دانشگاه تهران
پردیس دانشکده‌های فنی
دانشکده مهندسی برق و کامپیوتر

توضیح خطا در شبکه‌های مبتنی بر نرم‌افزار با استفاده از استدلال مبتنی بر علیت

پایان‌نامه برای دریافت درجهٔ کارشناسی ارشد در رشتهٔ مهندسی کامپیوتر
گرایش نرم‌افزار

امیرحسین صیحانی

اساتید راهنما

دکتر حسین حجت و دکتر محمدرضا موسوی

شهریور ۱۴۰۱

چکیده

واژگان کلیدی

فهرست مطالب

فهرست کارهای باقیمانده

فصل ۱

مقدمه

در شبکه‌های کامپیوتری رفتار اجزای شبکه را می‌توان در یکی از این دو دسته قرار داد: سطح کنترل [1] و سطح داده [2]. در سطح کنترل تصمیم‌گیری در مورد چگونگی رسیدگی به ترافیک انجام می‌شود، مثلا چه پورت‌هایی باید باز شوند یا چه نوع بسته‌هایی اجازه‌ی عبور دارند. در سطح داده، رفتارهایی که در سطح کنترل تصمیم‌گیری شده است اجرا می‌شود. مثلا باز کردن پورت‌ها یا عبور دادن بسته‌هایی از یک نوع خاص رفتارهایی هستند که در سطح داده طبقه‌بندی می‌شوند. در شبکه‌های کامپیوتری فعلی رفتارهای این دو سطح در اجزای شبکه تجمیع شده‌اند. به همین دلیل یک شبکه‌ی کامپیوتری عملا یک سیستم توزیع شده‌است که شامل برنامه‌هایی است که برای هر یک از اجزای شبکه به شکل مجزا نوشته شده است و این شبکه‌ها به وضوح پیچیده هستند و مدیریت آن‌ها دشوار است [۲۳]. شبکه‌های مبتنی بر نرم‌افزار [3] برای حل این مشکل از یک نرم‌افزار متمرکز برای کل شبکه استفاده می‌کنند. به طور دقیق‌تر، در شبکه‌های مبتنی بر نرم‌افزار، رفتارهای سطح کنترل و داده از یکدیگر جدا می‌شوند. در نتیجه‌ی این جداسازی اجزای شبکه مانند سوییچ‌ها یا روترها دستگاه‌های ساده‌ای در نظر گرفته می‌شوند که تنها رفتارهای سطح داده دارند و رفتارهای سطح کنترل توسط یک نرم‌افزار متمرکز توصیف می‌شود. بنابراین، در یک شبکه‌ی مبتنی بر نرم‌افزار، مدیر شبکه یک برنامه برای مدیریت کل شبکه می‌نویسد و دیگر نیازی به برنامه‌نویسی برای تک تک اجزای شبکه ندارد.

OpenFlow [۲۶] مطرح‌ترین رابط برنامه‌نویسی [4] برای شبکه‌های مبتنی بر نرم‌افزار است. اما برنامه‌های

[1] Control Plane
[2] Data Plane
[3] Software Defined Network
[4] Application Programming Interface

نوشته شده با OpenFlow معمولا سطح پایین هستند و کار کردن با آن‌ها برای کاربر معمولا دشوار است. به همین دلیل زبان‌های برنامه‌نویسی متعددی مانند [۱۳، ۳۲، ۲۷، ۳۳، ۲۸، ۲] که با استفاده از OpenFlow امکان برنامه‌نویسی برای شبکه‌های مبتنی بر نرم‌افزار در سطح بالاتر را فراهم می‌کنند.

با توجه به نقش حیاتی شبکه‌ها در سیستم‌های کامپیوتری، اطمینان از عملکرد درست آن‌ها از اهمیت بالایی برخوردار است [۱۲]. روش‌های صوری [۵] مجموعه‌ای از زبان‌های مبتنی بر ریاضی، تکنیک‌ها و ابزارهایی برای توصیف و درستی‌سنجی سیستم‌های سخت‌افزاری و نرم‌افزاری هستند [۱۱]. شبکه‌های مبتنی بر نرم‌افزار با متمرکز کردن رفتار کنترل‌کننده‌ی شبکه و ساده‌تر کردن اجزای دیگر شبکه امکان به کار گیری چنین روش‌هایی را تسهیل کرده‌اند. روش‌های متعددی مانند [۱، ۲۰، ۳۰، ۳۶] برای درستی‌سنجی شبکه‌های مبتنی بر نرم‌افزار ارائه شده‌اند.

نت‌کت [۶] [۲] یک زبان برنامه‌نویسی شبکه‌های مبتنی بر نرم‌افزار است که بر پایه‌ی KAT [۲۲] بنا شده است. استفاده از KAT و داشتن یک سیستم معادلاتی صحیح [۷] و کامل [۸] باعث می‌شود تا اثبات درستی برنامه‌ها در نت‌کت را بتوان با روش‌های جبری و اثبات تساوی برنامه‌های مختلف توصیف شده در این زبان انجام داد. نت‌کت پویا [۹][۶] برای بهبود برخی از قابلیت‌های نت‌کت ارائه شده است که از جمله این قابلیت‌ها می‌توان به امکان توصیف به‌روز رسانی‌های شبکه و استدلال در مورد چندین بسته در شبکه اشاره کرد. در درستی‌سنجی نرم‌افزار با روش‌های صوری یک مدل از سیستم و رفتار مورد انتظار آن توصیف می‌شود و با روش‌هایی مبتنی بر الگوریتم مانند وارسی مدل [۱۰] [۱۰] می‌توان ثابت کرد که سیستم با رفتار مورد انتظار تطابق دارد یا خیر

یکی از مهم‌ترین ویژگی‌های این الگوریتم‌ها امکان تولید مثال‌نقض [۱۱] یا گواهی [۱۲] برای اثبات نقض رفتار مورد انتظار توسط سیستم را دارند. این مثال‌نقض یا گواهی‌ها با اینکه می‌توانند در مورد رفتار سیستم توضیح دهند ولی درک درستی از این که چرا ویژگی مورد نظر در سیستم نقض شده است نمی‌دهند. به دست آوردن چنین درکی از اینکه چرا سیستم به درستی و مطابق انتظار کار نمی‌کند به آنالیز و تحلیلی فراتر نیاز دارد. استفاده از علیت [۱۳] یکی از راهکارها برای به دست آوردن درک بیشتر از مشکل سیستم است. مفهوم علیت و مبانی آن قرن‌ها در متون فلسفه مورد مطالعه قرار گرفته و استدلال مبتنی بر خلاف واقع روشی است که در نهایت برای پیدا کردن علت

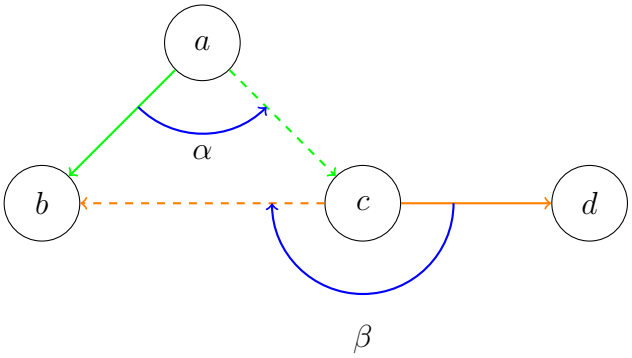
[5] Formal Methods
[6] NetKAT
[7] Sound
[8] Complete
[9] DyNetKAT
[10] Model Checking
[11] Counterexample
[12] Certificate
[13] Causality

واقعی مورد استفاده قرار گرفته است. با وجود اینکه چنین نظریه‌ای مدت‌ها پیش مورد توافق قرار گرفته است، فرمولاسیون دقیق ریاضی آن در سال‌های اخیر توسط هالپرن [۱۴] و پرل [۱۵] در [۱۷] ارائه شده است [۳]. به صورت خلاصه در [۱۷] سیستم توسط متغیرها و معادلات میان آن‌ها مدل می‌شود و شرایطی تعریف می‌شوند که تحت آن علت واقعی رویدادها پیدا می‌شوند.

شبکه‌های کامپیوتری یکی از مهم‌ترین اجزای زیرساخت سیستم‌های کامپیوتری هستند [۱۲]. شبکه‌های مبتنی بر نرم‌افزار [۱۶] با متمرکز کردن رفتار کنترل‌کننده‌ی شبکه و ساده‌تر کردن عناصر شبکه در سطح داده [۱۷] تست و درستی‌سنجی شبکه‌ها را تسهیل کرده‌اند. با این وجود به دلیل اینکه هر شبکه‌ی کامپیوتری ذاتا یک سیستم توزیع‌شده و ناهمگام [۱۸] است اطمینان از درستی شبکه‌های مبتنی بر نرم‌افزار همچنان فرآیندی پیچیده و سخت است. اما این همه‌ی ماجرا نیست. درستی‌سنجی [۱۹] و تست یک شبکه‌ی مبتنی بر نرم‌افزار یا به طور کلی یک نرم‌افزار کامپیوتری قدم اول در فرآیند اشکال‌زدایی [۲۰] آن است. روش‌های درستی‌سنجی نرم‌افزار در صورتی که سیستم مورد آزمون ویژگی [۲۱] مورد نظر را برآورده نکند یک مثال‌نقض [۲۲] پیدا کرده و کاربر بر می‌گردانند. به جز این مثال نقض اطلاعات دیگری به کاربر داده نمی‌شود و در نتیجه برای رفع مشکل سیستم کاربر مجبور است تا با روش‌های ابتکاری و با استفاده از دانش و شهود خود در مورد سیستم منشا مشکل را پیدا، آن را برطرف و فرآیند درستی‌سنجی را تکرار کند. انگیزه اصلی این پژوهش جایگزین کردن بخش انسانی این فرآیند با روش‌های خودکار است. در واقع در این پژوهش به دنبال پیدا کردن علت واقعی رخ‌دادن خطا در یک شبکه‌ی مبتنی بر نرم‌افزار هستیم.

به عنوان مثال شبکه‌ی رسم‌شده در شکل ۱.۱ را در نظر بگیرید. در این شبکه امکان انجام دو به‌روز رسانی α و β وجود دارد که به ترتیب مسیرهای سبز و نارنجی پر رنگ را با مسیرهای خط‌چین جایگزین می‌کنند. فرض کنید که ویژگی مورد انتظار در این شبکه نرسیدن بسته‌a [۲۳] ای از a به d باشد. پس از درستی‌سنجی می‌توان به مثال نقضی برای این ویژگی دست‌یافت که در آن ابتدا به روزرسانی α انجام می‌شود و سپس یک بسته از a به

[14] Joseph Y. Halpern
[15] Judea Pearl
[16] Software Defined Network
[17] Data Plane
[18] Asynchronous
[19] Verification
[20] Debug
[21] Property
[22] Counterexample
[23] Packet



شکل ۱.۱

d ارسال می‌شود. این مثال نقض ساده و کوچک است ولی در نگاه اول و بدون بررسی بیشتر کمک چندانی به اشکال‌زدایی این شبکه نمی‌کند. با بررسی بیشتر این شبکه می‌توان دریافت که انجام شدن به روز رسانی α پیش از تکمیل شدن به روز رسانی β سبب به وجود آمدن مسیر بین a و d و در نتیجه بروز خطا می‌شود. هدف از انجام این پژوهش پیدا کردن چنین عواملی (مثلا در اینجا ترتیب رخ‌دادن به‌روز رسانی‌ها) به عنوان علت واقعی رخ دادن خطا است.

۱.۱ اهداف پژوهش

شبکه‌های مبتنی بر نرم‌افزار به دلیل اینکه ذاتا توزیع‌شده و ناهمگام هستند مورد استفاده‌ی خوبی برای پیدا کردن علت واقعی خطا و تسهیل فرآیند رفع اشکال هستند. به همین دلیل در این پژوهش این دامنه از مسائل مورد بررسی قرار گرفته‌اند. برای توصیف این شبکه‌ها از زبان نت‌کت پویا [۲۴] [۶] استفاده شده است که یک زبان مینیمال و ساده بر پایه‌ی زبان نت‌کت [۲۵] [۲] که توصیف به‌روز رسانی‌های شبکه و استدلال در مورد چندین بسته موجود در شبکه را فراهم کرده است. هدف اصلی این پژوهش ارائه‌ی یک فرمولاسیون از علت واقعی [۲۶] بر اساس تعریف ارائه شده در [۱۷] برای برنامه‌های توصیف شده در زبان نت‌کت پویا و بررسی کارایی علت‌های واقعی پیدا شده در فرآیند اشکال‌زدایی از شبکه است.

[24] DyNetKAT
[25] NetKAT
[26] Actual Cause

۲.۱ ساختار فصل‌ها

در فصل دوم تعاریف و دانش پیش‌زمینه‌ی مورد نیاز برای بقیه فصول بیان می‌شود. فصل سوم روش ترجمه‌ی یک برنامه‌ی توصیف شده در زبان نت‌کت پویا به یک مدل علّی [۲۷] بیان می‌شود. فصل چهارم شامل به کار گیری روش ارائه شده در این پژوهش برای پیدا کردن علت خطا در چند دسته از ویژگی‌های رایج در شبکه است. در این فصل بررسی می‌شود که علت واقعی پیدا شده تا چه میزان با شهود موجود از مساله تطابق دارد و این فرمولاسیون تا چه حد موفق عمل می‌کند. فصل پنج شامل جمع‌بندی کارهای انجام شده در این پژوهش و بحث در مورد کاستی‌های آن و کارهای پیش‌رو است. در نهایت در فصل ششم مروری بر کارهای پیشین و مرتبط با این پژوهش انجام شده است.

[27] Causal Model

فصل ۲

تعاریف و دانش پیش‌زمینه

۱.۲ مقدمه

در این فصل مفاهیم مورد نیاز و استفاده در این پروژه مورد بررسی قرار می‌گیرند. این فصل شامل ۵ بخش است. ابتدا مفاهیم کلی شبکه‌های مبتنی بر نرم‌افزار بررسی می‌شوند. سپس زبان‌های نت‌کت و نت‌کت پویا که زبان‌های مورد استفاده در این تحقیق هستند شرح داده می‌شوند. در ادامه تعاریف اولیه ساختمان‌رویداد [۱] که به عنوان مدل معنایی در این تحقیق استفاده می‌شود شرح داده می‌شود. در بخش آخر مدل علّی [۲] که در این تحقیق برای پیدا کردن علت واقعی خطا مورد استفاده قرار می‌گیرد توصیف شده است.

۲.۲ شبکه‌های مبتنی بر نرم‌افزار

شبکه‌های کامپیوتری یکی از زیرساخت‌های حیاتی سیستم‌های کامپیوتری هستند. با وجود اینکه نیازمندی‌های این شبکه‌ها بسیار بیشتر و پیشرفته‌تر نسبت به گذشته شده است تکنیک‌ها و متدهای مدیریت و ساخت آن‌ها همچنان مانند گذشته است. این مساله مدیریت آن‌ها را در استفاده‌های امروزی پیچیده و مستعد خطا می‌کند. حتی شرکت‌های بزرگی مانند Amazon Github، یا GoDaddy مرتبا مشکلاتی در شبکه‌های خود پیدا می‌کنند

[1] Event Structure
[2] Causal Model

[۱۲] . شبکه‌های مبتنی بر نرم‌افزار یک پارادایم جدید برای طراحی و پیاده‌سازی شبکه‌های کامپیوتری هستند که علاوه بر اینکه مدیریت و درستی‌سنجی آن‌ها را با روش‌های اصولی‌تر امکان‌پذیر می‌کنند، انعطاف بالایی هم دارند. به صورت خلاصه در یک شبکه‌ی مبتنی بر نرم‌افزار رفتارهای کنترلی (تغییر و به‌روز رسانی قوانین ارسال [۳]) از عناصر شبکه مانند سوییچ‌ها یا روترها جدا می‌شوند و توسط یک کنترل‌کننده‌ی مرکزی انجام می‌شوند. به عنوان مثال در زبان OpenFlow [۲۶] رفتار سوییچ‌های شبکه تنها توسط تعدادی قانون به شکل تطبیق [۴] و اجرا [۵] توصیف می‌شود. قوانینی به این شکل ابتدا بررسی می‌کنند که بسته با قانون تطابق داشته باشد و اگر تطابق وجود داشت عملیات توصیف شده در قانون اجرا می‌شود. با ساده شدن عناصر شبکه، تمامی به‌روز رسانی‌ها و تغییرهای لازم در شبکه توسط یک کنترل کننده مرکزی انجام می‌شود. متمرکز شدن رفتار کنترلی سیستم باعث می‌شود تا استدلال و درستی سنجی در مورد رفتار شبکه آسان تر شود.

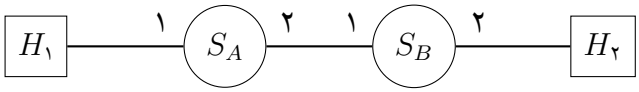
۳.۲ نت‌کت

نت‌کت ،یک زبان برای توصیف شبکه‌های مبتنی بر نرم‌افزار است [۲]. این زبان با وجود دستور زبان [۶] ساده‌ای که دارد، بر اساس KAT [۲۲] بنا شده و به همین دلیل یک سیستم معادلاتی صحیح و کامل [۷] دارد. این سیستم معادلاتی کمک می‌کند تا با استفاده از روش‌های جبری و اثبات تساوی برنامه‌های توصیف شده در این زبان بتوان در مورد آن‌ها استدلال کرد.

۱.۳.۲ دستور زبان نت‌کت

در نت‌کت هر بسته به عنوان یک نگاشت از یک مجموعه از فیلدهای $f_1, f_2, ..., f_n$ به اعداد طبیعی با تعداد ارقام ثابت در نظر گرفته می‌شود. آی‌پی [۸] های مبدا و مقصد، نوع بسته، پورت [۹] های مبدا و مقصد مثال‌هایی از

[3] Forwarding Rule
[4] Match
[5] Action
[6] Syntax
[7] Sound and Complete
[8] IP
[9] Port



شکل ۱.۲: مثال شبکه

این فیلدها هستند. دستور زبان نت‌کت به صورت زیر تعریف می‌شود:

$$a, b ::= 1 \mid 0 \mid f = n \mid a + b \mid a \cdot b \mid \neg a$$

$$p, q ::= a \mid f \leftarrow n \mid p + q \mid p \cdot q \mid p^* \mid dup$$

در این گرامر عبارت‌های a, b عملا معادل با تست‌های زبان کت [۱۰] هستند. عبارت‌های p, q عبارت‌های نت‌کت را تعریف می‌کنند که نسبت به دستور زبان کت جمله‌هایی به شکل dup و $f \leftarrow n$ به آن اضافه شده است. برای مثال شبکه‌ی ۱.۲ را در نظر بگیرید که شامل دو A [۱۱] و B و دو میزبان [۱۲] است. هر سوییچ دو پورت دارد که با شماره‌های ۱ و۲ مشخص شده‌اند. با استفاده از عبارت نت‌کت زیر می‌توانیم رفتار سوییچ‌های این شبکه را توصیف کنیم:

$$p \triangleq (dst = H_1 \cdot pt \leftarrow 1) + (dst = H_2 \cdot pt \leftarrow 2) \qquad (۱.۲)$$

این عبارت همه‌ی بسته‌هایی که مقصد آن‌ها میزبان ۱ باشد را به پورت ۱ و همه‌ی بسته‌هایی که مقصد آن‌ها میزبان ۲ باشد را به پورت شماره‌ی ۲ می‌فرستد.

