محیطی برای تولید برنامه از توصیف صوری

سید حسن‌میریان حسین‌آبادی (استادیار)
آرش جلالی (کارشناس ارشد)
دانشکده‌ی مهندسی کامپیوتر، دانشگاه صنعتی شریف

تولید نرم‌افزار به‌روش‌های سنتی با مشکلاتی نظیر عدم اطمینان از درستی عملکرد آن همراه است. استفاده از روش‌های صوری برای توصیف سیستم‌های نرم‌افزاری موجب کاهش این عدم اطمینان خواهد شد. از سوی دیگر، خلاء موجود بین فازهای توصیف و تولید برنامه باعث شده تا به اطمینان از درستی عملکرد، در مرحله‌ی تبدیل توصیف به برنامه خدشه وارد شود. روش‌های مختلفی برای رسیدن به برنامه از توصیف صوری پیشنهاد شده که استخراج برنامه از اثبات درستی توصیف یکی از این روش‌هاست. در این نوشتار، محیطی برای تولید نرم‌افزار مطمئن معرفی و پیشنهاد می‌شود که مبنای کار آن استخراج برنامه از توصیف صوری نرم‌افزار است. توصیف‌های صوری در قالب نسخه‌یی از زبان Z، با علامت اختصاری CZ، نوشته می‌شوند و اثبات درستی این توصیف‌ها به اثبات درستی در نظریه‌ی انواع مارتین لوف ترجمه می‌گردند. در نهایت برنامه از درخت اثبات ترجمه شده درستیِ تأیید شده استخراج می‌شود.

مقدمه

طی ۵۰ سال گذشته، روش‌های تولید نرم‌افزار، به ویژه روش‌های تولید برنامه به‌عنوان مؤلفه‌یی اساسی از نرم‌افزار، دچار دگرگونی بنیادین شده‌اند. این دگرگونی اساساً از تغییرات در ماهیت کاربردی نرم‌افزار و نیازهای کاربران ناشی شده است.[۱] به علاوه، تغییرات به افزایش پیچیدگی نرم‌افزارها انجامیده است. تغییر در پیچیدگی نرم‌افزارها را می‌توان در طیفی از کاربردهای عمده، از سیستم‌های پردازش دسته‌یی[۱] اوایل دهه‌ی ۵۰ میلادی تا سیستم‌های خبره و پردازش موازی دهه‌ی ۹۰ مشاهده کرد. از طرف دیگر، افزایش قدرت پردازش رایانه‌ها و نیز کاهش قیمت آن‌ها به این رشد فزاینده در پیچیدگی نرم‌افزارها کمک کرده است.

با این وجود، رشد روش‌های تولید نرم‌افزار با تغییرات در پیچیدگی نرم‌افزار هماهنگ نبوده است.[۲] زیرا لازمه‌ی هماهنگ کردن رشد این روش‌ها با نرخ تغییرات فوق، دگرگونی در همه‌ی جوانب از جمله شیوه‌ی برخورد آنها با مسئله‌ی قابلیت اطمینان و درستی است.

یکی از مشکلات عمده‌ی تولیدکنندگان نرم‌افزارها حصول اطمینان از درستی نرم‌افزار ـ میزان انطباق عملکرد نرم‌افزار بروظیفه‌ی مورد انتظار از آن ـ است. یکی از متداول‌ترین روش‌های موجود برای بررسی درستی نرم‌افزار، آزمون است که ساده‌ترین آنها آزمون جعبه‌ی سیاه می‌باشد که طی آن رفتار (خروجی) نرم‌افزار در قبال ورودی‌های معین بررسی می‌شود.[۱و۳]

اگر چه در حال حاضر، آزمون نرم‌افزار کاربردی‌ترین و عملی‌ترین روش برای بررسی درستی آن نرم‌افزار است، ارنست دیکسترا ایرادی اساسی بر آزمون، و به طور کلی روش‌های سنتی تولید نرم‌افزار قائل است:

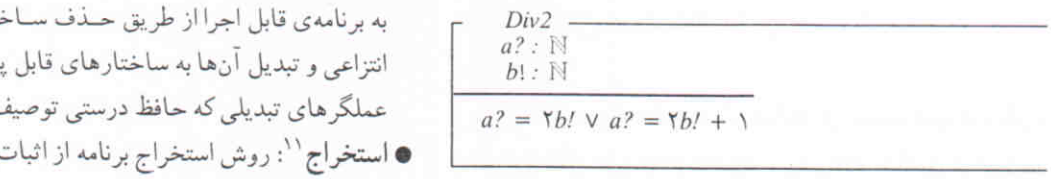
«آزمون برنامه را می‌توان برای نشان‌دادن وجود خطا در آن به کار گرفت ولی هرگز نمی‌توان از آن برای اثبات عدم حضور خطا استفاده کرد.»[۴]

سه مرحله‌ی کلی توصیف نیازها[۲]، طراحی و پیاده‌سازی تقریباً بین تمامی روش‌های تولید نرم‌افزار مشترک‌اند و بنابراین منطقی به نظر می‌رسد که ریشه‌ی خطاهای موجود در نرم‌افزار، در این مراحل جستجو شود. آزمون برنامه، فعالیتی است که همواره پس از مرحله‌ی پیاده‌سازی انجام می‌شود و بنابراین خطاهای مربوط به هر سه مرحله‌ی فوق در این مرحله آشکار می‌شوند. در روش‌های سنتی، برای کاهش خطاهای انسانی در دو مرحله‌ی اول نمی‌توان کار زیادی انجام داد؛ گرچه با به‌کارگیری ابزارهای یاریگر مهندسی نرم‌افزار[۳] به‌طور محدود می‌توان جلوی برخی از ناسازگاری‌های مرحله‌ی طراحی را گرفت.

روش‌های صوری تولید نرم‌افزار یکی از راه‌حل‌های پیشنهادی برای کاهش بروز خطا، به‌ویژه در دو مرحله‌ی اول از مراحل سه‌گانه‌ی فوق است.

روش‌های صوری تولید نرم‌افزار

روش صوری تولید نرم‌افزار روشی است برای تولید سیستم‌های نرم‌افزاری به‌کمک ریاضیات.[۱،۳و۵] در روش صوری، مرحله‌ی



شکل ۱. توصیف برنامه‌ی تقسیم عدد طبیعی ورودی بر ۲ به زبان Z.

توصیف عملکرد[۴] نیز به مراحل سه‌گانه‌ی توصیف نیازها، طراحی و پیاده‌سازی اضافه می‌شود. در این مرحله آنچه را که باید نرم‌افزار انجام دهد، بیان می‌کنند بی‌آن‌که به چگونگی انجام آن اشاره‌یی شود.[۵و۶]

روش‌های صوری به‌طور معمول، بر پایه‌ی زبان توصیف صوری بیان می‌شوند. مهم‌ترین ویژگی یک روش صوری، دقت و عدم وجود ابهام در توصیفی از نرم‌افزار است که به‌کمک زبان ریاضی موجود در آن می‌توان ارائه داد. در بین زبان‌های توصیف صوری موجود می‌توان به VDM[۷]، Z[۶،۸و۹] و B Method[۱۰] اشاره کرد که زبان Z در بین آن‌ها از پیشینه‌ی غنی‌تری در کاربردهای صنعتی برخوردار است.[۱۱] زبان Z، برمبنای نظریه‌ی مجموعه‌ها و منطق ریاضی بنا شده است. منطق به کاررفته در Z، منطق محمولات مرتبه‌ی اول[۵] است. از جمله موارد کاربرد موفقیت‌آمیز Z، به‌کارگیری آن در طراحی و توصیف نسخه‌یی از نرم‌افزار CICS است.[۶]

شکل ۱ نمونه‌یی از توصیف به زبان Z را نشان می‌دهد. شکل خطی این توصیف به‌صورت $[a?:\mathbb{N}, b!:\mathbb{N} \mid a? = 2b! \vee a? = 2b! + 1]$ است. در اینجا، از ساختار شِما[۶] برای توصیف برنامه‌یی که ورودی خود را، که با علامت(؟) مشخص می‌شود، بر دو تقسیم می‌کند و در خروجی، که با علامت (!) مشخص می‌شود، قرار می‌دهد استفاده شده است. «شِما» که برای توصیف حالت مجرد[۷] و نیز عملیات تغییردهنده‌ی آن به‌کار می‌رود، از ساختارهای اساسی زبان Z به‌عنوان یک زبان توصیف مبتنی بر مدل[۸] است.[۶،۸و۹و۱۲]

یکی از ضعف‌های زبان Z، آن است که میان مرحله‌ی توصیف نیازها و طراحی از یک طرف، و تولید برنامه از طرف دیگر خلاء ایجاد می‌کند. به‌عبارت دیگر، چگونگی تبدیل توصیف شکل ۱ به برنامه‌یی قابل اجرا مورد نظر است. روش‌های گوناگونی را که برای تولید برنامه‌های درست از توصیف صوری پیشنهاد شده است، می‌توان به سه دسته تقسیم کرد:

- **تقلید**[۹]: هر روشی که هدف آن تبدیل مستقیم توصیف به یک برنامه، یا اجرای مستقیم همان توصیف، با استفاده از تعدادی قاعده‌ی موردی است.[۱۳و۱۴]
- **پالایش**[۱۰]: روش تبدیل تدریجی (قدم به قدم) یک توصیف صوری به برنامه‌ی قابل اجرا از طریق حذف ساختارهای غیرقطعی و انتزاعی و تبدیل آن‌ها به ساختارهای قابل پیاده‌سازی به کمک عملگرهای تبدیلی که حافظ درستی توصیف‌اند.[۶]
- **استخراج**[۱۱]: روش استخراج برنامه از اثبات ساختاری درستی توصیف صوری.[۵] بدین منظور، لازم است اصول ریاضی که روش توصیف صوری بر آنها مبتنی است، از خواصی برخوردار باشند که به مجموعه‌ی آنها «ساختیگی»[۱۲] اطلاق می‌شود. به‌عبارت دیگر ریاضیاتی که روش صوری بر آن مبتنی است، نوعی خاص از ریاضیات، به‌نام «ریاضیات ساختی»[۱۳] است که با ریاضیات کلاسیک تفاوت‌هایی دارد.[۱۵-۱۸]

گرچه ریاضیات کلاسیک در بیان صریح و دقیق خواسته‌ها ابزار بسیار مناسبی است در دست‌یابی به این خواسته‌ها کمک چندانی نمی‌کند. در ریاضیات کلاسیک می‌توان وجود بسیاری از عناصر را اثبات کرد، بدون آنکه به‌طور صریح محاسبه شوند یا راهی برای محاسبه‌ی آنها ارائه شود. لذا اگر قرار باشد از اثبات درستی یک توصیف صوری برنامه‌یی استخراج شود، ریاضیات کلاسیک مناسب آن نخواهد بود.

ریاضیات ساختی به‌نوعی از ریاضیات اشاره دارد که در آن، هرگاه اثبات وجود یک شیء (با ویژگی‌هایی خاص) مدنظر باشد، باید نشان داد که چگونه می‌توان آن را یافت.[۱۷] با توجه به این تعریف، می‌توان دریافت که اگر یک توصیف صوری برمبنای یک سیستم صوری ساختی بیان شود، اثبات درستی آن توصیف، حاوی الگوریتم برنامه‌یی خواهد بودکه شرایط آن توصیف را برآورده (پیاده‌سازی) می‌کند.با چنین ایده‌یی، نمونه‌یی ساختی از زبان توصیف Z، با نام CZ معرفی شده است.[۵و۱۹] این زبان، علاوه برفراهم آوردن محیطی مشابه با زبان توصیف Z، قابل ترجمه به نظریه‌ی انواع مارتین‌لوف بوده[۲۰و۲۱] و به این ترتیب مبنایی نظری برای تولید سامانمند[۱۴] برنامه‌ها به زبان‌های تابع را فراهم می‌کند.[۲۲]

در این نوشتار، محیطی جامع برای تولید نرم‌افزار به روش استخراج معرفی، و مبانی نظری آن بیان می‌شود و در ادامه، طرح اولیه‌یی از معماری نرم‌افزاری که چنین محیطی را فراهم می‌آورد بررسی خواهد شد. سپس درباره‌ی بخش‌های توصیف صوری و وارسی این محیط، و نیز نحوه‌ی تعامل آنها با کاربر شرح داده خواهد شد. در پایان، در مورد واحد استخراج برنامه‌ی محیط تولید برنامه‌ی فوق، بحث خواهد شد.

معماری محیط تولید

محیط تولید برنامه‌ی صوری (EFPD)[۱۵] با هدف فراهم ساختن

ابزاری جامع برای تولید نرم‌افزاری مطمئن به روش استخراج پیشنهاد شده، که بخش‌هایی از آن طراحی و پیاده‌سازی شده است و بخش‌هایی از آن نیز در حال طراحی است.

به طور کلی EFPD از سه بخش اصلی تشکیل می‌شود: واحد توصیف، واحد وارسی، و واحد تولید (استخراج) برنامه. برای آشنایی با عملکرد هر یک از این بخش‌ها لازم است روش‌شناسی استخراج برنامه در توصیف صوری که EFPD را می‌توان یک ابزار CASE برای آن دانست تشریح شود. در این روش‌شناسی، کاربر پس از تجزیه و تحلیل مسئله، عملکرد نرم‌افزار مورد نظر را به گونه‌یی از زبان Z به نام CZ توصیف می‌کند. این توصیف در درجه‌ی اول می‌بایست از لحاظ سازگاری انواع و درستی ساختی کنترل شود. پس از حصول اطمینان از عدم وجود تناقض ساختی و انواع در توصیف، در مرحله‌ی بعد باید نسبت به درستی توصیف از لحاظ معنایی اطمینان حاصل کرد. بدین منظور، لازم است که کاربر توصیف را به کمک اصول نظریه‌ی CZ اثبات کند. چنان که پیش‌تر گفته شد، چون در روش استخراج برنامه از اثبات ساختی توصیف استخراج می‌شود، بعد از اثبات درستی، مرحله‌ی استخراج در پیش روست.

محیط توصیف

اولین مرحله در فرایند تولید برنامه، تهیه‌ی توصیف صوری از برنامه است. گفتیم که زبان توصیف CZ[5] که ساختی از زبان توصیف صوری Z است، در این محیط به‌عنوان زبان توصیف در اختیار کاربر قرار دارد. برای تشریح بخش توصیف EFPD، ابتدا لازم است CZ و مبنای نظری آن به‌اختصار معرفی شود.

زبان و نظریه‌ی مجموعه‌های CZ

مهم‌ترین مشخصه‌ی یک دستگاه صوری بنیاد ریاضی آن است و مهم‌ترین ابزار برای بیان صوری مفاهیم ریاضیات کلاسیک، نظریه‌ی مجموعه‌های گئورگ کانتور است که شکل اصل موضوعی آن نخستین بار توسط ارنست زرمِلو[16] ارائه شد. کانتور برای نخستین بار مجموعه را چنین تعریف کرد:[۲۳]

«مجموعه، گردایه‌یی از اشیاء متمایز در ذهن و شهود ماست در قالب یک کلیت.»

این تعریف بیش از حد ضعیف و آزاد است، و به‌همین دلیل است که در اواخر قرن نوزدهم و اوایل قرن بیستم، در نظریه‌ی مجموعه‌های کانتور، پارادوکس‌هایی از قبیل پارادوکس راسل[17] یافت شد.[۲۳] برای قوت بخشیدن به‌این نظریه، یکی از رهیافت‌ها بنا نهادن آن بر اصول موضوعه بود. گونه‌های مختلف اصل موضوعی از نظریه‌ی مجموعه‌ها ابداع شده است که اولین آنها، نظریه‌ی مجموعه‌های زرملو بود.

نظریه‌ی مجموعه‌های زرملو همانند هر سیستم صوری دیگر از دو جزء «منطق صوری» و «اصول موضوعه» تشکیل شده است. منطق صوری به‌کار رفته در آن همان منطق محمولات مرتبه‌ی اول است و اصول موضوعه‌ی آن، که با زبان این منطق بیان شده است، بیانگر مفاهیم مبنایی «مجموعه» و «عضویت» هستند. زبان توصیف Z نیز بر اساس نظریه‌ی مجموعه‌های زرملو است که آن را نظریه‌ی مجموعه‌های Z نیز می‌نامند. به‌دلیل وجود قانون طرد شِق ثالث[18] در نظریه‌ی مجموعه‌های کلاسیک، می‌توان با استفاده از برهان خلف در اثبات‌ها، وجود چیزی را بدون اشاره‌ی مستقیم به شیوه‌ی (الگوریتم) یافتن آن اثبات کرد؛ لذا زبان توصیفی که برمبنای چنین نظریه‌یی بیان شده باشد برای استخراج برنامه مناسب نخواهد بود. به همین دلیل، در EFPD از گونه‌ی ساختی Z با نام نظریه‌ی CZ استفاده می‌شود.[5] در ریاضیات ساختی، برای اثبات وجود یک شیء (با ویژگی‌هایی خاص)، باید نشان داد که چگونه می‌توان آن را یافت[17] و لذا قانون طرد شِق ثالث و به تبع آن برهان خلف معتبر نیستند.

نظریه‌ی CZ در بیان اصول از منطق شهودی[19] بهره می‌گیرد و زبان صوری آن نیز حساب شهودگرای محمولات مرتبه‌ی اول[20] است. نمادهای تساوی (=)، عضویت (∈)، گزاره‌ی همواره غلط (Ω)[21]، پیونددهنده‌های منطقی (∨ , ∧ و ⇒) و سورها (∃ و ∀) براساس منطق شهودی، به شرح زیر تفسیر می‌شوند:

- برای اثبات $A \vee B$، باید اثباتی برای A و یا برای B ارائه داد.
- برای اثبات $A \wedge B$، باید یک اثبات برای A و یک اثبات برای B ارائه داد.
- برای اثبات $A \Rightarrow B$، باید ساختمانی ارائه داد که هر اثبات A را به اثباتی برای B تبدیل کند.
- برای Ω هیچ اثباتی وجود ندارد.
- برای اثبات $\neg A$ باید $A \Rightarrow \Omega$ را ثابت کرد؛ یعنی با فرض برهانی بر روی A به تناقض رسید.
- برای اثبات $\forall x . A(x)$، باید ساختمانی ارائه دهیم که هر x_0 از دامنه را به اثباتی برای $A(x_0)$ تبدیل کند.
- برای اثبات $\exists x . A(x)$، باید c و d را به نحوی ارائه دهیم که d اثباتی برای $A(c)$ باشد.

