دانشگاه صنعتی شریف
دانشکده مهندسی کامپیوتر

پایان‌نامه کارشناسی ارشد
مهندسی نرم‌افزار

برخوردی صوری به آزمون میکروسرویس‌ها

نگارش

سجاد واحدی فرد

استاد راهنما

دکتر سید حسن میریان حسین آبادی

خرداد ۱۴۰۲


بسم الله الرحمن الرحیم

به نام خدا

دانشگاه صنعتی شریف

دانشکده مهندسی کامپیوتر

پایان‌نامه کارشناسی ارشد

این پایان‌نامه به عنوان تحقق بخشی از شرایط دریافت درجه کارشناسی ارشد است.

عنوان: برخوردی صوری به آزمون میکروسرویس‌ها

نگارش: سجاد واحدی فرد

کمیته ممتحنین

استاد راهنما: دکتر سید حسن میریان امضاء:

حسین آبادی

استاد مشاور: استاد مشاور امضاء:

استاد مدعو: دکتر جعفر حبیبی امضاء:

تاریخ:

سپاس

از استاد بزرگوارم که با کمک‌ها و راهنمایی‌های بی‌دریغشان، مرا در به سرانجام رساندن این پایان‌نامه یاری داده‌اند، تشکر و قدردانی می‌کنم. همچنین از همکاران عزیزی که با راهنمایی‌های خود در بهبود نگارش این نوشتار سهیم بوده‌اند، صمیمانه سپاسگزارم.

چکیده

معماری میکروسرویس[۱] یک سبک معماری نرم‌افزاری در حال گسترش است که اساساً با معماری یکپارچه[۲] و لایه‌ای متفاوت است. معماری میکروسرویس مبتنی بر خدمات کوچک است و مزیت‌هایی مانند استقلال، ترکیب‌پذیری، مقیاس‌پذیری و تحمل شکست را ارائه می‌دهد. با ظهور سبک معماری میکروسرویس، روشی که نرم‌افزار درک و طراحی می‌شود تغییر کرده است. بنابراین، نیاز به روش‌ها و ابزارهایی وجود دارد که به حل مسأله‌ی توصیف و درستی‌سنجی رفتار ارتباطی سیستم‌های میکروسرویس کمک می‌کنند.

در این پایان‌نامه، روشی مبتنی بر آزمون مدل‌ـ‌رانه[۳] و رویکرد صوری برای صحت‌سنجی[۴] و آزمون برنامه‌ها با معماری میکروسرویس ارائه شده است. این روش می‌تواند به صورت خودکار مدل گردش کاری برنامه‌ها را به توصیفی صوری ترجمه کند و با استفاده از توصیف به دست آمده، ساختار ارتباطی میکروسرویس‌ها را درستی‌سنجی کند و به صورت خودکار برای آن، موارد آزمون را تولید و حتی در فضای محدود آزمون‌ها را اجرا کند. روش ارائه‌شده درون یک ابزار موجود برای مدل‌سازی فرآروندهای کسب و کاری جای‌گذاری شده است. همچنین با استفاده از تحلیل جهش، بر روی نمونه‌های برگرفته از سیستم‌های واقعی، نشان داده شده است که روش ارائه‌شده کارایی بالایی دارد.

کلیدواژه‌ها: معماری میکروسرویس، آزمون مدل‌ـ‌رانه، گردش کار، توصیف صوری

---

[۱] Microservice Architecture (MSA)
[۲] Monolith
[۳] Model-Driven Testing
[۴] Verification

فهرست مطالب

۱ مقدمه ۱

۱-۱ تعریف مسئله ۱

۱-۲ اهمیت موضوع ۳

۱-۳ اهداف پژوهش ۳

۱-۴ ساختار پایان‌نامه ۴

۲ مفاهیم اولیه ۶

۲-۱ مفاهیم مربوط به میکروسرویس ۶

۲-۱-۱ میکروسرویس ۶

۲-۱-۲ میکروسرویس‌ها (معماری میکروسرویس) ۶

۲-۲ یاول ۸

۲-۲-۱ ساختار ذخیره‌سازی مدل‌ها در زبان یاول ۱۰

۲-۳ الوی ۱۰

۲-۳-۱ تحلیل‌گر الوی ۱۲

۲-۴ آزمون ۱۴

۳ کارهای پیشین ۲۲

۳-۱ پژوهش‌های مربوط به آزمون میکروسرویس‌ها ۲۲

۳-۱-۱ آزمون خودکار ۲۲

۳-۱-۲ معماری ۲۳
۳-۱-۳ توسعه‌ـعملیات و یکپارچه‌سازی مستمر ۲۴
۳-۱-۴ عملکرد ۲۴
۳-۱-۵ آزمون مبتنی بر مدل ۲۵

۴ روش پیشنهادی **۲۷**
۴-۱ چارچوب کلی ۲۷
۴-۲ مدل‌سازی سیستم‌های پیچیده با معماری میکروسرویس با یاول ۲۸
۴-۳ خودکارسازی ترجمه‌ی مدل‌های یاول به الوی ۲۹
۴-۳-۱ توصیف مدل‌ها در زبان الوی ۲۹
۴-۳-۲ جای‌گذاری روش در ابزار ویرایش یاول ۳۰
۴-۴ تحلیل صوری مدل‌ها ۳۱
۴-۵ ایجاد موارد آزمون و اعمال آن‌ها بر روی مدل ۳۳

۵ مورد مطالعاتی: برنامه‌ی خرید برخط **۳۶**

۶ ارزیابی **۴۷**
۶-۱ معیارهای ارزیابی ۴۷
۶-۲ قسمت ارزیابی برنامه‌ی ارائه‌ی محتوای چندرسانه‌ای ۴۸
۶-۳ قسمت ارزیابی برنامه‌ی خرید برخط ۴۹

مراجع **۵۳**

واژه‌نامه **۵۷**

آ مطالب تکمیلی **۶۱**

فهرست جداول

۲-۱ جهش‌یافته‌های نمونه ۲۰

۳-۱ دسته‌بندی موضوعات پژوهشی مرتبط با آزمون میکروسرویس ۲۶

۶-۱ جهش‌یافته‌های برنامه‌ی ارائه‌ی محتوای چندرسانه‌ای ۵۰

۶-۲ درصد آزمون‌های شکست‌خورده بعد از تولید جهش‌یافته‌ها در برنامه‌ی ارائه‌ی محتوای چندرسانه‌ای ۵۱

۶-۳ جهش‌یافته‌های برنامه‌ی خرید برخط ۵۱

۶-۴ درصد آزمون‌های شکست‌خورده بعد از تولید جهش‌یافته‌ها در برنامه‌ی خرید برخط . . ۵۲

۶-۵ ارزیابی کارایی روش ارائه‌شده از نظر زمانی ۵۲

فهرست تصاویر

۲-۱ معماری میکروسرویس برای یک برنامه‌ی ماشین حساب[۱] ۷

۲-۲ سیستم چراغ راهنماییalloybook ۱۳

۲-۳ آزمون مدل‌ـرانه[۲] ۱۵

۲-۴ مدل خطا/شکست و شرایط]۲RIPR[. ۱۶

۴-۱ روند کلی روش ارائه‌شده ۲۸

۴-۲ درخت عبارت منطقی برای گزاره‌های مسند $(a \wedge b) \vee c$ ۳۵

۵-۱ گردش کاری یک برنامه‌ی نمونه برای خرید برخط ۳۷

۵-۲ مدل گردش کاری برنامه‌ی خرید برخط در یاول ۳۷

۵-۳ وجود پیوند از نوع "یا" در مدل ۳۹

۵-۴ منتظر بودن دو پیوند از نوع "یا" در مثال خرید برخط ۴۰

۵-۵ میکروسرویس تایید پرداخت دارای گزاره‌های شرطی برای ایجاد آزمون

. . . . . ۴۱

۵-۶ شروع فرآیند ورود دارای گزاره‌های شرطی برای ایجاد آزمون

. . . . . ۴۶

۶-۱ گردش کاری یک برنامه‌ی نمونه‌ی ارائه محتوای چندرسانه‌ای ۴۸

فصل ۱

مقدمه

۱-۱ تعریف مسئله

معماری میکروسرویس سبکی از معماری است که در آن یک برنامه‌ی کاربردی در قالب مجموعه‌ای از اجزا یا پودمان‌ها ساخته می‌شود. هر پودمان از یک کار یا هدف تجاری خاص پشتیبانی می‌کند و از یک واسط که به صورت کامل تعریف شده و ساده است مانند واسط برنامه نویسی کاربردی[۱] برای برقراری ارتباط با دسته‌های دیگر از برنامه‌ها یا خدمات استفاده می‌کند[۳].

در سال‌های اخیر، میکروسرویس‌ها محبوبیت بیشتری پیدا کرده‌اند و شرکت‌ها ترجیح می‌دهند از معماری یکپارچه به معماری میکروسرویس حرکت کنند. به طور معمول، ممکن است صدها تا هزاران میکروسرویس در سیستم‌های مبتنی بر معماری میکروسرویس در برنامه‌ها با مقیاس بزرگ گنجانده شوند. به عنوان مثال، بیش از ۵۰۰ میکروسرویس نتفلیکس[۲] روزانه تقریباً ۲ میلیارد درخواست واسط برنامه‌نویسی کاربردی را انجام می‌دهند. [۴]، در حالی که سیستم وی‌چت تنسنت[۳] از بیش از ۳۰۰۰ سرویس تشکیل شده است که بر روی بیش از ۲۰۰۰۰ ماشین اجرا می‌شود[۵].

سبک معماری میکروسرویس از معماری خدمت‌گرا[۴] پدید آمده است و تاکید آن نیز بر ایجاد خدمات ریزدانه، توسعه‌ـعملیات[۵] و روش‌های چابک[۶] است[۶]. دو سبک معماری میکروسرویس و معماری خدمت‌گرا، به طور قابل توجهی از نظر ویژگی‌های هر خدمت، مانند ارتباطات خدمات و به اشتراک گذاری

[۱] Application Programming Interface (API)
[۲] Netflix
[۳] WeChat Tencent
[۴] Service Oriented Architecture (SOA)
[۵] Development-Operations (DevOps)
[۶] Agile

مولفه[۷]ها متفاوت هستند؛ به عنوان مثال، معماری میکروسرویس می‌تواند شامل صدها سرویس ریزدانه باشد که با هم یک خدمت را ارائه می‌کنند، در حالی که هر کدام از این میکروسرویس‌ها با سرعت زیاد تکامل می‌یابند. اما سبک معماری خدمت‌گرا شامل خدمات درشت‌دانه‌تر و پایدارتری است. در معماری میکروسرویس، ارتباط از طریق یک لایه واسط برنامه‌نویسی کاربردی برقرار می‌شود، در حالی که در معماری خدمت‌گرا می‌توان از طریق یک واسط برنامه‌نویسی کاربردی (API) در کنار یک گذرگاه خدمات سازمانی[۸] ارتباطات را برقرار کرد. در کل معماری میکروسرویس مقیاس‌پذیری، جداسازی و کنترل بهتری در فرآروند پیاده‌سازی برنامه (به عنوان مثال در مراحل توسعه، آزمون، استقرار) نسبت به معماری خدمت‌گرا ارائه می‌کند[۷]. این تفاوت‌های اساسی چالش‌هایی را در آزمون راهبردهای خاص برنامه‌های مبتنی بر معماری میکروسرویس ایجاد می‌کند. هدف از ایجاد آزمون‌ها یافتن احتمال خطاهای پیاده‌سازی یا خطاهایی است که به کاهش کیفیت منجر می‌شوند و یا خطاهایی است که قابلیت استفاده یک برنامه یا سیستم را کاهش می‌دهند.

آزمون نرم‌افزار از مجموعه تصدیق‌کننده‌های پویا تشکیل شده است که ارزیابی می‌کنند، آیا یک سیستم، رفتار مورد انتظار را در مجموعه‌ی محدودی از موارد آزمون (که به طور مناسب از دامنه‌ی اجرای معمولاً نامحدود انتخاب شده‌اند) ارائه می‌دهد یا خیر. پویا در این تعریف بر این حقیقت تاکید دارد که آزمون بر روی مجموعه‌ی متغیر از ورودی‌های انتخاب‌شده انجام می‌شود، البته دقیق‌تر آن است که برخی اوقات حتی یک ورودی انتخاب‌شده نیز برای مشخص کردن یک آزمون کافی نیست و ممکن است برنامه‌های غیرقطعی و پیچیده به ازای یک ورودی یکسان با توجه به حالت درونیشان، رفتارهای متفاوتی در مقابل آزمون نشان دهند و پاسخ‌های متفاوتی بدهند؛ در نتیجه برای مشخص کردن یک آزمون علاوه بر ورودی به حالت درونی برنامه نیز نیاز است.[۸] آزمون‌ها مجموعه‌ای از فعالیت‌ها هستند که هدفشان یافتن خطاهای احتمالی در طراحی، پیاده‌سازی، کیفیت یا قابلیت استفاده یک برنامه یا سیستم است. تعداد سرویس‌ها، فرآیندهای ارتباطی، وابستگی‌ها، نمونه‌ها، ارتباطات شبکه و سایر متغیرها بر روش‌های آزمون برنامه‌ها با معماری میکروسرویس تأثیر می‌گذارند. این نوع از برنامه‌ها به دلیل ماهیت پیچیده و رفتار پویای خود، چالش‌های مهمی را برای آزمون ایجاد می‌کنند. برای رسیدن به هدفی که گفته شد، در طول فرآروند آزمون، باید رفتارهای همزمان میکروسرویس‌های مختلف و تعاملات بین آنها درک شود[۹].

---

[۷] Component

[۸] Enterprise Service Bus (ESB)

۱-۲ اهمیت موضوع

تشخیص عیوب و اشکالات یک نرم‌افزار در مرحله طراحی، یک گام مهم در مهندسی نرم‌افزار است. اهمیت این فرآیند از این واقعیت ناشی می‌شود که به بالا رفتن کیفیت عملکرد نرم افزار کمک می‌کند و باعث اطمینان بیشتر و کارآمدتر شدن آن در خدمت به کاربران می‌شود. با تشخیص فعالانه نقص‌ها در مرحله طراحی، توسعه‌دهندگان می‌توانند از مشکلات بی‌شماری پایین‌دستی که در غیر این صورت پس از استقرار نرم‌افزار و استفاده توسط کاربران نهایی ایجاد می‌شوند، جلوگیری کنند. این فرآیند به‌ویژه در پروژه‌های مقیاس بزرگ، که در آن‌ها حتی اشکالات یا نقص‌های جزئی می‌تواند فاجعه‌بار باشد و منجر به خسارات مالی قابل‌توجه و آسیب به اعتبار سازمان شود، اهمیت دارد.

مزایای مالی تشخیص نقص در مرحله طراحی قابل توجه است و به ویژه برای پروژه‌های بزرگ جذاب است. با شناسایی و رفع خطاها در مراحل اولیه طراحی، سازمان‌ها می‌توانند خطرات ناشی از خطاها و تاخیرهای پرهزینه را به حداقل برسانند، در منابع با ارزش صرفه جویی کنند و کارایی کلی پروژه را بهبود بخشند. این رویکرد همچنین با اجازه دادن به مدیران پروژه برای برآورد نیازهای منابع و تخصیص موثرتر منابع، فرآیند توسعه را بهبود می‌بخشد. در مجموع، تشخیص عیوب و اشکالات در مرحله طراحی برای توسعه و استقرار موفق نرم‌افزار ضروری است و می‌تواند تأثیر قابل‌توجهی بر بازگشت سرمایه برای سازمان‌ها داشته باشد.

چالش‌هایی بر سر راه آزمون برنامه‌ها با معماری پیچیده و رفتار پویای میکروسرویس‌ها وجود دارد، در پژوهش‌های پیشین سعی شده است که برای چالش‌های ایجاد شده در این مسیر روش‌ها و رویکردهای جدیدی ارائه شود، در این روش‌ها تلاش شده است تا با وجود مولفه‌هایی که به طور مستقل از هم قابل استقرار هستند و در عین حال روابط پیچیده‌ای دارند، درستی عملکرد کلی سامانه سنجیده شود. یکی از رویکردهای آزمون معرفی شده رویکرد مدل‌رانه است، پژوهشگران در این رویکرد سعی کرده‌اند با روش‌های مدل‌سازی و ارائه‌ی توصیف صوری از میکروسرویس‌ها صحت عملکرد و نحوه تعامل میکروسرویس‌ها راصحت‌سنجی کنند، با وجود روش‌های پیشنهادی جدید، آزمون سامانه‌ها با معماری میکروسرویس در حوزه صنعت همچنان با چالش‌های زیادی روبه‌رو است[۶].

۱-۳ اهداف پژوهش

هدف از پژوهش در این پایان‌نامه ارائه‌ی روشی مبتنی بر آزمون مدل‌ـ‌رانه برای آزمون برنامه‌ها و سیستم‌ها با معماری میکروسرویس بوده است. روش آزمونی که در این پایان‌نامه ارائه شده، موارد آزمون را طبق یک

معیار پوشش بر اساس منطق، به صورت خودکار استخراج می‌کند و همچنین با استفاده از یک ابزار آماده به کار برای تحلیل توصیفات صوری آن‌ها را در محیطی محدود بر روی توصیف صوری برنامه اجرا می‌کند.

برای این که بتوان موارد آزمون را بر اساس معماری برنامه به صورت خودکار، تولید کرد، ابتدا باید مدل‌های گردش کار برنامه که به زبان یاول[۱۰] تولید شده‌اند، به ابزاری که تولید کردیم، ورودی داده شوند. سپس این ابزار با پردازش آن‌ها، مدل‌های ورودی را با یک روش که توسعه داده‌شده‌ی روش پیشنهادی ریواده و همکاران است[۱۱]، به توصیفی صوری تبدیل می‌کند. توصیف صوری مدل‌ها، در قالب زبان الوی[۱۲] تولید می‌شوند. پس از ترجمه‌ی به زبان الوی موارد آزمون، از مکان‌هایی از برنامه که جریان کنترل برنامه را تعیین می‌کنند استخراج می‌شوند و سپس با استفاده از مسند[۹]های موجود در آن‌ها موارد آزمون بر اساس پوشش گزاره‌ی فعال بند محدود[۱۰] تولید می‌شوند.

ترجمه‌ی مدل‌ها از یاول به الوی، به صورت خودکار، در ابزار موجود برای طراحی مدل‌های گردش کاری در یاول، انجام می‌شوند؛ دیگر نوآوری‌ای که ارائه شده است، انجام تحلیل‌هایی بر روی ساختار میکروسرویس‌ها با استفاده از ترجمه‌ی انجام‌شده است. این تحلیل‌ها برای شناسایی حالت‌های غیرمجاز و نامطلوب به کمک تحلیل‌گر برنامه می‌آیند. پس از انجام تحلیل، نتایج در ابزار نمایش داده می‌شوند.

علاوه بر این‌ها، تولید موارد آزمون نیز طبق روشی که ارائه داده‌ایم به صورت خودکار و با تولید همه‌ی مقادیر ممکن از روی مسندهای توصیف صوری و بر اساس معیار پوشش گزاره‌ی فعال محدود ایجاد می‌شوند و در همان ابزار گفته شده به آزمون‌کننده نمایش داده می‌شوند. همچنین اجرای موارد آزمون تولیدشده بر روی توصیف صوری و نمایش نتایج آزمون از نوآوری‌ها است.

۱-۴ ساختار پایان‌نامه

ساختار پایان‌نامه در ادامه به شرح زیر است: فصل دوم شامل مفاهیم اولیه‌ای است که برای درک روش پیشنهادی و کارهای مرتبط با این پایان‌نامه نیاز هستند. در فصل سوم کارهای پیشین مرتبط با موضوع این پایان‌نامه را بررسی کردیم و سعی کردیم آن‌ها را بر اساس فراوانی موضوعات اصلی تحقیقی رایج در موضوعات پنج‌گانه دسته‌بندی کنیم. در فصل چهارم روش ارائه‌شده‌ی خود را شرح داده‌ایم.

در فصل پنجم بر روی یک مورد مطالعاتی که برگرفته شده از یک برنامه‌ی واقعی است روش ارائه‌شده

[۹] predicate
[۱۰] Restricted Active Clause Coverage (RACC)

را اعمال کردیم و در هر مرحله نتایج به دست‌آمده را بیان کردیم .در بخش پایانی نیز به ارزیابی روش از نظر کارایی از نظر تحلیل جهش و همچنین از نظر کارایی زمانی پرداختیم.

فصل ۲

مفاهیم اولیه

در این قسمت مفاهیم مورد استفاده در پایان نامه و همچنین چند اصطلاح رایج در مبحث آزمون نرم‌افزار نوشته شده است

۲-۱ مفاهیم مربوط به میکروسرویس

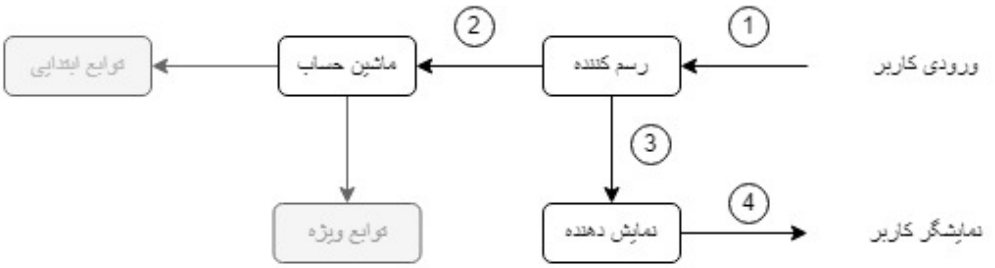
۲-۱-۱ میکروسرویس

میکروسرویس یک فرارونده[۱] منسجم و مستقل است که از طریق پیام در تعامل است. از دیدگاه فنی، میکروسرویس‌ها باید مولفه‌های مستقلی باشند که به صورت مفهومی‌به صورت مجزا مستقر شده و مجهز به ابزارهای اختصاصی ماندگاری حافظه (مانند پایگاه‌های داده) باشند. از آن‌جایی که تمام اجزای یک معماری میکروسرویس، میکروسرویس هستند، رفتار متمایزکننده‌ی آن از ترکیب و تعامل اجزای آن از طریق پیام‌ها ناشی می‌شود. تعاملات بین میکروسرویس‌ها اغلب از طریق مکانیسم‌های سبک، مانند API های منابع از نوع HTTP برقرار می‌شود[۱۳] [۱].

