تکنیک ها وزبانهای برنامه نویسی هوش مصنوعی

این انقلاب در اقتصاد امروز و نظم جامعه، به همان میزان انقلاب صنعتی در قرن 19 تأثیر داردواین تحولات قادر است الگوی فکری و فرم زندگی هر فرد را تغییر دهد.

انقلاب کامپیوتر توان ذهنی ما را گسترش می دهد.

عملکرد اولیهٔ برنامه نویسی هوش مصنوعی (AI) ایجاد ساختار کنترلی مورد لزوم برای محاسبه سمبولیک است خصوصیات این ساختارها به مقدار زیادی موجب تشخیص خصوصیاتی می شود که یک زبان کاربردی می بایستی فراهم کند.

در این مقدمه به یک سری خصوصیات مورد نظر برای زبان برنامه نویسی سمبولیک می پردازیم و زبانهای برنامه نویسی LISPو PROLOG را معرفی خواهیم کرد.

این دو زبان علاوه بر این که از مهمترین زبانهای مورد استفاده در هوش مصنوعی هستند، خصوصیات semantic و syntactic آنها نیز باعث شده که آنها شیوه ها و راه حل های قوی برای حل مسئله ارئه کنند.

تأثیر قابل توجه این زبانها بر روی توسعه AI از جمله توانائی آنها به عنوان «ابزارهای فکر کردن» می باشد که از جمله نقاط قوت آنها در زبانهای برنامه

نویسی می باشد.

همان طور که هوش مصنوعی مراحل رشد خود را طی می کند زبانهای LISPو PROLOG بیشتر مطرح می شوند.

این زبانها کار خود را در محدودهٔ توسعه و prototype سازی سیستم های AI در صنعت و دانشگاهها دنبال می کنند.

اطلاعات در مورد این زبانها به عنوان بخشی از مهارت هر برنامه نویس AI می باشد ما به بررسی این دو زبان در هوش مصنوعی می پردازیم.

آنـــــچه را کـــه نمی دانیم موجب دردسر و گرفتاری ما نخواهد شد، بلکه دردسرها از دانسته ها سرچشمه می گیرند.

W.ROGERS

زبان ، شناخت و خلاصه پردازی

توانایی شکل گیری خلاصه برداری از تجربیات از توانمند ترین و اساسی ترین توانائی های ذهن انسان است خلاصه پردازی به ما این اجازه را می دهد که به فهم جزئیات از یک محدوده ی کلی اطلاعات مربوط به یک خصوصیت کلی سازمان و رفتار برسیم . این خلاصه ها به ما اجازه شناخت و درک کامل موارد دریافت شده در حوزه خاص را می دهند . اگر ما وارد یک خانه شویم که به خوبی ساخته شده باشد ، راههای خود را به اطراف پیدا خواهیم کرد . ساختار

خصوصیات اطاق نشیمن ، اطاق خواب ،آشپزخانه و حمام عموماً از ویژگیهای یک مدل خانۀ استاندارد می باشد .

خلاصه پردازی به ما حس شناخت خانه های متفاوت را می دهد . یک تصویر ممکن است بیانی قوی تر از هزاران کلمه داشته باشد ، اما یک خلاصه مشخصاً بیان کنندۀ خصوصیات مهم یک کلیت از نوع تصویر است .

وقتی که ما به تئوری برای توصیف کلاس های یک پدیده می پردازیم ، خصوصیات و ویژگیهای کمی و کیفی مربوط به کلاس از کل جزئیات خلاصه می شود .

که اعضاء به خصوص خود را مشخص می کند . این کاهش جزئیات به وسیله قدرت توصیف و پیش بینی یک نظریه ارزشمند جبران می شود .

خلاصه سازی یکی از ابزارهای اساسی شناخت و ارزیابی کلیت های جهان اطراف ما و همچنین ساختار ذهنی ما است . در حقیقت این پروسه به طور مداوم براساس دانش و اطلاعات صورت می گیرد . دانش و اطلاعا نیز در لایه ها و بخش هایی از خلاصه پردازی ساخته می شود که از مکانیسم هایی که ساختار را فشرده ساخته و از حس اولیه به سمت یک سری تئوری های علمی سوق داده

می شود و در نهایت بیشتر این ایده ها دربارهٔ ایده های دیگر و نشأت گرفته از آنها می باشد .

خلاصه پردازی طبقه بندی شده (سلسله مراتبی) :

ساختار و سازمان آزمایش و تجربه در ارتباط با توصیفات کلاس های خلاصه سازی یکی از ابزارهای شناخت رفتار و ساختار سیستم های مرکب است که شامل برنامه های کامپیوتر می شوند .

همانند رفتار یک حیوان که ممکن است بدون توجه به فیزیولوژی سیستم عصبی نهفته در پشت آن مورد مطالعه قرار گیرد .

یک الگوریتم دارای خصوصیات مربوط به خود می باشد که کاملاً آن را از برنامه ای که آن را به کار می برد جدا می سازد .

به عنوان مثال دو نوع کاربر متفاوت جستجوی باینری را در نظر بگیرید .

یکی از آنها یعنی Fortran از محاسبات و طبقه بندی استفاده می کند و دیگری یعنی Ctt از Pointer استفاده می کند که بتواند در جستجوی درون شاخه های binary کاربرد داشته باشد .

