وارسی نمادین گزاره‌های منطق زمانی فازی روی گراف برنامه فازی

غلامرضا ستوده و علی موقر رحیم‌آبادی

چکیده: با ترکیب منطق‌های زمانی و منطق فازی می‌توان منطق‌های جدیدی ایجاد و از آن در وارسی خودکار مدل‌های پویای فازی استفاده نمود. تاکنون در چند مقاله مدل‌های کریپکه فازی FzKripke و گراف برنامه فازی FzPG به عنوان دو مدل زمانی فازی تعریف و جهت وارسی خواص زمانی روی این مدل‌ها، منطق زمانی FzCTL ارائه شده و بدون ارائه الگوریتم وارسی مدل، کاربردهایی از آنها در وارسی مدارات منطقی فازی مانند فلیپ- فلاپ‌های فازی معرفی شده است. در این مقاله جهت برخورد با مشکل انفجار فضای حالت در مدل‌های زمانی فازی، روشی نمادین ارائه شده که به کمک آن، مدل‌ها در قالبی بسیار فشرده ذخیره و پردازش می‌شوند. در این مقاله کارایی الگوریتم‌های طراحی‌شده نیز مورد ارزیابی تحلیلی و تجربی قرار می‌گیرند. به عنوان مطالعه موردی، کارایی روش در وارسی و کشف مخاطره پویای یک مدار فلیپ- فلاپ D فازی، مورد بررسی قرار گرفته و زمان اجرا و حافظه مصرفی الگوریتم در شرایط مختلف مدل، ارائه شده است.

کلیدواژه: **وارسی مدل، مدل کریپکه، منطق زمانی فازی، گراف برنامه فازی، وارسی نمادین مدل.**

۱- مقدمه

جهت وارسی رسمی خواص زمانی سیستم‌های سخت‌افزاری و نرم‌افزاری، تکنیکی پرقدرت با نام وارسی مدل[۱] توسط افرادی چون Clarke [۱] مطرح شده است. در این تکنیک، یک مدل ریاضی و رسمی (عمدتاً در قالب یک گراف) از روی مشخصات سیستم استخراج می‌شود و همچنین گزاره‌های مربوط به وارسی سیستم که به آن خواص گفته می‌شود، به زبانی رسمی (با نام منطق زمانی) ترجمه شده و نهایتاً روشی مکانیزه (در قالب یک الگوریتم) برای وارسی خواص در آن مدل، تعریف و پیاده‌سازی می‌شود. یکی از معروف‌ترین این مدل‌ها، مدل کریپکه[۲] نامیده می‌شود و برای توصیف خواص روی آن، منطق‌های زمانی CTL، LTL و CTL* ابداع شده است [۱] تا [۳]. تعمیم‌هایی از این مدل به زمان حقیقی[۳] صورت گرفته که اتوماتای زمانی (TA) نمونه‌ای از این مدل‌های تعمیم‌یافته است [۲].

یکی از مهم‌ترین مشکلات فرایند وارسی مدل، انفجار فضای حالت مدل‌هاست که در بعضی موارد به $۲^{۱۰۰}$ حالت می‌رسد. جهت برخورد با این مشکل، تکنیک‌هایی چون وارسی نمادین[۴] مدل و تجرید[۵] مدل‌های بزرگ به مدل‌های فشرده‌تر ارائه شده‌اند. یکی از مهم‌ترین ساختمان داده‌های استفاده‌شده در روش‌های نمادین، نمودار تصمیم‌گیری دودویی مرتب (OBDD) است که قابلیت فشرده‌سازی بالایی برای مدل‌ها دارد و زمان وارسی گزاره‌ها را نیز به خوبی کاهش می‌دهد [۲] و [۴] تا [۶]. در پردازش اتوماتای زمانی جهت فشرده‌سازی بیشتر فضای حالت از ساختمان داده‌های دیگر مثل ماتریس کران تفاضلی (DBM)، نمودار تصمیم‌گیری تفاضلی (DDD) و نمودار تصمیم‌گیری چندپایانه (MTBDD) نیز استفاده شده است [۷].

تاکنون تعمیم‌های مختلفی از مدل و منطق زمانی صورت گرفته است: تعمیم زمان گسسته به زمان حقیقی، لحاظ‌کردن مفهوم احتمال جهت برخورد با عدم قطعیت و فرایندهای تصادفی سیستم‌ها، در نظر گرفتن مفهوم هزینه در کنار زمان و ترکیب‌های مختلفی از این تعمیم‌ها [۲] و [۵] و [۶]. برای لحاظ‌کردن سطوح مختلف عدم قطعیت و ناسازگاری در فرایند وارسی مدل، مدل‌ها و منطق‌های زمانی چندمقداری تعریف شده که در آن به جای جبر بول، از جبرهای شبه‌بولی[۶] استفاده می‌شود (در این جبرها از یک شبکه[۷] عمدتاً متناهی استفاده می‌شود). برای جبرهای شبه‌بولی (با شبکه متناهی و بسیار کوچک) مدلی با نام XKripke و منطقی با نام XCTL ابداع شده و جهت وارسی نمادین این منطق الگوریتم‌هایی ارائه شده است [۴] و [۸] تا [۱۳]. اخیراً منطق چندمقداری جدیدتری با نام MCTL [۱۴] روی شبکه‌های متناهی تعریف گردیده و الگوریتمی برای وارسی آن ارائه شده است.

منطق فازی نیز نوعی منطق چندمقداری (با شبکه نامتناهی و پیوسته) است که در حوزه‌های مختلف کاربرد دارد [۱۵] و [۱۶]. با ترکیب منطق‌های زمانی و منطق فازی می‌توان مفاهیم فازی را در حوزه وارسی مدل، وارد کرد. در این زمینه کارهای محدودی انجام شده است که منطق‌های FTP [۱۷]، FBPL [۱۸] و [۱۹]، FzPLTL [۲۰] و Fuzzy Temporal Formula [۲۱] از آن جمله‌اند که عمدتاً جهت توصیف سیستم‌ها به کار می‌روند و برای عمده آنها اصلاً فرایند وارسی مدل دیده نشده و یا به طور محدود (فقط برای تست و نه وارسی کامل سیستم) پیاده‌سازی شده‌اند. در مواردی نیز از ابزارهای متداول وارسی مدل (های غیر فازی) برای وارسی کنترلرهای فازی استفاده شده است [۲۲] تا [۲۴].

به منظور تعریف مدل‌ها و منطق‌های زمانی فازی با قابلیت داشتن فرایند خودکار وارسی مدل، در [۲۵] و [۲۶] مدل کریپکه فازی[۸] که حالت خاصی از XKripke است تعریف شده و برای وارسی خواص آن منطق زمانی FzCTL که شباهت زیادی با XCTL دارد ارائه گردیده است. همچنین مدلی نسبتاً جدید و فشرده با نام گراف برنامه فازی (FzPG) با قابلیت تبدیل به مدل کریپکه فازی تعریف شده است. در [۲۶] و [۲۷] کاربردهایی از این منطق و مدل‌های زمانی فازی در وارسی مدارات

این مقاله در تاریخ ۲۵ اردیبهشت ۱۳۹۵ دریافت و در تاریخ ۹ آذر ماه ۱۳۹۵ بازنگری شد.

غلامرضا ستوده، گروه کامپیوتر، دانشگاه آزاد اسلامی واحد علوم و تحقیقات، تهران، ایران، (email: setoodeh@iaushiaz.ac.ir).

علی موقر رحیم‌آبادی، دانشکده مهندسی کامپیوتر، دانشگاه صنعتی شریف، تهران، ایران، (email: movaghar@shaif.edu).

1. Model Checking
2. Kripke
3. Real Time
4. Symbolic
5. Abstraction
6. Quasi-Boolean
7. Lattice
8. FzKripke

منطقی فازی (خصوصاً فلیپ- فلاپ‌های فازی) ارائه شده است. در [۲۸] مدلی مشابه با FzKripke به نام ساختار کریپکه فازی (FKS) به همراه منطق متناظر FCTL تعریف شده و یک الگوریتم وارسی برای مدل‌های کوچک ارائه شده است اما در آن به مشکل انفجار فضای حالت پرداخته نشده است.

در [۲۹] از تعمیم گراف برنامه فازی به زمان پیوسته، مدلی با نام FzTA تعریف گردیده و کاربردهایی از آن (مانند کنترل پروژه فازی) ارائه شده است. برای زمان پیوسته، منطق دیگری با نام FRTL [۳۰] تعریف شده و الگوریتمی برای وارسی گزاره‌های آن روی یک ردیابی محدود (و نه یک مدل زمانی کامل) ارائه شده است.

مدل کریپکه متناظر با گراف برنامه فازی (FzPG) ممکن است تعداد حالات زیادی (مثلاً بیش از 2^{100} گره) داشته باشد. از این رو در این مقاله یک سری الگوریتم جهت وارسی نمادین گراف برنامه فازی ارائه خواهد شد که با صرف حافظه مصرفی نسبتاً کمی وارسی گزاره‌ها را انجام می‌دهند. این الگوریتم‌ها از لحاظ کارایی زمان و حافظه مورد تحلیل ریاضی قرار می‌گیرند و علاوه بر آن به صورت تجربی (در وارسی یک مدار فلیپ- فلاپ D فازی) ارزیابی می‌گردند.

