فصل ۲۵ شیوه های رسمی

مفاهیم کلیدی (مرتب بر حروف الفبا)
پیش شرط ها و پس شرط ها ، توالی ها(سری) ، داده های تغییر ناپذیر ، رهنمودهای شیوه های رسمی ، زبانها ، شمای (طرح اصلی) Z ، علائم Z ، عملگرهای مجموعه عملگرهای منطقی ، عملیات ها ، مشخصات ساختاری ، وضعیت ها

KEY CONCEPTS
data invariant, constructive specification , formal method guidelines , languages , logic operators , pre- and post- conditions , sequences , set operators , states , Z schemas , Z notations

نگاه اجمالی

شیوه های رسمی چیست؟ این روش‌ها به مهندس نرم‌افزار امکان می‌دهند مشخصه‌ای خلق کند که کامل‌تر، یکنواخت‌تر و غیرمبهم‌تر از روش‌های قراردادی یا شیءگرا است. از مجموعه تئوری و عبارت منطقی برای خلق بیانیه مشخصی از حقایق استفاده می‌گردد. این مشخصه‌های ریاضی برای توان تحلیل نمود یا درستی و انسجامشان را اثبات کرد. از آن‌جا که این مشخصه‌ها با استفاده از عبارات ریاضی ایجاد می‌گردند، اساساً پیچیدگی کمتری نسبت به وضعیت های غیر رسمی ارائه شده دارند.

چه کسی عهده دار این امر می باشد؟ یک مهندس نرم‌افزار متخصص یک مشخصه رسمی را ایجاد می‌کند.

چرا شیوه های رسمی از اهمیت برخوردارند؟ در سیستم های مهم ایمنی یا مأموریتی، هرگونه نقصی ممکن است به قیمت گزافی تمام شود. ممکن است زندگی‌هایی از دست رفته یا نتایج شدید اقتصادی از صورت خرابی نرم‌افزار کامپیوتری به‌وقوع پیوندد. در چنین مواقعی، لازم است خطاها مشخص شده و این کار قبل از وارد شدن نرم‌افزار در عملیات صورت گیرد. (روش‌های رسمی، خطاهای مشخصاتی را تا حد زیادی کاهش داده و در نتیجه به‌عنوان پایه و اساس نرم‌افزاری کار می‌کنند که خطاهای بسیار محدودی در هنگام کار کاربر با آن دارا می‌باشد)

مراحل کار چیست؟ اولین مرحله در به‌کارگیری (شیوه های رسمی عبارتست از تعریف داده های نا متغیر، وضعیت و عملیاتی برای یک کارکرد سیستم. (داده های نامتغیر؛ وضعیتی است که در سراسر اجرای یک تابع درست و متغیربماند (True) و شامل مجموعه از داده ها است.) (وضعیت؛ عبارتست از اطلاعات ذخیره شده‌ای که یک تابع به آن دسترسی داشته و آن را تغییر می‌دهد.) و (عملیات؛ اعمالی هستند که در

سیستم در هنگام خواندن و یا نوشتن اطلاعات در یک وضعیت، صورت می‌گیرند) (هر عملیات به دو شرط مربوط است: یک پیش شرط و یک پس شرط) علامت‌گذاری و کارهای اکتشافی مربوط به مجموعه‌ها و مشخصات سازنده یعنی اپراتورهای مجموعه، اپراتورهای منطقی و توالی‌ها، اساس و پایه شیوه های رسمی را تشکیل می‌دهند.

محصول کار چیست؟ (مشخصات به‌نمایش درآمده به زبان رسمی مثل **Z** یا **VDM** وقتی ایجاد می‌شوند که روش‌های رسمی به‌کار گرفته می‌شوند.)

چگونه مطمئن شوم که کار را درست انجام داده ام؟ از آن‌جا که شیوه های رسمی از ریاضیات گسسته به‌عنوان مکانیزم تعیین مشخصات استفاده می‌کنند، اثبات منطقی را می‌توان در مورد کارکرد هر سیستم به‌کار گرفت تا معلوم شود که مشخصات درست هستند.

روش‌های مهندسی نرم‌افزار را می‌توان براساس طیف فرمالیته و رسمی، طبقه‌بندی کرد یعنی پیوند نه‌چندان محکمی با میزان جدی بودن و سختی ریاضی که در طول تحلیل و طراحی به‌کار گرفته شده است. به‌همین دلیل روش‌های طراحی و تحلیلی که پیش‌تر در این کتاب مورد بحث قرار گرفت همگی در انتهای غیررسمی این طیف قرار می‌گیرند. ترکیبی از نمودارها، متن‌ها، جداول و عبارات ساده برای حلق مدل‌های طراحی و تحلیل به‌کار گرفتهٔ شده اما میزان سختی ریاضی به‌کار رفته چندان زیاد نیست.

ما انتهای دیگر این طیف را که فرمالیته یا رسمی بودن است در نظر می‌گیریم. در این‌جا، مشخصات و طراحی با استفاده از نحوه و معنای رسمی توصیف شده‌اند که عملکرد و رفتار سیستم را مشخص می‌سازند. این مشخصه‌ها از نظر فرم به‌شکل ریاضی هستند. (مثلاً پیش‌بینی‌های ریاضی را می‌توان به‌عنوان پایه یک زبان رسمی اختصاصی استفاده نمود.)

در این مقدمه در مورد روش‌های رسمی، آنتونی هال **[HAL90]**[1] بیان می‌دارد:

(شیوه‌های رسمی، بحث برانگیزند. مدافعین آنها بیان می‌کنند که این روش‌ها می‌توانند تولید نرم‌افزار را دچار انقلاب کنند. مخالفانشان فکر می‌کنند که آنها بسیار مشکلند. در عین حال ، از نظر اکثر مردم، شیوه‌های رسمی به‌قدری ناآشنا هستند که قضاوت در مورد ادعاهای رقیب مشکل است.)

در این فصل، شیوه‌های رسمی را مورد بررسی قرار داده و تأثیر بالقوه‌شان را روی مهندسی نرم‌افزار در سال‌های آتی مطالعه می‌کنیم.

۲۵-۱ مفاهیم پایه

دائرة المعارف مهندسی نرم‌افزار[2] شیوه های رسمی را به صورت زیر تعریف می‌کند **[MAR94]**[3]

1.Hall,A.
2.Encyclopedia of Software Engineering
3.Maciniak,J.J.

روش‌های رسمی که در توسعه سیستم‌های کامپیوتری استفاده می‌شوند از نظر ریاضی برپایه فنونی هستند که برای توصیف خواص سیستم می‌باشند. چنین روش‌هایی چارچوبی مهیا می‌سازند که در آن افراد می‌توانند سیستم‌هایی را به‌صورت نظام مند نه بدون برنامه مشخص، توسعه و اثبات کنند.

روشی رسمی است که دارای پایه و اساس بی‌نقص ریاضی باشد به‌ویژه که با یک زبان رسمی مشخصاتی ارائه شده باشد. این بنیان وسیله‌ای برای تعریف دقیق عباراتی هم‌چون انسجام و کامل بودن، مشخصات، اجرا و درستی فراهم می‌سازد.

خواص مطلوب یک مشخصه رسمی مانند نبود ابهام، انسجام و تکمیل بودن، از اهداف هم روش‌های مشخصاتی است. استفاده از روش‌های رسمی احتمال رسیدن به این ایده آل را افزایش می دهد. شیوه نحوی زبان مشخصاتی باعث می‌شود که طراحی یا محتصات تنها به یک شیوه تفسیر شود که ابهامی را که اغلب وقتی یک زبان طبیعی یا عبارت گرافیکی باید برای خواننده تفسیر شود، رفع می‌کند. تسهیلات توصیفی نظری مجموعه و عبارت گرافیکی بیان صریح حقایق را ممکن می‌سازد. به‌منظور انسجام از نظر ریاضی با اثبات اینکه حقایق اولیه را می‌توان به‌صورت رسمی (با استفاده از قوانین تداخل) در حالات بعدی درون مشخصات طراحی نمود، قابلیت اطمینان برقرار می شود.

رسیدن به تکامل کار سختی است. حتی وقتی از روش‌های رسمی استفاده شود. ممکن است بعضی از جنبه‌های یک سیستم با ایجاد مشخصات، تعریف نشده باقی بمانند. ممکن است دیگر مشخصه‌ها را عمداً حذف نمود تا به طراحان آزادی عمل داد تا رهیافت پیاده سازی را انتخاب کنند. و نهایتاً در نظر گرفتن هر سناریو عملیاتی در یک سیستم پیچیده و بزرگ، غیرممکن است. احتمال دارد که آنها صرفاً از روی اشتباه حذف شوند.

گرچه رسمی سازی و فرمالیسم ارائه شده از جانب ریاضیات برای بعضی از مهندسین نرم‌افزار ظاهر دلفریبی دارد، اما بقیه در مورد بینش مبتنی بر ریاضی در تولید نرم‌افزار با شک و تردید می‌نگرند. برای این‌که بفهمیم شیوه رسمی به چه دلیل دارای مزیت است ابتدا باید نقایص مربوط به روش‌های کمتر رسمی را بررسی کنیم.

۲۵-۱-۱عدم کارآیی رهیافت های کمتر رسمی[1]

روش‌های مورد بحث برای تحلیل و طراحی در بخش‌های سه و چهار این کتاب استفاده سنگینی از زبان طبیعی و انواع نشانه‌گذاری‌های گرافیکی می‌کنند. گرچه به‌کارگیری دقیق روش‌های تحلیل و طراحی همراه با بازبینی کامل می‌تواند و مطمئناً منجربه نرم‌افزار با کیفیتی می‌شود اما نامرتبی و بی‌نظمی در

نقل قول

شیوه های رسمی دارای پتانسیل بالایی برای بهبود شفافیت و دقت مشخصات نیازمندیها و نیز یافتن خطاهای مهم و ظریف ، می باشند.

اگرچه یک شاخص خوب برای مستندات، رافع تمام نقائص ها نخواهد بود، اما به رفع آنها کمک خواهد نمود. زمانی را به ایجاد شاخصهای جامع برای مشخصه های سیستم و دیگر مستندات اختصاص دهید

۱. این قسمت و قسمتهای دیگر این فصل مطابق کار انجام شده توسط دارل اینس برای نسخه چهارم اروپایی کتاب مهندسی نرم افزار : رهیافت متخصصین ، می باشد.

به‌کارگیری این روش‌ها می‌تواند یک‌سری مشکلات ایجاد کند. مشخصات سیستم می‌تواند شامل تضادها، ابهامات، پیچیدگی‌ها ، حالات نا تمام و سطوح مختلفی از انتزاع و تجرید باشند.

تعارض و تضادها[1] مجموعه‌ای از حالاتی هستند که با یکدیگر متفاوتند. مثلاً، یک بخش از مشخصات سیستم بیان می‌دارد که سیستم باید تمام دماهای درون راکتور شیمیایی را کنترل کند در حالی‌که بخش دیگر که احتمالاً توسط فرد دیگری از گروه نوشته شده می‌گوید که تنها دماهای دامنه معینی از سیستم باید کنترل گردند. معمولاً تضادهایی که در یک صفحه از مشخصات سیستم رخ می‌دهند را می‌توان به آسانی تشخیص داد. اغلب تضادها توسط صفحات بی‌شماری جدا می‌شوند.

ابهام[2] حالاتی هستند که به شیوه‌های مختلفی تفسیر می‌گردند. مثلاً حالت زیر نوعی از ابهام است:

هویت اپراتور شامل نام اپراتور و کلمه عبور است. کلمه عبور شامل شش رقم می‌باشد. وقتی اپراتور به سیستم متصل می‌شود باید این آن را روی سیستم ایمنی ***VDU*** *به‌نمایش درآورده و در قابل اتصال قرار دارد.*

در این حالت آیا کلمه ((آن)) به کلمه عبور یا هویت اپراتور اشاره می کند؟

نامعلوم[3] اغلب از آن‌جایی رخ می‌دهد که مشخصات یک سیستم یک سند قطور می‌شود. دست یافتن به انسجامی سطح بالا تقریباً یک کار غیر ممکن است. این امر می‌تواند منجربه حالاتی هم‌چون این موارد شود:

نقل قول

انسان ممکن الخطاست، تکرار خطا نیز کار اوست.
مالکوم فور برز

«رابطی که در سیستم توسط اپراتورهای رادار استفاده می‌شود باید کاربرپسند و مبتنی بر مفاهیم کلی ساده باشد.» یا «رابط محاری باید براساس مفاهیم کلی باشد که از نظر درک و استفاده ساده هستند و از نظر تعداد معدود می‌باشند.» مرور این حالات به‌صورت تصادفی باعث تشخیص کمبود اطلاعات مفید نهفته نمی‌شود.

کامل نبودن[4] احتمالاً یکی از مشکلاتی است که در اکثر موارد در مورد مشخصات سیستم رخ می‌دهد. مثلاً، نیازمندیهای کارکردی زیر را درنظر بگیرند:

سیستم باید سطح مخازن را از سنسورهای عمق که در مخزن قرار دارد، به‌طور ساعتی بدست بیاورد. این مقادیر باید در مورد شش‌ماه گذشته ذخیره و ضبط شوند.

این کار بخش اصلی اطلاعات ذخیره شده یک سیستم را توصیف می‌کند.

اگر یکی از فرمان‌های سیستم عبارت بود از عملکرد فرمان ***Average*** *که عبارتست از نمایش سطح آب در مورد سنسور به‌خصوصی بین دو فاصله زمانی بر روی صفحه مونیتور کامپیوتر.*

1.Contradictions
2.Ambiguities
3.Vagueness
4.Incompleteness

با فرض این‌که جزئیات بیشتری برای این فرمان ارائه نشده باشد، جزئیات فرمان به‌شدت ناقصند. مثلاً توصیف فرمان شامل آنچه که درصورتی رخ می‌دهد که کاربر یک سیستم، زمانی را مشخص می‌کند که زودتر از شش ماه قبل از زمان کنونی است، نمی‌شود.

سطوح انتزاعی و تجریدی مخلوط[1] وقتی رخ می‌دهند که حالات تفکیک شده متعدد به‌صورت تصادفی با حالاتی که در سطح بسیار پایین‌تری قرار دارند، مخلوط می‌شوند. مثلاً وضعیتی مثل:

هدف از سیستم عبارتست از پی‌گیری موجودی انبار.

که ممکن است با مورد زیر مرتبط شود.

وقتی مسئول بارگیری فرمان **Withdraw** را تایپ می‌کند با شماره سفارش، هویت قلمی که باید جابه‌جا شود و کمیت جابه‌جا شده، ارتباط برقرار می‌کند. سیستم با تأیید این‌که جابه‌جایی مجاز است، پاسخ می‌دهد.

در حالیکه چنین وضعیت‌هایی در مشخصات سیستم مهم هستند، تعیین‌کننده‌ها اغلب آنها را تاحدی با هم مرتبط می‌سازند که دیدن معماری عملیاتی کلی سیستم بسیار مشکل می‌شود.

هر یک از این مسائل رایج‌تر از آن چیزی است که مایلیم باور کنیم. و هر یک نمایانگر یک نقص بالقوه در روش‌های قراردادی و شیءگرا برای تعیین مشخصه ها می باشد.

استفاده از فنون بازبینی رسمی طی فاز تعیین مشخصات سیستم بسیاری از مشکلات را مرتفع می سازد، هرچند برخی از آنها پوشش داده نشوند. بنابرین طی فازهای طراحی، برنامه نویسی و آزمون نیز مراقب دوری از خطا باشید.

۲۵-۱-۲ ریاضیات در توسعه نرم افزار

ریاضیات دارای خواص مفید متعددی است که برای تولیدکنندگان سیستم‌های بزرگ است. یکی از مفیدترین خواص آن عبارتست از این‌که قادر به توصیف معجزه دقیق یک موقعیت فیزیکی، یک شیء یا نتیجه یک عمل است. به‌طور ایده‌آل مهندس نرم‌افزار در همان موقعیت یک ریاضی‌دان تخصصی است. باید مشخصات ریاضی سیستم، به‌نمایش درآمده و راه‌حلی از نظر معماری نرم‌افزاری ارائه گردد که مشخصات تولیدی را پیاده سازی کنند.[2]

مزیت دیگر استفاده از ریاضی در فرآیند نرم‌افزاری این است که انتقال یکنواختی بین فعالیت‌های مهندسی نرم‌افزار به‌وجود می‌آورده نه‌تنها مشخصات عملیاتی بلکه طراحی‌های سیستم را می‌توان به‌صورت ریاضی بیان نمود و البته کد برنامه تک عبارت ریاضی است، البته یک عبارت بلندتر.