اگر دقیق تر نگاه کنیم این برنامه ها مثل هم می باشند چون اگر جز این باشد

کاربردهای آنها نیز تفاوت خواهد شد . جداسازی الگوریتم از که مورد استفاده در کاربرد آن یکی از نمونه های خلاصه سازی سلسله مراتبی می باشد .

Allen New ell بین سطح دانش و سطح نشانه ها برای توصیف یک سیستم هوشمند تفاوت قائل شده است.

سطح نشانه ها همراه سازماندهی به خصوصی مورد توجه قرار گرفته که برای بیان اطلاعات حل مسئله مورد استفاده قرار می گیرد. بحث مربوط به توجه به منطق به عنوان یک زبان یک نمونه از مواردی است که به سطح نشانه پرداخته است.

علاوه بر سطح نشانه سطح دانش است که توجه آن به مقدار و محتوی اطلاعات یک برنامه و شیوه استفاده از آن اطلاعات می باشد.

این نوع تمایز در ساختار و معماری سیستم هایی که بر اساس دانش و اطلاعات و سبک توسعه ای که آن را پشتیبانی می کتد منعکس می گردد.

به دلیل اینکه کاربرها برنامه ها را در قالب دانش و توانایی خودشان می شناسند بنابراین حائز اهمیت است که برنامه های AI دارای یک سطح خصوصیات اطلاعاتی باشند.

جداسازی اصل دانش و اطلاعات از ساختار کنترل این نظریه را آشکار می سازد

و توسعه رفتار سطح دانش را ساده می سازد.

همانند این نیز سطح نشانه ای یک زبان توصیفی را تشریح می کند که شبیه قوانین و روشهای تولید یا منطق براساس دانش و اطلاعات می باشد.

جداسازی آن از سطح و دانش و اطلاعات نه برنامه نویس این اجازه را می دهد که به سمت خلاصه پردازی ت،ثیر پذیری و راحتی برنامه نویسی سوق پیدا کندکه در ارتباط با رفتار و عملکرد بالای برنامه نمی باشد.

کاربرد بیان سطح نشانه ای شامل یک سطح دوره پائین تر از ساختار برنامه می شود و بیانگر یک سری ملاحظات طراحی اضافی می شود.

اهمیت نظریه چند مرحله ای نسبت به طراحی سیستم نمی تواند بیش از این مورد توجه قراار گیرد.

یعنی اینکه به برنامه نویس اجازه می دهد که با پیچیدگی نهفته شده در سطوح پائین تر خود را درگیر نکند و توجه و تاکیدش بر روی منابع مناسب با سطح فعلی خلاصه پردازی کند.

همچنین موجب می شود که اصول تئوری هوش مصنوعی عاری از کاربردهای خاص یا زبان برنامه نویسی باشد . این همچنین به ما قدرت توصیف یک کاربرد را می دهد و تاثیر گذاری خود را بر روی ماشین دیگر اثبات می کند بدون اینکه

بر رفتارش در سطوح بالاتر تاثیر بگذارد .

سطح اطلاعات توصیف کننده توانائی های یک سیستم هوشمند است. محتوی دانش و اطلاعات مستقل از شکل پذیری مورد استفاده برای بیان آن است به همان اندازه که زبان بیان کاملا مؤثر می باشد .

توجه به سطح دانش شامل سؤالاتی از این قبیل است:

از این سیستم چه چیزی ساخته خواهد شد؟ چه اشیا و چه ارتباطی در آن محدوده مؤثر و مفید است ؟ چگونه یک اطلاعات جدید به سیستم اضافه می گردد؟

آیا واقعیات در طی زمان تغییر می کنند؟ چگونه و چطور سیستم نیازمند است که دلائل اطلاعات خود را ثابت کند؟ آیا محدوده ارتباطی دارای یک طبقه بندی درست و شناخته شده است؟

آیا این محدوده شامل یک سری اطلاعات نادرست و غیر ممکن است؟

تجزیه و تحلیل دقیق در این سطح یک گام مبهم در طراحی کلی ساختار یک برنامه می باشد.

در سطح نشانه تصمیمات درباره ساختارها صورت می گیرد که برای بیان و ایجاد دانش مورد استفاده قرار می گیرند. انتخاب یک زبان برای بیان یک مورد مربوط به سطح نشانه می باشد.

منطق یکی از چندین نوع اشکال است که اصولا در حال حاضر برای بیان دانش و اطلاعات در دسترس می باشد.

زبان بیان نه تنها می بایستی توانایی بیان اطلاعات مورد لزوم برای کاربر را داشته باشد بلکه می بایستی خلاصه و قابل توصیف و دارای کاربرد مؤثر باشد و می بایستی به برنامه نویس برای دستیابی و سازماندهی اصل و اساس اطلاعات کمک کند.

وقتی که بین سطح اطلاعات و سطح نشانه یک برنامه تمایز به وجود آمد ما می توانیم بین سطح نشانه و الگوریتم و ساختمان داده ها مورد استفاده برای کاربرد آن نیز تمایز قایل شویم. به عنوان مثال بدون تاثیرگذاری رفتار و عملکرد یک تحلیل گر برنامه که اساس منطقی داشته باشد می بایستی تاثیر ناپذیر از انتخاب بین یک سری جزئیات و یک مجموعه و دسته بایزی باشد تا بتواند یک جدول مربوط به نشانه ها را به کار برد.

این تصمیمات کاربردی هستند و می بایستی در سطح نشانه قابل رؤیت باشند .