در ادامه فهرست اصول موضوعه‌ی CZ که بیانگر خواص مفاهیم «مجموعه» و «عضویت» در آن هستند، آورده شده است.

تساوی مصداقی[22]: $x = y \Leftrightarrow (\forall z \bullet z \in x \Leftrightarrow z \in y)$

مجموعه‌ی تهی[23]: $\exists z \; \forall x \bullet \neg (x \in z)$

این مجموعه طبق اصل محتوا یکتاست و آن را با ϕ نشان می‌دهند.

$\forall x\, \forall y\, \exists z \bullet x \in z \wedge y \in z$: زوج‌سازی[۲۴]

$\forall x\, \forall y\, \exists z \bullet \forall u \in x\, \forall v \in y \bullet (u, v) \in z$: حاصل ضرب دکارتی[۲۵]

$\forall x\, \exists z\, \forall y \in x\, \forall u \in y \bullet u \in z$: اجتماع[۲۶]

$\forall x\, \exists z\, \forall y \bullet y \in z \Leftrightarrow (y \in x \wedge \phi\ [y])$: جداسازی محدود[۲۷]

(ϕ یک فرمول است)

$(\forall y \bullet (\forall x \in y \bullet \phi\ [x]) \Rightarrow \phi\ [y])) \Rightarrow \forall x \bullet \phi\ [x]$: استقراء مجموعه[۲۸]

$\forall x\, \exists z\, \forall y \bullet y \in z \Leftrightarrow y \sqsubseteq x$: مجموعه‌ی توانی تصمیم‌پذیر[۲۹]

$\exists x\, \forall y \bullet y \in x \Leftrightarrow y = \phi \vee \exists\ u \in x \bullet y = \{u\}$: بی‌نهایت[۳۰]

مفهوم قواعد و اصول فوق به تفصیل آمده است.[۵] در اصل جداسازی محدود، فرمول محدود، فرمولی است که در ساختن آن فقط سورهای محدود به کار رود. از نماد $\sqsubseteq$ در اصل مجموعه‌ی توانی تصمیم‌پذیر برای بیان مفهوم زیرمجموعه‌ی تصمیم‌پذیر استفاده شده است. آنچه در اینجا اهمیت دارد زبان توصیف CZ و نحوه استفاده‌ی کاربر از آن در EFPD است. در مرحله‌ی توصیف کاربر به دو امکان نیاز دارد:

۱. ابزاری که به‌کمک آن بتوان عبارات CZ را نوشت که اساساً یک نوع امکان زبانی است. در EFPD بنا به دلایلی که گفته خواهد شد از دستورات نرم‌افزار LaTeX استفاده می‌شود.

۲. ابزاری که بتوان توصیف‌های نوشته شده را از لحاظ نحوه و درستی و عدم تناقض انواع وارسی کرد. بدین منظور یک واحد کنترل‌کننده‌ی نوع[۳۱] در EFPD پیش‌بینی شده که در ادامه آن را شرح خواهیم داد. بنابراین واحد توصیف EFPD اساساً متشکل از یک ابزار کنترل‌کننده‌ی انواع و ساختی نحوی برای CZ است.

واحد کنترل کننده‌ی انواع

کنترل انواع در EFPD در سه سطح انجام می‌گیرد:[۲۴-۲۶]

۱. تعیین نوع مافوق برای هر نوعِ استفاده شده در توصیف؛

۲. استنتاج نوع زیر عبارات، براساس نوع عناصر تشکیل‌دهنده‌ی آن؛

۳. تعیین درست بودن نوعِ عبارات (خوش نوعی[۳۲]).

در زیر شرح مختصری از اقدامات انجام شده در ارتباط با هر سطح ارائه می‌شود.

تعیین نوع مافوق

مجموعه‌هایی که در CZ به‌عنوان نوع به‌کار می‌روند به یکی از روش‌های زیر اعلان و یا تولید می‌شوند:[۶و۲۷]

۱. نوع داده شده: یک نوع می‌تواند یک مجموعه باشد. نوع‌های داده شده‌ی $T_1, T_2, ..., T_n$ در ابتدای توصیف به‌صورت $[T_1, T_2, ..., T_n]$ معرفی می‌شوند.

۲. حاصلضرب دکارتی: حاصلضرب دکارتی چند نوع، مانند $T_1, T_2, ..., T_n$، می‌تواند یک نوع به صورت $T_1 \times T_2 \times ... \times T_n$ باشد.

۳. مجموعه‌ی زیرمجموعه‌ها: مجموعه‌ی تمام زیرمجموعه‌های تصمیم‌پذیر یک نوع T، که به صورت $\mathbb{P}T$ نوشته می‌شود، یک نوع است.

۴. توصیف مجموعه: تعریف یک مجموعه با استفاده از مسندات ($\{a:T \mid p \bullet q\}$) یک نوع را مشخص می‌کند.

۵. شِما: ازیک شِما ($[a:T \mid p]$) می‌توان به عنوان یک نوع استفاده کرد.

۶. نوع آزاد: برای بیان یک نوع یا بیان عناصر آن در صورتی که تعداد محدودی باشد ($\{t_1, t_2, ..., t_n\}$) و یا بیان آن به‌صورت بازگشتی به شکل زیر از نوع آزاد استفاده می‌شود.

$$A ::= e_1 \mid e_2 \mid ... \mid e_m \mid f_1 \ll E_1[A] \gg \mid f_2 \ll E_2[A] \gg \mid ... \mid f_n \ll E_n[A] \gg$$

منظور از نوع مافوقِ نوع A مجموعه‌یی است مثل B که $A \subseteq B$ و B بزرگ‌ترین مجموعه‌ی ممکن دارای این خصوصیت در توصیف باشد. برای توصیف دقیق این بخش از ابزار، تابع $[]^s$، که نوع مافوق هر نوع را براساس تعریف آن و نوع مافوق نوع‌های ابتدایی به‌دست می‌دهد به صورت زیر تعریف می‌شود:

$$[\mathbb{Z}]^s = \mathbb{Z}$$
$$[\mathbb{N}]^s = \mathbb{Z}$$
$$[[T_1, T_2, ..., T_n]]^s = [T_1, T_2, ..., T_n]$$
$$[\mathbb{P}T]^s = \mathbb{P}[T]^s$$
$$[T_1 \times T_2 \times ... \times T_n]^s = [T_1]^s \times [T_2]^s \times ... \times [T_n]^s$$
$$[\{a_1: T_1; a_2: T_2; ...; a_n: T_n \mid p\}]^s = [T_1]^s \times [T_2]^s \times ... \times [T_n]^s$$
$$[\{a_1: T_1; a_2: T_2; ...; a_n: T_n \mid p \bullet e\}]^s = [\tau(e)]^s$$
$$[[a_1: T_1; a_2: T_2; ...; a_n: T_n \mid p]]^s = [a_1: [T_1]^s; a_2: [T_2]^s; ...; a_n: [T_n]^s]$$
$$[\{t_1, t_2, ..., t_n\}]^s = [t_1, t_2, ..., t_n]$$

در تعریف فوق، $\mathbb{Z}$ مجموعه اعداد صحیح و $\mathbb{N}$ مجموعه اعداد طبیعی است که از نوع‌های ابتدایی در CZ محسوب می‌شوند.[۵] همچنین $\tau(e)$ تابعی است که نوع عبارت e را براساس قوانین استنتاج نوع تعیین می‌کند. از آنجا که نوع آزاد را می‌توان با استفاده از سایر روش‌های بیان انواع و تعریفِ نوع داده شده‌ی جدید پیاده‌سازی کرد نیازی به تعیین نوع مافوق آن به طور مستقیم نیست.[۲۷]

قوانین استنتاج نوع

استنتاج نوع زیرعبارت‌ها

برای تعیین نوع زیرعبارت‌های یک عبارت، مجموعه‌یی از قوانین استنتاج به فرم کلی زیر تعریف شده‌اند:[۲۷و۲۸]

$$[\text{شرط‌ها}]\ \frac{\text{مقدمات}}{\text{نتیجه}}\ \text{نام قانون}$$

در این سیستم جملات به صورت $\rho \vdash e:T$ بیان می‌شوند که ρ محیط، e عبارت و T نوع است. آنچه که در محیط ρ تعریف می‌شود انواع نسبت داده شده به متغیرهای آزاد e هستند. جمله‌ی فوق چنین خوانده می‌شود: در محیط ρ عبارت e دارای نوع T است. این قوانین عبارت‌اند از:

(۱) مجموعه‌ی توانی تصمیم‌پذیر $\dfrac{}{\rho \vdash X:\mathbb{P}X}$ []

(۲) مجموعه‌ی توانی تصمیم‌پذیر $\dfrac{\rho \vdash A:\mathbb{P}X}{\rho \vdash \mathbb{P}A:\mathbb{P}\mathbb{P}X}$ []