۲.۳.۲ مدل معنایی نت‌کت

برای اینکه امکان استدلال در مورد مسیرهای طی شده توسط یک بسته در شبکه وجود داشته باشد،از مفهومی به نام تاریخچه‌ی بسته [۱۳] استفاده می‌شود. هر تاریخچه‌ی بسته، یک دنباله از بسته‌ها است که بسته نخست دنباله،

[10] KAT
[11] Switch
[12] Host
[13] Packet History

به عنوان بسته‌ی فعلی در نظر گرفته می‌شود. به صورت شهودی عبارت های ۱ و ۰ به ترتیب به معنای ارسال [۱۴] و رها کردن [۱۵] بدون شرط بسته هستند. عبارت $f = n$ در صورتی بسته را عبور می‌دهد که مقدار فیلد f آن برابر با n باشد. عبارت $f \leftarrow n$ مقدار n را به فیلد f بسته اختصاص می‌دهد. عبارت‌های dup باعث می‌شوند تا یک کپی از بسته‌ی فعلی ایجاد شود و به تاریخچه‌ی بسته‌ها اضافه شود. این عبارات در رفتار شبکه تاثیری ندارند اما امکان استدلال در مورد تمامی تغییرات ایجاد شده در حین جابه‌جایی بسته در شبکه را فراهم می‌سازند. به صورت دقیق، معنای هر عبارت [۱۶] نت‌کت با استفاده از معادلات زیر تعریف می‌شود:

$$[\![p]\!]H \in \mathcal{P}(H) \tag{۲.۲}$$

$$[\![1]\!]h \triangleq \{h\} \tag{۳.۲}$$

$$[\![0]\!]h \triangleq \{\} \tag{۴.۲}$$

$$[\![f = n]\!](pk :: h) \triangleq \begin{cases} \{pk :: h\} & \text{if } pk.f = n \\ \{\} & \text{otherwise} \end{cases} \tag{۵.۲}$$

$$[\![\neg a]\!]h \triangleq \{h\} \setminus ([\![a]\!]h) \tag{۶.۲}$$

$$\{f \leftarrow n\}(pk :: h) \triangleq \{pk[f := n] :: h\} \tag{۷.۲}$$

$$[\![p + q]\!]h \triangleq [\![p]\!]h \cup [\![q]\!]h \tag{۸.۲}$$

$$[\![p \cdot q]\!]h \triangleq ([\![p]\!] \bullet [\![q]\!])h \tag{۹.۲}$$

$$[\![p^*]\!]h \triangleq \bigcup_{i \in \mathbb{N}} F^i h \tag{۱۰.۲}$$

$$F^0 h \triangleq \{h\} \tag{۱۱.۲}$$

$$F^{i+1} h \triangleq ([\![p]\!] \bullet F^i)h \tag{۱۲.۲}$$

$$(f \bullet g)x \triangleq \bigcup \{g(y) | y \in f(x)\} \tag{۱۳.۲}$$

$$[\![dup]\!](pk :: h) \triangleq \{pk :: (pk :: h)\} \tag{۱۴.۲}$$

[14] Forward
[15] Drop
[16] Expression

در معادلات بالا فرض می‌شود که H مجموعه‌ی تمامی تاریخچه‌های ممکن است. معادله‌ی ۲.۲ بیان می‌کند که معنی هر عبارت نت‌کت روی یک تاریخچه‌ی بسته یک مجموعه از تاریخچه‌ی بسته‌های حاصل از اعمال این عبارت روی تاریخچه‌ی ورودی است. معادله‌ی ۳.۲ بیان می‌کند که عبارت ۱ بسته را بدون شرط عبور می‌دهد. در مقابل معادله‌ی ۴.۲ رها شدن بسته را با خروجی یک مجموعه‌ی خالی مدل می‌کند. معادله‌ی ۵.۲ بسته‌ی نخست ورودی را بررسی می‌کند و اگر مطابق با عبارت نبود بسته رها می‌شود. نقیض یک فیلتر در معادله‌ی ۶.۲ توصیف شده است. معادله‌ی ۷.۲ مقدار n را به فیلد f بسته‌ی نخست تاریخچه اختصاص می‌دهد. معادله‌ی ۸.۲ جمع دو جمله را به صورت اجتماع تاریخچه‌های حاصل از اعمال هر یک از عملوندها توصیف می‌کند. در معادله‌ی ۹.۲ ترکیب متوالی دو جمله را به صورت ترکیب Kleisli دو جمله که به شکل زیر تعریف می‌شود توصیف می‌کند:

$$(f \bullet g)x \triangleq \bigcup\{g\, y \mid y \in f\, x\}$$

در معادله‌ی ۱۰.۲ حاصل اپراتور ستاره‌ی کلینی [۱۷] معادل با اجتماع اعمال تابع‌های F_i روی تاریخچه‌ی ورودی در نظر گرفته شده است که تابع F_i حاصل i بار ترکیب Kleisli عبارت p است. در نهایت معادله‌ی ۱۴.۲ یک کپی از بسته‌ی نخست ورودی را به ابتدای خروجی اضافه می‌کند.

نت‌کت علاوه بر اصول موضوعه‌ی KAT [۱۸] اصول موضوعه‌ی زیر را هم شامل می‌شود تا دستگاه معادلاتی

[17] Kleene Star
[18] Axiom

صحیح و کامل [۱۹] داشته باشد:

$$f \leftarrow n \cdot f' \leftarrow n' \equiv f' \leftarrow n' \cdot f \leftarrow n, \text{ if } f \neq f' \quad (۱۵.۲)$$

$$f \leftarrow n \cdot f' = n' \equiv f' = n' \cdot f \leftarrow n, \text{ if } f \neq f' \quad (۱۶.۲)$$

$$dup \cdot f = n \equiv f = n \cdot dup \quad (۱۷.۲)$$

$$f \leftarrow n \cdot f = n \equiv f \leftarrow n \quad (۱۸.۲)$$

$$f = n \cdot f \leftarrow n \equiv f = n \quad (۱۹.۲)$$

$$f \leftarrow n \cdot f \leftarrow n' \equiv f \leftarrow n' \quad (۲۰.۲)$$

$$f = n \cdot f = n' \equiv 0, \text{ if } n \neq n' \quad (۲۱.۲)$$

$$\Sigma_i f = i \equiv 1 \quad (۲۲.۲)$$

اصل‌های ۱۵.۲، ۱۶.۲، ۱۷.۲ خواص جابه‌جایی [۲۰] عملیات‌ها را بیان می‌کنند. اصل ۱۸.۲ بیان می‌کند که اختصاص مقدار n به یک فیلد و سپس فیلتر کردن بر روی این فیلد با همین مقدار معادل با عملیات اختصاص [۲۱] به تنهایی است. مشابه همین اصل برای یک فیلتر و سپس یک اختصاص هم در اصل ۱۹.۲ مشخص شده. اصل ۲۰.۲ بیان می‌کند که در دنباله‌ای از اختصاص مقادیر به یک فیلد مشخص، تنها آخرین اختصاص تاثیر دارد. در اصل ۲۱.۲ مشخص شده است که مقدار یک فیلد نمی‌تواند دو مقدار متفاوت داشته باشد. در نهایت اصل ۲۲.۲ بیان می‌کند که عملیات مجموع فیلترها به ازای هر مقدار ممکن برای یک فیلد مشخص برابر عنصر همانی [۲۲] است.

۳.۳.۲ توصیف رفتار شبکه با نت‌کت

در ادامه فرض کنید که بخواهیم شبکه‌ای مانند شکل ۱.۲ را توصیف کنیم که در آن بسته‌هایی که از نوع SSH هستند اجازه‌ی عبور نداشته باشند. همچنان می‌توانیم از سیاست توصیف شده توسط ۱.۲ برای توصیف رفتار سوییچ‌ها استفاده کنیم. در ادامه می‌توان با اضافه کردن یک فیلتر به این عبارت، ویژگی دسترسی کنترل [۲۳] را به

[19] Sound and Complete
[20] Commutative
[21] Assignment
[22] Identity
[23] Access Control

این سیاست اضافه کرد تا همه‌ی بسته‌های از نوع SSH را رها کند:

$$p_{AC} \triangleq \neg(typ = SSH) \cdot p$$

اما استفاده از عبارت بالا به تنهایی برای توصیف رفتار شبکه شکل ۱.۲ کافی نیست. برای تکمیل این عبارت لازم است تا رفتار توپولوژی [۲۴] شبکه هم به آن افزوده شود. در نت‌کت توپولوژی شبکه به عنوان یک گراف جهت‌دار در نظر گرفته می‌شود و رفتار آن در قالب اجتماع رفتار هر یک از پیوندهای [۲۵] آن توصیف می‌شود. برای شبکه‌ی شکل ۱.۲ می‌توان از عبارت زیر برای توصیف توپولوژی شبکه استفاده کرد:

$$\begin{aligned} t \triangleq & (sw = A \cdot pt = 2 \cdot sw \leftarrow B \cdot pt \leftarrow 1) + \\ & (sw = b \cdot pt = 1 \cdot sw \leftarrow A \cdot pt \leftarrow 2) + \\ & (sw = b \cdot pt = 2) \end{aligned}$$

در نت‌کت در صورتی که سیاست و توپولوژی شبکه در قالب عبارت‌هایی توصیف شده‌باشند، رفتار کل شبکه در واقع دنباله‌ای از اعمال این عبارت‌ها به صورت یکی در میان است. به عنوان مثال در شکل ۱.۲ یک بسته از میزبان ۱ ابتدا توسط سوییچ A پردازش شده، سپس روی لینک بین دو سوییچ جا به جا می‌شود و در نهایت توسط سوییچ B پردازش می‌شود. در نت‌کت می‌توان این رفتار را به صورت $p_{AC} \cdot t \cdot p_{AC}$ توصیف کرد. با استفاده از همین شهود، رفتار کل شبکه را می‌توان در قالب عبارت زیر توصیف کرد:

$$(p_{AC} \cdot t)^*$$

در توصیف بالا فرض شده است که بسته‌ها می‌توانند به هر طریق ممکن وارد شبکه و از آن خارج شوند، اما این رفتار همیشه مورد قبول نیست. به عنوان مثال در شبکه شکل ۱.۲ اگر مکان‌های ورودی و خروجی شبکه را در

[24] Topology
[25] Link

قالب عبارت زیر توصیف کنیم:

$$e \triangleq sw = A \cdot port = 1 \vee sw = B \cdot port = 2$$

می‌توانیم رفتار انتها به انتهای [۲۶] شبکه را به شکل زیر توصیف کنیم:

$$p_{net} \triangleq e \cdot (p_{AC} \cdot t)^* e$$

در حالت کلی‌تر، نیازی به توصیف ورودی و خروجی‌های شبکه در قالب یک عبارت نیست. پس اگر فرض شود که مکان‌های ورودی شبکه توسط عبارت in و مکان‌های خروجی شبکه در قالب عبارت out توصیف شده باشند، رفتار یک شبکه در نت‌کت به صورت زیر تعریف می‌شود:

$$in \cdot (p \cdot t)^* \cdot out$$

که عبارت p سیاست شبکه و عبارت t توپولوژی شبکه است.

۴.۳.۲ درستی‌سنجی برنامه‌های نت‌کت

درستی‌سنجی یک شبکه و بررسی خواص آن در نت‌کت با استفاده از بررسی تساوی عبارت یک شبکه با عبارت‌های دیگر انجام می‌شود. به عنوان مثال در شبکه‌ی شکل ۱.۲ برای بررسی اینکه همه‌ی بسته‌ها با نوع SSH از میزبان ۱ رها می‌شوند کافی است تا تساوی زیر را ثابت کنیم:

$$\begin{pmatrix} type = SSH \cdot sw = A \cdot pt = 1 \cdot \\ (p_{AC} \cdot t)^* \cdot \\ sw = B \cdot pt = 2 \end{pmatrix} \equiv 0$$

[26] End to End

از طرفی برای بررسی یک خاصیت در شبکه، مثلا امکان فرستاده شدن همه‌ی بسته‌هایی که از نوع SSH نیستند از میزبان ۱ به میزبان ۲ می‌توان به جای بررسی تساوی دو عبارت از نامساوی $p \leq q$ استفاده کرد. این نامساوی که خلاصه شده‌ی تساوی $p+q \equiv q$ است بیان می‌کند که رفتار عبارت p بخشی از رفتار عبارت q است. بنابراین برای بررسی این مساله که شبکه‌ی شکل ۱.۲ بسته‌های غیر SSH از میزبان ۱ را عبور می‌دهد کافی است تا درستی نامعادله‌ی زیر را بررسی کرد:

$$\begin{pmatrix} \neg(type = SSH) \cdot sw = A \cdot pt = 1 \cdot \\ sw \leftarrow B \cdot pt \leftarrow 2 \end{pmatrix} \leq (p_{AC} \cdot t)^*$$

اصول موضوعه‌ی نت‌کت یک سیستم اثبات [۲۷] را تشکیل می‌دهند که امکان اثبات این معادله یا نامعادله‌ها را فراهم می‌کنند. مثلا فرض کنید که سیاست دسترسی‌کنترل برای رها کردن بسته‌های SSH ۱.۲ را برای افزایش کارایی فقط در سوییچ A انجام دهیم:

$$p_A \triangleq (sw = A \cdot \neg(typ = \texttt{SSH}) \cdot p) + (sw = B \cdot p)$$

به طریق مشابه می‌توانیم این کار را در سوییچ B هم انجام دهیم:

$$p_B(sw = A \cdot p) \triangleq (sw = B \cdot \neg(typ = \texttt{SSH}) \cdot p)$$

فرض کنید مکان‌های ورودی و خروجی به صورت زیر تعریف شده باشند:

$$in \triangleq (sw = A \cdot pt = 1)$$
$$out \triangleq (sw = B \cdot pt = 2)$$

اثبات معادل بودن دو سیاست p_A و p_B بر اساس این ورودی و خروجی در شکل ۲.۲ ذکر شده است.

[27] Proof System

$$
\begin{aligned}
& in \cdot \text{SSH} \cdot (p_A \cdot t)^* \cdot out \\
\equiv & \{ \text{KAT-Invariant, definition } p_A \} \\
& in \cdot \text{SSH} \cdot ((a_A \cdot \neg\text{SSH} \cdot p + a_B \cdot p) \cdot t \cdot \text{SSH})^* \cdot out \\
\equiv & \{ \text{KA-Seq-Dist-R} \} \\
& in \cdot \text{SSH} \cdot (a_A \cdot \neg\text{SSH} \cdot p \cdot t \cdot \text{SSH} + a_B \cdot p \cdot t \cdot \text{SSH})^* \cdot out \\
\equiv & \{ \text{KAT-Commute} \} \\
& in \cdot \text{SSH} \cdot (a_A \cdot \neg\text{SSH} \cdot \text{SSH} \cdot p \cdot t + a_B \cdot p \cdot t \cdot \text{SSH})^* \cdot out \\
\equiv & \{ \text{BA-Contra} \} \\
& in \cdot \text{SSH} \cdot (a_A \cdot 0 \cdot p \cdot t + a_B \cdot p \cdot t \cdot \text{SSH})^* \cdot out \\
\equiv & \{ \text{KA-Seq-Zero/Zero-Seq, KA-Plus-Comm, KA-Plus-Zero} \} \\
& in \cdot \text{SSH} \cdot (a_B \cdot p \cdot t \cdot \text{SSH})^* \cdot out \\
\equiv & \{ \text{KA-Unroll-L} \} \\
& in \cdot \text{SSH} \cdot (1 + (a_B \cdot p \cdot t \cdot \text{SSH}) \cdot (a_B \cdot p \cdot t \cdot \text{SSH})^*) \cdot out \\
\equiv & \{ \text{KA-Seq-Dist-L, KA-Seq-Dist-R, definition } out \} \\
& in \cdot \text{SSH} \cdot a_B \cdot a_2 + \\
& in \cdot \text{SSH} \cdot a_B \cdot p \cdot t \cdot \text{SSH} \cdot (a_B \cdot p \cdot t \cdot \text{SSH})^* \cdot a_B \cdot a_2 \\
\equiv & \{ \text{KAT-Commute} \} \\
& in \cdot a_B \cdot \text{SSH} \cdot a_2 + \\
& in \cdot a_B \cdot \text{SSH} \cdot p \cdot t \cdot \text{SSH} \cdot (a_B \cdot p \cdot t \cdot \text{SSH})^* \cdot a_B \cdot a_2 \\
\equiv & \{ \text{Lemma 1} \} \\
& 0 + 0 \\
\equiv & \{ \text{KA-Plus-Idem} \} \\
& 0 \\
\equiv & \{ \text{KA-Plus-Idem} \} \\
& 0 + 0 \\
\equiv & \{ \text{Lemma 1, Lemma 2} \} \\
& in \cdot a_B \cdot \text{SSH} \cdot a_2 + \\
& in \cdot \text{SSH} \cdot (a_A \cdot p \cdot t \cdot \text{SSH})^* \cdot p \cdot \text{SSH} \cdot a_A \cdot t \cdot out \\
\equiv & \{ \text{KAT-Commute, definition } out \} \\
& in \cdot \text{SSH} \cdot out + \\
& in \cdot \text{SSH} \cdot (a_A \cdot p \cdot t \cdot \text{SSH})^* \cdot a_A \cdot p \cdot t \cdot \text{SSH} \cdot out \\
\equiv & \{ \text{KA-Seq-Dist-L, KA-Seq-Dist-R} \} \\
& in \cdot \text{SSH} \cdot (1 + (a_A \cdot p \cdot t \cdot \text{SSH})^* \cdot (a_A \cdot p \cdot t \cdot \text{SSH})) \cdot out \\
\equiv & \{ \text{KA-Unroll-R} \} \\
& in \cdot \text{SSH} \cdot (a_A \cdot p \cdot t \cdot \text{SSH})^* \cdot out \\
\equiv & \{ \text{KA-Seq-Zero/Zero-Seq, KA-Plus-Zero} \} \\
& in \cdot \text{SSH} \cdot (a_A \cdot p \cdot t \cdot \text{SSH} + a_B \cdot 0 \cdot p \cdot t)^* \cdot out \\
\equiv & \{ \text{BA-Contra} \} \\
& in \cdot \text{SSH} \cdot (a_A \cdot p \cdot t \cdot \text{SSH} + a_B \cdot \neg\text{SSH} \cdot \text{SSH} \cdot p \cdot t)^* \cdot out \\
\equiv & \{ \text{KAT-Commute} \} \\
& in \cdot \text{SSH} \cdot (a_A \cdot p \cdot t \cdot \text{SSH} + a_B \cdot \neg\text{SSH} \cdot p \cdot t \cdot \text{SSH})^* \cdot out \\
\equiv & \{ \text{KA-Seq-Dist-R} \} \\
& in \cdot \text{SSH} \cdot ((a_A \cdot p + a_B \cdot \neg\text{SSH} \cdot p) \cdot t \cdot \text{SSH})^* \cdot out \\
\equiv & \{ \text{KAT-Invariant, definition } p_B \} \\
& in \cdot \text{SSH} \cdot (p_B \cdot t)^* \cdot out \qquad \square
\end{aligned}
$$

شکل ۲.۲: اثبات معادل بودن p_A و p_B [۲]

۴.۲ نت‌کت پویا

نک‌کت پویا [۲۸] برای رفع برخی از کاستی‌های نت‌کت ارائه شده است [۶]. به صورت دقیق‌تر نت‌کت پویا، امکان توصیف به‌روزرسانی سیاست‌های شبکه و همچنین رفتار شبکه در مقابل چندین بسته را ممکن می‌سازد.

۱.۴.۲ دستور زبان نت‌کت پویا

در نت‌کت پویا، از رفتار انتها به انتها [۲۹] ی توصیف‌های شبکه در قالب عبارت‌های نت‌کت استفاده می‌شود. به این معنا که در نت‌کت پویا تنها خروجی حاصل از اعمال عبارات نت‌کت روی بسته‌ها اهمیت دارد و مسیری که طی شده است در نظر گرفته نمی‌شود. به همین منظور دستور زبان نت‌کت پویا به صورت زیر تعریف می‌شود:

$$N ::= \mathrm{NetKAT}^{-dup}$$

$$D ::= \bot \mid N; D \mid x?N; D \mid x!N; D \mid D \parallel D \mid D \oplus D \mid X$$

$$X \triangleq D$$

در سینتکس بالا NetKAT^{-dup} قسمتی از زبان نت‌کت است که عبارت‌های dup از آن حذف شده است. عبارت‌های dup در توصیف‌های نت‌کت تاثیری در رفتار یک عبارت ندارند و هدف از استفاده از آن‌ها ثبت یک اثر از هر بسته پس از پردازش توسط یکی از عناصر شبکه است و امکان استدلال بر روی رفتار شبکه را ممکن می‌سازد. با توجه به این که در نت‌کت پویا رفتار انتها به انتهای یک عبارت نت‌کت مورد استفاده است، عبارت dup از دستور زبان کنار گذاشته شده است. نت‌کت‌پویا یک لیست از بسته‌های ورودی را پردازش می‌کند و یک لیست از مجموعه‌ی بسته‌های خروجی تولید می‌کند. اپراتور ترکیب متوالی [۳۰] $N; D$ باعث می‌شود که یک بسته از لیست بسته‌های ورودی توسط سیاست N پردازش شود و سپس این بسته توسط عبارت D پردازش شود. در نت‌کت پویا امکان ارتباط توسط عبارت‌هایی به شکل $x!N$ و $x?N$ توصیف می‌شوند که به ترتیب ارسال و دریافت یک عبارت نت‌کت را روی کانال x توصیف می‌کنند. ترکیب موازی [۳۱] دو عبارت توسط $D \parallel D$ توصیف

[28] DyNetKAT
[29] End to End
[30] Sequential Composition
[31] Parallel Composition

می‌شود. در نهایت رفتارهای غیرقطعی [۳۲] توسط عبارت‌هایی به شکل $D \oplus D$ توصیف می‌شوند.