بسیاری از الگوریتم و ساختمان داده ها در کاربرد بیان زبان AI به کار می روند که از روشهای معمول علم کامپیوتر می باشند مثل شاخه ها و جداول بایزی.

دیگر موارد در رابطه با AI بسیار تخصصی هستند و به گونه یک که مستعار بیان می شوند که از طریق متن و بخش های مربوط به LISP و PROLOG بیان می شوند .

در سطح پائین تر مربوط به الگوریتم و ساختمان داده ها (سطح زبان) واقع شده است در این جا ست که زبان کاربردی برای برنامه مشخص می شود .

با این حال سبک برنامه نویسی مطلوب احتیاج به این دارد که ما یک خلاصه داده ای بسازیم که بین خصوصیات ویژه یک زبان برنامه نویسی و لایه های بالای آن قرار گیرد . نیازهای منحصر به فرد برنامه نویسی سطح نشانه ای تأثیر به روی طراحی و استفاده از زبانهای برنامه نویسی AI ایجاد می کند . علاوه بر این طراحی زبان می بایستی در برگیرنده و مطابق با ساختار آن که بر گرفته از سطوح پائین تر ساختمان کامپیوتر که شامل زبان اسمبلی و سیستم عامل و دستور العملهای ماشین و سطوح سخت افزار ی باشد .

و محدودیت های فیزیکی کامپیوتر می بایستی بر روی منابعی همچون حافظه و سرعت پردازشگر تأ کید کند . روش های LISP , PROLOG در جهت

مستعاذل کردن نیازهای سطح نشانه و نیازهای نهفته در ساختار هر دو منبع مورد استفاده می باشند و هم چنین یک هدف هوشمند و ذهنی با اهمیت می باشند . در دنباله ما از ساختارهای سطح اطلاعات در محیطهای برنامه نویسی بر روی یک زبان کاربردی صحبت خواهیم کرد و سپس به مصزفی زبانهای عمده AI یعنی PROLOG , LISP می پردازیم .

خصوصیات مطلوب یک زبان AI

یکی از خصوصیات و ویژگیهای مهم خلاصه سازی سلسله مراتبی در ساختار برنامه غیر حساس بودن سطوح بالاتر نهفته در کاربرد زبان می باشد .

این مشاهده در عمل سنجیده می شود که همراه با سیستم های موفق دانش مدار می باشد که در زبانهای برنامه نویسی مختلفی مثل Pascal , C , Ctt , Java , PROLOG , LISP و حتی Fortran به کار می رود .

برنامه های مختلفی اصولاً در LISP , PROLOG و سپس در C به کار گرفته می شوند تا بتواند تاثیرپذیری و انتقال پذیری بهتر ایجاد کنند. در هر دوی این موارد رفتار و عملکرد در سطح نشانه به طور قطع بی اثر می باشد.

با این حال محدودیتهای خلاصه سازی در یک برنامه جامع بیان می شود که

کامل نمی باشد . ساختار سطح بالاتر باعث ایجاد ساختارهای قوی بر روی لایه های زیرین می شود و نیاز به این دارد که برنامه نویسی AI بر روی سطح نشانه ای قرار گیرد که در سطح زبان تکرار می شوند.

به عنوان مثال ساختارهای اطلاعاتی مورد لزوم برای ادغام سمبولیک خود را مقید به اشکال تکراری مثل فلش ها و لیست ها نمی کنند.

اهداف و پیش بینی های منطقی ابزارهای کاربردی طبیعی تر و انعطاف پذیرتر خواهند بود.

علاوه بر این به دلیل مشکلات موجود در بسیاری از مسائل مربوط به AI اغلب توسعه را قبل از اینکه یک شناخت کامل از نهایت فرم برنامه داشته باشیم شروع می کنیم.

توسعه AI لزوما در طبیعت به صورت کشف و تجزیه و آزمایش است.

این نیاز هم چنین وابسته به یک زبان و ابزارهایی است که باید فراهم ساخت . یک زبان نه تنها می بایستی متناسب با کاربرد ساختارهای سطح بالا باشد بلکه می بایستی یک ابزار مناسب برای انتقال کل چرخه نرم افزار از آنالیز و تجزیه و تحلیل تا حصول برنامه باشد.

در پنـــج زیر گـــروه بعدی ما به صورت جزئی و کامل در مورد نیازهایی که

ساختارهای سطح نشانه ای برنامه های AI که بر روی کاربرد زبان دارند بحث می کنیم.

این موارد عبارتند از :

1. ***پشتیبانی از محاسبه سمبولیک***
2. انعطاف پذیری کنترل
3. پشتیبانی از متدولوژی و روش های برنامه نویسی جستجویی
4. پویایی
5. مستنند سازی خوب و واضح

`پشتیبانی از محاسبات سمبولیک

گرچه روش های زیادی برای سازماندهی اطلاعات در یک سطح نشانه وجود دارد . ولی تمامی آنها نهایتاً به عنوان عملکردهایی بر روی نشانه ها به کار می روند . این روش در تئوری نشانه های آقای Simon , Newell آمده است . تئوری های سیستم فیزیکی نشانه نیاز اصلی برای زبان برنامه نویسی است که کاربردهای یک سری از عملیات سمبولیک را آسان می کند .