پس از این مقدمه در بخش‌های ۲ تا ۴ تعاریف مدل کریپکه فازی، منطق زمانی فازی FzCTL و گراف برنامه فازی ارائه می‌گردد. در بخش ۵ به طراحی الگوریتم وارسی نمادین گراف برنامه فازی پرداخته می‌شود و مرتبه زمانی الگوریتم تحلیل می‌گردد. در بخش ۶ به عنوان مطالعه موردی، کارایی الگوریتم در وارسی یک فلیپ- فلاپ D فازی ارزیابی خواهد شد و نهایتاً در بخش ۷ به جمع‌بندی مباحث پرداخته می‌شود و پیشنهادهایی جهت ادامه کار ارائه می‌گردد.

۲- مدل کریپکه فازی FZKRIPKE

هر چند منطق فازی عمدتاً بر روی بازه پیوسته $[0,1]$ تعریف و مورد استفاده قرار می‌گیرد اما در این مقاله جهت سادگی، مدل‌ها و منطق‌های زمانی فازی روی بازه کوانتیزه‌شده $[0,1]_\Delta$ تعریف می‌گردند که در آن Δ یک عدد کوچک (به صورت معکوس یک عدد طبیعی) است و بازه کوانتیزه به صورت (۱) تعریف می‌شود

$$[0,1]_\Delta = \{k\Delta : k \in 0 \dots 1/\Delta\} \tag{1}$$

مدل کریپکه فازی (FzKripke) کوانتیزه حالت خاصی از XKripke است که در جبر شبه‌بولی مرتبط با آن از مجموعه $[0,1]_\Delta$ استفاده می‌شود و از طرفی شباهت زیادی به گراف فازی [۳۱] دارد. این مدل به صورت چندتایی مرتب $M=(S,X,R,L,I)$ است و در آن $X=\langle x_1,\dots,x_m \rangle$ مجموعه‌ای از صفات و $S=\{s_1,\dots,s_n\}$ مجموعه وضعیت‌هاست. هر صفت دارای یک امکان است و مقادیر مختلف امکان برای کل صفات در قالب $Val_\Delta(X)$ قابل بیان است

$$Val_\Delta(X) = \{v_1,\dots,v_m \mid v_i \in [0,1]_\Delta\} \tag{2}$$

روی هر مقدارگیری خاص برای صفات، می‌توان دسترسی به یک مقدار متناظر با یک صفت را به کمک عمل نقطه مشخص کرد

$$\mu \in Val_\Delta(X)\ ,\ \mu = \langle v_1,\dots,v_m \rangle \Rightarrow \mu.x_i = v_i \tag{3}$$

تابع برچسب‌گذاری L برای هر وضعیت یک مقدارگیری را منسوب می‌کند و تابع R امکان انتقال از یک وضعیت به وضعیت دیگر را مشخص می‌سازد. امکان ورود به هر گره در موقع شروع به کمک تابع I بیان می‌شود

$$L : S \to Val_\Delta(X) \tag{4}$$

$$R : S \times S \to [0,1]_\Delta \tag{5}$$

$$I : S \to [0,1]_\Delta \tag{6}$$

به یک FzKripke که تابع I آن برای مبدأ s_0 مقدار یک و برای سایر گره‌ها مقدار صفر برمی‌گرداند، مدل کریپکه فازی تک‌مبدأ[1] اطلاق می‌شود.

رابطه (۷) تغییر بین دو وضعیت با امکان خاص را بیان می‌کند

$$s_i \xrightarrow{r} s_j \Leftrightarrow ((s_i,s_j),r) \in R \tag{7}$$

یک مسیر اجرایی متناهی با نام π با شروع از s'_0 به صورت (۸) قابل تعریف است

$$\pi \in Path_{fin}(s'_0) \Leftrightarrow \pi = s'_0 \xrightarrow{r_0} s'_1 \xrightarrow{r_1} s'_2 \xrightarrow{r_2} \dots \xrightarrow{r_{u-1}} s'_u \ ,\ \forall i \in 0 \dots u-1 \bullet r_i \in [0,1]_\Delta \tag{8}$$

مسیر اجرایی نامتناهی نیز به طور مشابه قابل بیان است

$$\pi \in Path_{\inf}(s'_0) \Leftrightarrow \pi = s'_0 \xrightarrow{r_0} s'_1 \xrightarrow{r_1} s'_2 \xrightarrow{r_2} \dots \ ,\ \forall i \in \mathbb{N} \bullet r_i \in [0,1]_\Delta \tag{9}$$

یک وضعیت خاص از مسیر و همچنین یک زیرمسیر به کمک نمادهای تعریف‌شده در (۱۰) بیان می‌شود

$$\forall i \in \mathbb{N} \bullet \quad \pi[i] = s'_i \wedge \pi[i..] = s'_i \xrightarrow{r_i} s'_{i+1} \xrightarrow{r_{i+1}} \dots \tag{10}$$

۳- منطق زمانی فازی FZCTL

قبل از توصیف این زبان، لازم است اعمال فازی مورد نیاز تعریف گردند. پیاده‌سازی اعمال فازی دارای تنوع بسیار بالایی است [۱۵] و [۱۶] اما در این مقاله از ساده‌ترین پیاده‌سازی اعمال فازی استفاده شده که خواص آنها با اعمال منطقی جبرهای شبه‌بولی مربوط به XCTL شباهت دارد. این پیاده‌سازی به صورت (۱۱) است

$$a \sqcap b = \min(a,b)\ ,\ \neg a = 1-a\ ,\ a \sqcup b = \max(a,b)\ ,\ a \to b = \max(b,1-a) = b \sqcup a\ ,\ \text{true} = 1\ ,\ \text{false} = 0 \tag{11}$$

غیر از این اعمال، عمل اشباع روی یک مقدار حقیقی به صورت (۱۲) تعریف می‌شود (از کلمه اشباع در [۲۳] با تعریفی مشابه استفاده شده است)

$$[\![a]\!] = \max(0,\min(1,a)) \tag{12}$$

عمل اشباع گسسته نیز به صورت (۱۳) تعریف می‌شود

$$[\![a]\!]_\varepsilon = [\![\varepsilon[\frac{a}{\varepsilon}]]\!] \tag{13}$$

با توجه به تعاریف فوق، نحو گزاره‌های روی وضعیت (با عنوان φ) و گزاره‌های روی مسیر (با عنوان Φ) در قالب گرامر (۱۴) قابل بیانند

$$\varphi ::= r \mid x \mid \varphi \mid \varphi \sqcap \varphi \mid \varphi \geq \varphi \mid [\![\varphi+\varphi]\!] \mid A\Phi \mid E\Phi \qquad \Phi ::= X\varphi \mid \varphi U \varphi \quad ,\quad r,\varepsilon \in [0,1]_\Delta, \varepsilon > 0, x \in X \tag{14}$$

جمع محدود $[\![\varphi+\varphi]\!]$ در XCTL دیده نشده ولی در این منطق لحاظ شده است. در این دو گزاره از عمل اشباع و اشباع گسسته استفاده می‌شود

1. Single Source

و علاوه بر قیدهای بعداً (X) و تا وقتی (U) قیدهای نهایتاً و دائماً به صورت (۱۵) قابل تعریفند

$$AF\Phi = A(true U \Phi)\ ,\ AG\Phi = EF\Phi\ , \\ EF\Phi = E(true U \Phi)\ ,\ EG\Phi = AF\Phi \tag{۱۵}$$

یک سری اعمال کمکی نیز به گرامر قابل اضافه‌کردن هستند که به کمک سایر اعمال قابل بیانند. این اعمال برای گزاره‌های روی وضعیت قابل استفاده‌اند. غیر از اعمال منطقی $\rightarrow$ و $\sqcup$ که قبلاً تعریف شد، عمل تفاضل محدود نیز قابل تعریف است

$$[\![a-b]\!] = \neg[\![(\neg a)+b]\!] \tag{۱۶}$$

سایر اعمال مقایسه غیر از $\leq$ به کمک عمل $\geq$ و اعمال منطقی قابل تعریف هستند.