حاصل ضرب دکارتی $\dfrac{\rho \vdash A:\mathbb{P}X \rho \vdash B:\mathbb{P}Y}{\rho \vdash A\times B:\mathbb{P}(X\times Y)}$ [] اتحاد $\dfrac{\rho \vdash A:\mathbb{P}\mathbb{P}X}{\rho \vdash \cup A:\mathbb{P}X}$ []

زوج شدن $\dfrac{\rho \vdash a:X \quad \rho \vdash b:X}{\rho \vdash \{a,b\}:\mathbb{P}X}$ [] زوج مرتب $\dfrac{\rho \vdash a:X \ \rho \vdash b:Y}{\rho \vdash (a,b):X\times Y}$ []

مجموعه‌ی تهی $\dfrac{\rho \vdash A:X}{\rho \vdash \phi_A:\mathbb{P}X}$ []

جداسازی $\dfrac{\rho \vdash A:\mathbb{P}X \quad \rho \vdash x:A \vdash \phi}{\rho \vdash \{x\in A \mid \phi\}:\mathbb{P}X}$ [ϕ خوش نوع است]

قواعد تطبیق نوع عملوند[۳۳] عملیات

علاوه بر قواعد بالا، قواعد دیگری نیز برای استنتاج نوع عملوندهای (کارگزارهای) عملیات مورد نیاز است. در هر توصیف رسمی، از عملیات روی رابطه‌ها، توابع، رشته‌ها و کیسه[۳۴]ها به‌مقیاس وسیع استفاده می‌شود.[۶] نوع نتیجه‌ی حاصل از انجام هر عمل براساس نوع عملوندهای آن مشخص می‌شود. برای استنتاج نوع بایستی این قواعد نیز تعریف شوند. لازم به توضیح است که توابع حالت خاصی از رابطه‌ها بوده و پیرو قواعد به کار برده شده روی رابطه‌ها هستند.

• **رابطه‌ها:** رابطه‌ی R بین دو نوع B و A $(R\in A\leftrightarrow B)$، مجموعه‌یی از زوج‌های مرتب (x,y) است که در آن $x\in A$ و $y\in B$ باشد.

$$A\leftrightarrow B == \mathbb{P}(A\times B)$$

عملیات زیر روی رابطه‌ها تعریف شده‌اند که برای هر یک نوع عملوندها و نوع حاصل عمل مشخص شده است.[۶و۲۹]

دامنه: $\text{dom}_- : (X\leftrightarrow Y) \to \mathbb{P}X$

برد: $\text{ran}_- : (X\leftrightarrow Y) \to \mathbb{P}Y$

رابطه‌ی همانی: $\text{id}_- : X\leftrightarrow Y$

ترکیب رابطه‌یی: $-\fatsemi- : (X\leftrightarrow Y)\times(Y\leftrightarrow Z)\to(X\leftrightarrow Z)$

ترکیب رابطه‌یی عکس: $-\circ- : (X\leftrightarrow Y)\times(Y\leftrightarrow Z)\to(X\leftrightarrow Z)$

تحدید دامنه: $-\lhd- : \mathbb{P}X\times(X\leftrightarrow Y)\to(X\leftrightarrow Y)$

تحدید برد: $-\rhd- : (X\leftrightarrow Y)\times\mathbb{P}Y\to(X\leftrightarrow Y)$

ضد تحدید دامنه: $-\lhd- : \mathbb{P}X\times(X\leftrightarrow Y)\to(X\leftrightarrow Y)$

ضد تحدید برد: $-\rhd- : (X\leftrightarrow Y)\times\mathbb{P}Y\to(X\leftrightarrow Y)$

معکوس رابطه: $-^{\sim} : (X\leftrightarrow Y)\to(Y\leftrightarrow X)$

بستار متعدی: $-^{+} : (X\leftrightarrow X)\to(X\leftrightarrow X)$

بستار بازتابی متعدی: $-^{*} : (X\leftrightarrow X)\to(X\leftrightarrow X)$

بازنویسی: $-\oplus- : (X\leftrightarrow Y)\times(X\leftrightarrow Y)\to(X\leftrightarrow Y)$

• **ردیف‌ها:** از ردیف برای مدل‌سازی مجموعه‌یی مرتب از مؤلفه‌ها استفاده می‌شود. در یک ردیف مؤلفه‌ها ممکن است بیش از یک‌بار ظاهر شوند. تعریف رسمی برای یک ردیف از مؤلفه‌ها از هر مجموعه‌ی X در نظریه‌ی CZ چنین است:[۱۹]

$$\text{seq}X == \{f:\mathbb{N}\to X \mid \exists n:\mathbb{N}\bullet \text{dom}\, f=\mathbb{N}_n\} \quad \text{که در آن} \quad \mathbb{N}_i == \{1,\dots,i\}$$

در زیر برای عملیات روی ردیف‌ها، نوع عملوندها و نوع حاصل مشخص شده است:

اتصال: $-^\frown- : \text{seq}X\times\text{seq}X\to\text{seq}X$

صافی: $-\restriction- : \text{seq}X\times\mathbb{P}X\to \text{seq}X$

عضو اول: $head_- : \text{seq}X \nrightarrow X$

همه‌ی اعضا، به‌جز عضو اول: $tail_- : \text{seq}X \nrightarrow \text{seq}X$

• **کیسه‌ها:** کیسه مشابه با مجموعه است با این تفاوت که تعداد دفعات وجود هر مؤلفه در کیسه دارای اهمیت است. کیسه به‌طور کلی چنین تعریف می‌شود:

$$\text{bag}X == X\to\mathbb{N}_1 \quad \text{که در آن} \quad \mathbb{N}_1 = \mathbb{N}\setminus\{0\}$$

در زیر برای عملیات روی کیسه‌ها، نوع عملوندها و نوع حاصل عمل مشخص شده است.

کثرت: $-\sharp- : \text{bag}X\times X\to\mathbb{N}$

عضویت: $-\text{in}- : X\leftrightarrow\text{bag}X$

زیر کیسه: $-\sqsubseteq- : \text{bag}X\leftrightarrow\text{bag}X$

اجتماع: $-\uplus- : \text{bag}X\times\text{bag}X\to\text{bag}X$

تفاضل: $-\uplus- : \text{bag}X\times\text{bag}X\to\text{bag}X$

تولید یک کیسه از اعضای ردیف: $items_- : \text{seq}X\to\text{bag}X$

قوانین تعیین خوش‌نوع بودن یک عبارت

برای استفاده از قوانین استنتاج مذکور در فوق، برای کنترل نوع‌ها در محیط‌های داده شده لازم است خوش‌نوع بودن یک عبارت با توجه

به این قوانین نشان داده شود.[۲۷] برای مثال در $e_1=e_2$ زیرعبارات موجود در دو طرف علامت تساوی باید دارای نوع یکسان (نوع مافوق یکسان) باشند. لذا جمله‌ی $\rho \vdash p\surd$ که به‌صورت: «در محیط ρ عبارت p خوش نوع است» خوانده می‌شود، تعریف شده است. قوانین خوش‌نوع بودن عبارات عبارت‌اند از:

$$\text{عضویت}\ \frac{\rho \vdash x{:}A \quad \rho \vdash y{:}\mathbb{P}A}{\rho \vdash x \in y\surd}[] \qquad \text{تساوی}\ \frac{\rho \vdash e_1{:}T \quad \rho \vdash e_2{:}T}{\rho \vdash e_1 = e_2\surd}[]$$

$$\Rightarrow \frac{\rho \vdash \phi\surd \quad \rho \vdash \psi\surd}{\rho \vdash \phi \Rightarrow \psi\surd}[] \qquad \wedge \frac{\rho \vdash \phi\surd \quad \rho \vdash \psi\surd}{\rho \vdash \phi \wedge \psi\surd}[] \qquad \vee \frac{\rho \vdash \phi\surd \quad \rho \vdash \psi\surd}{\rho \vdash \phi \vee \psi\surd}[]$$

$$\forall \frac{\rho, x{:}A \vdash \phi\surd}{\rho \vdash \forall x{:}A \bullet \phi\surd}[] \qquad \exists \frac{\rho, x{:}A \vdash \phi\surd}{\rho \vdash \exists x{:}A \bullet \phi\surd}[]$$

با توجه به استفاده‌ی وسیع از ساختار شِما در عبارات، باید قوانین زیر را بر قوانین بالا اضافه کرد:

$$\text{ترکیب شما}\ \frac{\rho \vdash S_1\surd \quad \rho \vdash S_2\surd}{\rho \vdash S_1 \fatsemi S_2\surd}[] \qquad \text{شِما}\ \frac{\rho \vdash \phi\surd}{[\rho]\phi\surd}[]$$

با توجه به مطالب فوق، واحد کنترل‌کننده‌ی ساختار نحوی و نوع‌ها در CZ در دست تهیه است که از قابلیت اتصال نیمه‌شفاف (از دید کاربر) به بخش وارسی، برخوردار است.