حتی شبکه های عصبی و دیگر شکل های ضروری محاسبه می بایستی شامل

اطلاعات سمبولیک در ورودیها و خروجی هایشان باشند . انواع کاربردها و اطلاعات دادهای عددی تاکید شان بر روی زبانهای برنامه نویسی معمول است که برای کاربردهای جستجوی الگوریتمی یا بیان زبان AI مناسب نمی باشند.

علاوه بر این یک زبان AI می بایستی ساختار ایجاد نشانه های اولیه را ساده سازد و بر روی آنها کار کند. این یکی از مهمترین نیازهای یک زبان برنامه نویسی AI می باشد.

محاسبات و پیش بینی یکی از قوی ترین و عمومی ترین ابزارهای ایجاد ساختار کیفی یک محدوده از مسئله می باشد.

خصوصیات بارز یک محدوده ممکن است به گونه یک سری واقعیات منطقی بیان شود. از طریق استفاده از متغیرها امکان ایجاد واقعیات کلی درابره ارتباط بین اهداف در یک محدوده به وجود می آید.

PROLOG یک زبان برنامه نویسی کلی است که بر اساس پیش بینی محاسباتی است.

به عنوان یک کاربرد رسمی منطق PROLOG بعضی اوقات مستقیما به عنوان یک زبان در سطح نشانه مورد استفاده قرار می گیرد.

با این حال قدرت واقعی آن به عنوان یک زبان برای کاربرد دقیق تر و کامل

همانند چهارچوب ها و شبکه ها در یک روش سیستماتیک و فشرده می باشد بسیاری از ساختارهای سطح نشانه ای به سادگی با استفاده از ساختارهای سطح بالای PROLOG ساخته می شوند.

PROLOG ممکن است برای کاربرد در جستجوی الگوریتم ها یک سیستم محافظ و یک شبکه سمانتیکی مورد استفاده قرار گیرد.

یک ابزار مهم دیگر برای ساخت ساختارهای نشانه لیست می باشد یک لیست شامل یک سری عناصر می شود که در آن هر عنصر ممکن است حتی یک لیست و یک نشانه باشد.

چند نمونه از لیست ها با استفاده از ساختار برنامه نویسی LISP عبارتند از :

(این یک لیست است)

(این هست) (یک لیست) (از لیست ها)

(زمانها (بعلاوه 13)(بعلاوه 23))

((123)(456)(789))

توجه داشته باشیم که اینها نمونه هایی می باشند که شامل لیستهای درون لیست های دیگر می شود این موجب می شود که ارتباطات ساختاری ایجاد گردد. قدرت لیست ها عمدتا در نتیجه توانایی بیان هر نوع ساختار نشانه ای

بدون در نظر گرفتن پیچیدگی یا عملکردهایی که می باید از آن پشتیبانی کند می باشد.

این شامل شاخه ها گراف های اولیه یک سری مشخصه های منطقی جهت ها اصول اطلاعاتی کلیدی می شود. به طور خلاصه هر نوع ساختار ممکن است بر اساس یک ترکیب مناسب متشکل از لیست ها و عملکردهای واقع شده بر روی آنها حاصل شوند.

لیست ها یک سری بلوک های مهم می باشند که PROLOG , LASP که موجب می شود که کاربر را با عناصر اطلاعاتی و عملیاتی برای دستیابی و تاکید بر آنها در درون یک سری ساختارهای پیچیده مهیا سازد. در حالیکه PROLOG مستقیما به محاسبات پیش بینی شده وابسته است و شامل یک سری لیست به عنوان ابزارهای بیان می شود.

LISP لیست را به عنوان اصول انواع داده ها و برنامه ها مورد استفاده قرار می دهد. تمامی ساختارهای LISP از لیست ها ساخته می شوند و زبان فراهم کردن یک سری ابزارهای قوی برای ترکیب اینها (ساختارها) را به عهده دارد و توصیف کننده عملیات جدید برای ایجاد توسعه و تغییر آنها است. یک شکل کردن ساختار LISP و توانائی توسعه آن توصیف هر نوع زبانی را برای ساختار

آن ساده می سازد . بوسیله پرداختن به نظریه جمع آوری اطلاعات فشرده برنامه نویس LISP می تواند ساختارهای نشانه را توصیف کند و عملیات مورد نیاز هر نوع شکل گیری سطح بالا شامل کنترل کننده های جستجو حل کننده های تئوریهای منطقی و دیگر اظهارات سطح بالا می باشد.

انعطاف پذیر بودن کنترل:

یکی از مشخصه های اساسی رفتار هوشمند قابلیت انعطاف پذیری آن می باشد . در حقیقت مشکل بتوان تصور کرد که هوشمندی می تواند از طریق توسعه گام به گام مراحل ثابت که بوسیله برنامه های معمول کامپیوتری نشان داده می شود حاصل شود. خوشبختانه این تنها راه سازماندهی محاسبات نمی باشد.

یکی از مهمترین و در عین حال قدیمی ترین نمونه های مربوط به ساخت یک برنامه AI سیستم تولید می باشد.

در سیستم تولیدی برنامه شامل یک سری قوانین می شود. در منطق اطلاعات این قوانین به گونه ای تنظیم می شود که بوسیله الگوی اطلاعات در یک نوع مسئله داده شده قابل تشخیص باشد.

قوانین تولید می تواند به هر گونه که پاسخگوی آ" موقعیت خاص باشد برنامه ریزی شود. بدین طریق یک سیستم تولسد می تواند ایجاد کننده انعطاف پذیری و ارتباط لازم برای رفتار هوشمند باشد.

