پردیس علوم
دانشکده ریاضی، آمار و علوم کامپیوتر

بهبود روش وارسی مدل با استفاده از نظریه تعبیر مجرد

نگارنده

پویا پرتو

استاد راهنمای اول: دکتر مجید علی‌زاده
استاد راهنمای دوم: دکتر مجتبی مجتهدی

پایاننامه برای دریافت درجه کارشناسی ارشد
در رشته علوم کامپیوتر

تاریخ دفاع

چکیده

تقدیم به

تقدیم به

سپاسگزاری

سپاسگزاری

پیشگفتار

با توجه به پیشرفت روز افزون علوم کامپیوتر و ورود کاربردهای آن به زندگی روزمره، پیشرفت در روش‌های ساخت و نگه‌داری برنامه‌ها نیازی آشکار به نظر می‌رسد. یکی از مسائل مهم در این زمینه بررسی صحت کارکرد برنامه‌های نوشته‌شده است. عدم صحت کارکرد برنامه‌های نوشته‌شده بسته به حساسیت کاری که یک برنامه انجام می‌دهد می‌تواند تبعات مختلفی داشته‌باشد. پرتاب ناموفق آریان ۵[۱۶] ، از مدار خارج شدن مدارگرد مریخ [۲] و تصادف هلیکوپتر چینوک [۱] چند نمونه از تبعات بزرگ این قضیه در گذشته بوده اند، هرچند که به‌سادگی می‌توان فجایع دیگری را مشابه این اتفاقات در زندگی روزمره‌ی انسان‌ها متصور شد. برای کشف صحت کارکرد برنامه‌های کامپیوتری روش های متفاوتی ابداع شده‌اند که در ادامه به طور مختصر از آن‌ها یاد می‌کنیم اما پیش از آن به یک خاصیت مشترک همه ی این روش‌ها می‌پردازیم که ناکامل بودن است. منظور از ناکامل بودن این است که با استفاده از هیچ یک از روش‌هایی که داریم نمی‌توانیم هر خاصیتی را برای هر برنامه‌ای بررسی کنیم. به عبارت دیگر استفاده از هر روشی، محدودیت‌هایی دارد. و البته قضیه رایس [۱۹] به ما این تضمین را داده که روش کاملی اصلا وجود ندارد. قضیه رایس به طور غیر رسمی بیان می‌کند که مسئله ی بررسی هر خاصیت غیر بدیهی، برای همه ی برنامه‌ها، تصمیم ناپذیر است. این دلیلی بر این شده که روش‌های مختلفی برای این کار درست شوند که هر کدام می‌توانند حالت‌های خاصی از مسأله را حل بکنند.

یک دسته‌بندی برای این روش‌ها تقسیم آن‌ها به دو دسته‌ی پویا و ایستا است. روش‌های پویا روش‌هایی هستند که در آن‌ها تست برنامه با اجرای برنامه همراه است و روش‌های ایستا بدون اجرای برنامه‌ها انجام می‌شوند. روش‌های پویا معمولاً با اجرای حالات محدودی از برنامه، تصمیم می‌گیرند که برنامه‌ای که نوشته‌ای ایم، انتظاراتمان را برآورده می‌کند یا خیر. اگر این روش بتواند تشخیص دهد برنامه‌ای درست کار نمی‌کند می‌توانیم با اطمینان نتیجه بگیریم که برنامه غلط نوشته‌شده، اما اگر برنامه‌ای از تست‌های ساخته‌شده با این روش‌ها با موفقیت عبور کند، نمی‌توان اطمینان حاصل کرد که برنامه درست کار می‌کند، زیرا ممکن است حالتی مشکل‌زا از اجرای برنامه وجود داشته باشد که در تست ها نیامده باشد. در کتاب [۱۷] به توضیح این روش‌ها پرداخته شده. این دسته از روش‌ها از موضوع اصلی کار ما دور هستند. روش‌های ایستا معمولاً روش‌هایی هستند که از

نظریه‌های مختلف در منطق ریاضی به عنوان ابزار بهره می‌برند تا بدون اجرای خود برنامه‌ها در مورد صحت اجرای آن‌ها نتیجه‌گیری کنند. به همین دلیل به بخشی مهم و بزرگی از این دستورات که از منطق استفاده می‌کنند روش‌های صوری هم گفته‌می‌شود. از معروف ترین این روش‌ها وارسی مدل، روش‌های استنتاجی و استفاده از نظریه تعبیر مجرد است. در روش وارسی مدل، یک مدل صوری متناهی از برنامه‌ی موردبررسی می‌سازیم که همه‌ی حالات اجرای برنامه با آن قابل‌توصیف است، سپس با استفاده از یک زبان صوری که بتواند در مورد مدل هایمان صحبت‌کند، ویژگی‌های مورد بررسیمان را بیان‌می‌کنیم و در نهایت صحت ویژگی‌های بیان‌شده را بررسی می کنیم. مقاله [۴] شروع این روش‌ها بوده که این کار را با استفاده از نوعی مدل کریپکی [۱۵] و نوعی منطق زمانی به نام منطق زمانی خطی [۴] انجام داده که روشی است با دقت و البته هزینه‌ی محاسباتی بسیار بالا. [۱۲] یک منبع بسیار مقدماتی در این زمینه و کتاب[۵] یک مرجع سنتی در این زمینه است. در روش‌های استنتاجی که شاید بتوان یکی از ابتدایی ترین آن‌ها را استفاده از منطق هور[۱۱] دانست، درستی کارکرد برنامه‌هایمان را با ارائه‌ی یک درخت اثبات در یک دستگاه استنتاجی که متناسب با زبان برنامه‌هایمان ساخته‌شده، نشان می‌دهیم. در این روش هم اگر بتوانیم درستی یک برنامه را اثبات کنیم، دیگر به‌طور تئوری، خیالی آسوده از درستی برنامه خواهیم داشت اما ساختن درخت اثبات در یک نظریه برهان می‌تواند چالش برانگیز باشد چون این یک مسئله‌ی NP-Hard است. در[۱۲] به منطق هور به طور مقدماتی پرداخته شده. همین‌طور کتاب[۱۸] نیز به پیاده‌سازی منطق هور در زبان coq پرداخته. coq نیز یک اثبات‌یار است که بر اساس یک نظریه نوع وابسته کار می‌کند. برای اطلاعات بیشتر در مورد چگونگی طرز کار این اثبات‌یار و تئوری بنیادین آن می شود به کتاب[۳] مراجعه‌کرد. تئوری مورد شرح در[۱۰] نیز می‌تواند در این مسیر به کار گرفته شود. نظریه تعبیر مجرد[۸] نیز یک نظریه ریاضیاتی است که در این بحث می‌توان گفت، به‌نوعی سعی می‌کند از روی معناشناسی یک برنامه‌ی کامپیوتری[۲۱] ، یک تقریب بسازد. منظور از تقریب، یک دستگاه کوچک‌تر از معناشناسی اصلی است که رفتارش زیرمجموعه‌ی رفتارهای دستگاه اصلی است. سعی بر این است که دستگاه جدیدی که می‌سازیم به لحاظ محاسباتی هم ساده‌تر از معناشناسی اصلی کار کند تا بتوانیم خواص آن را راحت‌تر بررسی کنیم. در این صورت هر نتیجه‌ای که در مورد خواص جدید بگیریم را می‌توانیم در مورد خود برنامه هم بیان کنیم اما می‌دانیم که در این صورت هم به همه‌ی حقایق دست پیدا نکرده ایم. در مورد این نظریه نیز به‌تازگی کتاب[۷] منتشر شده که حاصل نزدیک به ۵ دهه کار مبدع این نظریه، پاتریک کوزو، است. همین‌طور[۱۳] نیز در مورد پیاده‌سازی این نظریه بحث کرده.

فهرست مطالب

فصل ۱

برخی مفاهیم اولیه

در این فصل سعی شده خواننده با مفاهیم مقدماتی کار آشنا شود. بعضی از مفاهیم ممکن است مستقیما در ادامه استفاده شده باشند و بعضی دیگر ممکن است مستقیما در ادامه وارد بحث نشده باشند اما آشنایی با آن‌ها برای درک بهتر واجب است.

۱.۱ نظریه تعبیر مجرد

در این بخش مفاهیم اولیه‌ی نظریه تعبیر مجرد را معرفی می‌کنیم.

تعریف ۱.۱. (ترتیب جزئی): اگر $\mathbf{P}$ یک مجموعه باشد و $\leq$ یک رابطه روی این مجموعه باشد به‌طوریکه:

$$1.\forall \mathbf{a} \in \mathbf{P} : \mathbf{a} \leq \mathbf{a}$$

$$2.\forall \mathbf{a}, \mathbf{b} \in \mathbf{P} : (\mathbf{a} \leq \mathbf{b} \wedge \mathbf{b} \leq \mathbf{a}) \rightarrow \mathbf{a} = \mathbf{b}$$

$$3.\forall \mathbf{a}, \mathbf{b}, \mathbf{c} \in \mathbf{P} : a \leq b \wedge b \leq c \rightarrow a \leq c$$

آنگاه به زوج $(\mathbf{P}, \leq)$ یک ترتیب جزئی می‌گوییم.

