دانشگاه صنعتی امیرکبیر
دانشکده مهندسی کامپیوتر و فناوری اطلاعات

گزارش کار و شرح پروژه درس طراحی الگوریتم‌های موازی

طراحی و پیاده‌سازی یک زبان برنامه‌سازی برای توصیف ماشین‌های سلولی

توسط:
علی قنبری

استاد درس:
دکتر محمدرضا میبدی

اردیبهشت ۹۲

بسم الله الرحمن الرحیم

دانشگاه صنعتی امیرکبیر
دانشکده مهندسی کامپیوتر و فناوری اطلاعات

گزارش کار و شرح پروژه درس طراحی الگوریتم‌های موازی

طراحی و پیاده‌سازی یک زبان برنامه‌سازی برای توصیف ماشین‌های سلولی

توسط:
علی قنبری

استاد درس:
دکتر محمدرضا میبدی

اردیبهشت ۹۲

چکیده

ماشین‌های سلولی راه دیگری را برای توصیف محاسبات در پیش‌روی ما قرار می‌دهند، استفاده از آن‌ها برای توصیف بخش‌هایی از سیستم‌های نرم‌افزاری می‌تواند در برخی از موارد مفید باشد.

در این پروژه زبانی را طراحی و پیاده‌سازی کرده‌ایم که به ما اجازه می‌دهد ماشین‌های سلولی مورد نظر خود را تـوصیف کنیم. کامپایلر پیاده‌سازی شده، این امکان را به ما می‌دهد که توصیف‌های نوشته را به ماژول‌هایی به زبان برنامه‌سازی C ترجمه کنیم. ماژول‌های تولید شده داری واسط‌های تعریف شده برای ارتباط با دنیای خارج هستند. به این ترتیب، قادر خواهیم بود در توسعه سیستم‌های نرم‌افزاری اجزائی از سیستم‌های خـود را با استفاده از ماشین‌های سلولی تـعریف کنیم، و با استفاده از کامپایلر ارائه شده توصیف‌های خود را به ماژول‌های قابل اتصال به سیستم مورد نظر، ترجمه کنیم.

کلیدواژه‌ها: ماشین‌های سلولی، زبان‌های توصیف فرمال، کامپایلر.

فهرست مطالب

۱- مقدمه

در این پروژه یک زبان برنامه‌سازی برای توصیف ماشین‌های سلولی سنکرون طراحی و پیاده‌سازی کرده‌ایم. این زبان برنامه‌سازی در اصل بر اساس معناشناخت عملیاتی که در مقاله [۱] معرفی شده، طراحی شده است. این زبان برنامه‌سازی با دریافت توصیف یک ماشین سلولی، برنامه‌ایی را به زبان C تولید می‌کند که ساختار و رفتار ماشین سلولی توصیف شده را داشته باشد. بنابراین، این زبان برنامه‌سازی می‌تواند برای توصیف هر ماشین سلولی با هر ابعادی بکار رود. در بخش‌های بعد خواهیم دید که با استفاده از این زبان کافی است، پارامترهای ماشین سلولی دلخواه خود را در یک فایل متنی نوشته و آن را کامپایل کنیم. کامپایلر، برنامه‌ایی را به زبان C برای ما تولید خواهد کرد که در عمل ماشین سلولی توصیف شده را پیاده‌سازی می‌کند.

در زیر بخش ۱-۱، نگاهی بر معناشناخت عملیاتی ماشین‌های سلولی می‌اندازیم. در بخش ۲، نحو زبان ارائه شده را تعریف می‌کنیم. همچنین، نگاهی بر چند برنامه نمونه خواهیم انداخت، و شیوه استفاده از کامپایلر این زبان را آموزش خواهیم داد. در بخش ۳، قواعد سیستم‌نوع و نیز معناشناخت عملیاتی زبان و ارتباط آن با معناشناخت ارائه شده در مقاله [۱] را خواهیم دید. در نهایت در بخش ۴، ضمن جمع‌بندی مطالب ارائه شده، نتیجه‌گیری و کارهای آینده که می‌تواند این پروژه را بهبود دهد را ارائه خواهیم کرد.

۱-۱- معناشناخت عملیاتی ماشین‌های سلولی

در این زیربخش، معناشناخت عملیاتی ماشین‌های سلولی سنکرون را معرفی می‌کنیم. مطالب این بخش بر اساس مقاله [۱] است. در این زیربخش، با ارائه تعریف‌هایی مدل محاسباتی ماشین‌های سلولی را شرح می‌دهیم. با این تعاریف، می‌توان یک زبان برنامه‌سازی طراحی کرد که در آن شیوه توصیف یک ماشین سلولی و نیز ارتباط این توصیف با کدهای قابل اجرا بر روی کامپیوتر بیان شده است.

تعریف ۱-۱: توپولوژی یک ماشین سلولی یک شبکه نامتناهی d بعدی است. هر سلول این شبکه با یک چندتایی مرتب (از نوع $\mathbb{Z}^d$) آدرس‌دهی می‌شود.

در این تعریف، حالت کلی و ایده‌آل یک ماشین سلولی ارائه شده است. ماشین‌های سلولی می‌توانند بصورت شبکه‌های متناهی بیان شده باشند. اما در این حالت نیز برای اینکه با تعریف کلی مغایرت نداشته باشد، شبکه متناهی با لبه‌های پیچشی در نظر گرفته می‌شود. بنابراین، شیوه آدرس‌دهی در این حالت بدون تغییر قابل استفاده است. لازم به ذکر است که در ادامه برای سادگی، به جای ماشین سلولی با شبکه متناهی (یا نامتناهی) از واژه ماشین سلولی متناهی (یا نامتناهی) استفاده خواهیم کرد. همچنین، در تمام این گزارش، منظور ما از ماشین سلولی، یک ماشین سلولی سنکرون است.

برای درک بهتر تعریف ۱-۱ و حالت‌های خاص آن، بد نیست مثالی را مشاهده کنیم.

مثال ۱-۱: یک ماشین سلولی متناهی دو بعدی را می‌توان با استفاده از یک شبکه دو بعدی متناهی، بصورت شکل زیر نشان داد. در این شکل برای سادگی بحث، هر سلول را با یک حرف نشان داده‌ایم.

a	b	c	d
e	f	g	h
i	j	k	l
m	n	o	p

شکل ۱-۱- یک ماشین سلولی متناهی دو بعدی که با استفاده از یک شبکه دو بعدی نمایش داده شده است.