بنـــــابراین AI از یک تعداد متفاوتی ساختارهای کنترلی استفاده می کند که بسیاری از انها مرتبط با سیستم های تولید می باشند و همه آنها تابع الگو می باشند . کنترل الگویی موجب می شود که اطلاعات با توجه به نیاز به خصوصیات یک نوع مسئله خاص به کار گرفته شود. الگوی الگوریتم های انطباقی مثل به صورت واحد در آوردن باعث می شود که بتواند تشخیص دهد که چه موقع خصوصیات یک مسئله منطبق با یک برنامه اطلاعاتی است که بر این اساس اطلاعات لازم برای کاربرد در مسئله را انتخاب می کند.

بنابراین حائز اهمیت می باشد که یک زبان AI بتواند آن را مستقیما ایجاد کند و یا توسعه الگوی کنترل را ساده سازد.

در PROLOG یکی کردن و جستجوی الگوریتم ها در درون خود زمان ساخته می شوند و قلب و اساس PROLOG را تشکیل می دهند .

با استفاده از این یکی کردن الگوریتم ها به سادگی می توان هر نوع الگوی ساختاری کنترلی را ایجاد کرد .

LISP مستقیماً الگوی انطبقی ایجاد نمی کند اما محاسبات سمبولیک آن موجب گسترش ساده مربوط به زبان ساده ساختار الگوی منطق شونده و توصیف کننده اولیه ساختار می شود.

یکی از مزایای این نظریه این است که الگوی تطبیق و کنترل ساختارهای همراه با آن ممکن است به سادگی برای تطبیق با نیازهای یک مسئله بخصوص خود را منطبق سازد.

اغلـــــب نظـــریات فعلی در ارتباط با هوش مصنوعی همانند شبکه های عصبی عوامل تنظیم کننده و دیگر فرم های محاسبات ضروری ممکن است اجتناب از عملیات بر روی ساختارهای سمبولیک باشد.

ولی آنها نیاز به یک کنترل انعطاف پذیر را نفی نمی کنند. شبکه های عصبی می بایستی توانایی حرکتی شکل گیری خودشان را داشته باشند . عوامل متکی به پیام هستند که از بین ماحوبهای مختلف می گذرد.

الگوریتم های ژنتیکی نیاز به ایجاد واحد های شمارش به عنوان جمعیت کاندید شده حل مسئله دارند. توانایی زبان های AI برای ایجاد مشخصه ترکیب ساده طبقه بندی اتوماتیک حافظه امکان اطلاع رسانی ساده ایجاد متغیرها و روش های پویا و شکل های قوی ایجاد برنامه مثل یک برنامه شیء گرا موجب خواهد

شد که آنها را به سمت استفاده گسترده در کاربرد این ابزارهای جدیدتر AI سوق دهد.

پشتیبانی از روش های برنامه نویسی جستجویی.

مسائلی که AI به آن مرتبط می باشد همیشه پاسخگوی یک چنین نظریه های مهندسی نرم افزار استاندارد که شامل طراحی کامل و پردازش موفقیت آمیز و توسعه برنامه از خصوصیات و ویژگیهای دقیق است نمی تواند باشد. به دلیل طبیعت و ذات و نوع بخصوص AI به ندرت این احتمال به وجود می آید که بتوان ویژگیهای درست و کاملی از شکل نهایی یک برنامه AI قبل از ساخت حداقل یک proto type بدست آورد. اغلب موارد شناخت مسئله برنامه مربوط می شود به حل موارد درگیر مسئله از طریق توسعه برنامه . دلایل آن عبارت است از :

1 – بیشتر مسائل AI اصولا مشخصه های ضعیفی دارند.

به دلیل اینکه پیچیدگی زیادی برای پشتیبانی از سطح اطلاعات لازم می باشد به ندرت احتمال مشاهده یک مسئله و تشخیص کامل بودن نظریه دقیق که باید

در جایگاه خودش باشد وجود دارد.

بهترین ساختارهای سطح نشانه ای که در یک مسئله مورد استفاده قرار گیرند به ندرت در مشخصه های سطح دانش قرار می گیرند. این نوع پیچیدگی و نامفهومی خود را به روش های معمول مربوط به نرم افزارهای مهندسی مرتبط نمی دانند چون که در این نوع برنامه ها لازمه اش این است که مشخصه های مربوط به توسعه به خصوص مسئله قبل از اینکه مرحله کدبندی آغاز شود شکل می گیرد.

یک عملکرد منطقی خود ذاتا برای مشخصه ها و خصوصیات معمولش بسیار مشکل تر از عملکرد نوعی طبقه بندی لیست یا ایجاد یک فایل سیستم است .

حقیقتا این به چه معنی است؟

به عنوان مثال برای طراحی یک مدار یا بهبود یک بیماری این به چه معنی است؟ چگونه یک انسان ماهر و متخصص این عملیات ها را شکل می دهد؟ سطح رضایت بخش ایجاد یک محدوده مسئله داده شده چه چیزی است؟ چه نوع دانش و اطلاعاتی لازم می باشد؟ چه مشکلاتی ممکن است به دلایل نبود و یا غیر واقعی بودن اطلاعات پیش بیاید؟ به دلیل جوابهای به این قبیل سؤالات

و دیگر سؤالات که در یک دوره کلی مطرح می شود و بسیار تخصصی می باشند و هر وقت این طور باشد ساختار آن نیز عمیق تر و پیچیده تر می شود به همین نسبت حل آن نیز به دقت بیشتری نیاز دارد.

