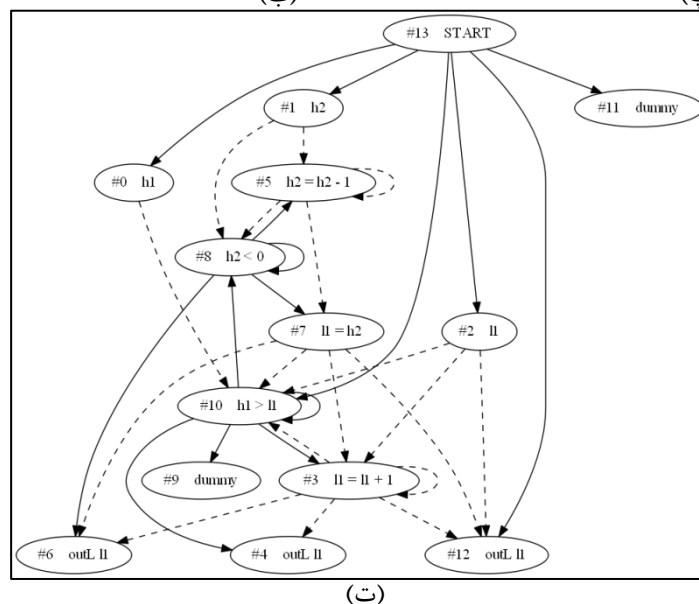
جدول ۷ - نمونه ورودی‌ها و خروجی‌ها برای برنامه به زبان C متناظر با برنامه شکل ۲۷- (پ)

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

همان‌طور که در جدول‌های فوق آمده است، با بازنویسی برنامه در حالت غیرحساس به پیشرفت، برنامه بازنویسی شده 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 ۱ 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.

<pre> 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 </pre>	<pre> 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 </pre>	<pre> 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 </pre>
(الف)	(ب)	(پ)



(ت)

شکل ۲۸ - (الف) برنامه مورد آزمون با نام `whilewhilenested.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 ۸ - نمونه ورودی‌ها و خروجی‌ها برای برنامه به زبان C متناظر با برنامه شکل ۲۸-الف)

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

جدول ۹ - نمونه ورودی‌ها و خروجی‌ها برای برنامه به زبان C متناظر با برنامه شکل ۲۸-ب)

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

جدول ۱۰ - نمونه ورودی‌ها و خروجی‌ها برای برنامه به زبان C متناظر با برنامه شکل ۲۸-پ)

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

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

این که 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 بازنویسی شده 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 جدول‌های 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

در طول فصول گذشته، ابتدا درباره امنیت و خط مشی امنیتی صحبت شد. سپس خط مشی امنیتی عدم تداخل را به عنوان خط مشی مورد نظر در این پروژه معرفی کردیم و محدودیت‌هایی که برای اعمال این خط مشی وجود داشت را اشاره کردیم. در ادامه نوعی از دسته‌بندی عدم تداخل؛ یعنی حالت غیرحساس به پیشرفت و حساس به پیشرفت مطرح شد و مشاهده شد که برای اعمال این خط مشی، مکانیزم‌های مختلفی وجود دارد که یکی از بهترین آن‌ها، 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. همچنین، زبان برنامه‌نویسی 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، با استفاده از بیان صوری خط مشی‌ها، صحت و شفافیت الگوریتم‌های بازنویسی قابل اثبات است. در ادامه به نحوه پیاده‌سازی و ابزارهای مورد استفاده پرداخته شد و با ارائه موارد آزمون کاربردی، صحت برنامه پیاده‌سازی شده را نشان دادیم.

ایده مورد استفاده در این پروژه، یکی از گام‌های ابتدایی و رو به جلو برای اعمال با حفظ شفافیت خط مشی‌های امنیتی است. برای کارهای آتی، می‌توان برای زبان‌های برنامه‌نویسی پیشرفته‌تر و رایج‌تر که ساختارهای زبانی پیچیده‌تری دارند، روش‌های مشابه و بهتری ارائه کرد. زبان‌هایی که از ساختارهای کلاس، شیء، چندنخی و سایر ویژگی‌های جدیدتری که زبان‌های برنامه‌نویسی امروزی پشتیبانی می‌کنند را می‌توان به عنوان آینده این پروژه قلمداد کرد. برای بهبود پیاده‌سازی انجام‌شده نیز می‌توان با بهینه‌سازی کد برنامه، به سرعت و استفاده کمتر از حافظه توجه کرد. تابع تحلیل‌گر حلقه نیز به تنهایی می‌تواند موضوع پژوهش جذابی برای علاقمندان این حوزه باشد که نقش بسزایی در بهبود بازنویس ایفا می‌کند. نکته دیگر این که از حیث پژوهش‌های نظری نیز می‌توان دسته‌بندی خط مشی‌های قابل اعمال توسط روش بازنویسی برنامه را به عنوان یکی از مسائل روز نام برد.

