استفاده از یادگیری تقویتی جهت افزایش سرعت حل‌کننده‌های SMT با هدف تشخیص سریع‌تر مسیرهای غیرقابل اجرا در نرم‌افزار

مجید فیضی
کارشناسی ارشد، دانشکده مهندسی کامپیوتر، دانشگاه صنعتی خواجه نصیرالدین طوسی
majid.feyzi@email.kntu.ac.ir

محمد مهدی اثنی عشری
استادیار، دانشکده مهندسی کامپیوتر، دانشگاه صنعتی خواجه نصیرالدین طوسی
esnaashari@kntu.ac.ir

چکیده

آزمون نرم‌افزار یک موضوع مهم و لازم در فرایند توسعه هر نرم‌افزاری می‌باشد. امروزه بخش عمده‌ای از آزمون نرم‌افزار به شیوه خودکار انجام می‌شود. در آزمون خودکار نرم‌افزار موانعی می‌توانند وجود داشته باشند که آزمون را با مشکل مواجه کنند. از این موانع، می‌توان به وجود مسیرهای غیرقابل اجرا[1] در گراف جریان کنترلی[2] حاصل از کدهای نرم‌افزار تحت آزمون اشاره کرد. مسیرهای غیرقابل اجرا مسیرهایی هستند که با هیچ ورودی نمی‌توان از آنها عبور کرد. وجود چنین مسیرهایی، می‌تواند در حوزه‌های مختلف، مانند تولید داده آزمون، امنیت و... مشکلاتی را به وجود آورد و باعث هدر رفت منابع شود.

برای پیشگیری از مشکلات ذکر شده، تشخیص مسیرهای غیرقابل اجرا بسیار مهم می‌باشد. اما تشخیص این مسیرها در نرم‌افزارهای بزرگ، به دلیل حجم زیاد کدها و زیاد بودن تعداد مسیرها در گراف جریان کنترلی، می‌تواند کاری زمانبر باشد. در روش ارائه شده در این مقاله، سرعت تشخیص مسیرهای غیرقابل اجرا با به کارگیری الگوریتم‌های یادگیری تقویتی[3] بهبود داده شده است. در این روش سرعت حل‌کننده SMT[4] به نام Z3 با بهره‌گیری از عامل DQN[5]، افزایش داده می‌شود. Z3 ابزاری است که برای بررسی قابل حل بودن فرمول‌های ساخته شده از شرط‌های موجود در گراف جریان کنترلی در تشخیص مسیرهای غیرقابل اجرا استفاده می‌شود. در واقع در این مقاله تاکتیک‌های ارائه شده توسط Z3 با استفاده از یادگیری تقویتی یاد گرفته شده و سپس با انتخاب و اعمال تاکتیک‌های Z3 براساس یادگیری انجام شده، سرعت حل Z3 افزایش می‌یابد. با افزایش سرعت Z3، سرعت تمامی روش‌ها و الگوریتم‌هایی که از این ابزار برای تشخیص مسیرهای غیرقابل اجرا استفاده می‌کنند نیز افزایش خواهد یافت. همچنین ابزار Z3 در هر حوزه‌ای که بتوان مسائل را به مسئله بررسی صدق‌پذیری یک فرمول کاهش داد نیز کاربرد دارد و بهبود عملکرد آن در چنین حوزه‌هایی نیز می‌تواند موثر باشد.

روش ارائه شده در این مقاله در منطق‌های QF_NRA و QF_NIA آزمایش شده است. در آزمایش‌های انجام شده، نشان داده شده است که با استفاده از روش پیشنهادی، سرعت Z3 را می‌توان تا ۴/۷ برابر افزایش داد. همچنین در آزمایش‌های انجام شده با استفاده از این روش، سرعت Z3 در منطق QF_NRA، به طور میانگین ۱/۹۵۴ برابر و در منطق QF_NIA، به طور میانگین ۱/۶۸۷ برابر افزایش یافته است.

کلمات کلیدی: آزمون خودکار نرم‌افزار، مسیرهای غیرقابل اجرا، حل‌کننده SMT، حل‌کننده Z3، یادگیری تقویتی، DQN.

---

[1] Infeasible Paths
[2] Control Flow Graph
[3] Reinforcement Learning
[4] Satisfiability Modulo Theories
[5] Deep Q-Network

۱. مقدمه

امروزه نرم‌افزار به بخشی از زندگی روزمره انسان‌ها مبدل شده است، و این موضوع باعث بزرگتر و پیچیده‌تر شدن نرم‌افزارها شده است. پیچیدگی نرم‌افزارها، فرایند توسعه آنها را دشوارتر و پرهزینه‌تر کرده است. یکی از مراحل مهم و تاثیرگذار در فرایند توسعه هر نرم‌افزاری، آزمون آن است. آزمون نرم‌افزار راهی برای اطمینان از درستی عملکرد نرم‌افزار می‌باشد. هدف در آزمون نرم‌افزار پیدا کردن بیشترین تعداد خطاها با کمترین تعداد آزمون است. البته باید به این نکته هم توجه کرد که دستیابی به نرم‌افزار کاملا عاری از خطا، عملا غیر ممکن است [1]. آزمون نرم‌افزار از مراحل مهم، زمانبر و پرهزینه در توسعه نرم‌افزار به شمار می‌آید. متاسفانه به دلیل وجود هزینه‌های مالی، نیاز به انجام انواع مختلف آزمون نرم افزار و نیز زمان‌بر و دشوار بودن این فرایند، در صنعت به خوبی به آن توجه نمی‌شود. این عدم توجه می‌تواند عواقب بدی مانند خاموشی بزرگ شمال شرقی آمریکا در سال ۲۰۰۳ و یا وجود نقص در دستگاه پرتو درمانی[6] که باعث مرگ افراد می‌شد [1] را به دنبال داشته باشد. بهترین راهکار برای برطرف کردن چنین مشکلاتی، خودکارسازی آزمون نرم‌افزار است. همانطور که در [2] بیان شده است، تا ۵۰ درصد هزینه‌های توسعه نرم‌افزار برای آزمون نرم‌افزار است. از این رو محققان، تحقیقاتی برای خودکارسازی فرایند آزمون نرم‌افزار انجام داده‌اند، تا با این کار، هزینه‌ها و زمان مصرفی برای انجام آن را بکاهند.

اما در خودکارسازی آزمون نرم‌افزار موانع و مشکلاتی نیز وجود دارد. یکی از مهم‌ترین این موانع، وجود مسیرهای غیرقابل اجرا در واحدهای تحت آزمون، نظیر رویه‌ها یا کلاس‌ها، است. این مسیرها، آنهایی هستند که عملا با هیچ داده ورودی، نمی‌توان آنها را طی کرد. به عبارت بهتر، در هیچ اجرایی از واحد مورد نظر، این مسیرها طی نشده و تاثیری در عملکرد واحد ندارند. وجود چنین مسیرهایی، فرایند تولید خودکار آزمون را در یک حلقه بی انتها قرار می‌دهد. چون تولید کننده خودکار می‌خواهد به هر قیمتی که شده، ورودی مناسبی را بیابد که بتواند این مسیرها را هم پوشش دهد؛ چیزی که عملا امکان پذیر نیست. در [3] برخی از روش‌های تولید داده آزمون معرفی شده‌اند. برای جلوگیری از چنین وضعیتی، لازم است که ابتدا مسیرهای غیرقابل اجرا شناسایی شده و از گراف واحد تحت آزمون کنار گذاشته شوند.

تا کنون روش‌های مختلفی برای تشخیص مسیرهای غیرقابل اجرا ارائه شده است. برای مثال مقاله [4] براساس اجرای نمادین[7]، یک روش داوری ایستا[8] پیشنهاد می‌کند که با محاسبه دو بازه عددی برای هر نماد، بخشی از مسیرهای غیرقابل اجرا را به شکل دقیق تشخیص می‌دهد. در مقاله [5] نیز، روشی برای تشخیص مسیرهای غیرقابل اجرای بین‌رویه‌ای با استفاده از الگوهای محدودیت مسیر صدق‌ناپذیر، پیشنهاد شده است که در آن مسیرهای غیرقابل اجرای بین‌رویه‌ای بررسی شده و سپس محدودیت مسیر مربوط به هر مسیر غیرقابل اجرا استخراج شده است. سپس ویژگی‌های محدودیت مسیر غیرقابل حل مسیرهای غیرقابل اجرا را کاوش کرده و به ۹ الگوی محدودیت مسیر صدق‌ناپذیر دست یافته‌اند. مقاله [6] نیز ابزار SMT-IPD[9] برای تشخیص مسیرهای غیرقابل اجرا را ارائه کرده است و از یک روش ایستا برای تشخیص آنها استفاده می‌کند که مبتنی بر حل‌کننده SMT است. در روش‌هایی که از حل‌کننده‌های SMT برای تشخیص مسیرهای غیرقابل اجرا استفاده می‌شود، سرعت حل‌کننده SMT تاثیر مستقیم در سرعت تشخیص مسیرهای غیرقابل اجرا دارد. بنابراین افزایش سرعت حل‌کننده SMT می‌تواند در تشخیص سریعتر مسیرهای غیرقابل اجرا موثر باشد.

در این مقاله مسیرهای غیرقابل اجرا و چگونگی تشخیص آنها بررسی شده و تلاش شده است تا بهبودهایی برای تشخیص آنها صورت گیرد. هدف این مقاله افزایش سرعت فرایند تشخیص مسیرهای غیرقابل اجرا با بهبود سرعت ابزارهای مورد استفاده برای تشخیص این مسیرها می‌باشد. ابزارهایی که در تشخیص مسیرهای غیرقابل اجرا کاربرد دارند، حل‌کننده‌های SMT هستند [6]. حل‌کننده SMT که در این مقاله مورد توجه و بررسی قرار گرفته است، حل‌کننده Z3 می‌باشد [7]. راهکار مورد

[6] Therac-25
[7] Symbolic Execution
[8] Static Judgment
[9] SMT solver based on Infeasible Path Detection

استفاده در این مقاله بدین منظور استفاده از یادگیری تقویتی است. در واقع در این مقاله با استفاده یادگیری تقویتی تاکتیک‌های حل‌کننده Z3 انتخاب و اعمال می‌شوند تا سرعت حل فرمول‌های استخراج شده از مسیرها افزایش یابد.

آزمایش‌های انجام شده نشان داده‌اند که با به کارگیری روش پیشنهادی، می‌توان سرعت حل‌کننده Z3 را تا ۴/۷ برابر افزایش داد. بهبود سرعت عملکرد حل‌کننده Z3 نه تنها در تشخیص مسیرهای غیرقابل اجرا، بلکه در مسائل مختلف دیگر نیز می‌تواند مفید باشد، چرا که حل‌کننده Z3 در زمینه‌های مختلف دیگری، نظیر تایید نرم‌افزار و سخت افزار [8]، زمانبندی و آنالیز ایستا نیز کاربرد دارد [9].

در بخش ۲ به معرفی و بررسی کارهای انجام شده در حیطه مورد بحث این مقاله می‌پردازیم. در بخش ۳ به شکل خلاصه مفاهیم مقدماتی مورد نیاز را شرح خواهیم داد. در بخش ۴ مسئله موجود را تعریف خواهیم کرد. در بخش ۵ روش پیشنهادی خود را به طور کامل شرح خواهیم داد. در بخش ۶ آزمایش‌های صورت گرفته و نتایج آنها را بررسی خواهیم کرد و در بخش ۷ نیز یک جمع‌بندی از مطالب ذکر شده خواهیم داشت.

۲. کارهای مرتبط

در این بخش برخی از کارهای انجام شده برای تشخیص مسیرهای غیرقابل اجرا و نیز کارهای انجام شده به منظور تسریع حل‌کننده‌ها را بررسی می‌نماییم.

مقاله [4] یک روش داوری ایستا مبتنی بر اجرای نمادین ارائه می‌کند که بطور دقیق بخشی از مسیرهای غیرقابل اجرا و نیز بخشی از مسیرهای قابل اجرا تشخیص می‌دهد. این روش برای هر متغیر نمادین[10] به طور همزمان ۲ مجموعه مقدار ممکن[11] و مجموعه مقدار ضروری[12] را محاسبه می‌کند و با استفاده از این اطلاعات بازه[13] به صورت قطعی تعیین می‌کند که مسیر قابل اجرا یا غیرقابل اجرا و یا نامعلوم است. به طور خلاصه روش ارائه شده در [4] برای هر نماد در هر مسیر که نشان دهنده یک متغیر می‌باشد، دو بازه عددی یافته و سپس با استفاده از آنها قابل اجرا بودن یا نبودن مسیر را تعیین می‌کند.

در اجرای نمادین به جای مقادیر واقعی، نمادها به ورودی‌ها اختصاص می‌یابند. بنابراین برای هر مسیر، یک عبارت از نمادهای ورودی به دست می‌آید. انفجار مسیر[14] یکی از محدودیت‌های اجرای نمادین می‌باشد. این به معنای افزایش نمایی تعداد مسیرهای قابل اجرا در برنامه با افزایش اندازه برنامه است که حتی در برنامه‌های دارای حلقه‌های نامحدود، تعداد مسیرها به بی‌نهایت می‌رسد [10]. اجرای نمادین نمی‌تواند همه مسیرهای قابل اجرا در برنامه‌های بزرگ را اجرا کند.

مجموعه مقدار ممکن یک تقریب کران بالا و مجموعه مقدار ضروری یک تقریب کران پایین از مقدار واقعی است. در مقاله [4] به ازای هر مسیر یک نماد برای هر یک از متغیرهای ورودی اختصاص یافته و سپس آن مسیر به صورت نمادین اجرا می‌شود و بازه هر نماد محاسبه می‌شود. حال اگر مجموعه مقدار ممکن نمادی خالی باشد، بنابراین تناقض‌هایی در این نماد وجود دارد و آن مسیر غیرقابل اجرا است. همچنین اگر حداقل مجموعه مقدار ضروری یکی از نمادها مقدار داشته باشد، بنابراین آن مسیر قابل اجرا است. ترکیب‌های مختلف مقادیر مجموعه‌های مقدار ضروری می‌تواند یک مورد آزمون[15] باشد که برنامه را به ازای یک مسیر اجرا می‌کند. به طور کلی در این روش قابل اجرا بودن یا نبودن هر مسیر براساس اطلاعات بازه‌های خروجی تعیین می‌شود.

مقاله [6] روشی ایستا برای تشخیص مسیرهای غیرقابل اجرا با به کارگیری حل‌کننده SMT ارائه می‌کند. در این مقاله ابزار SMT-IPD برای انجام این کار ارائه شده است. این روش در آغاز کار یک مجموعه زیرمسیر[16] ایجاد کرده و شرط‌های موجود در هر یک از آنها را استخراج می‌کند. پس از انجام این کار، محدودیت‌های[17] موجود در شرط‌های هر مسیر را به یک

[10] Symbolic Variable
[11] Possible Value Set
[12] Necessary Value Set
[13] Range Information
[14] Path Explosion
[15] Test Case
[16] Subpath
[17] Constraint

سری نامعادله تبدیل می‌کند. سپس، با استفاده از حل‌کننده SMT نامعادله‌ها را حل کرده و زیرمسیرها را به صورت زیرمسیرهای غیرقابل اجرا و نامشخص دسته‌بندی می‌کند. مسیرهایی که در دسته نامشخص قرار گرفته‌اند دوباره آزمون می‌شوند تا قابل اجرا بودن یا نبودنشان مشخص شود، و در پایان، قابل اجرا بودن یا نبودن همه مسیرها تعیین می‌شود. در روش پیشنهادی مقاله [6] با استفاده از زیرمسیر به جای مسیر سعی شده است تا جای ممکن مشکل انفجار مسیر مرتفع گردد. چرا که تعداد زیرمسیرها در برنامه‌های پیچیده به مراتب کمتر از تعداد مسیرها می‌باشد.

ابزار SMT-IPD برای تشخیص مسیرهای غیرقابل اجرا مراحل زیر را انجام می‌دهد: در مرحله اول، مجموعه‌ای از زیرمسیرها را ایجاد می‌کند. برای این کار ابتدا CFG برنامه تحت آزمون را بدست آورده و دورهای موجود در آن را حذف می‌کند. سپس وابستگی کنترلی برنامه آزمون را براساس روش محاسبه وابستگی کنترل بدست می‌آورد. سرانجام، CFG بدون دور حاصل از قسمت تحلیل ایستا را پیمایش کرده و یک مجموعه زیرمسیر ایجاد می‌کند. در مرحله دوم، قابل اجرا بودن هر زیرمسیر توسط این ابزار تعیین می‌شود. این ابزار برای انجام این کار، با الگوریتم ساخت نامعادله‌ها شرط‌های هر زیرمسیر را به نامعادله‌هایی تبدیل و سپس آنها را توسط حل‌کننده SMT حل می‌کند، و در آخر براساس نتایج بدست آمده از حل‌کننده SMT قابل اجرا بودن یا نبودن هر زیرمسیر را تعیین می‌کند. زیرمسیری که راه‌حلی برای نامعادله‌های آن یافت شده باشد قابل اجرا است، در غیر این صورت، غیرقابل اجرا یا تعیین نشده است. برای تعیین قابل اجرا بودن زیرمسیرهای تعیین نشده، این زیرمسیرها یک بار دیگر آزمون می‌شوند. در این مرحله، شرط‌های شاخه‌ای و نقاط تعریف متغیر وابستگی زیرمسیرها تحلیل می‌شوند. شرط‌های مسیر به صورت یک سیستم نامعادله تبدیل می‌شوند و سپس توسط حل‌کننده SMT حل می‌شود. اگر سیستم نامعادله‌ها قابل حل باشد، زیرمسیر قابل اجرا است ولی اگر قابل حل نباشد، غیرقابل اجرا است. اگر نتیجه سیستم نامعادله‌ها قابل تعیین نباشد، نمی‌تواند قابل اجرا بودن زیرمسیر را نیز تعیین کرد، که در این صورت زیرمسیر باید دوباره بررسی شود. در مرحله سوم، SMT-IPD زیرمسیرها را به مسیرها بسط می‌دهد و در نهایت، قابل اجرا بودن یا نبودن همه مسیرها را تعیین می‌کند. در این مرحله، SMT-IPD مجموعه زیرمسیر را بسط داده و نتایج تشخیص مسیرهای غیرقابل اجرا را بهبود می‌بخشد. SMT-IPD هر زیرمسیر را به مسیر بسط می‌دهد (تبدیل می‌کند) و قابل اجرا بودن یا نبودن مسیرهای بسط یافته را تعیین و نتایج تشخیص را تکمیل می‌کند.