خاصیت عمده ریاضی این است که از تجرید پشتیبانی نموده و یک محیط عالی برای مدل‌سازی است. از آن‌جا که این یک محیط واسط عالی است احتمال پیچیدگی کم است، مشخصات را می‌توان به‌صورت ریاضی از نظر تضادها و نواقص ارزیابی کرده و ابهام به‌طور کامل از بین می‌رود. علاوه بر این

---

1. Mixed levels of abstraction

۲. کلمه اخطار برای این نقطه مناسب می نماید. نمی توان مشخصه های ریاضی سیستم که در این فصل ارائه شده اند، بصورت یک عبارت ساده ریاضی خلاصه نمود. سیستم های نرم افزاری بسیار پیچیده هستند، و واقعیت آن است که نمی توان آنها را در یک خط ریاضی توصیف کرد.

می‌توان از ریاضی برای نمایش سطوح انتزاع و تجرید در مشخصات سیستم به شکلی سازمان یافته، استفاده نمود.

ریاضیات یک ابزار مناسب برای مدل‌سازی است. باعث می‌شود که اصول و کلیات مشخصه‌ها به نمایش گذارده شده و به تحلیلگر و تعیین‌کننده سیستم کمک می‌کند مشخصات را از نظر کارکردی، بدون موضوعاتی مثل زمان واکنش، رهنمودهای طراحی و پیاده سازی و محدودیت‌های پروژه، بررسی نماید. هم‌چنین به طراح کمک می‌کند زیرا مشخصات طراحی سیستم خواص مدل را نشان می‌دهند و این کار تنها جزئیات کافی را برای انجام وظیفه در دست اجرا مهیا می‌سازد.

نهایتاً، ریاضیات سطح اعتباری بالایی را وقتی به‌عنوان محیط تولید نرم‌افزار مورد استفاده قرار می‌گیرد مهیا می‌سازد. ممکن است از اثبات ها و تصدیقات ریاضی برای تشریح این‌که طراحی با مشخصاتی سازگاری دارد یا خیر یا این‌که کد برنامه‌ای، انعکاس درست طراحی است یا خیر، استفاده نمود. این کار در شیوه جاری ارجحیت دارد، که در آن اغلب تلاش کمی صرف اعتبارسنجی اولیه می‌شود و بیشتر کار کنترل نمودن سیستم نرم‌افزار در طول آزمون پذیرش و سیستم رخ می‌دهد.

۳-۱-۲۵ مفاهیم شیوه های رسمی

هدف از این بخش ارائه مفاهیم اصلی درگیر در مشخصات ریاضی سیستم‌های نرم‌افزاری است بدون این‌که خواننده را با ازدحام جزئیات ریاضی دچار توقف کنیم. به‌منظور دستیابی به این امر، از چند مثال ساده بهره می‌گیریم:

مثال ۱ : جدول نمادها . برنامه‌ای برای نگهداری جدول نمادها استفاده می‌شود. چنین جدولی اغلب بنا به انـــواع کاربردهای مختلف مورد استفاده قرار می‌گیرد. این برنامه شامل مجموعه‌ای از قلم‌ها بدون هیچ‌گونه نسخه‌برداری است. نمونه‌ای از این جدول در شکل ۲۵-۱ نشان داده شده است که نمایانگر جدول مورد استفاده سیستم عامل برای نگهداری اسم کاربران سیستم می‌باشد. سایر نمونه‌هایی از این جداول شامل مجموعه‌ای از اسامی کارکنان در سیستم پرداخت دستمزد، مجموعه‌ای از اسامی کامپیوترها در سیستم ارتباطات شبکه‌ای و مجموعه‌ای از مقصدهایی در یک سیستم، برای تهیه و ارائه جداول زمانی راه‌آهن، می باشد. فرض کنید که جدول ارائه شده در این مثال شامل بیشتر از اعضای کارکنان **MaxIds** نباشد. این وضعیت که محدودیتی بر جدول اعمال می‌کند، جزئی از شرطی است که به‌عنوان **data invariant** شناخته می‌شود ـ نظریه مهمی که بعداً در طول این فصل دوباره به آن رجوع می‌کنیم.

یک داده تا مغیر مجموعه ای از شرایط است که طی اجرای سیستم شامل مجموعه ز داده ها با مقدار صحیح خواهد بود.

این اطلاعات تغییرناپذیر یا **data invariant** شرطی است که در سراسر اجرای سیستم برقرار است و شامل مجموعه‌ای از اطلاعات می‌باشد. این شرط که برای جدول علایم مورد بحث فوق صادق است دارای دو جزء می‌باشد:

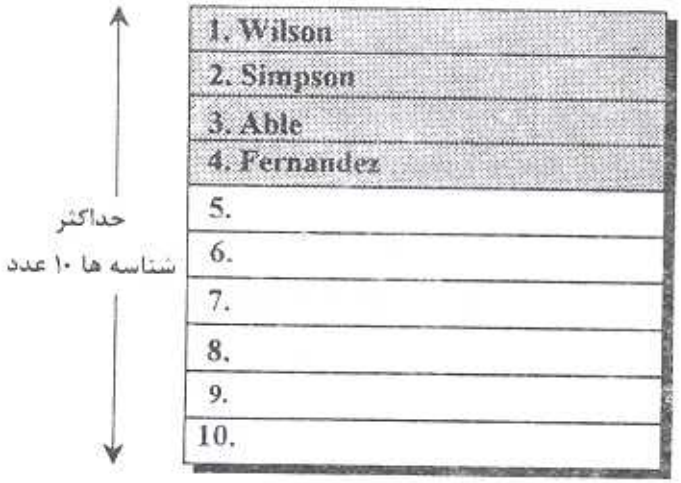
۱) جدول شامل چیزی غیر از اسامی **MaxIds** نخواهد بود

۲) اینکه اسامی کپی در جدول نخواهند بود. در مورد برنامه جدول نمادها که توضیح داده شد، یعنی این‌که بدون توجه به این‌که چه وقت جدول علایم در طول اجرای سیستم بررسی می‌شود، همیشه شامل چیزی بیشتر از اسامی شناسه کارکنان **MaxIds** نبوده و کپی نیز ندارد.

مفهوم مهم دیگری که مدنظر است **State** یا وضعیت است. در متن روش‌های رسمی،[1] وضعیت عبارتست از اطلاعات ذخیره شده‌ای که سیستم به آنها دسترسی داشته و آن را تغییر می‌دهد. در مثال برنامه جدول نمادها، وضعیت همان جدول نماد است.

در شیوه های رسمی، یک " وضعیت " داده ای که قابل دستیابی و تغییر است در خود ذخیره می کند. یک " عملیات " اقدامی است که داده را در یک وضعیت می خواند یا می نویسد.

مفهوم نهایی **operation** یا عملیات است. این عملی است که در سیستم صورت گرفته و اطلاعات را در یک وضعیت نوشته و می‌خواند. اگر برنامه جدول علایم با افزودن و حذف اسامی کارکنان از جدول مرتبط باشد، با دو عمل رابطه دارد یعنی: عملیات افزودن اسم خاصی به جدول و عملیات حذف اسم موجود از جدول. اگر برنامه تسهیلاتی برای چک کردن این مطلب مهیا کند که آیا نام مشخصی در جدول هست یا نه، عملیاتی وجود خواهد داشت که اشاراتی دارد به این‌که آیا نام در جدول بود یا خیر.



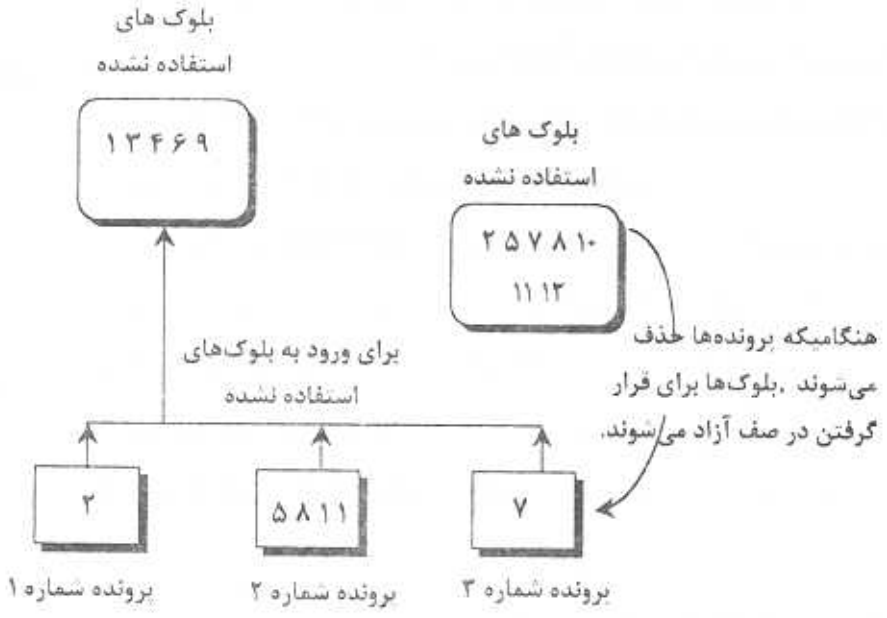
شکل ۲۵ـ۱ یک جدول نمادها که برای یک سیستم عامل مورد استفاده دارد

هر عملیات با دو شرط مرتبط است: **پیش شرط**[2] و **پس شرط**[1]. **پیش شرط** شرایطی را تعریف می‌کند که در آن عملیات خاص معتبر است. مثلاً، پیش شرط برای عملیاتی که نامی را به علایم شناسه جدول اضافه می‌کند، در صورتی معتبر است که نام افزوده شده در جدول نباشد و هم‌چنین اگر تعداد کمتری از شناسه‌های کارکنان **MaxIds** در جدول وجود نداشته باشد. **پس شرط** یک عملیات تعریف می‌کند که چه اتفاقاتی بعد از تکمیل شدن عملی توسط عملیات، رخ می‌دهند. این کار با تعریف اثر آن روی وضعیت

یک " پیش شرط " چگونگی اعتبار یک عملیات مشخص را تعریف می کند. " پس شرط " توضیح می دهد که پس از تکمیل اقدامات یک عملیات چه اتفاقی روی می دهد.

۱. به خاطر آورید که اصطلاح وضعیت در فصلهای ۱۲ و ۲۱ به عنوان بازنمای رفتار یک سیستم یا شئ به کار برده شد.
2. Precondition

صورت می‌گیرد. در مثالی از یک عملیات که شناسه ای را به جدول می‌افزاید، پس شرط از نظر ریاضی مشخص می‌کند که به جدول شناسه‌های جدید اضافه شده است.



شکل ۲۵ـ۲ یک اداره کننده بلوک

مثال ۲ : Block hander. یکی از مهم‌ترین بخش‌های سیستم عملیاتی کامپیوتر، سیستم فرعی است که فایل‌هایی را نگه می‌دارد که اخیراً توسط کاربر به‌وجود آمده‌اند. بخشی از سیستم فرعی کار با فایلها، **block hander** است. فایل‌های ذخیره شده متشکل از بلوک‌های ذخیره سازی هستند که در وسیله ذخیره فایل نگه‌داری می‌شوند. در طول عملیات کامپیوتر، فایل‌ها ایجاد و حذف می‌شوند که این کار مستلزم بدست آوردن و آزادسازی بلوک‌های ذخیره‌سازی است. به‌منظور تطابق با این کار، سیستم فرعی کار با فایلها مخزنی از بلوک‌های استفاده نشده دارد و مسیر بلوک‌هایی را که در حال حاضر در حال استفاده هستند، پی‌گیری می‌کند. وقتی بلوک‌ها از یک فایل حذف شده آزاد می‌شوند، معمولاً به یک دسته بلوک افزوده می‌شوند که منتظرند به یک مخزن بلوک‌های استفاده نشده، اضافه شوند. این عمل از شکل ۲۵-۲ نشان داده در این شکل تعدادی از اجزاء نشان داده شده اند: مخزن بلوک‌های استفاده نشده، بلوک‌هایی که درحال حاضر فایل‌هایی را تشکیل می‌دهند که تحت‌کنترل سیستم عامل هستند و بلوک‌هایی که منتظرند به مخزن اضافه شوند. بلوک‌های درحال حاضر انتظار، در یک صف قرار دارند همراه با هر عنصر این صف‌بندی مجموعه‌ای از بلوک‌های فایل حذف شده خواهد بود.

1.postcondition

در مورد این سیستم فرعی، وضعیت عبارتست از مجموعه بلوک‌های آزاد، مجموعه‌ای از بلوک‌های استفاده نشده و یک صف از بلوک‌های بازگردانده شده. اطلاعات تغییرناپذیر که به زبان طبیعی بیان شده‌اند عبارتند از:

- هیچ بلوکی دارای هر دو علامت « استفاده شده و استفاده نشده» نیست.
- همه مجموعه بلوک‌هایی که در یک صف هستند زیر مجموعه‌ای از مجموعه بلوک‌هایی هستند که در حال حاضر استفاده می‌شوند.
- هیچ عنصری از یک صف نیست که شامل تعداد بلوک‌های یکسانی باشد.
- مجموعه ای از بلوک‌های استفاده شده و استفاده نشده‌ها، مجموعه کل بلوک‌هایی هستند که فایلها را تشکیل می دهند.
- هیچ کپی از تعداد بلوک‌های استفاده نشده در مجموعه وجود ندارد.
- هیچ کپی از تعداد بلوک‌های استفاده شده در مجموعه وجود ندارد.

بعضی از عملیات مربوط به این اطلاعات عبارتند از:

- عملیاتی که مجموعه‌ای از بلوک‌ها را به انتهای صف اضافه می‌کند.
- عملیاتی که مجموعه‌ای از بلوک‌های استفاده شده را از جلوی صف برداشته و در مجموعه بلوک‌های استفاده نشده قرار می‌دهد.
- عملیاتی که بررسی می‌کند آیا صف بلوک‌ها خالی است یا خیر.

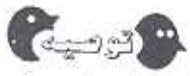
پیش شرط اولین عملیات این است که بلوک‌هایی باید به مجموعه بلوک‌های استفاده نشده، افزوده شوند. پس شرط این است که باید مجموعه‌ای از بلوک‌ها به‌انتهای صف بروند.

پیش شرط دومین عملیات این است که صف باید حداقل دارای یک قلم در آن باشد. پس شرط نیز این است که باید بلوک‌ها به مجموعه بلوک‌های استفاده نشده افزوده شوند.

عملیات نهایی که در آن چک می‌شود که آیا صف بلوک‌های بازگشتی خالی است یا نه، دارای پیش‌شرطی نیست. یعنی عملیات همیشه تعریف شده است بدون توجه به این‌که این وضعیت دارای چه ارزشی است. پس شرط ارزش واقعی را درصورتی ارائه می‌دهد که صف خالی باشد در غیر این‌صورت جواب غلط خواهد بود.

مثال ۳ : **Print Spooler**. (برنامه یا دستگاه خاصی که به کامپیوتر اجازه می‌دهد تا به هنگام اجرای کار دیگر، نسخه خاصی روی حافظ بوفر کند ـ برنامه ردیف‌گر)

در سیستم‌های عامل چندکاره، در چند کار تقاضای چاپ قابل می‌شود. اغلب دستگاه چاپ کافی برای انجام تمام این کارهای چاپی به‌طور همزمان وجود ندارد. هر تقاضای چاپی که فوراً انجام نمی‌شود،در یک صف قرار می‌گیرد که منتظر چاپ شدن خواهد بود. بخشی از سیستم عملیاتی که با اجرای چنین صف‌هایی سر و کار دارد و به‌عنوان برنامه اسپولر(جرخشگر چاپ) شناخته می‌شود.

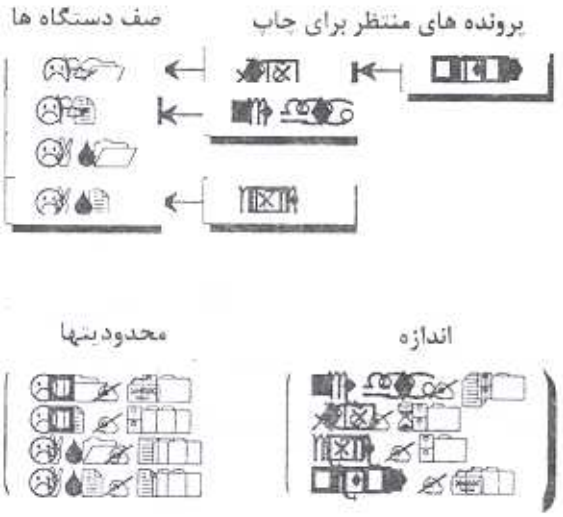


فنون " برین استورمنگ ". هنگامی که شما نیاز به داده های ثابت برای یک کارکرد پیچیده دارید، می تواند خوب کار کند. تعدادی افراد در اختیار داشته باشید که حدود و تعور- محدودیتها و شرایط را برای فانکشن ها مشخص سازند و آنگاه آنها را ترکیب و ویرایش نمایند.