از نماد $\mathbb{P}$ و $\vDash$ برای بیان امکان برقراری یک گزاره، مشروط به حضور در یک وضعیت یا مسیر، استفاده می‌شود

$$\mathbb{P}(M, s \vDash \varphi) = \mathbb{P}(\varphi | M, s)\ , \\ \mathbb{P}(M, \pi \vDash \Phi) = \mathbb{P}(\Phi | M, \pi) \tag{۱۷}$$

معنای گزاره‌های روی وضعیت‌ها در (۱۸) تا (۲۱) دیده می‌شود

$$\mathbb{P}(M, s \vDash r) = r \tag{۱۸}$$

$$\mathbb{P}(M, s \vDash x) = L(s).x \tag{۱۹}$$

$$\mathbb{P}(M, s \vDash \neg\varphi) = \neg\mathbb{P}(M, s \vDash \varphi) \tag{۲۰}$$

$$\mathbb{P}(M, s \vDash \varphi\, op\, \psi) = \mathbb{P}(M, s \vDash \varphi)\, op\, \mathbb{P}(M, s \vDash \psi) \tag{۲۱}$$

که در آن op می‌تواند اعمال منطقی دوتایی، مقایسه و جمع (و تفریق) محدود باشد. نتیجه اعمال مقایسه فقط صفر یا یک خواهد بود

$$\mathbb{P}(M, s \vDash A\Phi) = \prod_{\pi \in Path_{\inf}(s)} \mathbb{P}(M, \pi \vDash_A \Phi) \\ \mathbb{P}(M, s \vDash E\Phi) = \coprod_{\pi \in Path_{\inf}(s)} \mathbb{P}(M, \pi \vDash_E \Phi) \tag{۲۲}$$

معنای گزاره‌های روی مسیر در (۲۳) تا (۲۵) دیده می‌شود

$$\mathbb{P}(M, \pi \vDash_E X\varphi) = R(\pi[\cdot], \pi[۱]) \sqcap \mathbb{P}(M, \pi[۱] \vDash \varphi) \tag{۲۳}$$

$$\mathbb{P}(M, \pi \vDash_A X\varphi) = R(\pi[\cdot], \pi[۱]) \rightarrow \mathbb{P}(M, \pi[۱] \vDash \varphi) \tag{۲۴}$$

$$\mathbb{P}(M, \pi \vDash_Q \varphi U \psi) = \mathbb{P}(M, \pi \vDash_Q \psi) \sqcup \\ (\mathbb{P}(M, \pi \vDash_Q \varphi) \sqcap \mathbb{P}(M, \pi \vDash_Q X(\varphi U \psi))), \\ Q \in \{A E\} \tag{۲۵}$$

برقراری یک گزاره روی کل مدل به صورت (۲۶) است

$$\mathbb{P}(M \vDash \varphi) = \prod_{s \in S} (I(s) \rightarrow \mathbb{P}(M, s \vDash \varphi)) \tag{۲۶}$$

با توجه به معنای گزاره‌های این منطق، تساوی‌های (۲۷) برقرار خواهد بود

$$E(\varphi U \psi) = \mu Z.(\psi \sqcup (\varphi \sqcap EX\, Z))\ , \\ A(\varphi U \psi) = \nu Z.(\psi \sqcap (\varphi \sqcup EX\, Z)) \tag{۲۷}$$

که در آن نمادهای μ و ν به معنای بزرگ‌ترین و کوچک‌ترین نقطه ثابت[۱] است.

1. Fixed-Point

می‌توان هر مدل کریپکه فازی $M = (S, X, R, L, I)$ را به مدل کریپکه فازی تک‌مبدأ $M' = (S', X, R', L', I')$ تبدیل کرد. برای این کار باید یک گره جدید مثل ι را به عنوان مبدأ به S اضافه نمود. به ازای هر گره $s \in S$ تساوی $R'(\iota, s) = I(s)$ در تابع گذر مدل جدید باید لحاظ شود و برای گذر بین گره‌های S عملکرد R و R' یکسان در نظر گرفته شود. تابع L' را باید طوری در نظر گرفت که برای گره مبدأ، امکان تمام متغیرها را صفر برگرداند و برای سایر گره‌ها عملکرد L را داشته باشد و عملکرد I' نیز طبق تعریف مدل کریپکه تک‌مبدأ خواهد بود. برای وارسی خاصیتی مانند φ روی M کافی است گزاره $AX(\varphi)$ را روی M' وارسی کرد. مدل تک‌مبدأ M' با مبدأ ι را می‌توان در قالب چندتایی (S', X, R', L', i) نیز نمایش داد.

۴- گراف برنامه فازی FZPG

با تعمیم مدل گراف برنامه PG مطرح‌شده در [۲] می‌توان مدل جدیدی ایجاد کرد که در قالب چندتایی $G = (S, s_\cdot, X, Init, Act)$ قابل بیان است و در آن $s_\cdot \in S$ حالت شروع بوده، *Init* تابعی است که امکان ورود به گره شروع را بیان می‌کند و رابطه *Act* بیانگر یال‌های انتقال بین گره‌ها است. برچسب هر یال یک زوج مرتب است که بخش اول آن، تابعی است که امکان انتقال را بیان می‌کند و بخش دوم آن، تابعی است که امکان صفات وضعیت قبل را به امکان صفات وضعیت جدید، نگاشت می‌کند

$$Init \in \mathcal{F}_{\Delta,X}\ ,\ Act \in S \times S \leftrightarrow \mathcal{F}_{\Delta,X} \times \mathcal{G}_{\Delta,X} \tag{۲۸}$$

در این تعریف $\mathcal{F}_{\Delta,X} \in \mathcal{P}(Val_\Delta(X) \rightarrow [\cdot,۱]_\Delta)$ و $\mathcal{G}_{\Delta,X} = \mathcal{F}_{\Delta,X}^{|X|}$ و توابع متعلق به $\mathcal{F}_{\Delta,X}^{|X|}$ دارای این محدودیتند که باید از نحو گرامر (۲۹) تبعیت کنند

$$\varphi ::= r | x | \varphi | \varphi \sqcap \varphi | \varphi \geq \varphi | [\![\varphi + \varphi]\!] \\ x \in X\ ,\ r, \varepsilon \in [\cdot,۱]_\Delta\ ,\ \varepsilon > \cdot \tag{۲۹}$$

سایر اعمال منطقی، اعمال مقایسه و تفریق محدود از روی اعمال بیان‌شده در نحو فوق، قابل استخراج هستند.

برای بیان معنای یک FzPG می‌توان یک FzKripke معادل را تعریف کرد. معادل با مدل $G = (S, s_\cdot, X, Init, Act)$ کریپکه فازی $K_G = (S', X, R, L, I)$ قابل تصور است که در آن

$$S' = S \times Val_\Delta(X) \tag{۳۰}$$

$$\forall \eta \in Val_\Delta(X)\ ,\ s \in S \bullet I(s, \eta) = \begin{cases} Init(\eta)\ ,\ s = s_\cdot \\ \cdot\ ,\ s \neq s_\cdot \end{cases} \tag{۳۱}$$

$$\forall \eta \in Val_\Delta(X)\ ,\ s \in S \bullet L(s, \eta) = \eta \tag{۳۲}$$

$$\forall \eta, \eta' \in Val_\Delta(X), s, s' \in S \bullet \\ R((s,\eta),(s',\eta')) = \coprod_{((s,s'),(A,B)) \in Act,\ \eta' = B(\eta)} A(\eta) \tag{۳۳}$$

موقعی که FzKripke معادل، گزاره‌ای را با امکان خاصی بپذیرد می‌گوییم گراف برنامه نیز با همان امکان، گزاره را می‌پذیرد. در مباحث بعد فرض می‌شود که تابع *Init* برای گره شروع گراف برنامه فازی، مقدار ثابت یک داشته باشد و برای بقیه گره‌ها صفر باشد. اگر گراف برنامه فازی چنین نباشد می‌توان با اضافه‌کردن یک گره صوری مانند ι به آن مبدأ جدیدی ایجاد کرد که *Init* برای آن همیشه مقدار یک است و در عوض باید تابع *Init* قبلی را با *Act* ادغام کرد.

جدول ۱: توابع کار با بردارهای OBDD.

تابع	عملکرد
$\text{v\_const}(c)$	نمایش باینری عدد صحیح c در قالب یک بردار
$\text{v\_val}(v)$	تبدیل بردار v به عدد صحیح معادل
$\text{v\_if}(q,v۱,v۲)$	پارامتر q یک OBDD و دو تای دیگر دو بردار است و نتیجه تابع برداری از OBDDهاست که درایه i ام آن برابر با $\text{if}(d,v۱[i],v۲[i])$ است.
$\text{v\_eql}(v۱,v۲)$, $\text{v\_neq}(v۱,v۲)$, $\text{v\_grt}(v۱,v۲)$, $\text{v\_geq}(v۱,v۲)$, $\text{v\_lst}(v۱,v۲)$, $\text{v\_leq}(v۱,v۲)$	مقایسه دو بردار از OBDDها و ایجاد یک OBDD. برای اختصار می‌توان $v۱\geq v۲$ را به جای $\text{v\_geq}(v۱,v۲)$ به کار برد. به طور مشابه سایر اعمال مقایسه قابل تعریف هستند.
$\text{v\_sub}(v۱,v۲)$, $\text{v\_add}(v۱,v۲)$	جمع و تفریق دو بردار و ایجاد بردار سوم با همان تعداد درایه
$\text{v\_mult}(v۱,c)$, , $\text{v\_div}(v۱,c)$	ضرب و تقسیم صحیح یک بردار با یک عدد ثابت و ایجاد برداری جدید
False, True	معادل $\text{v\_const}(۰)$ و $\text{v\_const}(۲^d)$

۵- وارسی نمادین FZCTL روی گراف برنامه فازی

نمودار تصمیم‌گیری دودویی مرتب (OBDD) ساختمان داده فشرده‌ای از درخت تصمیم‌گیری است که کاربرد زیادی در طراحی مدارات سخت افزاری داشته و کاربرد دیگر آن در وارسی نمادین مدل‌های زمانی است و در ابزارهایی مانند NuSMV [۳۲] از آن استفاده گردیده است. این ساختمان داده یک گراف جهت‌دار بدون حلقه (DAG) است که گره‌های میانی آن متغیرهای یک تابع منطقی هستند و برگ‌های آن دو وضعیت صفر و یک می‌باشند. با توجه به یک دنباله مرتب از متغیرها، نمودار طوری شکل می‌گیرد که متغیرهای ابتدای دنباله، در بالای نمودار (نزدیک ریشه) و متغیرهای انتهای دنباله، در پایین (نزدیک برگ‌ها) قرار گیرند. از این ساختمان داده علاوه بر ذخیره‌سازی یک تابع منطقی می‌توان برای ذخیره‌سازی یک رابطه و یا یک مدل گرافی (مانند مدل کریپکه) استفاده نمود.