۲.۴.۲ معنای عملیاتی نت‌کت پویا

معنای عملیاتی [۳۳] نت‌کت پویا با استفاده از عبارت‌هایی به شکل (d, H, H') تعریف می‌شوند که d عبارت نت‌کت پویا فعلی است، H لیست بسته‌هایی که در ادامه باید پردازش شوند و H' لیست بسته‌هایی است که به صورت موفقیت‌آمیز توسط شبکه پردازش شده‌اند. در اینجا فرض می شود که $F = \{f_1, ..., f_n\}$ یک مجموعه از فیلدهای بسته‌ها است. یک بسته به شکل یک تابع $F \to \mathbb{N}$ توصیف می‌شود. برای یک بسته مانند σ تساوی $\sigma(f_i) = v_i$ بیان می‌کند که مقدار فیلد f_i در بسته‌ی σ برابر با v_i است. یک لیست خالی از بسته‌ها با $\langle\rangle$ نمایش داده می‌شود. اگر l یک لیست از بسته‌ها باشد $e :: l$ لیستی است که حاصل از اضافه کردن بسته σ به ابتدای لیست به دست می‌آید. برچسب هر قانون که با γ مشخص می‌شود به صورت یکی از شکل‌های $(\sigma, \sigma'), x!q, x?q$ یا $rcfg(x, q)$ تعریف می‌شود که $rcfg(x, q)$ به معنی انجام شدن $x!q$ و $x?q$ به صورت همگام [۳۴] است. قوانین

[32] Non-Deterministic
[33] Operational Seamntic
[34] Synchronized

زیر معنای عملیاتی نت‌کت پویا را تعریف می‌کنند:

$$(cpol^{\checkmark}_{;})\frac{\sigma' \in [\![p]\!](\sigma :: \langle\rangle)}{(p;q,\sigma :: H, H') \xrightarrow{(\sigma,\sigma')} (q, H, \sigma' :: H')} \quad (۲۳.۲)$$

$$(cpol_X)\frac{(p, H_0, H_1) \xrightarrow{\gamma} (p', H_0', H_1')}{(X, H_0, H_1) \xrightarrow{\gamma} (p', H_0', H_1')} X \triangleq p \quad (۲۴.۲)$$

$$(cpol_{\_\oplus})\frac{(p, H_0, H_0') \xrightarrow{\gamma} (p', H_1, H_1')}{(p \oplus q, H_0, H_0') \xrightarrow{\gamma} (p', H_1, H_1')} \quad (۲۵.۲)$$

$$(cpol_{\oplus\_})\frac{(q, H_0, H_0') \xrightarrow{\gamma} (q', H_1, H_1')}{(p \oplus q, H_0, H_0') \xrightarrow{\gamma} (p', H_1, H_1')} \quad (۲۶.۲)$$

$$(cpol_{\_\|})\frac{(p, H_0, H_0') \xrightarrow{\gamma} (p', H_1, H_1')}{(p \parallel q, H_0, H_0') \xrightarrow{\gamma} (p' \parallel q, H_1, H_1')} \quad (۲۷.۲)$$

$$(cpol_{\|\_})\frac{(q, H_0, H_0') \xrightarrow{\gamma} (q', H_1, H_1')}{(p \parallel q, H_0, H_0') \xrightarrow{\gamma} (p \parallel q', H_1, H_1')} \quad (۲۸.۲)$$

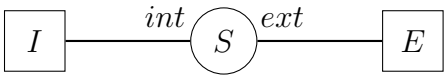
$$(cpol_{?})\frac{}{(x?p;q, H, H') \xrightarrow{x?p} (q, H, H')} \quad (۲۹.۲)$$

$$(cpol_{!})\frac{}{(x!p;q, H, H') \xrightarrow{x!p} (q, H, H')} \quad (۳۰.۲)$$

$$(cpol_{!?})\frac{(q, H, H') \xrightarrow{x!p} (q', H, H')(s, H, H') \xrightarrow{x?p} (s', H, H')}{(q \parallel s, H, H') \xrightarrow{rcfg(x,p)} (q' \parallel s', H, H')} \quad (۳۱.۲)$$

$$(cpol_{?!})\frac{(q, H, H') \xrightarrow{x?p} (q', H, H')(s, H, H') \xrightarrow{x!p} (s', H, H')}{(q \parallel s, H, H') \xrightarrow{rcfg(x,p)} (q' \parallel s', H, H')} \quad (۳۲.۲)$$

قانون ۲۳.۲ انجام یک عملیات مانند (σ, σ') که به معنای پردازش بسته‌ی ابتدایی لیست ورودی توسط عبارت p و افزودن خروجی حاصل از آن مانند σ' به لیست خروجی است را مشخص می‌کند. قانون ۲۴.۲ بیان می‌کند که رفتار متغیر X که برابر با عبارت p است معادل با رفتار عبارت p است. قوانین ۲۵.۲ و ۲۶.۲ رفتار غیرقطعی را توصیف می‌کنند که در آن یکی از عملوندها انتخاب شده و عملیات γ را انجام می دهد. قوانین ۲۷.۲ و ۲۸.۲ رفتار دو عبارت موازی را توصیف می‌کنند که در آن امکان اجرای عملیات توسط هر یک از عملوندها وجود دارد. قوانین ۲۹.۲ و ۳۰.۲ مشخص می‌کنند که ارسال یا دریافت پیام در نت‌کت پویا پردازشی روی بسته‌ها انجام نمی‌دهد.



شکل ۳.۲: مثال دیوار آتش

در نهایت همگام‌سازی [۳۵] ارسال و دریافت پیام در پردازه‌های موازی توسط قوانین ۲۹.۲ و ۳۰.۲ توصیف شده است که در آن دو یکی از پردازه‌ها امکان ارسال و دیگری امکان دریافت پیام را دارد و در نتیجه در پردازه حاصل از توازی آن‌ها امکان انجام یک عملیات هنگام از نوع $rcfg$ وجود دارد که معادل همگام‌سازی عملیات‌های ارسال و دریافت پیام است.

۳.۴.۲ توصیف برنامه‌ها در نت‌کت پویا

در ادامه چگونگی توصیف یک دیوار آتش [۳۶] حالت‌دار [۳۷] با استفاده از نت‌کت پویا بیان می‌شود. شبکه‌ی شکل ۳.۲ را در نظر بگیرید. در این شبکه هدف این است که امکان ارتباط از داخل شبکه به بیرون فراهم باشد ولی امکان ارسال بسته از خارج شبکه ممکن نباشد. اما زمانی که یک بسته به خارج شبکه ارسال شد، دیوار آتش باید اجازه‌ی عبور بسته‌ها از بیرون را بدهد تا پاسخ بسته‌ها دریافت شوند. برای توصیف این شبکه می‌توان از

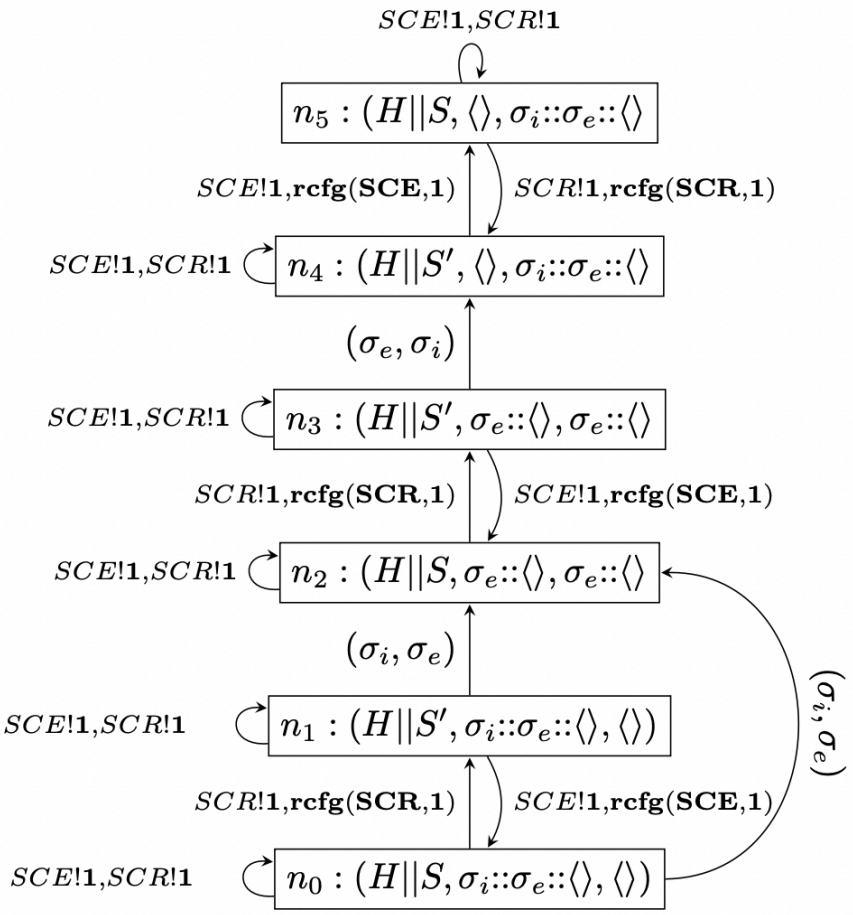
[35] Synchronization
[36] Firewall
[37] Stateful

عبارت نت‌کت پویای زیر استفاده کرد:

$$
\begin{aligned}
Host \triangleq & secConReq!1; Host \oplus secConEnd!1; Host \\
Switch \triangleq & (port = int) \cdot (port \leftarrow ext); Switch \oplus \\
& (port = ext) \cdot 0; Switch \oplus \\
& secConReq?1; Switch' \\
Switch' \triangleq & (port = int) \cdot (port \leftarrow ext); Switch' \oplus \\
& (port = ext) \cdot (port \leftarrow int); Switch' \oplus \\
& secConEnd?1; Switch \\
Init \triangleq & Host \parallel Switch
\end{aligned}
$$

در این توصیف عملیات $secConReq$ برای شروع ارتباط امن و $secConEnd$ برای خاتمه‌ی ارتباط امن در نظر گرفته شده‌است. بنابراین برنامه‌ی سوییچ پس از دریافت پیام برای شروع ارتباط امن تبدیل به برنامه‌ی $Switch'$ می‌شود که در آن اجازه‌ی ارسال بسته از پورت خارجی به پورت داخلی را می‌دهد. پس از دریافت پیام برای خاتمه‌ی ارتباط امن، برنامه به حالت اولیه‌ی خود بر می‌گردد و دوباره تمامی بسته‌های ورودی از پورت خارجی را رها می‌کند. در نهایت رفتار کل شبکه با استفاده از ترکیب موازی یک میزبان و یک سوییچ در حالت اولیه خود توصیف می‌شود. نمودار نمایش داده شده در شکل ۴.۲ سیستم انتقال برچسب‌دار [۳۸] این شبکه را در حالتی که یک بسته روی پورت ورودی و یک بسته روی پورت خروجی شبکه وجود دارد نشان می‌دهد. همانطور که در نمودار مشخص است، عملیات (σ_e, σ_i) که به معنای ارسال بسته از پورت ورودی به پورت خروجی است تنها در قسمتی از این سیستم انتقال قابل دسترسی است که پیش از آن یکی از عملیات‌های $SCR?۱$ یا $rcfg(SCR, ۱)$ انجام شده باشند. بنابراین در این حالت شبکه تنها در صورتی که بسته خارجی را به داخل ارسال می‌کند که پیش از آن پیام آغاز ارتباط امن دریافت کرده باشد.

[38] Labeled Transition System



شکل ۴.۲: سیستم انتقال برچسب‌دار برای شبکه‌ی دیوار آتش [۶]

۵.۲ ساختمان رویداد

ساختمان رویداد [۳۹] [۳۴] یک مدل محاسباتی [۴۰] غیرجای‌گذاری شده [۴۱] برای پردازه‌های هم‌روند [۴۲] است. در این مدل، برخلاف مدل‌های جایگذاری شده [۴۳] مانند سیستم‌انتقال که هم‌روندی پردازه‌های موازی با انتخاب غیرقطعی مدل می‌شود، هم‌روندی پردازه‌ها به صورت صریح در مدل توصیف می‌شوند [۳۱].

تعریف ۱.۵.۲. ساختمان رویداد یک ساختمان رویداد یک سه‌تایی $(E, \#, \vdash)$ است که در آن:

[39] Event Structure
[40] Computational Model
[41] Non-Interleaving
[42] Concurrent
[43] Interleaving

۱. E یک مجموعه از رویدادها است

۲. $\#$ رابطه‌ی تعارض [۴۴] ، یک رابطه‌ی دودویی متقارن و غیربازتابی بر روی مجموعه‌ی E است

۳. $\vdash \subseteq Con \times E$ رابطه‌ی فعال سازی [۴۵] است که شرط زیر را برقرار می‌کند:

$$X \vdash e \wedge X \subseteq Y \in Con \Rightarrow Y \vdash e$$

در رابطه‌ی بالا Con زیرمجموعه‌ای از مجموعه‌ی توانی رویدادها است که اعضای آن فاقد تعارض باشند. به صورت دقیق‌تر داریم:

$$Con = \{X \subseteq E | \forall e, e' \in X. \neg(e \# e')\}$$

تعریف ۲.۵.۲. به ازای هر ساختمان رویداد، می‌توانیم رابطه‌ی فعال‌سازی مینیمال را به صورت زیر تعریف کنیم:

$$X \vdash_{min} e \iff X \vdash e \wedge (\forall Y \subseteq X. Y \vdash e \Rightarrow Y = X)$$

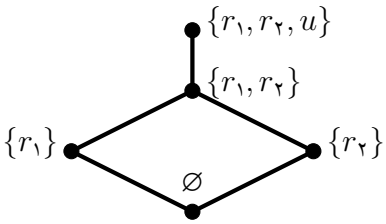
همچنین در هر ساختمان رویدادی شرط زیر برقرار است:

$$Y \vdash e \Rightarrow \exists X \subseteq Y. X \vdash_{min} e$$

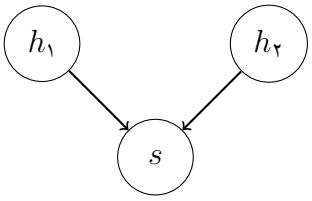
برای مشخص کردن وضعیت یک سیستم در هر لحظه از مفهومی به نام پیکربندی [۴۶] استفاده می‌شود و و یک مجموعه شامل رویدادهایی است که تا آن لحظه در سیستم رخ داده‌اند.

تعریف ۳.۵.۲. اگر $\mathrm{E} = (E, \#, \vdash)$ یک ساختمان رویداد باشد، یک پیکربندی آن یک زیرمجموعه از رویدادها $x \subseteq E$ است که شرایط زیر را داشته باشد:

[44] Conflict
[45] Enabling
[46] Configuration



شکل ۵.۲



شکل ۶.۲

۱. $x \in Con$

۲. $\forall e \in x \exists e_0, ..., e_n \in x. e_n = e \,\&\, \forall i \leq n. \{e_0, ..., e_{i-1}\} \vdash e_i$

مجموعه‌ی همه‌ی پیکربندی‌های یک ساختمان رویداد مانند E با $\mathcal{F}(\mathrm{E})$ نمایش داده می‌شود.

شبکه‌ی موجود در شکل ۶.۲ را در نظر بگیرید. در این شبکه دو میزبان ۱ و ۲ به صورت هم‌روند یک بسته را به سوییچ ارسال می‌کنند. این بسته‌ها شامل اطلاعات برای به روزرسانی مسیرهای دیگر در شبکه هستند، بنابراین سوییچ پس از دریافت هر دوی این بسته‌ها آن ها را پردازش کرده و مسیرهای خود را به روزرسانی می‌کند. برای مدل کردن این شبکه می‌توانیم از یک ساختمان رویداد به صورت زیر استفاده کنیم:

$$\mathrm{E} = (\{r_1, r_2, u\}, \varnothing, \{(\varnothing, r_1), (\varnothing, r_2), (\{r_1, r_2\}, u)\})$$

در این ساختمان رویداد، رویدادها به ترتیب دریافت یک بسته از میزبان ۱، دریافت یک بسته از میزبان ۲ و به روز رسانی سوییچ را مدل می‌کنند. یکی از روش‌های رسم نمودار برای ساختمان رویداد، رسم نمودار هس [۴۷] برای مجموعه‌ی پیکربندی‌های این ساختمان رویداد بر اساس رابطه‌ی زیرمجموعه است. برای مثالی که بیان شد می‌توان نموداری مطابق شکل ۵.۲ را رسم کرد.