بخش وارسی

پس از کنترل نحوی و عدم تناقض نوع‌های به‌کار گرفته شده در توصیف، به مرحله‌ی اثبات درستی توصیف نوشته شده به زبان CZ می‌رسیم. از آنجا که امکان اثبات یک حکم در حالت کلی به‌صورت تمام‌خودکار وجود ندارد، در این مرحله ابزاری در اختیار کاربر قرار می‌گیرد که وی را در اِعمال قواعد استنتاج و رسیدن به درخت اثبات یاری دهد. لذا به چنین ابزاری «یاریگر اثبات»[۳۵] گفته می‌شود. برای پیاده‌سازی یاریگر اثبات CZ در EFPD، از ابزارگان خاصی با عنوان کلی یاریگر پویای اثبات[۳۶] استفاده شده است. در این‌گونه ابزارگان، می‌توان با معرفی یک نظریه، یعنی با تعریف منطق صوری و قواعد استنتاج آن، یک یاریگر اثبات برای آن نظریه ساخت. ابزارگانی که برای پیاده‌سازی یاریگر اثبات EFPD به کار گرفته شده، ابزار JAPE است.[۳۰] در شکل ۲ نمونه‌یی از ورودی تعریف قواعد استنتاج که برخی از آنها در بخش قبل معرفی شدند، نشان داده شده است.

یکی دیگر از قابلیت‌های JAPE، امکان تولید (درخت اثبات) خروجی با قالب دستورات LaTeX است. در شکل ۳ نمونه‌یی از یک درخت اثبات CZ آورده شده که به کمک دستورات LaTeX تعریف و حروف‌چینی شده است.

```
TACTIC FAIL(x) IS JAPE (fail x)
RULE cut(B) IS FROM B AND B ⊢ C INFER C
RULE thin(A) IS FROM C INFER A ⊢ C

RULE "→-E"(A) IS FROM A AND A→B INFER B
RULE "∧-E(L)"(B) IS FROM A ∧ B INFER A
RULE "∧-E(R)"(A) IS FROM A ∧ B INFER B
RULE "∨-E"(A,B) IS FROM A ∨ B AND A ⊢ C AND
B ⊢ C INFER C
RULE "∀-E"(c) IS FROM ∀x. A(x) AND c inscope
INFER A(c)
RULE "∃-E"(OBJECT c) WHERE FRESH c AND c
NOTIN ∃x.A(x)
IS FROM ∃x.A(x) AND var c, A(c) ⊢ C INFER C

RULE "→-I" IS FROM A ⊢ B INFER A→B
RULE "∧-I" IS FROM A AND B INFER A ∧ B
RULE "∨-I(L)" (B) IS FROM A INFER A ∨ B
RULE "∨-I(R)" (A) IS FROM B INFER A ∨ B
RULE "∀-I"(OBJECT c) WHERE FRESH c
IS FROM var c ⊢ A(c) INFER ∀x . A(x)
RULE "∃-I"(c)IS FROM A(c) AND c inscope INFER
∃x.A(x)

RULE hyp(A) IS INFER A ⊢ A
AUTOMATCH hyp

STRUCTURERULE INDENTITY    hyp
STRUCTURERULE CUT          cut
STRUCTURERULE WEAKEN       thin

RULE "inscope" IS INFER var x ⊢ x inscope
AUTOMATCH "inscope"
```

شکل ۲. نمونه‌یی از ورودی تعریف قواعد استنتاج برای JAPE.

$$\dfrac{\dfrac{}{0 \in \mathbb{N}}\,\mathbb{N}_{i1} \quad \dfrac{\dfrac{}{0 \in \mathbb{N}}\,\mathbb{N}_{i1}}{0+1 \in \mathbb{N}}\,\mathbb{N}_{i2}}{\exists a \in \mathbb{N} \bullet a+1 \in \mathbb{N}}\,\exists_i$$

شکل ۳. یک نمونه درخت اثبات CZ.

محیط تولید برنامه

پس از تشکیل درخت اثبات توصیف، آخرین مرحله استخراج برنامه از درخت اثبات است. از آنجا که اثبات فوق یک اثبات ساختی است، قاعدتاً باید بتوان الگوریتم برنامه را از متن اثبات استخراج کرد. اما مشکل این است که زبان CZ ساختی کاملاً مشابه با زبان Z دارد، زیرا هر دو از زبان حساب محمولات مرتبه‌ی اول در نظریه‌ی مبنایی خود بهره می‌گیرند. بنابراین ماهیت توصیف‌های نوشته شده به این زبان به گونه‌یی است که از بیان صریح مفهوم محاسبه عاجزند. همان‌طور که پیش‌تر ذکر شد، در توصیف عملکرد نرم‌افزار، آنچه باید انجام شود توصیف می‌شود، نه چگونگی انجام آن. بنابراین گرچه CZ یک زبان توصیف ساختی است، نحوه‌ی انجام یک کار (محاسبه) صراحتاً در توصیف‌های آن وجود ندارد و باید از درون درخت اثبات استخراج شود. چنین خاصیتی برای فرایند تولید خودکار برنامه مشکل ایجاد می‌کند. لذا در استخراج خودکار برنامه، لازم است از روشی استفاده شود که در آن توصیف و الگوریتم به‌طور صریح در کنار هم باشند. نظریه‌ی انواع مارتین لوف، یک روش ساختی است که از این ویژگی برخوردار است. یعنی در آن امکان بیان توأم توصیف و برنامه در کنار هم وجود دارد. ساختی عبارت‌های نظریه‌ی انواع مطابق گونه‌یی از حساب λ است که مبنای زبان‌های تابعی است. مارتین لوف، واضع نظریه‌ی انواع این دیدگاه را مطرح کرده که نظریه‌ی انواع وی را می‌توان معادل با یک زبان برنامه‌سازی دانست و می‌توان از آن به‌عنوان محیطی جامع برای توصیف و برنامه‌سازی (اثبات درستی) استفاده کرد.[۲۱] ولی پیچیدگی ساختی و معنایی عبارات نظریه‌ی انواع مانع از فراگیری آن به‌عنوان یک محیط عملی برای تولید برنامه شده است.

به‌همین منظور، ترجمه‌یی از عبارات و قواعد استنتاج نظریه‌ی CZ به نظریه‌ی انواع مارتین‌لوف با هدف بهره‌گیری از امکانات و مزایای هر دو نظریه، یعنی سادگی کار و امکان تولید برنامه، ارائه شده است.[۵]

به این ترتیب اگر بتوان با پیاده‌سازی ابزاری، این ترجمه را به‌صورت خودکار انجام داد، تولید مکانیکی برنامه از توصیف‌های نوشته شده به CZ تسهیل شده و مشکِل‌گذار از توصیف‌های CZ به برنامه‌ی قابل اجرا برطرف می‌شود. لذا در بخش تولید برنامه‌ی EFPD، یک واحد مترجم وجود دارد که درخت‌های توصیف را به یک درخت اثبات متناظر در نظریه‌ی انواع ترجمه و عبارت لاندای متناظر با برنامه را به‌طور خودکار تولید می‌کند. واحد دیگری نیز در بخش تولید برنامه‌ی EFPD پیش‌بینی شده که عبارات لاندا را به عبارات یک زبان برنامه‌سازی تابعی تبدیل کند.

پیش از بحث در مورد واحد ترجمه، لازم است ابتدا مقدمه‌یی در باب نظریه‌ی انواع مارتین‌لوف ذکر شود. در ادامه، سعی داریم نظریه‌ی انواع را به‌نحوی، فارغ از جزئیات پیچیده‌ی نظری، شرح دهیم.

نظریه‌ی انواع مارتین‌لوف

نظریه‌ی انواع مارتین لوف (ML)، یکی از مهم‌ترین صورت‌های بیان ایده‌های ریاضیات ساختی است. ایده‌ی اصلی در نظریه‌ی انواع این است که برنامه، همچون سایر ساخت‌های داده‌یی رایج در زبان‌های برنامه‌سازی، دارای نوع است. به‌عنوان مثال، توصیف «تقسیم بر ۲» را که یک عدد طبیعی را بر دو تقسیم می‌کند، در نظر بگیرید. در زبان توصیف Z، نوع چنین تابعی را می‌توان چنین توصیف کرد: $Div2:\mathbb{N}\rightarrow\mathbb{N}$. این نوع، گرچه تا حدودی گویای خواص «تقسیم بر ۲» است، بیان دقیقی از کارکرد آن نیست؛ چراکه هر تابع دیگری که دامنه و برد آن مجموعه‌ی اعداد طبیعی است، از همین نوع است. در نظریه‌ی انواع، مفهوم نوع، یک قدم به جلو برده شده است. به‌طوری‌که نوع برنامه بیانگر عملکرد آن است، یا به‌عبارت دیگر توصیفی برای آن است مطابق مفهوم کلاسیک نوع، نوع یک شیء داده شده، بیانگر مجموعه‌ی مقادیر ممکن برای آن است. از طرف دیگر، نوع یک برنامه‌ی توصیفی برای مسئله‌یی است که این برنامه‌ی خاص، آن مسئله را حل می‌کند. بنابراین می‌توان تناظری بین مفهوم کلاسیک نوع، و مفهوم نوع یک برنامه در نظریه‌ی انواع برقرار کرد: «یک نوع در نظریه‌ی انواع معادل گزاره‌یی است که مسئله‌یی را توصیف می‌کند، و آن گزاره به‌نوبه‌ی خود معادل مجموعه‌ی راه‌حل‌های متصور برای آن مسئله است.»