طبق تعریف ۱-۱، برای آدرس‌دهی سلول‌های این شبکه از چندتایی‌های مرتب استفاده می‌شود. در این شکل فرض می‌کنیم که مبدإ آدرس‌دهی (یعنی سلول با آدرس (۰ ,۰)) سلول مربوط به حرف a است. بنابراین، سلول b با دوتایی (۰ ,۱)، و سلول p با دوتایی (۳, ۳) آدرس دهی می‌شود. در صورتی که شبکه را با لبه‌های پیچشی در نظر بگیریم (که همواره چنین در نظر گرفته می‌شود)، می‌توان هر سلول را به بیشمار شیوه آدرس‌دهی کرد. برای مثال سلول a را می‌توان در حالت کلی با مجموعه دوتایی‌های $\{(4n, 4n) \mid n \in \mathbb{Z}\}$ آدرس‌دهی کرد.

طبق تعریف ۱-۱ و مثال ارائه شده متوجه می‌شویم که تعداد ابعاد یک ماشین نامتناهی، و علاوه بر آن اندازه هر بعد از یک ماشین متناهی، به عنوان پارامترهای تعریف کننده ماشین‌های سلولی بکار می‌رود.

تعریف ۱-۲: حالت سلول‌های یک آتوماتای سلولی، مجموعه‌ایی متناهی از حالت‌ها است که آن را با S نشان می‌دهیم. در یک مقطع از زمان، هر یک از سلول‌های ماشین سلولی در یک حالت قرار می‌گیرند. بنابراین، هر ماشین سلولی در یک مقطع از زمان در یک پیکربندی قرار دارد که می‌توان آن را با استفاده از توابعی بصورت $c_i: \mathbb{Z}^d \to S$ تعریف کرد. تابع c_i پیکربندی iام ماشین را نشان می‌دهد.

مجموعه حالات یک ماشین سلولی به عنوان یکی از پارامترهای ماشین در نظر گرفته می‌شود. برای مثال، در ماشین سلولی که بازی زندگی را شبیه‌سازی می‌کند، مجموعه حالات بصورت S={زنده ,مرده} تعریف می‌شود. تابع c_i نیز در یک مقطع از زمان می‌تواند برای مشاهده اینکه هر یک از سلول‌های ماشین در چه حالتی قرار دارند، بکار رود. با استفاده از این تابع می‌توان قواعد محلی برای تغییر حالت را معرفی کرد که در ادامه تعریف خواهیم کرد.

طبق تعریف، سلول‌های یک ماشین سلولی حالت خود را در گام‌های زمانی گسسته تغییر می‌دهند. در حالت کلی، حالت بعد یک سلول (توسط فرمولی با نام قانون تغییر حالت محلی) وابسته به حالت فعلی آن سلول و حالت فعلی تعدادی (صفر یا بیشتر) از سلول‌های موجود در همسایگی آن سلول است. تمامی سلول‌ها در یک ماشین سلولی از یک قانون برای تغییر حالت استفاده می‌کنند. هنگام تغییر حالت، قانون تغییر حالت بصورت یکجا بر روی تمامی سلول‌های ماشین اعمال می‌شود.

سلول‌های موجود در همسایگی یک سلول می‌تواند سلول‌هایی در نظر گرفته شوند که سلول مورد نظر را محاصره کرده‌اند. اما در حالت کلی همسایگی یک سلول را بصورت زیر تعریف می‌کنند:

تعریف ۱-۳: فرض کنید بردار $\vec{N} = (x_1, x_2, \dots, x_n)$، یک بردار شامل n مؤلفه از نوع $\mathbb{Z}^d$ باشد که هر یک از این مؤلفه‌ها دو به دو با هم متمایز هستند. بنابراین، مجموعه آدرس‌های متناظر با سلول‌های موجود در همسایگی یک سلول با آدرس $x \in \mathbb{Z}^d$ بصورت زیر تعریف می‌شود:

$$\{x + x_i | i = 1, 2, \dots, n\}$$

در این صورت $\vec{N}$ را یک همسایگی برای سلول x می‌نامند.

تعریف ۱-۳، صورت کلی همسایگی سلول‌ها را تعریف می‌کند. این همسایگی‌ها می‌تواند با شعاع‌های دلخواه باشد. این در حالی است که تنها حالات خاصی از همسایگی در بسیاری از مسائل مورد استفاده قرار می‌گیرد. این همسایگی‌ها به ترتیب همسایگی‌های مور[1] و نیومن[2] نام دارند، حتی در این حالت نیز معمولاً همسایگی‌هایی با شعاع ۱ بیشتر مورد استفاده قرار می‌گیرد.

همسایگی $\vec{N}$ یک همسایگی نیومن با شعاع ۱ است، هر گاه هر مولفه x این همسایگی در رابطه $\|x\|_1 \leq 1$ صدق کند. در این رابطه نماد $\|x\|_1$ نُرم منهتن[3] است که بصورت زیر تعریف می‌شود:

$$\|x = (y, \dot{y}, \dots, \dot{y})\|_1 = |y| + |\dot{y}| + \cdots + |\dot{y}|$$ که در آن $y, \dot{y}, \dots, \dot{y} \in \mathbb{Z}$ است.

بطور مشابه، همسایگی $\vec{N}$ یک همسایگی مور با شعاع ۱ است، هر گاه هر مولفه x این همسایگی در رابطه $\|x\|_\infty \leq 1$ صدق کند. در این رابطه نماد $\|x\|_\infty$ نُرم ماکزیمم است که بصورت زیر تعریف می‌شود:

$$\|x = (y, \dot{y}, \dots, \dot{y})\|_\infty = \max\{|y|, |\dot{y}|, \dots, |\dot{y}|\}$$ که در آن $y, \dot{y}, \dots, \dot{y} \in \mathbb{Z}$ است.

این موضوع که یک ماشین در قانون تغییر حالت خود از کدام نوع همسایگی (یا همسایگی‌ها) برای تصمیم‌گیری استفاده کند، به عبارتی دیگر (تدوین) قانون تغییر حالت محلی یکی دیگر از پارامترهای یک ماشین سلولی است که در ادامه بصورت دقیق‌تر تعریف خواهیم کرد.

تعریف ۱-۴: قانون تغییر حالت محلی یک تابع بصورت $f: S^n \to S$ است که در آن n اندازه همسایگی است.