منابع و مراجع

- [۱] 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.
- [۲] D. Volpano and G. Smith, "A Type-Based Approach to Program Security", *TAPSOFT '97 Proceedings of the 7th International Joint Conference CAAP/FASE on Theory and Practice of Software Development*, 1997, pp. 607-621.
- [۳] 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.
- [۴] M.R. Clarkson and F.B. Schneider, "Hyperproperties", *Journal of Computer Security - 7th International Workshop on Issues in the Theory of Security (WITS'07)*, 2010, pp. 1157-1210.
- [۵] A. Lamei and M. S. Fallah, "Rewriting-Based Enforcement of Noninterference in Programs with Observable Intermediate Values", submitted to *Journal of Universal Computer Science*, 2015.
- [۶] V.N. Venkatakrishnan, W. Xu, D.C. DuVarney, and R. Sekar, "Provably Correct Runtime Enforcement of Non-interference Properties", in *Proceedings of the 8th International Conference on Information and Communications Security, ICICS'06*, Springer-Verlag Berlin, Heidelberg, 2006, pp. 332-351.
- [۷] J. Magazinius, A. Russo, and A. Sabelfeld, "On-the-fly inlining of dynamic security monitors", *Computers and Security-Silver Linings in the Cloud*, 2012, pp. 827-843.
- [۸] G. Le Guernic, A. Banerjee, T. Jensen, and D.A. Schmidt, "Automata-based confidentiality monitoring", in *Proceedings of the 11th Asian computing science conference on Advances in computer science: secure software and related issues, ASIAN'06*, Vol. 4435, Springer-Verlag Berlin, Heidelberg, 2007, pp. 75-89.
- [۹] 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.
- [۱۰] G.M. Kevin W. Hamlen and F.B. Schneider, "Computability classes for enforcement mechanisms", *ACM Transactions on Programming Languages and Systems*, Vol. 28, 2006, pp. 175-205.
- [۱۱] J. Ferrante, K.J. Ottenstein, and J.D. Warren, "The program dependence graph and its use in optimization", *ACM Transactions on Programming Languages and Systems*, Vol.9, 1987, pp. 319-349.
- [۱۲] H. Mantel and H. Sudbrock, "Types vs. pdgs in information flow analysis", in *Logic-Based Program Synthesis and Transformation*, Springer, 2013, pp. 106-121.
- [۱۳] "JFlex", Available: <http://jflex.de/> [Sep. 10, 2015].

- [۱۴] “Bison”, Available: <https://www.gnu.org/software/bison/> [Sep. 10, 2015].
- [۱۵] K. M. Anderson, Class Lecture, Topic: “Lecture 15: Control Dependence Graphs” CSCI 5828, University of Colorado at Boulder, Spring 2000, Available: <http://www.cs.colorado.edu/~kena/classes/5828/s00/lectures/lecture15.pdf> [Jul. 25 2015].
- [۱۶] T. Teitelbaum, Class Lecture, Topic: “Lecture 24: Control Flow Graphs” Introduction to Compilers, Cornell University, 2008, <http://www.cs.cornell.edu/courses/cs412/2008sp/lectures/lec24.pdf> [Jul. 25 2015].
- [۱۷] C. N. Fischer, Class Lecture, Topic: “The Program Dependence Graph: Control Flow and Control Dependences” S502 Compilers, Fall 2008, Available: <http://pages.cs.wisc.edu/~fischer/cs701.f08/lectures/Lecture19.4up.pdf> [Jul. 25 2015].
- [۱۸] 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.
- [۱۹] Roger S. Pressman, “Process Models” in *Software Engineering: A Practitioner’s Approach*, 7th ed., Mc Graw-Hill Higher Education, 2010, pp. 39-41.
- [۲۰] 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, 2000, pp. 1203-1233, Available: www.graphviz.org [Aug. 12 2015].
- [۲۱] “AProVE”, Available: <http://aprove.informatik.rwth-aachen.de/index.asp?subform=home.html> [Aug. 25 2015].
- [۲۲] “RSyntaxTextArea”, Available: <http://bobbylight.github.io/RSyntaxTextArea/> [Sep. 04 2015].

پیوست

بخش‌هایی از پیاده‌سازی

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 YYParse(new Lexer() {

    @Override
    public int yylex() {
        int yyl_return = -1;
        try {

            yyl_return = lexer.yytext();
        } 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;
    }

    /*-----*/
    /*-----*/

    }

    (ترتیب اولویت عملگرها)

    %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

    %%

    (عملیات مربوط به هر قاعده زبان در ذیل آن آمده است)
    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);
ar);

cSourceCodeForPSNI =
break;
((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).varia
bles);

        ((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).varia
bles);