مقاله [5]، روشی ارائه کرده است که در آن مسیرهای غیرقابل اجرای بین‌رویه‌ای با استفاده از الگوهای محدودیت مسیر صدق‌ناپذیر، تشخیص داده می‌شوند. در این روش، ابتدا با کاوش ویژگی‌های محدودیت مسیر، ۹ الگوی محدودیت مسیر صدق‌ناپذیر که در مسیرهای غیرقابل اجرا رایج هستند شناسایی می‌شوند و سپس با استفاده از آنها مسیرهای غیرقابل اجرای بین‌رویه‌ای تشخیص داده می‌شوند. اگر شرط‌های محدودیت مسیری با یکی از ۹ الگو یافت شده مطابقت داشته باشد بنابراین آن مسیر یک مسیر غیرقابل اجرا می‌باشد. هر ترکیبی از شرط‌ها می‌تواند به عنوان یک محدودیت مسیر تلقی شود. دو شرطی که متغیرهای مشترک دارند به یکدیگر وابسته هستند. اگر در مجموعه شرط‌های موجود در یک مسیر، هر شرط به شرط دیگری در همان مجموعه وابسته باشد، آنگاه به آن مجموعه شرط‌های وابسته می‌گویند. اگر بین شرط‌های وابسته یک مسیر تضادی وجود داشته باشد، آنگاه شرط‌های وابسته، یک محدودیت مسیر غیرقابل حل می‌باشند. الگوی محدودیت مسیر صدق‌ناپذیر، از محدودیت‌های غیرقابل حل موجود در یک مسیر بدست می‌آید. نویسندگان مقاله [5] برای کشف الگوی محدودیت غیرقابل حل ۱۲۶۹۵ مسیر غیرقابل اجرای بین‌رویه‌ای را که در ۶ برنامه به زبان C وجود داشت را به صورت دستی بررسی کرده‌اند. سپس محدودیت غیرقابل حل هر مسیر (شرط‌های محدودیت مسیری دارای تضاد) را با استخراج محدودیت مسیر هر یک از آنها مشخص کرده‌اند. در پایان، با کاوش ویژگی‌های محدودیت مسیر غیرقابل حل مسیرهای غیرقابل اجرا ۹ الگوی محدودیت مسیر صدق‌ناپذیر کشف کرده‌اند.

مقاله [9] سعی در یادگیری حل فرمول‌های SMT دارد. حل‌کننده Z3 تاکتیک‌هایی دارد که به ترتیب (مشخص شده توسط کاربر) با هم ترکیب می‌شوند و یک استراتژی[18] را تشکیل می‌دهند. در یک فضای جستجوی گسترده، دست‌یابی به یک

[18] Strategy

استراتژی مناسب که عملکرد خوبی داشته باشد، حتی برای متخصصان این حوزه نیز می‌تواند کار بسیار سختی باشد. مقاله [9] به مسئله یافتن استراتژی مناسب می‌پردازد.

مقاله [9] مسئله حل فرمول‌های SMT را به صورت یک مسئله جستجوی درخت بیان می‌کند که در هر مرحله آن یک تغییر شکل در فرمول ورودی صورت می‌پذیرد تا زمانی که فرمول حل شود. روش مقاله [9] از دو فاز تشکیل شده است. در فاز اول، یک سیاست را با استفاده از مجموعه‌ای از فرمول‌های حل نشده یاد می‌گیرد که با استفاده از آن یک تغییر مناسب را در هر گام به منظور حل هر فرمول انتخاب و اعمال کند. به بیان دیگر این روش سیاستی را که استراتژی‌های تسریع کننده در حل فرمول‌ها را می‌یابد، نتیجه می‌دهد. در فاز دوم هم یک استراتژی را به شکل یک برنامه بدون حلقه همراه با شاخه‌ها ترکیب[19] می‌کند.

هدف مقاله [9] یافتن یک استراتژی است که زمان لازم برای حل فرمول‌های $F = \{f_i\}_{i=1}^{N}$ را به حداقل برساند. در مقاله [9] ابتدا سیاستی یاد گرفته می‌شود که مجموعه‌ای از استراتژی‌های کاندید را که تنها شامل توالی تاکتیک‌هایی است که در آن هر استراتژی در زیرمجموعه‌های مختلف مجموعه داده F عملکرد خوبی داشته است، پیدا می‌کند و سپس استراتژی‌های کاندید را با یکدیگر ترکیب می‌کند.

مقاله [11] روش‌های آزمایش کارآمد و اصولی را برای حل‌کننده‌های SMT توسعه داده است. برای این منظور، یک سیستم فازی[20] مبتنی بر یادگیری تقویتی به نام BanditFuzz را ارائه کرده است که ساختارهای دستوری ورودی‌های حل‌کننده را که علت اصلی مشکلات کارایی یا صحت در حل‌کننده‌های تحت آزمایش هستند، هدف قرار می‌دهد. در مقاله [11] با استفاده از BanditFuzz، ۱۷۰۰ ورودی منحصر به فرد از نظر نحوی کشف شده‌اند که منجر به پاسخ‌های ناسازگار در حل‌کننده‌های SMT پیشرفته Z3، CVC4، Colibri، MathSAT و Z3str3 در منطق‌های SMT ممیز شناور[21] و رشته‌ای شده است. نویسندگان مقاله [11] همچنین، با استفاده از BanditFuzz دو مجموعه معیار (با ۴۰۰ نمونه برای منطق ممیز شناور و ۳۰۰ نمونه برای رشته‌ها) ایجاد کرده‌اند که مشکلات کارایی را در همه حل‌کننده‌های در نظر گرفته آشکار می‌کند.

BanditFuzz از یادگیری تقویتی برای یادگیری اینکه کدام ساختارهای دستوری بیشترین احتمال ایجاد خطا یا مشکلات کارایی را دارند، استفاده می‌کند. فازر[22]های جهش سنتی یک عملگر جهش را به صورت تصادفی انتخاب یا اجرا می‌کنند و از رفتار برنامه‌های تحت آزمایش غافل هستند. فازر برنامه‌ای است که به طور خودکار ورودی‌های یک برنامه هدف تحت آزمایش را تولید می‌کند. در مقابل، BanditFuzz مشکل چگونگی جهش بهینه یک ورودی را به نمونه‌ای از مسئله MAB[23] یادگیری تقویتی کاهش می‌دهد. مسئله MAB یک مسئله یادگیری تقویتی رایج است که بر اساس یک MDP با یک حالت واحد $S = \{s_0\}$ و مجموعه محدودی از عمل‌های A است. ایده اصلی در BanditFuzz، حفظ فهرستی از ساختارهای دستوری است که بر اساس احتمال اینکه علت اصلی یک خطا یا مشکل کارایی باشند، رتبه‌بندی می‌شوند. در ابتدا، تمام ساختارهای دستوری به طور یکنواخت در نظر گرفته می‌شوند که احتمال ایجاد خطا توسط عامل یادگیری تقویتی وجود دارد. BanditFuzz برای شروع به طور تصادفی یک ورودی تولید می‌کند و همه برنامه‌ها در P را در ورودی I اجرا می‌کند. در هر یک از تکرارهای بعدی حلقه بازخورد خود، BanditFuzz ورودی I از تکرار قبلی خود را با استفاده از لیست رتبه‌بندی شده ساختارهای دستوری جهش می‌دهد (یعنی عامل یک عمل را انجام می‌دهد)، و همه حل‌کننده‌ها را در P بر روی نسخه جهش یافته I اجرا می‌کند. نتایج این اجراها را تجزیه و تحلیل می‌کند تا بازخورد (یعنی پاداش) را به عامل یادگیری تقویتی ارائه دهد که به احتمال زیاد باعث ایجاد تفاوت کارایی نسبی بین برنامه هدف T نسبت به برنامه مرجع R می‌شود. سپس لیست ساختارهای دستوری خود را با هدف به حداکثر رساندن پاداش (یعنی به حداکثر رساندن تفاوت کارایی نسبی بین حل‌کننده‌ها در P) به روز و دوباره رتبه‌بندی می‌کند. این فرآیند تا زمانی ادامه می‌یابد که عامل یادگیری تقویتی به یک رتبه‌بندی همگرا شود و منابع آن تمام شود.

---

[19] Synthesize
[20] Fuzzing System
[21] Floating Point
[22] Fuzzer
[23] Multi-Armed Bandits

مقاله [12] با استفاده از الگوریتم اجرای کانکالیک[۲۴] و یادگیری تقویتی عمیق[۲۵] و راه حل فازی ترکیبی[۲۶] ابزاری برای یافتن باگ‌های عمیق به نام Dr.PathFinder ارائه می‌کند. در این مقاله، بر کاوش باگ‌های واقع در مسیر عمیق در مسیرهای اجرایی برنامه تمرکز شده است. برای دستیابی به این هدف، یک الگوریتم اجرای کانکالیک همراه با یادگیری تقویتی عمیق و راه حل فازی ترکیبی، برای یافتن باگ‌های عمیق پیشنهاد شده است. تجزیه و تحلیل برنامه، از جمله کشف آسیب‌پذیری، به دلیل فضای حالت نامتناهی آن به عنوان یک مسئله غیرقابل تصمیم‌گیری شناخته می‌شود. بنابراین، برای ردیابی همه حالت‌ها، Dr.PathFinder حالت‌ها را با استفاده از یک شبکه عصبی عمیق تخمین می‌زند. سپس، بر اساس این تقریب، Dr.PathFinder شاخه‌ای را انتخاب می‌کند که انتظار می‌رود مسیر طولانی‌تری داشته باشد. برای این هدف، نویسندگان مقاله [12] یک الگوریتم یادگیری پیشنهاد داده‌اند که به عامل اجازه می‌دهد تا مسیرهایی را جستجو کند که انتظار می‌رود در طول اجرای کانکالیک عمیق‌تر باشند. عامل آموزش دیده به طور انتخابی شاخه‌هایی را که برای بهبود پوشش مفید نیستند هرس می‌کند. این کار در نهایت با محدود کردن مناسب تولید ورودی‌ها، فضای جستجوی اجرای کانکالیک را کاهش می‌دهد و در نتیجه فازر را قادر می‌سازد تا باگ‌های عمیق‌تری را پیدا کند. برخلاف بسیاری از فازرها که تقریبا به صورت متوالی از مسیرهای کم عمق شروع به جستجو می‌کنند، Dr.PathFinder بر روی مسیرهای عمیق‌تر تمرکز می‌کند.

روش عملکرد این روش به صورت زیر است که ابتدا فازر یک برنامه هدف و یک صف با موارد آزمون ساده را به عنوان آرگومان می‌گیرد. فازر یک مورد آزمون را از صف می‌گیرد و از آن به عنوان دانه‌ای برای الگوریتم جهش برای ایجاد یک مورد آزمون جدید استفاده می‌کند. برنامه هدف با این آزمون به عنوان ورودی اجرا می‌شود. هنگامی که فازر یک انتقال (پوشش) بلوک پایه جدید را پیدا می‌کند یا در حین اجرای برنامه هدف دچار خطا می‌شود، فازر مورد آزمون را جالب در نظر می‌گیرد و آنرا در صف قرار می‌دهد. پس از تکرار مراحل قبل اگر پوشش جدیدی پیدا نشد، فازر موتور اجرای کانکالیک تقویت عمیق را فراخوانی می‌کند. این موتور ابتدا موارد آزمون را از صف دریافت می‌کند و برنامه هدف را اجرا می‌کند. سپس موتور، محدودیت‌های مسیر شاخه‌ای را که در حین اجرای برنامه با آن مواجه می‌شود منفی می‌کند و اجرای نمادین را آغاز می‌کند. هنگامی که عامل DQN با یک حالت شرطی روبرو می‌شود، اجرای نمادین را به شاخه‌ای که در آن مسیر عمیق‌تری انتظار می‌رود هدایت می‌کند. پس از تکرار انتخاب شاخه و ردیابی مسیرها با عامل DQN، حل‌کننده SMT محدودیت‌های مسیر جمع‌آوری شده را حل می‌کند و موارد تست جدیدی را برای تحقق مسیرها ایجاد می‌کند. موارد آزمون تولید شده در صف قرار داده می‌شوند و دوباره استفاده می‌شوند.

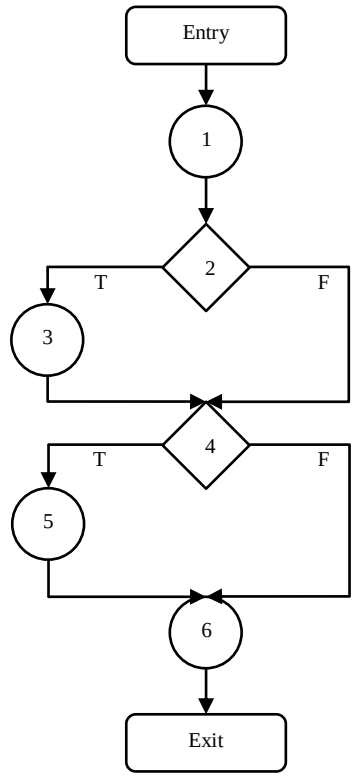
۳. مفاهیم مقدماتی

در این بخش به معرفی و تعریف برخی از مفاهیم مورد استفاده در این مقاله پرداخته شده است.

۳-۱. گراف جریان کنترلی

در آزمون نرم‌افزار می‌توان ساختارهای کمکی تهیه کرد که به آزمون کمک کنند. گراف‌ها رایج‌ترین ساختار برای کمک به آزمون نرم‌افزار هستند. گراف‌ها در حوزه های مختلف مانند [13] کاربرد دارند. گراف جریان کنترلی (با مخفف CFG)، مدل استاندارد نمایش جریان اجرای بین حالت‌های برنامه است [14]. در گراف جریان کنترلی گره‌ها نشان دهنده حالت‌ها و یال‌ها نشان دهنده جریان کنترلی بین حالت‌ها هستند. گراف جریان کنترلی در واقع یک مدل از نرم‌افزار ارائه می‌دهد. برای مثال شکل ۱ یک تابع به نام abs و گراف جریان کنترلی متناظر با آنرا نشان می‌دهد.

[24] Concolic Execution
[25] Deep Reinforcement Learning
[26] Hybrid Fuzzing Solution




```
void int abs(int x) {
1:      int abs_value = 0;
2:      if (x >= 0)
3:              abs_value = x;
4:      if (x < 0)
5:              abs_value = -x;
6:      return abs_value;
}
```

شکل ۱ تابع قدر مطلق در سمت راست شکل و گراف جریان کنترلی متناظر آن در سمت چپ شکل [14]

گراف جریان کنترلی بین‌رویه‌ای[۲۷]، (با مخفف ICFG نیز شناخته می‌شود) نوع دیگری از گراف جریان کنترلی در علوم کامپیوتر است. در این نوع گراف‌ها، گراف جریان کنترلی رویه‌های فراخوانی شده در داخل رویه اصلی نیز با گراف جریان کنترلی رویه اصلی ترکیب می‌شوند. به عبارت دیگر، در گراف جریان کنترلی بین‌رویه‌ای، ارتباط بین رویه‌ها نیز در نظر گرفته می‌شود، این ویژگی باعث می‌شود تعداد مسیرها و گره‌ها در هر مسیر گراف جریان کنترلی بین‌رویه‌ای بیشتر از گراف جریان کنترلی ساده باشد.

در آزمون نرم‌افزار هدف پوشش گرافی است که ساخته‌ایم. اما معیارهای پوشش[۲۸] متفاوت هستند. پوشش گره‌ها، یال‌ها و یا پوشش مسیرها هر کدام می‌توانند یک معیار پوشش باشند [1]. برای مثال، وقتی می‌گوییم معیار مورد نظر ما پوشش گره است، بدین معنی است که در آزمون، هر کدام از گره‌های گراف را حداقل یکبار ملاقات کنیم و گره ملاقات شده‌ای نداشته باشیم. به طور مشابه، در معیار پوشش مسیر نیز هدف پیمایش یا اجرای همه مسیرهای موجود در گراف جریان کنترلی است. هر مسیر در گراف جریان کنترلی، در واقع یک مسیر از گره شروع تا گره پایان گراف جریان کنترلی است.

۳-۲. مسیرهای غیرقابل اجرا

در گراف جریان کنترلی مسیرهایی وجود دارند که به ازای هیچ مقدار ورودی، نمی‌توان آنها را اجرا کرد که به این مسیرها، مسیرهای غیرقابل اجرا می‌گویند [14]. وجود مسیرهای غیرقابل اجرا در برنامه، تولید خودکار داده آزمون برای آن را دچار مشکل می‌کند. تولیدکننده خودکار داده آزمون[۲۹] تلاش می‌کند تا داده‌های آزمونی تولید کند که برنامه را، با توجه به معیار پوشش انتخاب شده، به صورت کامل تحت آزمون قرار دهد. اگر معیار پوشش مورد نظر، پوشش مسیر باشد، و مسیرهای

[27] Interprocedural Control Flow Graph
[28] Coverage Criteria
[29] Test Data Generator

غیرقابل اجرا در برنامه وجود داشته باشد، تولیدکننده داده آزمون در یک حلقه بی انتها تلاش خواهد کرد داده‌هایی برای آزمون این مسیرها بیابد و موفق نخواهد شد. این موضوع موجب هدر رفت منابع و زمان می‌شود [6]. با توجه به آنچه که بیان شد، با تشخیص و حذف این مسیرها می‌توان از این مشکل پیشگیری و در منابع و زمان صرفه‌جویی نمود.