نقل قول

اگر سازمانی به دنبال تحقیقات ریشه ای در امر استفاده مجدد باشد، لازم است که بداند در هر سیستم خاص چه اجرایی مورد نیاز است؟ دیوید رینه

در این مثال فرض می‌کنیم که سیستم عامل نمی‌تواند بیشتر از یک خروجی دستگاه **MaxDevs** را به‌کار گرفته و هر وسیله دارای یک صف مربوطه است. هم‌چنین فرض می‌کنیم که هر وسیله با تعداد خط‌های محدودی در یک فایل برای چاپ آن، مرتبط است. گاهی چرخشگر چاپ این محدودیت را به‌منظور پرهیز از انجام کارهای چاپی بزرگی انجام می‌دهند که ممکن است یک دستگاه جانگر کوچک را برای مدت طولانی مشغول سازد. طرح شماتیک این چرخشگر چاپ در شکل ۲۵-۳ آمده است.



شکل ۲۵ـ۳یک مدیر چاپ (چرخشگر)

با توجه به شکل، چرخشگر دارای چهار جزء است: صفی از فایل‌هایی که منتظر چاپ هستند، هر صف با یک وسیله خروجی خاص مرتبط است، مجموعه‌ای از وسایل خروجی که تحت کنترل اسلوپر هستند، ارتباط بین وسایل خروجی و حداکثر اندازه فایلی که می‌تواند چاپ شود و ارتباط بین فایل‌های منتظر چاپ و اندازه آنها از نظر خطوط. مثلاً شکل۲۵-۳ نشانگر وسیله خروجی **LP1** است که دارای محدودیت ۷۵۰ خطی بوده و دارای فایل‌های منتظر **Ftax** و **Persons** است و اندازه فایل‌ها به‌ترتیب ۶۵۰ و ۷۰۰ خط می‌باشد.

وضعیت ها و عملیات ها قیاسی بوده و برای سیستم شئ گرا بصورت های مختلفی می توانند رده بندی شوند. وضعیت ها، حوزه داده (صفات خاصه) و عملیات، فرآیند(شیوه ها) را که با داده ها کار می کنند، نشان می دهند.

وضعیت چرخشگر توسط چهار جزء به‌نمایش درآمده است، وسایل خروجی، صف‌ها، محدودیت‌ها و اندازه.

اطلاعات تغییر ناپذیر دارای ۵ جزء هستند:

- هر وسیله خروجی با یک سرحد بالایی در مورد خطوط چاپی مرتبط است.
- هر وسیله با یک صف احتمالاً غیرخالی از فایل‌های منتظر چاپ مرتبط است.
- هر فایل دارای یک اندازه است.

- هر صف با یک وسیله خروجی شامل فایل‌های ارتباط دارد که دارای اندازه‌ای کمتر از سرحد بالایی وسیله خروجی هستند.
- چیز دیگری به‌غیر از وسایل **MaxDevs** که تحت‌کنترل چرخشگر هستند به‌عنوان وسیله خروجی وجود ندارد.

می‌توان چندین عملیات را با هر چرخشگر مربوط کرد. مثلاً:

- عملیاتی که وسیله خروجی جدیدی را به چرخشگر همراه با سرحد چاپی مربوطه آن می‌افزاید.
- عملیاتی که فایلی را از صف مربوطه با وسیله خروجی خاصی انتقال می‌دهند.
- عملیاتی که فایلی را به‌وسیله دستگاه خروجی به‌خصوصی به صف می‌افزایند.
- عملیاتی که محدودیت چاپ تعداد خطوط یک دستگاه خروجی را تغییر می‌دهند.
- عملیاتی که فایل را از صف مربوط به یک دستگاه خروجی به صف مربوطه به دستگاه دوم انتقال می‌دهند.

هر یک از این عملیات با عملکرد چرخشگر ارتباط دارد. مثلا ، اولین عملیات با چرخشگری مرتبط است که به‌وسیله دستگاه جدیدی که به کامپیوتر حاوی سیستم عامل اجراکننده چرخشگر متصل است، مطلع شده است.

هم‌چون قبل، هر عملیات با یک پیش‌شرط و پس‌شرط مرتبط است. مثلاً پیش‌شرط اولین عملیات این است که نام وسیله خروجی هم‌اکنون وجود ندارد و درحال حاضر تعداد وسایل کمتری نسبت به وسایل کمتری نسبت به وسایل خروجی **MaxDevs** برای چرخشگر شناسایی شده اند. پس‌شرط این است که نام وسیله جدید به مجموعه نام‌های موجود وسیله افزوده شده، اطلاعات جدیدی برای وسیله بدون این‌که فایلی به این صف مربوط شود تشکیل می‌شود و وسیله با محدودیت چاپ آن مرتبط می‌گردد.

پیش‌شرط دومین عملیات این است که وسیله برای چرخشگر شناخته شده و حداقل یک ورودی در صف مربوط به این وسیله وجود داشته باشد. پس‌شرط این است که سر صف که با وسیله خروجی در ارتباط است برداشته شده و ورودی آن در بخشی از چرخشگر که مسیر اندازه فایل‌ها را ضبط می‌کند، پاک شود.

پیش شرط عملیات پنجم که در بالا توصیف شده عبارتست از: (برداشتن فایلی از یک صف مربوط به وسیله‌ای خروجی و انتقال آن به صف دیگری مربوط به دومین وسیله خروجی).

- اولین وسیله خروجی برای چرخشگر شناسایی می‌شود.
- دومین وسیله خروجی برای چرخشگر شناسایی می‌شود.
- صف مربوط به اولین وسیله حاوی فایلی است که باید پاک شود.
- اندازه فایل کمتر یا مساوی محدودیت چاپی مربوط به دومین وسیله خروجی است.

پس‌شرط این است که فایل باید از یک صف پاک شده و به صف دیگر اضافه شود.

در هر یک از مثال‌های آورده شده در این بخش مفاهیم اصلی مشخصات رسمی را معرفی کردیم. اما این کار بدون تأکید بر علم ریاضی لازم برای انجام این مشخصات رسمی، صورت گرفت. در بخش بعدی این موارد ریاضی را بررسی خواهیم کرد.

۲۵-۲ ریاضیات مقدماتی

برای به‌کارگیری مؤثر روش‌های رسمی، باید مهندس نرم‌افزار دارای شناخت کاری از عبارات ریاضی مربوط به مجموعه‌ها و توالی‌ها و عبارت منطقی به‌کار رفته در محاسبات پیش‌بینی و مسندی باشد. هدف از این بخش مهیا ساختن مقدمه‌ای جزئی است. در مورد بحث دقیق‌تر خوانندگان را به بررسی و مطالعه کتاب‌های مربوط به این موضوع ارجاع می‌دهیم (مثل [wil87][1]، [GRI93][2] و [ROS95][3])

۲۵-۲-۱ مجموعه ها و مشخصات ساختاری

مجموعه، جمعی از اشیا یا عناصر است و به‌عنوان یک سنگ بنای روش‌های رسمی است. عناصر موجود در یک مجموعه منحصر به فرد و یکتا می باشند (یعنی نسخه‌برداری و کپی آنها مجاز نیست). مجموعه‌هایی با تعداد اندکی عناصر، در داخل پرانتزهایی نوشته شده و هر یک از عناصر توسط ویرگول از هم‌جدا می‌شوند. مثلاً مجموعه زیر:

{ C++ , Pascal , Ada , COBOL , Java }

شامل اسامی پنج زبان برنامه‌سازی است.

ترتیب ظاهر شدن عناصر درون یک مجموعه چندان مهم نیست. تعداد قلم‌های مجموعه به‌عنوان کاردینالیته آن شناخته می‌شوند. شماره اپراتور به کاردینالیته[4] مجموعه بارمی‌گردد. مثلاً عبارت

{ A , B , C , D } = 4

نشانگر این است که اپراتورهای کاردینالیته، در مجموعه نشان داده به‌کار رفته و نتیجه نشانگر تعداد قلم‌های مجموعه است. دو شیوه برای تعریف مجموعه وجود دارد. ممکن است مجموعه‌ای با شمارش عناصرش تعریف شود (این شیوه‌ای است که در آن مجموعه‌های فوق‌الذکر تعریف شده‌اند). دومین روش عبارتست از ایجاد یک مشخصات سازنده مجموعه[5]. فرم کلی اعضای یک مجموعه عبارتست از این‌که مجموعه‌ای با استفاده از عبارت بولین مشخص شود. مشخصات سازنده مجموعه بر شمارش ارجحیت دارد

پرسش
مشخصات مجموعه ساختمانی چیست؟

1. Wiltala,S.A.
2. Gries,D. and F.B.
3. Rosen,K.H.
4. cardinality
5. constructive set specification

زیرا تعریف مختصر و مفید مجموعه‌های بزرگ را مقدور می‌سازد. هم‌چنین صراحتاً قانونی را تعریف می‌کند که در ساخت مجموعه استفاده شده بود.

مشخصات سازنده زیر را به‌عنوان مثال در نظر بگیرید:

$$\{ n : N \mid n < 3 \cdot n \}$$

این مشخصات دارای سه جزء، یک امضاء، $n : N$ ، یک گزاره $n < 3$ و یک عبارت n است. امضاء[1] مشخص‌کننده دامنه مقادیری است که در هنگام تشکیل مجموعه بررسی می‌شوند، گزاره[2] (عبارت بولین) تعریف می‌کند که چگونه مجموعه محدود می‌شود و نهایتاً عبارت **term** فرم کلی قلم مجموعه را می‌دهد. در مثال فوق **N** به‌جای اعداد طبیعی آمده است بنابراین، اعداد طبیعی باید درنظر گرفته شوند، گزاره نشانگر این است که تنها اعداد طبیعی کمتر از ۳ باید گنجانده شوند و عبارات مشخص می‌کنند که هر عنصر مجموعه به‌صورت **n** است.

بنابراین مشخصات فوق این مجموعه را تعریف می‌کنند:

$$\{ 0 , 1 , 2 \}$$

وقتی فرم عناصر یک مجموعه مشخص است، این عبارت را می‌توان حذف کرد. مثلاً، مجموعه فوق را می‌توان به‌صورت زیر مشخص کرد:

$$\{ n : N \mid n < 3 \}$$

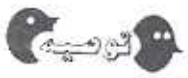
مجموعه‌هایی که در بالا آمده همگی دارای قلم‌های تکی می‌باشند. مجموعه‌ها را می‌توان از عناصری ساخت که جفت، سه تایی و غیره هستند. مثلاً، مشخصات مجموعه

$$\{ x , y : N \mid x + y = 10 \cdot (x , y^2) \}$$

مجموعه از جفت‌های اعداد طبیعی را توصیف می‌کند که دارای فرم (x , y^2) هستند و در آن مجموع x , y ۱۰ است. این مجموعه عبارتست از

$$\{ (1 , 81) , (2 , 64) , (3, 49) , \dots \}$$

مشخصاً، مشخصات سازنده مجموعه باید جزئی از نرم‌افزار کامپیوتری را نشان دهد که تا حد قابل توجهی پیچیده‌تر از موارد فوق است. فرم ابتدایی و ساختار اولیه همان می‌باشد.



دانش مجموعه عملیات، هنگام ساخت مشخصات رسمی، ضروری می نماید. زمانی را برای رسمی سازی هر عملیات اختصاص دهید.

۲۵-۲-۲ عملگرهای مجموعه

مجموعه ویژه‌ای از نشانه‌گذاری برای نمایش عملیات منطقی و مجموعه استفاده می‌شود. این نمادها باید به‌وسیله مهندس نرم‌افزاری شناسایی شوند که قصد دارد از روش‌های رسمی استفاده کند.

ایزاتور $\in$ برای اشاره به عضویت در یک مجموعه استفاده می‌شود. مثلاً عبارت

$$x \in X$$

1.signature
2.predicate

در صورتی دارای ارزش درست است که **x** عضوی از مجموعه **X** بوده در غیر این‌صورت ارزش آن نادرست است.

مثلاً : گزاره

$$12 \in \{ 6 , 1 , 12 , 22 \}$$

دارای ارزش صحیح است زیرا ۱۲ عضوی از این مجموعه است.

مخالف عملگر $\in$ ، علامت $\notin$ می باشد : عبارت :

$$x \notin X$$

در صورتی دارای ارزش صحیح است که **x** عضوی از مجموعه **X** نباشد و در غیر این صورت نادرست است.

مثلاً گزاره :

$$13 \notin \{ 13 , 1 , 124 , 22 \}$$

دارای ارزش نادرست است.

علایم $\subset$و $\subseteq$ مجموعه هایی را به‌عنوان عملوند خودشان می گیرند. گزاره

$$A \subset B$$

صحیح است اگر تمام اعضاء مجموعه **A** عضو مجموعه **B** باشد و در غیر اینصورت نادرست خواهد بود.

گزاره :

$$\{ 1 , 2 \} \subset \{ 4 , 3 , 1 , 2 \}$$

دارای ارزش صحیح است.

گزاره :

$$\{HD1 , LP4 , RC5 \} \subset \{ HD1 , RC2 , HD3 , LP1 , LP4 , LP6 \}$$

دارای ارزش نادرست است زیرا عنصری **RC5** در مجموعه سمت راست عملگر موجود نیست.

اپراتور $\subset$ مشابه $\subseteq$ است. با این تفاوت که اگر عملوندهای آن یکسان باشند دارای ارزش صحیح خواهد بود.

بنابراین ارزش گزاره :

$$\{ HD1 , LP4 , RC5 , \} \subseteq \{ HD1 , RC2 , HD3 , LP1 , LP4 , LP6\}$$

نادرست است.

گزاره :

$$\{ HD1 , LP4 , RC5 \} \subseteq \{ HD1 , LP4 , RC5 \}$$

درست است.

نقل قول

ساختارهای ریاضی زیباترین ساخته های فکر بشری است.
داگلاس هافستادلر

یک مجموعه خاص ، مجموعه تهی است Ø . این مجموعه در ریاضی عادی با صفر در ارتباط است. مجموعه تهی دارای این خاصیت است که زیر مجموعه‌ای از هر مجموعه دیگر است. دو مشخصه مفید مربوط به مجموعه تهی عبارتند از:

$$\emptyset \cup A = A \, , \, \emptyset \cap A = \emptyset$$

که در مورد هر مجموعه **A** ، علامت **U** که به‌عنوان اتحاد[1] شناخته می‌شود گاهی به‌عنوان **cup** (از آن روی که به شکل جام است م) آن را می‌شناسند. علامت ∩، اپراتور اشتراک[2] یا **cap** است.

علامت **U** اتحاد دو مجموعه را گرفته و مجموعه‌ای تشکیل می‌دهد که حاوی تمام عناصر مجموعه با نسخه‌های تکراری است که حذف شده‌اند. بنابراین نتیجه عبارت:

\{ File1 , File2 , Tax , Compiler } U { New tax , D2 , D3 , File2 }

می‌شود:

{ File1 , File2 , Tax , Compiler , New Tax , D2 , D3 }

علامت ∩ اشتراک، دو مجموعه را گرفته و از موارد مشترک میان آنها مجموعه‌ای شامل عناصر مشترک می‌سازد.

بنابراین عبارت:

{ 12 , 4 , 99 , 1 } ∩ { 1 , 13 , 12 , 77 }

منجر می‌شود به مجموعه:

{ 1 , 12 }

علامت تفاضل[3] ، \ ، همان‌گونه که اسم آن نشان می‌دهد، با پاک کردن عناصر دومین عملوند از عناصر اولین عملوند آن، مجموعه‌ای را تشکیل می‌دهد.

بنابراین ارزش عبارت:

{ New , Old , TaxFile , SysParam } \ { Old , SysParam }

منجر می‌شود به مجموعه:

{ New , TaxFile }

ارزش عبارت:

{ a , b , c , d } ∩ { x , y }

مجموعه Ø (تهی) است. اپراتور همیشه یک مجموعه را ارائه می‌دهد، در این مورد عناصر مشترکی بین عملوندهای آن وجود ندارد به‌طوری‌که مجموعه بدست آمده دارای عنصری نیست.