[47] Hasse

۶.۲ مدل علّی

پیدا کردن تعریفی برای علت واقعی [۴۸] مبحثی است که مورد مطالعه و تحقیق بسیاری قرار گرفته است. این مساله به طور خاص در متون فلسفه مورد توجه قرار گرفته است. یکی از تعاریف علت واقعی که مورد توجه بسیاری قراری گرفته است، تعریفی مبتنی بر وابستگی خلاف واقع [۴۹] است. مطابق این تعریف، رویداد الف علت رویداد ب است اگر در شرایطی که رویداد الف اتفاق نیافته باشد، رویداد ب هم اتفاق نیافتند. در اینجا اتفاق نیفتادن رویداد الف خلاف واقع است، چون در سناریوی واقعی (سناریو ای که واقعا اتفاق افتاده و مشاهده شده است) رویداد الف اتفاق افتاده است و در نظر گرفتن شرایطی که در آن رویداد الف اتفاق نیفتاده باشد بر خلاف واقعیت موجود است. اما این مدل به تنهایی امکان پیدا کردن علت مناسب را در همه‌ی موارد ندارد. به عنوان مثال سناریوی زیر را در نظر بگیرید که در آن سارا و بهرام هر کدام یک سنگ را برداشته و به سمت یک بطری شیشه‌ای پرتاب می‌کنند. در این سناریو، سنگ سارا زودتر از سنگ بهرام به بطری برخورد کرده و در نتیجه آن را می‌شکنند. در این سناریو واضح است که پرتاب سنگ توسط سارا علت شکسته شدن بطری است. فرض کنید بخواهیم از علیت مبتنی بر خلاف واقع برای پیدا کردن این علت استفاده کنیم. بنابراین باید شرایطی را در نظر بگیریم که سارا سنگ خود را پرتاب نکند. اما مشکل اینجاست که در این شرایط همچنان بطری شکسته می‌شود، چون اگر سارا سنگ خود را پرتاب نکند، بهرام همچنان سنگ خود را پرتاب می‌کند و در نتیجه این بار سنگ بهرام به بطری برخورد کرده و آن را می‌شکند. بنابراین در این سناریو امکان تعریف پرتاب سنگ توسط سارا به عنوان علت شکسته شدن بطری با استفاده از استدلال مبتنی بر خلاف واقع وجود ندارد. هالپرن [۵۰] و پرل [۵۱] برای حل کردن مشکلاتی از این دست، تعریف جدیدی از علت واقعی [۱۷] ارائه کردند. مدل ارائه شده توسط آن‌ها به دلیل اینکه مبنای ریاضی دارد امکان استفاده از آن را در آنالیز و تحلیل سیستم‌های محاسباتی فراهم می‌کند. به همین دلیل این تعریف در مقالات زیادی در حوزه‌ی دانش کامپیوتر مورد استفاده قرار گرفته است. برای تعریف علت واقعی ابتدا برخی مفاهیم اولیه مورد استفاده در این تعریف توضیح داده می‌شوند. به صورت کلی فرض می‌شود که دنیای مورد تحلیل توسط تعدادی متغیر تصادفی مدل شده است. اگر X یک متغیر تصادفی باشد، یک رویداد به شکل $X = x$ تعریف می‌شود. برخی از این متغیرها بر روی یکدیگر تاثیر گذارند. این وابستگی‌ها در قالب

[48] Actual Cause
[49] Counterfactual
[50] Halpern
[51] Pearl

مجموعه‌ای از معادلات ساختاری [۵۲] مدل می‌شوند. هر یک از این معادلات در واقع یک مکانیزم یا قانون مشخص در این دنیا را مدل می‌کنند. متغیرها به دو دسته درونی [۵۳] و برونی [۵۴] تقسیم می‌شوند. متغیرهای برونی متغیرهایی در نظر گرفته می‌شوند که مقدار آن‌ها توسط عواملی که درون مدل نیستند تعیین می‌شوند. بنابراین در مدل فرض می‌شود که مقدار این متغیرها از قبل مشخص است. اما متغیرهای درونی متغیرهایی هستند که مقدار آن‌ها بر اساس معادلات ساختاری تعیین می‌شود. به صورت دقیق‌تر، امضای [۵۵] یک مدل یک سه‌تایی $\mathcal{S} = (\mathcal{U}, \mathcal{V}, \mathcal{R})$ است که در آن $\mathcal{U}$ مجموعه‌ی متغیرهای بیرونی $\mathcal{V}$ مجموعه‌ی متغیرهای درونی و $\mathcal{R}$ دامنه‌ی مقادیر ممکن برای هر یک از متغیرها را مشخص می‌کند. در این مدل فرض می‌شود که مجموعه‌ی متغیرهای درونی محدود است. مدل علّی بر روی یک امضای $\mathcal{S}$ یک دوتایی $\mathcal{M} = (\mathcal{S}, \mathcal{F})$ است که در آن $\mathcal{F}$ به هر متغیر داخلی $X \in \mathcal{V}$ یک تابع $F_X : (\times_{U \in \mathcal{U}} \mathcal{R}(U)) \times (\times_{Y \in \mathcal{V}-\{X\}} \mathcal{R}(Y)) \to \mathcal{R}(X)$ اختصاص می‌دهد. هر تابع، معادله‌ی یک متغیر را به ازای مقادیر تمام متغیرهای دیگر مشخص می‌کند. به عنوان مثال اگر فرض کنیم $F_X(Y, Z, U) = Y + U$ اگر داشته باشیم $Y = ۳, U = ۲$ آنگاه مقدار X برابر ۵ خواهد شد. این معادلات امکان تفسیر آن‌ها بر اساس شرایط خلاف واقع را می‌دهند. به عنوان مثال در همین مدل اگر فرض کنیم که $U = u$ می‌توانیم نتیجه بگیریم که اگر مقدار متغیر Y برابر ۴ باشد آنگاه مستقل از اینکه مقدار بقیه‌ی متغیرها در دنیای واقعی چه مقداری دارند، مقدار متغیر X برابر $u + ۴$ خواهد بود که به صورت $(M, u) \models [Y \leftarrow ۴](X = u + ۴)$ نوشته می‌شود. توابع ذکر شده فقط برای متغیرهای درونی تعریف می‌شوند و همانطور که پیش‌تر اشاره شد، برای متغیرهای بیرونی تابعی تعریف نمی‌شود و فرض می‌شود که مقدار آن‌ها از قبل مشخص شده است.

مثال ۱.۶.۲. یک جنگل را در نظر بگیرید که می‌تواند توسط رعد و برق یا یک کبریت رها شده دچار آتش سوزی شود. برای مدل کردن این سناریو از سه متغیر بولی [۵۶] استفاده می‌کنیم:

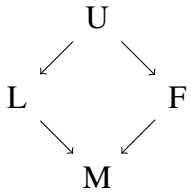
- متغیر F که اگر جنگل دچار آتش سوزی شود مقدار آن درست است و در غیر این صورت مقدار آن غلط است
- متغیر L که اگر رعد و برق اتفاق افتاده باشد مقدار آن درست است و در غیر این صورت غلط است

[52] Structural Equations
[53] Endogenous
[54] Exogenous
[55] Signature
[56] Boolean

- متغیر M که اگر یک کبریت در جنگل رها شده باشد مقدار آن درست است و در غیر این صورت غلط است

در این مثال فرض می کنیم که مقادیر متغیرهای برونی به گونه‌ای است که تمام شرایط لازم برای آتش سوزی جنگل در صورتی که رعد و برق اتفاق بیافتد یا کبریتی در جنگل رها شود را دارد (به عنوان مثال درختان جنگل به اندازه‌ی کافی خشک هستند و اکسیژن کافی در هوا وجود دارد). در این مدل تابع متغیر F را به گونه‌ای تعریف می‌کنیم که داشته باشیم: $F_F(\vec{u}, L, M) = L \vee M$. همانطور که پیش‌تر بیان شد، این مدل علّی امکان بررسی معادلات بر اساس شرایط خلاف واقع را می‌دهد. به صورت دقیق‌تر اگر $M = (\mathcal{S}, \mathcal{F})$ یک مدل علی، $\vec{X}$ یک بردار از متغیرهای درونی و $\vec{x}, \vec{u}$ برداری از مقادیر متغیرهای $\vec{X}, \mathcal{U}$ باشند مدل $M_{\vec{X}\leftarrow\vec{x}}$ را با امضای $S_{\vec{X}} = (\mathcal{U}, \mathcal{V} - \vec{X}, \mathcal{R}|_{-\vec{X}})$ یک زیرمدل [۵۷] از M تعریف می‌کنیم. به صورت شهودی این مدل حاصل مداخله [۵۸] ای در مدل M است که در آن مقادیر $\vec{x}$ را به متغیرهای $\vec{X}$ اختصاص داده‌ایم. به صورت دقیق‌تر تعریف می‌کنیم $M_{\vec{X}\leftarrow\vec{x}} = (\mathcal{S}_{\vec{X}}, \mathcal{F}^{\vec{X}\leftarrow\vec{x}})$ که $F_Y^{\vec{X}\leftarrow\vec{x}}$ از تابع F_Y که در آن مقادیر $\vec{x}$ را به متغیرهای $\vec{X}$ اختصاص داده‌ایم به دست می‌آید. به عنوان مثال اگر M مدل مثال ۱.۶.۲ باشد آنگاه در مدل $M_{L\leftarrow \mathrm{F}}$ معادله‌ی متغیر F به $F = M$ تبدیل می‌شود. این معادله دیگر به متغیر L وابسته نیست بلکه با توجه به مقدار آن که در اینجا غلط است معادله‌ی جدیدی دارد. علاوه براین توجه کنید که در مدل $M_{L\leftarrow \mathrm{F}}$ دیگر معادله‌ای برای متغیر L وجود ندارد. توجه کنید که در حالت کلی ممکن است یک بردار یکتا از مقادیر متغیرها برای یک مدل وجود نداشته باشد که همزمان تمامی معادلات را حل کند. در مدل علّی یک بردار از مقادیر متغیرهای برونی $\vec{u}$ یک هم‌بافت [۵۹] نامیده می‌شود. در مدل‌های بازگشتی به ازای یک هم‌بافت مشخص همیشه یک راه‌حل یکتا برای تمامی معادلات مدل وجود دارد. در ادامه فرض می‌شود که مدل‌ها بازگشتی هستند. تعمیم مدل علّی برای مدل‌های غیربازگشتی در [۱۷] توضیح داده است. برای یک مدل می‌توان یک شبکه‌ی علّی ترسیم کرد. این شبکه یک گراف جهت‌دار است که به ازای هر متغیر یک گره در آن وجود دارد و یک یال بین دو گره وجود دارد اگر تابع متغیر دوم به متغیر اول وابسته باشد. به عنوان مثال شکل زیر شبکه‌ی علّی مثال ۱.۶.۲ را نشان می‌دهد:

[57] Sub-Model
[58] Intervention
[59] Context



در ادامه برای سادگی رسم شبکه‌ی علی، متغیرهای برونی را از آن‌ها حذف می‌کنیم.

۱.۶.۲ علت واقعی

در ادامه فرمول‌های لازم برای تعریف علت واقعی توصیف می‌شوند. اگر $\mathcal{S} = (\mathcal{U}, \mathcal{V}, \mathcal{R})$ یک امضا باشد فرمول $X = x$ یک رویداد بدوی [۶۰] نامیده می‌شود که $X \in \mathcal{V}, x \in \mathcal{R}(X)$. فرمول $[Y_1 \leftarrow y_1, ..., Y_k \leftarrow y_k]\varphi$ یک فرمول علّی پایه [۶۱] نامیده می‌شود که در آن:

- φ یک ترکیب بولی از رویدادهای بدوی است
- $Y_1, ..., Y_k$ متغیرهای متمایز در $\mathcal{V}$ هستند
- $y_i \in \mathcal{R}(Y_i)$

این فرمول به صورت خلاصه به شکل $[\vec{Y} \leftarrow \vec{y}]\varphi$ نوشته می‌شود و اگر $k = 0$ باشد آنگاه به صورت φ نوشته می‌شود. به صورت شهودی یک فرمول به شکل $[\vec{Y} \leftarrow \vec{y}]\varphi$ بیان می‌کند که در شرایط خلاف واقع ای که در آن مقادیر $\vec{y}$ به متغیرهای $\vec{Y}$ اختصاص داده شده است فرمول φ برقرار است. یک فرمول علّی به صورت یک ترکیب بولی از فرمول‌های علّی پایه تعریف می‌شود. برقراری فرمول علی ψ در مدل M تحت هم‌بافت $\vec{u}$ را به صورت $(M, \vec{u}) \models \psi$ نشان می‌دهیم. به عنوان مثال $(M, \vec{u}) \models [\vec{Y} \leftarrow \vec{y}](X = x)$ برقرار است اگر مقدار متغیر X در راه حل معادلات مدل $M_{\vec{Y} \leftarrow \vec{y}}$ تحت هم‌بافت $\vec{u}$ برابر x باشد.

تعریف ۲.۶.۲. فرمول $\vec{X} = \vec{x}$ علت واقعی φ (که تاثیر [۶۲] نامیده می‌شود) در $(M, \vec{u})$ است اگر شرایط زیر برای آن برقرار باشد:

[60] Prime Event
[61] Basic Causal Formula
[62] Effect

۱. $(M, \vec{u}) \models (\vec{X} = \vec{x}) \wedge \varphi$

۲. یک افراز مانند $(\vec{Z}, \vec{W})$ از مجموعه‌ی متغیرهای $\mathcal{V}$ با شرط $\vec{X} \subseteq \vec{Z}$ و مقادیر $(\vec{x}, \vec{w}')$ برای متغیرهای $(\vec{X}, \vec{W})$ وجود داشته باشد که داشته باشیم $(M, \vec{u}) \models \vec{Z} = \vec{z}^*$ و شرایط زیر را برآورده کند:

(آ) $(M, \vec{u}) \models [\vec{X} \leftarrow \vec{x}', \vec{W} \leftarrow \vec{w}']\neg\varphi$

(ب) $\forall \vec{W}' \subseteq \vec{W}, \vec{Z}' \in \vec{Z}. (M, \vec{u}) \models [\vec{X} \leftarrow \vec{x}, \vec{W}' \leftarrow \vec{w}', \vec{Z}' \leftarrow \vec{z}^*]\varphi$

۳. $\vec{X}$ مینیمال باشد.

در این تعریف شرط اول بیان می‌کند که علت و تاثیر هر دو در شرایط واقعی برقرار هستند. شرط دوم به دنبال پیدا کردن شرایطی است که تحت آن تاثیر به صورت غیر واقع به علت وابسته باشد. این شرایط متغیرهای $\vec{W}$ و مقادیری مانند $\vec{w}'$ برای آن‌ها هستند. شرط ۲.الف بررسی می‌کند که تحت شرایطی که توسط $\vec{W} \leftarrow \vec{w}'$ به وجود می‌آید اگر علت مقداری متفاوت از مقدار خود در هم‌بافت واقعی داشته باشد اثر در مدل دیده نمی‌شود. شرط ۲.ب بررسی می‌کند که شرایط استفاده شده در ۲.الف عامل از بین رفتن اثر در ۲.الف نباشند. برای این منظور در شرایطی که علت مقدار واقعی خود را دارد در تمامی حالت‌هایی که متغیرهای شرایط می‌توانند داشته باشند بررسی می‌شود که اثر همچنان برقرار باشد. شرط سوم در واقع بیان می‌کند که زیرمجموعه‌ای از علت وجود نداشته باشد که همزمان شرایط ۱ و ۲ را برقرار کند. در تعریف بالا $(\vec{W}, \vec{w}', \vec{x}')$ یک شاهد [۶۳] بر اینکه $\vec{X} = \vec{x}$ علت φ است تعریف می‌شود.

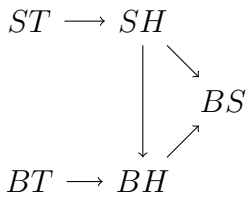
۲.۶.۲ پیدا کردن علت واقعی در مسائل

در ادامه مثال سارا و بهرام که در ابتدای این بخش ذکر شده بود را بررسی می‌کنیم.

برای مدل کردن این مساله متغیرهای زیر را در نظر می‌گیریم:

- BT: پرتاب سنگ توسط بهرام
- BH برخورد سنگ بهرام به بطری

[63] Witness



شکل ۷.۲

- ST: پرتاب سنگ توسط سارا
- SH: برخورد سنگ سارا به بطری
- BS: شکسته شدن بطری

ابتدا فرض می‌کنیم که متغیرهای BT, ST تنها به متغیرهای برونی وابسته‌اند. بطری در صورتی شکسته می‌شود که هر یک از سنگ‌های سارا یا بهرام با آن برخورد کنند. بنابراین برای شکسته شدن بطری معادله‌ی $BS = BH \vee SH$ را در نظر می‌گیریم. نکته‌ی اصلی در این مساله این است که سنگ سارا زودتر از سنگ بهرام به شیشه برخورد می‌کند، به همین دلیل لازم است تا این موضوع در مدل لحاظ شود. یک راه برای مدل کردن این مساله این است که معادله‌ی برخورد سنگ بهرام به شیشه را به گونه‌ای تعریف کنیم که تنها در صورتی که سنگ سارا به بطری برخورد نکرده باشد آنگاه سنگ بهرام به بطری برخورد کند. بنابراین می‌توانیم معادله‌ی $BH = BT \wedge \neg SH$ را تعریف کنیم. علاوه بر این معادله‌ی برخورد سنگ سارا را بدون وابستگی به برخورد سنگ بهرام تعریف می‌کنیم: $SH = ST$. با توجه به این تعاریف برای معادلات می‌توانیم گراف علّی شکل ۷.۲ را برای این مدل رسم کنیم در این مدل می‌توانیم $ST = \mathrm{T}$ را به عنوان علت $BS = \mathrm{T}$ تعریف کنیم. برای برقراری شرط ۲ در تعریف علت واقعی شرایط $\vec{W} = \{BT\}$ و $w' = \mathrm{F}$ را در نظر می‌گیریم. در این شرایط چون مقدار BH برابر F می‌شود، مقدار BS تنها وابسته به مقدار SH و در نتیجه ST می‌شود. همچنین در این مدل $BT = \mathrm{T}$ علت شکسته شدن شیشه نیست. مثلا فرض کنید که شرایط $\vec{W} = \{ST\}, w' = \mathrm{F}$ را در نظر بگیریم. در این شرایط اگر مقدار BT را به F تغیر دهیم مقدار BS هم غلط می‌شود. بنابراین شرط ۲.الف برقرار است. اما به ازای $\vec{Z}' = \{BH\}$ شرط ۲.ب برقرار نمی‌شود. در این حالت داریم: $(M, \vec{u}) \models [BT \leftarrow \mathrm{T}, ST \leftarrow \mathrm{F}, BH \leftarrow F]BS = \mathrm{F}$ توجه کنید با وجود اینکه مقدار درست به BT اختصاص یافته اما چون مقدار BH به مقدار آن در هم‌بافت واقعی برگردانده می‌شود در نتیجه مقدار BS همچنان غلط می‌ماند.

مثال بالا نشان می‌دهد که این تعریف از علت واقعی برخی از مشکلات موجود در تعاریف ساده مبتنی بر

خلاف واقع را برطرف می‌کند و می‌تواند توضیح مناسبی در برخی از این مثال‌ها پیدا کند. نکته‌ای که باید به آن توجه شود این است که هنوز روش یا معیاری برای این که چه تعریفی از علت واقعی تعریف مناسب است وجود ندارد. تنها روش ممکن مقایسه تعاریف مختلف استفاده از آن‌ها در مساله‌ها و سناریوهای مختلف و بررسی تطابق علت به دست آمده با استفاده از این تعریف‌ها با شهود موجود از مساله است.

۳.۶.۲ مدل تعمیم‌یافته

مدل علّی تعمیم یافته [۶۴] یک سه‌تایی $(\mathcal{S}, \mathcal{F}, \mathcal{E})$ است که $(\mathcal{S}, \mathcal{F})$ یک مدل علّی است و $\mathcal{E}$ یک مجموعه از مقداردهی‌های مجاز [۶۵] برای متغیرهای درونی است. به صورت دقیق‌تر اگر متغیرهای درونی $X_1, ..., X_n$ باشند آن‌گاه $(x_1, ..., x_n) \in \mathcal{E}$ اگر $X_1 = x_1, ..., X_n = x_n$ یک مقداردهی مجاز است. یک مقداردهی دلخواه به یک زیرمجموعه از متغیرهای درونی مجاز است اگر امکان تعمیم به یک مقداردهی مجاز در $\mathcal{E}$ را داشته باشد. هدف از این تعریف جلوگیری از در نظر گرفتن علت‌هایی است که شرایط رخ دادن آن‌ها غیر محتمل است. با توجه به تعریف مقداردهی مجاز، علت واقعی در یک مدل تعمیم یافته به گونه‌ای تعریف می‌شود که در شرط ۲ فقط امکان مقداردهی‌های مجاز وجود داشته باشد. در [۱۷] تعریف دقیق علت واقعی در مدل تعمیم یافته بیان نشده است. در بخش بعدی تعریفی از علت واقعی در مدل تعمیم یافته ارائه می‌شود.

۴.۶.۲ علت واقعی بدون شرط

فرض کنید که $\vec{X} = \vec{x}$ یک علت واقعی برای φ در $(M, \vec{u})$ با استفاده از شاهد $(\varnothing, \varnothing, \vec{x}')$ باشد. با توجه به اینکه در اینجا $\vec{W}$ یک بردار خالی است پس عملا شرط ۲.ب به بررسی شرط زیر تبدیل می‌شود:

$$\forall \vec{Z}' \in \vec{Z}.(M, \vec{u}) \vDash [\vec{X} \leftarrow \vec{x}, \vec{Z}' \leftarrow \vec{z}^*]\varphi$$

با دقت در شرط بالا می‌توان دریافت که مقدار متغیرها در فرمول‌های $[\vec{X} \leftarrow \vec{x}, \vec{Z}' \leftarrow \vec{z}^*]\varphi$ با مقدار متغیرها در هم‌بافت اولیه تفاوتی ندارد زیرا مقدار آن‌ها به مقداری که در هم‌بافت اولیه داشته‌اند برگردانده می‌شود. بنابراین

[64] Extended Causal Model
[65] Allowable Settings

در شرط بالا می‌توان نتیجه گرفت:

$$(M, \vec{u}) \vDash [\vec{X} \leftarrow \vec{x}, \vec{Z}' \leftarrow \vec{z}^*]\varphi \iff (M, \vec{u}) \vDash \varphi$$

بنابراین شرط ۲.ب معادل با شرط ۱ می‌شود. با توجه به این موضوع می‌توان قضیه زیر را نتیجه گرفت:

گزاره ۳.۶.۲. اگر $\vec{X} = \vec{x}$ در $(M, \vec{u})$ با شاهدی به شکل $(\varnothing, \varnothing, \vec{x}')$ شرط‌های ۱، ۲.الف و ۳ در تعریف ۲.۶.۲ را برای φ برآورده کند آنگاه $\vec{X} = \vec{x}$ یک علت واقعی برای φ در $(M, \vec{u})$ است.

