معنای عملیاتی برای یک زبان مبتنی براکتور

مرجان سیرجانی
استادیار دانشکده مهندسی برق و کامپیوتر – دانشکده‌های فنی - دانشگاه تهران
پژوهشکده علوم کامپیوتر – پژوهشگاه علوم بنیادی(IPM)
نیلوفر رضوی
دانشجوی کارشناسی ارشد دانشکده مهندسی برق وکامپیوتر – دانشکده‌های فنی - دانشگاه تهران
پژوهشکده علوم کامپیوتر – پژوهشگاه علوم بنیادی(IPM)
الهام موسوی
دانشجوی کارشناسی ارشد دانشکده علوم کامپیوتر – دانشگاه واترلو
علی موقر رحیم‌آبادی
دانشیار گروه مهندسی کامپیوتر - دانشکده مهندسی کامپیوتر - دانشگاه صنعتی شریف
(تاریخ دریافت ۸۳/۲/۱۹، تاریخ دریافت روایت اصلاح شده ۸۵/۵/۱۴ ، تاریخ تصویب ۸۵/۷/۸)

چکیده

باتوجه به رشد روز افزون استفاده از سیستم‌های همروند، داشتن مدلی مناسب برای توصیف این گونه سیستم‌ها امری ضروری است. در این مقاله یک زبان مبتنی براکتور، که زبانی شیء بنیاد و همروند است انتخاب شده و معنای عملیاتی آن به شکل صوری در سیستم گذار پایه بیان گردیده است. به این ترتیب درک و استفاده از زبان و همچنین تشخیص هرگونه ناسازگاری و یا ابهام و نقص در آن آسان تر شده - است. اشیاء واکنشی مستقل از هم، اجزاء تشکیل دهنده‌ی مدل هستند که با تبادل ناهمگام پیام با یکدیگر در ارتباط هستند. ایجاد پویای اشیاء واکنشی و تغییر پویای پیکربندی از ویژگی‌های این زبان است. این زبان به همراه توصیف صوری و قانونمند آن، پایه محکمی برای داشتن یک ابزار جهت وارسی سیستم‌های واکنشی بوده است.

واژه‌های کلیدی : همروندی، سیستم‌های واکنشی، مدل اکتور، سیستم گذار پایه، توصیف و وارسی صوری

مقدمه

توسعه‌ی فراگیر سیستم‌های واکنشی، استفاده از شبکه‌ها برای اشتراک منابع وحل مسائل توزیع شده، رشد فن آوری چند پردازنده‌ای و فراهم آوردن قدرت محاسبه بالا با هزینه‌ی کم از علت‌های مطرح شدن همروندی در محاسبات هستند. برای حل هرچه بهتر مسائل موجود دراین دامنه بایستی از مدلی مناسب و گویا استفاده کرد. در مدلسازی یک سیستم همروند، شیوه ارتباطات و نحوه هماهنگی بین مؤلفه‌های همروند بایستی به خوبی مشخص شود.

مدل اکتور یک مدل شیء بنیاد همروند است[۲]. اکتورها، مؤلفه‌های مستقل و خود شمولی هستند که با یکدیگر دارای برهم کنش هستند. در این مدل، ارتباطات به وسیله‌ی تبادل پیام انجام گرفته و این تبادل پیام به صورت ناهمگام است. هرپیام ارسالی می-تواند شامل آدرس یک اکتور نیز باشد. از این رو توپولوژی ارتباطی به صورت پویا قابل تغییر است. رسیدن پیام‌های ارسالی به مقصد در زمان محدود، دراین مدل تضمین شده است. به عبارت دیگر سیستم
با انصاف[1] است. ایجاد اکتور در این مدل می‌تواند به صورت پویا انجام گیرد.

مدل اکتور ابتدا توسط هیویت[۱۲] به عنوان پایه‌ای برای تولید یک جامعه از عوامل استدلال‌گر[2] پیشنهاد شد. سپس کلینگر [۱۱] یک مدل ریاضی از اکتورها را به صورت نمودارهای وقایع ایجاد کرد که از قواعد پردازش

همروند [۹] پیروی می‌کردند.

ادامه‌ی کار روی این مدل، به طورعمده توسط آقا انجام گرفت[۱][۲][۳][۴][۵]. او ابتدا یک سیستم گذار ساده برای اکتورها تعریف کرد که در آن مشکلات به وجود آمده در اثر بسته فرض کردن سیستم، مورد نظر قرار گرفته‌است[۱]. دراین کار مفاهیم پیکربندی سیستم، اکتورهای خارج از سیستم و اکتورهای دریافت‌کننده‌ی پیام از خارج به شکل صوری تعریف شده‌اند.

در [۳]، کارهای انجام شده تا سال ۱۹۹۱جمع‌بندی شده و ساختار معنای مدل اکتور بیان شده است. در آنجا معنای زبان اکتور به صورت یک سیستم گذار با تعاریف صوری بیان گردیده است. در این سیستم پیکربندی یک سیستم اکتور شامل اکتورها و وظایف درون آن است. هر وظیفه نشانگر پیامی است که فرستاده شده و هنوز پردازش نشده است.

درسال ۱۹۹۷ آقا به همراهی میسون، اسمیت و تالکوت کاری را برای رسیدن به یک پایه صوری قوی برای مدل اکتور انجام دادند [۴] که سپس در [۱۵] و [۲۱] تکمیل شد. زبان اکتوری که در آنها تعریف شده، شکل گسترش یافته‌ی یک زبان تابعی ساده است. پیکربندی سیستم به صورت توزیع شده و باز می‌باشد، بدان معنا که در توصیف یک سیستم اکتور، ارتباط با مؤلفه‌های خارجی به طور صریح در نظرگرفته شده است. معنای این زبان به صورت عملیاتی بیان شده است.

در سالهای اخیر، در طراحی زبان‌های شیءگرای بی‌درنگ، دربرخی موارد از زیرمجموعه‌ای از مدل اکتور استفاده شده است[۱۷][۱۸][۱۹]. جز در کاربردهای بی-درنگ، در موارد دیگر نیز برای ساده‌تر شدن مدل و پیاده-سازی و استفاده‌ی عملی از آن، زبان‌های متفاوتی مبتنی بر مدل اکتور به کار گرفته شده‌اند که در اغلب آنها مدل ازشکل تابعی خود خارج شده است[۲۴][۲۵][۲۶]. این زبان‌ها به صورت رویه‌ای در نظر گرفته شده اند تا استفاده از آنها ساده تر شود.