1.union operator
2.intersection operator
3.set difference

اپراتور نهایی $\times$، ضرب است که گاهی به‌عنوان ضرب کارتزین[1] نیز شناخته می‌شود. دارای دو علموند است که مجموعه‌ای از جفت‌ها هستند. نتیجه مجموعه‌ای از جفت‌ها است که در آن هر جفت شامل عنصری است که اولین عملوند گرفته شده با عنصری از دومین عملوند ترکیب شده است. نمونه‌ای از عبارت حاصل ضرب عبارت است از:

$$\{1,2\} \times \{4,5,6\}$$

نتیجه این عبارت می‌شود:

$$\{(1,4),(1,5),(1,6),(2,4),(2,5),(2,6)\}$$

توجه داشته باشید که هر عنصری از اولین عملوند با هر عنصری از دومین عملوند ترکیب شده است.

یکی از مفاهیم که برای روش‌های رسمی مهم است، مفهوم **Power Set** (مجموعه توان) است. مجموعه توان یک مجموعه از زیر مجموعه‌های آن مجموعه است. علامت مورد استفاده برای اپرانور **Power Set** در این فصل $\mathbb{P}$ است. این یک اپراتور **unary** (تک عملونده) است که وقتی در مجموعه‌ای به‌کار گرفته می‌شود، مجموعه زیر مجموعه‌های عملوند آن را برمی‌گرداند. مثلاً:

$$\mathbb{P}\{1,2,3\} = \{\ ,\{1\},\{2\},\{3\},\{1,2\},\{1,3\},\{2,3\},\{1,2,3\}\}$$

زیرا تمام مجموعه‌ها زیر مجموعه ای است از:

$$\{1,2,3\}$$

۳-۲-۲۵ عملگرهای منطقی

جزء مهم دیگر روش رسمی، منطق است: خبر عبارت درست و نادرست. مفهوم اپرانورهای منطقی رایج به‌درستی توسط هر مهندس نرم‌افزار شناخته می‌شود. اپرانورهای منطقی که با زبان برنامه‌نویسی رایج مرتبطند با استفاده از نشانه‌های موجود روی کیبوردها نوشته می‌شوند. اپراتورهای ریاضی معادل آنها عبارتند از:

- $\wedge$ **and** (و)
- $\vee$ **or** (یا)
- $\neg$ **not** (نه)
- $\Rightarrow$ **implies** (نتیجه می دهد)

تعیین مقادیر سراسری[2] شیوه‌ای برای ایجاد جمله ای در مورد عناصر یک مجموعه‌ای است که برای هرعنصری از مجموعه درست است. این شیوه از نشانه $\forall$ (سور عمومی)استفاده می‌کند.

نمونه ای از کاربرد آن عبارتست از:

1.Cartesian product
2.Universal quantification

$$\forall i, j : N \cdot i > j \rightarrow i^2 > j^2$$

که بیان می‌کند برای هر جفت از مقادیر مجموعه اعداد طبیعی اگر **i** بزرگتر از **j** باشد آن‌وقت i^2 بیشتر از j^2 است.

۴-۲-۲۵ دنباله ها

دنباله یک ساختار ریاضی است که این حقیقت را مدل‌سازی می‌کند که عناصر آن مرتب و به ترتیب هستند. هر دنباله مجموعه‌ای از جفت‌هایی است که عناصر آنها از ۱ تا بالاترین عدد عنصر هستند. بطور مثال:

{(استاوز , 4) , (شاپیرو , 3) , (ویلسون , 2) , (جونز , 1)}

یک دنباله است. فلم‌هایی که اولین عناصر جفت‌ها را تشکیل می‌دهند، مجموعاً به‌عنوان دامنه[1] دنباله و مجموعه دومین عناصر به‌عنوان برد[1] دنباله شناخته می‌شوند. در این کتاب این دنباله ها با استفاده آکولادهای زاویه‌دار، تخصیص یافته‌اند. مثلاً ، توالی فوق معمولاً به‌صورت زیر نوشته می‌شود.

⟨ استاوز، شاپیرو، ویلسون، جونز⟩

برخلاف مجموعه‌ها، تکرار در دنباله مجاز و ترتیب در دنباله ها مهم است. بنابراین:

⟨شاپیرو، ویلسون، جونز⟩ ≠ ⟨ ویلسون، شاپیرو، جونز⟩

دنباله تهی به صورت < > نشان داده می‌شود.

از تعدادی اپرانورهای دنباله در مشخصه های رسمی استفاده می‌شوند علامت ^ . یک اپرانور دودویی است که دنباله را تشکیل می‌دهد این امر با افزودن دومین عملوند آن به انتهای اولین عملوند ساخته می‌شود. مثلاً:

⟨ 2 , 3 , 34 , 1 ⟩ ^ ⟨ 12 , 33 , 34 , 200 ⟩

منجر به دنباله زیر می‌شود:

⟨ 2 , 3 , 34 , 1 , 12 , 33 , 34 , 200 ⟩

سایر اپرانورهایی را که می‌توان در دنباله ها به‌کار گرفت عبارتند از: **Last , Front , Tail , Head**

عملگر **head** اولین عنصر یک دنباله را بیرون می‌کشد.

tail ، **n − 1** عنصر از انتهای دنباله را باز می گرداند.

last . آخرین عنصر نهایی دنباله را بیرون می‌کشد.

front ، **n − 1** عنصر نخست دنباله ای به طول **n** را بر می‌گرداند. مثلاً:

head ⟨2, 3, 34, 1, 99, 101⟩ = 2

---

1 domain

tail ⟨2, 3, 34, 1, 99, 101⟩ = ⟨3, 34, 1, 99, 101⟩
last ⟨2, 3, 34, 1, 99, 101⟩ = 101
front ⟨2, 3, 34, 1, 99, 101⟩ = ⟨2, 3, 34, 1, 99⟩

از آن‌جا که دنباله مجموعه‌ای از جفت‌هاست، همه اپراتورهای مجموعه که در بخش ۲۵-۲-۲ توصیف شده‌اند قابل استفاده هستند. وقتی از دنباله در یک وضعیت استفاده می‌شود، باید آن را اختصاصی کرد به‌طوری که با استفاده از کلمه رمز **seq** این کار صورت گیرد. مثلاً :

File List : Seq FILS
No Users : N

حالتی را با دو جزء توصیف می‌کند: یک دنباله از فایل‌ها و یک عدد طبیعی .

۲۵-۳ علائم ریاضی کاربردی برای مشخصه های رسمی

به‌منظور تشریح استفاده از عبارت ریاضی در مشخصات رسمی جزء نرم‌افزاری، دوباره مثال **Block Hander** را در بخش ۲۵-۱-۳ بررسی می‌کنیم. به‌منظور باز بینی، جزء مهمی از سیستم عامل کامپیوتر فایل‌هایی را نگه می‌دارد که توسط کاربر ایجاد شده‌اند. **Block Hander** مخزنی از بلوک‌های استفاده نشده را ایجاد کرده و بلوک‌هایی را که درحال حاضر مورد استفاده قرار گرفته اند، پی‌گیری می‌کند. وقتی بلوک‌ها از فایل حذف شده آزاد می‌شوند معمولاً به صفی از بلوک‌هایی می‌پیوندند که منتظر اضافه شدن به مخزن بلوک‌های بدون استفاده هستند. این کار را به‌صورت طرحی در شکل۲۵-۲ مورد شناسایی قرار داده‌ایم.[۲]

یک مجموعه بلوک‌هایی به‌نام **Block** فرض شده‌اند که شامل مجموعه‌ای از هر تعداد بلوک هستند و مجموعه **AlBlock** که مجموعه‌ای از بلوک‌هایی است که بین **1** و **Max Block** قرار دارند. این وضعیت توسط دو مجموعه و یک زنجیره مدل‌سازی می‌شوند. دو مجموعه عبارتند از **Used** (استفاده شده) و **Free** (آزاد) . هر دو حاوی بلوک هستند ـ مجموعه **Used** شامل بلوک‌هایی است که در حال حضر در فایل‌ها استفاده می‌شوند و **Free** حاوی بلوک‌هایی است که برای فایل‌های جدید مهیا هستند. این زنجیره حاوی مجموعه‌ای از بلوک‌هاست که آماده آزاد شدن از فایل‌های حذف شده هستند. این وضعیت را می‌توان به‌صورت زیر توصیف کرد:

used , free : P BLOCKS
BlockQueue : seq P Blocks

پرسش

چگونه وضعیت ها و تابعهای داده ای را هنگام استفاده از مجموعه و عملیات منطقی تولید شده، می توان مورد استفاده قرار داد؟

1.range

۲. اگر مثال راه انداز (رسیدگی کننده) به بلاک مبهم می نماید، لطفاً" به قسمت ۲۵-۱-۳ رجوع نموده، تابعهای داده ای، عملگرها، پیش شرط ها و پس شرط های مربوط به راه انداز بلاک را مرور کنید.

این مسئله بسیار شبیه اعلام متغیرهای برنامه است. این امر بیان می‌دارد که **Free . Used** مجموعه‌هایی از بلوک‌ها بوده و **Block Queue** یک زنجیره خواهد بود که هر عنصری از آن مجموعه‌ای از بلوک‌هاست. اطلاعات تغییر ناپذیر (داده های نامتغیر) را می‌توان به‌صورت زیر نوشت:

$$\text{used} \cap \text{Free} = \emptyset \ \wedge$$
$$\text{used} \cup \text{Free} = \text{All Blocks} \ \wedge$$
$$\forall i : \text{dom Block Queue} \cdot \text{Block Queue } i \subseteq \text{used} \ \wedge$$
$$\forall i, j : \text{dom Block Queue} \cdot i \neq j \ \text{Block Queue} \Rightarrow \text{Block Queue } j = \emptyset$$

جزءهای ریاضی اطلاعات تغییرناپذیر با چهار جزء زبان طبیعی که بیش تر توصیف شدند، تطابق دارند. اولین خط این اطلاعات بیان‌کننده این است که هیچ بلوک مشترکی در مجموعه بلوک‌های استفاده شده و مجموعه بلوک‌های آزاد وجود ندارد. دومین خط بیان می‌کند که مجموعه بلوک‌های استفاده شده و آزاد همیشه برابر مجموع بلوک‌های سیستم است. سومین خط نشانگر این است که عنصر **i** ام صف بلوک‌ها همیشه زیر مجموعه‌ای از بلوک‌های استفاده شده است. آخرین خط می‌گوید که برای هر دو عنصر صف بلوک که یکسان نیستند، بلوک مشترکی در این دو عنصر وجود ندارد.دو جزء نهایی زبان طبیعی در اطلاعات تغییرناپذیر بخاطر این حقیقت اجرا می‌شوند که آزاد و استفاده شده مجموعه هستند و بنابراین شامل نسخه‌سازی‌ها نمی‌شوند.

اولین عملیاتی که باید تعریف کنیم عملیاتی است که عنصری را از نوک صف حذف می‌کند. پیش‌شرط این است که باید حداقل یک قلم در صف باشد:

$$\#\text{BlockQueue} > 0$$

پس‌شرط این است که ابتدای صف باید برداشته شده و در مجموعه بلوک‌های آزاد گذاشته شود و صف طوری تنظیم شود که این جابجایی را نشان دهد:

$$\text{used}' = \text{used} \setminus \text{head Block Queue} \ \wedge$$
$$\text{free}' = \text{free} \cup \text{head Block Queue} \ \wedge$$
$$\text{Block Queue}' = \text{tail Block Queue}$$

یک مورد قراردادی که بسیار از روش‌های رسمی دیده می‌شود عبارتست از مقدارگیری یک متغیر بعد از این‌که عملیاتی آماده شروع شد. بنابراین، اولین جزء عبارت فوق بیان می‌دارد که بلوکهای استفاده شده جدید مساوی بلوک‌های استفاده شده قدیمی است منهای بلوک‌هایی که پاک شده‌اند. دومین جزء بیان می‌دارد که بلوک‌های آزاد جدید همان بلوک‌های آزاد قدیمی هستند که ابتدای صف بلوک به آن افزوده شده است. جزء سوم بیان می‌کند که صف بلوک جدید مساوی انتهای ارزش قدیمی صف بلوک است یعنی همه عناصر صف از اولی جدا هستند. عملیات دوم آن است که مجموعه‌ای از بلوک‌های **Ablocks** را به

اطلاعات جامعی در خصوص شیوه های رسمی در آدرس زیر وجود دارند :
www.archive.com/ab.ox.ac.uk/formal_methods.html

چگونه پیش شرط ها و پس شرط ها را بازنمایی نماییم؟

صف بلوک می‌افزاید. پیش‌شرط این است که **Ablocks** درحال حاضر مجموعه‌ای از بلوک‌های استفاده شده است:

$$\mathbf{Ablock} \subseteq \mathbf{used}$$

پس‌شرط این است که مجموعه‌ای از بلوک‌ها به انتهای صف بلوک افزوده شده و مجموعه بلوک‌های استفاده شده و استفاده نشده بدون تغییر باقی می‌ماند:

$$\mathbf{Block\ Queue'} = \mathbf{Block\ Queue} \frown \langle \mathbf{Ablocks} \rangle \wedge$$
$$\mathbf{used'} = \mathbf{used} \wedge$$
$$\mathbf{free'} = \mathbf{free}$$

شکی نیست که مشخصات ریاضی صف بلوک تا حد قابل توجهی شدیدتر و سخت از روال زبان طبیعی یا مدل گرافیکی است. این تلاش اضافه، نیازمند کار است اما مزایای بدست آمده از این انسجام و تکامل بهبود یافته را می‌توان در موارد کاربرد مختلف، توجیه نمود.

۴-۲۵ زبان مشخصه های رسمی

زبان مشخصات رسمی معمولاً متشکل از سه جزء اولیه است یعنی : ۱) نحوی (syntax) که مورد خاصی را تعریف می‌کند که با آن مشخصات نمایش داده می‌شوند. ۲) معنایی‌که به تعریف در عالم اشیا که برای توصیف سیستم به‌کار می روند کمک می‌کند. ۳) مجموعه ارتباطاتی که قواعدی را تعریف می‌کنند که نشانگر این است که چه اشیایی به‌درستی این مشخصه‌ها را برقرار می‌سازند [WIN90][1]

دامنه نحوی این زبان اغلب براساس بافت نحوی است که از تئوری استاندارد مجموعه مشتق شده و محاسباتی را پیش‌بینی می‌نماید. مثلاً متغیرهایی مثل **x , y , z** مجموعه‌ای از اشیا را توصیف می‌کنند که به مسئله‌ای مربوط می‌شوند (گاهی به‌نام دامنه مباحثه[2]) و در ارتباط با عملیات توصیفی در بخش ۲۵-۲ به‌کار می‌روند. گرچه معمولاً از آیکون‌های (شمایل) سمبولیک می‌توان استفاده نمود (مثل نمادهای گرافیکی مثل کادر ها، جعبه ها (باکس‌ها)، فلش‌ها و دوایر) که در این صورتی است که آنها غیر مبهم باشد.

دامنه معنایی[3] این زبان نشانگر چگونگی بیان نیازمندیهای سیستم به‌وسیله این زبان است. مثلاً زبان برنامه‌نویسی دارای مجموعه‌ای قوانین معنای رسمی است که تولیدکننده نرم‌افزار را قادر می‌سازد الگوریتم‌هایی را مشخص سازد که ورودی را به خروجی تبدیل کنند. می‌توان از دستور زبان رسمی(مثل BNF) برای توصیف بافت نحوی زبان برنامه‌نویسی استفاده کرد. با این وجود زبان برنامه‌نویسی زبان تعیین مشخصه های خوبی نخواهد بود، زیرا تنها می‌تواند نمایانگر عملیات محاسباتی باشد. یک زبان

1. Wing,J.M.
2. discourse
3. semantic domain

مشخصاتی باید دارای دامنه معنایی باشد که گسترده‌تر است. یعنی دامنه معنایی یک زبان باید بتواند ایده‌هایی را مثل «برای تمام **x**های یک مجموعه نامحدود، یک **y** در مجموعه نامحدود **B** وجود دارد به‌طوری‌که خواص **P** در مورد **x** و **y** نیز وجود دارد.»، توصیف کند. [WIN90] سایر زبان‌های مشخصاتی، عنایی به‌کار می‌برند که باعث مشخصات رفتاری سیستم می‌شود. مثلاً نحو و معنا را می‌توان برای مشخص کردن حالات و انتقال وضعیت، حوادث و تأثیرشان روی انتقال وضعیت، هم‌زمانی و زمان‌بندی استفاده نمود.