فصل ۳

مروری بر کارهای پیشین

توضیح خطا [1] و متمرکز کردن خطا [2] روش‌هایی هستند که فرآیند رفع ایراد نرم‌افزار را تسهیل می‌کنند. توضیح خطا روش‌هایی را شامل می‌شود که به کاربر کمک می‌کند که با استفاده از یک مثال‌نقض یا یک دنباله از اجرای سیستم به ماهیت خطا و در نتیجه روش اصلاح خطا پی ببرد. در روش‌های متمرکز کردن خطا هدف مشخص کردن بخشی از سیستم است که عامل خطا بوده و امکان اندازه‌گیری کمی و مقایسه آن با دیگر بخش‌ها وجود دارد [۱۵]. یکی از روش‌های توضیح خطا پیدا کردن علت خطا بر اساس استدلال مبتنی بر خلاف واقع که توسط لوئیس در [۲۵] ارائه شده می‌باشد که در پژوهش‌هایی مانند [۳۵، ۱۵، ۱۶] استفاده شده است. هالپرن و پرل در [۱۷] تعریفی مبتنی بر استدلال خلاف واقع لوئیس ارائه کردند که توانسته است برخی از مشکلات تعریف لوئیس در پیدا کردن علت در سناریوهای پیچیده را بر طرف کند. در ادامه برخی از پژوهش‌هایی که از تعریف هالپرن و پرل برای توضیح خطا استفاده کرده‌اند را مورد بررسی قرار می‌دهیم.

۱.۳ تخمین پوشش

در [۹] نویسندگان از مدل HP برای تخمین میزان پوشش [3] سیستم توسط یک توصیف در فرآیند وارسی مدل استفاده کرده‌اند. معیارهای پوشش معمولا در فرآیند تست سیستم استفاده می‌شوند و مشخص کننده درصدی از

[1] Fault Explanation
[2] Fault Localization
[3] Covering

اجزا یا حالت‌های سیستم هستند که توسط مجموعه‌ی تست‌ها مورد استفاده یا بازدید قرار می‌گیرند. در فرآیند وارسی مدل همه‌ی حالت‌های سیستم بررسی می‌شوند به همین دلیل در این شرایط اگر تغییر یک حالت منجر به نقض ویژگی توصیف شده شود این حالت پوشش داده شده توسط ویژگی تعریف می‌شود. با استفاده از مفهوم مسئولیت [۴] که در [۱۸] تعریف شده است، نویسندگان این پژوهش به جای در نظر گرفتن مقدار ۰ و ۱ برای پوشیده شدن یا نشدن از درجه‌ی مسئولیت استفاده می‌کنند. همانطور که مشخص است این پژوهش به اصلاح و بهبود توصیف ویژگی کمک می‌کند و نه پیدا کردن علت خطا. در این پژوهش همانند پژوهش جاری به شکل مستقیم از تعریف HP استفاده شده است.

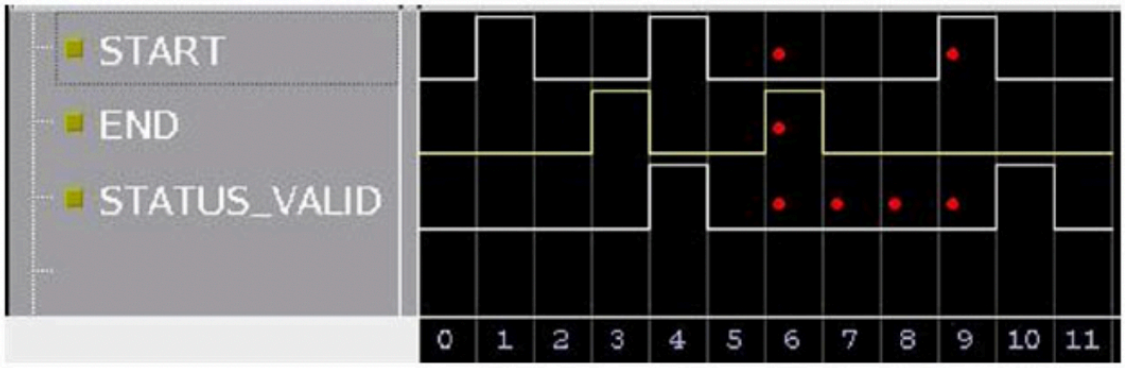
۲.۳ علت خطا در مثال نقض

در [۴] نویسندگان سیستم را به صورت یک سیستم انتقال [۵] در نظر می‌گیرند که در آن هر حالت یک نگاشت از یک مجموعه‌ی متغیرهای بولی به مقادیر درست و غلط است. در این پژوهش با استفاده از تعریف علت واقعی در یک مثال نقض یک ویژگی توصیف شده در LTL [۶] یک دوتایی متغیر و حالت به عنوان علت واقعی در نظر گرفته می‌شود. در همین پژوهش یک الگوریتم تقریبی برای پیدا کردن همه‌ی علت‌ها در یک مثال نقض داده شده ارائه شده است و ابزاری برای نمایش این علت‌ها به صورت گرافیکی به کاربر توسعه داده شده و در ابزار درستی‌سنجی RuleBase PE متعلق به IBM گنجانده شده است. تصویر ۱.۳ رابط کاربری ابزار RuleBase PE را پس از پیدا کردن یک مثال نقض برای ویژگی زیر نشان می‌دهد:

$$\mathbf{G}((\neg \texttt{START} \wedge \neg \texttt{STATUS\_VALID} \wedge \texttt{END}) \rightarrow [\neg \texttt{START}\ \mathbf{U}\ \texttt{STATUS\_VALID}])$$

در این تصویر نقاط قرمز علت‌های واقعی هستند که با الگوریتم تقریبی پیاده‌سازی شده پیدا شده‌اند. این پژوهش یکی از کاربردی‌ترین استفاده‌ها از توضیح خطا و پیدا کردن علت خطا را نشان می‌دهد. در این پژوهش سعی شده است تا علت خطا در یک مثال نقض پیدا شود و به همین دلیل مقدار متغیرها در حالت‌ها به عنوان علت پیدا می‌شوند در حالی که در پژوهش جاری هدف پیدا کردن علت خطا در کل سیستم است و در واقع ساختارهای

[4] Responsiblity
[5] Transition System
[6] Linear Temporal Logic



شکل ۱.۳: رابط کاربری ابزار PE RuleBase

سیستم، مثلا وجود یا عدم وجود روابط تعارض یا فعال‌سازی به عنوان علت خطا پیدا می‌شوند. اما همانند پژوهش جاری در این پژوهش هم به شکل مستقیم و بدون تغییر از تعریف HP استفاده شده است.

۳.۳ چک کردن علیت

در پژوهش [۲۴] نویسندگان تعریفی از علت واقعی که الهام گرفته از تعریف HP است ارائه می‌کنند و الگوریتم آن‌ها بر اساس این تعریف در حین اجرای فرآیند وارسی مدل [۷] علت‌ها را پیدا کرده و در نتیجه در انتهای وارسی مدل اگر سیستم ویژگی مورد نظر را نقض کرد به جای برگرداندن یک مثال نقض، رویدادهایی که علت رخداد خطا بوده‌اند را بر می‌گرداند. در این پژوهش یک منطق برای توصیف یک دنباله از رویداد عملیات‌های سیستم ارائه شده است و فرمول‌های این منطق به عنوان علت خطا در نظر گرفته می‌شوند. این پژوهش هم همانند [۴] سعی بر پیدا کردن همه‌ی علت‌های بروز خطا دارد و علت‌ها عملا دنباله‌هایی از اجرای سیستم هستند. تفاوت اصلی این کار با پژوهش جاری در این است که در این پژوهش علت خطا در رفتارهای سیستم جستجو می‌شود در حالی که در پژوهش جاری علت خطا در میان عناصر ساختاری سیستم جستجو می‌شود. این روش تنها برای ویژگی‌های دسترس‌پذیری ارائه شده است. در [۵] نویسندگان این روش را برای ویژگی‌های دلخواه توصیف شده توسط LTL تعمیم دادند.

[7] Model Checking

۴.۳ علت واقعی در خودکاره‌های زمان‌دار

در [۲۱] نویسندگان از تعریف HP برای پیدا کردن علت خطا در خودکاره‌های زمان‌دار [۸] استفاده کرده‌اند. در درستی‌سنجی خودکاره‌های زمان‌دار یک ابزار وارسی مدل بلا درنگ نقض ویژگی را در قالب یک رد تشخیصی زمان‌دار [۹] که در واقع یک مثال‌نقض است بر می‌گرداند. یک TDT در واقع یک دنباله متناوب از انتقال تاخیر [۱۰] و انتقال عملیات [۱۱] ها است که در آن مقدار تاخیرها به صورت سمبلیک مشخص شده‌اند. هدف این پژوهش پیدا کردن مقادیری یا دامنه‌ای از مقادیر برای این تاخیرهای سمبلیک است که بروز خطا را اجتناب ناپذیر می‌کنند یا به عبارت دیگر علت واقعی هستند. در این پژوهش اما به صورت مستقیم از تعریف HP استفاده نشده است و بر اساس آن تعریفی برای علت واقعی نقض ویژگی در یک TDT بیان شده است.

۵.۳ چارچوب علیت بر اساس رد سیستم

در [۱۴] نویسندگان این مساله را مطرح می‌کنند که تعریف ارائه شده توسط هالپرن و پرل ذاتا یک مدل بر اساس منطق گزاره‌ای [۱۲] است و به همین دلیل برای درستی‌سنجی پردازه‌ها ایده‌آل نیست. در این پژوهش یک فرمالیسم و تعریف جدید برای علیت بر اساس تعریف HP ارائه می‌شود که در آن از رد [۱۳] های سیستم به جای متغیرها در مدل HP استفاده می‌شود و امکان ترکیب [۱۴] چند مدل با یکدیگر را فراهم می‌کند.

[8] Timed Automata
[9] Timed Diagnostic Trace
[10] Transition Delay
[11] Delay Transition
[12] Propositional Logic
[13] Trace
[14] Composition

۶.۳ استدلال مبتنی بر علیت در HML

در [۷] نویسندگان از مفهوم استدلال مبتنی در سیستم‌انتقال برچسب‌دار [۱۵] و HML [۱۶] [۱۹] استفاده کرده‌اند. در این پژوهش سیستم با استفاده از یک سیستم انتقال برچسب‌دار مدل می‌شود و رفتار ناامن توسط یک فرمول در قالب HML توصیف می‌شود. سپس یک تعریف جدید که برگرفته شده از تعریف HP است با استفاده از این مدل‌ها برای علت واقعی بیان می‌شود. در این تعریف از مفهومی به نام عدم‌وقوع [۱۷] رویدادها که پیش‌تر در [۲۴] مطرح شده بود استفاده می‌شود. شهود کلی مفهوم عدم‌وقوع در علیت این است که در کنار اینکه رخ‌دادن برخی از رویدادها منجر به خطا می‌شود، رخ ندادن رویدادها هم می‌تواند به عنوان علت در نظر گرفته شود. در تعریف ارائه شده در این پژوهش مجموعه‌ای از محاسبه [۱۸] های سیستم به عنوان علت برقراری یک فرمول HML در سیستم که رفتار نا امن [۱۹] را توصیف می‌کند تعریف می‌شود. هر محاسبه شامل یک دنباله از عملیات‌های سیستم در کنار تعدادی عملیات دیگر، که عدم وقوع آن‌ها هم جزئی از علت است، در نظر گرفته می‌شود. به عبارت دیگر یک محاسبه را می‌توان شامل دو جز در نظر گرفت. جز اول یک اجرای سیستم است که منجر به خطا می‌شود. جز دوم مجموعه‌ای از اجراهای سیستم است که منجر به خطا نمی‌شوند و حاصل جایگذاری [۲۰] برخی از عملیات‌ها در جز اول این محاسبه هستند. عملیات‌های جایگذاری شده عملیات‌هایی هستند که عدم وقوع آن‌ها به عنوان علت بروز خطا در نظر گرفته می‌شود. در این تعریف علت واقعی به گونه‌ای تعریف شده است که محاسباتی که منجر به فعال شدن فرمول HML در سیستم می‌شوند به عنوان علت در نظر گرفته می‌شوند. در این تعریف شروطی مشابه با شروط موجود در تعریف HP در نظر گرفته شده است. در [۸] نویسندگان تعریف خود را بهبود دادند تا تطابق بیشتری با تعریف HP داشته باشد. علاوه بر این در این پژوهش ثابت شده است که این تعریف از علت در سیستم‌هایی که ارتباط همگام [۲۱] شده دارند قابل ترکیب نیست ولی در حالتی که سیستم‌ها ارتباط همگام نداشته باشند امکان ترکیب یا شکستن آن وجود دارد. نتایج حاصل از این پژوهش یکی از انگیزه‌های اصلی پژوهش جاری بود برای اینکه با انتخاب یک مدل معنایی یا تعریف علیت متفاوت امکان ترکیب آن برای سیستم‌های همگام شده بررسی شود. در ادامه به بررسی شباهت‌ها و تفاوت‌های این پژوهش و پژوهش جاری

[15] Labeled Transition System
[16] Hennesy Milner Loigc
[17] Non-Occurrence
[18] Computation
[19] Unsafe Behavior
[20] Interleaving
[21] Synchronized

می‌پردازیم اولا در این پژوهش تعریف جدیدی از علت واقعی ارائه شده است در حالی که در پژوهش جاری مستقیما از تعریف ارائه شده در [۱۷] استفاده شده است. در پژوهش جاری تمرکز بر پیدا کردن یک علت برای بروز خطا در سیستم است در حالی که در این پژوهش همه‌ی علل خطا مورد بررسی قرار می‌گیرند. پژوهش جاری علل خطا را در ساختارهای سیستم جستجو می‌کند در حالی که این پژوهش در میان رفتارهای سیستم به دنبال علل خطا می گردد.

۷.۳ جمع‌بندی

همان طور که بررسی شد پژوهش‌های متعددی در زمینه‌ی توضیح خطا ارائه شده است که نشان از اهمیت این مساله در فرآیند درستی‌سنجی و اشکال‌زدایی دارد. همچنین تعریف HP هم مورد توجه زیادی برای پیدا کردن علت خطا قرار گرفته است. یکی از مهم‌ترین تمایزهای پژوهش جاری با پژوهش‌های پیشین در المان‌هایی است که در آن علت خطا پیدا می‌شود. همانطور که بررسی شد در تمامی پژوهش‌های پیشین در این زمینه علت خطا در میان رفتارهای سیستم جستجو می‌شود. اما در پژوهش جاری رویکردی متفاوت استفاده شده است و علت خطا در میان ساختارهای سیستم، مثلا هم‌روند بودن یا نبودن پردازه، انجام می‌شود. مساله‌ی دیگری که باید به آن اشاره شود این است که در پژوهش جاری همانند [۴، ۹] به شکل مستقیم و بدون تغییر از تعریف HP استفاده می‌شود.

فصل ۴

روش و راه‌حل پیشنهادی

۱.۴ مقدمه

در این فصل روش پیدا کردن علت خطا در یک برنامه‌ی توصیف شده در نت‌کت پویا توضیح داده می‌شود. در بخش اول مدل معنایی عبارات نت‌کت پویا با در قالب ساختمان رویداد تعریف می‌شود. در بخش دوم یک مدل علّی برای توصیف ساختمان رویداد مطرح می‌شود. در نهایت در بخش سوم شامل استفاده از این چگونگی ترکیب این دو روش برای توضیح خطا در یک برنامه نت‌کت پویا توضیح داده می‌شود..

۲.۴ مدل معنایی عبارات نت‌کت پویا در قالب ساختمان رویداد

در این بخش ابتدا انواع ترکیب و محدودسازی ساختمان‌های رویداد تعریف می‌شود. سپس با استفاده از این تعاریف یک مدل معنایی برای عبارات نت‌کت پویا ارائه می‌شود.

تعریف ۱.۲.۴. فرض کنید $\mathrm{E} = (E, \#, \vdash)$ یک ساختمان رویداد باشد. به ازای یک مجموعه‌ی $A \subseteq E$ ،

محدودیت [1] E به A یک ساختمان رویداد به شکل زیر است:

$$\mathrm{E}\lceil A = (A, \#_A, \vdash_A)$$

که داشته باشیم:

$$X \subseteq Con_A \iff X \subseteq A \wedge X \in Con$$
$$X \vdash_A e \iff X \subseteq A \wedge e \in A \wedge X \vdash e$$

تعریف ۲.۲.۴. فرض کنید $\mathrm{E} = (E, \#, \vdash)$ یک ساختمان رویداد و a یک رویداد باشد. ساختمان رویداد $a\mathrm{E} = (E', \#', \vdash')$ به گونه‌ای تعریف می‌شود که داشته باشیم:

$$E' = \{(0, a)\} \cup \{(1, e) | e \in E\},$$
$$e'_0 \#' e'_1 \iff \exists e_0, e_1 . e'_0 = (1, e_0) \wedge e'_1 = (1, e_1) \wedge e_0 \# e_1$$
$$X \vdash' e' \iff e' = (0, a) \text{ or } [e' = (1, e_1) \wedge (0, a) \in X \wedge \{e | (1, e) \in X\} \vdash e_1]$$

تعریف ۳.۲.۴. یک ساختمان رویداد برچسب‌دار [2] یک پنج‌تایی به شکل $(E, \#, \vdash, L, l)$ است که در آن $(E, \#, \vdash)$ یک ساختمان رویداد، L یک مجموعه از برچسب‌ها (فاقد عنصر *) و l یک تابع به فرم $l : E \to L$ است که به هر رویداد یک برچسب اختصاص می‌دهد. یک ساختمان رویداد برچسب‌دار را به اختصار به صورت (E, L, l) نشان می‌دهیم.