علاوه بر کارهای انجام گرفته روی مدل اکتور، زبان-های برنامه نویسی شیءگرای همروندی وجود دارند که مبتنی بر این مدل نیستند. از جمله این زبان‌ها می توان از POOL نام برد[۷]. معنای این زبان به گونه‌های مختلف بیان شده است[۶][۸][۲۳]. مدل اکتور هم به منظور برنامه‌سازی همروند و هم به عنوان یک نظریه محاسبات مدلی تأثیرگذار بوده است[۱۶]. زبان مورد بحث دراین مقاله مبتنی بر اکتور بوده و امکان تغییر پویای پیکربندی، تبادل ناهمگام پیام و ایجاد پویای یک اکتور در آن وجود دارد. با این ویژگی از دو الگوی تابعی و امری به کار گرفته شده در مدل‌های مبتنی بر اکتور, الگوی امری انتخاب شده است. برای این انتخاب دلایل مختلفی وجود دارد, برای مثال می توان به [۱۵] مراجعه کرد. محیط خارجی در زبان ما درنظر گرفته نشده است اما افزودن آن، درصورت نیاز و درکاربردهای خاص، تنها با اضافه کردن تعاریف لازم به پیکربندی قابل انجام است. به وسیله‌ی تعمیم زبان، با افزودن مفهوم زمان به آن، می توان آن را برای مدلسازی سیستم‌های بی‌درنگ مناسب ساخت. معنای عملیاتی این مدل به شکل صوری، براساس سیستم گذار پایه [۱۴] بیان شده است.

روش صوری معمولا" از یک زبان توصیف و یک سری روش‌های اثبات برای وارسی صوری تشکیل شده است-[۱۴]. مطالعه و بیان صوری معنای یک مدل یا زبان دارای مزایای بسیاری است. دقیق و مختصر بودن تعریف صوری معنای یک زبان معیار خوبی برای ارزیابی جامعیت مفهومی آن است. اگر مفاهیم پایه‌ای یک زبان و یا نحوه ترکیب آنها با دقت انتخاب نشده باشد، آنگاه بیان معنای برنامه‌های نوشته شده با آن زبان به شکل صوری، به طور جدی با مشکل مواجه می‌شود. همچنین توصیف صوری معنای یک زبان

می‌تواند پایه‌ای برای اثبات درستی یک پیاده‌سازی خاص از آن زبان باشد. علاوه بر موارد ذکر شده, توصیف صوری معنای زبان می‌تواند به عنوان چارچوبی برای یک نظریه صوری وارسی (استدلال در مورد درستی برنامه‌های نوشته شده با آن زبان)، عمل کند[۸][۱۶].

سیستم گذار پایه، ارائه شده توسط منا [۱۴] (و قبل از آن کِلِر [۱۳]) این حسن را دارد که اغلب مدل‌ها و زبان‌های دارای همروندی را می‌توان با آن نمایش داد. به این ترتیب درک معنا و مقایسه‌ی این مدل‌ها براساس صورت بندی[۳] واحدی برای توصیف و وارسی سیستم‌های واکنشی ممکن می‌گردد. با بیان معنای مدل انتخابی به این روش، می‌توان کمبودها و تناقض‌های احتمالی را مشاهده کرد و راه‌های ممکن برای گسترش آن را بررسی نمود.

همانطور که بیان شد, برای توصیف معنای صوری مدل اکتور در شکل تابعی خود کارهای ارزشمندی انجام گرفته است. در این مقاله, یک زبان رویه‌ای مبتنی بر اکتور ارائه شده است و معنای عملیاتی آن با استفاده از سیستم گذار پایه بیان گردیده است. به این ترتیب مبنایی برای بررسی جامعیت مفهومی زبان و همچنین انجام عملیات وارسی صوری ایجاد شده است. کار انجام شده در این مقاله, پایه‌ی نظری برای ارائه‌ی زبان ربکا [۲۰] و تولید ابزار وارسی مدل‌های ربکا [۲۱] است. مدل‌های نوشته شده به زبان ربکا و ویژگی‌های دلخواه هر مدل به عنوان ورودی به ابزار داده شده و نتیجه‌ی وارسی پس از انجام آزمون مدل[۴] به کاربر نشان داده می‌شود.

در ادامه در بخش ۲ مدل اکتور و در بخش ۳ سیستم گذار پایه معرفی شده‌اند. در بخش ۴ زبان انتخابی و ویژگی‌های اساسی آن بیان شده و معنای این زبان براساس سیستم گذار پایه به شکل صوری تعریف شده است. بخش ۵ استفاده از این زبان را در قالب دو مثال نشان می‌دهد.

مدل اکتور

اکتورها مؤلفه‌های محاسباتی هستند که در زمان و فضای مدل محاسباتی پراکنده‌اند. هر اکتور یک آدرس پستی و یک رفتار دارد. تنها روش تأثیر یک اکتور روی کنش‌های اکتور دیگر فرستادن پیام است. یک اکتور، اگر آدرس پستی اکتور دیگر را داشته باشد
می‌تواند به او پیام بفرستد.

اکتورها مؤلفه‌هایی با میانگیرهای ارتباطی نامحدود هستند، تبادل پیام به صورت ناهمگام انجام می‌گیرد و فرض بر تضمین تحویل پیام یا به عبارت دیگر وجود انصاف در سیستم است. در یک سیستم اکتور، آدرس‌های پستی اکتورها می‌توانند مانند هرمقدار دیگری به آزادی ارسال و دریافت شوند. بنابراین توپولوژی شبکه ارتباطی اکتورها پویاست[۳].

دستورات ابتدایی مدل اکتور *send*, *become* و *create* است[۵]. دستور *send* یک پیام جدید را ایجاد کرده و درصندوق پستی گیرنده قرار می‌دهد.

دستور *become* برای تغییر رفتار است و یک اکتور بی‌نام را برای انجام بقیه محاسبات در جریان ایجاد می‌کند، رفتار اکتوری که دستور *become* را اجرا کرده است تغییر می‌دهد و آن اکتور را برای دریافت پیام دیگر آزاد می‌سازد. این دستور، خود تولیدکننده موازات است. اکتور بی‌نام می‌تواند برای به انجام رساندن محاسبات خود پیام بفرستد و یا اکتورهای جدیدی خلق کند، اما دیگر قادر به دریافت پیام نیست چون آدرس آن دیگر شناخته شده نمی‌باشد.

دستور *create* برای ایجاد یک اکتور جدید با رفتار اولیه و آدرس مشخص است. مشاهده می‌شود که در مدل اکتور، اکتورها می‌توانند در حین اجرا ایجاد شوند.