۲-۱-۲ میکروسرویس‌ها (معماری میکروسرویس)

سبک معماری میکروسرویس[۱۳] رویکردی برای توسعه یک برنامه کاربردی واحد به عنوان مجموعه ای از سرویس‌های کوچک است که هر کدام در فرآروند خاص خود اجرا می‌شوند. برای مثالی از معماری

[۱] Process



شکل ۲-۱: معماری میکروسرویس برای یک برنامه‌ی ماشین حساب[۱]

میکروسرویس، فرض می‌کنیم که می‌خواهیم عملکردی ارائه دهیم که نمودار یک تابع را رسم کند. وجود دو میکروسرویس را فرض می‌کنیم: ماشین حساب و نمایشگر. اولین مورد، میکروسرویس ماشین حساب است و دومی‌تصاویر را رنده می‌کند و نمایش می‌دهد. برای تحقق هدفمان، می‌توانیم یک میکروسرویس جدید به نام "رسم‌کننده[۲]" معرفی کنیم که ماشین حساب را برای محاسبه شکل نمودار تنظیم می‌کند[۳] و نمایشگر را برای ارائه شکل محاسبه‌شده فراخوانی می‌کند. در شکل ۲-۱، تصویری از معماری میکروسرویس برنامه‌ی ماشین حساب ذکرشده را نمایش داده‌ایم. توسعه‌دهندگان معماری فوق قادرند به طور جداگانه بر روی اجرای عملکردهای میکروسرویس اصلی، یعنی ماشین حساب و نمایش‌دهنده[۴] تمرکز کنند. در نهایت، آن‌ها می‌توانند رفتار برنامه‌ی توزیع شده را با رسم‌کننده پیاده‌سازی کنند که در مرحله‌ی ۱ تابع کاربر را می‌گیرد، در گام ۲ با ماشین حساب تعامل می‌کند تا یک نمایش نمادین از نمودار تابع را محاسبه کند، و در نهایت در گام ۳ از نمایش‌دهنده درخواست می‌کند تا نتیجه را به کاربر نشان دهد. در مرحله‌ی پایانی ۴. برای نشان دادن اینکه چگونه رویکرد میکروسرویس با ساخت بر روی معماری‌های میکروسرویسی که از قبل موجود هستند، مقیاس[۵] می‌شود؛ در شکل ۲-۱ ماشین حساب به گونه‌ای طراحی شده است که دو میکروسرویس اضافی (به رنگ خاکستری) را که توابع ابتدایی و ویژه ریاضی را اجرا می‌کنند، نیز تنظیم کند.

سبک معماری میکروسرویس هیچ انگاره[۶]ی برنامه نویسی خاصی را مورد حمایت قرار نمی‌دهد یا از هیچ انگاره‌ای منع نمی‌کند؛ بلکه یک دستورالعمل است برای شکستن اجزای یک برنامه کاربردی توزیع شده به موجودیت‌های مستقل که هر یک به یکی از نگرانی‌های آن برنامه می‌پردازد. این بدان معنی است که یک میکروسرویس، به شرطی که عملکردهای خود را از طریق ارسال پیام ارائه دهد، می‌تواند در داخل

[۲] Plotter
[۳] Orchestrate
[۴] Displayer
[۵] Scale
[۶] paradigm

برنامه با هر زبانی پیاده سازی شود[۱۳]. میکروسرویس‌ها ممکن است برای ارائه عملکردهای پیچیده‌تر و دقیق‌تر همکاری کنند. دو رویکرد برای ایجاد این همکاری وجود دارد - سبک ارکستراسیون[۷][۱۴] و سبک موزون[۸][۱۵]. ارکستراسیون نیاز به یک رهبر دارد. یک سرویس مرکزی که درخواست‌ها را به سرویس‌های دیگر ارسال می‌کند و با دریافت پاسخ‌ها بر فرآیند نظارت می‌کند. از سوی دیگر، سبک موزون هیچ مرکزیتی ندارد و از رویدادها و مکانیسم‌های انتشار[۹]/اشتراک[۱۰] برای ایجاد همکاری استفاده می‌کند. این دو مفهوم برای میکروسرویس‌ها جدید نیستند، بلکه از دنیای معماری بر محوریت سرویس به ارث رسیده‌اند. قبل از ظهور میکروسرویس‌ها و به طور خاص در آغاز ظهور معماری خدمت‌گرا، به دلیل سادگی استفاده و راه‌حل‌های با هزینه‌ی کمتر برای مدیریت پیچیدگی، عموماً ارکستراسیون محبوب‌تر بود و به طور گسترده‌تر استفاده می‌شد. با این حال، ارکستراسیون به وضوح منجر به جفت‌شدگی[۱۱] سرویس‌ها و توزیع نابرابر مسئولیت‌ها می‌شود، بنابراین برخی از سرویس‌ها نقش متمرکزتری نسبت به سایرین پیدا می‌کنند. فرهنگ تمرکززدایی و درجات بالای استقلال در میکروسرویس‌ها اتفاقا نشان‌دهنده‌ی سناریوی کاربردی ذاتی برای استفاده از سبک موزون برای دستیابی به همکاری است[۱].

۲-۲ یاول

بر اساس تجزیه و تحلیل دقیق سیستم‌های مدیریت گردش کار موجود و زبان‌های گردش کار، یک زبان گردش کار جدید به نام یاول (یک زبان دیگر گردش کار) توسط وندرآلست[۱۲] (استاد دانشگاه فناوری آیندهوون، هلند) و ترهافستید[۱۳] (استاد دانشگاه صنعتی کوئینزلند، استرالیا) در سال ۲۰۰۲ ایجاد شد. این زبان از یک سو بر شبکه‌های پتری، که یک نظریه همزمانی تثبیت شده با یک نمایش گرافیکی است، و از سوی دیگر بر روی الگوهای گردش کار معروف استوار است. الگوهای گردش کار یک معیار عمومی پذیرفته شده برای مناسب بودن یک زبان توصیف فرآیند است. شبکه‌های پتری[۱۴] می‌توانند تعداد کمی از الگوهای کنترل جریان شناسایی شده را جذب کنند، اما آنها از الگوهای چندگانه، الگوهای لغو و پیوند از نوع "یا"ی تعمیم‌یافته پشتیبانی نمی‌کنند. بنابراین یاول شبکه‌های پتری را با ساختارهای اختصاصی برای پشتیبانی این الگوها گسترش می‌دهد[۱۰].

یک مدل یاول از مجموعه ای از شبکه‌های یاول به شکل یک ساختار گراف ریشه‌دار تشکیل شده است.

---

[۷] Orchesteration
[۸] Choreography
[۹] Publish
[۱۰] Subscribe
[۱۱] Coupling
[۱۲] Wil van der Aalst
[۱۳] Arthur ter Hofstede
[۱۴] Nets Petri : یک زبان مدل‌سازی ریاضی برای توصیف سیستمهای توزیع‌شده.

هر شبکه یاول از یک سری وظایف تشکیل شده است. وظایف و شرایط در شبکه‌های یاول نقشی مشابه انتقال‌ها و مکان‌ها در شبکه‌های پتری ایفا می‌کنند. هر شبکه یاول دارای یک شرط ورودی و یک شرط خروجی منحصر به فرد است، که نقطه شروع و پایان برای یک نمونه فرآروند است.

وظایف در یک شبکه یاول می‌توانند رفتارهای پیوند و انشعاب خاص خودشان را داشته باشند. ساختارهای پیوند و انشعاب پشتیبانی شده عبارتند از پیوند از نوع "و"، پیوند از نوع "یا"ی انحصاری، پیوند از نوع "یا"، انشعاب از نوع "و"، انشعاب از نوع "یا" و انشعاب از نوع "یا"ی انحصاری. عملکرد هر یک از پیوند و انشعاب‌ها در یاول به شرح زیر است:

- پیوند از نوع "و" — شاخه‌ای که به دنبال پیوند از نوع "و" است کنترل را زمانی دریافت می‌کند که تمام شاخه‌های ورودی به پیوند از نوع "و" در یک مورد مشخص فعال شده باشند.
- پیوند از نوع "یا" - شاخه ای که پس از پیوند از نوع "یا" قرار می‌گیرد کنترل را در شرایط زیر دریافت می‌کند: زمانی که (۱) همه‌ی شاخه‌های ورودی فعال شده باشند یا (۲) همه‌ی شاخه‌های ورودی که هنوز فعال نشده‌اند، امکان فعال شدن در زمان آینده را نداشته باشند.
- پیوند از نوع "یا"ی انحصاری - شاخه‌ای که پس از پیوند از نوع "یا"ی اختصاصی قرار می‌گیرد، کنترل را زمانی دریافت می‌کند که یکی از شاخه‌های ورودی به پیوند از نوع "یا"ی انحصاری فعال شده باشد. در زبان‌های مختلف رویکردهای مختلف معنایی متفاوت (اغلب فقط شهودی) را به این نوع پیوند اختصاص می‌دهند، اگرچه همه‌ی آن‌ها، این موضوع مشترک را دارند که همگام‌سازی فقط برای شاخه‌های فعال انجام می‌شود. به بیان دقیق‌تر در یاول نگاه کلی به معنای پیوند از نوع "یا"، در حضور ویژگی‌های الغا و بدون محدودیت‌های ساختاری، با استفاده از فرمالیسم شبکه بازنشانی است. شبکه‌های بازنشانی شبکه‌های پتری با قوس‌های بازنشانی هستند. یک قوس بازنشانی زمانی که انتقال آن فعال می‌شود، همه توکن‌ها را از یک مکان حذف می‌کند.
- انشعاب از نوع "و" — زمانی که شاخه‌ی ورودی به انشعاب از نوع "و" فعال می‌شود، کنترل به همه شاخه‌های خروجی انشعاب از نوع "و" منتقل می‌شود.
- انشعاب از نوع "یا"— زمانی که شاخه‌ی ورودی به انشعاب از نوع "یا" فعال می‌شود، کنترل به یک یا چند شاخه‌ی خروجی انشعاب از نوع "یا" بر اساس ارزیابی شرایط مرتبط با هر یک از شاخه‌ها منتقل می‌شود.
- انشعاب از نوع "یا"ی انحصاری — زمانی که شاخه ورودی به انشعاب از نوع "یا"ی انحصاری فعال می‌شود، کنترل دقیقاً به یکی از شاخه‌های خروجی انشعاب از نوع "یا"ی انحصاری بر اساس ارزیابی شرایط مرتبط با هر یک از شاخه‌ها منتقل می‌شود.

یاول از مفهوم منطقه‌ی لغو پشتیبانی می‌کند، که شامل گروهی از وظایف در یک شبکه یاول است. منطقه‌ی لغو به یک وظیفه‌ی خاص در همان شبکه یاول مرتبط است. در زمان اجرا، زمانی که نمونه‌ای از وظیفه‌ای که منطقه لغو به آن متصل است، اجرا را کامل می‌کند، تمام وظایف موجود در منطقه لغو مرتبط که در حال حاضر برای همان مورد در حال اجرا هستند، لغو می‌شوند.

۲-۲-۱ ساختار ذخیره‌سازی مدل‌ها در زبان یاول

مدل‌های تولیدشده به زبان یاول در قالب استاندارد XML ذخیره می‌شوند؛ استفاده از فرمت استاندارد XML برای ذخیره مدل‌های یاول، امکان ذخیره و تبادل داده‌ها بین سیستم‌های مختلف را در یک فرمت قابل استفاده و قابل خواندن فراهم می‌کند. با استفاده از فرمت استاندارد، امکان تبادل ساده و راحت مدل‌های یاول بدون توجه به مکان و پلتفرم نرم‌افزاری آن‌ها، بین افراد و تیم‌های مختلف وجود دارد.

۲-۳ الوی

الوی یک زبان توصیف برای بیان محدودیت‌ها و رفتار ساختاری پیچیده در یک سیستم نرم افزاری است و همچنین یک ابزار مدل سازی ساختاری ساده بر اساس منطق مرتبه اول ارائه می‌دهد[۱۲].

الوی از موفقیت‌ها و محدودیت‌های چک کننده‌ی مدل‌ها الهام گرفته شده است و نوع جدیدی از زبان طراحی و تحلیل ارائه می‌دهد که با سه نوآوری امکان پذیر شده است:

- اولین آن‌ها منطق رابطه‌ای است، الوی از این منطق برای توصیف طراحی‌ها و ویژگی‌ها استفاده می‌کند. منطق رابطه‌ای[۱۶]، کمیت‌کننده[۱۵] های منطق مرتبه اول را با عملگرهای نظریه‌ی مجموعه‌ها و حساب رابطه‌ای می‌آمیزد. ایده‌ی مدل‌سازی طرح‌های نرم‌افزاری با مجموعه‌ها و روابط در زبان Z مطرح شده بود[۱۷] الوی از بخش زیادی از قدرت Z استفاده کرده است، در حالی که منطق موجود در آن را ساده‌تر می‌کند تا Z را قابل استفاده‌تر کند. در این راستا الوی فقط ساختارهای مرتبه اول را استفاده می‌کند. الوی همچنین تحت تأثیر زبان‌های مدل سازی مانند UML است.
- نوآوری دوم در الوی استفاده از تجزیه و تحلیل دامنه کوچک است. حتی منطق مرتبه اول ساده (بدون عملگرهای رابطه‌ای) تصمیم‌پذیر نیست. این به این معنی است که هیچ الگوریتمی نمی‌تواند وجود داشته باشد که بتواند طراحی نرم افزاری که به طور کامل به زبانی مانند الوی نوشته شده است را تجزیه و تحلیل کند. می‌توان زبان را تصمیم‌پذیر کرد، اما این امر قدرت بیان آن را ناقص می‌کند

[۱۵] quantifier

و باعث می‌شود که نتواند حتی ابتدایی‌ترین ویژگی‌های ساختارها را بیان کند. برای داشتن تجزیه و تحلیل چنین برنامه‌هایی یک راه این است که خودکارسازی تجزیه و تحلیل را حذف کرد و از کاربر برای این کار کمک گرفت. اما این کار مزایای یک ابزار تجزیه و تحلیل را ضایع می‌کند. در این صورت تجزیه و تحلیل دیگر پاداشی برای ساخت یک مدل طراحی نیست، بلکه یک سرمایه گذاری اضافی بزرگ و فراتر از مدل‌سازی است.

یک راه دیگر نیز محدود کردن تحلیل است که پیش از الوی از راه‌های انتزاع و شبیه‌سازی استفاده می‌شد. انتزاع معمولا آنقدر از جزئیات می‌کاست که به ایجاد مثبت نادرست منجر می‌شد که قابل تفسیر هم نبود و شبیه‌سازی نیز آنقدر بخش کوچکی از فضای حالت را پوشش می‌داد که نقص‌های ظریف از تشخیص فرار می‌کردند. اما یاول یک رویکرد جدید ارائه کرد: اجرای تمام آزمون‌های کوچک. طراح، محدوده‌ای را مشخص می‌کند که هر یک از انواع را در توصیف محدود می‌کند. این نوآوری بر پایه‌ی فرضیه دامنه کوچک[۱۸][۱۶] است، که ادعا می‌کند اکثر خطاها را می‌توان با مثال‌های نقض کوچک مشخص کرد.

- سومین نوآوری الوی در ترجمه‌ی مدل‌ها به یک مساله‌ی صدق‌پذیری است. حتی با محدوده‌های کوچک، فضای حالت یک مدلِ الوی، بسیار بزرگ است. حالت شامل مجموعه ای از متغیرها است که مقادیر آنها، روابط بین انواع هستند. فقط یک رابطه‌ی دودویی در یک محدوده‌ی پنج تایی دارای ۵ × ۵ = ۲۵ یال ممکن است، و بنابراین می‌تواند $۲^{۲۵}$ مقدار ممکن داشته باشد. یک طرح بسیار کوچک ممکن است دارای پنج رابطه باشد که به $(۲^{۲۵})^{۵}$ حالت ممکن - حدود $۱۰^{۳۷}$ حالت می‌شود. حتی با بررسی یک میلیارد مورد در ثانیه، چنین تحلیلی چندین برابر عمر جهان است. بنابراین الوی یک جستجوی صریح انجام نمی‌دهد، بلکه در عوض مسئله‌ی طراحی را به یک مسئله رضایت‌پذیری تبدیل می‌کند که متغیرهای آن روابط نیستند بلکه بیت‌های ساده هستند. با چرخش بیت‌ها به صورت جداگانه، یک حل‌کننده رضایت‌پذیری معمولاً می‌تواند راه‌حلی را بیابد (اگر وجود داشته باشد) یا تنها با بررسی بخش کوچکی از فضا نشان دهد که هیچ راه‌حلی وجود ندارد. ابزار تجزیه و تحلیل الوی اساساً یک کامپایلر برای مساله‌ی رضایت‌پذیری است که به آن اجازه می‌دهد از آخرین پیشرفت‌ها در حل کننده‌های مساله‌ی رضایت‌پذیری بهره‌برداری کند. موفقیت حل کننده‌های مساله‌ی رضایت‌پذیری یک داستان قابل توجه در علم کامپیوتر بوده است - نظریه‌پردازان نشان داده بودند که مساله‌ی رضایت‌پذیری ذاتاً غیر قابل حل است، اما معلوم شد که بیشتر مواردی که در عمل پدیدار می‌شوند، می‌توانند به طور موثر حل شوند. بنابراین مساله‌ی رضایت‌پذیری از یک مسئله‌ی حل‌نشدنی کهن الگو که برای نشان دادن غیرممکن بودن مسائل دیگر استفاده می‌شد،

۱۶

به یک مساله‌ی حل‌شدنی تبدیل شد که سایر مسائل را می‌توان به آن ترجمه کرد. الوی همچنین از روش‌های مختلفی برای کاهش مساله قبل از حل استفاده می‌کند، به ویژه افزودن محدودیت‌های شکست تقارن که حل‌کننده مساله‌ی رضایت‌پذیری را از در نظر گرفتن موارد مشابه با یکدیگر باز می‌دارد[۱۹].

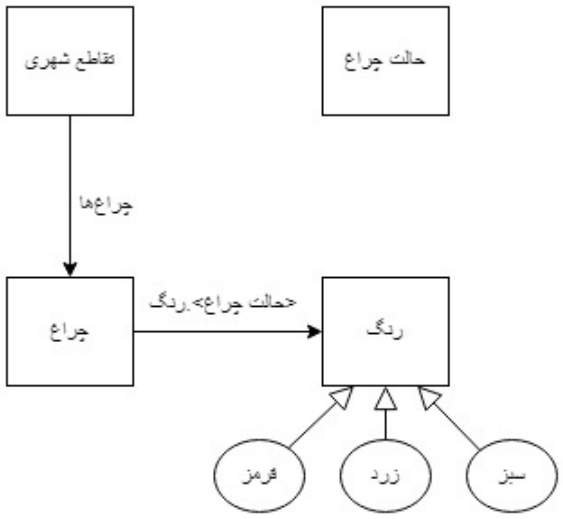
زبان الوی مانند زبان‌های توصیف نرم‌افزار دیگر دارای ابزارهایی برای نظام‌دهی مدل‌ها، ساخت مدل‌های بزرگ‌تر از مدل‌های کوچک‌تر و ساخت مولفه‌ها با قابلیت استفاده‌ی مجدد است، همچنین جزئیاتی در نحو آن وجود دارد تا الوی زبانی قابل استفاده در عمل باشد. مدل‌ها در الوی به صورتی مشخص و با نحو الوی توصیف می‌شوند، اجزایی از الوی که در این پایان‌نامه مورد استفاده قرار گرفته‌اند، را در ادامه آورده‌ایم و در انتها نیز در یک مثال از آن‌ها برای توصیف یک نرم‌افزار ساده استفاده کرده‌ایم. امضا: یک امضا مجموعه ای از اتم‌ها را معرفی می‌کند. امضا `sig A{}` مجموعه‌ای به نام A را معرفی می‌کند. امضا در واقع چیزی بیش از یک مجموعه است، زیرا می‌تواند شامل اعلام روابط باشد و می‌تواند به طور ضمنی نوع جدیدی را معرفی کند. یک مجموعه را می‌توان به عنوان زیرمجموعه‌ای دیگر معرفی کرد. از آن جایی که یک امضا در واقع یک مجموعه نیز هست در نتیجه `sig A1 extensions A {}` مجموعه‌ای به نام A۱ را معرفی می‌کند که زیرمجموعه A است. امضای A۱ توسعه‌داده‌شده یا زیرامضای امضای A است.

در الوی مفروضات در "حقیقت"ها قرار می‌گیرند. مفاهیمی که باید بررسی شوند در اظهارها گفته می‌شوند. محدودیت‌هایی که در زمینه‌های مختلف لازم‌اند. به عنوان مسند تعریف می‌شوند. با استفاده از یک مثال در زیر نشان می‌دهیم که چگونه می‌توان با استفاده از ابزارها و نحو الوی، یک مدل ساده را به صورت سازگار توصیف کرد.