جهت ذخیره‌سازی یک مدل کریپکه، ابتدا گره‌های مدل با یک سری متغیر بولی کد می‌شوند و تعداد متغیرها از مرتبه لگاریتم تعداد گره‌های مدل است. در این صورت هر زیرمجموعه از وضعیت‌های مدل، به کمک یک عبارت بولی قابل بیان است. علاوه بر گره‌ها برای رابطه انتقال نیز باید تابعی بولی تعریف کرد و در این رابطه متغیرهای بولی متناظری با نماد پریم (′) برای گره مقصد لحاظ می‌شود. بر روی OBDD ها اعمال بولی مانند $\wedge$، $\vee$، $\neg$ و $\oplus$ قابل انجام است و همچنین اعمال $\exists x$ و $\forall x$ برای حذف یک متغیر از OBDD قابل تعریف هستند. برای وارسی خواص CTL روی مدل کریپکه آنها را در قالب یک زبان دیگر با نام $\mu-$Calculus بیان نموده و به کمک تکرار اعمال روی OBDDها، زیرگزاره‌های آن خاصیت را با هم ترکیب می‌کنند [۳].

علاوه بر منطق‌های دومقداری (مانند CTL)، روش‌های نمادین مختلفی برای محاسبه ارزش گزاره‌های منطق‌های زمانی چندمقداری ارائه شده که در بعضی از آنها به کمک نمودارهای OBDD و بردارهایی از این نمودارها، مدل کریپکه چندمقداری و گزاره‌ها را در فضایی محدود ذخیره و پردازش می‌کنند [۹] و [۱۰].

برای وارسی خواص زمانی روی گراف برنامه فازی می‌توان در ابتدا یک مدل کریپکه به دست آورد و سپس از روش‌های نمادین متداول، استفاده کرد. اما مناسب‌تر آن است که بدون تشکیل مدل کریپکه مستقیماً گراف برنامه فازی را در قالبی نمادین ذخیره و مستقیماً ارزش گزاره‌ها را محاسبه کرد. در روش نمادین پیشنهادی از درخت تصمیم‌گیری مرتب (OBDD) و بردارهایی از آنها استفاده خواهد شد.

با فرض آن که دقت مدل و گزاره‌ها برابر $\Delta=۲^{-d}$ باشد (یعنی تمام اعداد شرکت‌کننده در گراف برنامه و فرمول‌های مورد وارسی ضریب Δ باشند) برای نمایش اعداد و امکان زیرفرمول‌ها می‌توان $d+۱$ بیت (با اندیس صفر برای کم‌ارزش‌ترین و d برای پرارزش‌ترین) در نظر گرفت. این بیت‌ها با ضرب این اعداد گویا در ۲^d و نمایش باینری حاصل در $d+۱$ رقم به دست می‌آیند. به عنوان نمونه برای بیان عدد اعشاری ۱ کافی است یک دنباله حاوی یک بیت ۱ و d بیت ۰ در نظر گرفت.

علاوه بر اعداد، صفات نیز باید در قالبی نمادین کد شوند. برای کدکردن هر متغیر از برداری حاوی $d+۱$ متغیر ساده استفاده می‌شود. اگر W_i بیانگر بردار معادل با صفت x_i باشد می‌توان آن را به صورت (۳۴) نمایش داد

$$W_i=\langle w_{i\cdot},\ldots,w_{id}\rangle \qquad (۳۴)$$

همچنین اجتماع تمام متغیرهای شرکت‌کننده در W_iها، W نامیده می‌شود

$$W=\langle W_۱,\ldots,W_k\rangle \qquad (۳۵)$$

برای کدکردن گره‌های گراف نیاز به $\lceil \log_۲^{|S|}\rceil$ متغیر وجود دارد که اجتماع آنها با U نام‌گذاری می‌شود (در الگوریتم‌های بعدی، علاوه بر مجموعه‌های U و W دو مجموعه متناظر با نام‌های U' و W' جهت بیان یال‌های انتقال استفاده خواهد شد).

در روش نمادین ارائه‌شده، به طور بازگشتی متناظر با هر زیرفرمول φ برداری از OBDDها با نام $\tau(\varphi)$ محاسبه می‌شود. برای اعداد، صفات و عملگرهای موجود در فرمول رویه‌ای برای ساخت τ باید ارائه شود. برای تعیین امکان گزاره روی کل گراف، پس از محاسبه $\tau(\varphi)$ رویه‌ای با نام Evaluate برای تبدیل آن به یک عدد اعشاری ارائه خواهد شد. برای پیاده‌سازی $\tau(\varphi)$ علاوه بر توابع کتاب‌خانه‌ای ساخت و ترکیب OBDDها، نیاز به یک سری توابع جهت کار با برداری از OBDDها وجود دارد. مهم‌ترین این توابع در جدول ۱ و روش بازگشتی محاسبه $\tau(\varphi)$ در شکل ۱ دیده می‌شود.

پیاده‌سازی دو عمل Eu و Au با توجه به مفهوم نقطه ثابت، در شکل ۲ دیده می‌شود. این دو عمل از Ex استفاده می‌کنند. تاکنون راجع به پیاده‌سازی آن عمل بحثی نشده اما در مباحث بعد، روش پیاده‌سازی آن ذکر خواهد شد.

برای به دست آوردن امکان یک گزاره φ روی کل گراف (که $\tau(\varphi)$ آن به صورت بردار θ است) تساوی (۳۶) را خواهیم داشت

$$\mathbb{P}(K_G\vDash\varphi)=\prod_{\eta\in Val(X)}\mathbb{P}(M,(s_\cdot,\eta)\vDash\varphi)=\prod_{\eta\in Val(X)}\theta(s_\cdot,\eta) \qquad (۳۶)$$

برای به دست آوردن $\sqcap$ در عبارت فوق می‌توان از (۳۷) و (۳۸) که روی هر مجموعه دلخواه I برقرار است استفاده نمود

```
Recursive Calculation of τ(φ)
τ(r) = Const(r)
τ(xi) = Wi
τ(φ) = Not(τ(φ))
τ(φ ⊓ ψ) = And(τ(φ),τ(ψ))
τ(⟦φ+ψ⟧) = Add(τ(φ),τ(ψ))
τ(φ ≥ ψ) = Geq(τ(φ),τ(ψ))
τ(EXφ) = Ex(τ(φ))
τ(AXφ) = Ax(τ(φ))
τ(EU(φ,ψ)) = Eu(τ(φ),τ(ψ))
τ(AU(φ,ψ)) = Au(τ(φ),τ(ψ))
-----------------------------------------
Const(r) = v_const(r * ۲^d )
Not(v) = v_sub(True, v)
And(v۱, v۲) = v_if(v_geq(v۱, v۲), v۲, v۱)
  Similarly Or(v۱, v۲) = v_if(v_geq(v۱, v۲), v۱, v۲)
Add(v۱, v۲)=v_if(v_geq(v1,Not(v۲)),True,v_add(v۱, v۲))
  Similarly Sub(v۱, v۲) =
                 v_if(v_leq(v۱, v۲),False, v_sub(v۱, v۲))
Geq(v۱, v۲) = v_geq(v۱, v۲)
  Similarly Grt(v۱, v۲) = v_grt(v۱, v۲), Eql(v) = v_eql(v)
Ax(v) = Not(Ex(Not(v)))
```

شکل ۱: الگوریتم تبدیل گزاره‌های FzCTL به بردارهای OBDD.