در ادامه خواهیم دید که معنای برنامه‌های کامپیوتری که به یک زبان برنامه نویسی نوشته می‌شوند را می‌شود به شکل یک ترتیب جزئی دید. ترتیب‌های جزئي به عنوان یک نظریه ریاضی مطالعه شده‌اند و اگر معناشناسی یک برنامه را بتوانیم به این شکل بیان کنیم می‌توانیم از خصوصیاتی که در مورد ترتیب جزئی می‌دانیم در جهت کار با معناشناسی استفاده کنیم.

تعریف ۲.۱. (اتصال گالو): اگر $(\mathbf{C}, \sqsubseteq)$ و $(\mathbf{A}, \preceq)$ دو ترتیب جزئی باشند و دو تابع $\alpha : \mathbf{C} \to \mathbf{A}$ و $\gamma : \mathbf{A} \to \mathbf{C}$ را داشته باشیم که:

$$\forall \mathbf{c} \in \mathbf{C} : \forall \mathbf{a} \in \mathbf{A} : \alpha(\mathbf{c}) \preceq \mathbf{a} \leftrightarrow \mathbf{c} \sqsubseteq \gamma(\mathbf{a})$$

۲.۱ روش وارسی مدل

روش وارسی مدل یک روش صوری است که برای درستی‌یابی سیستم‌های مختلف استفاده می‌شود. در این روش معمولا ابتدا یک ماشین حالات متناهی از روی سیستم مورد بررسی ساخته می‌شود، سپس بررسی‌هایی که قرار است روی سیستم اصلی انجام شوند، روی این ماشین(مدل) انجام می‌شود. در بررسی صحت کارکرد برنامه‌های کامپیوتری از این روش استفاده می‌شود اما این تنها مورد استفاده از این روش نیست و هر منظومه‌ی دیگری که قابلیت بیان به صورت صوری را داشته باشد قابل بررسی با این روش هست. مثلا می‌توان از این روش برای بررسی صحت عملکرد برنامه‌ی قطارهای شهری استفاده کرد؛ در حالتی که مثلا خصوصیات مورد بررسی ما عدم رخ دادن تصادف بین قطارها یا پوشش تمام مناطق شهر باشد. مثال های دیگر استفاده‌ی این روش در علوم کامپیوتر می تواند بررسی صحت عملکرد یک پردازنده یا مثلا الگوریتم تقسیم وظایف یک سیستم عامل باشد. این مثال‌ها هیچ یک برنامه‌ی کامپیوتری نیستند(هر چند که ممکن است مجبور باشیم از یک برنامه‌ی کامپیوتری برای پیاده سازی آن‌ها کمک بگیریم که در آن صورت بررسی صحت عملکرد آن برنامه‌ی کامپیوتری داستانی دیگر خواهد داشت) اما قابل بیان به صورت صوری به جای زبان طبیعی هستند.

ایده‌ی روش وارسی مدل از منطق‌های زمانی مختلف استفاده می‌کند. منطق زمانی یک نوع منطق موجهات است. منطق‌های موجهات از گسترش زبان منطق کلاسیک با اضافه کردن ادات وجهی گوناگون، با معانی متفاوتی که ممکن است در زبان طبیعی داشته باشند، ساخته می‌شوند. این ادوات غالبا در زبان طبیعی نقش قید را دارند. منطق‌های زمانی بخشی از منطق‌های موجهات هستند که به صوری‌گری ما مفهوم زمان را هم اضافه می‌کنند، یعنی قیدهایی مانند فعلا، بعدا و قبلا. منطقی که در اینجا بیان می‌کنیم LTL نام دارد که یکی از منطق‌های زمانی است که برای روش وارسی مدل استفاده می‌شود.

ابتدا زبان این منطق را بیان می‌کنیم و سعی می‌کنیم به طور غیر دقیق در مورد معنای فرمول‌های این زبان به خواننده یک درک شهودی بدهیم؛ سپس به سراغ معناشناسی صوری این منطق می‌رویم.

۱.۲.۱ زبان LTL

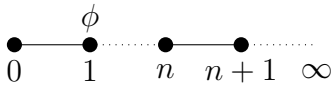
تعریف ۳.۱. هر عضو مجموعه‌ی Φ یک فرمول در زبان LTL است(و Π مجموعه‌ی فرمول‌های اتمی است و $\pi \in \Pi$):

$$\Pi \subset \Phi,$$

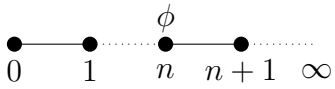
$$\phi \in \Phi \Leftrightarrow \phi ::= \pi|\phi \wedge \phi|\phi \vee \phi|\neg\phi|\phi \rightarrow \phi|\bigcirc \phi|\diamond \phi|\Box\phi|\phi\mathcal{U}\phi|\phi\mathcal{R}\phi|\phi\mathcal{W}\phi|\phi\mathcal{M}\phi$$

اولین نکته‌ای که برای فرمول‌های این زبان به آن نیاز داریم این است که در این منطق ما زمان را با اعداد طبیعی و هر خاصیتی که در موردشان تعریف شده نشان می‌دهیم. یعنی برای یک فرمول، زمان از عدد ۰ شروع شده و تا ابد ادامه خواهد داشت و حین گذر زمان ممکن است ارزش فرمول‌ها تغییر کند. مسلما پس از بررسی معناشناسی صوری بهتر می‌شود این مفهوم را به طور شهودی حس کرد، اما به هر حال به خواننده پیشنهاد می‌شود پیش از رسیدن به آن بخش به ادامه‌ی این بخش هم توجه شود.

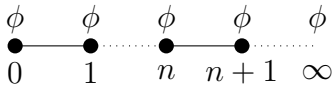
در این زبان ادوات کلاسیک $\neg, \rightarrow, \wedge, \vee$ هستند با همان معنایی که در منطق گزاره‌ای کلاسیک داشتند. در ادوات جدید $\bigcirc\phi$ به معنای برقرار بودن این فرمول دقیقا در لحظه‌ی بعدی(دقیقا یک لحظه) است، مثلا در شکل زیر با در نظر گرفتن اینکه در زمان ۰ هستیم، این فرمول در لحظه‌ی ۱ برقرار است.



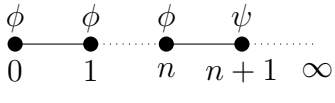
$\diamond\phi$ به معنای برقرار بودن این فرمول در یک لحظه‌ای در آینده است(که الزاما لحظه‌ی بعدی نیست، مثلا لحظه‌ی nام).



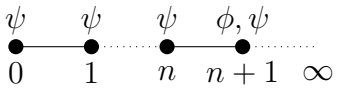
$\Box\phi$ به معنای برقرار بودن این فرمول در همه‌ی لحظات آینده است که دوگان ادات قبلی است(با این فرض که در زمان ۰ هستیم و این حرف زده شده).

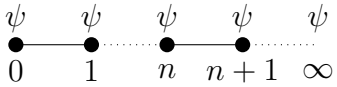


$\phi\mathcal{U}\psi$ به این معنی است که فرمول سمت چپی حداقل تا قبل از اینکه فرمول سمت راستی برقرار شود، برقرار است.(مثلا اگر بگوییم "تا وقتی که باران نباریده زمین خشک است" در این صورت "زمین خشک است" به جای فرمول سمت چپ و "باران باریده است" فرمول سمت راست است).

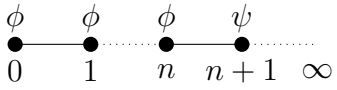


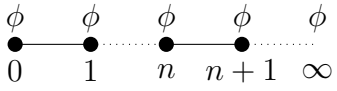
$\phi\mathcal{R}\psi$ که دوگان ادات قبلی است، به این معنی است که اگر خبر از وجود زمانی داشته باشیم که در آن فرمول سمت چپی برقرار باشد، تا آن لحظه و در خود آن لحظه فرمول سمت راستی برقرار است و اگر چنین لحظه‌ای وجود نداشته باشد، فرمول سمت راستی همیشه برقرار خواهد بود.



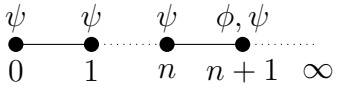


$\phi \mathcal{W} \psi$ که حالت ضعیف‌تر $\mathcal{U}$ است به این معنی است که فرمول سمت چپی حداقل تا قبل از اینکه فرمول سمت راستی برقرار شود، برقرار است و اگر خبر از زمانی نداشته باشیم که در آن فرمول سمت راستی برقرار باشد، فرمول سمت چپی تا ابد برقرار خواهد ماند.





$\phi \mathcal{M} \psi$ نیز دوگان ادات قبلی است و به این معنی است که تا قبل از اولین لحظه‌ای که در آن فرمول سمت چپ برقرار شده، به همراه همان لحظه، فرمول سمت راست برقرار است.



حال که به درکی شهودی از معنای فرمول‌های این زبان رسیده‌ایم، به بیان صوری این مفاهیم می‌پردازیم.