شکل ۲-۲، یک سیستم چراغ راهنمایی را نشان می‌دهد. در این سیستم می‌خواهیم که در هر تقاطع، در هر حالتی، بعضی از چراغ‌ها قرمز را نشان دهند. برای این سیستم در الوی قطعه توصیف زیر را می‌توان نوشت:

۲-۳-۱ تحلیل‌گر الوی

تحلیل‌گر الوی ابزاری است که در واقع توصیفات نوشته شده به زبان الوی را بررسی می‌کند. تحلیلگر مدل را به فرمول صدق‌پذیری برای حل تبدیل می‌کند. این تحلیل که در تحلیل‌گر الوی گنجانده شده است، بر پیشرفت‌های اخیر در فناوری صدق‌پذیری متکی است. تحلیل‌گر الوی محدودیت‌هایی را که باید حل شوند از الوی به محدودیت‌های بولی ترجمه می‌کند، که به یک حل‌کننده‌ی صدق‌پذیری آماده به کار داده می‌شود. همانطور که حل کننده‌های مساله‌ی صدق‌پذیری سریع‌تر می‌شوند، تجزیه و تحلیل الوی نیز سریع‌تر می‌شود



شکل ۲-۲: سیستم چراغ راهنمایی alloybook

```
sig LightState {color: Light -> one Color}

sig Light {}

abstract sig Color {}

one sig Red, Yellow, Green extends Color {}

sig Junction {lights: set Light}

fact {
      all s: LightState, j: Junction | some s.color.Red & j.lights
}
```

و به مسائل بزرگ‌تر می‌رسد. با استفاده از بهترین حل کننده‌های امروزی، تحلیلگر می‌تواند فضاهایی را که چند صد بیت عرض دارند بررسی کند.

تحلیلی که تحلیل‌گر الوی انجام می‌دهد نوعی حل محدودیت است. اما نقطه‌ی قوت تحلیل‌گر الوی در شبیه‌سازی سیستم مدل‌شده با الوی است. شبیه‌سازی شامل یافتن نمونه‌هایی از حالت‌ها یا اجراهایی است که یک ویژگی معین را برآورده می‌کند. همچنین می‌توان با تحلیل‌گر الوی مثال نقض برای حالت‌های خاص تعریف‌شده با الوی یافت. در این جا مقصود از مثال نقض نمونه‌ای است که یک ویژگی معین را نقض می‌کند. این ویژگی معین با اظهارها بیان می‌شوند. جستجوی نمونه‌ها در فضایی انجام می‌شود که ابعاد آن توسط کاربر در یک محدوده مشخص شده است، که یک کران به تعداد اشیاء از هر نوع اختصاص می‌دهد. حتی یک محدوده‌ی کوچک، یک فضای بزرگ را تعریف می‌کند، بنابراین استفاده از تحلیل‌گر الوی اغلب برای یافتن خطاها در فضاهای کوچک مناسب است.

۲-۴ آزمون

با رشد صنعت نرم‌افزار، آزمون نرم‌افزار نیز به عنوان فعالیتی کلیدی برای تضمین کیفیت پیشرفت کرده است، اگرچه عوامل بسیاری، از جمله، طراحی دقیق و مدیریت فرآیند کامل، بر مهندسی نرم افزار قابل اتکا تأثیر می‌گذارند، اما آزمون، اولین روشی است که صنعت، از آن برای ارزیابی نرم افزار در طول توسعه بهره می‌برد. بیزر[۱۷] هدف از آزمون محصول نرم‌افزاری را بر حسب «سطوح بلوغ فرآیند آزمون» یک سازمان بیان می‌کند، به طوری که این سطوح با اهداف آزمون‌کنندگان از آزمون تعیین می‌شوند[۲۰].

از آن‌جایی که واژه‌ی "درستی" در محصولات مهندسی واژه‌ای مبهم است و نمی‌توان منظور از درستی محصول را به صورت دقیق بیان کرد. مهندسان نرم‌افزار خردمند به جای جستجوی «درستی»، سعی می‌کنند «رفتار» نرم‌افزار را ارزیابی کنند تا تصمیم بگیرند که آیا این رفتار با در نظر گرفتن تعداد زیادی از عوامل از جمله قابلیت اطمینان، ایمنی، قابلیت نگهداری، امنیت و کارایی قابل قبول است یا خیر. بدیهی است که این بسیار پیچیده‌تر از میل ساده لوحانه برای نشان دادن درستی نرم‌افزار است. مهندسان نرم‌افزار برای مقابله با چنین پیچیدگی طاقت فرسایی از "بالا بردن سطح انتزاع" استفاده می‌کنند.

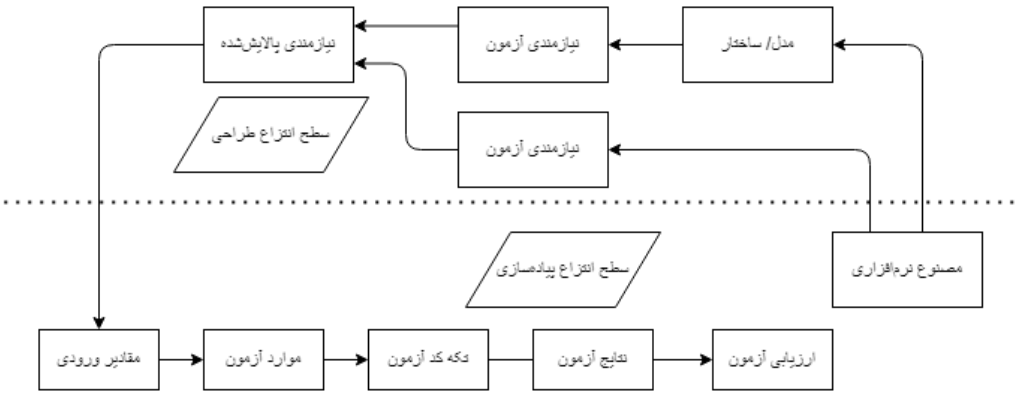
کار توسعه آزمون‌ها را می‌توان به چهار بخش مجزا تقسیم کرد: طراحی آزمون، خودکارسازی آزمون، اجرای آزمون و ارزیابی آزمون.

۱. طراحی آزمون فرآروند طراحی مقادیر ورودی است که به طور موثر نرم‌افزار را آزمایش می‌کند. این ریاضیاتی‌ترین و از نظر فنی چالش برانگیزترین بخش آزمون است. که می‌تواند به روش‌های مبتنی بر قاعده یا روش‌های مبتنی بر انسان انجام شود. استفاده از روش‌های مبتنی بر قاعده نیاز به دانش ریاضیات گسسته، برنامه‌نویسی و البته آزمون دارد. در حالی که روش‌های مبتنی بر انسان نیاز به دانش دامنه‌ی کاربرد نرم‌افزار مورد آزمون، واسط‌های کاربری و خود آزمون دارد.

۲. خودکارسازی آزمون، فرآروند جاسازی مقادیر آزمون در برنامه‌ی اجرا شونده است. جایی که مقادیر آزمون همان خروجی نهایی مرحله‌ی طراحی آزمون است.

۳. اجرای آزمون فرآروند اجرای آزمون‌ها بر روی نرم‌افزار و ثبت نتایج است. این مرحله، نیاز به دانش ابتدایی نرم‌افزار برای انجام دارد. امروزه بیشتر سازمان‌ها برای رسیدن به خودکارسازی صد در صد آزمون‌ها تلاش می‌کنند، این هدف اجرای آزمون را به طرز قابل ملاحظه‌ای ساده می‌کند. فرآروند طراحی آزمون مدل‌رانه، آزمون را به یک مجموعه کارهای کوچک تقسیم می‌کند که تولید آزمون را ساده می‌کند. سپس طراحان آزمون کار خود را به صورت جداگانه و فارغ از جزئیات نرم‌افزار یا

[۱۷] Beizer



شکل ۲-۳: آزمون مدل‌ـ‌رانه[۲]

مصنوعات طراحی، اتوماسیون آزمون و اجرای آزمون در سطح بالاتری از انتزاع پیش می‌برند و با استفاده از ساختارهای مهندسی ریاضیاتی مقادیر آزمون را طراحی می‌کنند.

۴. ارزیابی آزمون فرآروند ارزیابی نتایج آزمون و گزارش‌دهی به توسعه‌دهندگان است. انجام این مرحله نیاز به مهارت‌هایی مشابه مهارت‌های مورد نیاز برای طراحی آزمون با روش‌های مبتنی بر انسان دارد. ارزیابی آزمون اغلب با بیان خروجی‌های مورد انتظار در قالب اظهارها انجام می‌شود[۲].

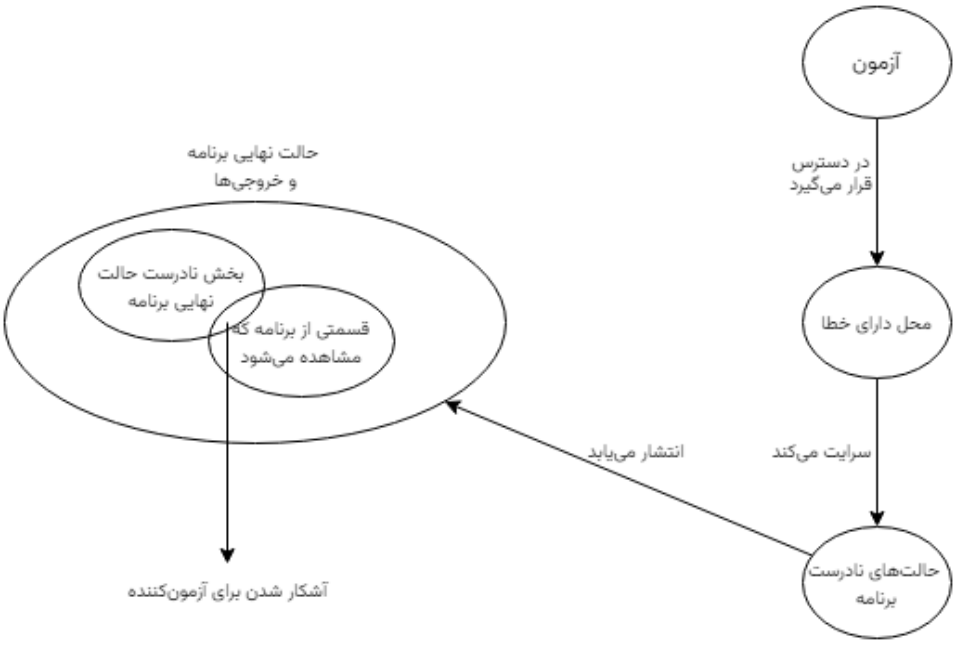
آزمون مدل‌ـ‌رانه

در طراحی آزمون مدل‌ـ‌رانه ما آزمون‌های خود را از ساختارهای انتزاعی که بسیار شبیه به مدل‌ها هستند استخراج می‌کنیم. این ساختارهای انتزاعی می‌توانند نمودارهای وضعیت در UML، شبکه‌های پتری، گراف‌ها یا توصیفات صوری باشند.

طراحی آزمون مدل‌ـ‌رانه به طراحان آزمون اجازه می‌دهد «سطح انتزاع خود را بالا ببرند» به طوری که زیر مجموعه‌ی کوچکی از آزمون‌کنندگان، جنبه‌های ریاضی طراحی و توسعه آزمون‌ها را انجام می‌دهند. سپس آزمون‌کنندگان و برنامه‌نویسان دیگر می‌توانند بخش‌های خود که شامل یافتن مقادیر آزمون، خودکار کردن آزمون‌ها، اجرای آزمون‌ها و ارزیابی آن‌ها، می‌شود را انجام دهند. این مشابه طراحی ساختمانی است، جایی که یک مهندس طرحی را ایجاد می‌کند که توسط بسیاری از متخصصین و کارگران دنبال می‌شود.

فرآیند طراحی آزمون مدل‌ـ‌رانه به صراحت از تقسیم کار پشتیبانی می‌کند. فرآروند این نوع از طراحی در شکل زیر نشان داده شده است، فعالیت‌های طراحی آزمون را در بالای خط و سایر فعالیت‌های آزمون را در شکل ۲-۳، می‌بینیم.

مدل خطا/شکست



شکل ۲-۴: مدل خطا/شکست و شرایط RIPR[۲]

تفاوتی اساسی بین عمل اشکال‌زدایی و عمل آزمون وجود دارد. در اشکال‌زدایی برای یک شکست موجود به دنبال علت هستیم در حالی که در فرآروند آزمون به دنبال کمینه کردن مخاطره‌ی استفاده از نرم‌افزار هستیم. مسئله اصلی این است که برای یک خطای معین، که در اشکال‌زدایی به ما داده شده‌است، همه ورودی‌ها باعث ایجاد خطا در ایجاد خروجی نادرست (شکست) نمی‌شوند. همچنین، ربط دادن یک شکست به خطایی که باعث آن شده است، اغلب بسیار دشوار است. تجزیه و تحلیل این ایده‌ها منجر به مدل خطا/شکست می‌شود که بیان می‌کند چهار شرط برای مشاهده‌ی شکست لازم است.

شکل ۲-۴ شرایط را نشان می‌دهد. ابتدا، یک آزمون باید به مکان یا مکان‌هایی در برنامه برسد که حاوی خطا (قابل دسترس بودن) هستند. دوما، پس از اجرای برنامه در آن مکان، حالت برنامه باید نادرست باشد (سرایت). سوما، حالت آلوده باید در بقیه مراحل اجرا منتشر شود و باعث شود برخی خروجی‌ها یا وضعیت نهایی برنامه نادرست شوند.(انتشار). در نهایت، آزمون‌کننده باید بخشی از قسمت نادرست حالت نهایی برنامه (قابل آشکار شدن) را مشاهده کند. اگر آزمون‌کننده فقط قسمت‌هایی از حالت نهایی برنامه را ببیند که صحیح هستند، خرابی آشکار نمی‌شود. مدل خطا/شکست در شکل ۲-۴ مشخص شده است.

مورد آزمون

یک مورد آزمون از مقادیر آزمون، مقادیر پیشوند، مقادیر پسوند و نتایج مورد انتظار لازم برای اجرای

کامل و ارزیابی نرم افزار تحت آزمایش تشکیل شده است.

مجموعه آزمون

مجموعه آزمون مجموعه ای از موارد آزمون است.

نیازمندی آزمون

یک نیازمندی آزمون عنصر خاصی از یک مصنوع نرم‌افزاری است که یک مورد آزمون باید آن را برآورده کند یا پوشش دهد.

معیار پوشش

تعداد ورودی‌های بالقوه برای اکثر برنامه‌ها به قدری زیاد است که می‌توان عملا آن را بی‌نهایت دانست زیرا که نمی‌توان به صراحت ورودی‌ها را برشمرد. اینجاست که معیارهای پوشش رسمی(یا صوری بهتره) وارد می‌شوند. از آنجایی که نمی‌توانیم با همه‌ی ورودی‌ها برنامه را بیازماییم، از معیارهای پوشش برای تصمیم‌گیری درباره‌ی ورودی‌های آزمون استفاده می‌کنیم. منطق پشت معیارهای پوشش این است که آنها فضای ورودی را برای به حداکثر رساندن تعداد خطاهای یافت‌شده در هر مورد آزمون تقسیم می‌کنند. مزیت دیگر برای استفاده از معیارهای پوشش در عمل این است که آن‌ها قوانین مفیدی را برای زمان توقف فرآروند آزمون ارائه می‌کنند.

تعریف رسمی معیار پوشش، آن را یک قاعده یا مجموعه قوانینی معرفی می‌کند که الزامات آزمون را بر یک مجموعه آزمون تحمیل می‌کند. معیارهای رایج که برخی از آن‌ها مطابق با مدل خطا/شکست تعریف شده‌اند و استفاده می‌شوند عبارتند از پارتیشن‌بندی فضای ورودی که طراحی تست را به روشی مستقل از مدل خطا/شکست انجام می‌دهد زیرا که فقط از فضای ورودی نرم‌افزار تحت آزمون استفاده می‌کند؛ معیارهای پوشش گراف که قابل دسترس بودن را تضمین می‌کند، استفاده از عبارات منطقی برای تولید آزمون که سرایت را تضمین می‌کند و تجزیه و تحلیل جهش که انتشار را تضمین می‌کند.

پوشش

با توجه به مجموعه ای از نیازمندی‌های آزمون TR برای یک معیار پوشش C، یک مجموعه آزمون T، C را برآورده می‌کند اگر و تنها اگر برای هر نیازمندی آزمون tr در TR، حداقل یک آزمون t در T وجود داشته باشد به طوری که t را ارضا کند.

معیارهای پوشش بر اساس منطق

معیارهای پوشش بر اساس منطق از عبارات منطقی برای تعریف معیارها و طراحی آزمون‌ها استفاده می‌کند. این دسته از معیارها به پیشرفت ما در مدل خطا/شکست کمک می‌کنند و اطمینان حاصل می‌کنند

که آزمون‌ها نه تنها به مکان‌های خاصی می‌رسند، بلکه حالت داخلی برنامه نیز متاثر از انتساب ترکیب‌های مختلفی از درستی به عبارات، آلوده می‌شود. از گزاره‌ها و مسندها برای معرفی انواع معیارهای پوشش استفاده می‌شود. فرض کنید P مجموعه‌ای از مسندها و C مجموعه‌ای از گزاره‌ها در مسندهای P باشد. برای هر مسند $p \in P$، C_p گزاره‌های در p باشد به طوری که $C_p = \{c|c \in p\}$ مجموعه‌ی C اجتماع گزاره‌های موجود در هر یک از مسندها در P خواهد بود، به عبارتی $C = \cup_{p \in P} C_{p^\wedge}$.

پوشش گزاره

برای هر $c \in C$ TR شامل دو نیازمندی است: اطلاق c به "درست" و اطلاق c به نادرست. گزاره‌ی $a \vee b$ را در نظر بگیرید. گزاره‌های C عبارتند از $\{a, b\}$ چهار ورودی آزمون که ترکیبی از مقادیر منطقی برای گزاره‌ها را برمی‌شمارد در جدول صحت زیر آمده، مشخص است.

	a	b	a ∨ b
1	T	T	T
2	T	F	T
3	F	T	T
4	F	F	F

مجموعه تست $T_{۲۳} = \{۲, ۳\}$ پوشش گزاره را برآورده می‌کند.

با توجه به تعاریف بالا می‌توان پوشش‌های زیر را بیان کرد

پوشش گزاره‌ی فعال

برای تعریف پوشش گزاره‌ی فعال ابتدا باید بدانیم گزاره‌ی اصلی یا تعیین‌کننده چیست. با در نظر گرفتن یک گزاره‌ی اصلی c_i در مسند p، می‌گوییم که اگر جملات فرعی $c_j \in pj \neq i$ مقادیری داشته باشند، p c_i را تعیین می‌کند به طوری که تغییر مقدار صدق c_i ، ارزش p را تغییر می‌دهد. با توجه به ادعای بالا می‌توان پوشش گزاره‌ی اصلی را این‌گونه تعریف کرد:

برای هر $p \in P$ و هر گزاره‌ی اصلی $c_i \in C_p$ ، گزاره‌های فرعی $c_j, j \neq i$ را طوری انتخاب کنید تا c_i تعیین کننده‌ی p باشد. نیازمندی آزمون در این حالت برای هر c_i دو شرط دارد: c_i به "درست" و c_i به "نادرست" مقداردهی .شود

برای مثال، برای $a \vee b$ ، در مجموع چهار نیازمندی در مجموعه نیازمندی‌های آزمون وجود دارد، دو مورد برای گزاره‌ی a و دو مورد برای گزاره‌ی b. برای گزاره‌ی a، a ارزش p را تعیین می‌کند اگر و فقط اگر b نادرست باشد. بنابراین ما دو نیازمندی آزمون = {(a = true b = false) (a = false b

{(false) را داریم. برای بند b، b ارزش کدp را تعیین می‌کند اگر و فقط اگر a نادرست باشد. بنابراین ما دو نیازمندی آزمون {(a = false b = false) (a = false b = true)} را داریم. این در جدول صحت جزئی زیر خلاصه شده است (مقادیر گزاره‌های اصلی به صورت پررنگ هستند).

	a	b
$c_i = a$	**T** **F**	f f
$c_i = b$	f f	**T** **F**

پوشش گزاره‌ی فعال محدود

برای هر $p \in P$ و هر گزاره‌ی اصلی $c_i \in C_p$ ، گزاره‌های فرعی $c_j, j \neq i$ را طوری انتخاب کنید که c_i ارزش p را تعیین کند. مجموعه نیازمندی‌های آزمون برای هر c_i دو شرط دارد: c_i به "درست" و c_i به "نادرست" مقداردهی شود. در هر دوی این حالت‌ها مقادیر انتخاب شده برای جملات فرعی c_j باید یکسان باشد.

برای مثال $p = a \vee (b \wedge c)$، را در نظر بگیرید برای اینکه a ارزش p را تعیین کند، عبارت $b \vee c$ باید "درست" باشد. این را می‌توان به سه طریق به دست آورد: b "درست" و c "نادرست" باشد، b "نادرست" و c "درست" باشد، و یا b و c هر دو "درست" باشند. بنابراین، می‌توان پوشش گزاره‌ی فعال محدود را با توجه به گزاره‌ی a را به کمک جدول زیر مشخص کرد.

در جدول درستی زیر ۶ حالتی را که گزاره‌ی a تعیین‌کننده‌ی ارزش مسند p است را آورده‌ایم طبق تعریف پوشش گزاره‌ی فعال محدود، میتوان این پوشش را با توجه به بند a این‌گونه ارضا کرد. ردیف ۲ را می‌توان با ردیف ۶، ردیف ۳ با ردیف ۷، یا ردیف ۱ با ردیف ۵ جفت کرد. بنابراین، تنها سه روش می‌توانند پوشش گزاره‌ی فعال محدود را برآورده کنند.

آزمون جهش

در همه‌ی برنامه‌ها می‌گوییم رشته‌های ورودی اگر به زبانی باشد که یک دستور زبان[۱۸] آن را مشخص کرده است معتبر است و در غیر این صورت نامعتبر است. به عنوان مثال، کاملاً معمول است که از یک

[۱۸] grammer

	a	b	c	a ∧ (b ∨ c)
1	T	T	T	T
2	T	T	F	T
3	T	F	T	T
5	F	T	T	F
6	F	T	F	F
7	F	F	T	F

جدول ۲-۱: جهش‌یافته‌های نمونه

مسند شرطی اصلی	نمونه مسند جهش‌یافته	عملگر جهش
$a < b$	$a \geqslant b$	ROR
$(a > b)\,\&\&\,(a > c)$	$(a > b)\,\|\|\,(a > c)$	COR

برنامه انتظار داشته باشیم ورودی‌های نادرست را رد کند. و این ویژگی باید صراحتا آزمون شود، زیرا برای توسعه‌دهندگان آسان است که آن را فراموش کنند یا اشتباه کنند. بنابراین، تولید رشته‌های نامعتبر از دستور زبان اغلب برای ورودی آزمون‌ها مفید است. همچنین در آزمون، استفاده از رشته‌هایی که معتبر هستند اما مشتق متفاوتی از یک رشته‌ی از قبل موجود هستند، مفید است. هر دو نوع این رشته‌ها جهش‌یافته نامیده می‌شوند. تولید رشته‌های ورودی نامعتبر را می‌توان با جهش[۱۹] دستور زبان، سپس تولید رشته[۲۰]ها، یا با جهش مقادیر در طول یک مشتق انجام داد.