Au(v۱, v۲):	Eu(v۱, v۲):

Z := Not(v۲)
While (true) do begin
  L := And(Not(v۲),
    Or(Not(v۱), Ex(Z)))
  if (Eql(L, Z) = ۱)
    return ot(Z)
  Z := L
end
	Z := v۲
	While (true) do begin
	  L := Or(v۲,
	    And(v۱, Ex(Z)))
	  if (Eql(L, Z) = ۱)
	    return Z
	  Z := L
	End

شکل ۲: پیاده‌سازی قیدهای Eu و Au.

$$\prod_{i\in I} a_i = D \Leftrightarrow \exists i \in I \bullet a_i = D \wedge \forall i \in I \bullet a_i \geq D \qquad (۳۷)$$

$$\coprod_{i\in I} a_i = D \Leftrightarrow \exists i \in I \bullet a_i = D \wedge \forall i \in I \bullet a_i \leq D \qquad (۳۸)$$

در مباحث بعد، جهت استفاده از D در وارسی نمادین، می‌توان آن را در قالب برداری از متغیرها به صورت (۳۹) نمایش داد

$$D = \langle z_\cdot, \ldots, z_d \rangle \qquad (۳۹)$$

در شبه‌کد Evaluate موجود در شکل ۳، طریقه محاسبه $\mathbb{P}(K_G \vDash \varphi)$ ارائه شده است. ابتدا اعضای بردار θ محدود به گره $s_\cdot$ می‌گردد و بردار λ ساخته می‌شود (نماد $U(s_\cdot)$ به معنای $v\_eql(U, v\_const(s_\cdot))$ برای این منظور استفاده شده است).

نمودار β با توجه به خواص $\sqcap$ به دست می‌آید که یک ترکیب منطقی (min-term) از متغیرهای $z_\cdot$ تا z_d است (نماد $D[i]$ به معنی z_i است). در صورت شرکت z_i در β حاصل عبارت $\exists D.(D[i] \wedge \beta)$ برابر یک و در صورت شرکت $\neg z_i$ در β حاصل عبارت برابر صفر می‌شود. با قراردادن این بیت‌ها در کنار هم (به کمک تابع Val) عدد صحیحی ساخته می‌شود و با تقسیم بر ۲^d عددی بین صفر و یک به عنوان امکان به دست می‌آید.

۵-۱ پیاده‌سازی تابع EX

با توجه به معنای EX در FzCTL روی مدل K_G داریم

```
Evaluate(θ(U,V))
for i := ۰ to d do
    λ[i] := θ[i]∧U(s۰)
β۱ := ∃U.∃W.Eql(λ,D)
β۲ := ∀U.∀W.Geq(λ, D)
β := β۱ ∧ β۲
for i := ۰ to d do
            ρ[i] := ∃D.(D[i] ∧ β)
return v_val(ρ) / ۲^d
```

شکل ۳: پیاده‌سازی تابع Evaluate.

```
ρ(U, W) = Ex(θ(U,W))
θ' := θ[U'/U, W'/W]
θ۱ := Grt(θ', D)
θ۲ := Geq(θ', D)
α := ۰, β := ۰
for each transition edge Act(s, t) = (A, ⟨B۱,...,Bk⟩) do
begin
    γ := ∃U'. (U'(t)∧ θ۱)
    η := ∃U'. (U'(t)∧ θ۲)
    γ̂ := γ[B۱/W'۱,...,Bk/W'k]
    η̂ := η[B۱/W'۱,...,Bk/W'k]
    α := α ∨ (γ̂ ∧ Grt(A,D) ∧ U(s))
    β := β ∨ (η̂ ∧ Geq(A,D) ∧ U(s))
end
ψ := β ∧ ¬α
for i := ۰ to d do
    ρ[i] := ∃D.(D[i] ∧ ψ)
return ρ
```

شکل ۴: پیاده‌سازی تابع Ex.

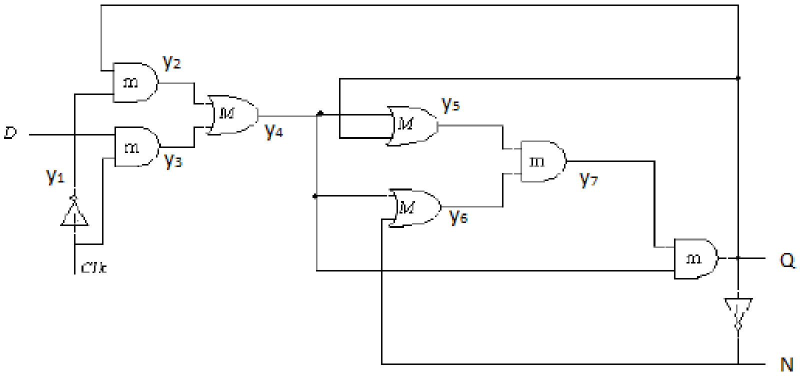
$$\mathbb{P}(M,(s,\eta) \vDash EX\,\varphi) = \coprod_{(s',\eta')\in S'} \mathbb{P}(M,(s',\eta') \vDash \varphi) \sqcap R((s,\eta),(s',\eta')) \qquad (۴۰)$$

اگر θ بردار $\tau(\varphi)$ و ρ بردار $\tau(EX\varphi)$ باشد، تساوی (۴۰) به صورت (۴۱) قابل باز نویسی خواهد بود

$$\rho(s,\eta) = \coprod_{(s',\eta')\in S'} \theta(s',\eta') \sqcap R((s,\eta),(s',\eta')) \qquad (۴۱)$$

برای محاسبه این عبارت، ساده‌ترین روش (که مشابه آن در [۸] استفاده شده است) آن است که ابتدا برداری از نمودارها برای R ایجاد و سپس با θ ترکیب گردد و نهایتاً بیشترین مقدار ممکن به کمک $\sqcup$ به دست آید. نمودار R به علت وجود متغیرها و متناظر پریم‌دار آنها، دارای ارتفاع مضاعفی است و در نتیجه دارای حجمی بسیار بالا است. پس از ترکیب R با θ از حجم نمودارهای نتیجه کاسته می‌شود اما پیش‌نیاز بودن ساخت کل نمودار R باعث می‌شود که این روش برای مدل‌های بزرگ دچار بن‌بست شود. از این رو، روش دیگری ارائه می‌شود که در آن هر یال گراف با توابعی از θ ترکیب شده و پس از جایگذاری‌های مناسب، نتایج ذره ذره با هم ترکیب می‌شوند تا ارتفاع نمودارها از حد مشخصی بالاتر نرود.

در شبه‌کد درج‌شده در شکل ۴، مراحل محاسبه ρ دیده می‌شود. در این کد، طریقه ساخت نمودار ψ روی مجموعه‌ای از مقادیر حقیقی بیان



شکل ۵: فلیپ- فلاپ D فازی [۳۳].

شده و در آن متغیرهای U، W و D شرکت دارند. ψ بیانگر تساوی برداری از نمودارها (بر حسب U و W) و بردار D است (اثبات درستی این الگوریتم به مقدمات زیادی نیاز دارد و در این مقاله از آن صرف نظر شده است).

۵-۲ ارزیابی مرتبه زمانی الگوریتم‌های طراحی‌شده

برای محاسبه Ax از Ex استفاده می‌شود و قسمت عمده زمان مصرفی آن را همین محاسبه Ex تشکیل می‌دهد و لذا زمان اجرای آن نیز از همین مرتبه است. در اجرای دو عملگر Eu و Au، عمل Ex چندین بار تکرار می‌شود و علاوه بر آن یک سری اعمال منطقی ساده نیز رخ می‌دهد که در برابر زمان مصرفی Ex هزینه چندانی ندارند.

جهت تحلیل زمان الگوریتم‌های استفاده‌شده در وارسی نمادین، یک سری علایم اختصاری تعریف می‌گردد

$$n=|U|\ ,\ d'=d+1\ ,\ h=|U|+|W|=n+kd' \qquad (۴۲)$$

با توجه به خطوط شکل ۴ می‌توان مرتبه زمانی الگوریتم Ex را به دست آورد

$$\begin{aligned}&O(d'2^{h}+d'2^{2(h+d')}+2^{h+d'})+\\&O(r(2^{(2k+1)d'}+2^{(2k+2)d'}(d'+2^{2n})))+\\&O(2^{2(h+d')}+d'2^{h+d'})=\\&O(r.2^{(2k+2)d'}(d'+2^{2n})+d'.2^{2(h+d')})\end{aligned} \qquad (۴۳)$$

معمولاً d' عدد کوچکی در نظر گرفته می‌شود و نسبت به $|S|$ (تعداد گره‌های گراف) کوچک است و می‌توان نتیجه گرفت که $d'=O(2^{n})$. از طرفی با توجه به آن که عمدتاً $r=O(|S|^{2})=O(2^{2n})$ زمان مصرفی کل الگوریتم Ex از مرتبه $O(2^{2(h+d'+n)})$ خواهد بود، زمان مصرفی Ax هم از همین مرتبه است. مرتبه زمان مصرفی اعمال Au و Eu با ضرب مرتبه زمانی Ex در $O(2^{h+d'})$ به دست می‌آید. در کل، زمان مصرفی فرایند وارسی مدل برای گزاره φ به صورت (۴۴) خواهد بود. سمت چپ حاصل جمع، با فرض استفاده فراوان (در حدود طول گزاره مورد وارسی) از اعمال سنگینی مانند Eu و Au در نظر گرفته شده و سمت راست حاصل جمع، زمان لازم برای Evaluate نتیجه نهایی است

$$O(|\varphi|(2^{3h+3d'+2n})+(d'2^{2(h+d')}))=O(|\varphi|(2^{3h+3d'+2n})) \qquad (۴۴)$$

مرتبه‌های زمانی به دست آمده دقیق و محکم[1] نیستند و بسته به مدل و گزاره مورد وارسی، مرتبه‌های زمانی پایین‌تری نیز محتمل هستند. برای تحلیل شهودی این الگوریتم‌ها، در بخش بعد به پیاده‌سازی و ارزیابی کارایی آنها در وارسی یک مدار فلیپ- فلاپ فازی پرداخته خواهد شد.

