فصلنامه‌ی علمی و پژوهشی شریف
تابستان ۱۳۸۶، شماره‌ی سی و هشتم، ص. ۸۱-۱۰۱

توصیف مبتنی بر رخداد (EBV) رویکردی برای ساخت خودکار راستی‌آزمای رفتار حین اجرای نرم‌افزارهای حساس به ایمنی

سید مرتضی بابامیر (دانشجوی دکتری)
سعید جلیلی (استادیار)
دانشکده فنی و مهندسی - گروه کامپیوتر، دانشگاه تربیت مدرس

راستی‌آزمایی پویا، که به راستی‌آزمایی رفتار حین اجرای نرم‌افزار در برابر نیازهای اولیه‌ی آن می‌پردازد، با چالش‌های ساخت راستی‌آزمای رفتار اجرایی نرم‌افزار و نگاشت بین این رفتار و رخدادهای محیطی مواجه است. ما در این نوشتار رویکردی بنام توصیف مبتنی بر رخداد[1] (EBV) را ارائه می‌دهیم که طی چهار مرحله راه‌حلی خودکار برای این چالش‌ها ارائه می‌کند. در مرحله‌ی اول، مستندسازی نیازها، مشتمل بر تعامل محیط و نرم‌افزار، را با روش Parnas مدل می‌کنیم و در مرحله‌ی دوم با استفاده از نگاشت‌های سه‌گانه، کمیت‌های مدل را به توصیف‌های سطح بالا و مبتنی بر رخداد که برحسب منطق حساب رخداد اظهار می‌شوند، نگاشت می‌کنیم. در مرحله‌ی سوم با استفاده از نگاشت‌های هفت‌گانه توصیف‌های مبتنی بر رخداد را به توصیف‌های سطح میانی نیازها که یک توصیف مبتنی بر حالت است، نگاشت می‌کنیم. این توصیف، مرجعی برای راستی‌آزمایی حالات درست و قابل انتظار نرم‌افزار به‌وسیله‌ی برنامه‌ی راستی‌آزما است. در مرحله‌ی چهارم با بهره‌گیری از الگوی طراحی حالت به پیاده‌سازی توصیف حالات می‌پردازیم. با طی این چهار مرحله، و با فراهم آوردن تناظر بین رخدادهای محیطی سطح بالا و فعالیت‌های اجرایی و سطح پایین نرم‌افزار، راستی‌آزما را می‌سازیم. به‌منظور اثبات عملی‌بودن توصیف مبتنی بر رخداد (EBV) ساخت راستی‌آزمای رفتار اجرایی را برای یک نرم‌افزار حساس به ایمنی از روی توصیف نیازها نشان می‌دهیم و آن را تا مرحله‌ی پیاده‌سازی دنبال می‌کنیم.

۱. مقدمه

نرم‌افزار علاوه بر آنکه در همه‌ی حوزه‌های صنعت، خدمات، اقتصاد، علوم و غیره نفوذ کرده است و به‌عنوان مؤلفه‌یی مهم در بسیاری از سیستم‌ها کاربرد دارد، اندازه و پیچیدگی آن در محصولات نیز به‌تدریج افزایش می‌یابد. بنابراین استفاده از نرم‌افزار در محصولات و سیستم‌ها یک امتیاز کلیدی رقابتی بین سازندگان آنها محسوب می‌شود و تهیه‌کنندگان محصولات که قادر به عرضه‌ی نرم‌افزار با کیفیت بالا و قابل پیش‌بینی با حداقل هزینه‌ی ساخت و به‌موقع باشند، یک امتیاز بزرگ نسبت به بقیه رقبا خواهند داشت.

از طرف دیگر افزایش پیچیدگی، احتمال بروز خطاهای ظریف را بیشتر می‌کند و این موجب افزایش احتمال شکست نرم‌افزار می‌شود. شکست نرم‌افزار می‌تواند بسیار پرهزینه باشد و در سیستم‌های حیاتی مانند پزشکی و کنترل پرواز باعث فقدان زندگی انسان‌ها شود. بنابراین یک هدف عمده در مهندسی نرم‌افزار تواناساختن سازندگان برای ساخت سیستم‌هایی است که علیرغم اندازه و پیچیدگی زیاد نرم‌افزار در آنها، قابل اعتماد باشند. رشد استفاده از نرم‌افزار در سیستم‌ها از یک طرف، و ضرورت اطمینان و اتکا به عملکرد صحیح آن از طرف دیگر، موجب شده است تا فرایند راستی‌آزمایی و اعتبارسنجی، نقش حیاتی در اثبات راستی توصیف، طراحی و پیاده‌سازی نرم‌افزار ایفا کند. اهداف راستی‌آزمایی و اعتبارسنجی را می‌توان به سه مورد خلاصه کرد: ۱. آشکار ساختن نقص‌های پنهانی که در توصیف، طراحی، و پیاده‌سازی نرم‌افزار وجود دارند؛ ۲. حصول اطمینان از این که نرم‌افزار دچار شکست نمی‌شود؛ ۳. کاهش هزینه با جلوگیری از شکست و تعمیر نرم‌افزار. اگرچه رویکرد راستی‌آزمایی توصیف و رویکرد آزمون نرم‌افزار برای کشف خطاهای توصیف، طراحی و پیاده‌سازی کاربرد معمول دارد، هنوز خطاهایی ممکن است رخ دهد که این دو رویکرد قادر به یافتن آنها نیستند. تصمیم‌ناپذیری برخی از ویژگی‌ها، مشکل‌شدن اثبات ویژگی‌ها در نرم‌افزارهای بزرگ و پیچیده و در نظر نگرفتن شرایط محیطی از موانع رویکرد راستی‌آزمایی توصیف و عملی‌نبودن آزمون به‌ازاء تمام مقادیر ممکن و گزینش مجموعه‌ی مناسبی از داده‌ها برای آزمون، از موانع رویکرد آزمون است.

بنابراین با توجه به این که اولاً نرم‌افزار نقص‌هایی بالقوه دارد که نمی‌تواند به‌وسیله‌ی رویکرد راستی‌آزمایی توصیف و رویکرد آزمون نرم‌افزار پیدا شود، دوماً امکان عدم انطباق پیاده‌سازی با توصیف نیازها، و سوماً عدم امکان پیش‌بینی شرایط محیطی که نرم‌افزار در آن اجرا می‌شود، لازم است نرم‌افزارهای حساس به ایمنی، که باید از اعتماد

بالایی برخوردار باشند، بیشتر مورد راستی‌آزمایی قرارگیرند تا به اندازه‌ی کافی مطمئن باشند. برای مثال، تحقیقات انجام شده مشخص کرده است که عمده دلیل نفوذهای انجام‌شده به نرم‌افزار عبارت است از: ۱. نقص‌هایی که در پیاده‌سازی آنها وجود دارد، مانند استفاده از اشاره‌گرهای کهنه[۲] (اشاره‌گرهای معلق)[۱و۲]؛ ۲. عوامل محیطی مانند کدی که مترجم زبان به کد اجرایی برنامه اضافه می‌کند. عوامل محیطی، مانند کدهای اضافه‌شده، برنامه را آسیب‌پذیرتر می‌سازد زیرا چیزی نیست که برنامه‌نویس آن‌را اضافه کرده باشد و بنابراین از آن ناآگاه است و در زمان توصیف مسئله، راستی‌آزمایی آن ممکن نیست. نفوذی ممکن است با ایجاد خطاهایی مانند، سرریز میانگیر[۳] در فضای حافظه‌ی برنامه، ایمنی آن را در معرض خطر قرار دهد. برای مثال LSASS [۴] در سرویسگر ویندوز ۲۰۰۰ و XP با استفاده از نقص سرریز میانگیر به‌وسیله‌ی نفوذی‌های تحت عنوان کار برگمنام[۵] از راه دور مورد حمله قرار گرفت.[۳] آسیب از طریق سرریز میانگیر به‌دلیل برنامه‌نویسی ضعیف است که عمدتاً در اداره‌ی توابع رشته‌یی در زبان برنامه‌نویسی C پیش می‌آید. بنابراین برای تشخیص انحراف برنامه از نیازها باید به تشخیص رفتار غیرعادی برنامه پرداخت. برنامه در یک وضعیت غیرمعمول از حالت عادی و قابل انتظار به حالت غیرقابل انتظارگذار می‌کند.

یک راهکار برای برخورد با مشکلات یاد شده، و تشخیص رفتار نادرست برنامه این است که در هر بار اجرای واقعی برنامه، آن را پایش و راستی‌آزمایی کنیم تا از عدم انحراف آن از نیازها مطمئن شویم. این رویکرد که «رویکرد راستی‌آزمایی پویا» (یا راستی‌آزمایی حین اجرا) نام دارد، کارایی خود را در نرم‌افزارهای حساس به ایمنی نشان داده است. استفاده از رویکرد راستی‌آزمایی پویا برای تعیین خطاهای نرم‌افزاری، که در ضمنِ فرایند راستی‌آزمایی توصیف و نیز در ضمنِ آزمونِ نرم‌افزار کشف نشده است، تأثیر به‌سزایی در جلوگیری از شکست نرم‌افزارهای حیاتی دارد. برای مثال پس از راستی‌آزمایی توصیف و آزمون نرم‌افزار مأموریت پرواز آریان ۵، رویکرد راستی‌آزمایی پویا (راستی‌آزمایی حین اجرا) عملاً توانست از فاجعه‌ی شکست این مأموریت که به دلیل خطای سرریز، مقدار اعشاری ۶۴ بیتی را به مقدار صحیح ۱۶ بیتی تبدیل کرده بود، ممانعت کند.[۴] گفتنی است که رویکرد آزمون به‌تنهایی قادر به کشف و کاهش نرخ شکست در حدود $۱۰^{-۴}$ شکست در ساعت است در صورتی که در نرم‌افزارهای حساس به ایمنی بیشنه‌ی قابل قبول $۱۰^{-۹}$ شکست در ساعت است.[۵و۶]

ما در این نوشتار رویکردی به نام توصیف مبتنی بر رخداد (EBV) ارائه می‌دهیم که به‌وسیله‌ی آن می‌توان راستی‌آزمای رفتار اجرایی نرم‌افزارهای حساس به ایمنی را به‌طور خودکار ساخت. به‌این منظور لازم است تا نگاشت بین رخدادهای سطح بالای محیطی و فعالیت‌های سطح پایین اجرایی نرم‌افزار را مشخص کنیم تا راستی‌آزما بتواند قیود وابسته به رخدادها را در فعالیت‌های اجرایی راستی‌آزمایی کند. این نگاشت در چهار مرحله انجام می‌شود که طی آن از توصیف نیازها به پیاده‌سازی راستی‌آزما می‌رسیم. در مرحله‌ی اول مستندسازی نیازها را، که شامل تعامل نرم‌افزار و محیط است، به روش Parnas مدل می‌کنیم زیرا با استفاده از این روش قادر خواهیم بود که محیط، سیستم و نرم‌افزار را به خوبی از یکدیگر جدا کنیم تا امکان توصیف انتزاعی و مبتنی بر رخداد برای محیط فراهم شود. در مرحله‌ی دوم با ارائه‌ی نگاشت‌های سه‌گانه، کمیت‌های مدل ساخته‌شده را به روان‌گزاره‌های حساب رخداد تبدیل می‌کنیم تا توصیفی مبتنی بر رخداد از محیط و نیازهای آن ارائه دهیم. استفاده از منطق حساب رخداد[۶] علاوه بر آن که ما را در توصیف‌های مبتنی بر رخداد (سطح بالا و نزدیک به محیط) و وابسته به زمان یاری می‌دهد، استخراج توصیف مبتنی بر حالت را نیز برای مرحله بعدی آسان می‌کند. در مرحله‌ی سوم با ارائه‌ی نگاشت‌های هفت‌گانه، گزاره‌های حساب رخداد را به عناصر جدول حالت SCR[۷] تبدیل می‌کنیم تا توصیف مبتنی بر حالت نیازها را از روی توصیف مبتنی بر رخداد آنها به‌صورت جدول حالت به دست آوریم. با ارائه‌ی جدول حالت، رفتار اجرایی نرم‌افزار را به صورت یک ماشین حالت انتزاعی نشان می‌دهیم که هر فعالیت اجرایی نرم‌افزار موجب تغییر حالت آن می‌شود. در مرحله‌ی چهارم به پیاده‌سازی عناصر جدول حالت با استفاده از الگوی طراحی حالت[۸] می‌پردازیم (پیاده‌سازی راستی‌آزما). الگوی طراحی حالت، الگویی برای پیاده‌سازی رفتارهای چندریخت و متغیر حین اجرای نرم‌افزار است. ماشین حالت به‌عنوان مرجعی برای راستی‌آزمایی حالات درست و قابل انتظار زمان اجرای نرم‌افزار به‌وسیله‌ی راستی‌آزما محسوب می‌شود.

EBV رویکردی است مناسب برای نرم‌افزارهای واکنشی که بخش عمده‌یی از سیستم‌های حیاتی را تشکیل می‌دهند، زیرا این نرم‌افزارها که در تعامل با محیط بیرون رخدادگرا هستند، در درون خود حالت‌گرا هستند. این بدان مفهوم است که در هنگام بروز واکنش به یک رخداد محیطی، حالت درونی آنها تغییر می‌کند. بنابراین حالت درست نرم‌افزار باید متناسب با رخدادهایی باشد که در محیط رخ می‌دهند. نرم‌افزارهای واکنشی غالباً دارای قیود زمانی‌اند، تعامل دائم با محیط دارند، پایان‌ناپذیرند، حالت نهایی ندارند، اما معین هستند. وظیفه‌ی این نرم‌افزارها ارائه‌ی پاسخ پیوسته، به موقع و مناسب به رخدادهای محیطی است.

در بخش دوم این نوشتار نگاهی داریم به انواع رویکردهای راستی‌آزمایی نرم‌افزار، و در بخش سوم به رویکردهای مرتبط و مقایسه‌ی آنها با EBV خواهیم پرداخت. در بخش چهارم به تشریح مراحل (چهار مرحله) رویکرد EBV می‌پردازیم، و در بخش پنجم، سیستم حساس به ایمنی پمپ انسولین همراه[۹] را معرفی می‌کنیم و رویکرد EBV را برای آن به‌کار می‌بریم.

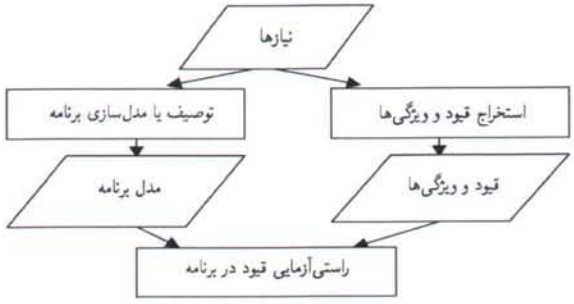
۲. رویکردهای راستی‌آزمایی

راستی‌آزمایی نرم‌افزار یک مرحله‌ی مهم در فرایند ساخت نرم‌افزار است و عاملی عمده در کسب اطمینان از کیفیت نرم‌افزار محسوب می‌شود. راستی‌آزمایی از یک طرف به‌دلیل هزینه‌ی زیادِ شکست و تعمیر نرم‌افزار یک موضوع اقتصادی است، و از طرف دیگر به‌دلیل فقدان زندگی بشری در شکست نرم‌افزارهای حساس به ایمنی، یک مسئله حیاتی است. بنابراین راستی‌آزمایی نرم‌افزار که در جهت کمک به کاهش خطرپذیری و ابتلا به هزینه‌های مالی و جانی صورت می‌گیرد، حصول اطمینان از کیفیت نرم‌افزار را به‌عهده دارد.

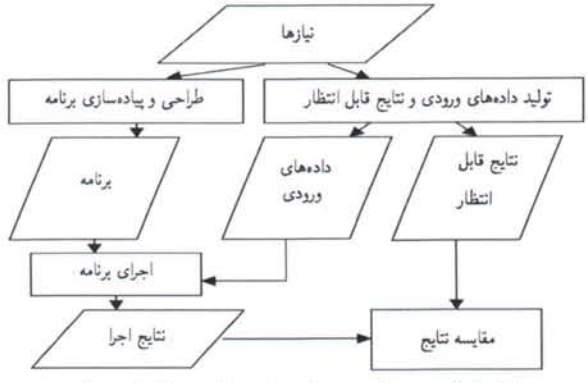
برای راستی‌آزمایی یک نرم‌افزار باید به راستی‌آزمایی آن در برابر رفتار مورد انتظار پرداخت. رفتار مورد انتظار، ویژگی‌ها یا گزاره‌های ایمنی[۱۰] هستند که نرم‌افزار باید همیشه آنها را ارضا کند. رویکردهای راستی‌آزمایی عبارت‌اند از: ۱. رویکرد راستی‌آزمایی توصیف (شکل ۱)؛ ۲. رویکرد آزمون (شکل ۲)؛ ۳. رویکرد راستی‌آزمایی حین اجرا (شکل ۳). لازم به ذکر است که در شکل‌های ۱ تا ۳ گراف متوازی‌الاضلاع نشان‌گر مستندات، و گراف مستطیل بیان‌گر فعالیت است.