جهش همیشه بر اساس مجموعه ای از "عملگرهای جهش" است که با توجه به یک رشته‌ی "پایه" تعریف می‌شوند. رشته‌ی پایه هر رشته‌ای در گرامر می‌تواند باشد و عملگر جهش قاعده‌ای است که تغییرات نحوی رشته‌های تولید شده از یک دستور زبان را مشخص می‌کند. همچنین یک جهش‌یافته نیز نتیجه‌ی اعمال یک عملگر جهش است. عملگرهای جهش معمولاً برای رشته‌های‌پایه اعمال می‌شوند، اما می‌توانند در دستور زبان یا به صورت پویا در طول اشتقاق نیز اعمال شوند[۲].

نمونه ی این عملگرها جایگزینی عملگرهای ریاضی یا رابطه‌ای، تغییر شرط شاخه[۲۱] و یا حذف یک عبارت[۲۲] است. نمونه ای از جهش یافته‌ها برای یک قطعه کد در ۲-۱ آمده است.

[۱۹] mutation
[۲۰] string
[۲۱] branch condition
[۲۲] expression

تحلیل جهش آزمون در موارد زیر کاربرد دارد:

- ارزیابی مجموعه آزمون
- انتخاب مجموعه آزمون
- کمینه سازی مجموعه آزمون
- تولید مجموعه آزمون
- مکان‌یابی خطا
- پیش‌بینی خطا

آزمون جهش شناخته شده‌ترین روش آزمون با استفاده از تزریق خطا است؛ آزمون جهش همچنین به طور گسترده برای ارزیابی عملکرد و بهبود مجموعه‌ای از موارد آزمون به کار گرفته می‌شود [۲۱]. آزمون جهش همچنین معمولاً به عنوان راهی برای ارزیابی کفایت مجموعه‌های آزمایشی است.این معیار با تولید یک مجموعه آزمون، نشان می‌دهد اشتباهات درج شده در جهش‌یافته‌ها در برنامه‌ی اصلی وجود ندارد، و با این کار قابلیت اطمینان برنامه را افزایش می‌دهد. برای اعمال این معیار ابتدا برنامه اصلی با مجموعه موارد آزمون اولیه اجرا می‌شود. سپس جهش یافته‌ها با همان مجموعه موارد آزمون تولید و اجرا می‌شوند. آنهایی که رفتاری متفاوت از برنامه‌ی اصلی دارند کشته‌شده در نظر گرفته می‌شوند و دیگر در آزمون استفاده نمی‌شوند. مجموعه‌ای از جهش یافته‌های زنده مورد تجزیه و تحلیل قرار گرفته و جهش یافته‌های هم‌ارز شناسایی می‌شوند. یک جهش زمانی هم‌ارز در نظر گرفته می‌شود که، برای همه موارد آزمون، دقیقاً همان رفتار برنامه‌ی تحت آزمون را نشان دهد. در نهایت، موارد آزمون جدید برای کشتن جهش یافته‌های زنده ایجاد می‌شود. علیرغم مزایای آزمایش جهش از نظر اثربخشی، برخی مشکلات مانند تعداد زیاد جهش تولیدشده، هزینه محاسباتی مورد نیاز برای اجرای آنها و تلاش زیاد لازم برای شناسایی جهش‌یافته‌های معادل مطرح می‌شود[۲۲].

فصل ۳

کارهای پیشین

در این فصل از پایان‌نامه، کارهای پیشین انجام‌شده روی مسئله به تفصیل توضیح داده می‌شود.

۳-۱ پژوهش‌های مربوط به آزمون میکروسرویس‌ها

در راستای آزمون برنامه‌ها با معماری میکروسرویس روش‌های گوناگونی ارائه شده‌اند. می‌توان اغلب روش‌های ارائه‌شده را در پنج زمینه[۱]ی اصلی دسته‌بندی کرد به طوری که آن‌ها مطالعات اصلی در راستای آزمون نرم‌افزارها با معماری میکروسرویس را مشخص می‌کنند، این ۵ دسته، شامل آزمون خودکار[۲]، معماری ، توسعه‌ـ عملیات و یکپارچه‌سازی مستمر[۳]، عملکرد[۴] و آزمون مبتنی بر مدل[۵]. البته این نکته قابل ذکر است که یک مطالعه ممکن است چندین موضوع از این ۵ زمینه‌ی گفته‌شده را در بر بگیرد.

۳-۱-۱ آزمون خودکار

این زمینه، مطالعات عمده‌ای را پوشش می‌دهد که آزمون‌های خاصی را در قالب آزمون خودکار مورد بحث قرار می‌دهند. به عنوان مثال، کوئنوم[۶] و آکنین[۷] یک رویکرد آزمون خودکار مبتنی بر توصیف صوری و عوامل هوشمند (به عنوان مثال، آزمون خودکارشده‌ی LASTA) برای استخراج موارد آزمون (به عنوان

[۱] theme
[۲] automated testing
[۳] continuous integration
[۴] performance
[۵] model-based
[۶] Quenum
[۷] Aknine

مثال، آزمون‌های واحد و پذیرش) برای برنامه‌ها با معماری میکروسرویس ارائه می‌کنند [۲۳].

به طور مشابه، شنگ[۸] و همکاران. یک طرح مبتنی بر گراف و سناریو_رانه را برای تحلیل، آزمون و استفاده مجدد از میکروسرویس‌ها معرفی کردند[۲۴]. این رویکرد از ابزار swagger برای استخراج خودکار فراخوانی‌ها بین میکروسرویس‌ها استفاده می‌کند و وابستگی‌های بین میکروسرویس‌ها به صورت خودکار به‌روز می‌شوند. در نتیجه امکان بازیابی خودکار موارد آزمون مورد نیاز در مواجهه با تغییرات میکروسرویس را فراهم می‌کند.

۳-۱-۲ معماری

در این زمینه‌ی تحقیق، بحث عمده بر استفاده یا در نظر گرفتن مصنوعات معماری (به عنوان مثال، اجزای معماری، ویژگی‌های کیفیت) برای آزمون برنامه‌های مبتنی بر میکروسرویس متمرکز است. به طور دقیق‌تر، مطالعات اصلی، رویکردهای آزمونی را ارائه می‌کنند که از اجزای طراحی میکروسرویس‌ها (به عنوان مثال، سرویس‌ها و رابط سرویس)، راهبردهای تجزیه‌ی برنامه‌ها، و روش‌های ارتباطی میکروسرویس‌ها (مانند پروتکل‌های همگام[۹] یا ناهمگام[۱۰]) برای ایجاد موارد آزمون برای آزمون برنامه‌های کاربردی مبتنی بر معماری میکروسرویس استفاده می‌کنند. علاوه بر این، چندین مطالعه، چگونگی آزمون و ارزیابی قابلیت اطمینان، انعطاف‌پذیری، و معماری برنامه‌های کاربردی مبتنی بر معماری میکروسرویس را در این زمینه می‌گویند.

به عنوان مثال هورهیادی[۱۱] و همکاران روش گرملین[۱۲] را با تمرکز بر شبکه‌های ارتباطی بین میکروسرویس‌ها ارائه داده‌اند. گرملین چارچوبی برای آزمون سیستماتیک قابلیت‌های مواجهه با شکست در میکروسرویس‌هاست. گرملین بر اساس این اصل است که میکروسرویس‌ها به طور معمول جفت‌شدگی کمی دارند و در عوض بر الگوهای استاندارد تبادل پیام در شبکه متکی هستند. گرملین به آزمون‌کننده اجازه می‌دهد تا به راحتی آزمون‌ها را طراحی کند و آن‌ها را با دستکاری پیام‌های بین سرویس‌ها در لایه شبکه اجرا کند. این پژوهش نشان می‌دهد که می‌توان از گرملین برای بیان سناریوهای شکست رایج به نحوی استفاده کرد تا توسعه‌دهندگان یک برنامه قادر باشند اشکالات ناشناخته‌ی قبلی در کد مدیریت شکست خود را بدون تغییر برنامه، شناسایی کنند[۲۵].

همچنین در پژوهشی که لوتز[۱۳] و همکارانش ارائه داده‌اند یک مطالعه موردی را بررسی می‌کنند که امکان‌سنجی و اثرات احتمالی تغییر معماری نرم‌افزار کمک‌کننده به راننده را به یک معماری میکروسرویس

[۸] Shang
[۹] synchronous
[۱۰] asynchronous
[۱۱] Heorhiadi
[۱۲] Gremlin
[۱۳] Lotz

شامل می‌شود. همچنین آزمون سیستم را برای این مورد مطالعه انجام می‌دهند. نتایج نشان می‌دهد که معماری میکروسرویس می‌تواند پیچیدگی و مراحل فرآروند زمان‌بر را کاهش دهد و سیستم‌های نرم‌افزار خودرو را برای چالش‌های آتی آماده کند تا زمانی که اصول معماری میکروسرویس به دقت دنبال شوند[۲۶].

۳-۱-۳ توسعه_عملیات و یکپارچه‌سازی مستمر

توسعه_عملیات و یکپارچه‌سازی مستمر، دارای طیف وسیعی از اقدامات است (به عنوان مثال، یکپارچه سازی مستمر[۱۴]، تحویل مستمر[۱۵]، آزمون، استقرار) آن‌ها با هدف ارائه‌ی سیستم‌های نرم‌افزاری قابل اعتماد با تشویق همکاری نزدیک بین کارکنان توسعه و عملیات پیشنهاد شده‌اند. این زمینه‌ی پژوهشی شامل مطالعاتی است که رویکردهای آزمون و ابزارهای مورد استفاده برای برنامه‌های مبتنی بر معماری میکروسرویس در توسعه_عملیات و یکپارچه‌سازی مستمر را گزارش می‌کنند. خودکارسازی آزمون یک عامل کلیدی در موفقیت با توسعه_عملیات است. به عنوان مثال، کارگر و حنیفی زاده یک روش خودکار را برای پشتیبانی از تست رگرسیون میکروسرویس‌ها در تحویل مداوم پیشنهاد کردند[۲۷].

مارسل[۱۶] و کریستوس گرابمن[۱۷] برنامه‌های میکروسرویسی را برای سیستم عامل شبکه باز (ONOS) با ایجاد محیط یکپارچه‌سازی مستمر توسعه دادند و آزمون کردند. آن‌ها همچنین یک توپولوژی آزمایشی را پیشنهاد کردند که برای ارزیابی برنامه‌های ONOS استفاده می‌شود[۲۸].

۴-۱-۳ عملکرد

این موضوع به جنبه‌های کمّی رفتاری آزمون می‌پردازد. نتایج نشان می‌دهد که مطالعات اولیه در این زمینه بر روی آزمون عملکرد، عمدتاً در مرحله‌ی تولید توسعه برنامه کاربردی مبتنی بر معماری میکروسرویس تمرکز دارد. کامارگو[۱۸] و همکاران. رویکردی برای ارزیابی عملکرد میکروسرویس به تنهایی ارائه کردند آن‌ها با یکپارچه‌سازی آزمون با میکروسرویس روش خود را توضیح داده‌اند[۲۹].

علاوه بر این، شارما[۱۹] و همکاران. با استفاده از مدل تحلیلی و اجرای آزمایش‌های بستر آزمون، عملکرد بین برنامه‌های یکپارچه و میکروسرویسی را در زمینه مجازی‌سازی عملکرد شبکه مورد آزمون و مقایسه قرار دادند[۳۰].

---

[۱۴] continuous integration (CI)
[۱۵] continuous delivery (CD)
[۱۶] Marcel
[۱۷] Grobmann Christos
[۱۸] Camargo
[۱۹] Sharma

۳-۱-۵ آزمون مبتنی بر مدل

این زمینه پژوهشی، آن دسته از مطالعات اولیه را که در مورد رویکردهای آزمون مبتنی بر مدل برای برنامه‌های کاربردی مبتنی بر معماری میکروسرویس بحث می‌کنند، جمع آوری می‌کند.

به عنوان مثال، کامیلی[۲۰] و همکاران. یک چارچوب رسمی مبتنی بر مدل‌های شبکه پتری ارائه می‌کند که در زمینه تست میکروسرویس قابل اجرا است. در این پژوهش یک آنتولوژی[۲۱] رسمی مبتنی بر شبکه‌های پتری برای جریان‌های فرآروند مبتنی بر ریزسرویس‌های مشخص شده با استفاده از زبان هماهنگ‌سازی کنداکتور ارائه شده است. کنداکتور یک زبان خاص دامنه مبتنی بر JSON است که توسط نتفلیکس طراحی شده است. همچنین یک معناشناسی صوری از ترجمه‌ی توصیف‌ها در کنداکتور به مدل‌های شبکه پتری که بر پایه‌ی زمان هستند ارائه می‌شود. این مدل‌ها یا شبکه‌های پتری، از تعریف محدودیت‌های زمانی پشتیبانی می‌کنند. این نوع از مدل‌های مبتنی بر شبکه‌ی پتری را می‌توان برای اهداف صحت‌سنجی به کمک کامپیوتر با استفاده از تکنیک‌های شناخته‌شده‌ی پیاده‌سازی شده توسط ابزارهای قدرتمند بررسی مدل آماده به کار، استفاده کرد[۳۱].

شولز[۲۲] و همکاران یک رویکرد آزمون مبتنی بر مدل را برای تولید مدل‌های فشار کاری برای آزمایش فشار یک یا چند میکرسرویس مشخص معرفی کرده‌اند[۳۲].

---

[۲۰] Camili
[۲۱] Ontology
[۲۲] Schulz

جدول ۳-۱: دسته‌بندی موضوعات پژوهشی مرتبط با آزمون میکروسرویس

زمینه‌ی پژوهش	نکات کلیدی پژوهش	پژوهش
آزمون خودکار	یک رویکرد مبتنی بر مشخصات رسمی برای استخراج موارد آزمایشی (به عنوان مثال، موارد آزمون پذیرش) برای تست میکروسرویس خودکار	[۲۳]
	طرح مبتنی بر گراف وابستگی سرویس‌ها برای تحلیل، آزمون و استفاده مجدد از میکروسرویس‌ها	[۲۴]
معماری	آزمون انعطاف‌پذیری میکروسرویس‌ها در زیرساخت‌های تولید	[۲۵]
	آزمون عملکردهای سیستم کمک راننده پیشرفته مبتنی بر معماری میکروسرویس (ADAS)	[۲۶]
توسعه‌ـعملیات و یکپارچه‌سازی مستمر	آزمون رگرسیون برنامه‌های کاربردی مبتنی بر معماری میکروسرویس در تحویل مداوم	[۲۷]
	آزمون مداوم سیستم عامل شبکه باز ONOS	[۲۸]
عملکرد	ارزیابی عملکردی که هر میکروسرویس می‌تواند ارائه دهد	[۲۹]
	آزمایش و مقایسه عملکرد برنامه‌های یکپارچه و میکروسرویس در NFV	[۳۰]
مبتنی بر مدل	شبکه‌های پتری به عنوان پایه‌ی آزمون‌های مبتنی بر مدل میکروسرویس‌ها	[۳۱]
	تولید مدل‌های فشار کاری مبتنی بر سشن برای آزمون فشار میکروسرویس‌ها	[۳۲]

فصل ۴

روش پیشنهادی

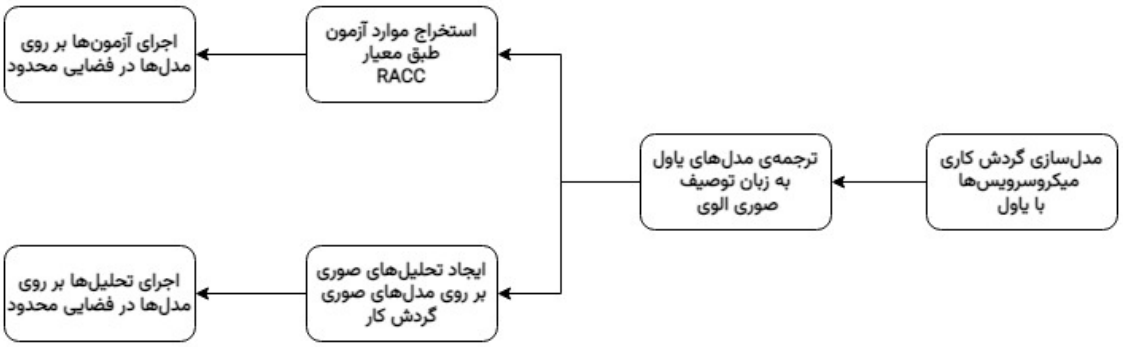
۴-۱ چارچوب کلی

چارچوب کلی روش پیشنهادی ارائه‌شده بدین صورت است که در ابتدا گردش‌های کاری میان میکروسرویس‌های تشکیل‌دهنده‌ی نرم‌افزار را با استفاده از زبان یاول مدل می‌کنیم. سپس این مدل‌ها را بر اساس روش ترجمه‌ای اثبات شده، به یک توصیف صوری در زبان الوی ترجمه می‌کنیم. بعد از آن بر روی توصیف صوری از میکروسرویس‌ها و ارتباطات آن‌ها تحلیل‌هایی صوری به زبان الوی انجام می‌دهیم تا وجود موقعیت‌های ناخواسته و غیرمجاز در اجرای گردش کاری نرم‌افزار را بررسی کنیم. به علاوه با استفاده از توصیفات صوری مذکور و بر اساس معیار پوشش گزاره‌ی فعال محدود، نیازمندی‌های آزمون و سپس برای برآورده کردن هر یک از نیازمندی‌های آزمون موارد آزمون را تولید می‌کنیم و در انتها موارد آزمون به دست آمده که آن‌ها نیز توصیفاتی صوری هستند را به توصیف اصلی اضافه می‌کنیم؛ سعی می‌کنیم یک نمونه از توصیف به‌دست آمده بسازیم، در صورت سازگار ماندن توصیف، این نمونه به دست می‌آید و موفقیت‌آمیز بودن اجرای آزمون را نشان می‌دهد.

روند کلی روش ارائه‌شده در شکل ۴-۱ آورده شده است.

تمام این روند، پس از مدل‌سازی میکروسرویس‌ها در زبان یاول توسط طراحان نرم‌افزار، تا ایجاد موارد آزمون و اعمال آن‌ها بر روی مدل صوری به صورت خودکار در ابزار مدل‌سازی یاول انجام می‌شود.

در ادامه به تشریح گام‌های ذکر شده در روش پیشنهادی می‌پردازیم. اما برای روشن‌تر شدن این روش یک نمونه‌ی برگرفته‌شده از یک سیستم نرم‌افزاری با معماری میکروسرویس در دنیای واقع را می‌آوریم و در هر گام، روش تشریحی خود را بر روی این نمونه‌ی واقعی اِعمال می‌کنیم.



شکل ۴-۱: روند کلی روش ارائه‌شده

۴-۲ مدل‌سازی سیستم‌های پیچیده با معماری میکروسرویس با یاول

امروزه خیلی از برنامه‌ها شامل مجموعه‌ای از سرویس‌ها هستند هر میکروسرویس به طور مستقل توسعه یافته، مستقر و مدیریت می‌شود. همکاری میکروسرویس‌ها با یکدیگر هدف برنامه را محقق می‌کند؛ هماهنگی و تعامل میان میکروسرویس‌ها با به کارگیری یکی از دو رویکرد ارکستراسیون یا موزون انجام می‌شود. به دلیل ذات غیرمتمرکز میکروسرویس‌ها به نظر می‌رسد استفاده از رویکرد موزون برای ترکیب[۱] آن‌ها مناسب تر باشد. در سبک موزون، هر میکروسرویس به طور مستقل کار می‌کند، در حالی که، در ارکستراسیون، یک کنترل‌کننده[۲] وجود دارد که تعاملات سرویس را هماهنگ می‌کند[۳۳][۳۴].

ما روشی برای آزمون هر دو این سبک‌ها ارائه می‌کنیم اما نمونه‌ای که برای روشن‌تر شدن موضوع استفاده شده‌است از رویکرد ارکستراسیون برای ساختن معماری میکروسرویسی خود استفاده می‌کند.

استفاده از زبان مدل‌سازی یاول که یک زبان مدیریت فرارونده کسب و کاری است یکی از رویکردها برای مدل‌سازی از این تعاملات بین میکروسرویس‌ها در یک برنامه‌ی پیچیده است[۳۳].

در روش پیشنهادی، ابتدا یک برنامه با استفاده از یاول مدل می‌شود، میکروسرویس‌ها واحدهای مستقلی هستند که هر کدام جزئی از کل کار برنامه را بر عهده دارند و گردش کار بین میکروسرویس‌ها معمولا با رد و بدل شدن پیام بین آن‌ها انجام می‌پذیرد [۲۵]، در سبک موزون این ارتباط بدون واسطه و مستقیم انجام می‌شود.

در یاول کوچک‌ترین واحدهای کاری مستقل، وظایف هستند و می‌توان آن‌ها را معادل میکروسرویس‌ها در یک برنامه در نظر گرفت. همچنین در یاول ارتباط میان وظایف با جریان‌ها برقرار می‌شوند، [۱۰] می‌توان

[۱] Compostion
[۲] Controller

برای نمایش ارسال پیام‌ها میان میکروسرویس‌ها از جریان‌ها استفاده کرد. از انشعاب‌ها و اتصال‌ها برای هدایت گردش کار در یاول استفاده می‌شود، در آن‌ها با توجه به متغیرهای ورودی و خروجی و شروط اعمال شده بر روی آن‌ها گردش کار توسط وظایف هدایت می‌شود. مقادیر متغیرها در برنامه را خروجی میکروسرویس‌ها تعیین می‌کنند و جریان کنترل در برنامه با توجه به خروجی هر میکروسرویس بین آن‌ها گردش می‌کند.