تعریف ۴.۲.۴. در یک ساختمان رویداد رابطه‌ی $\curlyvee$ را به صورت زیر تعریف می‌کنیم:

$$e \curlyvee e' \iff e \# e' \vee e = e'$$

تعریف ۵.۲.۴. فرض کنید (E, L, l) یک ساختمان رویداد برچسب‌دار و α یک برچسب باشد. $\alpha(\mathrm{E}, L, l)$

[1] Restriction
[2] Labeled Event Structure

را به صورت یک ساختمان رویداد برچسب‌دار به شکل $(\alpha\mathrm{E}, L', l)$ تعریف می‌کنیم که در آن $L' = \{\alpha\} \cup L$ و به ازای هر $e' \in E'$ داشته باشیم:

$$l'(e') = \begin{cases} \alpha & \text{if } e = (0, \alpha) \\ l(e) & \text{if } e = (1, e) \end{cases}$$

تعریف ۶.۲.۴. فرض کنید $\mathrm{E}_0 = (E_0, \#_0, \vdash_0, L_0, l_0)$ و $\mathrm{E}_1 = (E_1, \#_1, \vdash_1, L_1, l_1)$ دو ساختمان رویداد برچسب‌دار باشند. مجموع این دو ساختمان رویداد $\mathrm{E}_0 + \mathrm{E}_1$ را به صورت یک ساختمان رویداد برچسب‌دار $(E, \#, \vdash, L, l)$ تعریف می‌کنیم که در آن داشته باشیم:

$$E = \{(0, e) | e \in E_0\} \cup \{(1, e) | e \in E_1\}$$

با استفاده از این مجموعه از رویدادها توابع $\iota_k : E_k \to E$ را به ازای $k = 0, 1$ به شکل زیر تعریف می‌کنیم:

$$\iota_k(e) = (k, e)$$

رابطه‌ی تعارض را به گونه‌ای تعریف می‌کنیم که داشته باشیم:

$$\begin{aligned} e \# e' \iff & \exists e_0, e_0'. e = \iota_0(e_0) \wedge e' = \iota_0(e_0') \wedge e_0 \#_0 e_0' \\ & \bigvee \exists e_1, e_1'. e = \iota_1(e_1) \wedge e' = \iota_1(e_1') \wedge e_1 \#_1 e_1' \\ & \bigvee \exists e_0, e_1. (e = \iota_1(e_0) \wedge e' = \iota_1(e_1)) \vee (e' = \iota_1(e_0) \wedge e = \iota_1(e_1)) \end{aligned}$$

رابطه‌ی فعال‌سازی را به گونه‌ای تعریف می‌کنیم که داشته باشیم:

$$X \vdash e \iff X \in Con \wedge e \in E \wedge$$
$$(\exists X_0 \in Con_0, e_0 \in E_0 . X = \iota_0 X_0 \wedge e = \iota_0(e_0) \wedge X_0 \vdash_0 e_0) \text{ or}$$
$$(\exists X_1 \in Con_1, e_1 \in E_1 . X = \iota_1 X_1 \wedge e = \iota_1(e_1) \wedge X_1 \vdash_1 e_1)$$

مجموعه‌ی برچسب‌ها را به صورت $L = L_0 \cup L_1$ و تابع برچسب‌گذاری را به شکل تعریف می‌کنیم:

$$l(e) = \begin{cases} l_0(e_0) & \text{if } e = \iota_0(e_0) \\ l_1(e_1) & \text{if } e = \iota_1(e_1) \end{cases}$$

تعریف ۷.۲.۴. فرض کنید که $\mathrm{E}_0 = (E_0, \#_0, \vdash_0, L_0, l_0)$ و $\mathrm{E}_1 = (E_1, \#_1, \vdash_1, L_1, l_1)$ دو ساختار رویداد برچسب‌گذاری شده باشند. حاصلضرب آن‌ها $\mathrm{E}_0 \times \mathrm{E}_1$ را به صورت یک ساختمان رویداد برچسب‌گذاری شده $\mathrm{E} = (E, \#, \vdash, L, l)$ تعریف می‌کنیم که در آن رویدادها به صورت زیر تعریف می‌شوند:

$$E_0 \times_* E_1 = \{(e_0, *) | e_0 \in E_0\} \cup \{(*, e_1) | e_1 \in E_1\} \cup \{(e_0, e_1) | e_0 \in E_0 \wedge e_1 \in E_1\}$$

با توجه به این مجموعه رویدادها توابعی به شکل $\pi_i : E \to_* E_i$ تعریف می کنیم که به ازای $i = 0, 1$ داشته باشیم: $\pi_i(e_0, e_1) = e_i$. در اینجا رابطه‌ی تعارض را به کمک رابطه‌ی $\Cup$ که پیش‌تر تعریف شد، به شکل زیر به ازای تمامی رویدادهای $e, e' \in E$ توصیف می‌کنیم:

$$e \Cup e' \iff \pi_0(e) \Cup_0 \pi_0(e') \vee \pi_1(e) \Cup_1 \pi_1(e')$$

رابطه‌ی فعال‌سازی را به صورت زیر تعریف می‌کنیم:

$$X \vdash e \iff X \in Con \wedge e \in \mathcal{E} \wedge$$
$$(\pi_0(e) \text{ defined } \Rightarrow \pi_0 X \vdash_0 \pi_0(e)) \wedge (\pi_1(e) \text{ defined } \Rightarrow \pi_1 X \vdash_1 \pi_1(e))$$

مجموعه‌ی برچسب‌های حاصلضرب را به صورت زیر تعریف می‌کنیم:

$$L_0 \times_* L_1 = \{(\alpha_0, *) | \alpha_0 \in L_0\} \cup \{(*, \alpha_1) | \alpha_1 \in L_1\} \cup \{(\alpha_0, \alpha_1) | \alpha_0 \in L_0 \wedge \alpha_1 \in L_1\}$$

در انتها تابع برچسب‌گذاری را به صورت زیر تعریف می‌کنیم:

$$l(e) = (l_0(\pi_0(e)), l_1(\pi_1(e))))$$

تعریف ۸.۲.۴. فرض کنید که $\mathrm{E} = (E, \#, \vdash, L, l)$ یک ساختمان رویداد برچسب‌دار باشد. فرض کنید Λ یک زیرمجموعه از L باشد. محدودیت E به Λ را به صورت $\mathrm{E}\lceil\Lambda$ و به شکل یک ساختمان رویداد برچسب‌گذاری شده به شکل $(E', \#', \vdash', L \cap \Lambda, l')$ که در آن $(E', \#', \vdash') = (E, \#, \vdash)\lceil\{e \in E | l(e) \in \Lambda\}$ است و تابع برچسب‌گذاری معادل محدودسازی تابع l به دامنه‌ی $L \cap \Lambda$ است.

۱.۲.۴ معنای عبارات نت‌کت پویای نرمال

در ادامه ابتدا فرم نرمال عبارات نت‌کت پویا را تعریف می‌کنیم. فرض کنید که فیلدهای ممکن برای بسته‌ها $f_1, f_2, ..., f_k$ باشند. یک فیلتر کامل [۳] به صورت $\alpha = f_1 = n_1 ... f_k = n_k$ و یک اختصاص کامل [۴] به صورت $\pi = f_1 \leftarrow n_1 ... f_k \leftarrow n_k$ تعریف می‌شود. می‌گوییم یک عبارت q در $NetKAT^{-dup}$ به فرم نرمال است اگر به شکل $\Sigma_{\alpha \cdot \pi \in A} \alpha \cdot \pi$ باشد که داشته باشیم $\mathcal{A} = \{\alpha_i \cdot \pi_i | i \in I\}$. در عبارت قبل I مدل زبانی $NetKAT^{-dup}$ می‌باشد. بر اساس لم ۵ در [۶] به ازای هر عبارت p در $NetKAT^{-dup}$ یک عبارت p' به فرم نرمال وجود دارد که داشته باشیم: $p \equiv p'$.

[3] Complete Test
[4] Complete Assignment

تعریف ۹.۲.۴. زبان نت‌کت پویا نرمال را با دستور زبان زیر تعریف می‌کنیم:

$$F ::= \alpha \cdot \pi$$

$$D ::= \bot | F; D | x?F; D | x!F; D | D \parallel D | D \oplus D$$

با استفاده از این لم، در لم ۹ که در [۶] ثابت شده است، به ازای هر عبارت p در نت‌کت پویا یک عبارت معادل آن به فرم نرمال وجود دارد که داشته باشیم: $p \equiv q$. بنابراین در نهایت می‌توانیم هر عبارت نت‌کت پویا را به فرم یک عبارت نرمال با توجه به تعریف ۹.۲.۴ بنویسیم. در ادامه فرض کنیم که $\mathcal{A}$ مجموعه‌ی الفبا شامل تمامی حروف به شکل $\alpha \cdot \pi, x?F, x!F$ باشد و داشته باشیم $\alpha \in \mathcal{A}, L \subseteq \mathcal{A}$. معنای عبارات نت‌کت پویای نرمال را با به صورت زیر تعریف می‌کنیم:

$$[\![\bot]\!] = (\varnothing, \varnothing)$$

$$[\![\alpha; t]\!] = \alpha([\![t]\!])$$

$$[\![t_1 \oplus t_2]\!] = [\![t_1]\!] + [\![t_2]\!]$$

$$[\![t_1 \parallel t_2]\!] = [\![t_1]\!] \times [\![t_2]\!]$$

$$[\![\delta_L(t)]\!] = [\![t]\!] \lceil \mathcal{A} \setminus L$$

سمت چپ معادلات بالا عبارات نت‌کت پویای نرمال و در سمت راست ساختمان رویداد معادل هر یک مشخص شده است. در معادلات بالا $(\varnothing, \varnothing)$ یک ساختمان رویداد که مجموعه‌ی رویدادها و مجموعه‌ی برچسب‌های آن تهی است را نشان می‌دهد.

۳.۴ مدل علّی برای ساختمان رویداد

در ادامه نحوه‌ی توصیف یک ساختمان رویداد در قالب یک مدل علّی مطابق تعریف HP را بیان می کنیم.

فرض کنیم که $\mathrm{E} = (E, \#, \vdash)$ یک ساختمان رویداد باشد. مدل علّی این ساختمان رویداد را به صورت

$\mathcal{M} = (\mathcal{S}, \mathcal{F}, \mathcal{E})$ تعریف می‌کنیم که در آن $\mathcal{S} = (\mathcal{U}, \mathcal{V}, \mathcal{R})$. در این مدل فرض می‌کنیم همه‌ی متغیرها از نوع بولی هستند. همچنین در این مدل متغیر برونی در نظر نمی‌گیریم بنابراین داریم $\mathcal{U} = \varnothing$. اگر فرض کنیم مجموعه رویدادها به صورت $E = \{e_1, e_2, ..., e_n\}$ باشد مجموعه‌ی متغیرهای درونی را به صورت زیر تعریف می‌کنیم:

$$\begin{aligned}\mathcal{V} =& \{C_{e_i,e_j} | 1 \le i < j \le n . e_i \in E \wedge e_j \in E\} \\ & \cup \{EN_{s,e} | s \in \mathcal{P}(E), e \in E . e \notin s\} \\ & \cup \{M_{s,e} | s \in \mathcal{P}(E), e \in E . e \notin s\} \cup \{PV\}\end{aligned}$$

به صورت شهودی به ازای هر عضو از رابطه‌های $\#, \vdash, \vdash_{min}$ یک متغیر درونی در نظر می‌گیریم که درست بودن این متغیر به معنای وجود عنصر متناظر با آن در رابطه است.

به ازای $x, y \in \mathcal{P}(E)$ پوشیده شدن [۵] x توسط y را که با $x \prec y$ نمایش می‌دهیم به صورت زیر تعریف می‌کنیم:

$$x \subseteq y \wedge x \neq y \wedge (\forall z . x \subseteq z \subseteq y \Rightarrow x = z \text{ or } y = z)$$

همچنین به ازای هر متغیر $X \in \mathcal{V}$ بردار $\vec{V}_X$ را بردار شامل همه‌ی متغیرهای درونی به غیر از X تعریف می‌کنیم. با استفاده از این تعاریف توابع متغیرهای درونی را به صورت زیر تعریف می‌کنیم:

$$F_{C_{e,e'}}(\vec{V}_{C_{e,e'}}) = \begin{cases} true & \text{if } e \# e' \\ false & \text{otherwise} \end{cases}$$

$$F_{M_{s,e}}(\vec{V}_{M_{s,e}}) = \begin{cases} Min(s,e) \wedge Con(s) & \text{if } s \vdash_{min} e \\ false & \text{otherwise} \end{cases}$$

[5] Covering

$$F_{EN_{s,e}}(\vec{V}_{EN_{s,e}}) = \left(M_{s,e} \vee \left(\bigvee_{s' \prec s} EN_{s',e}\right)\right) \bigwedge Con(s)$$

که در آن‌ها داریم:

$$Con(s) = \left(\bigwedge_{1 \leq j < j' \leq n \wedge e_j, e_{j'} \in s} \neg C_{e_j, e_{j'}}\right)$$
$$Min(s,e) = \left(\bigwedge_{s' \subseteq E.(s' \subset s \vee s \subset s') \wedge e \notin s'} \neg M_{s',e}\right)$$

فرض کنید که $\mathbb{E}$ مجموعه‌ی تمامی سه‌تایی‌ها به فرم $(E, \#', \vdash')$ باشد که داشته باشیم:

$$\#' \subseteq E \times E$$
$$\vdash' \subseteq \mathcal{P} \times E$$

یک تابع به فرم $ES : \times_{V \in \mathcal{V} \setminus \{PV\}} \mathcal{R}(V) \to \mathbb{E}$ تعریف می‌کنیم که به صورت شهودی ساختمان رویداد حاصل از مقدار فعلی متغیرها در مدل علّی را به دست می‌دهد. فرض کنیم $\vec{v}$ برداری شامل مقادیر متغیرهای $\mathcal{V} \setminus \{PV\}$ باشد. به ازای هر متغیر مانند $V \in \mathcal{V}$ مقدار آن در $\vec{v}$ را با $\vec{v}(V)$ نمایش می‌دهیم. تابع ES را به گونه‌ای تعریف می‌کنیم که اگر $ES(\vec{v}) = (E, \#', \vdash')$ آنگاه داشته باشیم:

$$\forall e, e' \in E. e \#' e' \wedge e' \#' e \iff \vec{v}(C_{e,e'}) = \mathrm{T}$$
$$\forall s \in \mathcal{P}(E), e \in E. s \vdash' e \iff \vec{v}(EN_{s,e}) = \mathrm{T}$$

در ادامه فرض می‌کنیم که رفتار نا امن [۶] سیستمی که در قالب ساختمان رویداد مدل شده است، در قالب تابع متغیر PV توصیف شده است و در صورتی که رفتار ناامن در سیستم وجود داشته باشد مقدار آن درست و در غیر این صورت غلط است. در نهایت مقداردهی‌های مجاز $\mathcal{E}$ را مجموعه‌ی مقداردهی‌هایی مانند $\vec{v}$ در نظر می‌گیریم که خروجی تابع ES به ازای آن‌ها یک ساختمان رویداد باشد.

برای پیدا کردن علت واقعی در این مدل از تعریف زیر که در واقع سازگار شده‌ی تعریف برای مدل علّی تعمیم

[6] Unsafe Behavior

یافته است استفاده می‌کنیم:

تعریف ۱.۳.۴. اگر $M = (\mathcal{S}, \mathcal{F}, \mathcal{E})$ یک مدل علّی تعمیم یافته و $\vec{V}$ برداری از متغیرهای $\mathcal{V} \setminus PV$ باشد، فرمول $\vec{X} = \vec{x}$ علت واقعی φ در $(M, \vec{u})$ است اگر شرایط زیر برای آن برقرار باشد:

۱. $(M, \vec{u}) \vDash (\vec{X} = \vec{x}) \wedge \varphi$

۲. یک افراز مانند $(\vec{Z}, \vec{W})$ از مجموعه‌ی متغیرهای $\mathcal{V}$ با شرط $\vec{X} \subseteq \vec{Z}$ و مقادیر $(\vec{x}, \vec{w}')$ برای متغیرهای $(\vec{X}, \vec{W})$ وجود داشته باشد که داشته باشیم $(M, \vec{u}) \vDash \vec{Z} = \vec{z}^*$ و شرایط زیر را برآورده کند:

(آ) $(M, \vec{u}) \vDash [\vec{X} \leftarrow \vec{x}', \vec{W} \leftarrow \vec{w}']\neg\varphi \wedge \vec{V} = \vec{v} \wedge \vec{v} \in \mathcal{E}$

(ب) $\forall \vec{W}' \subseteq \vec{W}, \vec{Z}' \in \vec{Z}. (M, \vec{u}) \vDash [\vec{X} \leftarrow \vec{x}, \vec{W}' \leftarrow \vec{w}', \vec{Z}' \leftarrow \vec{z}^*]\vec{V} = \vec{v} \wedge \vec{v} \in \mathcal{E} \Rightarrow \varphi$

۳. $\vec{X}$ مینیمال باشد.

در این تعریف برای اینکه تنها مقداردهی‌های مجاز برای پیدا کردن علت واقعی در نظر گرفته شوند شرط ۲ تغییر یافته است. بند اضافه شده به شرط ۲.الف بیان می‌کند که تغییرات ایجاد شده در مدل یک مقدار دهی مجاز باشند. بند اضافه شده در ۲.ب باعث می‌شود این شرط تنها در حالت‌هایی بررسی شود که مقداردهی داده شده مجاز باشد.

۴.۴ پیدا کردن علت خطا در نت‌کت پویا

با استفاده از تعاریف بخش‌های قبلی در این بخش به چگونگی پیدا کردن علت خطا در یک برنامه توصیف شده در نت‌کت پویا می‌پردازیم.

فرض می‌کنیم که یک عبارت نت‌کت پویا p در اختیار داریم. ابتدا عبارت p را به فرم نرمال مطابق تعریف ۹.۲.۴ در می‌آوریم. فرض کنیم عبارت q فرم نرمال عبارت p باشد. اکنون فرض کنیم $\mathrm{E} = [\![q]\!]$ ساختمان رویداد معادل q باشد. اکنون مدل علّی $\mathcal{M}$ را بر اساس E می‌سازیم و رفتار نا امن را در قالب تابع متغیر PV این مدل و به شکل یک شرط بر روی مجموعه‌ی پیکربندی‌های مدل علّی توصیف می‌کنیم. در نهایت کافی است برای پیدا کردن علت واقعی رفتار نا امن، علت واقعی $PV = \mathrm{T}$ در $\mathcal{M}$ را بر اساس تعریف ۱.۳.۴ پیدا کنیم. توجه کنید

که در اینجا محدودیتی برای چگونگی تعریف رفتار نا امن وجود ندارد و یان تعریف می‌تواند هر شرطی بر روی مجموعه‌ی پیکربندی‌های مدل علّی باشد.

مثلا اگر مجموعه‌ای از مثال‌های نقض سیستم در قالب پیکربندی‌های ساختمان رویداد وجود داشته باشد (مثلا مجموعه‌ی C)، می‌توان رفتار نا امن را وجود یکی از این پیکربندی‌ها در سیستم توصیف کرد:

$$F_{PV}(\vec{V}_{PV}) = \bigvee_{\forall c \in C} c \in \mathcal{F}(ES(\vec{v}))$$

در مثالی دیگر اگر رفتار نا امن در قالب یک شرط unsafe روی برچسب‌های سیستم توصیف شده باشد می‌توان رفتار نا امن را وجود یک پیکربندی که شامل یک رویداد که شرط unsafe را برآورده می‌کند توصیف کرد:

$$F_{PV}(\vec{V}_{PV}) = \exists c \in \mathcal{F}(ES(\vec{v})).\exists e \in c.\text{unsafe}(l(e))$$

فصل ۵

نتایج

۱.۵ مقدمه

در این فصل با استفاده از مدل علّی تعریف شده در فصل پیشین، علت نقض چند دسته از ویژگی‌های رایج در شبکه را مورد بررسی قرار می‌دهیم.

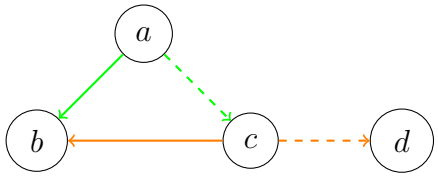
در ادامه فرض می‌کنیم که فیلد sw در همه‌ی توصیف‌های نت‌کت پویا وجود دارد. همچنین برای ساده‌تر شدن توصیف‌ها از اصل زیر استفاده می‌کنیم:

$$x \rightarrow y \triangleq sw = x \cdot sw \leftarrow y$$

۲.۵ لیست سیاه

در این ویژگی، یک لیست سیاه [1] از مکان‌هایی در شبکه وجود دارد که نباید در شبکه به آن‌ها دسترسی وجود داشته باشد [۲۹]. مهم‌ترین استفاده از لیست سیاه را می‌توان برای اعمال سیاست‌های کنترل دسترسی در نظر گرفت که مثلا برخی از میزبان‌ها که دارای اطلاعات حیاتی هستند در لیست سیاه قرار می‌گیرند تا از بیرون به آن‌ها دسترسی وجود نداشته باشد. به عنوان مثال دیگر ممکن است برخی از عناصر شبکه برای تعمیر برای مدتی

[1] Blacklist



شکل ۱.۵

کنار گذاشته شوند برای این منظور می‌توان آن‌ها را در لیست سیاه قرار داد تا دسترسی به آن‌ها سبب از دست رفتن بسته‌ها نشود.