1. Tight

۶- پیاده‌سازی الگوریتم‌ها و ارزیابی تجربی کارایی آنها

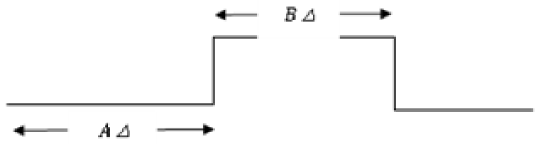
الگوریتم‌های ارائه‌شده در بخش قبل به کمک کتابخانه BuDDy [۳۱] پیاده‌سازی شده‌اند. در این کتابخانه امکانات مناسبی جهت کار با نمودارهای OBDD و بردارهایی از آنها وجود دارد. گره‌های مربوط به کل نمودارهای ساخته‌شده به کمک مکانیزم درهم‌سازی[2] ذخیره و بازیابی می‌گردند و اندازه هر گره ۲۰ بایت است. کران بالای تعداد کل گره‌ها در تابعی با نام bdd_init تنظیم می‌شود که مقدار آن صد میلیون گره در نظر گرفته شده است. در کل حجم حافظه RAM مصرفی در حدود ۲ GB خواهد بود (متأسفانه پردازش‌هایی که بیش از این مقدار حافظه نیاز دارند، دچار توقف می‌شوند).

یکی از کاربردهای منطق فازی در طراحی مدارات منطقی فازی است و برای این منظور نیازمند طراحی مدارات پایه‌ای (مانند فلیپ- فلاپ‌های فازی) هستیم و لازم است که صحت عملکرد این مدارات پایه تضمین شود. متأسفانه در این زمینه کار چندانی صورت نگرفته و فقط به شبیه‌سازی و تست مدارات با یک سری ردیابی ساده پرداخته شده و الگوریتمی برای وارسی کل حالات ورودی ممکن، ارائه نشده است. در [۳۳] یک فلیپ- فلاپ D چندمقداری (یا فازی) طراحی شده و صحت عملکرد آن به کمک شبیه‌سازی کامپیوتری بررسی شده است اما به کمک وارسی مدل می‌توان نشان داد که این مدار در شرایط خاصی به علت تأخیر در گیت‌ها دارای مخاطره پویا[3] است (بدین معنی که مقادیر غیر قابل پیش‌بینی تولیدشده، گذرا نبوده و دائماً تکرار می‌شوند و باعث ناپایداری مدار می‌گردند).

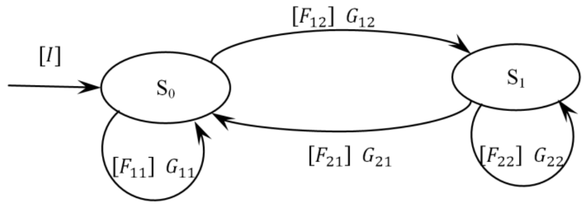
در شکل ۵، این فلیپ- فلاپ D فازی دیده می‌شود. نماد m بیانگر عمل min یا $\sqcap$ و نماد M بیانگر عمل max یا $\sqcup$ است. اسامی y_1 تا y_7 برای خروجی گیت‌های داخلی و Q برای خروجی فلیپ- فلاپ و N برای متمم Q در نظر گرفته شده است. همچنین جهت اختصار، $Clock$ با C نشان داده می‌شود.

برای سادگی فرض می‌شود که زمان تأخیر همه گیت‌های مدار، برابر مقدار ثابت $\Delta=2^{-h}$ واحد زمان باشد (h عددی صحیح مثبت است). فرض می‌شود که ورودی D مقداری پایدار[4] دارد اما C دارای یک پالس متناوب (به صورت شکل ۶) است. همچنین فرض می‌شود که A و B

2. Hash
3. Dynamic Hazard
4. Stable



شکل ۶: پالس ساعت ورودی مدار فلیپ- فلاپ D.



شکل ۷: گراف برنامه فازی معادل با مدار فلیپ- فلاپ D فازی.

اعدادی صحیح باشند. بزرگ‌بودن A و B مشکلی جز کندکردن سرعت مدار ندارد اما اگر A و B از حدی کوچک‌تر باشند، فرصت کافی برای پایداری خروجی‌ها به وجود نمی‌آید و عملکرد مدار نادرست خواهد بود. در مدار فلیپ- فلاپ فازی مورد بحث، حداقل A و B باید ۷ باشد (عملکرد نادرست مدار را برای اعداد کوچک‌تر از ۷ به کمک وارسی مدل می‌توان بررسی کرد). مقدار حداکثر A و B برابر $N=1/\Delta$ در نظر گرفته شده است (این فرض برای سادگی مدل‌سازی در نظر گرفته شده و بزرگ‌تر بودن A و B از مقدار N مشکلی جز کندی وارسی عملکرد مدار ایجاد نمی‌کند).

مقادیر گیت‌ها پس از Δ واحد زمان با اسامی پریم‌دار نام‌گذاری می‌شود. به راحتی می‌توان مقدار خروجی هر گیت را از روی ورودی‌ها محاسبه کرد

$$\begin{aligned}&y_1'=\neg C\ ,\ y_2'=y_1\sqcap Q\ ,\ y_3'=D\sqcap C\ ,\ y_4'=y_2\sqcup y_3\ ,\\&y_5'=y_4\sqcup Q\ ,\ y_6'=y_4\sqcup N\ ,\ y_7'=y_5\sqcap y_6\ ,\\&Q'=y_4\sqcap y_7\ ,\ N'=\neg Q\end{aligned} \qquad (45)$$

یک $FzPG$ با نام G (مطابق شکل ۷) با دو گره s_0 و s_1 در نظر گرفته می‌شود. در s_0 مقدار C برابر صفر و در s_1 برابر یک بوده و مجموعه صفات به صورت (۴۶) است

$$X=\langle T,u,D,C,y_1,y_2,y_3,y_4,y_5,y_6,y_7,Q,N\rangle \qquad (46)$$

متغیر T بیانگر گذر زمان با گام‌های Δ در گره‌های گراف برنامه فازی است. به محض ورود به هر گره، این متغیر مقدار صفر دارد و با تغییر این متغیر در هر گره، خروجی کلیه گیت‌ها تغییر می‌کند. متغیر u بیانگر اولین بالارفتن پالس ساعت است. پس از این که u مقدار یک گرفت، دائماً مقدار یک را نگه خواهد داشت.

جزئیات توابع شرکت‌کننده در $FzPG$ در (۴۷) تا (۵۱) دیده می‌شود

$$I=(T=0)\sqcap(C=0)\sqcap(u=0) \qquad (47)$$

$$\begin{aligned}&F_{11}=(T<A)\ ,\ F_{12}=(T=A)\ ,\\&F_{22}=(T<B)\ ,\ F_{21}=(T=B)\end{aligned} \qquad (48)$$

$$G_{12}=\langle 0,1,D,1,y_1,y_2,y_3,y_4,y_5,y_6,y_7,Q,N\rangle \qquad (49)$$

$$G_{21}=\langle 0,u,D,0,y_1,y_2,y_3,y_4,y_5,y_6,y_7,Q,N\rangle \qquad (50)$$

$$G_{11}=G_{22}=\left\langle\begin{matrix}[\![T+\Delta]\!],u,D,C,\neg C,y_1\sqcap Q,\\ D\sqcap C,y_2\sqcup y_3,y_4\sqcup Q,y_4\sqcup N,\\ y_5\sqcap y_6,y_4\sqcap y_7,\neg Q\end{matrix}\right\rangle \qquad (51)$$

جدول ۲: کارایی الگوریتم وارسی فلیپ- فلاپ D فازی به ازای $h=5$.