در صورتی که بخواهیم تعریف ۱-۴ را بصورت دقیق‌تر شرح دهیم، می‌توان گفت: $f(a, \dot{a}, \dots, \ddot{a})$ حالت یک سلول در زمان t +۱ است که همسایه‌های آن (طبق یک الگوی خاص از همسایگی که مورد نظر ما است)، در زمان t در حالت‌های $a, \dot{a}, \dots, \ddot{a}$ قرار دارند. هنگامی که قانون تغییر حالت محلی، بصورت مرحله به مرحله، همزمان بر روی تمامی سلول‌های یک ماشین سلولی اعمال می‌شود، ماشین از یک پیکربندی به

[1] Moore
[2] Neumann
[3] Manhattan Norm

یک پیکربندی دیگر منتقل می‌شود. در صورتی که بخواهیم از دید پیاده‌سازی به یک قانون تغییر حالت نگاه کنیم، می‌توان گفت تابع تغییر حالت یک برنامه است که پارامترهای ورودی این برنامه یک سلول (همان سلولی که می‌خواهیم قانون بر روی آن اعمال شود) و تمام همسایگی‌های آن است. دقت کنید که ممکن است در داخل برنامه، در شرایط خاص از انواع الگوهای همسایگی و هر یک با شعاع‌های مختلف استفاده کنیم. در این صورت ممکن است در هر بار که می‌خواهیم قانون تغییر حالت را اعمال کنیم، بسته به شرایط حاکم تعداد مختلفی از پارامترها نیاز باشد. بنابراین، دقیق‌تر این است که بگوئیم ممکن است n در تعریف ۱-۴، برابر با ماکزیمم تعداد همسایگی‌های یک سلول باشد. اما، در واقعیت و در تعریف صحیح یک ماشین سلولی چنین نیست و این بحث‌ها را کنار می‌گذارند و یک ماشین سلولی سنکرون نامتناهی dبعدی را با یک چهارتایی بصورت $(d, \vec{N}, S, f)$ شناسائی می‌کنند. در این تعریف d تعداد ابعاد ماشین، $\vec{N}$ تنها الگوی همسایگی بکار رفته در تعریف f، S مجموعه حالات سلول‌های ماشین (که یک مجموعه متناهی است) و در نهایت f قانون تغییر حالت محلی است.

در مورد ماشین‌های سنکرون متناهی d یک بردار بوده و مؤلفه‌های این بردار، طول هر بعد ماشین را نشان می‌دهد.

در صورتی که مطالبی را که تا کنون ارائه شده است را در کنار هم قرار دهیم، می‌توانیم تعاریفی را ارائه دهیم که معناشناخت عملیاتی ماشین‌های سلولی را تعریف می‌کنند. منظور ما از تعریف معناشناخت عملیاتی این است که تعریف کنیم که محاسبات در یک ماشین سلولی به چه صورت انجام می‌شود. در حوزه طراحی و پیاده‌سازی زبان‌های برنامه‌سازی، معناشناخت عملیاتی زبان‌های برنامه‌سازی را با استفاده از قوانین استنتاج نشان می‌دهند. خوشبختانه این نمایش دقیق و ساده در مورد بیان معناشناخت عملیاتی ماشین‌های سلولی هم می‌تواند مفید واقع شود.

فرض کنید t زمان حال را نشان دهد[۴]، و w زمان آخرین اعمال قانون (۱) در زیر باشد، در این صورت معناشناخت عملیاتی ماشین‌های سلولی بصورت زیر تعریف می‌شود:

$$\frac{w = t - ۱}{w = t \ \wedge \forall x \cdot e(x) = f(c(x + x_۱), c(x + x_۲), \ldots, c(x + x_n))} \qquad (۱)$$

قانون (۱) بیان می‌کند که در هر واحد زمانی قانون f را بر روی تک تک سلول‌های ماشین اعمال کن و ماشین را از پیکربندی c به پیکربندی e منتقل کن. دقت کنید که در ابتدا باید w را برابر با ۱- t فرض کنیم تا اینکه قانون (۱) سر هر واحد زمانی قابل اعمال باشد. اعمال این قانون را می‌توان تا زمانی که به پیکربندی مورد نظر نرسیده‌ایم ادامه دهیم.

[۴] برای مثال با استفاده از عددی طبیعی که از بدو تاریخ در هر واحد زمانی، یک واحد افزایش یافته است!

۲- نحو زبان توصیف ماشین‌های سلولی

در این بخش ابتدا قصد داریم نحو صوری زبان توصیف ماشین‌های سلولی که در این پروژه ارائه داده‌ایم را شرح دهیم، سپس طی مثال‌هایی زبان توصیف را آموزش خواهیم داد.

۲-۱- نحو صوری زبان

قواعد BNF زیر نحو صوری زبان توصیف ماشین‌های سلولی را تعریف می‌کند. برای سادگی نمایش و خوانا بودن نحو مجرد زبان را ارائه داده‌ایم. همچنین، توضیح مختصری در مورد هر یک از متغیرهای زبان ارائه داده‌ایم.

متغیر	تعریف	توضیحات
<*MACHINE*> ::=	**machine** <*IDENTIFIER*> **dim** <*INT_TUPLE*> **states** <*IDEN_TUPLE*> **transition** <*FUNC*> **final** <*EXPR*> **endmachine**	تعریف ماشین سلولی
<*INT_TUPLE*> ::=	<*NUMERAL*> | <*INT_TUPLE*>**,** <*NUMERAL*>	چندتایی اعداد صحیح
<*IDEN_TUPLE*> ::=	<*IDENTIFIER*> | <*IDEN_TUPLE*>**,** <*IDENTIFIER*>	چندتایی شناسه‌ها
< *FUNC*> ::=	<*STMT*>	بدنه توابع

<*STMT*> ::=	**{** <*STMT_LIST*> **}**
	| <*EXPR*> **:=** <EXPR>
	| **if** <EXPR> **then**
	<STMT>
	else
	<STMT>
	endif
	| **while** <EXPR> **do**
	<STMT>
	endwhile
	| **skip**
	| **return** <EXPR>
		دستورات

<*STMT_LIST*> ::=	<*STMT_LIST*>**;** <*STMT*> | <*STMT*>	دنباله دستورات
<*EXPR*> ::=	<*EXPR*> **+** <*EXPR*> | <*EXPR*> **-** <*EXPR*> | <*EXPR*> ***** <*EXPR*> | <*EXPR*> **/** <*EXPR*> | <*EXPR*> **&** <*EXPR*> | <*EXPR*> **|** <*EXPR*> | <*EXPR*> **=** <*EXPR*> | <*EXPR*> **<** <*EXPR*> | <*EXPR*> **>** <*EXPR*> | **(** <*EXPR*> **)** | **~** <*EXPR*> | **-** <*EXPR*> | **snn (** <*EXPR*> **)**	عبارت‌ها

	smn (<*EXPR*> **)** **this.**<*NUMERAL*> **tick** <*VARIABLE*> <*NUMERAL*> <*IDENTIFIER*>	
<*VARIABLE*> ::=	**a** | **b** | **c** | … | **z**	اسامی متغیرها
<*IDENTIFIER*> ::=	<*VARIABLE*>(<*VARIABLE*>|<*NUMERAL*>)$^{+}$	شناسه‌ها
<*NUMERAL*> ∈ ℕ		اعداد

لازم به ذکر است که در این قوانین نمادهای مورد استفاده در متن برنامه بصورت پر رنگ نشان داده شده است، همچنین بالانویس «+» به معنای تکرار به تعداد حداقل یک بار است. با توجه به این، طول شناسه‌های مورد استفاده در توصیف‌ها حتماً باید بیشتر دو کاراکتر باشد.