2 – نظریاتی که برای حل مسائل به آن پراخته می شود در محدوده بخصوصی قرار می گیرند.

گر چه چهار چوب های کلی برای حل مسائل AI وجود دارد به عنوان مثال سیستم تولید جستجو در زبان دامنه و محدوده هر مسئله نیازمند روش های خاص خود می باشد.

بنابراین راه حل موفقیت آمیز مسئله به ندرت به طور کامل برای محدودیتهای جدید عمومیت و کاربرد دارد هر کاربرد تا حدودی یک نوع مسئله جدید می باشد .

3- ساختارها و اشکال بیان AIبه طور پیوسته باید توسعه و تجدید شود

توسعه AI یک پروسه تحقیقی مداوم است . توسعه سیستم های AI کاربردی در بسیاری از روشها بسط و توسعهٔ این پروسه ها می باشند . گرچه تجربه عمدتاً به کاربرد زبان کمک می کند ولی عموماً هیچ جایگزینی برای کاربرد یک ایده و

اینکه چگونه عمل می کند وجود ندارد .

به همین دلیل AI اصولاً به صورت جستجوی است . برنامه اغلب به صورت ماشینی است که از طریق آن ما می توانیم دامن مسئله را کشف کنیم و روش های حل مسئله را کشف کنیم در حقیقت ابزاری است که با آن به شناخت مسئله نائل می شویم .

چالش در برنامه نویسی AI ، پشتیبانی برنامه ریزی کشفی است . در بین خصوصیاتی که یک زبان برنامه نویسی باید ایجاد کند موارد ذیل وجود دارد :

1- ***Modularity***

2- ***قابلیت گسترش***

3- ساختارهای سطح بالای مفید

4- پشتیبانی از Prototype سازی اولیه

5- قابل خواندن بودن برنامه

6- مترجم ها

7- پشتیبانی نرم افزاری برای برنامه نویسی جستجویی

ما این عناوین را در پاراگراف های زیر مورد بحث و بررسی قرار خواهیم داد :

1-قابلیت **Modularity** کدها

حائز اهمیت است که یک زبان برای برنامه نویسی کشفی از یک سری تعاریف متوالی مربوط به کدها پیروی کند این بیانگر این است که مسائل می بایستی شامل قسمت های کوچک و مطلوب باشد نه بدنه های پیچیده که بندی شده ارتباط متقابل بین محتوی برنامه باید محدود باشد و به خوبی نیز توصیف شده باشند.

این شامل پرهیز از تأثیرات جانبی و متغیرهای جهانی (global) و اطمینان از رفتار هر Module واحد در شناخت برنامه باشد که بتواند به خوبی قابل تشخیص باشد.

برنامه های LISP به صورت مجموعهٔ انتخابی از عملکردهای واحد می باشند در یک برنامه LISP که به صورت مطلوب نوشته شده باشد هر عملکرد کوچک می باشد که یک کارکرد خوب و واحد را شکل می دهند.

بنابراین اغلب جایگزینی و اصلاح علت های هر کمبودی، ساده می باشد. روش های اندازه گیری متغیر LISP و پارامترهای مربوط به آن اغلب برای کاهش تأثیرات عملکردی به کار گرفته می شوند. متغیرهای جهانی،گر چه به وسیله زبان پشتیبانی می شوند ولی استفاده در کدهای متناسب LISP نهی

شده اند.

علاوه بر این LISP دسته بندی شی گرا را از طریق سیستم شیء LISP به صورت CLOS پشتیبانی می شود.

در PROLOG واحد اصلی برنامه روش و قانون است، قوانین PROLOG همانند عملکردهای LISP کوچک و ویژه هستند.

به دلیل اینکه محدوده و قیاس متغیرها در PROLOG اغلب محدود به یک شیوه و قانون شده اند، و زبان اجازه تغییرات جهانی را نمی دهد. توصیف کردن اصولاً ساده می باشد.

LISP و PROLOG شامل مشخصه های سهل و آسانی می باشند که هنگامی که با یک ساختار برنامه مشخص ترکیب شوند، موجب آسان شدن پرداخت آن می گردند.

2-قابلیت گسترش

اصولاً برنامه نویسی جستجویی در قالب یک پروسه دارای ساختار سطح بالای برنامه ای است که به گونه کد توسعه یافته است. یک روش مهم برای انجام این پروسه در قالب سیستماتیک و با ساختار مناسب،توسعهٔ یک زبان نهفته در آن

است.

اغلب امکان توصیف شکل نهایی یک برنامه AI وجود ندارد،اما امکـان تـشخیص ساختارهای سطح بالا و مفید برای کشف و بررسی دامنه مسئله وجود دارد. ایـن ساختارها می توا نـد شـامل الگوهـای مناسـب ، کنتـرل کننـده هـای جـستجو وعملکردهای توصیف یک زبان توصیفی باشد.

اصـــولاً ایـــن نظـــریه که می گوید اگـر شـما سـاختار نهـایی یـک برنامـه را تشخیص ندهید می بایستی سعی کنید که ساختار زبان را توصیف کنید کـه بـه شما کمک خواهد کرد که آن ساختار را توسعه دهید.