ممکن است از تجریدهای مختلف معنایی برای توصیف همین سیستم به طرق مختلف استفاده کرد. ما این کار را به شکل کمتر رسمی در فصول ۱۲ و ۲۱ انجام دادیم. جریان داده‌ها و پردازش مربوطه با استفاده از نمودار جریان داده‌ها و رفتار سیستم با نمودار انتقال وضعیت مشخص شد. موارد مختلف مدل‌سازی را می‌توان برای توصیف سیستم‌های شیءگرا مورد استفاده قرار داد. معنای هر نمایش دیدگاه‌های تکمیلی را در مورد سیستم مهیا می‌کند. به‌منظور تشریح این روش وقتی از شیوه های رسمی استفاده می‌شود، فرض کنید که یک زبان مشخصاتی برای توصیف مجموعه حوادثی به‌کار می‌رود که باعث وضعیت خاصی در سیستم می‌شود. رابطه رسمی دیگری تمام عملکردهایی را که در وضعیت فرضی رخ می‌دهد، مشخص می‌کند. محل تقاطع این دو رابطه نشانگر حوادثی است که باعث رخ دادن عملکردهای به‌خصوصی می‌شوند.

یک‌سری زبان‌ها امروزه مورد استفاده قرار می‌گیرند. **CSP** [HIN95][1] و **LARCH** [HOR85][2] و **VDM** [JON91][3] و **Z** [SPI92][4] نماینده زبان‌های مشخصاتی رسمی هستند که مشخصه‌های فوق‌الذکر را نشان می‌دهند. در این فصل زبان **Z** برای اهداف تشریحی به‌کار خواهد رفت. **Z** با ابزار خودکاری همراه می‌شود که موارد حقیقی و به‌اثبات رسیده، قوانین استنتاج و برنامه‌های کاربردی نشأت گرفته از فضایایی که منجربه اثبات ریاضی درستی مشخصات می‌شود را، ذخیره می‌کند.

۲۵-۵ استفاده از **Z** برای بازنمایی مثالی از اجزاء نرم افزار

مشخصات **Z** به‌صورت مجموعه‌ای از طرح‌ها، ساختار جعبه مانندی که متغیرها را معرفی نموده و ارتباط بین این متغیرها را معرفی می‌کند، ساختاربندی می‌شوند. هر طرح اساساً آنالوگ مشخصات رسمی زیرروال یا رویه زبان برنامه‌نویسی می باشد. از جهاتی یکسان که رویه‌ها و زیر روال‌ها در ساختار سیستم استفاده می‌شوند، طرح‌ها برای ساخت مشخصات رسمی به‌کار می‌روند.

1.Hinchley,M.G.
2.Hoare,C.A.R.
3.Jones,C.B.
4.Spivey,J.M.

Z notations is based on typed set throry and first-order logic. Z provides a construct, called s schema, to describe a specification's state space and operations. A schema groups variable declarations with a list of predicates that constrain the possible value of variable. In Z, the schema X is defined by the form

X

declarations

predicates

Global functions and constants are defined by the form

declarations

predicates

The declaration gives the type of the function or constant, while the predicate gives it value. Only an abbreviated set of Z symbols is presented in this table.

Sets :

$S : \mathbb{P} X$	S is declared as a set of Xs.
$x \in S$	x is a member of S.
$x \notin S$	x is not member of S.
$S \subseteq T$	S is a subset of T: Every member of is also in T.
$S \cup T$	The union of S and T: It contains every member of S or T or both.
$S \cap T$	The intersection of S and T: It contains every member of both S and T.
$S \setminus T$	The difference of S and T: It contains every member of S except those also in T.
$\emptyset$	Empty set: It contains no members.
$\{x\}$	Single set: It contains just x.
$\mathbb{N}$	The set of natural numbers 0,1,2,…
$S : F X$	S is declared as a finite set of Xs.
max (S)	The maximum of the nonempty se of numbers S.

Functions:

$F : X \rightarrowtail Y$	f is declared as a partial injection from X to Y
dom f	The domain of f: the set of values x for which $f(x)$ is defined.
ran f	The range of f: the set of values taken by $f(x)$ as x varies over the domain of f.
$f \oplus \{x \mapsto y\}$	A function that agree with f except that x is mapped to y.
$\{x\} \trianglelefteq f$	A function like f, except that x is removed from its domain.

Logic:

$P \wedge Q$	P and Q : It is true if both P and Q are true.
$P \Rightarrow Q$	P implies Q: It is true if either Q is true or P is false.
$\theta \dot{S} = \theta S$	No components of schema S change in an operation.

جدول ۲۵-۱ خلاصه علائم (نشانه گذاری) زبان Z

در این بخش، ما از زبان مشخصاتی **Z** برای مدل‌سازی بلوک ارائه شده به‌عنوان مثال که در بخش ۲۵-۱-۳ آمده، استفاده کرده و در بخش ۲۵-۳ آن را بیشتر مورد بحث قرار می دهیم. خلاصه‌ای از عبارت زبانی **Z** در جدول ۲۵-۱ آمده است. مثال زیر از یک طرحی است که وضعیت بلوک و اطلاعات تغییرناپذیر را نشان می‌دهد:

BlockHandler

$$used, free : \mathbb{P}\ BLOCKS$$
$$BlockQueue : seq\ \mathbb{P}\ BLOCKS$$

$$used \cap Free = \emptyset \ \wedge$$
$$used \cup Free = All\ Blocks \ \wedge$$
$$\forall i : dom\ Block\ Queue \cdot Block\ Queue\ i \subseteq used \ \wedge$$
$$\forall i, j : dom\ Block\ Queue \cdot i \neq j \Rightarrow$$
$$Block\ Queue\ i \cap Block\ Queue\ j = \emptyset$$

این طرح شامل دو بخش است: بخش بالای خط مرکزی نمایشگر متغیرهای وضعیت است در حالی‌که بخش پایینی آن اطلاعات تغییرناپذیر را توصیف می‌کند. هرگاه طرح نمایانگر تغییرناپذیر بودن و وضعیت در طرح دیگری استفاده شود، به آن علامت Δ اضافه می‌شود. بنابراین، اگر طرح فوق در طرحی به‌کار رود که مثلاً عملیاتی را توصیف کند، پس باید آن را به‌صورت Δ **Block hander** نوشت. همان‌گونه که جمله آخر نشان می‌دهد این طرح‌ها را می‌توان برای توصیف عملیات استفاده نمود. مثال زیر در مورد یک طرح، عملیاتی را توصیف می‌کند که عنصری را از صف بلوک جابه‌جا می‌کند.

RemoveBlock

$$\Delta\ BlockHandler$$

$$Block\ Queue > 0,$$
$$Used' = used \setminus head\ Block\ Queue \ \wedge$$
$$free' = free \cup head\ Block\ Queue \ \wedge$$
$$Block\ Queue' = tail\ Block\ Queue$$

گنجاندن **Δ Block hander** سجر به تمام متغیرهایی می‌شود که باعث در دسترس بودن وضعیت در مورد طرح **Remove Block** بوده و این اطمینان را حاصل می‌کند که اطلاعات تغییرناپذیر است قبل و بعد از اجرای عملیات حفظ می‌شوند.

عملیات دوم که مجموعه ای را به انتهای صف می افزاید، بصورت زیر نمایش داده می‌شود :

AddBlock

Δ BlockHandler
Ablocks? : BLOCKS

Ablocks? used
Block Queue′ = Block Queue ^ ⟨Ablocks?⟩ ∧
used′ = used ∧
free′ = free

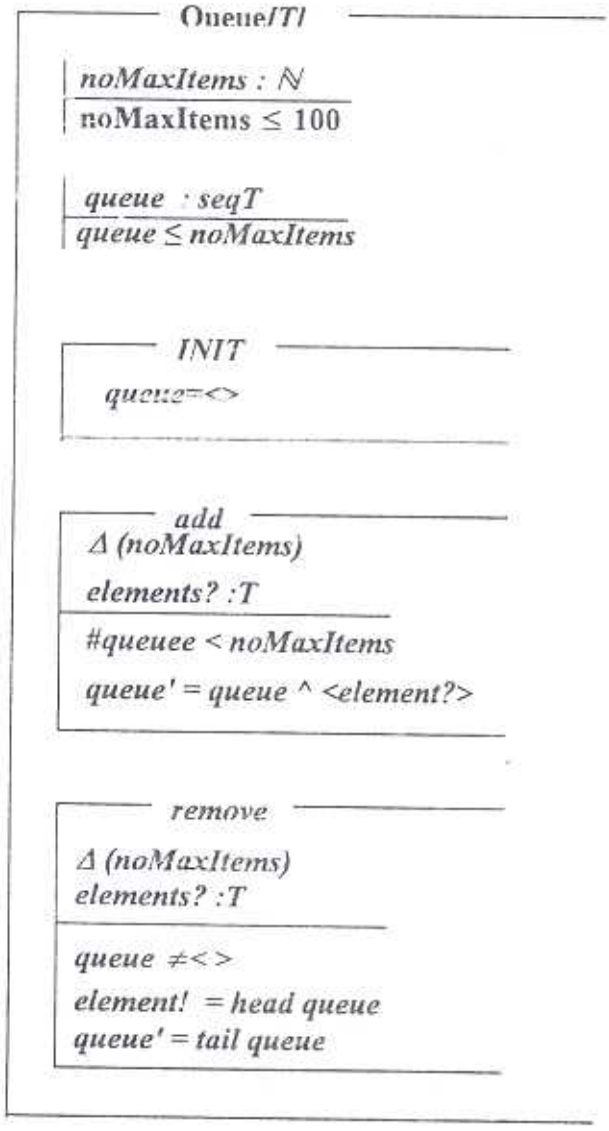
به‌صورت قراردادی در **Z** متغیر ورودی که از روی آن خوانده شده و بخشی از وضعیت را تشکیل نمی‌دهد به‌وسیله یک علامت سؤال خاتمه می‌یابد. بنابراین، **Ablocks ?** که به‌عنوان پارامتر ورودی عمل می‌کند به‌وسیله یک علامت سؤال خاتمه می‌یابد.

۶-۲۵ شیوه های رسمی شئ گرا

افزایش علاقه به فنآوری شیءگرایی به این معنی است که کارگران در محیط روش‌های رسمی کار تعریف عبارت ریاضی را شروع کرده‌اند که منعکس‌کننده ساختارهای مربوط به جهت‌گیری شیء است یعنی موارد مربوط به کلاس، وراثت و وقوع. تعدادی از متغیرهای عبارات موجود پیشنهاد شده بودند که عمدتاً براساس **Z** بودند و هدف از این بخش بررسی یکی از آنهاست یعنی، شیء **Z** که توسط محققان در مرکز شناسایی نرم‌افزار در دانشگاه کوئینزلند تولید شده است.

شیء **Z** از نظر جزئیات بسیار شبیه **Z** است. نقطه تفاوت آن در رابطه با ساختاربندی طرح‌ها و گنجاندن وراثت و تسهیلات عمل است.

نمونه‌ای از مشخصات شیء **Z** در زیر نشان داده شده است:



این تصویر نمایانگر مشخصات کلاس‌ای است که یک صف کلی را توصیف می کند که می‌تواند دربرگیرنده اشیایی از هر نوع باشد.

همچنین کلاس‌ای را تعریف می‌کند که دارای متغیر لحظه‌ای یا موردی **queue** است یعنی زنجیره‌ای که دارنده اشیای نوع **T** می‌تواند از هر نوعی باشد مثلاً می‌تواند یک انتگرال، یک

اتوبوس، مجموعه‌ای از قلمهای محاسباتی یا بلوکهای حافظه باشد. در پی تعریف صف، چندین طرح وجود دارد که کلاس را تعریف می‌کنند. اولی مقدار ثابتی را مشخص می‌کند که دارای ارزشی بیش از

۱ بیشتر طرح بعدی حداکثر طول صف را مشخص نموده و دو طرح بعدی یعنی **Add** و **Remove** فرایندهای افزون و حذف قلمی را از صف تعریف می‌کنند.

پیش‌شرط **Add** مشخص می‌کند که وقتی قلمی مثل **element ?** به صف افزوده می‌شود نباید از حداکثر اندازه مجاز بلندتر باشد، پس‌شرط اتفاقاتی را مشخص می‌کند که وقتی اضافه شدن عنصری

تکمیل شد، روی می‌دهند. پیش‌شرط عملیات **Remove** مشخص می‌کند که برای این‌که این عملیات موفقیت‌آمیز باشد نباید صف خالی باشد، پس‌شرط حذف نوک صف و جابه‌جایی آن را در عنصر متغیر مشخص می‌سازد.

پس این مشخصات مقدماتی کلاس شیء در شئ **Z** است. اگر می‌خواستیم از شیء تعریف شده‌ای توسط چنین کلاسی استفاده کنیم، نسبتاً ساده ای خواهد بود. مثلاً فرض کنید که ما بخشی از سیستم عاملی را تعریف می‌کنیم که فرایندهایی را ارائه می‌دهد. فرایندهایی که نمایانگر اجرای برنامه‌ها و همچنین نشان‌دهنده این است که ما تصمیم گرفته‌ایم بردارش‌ها در دو صف نگهداری شوند یکی شامل بردارش‌هایی که دارای اولویت زیاد هستند و دیگری بردازش‌هایی که اولویت پایین دارند و این اولویت‌بندی در اختیار سیستم عامل است تا برنامه‌ریزی کند چه بردازش در وهله بعدی انجام شود. اولین وضعیت شئ **Z** را که لازم داریم کلاس صف کلی ما را در یک کلاس که حاوی پردازش‌ها ست، قرار می‌دهد.

$$ProcQueue == Queue[PROCESSES]$$
$$[procQ / queue, proc! / element? Proc! / element!]$$

تمام چیزی که انجام می‌دهد این است که نوع «**T**» را به‌وسیله پردازش‌هایی در تعریف کلاس، صف کلی **queue** را به‌وسیله صف بردازش‌های **ProQ** و پارامترهای ورودی و خروجی کلی **element ?** و **elment !** را به‌وسیله پارامترهای بردارشی **Proc ?** و **Proc !** ، جایگزین می‌کند. علامت / را به‌عنوان فرمی از جایگزینی متنی بخوانید.

مرحله بعدی تعریف کلاس‌ای است که دو صف را توصیف می‌کند. این مورد در زیر نشان داده شده است. این کلاس از عملیات تعریف شده در کلاس **Queue** استفاده می‌کند. فرض می‌کنیم که عملیات زیر لازمند:

- ***addHigh*** که فرآیندی را به صف پردازش‌های دارای اولویت بالا می‌افزاید.
- ***addlow*** که فرآیندی را به صف پردازش‌های دارای اولویت کم می‌افزاید.
- ***INIT*** که صف‌های دارای اولویت کم و زیاد را راه‌اندازی می‌کنند.
- ***totalProcesses*** که با تعدادی کل فرآیندهای صف شده در هر دو صف بازمی گردد.

Queuepair

$high, low : ProcQueue$

$high \neq low$

INIT

$high.INIT \wedge low.INIT$

$addHigh \triangleq high.Add$

$add.Low \triangleq low.Add$

totalProcess

$total! : \mathbb{N}$

$total! = \#high + \#low$

اولین خط، طرح کلی را تعریف می‌کند. خط‌های دوم و سوم وضعیت و تغییرناپذیری آن را مشخص می‌سازند. در این مورد، وضعیت شامل دو صف **high** و **low** است که صف‌های پردازش هستند که متمایزند.

تعریف وضعیت، همراه با تعریف چهار عملیات است. **INIT** به‌عنوان برنامه کاربردی عملیات موروثی **INIT** روی صف‌های **high** و **low** تعریف می‌شود. **add high** برنامه کاربردی عملیات موروثی **add** روی جزء **high** می‌باشد. **add low** برنامه عملیات موروثی **add** روی جزء **low** بوده و در آخر، عملیات **total Processes** مجموع اندازه‌های صف‌های تکی **high** و **low** می‌باشد.

این مثال از ترسیم مادی یک مورد انتزاعی است که در آن اشیا به‌وسیله طرح کلی **object Z** که در طرح دیگری استفاده شده، تعریف می‌شوند. این کار ما را مجبور می‌سازد که وراثت را تعریف کنیم.

برای انجام این کار اجازه دهید مثال دیگری را بررسی کنیم، که از یک جدول نشانه است.چنین جداولی مرتباً در برنامه‌های کاربردی استفاده می‌شوند.