A	B	#EX	Time (ms)	#nodes
۸	۸	۲۳	۹۱۸	۶۳۲۹۴۴
۱۶	۸	۳۱	۱۶۳۹	۱۰۴۰۰۹۶
۸	۱۶	۳۱	۱۹۵۴	۱۱۷۱۲۳۱
۱۶	۱۶	۳۹	۲۴۴۱	۱۳۱۴۰۳۱
۲۴	۸	۳۹	۲۶۷۳	۱۴۳۶۲۰۸
۸	۲۴	۳۹	۳۰۴۶	۱۶۱۳۵۰۰
۳۲	۸	۴۷	۳۸۳۲	۱۷۴۲۴۴۹
۱۶	۲۴	۴۷	۳۹۲۴	۱۸۸۲۹۳۲
۲۴	۱۶	۴۷	۳۹۷۴	۱۸۳۸۵۴۸
۲۴	۲۴	۵۵	۴۶۴۵	۱۹۶۹۱۶۰
۳۲	۱۶	۵۵	۵۸۵۴	۲۱۸۳۱۱۷
۸	۳۲	۴۷	۶۳۶۵	۲۰۹۰۶۶۴
۳۲	۲۴	۶۳	۷۱۲۱	۲۴۵۷۸۲۴
۱۶	۳۲	۵۵	۸۰۸۲	۲۳۹۰۱۷۹
۲۴	۳۲	۶۳	۱۰۴۳۰	۲۶۲۲۰۷۳
۳۲	۳۲	۷۱	۱۳۰۰۶	۲۶۵۲۵۸۰

برای وارسی ردیف‌های جدول درستی مدار فلیپ- فلاپ چندمقداری، باید خواص (گزاره‌هایی) از منطق FzCTL را تعریف و به کمک روش وارسی مدل آنها را ارزیابی نمود. در این مقاله فقط به بررسی کارایی یکی از گزاره‌ها که بیانگر وجود مخاطره در مدار فلیپ- فلاپ است، پرداخته می‌شود (در [۲۷] بدون ذکر فرایند وارسی مدل، گزاره‌های بیشتری روی این مدار مورد بررسی قرار گرفته‌اند).

خاصیت یک: اگر ورودی مدار مقدار d داشته باشد، 6Δ بعد از اولین بالارفتن پالس ساعت، دائماً $Q=d$ خواهد بود

$$AG(u=1\rightarrow AX^{6}(AG(Q=D))) \qquad (52)$$

نماد AX^t به معنای اعمال t بار متوالی عمل AX است.

در وارسی گراف برنامه فازی طراحی‌شده، حالات مختلف Δ از 2^{-4} تا 2^{-9} در نظر گرفته شده و مقادیر A و B نیز از ۷ تا $1/\Delta$ تغییر داده شده‌اند اما خاصیت مزبور در همه حالات، ارزش صفر داشته است. با تغییر Δ هیچ تفاوتی در ارزش گزاره‌ها مشاهده نمی‌شود اما با کوچک‌کردن Δ، حافظه مصرفی، افزایش و سرعت وارسی مدل شدیداً کاهش می‌یابد.

این آزمایش‌ها به کمک یک PC با سیستم عامل ۳۲بیتی XP و پردازنده Intel® Core™ ۲ Due با سرعت ۲٫۶۷ GHz برای هر هسته و با حجم حافظه ۴ Gbyte صورت گرفته است. جزئیات و نتایج آزمایش‌ها به ازای مقادیر مختلف A، B و h در جداول ۲ و ۳ دیده می‌شود (چنان که قبلاً بیان شد $\Delta=2^{-h}$).

در جدول ۲ با فرض $h=5$ حالات مختلف A و B بررسی شده و به ازای هر یک، تعداد دفعات اجرای EX، کل زمان اجرا (بر حسب میلی‌ثانیه) و کل حافظه مصرفی (بر حسب جمع تعداد گره‌های OBDD) نمایش داده شده است. در جدول ۳ با فرض آن که $A\Delta=B\Delta=1/2$ به ازای حالات مختلف h، تعداد دفعات اجرای EX، کل زمان اجرا و کل حافظه مصرفی و حجم فضای حالت نمایش داده شده و در انتهای جدول، تقریبی از مرتبه زمان اجرا و حافظه مصرفی (به کمک رگرسیون) آمده است. با مشاهده این مرتبه‌ها، قدرت فشرده‌سازی و کارایی نمودارهای OBDD مشهود است و می‌توان دریافت که مرتبه زمان اجرا به مراتب از محاسبات ریاضی انجام‌شده در بخش ۵ پایین‌تر است.

جدول ۳: کارایی الگوریتم وارسی فلیپ- فلاپ D فازی در حالتی که $A\Delta = B\Delta =$ ۱/۲ .

h	State Space	#EX	Time (ms)	#nodes
۴	۲^۳۵	۲۳	۶۶۰	۵۲۶۳۹۵
۵	۲^۴۲	۳۹	۲۴۴۱	۱۳۱۴۰۳۱
۶	۲^۴۹	۷۱	۲۱۲۰۶	۲۸۲۱۹۶۵
۷	۲^۵۶	۱۳۵	۱۲۹۷۳۰	۵۷۱۵۱۱۴
۸	۲^۶۳	۲۶۳	۴۳۱۲۲۲	۱۱۳۵۸۴۷۵
۹	۲^۷۰	۵۱۹	۱۵۳۰۰۸۷	۲۲۴۸۵۱۲۹
Order	۲^(**h***۷)	۲^(**h***۰٫۹۰۵)	۲^(**h***۲٫۳۱)	۲^(**h***۱٫۰۷)

۷- نتیجه‌گیری و کارهای پیشنهادی

در این مقاله ابتدا مدل‌ها و منطق‌های زمانی فازی تعریف‌شده در [۲۵] تا [۲۷] و [۲۹] بررسی گردید. مدل کریپکه فازی FzKripke به عنوان حالت خاصی از مدل کریپکه چندمقداری (روی $[0,1]_\Delta$) ارائه گردید و بیان شد که جهت توصیف خواص زمانی این مدل، از منطق زمانی فازی FzCTL استفاده می‌شود. در این منطق به یک سری عملگر جدید (که در منطق‌های دیگر کمتر دیده می‌شود) مانند اعمال جمع و تفریق محدود، اشاره گردید. در مبحثی دیگر از مدلی فشرده با نام گراف برنامه فازی نام برده شد که قابلیت تبدیل به مدل کریپکه فازی را دارد.

در بخش اصلی این مقاله، الگوریتمی جهت وارسی گزاره‌ها روی گراف برنامه فازی به صورت نمادین (و به کمک بردارهایی از OBDD) مطرح شد و مرتبه زمانی الگوریتم اصلی و زیرالگوریتم‌های مطرح‌شده به طور تحلیلی محاسبه گردید. همچنین جهت ارزیابی دقیق‌تر کارایی این الگوریتم‌ها، به کمک کتابخانه BuDDy پیاده‌سازی شده و مورد آزمون تجربی قرار گرفته‌اند. در یک آزمون تجربی، وجود مخاطره پویا روی یک مدار فلیپ- فلاپ D فازی بررسی گردید و کارایی الگوریتم از لحاظ زمان و حافظه مصرفی در شرایط مختلف مدل ارائه گردید و در قالب چند جدول نمایش داده شد.

الگوریتم حاضر با وجود قابلیت ذخیره‌سازی و پردازش مدل‌های FzPG کوچک، همچنان از مرتبه نمایی است و برای مدل‌هایی که فضای حالت آنها از 2^{100} بیشتر باشد، جوابگو نیست و عمدتاً به علت کمبود حافظه متوقف می‌شود. جهت رفع این مشکل چندین پیشنهاد وجود دارد:

الف) استفاده از کتابخانه‌های جدیدتر با قابلیت ذخیره‌سازی و پردازش بیش از صد میلیون گره درخت‌های OBDD.

ب) استفاده از پردازنده‌های چندهسته‌ای جهت پردازش موازی درخت‌های OBDD.

ج) پیداکردن الگوریتمی که مرتبه زمانی آن فقط به تعداد گره‌های مدل گراف برنامه فازی وابسته باشد (حتی به صورت نمایی) و یا وابستگی آن به دقت Δ کم باشد. هر چند چنین کاری در نگاه اول غیر ممکن به نظر می‌رسد اما با نگاه دقیق‌تر به ساختمان داده‌های فشرده‌ای (مانند DDD) که در پردازش اتوماتای زمانی مورد استفاده قرار گرفته‌اند، امکان تعریف ساختمان داده مشابهی برای گراف برنامه فازی وجود دارد.

مراجع

[1] E. M. Clarke, J. O. Grumberge, and D. A. Peled, *Model Checking*, Cambridge, MA: MIT Press, 1999.

[2] C. Baier and J. P. Katoen, *Priciples of Model Checking*, Cambridge, MA: MIT Press, 2008.

[3] M. Huth and M. Ryan, *Logic Computer Science*, 2nd Ed. New York: Cambridge Univ. Press, 2004.

[4] B. Konikowska and W. Penczek, "On designated values in multi-valued CTL* model checking," *Fundamenta Informaticae*, vol. 60, no. 1-4, pp. 211-224, Sept. 2003.

[5] M. Kwiatkowska, G. Norman, J. Sproston, and J. Wang, "Symbolic model checking for probabilistic timed automata," *Information and Computation*, vol. 205, no. 7, pp. 1027-1077, Jul. 2007.