برای پیدا کردن علت نقض شدن ویژگی لیست سیاه شبکه‌ی رسم شده در شکل ۱.۵ را در نظر بگیرید. در این شبکه سوییچ d در لیست سیاه قرار دارد، بنابراین در هیچ لحظه نباید از a که ورودی شبکه است در دسترس باشد. بنابراین در شبکه عدم دسترسی a به d را به عنوان ویژگی در نظر می‌گیریم. در شبکه‌ی بالا ابتدا مسیرهایی که با خط پررنگ مشخص شده‌اند وجود دارند. در ادامه هر یک از مسیرها با مسیرهای خط‌چین جایگزین می‌شوند. فرض کنید به روز رسانی این مسیرها توسط دو پردازه هم‌روند انجام می‌شود. واضح است که اگر هر دوی این به‌روز رسانی‌ها انجام شوند دسترسی به سوییچی که در لیست سیاه قرار دارد ممکن می‌شود. اکنون فرض کنید که از عبارات زیر برای توصیف این شبکه در نت‌کت پویا استفاده کنیم:

$$P = p!1$$

$$Q = q!1$$

$$N = F \oplus p?1; N_p \oplus q?1; N_q$$

$$N_p = F_p \oplus q?1; F_{pq}$$

$$N_q = F_q \oplus p?1; F_{pq}$$

$$F = a \to b \oplus c \to b$$

$$F_p = a \to c \oplus c \to b \oplus a \to b$$

$$F_q = a \to b \oplus c \to d$$

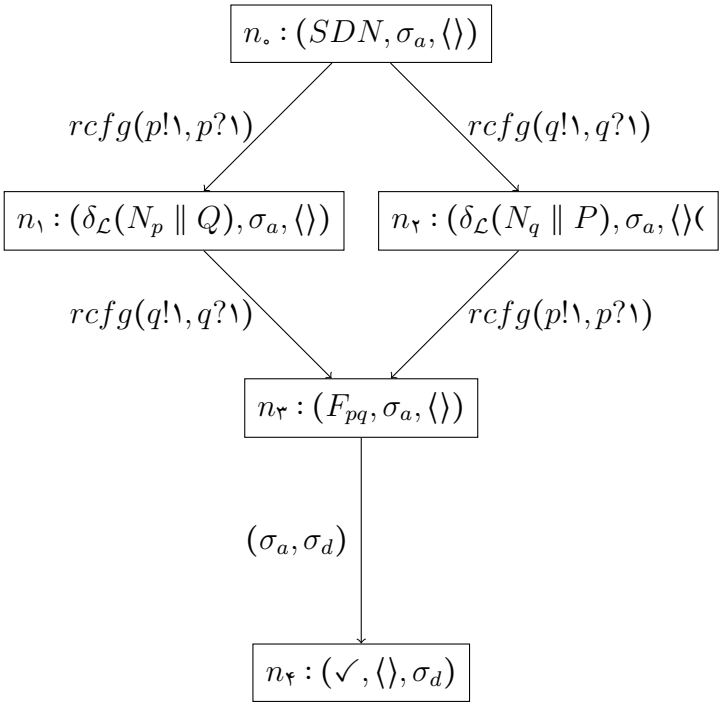
$$F_{pq} = a \to c \oplus c \to d \oplus a \to d$$

$$SDN = \delta_{\mathcal{L}}(N \parallel P \parallel Q)$$

$$\mathcal{L} = \{p!1, p?1, q?1, q?1\}$$

در توصیف بالا پردازه‌های P و Q به ترتیب وظیفه‌ای ارسال پیام برای به روز رسانی مسیرهای سبز و نارنجی را دارند. پردازه‌ی N رفتار ابتدایی شبکه و پردازه‌های N_p و N_q به ترتیب رفتار شبکه را پس از به روز رسانی مسیرهای سبز و نارنجی توصیف می‌کنند. پردازه های F, F_p, F_q, F_{pq} رفتارهای ارسالی [۲] شبکه را توصیف می‌کنند. در

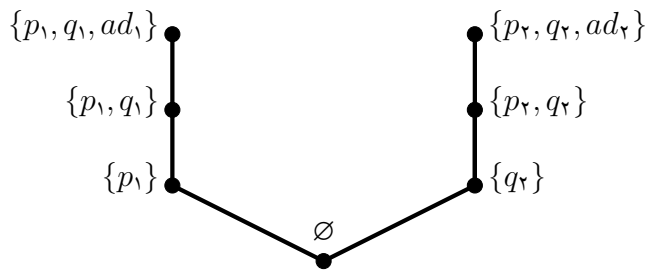
[2] Forwarding



شکل ۲.۵

نهایت رفتار کلی شبکه توسط پردازه‌ی SDN توصیف شده است که حاصل ترکیب موازی پردازه‌های N, P, Q و جلوگیری از اجرای عملیات‌های همگام نشده است. در توصیف بالا امکان اجرای هر دو به روز رسانی وجود دارد بنابراین شبکه این امکان را دارد که به حالتی برسد که مسیری از a به d در آن وجود داشته باشد. برای مثال فرض کنید که σ_a یک بسته وارد شده به شبکه باشد که داشته باشیم: $\sigma_a(sw) = a$. شکل ۲.۵ بخشی از نمودار سیستم انتقال این شبکه را زمانی که این بسته به شبکه وارد شود نشان می‌دهد. اگر فرض کنیم σ_d بسته‌ای باشد که $\sigma_d(sw) = d$ همانطور که در نمودار مشخص است دو مسیر به حالتی که بسته از سوییچ a به d برسد وجود دارد. به دلیل هم‌روندی پردازه‌های P و Q دو ترتیب برای اجرای این به‌روز رسانی‌ها وجود دارد و به همین دلیل دو مسیر منجر به خطا در این شبکه وجود دارد. اکنون می‌خواهیم علت بروز این خطا را پیدا کنیم. فرض کنید $\mathrm{E} = [\![SDN]\!]$ ساختمان رویداد این شبکه و $\mathcal{M}$ مدل علی E بر اساس مدل تعریف شده در ۳.۴ باشد. در این مدل تابع متغیر PV را به صورت زیر تعریف می‌کنیم:

$$F_{PV}(\vec{V}_{PV}) = \exists c \in \mathcal{F}(ES(\vec{v})).\exists e \in c.l(e) = a \to d$$

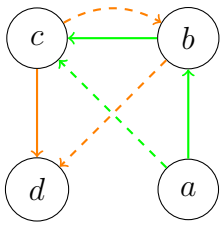


شکل ۳.۵

تابع بالا رفتار نا امن را وجود پیکربندی‌ای که شامل رویدادی با برچسب $a \to d$ باشد توصیف می‌کند. با توجه به ترتیب اجرای به‌روزرسانی‌ها در شبکه دو رویداد برای هر یک از عملیات‌های $rcfg(p!1, p?1)$، $rcfg(q!1, q?1)$ و $a \to d$ در ساختمان رویداد وجود دارد. فرض کنید برای رویدادهای مرتبط با این عملیات‌ها شش رویداد $p_1, p_2, q_1, q_2, ad_1, ad_2$ وجود داشته باشد که برچسب آن‌ها به صورت زیر باشد:

$$l(p_1) = rcfg(p!1, p?1)$$
$$l(p_2) = rcfg(p!1, p?1)$$
$$l(q_1) = rcfg(q!1, q?1)$$
$$l(q_2) = rcfg(q!1, q?1)$$
$$l(ad_1) = a \to d$$
$$l(ad_2) = a \to d$$

شکل ۳.۵ قسمتی از نمودار ساختمان رویداد این شبکه را نشان می‌دهد که در آن تمام پیکربندی‌هایی که یکی از رویدادهای ad_1 یا ad_2 را داشته باشند قابل دسترس باشد. با استفاده از مدل علّی در این مثال می‌توانیم $C(p_1, q_1) = \mathrm{F}$ را به عنوان یک علت برای نقض ویژگی لیست سیاه معرفی کنیم در صورتی که از $(C(p_2, q_2), \mathrm{T}, \mathrm{T})$ به عنوان شاهد استفاده کنیم. در E پیکربندی $\{p_1, q_1, ad_1\}$ قابل دسترسی است. بنابراین مقدار PV صحیح است. همچنین بین رویدادهای p_1 و q_1 تعارضی وجود ندارد پس $C(p_1, q_1) = \mathrm{F}$. بنابراین شرط ۱ در تعریف ۲.۶.۲ برقرار است. اکنون فرض کنید که مقدار $C(p_1, q_1)$ و $C(p_2, q_2)$ را برابر صحیح قرار دهیم. در این حالت هیچ یک از پیکربندی‌های $\{p_1, q_1, ad_1\}$ و $\{p_2, q_2, ad_2\}$ دیگر نمی‌توانند عضوی



شکل ۴.۵

از پیکربندی‌های $ES(\vec{v})$ در $\mathcal{M}$ باشند. پس در این حالت مقدار PV غلط می‌شود بنابراین شرط ۲.الف برقرار می‌شود. برای بررسی برقراری شرط ۲.ب باید فرض کنیم که مقدار $C(p_1, q_1)$ غلط است. توجه کنید که در این شرایط پیکربندی $\{p_1, q_1, ad_1\}$ عضوی از پیکربندی‌های $ES(\vec{v})$ است و مقدار $C(p_2, q_2)$ روی این مساله تاثیری ندارد. همچنین برگرداندن مقادیر بقیه متغیرها به مقدار اولیه آن‌ها باعث حذف $\{p_1, q_1, ad_1\}$ از مجموعه‌ی پیکربندی‌ها نمی‌شود بنابراین شرط ۲.ب هم برقرار است. با توجه به اینکه علت تنها شامل یک جمله است بنابراین شرط مینیمال بودن هم برقرار است. بنابراین در نهایت می‌توان نتیجه گرفت که $C(p_1, q_1)$ یک علت واقعی برای بروز خطا در این شبکه است. در این مثال مشخص است که علت پیدا شده با علتی که به صورت شهودی باعث بروز خطا بوده است تطبیق دارد.

۳.۵ نبود دور

این ویژگی بیان می‌کند که شبکه نباید هرگز دارای دور باشد [۱۲]. وجود دور در شبکه می‌تواند باعث مشکلاتی مانند دور زدن یک بسته در شبکه بدون رسیدن به مقصد و در نتیجه کاهش کارایی شبکه شود. به عنوان مثال شبکه‌ی رسم شده در شکل ۴.۵ را در نظر بگیرید. در ابتدا مسیری از a به d وجود دارد. در این شبکه دو به روز رسانی بر روی سوییچ‌های a و c انجام می‌شود تا مسیر جدیدی از a به d ایجاد شود که اینبار ابتدا از c

عبور می‌کند. می‌توانیم از توصیف نت‌کت پویای زیر برای توصیف این شبکه استفاده کنیم:

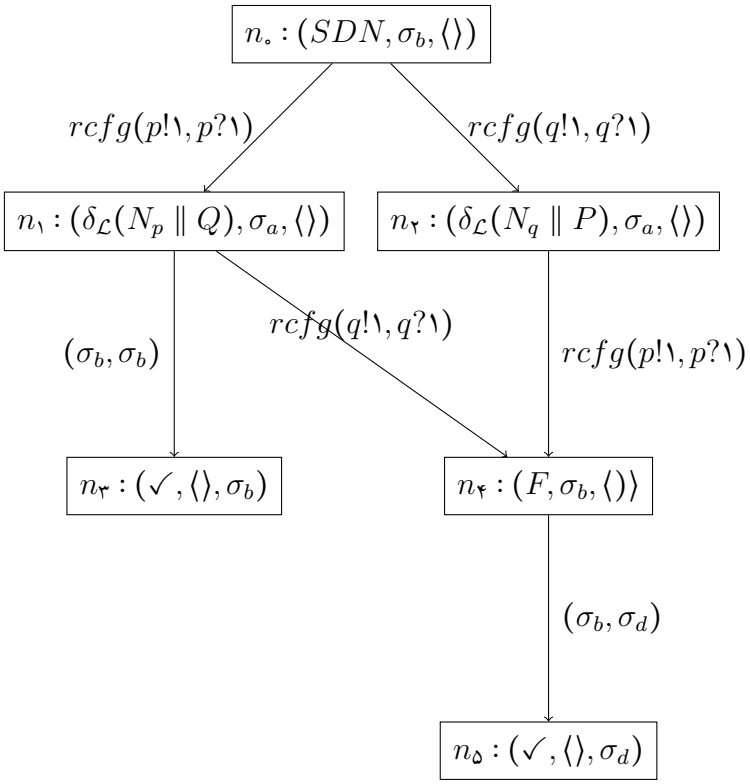
$$P = p!1$$
$$Q = q!1$$
$$N = F \oplus p?1; N_p \oplus q?1; N_q$$
$$N_p = F_p \oplus q?1; F$$
$$N_q = F_q \oplus p?1; F$$
$$SDN = \delta_{\mathcal{L}}(N \parallel P \parallel Q)$$
$$\mathcal{L} = \{p!1, p?1, q!1, q?1\}$$

$$F = a \to b \oplus a \to c \oplus a \to d \oplus b \to c \oplus b \to d \oplus c \to d$$
$$F_p = a \to c \oplus a \to d \oplus c \to d$$
$$F_q = a \to b \oplus a \to c \oplus a \to d \oplus b \to c \oplus b \to b \oplus b \to d \oplus c \to b \oplus c \to c \oplus c \to d$$

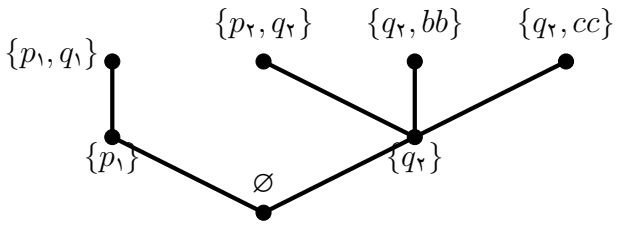
در توصیف بالا پردازه‌های P و Q به ترتیب وظیفه‌ای ارسال پیام برای به روز رسانی مسیرهای سبز و نارنجی را دارند. توجه کنید که در این توصیف پس از اجرای هر دو به روزرسانی رفتار ارسالی شبکه همانند رفتار اولیه خود می‌شود. همانطور که در شکل ۴.۵ مشخص است اگر به روز رسانی نارنجی پیش از به روز رسانی سبز انجام شود در شبکه یک دور شامل گره‌های c و b ایجاد می‌شود. شکل ۵.۵ قسمتی از سیستم انتقال برچسب‌دار شبکه را در حالتی که یک بسته ورودی روی سوییچ b وجود داشته باشد را نشان می‌دهد. همانطور که در شکل مشخص است پس از انجام به روز رسانی مسیر نارنجی امکان عملیاتی به شکل (σ_b, σ_b) وجود دارد که به معنی وجود حلقه در این شبکه است. اما اگر به روز رسانی مسیر سبز هم انجام شود تنها عملیات ممکن روی بسته ارسال آن به سوییچ d است. اکنون فرض کنید که $\mathrm{E} = [\![SDN]\!]$ ساختمان رویداد این شبکه و $\mathcal{M}$ مدل علی E بر اساس تعریف باشد. در این مدل تابع متغیر PV را به صورت زیر تعریف می‌کنیم:

$$F_{PV}(\vec{V}_{PV}) = \bigvee_{c \in C} c \in \mathcal{F}(ES(\vec{v}))$$
$$C = \{c \subset E | \exists e \in c.l(e) = b \to b \vee l(e) = c \to c\}$$

در این تابع رفتار نا امن وجود پیکربندی‌ای شامل یکی از برچسب‌های $b \to b$ یا $c \to c$ در شبکه است. همانند مثال قبل با توجه به ترتیب اجرای به‌روزرسانی‌ها در شبکه دو رویداد برای هر یک از عملیات‌های $rcfg(p!۱, p?۱)$ و $rcfg(q!۱, q?۱)$ در ساختمان رویداد وجود دارد. فرض کنید برای رویدادهای مرتبط با این عملیات‌ها چهار


$n_0 : (SDN, \sigma_b, \langle\rangle)$
$rcfg(p!1, p?1)$
$rcfg(q!1, q?1)$
$n_1 : (\delta_{\mathcal{L}}(N_p \parallel Q), \sigma_a, \langle\rangle)$
$n_2 : (\delta_{\mathcal{L}}(N_q \parallel P), \sigma_a, \langle\rangle)$
(σ_b, σ_b)
$rcfg(q!1, q?1)$
$rcfg(p!1, p?1)$
$n_3 : (\checkmark, \langle\rangle, \sigma_b)$
$n_4 : (F, \sigma_b, \langle\rangle)$
(σ_b, σ_d)
$n_5 : (\checkmark, \langle\rangle, \sigma_d)$

شکل ۵.۵

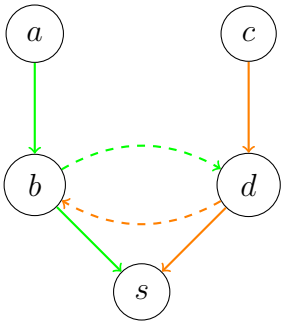


شکل ۶.۵

رویداد p_1, p_2, q_1, q_2 وجود داشته باشد که برچسب آن‌ها به صورت زیر باشد:

$$l(p_1) = rcfg(p!1, p?1)$$
$$l(p_2) = rcfg(p!1, p?1)$$
$$l(q_1) = rcfg(q!1, q?1)$$
$$l(q_2) = rcfg(q!1, q?1)$$

همچنین فرض کنید برچسب رویدادهای bb, cc به ترتیب $b \to b, c \to c$ باشد. شکل ۶.۵ قسمتی از نمودار ساختمان رویداد این شبکه را نشان می‌دهد که در آن تمام پیکربندی‌هایی وجود داشته باشد که یکی از رویدادهای bb یا cc قابل دسترس باشد. در اینجا می‌توان $M(\{p_2\}, q_2) = \mathrm{F}$ را به عنوان علت واقعی بروز خطا معرفی کرد. اگر مقدار $M(\{p_2\}, q_2)$ را برابر صحیح قرار دهیم آن‌گاه با توجه به تابع $M(\varnothing, q_2)$ که در بخش ۳.۴ تعریف شده است، مقدار $M(\varnothing, q_2)$ غلط شده و در نتیجه مقدار $EN(\varnothing, q_2)$ هم غلط می‌شود. این مساله باعث می‌شود که دیگر هیچ‌کدام از پیکربندی‌های شاخه‌ی سمت راست شکل ۶.۵ دیگر عضو پیکربندی‌های $ES(\vec{v})$ نباشند و در نتیجه مقدار PV غلط می‌شود. با توجه به اینکه در شاهد $\vec{W}$ خالی است می‌توان نتیجه گرفت که $M(\{p_2\}, q_2) = \mathrm{F}$ علت واقعی به وجود آمدن دور در این شبکه است.