از آنها برای قراردادی اسمی سیستم‌های پرسلی، اسامی چاپگرها در سیستم‌های عامل، اسامی جاده‌ها در سیستم حمل ونقل وسایط نقلیه و غیره استفاده می‌شود. به‌منظور توصیف وراثت ابتدا جدول کلی نماد را تعریف کرده، از آن استفاده نموده و سپس نشان می‌دهیم که چگونه می‌توانیم از طرح کلی **Z**

DisTable [T]

SymbolTable[T]
freqItem: T

find

Δ(freqItem)
toBeFound? : T
status: {found,notfound}

$toBefound? = freqItem \vee toBeFound? \in table \Rightarrow$

$status! = found \wedge freqItem' = toBeFound?$

$toBefound? \neq freqItem \wedge toBeFound? \notin table \Rightarrow$

$status! = notFound$

در این‌جا تمام دیگر عملیاتی که برای **Symbol Table** به ارث رسیده بدون تغییر بوده همان‌گونه که متغیر موردی **table** است. عملیات جدید **Find** از متغیر **Freq Item** استفاده می‌کند که بحثی از طرح کلی **object Z** جدید را در **Dist Table** نشان می‌دهد.

این عملیات چک می‌کند که آیا پارامتر ورودی **of find** همان قلم همیشگی است که چندین دفعه بازیافت شده یا در جدول وجود دارد. اگر این‌گونه است، پس از وضعیت درست دریافت شده بازگشت داده شده و قلم معمول به روز می‌شود. اگر این قلم یک قلم معمول نبوده و در جدول نباشد، وضعیت **not found** برگشت داده می‌شود.

از این طرح می‌توان در یک برنامه کاربردی کارمندان با مشخص کردن موارد زیر استفاده نمود :

FreqEmployeeTab = = DisTable[EMPLOYEE]
[emps/table,emp?/item?,emp!/item!,freqEmp/freqItem]

FreqEmptable

e: FreqEmployeeTab

$FreqEmpAdd \triangleq e.ADD$

$FreqEmpRemove \triangleq e.REMOVE$

$FreqEmpInit \triangleq e.INIT$

$FreqEmpNumber \triangleq e.NUMBER$

این معرفی کوتاهی بود در مورد این‌که چگونه عبارات رسمی شروع به استفاده شدن برای مشخص کردن سیستم‌های شیءگرا می‌کنند. اکثر کاری که در این حوزه انجام گرفته از عبارت **Z** استفاده برده و تلاش کرده تا ابده وضعیت، پیش‌شرط، پس‌شرط و اطلاعات تغییرناپذیر توصیف شده در بخش قبلی را به‌منظور اجرای تسهیلاتی برای تجسم و وراثت، ایجاد نماید. کار در این زمینه هنوز با چند برنامه کاربردی در سطح تحقیقاتی است. با این وجود، در طول مدت این چاپ، نشانه گذاری‌های رسمی شیءگرا سطح یکسانی از نفوذ صنعتی را به‌عنوان نشانه گذاری های استاندارد مثل **Z**، تجربه می‌کنند.

۲۵-۷ مشخصات جبری

یکی از مشخصه‌های اصلی دو تکنیک **Z** و **Z++** که قبلاً توصیف شدند این حقیقت است که آنها براساس مورد یک وضعیت هستند یعنی مجموعه‌ای از داده‌هایی که تحت‌تأثیر عملیاتی با عملیاتی که توسط عبارات مکتوب در محاسبات پیش‌بینی شده تعریف شده‌اند، می‌باشند.

تکنیک جایگزین دیگری به‌نام مشخصات جبری وجود دارد. این تکنیک مستلزم نوشتن جملات ریاضی است که به اثر عملیات مجاز در مورد بعضی داده‌ها مرتبط می‌شود. مزیت اصلی این تکنیک پشتیبانی بسیار ساده‌تر منطق نسبت به روش‌های مبتنی بر وضعیت است.

اولین مرحله در نوشتن مشخصات جبری عبارتست از تعیین این‌که چه عملیاتی لازمند. مثلاً، یک سیستم فرعی که صفی از پیام‌ها را در سیستم ارتباطات اجرا می‌کند ممکن است به عملیاتی نیاز داشته باشد که قلمی را از ابتدای صف حذف کرده، با چند قلم در صف بازگشته و چک می‌کند آیا صف خالی است یا خیر.

وقتی این عملیات اجرا شد، رابطه بین آنها را می‌توان مشخص کرد. مثلاً، مشخص‌کننده می‌تواند این حقیقت را توصیف کند که وقتی یک قلم به صف افزوده شد، تعداد قلم‌های آن صف یکی افزایش می‌یابد. به‌منظور درک شباهت مشخصات جبری، دو مشخصه را بازآفرینی می‌کنیم. اولی در مورد یک صف و دومی در مورد یک جدول نشانه است. در مورد صف فرض می‌کنیم که عملیات زیر لازمند:

- ***emptyQueue*** ـ این گزینه ارزش بولین را در صورتی حقیقی می‌گرداند که صف مربوط به آن خالی باشد وگرنه غلط می‌شود.
- ***addItem*** ـ قلمی را به انتهای صف اضافه می‌کند.
- ***removeItem*** ـ قلمی را از انتهای صف حذف می‌کند.
- ***first*** ـ این مورد به اولین قلم صف برمی‌گردد، صف تحت‌تأثیر این عملیات نیست.
- ***last*** ـ به آخرین مورد صف برمی‌گردد، صف تحت‌تأثیر این عملیات نیست.
- ***isEmpty?*** ـ به مقدار بولین درست برمی‌گردد، اگر که صف خالی باشد در غیر این‌صورت اشتباه است.

اولین خطوط مشخصات این صف و عملیات آن در زیر آمده است:

Name: partialQueue
Sorts: queue(Z)
Operators:

emptyQueue:	**→queue(Z)**
addItem:	**Z x queue(Z) → queue(Z)**
removeItem:	**Z x queue(Z) → queue(Z)**
first:	**queue(Z) → Z**
last:	**queue(Z) → Z**
isEmpty:	**queue(Z) → Boolaen**

اولین خط نامی به نوع صف می‌دهد. دومین خط بیان می‌دارد که صف از عهده هر نوع مدخل **Z** برمی‌آید. مثلاً می‌تواند پیام، فرآیند، کارکنان منتظر ورود به ساختمان یا هواپیماهای منتظر ورود در یک سیستم کنترل ترافیک هوایی باشد.

باقی مشخصات علایم عملیات را توصیف می‌کند:

نام و نوع قلمی که پردازش می‌کنند. تعریف ***emptyQueue*** بیان می‌دارد که بخشی وجود ندارد و صفی ارائه می‌دهد که خالی است، تعریف ***addItem*** می‌گوید که شئی از نوع **Z** را دارد و یک صف حاوی اشیائی از نوع **Z** و سپس صفی از اشیا نوع **Z** و غیره ارائه می‌دهد.

این تنها نیمی از مشخصه است، باقی آن معنای هر عملیات را از نظر ارتباط آن با دیگر عملیات توصیف می‌کند. بعضی از نمونه‌های این نوع مشخصه‌ها در زیر نشان داده شده است.

اولین مورد بیان می‌کند که ***isEmpty?*** درصورتی درست است که صف خالی و در غیر این‌صورت حتی اگر یک قلم در صف باشد غلط است. هر یک از این خواص به‌صورت یک تک خط بیان شده است:

isEmpty? (emptyQueue) = true
isEmpty? (addItem (z , q)) = false

تعریف اولین عملیات عبارتست از:

first (addItem (z , q)) = z

که بیان می‌دارد **first** به اولین قلم صف برمی‌گردد.

این کار کیفیت و مشخصات سبک مشخصات را به ما می‌دهد. برای نتیجه‌گیری مشخصات کلی را برای یک جدول نشانه ارائه می‌دهیم این ساختار اطلاعاتی است که حاوی مجموعه‌ای از اشیا بدون هیچ‌گونه نسخه‌ای است. چنین جداولی تقریباً در هر سیستم کامپیوتری یافت می‌شوند:

در اسمبلرها، در جدول کلیدواژه‌ها در سیستم‌های پرسشی، سامانه‌های هواپیما در سیستم کنترل ترافیک هوایی، برنامه‌هایی در سیستم‌های عامل و غیره استفاده می‌شود.

- ***emptyTable*** ـ یک جدول نشانه خالی می‌سازد

- ***addItem*** ـ نشانه‌ای را به جدول موجود می‌افزاید.
- ***removeItem*** ـ نشانه‌ای را از جدول موجود حذف می‌کند
- ***isInTable?*** ـ درصورت وجود نشانه‌ای در جدول علامت درست و در غیر این‌صورت علامت غلط می‌دهد.
- ***Join*** ـ محتویات در جدول نشانه‌ها را به‌هم مرتبط می‌کند.
- ***common*** ـ دو نشانه از جدول گرفته و عناصر مشترک از هر جدول را برمی‌گرداند.
- ***isPartOf?*** ـ درصورتی که یک جدول نشانه در یک جدول دیگر باشد، علامت درست نشان می‌دهد.
- ***isEqual?*** ـ اگر جدول نشانه‌ای با جدول دیگری مساوی باشد، علامت درست را نشان می‌دهد.
- ***isEmpty?*** ـ اگر جدول خالی باشد، درست است.

تعریف علایم اپراتورها در زیر نشان داده شده است:

Name table

Sorts: symbolTable(z) with =

Operators:

emptyTable:	**→symbolTable(Z)**
addItem:	**Z x symbolTable(Z) → symbolTable(Z)**
removeItem:	**Z x symbolTable(Z) → symbolTable(Z)**
isIntable?:	**Z x symbolTable(Z) → Boolean**
join:	**symbolTable(Z) x symbolTable(Z) → symbolTable(Z)**
common:	**symbolTable(Z) x symbolTable(Z) → symbolTable(Z)**
isPartOf?:	**symbolTable(Z) x symbolTable(Z) → Boolean**
isEqual?:	**symbolTable(Z) x symbolTable(Z) → Boolean**
isEmpty?:	**symbolTable(Z) → Boolean**

تمام تعریف خود توضیح می دهند که چه هستند به جز خط

Sorts : Symbol Table(Z) with =

که بیان می‌کند اشیای نوع **Z** با یک اپراتور برابر و مرتبطند.

تعریف اپراتور ***addItem*** عبارتست از:

addItem(s2,addItem(s1,s))= **if s1=s2 then addItem(s2,s)**

else addItem(s1,addItem(s2,s))

این بیان می‌دارد که اگر دو مورد اضافه شده در جدول نشانه وجود دارد و تلاش می‌شود همان قلم که دوباره قبلاً وجود داشته به آن اضافه شود، این کار برابر با اضافه کردن یکی از آن دو قلم خواهد بود. اگر قلمی متفاوت بود، تأثیر آن افزودن آن قلم به شیوه‌ای متفاوت به بقیه است.

تعریف ***removeItem*** عبارتست از:

removeItem(s1,emptyTable)= emptyTable
removeItem(s1,addItem(s2,s)) = if s1=s2 then removeItem(s1,s)
else addItem(s2,removeItem(s1,s))

این مورد بیان می‌کند که اگر بخواهیم قلمی را از جدول حالی حذف کنیم منجر به ساخت یک جدول حالی می‌شود و وقتی قلمی مثل **s1** را از یک جدول حاوی قلم **s2** حذف می‌کنیم درصورتی که **s1 , s2** مشابه باشند آن‌وقت تأثیر آن مثل حذف **s1** است. اگر مساوی و یکسان نباشند تأثیر آن باقی ماندن **s2** و حذف قلم **s1** است.

تعریف ***isInTable?*** عبارتست از:

isInTable?(s1,emptyTable)= false
isInTable?(s1,addItem(s2,s))= if s1=s2 then true
else isInTable?(s1,s)

که نشان می‌دهد که یک عنصر نمی‌تواند عضوی از یک عنصر خالی را نشانه باشد و نتیجه به‌کارگیری ***isInTable?*** و **s1** در یک جدول حاوی **? s1 , s2** عبارتست از علامت درست اگر **? s1** مساوی **s2** بوده و در غیر این‌صورت لازم است ***isInTable?*** در **s1** به‌کار گرفته شود.

تعریف ***Join*** در زیر آمده است:

join(s,emptyTable)= s
join(s,addItem(s1,t))= addItem(s1,join(s,t))

این مورد بیان می‌کند که اتصال یک جدول خالی و جدول **s** منجر به ایجاد یک جدول خالی **s** وی شود و اثر الحاق این دو جدول باهم، که یکی از آنها حاوی نماد **s1** است، برابر با افزودن **s1** به نتیجه الحاق دو جدول است.

تعریف ***common*** در زیر آمده است:

common(s,emptyTable)= emptyTable
common(s,addItem(s1,t))= if isInTable?(s1,s)
then addItem(s1,common(s,t))
else common(s,t)

موارد فوق نشانگر این است که جدول حاوی عناصر مشترکی از جدول حالی و هر جدول دیگری است که همیشه جدول خالی است. دومین خط بیانگر این است که اگر دو جدول را به هم مرتبط کنیم اگر عنصر مشترک **s1** وجود داشته باشد پس **s1** به مجموعه مشترک افزوده شده، در غیر این‌صورت مجموعه مشترک از دو مجموعه تشکیل شده است.

تعریف ***isPartOf?*** عبارتست از:

isPartOff?(emptyTable,s)= true
isPartOf?(addItem(s1,s),t)= if isInTable?(s1,s) then isPartOf?(s,t)
else false

که بیان می دارد که جدول خالی نشانه‌ها همیشه بخشی از هر جدول نشانه‌ای می‌باشد. دومین خط می‌گوید که جدول **t** حاوی عنصری در **s** باشد آن‌گاه ما ***isPartOf?*** را در آن به‌کار می‌گیریم تا ببینیم آیا **s** زیر جدولی از **t** است یا خیر.

تعریف ***isEqual?*** عبارتست از:

isPartOf?(s,t) ∧ isPartOf?(t,s)

که بیان می‌کند که دو جدول نشانه در صورتی مساوی‌اند که هر دو دیگری وجود داشته باشند.

آخرین تعریف که مربوط به ***isEmptyOf?*** است، بسیار ساده است.

isEmpty?(emptyTable) = true
isEmpty?(addItem (S1,t)) = false

این تنها بیان می‌دارد که جدول حالی، است و جدولی که حاوی حداقل یک قلم است خالی نخواهد بود. توصیف کلی جدول نشانه می‌شود:

Name table

Sorts: symbolTable(z) with =

Operators:

emplyTable:	**→symbolTable(Z)**
addItem:	**Z x symbolTable(Z) → symbolTable(Z)**
removeItem:	**Z x symbolTable(Z) → symbolTable(Z)**
isIntable?:	**Z x symbolTable(Z) → Boolean**
join:	**symbolTable(Z) x symbolTable(Z) → symbolTable(Z)**
common:	**symbolTable(Z) x symbolTable(Z) → symbolTable(Z)**
isPartOf?:	**symbolTable(Z) x symbolTable(Z) → Boolean**

```
isEqual?:                    symbolTable(Z) x symbolTable(Z) → Boolean
isEmpty?:                    symbolTable(Z) → Boolean
addItem(s2,addItem(s1,s))=   if s1=s2 then addItem(s2,s)
                             else addItem(s1,addItem(s2,s))
removeItem(s1,emptyTable)=   emptyTable
removeItem(s1,addItem(s2,s)) = if s1=s2 then removeItem(s1,s)
                             else addItem(s2,removeItem(s1,s))
isInTable?(s1,emptyTable)=   false
isInTable?(s1,addItem(s2,s))= if s1=s2 then true
                             else isInTable?(s1,s)
join(s,emptyTable)=          s
join(s,addItem(s1,t))=       addItem(s1,join(s,t))
common(s,emptyTable)=        emptyTable
common(s,addItem(s1,t))=     if isInTable?(s1,s)
                               then addItem(s1,common(s,t))
                               else common(s,t)

isPartOf?(emptyTable,s)=     true
isPartOf?(addItem(s1,s),t)=  if isInTable?(s1,s) then isPartOf?(s,t)
                             else false
isEqual?(s,t)=               isPartOf?(s,t) ^ isPartOf?(t,s)
isEmpty?(emptyTable)=        true
isEmpty?(addItem(s1,s))=     false
```

ایجاد چنین مشخصاتی بسیار سخت است، مشکل اصلی این است که شما نمی‌دانید چه وقت افزودن عباراتی را که عملیات را به‌هم مرتبط می‌کند متوقف کنید و آنها مایلند بسیار طولانی‌تر از معادلات مبتنی بر وضعیتشان باشند. آنها از نظر ریاضی خالص‌ترند و برای انجام دلیل آوردن و استدلال مناسب‌تر می‌باشند

۸-۲۵ شیوه های رسمی همروند

در بخش قبل به توصیف Z و انحراف شیء برای آن شیء Z (object Z) اختصاص داشت. مشکل اصلی این موارد و دیگر [illegible]

[illegible] برای [illegible] صلاح مواردی مثل Z [illegible] دارای درجه بالایی [illegible] نشده تا هم‌زمانی را نیز در برگیرد. موفقیتی نیز کسب نشده است زیرا آنها واقعاً با این فکر طراحی شده اند

در بعضی موارد مودیسیسی در ... طور اهداف ویژه از نظر ... کسب شده و هدف از این بخش توصیف یکی از مشهورترین آنها یعنی CSP است.