۲.۲.۱ معناشناسی LTL

معناشناسی صوری این منطق با استفاده از مدل‌های چهار مولفه‌ای $M = \langle S, T, V, I \rangle$ تعریف می‌شود. در این سه مولفه، به مولفه اول یعنی S مجموعه‌ی لحظه‌ها(یا جهان‌ها) می‌گوییم، T یک رابطه است روی S که قالب مدل ما را مشخص می‌کند و V یک تابع است که در هر لحظه ارزش گزاره‌های اتمی را مشخص می‌کند که با کمک آن می‌شود ارزش هر گزاره را در هر لحظه تعیین کرد و در نهایت I نیز یک عضو از S است که به عنوان لحظه‌ی ابتدایی مدل (که زمان در آن برابر ۰ است) در نظر گرفته می‌شود.

فصل ۲

صوری‌گری جدید برای روش وارسی مدل

محوریت کار ما قرار است[۶] باشد که در آن سعی شده روش وارسی مدل با کمک نظریه تعبیر مجرد، بهبود داده شود. در[۴] روشی که معرفی شده درواقع ویژگی برنامه‌ها را به کمک منطق های LTL یا CTL بیان می‌کند. خود برنامه‌ها هم با کمک معناشناسی این منطق ها که نوع خاصی از مدل های کریپیکی به اسم سیستم گذار هستند توصیف می شوند. اما در[۶] کاری که انجام شده به این شکل است که منطق های LTL و CTL با عبارات منظم[۱۴] جایگزین شده‌اند. علت این کار دو نکته بوده، اولی اینکه استفاده از عبارات منظم به جای منطق های نام برده شده می‌تواند برای برنامه نویسان ساده‌تر باشد و دومی اینکه عبارت منظم از از منطق های نام برده شده قدرت بیان بالاتری دارد.[۲۲] در ادامه ی کار وارسی مدل با استفاده از موجودات جدید تعریف شده به سه شکل مختلف بیان شده. در هر مرتبه بیان وارسی مدل، به‌گفته‌ی نویسنده، "ساختارمند"تر شده. می‌توان دریافت که فایده ساختارمندتر بودن بیان این است که پیاده‌سازی راحت‌تری دارد.
حال به بیان و بررسی تعاریف و خواص آن‌ها می‌پردازیم.

۱.۲ نحو زبان مورد بررسی

زبان بیان برنامه ها زیرمجموعه ای از دستورات زبان C است، به شکل زیر:

$$x, y, ... \in \mathbb{X}$$

$$A \in \mathbb{A} ::= 1 \mid x \mid A_1 - A_2$$

$$B \in \mathbb{B} ::= A_1 < A_2 \mid B_1 \text{ nand } B_2$$

$$E \in \mathbb{E} ::= A \mid B$$

$$S \in \mathbb{S} ::=$$

$$
\begin{aligned}
&\quad \mathsf{x = A;} \\
&\mid \ \mathsf{;} \\
&\mid \ \mathsf{if\ (B)\ S} \mid \mathsf{if\ (B)\ S\ else\ S} \\
&\mid \ \mathsf{while\ (B)\ S} \mid \mathsf{break;} \\
&\mid \ \mathsf{\{Sl\}} \\
\mathsf{Sl} \in \mathbb{SL} &::= \mathsf{Sl\ \ S} \mid \epsilon \\
\mathsf{P} \in \mathbb{P} &::= \mathsf{Sl}
\end{aligned}
$$

در اینجا زیر مجموعه‌ای از دستورات زبان C را داریم. همین‌طور که قابل مشاهده است این زبان تا حد ممکن کوچک شده. علت این کار را بعدا عمیق‌تر حس خواهیم‌کرد. علتْ ساده‌تر شدن کار برای ارائه‌ی معناشناسی و تعبیر مجرد آن است. در اینجا راحتی آن برنامه‌نویسی که قرار است با این زبان برنامه بنویسد مطرح نبوده چون اصلا این زبان برای این کار ساخته نشده. نویسنده‌ی [۶] در اینجا صرفا می‌خواسته فرآیند را نشان دهد. اگر به فرض برای زبانی مانند پایتون بخواهیم درستی‌یابی با استفاده از روش ارائه شده را درست کینم، می‌توانیم همه‌ی راهی که در [۶] برای زبان توصیف شده، رفته‌شده را برای پایتون هم برویم و به یک تحلیل‌گر ایستا برای پایتون برسیم. در مورد قدرت بیان این زبان هم می‌توانیم بگوییم که می‌توانیم باقی اعداد را از روی عدد ۱ و عملگر منها بسازیم. مثلا ابتدا ۰ را به کمک ۱ ـ ۱ می سازیم و سپس با استفاده از ۰ می‌توانیم یکی یکی اعداد منفی را بسازیم و سپس بعد از آن به سراغ اعداد مثبت می‌رویم که با کمک ۰ و هر عدد منفی‌ای که ساختیم، ساخته می شوند. باقی اعداد و حتی باقی عملگرها(یعنی به غیر از اعداد طبیعی) نیز از روی آنچه داریم قابل‌ساختن است. در مورد عبارت‌های بولی نیز داستان به همین منوال است. یعنی اینجا صرفا ادات شفر تعریف شده و باقی عملگرهای بولی را می‌توان با استفاده از همین عملگر ساخت. باقی دستورات نیز دستورات شرط و حلقه هستند. باقی دستورات قرار است مطابق چیزی که از زبان C انتظار داریم کار بکنند. در مورد دستور break; ذکر این نکته ضروری است که اجرای آن قرار است اجرای برنامه را از دستوری بعد از داخلی‌ترین حلقه‌ای که break; داخلش قرار دارد ادامه دهد. در پایان می توان ثابت کرد که این زبان هم قدرت با مدل دیویس[۹] است.
توجه داریم که هرچه در این بخش درمورد معنای دستورات این زبان گفتیم، به هیچ وجه صوری نبود و صرفا درک شهودی ای که از معنای اجرای هریک از دستورات داشتیم را بیان کردیم. بیان صوری معنای برنامه‌ها را، که برخلاف درک شهودی‌مان قابل انتقال به کامپیوتر است، در ادامه بیان خواهیم‌کرد. طبیعتا این بیان صوری از روی یک درک شهودی ساخته شده‌است.

۲.۲ معناشناسی زبان مورد بررسی

معناشناسی زبانی را که در بخش پیش آوردیم با کمک مفاهیمی به نام برچسب و رد پیشوندی و عملگر چسباندن روی دو رود پیشوندی مختلف تعریف خواهیم‌کرد و نام این معناشناسی نیز معناشناسی رد پیشوندی است.

۱.۲.۲ برچسب‌ها

باوجود اینکه خود زبان C در قسمتی از زبان خود چیزهایی به نام برچسب دارد اما همین‌طور که در بخش پیشین دیدیم، در زبانی که اینجا در حال بحث روی آن هستیم خبری از برچسب‌ها نیست. اما برای تعریف صوری معنای برنامه‌ها، به شکلی که مورد بحث است، به آن‌ها نیاز است. در این بخش ابتدا به توضیحی مختصر در مورد برچسب‌ها در معناشناسی زبان مورد بحث می‌پردازیم. تعاریف صوری دقیق این موجودات در پیوست [۶] آورده‌شده اند. برای اختصار از آوردن مستقیم این تعاریف در اینجا خودداری کرده‌ایم. در زبانمان Sها بخشی از موجودات موجود در زبان هستند. برچسب ها را برای Sها تعریف می‌کنیم. برچسب‌ها با کمک توابع labs, in, brks-of, brk-to, esc, aft, at تعریف می‌شوند. درواقع هر S به ازای بعضی از این توابع یک برجسب دارد و این‌ها درواقع نشان‌دهنده‌ی آن برچسب هستند. بعض دیگر این توابع برای هر S ممکن است یک مجموعه از برچسب‌ها را تعیین کند و یکی از آن‌ها هم با گرفتن S یک مقدار بولی را برمی‌گرداند.

at[S] : برچسب شروع S

aft[S] : برچسب پایان S، اگر پایانی داشته باشد

esc[S] : یک مقدار بولی را بازمی‌گرداند که بسته به اینکه در S دستور break; وجود دارد یا خیر، مقدار درست یا غلط را برمی‌گرداند.

brk-to[S] : برچسبی است که اگر حین S دستور break; اجرا شود، برنامه از آن نقطه ادامه پیدا می کند.

brks-of[S] : مجموعه‌ای از برچسب break; های S را برمی‌گرداند.

in[S] : مجموعه‌ای از تمام برچسب‌های درون S را برمی‌گرداند.

labs[S] : مجموعه‌ای از تمام برچسب‌هایی که با اجرای S قابل دسترسی هستند را برمی‌گرداند.

۲.۲.۲ رد پیشوندی

پس از تعریف برچسب‌ها به سراغ تعریف رد پیشوندی می‌رویم. پیش از آن باید وضعیت‌ها و محیط‌ها را تعریف کنیم.

تعریف ۱.۲. (محیط): به ازای مجموعه مقادیر $\mathbb{V}$ و مجموعه متغیرهای $\mathbb{X}$ تابع $\rho : \mathbb{X} \to \mathbb{V}$ را یک محیط می‌گوییم. مجموعه‌ی همه‌ی محیط‌ها را با $\mathbb{EV}$ نمایش می‌دهیم.