مدل اکتور آقا در سال ۱۹۹۱ از ویژگی‌های خاصی برخوردار است. درآن مدل، هر اکتور می‌تواند در خود ریسمان‌های اجرایی متعددی داشته باشد. هر زمان که یک پیام توسط اکتور پردازش می‌گردد، یک رفتار جدید ایجاد می‌شود تا پیام بعدی را دریافت کند. اگر رفتار اکتور به عنوان نتیجه پردازش یک پیام تغییر
نکند به معنای آن است که رفتار جایگزین همانند رفتار قبلی اکتور است. رفتار جدید می‌تواند شامل تغییر در متغیرهای حالت و یا حذف یا اضافه شدن متدهای پاسخگو به پیام‌ها باشد. پس از تغییر رفتار، رفتار قدیم اکتور به اجرای خود تا خاتمه ادامه می‌دهد اما دیگر قادر به دریافت پیام نیست، چون به عبارتی آدرس یا شناسه خود را به رفتار جدید واگذار کرده است.

سیستم گذار پایه

در روش ارائه شده توسط منا [۱۴]، برای مدلسازی یک سیستم واکنشی، جزء اصلی از مدل معنایی، سیستم گذار پایه است. این جزء مشخص کننده‌ی حالت‌ها و گذارهای موجود در سیستم است. گذارهای سیستم، کنش‌های تجزیه ناپذیری هستند که در یک محاسبه حالت سیستم را تغییر می‌دهند. معنای سیستم واکنشی با افزودن قیود انصاف به آن تکمیل می‌شود.

برای بیان نحو یک سیستم گذار پایه از یک زبان با مؤلفه‌های V, ε, A و I استفاده می‌شود.

V - واژه نامه

واژه نامه مجموعه‌ای شمارا [۵] ازمتغیرهای گونه‌دار[۶] است. این متغیرها می‌توانند متغیر داده‌ای باشند و یا متغیر کنترلی که پیشرفت در اجرای یک برنامه را نشان می‌دهند. گونه‌ی هر متغیر مشخص‌کننده دامنه‌ای است

که مقادیر آن متغیر در آن قرار می‌گیرد. یک متغیر داده-ای می‌تواند برای مثال از نوع بولی یا اعداد صحیح باشد. یک متغیر کنترلی در محدوده یک مجموعه متناهی از مکان‌ها در برنامه قرار دارد.

ε - عبارتها

عبارتها از متغیرهای V و ثابت‌هایی تشکیل شده است که توابع و مسندات در دامنه‌های تعریف شده، روی آنها عمل می‌کنند.

ثابت‌ها مانند $\varnothing, \Lambda, \circ$؛

توابع مانند $\cup, \bullet, +$؛

مسندات مانند $\subseteq, null, >$؛

دامنه‌ها مانند اعداد صحیح، لیست‌ها و مجموعه‌ها.

$\triangleq$ - ادعاها

ادعاها از عبارت‌های بولی تشکیل می‌شوند که در این عبارت‌ها رابطهای بولی و سورها $(\forall, \exists)$ روی برخی از متغیرها بکار رفته اند.

I - تفسیرها

یک تفسیر $I \in I$ ازمجموعه‌ای از متغیرهای گونه دار $V \subseteq V$ نگاشتی است که به هر متغیر $y \in V$ یک مقدار $I[y]$ را در دامنهy نسبت می‌دهد.

یک سیستم گذار پایه $<\Pi, \Sigma, T, \theta>$ یک برنامه واکنشی را با مؤلفه‌های زیر نمایش می‌دهد:

Π- مجموعه‌ای متناهی از متغیرهای حالت

$\Pi = \{u_1, \ldots u_n\} \subseteq V$

این متغیرهای حالت می‌توانند داده‌ای و یا کنترلی باشند.

Σ- مجموعه‌ای از حالت‌ها

هرحالت s در Σ تفسیری از Π است که به هر متغیرu در Πیک مقدار در دامنه‌اش نسبت می‌دهد. این مقدار را به صورت $s[u]$ نشان می‌دهیم.

T- مجموعه‌ی متناهی از گذارها

هرگذار t در T نمایانگر یک کنش تغییردهنده حالت در سیستم است که به صورت تابع $t : \Sigma \rightarrow 2^{\Sigma}$ تعریف شده و حالت s در Σ را به مجموعه‌ای از حالت‌ها $t(s)$ نگاشت می‌کند. این مجموعه‌ی حالت‌ها با به کارگیری کنش t روی حالت s بدست می‌آید.

θ- شرط اولیه

این ادعا، حالت‌هایی را مشخص می‌کند که اجرای برنامه می‌تواند در آنها آغاز شود. به حالت s که θ را برقرار کند، حالت اولیه گویند.

برای هر گذار t یک رابطه گذار $\rho_t[\Pi, \Pi']$داریم که مقادیر متغیرهای حالت در s را به مقادیر آنها در حالت بعدی s' ارتباط می‌دهد. s' از بکاربردن t روی s حاصل شده است. یک رابطه گذار را می‌توان تشکیل شده از یک شرط اولیه به همراه دستورات تغییر حالت دانست. شرط اولیه را به صورت $c_t[\Pi]$ نمایش می‌دهیم. درصورت وجود این شرط، گذار t می‌تواند روی حالت s عمل کند.

معنای یک زبان مبتنی براکتور

دراین بخش زیر مجموعه‌ای ازمدل اکتور اولیه-[۳] انتخاب شده است. دستورات اصلی در زبان مورد نظر، *send* و *create* است.

از آنجایی که وجود دستور *become* برای الگوی تابعی زبان مناسب است و پیاده سازی آن را درعمل دشوار می‌سازد از زبان مورد استفاده حذف شده است. به این ترتیب امکان داشتن ریسمان‌های اجرایی همروند درون یک اکتور وجود ندارد. هر اکتور یک ریسمان اجراست که به طور همروند با اکتورهای دیگر اجرا می-شود. این محدودیت در کاربردهای دیگری که از الگوی تابعی استفاده نکرده‌اند نیز اعمال شده است [۱۷][۱۸][۱۹][۲۶].

کنش *rcv* به عنوان یک گذار در سیستم وجود دارد که درشرایط تعریف شده اعمال می‌شود اما به صورت یک دستور صریح در زبان وجود ندارد.

به این ترتیب ویژگی‌های تغییر پویای پیکربندی و تبادل ناهمگام پیام درمدل حفظ شده است. همچنین امکان ایجاد پویای یک اکتور وجود دارد. محیط خارجی و تبادل پیام با آن، در زبان در نظر گرفته نشده است. به عبارتی سیستم مورد نظر یک سیستم بسته است. البته تعمیم این زبان به سیستم باز، کاری سرراست است. همچنین همواره می‌توان محیط را به صورت یک اکتور

درون سیستم مدلسازی نمود. معنای این زبان مبتنی براکتور بر مبنای سیستم گذار پایه در ادامه بیان می‌گردد.