۳-۴ خودکارسازی ترجمه‌ی مدل‌های یاول به الوی

۱-۳-۴ توصیف مدل‌ها در زبان الوی

برای ترجمه‌ی مدل‌های یاول به توصیفات صوری در زبان الوی، از روشی که در پژوهش ریواده و همکاران انجام شده است[۱۱] استفاده کردیم؛ در روشی که ریواده و همکاران برای ترجمه ارائه کرده‌اند، ساختار مدل‌ها در یاول به دو بخش ایستا و پویا تقسیم‌بندی شده‌است و برای هر یک از موجودیت[۳]ها در هر دسته معادلی در الوی ذکر شده‌است. همچنین برای ویژگی‌های ذاتی مدل‌های گردش کاری در یاول مانند هم‌بند بودن گراف، در الوی حقیقت[۴]هایی تعریف شده‌است. در نهایت روشی برای ترجمه‌ی مدل‌ها به دست آمده و سپس با نه قضیه نشان داده‌است که ترجمه‌ی به دست آمده از مدل‌های گردش کاری در یاول به زبان الوی کامل و صحیح[۵] هستند.

بخش ایستا در مدل‌ها، همان مفاهیم و مولفه‌های موجود در زبان یاول هستند. در پژوهش ریواده این بخش شامل وظیفه[۶]، شرط ورودی[۷]، شرط خروجی[۸] و شرط است؛ برای هر کدام از این مولفه‌ها در الوی یک معادل در قالب نشان[۹] آورده شده است. همچنین رفتار انواع پیوند[۱۰]ها و انشعاب[۱۱]ها نیز در قالب حقیقت‌ها بیان شده اند. علاوه بر این‌ها بخش ایستا در ترجمه‌ی تولیدشده شامل تعریف حالت[۱۲] در یک گردش کار نیز می‌شود به تعریف حالت در الوی ترتیب اضافه شده‌است، این کار اجازه می‌دهد بتوان تغییر حالت‌ها در زمان را مدل کرد. ترتیب حالت‌ها توسط پودمان کتابخانه util/ordering ارائه می‌شود. این پودمان عمومی است - یعنی می‌تواند به مجموعه‌ای از هر نوع ترتیب بدهد - بنابراین وقتی باز می‌شود باید

[۳] Entity
[۴] fact
[۵] sound
[۶] task
[۷] input condition
[۸] output condition
[۹] signature
[۱۰] join
[۱۱] split
[۱۲] state

با یک نوع (در این مورد، حالت) نمونه‌سازی[۱۳] شود[۱۲].

مجموعه‌ی توکن مجموعه‌ای از وظایف یا شرط ورودی یا شرط خروجی است که کاری در آن‌ها در حال انجام است در هر حالت از گردش کار، توکن در یک یا چند وظیفه وجود دارد و با تغییر حالت، در میان وظیفه‌ها جابجا می‌شود. در واقع و تغییر حالت متناظر با تغییر مجموعه‌ی توکن است. بخش پویا مرتبط با معماری میکروسرویسی است که در یاول مدل شده است.

برای ترجمه‌ی مدل تعریف شده به الوی، برای هر وظیفه که معادل یک میکروسرویس است، حقیقت یا حقیقت‌هایی در الوی تعریف می‌کنیم و در آن(ها) ویژگی‌های وظیفه شامل نام، نوع پیوند، نوع انشعاب و جریان‌های خروجی آن را ذکر می‌کنیم؛ همچنین مسندهایی که در جریان‌های خروجی وظیفه تعریف شده‌اند را نیز بسته به نوع انشعاب، در گزاره‌های شرطی ذکر می‌کنیم. بخش پویا در واقع نشان‌دهنده‌ی میکروسرویس‌ها و نحوه‌ی ارتباط آن‌ها با یکدیگر هستند و شامل وظیفه و جریان‌های بین آن‌ها می‌شود.

به روش ریواده و همکاران برای ترجمه‌ی مدل‌های یاول به الوی موارد جدیدی اضافه کردیم که در ادامه به آن‌ها می‌پردازیم؛ در ترجمه‌ای که از مدل‌های یاول تولید می‌کنیم، شامل توصیف ناحیه‌ی لغو در بخش ایستا نیز می‌باشد، همچنین در تعریف وظیفه‌ها نیز مجموعه‌ی وظیفه‌هایی که در ناحیه‌ی لغو آن وظیفه وجود دارند تعریف می‌شود. متغیرهای موجود در مدل گردش کاری که شامل ورودی خروجی وظیفه‌ها و معادل ورودی و خروجی میکروسرویس‌ها هستند، در تعریف حالت ذکر می‌شوند. مقادیر متغیرها در هر حالت امکان تغییر دارند و رفتار میکروسرویس‌ها در هر حالت بسته به مقدار متغیرها و شرایط درونی میکروسرویس می‌تواند تغییر کند.

۴-۳-۲ جای‌گذاری روش در ابزار ویرایش یاول

روش پیشنهادی گفته شده را در ابزار ویرایشگر یاول جاسازی کردیم و کار ترجمه در این ابزار به صورت خودکار انجام می‌شود و در خروجی به طراح نرم‌افزار نمایش داده می‌شود. برای تولید ترجمه به این صورت عمل می‌کنیم: قسمت‌های ایستا به صورت ثابت و ایستا به ترجمه اضافه می‌شوند اما برای ترجمه‌ی خودکار مولفه‌های متغیر در مدل‌ها، پیش‌پردازشی بر روی فرم استاندارد ذخیره‌شده‌ی مدل یاول انجام می‌دهیم و سپس با روشی هر کدام از مولفه‌های پویا را تجزیه و تحلیل می‌کنیم و آن‌ها را به اشیائی از کلاس‌های تعریف شده در زبان جاوا تبدیل می‌کنیم و سپس در انتهای کار ترجمه از هر کدام از آن اشیاء، توصیف آن‌ها را به زبان الوی پرس و جو می‌کنیم و سپس آن‌ها را تجمیع و به قسمت‌های ایستای توصیف اضافه می‌کنیم. در نهایت، توصیف به دست آمده شامل تعاریف اجزای ایستای مدل‌ها که شامل انواع پیوندها و انشعاب‌ها و

[۱۳] instantiate

همچنین تعاریف وظیفه و شرط ورودی و شرط خروجی است.

۴-۴ تحلیل صوری مدل‌ها

با توجه به اینکه گردش‌های کاری میکروسرویس‌های استقراریافته، ممکن است برای مدت طولانی به صورت مداوم اجرا شوند و ممکن است اقدامات زیادی انجام دهند که به سادگی قابل بازگرداندن نیستند، تشخیص خطاها در زمان طراحی بسیار می‌تواند کمک‌کننده باشد. هنگامی که از تحلیل گردش کار صحبت می‌کنیم در واقع به این می‌پردازیم که آیا یک گردش کار رفتارهای مطلوب خاصی را نشان می‌دهد یا خیر.

پژوهش‌های قبلی زیادی وجود دارد که بر روی تحلیل‌های گردش کار انجام شده‌اند مانند [۳۸] که در آن از تکنیک‌های تحلیل شبکه‌ی پتری برای تشخیص درست بودن یا نبودن شبکه گردش کار استفاده می‌شود. اما ضعف آن‌ها در این است که، نتایج پژوهش اشاره‌شده به راحتی قابل اعمال به برخی موقعیت‌ها نیستند. موقعیت‌هایی که در آن زبان‌هایی درگیر هستند که از مفاهیمی مانند منطقه‌ی لغو و پیوند از نوع "یا" استفاده می‌کنند. این ضعف به این دلیل است که این مفاهیم را نمی‌توان به راحتی از طریق شبکه‌های پتری بیان کرد. در پژوهش [۳۷] سعی شده است تا روشی بر مبنای شبکه‌های بازنشانی ارائه شود تا این ضعف را پوشش دهد. ما نیز در این بخش به تحلیل‌هایی که با استفاده از روش‌های صوری بر روی مدل گردش کاری در میکروسرویس‌ها انجام می‌دهیم، می‌پردازیم.

تحلیل‌های که در ادامه می‌آیند، با بررسی درستی اظهار در الوی انجام می‌شود به این صورت که یک اظهار به کل تعریف گردش کاری افزوده می‌شود و سپس توصیف حاصل‌شده را به موتور الوی می‌دهیم. اظهارها به صورت کلی به این فرمت نوشته می‌شوند که "در هیچ نمونه‌ای از توصیف موجود حالت غیرمجاز وجود ندارد" اگر موتور الوی بتواند مثال نقضی برای این اظهار پیدا کند، پیدا شدن نمونه‌ای که این اظهار را نقض کند، در کاربرد ما به منزله‌ی وجود حالتی غیرمجاز است که ساختار گردش کاری باعث پیدایش آن شده است.

- بررسی پیوند از نوع "یا" در حلقه:

بنا بر تعریف، این وضعیت که یک وظیفه که دارای پیوند از نوع "یا" است در حلقه قرار گیرد، غیرمجاز است؛ در این حالت، وظیفه‌ای که دارای پیوند از نوع "یا" است منتظر مشخص شدن وضعیت شاخه‌ی ورودی خود است در حالی که مشخص شدن وضعیت شاخه‌ی ورودی به خروجی همین وظیفه بستگی دارد و در نتیجه گردش کار دچار نوعی بن‌بست می‌شود. برای تشخیص این حالت غیرمجاز در روش پیشنهادی خود، تحلیلی را ارائه کردیم که با بررسی ساختار گردش کاری

ورودی که آیا وظیفه‌ای باعث ایجاد این حالت غیرمجاز می‌شود یا خیر.

همان‌طور که در بالاتر شرح داده‌شد، این تحلیل با افزودن یک اظهار و سپس بررسی آن در توصیف انجام می‌شود. اظهار به صورت زیر توصیف می‌شود.

```
assert no_or_join_in_loop {
    all t: task | t.label = "task_title" => t not in t.^(flowsInto.nextTask)
}
```

- قابل دسترس بودن وظایف:

میکروسرویس‌هایی که در معماری نرم‌افزار استفاده شده‌اند مسئولیت انجام قسمتی از خدمت را به عهده دارند در نتیجه در طول اجرای نرم‌افزار زمانی وجود خواهد داشت که هر میکروسرویس در حال انجام وظیفه‌ای است که به عهده دارد، اگر میکروسرویسی در جریان کنترل نرم‌افزار قابل دسترسی نباشد وجود آن میکروسرویس برای نرم‌افزار غیرضروری است؛ به طوری که گردش کار بدون وجود آن وظیفه‌ی غیر قابل دسترس نیز همان رفتاری را نشان می‌دهد که با وجود آن وظیفه دارد.

در این تحلیل، برای هر وظیفه به دنبال این هستیم که در حداقل یک حالت از گردش کار، وظیفه‌ی مورد بررسی، در مجموعه‌ی توکن آن حالت باشد. این تحلیل نیز با افزودن یک اظهار به توصیف و سپس پیدا کردن مثال نقض برای آن توسط موتور الوی انجام می‌شود. اظهار به صورت زیر نوشته می‌شود.

```
assert is_any_state_task_is_token_in_it {
    all t: task | t.label = "task_title" => {
        all s: State | t not in s.token
    }
}
```

اظهار بالا برای تک تک وظایف گردش کاری به صورت متوالی به توصیف گردش کاری افزوده می‌شود و بررسی نیز انجام می‌شود و نتیجه‌ی بررسی آن‌ها به صورت تجمیع شده در خروجی برمی‌گردد.

- منتظر ماندن دو وظیفه با پیوند از نوع "یا" برای یکدیگر:

در این تحلیل، پس از بررسی وجود دو وظیفه با پیوند از نوع "یا" در گردش کاری، با تحلیل ساختار بررسی میکنیم که آیا این دو وظیفه در تصمیم گیری برای جلو رفتن گردش کار در خودشان منتظر یکدیگر هستند یا خیر. چرا که با منتظر بودن این دو وظیفه برای مشخص شدن وضعیت دیگری، وضعیت بن‌بست در گردش کاری رخ می‌دهد که غیرمجاز است.

این تحلیل نیز با افزودن یک اظهار و سپس بررسی آن در توصیف انجام می‌شود. اظهار را به صورت زیر می‌نویسیم.

```
assert no_two_or_joins_pend_on_each_other {
    all t1, t2: task | t1.label = "service1_title" && t2.label = "service2_title" => t2 not in t1.^(flowsInto.nextTask)
}
```

۵-۴ ایجاد موارد آزمون و اعمال آن‌ها بر روی مدل

آزمون نرم افزار یک فرآیند حیاتی است که کیفیت محصولات نرم افزاری را تضمین می‌کند. در روش آزمون ارائه‌شده آزمون‌ها بر اساس معیارهای پوشش گزاره‌ی فعال محدود تولید شده‌اند.

پس از اطمینان از قابل دسترس بودن همه‌ی وظایف و هم‌چنین اطمینان از عدم برخورد با وضعیت‌های غیرمجاز به آزمون گردش کار می‌پردازیم در رویکرد ارائه‌شده، جریان کنترل گردش کار را مورد آزمون قرار می‌دهیم، برای آزمون مدل صوری به زبان الوی، موارد آزمون را در مکان‌هایی که در آن‌ها جریان کنترل تعیین می‌شود، تولید می‌کنیم.

در یک سیستم نرم‌افزاری با معماری میکروسرویس، هر خدمت یا یک مجموعه خدمت توسط یک میکروسرویس انجام می‌گیرد، پاسخ به درخواست کاربر از نقطه‌ای آغاز می‌شود و سپس گردش کار در میان میکروسرویس‌ها شامل ارتباط بین سرویس‌های مختلف برای تکمیل یک فراروند است. هر میکروسرویس ورودی را از سرویس‌های دیگر دریافت می‌کند، عملکرد خود را انجام می‌دهد و خروجی را به سرویس‌های دیگر می‌فرستد. به عبارتی گردش کار به عملکرد هر سرویس وابسته است و تعیین مسیر در گردش کاری با توجه به ورودی و خروجی‌های سرویس‌ها انجام می‌گیرد.

مدل گردش کاری میکروسرویس‌ها مدل‌سازی سطح بالایی است و به مدل‌سازی از عملکرد داخلی سرویس‌ها پرداخته نمی‌شود و تنها ارتباط آن‌ها و نحوه‌ی تعیین جریان کنترل در آن مشخص می‌شود. در نتیجه ترجمه‌ی این مدل‌ها نیز به صورت صوری شامل توصیف عملکرد داخلی میکروسرویس‌ها نیست و در آن صرفا به توصیف نحوه‌ی گردش کار و جریان کنترل در سیستم پرداخته می‌شود. جریان کنترل در زبان یاول، به وسیله‌ی پیوند یا انشعاب‌ها از انواع مختلف انجام می‌شود برای این درگاه‌ها بسته به نوع آن‌ها شروطی تعیین می‌شود و تصمیم‌گیری با توجه به این شروط و همچنین نوع درگاه گرفته می‌شود. ما برای تولید موارد آزمون مسندهایی که در انشعاب‌ها توسط طراح نوشته می‌شوند را مبنا قرار می‌دهیم. گزاره‌های شرطی موجود در انشعاب‌ها در ترجمه‌ی مدل‌ها به زبان الوی در نظر گرفته می‌شوند و در توصیف وظایف آن‌ها نیز توصیف می‌شوند. با پیش‌پردازش توصیف به دست آمده، انشعاب‌ها را می‌یابیم و سپس مسندهای آن‌ها را برای تولید موارد آزمون استخراج می‌کنیم.

برای تولید خودکار موارد آزمون به روش پوشش گزاره‌ی فعال محدود پودمانی ساختیم که با ورودی مسند، جفت‌های موارد آزمون را برای پوشش گزاره‌ی فعال محدود تولید می‌کند.

برای به دست آوردن زوج موارد آزمون بر اساس پوشش گزاره‌ی فعال محدود، نیاز است که گزاره‌های اصلی که ارزش کلی مسند را تعیین می‌کنند بیابیم. نیازمندی آزمون برای هر c_i دو شرط وجود دارد: گزاره‌ی اصلی به ارزش "درست" و به ارزش "نادرست" اطلاق شود. مقادیر انتخاب شده برای گزاره‌های فرعی باید در زمانی که گزاره‌ی اصلی "درست" است با زمانی که گزاره‌ی اصلی "نادرست" است یکسان باشد[۱۲]. می‌توان با محاسبه‌ی مسند به روش زیر برای یک گزاره‌ی خاص، تعیین ارزش مسند را به ارزش گزاره‌ی اصلی وابسته کرد.

$$P_a = p_{a=True} \oplus p_{a=False}$$

برای تولید زوج موارد آزمون بعد از محاسبه‌ی مسند به صورتی که ارزش آن را گزاره‌ی اصلی تعیین کند، ارزش گزاره اصلی را یک بار برابر با "درست" و یک بار نیز برابر "نادرست" قرار می‌دهیم. برای این که بتوانیم ارزش گزاره‌ی اصلی را برابر "درست" یا "نادرست" قرار بدهیم به گونه‌ای متغیرهای مورد استفاده در گزاره‌ی شرطی را مقداردهی می‌کنیم تا گزاره ارزش مورد نظر را پیدا کند. در روشی که برای تولید موارد آزمون استفاده کردیم، ابتدا با تجزیه و تحلیل مسند موجود در جریان خروجی گزاره‌های شرطی تشکیل‌دهنده‌ی مسند و متغیرهای موجود در آن‌ها را را استخراج می‌کنیم و سپس درخت عبارت منطقی را می‌سازیم. درخت عبارت منطقی درختی دودویی است که برای نشان دادن عبارات منطقی و استدلال‌های منطقی استفاده می‌شود. برگ‌های این درخت همان گزاره‌های شرطی هستند و راس‌های میانی آن عملگرهای منطقی هستند. مسند

$$(a \wedge b) \vee c$$

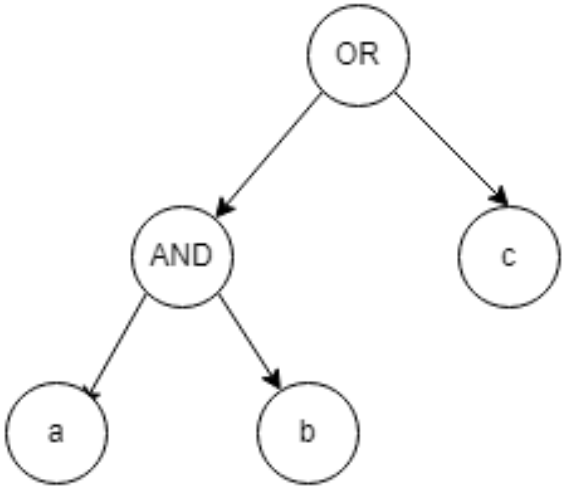
را در نظر بگیرید شکل ۲-۴ نمایانگر درخت عبارت منطقی آن است.

با استفاده درخت عبارت منطقی به ازای همه‌ی P_iها عبارت

$$P_a = p_{a=True} \oplus p_{a=False}$$

را محاسبه می‌کنیم، سپس با ثابت نگه‌داشتن ارزش گزاره‌های فرعی ارزش P_i را یک بار برابر "درست" و یک بار برابر "نادرست" قرار می‌دهیم. در انتها با تحلیل گزاره‌ها و تجزیه‌ی آن‌ها به عملگرها و عملوندی تشکیل‌دهنده، مقادیر متغیرها را به گونه‌ای به آن‌ها نسبت می‌دهیم که ارزش مورد نظر برای گزاره حاصل شود.

پس از تولید موارد آزمون، آن‌ها را بر روی توصیف اعمال می‌کنیم. هر یک از موارد آزمون را با زبان الوی به صورت صوری توصیف می‌کنیم، سپس توصیف را به توصیف کلی گردش کار اضافه می‌کنیم و آن



شکل ۴-۲: درخت عبارت منطقی برای گزاره‌های مسند $(a \wedge b) \vee c$

را با واسطه‌های برنامه‌نویسی کاربردی ابزار تحلیل الوی اجرا می‌کنیم.

اجرای آزمون‌ها با استفاده از تحلیل‌گر الوی انجام می‌شود و در نتیجه در فضایی کوچک و با حالت‌های محدودشده انجام می‌گیرد[۱۹]. در نتیجه نمی‌توان موفقیت اجرای یک آزمون را معادل موفقیت همیشگی برنامه با ورودی‌های مورد آزمون دانست، اما ناموفق بودن آزمون وجود خطا در برنامه با وجود ورودی‌های مورد آزمون را تضمین می‌کند. به همین دلیل استفاده از تحلیل‌گر الوی به قدرت روش ارائه‌شده در تشخیص خطاها در برنامه می‌افزاید.

زوج‌های موارد آزمون تولید شده برای مجموعه متغیرهای استفاده شده در عبارات شرطی در هر درگاه تولید می‌شوند. طبق تعریف پوشش گزاره‌ی فعال محدود ارزش هر متغیری که مورد آزمون است، تعیین‌کننده‌ی ارزش کلی مسند است. سازگار بودن توصیفی که از اعمال موارد آزمون به دست می‌آید، به معنی موفق بودن آزمون است.