تعریف ۲.۲. (وضعیت): به هر زوج مرتبْ به ترتیب متشکل از یک برچسب l و یک محیط ρ یک وضعیت $\langle l, \rho \rangle$ می‌گوییم. مجموعه‌ی همه‌ی وضعیت‌ها را با $\mathbb{S}$ نشان می‌دهیم.

تعریف ۳.۲. (رد پیشوندی): به یک دنباله از وضعیت‌ها(با امکان تهی بودن) یک رد پیشوندی می‌گوییم.

هر رد پیشوندی یک دنباله است که قرار است توصیفی از چگونگی اجرای برنامه باشد. وضعیت‌ها همان‌طور که از نامشان پیداست قرار است موقعیت لحظه‌ای برنامه را توصیف کنند. l قرار است برچسب برنامه‌ی در حال اجرا باشد و ρ مقدار متغیرها را در آن موقع از اجرای برنامه نشان می‌دهد. دنباله‌های ما می‌توانند متناهی یا نامتناهی باشند. مجموعه‌ی ردهای پیشوندی متناهی را با $\mathbb{S}^+$ و مجموعه‌ی ردهای پیشوندی نامتناهی را با $\mathbb{S}^\infty$ نمایش می‌دهیم. مجموعه‌ی همه‌ی ردهای پیشوندی را هم با $\mathbb{S}^{+\infty}$ نمایش می‌دهیم. باتوجه به آنچه گفتیم، یک عملگر چسباندن $\bowtie$ را روی ردهای پیشوندی تعریف می‌کنیم.

تعریف ۴.۲. (عملگر چسباندن): اگر داشته باشیم $\pi_1, \pi_2 \in \mathbb{S}^{+\infty}, \sigma_1, \sigma_2 \in \mathbb{S}$ داریم:

اگر $\pi_1 \in \mathbb{S}^+$ داریم	$\pi_1 \bowtie \pi_2 = \pi_1$
اگر $\sigma_1 \neq \sigma_2$	$\pi_1 \bowtie \pi_2$ تعریف نشده است
اگر $\pi_1 \in \mathbb{S}^\infty$ داریم	$\pi_1\sigma_1 \bowtie \sigma_1\pi_2 = \pi_1\sigma_1\pi_2$

همینطور ϵ هم یک رد پیشوندی است که حاوی هیچ وضعیتی نیست. به عبارت دیگر یک دنباله‌ی تهی است.

۳.۲.۲ تعریف صوری معناشناسی رد پیشوندی

در این بخش قرار است دو تابع $\mathcal{A}$ و $\mathcal{B}$ را به ترتیب روی عبارات حسابی و بولی زبانمان یعنی Aها و Bها تعریف کنیم سپس با کمک آنها $\mathcal{S}^*$ را روی مجموعه‌ای از اجتماع معنای Sها و SIها تعریف می کنیم. پس در نهایت هدف ما تعریف $\mathcal{S}^*$ است.

تعریف ۵.۲. (معنای عبارات حسابی ـ تابع $\mathcal{A}$): تابع $\mathcal{A} : \mathbb{A} \to \mathbb{EV} \to \mathbb{V}$ را به صورت بازگشتی روی ساختار $\mathsf{A} \in \mathbb{A}$ به شکل زیر تعریف می‌کنیم:

$$\mathcal{A}[1]\rho = 1$$

$$\mathcal{A}[\mathsf{x}]\rho = \rho(\mathsf{x})$$

$$\mathcal{A}[\mathsf{A}_1 - \mathsf{A}_2]\rho = \mathcal{A}[\mathsf{A}_1]\rho - \mathcal{A}[\mathsf{A}_2]\rho$$

تعریف ۶.۲. (معنای عبارات بولی ـ تابع $\mathcal{B}$): تابع $\mathcal{B} : \mathbb{B} \to \mathbb{EV} \to \mathbb{BOOL}$ را به صورت بازگشتی روی ساختار $\mathsf{B} \in \mathbb{B}$ به شکل زیر تعریف می‌کنیم:

$$\mathcal{B}[\mathsf{A}_1 < \mathsf{A}_2]\rho = True \quad \text{اگر } \mathcal{A}[\mathsf{A}_1]\rho \text{ کوچکتر از } \mathcal{A}[\mathsf{A}_2]\rho \text{ باشد}$$

$$\mathcal{B}[\mathsf{A}_1 < \mathsf{A}_2]\rho = False \quad \text{اگر } \mathcal{A}[\mathsf{A}_1]\rho \text{ بزرگتر از } \mathcal{A}[\mathsf{A}_2]\rho \text{ باشد}$$

$$\mathcal{B}[\mathsf{B}_1 \mathsf{nand} \mathsf{B}_2]\rho = \neg(\mathcal{B}[\mathsf{B}_1]\rho \wedge \mathcal{B}[\mathsf{B}_2]\rho)$$

طبعا $\wedge$ و $\neg$ در فرازبان هستند.

در ادامه به تعریف $\mathcal{S}^*$ می‌پردازیم. این کار را با تعریف $\mathcal{S}^*$ روی هر ساخت S و SI انجام می‌دهیم. پیش از ادامه‌ی بحث باید این نکته را درمورد علامت‌گذاری‌هایمان ذکر کنیم که منظور از $\mathsf{S} ::= l\mathsf{break};$ این است که تاکید کرده‌ایم که S با برچسب l شروع شده‌است وگرنه همین طور که گفتیم l جزو زبان نیست.

تعریف ۷.۲. (معنای برنامه‌ها ـ تابع $\mathcal{S}^*$): اگر $\mathsf{S} ::= \mathsf{break};$ باشد، ردهای پیشوندی متناظر با اجرای این دستور را به شکل مجموعه‌ی زیر تعریف می کنیم:

$$\mathcal{S}^*[\mathsf{S}] = \{\langle at[\mathsf{S}], \rho\rangle | \rho \in \mathbb{EV}\} \cup \{\langle at[\mathsf{S}], \rho\rangle\langle brk - to[\mathsf{S}], \rho\rangle | \rho \in \mathbb{EV}\}$$

اگر $\mathsf{S} ::= \mathsf{x} = \mathsf{A};$ باشد، ردهای پیشوندی متناظر با اجرای این دستور را به شکل مجموعه‌ی زیر تعریف می کنیم:

$$\mathcal{S}^*[\mathsf{S}] = \{\langle at[\mathsf{S}], \rho\rangle | \rho \in \mathbb{EV}\} \cup \{\langle at[\mathsf{S}], \rho\rangle\langle aft[\mathsf{S}], \rho[\mathsf{x} \leftarrow \mathcal{A}[\mathsf{A}]\rho]\rangle | \rho \in \mathbb{EV}\}$$

اگر $\mathsf{S} ::= \mathsf{if(B)S_t}$ باشد، ردهای پیشوندی متناظر با اجرای این دستور را به شکل مجموعه‌ی زیر تعریف می کنیم:

$$\mathcal{S}^*[\mathsf{S}] = \{\langle at[\mathsf{S}], \rho\rangle | \rho \in \mathbb{EV}\} \cup \{\langle at[\mathsf{S}], \rho\rangle\langle aft[\mathsf{S}], \rho\rangle | \mathcal{B}[\mathsf{B}]\rho = False\}$$

$$\cup\{\langle at[\mathsf{S}],\rho\rangle\langle at[\mathsf{S_t}],\rho\rangle\pi|\mathcal{B}[\mathsf{B}]\rho = True \wedge \langle at[\mathsf{S_t}],\rho\rangle\pi \in \mathcal{S}[\mathsf{S_t}]\}$$

اگر $\mathsf{S} ::= \mathsf{if(B)S_t elseS_f}$ باشد، ردهای پیشوندی متناظر با اجرای این دستور را به شکل مجموعه‌ی زیر تعریف می کنیم:

$$\mathcal{S}[\mathsf{S}] = \{\langle at[\mathsf{S}],\rho\rangle|\rho \in \mathbb{EV}\}$$

$$\cup\{\langle at[\mathsf{S}],\rho\rangle\langle at[\mathsf{S_f}],\rho\rangle\pi|\mathcal{B}[\mathsf{B}]\rho = False \wedge \langle at[\mathsf{S_f}],\rho\rangle\pi \in \mathcal{S}[\mathsf{S_f}]\}$$
$$\cup\{\langle at[\mathsf{S}],\rho\rangle\langle at[\mathsf{S_t}],\rho\rangle\pi|\mathcal{B}[\mathsf{B}]\rho = True \wedge \langle at[\mathsf{S_t}],\rho\rangle\pi \in \mathcal{S}[\mathsf{S_t}]\}$$

اگر $\mathsf{SI} ::= \ni$ باشد، ردهای پیشوندی متناظر با اجرای این دستور را به شکل مجموعه‌ی زیر تعریف می کنیم:

$$\mathcal{S}[\mathsf{SI}] = \{\langle at[\mathsf{SI}],\rho\rangle|\rho \in \mathbb{EV}\}$$

اگر $\mathsf{S} ::= \mathsf{SI}'$ باشد، ردهای پیشوندی متناظر با اجرای این دستور را به شکل مجموعه‌ی زیر تعریف می کنیم:

$$\mathcal{S}[\mathsf{SI}] = \mathcal{S}[\mathsf{SI}'] \cup (\mathcal{S}[\mathsf{SI}'] \bowtie \mathcal{S}[\mathsf{S}])$$