زبان‌های شیءگرا، در سطح تجرید بالاتری نسبت به سیستم‌های مبتنی بر پردازه‌ها قرار دارند. بنابراین بیان حالت در چنین زبان‌هایی براساس متغیرهای ساده موجود در واژه نامه، نمایش طولانی و پیچیده‌ای را به وجود می‌آورد. از این رو تعریف اعضای مجموعه‌ی V^* درسطح تجرید بالاتر، برمبنای متغیرهای V بیان شده است. با تجزیه‌ی ساختارهای V^* به متغیرهای ساده‌ی تشکیل دهنده‌ی آنها، V به طور سرراست از V^* به دست می‌آید. مجموعه‌های Π و Σ براساس تعاریف موجود در V^* مشخص شده اند.

V^*: مجموعه‌ای شمارا شامل اکتورها و پیام‌هاست که تعریف دقیق آن در زیر آمده است.

$$V^* = allactors \cup msgs$$

$$allactors : actorid \rightarrow actorstate \times behavior$$

$$actorid \subset ids$$

ids: مجموعه‌ی شناسه‌های اکتورها (به عبارت دیگر آدرس پستی اکتورها)، از نوع رشته‌های کاراکتری

$$actorstate \subseteq statevars \times compstatus$$

statevars: متغیرهای حالت هر اکتور که می‌توانند از گونه‌های ساده‌ی متعارف باشند.

$$compstatus \subseteq actions \times comppoints$$

actions: مجموعه‌ی حالت‌های اجرا در اکتورها، از نوع بولین $actions = \{busy, idle\}$.

comppoints: مجموعه‌ی نقاط اجرا در متدهای اکتور، از نوع رشته‌های کاراکتری که قسمتی از این رشته کاراکتری شناسه‌ی متد مربوطه را مشخص می‌کند و قسمت دیگر مشخص‌کننده‌ی برچسب دستورالعمل اجرایی درآن متد است. برای $idle$, $comppoint$ (عضوی از مجموعه‌ی $comppoints$) همیشه '0' است، یا به بیان دیگر :

$$compstatus = (idle,'0') \,/\, (busy, comppoint)$$

behavior: مجموعه‌ی دنباله‌هایی از متدهای یک اکتور که به صورت رشته‌ی کاراکتری به دنبال نام آن آمده است. هرمتد شناسه‌ی مخصوص به خود را دارد و هر دستورالعمل درون متد دارای برچسب است. معنای *behavior* منطبق برمعنای عملیاتی متعارف دستورالعمل‌های ترتیبی است، ازاین رو از توضیح بیشتر آن خودداری شده است و آن را به عنوان یک رشته‌ی کاراکتری درنظر گرفته‌ایم. معنای دستورالعمل‌های ترتیبی رایج را می توان در [۱۰] یافت.

$$msgs : actorid \rightarrow msgvals$$

msgvals: مجموعه‌ی پیام‌ها شامل متد پاسخگوی پیام و محتوای پیام، از نوع رشته‌های کاراکتری که قسمتی از رشته، متد پاسخگوی پیام را مشخص می‌کند و قسمت دیگر مشخص‌کننده‌ی محتوای پیام است.

$$actors \subset allactors$$

$$pendingmsgs \subseteq msgs$$

Π – مجموعه‌ی متناهی ازمتغیرهای V^*

$$\Pi = actors \cup pendingmsgs$$

Σ: مجموعه‌ای از حالت‌ها، در هرحالت مقادیر موجود درمجموعه‌ی *actors* و *pendingmsgs* مشخص است.

T – مجموعه‌ی گذارهای سیستم

$$T = \{T_{send}, T_{rcv}, T_{create}, T_{func}\}$$

T_{func} نماینده‌ی گروهی از گذارهاست که در اثر اجرای دستورالعمل‌های معمول در زبان‌های ترتیبی عمل می‌کنند (مانند گمارش، دستور شرطی و حلقه). این گروه از گذارها روی مجموعه‌ی *pendingmsgs* تاثیری ندارد و در *actors* باعث تغییر *compstatus* می‌شود. تعریف بقیه‌ی رابطه‌های گذار در ادامه آمده است. برای هردستور، پیش‌شرطی که برای اجرای آن و انجام گذار لازم است نوشته شده و حالت بعدی سیستم با متغیرهای پریم‌دار مشخص شده است.

ارسال پیام

$$\begin{aligned} &a : send\,(b, m) \quad : \\ &\quad a, b \in actors \wedge (b \rightarrow m)\ msgs\ (b \rightarrow m) \\ &\qquad \notin pendingmsgs \wedge \\ &\quad pendingmsgs' = pendingmsgs \cup (\mathrm{b} \rightarrow \mathrm{m}) \wedge \\ &\quad actors'(a)\ .\ statevars = actors(a). \\ &\qquad statevars \wedge \\ &\quad actors'(a)\ .\ compstatus = actors(a)\ . \\ &\qquad succ(compstatus) \\ &\quad actors'(a)\ .\ behavior = actors(a)\ .\ behavior \end{aligned}$$

تابع *succ* در اینجا به این صورت تعریف می‌شود: اگر دستور اجرا شده آخرین دستور اجرایی متد باشد، اکتور به حالت idle می‌رود. درغیر این صورت جزء *actions* از *compstatus*, *busy* باقی مانده و جزء *comppoint* به دستوربعدی درهمان متد اشاره خواهد کرد.

دریافت پیام

$a : rcv\ (b, m)$:
$a \in actors \wedge m \in pendingmsgs(a) \wedge$
$actors(a)\ .\ compstatus = idle \wedge$
$pendingmsgs' = pendingmsgs - \{a \rightarrow m\} \wedge$
$actors'(a)\ .\ statevars = actors(a)\ .\ statevars \wedge$
$actors'(a)\ .\ compstatus = (busy,$
$matchstatement)$
$actors'(a)\ .\ behavior = actors(a)\ .\ behavior$

matchstatement اولین دستورالعمل اجرایی در متد سرویس دهنده به پیام، دراکتور گیرنده‌ی پیام است.

ایجاد اکتور

$create(a,beh,sv)$:
$a \in ids \wedge a \notin actorids \wedge beh \in behaviors$
$actorid' = actorid + \{a\} \wedge$
$actors'(a)\ .\ statevars = sv \wedge$
$actors'(a)\ .\ compstatus = (idle, 0) \wedge$
$actors'(a)\ .\ behavior = beh$

Θ: شرط اولیه مبین اکتورها و پیام‌های موجود در حالت اولیه سیستم است.

مثال

در این بخش یک سیستم کنترل‌کننده دیگ بخار به عنوان یک مثال ساده و یک سیستم نظارت بر تردد روی پل به عنوان یک مثال پیچیده‌تر بررسی می‌شود.