یکی دیگر از دلایل اهمیت مسیرهای غیرقابل اجرا، مرتبط با امنیت است [15]. در مقاله مورد نظر، نویسندگان به این موضوع اشاره کرده‌اند که برخی از بدافزارها کدهای مخرب خود را در مسیرهای غیرقابل اجرا مخفی می‌کنند تا از شناسایی شدن آنها جلوگیری کنند. با تشخیص مسیرهای غیرقابل اجرا و حذف آنها، می‌توان کدهای مخرب بدافزارها را از بین برد. همچنین تشخیص مسیرهای غیرقابل اجرا در بهینه‌سازی کد نیز کاربرد دارد [14]. امنیت یکی از مسائل مهم و حیاتی است که در [16] مورد بررسی قرار گرفته است.

به طور کلی، مسیرهای غیرقابل اجرا در گراف جریان کنترلی به دلایل مختلفی وجود دارند. یکی از اصلی‌ترین دلایل، تضاد میان حالت‌های شرطی مسیر، یا به عبارت دیگر، وابستگی بین حالت‌های شرطی به واسطه یک متغیر مشترک است [17]. این وابستگی معمولاً با نماد $X \rightarrow Y$ نشان داده می‌شود، که در آن X نتیجه حالت شرطی اول (صحیح یا غلط) و Y نتیجه حالت شرطی دوم را نمایش می‌دهد. به عنوان مثال، در تابع شکل ۱ و مسیر $P4$، دو حالت شرطی وجود دارند: حالت‌های ۲ و ۴. هر دو حالت شرطی در متغیر x مشترک هستند و با توجه به آن تعریف شده‌اند، اما شرط هرکدام با دیگری در تضاد است؛ به عبارت دیگر، $x >= 0$ با $x < 0$ در تضاد است. وابستگی بین این دو حالت شرطی به صورت $true \rightarrow false$ و $false \rightarrow true$ نشان داده می‌شود، به این معنی که در مسیری که از این دو حالت شرطی عبور می‌کند، زمانی که نتیجه شرط اول صحیح باشد، باید نتیجه شرط دوم غلط باشد و یا بالعکس تا آن مسیر قابل اجرا باشد. بنابراین هر مسیری که وابستگی‌های $false \rightarrow false$ و $true \rightarrow true$ داشته باشد (مانند مسیرهای $P1$ و $P4$)، یک مسیر غیرقابل اجرا خواهد بود.

همچنین، وابستگی بین حالت‌های شرطی و حالت‌های تخصیص[30] (حالت‌های غیرشرطی) در برنامه‌نویسی نیز می‌توانند منجر به مسیرهای غیرقابل اجرا شوند [6]. به حالتی که در آن یک مقدار به یک متغیر تخصیص می‌یابد، حالت تخصیص گفته می‌شود. برای مثال فرض کنید یک حالت تخصیص مانند $d = 20$ داریم که بدون اینکه مقدار متغیر آن در حالت دیگری تغییر یابد، در یک حالت شرطی با شرط $d > 30$ به کار برده می‌شود. بنابراین هر مسیری که در آن ابتدا باید از حالت تخصیص گفته شده و سپس از شاخه صحیح حالت شرطی ذکر شده عبور شود یک مسیر غیرقابل اجرا می‌باشد.

علاوه بر موارد ذکر شده، وجود حالت‌های شرطی که هیچ وقت مقدار آنها صحیح نخواهد شد نیز می‌توانند باعث ایجاد کدهای مرده[31] شوند [17]. از آنجایی که حالت‌های کدهای مرده حالت‌هایی هستند که قابل دستیابی نیستند و هرگز نمی‌توانند اجرا شوند، می‌توانند باعث ایجاد مسیرهای غیرقابل اجرا شوند.

در مقاله [18] گفته شده است که احتمال غیرقابل اجرا بودن یک مسیر با تعداد حالت‌های شرطی در آن مسیر رابطه مستقیم داد و هرچه تعداد حالت‌های شرطی بیشتر باشد احتمال غیرقابل اجرا بودن آن مسیر نیز بیشتر است. به عبارت دیگر شرط‌های کمتر در هر مسیر به معنی بالاتر بودن احتمال قابل اجرا بودن آن مسیر است [19].

۳-۳. صدق‌پذیری پیمانه نظریه‌ها

صدق‌پذیری پیمانه نظریه‌ها که به صورت مخفف SMT به کار برده می‌شود، مسئله تصمیم‌گیری در مورد صدق‌پذیری فرمول‌های منطقی مرتبه اول، براساس برخی از نظریه‌های منطقی است [8]. نسخه ساده‌تر مسئله SMT، مسئله صدق‌پذیری دودویی است که به صورت مخفف SAT به کار برده می‌شود. هدف مسئله صدق‌پذیری دودویی، بررسی صدق‌پذیری یک مسئله بولی[32] است. منظور از صدق‌پذیری تعیین این موضوع است که آیا می‌توان برای متغیرهای یک رابطه یا فرمول، مقادیری تعیین کرد که در آن رابطه صدق کند یا نه؟ برای مثال عبارت $a + 1 = 0$ به ازای $a = -1$ صادق (صدق‌پذیر) می‌باشد اما به ازای $a = 1$ صادق نمی‌باشد. در حوزه‌های مختلف علوم کامپیوتر، بسیاری از مسائل را می‌توان به مسئله بررسی صدق‌پذیری یک

[30] Assignment Statement
[31] Dead Code
[32] Boolean

فرمول در یک منطق کاهش داد. حل‌کننده‌های مختلفی وجود دارند که مسائل SMT را حل می‌کنند. یکی از بهترین حل‌کننده‌ها، حل‌کننده Z3 می‌باشد که توسط Microsoft Research به‌طور رایگان در اختیار عموم قرار داده شده است که در این مقاله نیز به کار گرفته شده است. Z3 برای تجزیه و تحلیل برنامه‌های کاربردی و تایید آنها استفاده می‌شود [7]. حل‌کننده Z3 از ۳ قالب[۳۳] ورودی SMT-LIB و Simplify و DIMACS پشتیبانی می‌کند. قالب ورودی پیش‌فرض Z3، قالب SMT-LIB است. SMT-LIB یک نوآوری بین‌المللی با هدف تسهیل تحقیق و توسعه در صدق‌پذیری پیمانه نظریه‌ها یا همان SMT است [20]. SMT-LIB یک زبان استاندارد برای بیان فرمول‌ها و رابطه‌ها ارائه می‌کند. در SMT-LIB فرمول‌ها دارای منطق‌های مختلفی مانند QF_NRA، QF_NIA و... می‌باشند [20]. دلیل وجود منطق‌های مختلف این است که بتوان در عمل از روش‌های کارآمدتری برای حل فرمول‌های آن منطق استفاده کرد. این منطق‌ها با دسته‌ای از حروف نام‌گذاری می‌شوند که تئوری‌های استفاده شده در آن منطق را نشان می‌دهند. برای مثال عبارت QF نشان‌دهنده فرمول‌های بدون سور[۳۴]، IA نشان‌دهنده محاسبات اعداد صحیح[۳۵]، RA نشان‌دهنده محاسبات اعداد حقیقی و N قبل از RA و یا IA به معنای غیرخطی بودن است. بنابراین QF_NRA نشان‌دهنده فرمول‌های اعداد حقیقی غیرخطی بدون سور است. برای اطلاعات بیشتر در مورد منطق‌ها می‌توان به [20] مراجعه کرد.

حل‌کننده Z3، حاوی تعداد زیادی ترکیب اکتشافی[۳۶] دست‌ساز و کاملا یکپارچه از روش‌های اثبات الگوریتمی است که به آنها تاکتیک گفته می‌شود. در Z3 می‌توان این تاکتیک‌ها را روی فرمول ورودی اعمال کرد و هر تاکتیک می‌تواند فرمول ورودی را به یک فرمول دیگر که حل آن توسط حل‌کننده Z3 آسانتر و سریعتر است، تبدیل نماید. حل‌کننده Z3 دارای بیش از ۱۰۰ عدد تاکتیک می‌باشد که هر کدام از آنها می‌توانند پارامترهای مخصوص خود را نیز داشته باشند. مانند تاکتیک simplify که بیش از ۵۰ پارامتر متفاوت دارد. برای مثال تاکتیک simplify، ثابت‌ها و عبارات تکراری را حذف می‌کند (مانند حذف عدد ثابت صفر از $x - 0$ یا جایگزین کردن عبارت $x - x$ با عدد صفر).

علاوه بر تاکتیک‌ها، حل‌کننده Z3 قابلیت‌های دیگری را نیز ارائه می‌کند. یکی از این قابلیت‌ها امکان اندازه‌گیری و شمارش برخی مقادیر مانند تعداد اعداد ثابت فرمول و... در فرمول ورودی می‌باشد، که به آنها سنجه قاعده[۳۷] گفته می‌شود. سنجه‌های قاعده با استفاده از فرمول ارزیابی می‌شوند. مانند سنجه قاعده num-consts که تعداد ثابت‌های تفسیر نشده[۳۸] غیر بولی موجود در فرمول هدف را می‌شمارد. اعمال تاکتیک روی فرمول می‌تواند مقادیر سنجه‌های قاعده را تغییر دهد.

Z3 همچنین این امکان را می‌دهد تا بتوان آمار[۳۹]هایی مانند مقدار حافظه مورد استفاده یا زمان لازم برای حل فرمول مورد نظر را بدست آورد. یکی از کاربردی‌ترین آمارها که در این مقاله نیز مورد استفاده قرار گرفته است، rlimit می‌باشد. به بیان ساده، rlimit تعداد عملیات اساسی انجام شده توسط حل‌کننده برای حل فرمول را نشان می‌دهد.

۳-۴. یادگیری تقویتی

به طور کلی، یادگیری تقویتی به معنای یادگیری راه‌هایی است که باعث بهبود عملکرد یک عامل در محیط خاص می‌شود. در این نوع یادگیری، عامل با تعامل با محیط، عمل[۴۰]های مختلف را انجام می‌دهد و سعی می‌کند تا سیگنال پاداش عددی را به حداکثر برساند. یکی از ویژگی‌های مهم یادگیری تقویتی، عدم دانستن عمل‌های بهینه از قبل است. به عبارت دیگر، به یادگیرنده نمی‌گوییم که کدام عمل‌ها را انجام دهد؛ بلکه او باید با آزمون و خطا کشف کند که کدام عمل‌ها بیشترین پاداش را در پی خواهند داشت. جستجو با آزمون و خطا و پاداش با تاخیر، دو ویژگی برجسته یادگیری تقویتی هستند [21]. به طور کلی، یادگیری تقویتی ایده یادگیری از طریق تعامل با یک محیط است.

---

[33] Format
[34] Quantifier Free
[35] Integer Arithmetic
[36] Heuristic
[37] Probes
[38] Uninterpreted constants
[39] Statistics
[40] Action

در یادگیری تقویتی یک محیط وجود دارد که عامل عمل‌هایی را در آن انجام می‌دهد و براساس عمل‌هایی که انجام داده است، پاداش‌ها و تنبیه‌هایی را دریافت می‌کند و براساس آنها یک سیاست بهینه را یاد می‌گیرد. در مراحل بعدی عامل برای انتخاب و انجام عمل‌های خود در محیط از سیاستی که یاد گرفته است استفاده می‌نماید. علاوه بر عامل و محیط که به آنها اشاره شد، سیاست[۴۱]، سیگنال پاداش[۴۲] و تابع ارزش[۴۳] سه عنصر اصلی دیگر در یادگیری تقویتی می‌باشند. سیاست نحوه رفتار عامل در یک زمان معین را مشخص می‌کند. به طور کلی، یک سیاست نگاشت از حالت[۴۴]های درک شده محیط به عمل‌هایی است که باید در آن حالت‌ها انجام شود. در واقع، سیاست‌ها احتمالات را برای هر عمل مشخص می‌کنند. سیگنال پاداش، هدف یک مسئله یادگیری تقویتی را مشخص می‌کند. محیط در هر گام زمانی[۴۵]، یک عدد واحد را به عنوان پاداش به عامل یادگیری تقویتی می‌فرستد. هدف عامل به حداکثر رساندن پاداش با تاخیر است. منظور از پاداش با تاخیر، کل پاداشی است که عامل در بلند مدت دریافت می‌کند. بنابراین سیگنال پاداش، مشخص کننده عمل‌های خوب و بد برای عامل است. مبنای اصلی برای تغییر سیاست، عنصر موثر برای تغییر سیاست سیگنال پاداش است که می‌تواند باعث تغییر سیاست در آینده شود. در حالی که سیگنال پاداش میزان خوب بودن عمل‌ها در کوتاه مدت را نشان می‌دهد، یک تابع ارزش مشخص می‌کند که چه عمل‌هایی در بلند مدت خوب هستند. به طور کلی، مقدار کل پاداشی که عامل می‌تواند با شروع از یک حالت در بلند مدت بدست آورد، ارزش آن حالت را تعیین می‌کند. در حالی که پاداش‌ها، مطلوبیت ذاتی و فوری حالات محیطی را تعیین می‌کنند، ارزش‌ها مطلوبیت بلند مدت حالت‌ها را نشان می‌دهند.

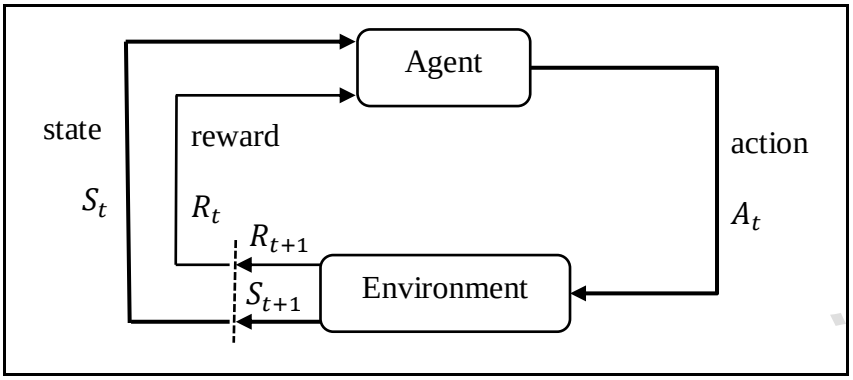
در یادگیری تقویتی در ابتدای کار عامل در یک حالت اولیه از محیط قرار دارد. عامل در هر حالت با توجه به وضعیتی که در آن قرار دارد و نیز براساس تجربیاتی که یاد گرفته است یک عمل را انتخاب می‌کند و آنرا در محیط انجام دهد. پس از اعمال عمل انتخاب شده در محیط توسط عامل، محیط پاداشی به عامل می‌دهد و عامل به یک حالت یل وضعیت دیگر در محیط انتقال می‌یابد. عمل انتخاب و انجام عمل توسط عامل در حالت‌های مختلف محیط تا رسیدن به یکی از حالت‌های پایانی محیط ادامه می‌یابد. به حالت‌هایی که عامل با شروع از حالت اول تا رسیدن به حالت آخر (حالت پایان) مشاهده می‌کند، یک دوره[۴۶] گفته می‌شود. عامل دوره‌های مختلف و حتی تکراری را در محیط سپری می‌کند. هدف عامل در یادگیری تقویتی، بهبود سیاست‌های خود است، به‌طوری که مجموع پاداش‌های دریافتی در طول زمان بیشینه[۴۷] شود.

اکتشاف[۴۸] و بهره‌برداری[۴۹] دو مفهوم کلیدی در یادگیری تقویتی هستند. در بهره‌برداری، عامل از دانشی که تاکنون جمع‌آوری کرده است، برای انتخاب عمل‌ها استفاده می‌کند که انتظار دارد پاداش بیشتری داشته باشند. هدف از بهره‌برداری، بهبود پاداش در کوتاه‌مدت است. در اکتشاف، عامل عمل‌هایی را انتخاب می‌کند که نتایج آن‌ها نامعلوم است. اکتشاف به عامل امکان می‌دهد تا اطلاعات جدیدی از محیط جمع‌آوری کند و دانش خود را افزایش دهد. تعادل بین اکتشاف و بهره‌برداری، در مسائل یادگیری تقویتی بسیار مهم است. اگر عامل فقط از تجربیات گذشته استفاده کند، ممکن است در یک سیاست نامناسب گیر کند.

در یادگیری تقویتی، مسائل به صورت یک فرایند تصمیم‌گیری مارکوف[۵۰] (MDP) مدل می‌شوند. فرایند تصمیم‌گیری مارکوف، رسمی‌سازی[۵۱] تصمیم‌گیری ترتیبی است. در این فرایند، عامل در حالت s قرار دارد و یک عمل a را انتخاب می‌کند. پس از انجام هر عمل، عامل به حالت جدیدی منتقل می‌شود و پاداش مربوط به عمل a را دریافت می‌کند. شکل ۲ نحوه تعامل یک عامل با محیط را در یک فرایند تصمیم‌گیری مارکوف نشان می‌دهد.

---

[41] Policy
[42] Reward Signal
[43] Value Function
[44] State
[45] Time Step
[46] Episode
[47] Maximize
[48] Exploration
[49] Exploitation
[50] Markov Decision Process
[51] Formalization



شکل ۲ تعامل عامل با محیط در فرایند تصمیم‌گیری مارکوف [21]

در شکل ۲، t نشان‌دهنده گام زمانی، S_t حالتی است که عامل در گام زمانی t در آن حالت قرار دارد، A_t عملی است که عامل در حالت S_t انجام می‌دهد. عامل بعد از انجام عمل A_t از حالت S_t به حالت بعدی یعنی S_{t+1} منتقل می‌شود و پاداش R_{t+1} را دریافت می‌کند.