اگر $\mathsf{S} ::= \mathsf{while(B)S_b}$ باشد، ماجرا نسبت به حالات قبل اندکی پیچیده‌تر می‌شود. تابعی به اسم $\mathcal{F}$ را تعریف خواهیم‌کرد که در حقیت دو ورودی دارد. ورودی اول آن یک دستور حلقه است و ورودی دوم آن یک مجموعه. به عبارتی دیگر می‌توانیم بگوییم به ازای هر حلقه یک تابع $\mathcal{F}$ جداگانه تعریف می‌شود که مجموعه‌ای از ردهای پیشوندی را می گیرد و مجموعه‌ای دیگر از همین موجودات را بازمی‌گرداند. کاری که این تابع قرار است انجام دهد این است که انگار یک دور دستورات داخل حلقه را اجرا می کند و دنباله‌هایی جدید را از دنباله‌های قبلی می‌سازد. معنای یک حلقه را کوچکترین نقطه ثابت این تابع در نظر می‌گیریم. در ادامه تعریف $\mathcal{F}$ آمده. با دیدن تعریف می توان به دلیل این کار پی‌برد. آن نقطه‌ای که دیگر $\mathcal{F}$ روی آن اثر نمی‌کند یا حالتی است که در آن دیگر شرط حلقه برقرار نیست و اصولا قرار نیست دستورات داخل حلقه اجرا شوند که طبق تعریف $\mathcal{F}$ می‌توانیم ببینیم که $\mathcal{F}$ در این حالت چیزی به ردهای پیشوندی اضافه نمی‌کند. یا اینکه حلقه به دستور break; خورده که در آن صورت وضعیتی به ته ردهای پیشوندی اضافه می‌شود که برچسبش خارج از مجموعه برچسب دستورات حلقه است و همین اضافه کردن هر چیزی را به ته ردهای پیشوندی موجود، توسط $\mathcal{F}$ غیرممکن می‌کند. بنابراین نقطه ثابت مفهوم مناسبی است برای اینکه از آن در تعریف صوری معنای حلقه استفاده کنیم. علت اینکه کوچکترین نقطه ثابت

را به عنوان معنای حلقه در نظر می‌گیریم هم این است که مطمئن هستیم کوچکترین نقطه ثابت، هر رد پیشوندی ای را در خود داشته باشد به معنای اجرای برنامه مرتبط است. برای درک بهتر این نکته می‌توان به این نکته توجه کرد که با اضافه کردن وضعیت‌هایی کاملا بی‌ربط به اجرای برنامه به ته ردهای پیشوندی، که صرفا برچسب متفاوتی با آخرین وضعیت هر رد پیشوندی دارند، نقطه ثابت جدیدی ساخته ایم. پس اگر خودمان را محدود به انتخاب کوچکترین نقطه ثابت نکنیم، به توصیفات صوری خوبی از برنامه‌ها دست پیدا نخواهیم‌کرد. در مورد نقطه ثابت تنها این نکته باقی می‌ماند که اصلا از کجا می‌دانیم که چنین نقطه ثابتی وجود دارد که در این صورت باید گفت مجموعه‌هایی که از ردهای پیشوندی تشکیل می‌شوند با عملگر زیرمجموعه بودن یک مشبکه را تشکیل می‌دهند و بنا به قضیه تارسکی[۲۰] برای چنین موجودی نقطه ثابت وجود دارد. تعاریف موجوداتی که درموردشان صحبت کردیم به این شکل است:

$$\mathcal{S}[\mathsf{S}] = lfp^{\subseteq}\, \mathcal{F}[\mathsf{S}]$$

$$\mathcal{F}[\mathsf{S}]X = \{\langle at[\mathsf{S}], \rho\rangle | \rho \in \mathbb{EV}\} \cup$$

$$\{\pi_2\langle l, \rho\rangle\langle aft[\mathsf{S}], \rho\rangle | \pi_2\langle l, \rho\rangle \in X \wedge \mathcal{B}[\mathsf{B}]\rho = False \wedge l = at[\mathsf{S}]\} \cup$$

$$\{\pi_2\langle l, \rho\rangle\langle at[\mathsf{S_b}], \rho\rangle\pi_3 | \pi_2\langle l, \rho\rangle \in X \wedge \mathcal{B}[\mathsf{B}]\rho = True \wedge$$

$$\langle at[\mathsf{S_b}], \rho\rangle\pi_3 \in \mathcal{S}[\mathsf{S_b}] \wedge l = at[\mathsf{S}]\}$$

اگر ;=:: S باشد، ردهای پیشوندی متناظر با اجرای این دستور را به شکل مجموعه‌ی زیر تعریف می کنیم:

$$\mathcal{S}[\mathsf{S}] = \{\langle at[\mathsf{S}], \rho\rangle | \rho \in \mathbb{EV}\} \cup \{\langle at[\mathsf{S}], \rho\rangle\langle aft[\mathsf{S}], \rho\rangle | \rho \in \mathbb{EV}\}$$

اگر {Sl} =:: S باشد، ردهای پیشوندی متناظر با اجرای این دستور را به شکل مجموعه‌ی زیر تعریف می کنیم:

$$\mathcal{S}[\mathsf{S}] = \mathcal{S}[\mathsf{Sl}]$$

در ادامه چند مثال می‌زنیم تا مفهوم موجودات تعریف شده مشخص‌تر شود.

مثال ۸.۲.

مثال ۹.۲.

مثال ۱۰.۲.

۳.۲ ویژگی‌های معنایی برنامه‌ها

تا به اینجای کار یک زبان آورده‌ایم و برای آن معنا تعریف کرده‌ایم. در این فصل می‌خواهیم در مورد ویژگی‌های برنامه‌هایی که در این زبان نوشته می‌شوند با توجه به معنای صوری‌ای که تعریف کرده‌ایم، صحبت کنیم. دقت شود که برای برنامه‌هایی که در یک زبان برنامه‌نویسی نوشته می‌شوند می‌توان به اشکال مختلفی ویژگی تعریف کرد؛ مثلا ویژگی‌های نحوی، مثل اینکه طول برنامه چند خط است یا هر کاراکتر چند بار به کار رفته، یا ویژگی‌های محاسباتی، مثل بررسی سرعت برنامه یا میزان استفاده‌ی آن از حافظه که عموما در نظریه الگوریتم و پیچیدگی محاسبات بررسی می‌شود. منظور ما در اینجا از تعریف ویژگی، متناسب است با معناشناسی‌ای که برای برنامه‌هایمان تعریف کرده‌ایم. معناشناسی‌ای که تعریف کرده‌ایم درواقع سیر محاسباتی برنامه را توصیف می‌کند و ما می‌خواهیم ویژگی‌ها را با توجه به این موضوع تعریف کنیم. در این صورت می‌توانیم صحت عملکرد برنامه‌ها را با توجه به صادق بودن ویژگی‌هایی که در مورد آن‌ها تعریف شده بفهمیم.
ابتدا به تعریف ویژگی‌ها می‌پردازیم، سپس به سراغ تعریف یک نوع عبارت منظم می‌رویم که از آن برای بیان ویژگی‌ها استفاده می‌شود.

۱.۳.۲ ویژگی‌های معنایی

همان‌طور که در بخش قبلی دیدیم، معنای هر برنامه با یک مجموعه‌ی $\mathcal{S}^*[\![S]\!]$ مشخص می‌شود. وقتی می‌خواهیم ویژگی‌هایی را برای موجوداتی که به کمک مجموعه‌ها تعریف شده اند بیان کنیم، اینکه ویژگی‌ها را هم با مجموعه‌ها بیان کنیم کار معقولی به نظر می‌رسد. مثل اینکه بخواهیم ویژگی زوج بودن را در مورد اعداد طبیعی بیان کنیم. می توانیم مجموعه‌ی $\mathbb{E}$ را به عنوان مجموعه‌ی همه‌ی اعداد زوج در نظر بگیریم و اینکه یک عدد زوج هست یا نه را عضویتش در مجموعه‌ی $\mathbb{E}$ تعریف کنیم. پس یعنی در مورد اعداد طبیعی قرار است هر ویژگی به شکل زیرمجموعه‌ای از تمام این اعداد در نظر گرفته شود. یعنی هر عضو $P(\mathbb{N})$ بنا به تعریف ما یک ویژگی از اعداد طبیعی است. در مورد برنامه‌ها نیز قرار است همین رویه را پیش بگیریم. تابع $\mathcal{S}^*$ از نوع $\mathbb{P} \to P(\mathbb{S}^+)$ است. یعنی یک برنامه را در ورودی می‌گیرد و یک مجموعه از ردهای پیشوندی را باز می‌گرداند. پس می‌توانیم هر ویژگی را به عنوان زیر مجموعه‌ای از $P(\mathbb{S}^+)$ تعریف کنیم، به عبارت دیگر عضوی از $P(P(\mathbb{S}^+))$.