بنابراین در نظریه‌ی انواع، مفاهیم مجموعه (نوع[۳۷])، گزاره، توصیف و مسئله‌ی معادل‌اند. به‌عبارت دیگر، در نظریه‌ی انواع حکم $a\in A$ را می‌توان حداقل به چهار صورت خواند:[۲۰]

- a عضوی از مجموعه‌ی A است.
- a اثباتی است برای گزاره‌ی A.
- a برنامه‌یی است که توصیف A را پیاده‌سازی می‌کند.
- a راه‌حلی است برای مسئله‌ی A.

در واقع ایده‌ی اصلی در نظریه‌ی انواع را می‌توان چنین بیان کرد:

«اثبات کردن همان ساختن عضو نوع، یعنی برنامه، است»

ایده‌ی فوق را می‌توان به‌صورت بازگشتی برای یک عبارت براساس اجزای آن بیان کرد:

- یک اثبات برای $A\wedge B$ یک زوج (a,b) است که a و b به‌ترتیب اثبات‌های A و B اند.

- یک اثبات برای $A \Rightarrow B$ تابعی است که هر اثبات Aرا به اثباتی از B می‌نگارد.
- یک اثبات برای $A \vee B$ اثباتی است برای Aیا برای B.
- یک اثبات برای $\forall x{:}\, A \bullet B(x)$ تابعی است که هر اثبات aاز Aرا اثباتی از $B(a)$می‌نگارد.
- یک اثبات برای $\exists x{:}\, A \bullet B(x)$ یک زوج (a,b)است که در آن a اثباتی برای A و bاثباتی برای $B(a)$است.

برخی از نوع‌های ساخته شده در نظریه‌ی انواع عبارت‌اند از:

- **حاصل‌ضرب دکارتی**: $A \times B$، متناظر با گزاره‌ی $A \wedge B$.
- **اجتماع مجزا**[38]: $A+B$، متناظر با گزاره‌ی $A \vee B$.
- **فضای تابعی**[39]: $A \to B$، متناظر با گزاره‌ی $A \Rightarrow B$.
- **اجتماع مجزای خانواده‌یی از مجموعه‌ها (جمع وابسته)**: $(\Sigma x{:}A)B(x)$ متناظر با گزاره‌ی $\exists x{:}\, A \bullet B(x)$.
- **حاصل‌ضرب دکارتی خانواده‌یی از مجموعه‌ها**: $(\Pi x{:}A)B(x)$ متناظر با گزاره‌ی $\forall x{:}\, A \bullet B(x)$.

عضو هر نوع در نظریه‌ی انواع، به‌وسیله‌ی یک عبارت حساب λ-، بیان می‌شود. قواعد نحوی ناظر بر این عبارات توسط یک نظریه‌ی عبارات بیان می‌شود.[20]

حساب λ- مبنای معنایی زبان‌های تابعی است و بنابراین با داشتن عضو یک نوع می‌توان ادعا کرد که برنامه‌ی تابعی یک توصیف درست است.

برای هر نوع در نظریه‌ی انواع، چهار نوع قاعده باید تعریف شود:[20]

قاعده‌ی ایجاد[40]: این قاعده مشخص می‌کند که تحت چه شرایطی می‌توان ادعا کرد که Aیک نوع است و چه موقع دو عضو از نوع A با هم برابرند.

قاعده‌ی معرفی[41]: این قاعده مشخص می‌کند که عضوهای کانونی چگونه ساخته می‌شوند و چه موقع دو عضو کانونی با هم برابرند. عملگرهای سازنده‌ی عضو برای هر نوع، در این نوع قاعده معرفی می‌شوند.

قاعده‌ی حذف[42]: این قاعده بیان می‌کند که چگونه می‌توان یک ویژگی را در مورد یک عضو از این نوع اثبات کرد. عملگرهای انتخاب‌کننده‌ی عضو، که یک ثابتِ غیر کانونی‌اند، در همین نوع قاعده معرفی می‌شوند.

قاعده‌ی تساوی یا قاعده‌ی محاسباتی: این قاعده نحوه‌ی محاسبه‌ی عملگر انتخاب کننده‌ی عضو را بیان می‌کند.

مثلاً قواعد فوق برای مجموعه‌ی حاصل‌ضرب چنین است:

$$\frac{A\ type \quad B\ type}{A \times B\ type} \times f$$

$$\frac{a \in A \quad b \in B}{(a,b) \in A \times B} \times i$$

$$\frac{C(v) type\ [v \in A \times B]\ a \in A \times B\ e(x,y) \in C((x,y))[x \in A, y \in B]}{split\ (p,e) \in C(p)} \times e$$

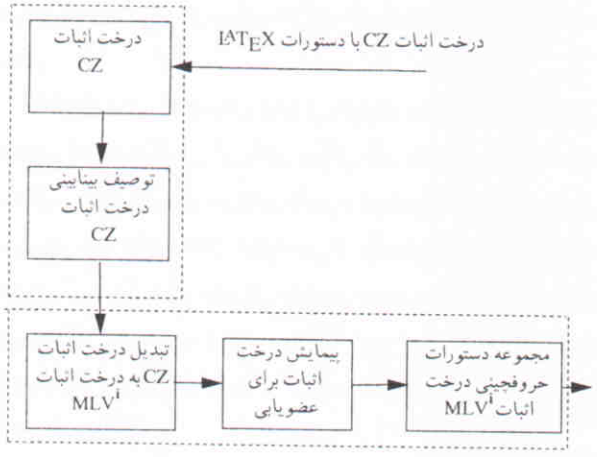
$$\frac{a \in A \quad b \in B \quad e(x,y) \in C((x,y))[x \in A, y \in B]}{split\ ((a,b),e) = e(a,b) \in C((a,b))} \times c$$

– فهرست کامل قواعد برای نوع‌های دیگر نیز قابل دسترسی است.[15،20،21] تفاوت گونه‌های مختلف نظریه‌ی انواع مارتین لوف، در نوع‌های تعریف شده در آن‌گونه و نیز مصداقی بودن یا نبودن مفهوم تساوی نوع‌ها است. گونه‌یی که CZ به آن ترجمه می‌شود، MLV^i نام دارد.

واحد ترجمه‌ی نظریه‌ی CZ به نظریه‌ی انواع مارتین‌لوف

شکل ۴ شمایی از معماری برای واحد مترجم اثبات‌های CZ به MLV^i را نشان می‌دهد. همان‌طور که در شکل مشخص شده است، درخت اثبات CZ که به‌صورت دستورات متنی LaTeX بیان می‌شود، در مرحله‌ی نخست توسط یک مبدل به درختی بینابینی (نیمه‌تمام) در MLV^i تبدیل می‌شود. آنچه در این مرحله صورت می‌گیرد، ترجمه‌ی ساختی عبارات منطق شهودی به عبارات نظریه‌ی انواع است.

در مرحله‌ی بعد، توصیف بینابینی توسط واحد عضو یاب پردازش شده و با توجه به ترجمه‌ی قواعد استنتاج CZ به MLV^i عضوهای هر نوع در درخت بینابینی محاسبه می‌شوند. به این ترتیب در انتهای این فاز، یک درخت کامل MLV^i به دست می‌آید.



شکل ۴. معماری واحد مترجم EFPD.

در آخرین مرحله لازم است از نمایش درون حافظه‌یی درخت اثبات MLV[i]، خروجی ماندگار در قالب دستورات LaTeX تولید شود. خروجی به‌صورت درختی، خطی، یا عبارت لاندای عضوِ نوعِ نتیجه، توسط واحد مترجم تولید می‌شود.

واحد مترجم CZ به MLV[i] با قابلیت افزودن قواعد استنتاج جدید برمبنای روش مبتنی بر خانواده[۴۳]، طراحی، و به کمک زبان ابزار PRECC، که یک مولّد تحلیل‌گر نحوی است[۳۱]، قواعد استنتاج و ترجمه‌ی آنها پیاده‌سازی شده است.[۳۲]

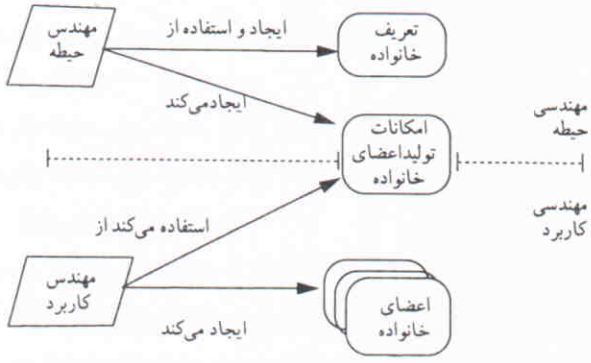
روش به کار گرفته شده در ساخت واحد مترجم

در این قسمت مختصری درباره‌ی فرایند و روش‌های به کار رفته در طراحی و ساخت واحد مترجم شرح داده خواهد شد. از آنجا که واحد مترجم به عنوان یک نرم‌افزار مستقل پیاده‌سازی شده است، در شرحی که خواهد آمد، از آن با عنوان ابزار مترجم و یا نرم‌افزار مترجم یاد می‌شود.