CSP یا Communicating Sequential Processes (برقراری ارتباط میان فرآیندهای پردازشی تسلسلی) به‌وسیله تونی هور دانشمند علوم کامپیوتری در دانشگاه آکسفورد ارائه گردید. ایده اصلی در پس این عبارت این است که هر سیستمی می‌تواند به‌عنوان مجموعه‌ای از پردازش‌های تسلسلی (برنامه‌های ساده غیر هم‌زمان) نگریسته شود که به‌طور خود گران اجرا و با سایر فرایندها ارتباط برقرار می‌کند. علاوه بر این، هور یک‌سری قوانین جبری ارائه داد که امکان استدلال را فراهم می‌آورد. مثلاً قوانین او تولیدکننده را قادر می‌سازد که توضیح دهد فرآیند P_1 منتظر عمل دیگر P_2 نیست که آن هم به‌خودی خود منتظر P_1 است تا عمل دیگری را انجام دهد.

مورد CSP در مقایسه با مواردی هم‌چون Z یا object Z بسیار ساده است. این مورد بر مفهوم یک حادثه و واقعه تکیه دارد. حادثه چیزی است که می‌توان رؤیت نمود، ملموس و اتفاقی یعنی به‌طور مثال، آن را نمی‌توان دچار وقفه نمود و بدون توجه به آن‌چه که در سیستم رخ می‌دهد، به تکامل و پایان می‌رسد. مجموعه حوادث مربوط به فرآیندی مثل P با حرف لفاز P شناخته شده و به‌صورت ∂P نوشته می‌شود. نمونه‌های رایج آنها قطع و وصل والو (شیر اطمینان) در یک کنترل‌کننده صنعتی، به روزسازی یک متغیر جهانی در یک برنامه یا انتقال بعضی از داده‌ها به کامپیوتر دیگر است. پردازش‌ها بطور باز گشتی از نظر حوادثشان تعریف می‌شوند. مثلاً:

$$(e \rightarrow P)$$

این حقیقت را توصیف می‌کند که یک پردازش درگیر با حادثه‌ای مثل e در ∂P است و سپس مثل P عمل می‌کند. در CSP می‌توانیم تعاریف پردازش‌ها را قرار دهیم. مثلاً P فوق‌الذکر را می‌توان از نظر پردازش دیگر P_1 به‌صورت زیر تعریف کرد:

$$(e \rightarrow (e_0 \rightarrow P_1))$$

که در آن حادثه e_0 رخ داده و پردازش هم‌چون پردازش P_1 عمل می‌کند.

تسهیلاتی که توصیف کردیم چندان مفید نیستند زیرا شامل هر انتخابی نمی‌شوند، مثلاً یک پردازش می‌تواند درگیر دو حادثه شود. انتخاب اپراتور | به مشخصات CSP امکان می‌دهد شامل این عنصر تصمیم‌گیری شود. مثلاً مشخصات زیر پردازش P را تعریف می‌کند که امکان انتخاب را فراهم می‌آورد.

$$P = (e_1 \rightarrow P_1 \mid e_2 \rightarrow P_2)$$

در این‌جا پردازش P طوری تعریف شده که قادر به درگیری در دو حادثه است یعنی e1 , e2 اگر اولی رخ دهد پس فرآیند مثل p_1 عمل کرده و اگر دومی رخ دهد آن‌وقت مثل p_2 عمل می‌کند. چند پردازش استاندارد در CSP وجود دارد. Stop فرایندی است که نشانگر این است که سیستم به‌طور ... قبلی نامطلوب مانند dead lock (بن بست) خاتمه یافته است. ...

فرآیند دیگری است که با موفقیت کاری را پایان می‌بخشد. **RON** می‌تواند در هر رویدادی در الفبای آن درگیر شود. این مورد نیز هم‌چون **Stop** نامطلوب بوده و نشانگر این است که **live lock**رخ داده است.

به‌منظور بدست آوردن مشخصه‌های چگونگی استفاده **CSP** در مورد مشخصات پردازش‌ها، چند سیستم بسیار ساده را مشخص می‌کنیم. اولین آنها سیستمی است که شامل یک پردازش **C** است. وقتی این فرآیند فعال شد، شیر ایمنی (والو) را در راکتور شیمیایی بسته و سپس متوقف می‌شود.

$$\alpha C = \{close\}$$
$$C = (close \rightarrow Stop)$$

اولین خط، الفبای فرآیند را تعریف می‌کند که شامل یک حادثه بوده و دومی بیان می‌کند که پردازش **C** درگیر عمل بستن و سپس توقف است.

این مشخصه‌ای ساده و تا حدی غیر واقع بینانه است. اکنون فرض کنید فرآیند دیگری به نام C_1 را تعریف می‌کنیم که شیر را باز نموده، آن را می بندد و سپس دوباره مثل C_1 عمل می‌کند.

$$\alpha C_1 = \{open , close\}$$
$$C_1 = (open \rightarrow close \rightarrow C_1)$$

تعریف جایگزین دیگری که در آن ما دو فرآیند را تعریف می‌کنیم خواهد بود:

$$\alpha C_1 = \{open\}$$
$$C_1 = (open \rightarrow C_2)$$

و

$$\alpha C_2 = \{close\}$$
$$C_2 = (close \rightarrow C_1)$$

در این‌جا اولین فرآیند C_1 دارای حروف { **open** } است که در یک حادثه درگیر است که در حروف رخ داده و سپس مانند فرآیند C_2 عمل می‌کند. فرآیند C_2 نیز دارای یک تک حرف حادثه است که در این رویداد درگیر بوده و سپس هم‌چون مانند C_1 عمل می‌کند. ناظر خارجی که به سیستم بیان شده به این شکل، نگاه می‌کند توالی حادثه را می بیند.

open , close , open , close , ...

این توالی به‌عنوان پی‌گیری فرآیند یا **trace** شناخته می‌شود. مثال دیگر مشخصات **CSP**در زیر نشان داده شده است. این مثال نمایانگر روباتی است که می‌تواند در دو واقعه مربوط به حرکت جلو و عقب

در یک خط درگیر شود. اگر کسی یک مسیر نامحدود برای حرکت بدون نقطه پایانی برای روبات در نظر بگیرد آن‌وقت به صورت زیر تعریف می‌شود:

$$\alpha ROBOT = \{forward , back\}$$
$$ROBOT = (forward \rightarrow ROBOT \mid back \rightarrow ROBOT)$$

در این‌جا روبات می‌تواند در دو امر جلو و عقب رفتن درگیر شود و سپس مانند یک روبات رفتار کرده و می‌تواند جلو یا عقب برود. دوباره، اجازه دهید با دادن فرضی برای معرفی بیشتر تسهیلات CSP این مشخصه را واقعی‌تر کنیم و مشکل این است که مسیری که روبات روی آن حرکت می‌کند شامل صدها حرکت جلو و عقب است که به یک حرکت اختصاص یافته است. هم‌چنین

فرض می‌کنیم که روبات در موقعیت ۱ روی این مسیر حرکت را شروع می‌کند. تعریف این $ROBOT_c$ در زیر آمده است:

$$\alpha ROBOT_c = \{forward , back\}$$
$$ROBOT_c = R_1$$
$$R_1 = \{forward \rightarrow R2)$$
$$R_i = (forward \rightarrow R_{i+1} \mid back \rightarrow R_{i-1}) \ (i<100 \wedge i>1)$$
$$R_{100} = (back \rightarrow R_{99})$$

در این‌جا روبات این‌گونه تعریف شده که دارای همان دو حرف حادثه مانند روبات قبلی است. مشخصات درگیری آن در این حوادث پیچیده‌تر است. دومین خط بیان می‌کند که روبات در موقعیت اولیه خاموش شده و همان‌گونه که در فرآیند R_1 توصیف شده که نمایانگر جای‌گیری آن روی اولین مربع است، رفتار می‌کند. سومین خط می‌گوید که فرآیند R_1 تنها می‌تواند در حادثه حرکت جلوی درگیر شود زیرا در نقطه انتهایی است. چهارمین خط می‌گوید یک‌سری فرآیند از R2 تا R99 را تعریف می‌کند. این کار این حقیقت را تعریف می‌کند که در هر نقطه‌ای روبات می‌تواند چه به جلو چه به عقب حرکت کند. از این خط آن‌چه را که وقتی روبات به انتهای مسیرش رسیده و رخ داده، مشخص می‌سازد. روبات تنها می‌تواند به عقب حرکت کند.

پس این مشخصه چگونگی تعریف فرآیندها را در CSP است. در سیستم‌های واقعی چندین فرآیند پردازشی وجود دارند که کار هم‌زمانی و برقراری ارتباط با یکدیگر را انجام می‌دهند، مثال‌هایی در زیر آمده است:

- در یک سیستم حادم/ محدوم، یک حادم وقتی فرآیندی با تعدادی مشتری ارتباط برقرار می‌کند، شروع به‌کار می‌نماید به‌همین شکل کار پردازش صورت می‌گیرد.

- در یک سیستم زمان واقعی که به یک راکتور شیمیایی را کنترل می‌کند فرآیندهایی وجود دارند که دمای راکتورها را کنترل می‌کنند که با فرآیندهایی مرتبطند که شیرهای اطمینان این راکتورها را باز و بسته می‌کنند.
- در سیستم کنترل ترافیک هوایی، عملکرد رادار می‌تواند با فرآیندهایی اجرا گردد که با فرآیندهایی در ارتباط هستند که کار نمایش موقعیت هواپیما را روی صفحه نمایش انجام می‌دهند.

بنابراین به اجرای موازی فرایندهای بهنمایش درآمده در **CSP** نیاز داریم. همراه با آن، به وسایلی نیاز داریم که در آن برقراری ارتباط و همزمان‌سازی بین این فرآیندها توسط آنها انجام شود. ابزاری که اجرای موازی در **CSP** را مشخص می سازد $\|$ است. بنابراین فرآیند **P** که نمایانگر اجرای موازی فرآیندهای P_1 , P_2 است به‌صورت زیر تعریف شده است.

$$P = P_1 \| P_2$$

همزمان ساری بین فرآیندها بهوسیله فرآیندهایی حاصل می‌گردد که دارای هم‌پوشانی در الفبایشان هستند. قانون مربوط به ارتباطات و همزمان‌سازی این است که وقتی دو فرآیند به‌صورت موازی اجرا می‌شوند و در آن دارای حوادثی مشترک هستند، باید آن حادثه را به‌طور همزمان اجرا کنند. به‌عنوان مثال یک سیستم ساده و غیرواقعی را درنظر بگیرید که چراغ‌ها را روشن و خاموش کرده و براساس مورد مشابهی در [**HIN 95**] است. تعاریف این دو فرآیند در این سیستم عبارتند از:

$$\alpha LIGHT_1 = \{on,off\}$$
$$LIGHT_1 = (on \rightarrow on \rightarrow STOP \mid off \rightarrow off \rightarrow STOP)$$
$$\alpha LIGHT_2 = \{on,off\}$$
$$LIGHT_2 = (on \rightarrow off \rightarrow LIGHT_2)$$

هر دو این فرآیندها دارای الفبای یکسانی هستند. اولین فرآیند چراغ را روشن کرده و دوباره چراغی را روشن می‌سازد. (به‌یاد داشته باشید که دیگر فرآیندهای سیستم ممکن است آن چراغ را در این بین خاموش کرده و سپس آن را محتل کنند) یا چراغی را خاموش نموده و سپس دوباره خاموش کند. فرآیند **LIGHT 2** ابتدا چراغ را روشن و سپس خاموش نموده و بعد دوباره مانند **LIGHT 2** عمل می‌کند. دو فرآیند موازی به‌صورت زیر آمده‌اند:

$$LIGHT_1 \| LIGHT_2$$

تأثیر اجرای این فرآیندها به‌صورت موازی چیست؟ در مقدمه‌ای همچون این، ما نمی‌توانیم زیاد وارد جزئیات شویم. می‌توانیم دلایل استعدلال به‌کار رفته در چنین عبارتی را بیان کنیم. به‌خاطرخواهید آورد که ،قبل **CSP** را معرفی کردیم بیان کردیم که تنها شاخل مورد نیاز، مشخص کردن فرآیندهای

استدلال بیاوریم. مثلاً تعیین کنیم که آیا حوادث بد مثل به بن‌بست رسیدن در سیستمی که در CSP مشخص شده رخ می‌دهد یا خیر. به‌منظور مشاهده آن‌چه که روی می‌دهد به‌کارگیری بعضی از قوانینی که هور برای CSP بیان کرده، ارزشمند است. به‌خاطر دارید که می‌خواهیم بفهمیم چه فرآیندی توسط احرای موازی دو فرآیند $LIGHT_1$, $LIGHT_2$ تعریف شده است. این عبارت مساوی است با :

$$LIGHT_1 = (on \rightarrow on \rightarrow STOP \mid off \rightarrow off \rightarrow STOP)$$
$$LIGHT_2 = (on \rightarrow off \rightarrow LIGHT_2)$$

یکی از قوانین هورر به قرار ذیل بود :

$$(e \rightarrow P) \parallel (e \rightarrow q) = e \rightarrow (P \parallel Q)$$

و این به ما اجازه می‌دهد که عبارت را با استفاده از $LIGHT_1$ و $LIGHT_2$ بسط دهیم:

$$(on \rightarrow ((on \rightarrow STOP) \parallel (off \rightarrow LIGHT_2)))$$

که این عبارت نیز با استفاده از قانون دیگری از هورر به عبارت ذیل ساده خواهد شد:

$$(on \rightarrow STOP)$$

و به این معناست که اجرای موازی دو فرآیند هم‌ارز است با روشن ساختن یک لامپ و آنگاه رفتن به بن‌بست.

این یک توصیف ساده از CSP بود - چون تمام شیوه‌های رسمی دیگر، نشانه‌گذاری‌های ریاضی و قوانین استدلال را شامل می‌شد. البته آنچه ما از Z و Object Z مطرح دادیم، به دنبال اثبات قوانین و تمرکز بر رسمی‌گرایی و فرمالیسم نبودیم حال آنکه انها شامل تسهیلات مهم و قابل توجهی برای استدلال و اثبات خصوصیات یک سیستم می باشند.

تذکر

تصمیم گیری در خصوص استفاده از شیوه های رسمی شفاف نخواهد بود. این احکام را دنبال کنید و از آموزش صحیح همگان اطمینان [illegible]

۹-۲۵ ده فرمان برای شیوه های رسمی

تصمیم در خصوص استفاده از شیوه رسمی در دنیای واقعی، موردی نیست که بتوان آن را آسان گرفت. بوان و مسکلی " ده فرمان برای استفاده رسمی از شیوه ها " به عنوان راهنمایی برای افرادی که می خواهند این مبحث مهم در مهندسی نرم افزار دست یابند، وضع کرده اند.

۱- **یک نشانه گذاری اختصاصی برای خود انتخاب کنید.**

برای استفاده موثر از زبانهای اختصاصی و رسمی بطور گسترده، مهندسی نرم افزار باید لغات زبان، انواع خاص ساختار زبان را بررسی نموده و استفاده از آن را گسترش دهد.

۲- به روشها رسمیت دهید اما نه بیش از حد لازم.

به طور عمومی نیازی به اختصاص شیوه های رسمی به هر جزء کوچکی از سیستم اصلی نمی باشد. این اجزاء که از نظر انتقادی ایمن هستند جزء اولین انتخابها می باشند و همراه اجزائی می آیند که شکست آنها جایز نمی باشد.(از لحاظ دلایل تجاری)

۳- هزینه ها را برآورد کنید.