۱.۲. رویکرد راستی‌آزمایی توصیف

در رویکرد راستی‌آزمایی توصیف ابتدا با استفاده از یکی از روش‌های رسمی مانند جبر، منطق یا گراف حالت برنامه را توصیف می‌کنند و سپس با استفاده از روش‌هایی مانند اثبات قضیه یا بررسی مدل به راستی‌آزمایی قیود و ویژگی‌ها در توصیف می‌پردازند (شکل ۱).



شکل ۱. راستی‌آزمایی توصیف برنامه در برابر قیود.



شکل ۲. آزمون برنامه در برابر نیازها با اجرای آزمایشی آن.

رویکرد راستی‌آزمایی توصیف که یک راستی‌آزمایی ایستا است و بدون اجرای برنامه صورت می‌گیرد، علاوه بر راستی‌آزمایی توصیف در برابر قیود به راستی‌آزمایی «سازگاری» بین اجزاء توصیف نیز می‌پردازد. دو رویکرد عمده در راستی‌آزمایی توصیف عبارت‌اند از «بررسی مدل» و «اثبات قضیه»؛ بررسی مدل متکی به ساخت یک مدل (ماشین حالت) متناهی از برنامه و بررسی قیود در این مدل است. اثبات قضیه نخست برنامه و قیود را با فرمول‌های منطق ریاضی توصیف و سپس با استفاده از اصول و قواعد استنتاجی به اثبات آنها در نرم‌افزار می‌پردازد.

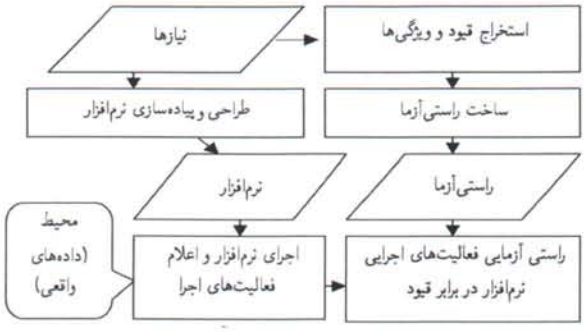
۲.۲. رویکرد آزمون

در رویکرد آزمون به جای راستی‌آزمایی توصیفِ برنامه در برابرِ قیود و ویژگی‌ها، به بررسی اجرای آزمایشی برنامه در برابر قیود می‌پردازند. آزمون رویکردی پویا است، زیرا با استفاده از اجرای برنامه صورت می‌گیرد. در این رویکرد، ابتدا داده‌های آزمایشی تولید و سپس برنامه به‌صورت آزمایشی (و نه واقعی) اجرا می‌شود و بعد نتیجه‌ی اجرا با مقادیر قابل انتظار مقایسه می‌شود (شکل ۲). آزمون برنامه برای همه‌ی انواع نرم‌افزارها ــ اعم از حیاتی و عادی ــ کاربرد دارد و نسبت به رویکرد راستی‌آزمایی توصیف استفاده‌ی وسیع‌تری دارد.

رویکرد آزمون به دو رویکرد آزمون جعبه سیاه و رویکرد آزمون جعبه سفید تقسیم می‌شود.[۷] در آزمون جعبه سیاه، نرم‌افزار به‌صورت جعبه‌یی در نظر گرفته می‌شود که هیچ آگاهی از داخل آن وجود ندارد و فقط، وظایفی که باید انجام دهد مشخص شده است. مقادیر ورودی به نرم‌افزار در حالت‌های مختلف اعمال و نرم‌افزار اجرا می‌شود، و سپس مقادیر حاصل از اجرا با مقادیر قابل انتظار مقایسه می‌شود. برای نرم‌افزارهای بزرگ و پیچیده ترکیب همه‌ی مقادیر ممکن ورودی، حالت‌های شروع و خروجی‌های قابل‌انتظار بسیار زیاد است که آزمونِ همه ترکیب‌ها در عمل غیرممکن است. آزمون جعبه سفید، که آن را آزمون مبتنی بر کد یا آزمون ساختاری نیز می‌نامند، متکی به دانش ساختاری کد و طراحی رویه‌ها برای تولید داده‌های آزمون است. در ضمن آزمون جعبه سفید هر دستورالعمل حداقل یک بار باید اجرا شود (آزمون پوشش کد) و همه‌ی مسیرهای برنامه دنبال شوند (آزمون پوشش مسیر).

۳.۲. رویکرد راستی‌آزمایی حین اجرا

رویکرد راستی‌آزمایی حین اجرا به جای اثبات و راستی‌آزمایی ویژگی‌ها در توصیف یا مدلی از نرم‌افزار، آنها را در زمان اجرای واقعی نرم‌افزار و در محیط حقیقی راستی‌آزمایی می‌کند. در واقع، این رویکرد مکملِ رویکرد راستی‌آزمایی توصیف و رویکرد آزمون است به این معنی که خطاهایی که در راستی‌آزمایی رسمی توصیف و آزمون نرم‌افزار کشف



شکل ۳. رویکرد راستی‌آزمایی حین اجرا.

نشده‌اند، مشخص می‌شوند. رویکرد راستی‌آزمایی حین اجرا، بر عکس دو رویکرد دیگر، عهده‌دار وظیفه‌ی قضاوت کلی برای همه‌ی اجراهای نرم‌افزار نیست، بلکه فقط اجرای جاری نرم‌افزار را راستی‌آزمایی می‌کند.

رویکرد راستی‌آزمایی حین اجرا، اساساً برای نرم‌افزارهای حیاتی و حساس به ایمنی به‌کارگرفته می‌شود زیرا استفاده از رویکرد راستی‌آزمایی توصیف و رویکرد آزمون در این گونه نرم‌افزارها با موانعی مواجه است. رویکرد راستی‌آزمایی توصیف که مبنای ریاضی و منطقی دارد، برای نرم‌افزارهای حیاتی و حساس به ایمنی که زیان‌های ناشی از شکست در آنها بسیار جدی است، یک امیدواری است. اما ناتوانی روش‌هایی مانند اثبات قضیه در اثبات ویژگی‌های تصمیم‌ناپذیر نرم‌افزارهای حیاتی پیچیده و بزرگ[۸] و همچنین زیادشدن تعداد حالت‌ها در روش بررسی مدل، موانع عمده‌ی موجود در استفاده از رویکرد راستی‌آزمایی رسمی است. همچنین، اثبات قضیه و بررسی مدل در سیستم‌های گسترده و با مقیاس بزرگ مانند سیستم‌های توزیع‌شده، دشوار و در برخی از موارد غیرممکن است. علاوه بر این، ممکن است برخی از شرایط محیط اجرا در زمان توصیف قابل پیش‌بینی نباشد.[۸]

۴.۲. تحلیل رویکردهای راستی‌آزمایی

برای آزمون نرم‌افزارهای حیاتی ــ که غالباً بزرگ و پیچیده‌اند ــ ترکیب همه‌ی مقادیر ممکن ورودی، و نیز حالت‌های شروع و خروجی‌های قابل‌انتظار بسیار زیاداند و عملاً آزمون همه‌ی این ترکیب‌ها غیرممکن است. به همین دلیل محققین معتقدند که آزمون اغلب نشانگر حضور خطاها، و نه غیبت آنها است.[۹و۱۰] علاوه بر این، آزمون عملی یک برنامه برای تمام داده‌ها غیرممکن است. بنابراین در این رویکرد با گزینش مجموعه‌ی مناسبی از داده‌ها مواجه هستیم و تولید مجموعه داده‌های مناسب برای آزمون و تعیین دقیق کفایت آزمون از مشکلات دیگر این روش است. آزمونی که اجرای برنامه را با داده‌های آزمایشی می‌آزماید، به‌تنهایی قادر به کشف و کاهش نرخ شکست درحدود حداکثر 10^{-4} شکست در ساعت است، در صورتی که در نرم‌افزارهای حساس به ایمنی حداکثر 10^{-9} شکست در ساعت قابل قبول است.[۵و۶]

دو مرور زیر برای هر نرم‌افزار به‌عنوان یک مؤلفه‌ی حساس به ایمنی در یک سیستم بزرگ لازم است[۱۰]: ۱. مرور هر پیمانه برای سازگاری طراحی الگوریتم و ساختار داده با رفتار توصیف شده؛ ۲. مرور کد برای سازگاری با طراحی الگوریتم و ساختار داده و این که آیا کامپایلر کد اجرایی را درست می‌سازد یا نه[۱۰]. همچنین برآورد شده است که به‌طور تقریب در هر ۱۰۰۰ خط از کد برنامه‌های سیستم‌های نرم‌افزاری بزرگ ۳/۳ خطا وجود دارد[۱۱]. علاوه بر این، سرایت مشکلات سخت‌افزاری به نرم‌افزار و یافتن به موقع خطاها نیز مسئله‌ی مهمی است. مشخص شده است که تقریباً ۱۰٪ از خطاهای نرم‌افزار و ۳۵٪ از شکست‌های نرم‌افزار دیر پیدا شده‌اند و علت آن‌ها خطاهای گذرایی (مانند خرابی داده‌ها) بوده‌اند که از قبل قابل تشخیص نبوده‌اند.[۱۲] موارد یاد شده و آنچه در بخش اول مقاله بیان شد، نشان‌دهنده‌ی لزوم راستی‌آزمایی حین اجرای نرم‌افزار است.

بنابراین پس از راستی‌آزمایی توصیف و آزمون نرم‌افزار، به‌منظور کشف خطاهای باقی‌مانده و جبران ضعف‌های یاد شده، لازم است نرم‌افزار را در زمان اجرای آن پایش و راستی‌آزمایی کنیم. در این رویکرد براساس قیود و ویژگی‌ها، راستی‌آزمایی ساخته می‌شود تا رفتار برنامه را در هنگام اجرای واقعی آن، در برابر قیود راستی‌آزمایی کند (شکل ۳).

پایش و راستی‌آزمایی حین اجرا[۱۱] که «راستی‌آزمایی پویا» نیز نام دارد، رویکردی سبک وزن است که در آن از اثبات‌کننده‌های قضیه یا بررسی‌کننده‌های مدل استفاده نمی‌شود. چون در این رویکرد مدل‌سازی محدود می‌شود، جلوه‌های ریاضی روش برای کاربر کاهش می‌یابد و آستانه‌ی مطالعه‌ی ریاضی مرتبط پایین می‌آید. این رویکرد از ویژگی‌های رویکرد راستی‌آزمایی توصیف ــ که در آن رفتار مورد انتظار نرم‌افزار راستی‌آزمایی می‌شود ــ و رویکرد آزمون ــ که در آن اجرای نرم‌افزار راستی‌آزمایی می‌شود ــ استفاده می‌کند اما با محدودساختن دامنه‌ی مسئله از ضعف‌های آنها می‌پرهیزد. این بدان مفهوم است که با محدودکردن خود به راستی‌آزمایی اجرای جاری برنامه (به جای راستی‌آزمایی همه‌ی اجراهای ممکن) از پیچیدگی‌های اثبات قضیه و بررسی مدل (مشکل رویکرد راستی‌آزمایی توصیف) و گزینش داده‌های آزمون مناسب از میان مجموعه داده‌های نامتناهی (مشکل رویکرد آزمون) اجتناب می‌کند و فقط به راستی‌آزمایی رفتار مورد انتظار در زمان اجرای واقعی برنامه می‌پردازد و اجتناب از حالت‌های مخاطره‌آمیز را مد نظر قرار می‌دهد.[۴و۱۳]

ویژگی‌های رویکرد راستی‌آزمایی حین اجرای نرم‌افزار عبارت است از: ۱. در زمان اجرای برنامه، تخلف از نیازها را می‌یابد و به کاربران کمک می‌کند تا عملیات ترمیمی را قبل از آنکه منجر به شکست نرم‌افزار شود، انجام دهند و در صورت عدم تخلف از نیازها اطمینان می‌دهد که اجرای جاری برنامه نسبت به نیازها صحیح است؛ ۲. اگر هنگام توصیف

نرم‌افزار، پیش‌بینی کامل شرایط محیط اجرایی و فیزیکی خارجی ناممکن باشد، راستی‌آزمایی حین اجرا قادر به بررسی امکان برقراری ویژگی‌ها در محیط اجرای واقعی است؛ ۳. از آنجا که کاربرد روش‌های راستی‌آزمایی فعلی برای بررسی توصیف‌های نرم‌افزاری در مقیاس بزرگ آسان نیست، یک راه حل برای بررسی این گونه نرم‌افزارها راستی‌آزمایی حین اجرای آنها است؛ ۴. هنگامی که توصیف نرم‌افزار به‌طور کامل در دسترس نیست، یا آنکه نرم‌افزار دارای مؤلفه‌های ثالثی[۱۲] است، تحلیل حین اجرا بسیار مفید خواهد بود؛ ۵. هنگامی که نمی‌توانیم یک نرم‌افزار حساس به ایمنی را به حد کافی بیازمائیم تا قابل اتکا شود، جمع‌آوری اطلاعات در ضمن اجرای نرم‌افزار و تحلیل آنها تنها راهکار ممکن است. راستی‌آزمایی حین اجرا علاوه بر این که در راستی‌آزمایی نرم‌افزار کاربرد دارد[۱۴]، برای ارزیابی کارایی[۱۵]، اشکال‌زدایی[۱۶] و کنترل سیستم[۱۷] نیز استفاده شده است. در پایان این بخش روش‌های تحلیل درستی را که به‌وسیله‌ی رویکردهای راستی‌آزمایی استفاده می‌شوند دسته‌بندی می‌کنیم.

روش‌های تحلیل درستی نرم‌افزار: به‌طور کلی استدلال درباره‌ی درستی نرم‌افزار را می‌توان به چهار روش انجام داد[۱۸]: ۱. استنتاج، که به‌وسیله‌ی رویکرد راستی‌آزمایی توصیف استفاده می‌شود و یک روش ایستا است و هدف آن این است که از نرم‌افزار، به‌عنوان یک واحد انتزاعی و کلی، به نتایج خاصی که ممکن است از اجرای آن به‌دست آید، برسد. این روشِ استدلال، نرم‌افزار را به‌صورت ایستا (یعنی بدون اجرا) تحلیل می‌کند تا استنتاج کند که اگر نرم‌افزار اجرا شود، چه نتایجی «می‌تواند» و چه نتایجی «نمی‌تواند» به‌دست آید؛ ۲. مشاهده، که به‌وسیله‌ی رویکرد راستی‌آزمایی حین اجرا استفاده می‌شود و یک روش پویا است و هدف آن بررسیِ اجرای واقعی نرم‌افزار به‌منظور کسب اطمینان از عدم رخدادهای مخاطره‌آمیز (قیود ایمنی) و وقوع رخدادهای لازم (قیود حیاتی) است؛ ۳. استقرا، که در رویکردهای آزمون و راستی‌آزمایی حین اجرا کاربرد دارد و هدف آن خلاصه‌کردن مشاهدات حاصل از نتیجه‌ی اجرای نرم‌افزار به یک انتزاع و قانون‌مندی کلی (یعنی موارد همیشه برقرار[۱۳]) است؛ ۴. آزمایش، که طی رویکرد آزمون استفاده می‌شود و هدف آن مشاهده‌ی اجرای آزمایشی برنامه به‌منظور تشخیص شرایطی است که موجب شکست برنامه و نتایج نادرست می‌شود. در رویکرد پیشنهادی، ما از مشاهده، که یک روش تحلیل پویاست، بهره می‌گیریم.

جدول ۱. ویژگی‌های روش‌های مختلف استدلال.

روش	ویژگی
استنتاج	استنتاج از انتزاع به واقعیت (چه می‌تواند و چه نمی‌تواند رخ دهد)
مشاهده	مشاهده‌ی رخدادها (چه باید و چه نباید رخ دهد)
استقرا	استقرا از مشاهدات (از آنچه که رخ داده است) به قانون‌مندی کلی
آزمایش	تعیین علل رخدادها (چرا چنین رخدادهایی حادث شده است؟)

جدول ۱ ویژگی‌های روش‌های مختلف استدلال را نشان می‌دهد. این روش‌ها دارای ساختار سلسله‌مراتبی هستند. برای مثال استقرا می‌تواند از مشاهده و استنتاج استفاده کند.

۳. رویکردهای مرتبط

رویکردهای راستی‌آزمایی حین اجرا را می‌توان به ارائه‌ی رویکردهایی برای راستی‌آزمایی رفتار اجرایی نرم‌افزار، و ارائه‌ی رویکردهایی برای ساخت راستی‌آزما حین اجرا تقسیم کرد. در هر دو نوع رویکرد، نیازها با یک رویکرد رسمی (معمولاً منطق‌های زمانی یا جبر) توصیف می‌شود تا حالات یا رخدادهای حین اجرای نرم‌افزار در برابر آنها، راستی‌آزمایی شود.