۲.۳.۲ عبارات منظم

در اینجا توصیف ویژگی‌ها برای هر برنامه باید یک چارچوب داشته باشد. در صورت قدیمی روش وارسی مدل ما از منطق های زمانی برای بیان ویژگی‌ها به صورت صوری استفاده می‌کردیم و این احتیاج به یک زبان برای صوری کردن کامل کار را، که رسیدن به بیان مسئله‌ی وارسی مدل است،

به ما نشان می‌دهد. در اینجا ما با داستان دیگری هم رو به رو هستیم و آن این است که از آنجایی که با مجموعه‌ها سر و کار داریم و مجموعه‌ها چندان موجودات ساختنی‌ای نیستند(برخلاف مدل کریپکی)، بهتر است یک موجود ساختنی مثل یک زبان صوری برای بیان آن‌ها داشته باشیم. در این فصل قصد داریم یک نوع عبارت منظم را برای این منظور تعریف کنیم. ابتدا زبان این عبارت منظم را تعریف می‌کنیم، سپس به سراغ معناشناسی آن می‌رویم.

۳.۳.۲ زبان عبارات منظم

فرق عمده‌ای که زبان عبارات منظم ما با عبارات منظم کلاسیک دارد در کاراکترهاست. کاراکترها در زبان کلاسیک موجوداتی اتمی بودند، اما در اینجا ساختار دارند. در اینجا به جای هر کاراکتر یک زوج متشکل از مجموعه‌ی L و عبارت بولی B تشکیل شده‌اند که این زوج را به شکل $\mathsf{L} : \mathsf{B}$ در زبانمان نمایش می‌دهیم.
زبان ما به شکل BNF زیر است:

تعریف ۱۱.۲.

$$\mathsf{L} \in P(\mathbb{L})$$
$$\mathsf{x}, \mathsf{y}, \ldots \in \mathbb{X}$$
$$\underline{\mathsf{x}}, \underline{\mathsf{y}}, \ldots \in \underline{\mathbb{X}}$$
$$\mathsf{B} \in \mathbb{B}$$
$$\mathsf{R} \in \mathbb{R}$$

$$
\begin{aligned}
\mathsf{R} ::= \quad & \varepsilon \\
| \quad & \mathsf{L} : \mathsf{B} \\
| \quad & \mathsf{R}_1\mathsf{R}_2 \quad (or\ \mathsf{R}_1 \bullet \mathsf{R}_2) \\
| \quad & \mathsf{R}_1 \mid \mathsf{R}_2 \quad (or\ \mathsf{R}_1 + \mathsf{R}_2) \\
| \quad & \mathsf{R}_1^* \\
| \quad & \mathsf{R}_1^+ \\
| \quad & (\mathsf{R}_1)
\end{aligned}
$$

همان طور که قابل مشاهده است در اینجا عملگرهای دوتایی چسباندن ($\bullet$) و انتخاب ($|$) را داریم، به همراه عملگرهای یگانی * و $^+$. در ادامه خواهیم دید که در فرازبان معنی عملگر یگانی $^+$ به وسیله‌ی عملگر یگانی دیگر قابل بیان است، هرچند که در زبانمان هم برای سهولت کار از بیان این عملگر اجتناب نشده. توجه شود که پرانتزها هم جزئی از زبان قرار داده شده‌اند.

همین‌طور در اینجا می‌خواهیم از تعدادی عبارات مخفف که در ادامه کارمان را راحت‌تر می‌کنند صحبت کنیم. منظور از زوج $? : \mathsf{B}$ همان $\mathbb{L} : \mathsf{B}$ است. عبارت $l : \mathsf{B}$ به جای عبارت $\{l\} : \mathsf{B}$ به کار می‌رود و منظور از عبارت $\neg l : \mathsf{B}$ نیز عبارت $\mathbb{L} \setminus \{l\} : \mathsf{B}$ است.

با یک نگاه به دستور این زبان یک نکته‌ی چشمگیر برای ما، با توجه به موجوداتی که در بخش قبل تعریف کردیم، با نگاه به قواعد این زبان می‌تواند وجود یک مجموعه‌ی $\underline{\mathbb{X}}$ در کنار $\mathbb{X}$ که از قبل داشتیم باشد. قرار است به ازای هر $\mathsf{x} \in \mathbb{X}$ یک $\underline{\mathsf{x}} \in \underline{\mathbb{X}}$ داشته‌باشیم. منظور از $\underline{\mathsf{x}}$ مقدار متغیر x در ابتدای هر برنامه است. این یعنی تابع $\underline{\rho} : \underline{\mathbb{X}} \to \mathbb{V}$ که $\mathbb{V}$ مجموعه‌ی مقادیر متغیرهاست(در بخش قبل به این اشاره نشد اما خود ρ هایی که در بخش قبل داشتیم هم از نوع $\mathbb{X} \to \mathbb{V}$ بود. با توجه به زبانمان و توضیحاتی که در گذشته دادیم، می‌توان در نظر گرفت که در اینجا $\mathbb{V}$ همان $\mathbb{Z}$ یعنی اعداد صحیح است. دلیل استفاده از نماد $\mathbb{V}$ به عنوان مجموعه مقادیر این است که روح کلی نگرانه‌ی کار حفظ شود). همان‌طور که پیش‌تر گفتیم برای اشاره به یک تابع ρ از کلمه‌ی "محیط" استفاده می‌شود. به همین منوال در ادامه برای اشاره به $\underline{\rho}$ از "محیط اولیه" استفاده می‌کنیم. برای اشاره به مجموعه‌ی همه‌ی محیط‌های اولیه هم از نماد $\underline{\mathbb{EV}}$ استفاده می‌کنیم. بقیه‌ی موجودات از جمله برچسب‌ها و عبارات بولی را هم که قبلا داشتیم.

در ادامه به بیان صوری معنای زبان بیان شده می‌پردازیم. پس از آن می‌توانیم با بررسی چند مثال، از اینکه معنای هر عبارت منظم چیست درکی شهودی به دست آوریم.

۴.۳.۲ معناشناسی عبارات منظم

معنای عبارات منظم را با استفاده از تابع $\mathcal{S}^r$ نشان می‌دهیم. این تابع به این شکل تعریف می‌شود که در ورودی یک عبارت منظم R را می‌گیرد، سپس یک مجموعه از زوج مرتب‌های(یا همان‌طور که پیش‌تر نام‌گذاری کردیم "وضعیت‌ها"ی) $\langle \underline{\rho}, \pi \rangle$ را که $\pi \in \mathbb{S}^*$ و $\underline{\rho} \in \underline{\mathbb{EV}}$ باز می‌گرداند. بنابراین این تابع از نوع $\mathbb{R} \to P(\underline{\mathbb{EV}} \times \mathbb{S}^*)$ است. همین‌طور دقت شود که تا به حال از $\mathbb{S}^*$ صحبتی نکرده بودیم و فقط $\mathbb{S}^+$ را معرفی کرده بودیم. $\mathbb{S}^*$ نیز برابر است با $\mathbb{S}^+ \cup \{\epsilon\}$ (به لحاظ معنایی همان عملگر * است که در زبان عبارات منظمهم هست، مشهور به ستاره‌ی کلینی).

تعریف استقرایی تابع $\mathcal{S}^r$ به شکل زیر است:

تعریف ۱۲.۲. تابع $\mathcal{S}^r : \mathbb{R} \to P(\underline{\mathbb{EV}} \times \mathbb{S}^*)$ به صورت استقرایی روی ساختار عبارت منظم R به صورت زیر تعریف می‌شود:

$$\mathcal{S}^r[\varepsilon] = \{\langle \underline{\rho}, \epsilon \rangle | \underline{\rho} \in \underline{\mathbb{EV}}\}$$

[یعنی معنای عبارت منظم ε مجموعه‌ای شامل زوج مرتب‌هایی از محیط‌های اولیه‌ی مختلف در کنار رد پیشوندی تهی استفاده می‌کند.]

$$\mathcal{S}^r[\mathsf{L} : \mathsf{B}] = \{\langle \underline{\rho}, \langle l, \rho \rangle \rangle | l \in \mathsf{L} \wedge \mathcal{B}[\mathsf{B}]\underline{\rho}, \rho\}$$

[این یعنی معنای عبارت $\mathcal{S}^r[\mathsf{L} : \mathsf{B}]$ زوج مرتب‌هایی هستند که عضو اول آن‌ها محیط‌های اولیه مختلف هستند(مانند مورد قبلی و البته در موارد آتی!) و عضو دوم آن‌ها ردهای پیشوندی تک‌عضوی $\langle l, \rho \rangle$ هستند که در آن‌ها برچسب l باید در L که مجموعه‌ای از برچسب‌هاست حضور داشته باشد و عبارت بولی B باید درباره‌ی محیط اولیه $\underline{\rho}$ و محیط ρ برقرار باشد. حتما متوجه این نکته شدید که $\mathcal{B}$ در اینجا به جای اینکه از نوع $\mathbb{EV} \to \mathbb{BOOL}$ باشد،همان‌طور که قبلا تعریف کردیم، از نوع $\underline{\mathbb{EV}} \to \mathbb{EV} \to \mathbb{BOOL}$ است.(منظور از $\mathbb{BOOL}$ همان مجموعه‌ی $\{True, False\}$ است.) در اینجا $\mathcal{A}$ و $\mathcal{B}$ را در ادامه با نوع‌های متفاوت دوباره تعریف خواهیم کرد، که البته فرق اساسی‌ای با تعریف قبلی ندارد و صرفا گسترشی ساده از آن است.]