شیوه های رسمی آغازی پر هزینه دارندآموزش کارکنان، کسب ابزار پشتیبانی و استفاده از قراردادهای مشورتی در ابتدای کار که متحمل هزینه های زیادی می باشد. این هزینه ها باید هنگام کار و تحقیق بر بازگشت سرمایه گذاری مربوط به شیوه های رسمی بررسی گردند.

۴- یک خط ارتباطی برای راهنمایی روشهای رسمی بر قرار سازید.

هنگامی که برای اولین بار از شیوه های رسمی استفاده می شود، آموزش پیشرفته و تخصصی و مشورتهای زنده و جاری، مواردی ضروری برای دستیابی به موفقیت قلمداد می شوند.

۵- از توسعه شیوه های سنتی و کلاسیک خود نیز صرفنظر نکنید.

استفاده از شیوه های رسمی به همراه روشهای متداول یا روشهای ش، گرا، در برخی موارد مطلوب نیز می باشد. هرکدام آن ها نقاط ضعف و قوتی برای خود دارند ترکیب آنها اگر ممکن باشد، نتایج فوق العادة ای را در بر خواهد داشت.

۶- مدارک و اسناد واجد شرایط ارائه کنید.

شیوه های رسمی ، اصول فشرده، نامبهم و ثابتی را برای مستندسازی نیازمندیهای سیستم تهیه می کند. اگرچه پیشنهاد شده است که تفسیر زبان طبیعی، برای خوانندگانی که زبان سیستم را منوجه نمی شوند، به همراه زبان تخصصی رسمی و به عنوان مکانیزمی تقویتی به کار رود.

۷- از استانداردهای کیفیت غافل نشوید. شیوه های رسمی معجزه نمی کنند[BOW95] و به همین علت، فعالیتهای دیگر تضمین کیفیت نرم افزار (فصل ۸) باید ادامه یابند.

۸- تعصب به خرج ندهید. یک مهندس نرم افزار باید بداند که شیوه های رسمی نسبت به صحت و درستی، تضمین و گارانتی نمی دهند. محتمل است (شاید هم بیشتر!) که سیستم نمایی، حتی اگر با شیوه های رسمی توسعه یافته باشد، از قلم افتادگی ها، اشکالات پنهان و دیگر استثناء هایی را در برگیرد.

۹- شما را سفارش می کنم به آزمون، آزمون و آزمون دوباره. اهمیت آزمون نرم افزار در فصول ۱۷ و ۱۸ و ۲۳ آورده شده است. شیوه های رسمی، مهندسی نرم افزار را از کاربران آزمون های خوب برنامه ریزی شده، بی نیاز نمی سازد.

۱۰- **استفاده مجدد را بکار برید.** در دراز مدت تنها یک راه برای کاهش هزینه و افزایش کیفیت، متصور است و آن، استفاده مجدد می باشد (فصل ۲۷). شیوه های رسمی این حقیقت را تغییر نمی دهند. در واقع، شیوه های رسمی هنگامی یک رهیافت وطلوب خواهند بود که اجزاء در یک کتابخانه برای استفاده مجدد، قرار گیرند.

۲۵-۱۰ شیوه های رسمی – راهی فراروی

هرچند فنون مبتنی بر مشخصه های ریاضی و رسمی هنوز در صنعت به طور گسترده ای مورد استفاده ندارد، امتیازات بسیاری نسبت به سیستمهای کمتر رسمی از خود نشان می دهد. این امر را لیسکو و برسینس [LIS86] در جملات زیر خلاصه کرده اند :

مشخصه های رسمی بطور ریاضی قابل مطالعه می باشند در حالیکه مشخصه های غیر رسمی اینگونه نیستند. برای مثال یک برنامه درست، می تواند جهت تصدیق مشخصه هایش تضمین گردد، یا دو مجموعه جایگزین از مشخصه ها می توانند بطور معادل ارزیابی و تصدیق شوند... شکلهای اصلی کامل نبودن ها و ناسازگاری ها می تواند به طور خودکار تشخیص داده شود.

به علاوه، مشخصه های رسمی ابهام را زایل می نماید و در مقابل سختی های بزرگتری که در مراحل اولیه فرآیند مهندسی نرم افزار وجود دارد، دلگرم کننده است.

ولی مسائلی نیز باقی می مانند. مشخصه های رسمی بر توابع و داده ها متمرکز می شوند. زمان، کنترل، و برخی عناصر دیگر یک مسئله (مانند رابط انسان / ماشین)، توسط نمونه سازی ها یا فنون تصویری و گرافیکی به طرز مناسب تری مشخص خواهند شد. در نهایت فراگیری مشخصه های به کار رفته در شیوه رسمی سخت تر و پیچیده تر از یادگیری تحلیل ساختیافته خواهند بود و به این ترتیب بیم بروز یک " شوک فرهنگی " برای برخی متخصصین نرم افزاری وجود دارد. این دلایل باعث می شود که فنون مشخصه های رسمی و ریاضی در قالب ابزاری های نسل آینده **CASE** نمود پیدا نمایند. هنگامی که این امر روی دهد، البته اگر روی دهد مشخصه های مبتنی بر ریاضی توسط قشر وسیعی از جامعه نرم افزاری مورد اقبال قرار گرفته و پذیرفته خواهند شد.

۲۵-۱۱ خلاصه

شیوه های رسمی زیربنای محیط های خاص می باشند که رسیدن به مدل های تحلیلی کامل را هدایت می نمایند. مدل هایی که کامل تر، سازگارتر بوده و نسبت به شیوه های سنتی و شئ گرا از ابهام کمتری برخوردارند. تسهیلات توصیفی تئوری مجموعه ها و نشانه گذاری منطقی، یک مهندس نرم افزار را قادر می سازد که جملاتی شفاف از حقایق و نیازمندیها ارائه کند.

مفاهیمی که در شیوه های رسمی به کار می رود عبارتند از : ۱) داده های تغییر ناپذیر شرطی که در خلال اجرای سیستم مملو از داده ها، صحیح خواهد بود. ۲) وضعیت، داده ذخیره شده که یک سیستم آن را مورد استفاده و ارجاع قرار می دهد، و ۳) عملیاتی که از دو شرط تشکلی شده است : یک پیش شرط و یک پس شرط.

ریاضیات گسسته - علائمی از مجموعه ها و مشخصه های ساختمانی، دارد، عملگرهای مجموعه ها، عملگرهای منطقی، و دنباله ها - شکلهای پایه ای شیوه های رسمی را تشکیل می دهند. ریاضیات گسسته به صورت بافت یک زبان مشخصه رسمی همچون **Z** پیاده سازی می شود.

Z ، مانند تمام زبانهای مشخصه رسمی، هردو حوزه معنایی و نحوی را پوشش می دهد. حوزه نحوی از یک سری نمادها بهره می برد که معادل علائم مجموعه ها و گراره های ریاضی است.

حوزه معنایی زبان را قادر می سازد که نیازمندیها بصورت کوتاه و موجز بیان گردند. ساختار **Z** متشکل از الگوهایی - جعبه ای- مانند ساختارهایی است که متغیرند و رابطه ای را میان این متغیرها برقرار می سازد.

تصمیم در خصوص استفاده از شیوه های رسمی باید هزینه های شروع کار را در نظر گرفته و به همان صورت هم، فرهنگ اعمال تغییرات لازم را مورد توجه قرار دهد. با فن آوری ای که بسیار متفاوت است. در بیشتر موارد شیوه های رسمی سیستم های بحرانی و حساس تجاری و امنیتی را در سطح بالایی پوشش می دهد

مسایل و نکاتی برای تفکر و تعمق بیشتر

۲۵-۱ انواع مختلف عیب ها و نقص های مربوطه را با شیوه های دستیابی رسمی مهندسی نرم افزار در فصل ۲۵-۱ مرور کنید. سه مثال از هر کدام برای افزودن تجربیات خود تهیه نمایید

۲۵-۲ قواعد ریاضی به عنوان مکانیزم مخصوص به طور کامل در این فصل مورد بحث قرار گرفته شده است. آیا نقطه ضعفی هم دارد؟

۲۵-۳ شما به گروهی تخصیص یافته اید که کارشان توسعه نرم افزار برای فکس / مودم می باشد. کار شما ساخت قسمت " دفتر تلفن "دستورالعمل خواهد بود. ساختار دفتر تلفن به گونه ای است که امکان ذخیره سازی افراد به همراه اسم شرکت، شماره فکس و دیگر اطلاعات وابسته را فراهم می سازد. از زبان طبیعی استفاده کرده و موارد زیر را تعریف کنید:

الف- داده های ثابت

ب- چگونگی و کیفیت (case)

ب- عملیات محتمل (likely)

۲۵-۴ شما به گروه نرم افزاری تخصیص یافته اید که کارش توسعه نرم افزاری است و دوبرابرکننده حافظه" نامیده می شود. کار آن تهیه حافظه ای بزرگ برای PC است صرفنظر از حافظه فیزیکی در اختیار. این شیوه به وسیله شناسایی، گردآوری و دوباره سازی بخشهای حافظه که به دستورالعمل های موجود اختصاص داده شده اما مورد استفاده قرار نگرفته اند، تکمیل می شود. بخشهای استفاده نشده به دستورانی که نیاز به حافظه اضافی دارند، تخصیص داده می شوند. با استفاده از زبان طبیعی و فرضیات مناسب موارد زیر را تعریف کنید:

الف – داده های ثابت

ب- چگونگی کیفیت

ب- عملیات محتمل

۲۵-۵ پساخبار خاصی را به گروهی اختاص دهید که شامل اعداد طبیعی چندتایی به شکل $(x,y,z)^2$ بوده و به طور مثال جمع x و y برابر با z شود.

۲۵-۶ نصب کننده دستورالعمل های PC ابتدا مشخص می سازد که آیا گروه قابل قبول سخت افزاری و منابع سیستم آماده هست یا خیر. آن شکل ظاهری سخت افزار را برای مشخص کردن وجود قطعات منبع و نوع خاصی از سخت نرم افزاری و این مورد که راه اندازها قبلا" نصب شده اند یا خیر، بررسی می کند.

چه گروه عملیاتی برای این کار می تواند مورد استفاده قرار گیرد؟ مثالی را در این زمینه تهیه کنید.

۲۵-۷ سعی کنید برای مثال های زیر از بیان منطقی و عملیات گروهی استفاده نمایید. " برای همه x و y ها اگر x والد y و y والد x باشد ، در نتیجه x جد z می شود. هرکس یک والد دارد. نکته : از ساختار P(x,y) و G(x,z) برای سازگار کردن والد و جد استفاده نمایید.

۲۵-۸ ساختاری از گروه خاصی از یک جفت عدد را شرح دهید که اولین عنصر هر جفت، مجموعه دو عدد طبیعی غیرصفر و دومین عنصر، تفاوت بین همان اعداد باشد. هر دو عدد باید بین ۱۰۰ و ۲۰۰ باشند.

۲۵-۹ توضیحی به شیوه ریاضی برای داده های ثابت و حالت و چگونگی در مثال ۲۵-۳ ارائه دهید. سپس این توضیح را به زبان خاص Z بهبود بخشید.

۲۵-۱۰ توضیحی به شیوه ریاضی برای داده های ثابت و حالت و چگونگی در مثال ۲۵-۴ ارائه دهید. سپس این توضیح را به زبان خاص Z بهبود بخشید.

۲۵-۱۱ از نشانه گذاری z که در مثال ۲۵-۱ آمده است، استفاده کنید. قسمتهایی از سیستم خانه امن که در بخشهای قبل آمده بود، انتخاب کرده و سعی کنید آن را با روش Z مشخص نمایید.

۲۵-۱۲ با استفاده از اطلاعات اشاره شده در این فصل یا اطلاعات متنوعی که در منابع دیگر موجود اند، در مورد زبان اختصاصی رسمی علاوه بر زبان Z و ساختارهای اصلی دستوری یا معنایی آن شرحی بیاورید.

فهرست منابع و مراجع

[BOW95] Bowan, J.P. and M.G. Hinchley, "Ten Commandments ofFormal Methods," *Computer*, vol. 28, no. 4, April 1995.

[GRI93] Gries, D. and F.B. Schneider, *A Logical Approach to Discrete Math*, Springer-Verlag, 1993.

[GUT93] Guttag, J.V., and J.J. Horning, *Larch: Languages and Tools for Formal Specification*, Springer-Verlag, 1993.

[HAL90] Hall, A., "Seven Myths of Formal Methods," *IEEE Software*, September 1990 pp. 11-20.

[HIN95] Hinchley, M.G. and S.A. Jarvis, *Concurrent Systems: Formal Developme nt -, CSp*, McGraw-Hill, 1995.

[HOR85] Hoare, C.A.R., *Communicating Sequential Processes*, Prentice-Hall International, 1985.

[JON91] Jones, C B., *Systematic Software Development Using VDM*, 2nd ed.,Prentice-Hall, 1991.

[LIS86] Liskov, B.H., and V. Berzins, "An Appraisal of Program Specifications," in *Software Specification Techniques*, N. Gehani and A.T. McKittrick *(eds.)*, Addison-Wesley, 1986, p. 3.

[MAR94] Marciniak, J.J. (ed.), *Encyclopedia of Software Engineering*, Wiley, 1994.

[ROS95] Rosen, K.H., *Discrete Mathematics and Its Applications*, 3rd ed., McGraw-Hill, 1995.

[SPI88] Spivey, J.M., *Understanding Z: A Specification Language and Its Formal Semantics*, Cambridge University Press, 1988.

[SPI92] Spivey, J.M., *The Z Notation: A Reference Manual*, Prentice-Hall, 1992.

[WIL87] Wiltala, S.A., *Discrete Mathematics: A Unified Approach*, McGraw-Hill, 1987.

[WIN90] Wing, J.M., "A Specifier's Introduction to Formal Methods," *Computer*, vol. 23, no. 9, September 1990, pp. 8-24.

[YOU94] Yourdon, E., "Formal Methods," *Guerrilla Programmer*, Cutter Information Corp., October 1 994.

خواندنیهای دیگر و منابع اطلاعاتی

In addition to the books used as references in this chapter, a fairly large number of books on formal methods topics have been published over the past decade. A listing of some of the more useful offerings follows:

Bowan, J., *Formal Specification and Documentation using Z: A Case Study Approach*, International Thomson Computer Press, 1996.

Casey, C., *A Programming Approach to Formal Methods*, McGraw-Hill, 2000.

Cooper, D. and R. Barden, *Z in Practice*, Prentice-Hall, 1995.

Craigen, D., S. Gerhart, and T. Ralston, *Industrial Applicanon of Formal Methods to Model, Design and Analyze Computer Systems*, Noyes Data Corp., 1995.

Diller, A., *Z: An Introduction to Formal Methods*, 2nd ed., Wiley, 1994.

Harry, A., *Formal Methods Fact File: VDM and Z*, Wiley, 1997.

Hinchley, M. and J. Bowan, *Applications of Formal Methods*, Prentice-Hall, 1995.

Hinchley, M. and J. Bowan, *Industrial Strength Formal Methods*, Academic Press, 1997.

Hussmann, H., *Formal Foundations for Software Engineering Methods*, Springer-Verlag, 1997.
Jacky, J., *The Way of z: Practical Programming with Formal Methods*, Cambridge University Press, 1997.
Lano, J. and H. Haughton (eds.), *Object-Oriented Specification Case Studies*, Prentice-Hall, 1993.
Rann, D., J. Turner, and J. Whitworth, Z: *A Beginner's Guide*, Chapman and Hall, 1994.
Ratcliff, B., *Introducing Specification Using Z: A Practical Case Study Approach*, McGraw-Hill, 1994.
D. Sheppard, *An Introduction to Formal Specification with Z and VDM*, McGraw-Hill, 1995.

The September 1 990, issues of *IEEE Transactions on Software Engineering*, *IEEE Software*, and *IEEE Computer* were dedicated to formal methods. They remain an excellent source of useful information.

Schuman *(Formal Object-Oriented Development*, Springer-Verlag, 1996) has edited a book that addresses formal methods and object technologies, providing guidelines on the selective use of formal methods and showing how such methods can be used in conjunction with 00 approaches. Bowman and Derrick *(Formal Methods for Open Object-Based Distributed Systems*, Kluwer Academic Publishers, 1997) address the issues of formal methods when coupled with 00 applications in a distributed environment.

A wide variety of information sources on formal methods and related subjects is available on the Internet. An up-to-date list of World Wide Web references that are relevant to formal methods can be found at the SEPA Web site:
http://www.mhhe.com/engcs/compsci/pressman/resources/formal-methods.mhtml