انتخاب روش طراحی نرم‌افزار مترجم، متأثر از ویژگی‌ها و خواست‌هایی بوده که در بررسی فنی مسئله‌ی ترجمه‌ی نظریه‌ی CZ به نظریه‌ی انواع مشخص شد. مهم‌ترین این ویژگی‌ها، نیاز به انعطاف در قبال تغییر و گسترش است. نرم‌افزار مترجم لازم است قابلیت قبول قواعد جدید و نیز تغییر در قواعد (چه از لحاظ ساختی و چه از لحاظ معنایی) را داشته باشد. به این منظور روش طراحی با این دید انتخاب شده که تعریف عملکرد نرم‌افزار در سطحی بالاتر از انتزاع، قابل ارائه باشد و نیز برنامه(ها)ی نهایی را به کمک ابزارهایی به‌طور نیمه خودکار بتوان تولید کرد، و در نتیجه برای اعمال تغییرات در فرایند ترجمه، تا حد امکان نیازی به رجوع به متن برنامه نباشد و بتوان آن تغییرات را در سطح بالاتر از انتزاع اعمال کرد. روش FAST[۴۴] روشی است که چنین سبکی از تولید نرم‌افزار را ممکن می‌سازد.[۳۳] لذا در طراحی و تولید نرم‌افزار مترجم، از ایده‌های این روش استفاده شده است.

FAST روشی است برای تولید نرم‌افزار به سبک خطوط تولید صنعتی، که ایده‌ی اصلی آن یافتن ویژگی‌های مشترک بین گونه‌های مختلف نرم‌افزار در یک حوزه‌ی خاص و ایجاد یک خط تولید از آن است. هر خط تولید معادل خانواده‌یی از یک سری نرم‌افزار است که اعضای آن ویژگی‌های مشترکی دارند، و در عین حال هر یک دارای خصوصیاتی منحصر به فرد هستند.[۳۴] این خصوصیات منحصر به فرد نشانگر نیازهای خاص یک کاربرد خاص در یک مقطع زمانی خاص است.[۳۴]

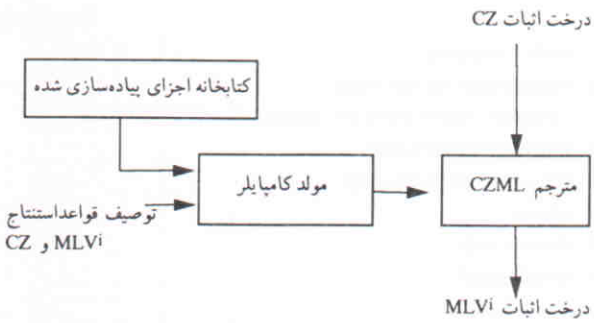
در روش FAST ویژگی‌های مشترک و منحصر به فرد اعضای یک خانواده طی فرایندی به‌نام تحلیل مشترکات[۴۵] تشریح می‌شوند. براساس این تحلیل، یک محیط مهندسی کاربردی[۴۶] برای ساخت سریع و کم‌هزینه‌ی اعضای خانواده ایجاد می‌شود.[۳۳و۳۴] شکل ۵ فرایندهای موجود در یک روش تولید نرم‌افزار خانواده محور[۴۷] (FBSD) را نشان می‌دهد. مطابق این شکل، فرایند FBSD را می‌توان به دو فاز مهندسی حوزه و تولید مهندسی نرم‌افزار کاربردی تقسیم کرد. در فاز اول، خانواده‌ها از طرح تحلیل مشترکات شناسایی می‌شوند. یک خانواده مجموعه‌یی از نرم‌افزارهاست که مشترکاتی دارند و تفاوت‌های بین آنها نیز قابل پیش‌بینی است.[۳۳] پس از شناسایی خانواده‌ها، فعالیت بعدی در فاز اول عبارت است از ایجاد ابزارهایی مناسب برای تولید خانواده‌ها. این ابزارها در حکم تجهیزات لازم برای راه‌اندازی یک خط تولید هستند. در فاز دوم، از ابزارهای ایجاد شده برای تولید اعضای مختلف خانواده استفاده می‌شود. FAST، گونه‌یی از روش FBSD است که در آن پس از تشخیص یک خانواده زبانی برای توصیف اعضای مختلف خانواده، و نیز ابزار لازم برای پردازش توصیف‌های نوشته شده به این زبان و تبدیل آنها به برنامه ایجاد می‌شود. از آنجا که هر عضو از خانواده معادل یک نمونه برنامه‌ی کاربردی است، زبان توصیف عضوهای یک خانواده را زبان مدل‌سازی برنامه‌ی کاربردی[۴۸] می‌نامند.



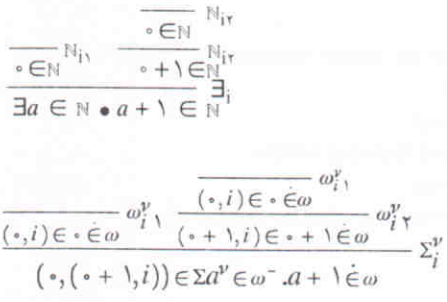
شکل ۵. فرایندهای موجود در یک روش تولید نرم‌افزار مبتنی بر خانواده.

با توجه به ایده‌های مطرح شده در مورد رهیافت مبتنی بر خانواده و به‌طور مشخص فرایند FAST ابزار مترجم طراحی شده است. مبنای این طراحی، معماری شکل ۴ است. در اولین قدم می‌بایست خانواده‌ی مورد نظر تشخیص داده می‌شد. مجموعه‌ی قواعد استنتاج نظریه‌ی CZ همراه با ترجمه‌ی آنها در MLV[i] خانواده‌یی از نرم‌افزارها را تشکیل می‌دهند که برای سهولت در نوشتار، خانواده‌ی CZML نامیده می‌شود. در اینجا به‌لحاظ رعایت ایجاز، جزئیات تحلیل مشترکات خانواده‌ی ارجاع CZML بیان نشده است.[۳۲]

مرحله‌ی پس از تحلیل مشترکات، طراحی و پیاده‌سازی AML است. در مرحله‌ی نخست از طرح ساخت محیط EFPD به‌لحاظ



شکل ۶. فرایند ساده‌شده‌ی تولید اعضای خانواده CZML.



شکل ۷. یک نمونه درخت اثبات CZ و تبدیل شده‌ی آن در MLV^i.

تسریع در کار، تصمیم بر این شد که به جای طراحی یک AML جدید، از زبان یک مولد پارسر استفاده شود.

PRECC یک ابزار مولد پارسر است که امکانات و ویژگی‌های منحصر به فردش، آن را برای چنین کاربردی مناسب می‌سازد. به این ترتیب فرایندساده شده‌ی تولید اعضای خانواده‌ی CZML به صورت شکل ۶ است.

هر عضو از خانواده‌ی CZML، یک درخت اثبات تولید شده توسط JAPE را در قالب دستورات متنی LaTeX دریافت کرده و یک درخت اثبات در نظریه‌ی انواع به عنوان خروجی تولید خواهد کرد. شکل ۷ یک نمونه درخت اثبات CZ و تبدیل شده‌ی آن به کمک ابزار مترجم را نشان می‌دهد.

واحد مولد برنامه‌ی تابعی

با انجام ترجمه‌ی درخت اثبات CZ به درخت اثبات متناظر در انواع، نظریه‌ی عبارت لاندای عضو نوع نتیجه در این درخت به دست می‌آید. این عبارت همان برنامه‌ی مورد نظر است که با حذف برخی اطلاعات اضافی مستقیماً قابل تبدیل به یک عبارت در یک زبان برنامه‌سازی تابعی، مثل Lisp، است.[۳۵]

پیچیدگی اصلی در کار این واحد، همان حذف اطلاعات اضافی از عبارت لانداست. در حال حاضر کار بر روی طراحی و پیاده‌سازی واحد مولدبرنامه‌ی Lisp در محیط EFPD کماکان ادامه دارد.

نتیجه‌گیری

یکی از بزرگ‌ترین مشکلات تولیدکنندگان محصولات نرم‌افزاری حصول اطمینان از دسترسی آن است. در این نوشتار، محیطی جامع برای تولید نرم‌افزار درست به‌روش استخراج معرفی و مبانی نظری آن بررسی شده است.

ابتدا طرح اولیه‌یی از معماری نرم‌افزاری که چنین محیطی را فراهم می‌آورد، بررسی شده و دلایل انتخاب زبان و نظریه‌ی مجموعه‌های CZ برای تهیه توصیف صوری ذکر شده‌اند. سپس برای بخش وارسی، از بین نرم‌افزارهای موجود، استفاده از ابزار یاریگر اثبات JAPE پیشنهاد شده است. در نهایت در مورد واحد استخراج برنامه‌ی این محیط، استفاده از روش ترجمه‌ی درخت اثبات در CZ به اثبات در نظریه‌ی انواع مارتین لوف و استخراج برنامه از آن پیشنهاد شده است.