[6] F. Wang, *Symbolic Implementation of Model-Checking Probabilistic Timed Automata*, Ph.D Dissertation, School of Comput. Sci., Univ. of Birmingham, UK, 2006.

[7] F. Wang and M. Kwiatkowska, "An MTBDD-based implementation of forward reachability for probabilistic timed automata," in *Proc. 3rd Int. Sym. on Automated Technology for Verification and Analysis, ATVA'05*, pp. 385-399, Oct. 2005.

[8] M. Chechik, B. Devereux, S. Easterbrook, and A. Gurfinkel, "Multi-valued symbolic model-checking," *ACM Trans. Softw. Eng. Methodol.*, vol. 12, no. 4, pp. 371-408, Oct. 2003.

[9] M. Chechik, S. Easterbrook, and V. Petrovykh, "Model-checking over multi-valued logics," in *Proc. Int. Symp. Formal Methods Europe on Formal Methods for Increasing Software Productivity, FME'01,* pp. 72-98, Berlin, Germany, Mar. 2001.

[10] M. Chechik, A. Gurfinkel, B. Devereux, A. Lai, and S. Easterbrook, "Data structures for symbolic multi-valued model-checking," *Form. Method Syst. Des.*, vol. 29, no. 3, pp. 295-344, Nov. 2006.

[11] G. E. Fainekos, *An Introduction to Multi-Valued Model Checking*, Dept. Computer and Information Science, Univ. of Pennsylvania, Tech. Rep. MS-CIS-05-16, 2005.

[12] A. Gurfinkel, *Multi-Valued Symbolic Model-Checking: Fairness, Counter-Examples, Running Time*, M.S Thesis, Dept. Comput. Sci., Univ. of Toronto, Canada, 2003.

[13] A. F. Vilas, J. J. P. Arias, A. B. B. Martınez, M. L. Nores, R. P. D. Redondo, A. G. Solla, J. G. Duque, and M. R. Cabrer, "Multi-valued model checking in dense-time," *Lecture Notes in Computer Science*, vol. 3751, pp. 638-649, Dec. 2005.

[14] H. Pan, Y. Li, Y. Cao, and Z. Ma, "Model checking computation tree logic over finite lattices," *Theoretical Computer Science*, vol. 612, pp. 45-62, Jan. 2016.

[15] N. Sladoje, *On Analysis of Discrete Spatial Fuzzy Sets in 2 and 3 Dimensions*, Ph.D. Dissertation, Centre for Image Analysis, SLU and Uppsala Univ., Uppsala, Sweden, 2005.

[16] M. J. Wierman, An Introduction to the Mathematics of Uncertainty, 1st. ed., 20 Aug. 2010.

[17] P. Carinena, A. Burgarin, M. Mucientes, and S. Barro, "A language for expressing expert knowledge using fuzzy temporal rules," in *Proc. EUSFLAT-ESTYLF Joint Conf.,* pp. 171-174, Palma de Mallorca, Spain, Sept. 1999.

[18] S. Moon, K. Lee, and D. Lee, "Fuzzy branching temporal logic," *IEEE Trans. Syst. Man Cybern. B*, vol. 34, no. 2, pp. 1045-1055, Sept. 2004.

[19] J. Whittle, P. Sawyer, N. Bencomo, B. H. C. Cheng, and J. M. Bruel, "RELAX: incorporating uncertainty into the specification of self-adaptive systems," in *Proc. 17th IEEE Int. Requirements Eng. Conf,. RE'09,* pp. 79-88, Atlanta, GA, USA, Apr. 2009.

[20] G. Palshikar, "Representing fuzzy temporal knowledge," in *Proc. Int. Conf. on Knowledge-Based Systems, KBCS'00,* pp. 252-263, Mumbai, India, Dec. 2000.

[21] G. Bruns and P. Godefroid, "Model checking with multi-valued logics," *Lecture Notes in Computer Science*, vol. 3142, pp. 281-293, Jul. 2004.

[22] B. Intrigila, D. Magazzeni, A. Tofani, I. Melatti, and E. Tronci, "A model checking technique for the verification of fuzzy control

systems," in *Proc. Int. Conf. on Computational Intell. for Modelling, Control and Automation and Int. Conf. on Intelligent Agents, Web Tech. and Internet Commerce*, vol. 2, pp. 536-542, Nov. 2005.

[23] W. Liang, W. Bing-Wen, and G. Yi-Ping, "Cell mapping description for digital control system with quantization effect," presented at *Computing Research Repository, CoRR*, 2007, http://arxiv.org/abs/0712.2501

[24] J. Y. Yen and S. W. Tarng, "A fuzzy cell-mapping feedback control algorithm for the satellite attitude maneuvering control," in *Proc. Asian Fuzzy Syst. Symp. Soft Computing in Intelligent Syst. and Inform. Process.,* pp. 567-572, Kenting, Taiwan, Dec. 1996.

[۲۵] غ. ر. ستوده و ع. موقر رحیم‌آبادی، "تقریب و تجدید در منطق و مدل‌های زمانی فازی"، *شانزدهمین کنفرانس بین‌المللی سالانه انجمن کامپیوتر ایران*، صص. ۱۶۶-۱۶۰، تهران، اسفند ۱۳۸۹.

[26] G. R. Sotudeh and A. Movaghar, "Abstraction and approximation in fuzzy temporal logics and models," *Formal Aspects of Computing, London, UK, Springer-Verlag*, vol. 27, no. 2, pp. 309-334, Mar. 2015.

[27] G. R. Sotudeh and A. Movaghar, "Applications of fuzzy program graph in symbolic checking of fuzzy flip-flops," *J. of Computer & Robotics,* vol. 7, no. 1, pp. 22-36, Winter/Spring 2014.

[28] H. Pan, Y. Li, Y. Cao, and Z. Ma, "Model checking fuzzy computation tree logic," *Fuzzy Sets and Systems*, vol. 262, pp. 60-77, Mar. 2015.

[۲۹] غ. ر. ستوده و ع. موقر رحیم‌آبادی، "تعمیم مدل و منطق زمانی فازی به زمان حقیقی،" مجموعه مقالات *هفدهمین کنفرانس سالانه انجمن کامپیوتر ایران، تهران*، صص. ۵۲۴-۵۱۷، اسفند ۱۳۹۰.

[30] S. Mukherjee and P. Dasgupta, "A fuzzy real-time temporal logic," *International J. of Approximate Reasoning*, vol. 54, no. 9, pp. 1452-1470, Nov. 2013.

[31] J. Nild-Nielson, *BuDDy-A Binary Decision Diagram Package*, Dep. Inform. Tech., Technical Univ. of Denmark, 1999.

[32] http://nusmv.fbk.eu

[33] B. Choi and K. Shukla, "Multi-valued logic circuit design and implementation," *International J. of Electronics and Electrical Engineering*, vol. 3, no. 4, pp. 256-262, Aug. 2015.

غلامرضا ستوده تحصیلات خود را در مقاطع کارشناسی و کارشناسی ارشد مهندسی کامپیوتر به‌ترتیب در سال‌های ۱۳۷۸ و ۱۳۸۰ از دانشگاه صنعتی شریف و در مقطع دکتری مهندسی کامپیوتر در سال۱۳۹۳ از دانشگاه آزاد اسلامی واحد علوم و تحقیقات به پایان رسانده است و هم اکنون استادیار دانشکده مهندسی کامپیوتر دانشگاه آزاد اسلامی واحد شیراز می‌باشد. نام‌برده از سال ۱۳۸۱ تاکنون علاوه بر عضویت در هیأت علمی دانشگاه آزاد اسلامی واحد شیراز، به عنوان کارشناس نرم‌افزار در شرکت خدمات مهندسی عصر اندیشه شیراز به کار مشغول بوده است. زمینه‌های تحقیقاتی مورد علاقه ایشان عبارتند از: طراحی زبان‌های برنامه‌سازی و کامپایلرها، الگوریتم‌های موازی، وارسی رسمی سیستم‌های فازی، تست و وارسی نرم افزار، مدل‌سازی و ارزیابی کارایی سیستم‌های کامپیوتری، تحلیل و طراحی سیستم‌های اطلاعات مدیریت

علی موقر رحیم‌آبادی هم اکنون استاد دانشکده مهندسی کامپیوتر دانشگاه صنعتی شریف است و از بهمن ماه سال ۱۳۷۲ با این دانشگاه مشغول همکاری بوده است. نام‌برده در سال ۱۳۵۶ مدرک کارشناسی را در مهندسی برق از دانشکده فنی دانشگاه تهران و در سال‌های ۱۳۵۸ و ۱۳۶۴ مدارک کارشناسی ارشد و دکتری را در مهندسی کامپیوتر، اطلاعات و کنترل از دانشگاه میشیگان در آن آربور اخذ نمود. زمینه‌های پژوهشی مورد علاقه ایشان عبارتند از: مدل‌سازی کارایی/اتکاپذیری و درستی‌یابی شبکه‌های بی‌سیم، سیستم‌های توزیع شده بی‌درنگ و سیستم‌های سایبری فیزیکی.