۲-۲- آموزش زبان توصیف ماشین‌های سلولی

در زیربخش قبل، نحو زبان توصیف ماشین‌های سلولی را به‌صورت صوری توصیف کردیم. در این زیربخش قصد داریم با توضیح غیر صوری‌تر نحو زبان، شیوه استفاده از آن را آموزش دهیم. قبل از اینکه وارد بحث شویم لازم به ذکر است که کامپایلر این زبان برای سیستم‌عامل لینوکس (و تمام سیستم‌عامل‌های شبه یونیکس) طراحی شده است. توصیف‌های این زبان در داخل یک فایل متنی نوشته می‌شوند و می‌توان از طریق ترمینال سیستم آن‌ها را کامپایل کرد. دستور کامپایل کردن یک توصیف ماشین سلولی به‌صورت زیر است:

```
$ cac file.ca
```

در این دستور، cac کوتاه شده CA Compiler بوده و برنامه کامپایلر زبان توصیف ماشین‌های سلولی است. فایل ورودی با نام file.ca نشان داده شده است. در حالت کلی این نام مهم نیست و می‌توان از هر فایل متنی حاوی توصیف ماشین‌های سلولی استفاده کرد. در صورتی که فایل وردی فاقد خطی‌های نحوی و یا معنایی باشد، یک فایل با پسوند C در پوشه جاری تولید می‌شود (اسم این فایل همان اسم ماشین سلولی است که در توصیف ماشین انتخاب شده است). این فایل در اصل برنامه‌ایی به زبان برنامه‌نویسی C است که ماشین سلولی توصیف شده را پیاده‌سازی می‌کند. فایل خروجی بعداً می‌تواند توسط کامپایلر C به فایل اجرایی ترجمه شود.

در حالت کلی، ماشین‌های سلولی در قالب بندهایی توصیف می‌شوند. این بندها به نحوی با پارامترهای شناساننده ماشین‌های سلولی که در بخش اول ارائه دادیم متناظر هستند. طبق نحو صوری که در زیربخش قبل ارائه دادیم، بندها باید طبق ترتیب خاصی نوشته شوند که در اینجا نیز آن‌ها را با همان ترتیب شرح می‌دهیم. هر توصیف با یک بند machine شروع می‌شود، در این بند می‌بایست نام ماشین سلولی را بنویسیم. نام ماشین برای تعیین نام فایل خروجی کامپایلر مورد استفاده قرار می‌گیرد. بند dim برای تعیین ابعاد ماشین سلولی مورد استفاده قرار می‌گیرد. در این بند ابعاد ماشین را با استفاده از یک چندتایی از اعداد طبیعی تعریف می‌کنیم. بند states برای تعریف مجموعه حالات هر سلول مورد استفاده قرار می‌گیرد. هر

حالت یک شناسه است که بعد از تعریف در سیستم به عنوان یک ثابت از نوع State تعریف می‌شود. بنابراین، نمی‌توان مقداری را در طول محاسبات به آن‌ها منتسب کرد یا اینکه در محاسبات اعداد صحیح از آن‌ها استفاده کرد. هر یک از سلول‌های ماشین توصیف شده، در هر لحظه می‌توانند در یکی از این حالات قرار بگیرند. بند transition برای تعریف قانون محلی ماشین سلولی مورد استفاده قرار می‌گیرد. در این بند مجموعه دستوراتی نوشته می‌شود که قانون محلی ماشین مورد نظر را تعریف می‌کند. مجموعه دستورات بند transition به ازای تک تک سلول‌های ماشین اجرا می‌شوند. دستورات بند transition در هر تیک ساعت برای تغییر پیکربندی ماشین اجرا می‌شوند. در مورد این دستورات در ادامه بحث خواهیم کرد. بند final یک عبارت را تعریف می‌کند که مشخص کننده شرط خاتمه محاسبات است. برنامه‌ایی که برای پیاده‌سازی ماشین سلولی تولید می‌شود تا زمانی که این شرط بر قرار نشده باشد، تغییر پیکربندی را ادامه می‌دهد. نهایتاً توصیف ماشین سلولی با بند endmachine خاتمه می‌یابد. بعد از این بند نیز نباید هیچ نمادی به جز انتهای فایل نوشته شود. در شکل زیر قالب کلی یک فایل توصیف ماشین سلولی نشان داده شده است.

machine
نام ماشین سلولی
dim
$n_1, n_2, \dots, n_d$
states
$i_1, i_2, \dots, i_s$
transition
دنباله دستورات برای تعریف قانون محلی ماشین
final
عبارتی که شرط پایان محاسبات را مشخص می‌کند
endmachine

شکل ۲-۱- قالب کلی فایل توصیف ماشین‌های سلولی.

دنباله دستورات زبان توصیف ماشین‌های سلولی که در بند transition استفاده می‌شود، می‌تواند با طول یک و یا بیشتر از یک باشد. دنباله دستورات با طول بیشتر از یک را در داخل کروشه قرار می‌دهیم و دستورات داخل کروشه را به نماد نقطه ویرگول «;» از هم جدا می‌کنیم.

یک دستور می‌تواند انتساب یک مقدار به یک متغیر باشد. همانطور که از تعریف صوری نحو زبان می‌توان فهمید، متغیرها در این زبان فقط از حروف یک حرفی تشکیل شده‌اند. بنابراین، تنها از ۲۶ متغیر می‌توانیم در توصیف‌های استفاده کنیم. لازم به ذکر است که این محدودیت در مورد تعداد متغیرها به دلیل سادگی قوانین محلی ماشین‌ها محسوس نخواهد بود. متغیرها در این زبان، همگی از نوع اعداد صحیح هستند. بنابراین، انتساب یک مقدار غیر صحیح به متغیرها موجب تولید خطا از سوی کامپایلر خواهد شد. در حالت کلی دستور انتساب بصورت زیر نوشته می‌شود:

v := ex

در این دستور v نام یک متغیر و ex یک عبارت با مقدار صحیح است.