فصل ۵

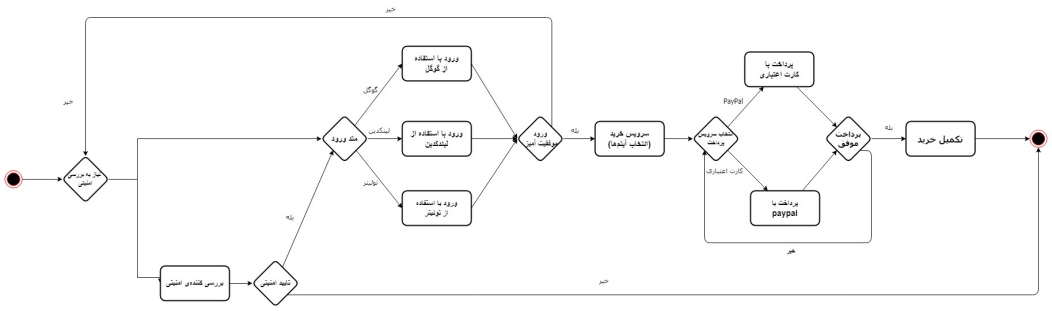
مورد مطالعاتی: برنامه‌ی خرید برخط

در این فصل روش ارائه‌شده در قسمت ۴-۱ را بر روی نمونه‌ای اجرا خواهیم کرد، در ابتدا نمونه را تشریح می‌کنیم و سپس مرحله به مرحله بر روی آن روش خود را اجرا می‌کنیم. در طول این قسمت ممکن است بر روی مورد مطالعاتی تغییراتی اعمال شود تا کارایی تحلیل‌ها یا روش آزمون ارائه‌شده بیشتر مشخص شود.

تشریح مورد مطالعاتی

نمونه‌ای که گفته خواهد شد، یک برنامه‌ی خرید به صورت برخط است که از میکروسرویس‌های مستقل برای ارائه‌ی خدمت خود تشکیل شده است. این میکروسرویس‌ها شامل میکروسرویس ورود با استفاده از گوگل[۱]، ورود با استفاده از توئیتر[۲] ورود با استفاده از لینکدین[۳]، سرویس خرید (انتخاب موارد خرید)، پرداخت با کارت اعتباری، پرداخت با پی‌پل[۴] و تایید پرداخت است. می‌توان گفت که این خدمت با همکاری چند میکروسرویس ریزدانه‌تر محقق می‌شود؛ در این برنامه با آغاز روند خرید و اولین درخواست کاربر، اگر برنامه تشخیص بدهد که نیاز به بررسی امنیتی درخواست کاربر وجود دارد میکروسرویس بررسی‌کننده‌ی امنیت این وظیفه را انجام می‌دهد و اگر درخواست را غیر مجاز یا خرابکارانه تشخیص دهد، بلافاصله فرآیند برنامه به اتمام می‌رسد و کاربر مجبور است درخواست دیگری برای آغاز خرید ارسال کند؛ اما در صورتی که درخواست از جنبه‌ی امنیتی تایید شود، کاربر باید احراز هویت خود را انجام دهد و این هدف توسط یکی از سه میکروسرویس ورود با استفاده از گوگل، ورود با استفاده از توئیتر ورود با استفاده از لینکدین به انتخاب کاربر، محقق می‌شود. اگر برنامه، بررسی امنیتی درخواست کاربر را لازم نداند، احراز هویت کاربر بلافاصله پس از درخواست کاربر صورت می‌پذیرد و اگر احراز هویت کاربر به

[۱] Google
[۲] Twitter
[۳] Linkedin
[۴] PayPal



شکل ۵-۱: گردش کاری یک برنامه‌ی نمونه برای خرید برخط

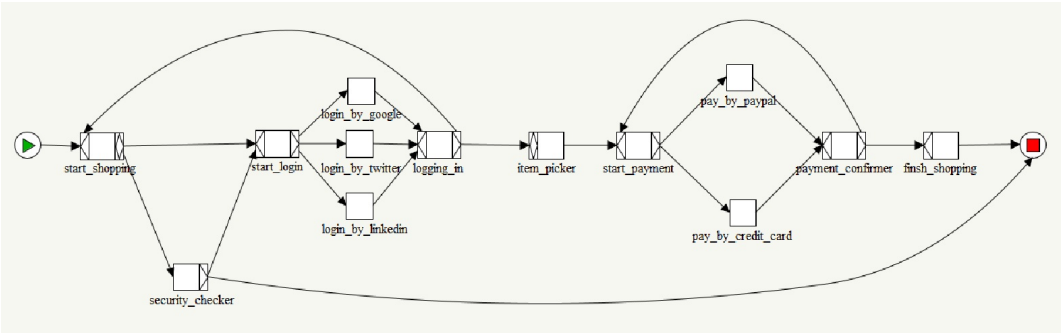
درستی صورت نگیرد کاربر به حالت ابتدایی خود پس از درخواست شروع برنامه منتقل می‌شود و روند گفته‌شده در تلاش بعدی کاربر تکرار می‌شود؛ در غیر این صورت پس از احراز هویت کاربر، انتخاب کالاها و پر کردن سبد خرید توسط میکروسرویس نمایش و انتخاب کالاها، در تعامل با کاربر انجام می‌شود.

قطعه‌ی کدی که در پیوست ۱ آمده، ترجمه‌ی نمونه‌ای است که در شکل ۵-۲ آمده است.

پس از انتخاب کالاها، کاربر باید برای تکمیل خرید خود سبد خود را تسویه حساب کند، این وظیفه نیز توسط یکی از دو میکروسرویس پرداخت یعنی پرداخت با کارت اعتباری یا پرداخت با پی‌پل انجام می‌شود. اگر پرداخت موفقیت‌آمیز باشد جریان کنترل برنامه به میکروسرویس انتخاب کالا باز گردانده می‌شود و اگر پرداخت موفق باشد نیز میکروسرویس اتمام خرید که نحوه‌ی دریافت و تاییدیه‌ی پرداخت کاربر را به او نمایش می‌دهد خدمت خرید را به اتمام می‌رساند.

شکل ۵-۱ نشان‌دهنده‌ی گردش کاری این برنامه است و شکل ۵-۲ مدل‌سازی از برنامه‌ی پرداخت، به زبان یاول است.

در این مثال وظیفه‌ی "بررسی‌کننده‌ی امنیتی" انشعاب از نوع "یا" دارد و بعد از اتمام این وظیفه با توجه



شکل ۵-۲: مدل گردش کاری برنامه‌ی خرید برخط در یاول

به پاسخ آن یا "شروع فرآیند ورود" بلافاصله آغاز می‌شود و یا این که بررسی‌کننده‌ی امنیتی خدمت برنامه، را منع می‌کند. همچنین وظیفه‌ی "شروع فرآیند ورود" دارای انشعاب از نوع "یای انحصاری" است زیرا کاربر تنها با یک روش از روش‌های سه‌گانه‌ی ورود ممکن (یعنی "ورود با گوگل"، "ورود با لینکدین" و "ورود با توئیتر")، می‌تواند احراز هویتش را انجام دهد و وارد حساب کاربری خود شود. اگر که پاسخ میکروسرویس‌هایی که عمل احراز هویت را انجام می‌دهند تایید هویت کاربر باشد، ورود کاربر موفقیت آمیز است و جریان کنترل برنامه به میکروسرویس "خرید (انتخاب موارد)" می‌رسد و در غیر این صورت جریان کنترل به ابتدای برنامه، یعنی "شروع خرید" باز می‌گردد.

پس از انتخاب موارد خرید توسط میکروسرویس "خرید"، بسته به انتخاب کاربر، یکی از میکروسرویس‌های پرداخت ("پرداخت با کارت اعتباری" و "پرداخت با پی‌پل") شروع به کار می‌کنند در نتیجه انشعاب در وظیفه‌ی "شروع پرداخت" از نوع "یای انحصاری" است. اگر پاسخ میکروسرویس پرداخت، پرداخت موفقیت آمیز باشد، روند خرید تکمیل می‌شود و خدمت برنامه به پایان می‌رسد؛ اما اگر "پرداخت" ناموفق باشد، جریان کنترل به وظیفه‌ی "شروع پرداخت" بازگردانده می‌شود که در حین آن وظیفه کاربر مجددا به انتخاب سرویس پرداخت می‌پردازد. با توجه به عملکرد مورد انتظار از برنامه و شرایط گفته‌شده، وظیفه‌ی "شروع پرداخت" پیوند از نوع "یای انحصاری" دارد.

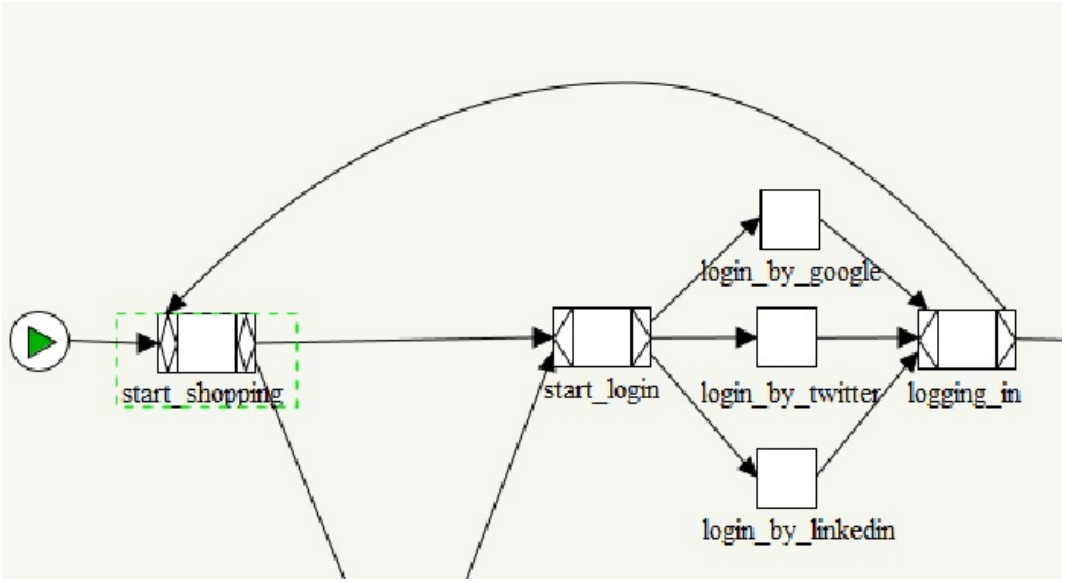
تحلیل بررسی پیوند از نوع "یا" در حلقه

در برنامه‌ی خرید برخط، سناریویی را در نظر بگیرید که طراح برای شروع کار میکروسرویس "شروع خرید"، شرط انجام کار در میکروسرویس "احراز هویت و ورود" قبلی را لحاظ می‌کند و گردش کار مشخص شده در شکل ۵-۳ را تعریف می‌کند. در این گردش کاری وظیفه‌ی "شروع خرید" که دارای پیوند از نوع "یا" است در انتظار مشخص شدن وضعیت جریان‌های ورودی خود است تا پس از آن کار خود را آغاز کند، در حالی که مشخص شدن وضعیت ورودی‌های آن به فعال بودن با نبودن خروجی وظیفه‌ی "احراز هویت و ورود" بستگی دارد؛ در حالی که مشخص شدن فعال یا غیر فعال بودن خروجی این وظیفه به مشخص بودن وضعیت "آغاز خرید" بستگی دارد؛ در نتیجه گردش کار در وضعیت بن‌بست قرار گرفته است که وضعیتی نامطلوب است و دلیل آن نیز قرار گرفتن وظیفه‌ای با پیوند از وضعیت "یا" در حلقه است.

اظهار تولید شده برای گردش کاری در شکل ۵-۳، در زیر آمده است.

```
assert no_or_join_in_loop {
all t: task | t.label = "item_picker" => t not in t.^(flowsInto.nextTask)
}
```

تحلیل بررسی قابل دستیابی بودن میکروسرویس‌ها



شکل ۵-۳: وجود پیوند از نوع "یا" در مدل

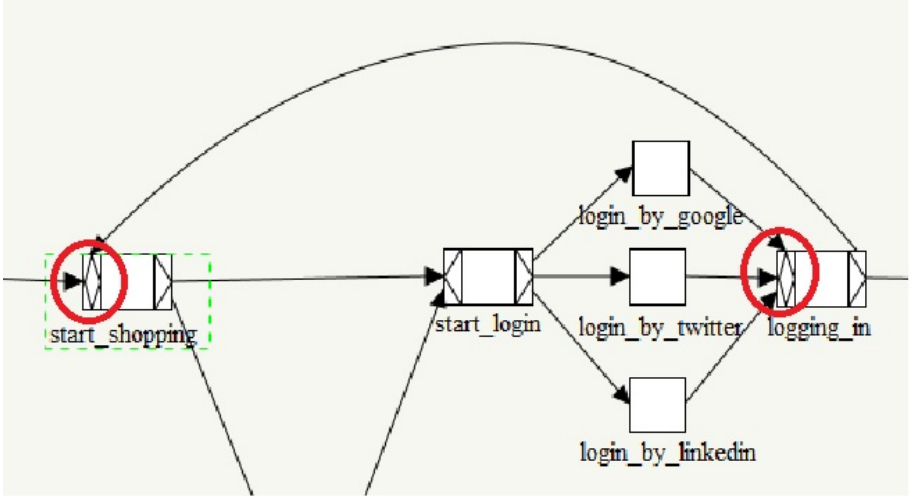
در مثال برنامه‌ی خرید برخط، فرض کنید که طراح در میکروسرویس "تاییدکننده‌ی پرداخت" شرط فعال شدن میکروسرویس "اتمام خرید" را این‌گونه تعریف کند:

```
{
    t1, t2: Task && f: Flow | t1.name = "payment_confirmer" && t2.name =
    "finish_shopping" && f = (t1, t2) =>
    {f.predicate =
        ((s."pay_by_credit_card" == 1 and s."pay_by_credit_card" == 0) or
        ((s."pay_by_credit_card_succesfull" == 0 and
        s."pay_by_credit_card_succesfull" == 1))
    }
}
```

در حالی که مقدار شرط صحیح این‌گونه است:

```
{
    t1, t2: Task && f: Flow | t1.name = "payment_confirmer" && t2.name =
    "finish_shopping" && f = (t1, t2) =>
    {f.predicate =
        ((s."pay_by_credit_card" == 1 and s."pay_by_credit_card_succesfull" == 1)
        or ((s."pay_by_paypal" == 1 and s."pay_by_paypal_succesfull" == 1))
    }
}
```

در این صورت هرگز ارزش مسند



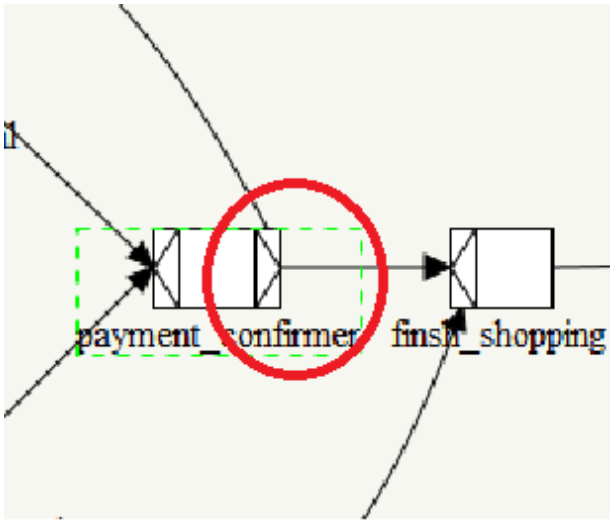
شکل ۵-۴: منتظر بودن دو پیوند از نوع "یا" در مثال خرید برخط

```
((s."pay_by_credit_card" == 1 and s."pay_by_credit_card" == 0) or
((s."pay_by_credit_card_succesfull" == 0 and
s."pay_by_credit_card_succesfull" == 1))
```

برابر "درست" نخواهد شد. در نتیجه هیچ گاه به وظیفه‌ی "اتمام خرید" در زمان اجرای گردش کار نخواهیم رسید و در حقیقت وظیفه‌ی "اتمام خرید" در گردش کاری طراحی شده غیرقابل دسترسی است. اظهار افزوده شده برای یافتن مثال نقض این گزاره به صورت زیر به توصیف گردش کار افزوده می‌شود.

```
assert is_any_state_task_is_token_in_it {
    all t: task | t.label = "finish_shopping" => {
        all s: State | t not in s.token
    }
}
```

تحلیل بررسی منتظر ماندن دو پیوند از نوع "یا" برای یکدیگر مجددا برنامه‌ی خرید برخط را در نظر بگیرید فرض کنید در این نمونه طراح برای نمایش اجرای دو بار "خرید" به صورت پشت سر هم، توسط کاربر گردش کار زیر را طراحی می‌کند، در این گردش کار، میکروسرویس "آغاز خرید"، دارای پیوند از نوع "یا" است و برای آغاز به کار در انتظار مشخص شدن وضعیت جریان‌های ورودی خود است.در حالی که یکی از ورودی‌های آن میکروسرویس "احراز هویت و ورود" است که خود دارای پیوند از نوع "یا" است و مشخص شدن وضعیت ورودی‌های آن به مشخص شدن وضعیت وظیفه‌ی "آغاز خرید"، بستگی دارد. در نتیجه گردش کار دچار بن‌بست می‌شود که وضعیتی نامطلوب است.



شکل ۵-۵: میکروسرویس تایید پرداخت دارای گزاره‌های شرطی برای ایجاد آزمون

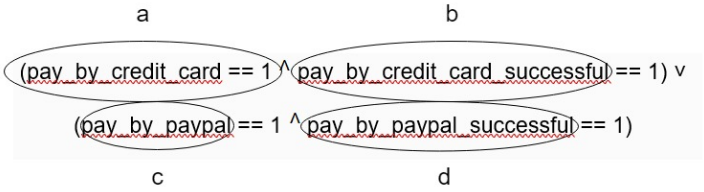
برای پیدا کردن این وضعیت در گردش کار اظهار زیر را به توصیف کلی اضافه می‌کنیم:

```
assert no_two_or_joins_pend_on_each_other {
    all t1, t2: task | t1.label = "start_shopping" && t2.label = "logging_in" => t2 not in t1.^(flowsInto.nextTask)
}
```

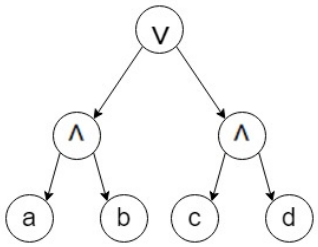
تولید موارد آزمون

در روشی که گفته شد برای تولید موارد آزمون، از مسندهای شرطی در انشعاب‌ها استفاده می‌کنیم، اکنون برای مورد مطالعاتی، یعنی برنامه‌ی خرید برخط، به تولید موارد آزمون می‌پردازیم.

همان‌طور که در شکل زیر می‌بینید خروج از "تاییدکننده‌ی پرداخت" و ورود به "اتمام خرید" شرط زیر را داراست:



برای تولید موارد آزمون بر اساس معیار پوشش گزاره‌ی فعال محدود، درخت گزاره‌های این مسند را تشکیل می‌دهیم و آن را حل میکنیم و سپس زوج موارد آزمون را از طبق جدول صحت گزاره‌ها استخراج می‌کنیم.



	a	b	c	d	p	p_a	p_b	p_c	p_d
1	T	T	T	T	T				
2	T	T	T	F	T	T	T		
3	T	T	F	T	T	T	T		
4	T	T	F	F	T	T	T		
5	T	F	T	T	T			T	T
6	T	F	T	F	F		T		T
7	T	F	F	T	F		T	T	
8	T	F	F	F	F		T		
9	F	T	T	T	T			T	T
10	F	T	T	F	F	T			T
11	F	T	F	T	F	T		T	
12	F	T	F	F	F	T			
13	F	F	T	T	T			T	T
14	F	F	T	F	F				T
15	F	F	F	T	F			T	
16	F	F	F	F	F				

طبق جدول صحتی که در بالا آمده است زوج موارد آزمون برای هر کدام از گزاره‌های $\{a, b, c, d\}$ به صورت زیر به دست می‌آید:

- برای a : $\{(۲, ۱۰), (۳, ۱۱), (۴, ۱۲)\}$
- برای b : $\{(۲, ۶), (۳, ۷), (۴, ۸)\}$
- برای c : $\{(۵, ۷), (۹, ۱۱), (۱۳, ۱۵)\}$

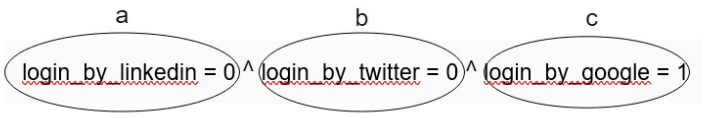
- برای d : {(۵, ۶), (۹, ۱۰), (۱۳, ۱۴)}

برای نمونه دو توصیفی که برای اجرای زوج موارد آزمون (۲و ۶) برای گزاره‌ی b ایجاد می‌شود به صورت زیر است:

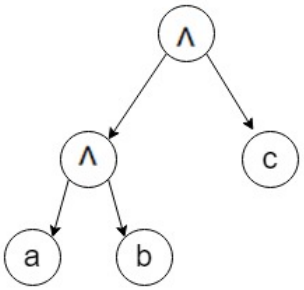
```
fact test {
        all t, t': task | t.label = "payment_confirmer" && t'.label = "finsh_shopping" => {
        one s: State |  t in s.token && s.pay_by_paypal = 1 &&
                        s.pay_by_credit_card_successful = 1 && s.pay_by_credit_card = 1
                        && s.pay_by_paypal_successful = 0 && t' not in s.next.token
        }
}
```

```
fact test {
        all t, t': task | t.label = "payment_confirmer" && t'.label = "finsh_shopping" => {
        one s: State |  t in s.token && s.pay_by_paypal = 1 &&
                        s.pay_by_credit_card_successful = 0 && s.pay_by_credit_card = 1
                        && s.pay_by_paypal_successful = 0 && t' not in s.next.token
        }
}
```

پس از تایید امنیتی (در صورت نیاز)، کاربر می‌تواند روش ورود خود را بر اساس سلیقه‌ی خود انتخاب کند. برای این منظور ۳ متغیر برای کاربر در نظر گرفته شده است و او می‌تواند فقط یکی از آن‌ها را انتخاب کند و بسته به متغیر انتخاب شده یکی از میکروسرویس‌های ورود شروع به کار می‌کنند و هویت کاربر را احراز می‌کنند. برای مثال، خروج از "شروع فرآیند ورود" و ورود به روش "ورود با گوگل" شرط زیر را داراست:



برای این مسند هم، درخت گزاره‌ها را تشکیل می‌دهیم و آن را حل میکنیم و سپس زوج موارد آزمون را از طبق جدول صحت گزاره‌ها استخراج می‌کنیم.



	a	b	c	p	p_a	p_b	p_c
1	T	T	T	T	T	T	T
2	T	T	F	F			T
3	T	F	T	F		T	
4	T	F	F	F			
5	F	T	T	F	T		
6	F	T	F	F			
7	F	F	T	F			
8	F	F	F	F			

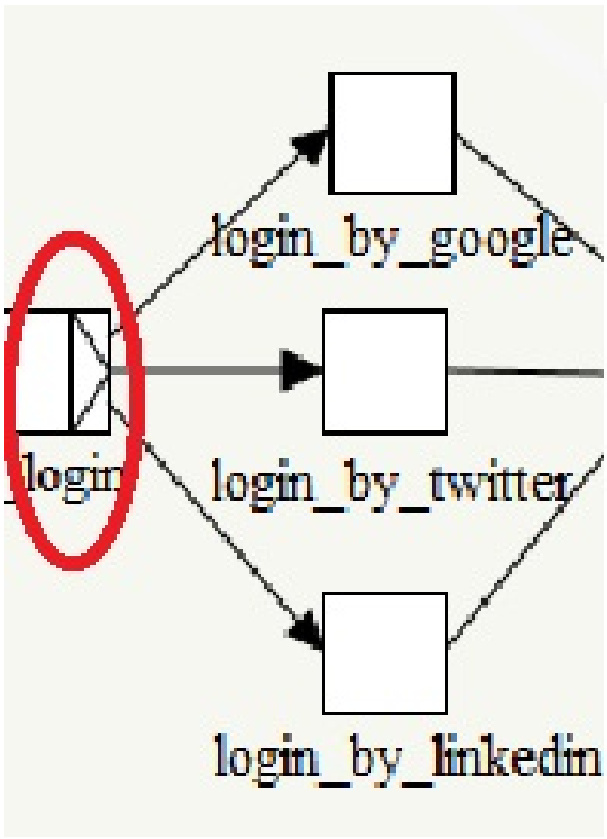
طبق جدول صحتی که در بالا آمده است زوج موارد آزمون برای هر کدام از گزاره‌های $\{a, b, c\}$ به صورت زیر به دست می‌آید:

- برای a: $\{(۱, ۵)\}$
- برای b: $\{(۱, ۳)\}$
- برای c: $\{(۱, ۲)\}$

برای نمونه دو توصیفی که برای اجرای زوج موارد آزمون (۱ و ۵) برای گزاره‌ی a ایجاد می‌شود به صورت زیر است:

```
fact test {
        all t, t': task | t.label = "start_login" && t'.label = "login_by_google" => {
        one s: State |  t in s.token && s.login_by_google = 1 &&  s.login_by_twitter = 0 &&
                        s.login_by_linkedin = 0 && t' not in s.next.token
    }
}
```

```
fact test {
        all t, t': task | t.label = "start_login" && t'.label = "login_by_google" => {
        one s: State |  t in s.token && s.login_by_google = 0 &&  s.login_by_twitter = 0 &&
                        s.login_by_linkedin = 0 && t' not in s.next.token
    }
}
```



شکل ۵-۶: شروع فرآیند ورود دارای گزاره‌های شرطی برای ایجاد آزمون

فصل ۶

ارزیابی

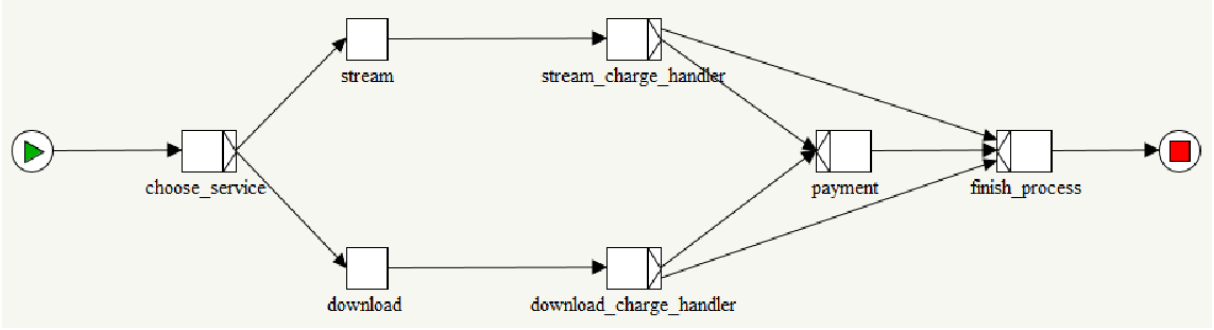
۶-۱ معیارهای ارزیابی

در این بخش به ارزیابی روش ارائه شده در فصل ۴ می‌پردازیم ما برای ارزیابی روش ارائه‌شده‌ی خود از آزمون جهش استفاده کردیم. قصد ما ارزیابی آزمون‌هایی ست که از توصیفات صوری استخراج شده‌اند، همچنین همان‌طور که در روش ارائه‌شده بیان‌شد، موارد آزمون بر اساس معیارپوشش گزاره‌ی فعال محدود از روی مسندهای موجود در جریان‌های خروجی وظایف گردش کار به دست می‌آیند. در نتیجه سعی کردیم با انتخاب عملگرهای جهش مناسب، آن‌ها را بر روی گزاره‌های تشکیل‌دهنده‌ی مسندها اعمال کنیم و موارد آزمون استخراج‌شده را از این نظر که چقدر توانایی کشتن جهش‌یافته‌ها را دارند، ارزیابی می‌کنیم.

در ارزیابی، از دو برنامه با اندازه‌های متفاوت استفاده کرده‌ایم اما از عملگرهای جهش مشابه برای تغییر در آن‌ها استفاده کرده‌ایم. دو برنامه‌ای که برای ارزیابی استفاده شده‌اند به ترتیب دارای ۵ و ۱۰ میکروسرویس هستند و بر روی هر کدام، ۴ معیار جهش اعمال شده است. عملگرهای جهش انتخابی، عبارتند از، جایگزینی عملگرهای رابطه‌ای، جایگزینی عملگرهای شرطی، حذف عملگرهای یگانی[۱] و جایگزینی متغیرهای عددی[۲]

---

[۱] unary
[۲] scalar



شکل ۶-۱: گردش کاری یک برنامه‌ی نمونه‌ی ارائه محتوای چندرسانه‌ای

۶-۲ قسمت ارزیابی برنامه‌ی ارائه‌ی محتوای چندرسانه‌ای

ما برای ارزیابی روش خود، یک برنامه‌ی ارائه‌ی محتوای چندرسانه‌ای را طراحی کردیم، این برنامه نیز مشابه با برنامه‌ی خرید برخط، برگرفته از یک برنامه‌ی واقعی، ایجاد شده است[۳۵]. برنامه‌ی ارائه‌ی محتوای چندرسانه‌ای در دنیای واقع است.

برنامه‌ای که طراحی کرده‌ایم شامل ۵ میکروسرویس است که طبق شکل ۶-۱ برای ارائه‌ی خدمت نهایی، یک گردش کاری را تشکیل دادند. همان طور که در شکل مشخص است، برنامه دارای دو نوع سرویس تماشای برخط[۳] و بارگیری[۴] است. برنامه پس از استفاده‌ی کاربر از هر یک از این سرویس‌ها، کاربر را به اندازه‌ی قیمت محصول ارائه‌شده شارژ می‌کند. اگر موجودی حساب کاربر به اندازه‌ی کافی باشد، مبلغ محصول از حساب کاربر کسز می‌شود و در غیر این صورت، سرویس پرداخت کاربر را به درگاه بانکی می‌برد تا کاربر حساب خود را افزایش اعتبار بدهد، و در نهایت نیز خدمت برنامه به اتمام می‌رسد.

در برنامه‌ی طراحی‌شده، ۵ سرویس تماشای برخط، بارگیری، مدیریت پرداخت تماشای برخط، مدیریت پرداخت بارگیری و پرداخت وجود دارند.

شکل ۶-۱ نشان‌دهنده‌ی گردش کاری این برنامه است.

عملگرهای جهشی که برای ارزیابی موارد آزمون استخراج‌شده به کار رفته‌اند، جایگزینی عملگرهای رابطه‌ای، جایگزینی عملگرهای شرطی، حذف عملگرهای یگانی[۵] و جایگزینی متغیرهای عددی هستند. این عملگرهای جهش بر روی مسندهای شرطی نوشته‌شده در خروجی وظایف مدیریت پرداخت تماشای برخط و مدیریت پرداخت بارگیری اِعمال شده‌اند، در جدول ۶-۱ مسند شرطی استفاده شده برای تولید

[۳] streaming
[۴] download
[۵] unary

جهش‌یافته، نمونه مسند شرطی جهش‌یافته و عملگر جهش استفاده‌شده را نمایش می‌دهد. این مسندها در میکروسرویس‌های مدیریت پرداخت تماشای برخط و مدیریت پرداخت بارگیری استفاده شده‌اند.

با اعمال عملگرهای جهش بر روی مسندهای شرطی و تکرار اجرای آزمون‌ها . درصد از آزمون‌ها به شکست انجامید. در جدول ۶-۲ به تفکیک عملگرها درصد شکست آزمون‌ها بعد از تولید جهش‌یافته‌ها و اجرای دوباره‌ی آزمون‌ها آمده است. می‌توان گفت روش ما کارایی بالایی دارد و فقط در عملگر حذف عملگرهای یگانی کارایی خوبی ندارد و باعث شکست خوردن موارد آزمون استخراج‌شده نشده‌است.

۶-۳ قسمت ارزیابی برنامه‌ی خرید برخط

برنامه‌ای که در این قسمت روش ارائه‌ی شده‌ی خود را بر روی آن ارزیابی می‌کنیم همان برنامه‌ای است که در فصل ۴ گفتیم است.

برنامه‌ای که گفته‌شد برنامه‌ی خرید برخط است و شامل میکروسرویس‌های بررسی‌کننده‌ی امنیتی درخواست‌ها، ورود با استفاده از گوگل، ورود با استفاده از توئیتر، ورود با استفاده از لینکدین، سرویس خرید (انتخاب موارد خرید)، پرداخت با کارت اعتباری، پرداخت با پی‌پل و تایید پرداخت است. شکل ۵-۱ نشان‌دهنده‌ی گردش کاری این برنامه است.

عملگرهای جهشی که برای ارزیابی موارد آزمون استخراج‌شده به کار رفته‌اند، جایگزینی عملگرهای رابطه‌ای، جایگزینی عملگرهای شرطی، حذف عملگرهای یگانی و جایگزینی متغیرهای عددی هستند. این عملگرهای جهش بر روی مسندهای شرطی نوشته‌شده در خروجی وظایف انتخاب روش ورود و بررسی‌کننده‌ی امنیتی اِعمال شده‌اند، در جدول ۶-۳ مسند شرطی استفاده شده برای تولید جهش‌یافته، نمونه مسند شرطی جهش‌یافته و عملگر جهش استفاده‌شده را نمایش می‌دهد.

با اعمال عملگرهای جهش بر روی مسندهای شرطی و تکرار اجرای آزمون‌ها ۱۵/۸۲ درصد از آزمون‌ها به شکست انجامید. در جدول ۶-۴ به تفکیک عملگرها درصد شکست آزمون‌ها بعد از تولید جهش‌یافته‌ها و اجرای دوباره‌ی آزمون‌ها آمده است. می‌توان گفت روش ما در این ارزیابی هم کارایی بالایی دارد.

همچنین از نظر کارایی زمانی روش ارائه‌شده را بررسی کردیم. برای بررسی کارایی زمانی نیز بر روی دو نمونه‌ی گفته‌شده یعنی برنامه‌های خرید برخط و ارائه‌ی محتوای چندرسانه‌ای روش ارائه‌شده را آزمایش کردیم. نتایج جدول ۶-۵ حاصل شد. می‌توان گفت که روش اارائه‌شده در بحث مربوط به کارایی زمانی همچنان می‌تواند بهتر شود و در حال حاضر در مقایسه با روش‌های قبلی ارائه‌شده کارایی خوبی ندارد[۲۴].

جدول ۶-۱: جهش‌یافته‌های برنامه‌ی ارائه‌ی محتوای چندرسانه‌ای

مسند شرطی اصلی	نمونه مسند جهش‌یافته	عملگر جهش
$((fee > customer\_credit \&\& service\_type = "stream") \|\| is\_free = ۱)$	$((fee < customer\_credit \&\& service\_type = "stream") \|\| is\_free = ۱)$	ROR
	$((fee > customer\_credit \&\& service\_type = "stream" \&\& is\_free = ۱)$	COR
	$((fee > customer\_credit \&\& service\_type = "stream") \|\| \sim is\_free = ۱)$	UOD
	$((fee > customer\_credit \&\& service\_type = "stream") \|\| customer\_credit = ۱)$	SVR
$((fee > customer\_credit \&\& service\_type = "download") \|\| is\_free = ۱)$	$((fee > customer\_credit \&\& service\_type = "download") \|\| is\_free > ۱)$	ROR
	$((fee > customer\_credit \|\| service\_type = "download" \|\| is\_free = ۱)$	COR
	$((fee > customer\_credit \&\& \sim (service\_type = "download")) \|\| is\_free = ۱)$	UOD
	$((fee > is\_free \&\& service\_type = "download") \|\| is\_free = ۱)$	SVR

جدول ۶-۲: درصد آزمون‌های شکست‌خورده بعد از تولید جهش‌یافته‌ها در برنامه‌ی ارائه‌ی محتوای چندرسانه‌ای

عملگر جهش	درصد آزمون‌های شکست‌خورده بعد از تولید جهش‌یافته‌ها
ROR	۱۸/۷۵
COR	۱۲/۵
UOD	۰
SVR	۱۲/۵

جدول ۶-۳: جهش‌یافته‌های برنامه‌ی خرید برخط

عملگر جهش	نمونه مسند جهش‌یافته	مسند شرطی اصلی
ROR	$(((login\_by\_linkedin > ۱) \&\& (login\_by\_twitter = ۰)) \&\& and(login\_by\_google = ۰))$	$(((login\_by\_linkedin = ۱) \&\& (login\_by\_twitter = ۰)) \&\& (login\_by\_google = ۰))$
COR	$(((login\_by\_linkedin = ۱) \|\| (login\_by\_twitter = ۰)) \&\& (login\_by\_google = ۰))$	
UOD	$(((login\_by\_linkedin = ۱) \&\& (login\_by\_twitter = ۰)) \&\& (login\_by\_google \neq ۰))$	
SVR	$(((login\_by\_linkedin = ۱) \&\& (login\_by\_twitter = ۰)) \&\& (login\_by\_linkedin = ۰))$	

جدول ۶-۴: درصد آزمون‌های شکست‌خورده بعد از تولید جهش‌یافته‌ها در برنامه‌ی خرید برخط

عملگر جهش	درصد آزمون‌های شکست‌خورده بعد از تولید جهش‌یافته‌ها
ROR	۲۰
COR	۱۳/۳
UOD	۶/۶۷
SVR	۲۳/۳

جدول ۶-۵: ارزیابی کارایی روش ارائه‌شده از نظر زمانی

برنامه‌ی تحت آزمون	تعداد آزمون‌ها	تعداد میکروسرویس‌های برنامه	زمان میانگین اجرای کل آزمون‌ها و ارائه‌ی نتیجه بر حسب ثانیه
برنامه‌ی ارائه‌ی محتوای چندرسانه‌ای	۱۶	۶	۸۰
برنامه‌ی خرید برخط	۳۰	۱۲	۴۸۰

Bibliography

[1] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina. *Microservices: Yesterday, Today, and Tomorrow*, pages 195–216. Springer International Publishing, 2017.

[2] P. Ammann and J. Offutt. *Introduction to Software Testing*. Cambridge University Press, USA, 2nd edition, 2017.

[3] N. Sam. *Building Microservices*. O'Reilly Media, Inc., 2nd edition, 2015.

[4] S. Software. Why you can't talk about microservices without mentioning netflix, 2015.

[5] H. Zhou, M. Chen, Q. Lin, Y. Wang, X. She, S. Liu, R. Gu, B. C. Ooi, and J. Yang. Overload control for scaling wechat microservices. SoCC '18. Association for Computing Machinery, 2018.

[6] M. Waseem, P. Liang, G. Márquez, and A. D. Salle. Testing microservices architecture-based applications: A systematic mapping study. In *2020 27th Asia-Pacific Software Engineering Conference (APSEC)*, pages 119–128, 2020.

[7] R. Mark. *Microservices vs. Service-Oriented Architecture*. O'Reilly Media, Inc., 1st edition, 2016.

[8] P. Bourque and R. E. Fairley, editors. *SWEBOK: Guide to the Software Engineering Body of Knowledge*. IEEE Computer Society, version 3.0 edition, 2014.

[9] I. Beschastnikh, P. Wang, Y. Brun, and M. D. Ernst. Debugging distributed systems. *Commun. ACM*, 59(8):32–37, jul 2016.

[10] A. Hofstede, W. van der Aalst, M. Adams, and N. Russell. *Modern Business Process Automation: YAWL and Its Support Environment*. Springer Publishing Company, Incorporated, 1st edition, 2009.

[11] M. Rivadeh and S. Mirian Hosseinabadi. Formal translation of yawl workflow models to the alloy formal specifications: a testing application. 11 2022.

[12] D. Jackson. *Software Abstractions: Logic, Language, and Analysis*. The MIT Press, 2012.

[13] J. Lewis and M. Fowler. Microservices, a definition of this new architectural term, 2014.

[14] M. Mazzara and S. Govoni. A case study of web services orchestration. volume 3454, pages 1–16, 04 2005.

[15] C. Peltz. Web services orchestration and choreography. *Computer*, 36(10):46–52, 2003.

[16] A. AGUIRRE, G. BARTHE, M. GABOARDI, D. GARG, and P.-Y. STRUB. A relational logic for higher-order programs. *Journal of Functional Programming*, 29, 2019.

[17] J. M. Spivey. *The Z Notation: A Reference Manual*. Prentice-Hall, Inc., USA, 1989.

[18] J. Oetsch, M. Prischink, J. Pührer, M. Schwengerer, and H. Tompits. On the small-scope hypothesis for testing answer-set programs. *13th International Conference on the Principles of Knowledge Representation and Reasoning, KR 2012*, pages 43–53, 01 2012.

[19] D. Jackson. Alloy: A language and tool for exploring software designs. *Commun. ACM*, 62(9):66–76, aug 2019.

[20] B. Beizer. *Software Testing Techniques (2nd Ed.)*. Van Nostrand Reinhold Co., USA, 1990.

[21] R. A. Silva, S. do Rocio Senger de Souza, and P. S. Lopes de Souza. A systematic review on search based mutation testing. *Information and Software Technology*, 81:19–35, 2017.

[22] A. Roman and M. Mnich. Test-driven development with mutation testing – an experimental study. *Software Quality Journal*, 29:1–38, 03 2021.

[23] J. G. Quenum and S. Aknine. Towards executable specifications for microservices. In *2018 IEEE International Conference on Services Computing (SCC)*, pages 41–48, 2018.

[24] S.-P. Ma, C.-Y. Fan, Y. Chuang, W.-T. Lee, S.-J. Lee, and N.-L. Hsueh. Using service dependency graph to analyze and test microservices. In *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, volume 02, pages 81–86, 2018.

[25] V. Heorhiadi, S. Rajagopalan, H. Jamjoom, M. K. Reiter, and V. Sekar. Gremlin: Systematic resilience testing of microservices. In *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*, pages 57–66, 2016.

[26] J. Lotz, A. Vogelsang, O. Benderius, and C. Berger. Microservice architectures for advanced driver assistance systems: A case-study. In *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*, pages 45–52, 2019.

[27] M. J. Kargar and A. Hanifizade. Automation of regression test in microservice architecture. In *2018 4th International Conference on Web Research (ICWR)*, pages 133–137, 2018.

[28] M. Großmann and C. Ioannidis. Continuous integration of applications for onos. In *2019 IEEE Conference on Network Softwarization (NetSoft)*, pages 213–217, 2019.

[29] A. Camargo, I. Salvadori, R. Mello, and F. Siqueira. An architecture to automate performance tests on microservices. pages 422–429, 11 2016.

[30] S. Sharma, N. Uniyal, B. Tola, and Y. Jiang. On monolithic and microservice deployment of network functions. In *2019 IEEE Conference on Network Softwarization (NetSoft)*, pages 387–395, 2019.

[31] M. Camilli, C. Bellettini, L. Capra, and M. Monga. *A Formal Framework for Specifying and Verifying Microservices Based Process Flows*, pages 187–202. 02 2018.

[32] H. Schulz, T. Angerstein, D. Okanović, and A. van Hoorn. Microservice-tailored generation of session-based workload models for representative load testing. In *2019 IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 323–335, 2019.

[33] P. Valderas, V. Torres, and V. Pelechano. A microservice composition approach based on the choreography of bpmn fragments. *Information and Software Technology*, 127:106370, 06 2020.

[34] A. Nadeem and M. Malik. A case for microservices orchestration using workflow engines. pages 6–10, 10 2022.

[35] n. ci. 30nama multimedia app. available:, 2021.

[36] R. Pietrantuono, S. Russo, and A. Guerriero. Run-time reliability estimation of microservice architectures. In *2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE)*, pages 25–35, 2018.

[37] M. T. Wynn, W. M. P. van der Aalst, A. H. M. ter Hofstede, and D. Edmond. Verifying workflows with cancellation regions and or-joins: An approach based on reset nets and reachability analysis. In *Business Process Management*, pages 389–394, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[38] W. Aalst. Verification of workflow nets. pages 407–426, 01 1997.