یادگیری تقویتی عمیق زیرمجموعه‌ای از یادگیری تقویتی است که اصول یادگیری عمیق[۵۲] را با روش‌های یادگیری تقویتی ترکیب می‌کند. در این رویکرد، شبکه‌های عصبی عمیق به عنوان تقریب‌گرهای تابع ارزش یا سیاست در فرایند تصمیم‌گیری مارکوف (MDP) استفاده می‌شوند. این روش‌ها به خصوص در مسائلی که تعداد حالت‌ها زیاد است، عملکرد مناسبی دارند. به همین دلیل در این مقاله نیز از عامل یادگیری عمیق استفاده شده است.

۴. تعریف مسئله

همانطور که در بخش ۳ گفته شد، مسیرهای غیرقابل اجرا مسیرهایی هستند که با هیچ داده ورودی نمی‌توان آنها را اجرا کرد. برای مثال، در گراف جریان کنترلی شکل ۱، چهار مسیر مختلف به شرح زیر از گره شروع تا گره پایان وجود دارد:

$$P1 = (entry, 1, 2, 4, 6, exit)$$

$$P2 = (entry, 1, 2, 4, 5, 6, exit)$$

$$P3 = (entry, 1, 2, 3, 4, 6, exit)$$

$$P4 = (entry, 1, 2, 3, 4, 5, 6, exit)$$

اگر با تخصیص مقادیر مختلف به پارامتر x، هر مسیر حداقل یکبار پیمایش یا به عبارت دیگر اجرا شود، در اینصورت گفته می‌شود که آزمون‌ها تابع را به طور کامل پوشش داده‌اند. در مورد مثال شکل ۱، هیچ مقداری برای پارامتر ورودی x وجود ندارد که بتواند دو مسیر $P1$ و $P4$ را پیمایش کند. لذا، این دو مسیر غیرقابل اجرا هستند.

تشخیص قابل اجرا بودن یا نبودن یک مسیر، یک مسئله تصمیم‌ناپذیر است و این موضوع در مقاله [22] نشان داده شده است. بنابراین، یک روش کلی برای تشخیص مسیرهای غیرقابل اجرا وجود ندارد. اما با این حال روش‌های جزئی مختلفی برای تشخیص مسیرهای غیرقابل اجرا ارائه شده‌اند که تلاش می‌کنند تا جای ممکن، مسیرهای غیرقابل اجرا را تشخیص دهند. این روش‌ها به دو دسته روش‌های تحلیل ایستا[۵۳] و روش‌های تحلیل پویا[۵۴] تقسیم می‌شوند [6]، [23]، [24]. در روش تحلیل ایستا

[52] Deep Learning
[53] Static Analysis
[54] Dynamic Analysis

بدون اجرا کردن کدهای برنامه تحت آزمون، کدهای آن تجزیه و تحلیل شده و آزمون می‌شوند. در مقابل روش‌های تحلیل ایستا، روش‌های تحلیل پویا قرار دارند که در آنها کدهای برنامه تحت آزمون برای تجزیه و تحلیل عملکرد برنامه و آزمون آن اجرا می‌شوند. هر کدام از این روش‌ها می‌توانند برخی از مسیرهای غیرقابل اجرا را تشخیص دهند. ما در این مقاله بیشتر روی روش‌های تحلیل ایستا تمرکز داریم.

یکی از روش‌های ایستا برای تشخیص مسیرهای غیرقابل اجرا، استفاده از حل‌کننده‌های SMT برای بررسی قابل حل بودن یا نبودن شرط‌های موجود در هر مسیر است [6]. در این روش شرط‌های موجود در هر مسیر از گراف جریان کنترلی استخراج شده، سپس به صورت نامعادله‌هایی به حل‌کننده‌های SMT ارسال می‌شوند. پس از حل نامعادله‌ها، سازگاری و یا ناسازگاری بین شرط‌ها بررسی شده و براساس آن در مورد قابل اجرا بودن یا نبودن مسیر تصمیم‌گیری می‌شود. برای مثال نامعادله $x > 1 \hat{} x \leq 4$ به ازای $x = \{2\}$ قابل حل می‌باشد. حل‌کننده SMT قابل حل بودن و یا نبودن نامعادله‌ها را تعیین می‌کند. اگر نامعادله‌ها قابل حل باشند، پس مسیر متناظر آنها قابل اجرا می‌باشد، در غیر اینصورت، آن مسیر غیرقابل اجرا است.

حل‌کننده‌های SMT برای حل نامعادله‌ها و فرمول‌های منطقی با قوانین خاص، استفاده می‌شوند. یکی از معروف‌ترین حل‌کننده‌های SMT حل‌کننده Z3 است [9]. این ابزار در مسائل مختلف کاربر دارد، از جمله تحلیل ایستا و تولید موارد آزمون [7]. حل‌کننده Z3 دارای تاکتیک‌هایی می‌باشد. تاکتیک‌ها در حل‌کننده Z3 برای بهبود و سرعت‌بخشی به حل نامعادله‌ها استفاده می‌شوند. هر تاکتیک، نامعادله یا فرمول ورودی را به یک نامعادله یا فرمول دیگر تبدیل می‌کند تا حل آن ساده‌تر شود. انتخاب تاکتیک‌های مناسب برای هر ورودی مهم است و می‌تواند تاثیر زیادی در سرعت حل داشته باشد. به طور خلاصه، حل‌کننده Z3 با استفاده از تاکتیک‌های مختلف، نامعادله‌ها را به صورت سریع‌تر و دقیق‌تر حل می‌کنند. انتخاب تاکتیک‌های مناسب و تجربه در استفاده از آنها، کلید موفقیت در حل مسائل پیچیده است.

در صورتی که تعداد شرط‌های موجود در هر مسیر از گراف جریان کنترلی زیاد باشد، نامعادله/فرمول ایجاد شده نیز بزرگتر و پیچیده‌تر خواهد بود. در گراف جریان کنترلی بین رویه‌ای، تعداد شرط‌های موجود در هر مسیر حتی بیشتر هم می‌شود. در چنین شرایطی حل نامعادله/فرمول توسط Z3 زمانبر خواهد بود. بنابراین سرعت تشخیص مسیرهای غیرقابل اجرا به سرعت حل نامعادله‌ها/فرمول‌ها توسط Z3 وابسته است و با افزایش سرعت حل کننده Z3، سرعت تشخیص مسیرهای غیرقابل اجرا نیز افزایش می‌یابد. انتخاب و اعمال تاکتیک‌های مناسب روی نامعادله/فرمول می‌تواند در سرعت حل آن توسط Z3 تاثیر بگذارد. اما همانطور که گفته شد، انتخاب تاکتیک مناسب برای هر فرمول کار آسانی نیست. در روش پیشنهادی مقاله با استفاده از یادگیری تقویتی و انتخاب تاکتیک‌های مناسب برای فرمول‌های مختلف سرعت حل‌کننده Z3 افزایش می‌یابد.

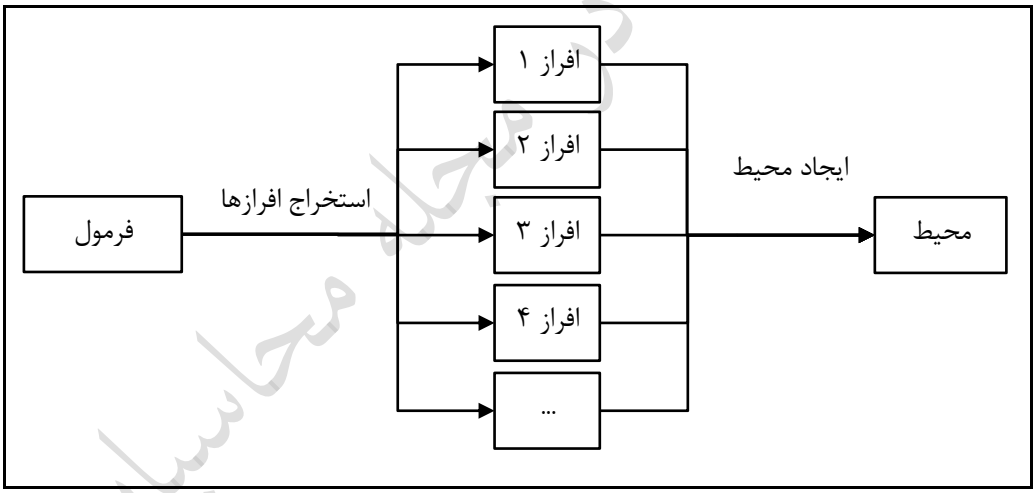
۵. روش پیشنهادی

از آنجایی که انتخاب تاکتیک‌های مناسب در حل‌کننده‌های SMT تاثیر قابل توجهی در سرعت و دقت این حل‌کننده‌ها دارد، روش پیشنهادی مقاله استفاده از یادگیری تقویتی برای انتخاب تاکتیک‌ها و بهبود عملکرد حل‌کننده‌های SMT است که این کار تشخیص هر چه بهتر و سریعتر مسیرهای غیرقابل اجرا را به دنبال دارد. نحوه عملکرد یادگیری تقویتی از طریق آزمون و خطا و تنبیه و پاداش می‌باشد، بنابراین نیازی به داشتن دانش قبلی نیست. با استفاده از یادگیری تقویتی می‌توان بدون داشتن دانش قبلی و اینکه یک تاکتیک روی یک فرمول چه تاثیری می‌گذارد، در مورد تاکتیک‌ها یاد گرفت. بدین شکل که عامل یادگیری تقویتی به ازای تاکتیک‌های موثر بر فرمول، پاداش دریافت می‌کند. تاکتیک‌های موثر تاکتیک‌هایی هستند که باعث سریعتر حل شدن فرمول می‌شوند و یا اینکه آنرا به طور صریح قابل حل می‌کنند. در واقع عامل یاد میگیرد که چه تاکتیک‌هایی برای کدام مجموعه فرمول‌ها مناسب هستند. با استفاده از یادگیری تقویتی می‌توان در مورد مسائل مختلف تصمیم‌گیری کرد و انتخاب تاکتیک مناسب یک عمل تصمیم‌گیری است. بنابراین استفاده از یادگیری تقویتی می‌تواند مفید واقع شود.

همانطور که در بخش ۳ (مفاهیم مقدماتی) ذکر شد، یادگیری تقویتی از ۵ عنصر اصلی تشکیل شده است، که ابتدا به معرفی آنها در روش پیشنهادی می‌پردازیم و سپس نحوه عملکرد روش پیشنهادی را شرح می‌دهیم. کدهای منبع روش پیشنهادی به صورت متن باز در دسترس می‌باشد[۵۵].

۵-۱. محیط

در ابتدای کار یک محیط باید وجود داشته باشد تا عامل بتواند در آن آموزش ببیند. محیطی که برای آموزش عامل طراحی شده است، با استفاده از یک فرمول در قالب SMT که به صورت یک فایل با پسوند smt2. در دسترس است و توسط حل‌کننده Z3 قابل خواندن است، ایجاد می‌شود. بنابراین می‌توان با هر فایل SMT یک محیط جداگانه ایجاد کرد. محیط باید به گونه‌ای طراحی شود که وابسته به یک فرمول خاص نباشد. همچنین نمی‌توان از همه فرمول‌ها برای ایجاد یک محیط استفاده کرد، چرا بی‌نهایت فرمول می‌تواند وجود داشته باشد. بنابراین محیط به جای اینکه وابسته به یک یا چند فرمول باشد، می‌تواند وابسته به افرازهایی باشد که بین فرمول‌های مختلف مشترک هستند. این افرازها می‌توانند گسترده و مختلف باشند و برخی از آنها باید انتخاب و برای طراحی محیط استفاده شوند. در واقع محیط می‌تواند با استفاده از افرازهایی که از یک فرمول بدست می‌آید ساخته شود. در این حالت، با اینکه محیط با استفاده از یک فرمول ایجاد می‌شود اما در واقع مجموعه‌ای از فرمول‌ها را شامل می‌شود. این افرازها می‌توانند ویژگی‌هایی مانند منطق فرمول، تعداد عملگرهای مختلف و... باشند که می‌توان از یک فرمول استخراج کرد. شکل ۳ این فرایند را نشان می‌دهد.



شکل ۳ فرایند ایجاد محیط با استفاده از افرازها یا ویژگی‌های فرمول

در این مقاله سعی شده است محیط به گونه‌ای طراحی شود که بتوان عامل‌های مختلف یادگیری تقویتی را در این محیط آموزش داد. از این رو تلاش شده است محیط طراحی شده، تا حد امکان به محیط‌های کتابخانه OpenAI Gym [25] تشابه داشته باشد. طراحی محیط به این شکل، این امکان را برای محققان فراهم می‌کند تا در صورت نیاز بتوانند به آسانی از این محیط در مطالعات خود بهره بگیرند.

۵-۱-۱. حالت‌ها

با توجه به اینکه تعداد فرمول‌ها بسیار زیاد و بلکه نامتناهی است، در نظر گرفتن هر فرمول به عنوان یک حالت از محیط ایده چندان مناسبی به نظر نمی‌رسد. این روش در مقاله [9] مورد استفاده قرار گرفته و باعث شده که نتیجه قابل تعمیم روی

[55] https://github.com/majidfeyzi/Z3-tactics-learning

فرمول‌های مختلف نباشد. به جای آن، در این مقاله سعی شده است ویژگی‌هایی برای فرمول‌ها در نظر گرفته شود که بر اساس این ویژگی‌ها و مقادیر آنها، بتوان فرمول‌ها را دسته بندی کرده و هر دسته را به عنوان یک حالت از محیط محسوب نمود. این ویژگی‌ها را، که تعداد آنها در این مقاله ۲۷ عدد در نظر گرفته شده است، تحت عنوان «سنجه‌های قاعده» نامگذاری نموده‌ایم. به عبارت دیگر، هر حالت با مجموعه مشخص و متفاوتی از مقادیر برای این ۲۷ سنجه بازنمایی می‌شود. این سنجه‌های قاعده را می‌توان در جدول ۱ مشاهده نمود. مقادیر برخی از این سنجه‌های قاعده از حل‌کننده Z3 دریافت می‌شوند (با دادن فرمول به آن و دریافت مقدار سنجه به عنوان خروجی Z3) و مقادیر برخی نیز مستقل از Z3 تعیین می‌شوند. از ۲۷ سنجه قاعده در نظر گرفته شده، ۲۳ سنجه مقدار عددی و ۴ سنجه مقدار بولی دارند. مقادیر ۲۳ سنجه عددی و نیز مقدار یکی از سنجه‌های بولی، با انجام عمل‌ها توسط عامل می‌تواند تغییر می‌کند، که به دنبال آن، عامل وارد یک حالت جدید می‌شود. اما مقادیر ۳ عدد از سنجه‌های قاعده بولی تنها به منظور ایجاد تمایز بین حالت‌ها در منطق‌های مختلف SMT، به حالت‌های محیط اضافه شده‌اند و لذا با انجام عمل‌های مختلف توسط عامل تغییر نمی‌کنند؛ مقادیر آنها تنها براساس فرمول اولیه سازنده محیط تعیین می‌شود.

بدیهی است که انتخاب سنجه‌های قاعده، تأثیر مستقیم و شدیدی روی نتیجه کار خواهد داشت و لذا انتخاب آنها باید با دقت انجام شود. همچنین، سنجه‌های قاعده باید به شکلی انتخاب شوند که به فرمول‌ها مرتبط باشند. به عنوان مثال، سنجه قاعده depth، که تعداد تاکتیک‌های اعمال شده روی فرمول را ارائه می‌دهد، ارتباطی با خود فرمول ندارد و لذا سنجه مناسبی محسوب نمی‌شود.

جدول ۱ لیست سنجه‌های قاعده استفاده شده برای ایجاد حالت‌های محیط

	سنجه قاعده	توضیحات
۱	size	تعداد ادعاها[۵۶] در فرمول داده شده
۲	num-exprs	تعداد عبارات[۵۷]/اصطلاحات[۵۸] در فرمول داده شده
۳	num-consts	تعداد ثابت‌های غیر بولی در فرمول داده شده
۴	num-bool-consts	تعداد ثابت‌های بولی در فرمول داده شده
۵	num-arith-consts	تعداد ثابت‌های حسابی[۵۹] در فرمول داده شده
۶	num-bv-consts	تعداد ثابت‌های بردار بیتی[۶۰] در فرمول داده شده
۷	+	تعداد عملگرهای جمع در فرمول داده شده
۸	-	تعداد عملگرهای تفریق در فرمول داده شده
۹	/	تعداد عملگرهای تقسیم در فرمول داده شده
۱۰	*	تعداد عملگرهای ضرب در فرمول داده شده
۱۱	==	تعداد عملگرهای برابری در فرمول داده شده
۱۲	!=	تعداد عملگرهای مخالف در فرمول داده شده
۱۳	>=	تعداد عملگرهای بزرگتر مساوی در فرمول داده شده
۱۴	>	تعداد عملگرهای بزرگتر در فرمول داده شده
۱۵	<=	تعداد عملگرهای کوچکتر مساوی در فرمول داده شده
۱۶	<	تعداد عملگرهای کوچکتر در فرمول داده شده

[56] Assertions
[57] Expressions
[58] Terms
[59] Arithmetic
[60] Bit-Vector

۱۷	and	تعداد عملگرهای and در فرمول داده شده
۱۸	or	تعداد عملگرهای or در فرمول داده شده
۱۹	bvand	تعداد عملگرهای bvand در فرمول داده شده
۲۰	bvor	تعداد عملگرهای bvor در فرمول داده شده
۲۱	bvxor	تعداد عملگرهای bvxor در فرمول داده شده
۲۲	bvashr	تعداد عملگرهای bvashr در فرمول داده شده
۲۳	bvshl	تعداد عملگرهای bvshl در فرمول داده شده
۲۴	is-unbounded	تعیین می‌کند که آیا فرمول داده شده دارای ثابت‌های صحیح/واقعی که کران بالا/ پایین ندارند می‌باشد یا خیر
۲۵	is-qfbv	تعیین می‌کند که آیا فرمول داده شده در منطق QF_BV است یا خیر
۲۶	is-qfnia	تعیین می‌کند که آیا فرمول داده شده در منطق QF_NIA است یا خیر
۲۷	is-qfnra	تعیین می‌کند که آیا فرمول داده شده در منطق QF_NRA است یا خیر

محیط باید دارای یک حالت شروع باشد تا عامل کار خود را از آن حالت شروع کند. حالت شروع حالتی است که در آن مقادیر سنجه‌های قاعده بدون استفاده از تاکتیک و با استفاده از فرمول اولیه ورودی تعیین می‌شوند. حالت پایانی مشخصی در این مقاله در نظر گرفته نشده است. اما در یکی از دو حالت زیر، دوره[۶۱] را خاتمه یافته تلقی می‌نماییم: (۱) فرمولی که به آن رسیده‌ایم به اندازه‌ای ساده باشد که به طور صریح قابل حل باشد. منظور از فرمولی که به طور صریح قابل حل باشد، فرمولی است که به اندازه‌ای ساده شده باشد که دیگر هیچ فرمول هدف فرعی نداشته باشد. (۲) عامل از حداکثر تعداد عمل‌های قابل انتخاب در یک دوره عبور کرده باشد.