سیستم کنترل‌کننده دیگ بخار

برنامه زیر قسمتی از یک سیستم کنترل‌کننده دیگ بخار است که شامل یک سنسور فشار، یک کنترل‌کننده و یک تنظیم‌کننده شیر است. پس از درخواست کنترل-کننده، سنسور فشار پیامی حاوی مقدار فشار را به آن می‌فرستد. براساس مقدار فشار، کنترل‌کننده مقدار مناسب برای شیر بخار را محاسبه می‌کند. کنترل‌کننده این عملیات را به صورت دوره‌ای تکرار می‌کند. تکرار عملیات با فرستادن پیام loop به خود، انجام می‌گیرد (این مثال از [۱۸] انتخاب شده است).

```
actor PressureSensor() {
  real value ;
  method read(actorRef customer ) {
    send customer . reading(value);
  }
}
```

```
actor steamValve() {…} //unspecified actor actor
controller(actorRef sensor, valve) {
  method loop() {
    send self . loop();
    send sensor . read (self);
  }
  method reading(real pressure ) {
    newValvepos = computeValvepos
                          (pressure )
    send valve . move(newValvepos);
  }
}
```

پیکربندی و حالت اولیه سیستم، همچنین دیگر حالات و گذارهایی که باعث تبدیل هر حالت به حالت بعدی می‌شود، در ادامه آمده است. در حالت اولیه یک پیام *loop()* در صف پیام‌های متناظر اکتور *Controller* قرار دارد. در برخی از حالات بیش از یک گذار می‌تواند عمل کند. این سیستم در یک حلقه به طور مرتب عملیات ذکر شده را تکرار می‌کند و رشته محاسبات نامتناهی است. در اینجا برای نشان دادن تغییر حالات بر مبنای تعریف سیستم گذار پایه، تعدادی از گذارها نمایش داده شده است. هر حالت به صورت s_i : $\langle actors, pendingmsgs \rangle$ و هرگذار به صورت یک پیکان نمایش داده شده است. پیام یا کنشی که موجب انجام گذار می‌گردد در شکل(۱) نشان داده شده است. همان-گونه که در سیستم‌های همروند متداول است، در هر مرحله امکان انتخاب گذارهای متفاوتی وجود دارد. در هر مرحله این انتخاب‌ها ذکر شده است. با توجه به توضیحات بخش قبل، در اینجا از آوردن کدهای حاصل از اجرای دستورات متداول در زبان‌های ترتیبی صرفنظر شده است. حالت اولیه که Θ را برقرار می‌کند همان S_0 است.

S_0 : ⟨ *actors* : {
(*pressureSensor*, v_0, (*idle*, *'0'*))
,(*steam Valve*, , (*idle*, *'0'*))
,(*controller*, , (*idle*, *'0'*))}
,*pendingmsgs*:
{(*controller*, (*loop* ()))} ⟩

شکل(۱) گذار از حالت S_0 به S_1 توسط گذار *controller : rcv(loop ())* را نشان می‌دهد.

S_1 : ⟨ *actors* : {
(*pressureSensor*, v_0, (*idle*, *'0'*))
,(*steam Valve*, , (*idle*, *'0'*))
,(*controller*, , (*busy*, *loop*.1))}
,*pendingmsgs*: {} ⟩

شکل(۱) گذار از حالت S_1 به S_2 توسط گذار *controller : send(self, loop ())* را نشان می‌دهد.

S_2 : ⟨ *actors* : {
(*pressureSensor*, v_0, (*idle*, *'0'*))
,(*steamValve*, , (*idle*, *'0'*))
,(*controller*, , (*idle*, *loop*.2))}
,*pendingmsgs*:
{(*controller*, (*loop* ()))} ⟩

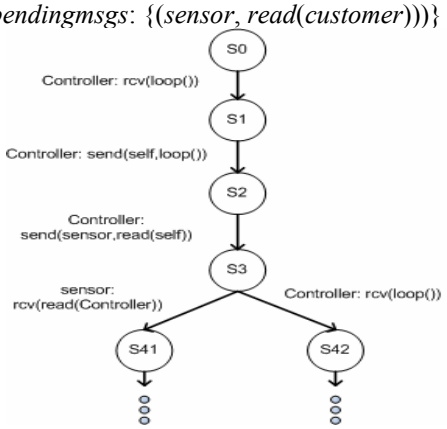
در s_2 اجرای دستورالعمل بعدی در متد انتخاب شده است. امکان انتخاب کنش سرویس به پیام نیز وجود دارد. شکل(۱) گذار از حالت S_2 به S_3 توسط گذار *controller: send(sensor, read(self))* را نشان می‌دهد.

S_3 : ⟨ *actors* : {
(*pressureSensor*, v_0, (*idle*, *'0'*))
,(*steamValve*, , (*idle*, *'0'*))
,(*controller*, , (*idle*, *'0'*))}
,*pendingmsgs*: {(*controller*, (*loop* ())),
(*sensor*, *read*(*controller*)))} ⟩

در S_3 دو انتخاب امکان‌پذیر است. S_{41} و S_{42} دو حالتی هستند که در اثر هر یک از این انتخاب‌ها به وجود می‌آیند. شکل(۱) گذار از حالت S_3 به S_{41} توسط گذار *sensor:rcv(read(controller))* و گذار از حالت S_3 به S_{42} توسط گذار *controller: rcv(loop ())* را نشان می‌دهد.

S_{41}: ⟨ *actors* : {
(*pressureSensor*, v_0, (*busy*, *read*.1))
,(*steamValve*, , (*idle*, *'0'*))
,(*controller*, , (*idle*, *'0'*))}
,*pendingmsgs* : {(*controller*, (*loop*, ()))} ⟩

S_{42}: ⟨ *actors* : {
(*pressureSensor*, v_0, (*idle*, *'0'*))
,(*steamValve*, , (*idle*, *'0'*))
,(*controller*, , (*busy*, *loop*.1))}
,*pendingmsgs*: {(*sensor*, *read*(*customer*))))} ⟩



شکل۱: بخشی از سیستم گذار مربوط به کنترل کننده دیگ بخار.

سیستم نظارت بر تردد روی پل

پلی را در نظر بگیرید که دارای یک خط راه آهن است. از این پل در هر لحظه فقط یک قطار می‌تواند عبور کند. دو قطار در جهات مخالف خواهان ورود به پل هستند و یک کنترل‌کننده با استفاده از چراغ‌های راهنما مانع برخورد قطارها بایکدیگر می‌شود. این کنترل‌کننده، با فرض اینکه قطارها بعد از ورود به پل آنرا ترک خواهند کرد، ضمانت می‌کند که هر قطار در نهایت وارد پل شود و هیچگاه دو قطار به طور همزمان روی پل نباشند.