ایجاد محیطی برای تولید برنامه‌ی درست دارای کاربردهای آموزشی زیادی است. کمبود چنین نرم‌افزاری در محیط‌های آموزشی به شدت احساس می‌شود. وجود چنین نرم‌افزاری آموزش درس‌هایی مانند مهندسی نرم‌افزار پیشرفته، قابلیت اطمینان نرم‌افزار، توصیف و وارسی صوری برنامه و تولید برنامه از توصیف صوری را تسهیل می‌کند. دانشجویان این دروس با استفاده از چنین محیطی درک بهتری از مفاهیم انتزاعی تدریس شده پیدا خواهند کرد. علاوه بر مزایای آموزشی یاد شده تهیه چنین محیطی مزایای دیگری نیز دارد که در زیر به چند مورد آن اشاره شده است:

- تسهیل استفاده از روش‌های صوری در فرایند تولید برنامه؛
- گسترش استفاده از روش‌های صوری در کاربردهای عملی در صنعت؛
- تبدیل فرایندتولید برنامه از یک فرایند مبتنی بر توانایی‌های برنامه‌ساز، به فرایندی مبتنی بر ریاضیات؛
- کاهش هزینه‌های خطازدایی که سهم قابل توجهی از هزینه‌ی تولید نرم‌افزار را در روش‌های سنتی به خود اختصاص می‌دهد.

رسیدن به محیطی که به‌طور عملی مورد استفاده‌ی صنعت قرار گیرد نیاز به تحقیقات بیشتری در خصوص هرچه کاربردی‌تر کردن این محیط دارد. اقدام برای تهیه‌ی واسط گرافیکی برای چنین محیطی در همین راستا صورت گرفته است که فعالیت‌های مربوط به آن ادامه دارد.

پانوشت‌ها

1. batch processing
2. requirements specification
3. Computer Aided Software Engineering (CASE) tools
4. functional specification
5. first order predicate logic
6. schema
7. abstract state
8. model based
9. animation
10. refinement
11. derivation
12. constructivity
13. constructive mathematics
14. systematic
15. an environment for formal program development
16. Ernst Zermelo
17. Russell's paradox
18. $(\forall x.\forall y.x\in y \vee x\notin y)$ Excluded Middle
19. intuitionistic logic
20. first-order intuitionistic predicat calculus
21. absurdity
22. extensionality
23. empty set
24. pairing
25. cartesian product
26. union
27. restricted separation
28. set induction
29. decidable power set
30. infinity
31. type checker
32. well-typedness
33. operand
34. bag
35. proof assistant
36. generic proof assistant

۳۷. در شرح نظریه‌ی انواع، مفاهیم مجموعه و نوع یکسان تلقی می‌شوند.

38. disjoint union
39. functional space
40. formation rule
41. introduction rule
42. elimination rule
43. family based
44. family-oriented abstraction, specification and translation
45. commonality analysis
46. application engineering environment
47. family based software development
48. Application Modelling Language (AML)

منابع

1. Pressman, R.S. *Software Engineering: A practitioners Approach.* McGraw-Hill, UK, Adapted by Darrel Ince, (1994). 24. J.M. Spivey. *Understanding Z: A Specification language and its formal semantics.* Cambridge University Press, (1988).
2. Bowen, J.P. and Stavridou, V. "The industrial take-up of formal methods in saftey-critical and other areas: A perspective". In *Proceedings of FME'93, Formal Methods, Europe, LNCS,* pp 183-195. Springer-Verlag (April 1993).
3. Ghezzi, C. Jazayeri, M. and Madrioli, D. *Fundamantals of Software Engineering.* Prentice-Hall, U.S.A (1991).
4. Dahl, D.J., Dijkstra, E.W. and Hoare, C.A.R. *Structured Programming.* Academic Press (New Yourk 1972).
5. Mirian-Hosseinabadi, S.-H. "Constructive Z" PhD Thesis, University of Essex, UK, (March 1997).
6. Woodcock, J. and Davies, J. *Using Z, Specifications, Refinement and Proof* Prentice-Hall (1996).
7. Jones, C.B. *Systematic software Development Using VDM.* Prentice-Hall (1986).
8. Diller, A. *Z An Introduction to Formal Methods* John Wiley & Sons (1990).
9. Spivey, J.M. *The Z notation* Prentice-Hall (1992).
10. Lano, K. and Haughton, H. *Specification in B, An Introduction Using the B Toolkit* Imperial College Press (1995).
11. Scharbach, P.N. *Formal Methods: Theory and Practice, In Formal Methods: Theory and Practice,* P.N. Scharbach(Editor), BSP Professional Books, pp 2-4 (1989).
12. Spivey, J.M. *Understanding Z: A Specification language and its formal semantis* Cambridge University Press (1988).
13. Jia, X. "An approach to animating Z specifications", *In Proceedings of 19th Annual International Computer Softwrare and Applications Conference,* Dallas, Texas, U.S.A., August, (1995). Also electronically available at: ftp://ise.cs.depaul.edu.
14. Hewitt, M.A. "Automated animation of Z using prolog. Technical Report, Department of Computing, Lancaster University, UK, (1991).
15. Troelstra, A.S. and VanDalen, D. "Constructivism in Mathematics", **II**. North-Holland Studies in Logic and Foundations of Mathematics, **123** (1988).
16. Troelstra, A.S. and van Dalen, D. "Constructivism in Mathematics", **I**. North-Holland Studies in Logic and Foundations of Mathematics **123,** (1988).
17. Beeson, M.J. *Foundations of Constructive Mathematics.* Springer-Verlag, (1985).

۱۸. اردشیر، محمد. (مترجم). شهودگرایی و صورتگرایی، نوشته‌ی اِخبِرتوس یان‌براور، نشر ریاضی، سال ۹، شماره‌ی ۱، صص ۳۵-۳۰.

19. Mirian-Hosseinabadi, S.-H. and Turner, R. "Constructive Z." *Jourant of Logic and Computation,* **8**(1) pp 49-70 (February 1998).
20. Nordstrom, B. and Petersson, K. *Programming in Martin-Lof's Type Theory* Oxford University Press (1990).
21. Per Martin-Lof. "Constructive mathematics and computer programming". In C.A.R. Hoare and J.C. Shepherdson, editors, *Mathematical Logic and Computer Programming,* pp 167-184 Prentice-Hall (1985).
22. Mirian-Hosseinabadi, S.-H. "A cnstructive approach to set-theoretical formal specification". *In Proceedings of 4the Annual International CSI Computer Software CSICC'98,* pp 158-165, Sharif University of Technology, Tehran, Iarn (January 1999).

23. Fraenkel, A. et al, *Foundations of Set Theory* North-Holland Publishing Company, Amsterdam, Second Revised Edition (1973).

24. Damm, F. Hansen, B.S. and Bruun, H. "On Type Checking in VDM and Related Consis-tency Issues", *In the Proceedings of VDM'91*, LNCS 551, **1,** pp 45-62 (1991).

25. deVasconcelos, A.M.L. and McDermid, J.A. "Incremental Type-checking in Z", University of York, Department of Computer Science, Computer Science Report, YCS 185, (1992).

26. Dekkers, W. *Typed Lambda Calculi,* Fourth European Summer School in Logic, Language and Information, University of Essex (August 1992).

27. Spivey, J.M. and Sufrin, B.A. "Type Inference in Z", In *the Proceedings of VDM'90, VDM and Z-Formal Methods in Software Development*, LNCS 425, Springer-Verlag, pp. 426-451, (April 1990).

28. Turner, R. *A Logic for Z,* Department of Computer Science, University of Essex (1998).

29. Nicholls, J. (Editor). *Z Notation Version 1.2,* Prepared by members of the Z Standard Panel, Electronically available from: *http://www.comlab.ox.ac.uk/ Zforum/ ZSTA/ versions/,* (September 1995).

30. Bornat, R. and Sufrin, B. *Rool your own Jape logic - Encoding Logics for the Jape proof calculator* Oxford Programming Research Group, (March 1996).

31. Breuer, P.T. and Bowen, J.P. *PRECCX USER MANUAL* Oxford University Computing Lab., (August 1994).

۳۲. جلالی قمیوانی، آرش. ترجمه نظریه‌ی مجموعه‌های CZ به نظریه‌ی انواع مارتین لوف و پیاده‌سازی آن ، پایان‌نامه کارشناسی ارشد، دانشکده مهندسی کامپیوتر، دانشگاه صنعتی شریف (آبان ماه ۱۳۷۹).

33. Weiss, D.M. and Lai, C.T.R. *Software Product -Line Engineering, A Family-Based Soft-ware Development Process* Addison-Wesley (1999).

34. Ardis, M. et al, "Sofrware product lines: a case study". *Software Practice and Experience,* **30** (7) pp 825-847 (June 2000)

۳۵. میریان حسین‌آبادی، سید حسن و سیرجانی،مرجان. طراحی و پیاده‌سازی ابزارگان توصیف، وارسی و تولید سیستم‌های نرم‌افزاری، گزارش نهایی طرح تحقیقاتی با عنوان «طراحی و پیاده‌سازی ابزارگان توصیف، وارسی و تولید سیستم‌های نرم‌افزاری»، مجموعه مقالات پژوهشی سال ۷۷، دانشکده مهندسی کامپیوتر، دانشگاه صنعتی شریف (۱۳۷۸).