۵-۲. عمل‌ها

هر عمل از یک تاکتیک و پارامترهای آن (در صورت وجود) تشکیل شده است. انجام هر یک از عمل‌ها توسط عامل در محیط به معنی اِعمال تاکتیک متناظر روی فرمول است. این کار باعث تغییر شکل فرمول شده و به دنبال آن، مقادیر سنجه‌های قاعده نیز تغییر می‌کنند و لذا، عامل به یک حالت جدید در محیط منتقل می‌شود. لیست عمل‌ها و تاکتیک و پارامتر سازنده هر عمل در جدول ۲ نمایش داده شده‌اند. در کل ۱۴ تاکتیک و ۹ پارامتر در این مقاله در نظر گرفته شده است. نوع همه پارامترهای در نظر گرفته شده بولی می‌باشد. با لحاظ کردن پارامترهای مختلف برای هر تاکتیک، در مجموع عامل دارای ۲۳ عمل خواهد بود. برای مثال برای تاکتیک aig دو عمل وجود دارد. عمل اول، تاکتیک aig بدون پارامتر است و عمل دوم، تاکتیک aig با پارامتر aig_per_assertion است که مقدار پارامتر true می‌باشد. تاکتیک aig با پارامتر aig_per_assertion و مقدار پارامتر false به عنوان عمل مجزا در نظر گرفته نمی‌شود. دلیل این موضوع این است که مقدار پیش‌فرض پارامترهای بولی، false است و تاکتیک aig با پارامتر aig_per_assertion و مقدار پارامتر false معادل همان تاکتیک aig بدون پارامتر است.

جدول ۲ لیست عمل‌های ممکن در محیط و تاکتیک‌ها و پارامترهای متناظر با هر عمل[۶۲]

	عمل	تاکتیک	پارامترها

[61] Episode

[۶۲] **برای اطلاعات بیشتر در مورد تاکتیک‌ها می‌توانید به صفحه راهنمای پروژه Z3 به آدرس** https://microsoft.github.io/z3guide، **بخش تاکتیک‌ها مراجعه نمایید.**

۱	simplify	simplify	-
۲	simplify(elim_and=true)		elim_and
۳	simplify(blast_distinct=true)		blast_distinct
۴	simplify(som=true)		som
۵	simplify(pull_cheap_ite=true)		pull_cheap_ite
۶	simplify(hoist_mul=true)		hoist_mul
۷	simplify(local_ctx=true)		local_ctx
۸	simplify(flat=true)		flat
۹	smt	smt	-
۱۰	bit-blast	bit-blast	-
۱۱	bv1-blast	bv1-blast	-
۱۲	solve-eqs	solve-eqs	-
۱۳	aig(aig_per_assertion=true)	aig	aig_per_assertion
۱۴	aig		-
۱۵	qfnra-nlsat	qfnra-nlsat	-
۱۶	sat	sat	-
۱۷	max-bv-sharing	max-bv-sharing	-
۱۸	reduce-bv-size	reduce-bv-size	-
۱۹	purify-arith	purify-arith	-
۲۰	propagate-values(push_ite_bv=true)	propagate-values	push_ite_bv
۲۱	propagate-values		-
۲۲	elim-uncnstr	elim-uncnstr	-
۲۳	ackermannize_bv	ackermannize_bv	-

۵-۳. سیگنال پاداش

به طور کلی، هدف یادگیری تاکتیک‌ها و پارامترهای موثر در هر حالت از محیط است. به عبارت دیگر، هدف تشخیص تاکتیک‌ها و پارامترها به ازای حالت‌های مختلف محیط است که سرعت حل فرمول توسط حل‌کننده را کمینه می‌کنند. بنابراین، سیگنال پاداش باید به گونه‌ای تعیین شود که با افزایش سرعت حل فرمول توسط حل‌کننده، رابطه مستقیم داشته باشد. در جدول ۳، پاداش‌ها برای هر عمل مشخص شده‌اند. عامل به طور پیش‌فرض، پاداش ۲- برای هر عملی که در محیط انجام می‌دهد دریافت می‌کند. این پاداش به عنوان یک جریمه برای عامل عمل می‌کند و عامل را مجاب می‌کند تا کمترین تعداد عمل را در محیط انجام دهد. اگر عامل عملی انجام دهد که منجر به کاهش rlimit شود، جریمه ۲- به ۱- کاهش می‌یابد. همچنین، اگر عامل عملی انجام دهد که باعث شود تا فرمول به طور صریح قابل حل شود، بیشترین پاداش، یعنی مقدار ۵ را دریافت می‌کند. پس از انجام عملی که منجر به فرمولی با قابلیت حل شدن صریح می‌شود، دوره خاتمه می‌یابد که معادل دریافت پاداش ۵ می‌باشد. اما اگر عامل عملی انجام دهد که باعث بروز خطا در حل‌کننده شود، جریمه ۵- دریافت می‌کند. با این جریمه، عامل به مرور دیگر چنین عمل‌هایی را در محیط انجام نمی‌دهد.

دریافت پاداش ۱- به جای پاداش ۲- در ازای انجام عمل‌هایی که باعث کاهش rlimit می‌شوند، باعث می‌شود تا این عمل‌ها نسبت به عمل‌هایی که تاثیری روی فرمول ندارند اولویت بیشتری برای عامل داشته باشند، چرا که عامل با انجام این عمل‌ها جریمه کمتری دریافت می‌کند. همچنین مقدار پاداش ۱- برای عمل‌هایی که باعث کاهش rlimit می‌شوند، باعث

می‌شود عامل تلاش کند تا با کمترین تعداد عمل به بیشترین پاداش دست پیدا کند. تخصیص مقدار پاداش ۰ و یا مثبت (مثلا پاداش ۱+) به چنین عمل‌هایی می‌تواند عامل را با خطا مواجه کند. در چنین حالتی ممکن است عامل عمل‌های اضافی و ناکارآمد انجام دهد و سراغ عمل‌هایی که منجر به فرمولی با قابلیت حل شدن صریح می‌شوند و بیشترین پاداش را به عامل می‌دهند نرود و یا آنها را اولویت‌های بعدی خود قرار دهد. دلیل این امر این است که عامل تلاش می‌کند تا پاداشی را که در طولانی مدت دریافت می‌کند بیشینه کند.

نکته‌ای که در اینجا باید بدان توجه کرد این است که اِعمال تاکتیک‌ها در Z3، خود باعث افزایش مقدار rlimit می‌شود. اما این افزایش نباید در rlimit حل فرمول لحاظ شود. به همین دلیل، محاسبه مقدار rlimit باید طبق روال زیر انجام شود:

(۱) اِعمال تاکتیک روی فرمول مورد نظر
(۲) اندازه‌گیری مقدار اولیه rlimit
(۳) حل فرمول بعد از اِعمال تاکتیک
(۴) اندازه‌گیری مقدار ثانویه rlimit
(۵) محاسبه اختلاف rlimit اندازه‌گیری شده در گام ۲ و گام ۴ و در نظر گرفتن این مقدار به عنوان مقدار اصلی rlimit

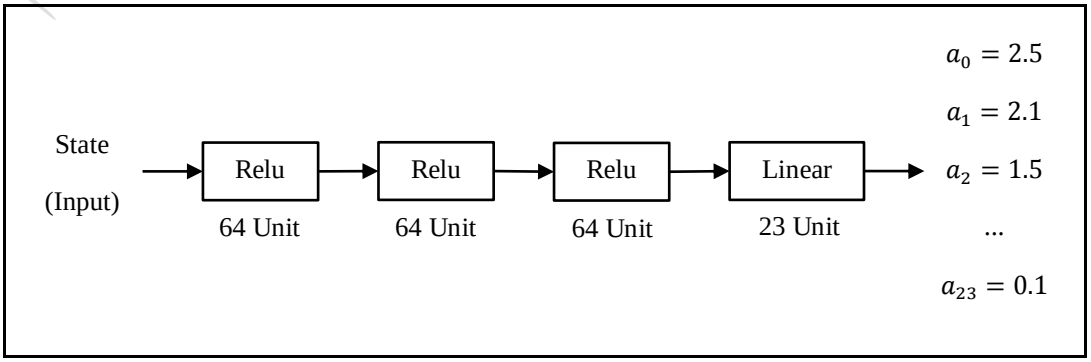
جدول ۳ پاداش‌های محیط

مقدار پاداش	توضیحات
۵	پاداش ساده‌سازی فرمول تا حدی که به طور صریح قابل حل باشد
-۵	جریمه انجام عملی که مناسب فرمول نباشد و باعث بروز خطا شود
-۱	پاداش عملی که بتواند rlimit را کاهش دهد
-۲	پاداش پیش‌فرض هر گام زمانی

۵-۴. عامل

در این مقاله از عامل DQN [26] استفاده شده است که در ادامه همین بخش معرفی می‌شود. دلیل انتخاب عامل DQN آن است که عموما در محیط‌های با تعداد حالت‌ها و عمل‌های زیاد عملکرد مناسبی دارد.

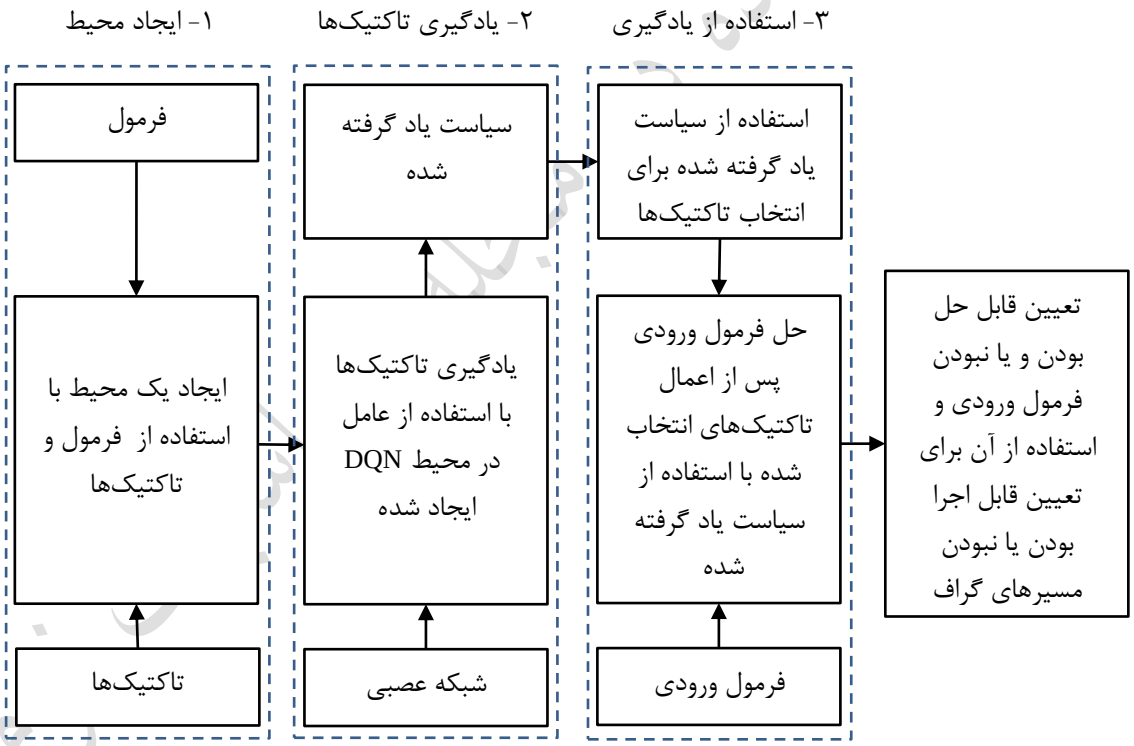
عامل از یک شبکه عصبی که متشکل از سه لایه Dense (Relu) با ۶۴ واحد در هر لایه، و یک لایه Dense (Linear) که تعداد واحدهای آن برابر با تعداد عمل‌های محیط است، بهره‌مند شده است. خروجی هر لایه ورودی لایه بعدی است. همچنین بهینه‌سازی که در شبکه عصبی عامل از آن استفاده شده است بهینه‌ساز Adam است. ورودی شبکه عصبی در حقیقت حالت محیط است و خروجی آن عملی است، که عامل باید طبق تجویز شبکه، در حالت فعلی انجام دهد. در واقع خروجی شبکه احتمال انتخاب هر عمل در حالت ورودی شبکه عصبی را تعیین می‌کند و عمل با احتمال بیشتر به معنای بهتر بودن آن عمل برای انجام در آن حالت است. شکل ۴ این شبکه عصبی را نشان می‌دهد.



شکل ۴ شبکه عصبی استفاده شده در عامل یادگیری تقویتی عمیق

۵-۵. روش پیشنهادی در یک نگاه کلی

شکل ۵ روند کلی روش پیشنهادی را نشان می‌دهد. همان گونه که در بخش قبل بیان شد، در این روش از الگوریتم DQN استفاده شده است. به منظور آموزش عامل DQN ابتدا یک فرمول ورودی در نظر گرفته می‌شود. با توجه به سنجه‌های قاعده آن فرمول، وضعیت محیط مشخص شده و به عنوان ورودی به شبکه عصبی داده می‌شود. عامل در هر گام زمانی یک عمل را (براساس مقدار اکتشاف و بهره‌برداری تعیین شده) انتخاب می‌کند و آن عمل را در محیط انجام می‌دهد. بدین ترتیب، فرمول با توجه به تاکتیک انتخابی بروزرسانی می‌شود. پاداش عملی که عامل انجام داده و مقدار تعیین‌کننده پایان دوره به عامل داده می‌شود. با توجه به اعمال تاکتیک انتخابی، بروز شدن فرمول و تغییر مقادیر سنجه‌های قاعده آن، عامل وارد حالت جدیدی از محیط می‌شود. چرخه فوق مجددا و تا زمان پایان یافتن دوره تکرار می‌شود. در صورتی که یک دوره به پایان برسد، محیط بازنشانی[۶۳] می‌شود و عامل مجددا یک دوره جدید یادگیری را شروع می‌کند. عامل این کار را برای چندین دوره پیاپی تکرار می‌کند تا زمانی که یک سیاست بهینه را یاد بگیرد. بعد از اینکه عامل به یک سیاست بهینه دست یافت، می‌توان از این سیاست برای انتخاب تاکتیک‌ها برای فرمول‌های مختلف استفاده کرد.



شکل ۵ روند کلی روش پیشنهادی

۵-۶. مثالی از روش پیشنهادی

در این بخش روش پیشنهادی با ذکر مثالی شرح داده می‌شود. دستورات SMT شکل ۶ را در نظر بگیرید. این دستورات به صورت یک فایل با پسوند smt2 قابل ذخیره‌سازی و نگهداری است. حل‌کننده Z3 برای حل این فرمول به صورت پیش‌فرض باید ۵۱۳ عدد عملیات پایه انجام دهد که این عدد در واقع همان مقدار rlimit می‌باشد. تاکتیک‌های smt و qfnra-nlsat

[63] Reset

بهترین تاثیر را روی این فرمول دارند و اعمال هر کدام از آنها روی این فرمول، مقدار rlimit را به ۱۷ کاهش می‌دهد. بعد از اِعمال هر کدام از این تاکتیک‌ها، دستورات SMT به صورت تنها یک مقدار صحیح ساده خواهند شد که به صورت صریح قابل حل می‌باشد.

در ابتدا با استفاده از فایل حاوی دستورات SMT ارائه شده در شکل ۶ و نیز تاکتیک‌های موجود در جدول ۲، حالت ابتدایی محیط مشخص می‌شود. جدول ۴ تمامی عمل‌های ممکن و همچنین مقادیر rlimit و پاداش متناظر با هر عمل را که براساس مقدار rlimit محاسبه می‌شود نشان می‌دهد.

```
(set-info :smt-lib-version 2.6)
(set-logic QF_NIA)
(set-info :source |AProve team, see http://aprove.informatik.rwth-aachen.de/, submitted for SMT-COMP 2014|)
(set-info :category "industrial")
(set-info :status sat)
(declare-fun a__4 () Int)
(declare-fun a__2 () Int)
(declare-fun a__3 () Int)
(declare-fun a__5 () Int)
(assert (and (>= (+ (* a__4 a__2) (* (- 0 1) (* a__4 a__3)) (* (- 0 1) (* a__4 a__5 a__3)) (- 0 1)) 0) (>= (+ (* a__5 a__2) (* (- 0 1) a__2)) 0) (>= (+ a__3 (* (- 0 1) (* a__5 a__5 a__3))) 0) (>= a__4 0) (>= a__2 0) (>= a__3 0) (>= a__5 0)))
(check-sat)
(exit)
```

شکل ۶ نمونه‌ای از دستورات صدق‌پذیری پیمانه نظریه‌ها برای شرح مثال از نحوه عملکرد روش پیشنهادی

در طول یادگیری، هر زمانی که عامل تاکتیک‌های smt و qfnra-nlsat را به کار گیرد پاداش ۵+ را که بیشترین میزان پاداش است دریافت خواهد کرد، چرا که این تاکتیک‌ها باعث می‌شوند فرمول به طور صریح قابل حل شود.