برای راستی‌آزمایی رفتار اجرایی نرم‌افزار رویکردهایی[۱۹و۲۰] معرفی شده‌اند که نیازهای سیستم را با روش مبتنی بر حالت توصیف کرده‌اند و به راستی‌آزمایی حالات اجرایی نرم‌افزار در برابر آنها پرداخته‌اند. از راستی‌آزمای Eagle که یک راستی‌آزمای مبتنی بر قاعده[۱۴] و بر مبنای منطق Eagle است[۲۰] استفاده می‌شود. این منطق که گسترشی از حساب μ است، سعی در پشتیبانی کاربر در توصیف نیازها برحسب انواع منطق‌های زمانی (گذشته، آینده، حال) دارد. اما Eagle یک راستی‌آزمای مبتنی بر حالت است و حالات برنامه را در برابر توصیف‌هایی که برحسب منطق Eagle نوشته شده است، راستی‌آزمایی می‌کند. به‌منظور گسترش Eagle برای پشتیبانی توصیف‌های مبتنی بر رخداد، منطق و ابزار جدیدی به نام HAWK[۲۱] برای برنامه‌های Java فراهم شده است. در این روش برای توصیف‌های سطح بالا و برای فعالیت‌های اجرایی برنامه از زبان (HAWK) استفاده می‌شود.

در یکی از رویکردهای ساخت راستی‌آزمای سیستم‌های بی‌درنگ[۲۲و۲۳] نیازها با منطق RTL[۱۵] که توانایی بالایی در بیان ویژگی‌های زمان واقعی به‌خصوص تقدم و تأخر رخدادها دارد، توصیف می‌شوند و از روی این توصیف‌ها، راستی‌آزمایی حین اجرا با تشکیل گراف بین ویژگی‌های ایمنی و حیاتی نیازها ساخته می‌شود تا حالات اجرایی برنامه را در گراف بررسی کند. یال‌های این گراف معرف تأخیر (با عدد منفی) و مهلت (با عدد مثبت) است. با RTL فقط می‌توان توصیف‌های مبتنی بر رخدادهای زمان واقعی را پشتیبانی کرد و بنابراین، استفاده از آن برای توصیف‌های مبتنی بر حالت مناسب نیست.

به ارائه رویکردی برای ساخت راستی‌آزما از روی نیازهایی که برحسب منطق زمانی خطی توصیف می‌شوند، رویکردی ارائه شده است.[۲۴و۲۵] این راستی‌آزما، دنباله‌یی از حالات حین اجرای نرم‌افزار را می‌گیرد و آن را در برابر توصیف‌های منطق زمانی راستی‌آزمایی می‌کند.[۲۶] دنباله‌ی حالات حین اجرای نرم‌افزار به‌صورت رشته‌یی از

عبارات منظم[16] ساخته شده[۲۶] و راستی‌آزما، یک ماشین حالت است که به راستی‌آزمایی این عبارات منظم می‌پردازد.

ما رویکردی مبتنی بر قانون را برای راستی‌آزمایی حین اجرا معرفی کرده‌ایم[۲۷] که در آن، نیازهای سیستم‌های رخدادگرا را با استفاده از نمودارهای حالت[17] Harel توصیف کرده‌ایم و سپس راستی‌آزما را از روی آن برحسب قوانین رخداد ـ شرایط ـ عمل[18] (ECA) ساخته‌ایم. در نمودارهای حالت Harel، گذار بین دو حالت می‌تواند با رخداد واقعه و برقراری شرط انجام شود. این نمودارها در نمایش حالت‌های هم‌روند و سلسله‌مراتبی توانمندند. قوانین ECA در پایگاه داده‌های فعال استفاده‌ی گسترده‌یی دارند و به‌منظور برخورد فعال در برابر رخدادها وضع شده‌اند.

۴. رویکرد پیشنهادی، EBV

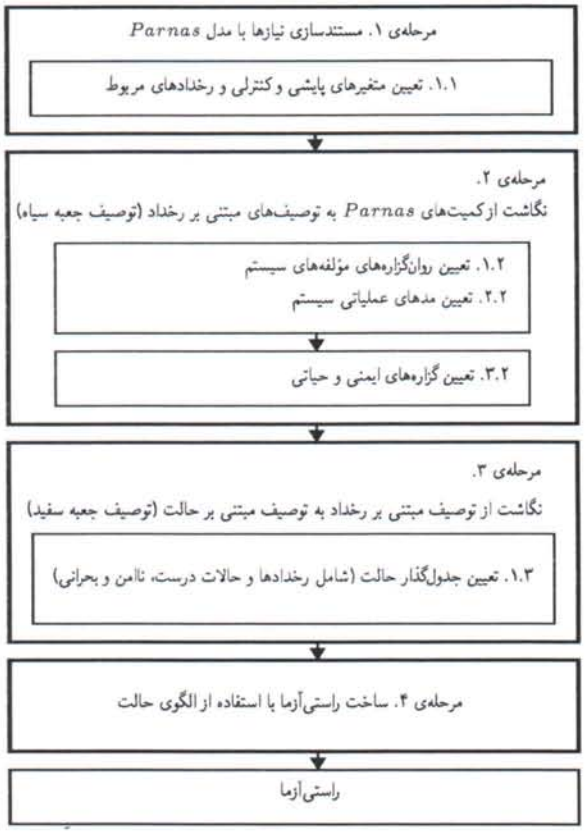
ما در این بخش رویکردی پویا و مبتنی بر رخداد (EBV) را به‌منظور ساخت خودکار راستی‌آزمای رفتار زمان اجرای برنامه‌های حساس به ایمنی، در چهار مرحله ارائه می‌دهیم (شکل۴). در بخش آتی، به‌منظور نشان‌دادن عملی‌بودن EBV، آن را برای سیستم حساس به ایمنیِ پمپ انسولین همراه و محیط (انسان)، از توصیف نیازها تا پیاده‌سازی راستی‌آزما، به کار می‌گیریم.

۱.۴. مرحله‌ی اول

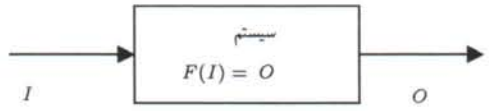
در این مرحله ابتدا با استفاده از روش Parnas مستندات نیازهای سیستم حساس به ایمنی را مدل می‌کنیم (مرحله‌ی یک عددی از شکل ۴). از آن‌جا که سیستم، نرم‌افزار و محیط به‌صورت جدانشدنی درهم آمیخته‌اند، یکی از بزرگ‌ترین امتیازات مدل‌سازی به‌روش Parnas این است که به ما اجازه می‌دهد تا با استفاده از متغیرهای پایشی و کنترلی، محیط، سیستم و نرم‌افزار را صریحاً از هم جدا کنیم.[۲۸] این جداسازی ما را برای توصیف نیازهای محیطی، با استفاده از منطق حساب رخداد در مرحله‌ی دوم آماده می‌سازد. در مقایسه با روش‌شناسی‌های رایج، رابطه‌های Parnas از قابلیت تأمین موجزتر، روشن‌تر و آسان‌تر نیازها و دانش دامنه (محیط) برای فهم کاربران و متخصصین دامنه، برخوردار است.

مستندسازی نیازها با روش Parnas: مدل سنتی یک سیستم نرم‌افزاری-سخت‌افزاری براساس یک مدل دو متغیره (متغیرهای ورودی به سیستم و متغیرهای خروجی از آن) و با دو فرض است: ۱. سیستم دارای ورودی‌ها و خروجی‌هاست؛ ۲. خروجی‌ها اغلب تابعی ریاضی از ورودی‌ها هستند (شکل ۵).

در این مدل، اولاً ورودی‌ها همان ورودی‌های فیزیکی واقعی به سیستم‌اند و توابع ریاضی اغلب پیچیده و برای توصیف سخت هستند



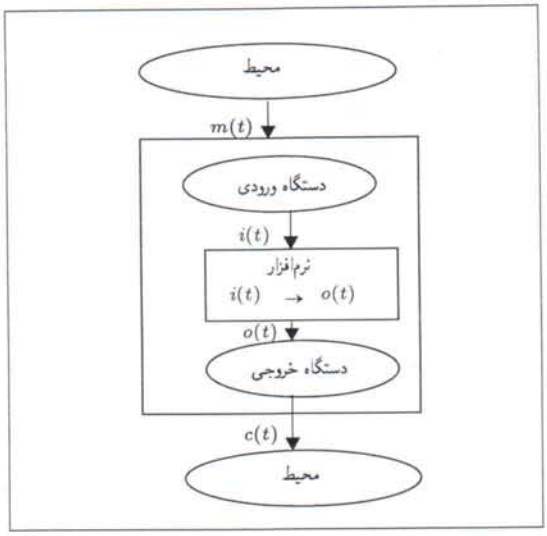
شکل ۴. رویکرد EBV برای ساخت خودکار راستی‌آزما.



شکل ۵. مدل دو متغیره از یک سیستم نرم‌افزاری ـ سخت‌افزاری.

و بنابراین مرور سیستم به‌وسیله‌ی متخصصین آن سخت می‌شود، و ثانیاً اغلب روابط بین سیستم و محیط بیرون آن واضح نیست؛ بنابراین به‌منظور توصیف و مستندسازی طبیعی، شفاف و ساده‌تر نیازهای یک سیستم برای تعامل با محیط بیرون، لازم است در توصیف آن به‌جای پرداختن به توابع ریاضی پیچیده، آن را با ویژگی‌های محیطی نشان دهیم. به همین منظور یک مدل چهار متغیره برای مستندسازی نیازهای یک سیستم معرفی شده است.[۲۹] این مدل نیازهای سیستم را برحسب چهار نوع متغیر: متغیر پایشی[19] $m(t)$، متغیر ورودی $i(t)$، متغیر خروجی $o(t)$ و متغیر کنترلی $c(t)$ بیان می‌کند (شکل ۶).[۳۰]

مجموعه‌ی کمیت‌های پایشی و کنترلی، کمیت‌های محیطی را تشکیل می‌دهند و برحسب فرهنگ کلمات دامنه‌ی سیستم بیان می‌شوند. کمیت‌های پایشی کمیت‌هایی هستند که کاربر می‌خواهد آنها را در سیستم اندازه بگیرد (مانند مقدار قند خون بیمار در سیستم کنترل قند بیمار)، و کمیت‌های کنترلی کمیت‌هایی هستند که کاربر می‌خواهد



شکل ۶. مدل ۴ متغیره‌ی Parnas از یک سیستم.[۲۹و۳۰]

به‌وسیله‌ی آنها محیط را کنترل کند (مانند مقدار انسولینی که باید برای کنترل قند بیمار قندی تزریق شود). به‌عبارت دیگر، کمیت‌های کنترلی آن دسته از کمیت‌های محیطی‌اند که سیستم بر آنها تأثیر می‌گذارد، و کمیت‌های پایشی آن دسته از کمیت‌های محیطی‌اند که بر رفتار سیستم تأثیر می‌گذارند. کمیت‌های ورودی که از طرف دستگاه ورودی برای نرم‌افزار و کمیت‌های خروجی که به‌وسیله نرم‌افزار برای دستگاه خروجی تعیین می‌شود، برحسب فرهنگ کلمات سخت‌افزار سیستم بیان می‌شود. بین این کمیت‌ها پنج نگاشت وجود دارد که در جدول ۲ نشان داده شده است.

نگاشت IN که توسط دستگاه ورودی انجام می‌شود، کمیت‌های پایشی محیطی ($m(t)$) را به مقادیر ورودی مناسب ($i(t)$) برای نرم‌افزار تبدیل می‌کند تا نرم‌افزار با استفاده از نگاشت SOF این مقادیر را به مقادیر مناسب خروجی ($o(t)$) تبدیل کند. نگاشت SOF باید رابطه‌ی R۱ (فرمول ۱) را ارضا کند تا قابل قبول باشد. مقادیر خروجی با

جدول ۲. نگاشت بین کمیت‌ها در مدل Parnas.[۲۹]

نگاشت	رابطه	شرح
IN	$m(t) \rightarrow i(t)$	توصیف رفتار دستگاه ورودی
SOF	$i(t) \rightarrow o(t)$	توصیف رفتار نرم‌افزار
OUT	$o(t) \rightarrow c(t)$	توصیف رفتار دستگاه خروجی
NAT	$m(t) \rightarrow c(t)$	توصیف قیود و محدودیت‌های محیطی (قوانین فیزیکی و طبیعی حاکم بر محیط) در غیاب سیستم
REQ	$m(t) \rightarrow c(t)$	توصیف رفتار مورد انتظار سیستم با قید اضافی نسبت به NAT

استفاده از نگاشت OUT و توسط دستگاه خروجی به کمیت‌های کنترلی ($c(t)$) تبدیل می‌شود تا به محیط اعمال شود. نگاشت NAT و نگاشت REQ رابطه‌ی بین کمیت‌های پایشی و کمیت‌های کنترلی را نشان می‌دهند. این نگاشت‌ها به‌ترتیب معرف قیود فیزیکی و طبیعی محیط، و رفتار مورد انتظار سیستم هستند؛ دامنه‌ی REQ نیز مجموعه‌ی بالاسری دامنه‌ی NAT است زیرا قیود بیشتری دارد. یک توصیف جعبه‌سیاه از رفتار مورد نیاز سیستم با دو رابطه‌ی REQ و NAT و یک توصیف جعبه‌سفید از آن (رفتار مورد نیاز نرم‌افزار) با IN و OUT مشخص می‌شود.

$$\begin{aligned}&\mathbf{R1}: \forall m(t), i(t), o(t), c(t)\ [IN(m(t), i(t)) \wedge \\ &SOF(i(t), o(t)) \wedge OUT(o(t), c(t)) \wedge \\ &NAT(m(t), c(t)) \Rightarrow REQ(m(t), c(t))] \\ &Domain(REQ) \supseteq Domain(NAT)\end{aligned}$$

فرمول ۱. رابطه‌ی R۱ برای راستی‌آزمایی نرم‌افزار در مدل Par-nas.[۲۹]

از آنجا که مؤلفه‌های سیستم، دستگاه‌های ورودی، خروجی و نرم‌افزار را شامل می‌شود، و مؤلفه‌ی نرم‌افزار عهده‌دار وظیفه‌ی کنترل دستگاه‌های ورودی و خروجی، محاسبات و نگاشت مقادیر از ورودی به خروجی است، راستی‌آزما وظیفه دارد نرم‌افزار را در کنترل درست دستگاه‌ها، مشتمل‌بر نگاشت بین مقادیر و محاسبات، راستی‌آزمایی کند. نگاشت‌ها عبارت‌اند از: ۱. نگاشت از کمیت‌های پایشی محیط به مقادیر ورودی نرم‌افزار به‌وسیله‌ی دستگاه ورودی؛ ۲. نگاشت از مقادیر ورودی به مقادیر خروجی به‌وسیله‌ی نرم‌افزار؛ ۳. نگاشت از مقادیر خروجی نرم‌افزار به کمیت‌های کنترلی محیط به‌وسیله‌ی دستگاه خروجی. در این راستا لازم است بررسی شود که آیا: ۱. داده‌ها از دستگاه ورودی درست و به‌موقع می‌رسند؟، ۲. رفتار و حالات مؤلفه‌ی نرم‌افزار به‌ازاء داده‌های ورودی معتبر درست است؟ ۳. مقادیر خروجی محاسبه‌شده به‌وسیله‌ی نرم‌افزار به‌درستی به‌وسیله دستگاه خروجی اعمال می‌شود؟

۲.۴. مرحله‌ی دوم

در این مرحله، با استفاده از منطق حساب رخداد[۳۱] و برحسب رخدادهای محیطی، یک توصیف جعبه‌سیاه ارائه می‌دهیم (مرحله‌ی ۲ از شکل ۴). در این توصیف، اعتبار داده‌های ورودی (مقدار و زمان)، محاسبات، نگاشت از دستگاه ورودی (IN) به دستگاه خروجی (OUT) و به‌کارگیری درست نتایج خروجی را با گزاره‌های حساب

رخداد بیان می‌کنیم. این توصیف، که یک توصیف سطح بالا و مبتنی بر رخداد است، انتزاعی از رفتار خارجی درست و مورد انتظار مؤلفه‌های سیستم را در برابر رخدادهای محیطی از نظر کاربر نشان می‌دهد و حالات و رفتار داخلی سیستم را پنهان می‌دارد.