دستور if … then … else … endif برای تعریف بلوک‌های شرطی مورد استفاده می‌شود. برای سادگی در طراحی زبان، فرض کرده‌ایم که هر دستور if باید به همراه else و endif بیاید. در حالت کلی به جای علامت‌های … در قالب کلی ذکر شده، می‌توان دستور و یا دنباله‌ایی از دستورات را بنویسیم. برای مثال، دستور زیر یک بلوک شرطی است که به شرط مثبت بودن جمع متغیرهای a و b میانگین آن‌ها را حساب می‌کند و در غیر این صورت ماکزیمم a و b را محاسبه کرده و در متغیر m قرار می‌دهد.

```
if a+b > 0 then
    m := (a+b) / 2
else {
    t := a;
    if t > b then m := t
    else m := b
}
```

همان‌طور که مشاهده می‌شود، دنباله دستورات را در داخل نماد کروشه و آن‌ها را با استفاده از نماد نقطه ویرگول، دو به دو از هم جدا می‌کنیم. این دستورات یکی پس از دیگری اجرا می‌شوند.

دستور while … do … endwhile برای تعریف بلوک تکرار دستورات (حلقه) مورد استفاده قرار می‌گیرد. اولین نماد ... در قالب دستور مکان یک عبارت منطقی است، و نماد ... بعدی مکان دستور و یا دنباله‌ایی از دستورات است که می‌خواهیم به تعداد مورد نظر تکرار شود. لازم به ذکر است که در این زبان می‌توان حلقه‌هایی را توصیف کرد که هیچ‌گاه خاتمه پیدا نمی‌کنند. این موضوع بر قدرت زبان می‌افزاید، ولی در مورد زبان توصیف ماشین‌های سلولی، این قدرت آن‌چنان که باید مفید نیست. در صورتی که بخواهیم دقیق‌تر بحث کنیم، می‌توان گفت از آنجایی که تابع قانون محلی باید یک تابع Total باشد، می‌بایست در توصیف آن حتماً از حلقه‌هایی استفاده کنیم که خاتمه پیدا می‌کنند. برنامه زیر، می‌تواند برای محاسبه فاکتوریل عددی که در متغیر x ذخیره شده است مورد استفاده قرار بگیرد. نتیجه محاسبات این دستورات در متغیر f قرار می‌گیرد.

```
{
    f := 1;
    c := x;
    while c > 0 do {
        f := f * c;
        c := c - 1
    }
    endwhile
}
```

دستور skip برای توصیف عمل خنثی مورد استفاده قرار می‌گیرد. این دستور هیچ اثر و یا مقدار برگشتی ندارد، و صرفاً برای مواردی از قبیل استفاده در شاخه‌های دستور if مورد استفاده قرار می‌گیرد. برای مثال، فرض کنید می‌خواهید مقدار متغیر f را در صورت کوچکتر بودن آن از ۶ یک واحد افزایش دهید

و در غیر اینصورت مقدار آن بدون تغییر باقی بماند. در این صورت می‌توان برنامه‌هایی مانند برنامه‌های زیر ارائه داد:

if f < 6 **then** f := f + 1 **else skip** یا **if** f >= 7 **then skip** **else** f := f + 1

دستور return در فقط در بند transition کاربرد دارد و برای بر گرداندن نتیجه تابع قانون محلی مورد استفاده قرار می‌گیرد. نوع داده‌ایی که این دستور لازم دارد باید از نوع State باشد. به عبارتی دیگر، در مقابل دستور return باید یکی از ثابت‌هایی که در بند states تعریف کرده‌ایم نوشته شود.

بـعد از معرفی دستورات، نوبت به معرفی عبارت‌ها می‌رسد. دسته‌ایی از عبارت‌ها در زبان توصیف ماشین‌های سلولی برای محاسبات اعداد صحیح مورد استفاده قرار می‌گیرند. در حالت کلی طبق تعریف صوری نحو زبان، در صورتی که e_1 و e_2 عبارت‌هایی از زبان توصیف ماشین‌های سلولی باشند، آنگاه $e_1\ op\ e_2$ نیز یک عبارت از این زبان است، که در آن op یکی از چهار عمل اصلی محاسبات است. همچنین، اگر e یک عبارت باشد، (e) نیز یک عبارت است. عملگرهای منطقی نیز در عبارت‌های این زبان می‌توانند ظاهر شوند. در این زبان سه عملگر منطقی AND، OR، و NOT تعریف کرده‌ایم که به ترتیب با استفاده از نمادهای &، |، و ~ نمایش داده می‌شوند. سه عملگر برای مقایسه عبارات در زبان تعبیه کرده‌ایم. این عملگرها برای بررسی تساوی، بزرگتر بودن، و کوچکتر بودن به‌کار می‌روند. برای انجام سایر مقایسه‌ها مانند عدم تساوی و کوچکتر بودن غیر اکید، می‌بایست از عملگرهای منطقی نیز کمک بگیریم. عملگری را نیز برای منفی کردن اعداد صحیح در نظر گرفته‌ایم. در حالت کلی اگر e یک عبارت باشد، e- نیز یک عبارت است که مقدار آن منفی مقدار e است. توجه کنید که در این زبان هیچ عبارتی از نوع Boolean وجود ندارد، و هر عبارت صحیح غیر صفر دارای ارزش منطقی True و هر عبارت صحیح برابر با صفر دارای ارزش منطقی False است. با این حساب، عملگرهای منطقی مانند AND بر روی نمایش بیتی عملوندها اعمال می‌شوند.

در زبان توصیف ماشین‌های سلولی عملگرهایی برای دسترسی به سلول‌های همسایه سلول جاری وجود دارد. در زبان ارائه شده، صرفاً دو نوع همسایگی مُور و نیومن پیاده‌سازی شده‌اند و آن‌ها را نیز با استفاده از دنباله‌هایی پیاده‌سازی کرده‌ایم. ایده‌ایی که به‌کار برده‌ایم بر اساس تعاریفی است که از این نوع همسایگی‌ها در بخش اول ارائه دادیم. در صورتی که بخواهیم بصورت کلی این دو دنباله را تعریف کنیم، می‌توان گفت دنباله همسایگی‌های نیومن دنباله‌ایی از حالات است که بصورت زیر تعریف می‌شود:

$$snn: \mathbb{N} \nrightarrow State$$

همانطور که مشاهده می‌شود، این دنباله بصورت جزئی تعریف شده است. در اصل دنباله snn دارای طول متناهی است. طول این دنباله ۳d است که در آن d تعداد بعدهای ماشین سلولی است. می‌دانیم که در یک ماشین سلولی d بعدی، می‌توان سلول‌ها را با یک dتایی به‌صورت $(c_0, c_1, \ldots, c_{d-1})$ آدرس‌دهی کرد. تـوجـه کنید که مـؤلفه‌های dتایی بـه ترتیب در دنباله C قرار گرفته‌اند. همچنین فرض کنید دنباله B را به‌صورت B=<-۱, ۰, ۱> تعریف کنیم. علت تعریف دنباله B به تعاریف ارائه شده در بخش اول بر می‌گردد.