جدول ۴ عمل‌های ممکن در حالت ایجاد شده با فرمول شکل ۶ و پاداش‌های متناظر با هر عمل

	عمل	مقدار rlimit بعد از انجام عمل در محیط	آیا این عمل فرمول ورودی را به طور صریح قابل حل می‌کند؟	آیا این عمل مقدار rlimit را کاهش می‌دهد؟	پاداش عمل
۱	ackermannize_bv	۵۱۳	خیر	خیر	-۲
۲	purify-arith	۴۶۷	خیر	بلی	-۱
۳	reduce-bv-size	۴۷۳	خیر	بلی	-۱
۴	solve-eqs	۴۶۷	خیر	بلی	-۱

۵	propagate-values	۴۷۳	خیر	بلی	-۱
۶	simplify	۴۶۷	خیر	بلی	-۱
۷	sat	۴۷۳	خیر	بلی	-۱
۸	qfnra-nlsat	۱۷	بلی	بلی	+۵
۹	simplify(hoist_mul=True)	۵۰۹	خیر	بلی	-۱
۱۰	bv1-blast	۴۷۳	خیر	بلی	-۱
۱۱	simplify(pull_cheap_ite=True)	۴۶۷	خیر	بلی	-۱
۱۲	simplify(som=True)	۴۶۷	خیر	بلی	-۱
۱۳	aig(aig_per_assertion=True)	۴۶۷	خیر	بلی	-۱
۱۴	smt	۱۷	بلی	بلی	+۵
۱۵	simplify(flat=True)	۴۶۷	خیر	بلی	-۱
۱۶	propagate-values(push_ite_bv=True)	۴۷۳	خیر	بلی	-۱
۱۷	simplify(elim_and=True)	۴۶۷	خیر	بلی	-۱
۱۸	simplify(local_ctx=True)	۴۶۷	خیر	بلی	-۱
۱۹	simplify(blast_distinct=True)	۴۶۷	خیر	بلی	-۱
۲۰	aig	۴۷۳	خیر	بلی	-۱
۲۱	elim-uncnstr	۴۷۳	خیر	بلی	-۱
۲۲	max-bv-sharing	۴۷۳	خیر	بلی	-۱
۲۳	bit-blast	۴۷۳	خیر	بلی	-۱

جدول ۵ نمونه‌ای از حالت‌های محیط را، که در یک دوره از عملکرد، عامل در آنها قرار گرفته است نمـایش مـی‌دهـد. هر حالت از لیستی از مقادیر سنجه‌های قاعده تشکیل شده است. هر سطر جدول ۵ یک حالت را نشان می‌دهد. حالت شـماره ۱ در جدول ۵ حالت شروع است. در این دوره عامل نتوانسته هیچ یک از عمل‌های شماره ۸ و ۱۴ جدول ۴ را انجام دهـد و دوره را به پایان برساند و دلیل خاتمه یافتن این دوره به حداکثر رسیدن تعداد عمل‌های انجام شده در محیط است. در این مثال، عامل در این دوره تنها سه حالت متمایز را مشاهده کرده است، اما حالت‌ها ممکن است به صورت تکراری توسط عامل مشاهده شوند.

جدول ۵ نمونه‌ای از حالت‌هایی که عامل در یک دوره در آنها قرار گرفته است

شماره حالت	حالت (سنجه‌ها)										
	Other probes	is-qfnia	is-unbounded	>=	*	-	+	num-arith-consts	num-consts	num-exprs	size
۱	۰	۱	۱	۴	۱	۱	۱	۲	۲	۱۱	۴

۲	۰	۱	۱	۴	۲	۱	۱	۲	۲	۴
۳	۰	۱	۱	۴	۲	۲	۲	۲	۲	۴

جدول ۶ نمونه‌ای دیگر از حالت‌های محیط که در یک دوره دیگر عامل در آن حالت‌ها قرار گرفته است را نمایش می‌دهد. هر سطر جدول ۶ نیز یک حالت را نشان می‌دهد. حالت شماره ۱ در جدول ۶، حالت شروع است که معادل همان حالت ۱ در جدول ۵ است. حالت شماره ۲ در جدول ۶ حالت پایان است. در این دوره عامل موفق شده یکی از عمل‌های شماره ۸ و ۱۴ جدول ۴ را در محیط انجام دهد و دوره را به پایان برساند. عامل در این دوره تنها چهار حالت متمایز را مشاهده کرده است.

جدول ۶ نمونه‌ای دیگر از حالت‌هایی که عامل در یک دوره در آنها قرار گرفته است

شماره حالت	حالت (سنجه‌ها)										
	Other probes	is-qfnia	Is-qfbv	is-unbounded	>=	*	-	+	num-arith-consta	num-consts	num-exprs	size
۱	۰	۱	۰	۱	۴	۱	۱	۱	۲	۲	۱۱	۴
۲	۰	۰	۱	۰	۰	۰	۰	۰	۰	۰	۰	۰
۳	۰	۱	۰	۱	۴	۲	۱	۱	۲	۲	۱۲	۴
۴	۰	۱	۰	۱	۴	۲	۲	۲	۲	۲	۱۳	۴

۶. آزمایش‌ها

به منظور بررسی روش پیشنهادی، منطق QF_NIA با مجموعه داده‌های AProVE[64] و منطق QF_NRA با مجموعه داده‌های hycomp[65] مورد استفاده قرار گرفته است. همانطور که در بخش ۵-۴ گفته شد، در روش پیشنهادی از عامل DQN استفاده شده است. پارامترهای این عامل در جدول ۷ نشان‌داده شده‌اند. نکته‌ای که در مورد پارامتر memory size باید به آن توجه کرد این است که هر خانه حافظه باید حالت فعلی، حالت بعدی، عمل، میزان پاداش دریافت شده و نیز مقدار تعیین‌کننده پایان دوره را در خود نگه دارد. به جهت مشخص شدن کیفیت رفتار عامل DQN، یک عامل تصادفی[66] نیز در نظر گرفته شده که در هر مرحله، یک اقدام را به صورت کاملا تصادفی انتخاب و در محیط اِعمال می‌کند. در آزمایش‌ها، نتایج حاصل از رفتار عامل DQN با نتایج حاصل از رفتار عامل تصادفی مورد مقایسه قرار گرفته است.

فاز آموزش در هر منطق به صورت جداگانه و روی ۵۰ فرمول انجام شده است. پس از آموزش، به ازای هر منطق به صورت جداگانه و روی ۱۰۰ فرمول آزمایش صورت پذیرفته است. فرمول‌های مربوط به آموزش و آزمایش از مقاله [9] انتخاب شده است.

جدول ۷ پارامترهای عامل یادگیری تقویتی عمیق

[64] https://clc-gitlab.cs.uiowa.edu:2443/SMT-LIB-benchmarks/QF_NIA/tree/master/AProVE
[65] https://clc-gitlab.cs.uiowa.edu:2443/SMT-LIB-benchmarks/QF_NRA/tree/master/hycomp
[66] Pure Chance

پارامتر	مقدار	توضیحات
discount factor (gamma)	۰/۹	میزان اهمیت پاداش‌های جدید نسبت به پاداش‌های قدیمی‌تر برای عامل را تعیین می‌کند
learning rate	۰/۰۰۱	نرخ یادگیری بهینه‌ساز Adam را تعیین می‌کند
epsilon	۱/۰	مبادله بین اکتشاف و بهره‌برداری را تعیین می‌کند که در ابتدا بیشترین مقدار را دارد (اولویت با اکتشاف است)
epsilon minimum	۰/۱	حداقل مقدار ممکن برای epsilon را تعیین می‌کند
epsilon decay	۰/۹۹۹	میزان کاهش epsilon را در فاصله مشخص شده توسط پارامتر target update interval تعیین می‌کند
batch size	۶۴	اندازه دسته‌های نمونه‌برداری از حافظه عامل را تعیین می‌کند
memory size	۵۰۰۰۰۰	اندازه حافظه عامل را تعیین می‌کند
target update interval	۱۰	فاصله بین به‌روزرسانی شبکه عصبی و مقدار epsilon (برحسب گام زمانی) را تعیین می‌کند
training interval	۱۰	فاصله بین آموزش شبکه عصبی (برحسب گام زمانی) را تعیین می‌کند
learning start	۱۰۰۰	گام زمانی شروع یادگیری را تعیین می‌کند

با توجه به مقدار rlimit و زمان حل فرمول‌های ورودی فاز آزمایش (بدون اعمال هیچ‌گونه تاکتیکی)، فرمول‌ها به سه دسته آسان[67]، معمولی[68] و سخت[69] تقسیم بندی شدند. در جدول ۸ مقادیر مرزی rlimit برای هر منطق نشان داده شده است. توجه نمایید که در دسته سخت منطق QF_NIA جدول ۸، مقدار حداکثر با مهلت زمانی ۵ دقیقه‌ای بدست آمده است.

جدول ۸ مقادیر مرزی rlimit برای دسته‌های فرمول‌های آزمایش

منطق	آسان		معمولی		سخت	
QF_NRA (hycomp)	حداقل	حداکثر	حداقل	حداکثر	حداقل	حداکثر
	۸۵	۸۲۸	۸۱۸	۱۰۶۷	۱۰۷۱	۱۸۱۶
QF_NIA (AProVE)	حداقل	حداکثر	حداقل	حداکثر	حداقل	حداکثر
	۲۳۰	۳۰۱۰۳	۵۱۰۶	۵۹۳۲۶۵۷	۱۸۱۳۶۳۶	۱۴۵۲۶۴۴۴۳

در فاز آموزش، یک مهلت زمانی ۵ ثانیه‌ای برای حل‌کننده در نظر گرفته شده است. در فاز آزمایش نیز مهلت زمانی ۵ دقیقه‌ای برای حل‌کننده در نظر گرفته شده است. این مهلت‌های زمانی براساس زمان حل فرمول‌ها بدون اِعمال تاکتیک، در نظر گرفته شده‌اند. در صورتی که حل‌کننده نتواند در مهلت زمانی تعیین شده فرمول ورودی را حل کند، مقدار rlimit بدست آمده در آن مهلت زمانی در نظر گرفته می‌شود. هر دوره به انجام ۲۳ عمل توسط عامل محدود شده و پس از آن دوره خاتمه می‌یابد. در طی آموزش، هر فرمول ۵۰۰۰ بار به عامل داده شده و در هر بار، عامل یک دوره کامل روی آن اقدام نموده و سعی در ساده‌سازی آن نموده است.

معیارهای مورد استفاده برای ارزیابی نتایج به شرح زیر هستند: (۱) میزان افزایش سرعت حل‌کننده در حل فرمول و (۲) میزان کاهش rlimit.

[67] Easy
[68] Normal
[69] Hard

تمامی آزمایش‌ها روی یک سیستم Core i7 با ۶۴ گیگابات حافظه RAM انجام شده است. هر نتیجه ذکر شده، میانگین ۱۰ بار اجراست. در ادامه نتایج آزمایش‌ها انجام پذیرفته به صورت جداگانه برای هر منطق آورده شده است.

۶-۱. نتایج آزمایش‌ها در منطق QF_NRA

همانطور که در جدول ۹ می‌توان مشاهده کرد، در منطق QF_NRA با استفاده از یادگیری انجام شده، سرعت حل حل‌کننده را می‌توان تا ۲/۱۳۶ برابر افزایش داد. همچنین می‌توان مشاهده کرد که مقدار rlimit تا ۴۵۱ برابر کاهش می‌یابد. با مقایسه سرعت و rlimit اندازه‌گیری شده برای دو عامل آموزش دیده و عامل تصادفی، می‌توان نتیجه گرفت که عملکرد عامل آموزش دیده بسیار بهتر از عامل تصادفی بوده است. با بررسی جدول ۹ متوجه می‌شویم که با پیچیده‌تر و سخت‌تر شدن فرمول‌ها، عامل آموزش دیده با انتخاب عمل‌های مناسب، عملکرد بهتری از خود نشان داده است.

جدول ۹ نتایج آزمایش‌ها در منطق QF_NRA

دسته	با عامل آموزش دیده (DQN)				با عامل تصادفی				تعداد فایل در هر دسته
	کمترین میزان افزایش سرعت	بیشترین میزان افزایش سرعت	کمترین مقدار کاسته شده از rlimit	بیشترین مقدار کاسته شده از rlimit	کمترین میزان افزایش سرعت	بیشترین میزان افزایش سرعت	کمترین مقدار کاسته شده از rlimit	بیشترین مقدار کاسته شده از rlimit	
آسان	۱/۶۹۵ برابر	۲/۱۳۶ برابر	۲۱ برابر	۲۰۳ برابر	۱/۱۰۴ برابر	۱/۹۰۱ برابر	۱/۱۸۶ برابر	۲/۴۰۳ برابر	۳۳
معمولی	۱/۷۴۳ برابر	۲/۰۷ برابر	۲۰۱ برابر	۲۶۶ برابر	۱/۰۴۵ برابر	۱/۶۴۲ برابر	۱/۱۸۵ برابر	۲/۵۷۹ برابر	۳۴
سخت	۱/۸۸۳ برابر	۲/۰۶۷ برابر	۲۶۳ برابر	۴۵۱ برابر	۱/۰۹۷ برابر	۱/۷۹۵ برابر	۱/۱۸۶ برابر	۲/۴۰۳ برابر	۳۳

در جدول ۱۰ نیز می‌توان میزان افزایش سرعت به ازای هر فرمول را به تفکیک مشاهده کرد.

جدول ۱۰ میزان افزایش سرعت در منطق QF_NRA به تفکیک فرمول

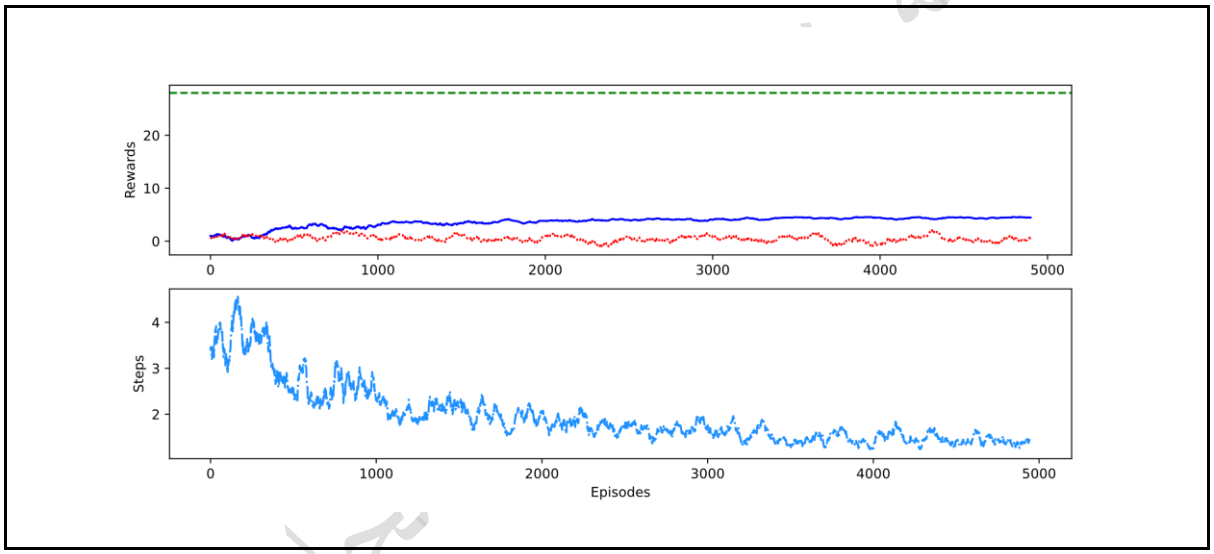
ردیف	نام فایل (فرمول)	افزایش سرعت حل	ردیف	نام فایل (فرمول)	افزایش سرعت حل
۱	ball_count_1d_plain.01.qfree_global_0.smt2	۲/۰۱۷	۵۱	ball_count_2d_plain.02.qfree_global_0.smt2	۱/۸۹۳