$$\mathcal{S}^r[\mathsf{R}_1\mathsf{R}_2] = \mathcal{S}^r[\mathsf{R}_1] \bowtie \mathcal{S}^r[\mathsf{R}_2]$$

به‌طوری که در آن برای هر دو مجموعه‌ی $\mathcal{S}$ و $\mathcal{S}'$ از ردهای پیشوندی:

$$\mathcal{S} \bowtie \mathcal{S}' = \{\langle \underline{\rho}, \pi\pi' \rangle | \langle \underline{\rho}, \pi \rangle \in \mathcal{S} \wedge \langle \underline{\rho}, \pi' \rangle \in \mathcal{S}'\}$$

[این یعنی اگر یک عبارت منظم داشته باشیم که از چسباندن R_1 و R_2 به هم ساخته شده باشد، آنگاه معنای این عبارت منظم با چسباندن ردهای پیشوندی موجود در مولفه‌ی دوم زوج مرتب‌هایی

که اعضای مجموعه‌ی معنای این دو عبارت منظم هستند و گذاشتن این رد پیشوندی‌های حاصل از چسباندنْ در معنای عبارت منظم جدید تعریف می‌شود. همین‌طور که می‌بینید یک عملگر چسباندن برای دو مجموعه از این زوج‌های $\langle \underline{\rho}, \pi \rangle$ تعریف شده و در تعریف $\mathcal{S}^r[\mathsf{R}_1\mathsf{R}_2]$ از آن کمک گرفته شده.
تا این تکه از تعریف معنای عبارت منظم که رسیده‌ایم، تا حدی به دستیابی به درکی شهودی از اینکه به چه نحوی قرار است عبارات منظم راهی برای توصیف ویژگی در مورد برنامه‌ها باشد نزدیک‌تر شده‌ایم. همان‌طور که در مورد قبل دیدیم هر زوج $\mathsf{B} : \mathsf{L}$ دقیقا به یک وضعیتْ داخل یک رد پیشوندی اشاره می‌کند. انگار که قرار است این زوج‌ها موازی با وضعیت‌ها در ردهای پیشوندی موجود در معنای یک برنامه جلو روند و منطبق باشند تا وارسی مدل انجام شود. درک این موضوع اولین قدم ماست در دیدن عصاره‌ی روش وارسی مدل در ادبیاتی که از اول این فصلْ عَلَم کرده‌ایم.]

$$\mathcal{S}^r[\mathsf{R}_1 \mid \mathsf{R}_2] = \mathcal{S}^r[R_1] \cup \mathcal{S}^r[R_2]$$

[این موردْ معنای اعمال عملگر انتخاب روی دو عبارت منظم را توصیف می‌کند. معنای اعمال این عملگر به‌سادگی به صورت اجتماع معنای هر دو عبارت منظم تعریف شده.]

$$\mathcal{S}^r[\mathsf{R}]^0 = \mathcal{S}^r[\varepsilon]$$
$$\mathcal{S}^r[\mathsf{R}]^{n+1} = \mathcal{S}^r[\mathsf{R}]^n \bowtie \mathcal{S}^r[\mathsf{R}]$$

[دو عبارت اخیر برای توصیف معنای عملگرهای * و $^+$ تعریف شده‌اند. عملگر $\bowtie$ و معنای $\mathcal{S}^r[\varepsilon]$ را هم که قبلا تعریف کرده بودیم و 0 و n و $n+1$ هم اعداد طبیعی‌اند و $+$ لاجرم همان جمع اعداد طبیعی است.]

$$\mathcal{S}^r[\mathsf{R}^*] = \bigcup_{n \in \mathbb{N}} \mathcal{S}^r[\mathsf{R}^n]$$
$$\mathcal{S}^r[\mathsf{R}^+] = \bigcup_{n \in \mathbb{N} \setminus \{0\}} \mathcal{S}^r[\mathsf{R}^n]$$

[این دو عبارت هم تعریف معنای خود دو عملگر * و $^+$ هستند. منظور از $\mathbb{N}$ مجموعه‌ی اعداد طبیعی است. همان‌طور که قبل‌تر هم اشاره شد $^+$ را می‌توان در فرازبان با * تعریف کرد. اضافه می‌کنیم که خود * را هم در فرازبان می‌توان با عملگر انتخاب تعریف کرد و در اینجا می‌توان این نکته را هم دید.]

$$\mathcal{S}^r[(\mathsf{B})] = \mathcal{S}^r[\mathsf{B}]$$

[این تکه از تعریف هم صرفا بیان می‌کند که پرانتزها تاثیری در معنای عبارات منظم ندارند که کاملا قابل انتظار است چرا که وجود پرانتز قرار است در صرفا در خواص نحوی زبان اثر بگذارد.]

تعریف معنای عبارات منظم در اینجا تمام می‌شود اما همان‌گونه که در لابه‌لای تعاریف گفتیم، احتیاج داریم که $\mathcal{A}$ و $\mathcal{B}$ را از نو تعریف کنیم:

تعریف ۱۳.۲. توابع $\mathcal{A} : \mathbb{A} \to \underline{\mathbb{EV}} \to \mathbb{EV} \to \mathbb{V}$ و $\mathcal{B} : \mathbb{B} \to \underline{\mathbb{EV}} \to \mathbb{EV} \to \mathbb{BOOL}$ به شکل استقرایی به ترتیب روی ساختارهای $\mathsf{A} \in \mathbb{A}$ و $\mathsf{B} \in \mathbb{B}$ به شکل زیر تعریف می‌شوند:

$$\mathcal{A}[1]\underline{\rho}, \rho = 1$$
$$\mathcal{A}[\underline{\mathsf{x}}]\underline{\rho}, \rho = \underline{\rho}(\mathsf{x})$$
$$\mathcal{A}[\mathsf{x}]\underline{\rho}, \rho = \rho(\mathsf{x})$$
$$\mathcal{A}[\mathsf{A}_1 - \mathsf{A}_2]\underline{\rho}, \rho = \mathcal{A}[\mathsf{A}_1]\underline{\rho}, \rho - \mathcal{A}[\mathsf{A}_2]\underline{\rho}, \rho$$
$$\mathcal{B}[\mathsf{A}_1 < \mathsf{A}_2]\underline{\rho}, \rho = \mathcal{A}[\mathsf{A}_1]\underline{\rho}, \rho \ < \ \mathcal{A}[\mathsf{A}_2]\underline{\rho}, \rho$$
$$\mathcal{B}[\mathsf{B}_1 \text{ nand } \mathsf{B}_2]\underline{\rho}, \rho = \mathcal{B}[\mathsf{B}_1]\underline{\rho}, \rho \uparrow \mathcal{B}[\mathsf{B}_2]\underline{\rho}, \rho$$

به‌راحتی قابل مشاهده است که تعاریف جدید تا حد خوبی به تعاریف قبلی شبیه هستند و فرق عمده صرفا وارد شدن $\underline{\rho}$ است.
حال به سراغ چند مثال از عبارات منظم و معنای آن‌ها می‌رویم.

مثال ۱۴.۲. فرض کنید عبارت منظم ما

مثال ۱۵.۲. این سه تا مثال باید با سه تا مثال بخش معناشناسی برنامه‌ها سینک باشن! همین طور در ادامه بعد از تعریف مدل چکینگ در این فصل هم ۳ تا مثال خواهیم داشت که قراره هر کدومشون از این ۳ تا مثال و ۳ تا مثالی که قبلا تعریف کردیم تشکیل شده باشن.

مثال ۱۶.۲.

تا اینجای کار بیشتر مفاهیمی که برای بیان صورت جدید مسئله‌ی وارسی مدل احتیاج داریم را بیان کرده‌ایم.

۵.۳.۲ واریته‌های مختلف زبان عبارات منظم

به عنوان قسمت آخر این بخشْ واریته‌های مختلفی از زبان عبارات منظم را بیان می‌کنیم. ، که هر کدام در واقع زیرمجموعه‌هایی از کل عبارات زبانی که توصیف کرده‌ایم را توصیف می‌کنند. بعضی از آن‌ها را در همین فصل برای هدف نهایی این فصل و بعضی دیگر را در فصل بعدی استفاده می‌کنیم.
اولین واریته‌ای که می‌خواهیم بیان کنیم، واریته‌ای است که در اعضای آن اصلا عبارت $\mathsf{L} : \mathsf{B}$ حضور ندارد و کل عبارت‌های زبان از ε ها تشکیل شده‌اند.

تعریف ۱۷.۲. (عبارت منظم تهی ـ $\mathbb{R}_\varepsilon$):

$$\mathsf{R} \quad \in \quad \mathbb{R}_\varepsilon$$

$$\mathsf{R} \quad ::= \quad \varepsilon \mid \mathsf{R}_1\mathsf{R}_2 \mid \mathsf{R}_1 + \mathsf{R}_2 \mid \mathsf{R}_1^* \mid \mathsf{R}_1^+ \mid (\mathsf{R}_1)$$

با توجه به بخش قبل متوجه هستیم که معنای همه‌ی این عبارت‌ها برابر $\{\langle \underline{\rho}, \epsilon \rangle\}$ خواهد بود.
واریته‌ی بعدی عبارت منظم ناتهی است.