برای پشتیبانی از این روش، یک زبان برنامه نویسی باید به صورت سهل و آسـان قابل گسترش باشد و به طـور سـاده آنهـا را توسـعه دهـد. بـه وسـیله توسـعه و گسترش که همان توانـایی توصـیف سـاختارهای زبـانی جدیـد اسـت کـه دارای حداکثر آزادی و انعطاف باشند.

LISP و PROLOG و همچنین توسعهٔ شیء گرا آنها هماننـد CLOS همگـی موجب می شوند که توصیف سادهٔ اهداف، پیش بینی ها و عملکردهـای جدیـد ، صورت پذیرد.

هنگامی که توصیف صورت پذیرفت، این ساختارهایی که کاربر ایجاد کرده دارای

رفتاری شبیه به ساختارهای اساسی زبان می باشند.

این زبانها به وسیله توسعهٔ توانایی های اصولی از ابتدا تا حل آن برنامه ریزی می شوند. دراین صورت،گفته می شود که برنامه های معمول، ساختاربندی می شوند ولی برنامه های AI رشد و توسعه می یابند.

این مورد با تشخیص سریع مقایسه می شود که در این مورد زبانهای معمولی مابین خصوصیات ساختاری و برنامه های توسعه یافته، کاربردی واقع می شوند.

در یک برنامه ما ممکن است عملکردهای جدیدی را تعریف کنیم اما ساختار آنها بسیار محدودتر از ساختارهای از قبل ساخته شده است. این موجب محدودیت انعطاف پذیری و استفاده از این توسعه و گسترش ها می شود.

LISP و PROLOG همچنین موجب ساده شدن نوشتن توصیف متغیرهای ویژه یک زبان خاص می شوند. در LISP برنامه ها و اطلاعات به گونه ساختاری لیست می شوند. این باعث سادگی نوشتن برنامه ای می شود که از کد LISP به عنوان داده (Data) استفاده می کنند که در این صورت باعث ساده تر شدن توسعه، تصویفی می شوند.

بسیاری از زبانهایی که از نظر سابقه و همچنین اقتصادی در نوع زبانهای AI حائز اهمیت می باشند، مثل PLANNER و ROSIE و KEE و OPS بر

اساس توانائی های LISP ساخته می شوند.

PROLOG این توانائی ها را در قالب تعدادی "meta – predicates" که قابل پیش بینی برای ترکیب با دیگر مشخصه های PROLOG باشند. ایجاد می کند که در این صورت باعث ساده شدن نوشتاری آن می گردد.

همراه با LISP تعدادی زبانهای سطح بالا AI بر اساس PROLOG ساخته شده اند که از این روش استفاده می کنند.

3-وجود ساختارهای مفید سطح بالا

برنا مه نویسی جستجویی به کمک یک ساختار قوی سطح بالا در زبان به وجود می آید،این ساختارهای قوی و کلی به برنامه نویس اجازه توسعه سریع ساختارهای ویژه برای بیان اطلاعات توصیفی و کنترل برنامه را می دهند.

در LISP اینها شامل اصول اساسی نوع اطلاعاتت می شود که موجب ایجاد ساختارهای پیچیده اطلاعاتی و عملکردهای قوی برای توصیف عملیات بر روی آنها می شود. به دلیل اینکه LISP قابل گسترش می باشد و برای چندین دهه است که مورد استفاده قرار می گیرد. مهمترین و قوی ترین عملکردهای توصیفی LISP همان خصوصیات استاندارد زبانی آنها می باشد. ویژگیهای معمول LISP

شامل جیدها عملکرد برای ایجاد ساختارهای اطلاعاتی، ساخت تداخلگرها و قابلیت Edit کردن ساختارهای LISP می شوند.

PROLOG به عنوان یک زبان مقایسه ای کوچک مطرح است که بخشی از آن به دلیل نو بودن و بخش دیـــــگر به دلیل عدم سادگی و کامل بودن آن است با این حال PROLOG به کاربرها اجازۀ ایجاد پیشگوئی های به خصوص را می دهد و مفیدترین اینها راه خود را برای استاندارد شدن باز کرده اند.

4-پشتیبانی برای ساخت Prototype اولیه

یکی از روش های برنامه نویسی جستجویی و مهم، Prototype سازی اولیه می باشد. در اینجا برنامه نویس یک راه حل سریع برای مسئله پیدا می کند و از آن برای جستجو فضای مسئله استفاده می کند. وقتی که مسئله مورد بررسی قرار گرفت و روش حل آن مشخص شد، Prototype کنار گذاشته می شود و یک برنامه نهایی که تأکید آن بر روی صحت و مؤثر بودن می باشد، ساخته می شود. گر چه مشکل است که چیزهایی را که زیاد مورد استفاده قرار می گیرند تا برای ساخت یک برنامه کامپیوتری به کار روند، کنار گذاشت، ولی انجام چنین کاری باعث صرفه جوئی در زمان و بهبود کیفیت نهائی کار می شود.

ساختارهای ایجاد شده به وسیله زبانهای AI عمدتاً باعث افزایش سرعت توسعهٔ Prototypeها می شوند.

5-قابلیت خواندن برنامه و مستندسازی آن

به دلیل اینکه اغلب برنامه های AI به طور گسترده ای از طریق خودشان توصیف می شوند ولی این نکته حائز اهمیت است که کد بتواند قابل خواندن و قابل مستندسازی باشد. در عین حالیکه هیچ نوع جایگزینی برای محتوی زبانهای معمول در کد وجود ندارد، ولی با این حال زبان های AI همراه با Moduleهای با ساختار سطح بالا باعث ساده شدن این عمل می شوند.