دنباله B در اصل دنباله مقادیر ممکن برداری را نشان می‌دهد که یک همسایگی نیومن را تعریف می‌کند (به خاطر بیاورید که بردار $\vec{N}$ یک همسایگی نیومن را تعریف می‌کند اگر نُرم منهتن آن کمتر یا مساوی یک باشد، به عبارت دیگر $\vec{N}$ باید برداری با حداکثر یک مؤلفه غیرصفر باشد). در صورتی که فرض کنیم ورودی دنباله *snn* اعداد طبیعی مانند i است، می‌توان گفت:

«اگر سلول جاری در آدرس $(c_0, c_1, \ldots, c_{d-1})$ قرار داشته باشد، همسایگی iام سلول جاری در آدرس $(n_0, n_1, \ldots, n_{d-1})$ قرار دارد که در آن $n_{i/3} = c_{i/3} + B(i \bmod 3)$ و به ازای هر $j \neq i/3$ داریم $n_j = c_j$.»

برای مثال، در ماشین سلولی زیر همسایگی نیومن سلول f با رنگ خاکستری مشخص شده است.

a	b	c	d
e	f	g	h
i	j	k	l
m	n	o	p

با فرض اینکه حالت یک سلول s با نماد σ_s نمایش داده می‌شود، دنباله snn برای سلول f بصورت زیر تعریف می‌گردد:

$$snn = \{0 \mapsto \sigma_e, 1 \mapsto \sigma_f, 2 \mapsto \sigma_g, 3 \mapsto \sigma_b, 4 \mapsto \sigma_f, 5 \mapsto \sigma_j\}$$

بـطور مشابه مـی‌توان دنباله حالات همسایگی مُور یک سلول را تعریف کنیم. برای اینکار دنباله‌ایی را به‌صورت زیر تعریف می‌کنیم:

$$smn: \mathbb{N} \nrightarrow States$$

درست همانند دنباله snn، دنباله smn نیز بصورت تابعی جزئی (و نیز با طول متناهی تعریف می‌شود). بـرای تعریف بـدنه این تـابع، ابتدا فـرض می‌کنیم که عدد طبیعی i ورودی دنباله است. همچنین فرض می‌کنیم dتایی $(c_0, c_1, \ldots, c_{d-1})$ آدرس سلول جاری در ماشین است، و دنباله $D_i = \langle d_0, d_1, \ldots, d_{d-1} \rangle$ نمایش dرقمی i در مبنای ۳ است. در دنباله D_i، مقدار $D_i(0)$ با ارزش‌ترین رقم عدد را نشان می‌دهد. در این صورت می‌توان گفت:

« همسایگی iام سلول جاری در آدرس $(n_0, n_1, \ldots, n_{d-1})$ قرار دارد، که در آن $n_j = B(d_j) + c_j$. در این فرمول دنباله B همان دنباله‌ایی است که پیش‌تر تعریف شد.»

طبق این تعریف می‌توان گفت دنباله smn برای یک ماشین d بعدی، دارای طول 3^d است.

برای مثال، در ماشین سلولی زیر همسایگی مُور سلول f با رنگ خاکستری مشخص شده است. همانند مثال قبل، فرض فرض کنید که حالت یک سلول s با نماد σ_s نمایش داده می‌شود، دنباله smn برای سلول f بصورت زیر تعریف می‌گردد:

a	b	c	d
e	f	g	h
i	j	k	l
m	n	o	p

$$smn = \{0 \mapsto \sigma_a, 1 \mapsto \sigma_e, 2 \mapsto \sigma_i, 3 \mapsto \sigma_b, 4 \mapsto \sigma_f, 5 \mapsto \sigma_j, 6 \mapsto \sigma_c, 7 \mapsto \sigma_g, 8 \mapsto \sigma_k\}$$

عبارت this.i که در آن i عددی طبیعی و $0 \leq i \leq d-1$ است، برای دسترسی به مختصات سلول جاری مورد استفاده قرار می‌گیرد. از این عبارت در فاز مقداردهی اولیه (اولین تیک ساعت) استفاده می‌شود. در صورتی که ($c_0, c_1, \ldots, c_{d-1}$) آدرس سلول جاری باشد، this.i = c_i است. در صورتی که i بیشتر از d - 1 باشد که در آن d تعداد بعدهای ماشین است، خطایی از سوی کامپایلر صادر خواهد شد.

آخرین عبارتی که لازم است توضیح داده شود، عبارت tick است که تیک ساعت جاری را نشان می‌دهد. هر بار که دستورات بند transition به ازای تمامی سلول‌های ماشین اجرا شد، مقدار tick یک واحد افزایش پیدا می‌کند. اولین بار، مقدار tick صفر است و با استفاده از این مقدار می‌توان فاز مقداردهی اولیه را تشخیص داد. کاربرد دیگر عبارت tick در تعیین شرط بند final است که با استفاده از آن می‌توان گفت که می‌خواهیم محاسبات ماشین بعد از چند تیک ساعت متوقف شود. عبارت tick را مقدار ثابتی تعریف کرده‌ایم و هر گونه مقدار دهی به این عبارت با ممانعت از سوی کامپایلر همراه است.

نکته‌ایی که در پایان باید به آن اشاره کنیم، این است که در صورتی که در تیک صفر هیچ سلولی به‌صورت مستقیم مقداردهی اولیه نشود، سلول‌ها به‌طور خودکار با اولین حالت ذکر شده در بند states مقداردهی خواهند شد. با توجه به اینکه دستورات و عبارات زبان توصیف ماشین‌های سلولی شرح داده شد، بهتر است چند نمونه از توصیف ماشین‌های سلولی در این زبان را مشاهده کنیم.

مثال: بازی زندگی (یا Game of Life) یک ماشین سلولی دوبعدی سنکرون است که در سال ۱۹۷۰ توسط یک ریاضیدان انگلیسی معرفی شد. بازی زندگی یک بازی صفر نفره است، به این معنا که ماشین سلولی پیاده‌سازی کننده آن به هیچ ورودی احتیاج ندارد. ماشین سلولی برای بازی زندگی باید یک ماشین نامتناهی باشد، ما در این مثال آن را با استفاده از یک ماشین متناهی بزرگ توصیف می‌کنیم.

بازی زندگی قوانین ساده‌ایی دارد، این قوانین به شرح زیر است:

- هر سلول زنده با کمتر از دو سلول زنده در همسایگی آن، می‌میرد.
- هر سلول زنده با دو یا سه همسایه زنده، زنده باقی می‌ماند.