شکل ۷.۵

۴.۵ نبود سیاه‌چاله

در یک شبکه سیاه‌چاله‌ها [3] عناصری در شبکه هستند که وظیفه ارسال بسته‌ها را دارند (مثلا سوییچ یا روتر) ولی برخی از بسته‌ها را پس از دریافت به جایی ارسال نمی‌کنند و در واقع مانند سیاه‌چاله این بسته‌ها در آن‌ها گم می‌شوند [۲۹]. در یک شبکه که مکان‌های ورودی و خروجی مشخص دارد عدم وجود سیاه‌چاله در شبکه را می‌توان معادل این ویژگی که همه‌ی بسته‌های ورودی به شبکه از آن خارج شوند دانست. به عنوان مثال شبکه‌ی موجود در شکل ۷.۵ را در نظر بگیرید. فرض کنید که در این شبکه a, c ورودی‌های شبکه و s خروجی شبکه باشد. در این شبکه دو به روز رسانی برای جایگزینی مسیر ds با db و مسیر bs با bd انجام می‌شود. در این شبکه در حالت ابتدایی و پس از انجام یکی از به روزرسانی‌ها ورودی‌ها به خروجی مسیر وجود دارد اما اگر هر دوی این به روز رسانی‌ها انجام شوند دیگر s قابل دسترسی نیست و عملا بسته‌های ورودی به شبکه به خروجی نمی‌رسند. این شبکه را می‌توانیم به فرم زیر در نت‌کت پویا توصیف کنیم:

$$P = p!1$$

$$Q = q!1$$

$$N = F \oplus p?1; N_p \oplus q?1; N_q$$

$$N_p = F \oplus q?1; F_{pq}$$

$$N_q = F \oplus p?1; F_{pq}$$

$$F = a \to s \oplus c \to s$$

$$F_{pq} = a \to b \oplus c \to d$$

$$SDN = \delta_{\mathcal{L}}(N \parallel P \parallel Q)$$

$$\mathcal{L} = \{p!1, p?1, q?1, q?1\}$$

[3] Blackhole

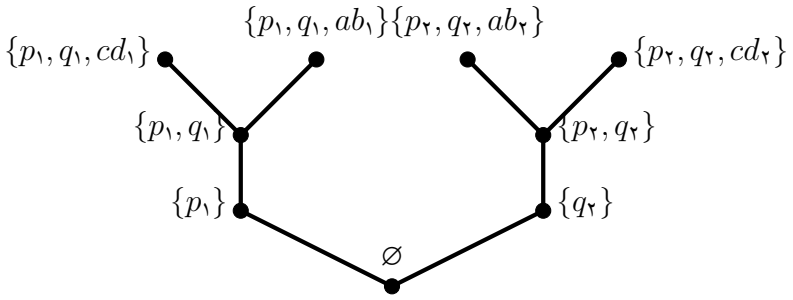
در ادامه فرض کنید که $\mathcal{M}$ مدل علّی این شبکه باشد. در این مدل تابع متغیر PV را به صورت زیر تعریف می‌کنیم:

$$F_{PV}(\vec{V}_{PV}) = \exists c \in \mathcal{F}(ES(\vec{v})), \exists e \in c.l(e) = \alpha \cdot \pi \wedge \pi(sw) \neq s$$

در تعریف این تابع رفتار نا امن وجود یک پیکربندی شامل رویدادی با برچسب از نوع $\alpha \cdot \pi$ یا به عبارت دیگر رویدادی از نوع ارسال بسته است که سوییچ مقصد ارسال آن سوییچ s نباشد. همانند مثال مربوط نبود دور در شبکه، در این مثال هم با توجه به ترتیب اجرای به روز رسانی‌ها دو رویداد برای هر یک از عملیات‌های $rcfg(p!۱, p?۱)$، $rcfg(q!۱, q?۱)$ و $a \rightarrow b, ac \rightarrow d$ وجود دارد. بنابراین فرض کنید که رویدادهای p_i, q_i, ab_i, cd_i به ازای $i = ۱, ۲$ در ساختمان رویداد این مدل وجود داشته باشند و برچسب‌گذاری آن‌ها به صورت زیر باشد:

$$l(p_1) = rcfg(p!1, p?1)$$
$$l(p_2) = rcfg(p!1, p?1)$$
$$l(q_1) = rcfg(q!1, q?1)$$
$$l(q_2) = rcfg(q!1, q?1)$$
$$l(ab_1) = a \rightarrow b$$
$$l(ab_2) = a \rightarrow b$$
$$l(cd_1) = c \rightarrow d$$
$$l(cd_2) = c \rightarrow d$$

پیکربندی‌هایی از این ساختمان رویداد که شامل رویدادی با برچسب $a \rightarrow b$ یا $c \rightarrow d$ باشند نقض شدن ویژگی



شکل ۸.۵

نبود سیاه‌چاله را نشان می‌دهند. بنابراین تابع متغیر PV را به فرم زیر توصیف می کنیم:

$$F_{PV}(\vec{V}_{PV}) = \exists c \in \mathcal{F}(ES(\vec{v})), \exists e \in c.l(e) = \alpha \cdot \pi \wedge \pi(sw) \neq s$$

در این مدل هم همانند مثال نبود دور در شبکه عدم وجود تعارض بین رویدادهای p_1 و q_1 را می‌توان به عنوان علت واقعی نقض شدن ویژگی در نظر گرفت. برای این منظور از شاهد $(C_{p_2,q_2}, \mathrm{T}, \mathrm{T})$ استفاده می‌کنیم. واضح است که اگر مقدار هر دو متغیر C_{p_1,q_1} و C_{p_2,q_2} را برابر غلط قرار دهیم آنگاه هیچ یک از پیکربندی‌های $\{p_1, q_1, cd_1\}, \{p_1, q_1, ab_1\}, \{p_2, q_2, ab_2\}, \{p_2, q_2, cd_2\}$ دیگر عضوی از پیکربندی‌های $ES(\vec{v})$ نیستند. از طرفی در شرایطی که C_{p_1,q_1} مقدار درست داشته باشد آنگاه پیکربندی‌های $\{p_1, q_1, ab_1\}, \{p_1, q_1, cd_1\}$ عضو $ES(\vec{v})$ هستند و مقدار متغیر C_{p_2,q_2} تاثیری روی این مساله ندارد بنابراین در نهایت می‌توان نتیجه گرفت که $C_{p_1,q_1} = \mathrm{F}$ علت واقعی نقض ویژگی است.

فصل ۶

جمع‌بندی و کارهای آینده

۱.۶ جمع‌بندی کارهای انجام‌شده

در این پژوهش روشی برای استفاده از تعریف علت واقعی مطابق [۱۷] برای پیدا کردن علت خطا در برنامه‌های توصیف شده در زبان نت‌کت پویا ارائه شد.

در این پژوهش سعی شد تا از مفهوم علت واقعی برای پیدا کردن علت خطا در شبکه‌های نرم‌افزاری استفاده شود. زبان نت‌کت پویا برای توصیف شبکه‌های نرم‌افزاری انتخاب شد. برای این منظور زبان نت‌کت پویا به عنوان زبان برنامه‌ی شبکه انتخاب شد. دلیل انتخاب نت‌کت پویا این است که اولا این زبان چون بر اساس نت‌کت بنا شده سادگی و مینیمال بودن خود را حفظ کرده است که این مساله به ساده‌تر شدن مساله کمک می‌کند. ثانیا نت‌کت پویا امکان به روز رسانی شبکه را فراهم می‌کند و از آن‌جایی که هدف نهایی پیدا کردن علت خطا تسهیل کردن فرآیند رفع خطا است، وجود ساختارهایی در زبان برای به روز رسانی‌های شبکه کمک می کند تا این ساختار ها هم در پیدا کردن علت نقش داشته باشند و کمک بیشتری به رفع خطا کنند. برای اختصاص معنا به عبارات نت‌کت پویا از ساختمان رویداد استفاده شده است. در نهایت برای بر اساس تعریف مدل علّی ارائه شده در [۱۷] مدل علّی ساختمان رویداد در این پژوهش ارائه شده است. برای بررسی کارایی این مدل چند دسته از ویژگی‌های مطرح شبکه مورد بررسی قرار گرفته‌اند و تطابق علت واقعی پیدا شده با شهود موجود از مساله مورد بحث قرار گرفته است.

۲.۶ نوآوری‌ها و دستاوردها

۱.۲.۶ جستجو در ساختار

۲.۲.۶ ساختمان رویداد

۳.۲.۶ استفاده مستقیم از تعریف علت

در این پژوهش سعی شد تا از رویکرد متفاوتی نسبت به روش‌های پیشین برای توصیف خطا استفاده شود. در پژوهش‌هایی مانند [۲۴، ۸، ۴] المان‌هایی در رفتار یا وضعیت سیستم به عنوان علت بروز خطا در نظر گرفته می‌شوند. اما در این پژوهش در میان المان‌های ساختاری سیستم به جستجوی علت پرداخته شده است. یکی از مزایایی که برای این روش می‌توان در نظر گرفت استفاده آسان‌تر از آن برای فرآیند تعمیر یا سنتز خودکار نرم‌افزار است. چون در اینجا ساختارهای سیستم به عنوان علت پیدا می‌شوند این موضوع به تولید تعمیر خودکار کمک می‌کند. مساله‌ی دیگر استفاده از ساختمان رویداد به عنوان مدل معنایی است. همانطور که پیش‌تر بیان شد ساختمان رویداد یک مدل غیرجایگذاری شده است که در آن هم‌روندی به صورت صریح مشخص می‌شود. این موضوع سبب می‌شود که اولا در فرآیند پیدا کردن علت، ساختارهای جعلّی مانند جایگذاری پردازه‌های هم روند که یک ترتیب برای اجرای آن‌ها ایجاد می‌کند به عنوان علت پیدا نشوند، ثانیا بتوان هم‌روندی دو پردازه یا عملیات را به عنوان علت تعریف کرد، کاری که در مدل‌های جایگذاری شده به دلیل اینکه هم‌روندی صراحتا توصیف نمی‌شود ممکن نیست. یکی دیگر از تفاوت‌های این روش با روش‌هایی مانند [۸، ۲۴] در استفاده از تعریف علت واقعی است. در پژوهش‌های ذکر شده با اقتباس از تعریف HP تعریف جدیدی برای علت واقعی ارائه شده است و در مورد معادل بودن این تعاریف بحثی نشده است. در مقابل در پژوهش جاری به صورت مستقیم از تعریف HP استفاده شده است.

۳.۶ محدودیت‌ها

۱.۳.۶ پیچیدگی زمانی

یک ساختمان رویداد با n رویداد را در نظر بگیرید. مدل علّی این ساختمان رویداد شامل $O(n2^n)$ متغیر است. برای پیدا کردن علت واقعی در این مدل و به طور خاص برای بررسی شرط ۲.ب لازم است تا تمامی زیر مجموعه‌های یک افراز از این متغیرها بررسی شود که در بهترین حالت پیچیدگی زمانی $O(2^{n2^n})$ دارد. بنابراین پیاده‌سازی این روش بدون بهینه‌سازی یا استفاده از روش‌های ابتکاری عملا ممکن نیست.

۲.۳.۶ توصیف خطا در سطح مدل علی

در روش ارائه شده در این پژوهش برای پیدا کردن علت خطا در یک برنامه توصیف شده در زبان نت‌کت پویا، لازم است تا رفتار نا امن در قالب یک تابع در مدل علّی و به عنوان یک شرط بر روی مجموعه‌ی پیکربندی‌های ساختمان رویداد منتج شده از آن توصیف شود. این مساله کار کردن با این روش را برای کاربر سخت می‌کند. راه حل مناسب ارائه یک منطق در سطح زبان است که به کاربر اجازه‌ی توصیف رفتار نا امن یا در روش بهتر اجازه‌ی توصیف ویژگی مورد نظر در قالب یک منطق را بدهد.

۳.۳.۶ استدلال در مورد یک علت

روش ارائه شده در این پژوهش می‌تواند برای اثبات اینکه چه ساختاری از برنامه علت خطا است به کار رود. ولی این مساله به تنهایی برای تسهیل فرآیند اشکال‌زدایی سیستم کافی نیست. برای اینکه علت خطا بتواند به شکل کاربردی در فرآیند اشکال‌زدایی مورد استفاده قرار گیرد لازم است تا مشابه روش‌هایی مانند [۴] تمامی علت‌های ممکن برای خطا پیدا شوند و به کاربر نمایش داده شوند.

۴.۶ کارهای آینده

۱.۴.۶ ترکیب علت

در [۸] نویسندگان ثابت کرده‌اند که امکان ترکیب علت‌ها در اجزای یک پردازه برای پیدا کردن علت در آن پردازه در شرایطی که پردازه‌ها ارتباط

۲.۴.۶ سنتز تعمیر

از علت‌های پیدا شده در سیستم برای بروز خطا می‌توان برای سنتز تعمیر مناسب برای سیستم استفاده کرد. به طور خاص در این روش که علت خطا در ساختارهای سیستم جستجو می‌شود سنتز کردن تعمیر فرآیند ساده‌تری خواهد بود.

مراجع

[1] Al-Shaer, Ehab and Al-Haj, Saeed. Flowchecker: Configuration analysis and verification of federated openflow infrastructures. In *Proceedings of the 3rd ACM workshop on Assurable and usable security configuration*, pages 37–44, 2010. 4

[2] Anderson, Carolyn Jane, Foster, Nate, Guha, Arjun, Jeannin, Jean-Baptiste, Kozen, Dexter, Schlesinger, Cole, and Walker, David. Netkat: Semantic foundations for networks. *Acm sigplan notices*, 49(1):113–126, 2014. 4, 6, 10, 18

[3] Baier, Christel, Dubslaff, Clemens, Funke, Florian, Jantsch, Simon, Majumdar, Rupak, Piribauer, Jakob, and Ziemek, Robin. From verification to causality-based explications. *arXiv:2105.09533 [cs]*, May 2021. arXiv: 2105.09533. 5

[4] Beer, Ilan, Ben-David, Shoham, Chockler, Hana, Orni, Avigail, and Trefler, Richard. Explaining counterexamples using causality. *Formal Methods in System Design*, 40(1):20–40, Feb 2012. 36, 37, 40, 64, 65

[5] Caltais, Georgiana, Guetlein, Sophie Linnea, and Leue, Stefan. Causality for general ltl-definable properties. *Electronic Proceedings in Theoretical Computer Science*, 286:1–15, Jan 2019. 37

[6] Caltais, Georgiana, Hojjat, Hossein, Mousavi, Mohammad, and Tunc, Hunkar Can. Dynetkat: An algebra of dynamic networks. *arXiv preprint arXiv:2102.10035*, 2021. 4, 6, 19, 24, 45, 46

[7] Caltais, Georgiana, Leue, Stefan, and Mousavi, Mohammad Reza. (de-)composing causality in labeled transition systems. *Electronic Proceedings in Theoretical Computer Science*, 224:10–24, Aug 2016. 39

[8] Caltais, Georgiana, Mousavi, Mohammad Reza, and Singh, Hargurbir. Causal reasoning for safety in hennessy milner logic. *Fundamenta Informaticae*, 173(2–3):217–251, Mar 2020. 39, 64, 66

[9] Chockler, Hana, Halpern, Joseph Y., and Kupferman, Orna. What causes a system to satisfy a specification? *ACM Transactions on Computational Logic*, 9(3):1–26, Jun 2008. 35, 40

[10] Clarke, Edmund M. Model checking. In *International Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 54–56. Springer, 1997. 4

[11] Clarke, Edmund M and Wing, Jeannette M. Formal methods: State of the art and future directions. *ACM Computing Surveys (CSUR)*, 28(4):626–643, 1996. 4

[12] Foerster, Klaus-Tycho, Schmid, Stefan, and Vissicchio, Stefano. Survey of consistent software-defined network updates. *IEEE Communications Surveys & Tutorials*, 21(2):1435–1461, 2018. 4, 5, 10, 55

[13] Foster, Nate, Harrison, Rob, Freedman, Michael J, Monsanto, Christopher, Rexford, Jennifer, Story, Alec, and Walker, David. Frenetic: A network programming language. *ACM Sigplan Notices*, 46(9):279–291, 2011. 4

[14] Gössler, Gregor and Le Métayer, Daniel. A general trace-based framework of logical causality. In *International Workshop on Formal Aspects of Component Software*, pages 157–173. Springer, 2013. 38

[15] Groce, Alex, Chaki, Sagar, Kroening, Daniel, and Strichman, Ofer. Error explanation with distance metrics. *International Journal on Software Tools for Technology Transfer*, 8(3):229–247, 2006. 35

[16] Groce, Alex and Visser, Willem. What went wrong: Explaining counterexamples. In *International SPIN Workshop on Model Checking of Software*, pages 121–136. Springer, 2003. 35

[17] Halpern, Joseph Y. and Pearl, Judea. Causes and explanations: A structural-model approach, part i: Causes. *arXiv:cs/0011012*, Nov 2005. arXiv: cs/0011012. 5, 6, 27, 29, 33, 35, 40, 63

[18] Halpern, Joseph Y. and Pearl, Judea. Causes and explanations: A structural-model approach. part ii: Explanations. *The British journal for the philosophy of science*, 56(4):889–911, 2005. 36

[19] Hennessy, Matthew and Milner, Robin. On observing nondeterminism and concurrency. In *International Colloquium on Automata, Languages, and Programming*, pages 299–309. Springer, 1980. 39

[20] Khurshid, Ahmed, Zou, Xuan, Zhou, Wenxuan, Caesar, Matthew, and Godfrey, P Brighten. {VeriFlow}: Verifying {Network-Wide} invariants in real time. In *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 15–27, 2013. 4

[21] Kölbl, Martin, Leue, Stefan, and Schmid, Robert. Dynamic causes for the violation of timed reachability properties. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 127–143. Springer, 2020. 38

[22] Kozen, Dexter. Kleene algebra with tests. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 19(3):427–443, 1997. 4, 10

[23] Kreutz, Diego, Ramos, Fernando MV, Verissimo, Paulo Esteves, Rothenberg, Christian Esteve, Azodolmolky, Siamak, and Uhlig, Steve. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, 2014. 3

[24] Leitner-Fischer, Florian and Leue, Stefan. Causality checking for complex system models. In *International Workshop on Verification, Model Checking, and Abstract Interpretation*, page 248–267. Springer, 2013. 37, 39, 64

[25] Lewis, David. *Counterfactuals*. Wiley, 1973. 35

[26] McKeown, Nick, Anderson, Tom, Balakrishnan, Hari, Parulkar, Guru, Peterson, Larry, Rexford, Jennifer, Shenker, Scott, and Turner, Jonathan. Openflow: enabling innovation in campus networks. *ACM SIGCOMM computer communication review*, 38(2):69–74, 2008. 3, 10

[27] Monsanto, Christopher, Foster, Nate, Harrison, Rob, and Walker, David. A compiler and run-time system for network programming languages. *Acm sigplan notices*, 47(1):217–230, 2012. 4

[28] Monsanto, Christopher, Reich, Joshua, Foster, Nate, Rexford, Jennifer, and Walker, David. Composing software defined networks. In *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 1–13, 2013. 4

[29] Reitblatt, Mark, Foster, Nate, Rexford, Jennifer, Schlesinger, Cole, and Walker, David. Abstractions for network update. *ACM SIGCOMM Computer Communication Review*, 42(4):323–334, 2012. 51, 59

[30] Ruchansky, Natali and Proserpio, Davide. A (not) nice way to verify the openflow switch specification: Formal modelling of the openflow switch using alloy. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, pages 527–528, 2013. 4

[31] Sassone, Vladimiro, Nielsen, Mogens, and Winskel, Glynn. Models for concurrency: Towards a classification. *Theoretical Computer Science*, 170(1-2):297–348, 1996. 24

[32] Voellmy, Andreas and Hudak, Paul. Nettle: Taking the sting out of programming network routers. In *International Symposium on Practical Aspects of Declarative Languages*, pages 235–249. Springer, 2011. 4

[33] Voellmy, Andreas, Kim, Hyojoon, and Feamster, Nick. Procera: A language for high-level reactive network control. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 43–48, 2012. 4

[34] Winskel, Glynn. *Event structures*, volume 255 of *Lecture Notes in Computer Science*, page 325–392. Springer Berlin Heidelberg, Berlin, Heidelberg, 1987. 24

[35] Zeller, Andreas. *Why programs fail: a guide to systematic debugging*. Elsevier, 2009. 35

[36] Zeng, Hongyi, Zhang, Shidong, Ye, Fei, Jeyakumar, Vimalkumar, Ju, Mickey, Liu, Junda, McKeown, Nick, and Vahdat, Amin. Libra: Divide and conquer to verify forwarding tables in huge networks. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 87–99, 2014. 4

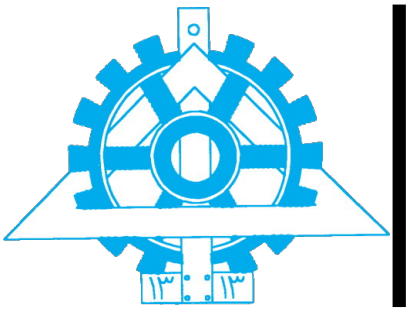
واژه‌نامهٔ فارسی به انگلیسی

واژه‌نامهٔ انگلیسی به فارسی

Abstract

This thesis studies on writing projects, theses and dissertations using tehran-thesis class. It ...

Keywords Writing Thesis, Template, LaTeX, XeLaTeX, Persian



University of Tehran
College of Engineering
Faculty of Electrical and Computer Engineering

Explaining Failures in Software-Defined Networks Using Casual Reasoning

A Thesis submitted to the Graduate Studies Office
In partial fulfillment of the requirements for
The degree of Master of Science
in Computer Engineering - Software Engineering

By:

Amir Hossein Seyhani

Supervisors:

Dr. Hossein Hojjat and Dr. Mohammad Reza Mousavi

Advisor:

First Advisor

September 2022