در این توصیف، اولاً متغیرهای ایمنی مرتبط با مؤلفه‌ها (سخت‌افزار و نرم‌افزار) را تعیین می‌کنیم. این متغیرها، روان‌گزاره‌های [۲۰] حساب رخداد (یعنی φ_i از فرمول ۲) را تشکیل می‌دهند که درستی آنها در طول زمان تغییر می‌کند (مرحله‌ی ۱.۲ از شکل ۴). روان‌گزاره‌ها را از روی قیود رفتاری سیستم (قیود REQ در جدول ۲) به‌دست می‌آوریم. ثانیاً مدهای عملیاتی (یعنی α_i از فرمول ۲) مؤلفه‌های سیستم را با توجه به قیود REQ مشخص می‌کنیم (مرحله‌ی ۲.۲ از شکل ۴). در هر مد عملیاتی، مؤلفه در حال انجام یک عمل خاص است. در نهایت در این مرحله، با توجه به روان‌گزاره‌ها و مدهای عملیاتی به‌دست آمده، گزاره‌های ایمنی را برای راستی‌آزما می‌سازیم (مرحله‌ی ۳.۲ از شکل ۴).

حساب رخداد: حساب رخداد به‌عنوان یک زبان منطق رتبه اول [۲۱] برای نمایش رخدادها برحسب دوره‌های زمانی معرفی شده است.[۳۱] حساب رخداد انواع «رخداد یا کنش»، «روان‌گزاره» و «زمان» را شامل می‌شود. روان‌گزاره عبارت است از یک کمیت (مانند «درجه حرارت») یا یک گزاره (مانند «دستگاه خراب است») که مقدار یا درستی آن در طول زمان تغییر می‌کند. در حقیقت حساب رخداد بر پایه‌ی سه دامنه قرار دارد که در مجموعه‌ی فرمول ۲ ارائه شده است.

$$
\begin{aligned}
&۱ - \tau := <, = \text{ زمان با ساختار جبری با رابطه‌های ترتیب خطی}\\
&۲ - \Phi = \{\varphi_۱, \varphi_۲, \ldots \varphi_n\} \therefore \varphi_i \text{یک روان‌گزاره است}\\
&۳ - A = \{\alpha_۱, \alpha_۲, \ldots, \alpha_n\} \therefore \alpha_i \text{عمل یا رخداد است}
\end{aligned}
$$

فرمول ۲. دامنه‌های حساب رخداد.

دستگاه استنتاجی حساب رخداد، کنش‌ها (happens) را بر حسب زمان، و تأثیرات کنش‌ها(Treminates, Initiates) را به‌عنوان ورودی می‌گیرد، و حقایق (Holds at) را در خروجی تولید می‌کند. ما در این نوشتار از حساب رخداد ساده شده به‌نام SEC [۲۲] استفاده می‌کنیم[۳۲] که در آن نقاط زمانی به‌جای دوره‌های زمانی به‌کار می‌رود. SEC شامل یک اصل مرکزی S است (فرمول ۳) که دارای سه گزاره در فرض (Happens، Initiates و Terminates) و یک گزاره در نتیجه (HoldsAt) است. این گزاره‌ها در جدول ۳ شرح داده شده‌اند.[۳۳] α معرف رخداد یا عمل، β معرف روان‌گزاره و τ معرف نقطه‌یی از زمان است. اصل S بیان می‌دارد که روان‌گزاره‌ی β به‌واسطه‌ی رخداد $\alpha_۱$ در گذشته برقرار شده است (درست) و تا لحظه‌ی جاری، رخدادی رخ

جدول ۳. گزاره‌های پایه در حساب رخداد.[۳۳]

گزاره	معنی
$\text{Initiates}(\alpha, \beta)$	گزاره β پس از رخداد α برقرار است
$\text{Terminates}(\alpha, \beta)$	گزاره β پس از رخداد α برقرار نیست
$\tau_۱ \prec \tau_۲$	زمان $\tau_۱$ قبل از زمان $\tau_۲$ قرار دارد
$\text{Happens}(\alpha, \tau)$	رخداد α در زمان τ رخ می‌دهد
$\text{HoldsAt}(\beta, \tau)$	گزاره β در زمان τ برقرار است

نداده است که آن را نادرست ساخته باشد. اصول وابسته، گزاره‌های اصل مرکزی را تعریف می‌کنند.

$$
\begin{aligned}
&\underline{SAxiom:} HoldsAt(\beta, \tau) \leftarrow Happens(\alpha_۱, \tau_۱) \wedge\\
&Initiates(\alpha_۱, \beta) \wedge \tau_۱ < \tau \wedge\\
&\neg(\alpha_۲, \tau_۲) : [Happens(\alpha_۲, \tau_۲) \wedge\\
&Terminates(\alpha_۲, \beta) \wedge \tau_۱ < \tau_۲ \wedge \tau_۲ < \tau]
\end{aligned}
$$

فرمول ۳. اصل مرکزی S در SEC.[۳۲]

استفاده از حساب رخداد دو ویژگی را برای ما فراهم می‌سازد: ۱. یک زبان منطق رتبه‌ی اول است، بنابراین روشی متقن و قابل اتکا برای توصیف است و درصورت نیاز می‌توان به اثبات سازگاری بین توصیف‌ها پرداخت؛ ۲. به‌دلیل آن‌که عامل زمان و همچنین گزاره‌های کافی را برای نشان‌دادن تقدم و تأخر رخدادها دارد، از توانایی لازم برای توصیف محیط‌های مبتنی بر رخداد برخوردار است. لازم به ذکر است که در سیستم‌های رخدادگرا که نرم‌افزار هدایت‌گر سخت‌افزار در تصمیم‌گیری واکنش به رخدادها و محرک‌های محیطی است، جنبه‌های رفتاری نه‌تنها شامل وظیفه‌مندی، بلکه شامل جنبه‌های زمانی و ایمنی نیز هست. یک قید ایمنی می‌تواند به‌صورت یک فرمول منطقی بیان و سپس راستی‌آزمایی شود. این نوع قید معرف این است که واقعه‌ی بدی رخ نخواهد داد مانند واقعه بن‌بست در تخصیص منابع به‌وسیله‌ی سیستم عامل.

نگاشت‌های سه‌گانه: در این مرحله با سه نگاشت، توصیف‌های مبتنی بر مدل Parnas را به‌توصیف‌های مبتنی بر حساب رخداد تبدیل می‌کنیم و سپس قیود ایمنی را با استفاده از تبدیل‌های انجام‌شده به‌دست می‌آوریم. هر نگاشت، مقدار یک کمیت را به گزاره‌های حساب رخداد تبدیل می‌کند. نگاشت اول مقدار کمیت‌های پایشی را به گزاره‌های حقایق (HoldsAt)، نگاشت دوم رخدادهای محیطی را به گزاره‌های کنش (Happens) و نگاشت سوم حالات سیستم را به گزاره تأثیرات کنش (Initiates) تبدیل می‌کند.

در نگاشت اول (رابطه‌ی ۱)، هر مقدار از یک کمیت پایشی در مدل Parnasرا به یک گزاره‌ی HoldsAt در حساب رخداد نگاشت می‌کنیم که τ زمان، β روان‌گزاره و m_τ مقدار کمیت پایشی m در زمان τ است. نگاشت ۱ اظهار می‌دارد که برای هر کمیت پایشی یک روان‌گزاره وجود دارد که می‌توان مقدار آن را به یک گزاره‌ی HoldsAt روی آن روان‌گزاره نگاشت کرد.

$$\forall\tau\exists\beta : m_\tau \Rightarrow \text{HoldsAt}(\beta,\tau)\ \tau \in \text{Time},$$
$$\beta \in \text{Fluents} \qquad (۱)$$

m_τ یک مقدار حقیقی، یک مقدار صحیح یا یک مقدار منطقی است. در حقیقت برای ممکن بودن نگاشت ۱ لازم است نشان دهیم روشی وجود دارد که الف) می‌توان β را به دست آورد؛ ب) این β یکتا است (در نتیجه فقط یکHoldsAt داریم). به این منظور ابتدا مجموعه مقادیر ممکن m را مشخص می‌کنیم (برای مثال مجموعه مقادیر ممکن برای قند خون بیمار اعداد صحیح ۱ تا ۲۰ است). mزیرمجموعه‌یی غیر تهی از اعداد حقیقی (R)، اعداد صحیح (Z) یا مقادیر منطقی (L) است:

$$m = \{m_\circ, m_۱, ..., m_k\} :$$
$$m \neq \emptyset \wedge m \subset R \vee m \subset Z \vee m \subseteq L$$

سپس در مجموعه‌ی m یک (یا تعدادی) بازه از مقادیری که سیستم نسبت به آن بازه‌ها حساس است مشخص می‌کنیم. تغییرات بین بازه‌ها مهم است و باید به‌وسیله‌ی سیستم پایش شود. بعد به هر بازه یک متغیر منطقی نسبت می‌دهیم و آن راβ (روان‌گزاره) می‌نامیم (برای مثال در قند خون بیمار، low را به بازه‌ی ۱ تا ۴، normal را به بازه‌ی ۵ تا ۹ و high را به بازه‌ی ۱۰ تا ۲۰ نسبت می‌دهیم). به این ترتیب یک (یا تعدادی) β مرتبط با بازه‌های مقادیر مهم m به‌دست می‌آوریم.

با توجه به مطالب یاد شده، اولاً برای هر m قادر به تعیین حداقل یک β هستیم ($\exists\beta$) زیرا هر مجموعه m یک مجموعه‌ی غیرتهی است و حداقل یک بازه‌ی تک‌عنصری دارد که آن بازه را β (روان‌گزاره) در نظر می‌گیریم؛ ثانیاً در هر زمان فقط و فقط یک β_i درست وجود دارد (β_i درست یکتا است، در نتیجه یک HoldsAt داریم) زیرا با توجه به این که بازه‌ها هم‌پوشانی ندارند، هر m_τ در زمان τ فقط در یکی از بازه‌ها قرار دارد. در نتیجه β_i مرتبط با بازه‌یی است که $m\tau$ در آن قرار دارد، و ارزش آن true است، و $(\forall j, j \neq i)_j\beta_j$ منتسب به دیگر بازه‌ها است و ارزش آن false است. در حقیقت β_i درست یعنی HoldsAt (β_i,τ) برای هر m نشان‌دهنده‌ی این است که سایر β_jها برای این m نادرست‌اند، یعنی:

$$\forall j :\sim \text{HoldsAt}(\beta_j,\tau), j \neq i \ \square$$

برای مثال، کمیت پایشی قند خون بیمار، در هر زمان یکی از اعداد صحیح ۱ تا ۲۰ است که در آن با توجه به دو مقدار ۵ و ۱۰ (دو مقدار مرزی مهم برای سیستم) سه بازه برای کمیت پایشی قند خون بیمار تشکیل می‌شود.

$$m = \{۱, ..., ۵, .., ۱۰, .., ۲۰\} = \{۱..۴\} \cup \{۵..۹\} \cup$$
$$\{۱۰..۲۰\}$$
$$\{۱..۴\} \Rightarrow low, \{۵..۹\} \Rightarrow \text{normal}, \{۱۰..۲۰\} \Rightarrow \text{high}$$

که در هر زمان، تنها یکی از روان‌گزاره‌های low، normal و high درست است. ستون REQ از جدول ۶، نگاشت کمیت‌های پایشی محیط سیستم پمپ انسولین همراه را به روان‌گزاره‌ها نشان می‌دهد.

در نگاشت دوم (رابطه ۲)، هر رخداد محیطی را به یک گزاره‌ی Happens در حساب رخداد نگاشت می‌کنیم. رابطه‌ی ۲ بیان می‌دارد که برای هر رخداد محیطی در مدل Parnas (رخداد) یک روان‌گزاره وجود دارد که می‌توان آن را به گزاره‌ی Happens روی آن روان‌گزاره نگاشت کرد.

$$\text{event}(\tau) \Rightarrow \text{Happens}(\alpha,\tau) \equiv \text{Happens}(@T(\beta),\tau) \vee$$
$$\text{Happens}(@F(\beta),\tau) \qquad (۲)$$

یک رخداد نشان‌دهنده‌ی تغییر مقدار یک کمیت پایشی بین دو بازه است (خروج از بازه‌ی جاری و ورود به بازه‌ی بعد)، که موجب نادرستی β_i منتسب به بازه‌ی جاری و درستی β_j منتسب به بازه‌ی بعد می‌شود.

$$[\text{HoldsAt}(\beta_i,\tau) \wedge \sim \text{HoldsAt}(\beta_i,\tau+۱)] \wedge$$
$$[\sim \text{HoldsAt}(\beta_j,\tau) \wedge \text{HoldsAT}(\beta_j,\tau+۱)] \qquad (۳)$$

در هر لحظه، پایشگر فقط به یکی از دو روان‌گزاره‌ی β_i یا β_j یعنی رابطه‌ی ۴ یا رابطه‌ی ۵ توجه دارد.

$$[\ \text{HoldsAt}(\beta_i,\tau) \wedge \sim \text{HoldsAt}(\beta_i,\tau+۱)] \qquad (۴)$$
$$[\sim \text{HoldsAt}(\beta_j,\tau) \wedge \text{HoldsAt}(\beta_j,\tau+۱)] \qquad (۵)$$

حساب رخداد یک رویکرد رسمی مبتنی بر رخداد است، در نتیجه هر تغییر روان‌گزاره‌یی مانند β_i که مطابق یکی از دو رابطه‌ی ۴ یا ۵ بیان می‌شود را به یک رخداد در حساب رخداد نسبت می‌دهیم و آن را با رابطه‌ی ۶ بیان می‌کنیم.

$$[\text{HoldsAt}(\beta_i,\tau) \wedge \sim \text{HoldsAt}(\beta_i,\tau+۱)]$$
$$\equiv \text{Happens}(\alpha,\tau) \qquad (۶)$$

رابطه‌ی ۶ در حساب رخداد بیان می‌دارد که رخداد α در زمان τ رخ داده است و موجب تغییر روان‌گزاره β_i در زمان $\tau+1$ شده است. با توجه به این که نام رخداد یعنی α برای پایشگر مهم نیست، بلکه تأثیر تغییر یک β_i مورد توجه است، ماگزاره‌ی $\text{Happens}(\alpha,\tau)$ را برای رابطه‌های ۴ و ۵ به‌ترتیب با گزاره‌های $\text{Happens}(@\text{F}(\beta_i),\tau)$ و $\text{Happens}(@\text{T}(\beta_j),\tau)$ بیان می‌کنیم. گزاره‌ی $\text{Happens}(@\text{F}(\beta_i),\tau)$ بیان می‌دارد که در زمان τ رخدادی حادث شده است که موجب تغییر β_i از مقدار درست به مقدار نادرست در زمان $\tau+1$ شده است. برای مثال در قند خون بیمار برای پایشگر مهم است که بداند تغییر قند از normal به high در چه زمانی رخ می‌دهد تا انسولین تزریق شود، اما به‌رخدادی که موجب این تغییر شده است علاقه‌یی ندارد. □

در نگاشت سوم (رابطه‌ی ۷)، تغییر حالت سیستم را به‌گزاره‌ی Initiates در حساب رخداد نگاشت می‌کنیم. این نگاشت بیان می‌دارد که در هر تغییر حالت سیستم می‌توان روان‌گزاره‌یی (θ) در حساب رخداد یافت که تغییر حالت را به برقراری این روان‌گزاره (Initiates) نگاشت می‌کند.

$$OldMode \rightarrow NewMode \Rightarrow \text{Initiates}(@T(\beta),\theta) \quad (۷)$$

طرف چپ رابطه‌ی ۷ به این معنی است که سیستم حالت قبلی خود را ترک و به حالت جدید می‌رود. با توجه به این که سیستم در واکنش به رخدادهای محیطی تغییر حالت می‌دهد، اگر رخداد را با نگاشت آن در حساب رخداد یعنی Happens (رابطه‌ی ۲) جایگزین کنیم، آنگاه رابطه‌ی ۸ را داریم:

$$\text{Happens}(\alpha,\tau) \equiv (OldMode \rightarrow NewMode) \quad (۸)$$

برای هر حالت سیستم یک روان‌گزاره در نظر می‌گیریم که بودن در آن حالت را با گزاره‌ی Holds At نشان می‌دهیم. بنابراین رابطه‌ی ۸ به این معنی خواهد بود که اگر α رخ دهد، آنگاه θ (روان‌گزاره‌ی منتسب به حالت $NewMode$) برقرار می‌شود. در حساب رخداد بیان دیگری از گزاره‌ی $\text{HoldsAt}(\theta,\tau)$ به‌نام گزاره‌ی $Initiates(\alpha,\theta)$ وجود دارد که برقراری θ را وابسته به رخداد α می‌داند. بنابراین رابطه‌ی ۸ را می‌توانیم به‌صورت رابطه‌ی ۹ بنویسیم.

$$\text{Happens}(\alpha,\tau) \equiv \text{Initiates}(\alpha,\theta) \quad (۹)$$

اما همان‌طور که در رابطه‌ی ۶ گفته شد، به‌جای رخداد α تأثیر آن را در نظر می‌گیریم. بنابراین رابطه‌ی ۹ را می‌توان به‌صورت رابطه‌ی ۱۰ یا رابطه‌ی ۱۱ نوشت.

$$\text{Happens}(@T(\beta),\tau) \equiv \text{Initiates}(@T(\beta),\theta) \quad (۱۰)$$

$$\text{Happens}(@F(\beta),\tau) \equiv \text{Initiates}(@F(\beta),\theta) \quad (۱۱)$$

در قیود ایمنی، به رابطه‌های بین کمیت‌های پایشی و کنترلی ($\text{m(t)}\rightarrow\text{c(t)}$) که به‌وسیله‌ی نگاشت REQ در مدل Parnas (جدول ۲) ارائه می‌شود، توجه می‌کنیم. این رابطه‌ها نشان‌گر نیازهایی هستند که در آنها با تغییر مقدار یک کمیت پایشی، مقدار کمیت کنترلی باید به‌وسیله‌ی سیستم تعیین شود. اما مقدار کمیت کنترلی با قیودی همراه است (قیود نیازها) که باید رعایت شوند. برای محاسبه‌ی کمیت کنترلی، سیستم به حالت جدیدی می‌رود که اگر قید روی کمیت کنترلی رعایت نشده باشد، محاسبه نادرست و بنابراین سیستم در حالت ناامن قرار دارد.