واژه‌نامه

الف

انشعاب ..... split
الوی ..... Alloy
انتزاع ..... abstraction
امضا ..... signature
اتم ..... atom
اظهار ..... assertion
انتقال ..... transition
ایمنی ..... safety
امنیت ..... security
آزمون مدل‌ـرانه ..... model-driven testing
اشکال زدایی ..... debugging
انتشار ..... propagation
آزمون خودکار ..... automated testing
آزمون مبتنی بر مدل ..... model-based testing
آزمون پذیرش ..... acceptance testing
آزمون واحد ..... unit testing
استقرار ..... deployment
آزمون رگرسیون ..... regression testing
آنتولوژی ..... ontology
اشتقاق ..... derivation
انگاره ..... paradigm

ب

برخط ..... online
بازنشانی ..... reset
بیت ..... bit
بن‌بست ..... deadlock
بارگیری ..... download

پ

پیوند ..... join
پرسمان ..... query
پوشش ..... coverage
پیچیدگی ..... complexity
پیشوند ..... prefix
پسوند ..... postfix
پوشش گزاره‌ی فعال محدود restricted acive clause coverage (RACC)
پودمان ..... module

ت

توکن ..... Token
تصمیم‌پذیر ..... decidable
توسعه یافته ..... extended
تقسیم‌بندی ..... partition
توری ..... mesh
توزیع‌شده ..... distributed

تحویل مداوم ..... continuous delivery
تماشای برخط ..... streaming

ج

جفت‌شدگی ..... coupling
جعبه سیاه ..... black box
جریان ..... flow
جهش ..... mutation
جهش‌یافته ..... mutant
جایگزینی ..... replacement

چ

چندرسانه‌ای ..... multimedia

ح

حقیقت ..... fact
حریصانه ..... greedy

خ

خوشه ..... cluster
خطی ..... linear

د

داده ..... data
درستی ..... correctness
دودویی ..... binary
دستور زبان ..... grammer

ر

رأس ..... vertex
رسمی ..... formal
رشته ..... string
ریاضیات گسسته ..... discrete mathematics

ز

زیرامضا ..... subsignature
زمینه ..... theme

س

سرایت ..... infection
سلسه‌مراتبی ..... hierarchichal

ش

شرط ورودی ..... input condition
شرط خروجی ..... output condition
شیء ..... object
شبیه‌سازی ..... simulation

ص

صدق‌پذیری ..... satisfiability
صحیح ..... sound
صوری ..... formal

ط

طراحی ..... design

ع

عملکرد ..... performance
عددی ..... scalar

عملگر ..... operator

غ

غلبه ..... dominate

ف

فاصله ..... distance

فضا ..... space

ق

قابلیت اطمینان ..... reliability

قابل دسترس بودن ..... reachability

قابلیت نگه‌داری ..... maintainability

قابل آشکار شدن ..... revealable

ک

کاهش ..... reduce

کارایی ..... efficiency

گ

گراف ..... graph

گزاره ..... clause

گردش کار ..... workflow

م

مجموعه ..... set

منطقه‌ی لغو ..... cancellation region

منطق رابطه‌ای ..... relational logic

مثبت نادرست ..... false positive

محدودیت‌های شکست تقارن . symmetry breaking constraints

مبتنی بر معیار ..... criteria based

مبتنی بر انسان ..... human based

خطا/شکست ..... fault/failure

منطق ..... logic

مسند ..... predicate

معماری ..... acrchitecture

موجودیت ..... entity

موتور ..... engine

متغیر ..... variable

ن

نحو ..... syntax

نوع ..... type

نقطه‌ی مرکزی ..... center point

نیازمندی آزمون ..... test requirement

ناهمگام ..... asynchronous

و

و ..... AND

وظیفه ..... task

ویژگی‌ها ..... properties

هـ

همگام‌سازی ..... Synchronization

ی

یال ..... edge

یا ..... OR

یای انحصاری ..... XOR

یکپارچه‌سازی مستمر ..... continuous integration

یگانی ..... unary

پیوست آ

مطالب تکمیلی

ترجمه‌ی مدل گردش کاری برنامه‌ی خرید بر خط به زبان الوی

```
   /* Impose an ordering on the State. */
open util/ordering[State]
sig State {
	token, n_token: some Object1,
	login_by_twitter: lone Int,
	pay_by_credit_card: lone Int,
	login_by_google: lone Int,
	need_security_checking: lone Int,
	security_check_pass: lone Int,
	payment_successful: lone Int,
	login_by_linkedin: lone Int,
	login_successful: lone Int,
	pay_by_paypal: lone Int,
}

// formal defintion of model objects
abstract sig Object1
{
	flowsInto: set Flows,
	status: String
}



one sig input_condition	extends Object1 {}
one sig output_condition	extends Object1 {}


sig task extends Object1
{
	split, join, label: String,
	last_deactive_task: Object1,
	cancellation_reigon_objects: Object1
}


fact{
	all o: Object1 | o.status = "Activated" || o.status = "Deactive" || o.status = "N/A"
}


sig Flows {
	predicate: lone Boolean,
	nextTask: one Object1}//formal definition of boolean

sig Boolean {
	value: Int
}
```

```
fact {
	all b: Boolean | b.value = 0 || b.value = 1
}


//formal definition of states of the model
//formal definition of the initial state
fact {
	all s: State {s.token + s.n_token = Object1 && s.token & s.n_token = none}
}

fact {
	first.token = input_condition && first.n_token = Object1 - input_condition
			&& first.token.status = "Activated"
}


 //formal definition of the final state
fact {
	last.token = output_condition
}


fact {
	no s: State | output_condition in s.token && s != last
}



 // task with None split have no predicate
fact {
	all t: task{
		t.split = "None" => #t.flowsInto.predicate = 0
	}
}


 // Definition of task statuses
 fact {
	one o: input_condition | o.status = "Activated"
}


fact{
	all s: State {
		all x: s.token {
			(x.join = "None" && x != input_condition &&
				(flowsInto.nextTask.x.split = "None"
				|| flowsInto.nextTask.x = input_condition)) =>
```

```
                    x.status = flowsInto.nextTask.x.status
        }
    }
}


fact {
    all t: task {
            flowsInto.nextTask.t.status = "Deactive" && t.join = "None" =>
                t.status = "Deactive"
        }
}




fact {
    all s: State, s': s.next {
        all x: s.token {
            x.split = "Xor" => {one f: x.flowsInto | f.predicate.value = 1 &&
                f.nextTask in s'.token && f.nextTask.status = "Activated" &&
                all f': (x.flowsInto  -  f)| f'.nextTask.status = "Deactive" }
        }
    }
}


fact{
    all s: State, s': s.next {
        all x: s.token {
            x.split = "And" => {all f: x.flowsInto | f.predicate.value = 1 &&
            f.nextTask in s'.token
            && f.nextTask.status = "Activated" }
        }
    }
}


fact{
    all s: State, s': s.next {
        all x: s.token {
            x.split = "Or" => {some f: x.flowsInto | f.predicate.value = 1 &&
                                f.nextTask in s'.token
                                 && f.nextTask.status = "Activated" &&
                        all f': (x.flowsInto  -  f)| f'.nextTask.status = "Deactive" }
        }
```

```
}

// Definition of Cancellation Reigons


fact {
	all s: State, s':s.next {
		all x: s.token {
			all t:x.flowsInto.nextTask | t in s'.token =>
				{no o: x.cancellation_reigon_objects | o in s'.token } &&
				{all o: x.cancellation_reigon_objects { all f: o.flowsInto |
									f.nextTask not in s'.token }}
		}
	}
}


 // Definition of the None split gateway behavior
fact {
	all s: State, s': s.next {
		all x: s.token {
			x.split = "None" => ( (x.flowsInto.nextTask in s'.token || x in s'.token) &&
					!(x.flowsInto.nextTask in s'.token && x in s'.token))
		}
	}
}

// Definition of the And split gateway behavior
fact {
	all s: State, s': s.next {
		all x: s.token {
			x.split = "And" =>((s'.token = x || all y: x.flowsInto.nextTask | y in s'.token)
			 && !(s'.token = x && all y: x.flowsInto.nextTask | y in s'.token))
		}
	}
}


// Definition of the Xor split gateway behavior
fact {
	all s: State, s': s.next {
		all x: s.token{x.split = "Xor" =>((s'.token = x || all f: x.flowsInto |
				f.predicate.value = 1 =>
					{one t: f.nextTask | t in s'.token &&
						no z: f.nextTask | z in s'.token   && z != t})
				&& !(s'.token = x && all f: x.flowsInto |
						f.predicate.value = 1 =>
							{one t: f.nextTask | t in s'.token &&
							 no z: f.nextTask | z in s'.token && z != t}))
		}
```

```
    }
}

// Definition of the Xor split gateway behavior
fact {
    all s: State, s': s.next {
        all x: s.token{x.split = "Xor" =>((s'.token = x || all f: x.flowsInto |
                                f.predicate.value = 1 =>
                                    {one t: f.nextTask | t in s'.token &&
                                                no z: f.nextTask | z in s'.token    && z != t})
                                && !(s'.token = x && all f: x.flowsInto |
                                                f.predicate.value = 1 =>
                                                    {one t: f.nextTask | t in s'.token &&
                                                     no z: f.nextTask | z in s'.token && z != t}))
        }
    }
}

// Definition of the Or split gateway behavior
fact {
    all s: State, s': s.next {
        all x: s.token {
            x.split = "Or" => ((x in s'.token || some y: x.flowsInto.nextTask | y in s'.token)
                               &&
                              !(x in s'.token && some y: x.flowsInto.nextTask | y in s'.token))
        }
    }
}

fact all_splits_flows_intos_has_predicate {
    all t:task | (t.split = "Xor" || t.split = "And" || t.split = "Or") => all f:t.flowsInto | #f.predicate = 1
}

fact xor_split_definition {
    all s: State { all x: s.token | x.split = "Xor" => one f: x.flowsInto | f.predicate.value = 1
    }
}

//Definition of the None join behavior
fact {
    all s: State, s': s.next {
        all x: s'.token {
            x.join = "None" =>((flowsInto.nextTask.x in s.token || x in s.token) &&
                               !(flowsInto.nextTask.x in s.token && x in s.token))
        }
```

```
    }
}

// Definition of the And join behavior
fact
{
all s: State, s': s.next
{
all x: s'.token
{
x.join = "And" =>      x.status = "Activated" && (
(all y: flowsInto.nextTask.x | y in s.token || x in s.token) &&
!(all y: flowsInto.nextTask.x | y in s.token && x in s.token)
                        )
}
}
}
// Definition of the Xor join gateway behavior
fact{
    all s: State, s': s.next{
        all x: s'.token{
            x.join = "Xor" => x.status = "Activated" && ((one y: flowsInto.nextTask.x |
                                                    y in s.token || x in s.token) &&
                                                !(one y: flowsInto.nextTask.x | y in s.token &&
                                                                        x in s.token))
        }
    }
}


// Definition of the Or join gateway behavior
fact{
    all s: State, s': s.next{
        all x: s'.token{
            x.join = "Or" =>((some y: flowsInto.nextTask.x | y in s.token || x in s.token) &&
                                !(some y: flowsInto.nextTask.x | y in s.token && x in s.token))
        }
    }
}


fact{
    all s: State, s': s.next{
        all x: s'.token{
            x.join = "Or" => x.status = "Activated" && all y: flowsInto.nextTask.x |
                                                    y.status != "N/A"
        }
    }
}
```

```
//Output condition has no output
fact {
	#output_condition.flowsInto = 0
}

// Input condition next state tokens are all input condition next tasks
fact {
	first.next.token = input_condition.flowsInto.nextTask
}

// Input condition Flows has no predicate
fact {
	#first.token.flowsInto.predicate = 0
}

// Two distinct Object1 has no same Flow
fact {
	all o1, o2: Object1 | all f1: o1.flowsInto, f2: o2.flowsInto | o1 != o2 => f1 != f2
}

// To prevent state token jump forward or backward
fact {
	all s: State, s': s.next | all t': Object1 | t' in s'.token =>
		some t: Object1 | t in s.token && (t' = t || t in flowsInto.nextTask.t')
}


// Dynamic Part
fact {
	all s: State | all i: input_condition |  i in s.token => {
		one t: Object1 | t in i.flowsInto.nextTask && t.label = "start_shopping" &&
			t.join = "Xor" && t.split = "Xor" && t in s.next.token
	}
}

fact {
	all t: Object1 | (t.label = "start_shopping" || t.label = "security_checker") => {
		one t2: Object1 | t2 in t.flowsInto.nextTask && t2.label = "start_login" &&
					t2.join = "Xor" && t2.split = "Xor"
	}
}


fact {
	all t: task | t.label = "start_login" => {
		{one t0: Object1 | t0 in t.flowsInto.nextTask && t0.label = "login_by_google"
				&& t0.split = "None" && t0.join = "None"
		} &&
		{ one t1: Object1 | t1 in t.flowsInto.nextTask && t1.label = "login_by_twitter"
```

```
                && t1.split = "None" && t1.join = "None"
            } &&
            { one t2: Object1 | t2 in t.flowsInto.nextTask && t2.label = "login_by_linkedin"
                && t2.split = "None" && t2.join = "None"
            }
        }
}

fact {
        all s: State, s': s.next | all t: task, t':Object1 | t in s.token && t.label = "start_login"
            && t' in s'.token && t'.label = "login_by_twitter" => { one f: t.flowsInto |
                f.nextTask.label = "login_by_twitter" && f.predicate.value = 1 &&
                    (((s.login_by_linkedin = 0) && (s.login_by_twitter = 1)) &&
                    (s.login_by_google = 0))
        }
}


fact {
        all s: State, s': s.next | all t: task, t':Object1 | t in s.token && t.label = "start_login"
            && t' in s'.token && t'.label = "login_by_linkedin" => { one f: t.flowsInto |
                f.nextTask.label = "login_by_linkedin" && f.predicate.value = 1 &&
                (((s.login_by_linkedin = 1) && (s.login_by_twitter = 0)) && (s.login_by_google = 0))
        }
}


fact {
        all s: State, s': s.next | all t: task, t':Object1 | t in s.token && t.label = "start_login" &&
            t' in s'.token && t'.label = "login_by_google" => { one f: t.flowsInto |
                f.nextTask.label = "login_by_google" && f.predicate.value = 1
        }
}

fact {
        all t: Object1 | (t.label = "pay_by_paypal" || t.label = "pay_by_credit_card") =>{
            one t2: Object1 | t2 in t.flowsInto.nextTask && t2.label = "payment_confirmer" &&
                t2.join = "Xor" && t2.split = "Xor"
        }
}


fact {
        all t: task | t.label = "payment_confirmer" => {
            { one t0: Object1 | t0 in t.flowsInto.nextTask && t0.label = "finsh_shopping" &&
                t0.split = "None" && t0.join = "Xor" } && { one t1: Object1 | t1 in t.flowsInto.nextTask
                    && t1.label = "start_payment" && t1.split = "Xor" && t1.join = "Xor" }
        }
}
```

```
fact {
	all s: State, s': s.next | all t: task, t':Object1 | t in s.token && t.label = "payment_confirmer"
		&& t' in s'.token && t'.label = "finsh_shopping" => { one f: t.flowsInto |
			f.nextTask.label = "finsh_shopping" && f.predicate.value = 1 &&
			s.payment_successful = 1
	}
}


fact {
	all s: State, s': s.next | all t: task, t':Object1 | t in s.token && t.label = "payment_confirmer"
	&& t' in s'.token && t'.label = "start_payment" => {
		one f: t.flowsInto | f.nextTask.label = "start_payment" && f.predicate.value = 1 }
}

fact {
	all t: task | t.label = "item_picker" => { one t1: task | t1 = t.flowsInto.nextTask &&
		t1.label = "start_payment" && t1.split = "Xor" && t1.join = "Xor"}
}

fact {
	all t: Object1 | (t.label = "payment_confirmer" || t.label = "security_checker") => {
		one t2: Object1 | t2 in t.flowsInto.nextTask && t2.label = "finsh_shopping" &&
			t2.join = "Xor" && t2.split = "None"
	}
}


fact {
	all t: task | t.label = "finsh_shopping" => {
		one t1: Object1 | t1 = t.flowsInto.nextTask && t1= output_condition
	}
}

fact {
	all t: Object1 | (t.label = "login_by_google" || t.label = "login_by_twitter" ||
		t.label = "login_by_linkedin") =>{ one t2: Object1 | t2 in t.flowsInto.nextTask &&
			t2.label = "logging_in" && t2.join = "Xor" && t2.split = "Xor"
	}
}


fact {
	all t: task | t.label = "logging_in" => {{ one t0: Object1 | t0 in t.flowsInto.nextTask &&
		t0.label = "start_shopping" && t0.split = "Xor" && t0.join = "Xor" } &&
			{ one t1: Object1 | t1 in t.flowsInto.nextTask && t1.label = "item_picker" &&
				t1.split = "None" && t1.join = "None" }
	}
}
```

```
fact {
	all s: State, s': s.next | all t: task, t':Object1 | t in s.token && t.label = "logging_in" &&
		t' in s'.token && t'.label = "start_shopping" => { one f: t.flowsInto |
			f.nextTask.label = "start_shopping" && f.predicate.value = 1
	 }
}


fact {
	all s: State, s': s.next | all t: task, t':Object1 | t in s.token && t.label = "logging_in" &&
		 t' in s'.token && t'.label = "item_picker" => { one f: t.flowsInto |
			f.nextTask.label = "item_picker" && f.predicate.value = 1 && s.login_successful = 1
	}
}

fact {
	all t: Object1 | (t.label = "item_picker" || t.label = "payment_confirmer") => {
		one t2: Object1 | t2 in t.flowsInto.nextTask && t2.label = "start_payment" &&
			t2.join = "Xor" && t2.split = "Xor"
	}
}


fact {
	all t: task | t.label = "start_payment" => {
		{ one t0: Object1 | t0 in t.flowsInto.nextTask && t0.label = "pay_by_credit_card" &&
			t0.split = "None" && t0.join = "None" } &&
		{ one t1: Object1 | t1 in t.flowsInto.nextTask && t1.label = "pay_by_paypal"&&
			t1.split = "None" && t1.join = "None" }
	}
}

fact {
	all s: State, s': s.next | all t: task, t':Object1 | t in s.token && t.label = "start_payment" &&
		t' in s'.token && t'.label = "pay_by_credit_card" => {
		 one f: t.flowsInto | f.nextTask.label = "pay_by_credit_card" && f.predicate.value = 1
	}
}


fact {
	all s: State, s': s.next | all t: task, t':Object1 | t in s.token && t.label = "start_payment" &&
		t' in s'.token && t'.label = "pay_by_paypal" => {
			one f: t.flowsInto | f.nextTask.label = "pay_by_paypal" && f.predicate.value = 1
				&& ((s.pay_by_paypal = 1) && (s.pay_by_credit_card = 0))
	}
}
```

```
fact {
	all t: task | t.label = "pay_by_credit_card" => {
		one t1: task | t1 = t.flowsInto.nextTask && t1.label = "payment_confirmer" &&
			t1.split = "Xor" && t1.join = "Xor"
	}
}

fact {
	all t: task | t.label = "login_by_twitter" => {
		one t1: task | t1 = t.flowsInto.nextTask && t1.label = "logging_in" &&
			t1.split = "Xor" && t1.join = "Xor"
	}
}

fact {
	all t: task | t.label = "security_checker" => {
		{ one t0: Object1 | t0 in t.flowsInto.nextTask && t0.label = "finsh_shopping"
			&& t0.split = "None" && t0.join = "Xor" } && { one t1: Object1 |
				t1 in t.flowsInto.nextTask && t1.label = "start_login" && t1.split = "Xor"
				&& t1.join = "Xor"
		}
	}
}

fact {
	all s: State, s': s.next | all t: task, t':Object1 | t in s.token && t.label = "security_checker" &&
		t' in s'.token && t'.label = "start_login" => { one f: t.flowsInto |
			f.nextTask.label = "start_login" && f.predicate.value = 1 && s.security_check_pass = 1
	}
}


fact {
	all s: State, s': s.next | all t: task, t':Object1 | t in s.token && t.label = "security_checker" &&
		t' in s'.token && t'.label = "finsh_shopping" => { one f: t.flowsInto |
			f.nextTask.label = "finsh_shopping" && f.predicate.value = 1
	}
}


fact {
	all t: Object1 | (t.label = "logging_in" || t = input_condition) =>{
		one t2: Object1 | t2 in t.flowsInto.nextTask && t2.label = "start_shopping" &&
			t2.join = "Xor" && t2.split = "Xor"
	}
}
```

```
fact {
	all t: task | t.label = "start_shopping" => {
		{ one t0: Object1 | t0 in t.flowsInto.nextTask && t0.label = "start_login"
			&& t0.split = "Xor" && t0.join = "Xor" } && { one t1: Object1 |
				t1 in t.flowsInto.nextTask && t1.label = "security_checker" && t1.split = "Xor"
				&& t1.join = "None"
		}
	}
}

fact {
	all s: State, s': s.next | all t: task, t':Object1 | t in s.token && t.label = "start_shopping" &&
		t' in s'.token && t'.label = "security_checker" => { one f: t.flowsInto |
			f.nextTask.label = "security_checker" && f.predicate.value = 1 &&
				s.need_security_checking = 1
	}
}


fact {
	all s: State, s': s.next | all t: task, t':Object1 | t in s.token && t.label = "start_shopping" &&
		t' in s'.token && t'.label = "start_login" => { one f: t.flowsInto |
			f.nextTask.label = "start_login" && f.predicate.value = 1
	}
}

fact {
	all t: task | t.label = "login_by_linkedin" => { one t1: task | t1 = t.flowsInto.nextTask &&
		t1.label = "logging_in" && t1.split = "Xor" && t1.join = "Xor"
	}
}

fact {
	all t: task | t.label = "login_by_google" => { one t1: task | t1 = t.flowsInto.nextTask &&
		t1.label = "logging_in" && t1.split = "Xor" && t1.join = "Xor"}
}

fact {
	all t: task | t.label = "pay_by_paypal" => { one t1: task | t1 = t.flowsInto.nextTask &&
		t1.label = "payment_confirmer" && t1.split = "Xor" && t1.join = "Xor"
	}
}

pred show{}
 run show for 13 but 13 task, 21 Flows
```