- هر سلول زنده با بیش از سه همسایه زنده، می‌میرد.
- هر سلول مرده با دقیقا سه همسایه زنده، دوباره زنده می‌شود.

در این مثال فرض می‌کنیم ماشین در ابتدا در یک حالت اولیه مطلوب قرار دارد. با توجه به قانون‌های ذکر شده، در می‌یابیم که سلول‌های این ماشین می‌توانند در دو حالت زنده یا مرده قرار داشته باشند که ما این دو حالت را به ترتیب با alive و dead نشان می‌دهیم. این ماشین سلولی را در زبان توصیف ماشین‌های سلولی به‌صورت زیر توصیف می‌کنیم:

```
machine
    GameOfLife
dim
    100, 100
states
    dead, alive
transition
{
    i := 0;
    l := 0;
    while i < 9 do {
        if ~(i = 5) & smn (i) = alive then
            l := l + 1
        else skip
        endif;
        i := i + 1
    }
    endwhile;
    if smn (5) = alive then
        if l < 2 then return dead
        else if l = 3 then return alive
        else return dead
        endif
        endif
    else
        if l = 3 then return alive
        else return dead
        endif
    endif
}
final
    tick = 100
endmachine
```

شکل ۲-۲- توصیف بازی زندگی در زبان توصیف ماشین‌های سلولی.

دقت کنید که در این مثال، ماشین را طوری طراحی کرده‌ایم که ۱۰۰ بار قانون محلی تعریف شده را اجرا کند. در صورتی که توصیف ارائه شده در شکل ۲-۲ را در یک فایل متنی بنویسیم و توسط برنامه cac کامپایل کنیم، برنامه C که در صفحات بعد ارائه داده‌ایم بطور خودکار تولید خواهد شد.

```
/*Data-type definitions*/
typedef enum _state_t {alive, dead} state_t;
typedef state_t array_t [100][100];

/*Private data information*/
static int turn = 0;
static array_t ca [2] = {{0}, {0}};
static const int base [3] = {-1, 0, 1};
static int var_pool [16] = {0};
static int tick = 0;

/*The only globally available function*/
void compute (array_t* out);
/*Private functions declarations*/
static int final ();
static void do_transition ();
static state_t transition (int c0, int c1);
static state_t get_cell (int c0, int c1);
static void set_cell (int c0, int c1, state_t st);
static state_t snn (int c0, int c1, int ndex);
static state_t smn (int c0, int c1, int ndex);

/*The implementation*/
state_t get_cell (int c0, int c1) {
        c0 = c0 < 0 ? c0 + 100 : c0 % 100;
        c1 = c1 < 0 ? c1 + 100 : c1 % 100;

        return ca [turn] [c0][c1];
}

void set_cell (int c0, int c1, state_t st) {
        c0 = c0 < 0 ? c0 + 100 : c0 % 100;
        c1 = c1 < 0 ? c1 + 100 : c1 % 100;

        ca [(turn + 1) % 2] [c0][c1] = st;
}

state_t snn (int c0, int c1, int ndex) {
        int n [2] = {c0, c1};

        n [ndex / 3] += base [ndex % 3];
        return get_cell (n [0], n [1]);
}

state_t smn (int c0, int c1, int ndex) {
        int n [2] = {c0, c1};
        int bvec [2];
        int len;
        int i;

        for (len = 0; ndex >= 3; len ++, ndex /= 3)
                bvec [len] = base [ndex % 3];
        bvec [len] = base [ndex];
        for (i = 0; len >= 0; i ++, len --)
                n [i] += bvec [len];
        return get_cell (n [0], n [1]);
}

state_t transition (int c0, int c1) {
        var_pool [8] = 0;
        var_pool [11] = 0;
        while (var_pool [8] < 9) {
                if (! var_pool [8] == 5 && smn (c0, c1, var_pool [8]) == alive) {
                        var_pool [11] = var_pool [11] + 1;
                } else {
                }
                var_pool [8] = var_pool [8] + 1;
        }
        if (smn (c0, c1, 5) == alive) {
                if (var_pool [11] < 2) {
                        return dead;
                } else {
                        if (var_pool [11] == 3) {
                                return alive;
                        } else {
                                return dead;
```

```
                        }
                }
        } else {
                if (var_pool [11] == 3) {
                        return alive;
                } else {
                        return dead;
                }
        }

        return (state_t) 0; /*default behavior, to be safe!*/
}

void do_transition () {
        int c0;
        int c1;

        for (c0 = 0; c0 < 100; c0 ++)
                for (c1 = 0; c1 < 100; c1 ++)
                        set_cell (c0, c1, transition (c0, c1));
        turn = (turn + 1) % 2;
}

int final () {return tick == 100;}

void compute (array_t* out) {
        while (!final ()) {
                do_transition ();
                tick ++;
        }
        memcpy (out, &ca [turn], sizeof (array_t));
}
```

شکل ۲-۳- فایل **GameOfLife.c** که توسط کامپایلر تولید شده است.

همانطور که مشاهده می‌شود، این فایل فاقد تعریف تابع main است. چرا که، این فایل در اصل باید یک ماژول از یک برنامه بزرگتر باشد. با مشاهده شکل ۲-۳ متوجه می‌شویم که تنها یک تابع عمومی در این ماژول تعریف شده است. تابع compute در اصل واسط ماشین سلولی با دنیای بیرون است، این تابع پارامتری دارد که از طریق آن می‌توان از پیکربندی نهایی ماشین اطلاع حاصل کرد.

۳- معناشناخت صوری زبان

در بخش اول معناشناخت عملیاتی زبان را بصورت صوری تعریف کردیم. در انتهای این بخش اشاره‌ایی به نحوه پیاده‌سازی این قانون معناشناخت بحث خواهیم کرد. در این بخش بطور عمده، بر روی معناشناخت صوری استاتیک زبان مطالبی را ارائه خواهیم داد.

۳-۱- معناشناخت استاتیک زبان

در زبان توصیف ماشین‌های سلولی، سه نوع[۵] وجود دارد. نوع‌های این زبان عبارت‌اند از نوع داده‌های صحیح، نوع داده‌های حالت، و نوع تک‌عضوی به ترتیب با Integer، State، و Unit می‌شناسیم. در یک نگاه غیرصوری می‌توان گفت متغیرهای a تا z، اعداد، نتیجه عملگرهای محاسباتی و منطقی، و نتیجه عملگر this از نوع Integer است. یک دستور و نیز دنباله‌ایی از دستورات از نوع تابعی به‌صورت Unit→Unit است. بنابراین، اجرای بند transition به معنای اعمال نمودن تابع توصیف شده در بند transition بر روی مقدار unit است. نتیجه اجرای هر دستور برای ما مهم نیست برای همین آن را از نوع Unit انتخاب کرده‌ایم. تنها چیزی که از یک دستور برای ما اهمیت دارد این است که آیا محاسبات توصیف شده توسط آن خاتمه یافته است یا خیر. به عبارتی بهتر، در مورد دستورات آثار جانبی دستور بیشتر برای ما اهمیت دارد تا مقدار برگشتی. نهایتاً، هر ثابت فهرست شده در بند states از نوع State است. سیستم‌نوع زبان، که در ادامه ارائه خواهیم داد، اجازه ترکیب داده‌ها از نوع‌های مختلف و ناسازگار را نمی‌دهد.