برای پایش هر رابطه $\text{m(t)}\rightarrow\text{c(t)}$، یک جمله‌ی شرطی برحسب حساب رخداد ارائه می‌دهیم که فرض آن از گزاره‌های HoldsAt و Happens (نگاشت‌های ۱ و ۲) و نتیجه‌ی آن از گزاره‌ی Initiates (نگاشت ۷) تشکیل شده است. گزاره‌ی Initiates متناظر با حالتی است که سیستم برای محاسبه‌ی c(t) در آن قرار دارد. این جملات شرطی، قیود ایمنی و حیاتی‌اند. قیود ایمنی، جملاتی به‌صورت رابطه‌ی ۱۲ هستند که θ در جمله‌ی Initiates آنها برابر با Unsafe یا Critical است.

$$\text{if Happens}(@T(\beta),\tau) \wedge \sum_{*}^{n} \text{HoldsAt}(\gamma_i,\tau) \rightarrow$$

$$\text{Initiates}(@T(\beta),\theta)\ \theta \in \{\text{Unsafe},\ \text{Critical}\} \quad (۱۲)$$

که در آن عبارت $\sum_{*}^{n} \text{HoldsAt}(\gamma_i,\tau)$ به‌معنی تعداد صفر یا بیشتر گزاره $\text{HoldsAt}(\gamma_i,\tau)$ است. این قیود در مراحل بعدی، هسته‌ی راستی‌آزما را تشکیل می‌دهند. از آنجا که قیود ایمنی، قیودی خاص با یک جمله‌ی Initiates و حالات Unsafe و Critical هستند، راستی‌آزمایی حین اجرا تنها این قیود خاص را در نظر می‌گیرد و به استنتاج کلی درباره رفتار نرم‌افزار نمی‌پردازد (برخلاف رویکرد وارسی توصیف). □

۳.۴. مرحله‌ی سوم

در این مرحله، از روی گزاره‌های ایمنی و حیاتی مبتنی بر رخداد، توصیف رفتار درست نرم‌افزار را برحسب ماشین حالت به دست می‌آوریم. این توصیف، یک توصیف سطح میانی و جعبه سفید است که آن را با استفاده از روش SCR [۳۴] و به‌صورت جدول حالات نمایش می‌دهیم. در این جدول، گذار بین حالات را براساس روان‌گزاره‌های حساب رخداد تعیین می‌کنیم (مرحله‌ی ۳ از شکل ۴). این توصیف که حالات نرم‌افزار را نشان می‌دهد، مرجعی است برای راستی‌آزما تا از آن برای راستی‌آزمایی فعالیت‌های زمان اجرای نرم‌افزار استفاده کند. علاوه بر این، توصیف جعبه‌سفید، پل بین فعالیت‌های اجرایی نرم‌افزار

و رخدادهای محیطی را، که یکی از چالش‌های راستی‌آزما است، نیز فراهم می‌کند.

روش SCR : این روش که بر پایه‌ی مدل چهار متغیره‌ی Par-nas و Madey بنا شده است از یک استخوان‌بندی رسمی برخودار است. SCR که به‌وسیله‌ی آزمایشگاه‌های تحقیقاتی نیروی دریایی آمریکا تدوین شده است، برای سیستم‌های الکترونیک هواپیما، کنترل روشنایی ساختمان، سیستم ارتباطات زیردریایی و مؤلفه‌های حساس به ایمنی کارخانه انرژی هسته‌یی Darlington در کانادا به‌کار گرفته شده است.[۳۴] علاوه بر این تعدادی از سازمان‌های صنعتی، مانند آزمایشگاه‌های بل وگرومن، این روش را برای توصیف نیازهای سیستم‌های عملی پذیرفته‌اند.[۳۴] در روش SCR، توصیف سیستم با ۳ جدول گذار حالت، رخداد، و شرط مشخص می‌شود. جدول گذار حالت مشتمل‌بر گذارهایی است که به‌دلیل بروز رخدادها، سیستم را از حالت جاری به حالت جدید تغییر می‌دهد. هر تغییر در مقادیر کمیت‌های پایشی محیط، یک رخداد است و باعث تغییر حالت سیستم می‌شود. از آن‌جا که تغییر حالت سیستم باعث تولید یک یا چند خروجی در محیط می‌شود، جدول رخداد برای نشان‌دادن ارتباط رخدادها و حالات با خروجی‌ها، و جدول شرط برای نشان‌دادن ارتباط شروط و حالات با خروجی‌ها تعیین می‌شود. به عبارت دیگر این جدول‌ها مشخص می‌کنند که سیستم تحت چه رخدادها، شروط و حالاتی کمیت‌های خروجی را تغییر می‌دهد. ما در این نوشتار از شکل ساده‌ی SCR که جدول گذار حالت را شامل می‌شود استفاده می‌کنیم. جدول ۴ شکل کلی جدول گذار حالت[۳۵] و جدول ۵ کاربردی از آن را برای حالات مختلف یک واکنشگر نشان می‌دهد که در آن فشار آب به‌منظزله‌ی کمیت پایشی‌واکنشگر است و تغییر در مقدار آن موجب رخداد می‌شود. مثلاً ردیف اول جدول ۵ نشان می‌دهد که اگر واکنشگر در حالت فشار خیلی پایین قرار داشته باشد و فشار آب از مقدار «کم» بیشتر شود، آنگاه واکنشگر به حالت مجاز گذار می‌کند.

جدول ۴. جدول گذار حالت در SCR.[۳۵]

OldMode	Event	NewMode
m_1	$e_{1,1}$	$m_{1,1}$
	$e_{1,2}$	$m_{1,2}$
	...	...
	$e_{1,k1}$	$m_{1,k1}$
...	...	...
m_n	$e_{n,1}$	$m_{n,1}$
	$e_{n,2}$	$m_{n,2}$
	...	...
	$e_{n,kn}$	$m_{n,kn}$

جدول ۵. جدول گذار حالت یک واکنشگر در SCR.[۳۵]

OldMode	Event	NewMode
ToolLow	فشار آب ≤ کم	Permitted
Permitted	@T(WaterPres ≥Permit)	High
Permitted	@T(WaterPres < Low)	ToolLow
High	@T(WaterPres < Permit)	Permitted

نگاشت‌های هفت‌گانه: در این مرحله گزاره‌های حساب رخداد و ترکیب آنها را به عناصر جدول حالت در SCR نگاشت می‌کنیم. در حقیقت روشی مرکب از هفت نگاشت ارائه می‌دهیم که توصیف‌های مبتنی بر رخداد را که با حساب رخداد بیان می‌شود، به توصیف‌های مبتنی بر حالت تبدیل می‌کند. توصیف مبتنی بر حالت را با الهام از جدول گذار حالت در SCR بیان می‌کنیم. با این جدول (توصیف جعبه‌سفید) نگاشت بین حالات و رخدادهای سطح بالا (توصیف جعبه‌سیاه) فراهم می‌شود.

ما در جدول ۶، هفت رابطه مشخص کرده‌ایم که براساس آنها می‌توان گزاره‌های حساب رخداد را به‌عناصر جدول حالت نگاشت کرد و به این وسیله قیود ایمنی را برحسب عناصر جدول حالات تعیین کرد. برای مثال ردیف اول جدول ۶ بیان می‌دارد که جمله‌ی «گزاره φ در زمان τ برقرار نیست و در این زمان رخدادی رخ نمی‌دهد که گزاره φ در لحظه‌ی بعد نیز برقرار باشد» معادل با این جمله است که «شرط φ در حالت جاری و حالت جدید باید «نادرست» باشد». هم‌چنین عبارت $(\varphi = f) \vee @\mathrm{T}(\varphi)$در ردیف سوم جدول ۶ بیان می‌دارد که شرط φ حداقل در حالت جاری باید «نادرست» باشد. اکنون به بررسی درستی نگاشت‌های جدول ۶ می‌پردازیم:

۱. **نگاشت ردیف اول** جدول ۶ به‌صورت رابطه‌ی ۱۳ است:

$$[\sim \mathrm{HoldsAt}(\varphi,\tau) \wedge \sim \mathrm{Happens}(@T(\varphi),\tau)] \Rightarrow \varphi = f \quad (۱۳)$$

طبق تعریف، گزاره‌ی $\sim \mathrm{Happens}(\ @\ \mathrm{T}(\varphi),\tau)$ به‌این معنی است که در زمان τ رخدادی رخ نمی‌دهد که موجب شود مقدار روان‌گزاره‌ی φ در زمان $\tau+1$ برابر با «درست» شود، یعنی داریم: $\sim \mathrm{HoldsAt}(\varphi,\tau+1)$. بنابراین رابطه‌ی ۱۳ را می‌توان به‌صورت رابطه‌ی ۱۴ نوشت:

$$[\sim \mathrm{HoldsAt}(\varphi,\tau) \wedge \sim \mathrm{HoldsAt}(\varphi,\tau+1)] \Rightarrow \varphi = f \quad (۱۴)$$

اگر چنین فرض کنیم که زمان τ زمانی است که سیستم در حالت جاری قرار دارد و زمان $\tau+1$ زمانی است که سیستم در حالت جدید قرار می‌گیرد، آنگاه $\sim \mathrm{HoldsAt}(\varphi,\tau)$ به این معنی است که مقدار φ

جدول ۶. نگاشت گزاره‌های حساب رخداد به عناصر جدول حالت در EBV.

عنصر جدول	گزاره حساب رخداد
$\varphi = f$	$[\sim \text{HoldsAt}(\varphi,\tau) \wedge \sim \text{Happens}(@T(\varphi),\tau)]$
$\varphi = t$	$[\text{HoldsAt}(\varphi,\tau) \wedge \sim \text{Happens}(@F(\varphi),\tau)]$
$(\varphi = f) \vee @T(\varphi)$	$\sim \text{HoldsAt}(\varphi,\tau)$
$(\varphi = t) \vee @F(\varphi)$	$\text{HoldsAt}(\varphi,\tau)$
Enter mode φ	$\text{Initiates}(\alpha,\varphi)$
$\varphi = @T(\varphi)$	$\text{Happens}(@T(\varphi),\tau)$
$\varphi = @F(\varphi)$	$\text{Happens}(@F(\varphi),\tau)$

در حالت جاری برابر با «نادرست» و $\sim \text{HoldsAt}(\varphi,\tau+1)$ به این معنی است که مقدار φ در حالت جدید نیز برابر با «نادرست» است که این تعریف $\varphi = f$ در SCR است. علت حضور گزاره‌ی Happens در جمله‌ی دوم سمت چپ رابطه‌ی ۱۳ این است که قضاوت درباره‌ی لحظه‌ی بعدی برمبنای لحظه‌ی جاری انجام می‌شود.□

۲. نگاشت ردیف دوم جدول ۶ به‌صورت رابطه‌ی ۱۵ است که استدلال برای آن مشابه ردیف اول جدول ۶ است:

$$[\text{HoldsAt}(\varphi,\tau) \wedge \sim \text{Happens}\ (@\ \text{F}(\varphi),\tau)] \Rightarrow \varphi = \text{t} \qquad (۱۵)$$

۳. نگاشت ردیف سوم جدول ۶ به‌صورت رابطه‌ی ۱۶ است.

$$\sim \text{HoldsAt}(\varphi,\tau) \Rightarrow [(\varphi = f) \vee @\text{T}(\varphi)] \qquad (۱۶)$$

گزاره‌ی $\sim \text{HoldsAt}(\varphi,\tau)$ بیان می‌دارد که در زمان τ روان‌گزاره‌ی φ برقرار نیست. با توجه به این‌که درباره‌ی زمان $\tau+1$ اظهاری نداریم، می‌توان دو وضعیت ۱۷ و ۱۸ را پیش‌بینی کرد:

$$\sim \text{HoldsAt}(\varphi,\tau) \wedge \sim \text{HoldsAt}(\varphi,\tau+1) \qquad (۱۷)$$

$$\sim \text{HoldsAt}(\varphi,\tau) \wedge \text{HoldsAt}(\varphi,\tau+1) \qquad (۱۸)$$

وضعیت ۱۷، معرف $\varphi = f$، و وضعیت ۱۸، معرف $@\text{T}(\varphi)$است. بنابراین می‌توان رابطه‌ی ۱۹ را داشت:

$$\sim \text{HoldsAt}(\varphi,\tau) \Rightarrow \varphi = \text{f} \vee @\text{T}(\varphi) \qquad (۱۹)$$

□

۴. نگاشت ردیف چهارم جدول ۶ به‌صورت رابطه‌ی ۲۰ است که استدلال برای آن مشابه ردیف سوم جدول ۶ است.

$$\text{HoldsAt}(\varphi,\tau) \Rightarrow (\varphi = \text{t}) \vee @\text{F}(\varphi) \qquad (۲۰)$$

□

۵. نگاشت ردیف پنجم جدول ۶ به‌صورت رابطه‌ی ۲۱ است.

$$\text{Initiates}(\alpha,\varphi) \Rightarrow \text{Enter mode } \varphi \qquad (۲۱)$$

رابطه‌ی ۲۱ بیان می‌دارد که هر روان‌گزاره وابسته به یک رخداد را می‌توان به آغاز یک مد عملیاتی نگاشت کرد. روان‌گزاره وابسته‌ی به یک رخداد، روان‌گزاره‌یی است که برقراری آن بسته به وقوع آن رخداد است. گزاره‌ی Initiates بیان دیگری از گزاره‌ی HoldsAt است که برقراری روان‌گزاره‌ی φ را بسته به رخداد α می‌داند. اگر رخداد α در زمان τ رخ دهد، در این‌صورت، φ در زمان $\tau+1$ برقرار می‌شود. بنابراین رابطه‌ی ۲۱ را می‌توان به‌صورت رابطه‌ی ۲۲ نوشت.

$$\text{Initiates}(\alpha,\varphi) \equiv \text{HoldsAt}(\varphi,\tau+1) \Rightarrow \text{Enter mode } \varphi \qquad (۲۲)$$

اگر φ را یک مد عملیاتی برای سیستم در نظر بگیریم، رابطه‌ی ۲۲ بیان می‌دارد که در زمان τ رخداد φ حادث شده است که موجب برقراری مد عملیاتی φ در زمان $\tau+1$ شده است و این به‌معنی ورود به یک مد عملیاتی جدید است.□

۶. نگاشت ردیف ششم جدول ۶ به‌صورت رابطه‌ی ۲۳ است.

$$\text{Happens}\ (@\ \text{T}(\varphi),\tau) \Rightarrow \varphi = @\text{T}(\varphi) \qquad (۲۳)$$

گزاره‌ی $\text{Happens}(@\text{T}(\varphi),\tau)$ بیان می‌دارد که رخدادی در زمان τ حادث شده است که موجب تغییر روان‌گزاره‌ی φ از «نادرست» در زمان τ (یعنی $\sim \text{HoldsAt}(\varphi,\tau)$) به «درست» در زمان $\tau+1$ (یعنی $\text{HoldsAt}(\varphi,\tau+1)$) شده است. بنابراین رابطه‌ی ۲۳ را می‌توان به‌صورت رابطه‌ی ۲۴ نوشت.

$$[\sim \text{HoldsAt}(\varphi,\tau) \wedge \mathit{HoldsAt}(\varphi,\tau+1)] \Rightarrow \varphi = @T(\varphi) \qquad (۲۴)$$

اگر چنین فرض کنیم که زمان τ زمانی است که سیستم در حالت جاری قرار دارد و زمان $\tau+1$ زمانی است که سیستم در حالت جدید قرار می‌گیرد، آنگاه به‌تعریف $@\text{T}(\varphi)$ در روش SCR می‌رسیم که کمیت φ در حالت جاری برقرار نیست اما در حالت جدید برقرار می‌شود.□

۷. نگاشت ردیف هفتم جدول ۶ به‌صورت رابطه‌ی ۲۵ است که استدلال برای آن مشابه ردیف ششم جدول ۶ است.□

$$\text{Happens}\ (@\ \text{F}(\varphi),\tau) \Rightarrow \varphi = @\text{F}(\varphi) \qquad (۲۵)$$

۴.۴. مرحله چهارم

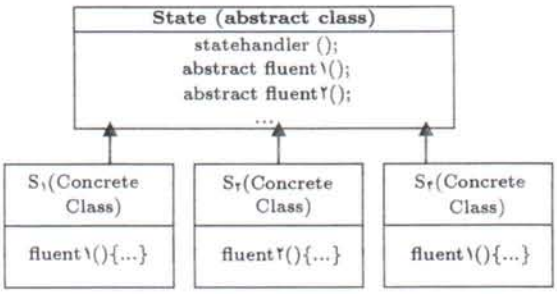
در این مرحله، با بهره‌گیری از الگوی طراحی حالت به پیاده‌سازی راستی‌آزما از روی جدول حالت می‌پردازیم. الگوی طراحی حالت برای پیاده‌سازی شیءگرای ماشین حالت به‌کار گرفته می‌شود و برای اشیایی مناسب است که رفتار آنها برحسب حالتی که در زمان اجرا می‌گیرند، تغییر می‌کند.[۳۶و۳۷] این رفتار برحسب نوع رخدادهایی که در زمان اجرا رخ می‌دهد متفاوت است. همچنین این الگو نشان‌دهنده‌ی چندریختی حین اجرا[۲۳] است که در آن رفتار زمان اجرای اشیا تابع حالات آنهاست. بنابراین جدول حالت به‌عنوان مرجع و الگوی طراحی حالت به‌عنوان روش پیاده‌سازی حالات استفاده می‌شود.