برنامه زیر مدلی برای مثال ذکر شده است.

```
actor Controller(actorRef Train1, Train2){
  bool isWaiting1, isWaiting2;
  bool signal1, signal2;
  method Arrive(actorRef sender){
    if (sender == Train1){
      if (signal2 == false){
        signal1 = true;
        send Train1.youMayPass();
      }else
        isWaiting1 = true;
    }else{
      if (signal2 ==false){
        signal2 = true;
        send Train2.youMayPass();
      }else
        isWiting2 = true;
    }
  }
  method Leave(actorRef sender){
    if (sender == Train1){
      signal1 = false;
      if (isWaiting2 == true){
        signal2 = true;
        send Train2.youMayPass();
        isWaiting2 = false;}
    }else{
      signal2 = false;
      if (isWaiting1 == true){
        signal1 = true;
        send Train1.youMayPass();
        isWaiting1 = false;}
    }
  }
}
actor Train1(actorRef Controller){
  bool onTheBridge;
  method youMayPass(){
    onTheBridge = true;
    send self.Passed();
  }
  method Passed(){
    onTheBridge = false;
    send Controller.Leave();
    send Controller.ReachBridge();
  }
```

```
    method ReachBridge(){
        send Controller.Arrive(self);
    }
}
actor Train2(actorRef Controller){
    bool onTheBridge;
    method youMayPass(){
        onTheBridge = true;
        send self.Passed();
    }
    method Passed(){
        onTheBridge = false;
        send Controller.Leave();
        send Controller.ReachBridge();
    }
    method ReachBridge(){
        send Controller.Arrive(self);
    }
}
```

کنترل‌کننده دارای متغیرهایی برای نگه داشتن حالت چراغ‌های راهنما در دو طرف پل و همچنین متغیرهایی برای تشخیص انتظار قطارها جهت عبور از پل است. در پیوست، قسمتی از سیستم گذار مربوط به کنترل‌کننده و قطارها آمده است. حالت اولیه که Θ را برقرار می‌کند همان S_0 است. در این حالت متغیرهای *isWaiting1*، *isWaiting2*، *signal1* و *signal2* از اکتور کنترلر همگی False هستند. متغیرهای *onTheBridge* قطارها نیز مقدار False دارند.

در حالت اولیه پیام *Passed()* در صف پیام‌های متناظر اکتورهای قطار قرار دارد.

نتیجه گیری

کار حاضر را می‌توان از دو دیدگاه بررسی کرد. نخست زیرمجموعه انتخاب شده از مدل اکتور و ویژگی‌های آن و دوم نحوه بیان معنای این مدل. زبان اکتور پیشنهادی دارای ویژگی‌های یک زبان امری است. امکان تغییر پویای پیکربندی، تبادل ناهمگام پیام و ایجاد پویای یک اکتور در این زبان وجود دارد. محیط خارجی و تبادل پیام با آن، درمدل درنظرگرفته نشده است. به عبارتی سیستم مورد نظر یک سیستم بسته است. معنای عملیاتی مدل بر اساس سیستم گذار پایه بیان گردیده است که با نحوه‌ی بیان معنا در منابع دیگر متفاوت است. این شیوه، به علت سادگی و روشنی آن انتخاب شده است. درک معنای مدل، هنگامی که به این روش بیان گردد ساده تر بوده و کمبودهای احتمالی به سهولت قابل مشاهده است. به علاوه تعمیم مدل به یک سیستم باز و بی‌درنگ و درنظرگرفتن انصاف به گونه‌ای طبیعی و مستقیم امکان-پذیر است.

زبان پیشنهادی و معنای صوری ارائه شده در این مقاله پایه‌ی نظری برای ارائه‌ی زبان ربکا و ابزار وارسی آن بوده است.

مراجع

1 - Agha, G. and Hewitt, C. (1988). *Concurrent programming using actors.* in Yonezawa A., Tokoro M., ed., Object-Oriented Concurrent Programming, MIT Press, PP. 37-53.

2 - Agha, G. (1990). "Concurrent object-oriented programming." *J. of Communications of the ACM*, Vol. 33, No. 9, PP. 125-141.

3 - Agha, G. (1991). "The structure and semantics of actor languages." Foundations of Object-Oriented Languages: REX School Workshop, LNCS 489, PP. 1-59.

4 - Agha, G., Mason, I., Smith, S. and Talcott, C. (1997). "A foundation for actor computation." *J. of Functional Progtamming*, Vol. 7, PP. 1-72.

5 - Agha G. and Kim W. (1998). "Actors: a unifying model for parallel and distributed computing." *J. of Systems Architecture*, EuroMicro Society in cooperation with Elsevier, PP. 1-15.

6 - America, P., Bakker J., Kok J. and Rutten J. (1986). "Operational semantics of a parallel object-oriented language." *13th Conf. on Principles of Programming Languages*, PP. 194-208.

7 - America, P. (1988). *POOL-T:A parallel object-oriented language*. in Yonezawa A., Tokoro M., ed., Object-Oriented Concurrent Programming, MIT press, PP. 37-58.

8 - America, P. and Rutten, J. (1991). “A layered semantics for a parallel object-oriented language.” *in Foundations of Object–Oriented Languages: REX School Workshop*, LNCS 489, PP. 91-123.

9 - Baker, H. G. and Hewitt, C. (1997). *Laws for communicating parallel processes*. IFIP Congress, PP. 987-992.

10 - Best, E. (1996). *Semantics of sequential and parallel programs*. Prentice Hall.

11 - Clinger, W. (1981). *Foundations of Actor Semantics*. AI-TR-, MIT Artificial Intelligence Laboratory.

12 - Hewitt, C. (1971). *Description and theoretical analysis (Using Schemata) of PLANNER: a language for proving theorems and manipulating models in a robot*. PhD Thesis, MIT.

13 - Keller, R. M. (1976). “Formal verification of parallel programs.” *Communications of ACM*, Vol. 19, No. 7, PP. 371-384.

14 - Manna, A. and Pnueli, A. (1991). *The Temporal Logic of Reactive and Concurrent Systems.* Springer-Verlag.

15 - Mason, I. A. and Talcott, C. L. (1999). “Actor languages: their syntax, semantics, translation and equivalence.” *Theoretical Computer Science*, Vol. 220, No. 2, PP. 409-467.

16 - Meyer, B. (1991). *Introduction to the Theory of programming languages.* Prentice Hall.

17 - Nielsen, B. and Agha, G. (1996). “Semantics for an actor-based real-time langage.” *Proc. 4th Int. Wrokshop on Parallel and Distributed Real-Time Systems*, PP. 223-228.