معناشناخت استاتیک (سیستم‌نوع) زبان توسط قوانین زیر تعریف می‌شوند:

$$\frac{n \in \mathbb{N}}{n: Integer} \quad (۲)$$

$$\frac{o: Integer}{-o: Integer} \qquad \frac{o: Integer}{\sim o: Integer} \quad (۳)$$

$$\frac{\begin{array}{c} o_1: Integer \\ o_2: Integer \\ op\ is\ either\ +, -, *, /, <, >, =, \&, or\ | \end{array}}{o_1 op\ o_2: Integer} \quad (۴)$$

$$\frac{\begin{array}{c} l: Integer \\ r: Integer \\ l\ is\ a\ variable \end{array}}{l := r : Unit \to Unit} \quad (۵)$$

$$\frac{\begin{array}{c} g: Integer \\ t: Unit \to Unit \\ e: Unit \to Unit \end{array}}{if\ g\ then\ t\ else\ e\ endif: Unit \to Unit} \qquad \frac{\begin{array}{c} g: Integer \\ w: Unit \to Unit \end{array}}{while\ g\ do\ w\ endwhile: Unit \to Unit} \quad (۶)$$

[۵] Type

$$\frac{s_1: Unit \to Unit \quad s_2: Unit \to Unit}{s_1; s_2: Unit \to Unit} \tag{۷}$$

$$\frac{v: State}{return\ v: Unit \to Unit} \qquad \frac{}{skip: Unit \to Unit} \tag{۸}$$

$$\frac{p: Integer}{snn(p): State} \qquad \frac{p: Integer}{smn(p): State} \tag{۹}$$

$$\frac{p \in \mathbb{N} \quad 0 \le p < d}{this.p: Integer} \tag{۱۰}$$

$$\frac{}{tick: Integer} \tag{۱۱}$$

همانطور که از قوانین و اصل‌های (۲) تا (۱۱) می‌توان دریافت، زبان راهی برای ساخت نوع Unit→Unit وجود ندارد، اما این نوع در حین بررسی نوع تولید می‌شود. هرچند، می‌توانستیم به جای Unit→Unit از یک نوع ساختگی دیگر مثالاً نوعی با نام Statement استفاده کنیم. اما، با انتخاب نوع تابعی توجیه دقیق و صوری از اجرای یک دستور را می‌توان ارائه داد. نکته‌ایی که لازم است ذکر کنیم، این است که در قانون (۱۹) حرف d نشان دهنده تعداد بعدهای ماشین است.

با توجه به قوانین و اصول (۲) تا (۱۱) ویژگی‌های جالب توجهی را می‌توان اثبات کرد. اما، در این پروژه و گزارش هدف اثبات و یا پرداختن به این ویژگی‌ها نیست.

۳-۲- ترجمه کد به زبان C

در بخش ۱، معناشناخت عملیاتی زبان توصیف ماشین‌های سلولی را با ارائه یک قانون به‌طور صوری تعریف کردیم. ترجمه کد به زبان برنامه‌سازی C نیز بر اساس این قانون انجام می‌شود. مطابق قانون (۱) در بخش ۱، یک تابع ماشین را از یک پیکربندی به پیکربندی دیگر منتقل می‌کند، این تابع در بند transition ماشین توصیف می‌شود. بخشی از سیستم مسئول اعمال قانون محلی بر روی تک تک سلول‌ها و افزایش مقدار متغیر tick است، این بخش بعد از دور اعمال قانون محلی، برقرار بودن شرط ذکر شده در بند final را نیز بررسی می‌کند. در صورتی که شرط final برقرار باشد، محاسبات پایان یافته و ماشین متوقف می‌شود.

بعد از اینکه درخت نحوی توصیف ورودی ساخته شد، تابعی این درخت را پیمایش کرده و مطابق آن کدهای لازم را تولید می‌کند. برای بدست آوردن اطلاعات بیشتر می‌توان به CD همراه گزارش رجوع کرد.

۴- نتیجه‌گیری و کارهای آینده

گزارش حاضر، حاوی شرح کامپایلر و زبان طراحی شده برای زبان توصیف ماشین‌های سلولی است. زبان توصیف ماشین‌های سلولی برای سیستم‌های لینوکس پیاده‌سازی و تست شده است. از این زبان می‌توان برای توصیف ماشین‌های سلولی متناهی و سنکرون استفاده کرد. در کل، استفاده از کامپایلر به این صورت است که توصیف ماشین سلولی در یک فایل متنی نوشته می‌شود و به کامپایلر پاس می‌شود، در صورتی که توصیف فاقد هر گونه خطا باشد، کد C معادل با آن توصیف تولید می‌شود.

در این گزارش، در بخش ۱، تعاریف اصلی ماشین‌های سلولی و معناشناخت عملیاتی این ماشین‌ها را تعریف کردیم. در بخش ۲، ضمن آموزش استفاده از زبان و کامپایلر پیاده‌سازی شده، نحو صوری زبان را ارائه دادیم. در بخش ۳، معناشناخت صوری زبان را تعریف کردیم، و نهایتاً اشاره‌ایی به نحوه ترجمه توصیف‌های زبان توصیف ماشین‌های سلولی به برنامه C را شرح دادیم.

برای بهبود کیفیت این زبان می‌توان آن را بگونه‌ایی طراحی کرد که Safe باشد، و نیز Safe بودن آن را نیز اثبات کرد. همچنین خوب است نحو زبان به طوری تعریف شود که امکان نوشتن توضیحات در متن توصیف‌ها موجود باشد. به این ترتیب، توصیف‌های خواناتری را می‌توان نوشت.

در پایان لازم به ذکر است که به همراه این گزارش لوح فشرده‌ایی ارائه شده است که حاوی کد منبع کامپایلر و نیز چند نمونه توصیف نمونه است.

۵- منابع

[۱] B. T. Widemann, “Structural Operational Semantics for Cellular Automata,” Lecture Notes in Computer Science, vol. 7495, pp. 184-193, 2012.