پیاده‌سازی راستی‌آزما با الگوی طراحی حالت

برای بهره‌گیری از الگوی طراحی حالت در پیاده‌سازی راستی‌آزما (مورد ۴ از شکل ۴)، جدول حالت (جدول ۴) را که در مرحله‌ی سوم با استفاده از روش SCR به دست آمد، به‌عنوان مرجع حالات درست در نظر می‌گیریم و الگوی طراحی حالت را برای پیاده‌سازی آن به‌کار می‌گیریم تا به این وسیله راستی‌آزما را بسازیم. راستی‌آزما، فعالیت‌های زمان اجرای نرم‌افزار را در برابر حالات درست، که متناظر با رخدادهای سطح بالا است، راستی‌آزمایی می‌کند.

الگوی طراحی حالت که در رویکرد شیءگرا برای پیاده‌سازی ماشین حالت به‌کار گرفته شده است، برای اشیایی مناسب است که رفتار آنها برحسب حالتی که در زمان اجرا می‌گیرند، تغییر می‌کند. این رفتار بر حسب نوع رخدادهایی که در زمان اجرا رخ می‌دهد، متفاوت است. همچنین این الگو نشان‌دهنده‌ی چندریختی حین اجراست که در آن رفتار زمان اجرای اشیاء تابع حالات آنهاست. الگوی طراحی حالت به‌عنوان یک الگوی مرجع برای پیاده‌سازی انواع ماشین حالت مورد استفاده قرار گرفته است.[۳۸] برای پیاده‌سازی الگوی طراحی حالت، یک کلاس حالت انتزاعی که نقش واسط را دارد، و تعدادی کلاس واقعی که هرکدام پیاده‌سازی‌های جداگانه‌یی از برخی یا همه‌ی متدهای کلاس انتزاعی را دارند، تعریف می‌شود. هر کلاس واقعی، معرف یکی از حالات شی و متدهای این کلاس، معرف رفتاری است که شی در آن حالت دارد (شکل ۷).[۳۶]

جدول ۷ الگوی طراحی حالت را برای حالات ناامن نشان می‌دهد. چنان که مشاهد می‌شود، برای حالات ناامن در جدول فرضی حالات (جدول ۷) یک کلاس واقعی، و برای تغییر هر روان‌گزاره (وقوع یک رخداد)، که درستی آن از حالت جاری به حالت جدید تغییر می‌کند، یک پیاده‌سازی (تابع) جداگانه در کلاس واقعی منظور می‌شود. تغییر روان‌گزاره‌ها با @T یا @F مشخص شده است. برای هر مقدار ثابت t یا f یک شرط در متد منظور می‌شود.



شکل ۷. الگوی طراحی حالت برای حالات ناامن جدول ۷.

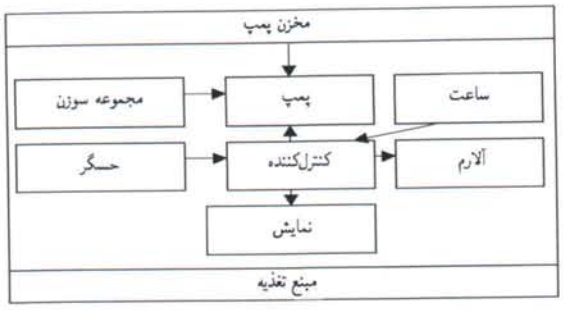
جدول ۷. جدول فرضی حالات.

حالت جاری	روان‌گزاره‌ی ۱	روان‌گزاره‌ی ۲	...	حالت جدید
S۱	@T	t		ناامن
S۲		@T		ناامن
S۳	@T	@T	...	امن
S۴	@T	t	...	ناامن

۵. توصیف سیستم حساس به ایمنی پمپ انسولین همراه

در این بخش در مورد سیستم حساس به ایمنی پمپ انسولین همراه[۳۹] بحث خواهیم کرد و سپس در بخش‌های بعدی هر یک از مراحل رویکرد EBV را برای آن به‌کار می‌گیریم. از این سیستم برای بیمارانِ مبتلا به قند نوع ۱ استفاده می‌شود. این بیماری یک بیماری مادام‌العمر است که به‌دلیل توقف تولید انسولین توسط لوزالمعده به‌وجود می‌آید. انسولین هورمونی است که موجب انتقال قند (گلوکز) از خون به سلول‌های بدن، و تولید انرژی می‌شود. اگر قند از خون به سلول‌ها منتقل نشود، مقدار آن در خون از سطح معینی بالاتر می‌رود و همچنین کارکرد سلول‌ها مختل می‌شود. قند نوع ۱ که قند جوانی نام دارد ممکن است در هر سنی بروز کند، اگرچه معمولاً در کودکان و جوانان و قبل از سن ۳۰ ایجاد می‌شود. به‌منظور کنترل دائم بیماری قند نوع ۱، به‌خصوص در کودکان و نوجوانان، که به‌دلیل کمی سن نمی‌توان رژیم غذایی خاصی برای آنها وضع کرد و باید اجازه‌ی فعالیت دائم را به آنها داد، پمپ مینیاتوری انسولین به بدن بیمار متصل می‌شود تا همیشه همراه بیمار باشد و قندخون را به‌طور پیوسته کنترل، و در صورت لزوم انسولین را تزریق کند.

سیستم پمپ انسولین (شکل ۸) در بخش کنترل‌کننده دارای نرم‌افزاری است که سیستم را کنترل می‌کند. سیستم در زمان شروع، در حالت STARTUP قرار دارد و سپس رویه‌ی TEST برای آزمون سخت‌افزار آن اجرا می‌شود. همچنین این رویه به‌منظور کسب اطمینان بالا از درستی عمل سخت‌افزار، در هر ساعت ۴ بار اجرا می‌شود



شکل ۸. معماری سیستم پمپ انسولین.

(راستی‌آزمایی سخت‌افزار). پس از خودآزمایی اگر نقصی پیدا نشود سیستم به حالت RUN می‌رود. هر زمان که مخزن انسولین در پمپ یا هر قطعه دیگر با قطعه‌ی جدیدی جایگزین شود، پمپ RESET می‌شود. همچنین اگر پس از اجرای رویه‌ی TEST نقصی در سخت‌افزار پیدا شود یا پمپ RESET شود، پمپ در وضعیت Out Of Action قرار می‌گیرد. حسگر ورودی باید هر ۱۰ دقیقه یک بار قند خون را اندازه بگیرد و کنترل‌کننده به‌منظور محاسبه‌ی مقدار انسولینی که باید تزریق شود، از ۳ مقدار قند اندازه‌گیری شده‌ی آخر استفاده می‌کند. با توجه به مقادیر اندازه‌گیری‌شده، وضعیت بیمار در یکی از سه حالت قند پائین، قند معمولی و قند بالا تشخیص داده می‌شود. برای مخزن پمپ و مجموعه سوزن حسگرهایی قرار داده شده است که اتصال یا عدم اتصال آنها را نشان می‌دهد. همچنین هنگام تزریق انسولین، مخزن پمپ باید مقدار انسولین کافی را برای یک وعده داشته باشد. خطاهای سیستم را می‌توان به سه دسته تقسیم کرد:

الف) خطاهایی که در رویه‌ی TEST می‌توان تشخیص داد:

- نقص یا نبود مجموعه سوزن،
- نقص یا نبود مخزن انسولین، اتمام یا ناکافی‌بودن انسولین در مخزن برای یک وعده تزریق،
- نقص یا افت باطری،
- نقص کنترل‌کننده‌ی پمپ، نقص حسگر اندازه‌گیر قند خون،
- ازکارافتادن ساعت

ب) خطاهای نرم‌افزاری که به‌وسیله‌ی رویه‌ی TEST قابل تشخیص نیست و باید راستی‌آزما آن را تشخیص دهد:

- خطاهای محاسباتی،
- خطاهای الگوریتمی،
- خطای ساعت

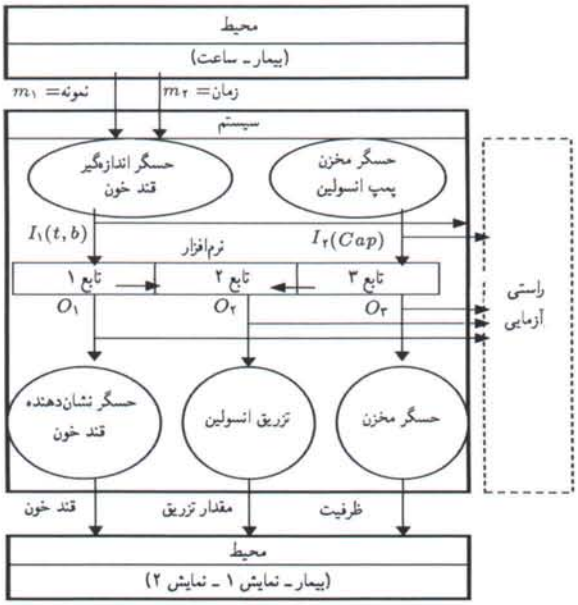
ج) خطاهایی که فقط بیمار می‌تواند تشخیص دهد:

- واکنش آلرژی،
- تداخل با تجهیزات استفاده شده‌ی دیگر مانند دستگاه تنظیم‌کننده‌ی[۲۴] ضربان قلب.

خطاهای سیستم پمپ انسولین به‌طورکامل در ضمیمه‌ی ۱ ارائه شده است. برگ‌های انتهایی درخت، نشان‌گر خطاهای نرم‌افزاری (محاسباتی و الگوریتمی) هستند که باید راستی‌آزمایی شود.

۱.۵. مستندسازی نیازهای سیستم پمپ انسولین همراه

در این بخش، ابتدا مدل Parnas را به‌منظور تهیه‌ی مستندات نیازهای سیستم پمپ انسولین همراه می‌سازیم (شکل ۹) و سپس این مستندات را با تعیین کمیت‌های پایشی و کنترلی مشخص می‌کنیم (جدول ۸) مدل Parnasسیستم را در دو بخش محیط و سیستم نشان می‌دهد که در آن حسگر اندازه‌گیر و حسگر مخزن کمیت‌های پایشی m_1 و m_2 را به مقادیر I_1 و I_2 تبدیل می‌کنند (نگاشت IN در جدول ۲) و مقادیر ورودی به نرم‌افزار را می‌سازند. برای مثال، I_1 مقدار صحیح خاصی است که از تبدیل قند بیمار برحسب gr/ml به دست آمده است. تابع ۱، تابع ۲ و تابع ۳ مجموعه‌ی توابع نرم‌افزار هستند که مقادیر O_1،O_2، و O_3 را برای دستگاه‌های خروجی محاسبه می‌کنند (نگاشت SOF در جدول ۲). مثلاً O_2 مقدار انسولین قابل تزریق را مشخص می‌کند. دامنه‌ی روابط REQ(جدول ۲) مجموعه مقادیر مجاز زمان و قند خون (m_1 و m_2 در شکل ۹) و برد آن مجموعه مقادیر مجاز انسولین (dose در شکل ۹) را مشخص می‌کند. روابط REQ که قیود روی کمیت‌های پایشی و کنترلی را نشان می‌دهد، در جدول ۸ مشخص شده‌اند. ستون‌های ۲ و ۳ از جدول ۸، کمیت‌های پایشی و کنترلی و ستون ۵ آن، روابط REQ را برای این کمیت‌ها نشان می‌دهد.



شکل ۹. مدل Parnas سیستم پمپ انسولین همراه.

جدول ۸. متغیرهای پایشی (M) و کنترل (C) سیستم پمپ انسولین همراه.

متغیر	M	C	نوع	REQ
m_{Time}	√		Real	bool Run-out $=(m_{Time} > ۱۰)$
$m_{GetSample}$	√		{T,F}	-
$m_{BloodSugar}$	√		{۱..۲۰}	bool Low $= (m_{BloodSugar} <$ Norm), bool High $= (m_{BloodSugar} >$ Norm)
c_{Dose}		√	Real	bool NDose $= (۰ < c_{Dose} <= ۵)$ bool Ovr $= (c_{Dose} > ۵)$
m_{Cap}	√		{۱..۱۰۰}	$boolDosAv۱ = (m_{Cap} >= NDose)$
$c_{Deliver}$		√	{T,F}	$boolHigh \wedge NDose \wedge DosAv۱$

متغیر m_{Time} برای پایش زمان است که در صورت گذشتن از ۱۰ دقیقه، Run-out رخ می‌دهد. متغیر $m_{GetSample}$ برای پایش شروع نمونه‌گیری است. متغیر $m_{BloodSugar}$ برای پایش قند خون است که اگر مقدار آن کم‌تر از معمول باشد، رخداد وضعیت قند پایین (Low) و اگر بیشتر از معمول باشد، رخداد وضعیت قند بالا (High) را داریم. متغیر c_{Dose} برای کنترل مقدار انسولینی است که باید تزریق شود. اگر این مقدار کم‌تر از ۵ باشد، وضعیت نرمال (N_{Dose}) است، در غیر این صورت رخداد Ovr را داریم. m_{Cap} برای پایش مقدار انسولین موجود مخزن است که اگر بیشتر یا مساوی مقدار انسولین تحویلی باشد، رخداد DosAvl را داریم. متغیر $c_{Deliver}$ برای کنترل شروع تحویل انسولین است که در صورت تحقق سه رخداد: افزایش قند؛ تعیین درست مقدار انسولین؛ وجود انسولین کافی، ممکن است رخ دهد.

۲.۵. توصیف جعبه سیاه سیستم پمپ انسولین همراه

در این بخش با استفاده از حساب رخداد، سیستم پمپ انسولین همراه را به‌صورت جعبه‌سیاه توصیف می‌کنیم. در این توصیف، رفتار سیستم در برابر رخدادهای خارجی (یعنی تغییر متغیرهای پایشی) و همچنین کنترل محیط به‌وسیله‌ی سیستم (یعنی تغییر متغیرهای کنترلی) را بدون توجه به جزئیات داخلی سیستم بیان می‌کنیم.

تعیین روان‌گزاره‌ها: روان‌گزاره‌ها (دامنه‌ی دوم از مجموعه‌ی فرمول ۲) را از روی قیود رفتاری سیستم (قیود نگاشت REQ در ستون ۵ جدول ۸)، که مرتبط با متغیرهای پایشی و کنترلی است، به دست می‌آوریم. از ردیف‌های اول تا سوم جدول ۸، برای تعیین روان‌گزاره‌های حس‌گر اندازه‌گیر قند خون (ستون‌های ۱ و۲ جدول ۹) و از ردیف‌های چهارم تا ششم آن برای تعیین روان‌گزاره‌های واحد تزریق (ستون‌های ۳ و۴ جدول ۹) استفاده می‌کنیم.