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

        ((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)$$.variables.addAll(((eval)$1).variabl
es);
    ((eval)$$.variables.addAll(((eval)$3).variabl
es);
};
| 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)$$.variables.addAll(((eval)$1).variabl
es);
    ((eval)$$.variables.addAll(((eval)$3).variabl
es);
};
| 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)$$.variables.addAll(((eval)$1).variabl
es);
    ((eval)$$.variables.addAll(((eval)$3).variabl
es);
};
| 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)$$.variables.addAll(((eval)$1).variabl
es);
    ((eval)$$.variables.addAll(((eval)$3).variabl
es);
};
| 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)$$.variables.addAll(((eval)$1).variabl
es);
    ((eval)$$.variables.addAll(((eval)$3).variabl
es);
};
| 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).variabl
es);

```

```

    ((eval)$$.variables.addAll(((eval)$3).variabl
es);
};
| exp MINUS_KW exp
{
    writer.print("\t exp -> exp
MINUS_KW exp \n") ;
    $$=new eval();
    ((eval)$$.stmt += ((eval)$1).stmt
+ " - " + ((eval)$3).stmt;
    ((eval)$$.cSourceCode +=
((eval)$1).cSourceCode + " - " +
((eval)$3).cSourceCode;
    writer.print(((eval)$$.stmt+ "\n");

    ((eval)$$.variables.addAll(((eval)$1).variabl
es);
    ((eval)$$.variables.addAll(((eval)$3).variabl
es);
};
| 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)$$.variables.addAll(((eval)$1).variabl
es);
    ((eval)$$.variables.addAll(((eval)$4).variabl
es);
};
| 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)$$.variables.addAll(((eval)$1).variabl
es);
        ((eval)$$.variables.addAll(((eval)$4).variabl
es);
    };
    | 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)$$.variables.addAll(((eval)$2).variabl
es);
    };
    | LPAR_KW exp RPAR_KW %prec p
    {
        writer.print("\t exp -> LPAR_KW
exp RPAR_KW \n");
        $$=new eval();
        ((eval)$$.stmt += "("+
((eval)$2).stmt + ")";
        ((eval)$$.cSourceCode += "("+
((eval)$2).cSourceCode + ")";
        writer.print(((eval)$$.stmt+ "\n");

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

    };

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).variabl
es);

        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);
ar);

        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).variabl
es);

    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(current
Var);

        //((eval)$$.node.addToVariablesOfNode(cur
rentVar);

        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)$$.cSourceCode
+= "scanf(\"%d\\n\", &" + alpha + ");\n";
                                }
                                writer.print(((eval)$$.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(current
Var);

                                (((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(varv
ar);

        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);

};
| 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).variabl
es);

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

    ((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
*****/
(هر یک از نشانه‌ها و غیر پایانه‌های زبان، نمونه‌ای کلاس زیر
هستند که به عنوان یک ساختمان داده است.)

class eval {

    public String stmt="";

```

(کاراکترهای برنامه ورودی تا قاعده مربوط، در این رشته ذخیره می‌شود.)

public String nodeIdAndStmt="";

(کاراکترهای برنامه ورودی و شماره گره آن‌ها در گراف وابستگی برنامه در این رشته ذخیره می‌شود.)

public String cSourceCode = "";

(کد مبدأ متناظر با زبان c در این رشته ذخیره می‌شود.)

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

(متغیرهای موجود در یک عبارت یا ساختار برنامه در ساختمان داده مجموعه درهم نگه‌داری می‌شود. از این ساختمان داده برای به دست آوردن گراف وابستگی داده استفاده می‌شود.)

public Node node;

(مشخصات گزاره مربوط در یک ساختمان داده گره نگه‌داری می‌شود.)

public MyLinkedList list;

(گراف‌های وابستگی برنامه به کمک کنار هم قرار گرفتن گره‌ها و لیست‌های پیوندی این ساختمان داده تشکیل و ذخیره می‌شوند.)

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, FDTTree, PDG; //FDTTree
    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 =
        YYParse.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 Dependencies from PDFs
computeControlDep();

//print the Control Dependencies
printControlDeps();

//compute the Data Dependencies from CFG
computeDataDep();

//print the Data Dependencies
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));
    }

    FDTTree = 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.getPost
Dominators())) {
            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.getControlDep().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.getAssigned
Variable().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) {

```

```
private void printDataDeps() {
    String DDgraph = "";

    for (Node n : cfg.getNodeSet()) {
        for (Node q : n.getDataDepsForThisNode()) {
```

```

        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.getPa
rentOfControlDep());

for (Node temp :
nodeInFDT.getDataDepsForThisNode()) {
    for (Node temp2 : PDG.getNodeSet()) {
        if (temp.getNodeID() ==
temp2.getNodeID()) {

nodeInPDG.getDataDepsForThisNode().add(temp2);
        }
    }
}

nodeInPDG.setParentsOfDataDep(nodeInFDT.getPare
ntsOfDataDep());

    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 While Programs**

By
Seyed Mohammad Mehdi Ahmadpanah

Supervisor
Dr. Mehran S. Fallah

September 2015