			برابر		برابر
۲	ball_count_1d_plain.02.qfree_global_0.smt2	۵۲	۱/۹۸۰ برابر	ball_count_1d_plain.01.redlog_global_0.smt2	۱/۸۶۱ برابر
۳	ball_count_1d_plain.03.qfree_global_0.smt2	۵۳	۱/۸۷۳ برابر	ball_count_2d_plain.03.qfree_global_0.smt2	۱/۹۵۳ برابر
۴	ball_count_2d_hill_simple.01.qfree_global_0.smt2	۵۴	۱/۸۶۵ برابر	ball_count_1d_plain.02.seq_lazy_lemmas_global_0.smt2	۱/۹۸۸ برابر
۵	ball_count_2d_hill_simple.02.qfree_global_0.smt2	۵۵	۲/۱۳۶ برابر	ball_count_1d_plain.04.redlog_global_0.smt2	۱/۹۹۳ برابر
۶	ball_count_1d_plain.01.seq_lazy_global_0.smt2	۵۶	۱/۸۷۹ برابر	ball_count_1d_plain.03.seq_lazy_linear_enc_lemmas_global_0.smt2	۲/۰۶۰ برابر
۷	ball_count_1d_plain.02.seq_lazy_global_0.smt2	۵۷	۱/۷۶۷ برابر	ball_count_2d_hill_simple.01.redlog_global_0.smt2	۱/۹۶۱ برابر
۸	ball_count_1d_plain.01.seq_lazy_linear_enc_global_0.smt2	۵۸	۱/۷۹۳ برابر	ball_count_2d_hill_simple.02.redlog_global_0.smt2	۲ برابر
۹	ball_count_1d_plain.02.seq_lazy_linear_enc_global_0.smt2	۵۹	۱/۸۲۸ برابر	ball_count_1d_plain.04.seq_lazy_linear_enc_lemmas_global_0.smt2	۱/۹۶۸ برابر
۱۰	ball_count_2d_hill_simple.10.qfree_global_0.smt2	۶۰	۱/۹۷۱ برابر	ball_count_2d_hill_simple.03.redlog_global_0.smt2	۲/۰۰۹ برابر
۱۱	ball_count_1d_plain.10.seq_lazy_global_0.smt2	۶۱	۱/۶۹۵ برابر	ball_count_2d_hill.01.seq_lazy_linear_enc_global_0.smt2	۱/۹۸۷ برابر
۱۲	ball_count_1d_plain.10.seq_lazy_linear_enc_global_0.smt2	۶۲	۱/۸۰۴ برابر	ball_count_2d_hill.02.seq_lazy_linear_enc_global_0.smt2	۱/۹۹۴ برابر
۱۳	ball_count_2d_plain.02.seq_lazy_linear_enc_global_0.smt2	۶۳	۱/۷۸۶ برابر	ball_count_2d_hill_simple.05.redlog_global_0.smt2	۱/۹۷۰ برابر
۱۴	ball_count_1d_plain.04.seq_lazy_global_0.smt2	۶۴	۱/۸۱۰ برابر	ball_count_2d_hill.04.seq_lazy_linear_enc_global_0.smt2	۲/۰۰۱ برابر
۱۵	ball_count_2d_hill_simple.10.seq_lazy_linear_enc_global_0.smt2	۶۵	۱/۹۰۱ برابر	ball_count_2d_hill.01.seq_lazy_global_0.smt2	۲/۰۳۵ برابر
۱۶	ball_count_2d_plain.01.seq_lazy_global_0.smt2	۶۶	۱/۹۲۶ برابر	ball_count_2d_hill.03.seq_lazy_global_0.smt2	۱/۹۸۹ برابر
۱۷	ball_count_2d_plain.02.seq_lazy_global_0.smt2	۶۷	۱/۸۷۵ برابر	ball_count_2d_hill.05.seq_lazy_linear_enc_global_0.smt2	۱/۹۹۴ برابر
۱۸	ball_count_2d_hill_simple.10.seq_lazy_global_0.smt2	۶۸	۱/۹۱۸ برابر	ball_count_1d_plain.01.seq_lazy_lemmas_global_0.smt2	۲/۰۱۲ برابر

۱/۸۸۳ برابر	ball_count_2d_plain.04.qfree_global_0.smt2	۶۹	۱/۷۹۷ برابر	ball_count_2d_plain.10.seq_lazy_global_0.smt2	۱۹
۱/۹۸۹ برابر	ball_count_1d_plain.03.seq_lazy_lemmas_global_0.smt2	۷۰	۱/۹۸۶ برابر	ball_count_1d_plain.02.redlog_global_0.smt2	۲۰
۲/۰۵۶ برابر	ball_count_2d_plain.05.qfree_global_0.smt2	۷۱	۱/۹۲۶ برابر	ball_count_1d_plain.10.redlog_global_0.smt2	۲۱
۱/۹۷۱ برابر	ball_count_1d_plain.04.seq_lazy_lemmas_global_0.smt2	۷۲	۱/۹۸۶ برابر	ball_count_1d_plain.02.seq_lazy_linear_enc_lemmas_global_0.smt2	۲۲
۱/۹۴۸ برابر	ball_count_1d_plain.05.seq_lazy_lemmas_global_0.smt2	۷۳	۱/۹۰۸ برابر	ball_count_1d_plain.10.seq_lazy_linear_enc_lemmas_global_0.smt2	۲۳
۱/۹۷۳ برابر	ball_count_2d_hill_simple.01.seq_lazy_linear_enc_lemmas_global_0.smt2	۷۴	۱/۹۷۰ برابر	ball_count_1d_plain.10.seq_lazy_lemmas_global_0.smt2	۲۴
۱/۹۴۲ برابر	ball_count_2d_plain.01.seq_lazy_linear_enc_lemmas_global_0.smt2	۷۵	۱/۹۶۰ برابر	ball_count_2d_hill_simple.10.redlog_global_0.smt2	۲۵
۲/۰۱۶ برابر	ball_count_2d_hill_simple.03.seq_lazy_linear_enc_lemmas_global_0.smt2	۷۶	۲/۰۰۱ برابر	ball_count_2d_hill.10.seq_lazy_linear_enc_global_0.smt2	۲۶
۲/۰۴۵ برابر	ball_count_2d_plain.03.seq_lazy_linear_enc_lemmas_global_0.smt2	۷۷	۱/۹۸۰ برابر	ball_count_2d_hill_simple.10.seq_lazy_linear_enc_lemmas_global_0.smt2	۲۷
۱/۹۴۹ برابر	ball_count_2d_hill_simple.04.seq_lazy_linear_enc_lemmas_global_0.smt2	۷۸	۱/۹۸۳ برابر	ball_count_2d_plain.10.seq_lazy_linear_enc_lemmas_global_0.smt2	۲۸
۱/۹۵۵ برابر	ball_count_2d_plain.05.seq_lazy_linear_enc_lemmas_global_0.smt2	۷۹	۱/۹۹۵ برابر	ball_count_2d_hill.10.seq_lazy_global_0.smt2	۲۹
۲/۰۲۳ برابر	ball_count_2d_hill.05.seq_lazy_global_0.smt2	۸۰	۱/۹۷۶ برابر	ball_count_2d_hill_simple.10.seq_lazy_lemmas_global_0.smt2	۳۰
۲/۰۲۲ برابر	ball_count_2d_hill_simple.01.seq_lazy_lemmas_global_0.smt2	۸۱	۲/۰۲۲ برابر	ball_count_2d_plain.10.seq_lazy_lemmas_global_0.smt2	۳۱
۲/۰۰۱ برابر	ball_count_2d_hill_simple.02.seq_lazy_lemmas_global_0.smt2	۸۲	۱/۹۸۰ برابر	ball_count_2d_hill.10.seq_lazy_linear_enc_lemmas_global_0.smt2	۳۲
۱/۹۴۹ برابر	ball_count_2d_plain.01.seq_lazy_lemmas_global_0.smt2	۸۳	۱/۹۵۹ برابر	ball_count_2d_plain.10.redlog_global_0.smt2	۳۳
۱/۸۸۷ برابر	ball_count_2d_hill_simple.03.seq_lazy_lemmas_global_0.smt2	۸۴	۲/۰۴۹ برابر	ball_count_1d_plain.04.qfree_global_0.smt2	۳۴
۱/۹۹۱ برابر	ball_count_2d_hill_simple.04.seq_lazy_lemmas_globa	۸۵	۱/۸۲۴ برابر	ball_count_2d_hill_simple.03.qfree_global_0.smt2	۳۵

	l_0.smt2				
۱/۹۷۰ برابر	ball_count_2d_plain.04.seq_lazy_lemmas_global_0.smt2	۸۶	۱/۹۷۲ برابر	ball_count_2d_hill_simple.01.seq_lazy_linear_enc_global_0.smt2	۳۶
۲/۰۵۶ برابر	ball_count_2d_slope.02.seq_lazy_linear_enc_global_0.smt2	۸۷	۱/۹۸۷ برابر	ball_count_2d_hill_simple.04.qfree_global_0.smt2	۳۷
۲/۰۳۰ برابر	ball_count_2d_slope.03.seq_lazy_linear_enc_global_0.smt2	۸۸	۲/۰۷۰ برابر	ball_count_1d_plain.05.qfree_global_0.smt2	۳۸
۱/۹۸۸ برابر	ball_count_2d_hill.02.seq_lazy_linear_enc_lemmas_global_0.smt2	۸۹	۱/۸۶۴ برابر	ball_count_2d_plain.01.seq_lazy_linear_enc_global_0.smt2	۳۹
۲/۰۰۹ برابر	ball_count_2d_slope.04.seq_lazy_linear_enc_global_0.smt2	۹۰	۱/۸۵۵ برابر	ball_count_1d_plain.04.seq_lazy_linear_enc_global_0.smt2	۴۰
۱/۹۸۴ برابر	ball_count_2d_hill.03.seq_lazy_linear_enc_lemmas_global_0.smt2	۹۱	۱/۷۶۵ برابر	ball_count_2d_plain.03.seq_lazy_linear_enc_global_0.smt2	۴۱
۲/۰۲۳ برابر	ball_count_2d_hill.04.seq_lazy_linear_enc_lemmas_global_0.smt2	۹۲	۱/۹۵۶ برابر	ball_count_2d_hill_simple.03.seq_lazy_global_0.smt2	۴۲
۱/۹۷۷ برابر	ball_count_2d_slope.05.seq_lazy_linear_enc_global_0.smt2	۹۳	۱/۸۹۲ برابر	ball_count_2d_hill_simple.05.qfree_global_0.smt2	۴۳
۲/۰۶۷ برابر	ball_count_2d_plain.01.redlog_global_0.smt2	۹۴	۱/۷۶۴ برابر	ball_count_1d_plain.05.seq_lazy_global_0.smt2	۴۴
۲/۰۲۸ برابر	ball_count_2d_hill.05.seq_lazy_linear_enc_lemmas_global_0.smt2	۹۵	۲/۰۰۷ برابر	ball_count_2d_hill_simple.04.seq_lazy_linear_enc_global_0.smt2	۴۵
۲/۰۰۷ برابر	ball_count_2d_plain.04.redlog_global_0.smt2	۹۶	۱/۷۴۳ برابر	ball_count_1d_plain.05.seq_lazy_linear_enc_global_0.smt2	۴۶
۲/۰۰۷ برابر	ball_count_2d_plain.05.redlog_global_0.smt2	۹۷	۱/۹۹۴ برابر	ball_count_2d_hill_simple.04.seq_lazy_global_0.smt2	۴۷
۱/۹۸۱ برابر	ball_count_2d_hill.01.seq_lazy_lemmas_global_0.smt2	۹۸	۱/۸۷۷ برابر	ball_count_2d_hill_simple.05.seq_lazy_linear_enc_global_0.smt2	۴۸
۱/۹۲۵ برابر	ball_count_2d_hill.02.seq_lazy_lemmas_global_0.smt2	۹۹	۲/۰۰۸ برابر	ball_count_2d_plain.05.seq_lazy_global_0.smt2	۴۹
۲/۰۱۱ برابر	ball_count_2d_hill.03.seq_lazy_lemmas_global_0.smt2	۱۰۰	۱/۹۶۷ برابر	ball_count_2d_plain.01.qfree_global_0.smt2	۵۰

شکل ۷ به صورت نمونه روند یادگیری عامل DQN در محیط ایجاد شده در منطق QF_NRA را نشان می‌دهد. نمودار بالای شکل میزان پاداش دریافتی در هر دوره از یادگیری عامل DQN (خط صاف آبی رنگ)، میزان پاداش دریافتی در هر دوره برای عامل تصادفی (خط نقطه چین قرمز رنگ) و میزان پاداش حداکثری که عامل می‌تواند دریافت کند (خط حاشور سبز رنگ) را نشان می‌دهد. میزان پاداش حداکثری از طریق محاسبه پاداش برای ترکیب‌های مختلف عمل‌ها به دست آمده است. نمودار پایین شکل ۷ تعداد گام‌های انجام شده توسط عامل DQN در هر دوره را نشان می‌دهد. همانطور که در شکل ۷ مشخص است برخلاف عامل تصادفی که نمی‌تواند پاداش دریافتی خود را افزایش دهد، عامل DQN در طول یادگیری توانسته است پاداش دریافتی خود را افزایش دهد. با توجه به نمودار پایین شکل ۷، عامل DQN توانسته است همزمان با افزایش پاداش دریافتی، تعداد عمل‌های انجام شده در هر دوره را نیز کاهش دهد و از آنجایی که هر عمل معادل یک تاکتیک (و پارامترهای آن) است، بنابراین عامل به درستی آموزش دیده است و عمل‌ها (تاکتیک‌های) مناسب را انتخاب کرده و انجام می‌دهد و عمل اضافی انجام نمی‌دهد.



شکل ۷ نمودار روند یادگیری عامل یادگیری تقویتی عمیق در منطق QF_NRA

۶-۲. نتایج آزمایش‌ها در منطق QF_NIA

همانطور که در جدول ۱۱ می‌توان مشاهده کرد، در منطق QF_NIA با استفاده از یادگیری انجام شده، سرعت حل حل‌کننده را می‌توان تا ۴/۷۴۴ برابر افزایش داد. همچنین می‌توان مشاهده کرد که مقدار rlimit تا ۴۵۹۰۶۰ برابر کاهش می‌یابد. در برخی موارد عامل آموزش دیده نتوانسته مقدار rlimit را کاهش و یا سرعت حل را افزایش دهد. دلیل این امر این است که عامل به اندازه کافی آموزش ندیده است و نیاز به آموزش بیشتر دارد. همچنین محدودیت زمانی در نظر گرفته شده نیز در این امر تاثیرگذار است.

جدول ۱۱ نتایج آزمایش‌ها در منطق QF_NIA

	با عامل آموزش دیده (DQN)	با عامل تصادفی	قابل

	کمترین میزان افزایش سرعت	بیشترین میزان افزایش سرعت	کمترین مقدار کاسته شده از rlimit	بیشترین مقدار کاسته شده از rlimit	کمترین میزان افزایش سرعت	بیشترین میزان افزایش سرعت	کمترین مقدار کاسته شده از rlimit	بیشترین مقدار کاسته شده از rlimit	
آسان	۱ برابر	۲/۰۹۹ برابر	۱۳ برابر	۶۰۹۸ برابر	۰/۷۸ برابر	۱/۲۶۳ برابر	۰/۹۹ برابر	۱/۴۳۴ برابر	۳۹
معمولی	۱/۱۱۴ برابر	۴/۷۴۴ برابر	۳۰۶ برابر	۴۵۹۰۶۰ برابر	۰/۹۶۹ برابر	۱/۲۱۶ برابر	۰/۹۴۹ برابر	۱/۴۲۹ برابر	۳۸
سخت	۱/۹۴۷ برابر	۱/۴۳ برابر	۱ برابر	۱۳۸۵۵۳ برابر	۰/۱۷ برابر	۱/۱۴۵ برابر	۰/۹۸۴ برابر	۱/۱۱۱ برابر	۲۳

در جدول ۱۲ نیز می‌توان میزان افزایش سرعت به ازای هر فرمول را به تفکیک مشاهده کرد.

جدول ۱۲ میزان افزایش سرعت در منطق QF_NIA به تفکیک فرمول

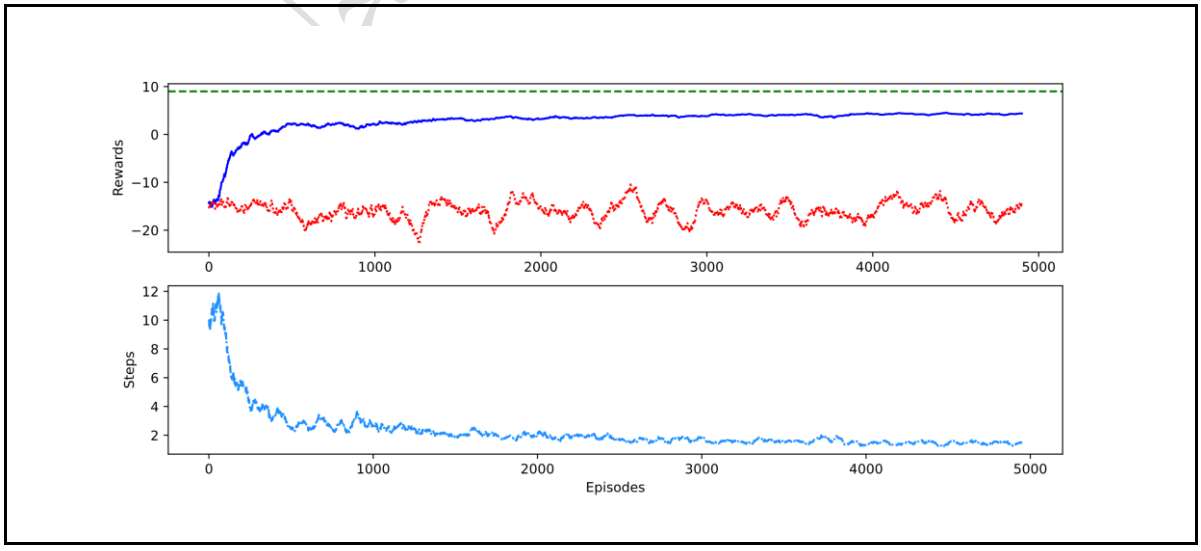
ردیف	نام فایل (فرمول)	افزایش سرعت حل	ردیف	نام فایل (فرمول)	افزایش سرعت حل
۱	aproveSMT106352422603502658.smt2	۱/۵ برابر	۵۱	aproveSMT201351602224939005.smt2	۱/۶۳۱ برابر
۲	aproveSMT110406245392905194.smt2	۱ برابر	۵۲	aproveSMT2430755879480044402.smt2	۲ برابر
۳	aproveSMT164281113689408140.smt2	۱/۶۳۲ برابر	۵۳	aproveSMT752462605509722678.smt2	۱/۲۶۷ برابر
۴	aproveSMT237454731236054743.smt2	۱/۶۶۷ برابر	۵۴	aproveSMT125187419356719603.smt2	۱/۳۷۴ برابر
۵	aproveSMT263445627778943920.smt2	۱/۳۳۳ برابر	۵۵	aproveSMT233295163504864559.smt2	۱/۲۶۰ برابر
۶	aproveSMT472666032770 1427.smt2	۱/۵۳۱ برابر	۵۶	aproveSMT991944918242922947.smt2	۱/۹۶۸ برابر
۷	aproveSMT191275437130012394.smt2	۱/۹۶۴ برابر	۵۷	aproveSMT358912782933805782.smt2	۲/۰۳۳ برابر