تعیین مدهای عملیاتی: مدهای عملیاتی (دامنه‌ی سوم از مجموعه‌ی فرمول ۲) را نیز با توجه به قیود نگاشت REQ از جدول ۸ تعیین می‌کنیم و آنها را در جدول ۱۰ نشان می‌دهیم. هر ردیف از جدول ۸ متناظر با یک ردیف از جدول ۱۰ است. مثلاً مد عملیاتی ناامن و عادی در جدول ۱۰ به‌ترتیب معرف $m_{Time} > ۱۰$ و $m_{Time} \leq ۱۰$ است که متناظر با ردیف اول جدول ۸ است. در مدهای ناامن و بحرانی، مؤلفه رفتار نادرستی دارد. اکنون قیود ایمنی و حیاتی را با ترکیب‌های مناسب روان‌گزاره‌ها و مدهای عملیاتی که در دو جدول ۹ و ۱۰ تعریف کرده‌ایم، برحسب حساب رخداد توصیف

جدول ۹. روان‌گزاره‌های حس‌گر اندازه‌گیر قند خون و واحد تزریق انسولین.

شرح	روان گزاره	شرح	روان‌گزاره
عادی‌بودن مقدار انسولین	NDose	پایان بازه ۱۰ دقیقه‌یی	Run-out
زیاد بودن مقدار انسولین	Ovr	شروع نمونه‌گیری	Sample
شروع تزریق	Inject	پایین رفتن قند خون	Low
کافی‌بودن مقدار انسولین	DosAvl	نرمال شدن قند خون	Norm
-	-	بالا رفتن قند خون	High

جدول ۱۰. مدهای عملیاتی حسگر و پمپ انسولین.

توصیف	مؤلفه	مد عملیاتی
پایش بازه زمانی	حس‌گر	عادی
نمونه‌گیری قند خون	حس‌گر	اندازه‌گیری
محاسبه مقدار انسولین تزریقی	پمپ	محاسبه
تأخیر در نمونه‌گیری خون	حسگر	ناامن
عدم تحویل انسولین کافی،	پمپ	
کافی‌نبودن انسولین موجود	پمپ	
پایین افتادن قند خون	محیط	بحرانی
تزریق انسولین غیرضروری	پمپ	
تزریق انسولین اضافی	پمپ	
تحویل انسولین لازم	پمپ	تحویل

جدول ۱۱. دو قید ایمنی و قید حیاتی نمونه بر حسب حساب رخداد برای پمپ.

قید	گزاره	شرح
SP_1	if Happens(@T(Run-out),τ)$\wedge$ [$\sim$HoldsAt(Sample,τ)$\wedge$ $\sim$Happens(@T(Sample),τ)] $\rightarrow$ Initiates(@T(Run-out), Unsafe)	نمونه‌گیری نباید به تأخیر بیفتد
LP_1	if $\sim$ HoldsAt(Run-out,τ)$\wedge$ Happens(@T(Sample),τ) $\rightarrow$ Initiates(@T(Sample), Measure)	نمونه‌گیری باید به موقع انجام شود

می‌کنیم. برای مثال، قرارداشتن در مد عملیاتی اندازه‌گیری (جدول ۱۰) و تغییر روان‌گزاره‌های Run-Out و Sample(جدول ۹) منجر به قید ایمنی SP_1 و قید حیاتی LP_1 می‌شود (جدول ۱۱). @T معرف این است که مقدار روان‌گزاره از «نادرست» به «درست» و @F معرف این است که مقدار روان‌گزاره از «درست» به «نادرست» تغییر می‌کند. گزاره‌های Happens، HoldsAt و Initiates در بخش ۲.۴ (حساب رخداد) توصیف شد. جدول کامل قیود ایمنی و قیود حیاتی که شامل ۱۰ قید است، در ضمیمه‌ی ۲ آمده است. قیود ایمنی معرف مدهای ناامن و بحرانی و قیود حیاتی معرف سایر مدها است.

۳.۵. توصیف جعبه‌سفید سیستم پمپ انسولین همراه

قیود ایمنی که یک نمونه از آنها در جدول ۱۱ مشخص شده است، قیودی هستند که راستی‌آزما از آنها برای راستی‌آزمایی رفتار اجرایی مخاطره‌آمیز نرم‌افزار استفاده می‌کند. اکنون با استفاده از جدول ۶، جدول حالات را برای گزاره‌های ایمنی و حیاتی سیستم پمپ انسولین همراه، که دو نمونه از آنها در جدول ۱۱ ارائه شده است، در جدول ۱۲ ارائه می‌دهیم. جدول ۱۲، جدول حالات کامل برای گزاره‌های ایمنی و حیاتی سیستم پمپ انسولین است که در ضمیمه‌ی ۲ آمده است. این جدول، توصیف جعبه‌سفیدی از حالات درست و مرجعی برای راستی‌آزما است تا به‌وسیله‌ی آن رفتار حین اجرای سیستم را راستی‌آزمایی کند. در حقیقت ردیف‌هایی از جدول ۱۲ که حالات ناامن یا بحرانی دارند، هسته‌ی راستی‌آزما را تشکیل می‌دهند. مثلاً ردیف اول جدول ۱۲ بیان می‌دارد که اگر نمونه‌گیری خون شروع نشده باشد (شرط Sample=f) و بازه زمانی به پایان رسیده باشد (رخداد@T(Run-Out))، سیستم در حالت ناامن قرار می‌گیرد. این شرط و این رخداد از قید SP_1 در جدول ۱۱، و با توجه به نگاشت‌های ردیف اول و ردیف ششم جدول ۶ به دست آمده است.

۴.۵. پیاده‌سازی راستی‌آزمای پمپ انسولین همراه

اکنون با توجه به ماشین حالتی که در شکل ۷ برای پیاده‌سازی جدول حالات نمونه ۷ نشان داده شده است، نخست ماشین حالت راستی‌آزمای سیستم پمپ انسولین همراه (شکل ۱۰) را با توجه به ردیف‌هایی از

جدول ۱۲. گذار حالات سیستم برای سیستم پمپ انسولین همراه.

C	رخداد ـ شرط									حالت جدید
	A	B	C	D	E	F	G	H	I	
۱	f	@T	—	—	—	—	—	—	—	U(SP_1)
	@T	@T	—	—	—	—	—	—	—	۲(LP_1)
	@T	f	—	—	—	—	—	—	—	۲(LP_1)
۲	—	—	@T	—	—	—	—	—	—	R(SP_2)
	—	—	—	@T	—	—	—	—	—	۱(LP_2)
	—	—	—	—	@T	—	—	—	—	۳(LP_3)
۳	—	—	—	—	—	@F	—	—	—	U(SP_3)
	—	—	—	—	—	@T	f	—	—	U(SP_4)
	—	—	—	—	—	—	—	@T	—	R(SP_5)
	—	—	—	—	—	@T	t	—	—	۴(LP_4)
۴	—	—	—	—	—	—	—	—	@T	۱(LP_5)

نمادهای جدول ۱۲.

C: حالت جاری

A: Sample، **B**: Run-Out، **C**: Low، **D**: Norm، **E**: High،

F: NDose، **G**: DosAv1، **H**: Ovr، **I**: Inject

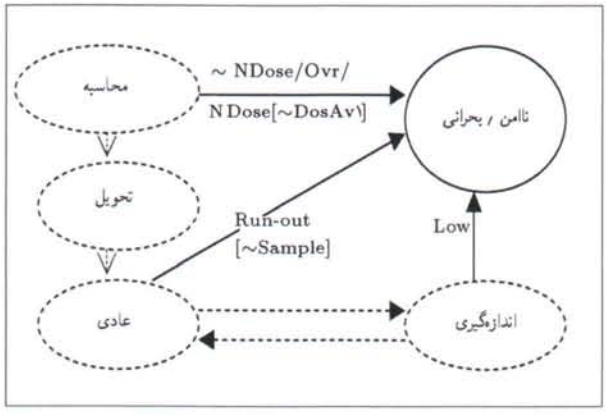
۱: عادی، ۲: اندازه‌گیری، ۳: محاسبه، ۴: تحویل

U: ناامن، **R**: بحرانی

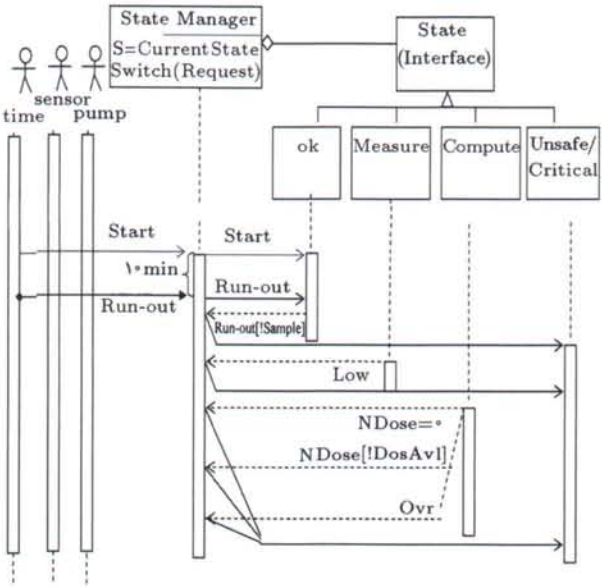
جدول ۱۲ که حالات ناامن و بحرانی دارند، به دست می‌آوریم. برای مثال انتقال از حالت محاسبه به حالت ناامن یا بحرانی با توجه به قیود ایمنی SP_3، SP_4 و SP_5 از ردیف سوم جدول ۱۲ مشخص شده است. سپس پیاده‌سازی راستی‌آزما را با نمودار توالی[۲۵] (شکل ۱۱) نشان می‌دهیم.

از ماشین حالت شکل ۱۰ مشخص است که پس از سه حالت عادی، اندازه‌گیری و محاسبه می‌تواند حالت‌های ناامن یا بحرانی پیش آید که رخدادها و شرایطی که اجرای برنامه را به حالت‌های ناامن یا بحرانی می‌برد، روی جریان‌های داده (خطوط پر) مشخص شده است. بنابراین، این مسیرها ناامن هستند که مورد دغدغه‌ی راستی‌آزما

هستند.با توجه به پیاده‌سازی نمونه شکل ۷، باید چهار کلاس حقیقی برای حالت‌های عادی، اندازه‌گیری، محاسبه و بحرانی/ناامن در نظر بگیریم (شکل ۱۱). هنگامی که یک بازیگر یا یک متد از یک کلاس حقیقی پیامی را به کلاس StateManager برمی‌گرداند این کلاس برحسب نوع پیام (رخداد و شرط) اجرای برنامه را به متدی واقع در کلاس دیگر واگذار می‌کند. ارتباط بین کلاس‌ها غیرمستقیم است، به این صورت که وقتی یک متد از هر کلاس از کلاس StateManager پیامی دریافت کرد، در پایان پیام نتیجه را به آن برمی‌گرداند که State-Manager با توجه به آن انتقال به کلاس جدید را انجام می‌دهد. خطوط پر معرف ارسال پیام از StateManager به یکی از چهار کلاس و خطوط توخالی معرف پیام بازگشت از یکی از چهار کلاس به StateManagerاست.



شکل ۱۰. ماشین حالت راستی‌آزمای سیستم پمپ انسولین همراه.



شکل ۱۱. نمودار توالی پیاده‌سازی پایشگر.

۶. نتیجه‌گیری

ما در این نوشتار رویکردی بنام توصیف مبتنی بر رخداد (EBV) را ارائه دادیم که به‌وسیله‌ی آن در طی چهار مرحله از توصیف نیازها تا پیاده‌سازی، می‌توان یک راستی‌آزما را برای راستی‌آزمایی رفتار زمان اجرای نرم‌افزارهای حساس به ایمنی، به‌صورت خودکار ساخت. مرحله‌ی نخستِ این رویکرد را با مستندسازی نیازهای اولیه‌ی کاربر آغاز کردیم و در طی سه مرحله بعدی به ساخت برنامه‌ی راستی‌آزما پرداختیم. در هر مرحله روشی را ارائه دادیم تا مرحله‌ی بعدی بتواند از روی خروجی مرحله‌ی قبل، ورودی خود را بسازد.

در مرحله‌ی نخست، نیازهای یک سیستم حساس به ایمنی را با روش Parnasمدل کردیم. روش مدل‌سازی Parnas به ما کمک کرد تا بتوانیم محیط را برحسب فرهنگ واژگان کاربران در قالب متغیرهای پایشی و کنترلی تعریف کنیم و با بیان روابط بین این متغیرها، نیازهای کاربران را از سیستم نشان دهیم. این موضوع ما را برای توصیف نیازهای محیط براساس رخدادها (مرحله دوم) آماده کرد.

در مرحله‌ی دوم، مدل Parnas نیازها را به توصیف مبتنی بر رخداد برحسب منطق رتبه اول حساب رخداد نگاشت کردیم. به این وسیله یک توصیف جعبه‌سیاه از رفتار مورد انتظار سیستم در برابر رخدادها را نشان دادیم و نیازها را برحسب قیود ایمنی و حیاتی از یکدیگر جدا کردیم با این هدف که راستی‌آزما را برای راستی‌آزمایی رفتار حین اجرا در برابر قیود ایمنی بسازیم. قیود ایمنی مجموعه‌یی از نبایدها برای رخدادهای مخاطره‌آمیز است که باید از رعایت آنها در هر بار اجرای نرم‌افزار مطمئن شویم.

در مرحله‌ی سوم پلی ایجاد کردیم که به‌وسیله آن بتوان از رخدادهای سطح بالای محیطی به فعالیت‌های سطح پایین اجرا گذار کرد. به‌منظور ساخت این پل، با مشخص‌کردن هفت نگاشت و با استفاده از روش SCR توصیف مبتنی بر رخداد نیازها را به توصیف مبتنی بر حالت سیستم تبدیل کنیم. توصیف حاصل یک توصیف جعبه‌سفید از رفتار درست نرم‌افزار درکنترل محیط است. در حالی که توصیف جعبه‌سفید، بیانگر حالات اجرا است. توصیف جعبه‌سیاه در مرحله‌ی دوم بیانگر دید کاربر است. در توصیف جعبه‌سفید، حالات بحرانی و ناامن مشخص، و از بقیه حالات جدا شدند تا گذار به این حالات در زمان اجرا به‌وسیله‌ی راستی‌آزما گزارش شود. بنابراین راستی‌آزما با استفاده از جدول حالات از همه رخدادها و شرایطی که منتهی به حالات وخیم و ناامن می‌شود، مراقبت می‌کند.

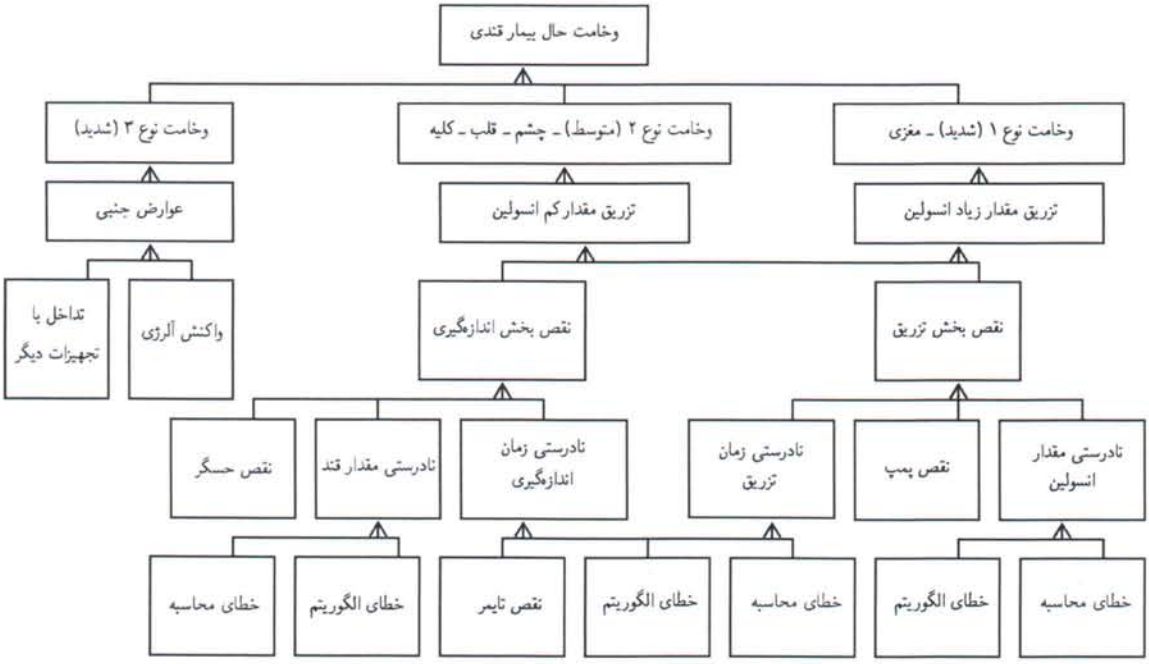
در مرحله‌ی چهارم با استفاده از الگوی طراحی حالت برنامه‌ی راستی‌آزما پیاده کرده‌ایم. استفاده از این الگو ما را در تبدیل خودکار جدول حالات که در مرحله‌ی سوم به دست آمد، به کد یاری کرد. از آنجا که حالات زمان اجرا برحسب نوع رخدادها در محیط تغییر می‌کند،

جدول ۱۳. مقایسه‌ی رویکرد EBV و رویکردهای مرتبط.