تعریف ۱۸.۲. (عبارت منظم ناتهی ـ $\mathbb{R}^+$):

$$\mathsf{R} \quad \in \quad \mathbb{R}^+$$

$$\mathsf{R} \quad ::= \quad \mathsf{L} : \mathsf{B} \mid \varepsilon\mathsf{R}_2 \mid \mathsf{R}_1\varepsilon \mid \mathsf{R}_1\mathsf{R}_2 \mid \mathsf{R}_1 + \mathsf{R}_2 \mid \mathsf{R}_1^+ \mid (\mathsf{R}_1)$$

دلیل وجود $\varepsilon\mathsf{R}_2$ و $\mathsf{R}_1\varepsilon$ در تعریف این است که ممکن است معنای عبارتی با معنای عبارات عضو $\mathbb{R}_\varepsilon$ برابر نباشد(یعنی برابر $\langle \underline{\rho}, \epsilon \rangle$ نباشد)، اما در خود عبارتْ ε حضور داشته باشد. با این تفاصیل می‌توان دید که دو مجموعه‌ی $\mathbb{R}_\varepsilon$ و $\mathbb{R}^+$ یک افراز برای مجموعه‌ی $\mathbb{R}$ هستند، براساس اینکه معنای هر عبارت در $\mathbb{R}$ برابر $\langle \underline{\rho}, \epsilon \rangle$ هست یا خیر. بنابراین شاید به نظر برسد که تعریف یکی از آن‌ها به طور ساختاری کافی بود، اما ممکن است درجایی احتیاج داشته باشیم که ساختاری استقرایی روی هر یک از آن‌ها عَلَم کنیم یا اینکه در اثبات حکمی بخواهیم از استقرا روی یکی از این دو ساختار استفاده کنیم.
واریته‌ی آخر عبارات منظم ما نیز عبارات منظم بدون انتخاب است.

تعریف ۱۹.۲. (عبارت منظم بدون انتخاب ـ $\mathbb{R}^\dagger$):

$$\mathsf{R} \quad \in \quad \mathbb{R}^\dagger$$

$$\mathsf{R} \quad ::= \quad \varepsilon \mid \mathsf{L} : \mathsf{B} \mid \mathsf{R}_1\mathsf{R}_2 \mid \mathsf{R}_1{}^* \mid \mathsf{R}_1{}^+ \mid (\mathsf{R}_1)$$

۴.۲ صورت جدید مسئله‌ی وارسی مدل

بالاخره به هدف نهایی این فصل رسیدیم. می‌خواهیم صورت جدیدی از مسئله‌ی وارسی مدل را بیان کنیم.
پیش از ارائه‌ی تعریف وارسی مدل نیاز داریم تا عملگر بستار پیشوندی را برای یک مجموعه از ردهای پیشوندی معرفی کنیم.

تعریف ۲۰.۲. اگر $\Pi \in P(\underline{\mathbb{EV}} \times \mathbb{S}^+)$ آنگاه بستار پیشوندی Πرا به صورت زیر تعریف می‌کنیم:

$$\mathsf{prefix}(\Pi) = \{\langle \underline{\rho}, \pi \rangle | \pi \in \mathbb{S}^+ \wedge \exists\, \pi' \in \mathbb{S}^* : \langle \underline{\rho}, \pi\pi' \rangle \in \Pi\}$$

به زبان ساده‌تر همان‌طور که پیش‌تر گفتیم ردهای پیشوندی دنباله هستند و منظور از بستار یک رد پیشوندی، مجموعه‌ی همه‌ی ردهای پیشوندی ناتهی است که زیردنباله های آن رد پیشوندی هستند، در ادامه بستار یک مجموعه از ردهای پیشوندی هم می‌شود هر رد پیشوندی‌ای که زیردنباله‌ی یک رد پیشوندی موجود در آن مجموعه باشد.

مثال ۲۱.۲. اگر $\Pi = \{\langle l_1, \rho_1\rangle\langle l_2\rho_2\rangle\langle l_3, \rho_3\rangle, \langle {l_1}', {\rho_1}'\rangle\langle {l_2}'{\rho_2}'\rangle\}$ باشد(Π شامل دو عضو است) آنگاه:

$$\mathsf{prefix}(\Pi) = \{\langle l_1, \rho_1\rangle, \langle l_1, \rho_1\rangle\langle l_2\rho_2\rangle, \langle l_1, \rho_1\rangle\langle l_2\rho_2\rangle\langle l_3, \rho_3\rangle, \langle {l_1}', {\rho_1}'\rangle, \langle {l_1}', {\rho_1}'\rangle\langle {l_2}'{\rho_2}'\rangle\}$$

که شامل ۵ عضو است.

تعریف ۲۲.۲. اگر $\mathsf{P} \in \mathbb{P}, \mathsf{R} \in \mathbb{R}^+, \underline{\rho} \in \underline{\mathbb{EV}}$ آنگاه:

$$\mathsf{P}, \underline{\rho} \models \mathsf{R} \Leftrightarrow (\{\underline{\rho}\} \times \mathcal{S}^*[\mathsf{P}]) \subseteq \mathsf{prefix}(\mathcal{S}^r[\mathsf{R} \bullet (? : T)^*])$$

فصل ۳

وارسی مدل منظم

در این فصل قرار است بیانی ساختارمندتر از روش وارسی مدل داشته باشیم. اهمیت ساختارمند تر بودن در این است که بیانی که در فصل پیش داشتیم تا پیاده سازی فاصله‌ی بسیاری دارد، چون همان‌طور که پیش‌تر گفته شد مجموعه‌ها موجودات ساختنی‌ای نیستند و کار با آن‌ها حین نوشتن برنامه‌ای کامپیوتری که قرار است پیاده‌سازی روش مورد بحث ما باشد را سخت می‌کند.

فصل ۴

وارسی مدل ساختارمند

فصل ۵

نتیجه گیری

واژه‌نامه فارسی به انگلیسی

واژه‌نامه انگلیسی به فارسی

Bibliography

[1] Committee to review chinook zd 576 crash. report from the select committee on chinook zd 576., Feb 2002.

[2] A. S. E. Al. Mars climate orbiter mishap investigation boord phase i report., November 1999.

[3] A. Chlipala. *Certified Programming with Dependent Types: A Pragmatic Introduction to Coq Proof Assistant.* MIT Press, 2022.

[4] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In D. Kozen, editor, *Logic of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer, 1981.

[5] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model checking.* MIT Press, London, Cambridge, 1999.

[6] P. Cousot. Calculational design of a regular model checker by abstract interpretation. In R. M. Hierons and M. Mosbah, editors, *ICTAC*, volume 11884 of *Lecture Notes in Computer Science*, pages 3–21. Springer, 2019.

[7] P. Cousot. *Principals of Abstract Interpretation.* MIT Press, 2021.

[8] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL '77: Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 238–252. ACM Press, 1977.

[9] M. Davis and E. Weyuker. *Computability, Complexity, and Languages.* Academic Press, New York, 1983.

[10] D. Harel, D. Kozen, and J. Tiuryn. Dynamic logic. In *Handbook of philosophical logic*, pages 99–217. Springer, 2001.

[11] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 1969.

[12] M. Huth and M. Ryan. *Logic in computer science : modelling and reasoning about systems*. Cambridge University Press, Cambridge [U.K.]; New York, 2004.

[13] X. R. K. Yi. *Introduction to Static Analysis: An Abstract Interpretation Perspective.* MIT Press, 2020.

[14] S. Kleene. Representation of Events in Nerve Nets and Finite Automata. In C. Shannon and J. McCarthy, editors, *Automata Studies*, pages 3–41. Princeton University Press, 1956.

[15] S. A. Kripke. A completeness theorem in modal logic1. *The journal of symbolic logic*, 24(1):1–14, 1959.

[16] J. Lions. Ariane 5 Flight 501 Failure: Report of the Inquiry Board, July 1996.

[17] G. J. Myers, C. Sandler, and T. Badgett. *The art of software testing*. John Wiley & Sons, Hoboken and N.J, 3rd ed edition, 2012.

[18] B. C. Pierce, A. Azevedo de Amorimand Chris Casinghino, M. Gaboardi, M. Greenberg, C. Hriţcu, V. Sjöberg, A. Tolmach, and B. Yorgey. *Programming Language Foundations*. Software Foundations series, volume 2. Electronic textbook, May 2018.

[19] H. G. Rice. Classes of recursively enumerable sets and their decision problems. *Transactions of the American Mathematical Society*, 74(2):358–366, 1953.

[20] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific journal of Mathematics*, 5(2):285–309, 1955.

[21] G. Winskel. *The formal semantics of programming languages - an introduction.* Foundation of computing series. MIT Press, 1993.

[22] P. Wolper. Temporal logic can be more expressive. *Inf. Control.*, 56(1/2):72–99, January/February 1983.

Abstract

Abstract goes here...

College of Science
School of Mathematics, Statistics, and Computer Science

Thesis Title

Author name

Supervisor: name
Co-Supervisor: name
Advisor: name

A thesis submitted to Graduate Studies Office
in partial fulfillment of the requirements for the degree of
B.Sc./Master of Science/Doctor of Philosophy in
Pure Mathematics/ Applied Mathematics/ Statistics/ Computer Science

yyyy