۸	aproveSMT172200405223484420.smt2	۱/۴۸۸ برابر	۵۸	aproveSMT206305505307433843.smt2	۱/۷۲۶ برابر
۹	aproveSMT32025504445261213.smt2	۱/۶۳۶ برابر	۵۹	aproveSMT265925057088562932.smt2	۱/۴۷۰ برابر
۱۰	aproveSMT106628954746449373.smt2	۲/۰۲۳ برابر	۶۰	aproveSMT257699921050028985.smt2	۱/۴۶۳ برابر
۱۱	aproveSMT141510234371783052.smt2	۱/۹۱۹ برابر	۶۱	aproveSMT474375145462252696.smt2	۱/۹۹۰ برابر
۱۲	aproveSMT335045203561853343.smt2	۱/۷۸۶ برابر	۶۲	aproveSMT314649917464608353.smt2	۲/۰۲۹ برابر
۱۳	aproveSMT199924319808952062.smt2	۲/۰۸۹ برابر	۶۳	aproveSMT269283320009384599.smt2	۱/۱۱۴ برابر
۱۴	aproveSMT465387089974364589.smt2	۱/۷۰۹ برابر	۶۴	aproveSMT208617998158592834.smt2	۱/۹۴۰ برابر
۱۵	aproveSMT475511664945327223.smt2	۱/۹۳۰ برابر	۶۵	aproveSMT425810002589506085.smt2	۱/۲۵۴ برابر
۱۶	aproveSMT338922425342401784.smt2	۱/۸۰۲ برابر	۶۶	aproveSMT205711926841620090.smt2	۱/۸۱۸ برابر
۱۷	aproveSMT188178440840 34737.smt2	۱/۹۱۷ برابر	۶۷	aproveSMT226894020986703971.smt2	۱/۷۷۶ برابر
۱۸	aproveSMT426987210793294180.smt2	۱/۹۸۹ برابر	۶۸	aproveSMT283487298308284366.smt2	۱/۳۳۶ برابر
۱۹	aproveSMT299708961479900596.smt2	۲/۰۸۰ برابر	۶۹	aproveSMT201132202294598618.smt2	۱/۷۴۵ برابر
۲۰	aproveSMT216462801678736982.smt2	۲/۰۱۹ برابر	۷۰	aproveSMT342988194676879281.smt2	۱/۹۸۲ برابر
۲۱	aproveSMT374020975285891691.smt2	۲/۰۴۷ برابر	۷۱	aproveSMT277795045419264694.smt2	۱/۷۷۰ برابر
۲۲	aproveSMT233357233309295005.smt2	۲/۰۷۳ برابر	۷۲	aproveSMT130911952525388558.smt2	۱/۵۸۵ برابر
۲۳	aproveSMT236580308612717708.smt2	۱/۸۸۹ برابر	۷۳	aproveSMT458300490583062727.smt2	۱/۴۸۳ برابر
۲۴	aproveSMT110719039202 52280.smt2	۱/۹۴۲ برابر	۷۴	aproveSMT390431825616874015.smt2	۲/۰۳۴ برابر
۲۵	aproveSMT297536522076030542.smt2	۲/۰۸۱ برابر	۷۵	aproveSMT271923944066788666.smt2	۱/۵۳۳ برابر
۲۶	aproveSMT359820136277819026.smt2	۱/۸۸۴ برابر	۷۶	aproveSMT233512211341539 85.smt2	۱/۷۹۷ برابر

۲۷	aproveSMT405002793410787217.smt2	۱/۹۴۹ برابر	۷۷	aproveSMT32533071786363354.smt2	۱/۴۱۲ برابر
۲۸	aproveSMT388188197435784909.smt2	۱/۹۸۳ برابر	۷۸	aproveSMT205663757693830508.smt2	۱ برابر
۲۹	aproveSMT97032048972614298.smt2	۱/۹۳۷ برابر	۷۹	aproveSMT353922419320417126.smt2	۱ برابر
۳۰	aproveSMT309344887399351087.smt2	۱/۹۲۰ برابر	۸۰	aproveSMT191866259888385 2.smt2	۱ برابر
۳۱	aproveSMT790359166007 4047.smt2	۲/۰۹۹ برابر	۸۱	aproveSMT409650305449726832.smt2	۱ برابر
۳۲	aproveSMT221654206070531325.smt2	۱/۹۱۰ برابر	۸۲	aproveSMT441996396180892764.smt2	۱/۲۷۸ برابر
۳۳	aproveSMT185813161521293895.smt2	۲/۰۸۲ برابر	۸۳	aproveSMT265556433380811208.smt2	۱/۴۳۰ برابر
۳۴	aproveSMT395451549965794310.smt2	۲/۰۸۷ برابر	۸۴	aproveSMT293614158054158832.smt2	۱ برابر
۳۵	aproveSMT166207192320126878.smt2	۱/۹۷۲ برابر	۸۵	aproveSMT334180970798087384.smt2	۱/۱۰۱ برابر
۳۶	aproveSMT479758788885008279.smt2	۲/۰۷۴ برابر	۸۶	aproveSMT463192377960637155.smt2	۱ برابر
۳۷	aproveSMT346224908798084830.smt2	۲/۰۰۶ برابر	۸۷	aproveSMT451682996481631551.smt2	۱ برابر
۳۸	aproveSMT414771145475661582.smt2	۱/۹۵۹ برابر	۸۸	aproveSMT463905008853206218.smt2	۱ برابر
۳۹	aproveSMT256227000274928233.smt2	۱/۹۲۶ برابر	۸۹	aproveSMT868126669190659 88.smt2	۱ برابر
۴۰	aproveSMT101417910736011502.smt2	۴/۰۳۸ برابر	۹۰	aproveSMT143564062931235708.smt2	۱ برابر
۴۱	aproveSMT379723609808799032.smt2	۱/۴۰۱ برابر	۹۱	aproveSMT174156337543845992.smt2	۱ برابر
۴۲	aproveSMT324761792014523705.smt2	۱/۲۲۷ برابر	۹۲	aproveSMT829803533355221 03.smt2	۱ برابر
۴۳	aproveSMT201343392445663280.smt2	۳/۴۴۶ برابر	۹۳	aproveSMT433457179563078955.smt2	۱ برابر
۴۴	aproveSMT370632052300982227.smt2	۲/۸۷۷ برابر	۹۴	aproveSMT397893769027547 99.smt2	۱ برابر
۴۵	aproveSMT258682010928380630.smt2	۱/۷۰۴ برابر	۹۵	aproveSMT365922853089376869.smt2	۱ برابر

۴۶	aproveSMT474292193369 83550.smt2	۱/۶۵۵ برابر	۹۶	aproveSMT10968038249005 4156.smt2	۱ برابر
۴۷	aproveSMT341225430958 194141.smt2	۱/۸۶۵ برابر	۹۷	aproveSMT38828301237259 7473.smt2	۱ برابر
۴۸	aproveSMT146826416665 60919.smt2	۱/۵۴۳ برابر	۹۸	aproveSMT45319836306326 3721.smt2	۱ برابر
۴۹	aproveSMT337115653666 620019.smt2	۱/۹۷۱ برابر	۹۹	aproveSMT18585716959443 3069.smt2	۱ برابر
۵۰	aproveSMT448542283443 020288.smt2	۴/۷۴۴ برابر	۱۰۰	aproveSMT40797630118971 6929.smt2	۱ برابر

شکل ۸ نیز به صورت نمونه روند یادگیری عامل DQN در محیط ایجاد شده در منطق QF_NIA را نشان می‌دهد. نمودار بالای شکل میزان پاداش دریافتی در هر دوره از یادگیری عامل DQN (خط صاف آبی رنگ)، میزان پاداش دریافتی در هر دوره برای عامل تصادفی (خط نقطه چین قرمز رنگ) و میزان پاداش حداکثری که عامل می‌تواند دریافت کند (خط حاشور سبز رنگ) را نشان می‌دهد. در این منطق نیز، میزان پاداش حداکثری از طریق محاسبه پاداش برای ترکیب‌های مختلف عمل‌ها به دست آمده است. نمودار پایین شکل ۸ نیز تعداد گام‌های انجام شده توسط عامل DQN در هر دوره را نشان می‌دهد. در این منطق نیز عامل DQN بر خلاف عامل تصادفی، در طول یادگیری توانسته است پاداش دریافتی را بیشینه کند. همچنین عامل DQN توانسته است تعداد عمل‌های انجام شده در هر اپیزود را کاهش دهد و بنابراین عامل به درستی آموزش دیده است و عمل‌ها (تاکتیک‌های) مناسب را انجام می‌دهد و عمل اضافی انجام نمی‌دهد.



شکل ۸ نمودار روند یادگیری عامل یادگیری تقویتی عمیق در منطق QF_NIA

۶-۳. مقایسه نتایج آزمایش‌ها

با استفاده از جدول ۱۳ می‌توان نتایج آزمایش‌ها را مقایسه کرد. با بررسی جدول ۱۳ می‌توان متوجه شد که در منطق QF_NRA عامل تصادفی به این علت که تعداد تاکتیک‌های موثر بر فرمول‌های این منطق نسبت به منطق‌های دیگر بیشتر است و هر چه تعداد تاکتیک‌های موثر بیشتر شود، احتمال انتخاب تاکتیک موثر توسط عامل تصادفی نیز بیشتر می‌شود، عملکرد نسبتا خوبی از خود نشان داده است. اما با این وجود عامل آموزش دیده در هر دو منطق خیلی بهتر از عامل تصادفی عمل کرده است.

جدول ۱۳ مقایسه نتایج آزمایش‌ها

منطق	با عامل آموزش دیده (DQN)				با عامل تصادفی			
	حداقل افزایش سرعت	حداکثر افزایش سرعت	حداقل میزان کاهش rlimit	حداکثر میزان کاهش rlimit	حداقل افزایش سرعت	حداکثر افزایش سرعت	حداقل میزان کاهش rlimit	حداکثر میزان کاهش rlimit
QF_NRA	۱/۶۹۵ برابر	۲/۱۳۶ برابر	۲۱ برابر	۴۵۱ برابر	۱/۰۴۵ برابر	۱/۹۰۱ برابر	۱/۱۸۵ برابر	۲/۵۷۹ برابر
QF_NIA	۱ برابر	۴/۷۴۴ برابر	۱ برابر	۴۵۹۰۶۰ برابر	۰/۱۷ برابر	۱/۲۶۳ برابر	۰/۹۸۴ برابر	۱/۴۳۴ برابر

۷. نتیجه‌گیری

Z3 ابزاری است که در کاربردهای متفاوتی در حوزه‌های مختلف دارد که یکی از آنها تشخیص مسیرهای غیرقابل اجرا است. بنابراین افزایش سرعت عملکرد این ابزار می‌تواند موجب بهبودهایی در حوزه‌های کاربرد آن نیز شود. برای مثال افزایش سرعت حل‌کننده‌های SMT می‌تواند کار را برای استفاده از گراف جریان کنترلی بین‌رویه‌ای آسان‌تر کند که در نتیجه آن می‌توان مسیرهای غیرقابل اجرای بشیتری را تشخیص داد. در این مقاله نشان داده شد که می‌توان با استفاده از یادگیری تقویتی، به ازای هر دسته از فرمول‌ها، تاکتیک‌های مناسبی را در حل کننده Z3 یاد گرفت، که استفاده از آنها منجر به حل سریع‌تر فرمول‌ها بشود. در روش پیشنهادی مقاله، با استفاده از عامل DQN تاکتیک‌های مختلف حل‌کننده Z3 یاد گرفته می‌شوند و سپس براساس یادگیری انجام شده تاکتیک‌ها انتخاب و روی فرمول‌های مختلف اعمال می‌شوند تا سرعت حل‌کننده Z3 افزایش یابد. در آزمایش‌هایی که در منطق‌های QF_NRA و QF_NIA انجام شد، نشان داده شد که با این روش سرعت حل‌کننده Z3 تا ۴/۷ برابر افزایش می‌یابد.

در این مقاله یک محیط برای تاکتیک‌های Z3 طراحی و ارائه شد که محققان می‌توانند از آن استفاده کنند[۷۰]. در طراحی محیط تلاش بر این بود که به گونه‌ای طراحی شود که قابلیت به کارگیری در الگوریتم‌های مختلف یادگیری تقویتی را داشته باشد. به همین دلیل این محیط همانند محیط‌های کتابخانه OpenAI Gym [25] طراحی و ارائه گردید.

[70] https://github.com/majidfeyzi/Z3-tactics-learning

مراجع

[1] P. Ammann and J. Offutt , *Introduction to Software Testing*, Cambridge University Press: Springer, 2016. doi: 10.1017/9781316771273

[2] B. Beizer, Software testing techniques, New York, 1990.

[3] ح. فرزانه، س. بخشایشی، ر. ابراهیمی آتانی و ا. شاه بهرامی، «مروری بر روش های تولید داده های آزمون در آزمون جهشی»، *مجله محاسبات نرم*، جلد ۲، ص. ۷۲-۸۵، ۱۳۹۲.

[4] Y.-W. Wang, Y. Xing and X.-Z. Zhang, "A Method of Path Feasibility Judgment Based on Symbolic Execution and Range Analysis," *International Journal of Future Generation Communication and Networking*, vol. 7, no. 3, pp. 205-212, 2014. doi: 10.14257/ijfgcn.2014.7.3.19

[5] H. Zhu, D. Jin, Y. Gong, Y. Xing and M. Zhou, "Detecting Interprocedural Infeasible Paths Based on Unsatisfiable Path Constraint Patterns," *IEEE Access*, vol. 7, pp. 15040-15055, 2019. doi: 10.1109/ACCESS.2019.2894593

[6] S. Jiang, H. Wang, Y. Zhang, M. Xue, J. Qian and M. Zhang, "An Approach for Detecting Infeasible Paths Based on a SMT Solver," *IEEE Access*, vol. 7, pp. 69058-69069, 2019. doi: 10.1109/ACCESS.2019.2918558

[7] L. d. Moura and N. Bjorner, "Z3: An efficient SMT solver," in *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, Springer, 2008, pp. 337-340. doi: 10.1007/978-3-540-78800-3_24

[8] C. Barrett and C. Tinelli, "Satisfiability modulo theories," in *Handbook of model checking*, Springer, 2018, pp. 305-343. doi: 10.1007/978-3-319-10575-8_11

[9] M. Balunovic, P. Bielik and M. Vechev, "Learning to solve SMT formulas," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[10] S. Anand, P. Godefroid and N. Tillmann, "Demand-driven compositional symbolic execution," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Springer, 2008, pp. 367-381. doi: 10.1007/978-3-540-78800-3_28

[11] J. Scott, F. Mora and V. Ganesh, "BanditFuzz: Fuzzing SMT solvers with reinforcement learning," 2020. doi: 10012/15753

[12] S. Jeon and J. Moon, "Dr. PathFinder: hybrid fuzzing with deep reinforcement concolic execution toward deeper path-first search," *Neural Computing and Applications*, vol. 34, pp. 10731-10750, 2022. doi: 10.1007/s00521-022-07008-8

[13] م. حاجی بابا و س. پارسا، «مکانیابی خطاهای پنهان نرم افزار با استفاده از آنتروپی متقاطع و مدلهای-n گرام»، *مجله محاسبات نرم*، جلد ۲، شماره ۱، صفحات ۴۴-۵۹، ۱۳۹۲.

[14] S. Ding, H. B. K. Tan and K. P. Liu, "A Survey of Infeasible Path Detection," *7th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE-2012)*,, pp. 43-52, 2012.

[15] S. Jang, H.-Y. Kim, Y.-H. Choi and T.-M. Chung, "A Study of Advanced Hybrid Execution Using Reverse Traversal," in *2011 International Conference on Information Management, Innovation Management and Industrial Engineering*, vol. 2, IEEE, 2011, pp. 557-559. doi: 10.1109/ICIII.2011.278

[16] ح. قربانی مقدم، ب. جاماسب و ح. دهدشتی جهرمی، «مروری بر نیازمندی‌های امنیتی در فرآیند تولید نرم‌افزار»، *مجله محاسبات نرم*، جلد ۱۰، شماره ۲، صفحات ۷۲-۸۳، ۱۴۰۰.

[17] B. Barhoush and I. Alsmadi, "Infeasible Paths Detection Using Static Analysis," *The Research Bulletin of Jordan ACM*, vol. 2, no. 3, pp. 120-126, 2013.

[18] N. Malevris, D. Yates and A. Veevers, "Predictive metric for likely feasibility of program paths," *Information and Software Technology*, vol. 32, no. 2, pp. 115-118, 1990. doi: 10.1016/0950-5849(90)90110-D

[19] D. Yates and N. Malevris, "Reducing The Effects Of Infeasible Paths In Branch Testing," *ACM SIGSOFT Software Engineering Notes*, vol. 14, no. 8, pp. 48-54, 1989. doi: 10.1145/75309.75315

[20] "SMT-LIB," [Online]. Available: https://smtlib.cs.uiowa.edu/. [Accessed 02 06 2022].

[21] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, London: The MIT Press, 2018.

[22] E. J. Weyuker, "The Applicability of Program Schema Results to Programs," *International Journal of Computer & Information Sciences*, vol. 8, no. 5, pp. 387-403, 1979. doi: 10.1007/BF00995175

[23] D. Gong and X. Yao, "Automatic detection of infeasible paths in software testing," *IET software*, vol. 4, no. 5, pp. 361-370, 2010. doi: 10.1049/iet-sen.2009.0092

[24] D. Kundu, M. Sarma and D. Samanta, "A UML model-based approach to detect infeasible paths," *Journal of Systems and Software*, vol. 107, pp. 71-92, 2015. doi: 10.1016/j.jss.2015.05.007

[25] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang and W. Zaremba, "OpenAI Gym," *arXiv preprint arXiv:1606.01540*, 2016. doi: 10.48550/arXiv.1606.01540

[26] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg and D. Hassabis, "Human-level control through deep reinforcement learning," *nature*, vol. 518, pp. 529-533, 2015. doi: 10.1038/nature14236