معیار	رویکرد			
	EBV	HAWK	RTL	[۲۴]،[۲۵]
پشتیبانی سطح انتزاع در توصیف و مستندسازی نیازها	برحسب فرهنگ کلمات کاربران و براساس روش Parnas	زبان HAWK(مبتنی بر منطق زمانی و زبان Java)	منطق بی‌درنگ	منطق زمانی خطی
پشتیبانی نوع توصیف نیازها	رخداد (براساس کمیت‌های سطح بالای محیطی) و حالت	رخداد و حالت (برحسب فرمول‌های منطق زمانیEagle)	مبتنی بر تقدم و تاخر رخدادهای محیطی	برحسب فرمول‌های منطق زمانی
پشتیبانی نوع و بیان قیود	قیود ایمنی، حساب رخداد	قیود ایمنی، منطقEagle	قیود ایمنی و حیاتی،RTL	قیود ایمنی، منطقLTL
بهره‌گیری از روش‌های رسمی	حساب رخداد و SCR	منطق Eagle	منطق RTL و الگوریتم‌های گراف	جبر و منطق زمانی خطی
پیاده‌سازی راستی‌آزما	استفاده از الگوی حالت	ندارد	استفاده از گراف	منطق زمانی
پشتیبانی نگاشت بین رخدادهای محیطی و حالات اجرایی نرم‌افزار	خودکار و با استفاده از SCR و الگوی حالت	خودکار و از رخدادهای حین اجرا (برحسب HAWK به فرمول‌های منطقیEagle)	خودکار و از رخدادهای حین اجرا به اجزا گراف حالت	خودکار و از دنباله حالات به فرمول‌های منطقی
روش و نوع ساخت راستی‌آزما	خودکار و مبتنی برتوصیف کمیت‌های محیطی	ندارد	خودکار و مبتنی بر گزاره‌های RTL	خودکار و مبتنی بر گزاره‌های منطقی

الگوی طراحی حالت روشی مناسب برای پیاده‌سازی راستی‌آزما در بررسی رخدادهای منتهی به حالات ناامن و بحرانی است. جزییات پیاده‌سازی راستی‌آزما را که در آن از الگوی طراحی حالت استفاده شده است، به‌وسیله‌ی نمودار توالی نشان دادیم. برای هر حالت یک کلاس، و برای هر رخداد یک متد از آن کلاس منظور شده است. هنگامی که رخدادی حادث می‌شود، متد جاری از یک کلاس خاتمه می‌یابد و متد جدیدی از کلاس‌دیگر فراخوانی می‌شود و به این وسیله گذار حالت رخ می‌دهد.

رویکرد EBV نسبت به رویکردهای مرتبط که آنها را در بخش سوم مرور کردیم، نوآوری‌هایی دارد: ۱. قابلیت به‌کارگیری EBV برای سیستم‌های رخدادگرا (به‌وسیله‌ی حساب رخداد) و سیستم‌های مبتنی بر حالت (به‌وسیله‌ی روش SCR)، گستره‌ی استفاده‌ی وسیع‌تر آن را نشان می‌دهد؛ ۲. در حالی که رویکردهای دیگر سعی دارند مستقیماً نیازها را با منطق زمان خطی[۲۴و۲۵] و مشابه آن[۲۲و۲۳]، یا منطق‌های ویژه[۲۰و۲۱] توصیف کنند، EBV تلاش دارد تا با بهره‌گیری از مدل‌سازی به روش Parnas محیط را برحسب فرهنگ واژگان کاربران و متخصصین دامنه توصیف کند و سپس با ارائه‌ی روشی آن را به گزاره‌های حساب رخداد تبدیل کند. ما به این وسیله سیستم را از محیط منتزع ساختیم تا ویژگی‌های آن را در برابر محیط از دید کاربران توصیف کنیم. پشتیبانی انتزاع توصیف نیازها در سطح فرهنگ واژگان کاربران و متخصصین دامنه (یعنی برحسب کمیت‌های محیطی) موجب شد تا در توصیف نیازها نسبت به برخی از رویکردها مانند HAWK [۲۱] که برای راستی‌آزمایی برنامه‌های Java تدوین شده است و در دستورالعمل‌های خود تعاریف سطح برنامه‌نویسی را لحاظ کرده است، وضوح بیشتری را فراهم سازیم. ما در رویکرد EBV با استفاده از مدل Parnas توصیف روشنی که مبتنی بر فرهنگ واژگان کاربران سیستم است، ارائه دادیم و سپس آن را به توصیف حالات (جعبه‌سفید) نگاشتیم و به این وسیله توصیف‌های سطح کاربر را از توصیف‌های سطح برنامه جدا کردیم. براساس یک دسته‌بندی از پایشگرها (راستی‌آزماها)[۴۰]، یکی از موارد اولیه در ساخت پایشگرها، پشتیبانی سطح انتزاع است که به سه سطح دامنه، طراحی، و پیاده‌سازی تقسیم می‌شود. ما در رویکرد EBV بالاترین سطح انتزاع، یعنی دامنه را، پشتیبانی کردیم و در آن نیازهای سیستم را برحسب فرهنگ واژگان متخصصین و کاربران سیستم بیان کردیم؛ ۳. به‌دلیل بهره‌گیری از منطق حساب رخداد، امکان راستی‌آزمایی و سازگاری ویژگی‌های سیستم در سطح توصیف نیز فراهم شد؛ ۴. خودکار ساختن مراحل ساخت راستی‌آزما که با فراهم‌نمودن چارچوبی به‌منظور نگاشت از رخدادهای محیطی به فعالیت‌های حین اجرا صورت گرفت، نشان‌دهنده‌ی مزیت دیگر EBV بود؛ ۵. استفاده از الگوی طراحی حالت درEBV، که به‌طور عملی در پیاده‌سازی ماشین‌های حالت به‌کارگرفته شده است، این رویکرد را در پیاده‌سازی راستی‌آزما هنگامی که جدول حالات بزرگ می‌شود (یعنی تعداد حالات یا رخدادها و شروط زیاد می‌شوند) توانا ساخت. جدول ۱۳ خلاصه‌یی از جمع‌بندی ویژگی‌های رویکرد EBV را نسبت به رویکردهای دیگر ساخت راستی‌آزما نشان می‌دهد.



ضمیمه‌ی ۱. درخت خطای سیستم پمپ انسولین.

ضمیمه‌ی ۲. جدول قیود ایمنی و حیاتی سیستم پمپ انسولین همراه برحسب گزاره‌های حساب رخداد.

	P	توصیف	گزاره	مؤلفه
۱	SP۱	نمونه‌گیری نباید به تأخیر بیفتد.	if Happens(@T(Run-out),τ)$\wedge$ [$\sim$ HoldsAt(Sample, τ)$\wedge$ $\sim$ Happens(@T(Sample),τ)] $\rightarrow$ Initiates(@T(Run-out),Unsafe)	حسگر
۲	LP۱	نمونه‌گیری باید به موقع انجام شود.	if $\sim$HoldsAt(Run-out,τ)$\wedge$ Happens(@T(Sample),τ) $\rightarrow$ Initiates(@T(Sample), Measure)	حسگر
۳	SP۲	قند خون نباید پایین بیفتد.	if Happens(@F(Measure),τ)$\wedge$ Happens (@T(Low,τ) $\rightarrow$ Initiates(@T(Low),Critical)	حسگر
۴	LP۲	نمونه‌گیری مجدد باید انجام شود.	if Happens(@F(Measure),τ)$\wedge$ Happens (@T(Norm,τ) $\rightarrow$ Initiates(@T(Norm),Ok)	حسگر
۵	LP۳	انسولین باید به موقع محاسبه شود.	if Happens(@F(Measure),τ) $\wedge$ Happens(@T(High,τ) $\rightarrow$ Initiates(@T(High),Compute))	پمپ
۶	SP۳	انسولین لازم باید محاسبه شود.	if Happens(@T(High),τ)$\wedge$ Happens(@F(NDose),τ) $\rightarrow$ Initiates(@F(NDose),Unsafe)	پمپ
۷	SP۴	تحویل انسولین نباید به تأخیر افتد.	if Happens(@T(NDose),τ) $\wedge$ [$\sim$ HoldsAt(DosAvl,τ) $\wedge$ $\sim$ Happens(@T(DosAvl),τ)] $\rightarrow$ Initiates(@T(NDose),Unsafe)	پمپ
۸	SP۵	انسولین اضافی نباید محاسبه شود.	if Happens(@T(Ovr),τ) $\rightarrow$ Initiates(@T(Ovr),Unsafe)	پمپ
۹	LP۴	انسولین لازم باید تحویل شود.	if Happens(@T(NDose),τ) $\wedge$ [HoldsAt(DosAvl,τ) $\wedge$ $\sim$ Happens(@F(DosAvl),τ)] $\rightarrow$ Initiates(@T(NDose),Deliver)	پمپ
۱۰	LP۵	نمونه‌گیری مجدد باید انجام شود.	if Happens(@T(Inject),τ) $\rightarrow$ Initiates(@T(Inject), Ok)	پمپ و حسگر

پانوشت

1. event-based verification
2. stale pointer dereferencing
3. buffer overflow attack
4. local security authority subsystem service
5. anonymous user
6. event calculus
7. software cost reduction
8. state design pattern
9. continuous insulin infusion pump
10. safety assertions
11. run-time monitoring and verification
12. third party components
13. invariants
14. rule-based
15. real-time logic
16. requlare expressions
17. harel statecharts
18. event-condition-action
19. monitored variables
20. fluent
21. first order logic
22. simplified event calculus
23. run-time polymorphism
24. Pacemaker
25. sequence diagram

منابع

1. Ghosh, A. Wanken, J. and Charron, F. Detecting anomalous and unknown intrusions against programs, In Proceedings of 14th Annual Computer Security Applications Conference (December 1998).
2. Austin, T. Breach, S. and Sohi, G. "Efficient detection of all pointer and array access errors", In ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'94), **29**(6), pp. 290-301 (June 1994).
3. Microsoft Security Bulletin MS04-028, Version 3.0. (September 2004).
4. Gates, A.Q. and Teller, P.J. "DynaMICs: an automated and independent software-fault detection approach", In Proceedings of 4th International High-Assurance System Engineering Symposium (1999).
5. Butler, R.W. and Finelli, G.B. "The infeasibility of quantifying the reliability of life-critical real-time software", *IEEE Transactions on Software Engineering*, (19), pp. 3-12 (January 1993).
6. Littlewood, B. and Stringini, L. "Validation of ultrahigh dependability for software-based systems", *Communications of the ACM*, **11**(36), pp. 69-80 (November 1993).
7. Myers, G.J. The Art of Software Testing, Second Edition, John Wiley (2004).
8. Xu, J "On inspection and verification of software with timing requirements", *IEEE Transactions on Software Engineering*, **29**(8), pp. 705-720 (August 2003).
9. Gahl, D.J. DijKstra, E.J. and Hoare, C.A.R. Notes on Structured Programming, Academic Press London (1972).
10. Parnas, D.L. Schouwen, A.J.V. and Kwan, S.P. "Evaluation of safety-critical software", *Communications of the ACM*, **33**(6), pp.636-648 (1990).
11. Myers, W. "Can software for the strategic defence initiative ever be error-free?", *IEEE Computer*, **19**(11), pp. 61-67 (November 1986).
12. Iyer, R.K. and VerLardi, P. "Hardware-related software errors:measurement and analysis", *IEEE Transaction on Software Engineering*, **11**(2), pp.223-231 (Febury 1986).
13. Gates, A.Q. Roach, S. Mondragon, O. and Delgado, N. "DynaMICs: comprehensive support for run-time monitoring", In Proceedings of the First Workshop on Runtime Verification(RV'01), ENTCS, **55**(2), pp. 61-67 (Elsevier 2001).
14. Mok, A.K. and Liu, G. "Efficient run-time monitoring of timing constraints", In IEEE Real-Time Technology and Applications Symposium (June 1997).
15. Liao, Y. and Cohen, D. "A specificational approach to high level program monitoring and measuring", *IEEE Transaction on Software Engineering*, **18**(11), pp. 969-979 (November 1992).
16. Peleska, j. "Test automation for safety-critical systems: industrial application and future developments", In Proceedings of the 3^{rd} International Symposium of Formal Methods Europe (1996).
17. Schroeder, B.A. "On-Line monitoring: a tutorial", *IEEE Computer*, **28**(6), pp.72-78 (June 1995).
18. Zeller, A. "Program analysis: a hierarchy", In Proceedings of the 3^{rd} Workshop of ICSE(International Conference on Software Engineering) on Dynamic Analysis (May 2003).
19. Peters, D.K. and Parnas, D.L. "Requirements-Based monitors for real-time systems", *IEEE Transactions on Software Engineering*, **28**(2), pp. 146-158 (February 2002).
20. Barringer, H. Goldberg, A. Havelund, K. and Sen, K. "Rule-Based runtime verification", In Proceedings of 5^{th} International Conference on Verification, Model Checking and Abstract Interpretation(VMCAI'04), LNCS 2937, pp.44-57 (Springer 2004).
21. d'Amorim, M. and Havelund, K. Event-Based Runtime Verification of Java Programs, In Proceedings of the 5th Workshop of ICSE(International Conference on Software Engineering) on Dynamic Analysis (2005).

22. Mok, A.K. and Liu, G. Efficient Run-time Monitoring of Timing Constraints, In Proceedings of 3rd IEEE Real-time Technology and Applications Symposium (June 1997).

23. Jahanian, F. and Mok, A.K. "Modechart: a specification language for real-time systems", *IEEE Transactions on Software Engineering*, **20**(12), pp. 933-947 (1994).

24. Sen, K. Rosu, G. and Agha, G. "Generating optimal linear temporal logic monitors by coinduction", In Proceedings of 8th Asian Computing Science Conference (ASIAN'03), LNCS 2896, pp. 260-275 (Springer 2003).

25. Havelund, K. and Rosu, G. "Synthesizing monitors for safe properties", In Proceedings of 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, LNCS, 2280, pp. 342-359 (Springer 2002).

26. Sen, K. and Rosu, G. "Generating optimal monitors for extended regular expressions", In Proceedings of Runtime Verification (RV'03), ENTCS, **89**(2), pp. 162-181 (Elsevier 2003).

27. Babamir, S.M. and Jalili, S. "Dynamic analysis of object-oriented programs using state machines and ECA rules", In Proceedings of 14th International Conference on Intelligent and Adaptive Systems and Software Engineering (IASSE-2005), pp. 243-248, Toronto, Canada (July 2005).

28. Miller, S.P. and Tribble, A.C. "Extending the four-variable model to bridge the system-software gap", In Proceedings of the 20th Digital Avionics Systems Conferene (DASC01), Daytona Beach, Florida (Oct 2001).

29. Parnas, D.L. and Madey, J. "Functional documents for computer systems", *Journal of Science of Computer Programming*, **25**(1), pp. 41-61, Elsevier (October 1995).

30. Heitmeyer, C.L. Jeffords, R.D. and Labaw, B.G. "Automated consistency checking of requirements specifications", *ACM Transactions on Software Engineering and Methodology*, **5**(3), pp. 231-261 (July 1996).

31. Kowalski, R.A. and Sergot, M.J. "A logic based calculus of events", *New Generation Computing*, (4), pp. 67-95 (1986).

32. Sadri, F. and Kowalski, R. "Variants of the event calculus", in proceedings of the 12th international conference on logic Programming(ICLP), MIT Press, pp. 67-82 (June 1995).

33. Shanahan, M. "The event calculus explained, in proceedings of the conference on artificial intelligence today", *Recent Trends and Development*, LNCS 1600, pp. 409-430 (Springer 1999).

34. Bharadwaj, R. and Heitmryer, C.L. "Model checking complete requirements specifications using abstraction", *Automated Software Engineering*, (6), pp. 37-68, Kluwer Academic Publishers (1999).

35. Heitmeyer, C.L. Labaw, B. and Kiskisy, D. "Consistency checking of SCR-Style requirements specifications", In Proceedings of International Symposium on Requirements Engineering (March 1995).

36. Gamma, E. Helm, R. Johnson, R. and Vlissides, J. Design Patterns, Elements of Reusable Object-Oriented Software, Addison-Wesley (1995).

37. Douglass, B.P. Doing Hard Time, Developing Real-Time Systems with UML, Objects, Framework and Patterns, Addison-Wesley (1999).

38. Samek, M. Practical Statecharts in C/C++, Quantum Programming for Embedded Systems, CMP Books (2002).

39. Roberson, K.E. Glazer, N.B. and Campbell, R.K. The Latest Developments in Insulin Injection Devices, Journal of American Association of Diabetes Educators, (26), pp. 135-152 (January/February 2000).

40. Delgado, N. Gates, A.Q. and Roach, S. A Taxonomy and Catalog of Runtime Software-Fault Monitoring Tools, IEEE Transactions on Software Engineering, **30**(22), pp. 859-872 (December 2004).