18 - Nielsen, B., Ren, S. and Agha, G. (1998). “Specification of real-time interaction constraints.” *Proc. first Int. Symp. on Object-Orinted Real-Time Computing*, IEEE Computer Society, PP. 206-214

19 - Ren, S. and Agha, G. (1995). “RTSynchronizer: language support for real-time specifications in distributed systems.” *ACM SIGPLAN Workshop on Languages, Compilers and Tools for Real-Time Systems*, California, PP. 50-59.

20 - Sirjani, M., Movaghar, A. and Mousavi, M. R. (2001). “Compositional verification of an actor-based model for reactive systems.” *Proc. Workshop on Automated Verification of Critical Systems (AVoCS'01)*, Oxford University, PP. 114-118.

21 - Sirjani, M., Shali, A., Jaghoori, M.M., Iravanchi, H. and Movaghar, A. (2004). **“**A front-end tool for automated abstraction and modular verification of actor-based models.” *Int. Conf. on Application of Concurrency to System Design (ACSD)*. (to appear)

22 - Talcott, C. (1997). “Actor theories in rewriting logic.” *Theoretical Computer Science*, Vol. 1346, PP. 250-267.

23 - Vaandrager F. W. (1986). “Process algebra semantics for POOL.” *Report CS-R8629*, Center for Mathematics and Computer Science, Netherlands.

24 - Varela, G. and Agha, G. (2001). “Programming dynamically reconfigurable open systems with SALSA.” *ACM SIGNPLAN Notices*, Vol. 36, No. 12, PP. 20-34.

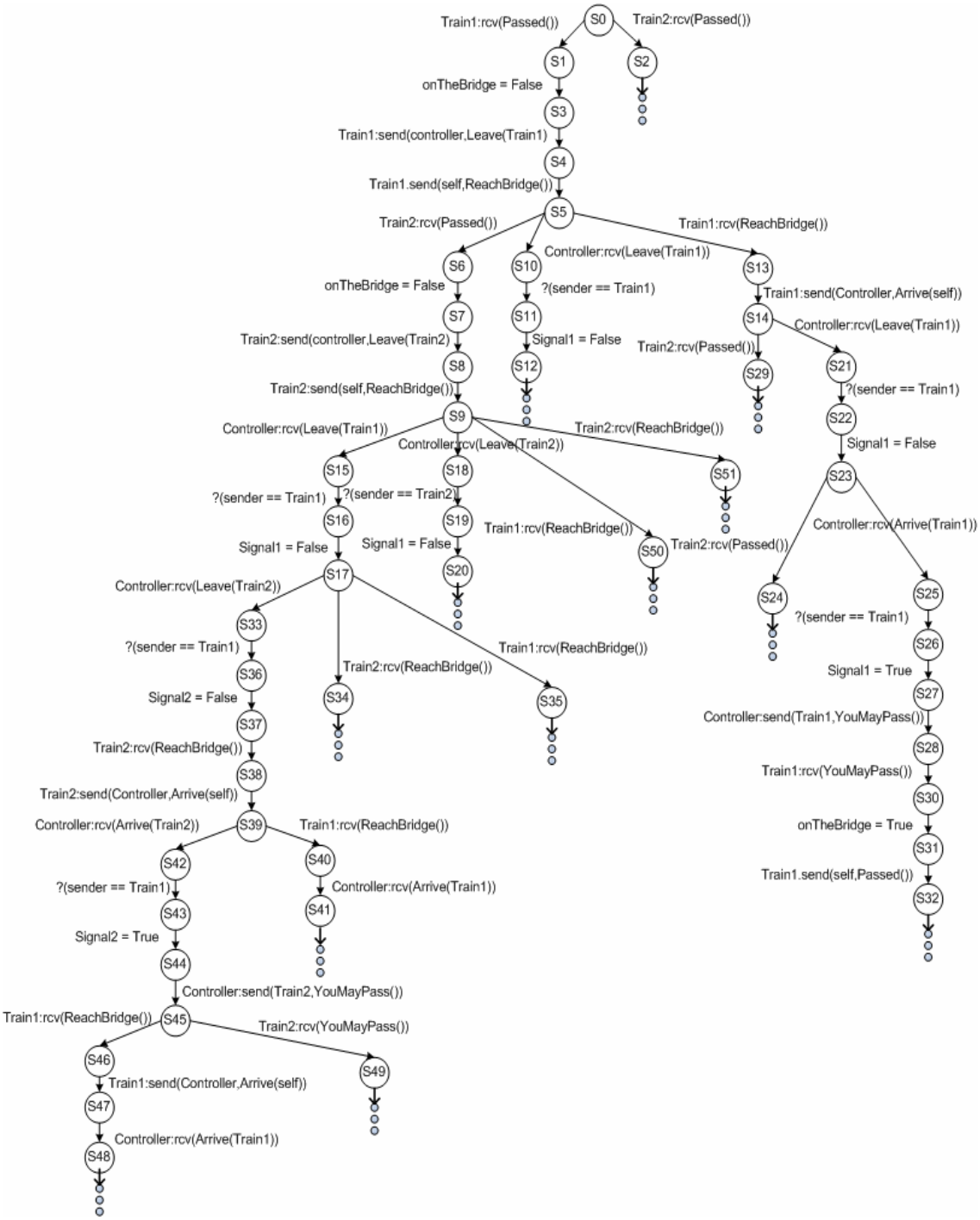
25 - Yonezawa A., Shibayama E., Takada T. and Honda Y. (1988). “Modeling and programming in an object-oriented concurrent language ABCL/1.” In Yonezawa A., Tokoro M., ed., *Object-Oriented Concurrent programming*, MIT Press , PP. 37-53.

۲۶- خسروی، ر. و موقر، ع. "اعتبار سنجی سیستم‌های همروند به کمک مدل اکتور و منطق زمانی." مجموعه مقالات سومین کنفرانس بین المللی سالانه انجمن کامپیوتر ایران، ص ص. ۵۶ – ۶۳، (۱۳۷۶).

واژه های انگلیسی به ترتیب استفاده در متن

1 - Fair
2 - Reasoning Agents
3 - Formalism
4 - Model Checking
5 - Countable
6 - Typed

پیوست ۱



بخشی از سیستم گذار مربوط به سیستم نظارت بر تردد روی پل

پیوست ۲

S_0: ⟨*actors*: {*(Controller, F, F, F, F, (idle, '0'))*, *(Train1, F, (idle, '0'))*, *(Train2, F, (idle, '0'))*}
,pendingmsgs: {(Train1, Passed()), (Train2, Passed())} ⟩

S_1: ⟨*actors*: {*(Controller, F, F, F, F, (idle, '0'))*, *(Train1, F, (busy, Passed.1))*, *(Train2, F, (idle, '0'))*}
,pendingmsgs: {(Train2, Passed())} ⟩

S_2 : ⟨*actors*: {*(Controller, F, F, F, F, (idle, '0'))*, *(Train1, F, (idle, '0'))*, *(Train2, F, (busy, Passed.1))*}
,pendingmsgs: {(Train1, Passed())} ⟩

S_3: ⟨*actors*: {*(Controller, F, F, F, F, (idle, '0'))*, *(Train1, F, (busy, Passed.2))*, *(Train2, F, (idle, '0'))*}
,pendingmsgs: {(Train2,Passed())} ⟩

S_4: ⟨*actors*: {*(Controller, F, F, F, F, (idle, '0'))*, *(Train1, F, (busy, Passed.3))*, *(Train2, F, (idle, '0'))*}
,pendingmsgs: {(Train2,Passed()),Controller.Leave(Train1)} ⟩

S_5: ⟨*actors*: {*(Controller, F, F, F, F, (idle, '0'))*, *(Train1, F, (idle, '0'))*, *(Train2, F, (idle, '0'))*}
,pendingmsgs:{(Train2,Passed()),(Controller.Leave(Train1)), (Train1, ReachBridge())} ⟩

S_6 : ⟨*actors*: {*(Controller, F, F, F, F, (idle, '0'))*, *(Train1, F, (idle, '0'))*, *(Train2, F, (busy, Passed.1))*}
,pendingmsgs: {(Controller.Leave(Train1)), (Train1, ReachBridge())} ⟩

S_7: ⟨*actors*: {*(Controller, F, F, F, F, (idle, '0'))*, *(Train1, F, (idle, '0'))*, *(Train2, F, (busy, Passed.2))*}
,pendingmsgs:{(Controller.Leave(Train1)), (Train1, ReachBridge())}⟩

S_8: ⟨*actors*: {*(Controller, F, F, F, F, (idle, '0'))*, *(Train1, F, (idle, '0'))*, *(Train2, F, (busy, Passed.3))*}
,pendingmsgs: {(Controller.Leave(Train1)), (Train1,ReachBridge()),(Controller.Leave(Train2))} ⟩

S_9: ⟨*actors*: {*(Controller, F, F, F, F, (idle, '0'))*, *(Train1, F, (idle, '0'))*, *(Train2, F, (idle,'0'))*}
,pendingmsgs:{(Controller.Leave(Train1)), (Train1,ReachBridge()),(Controller.Leave(Train2)),
(Train2.ReachBridge())} ⟩

S_{10}: ⟨*actors*: {*(Controller, F, F, F, F, (busy, Leave.1))*, *(Train1, F, (idle, '0'))*, *(Train2, F, (idle, '0'))*}
,pendingmsgs:{(Train2,Passed()), (Train1, ReachBridge())} ⟩

S_{11}: ⟨*actors*: {*(Controller, F, F, F, F, (busy, Leave.2))*, *(Train1, F, (idle, '0'))*, *(Train2, F, (idle, '0'))*}
,pendingmsgs:{(Train2,Passed()), (Train1, ReachBridge())} ⟩

S_{12}: ⟨*actors*: {*(Controller, F, F, F, F, (idle, '0'))*, *(Train1, F, (idle, '0'))*, *(Train2, F, (idle, '0'))*}
,pendingmsgs:{(Train2,Passed()), (Train1, ReachBridge())} ⟩

S_{13}: ⟨*actors*: {*(Controller, F, F, F, F, (idle, '0'))*, *(Train1, F, (busy, RichBridge.1))*, *(Train2,F, (idle, '0'))*}
,pendingmsgs:{(Train2,Passed()),(Controller.Leave(Train1))} ⟩

S_{14}: ⟨*actors*: {*(Controller, F, F, F, F, (idle, '0'))*, *(Train1, F, (idle, '0'))*, *(Train2, F, (idle, '0'))*}
,pendingmsgs:{(Train2,Passed()),(Controller.Leave(Train1)), (Controller.Arrive(Train1))} ⟩

S_{15}: ⟨*actors*: {*(Controller, F, F, F, F, (busy,Leave.1))*, *(Train1, F, (idle, '0'))*, *(Train2, F, (idle,'0'))*}
,pendingmsgs: { (Train1,ReachBridge()), (Controller.Leave(Train2)), (Train2.ReachBridge())} ⟩

S_{16}: ⟨*actors*: {*(Controller, F, F, F, F, (busy, Leave.2))*, *(Train1, F, (idle, '0'))*, *(Train2, F, (idle,'0'))*}
,pendingmsgs: { (Train1,ReachBridge()), (Controller.Leave(Train2)), (Train2.ReachBridge())} ⟩

S_{17}: ⟨*actors*: {*(Controller, F, F, F, F, (idle,'0'))*, *(Train1, F, (idle, '0'))*, *(Train2, F, (idle,'0'))*}
,pendingmsgs: { (Train1,ReachBridge()), (Controller.Leave(Train2)), (Train2.ReachBridge())} ⟩

S_{18}: ⟨*actors*: {*(Controller, F, F, F, F, (busy,Leave.1))*, *(Train1, F, (idle, '0'))*, *(Train2, F, (idle,'0'))*}
,pendingmsgs: {(Controller,Leave(Train1)), (Train1,ReachBridge()),(Train2.ReachBridge())} ⟩

S_{19}: ⟨*actors*: {*(Controller, F, F, F, F, (busy, Leave.8))*, *(Train1, F, (idle, '0'))*, *(Train2, F, (idle,'0'))*}
,pendingmsgs:{(Controller,Leave(Train1)), (Train1,ReachBridge()), (Train2.ReachBridge())} ⟩

S_{20}: ⟨*actors*: {*(Controller, F, F, F, F, (idle,'0'))*, *(Train1, F, (idle, '0'))*, *(Train2, F, (idle,'0'))*}
,pendingmsgs:{(Controller,Leave(Train1)), (Train1,ReachBridge()),(Train2.ReachBridge())} ⟩

S_{21}: ⟨*actors*: {*(Controller, F, F, F, F, (busy,Leave.1))*, *(Train1, F, (idle, '0'))*, *(Train2, F, (idle, '0'))*}
,pendingmsgs:{(Train2,Passed()),(Controller.Arrive(Train1))} ⟩

S_{22}: ⟨*actors*: {*(Controller, F, F, F, F, (busy,Leave.2))*, *(Train1, F, (idle, '0'))*, *(Train2, F, (idle, '0'))*}
,pendingmsgs:{(Train2,Passed()),(Controller.Arrive(Train1))} ⟩

بخشی از حالات سیستم گذار مربوط به سیستم نظارت بر تردد روی پل