6-مفسرها

بیشتر زبانهای AI قبلاً ترجمه شده هستند نه اینکه در طول برنامه بخواهند ترجمه شوند. این بدان معنی است که برنامه نویس لازم نیست به مدت طولانی هر زمان که کد تغییر کرد برنامه را تعریف مجدد کند.

با توجه به مسائل عملکردی در ترجمه کد، زبانهای AI مدرن به Module های ویژه اجازه تعریف مجدد برای یک موقعیت متوسط را می دهند که از این طریق برنامه های سطح بالاتر بهتر تعریف می شوند. علاوه بر این بسیاری از کاربردها

به برنا مه ها اجازه تکمیل شدن نهایی برنامه ها را می دهند.

7-محیطهای توسعه

زبانهای جدید AI در برگیرنده محیطهای برنامه ریزی مـی شـوند کـه ابزارهـای ایجاد کلی و یا بخشـــی از برنامه را فراهم می کنند.بسیاری از کاربردهای زبانی شامل ویرایش هوشمند می شـــوند که اشتباهات را به عنوان یک کـد نوشـتاری در نظر می گیرد. به دلیل پیچیدگی برنامه های AI و مشکل بودن پـیش بینـی عملکرد هر سیستم تولید، اهمیت این پشتیبانی های سهل نمی تواند قابل تصور باشد.

Dynamic Binding and constraint propagation

زبانهای معمول نیاز به این دارند که بیشتر برنامه های مرتبط با آن در یک مدت زمان خاص تشخیص داده شوند.

شامل اتصال دادن متغیرها به محیط حافظه و انتقال روش های به نـام هایـشان می باشد. با این حال بسیاری از روش های برنامه نویسی پیـشرفته مثـل، برنامـه نویسی های شیء گرا نیاز به این اتصال ها برای تشخیص دینامیکی دارند.

برنامه های Prolog و LISP پشتیبان قیدگـذاری دینـامیکی هـستند. از یـک

نقطه نظر AI یکی از مهمترین منافع قیدگذاری دینـامیکی پـشتیبانی از برنامـه نویسی ساختاری است. اغلب مسائل مربوط به یک برنامه AI نیـــاز به ایـــــن دارد که ارزش های مشخصه های خاص ناشناخته باقی مـی مانـد تـا زمـانی کـه اطلاعات لازم جمع آوری شوند.

این اطلاعات ممکن است به گونه یک سری از ساختارها بر اساس ارزش ها باشد که یک متـــغیر از آن انتظار دارد. هم چنانکه ساختارها جمع شوند یک سـری از احتمالات کاهش می یابد و در نهایت به یک راه حل منتهی می شود که تمـامی ساختارها را تحت پوشش مطلوب قرار می دهد.

یک نمونه ساده از این نظریه ممکن است در یک سیستم تشخیص پزشکی دیده شود که اطلاعات دربارۀ نوع بیماری مریض جمع آوری می شـود تـا زمـانی کـه اطلاعات مربوطه محدود به نوع خاصی از بیمار شوند زبان برنامه نویسی مقایـسه ای این روش از نوع متغیر قیدگذاری اولیه یا توانایی حصول یک متغیـر نامرکـب می باشد در حالیکه آن را در کد برنامه جمع می کند.

LISP و PROLOG به متغیرها اجازۀ ترجمه وافزایش غیرمرکب را می دهند، در حالیکه توصـــیف ارتبـــاطات و وابستگی هـای بـین ایـن متغیرهـا و دیگـر واحدهای برنامه را انجام می دهد. این موجب کاربرد آسان و طبیعی نوع قید می

شود.

6. تعاریف مشخص و واضح

لازم است که زبانهای AI همراه با زبانهای دیگر برنامه نویسی برای توسعه گسترده کامل و در عین حال منطقی سیستم، به کار گرفته شوند.

متأسفانه زبانهای برنامه نویسی معمول مثل Fortran و پاسکال دارای تعاریف مشکل و پیچیده ای هستند این موارد می تواند ناشی از واقعیت خود زبان باشد که اصولاً دارای خصوصیات ساختاری سطح بالایی در کامپیوتر دارند و در خودشان سیستم های فیزیکی و پیچیده ای دارند. به دلیل اینکه زبانهای AI دارای اساس و پایه ریاضی هستند مثل PROLOG و LISP ،آنها می بایستی معانی ساده تری باشند که دارای قدرت و ظرافت نهفته در ریاضی باشند.

این موجب می شود که این زبانها عمدتاً برای تحقیقات در محدودهٔ به کارگیری دانش ابزارهای زبان، ایجاد برن امه درست،و اتوماتیک کردن تأثیر گذاری کد، مفید واقع شوند.

همچنین باید توجه داشت که گر چه عملکرد بسیاری از برنامه های AI کاملاً پیچیده می باشد ولی کدی که دارای این عملکرد است باید ساده و

مشخص باشد.

بلوک های بزرگ مرکب و پیچیده با کد مشخص دارای AI مناسب نمـی باشـند یک زبان خوب توصیف شده،یک ابزار مهم برای دریافت این اهداف می باشد.

خلاصه ای دربارۀ LISP و PROLOG

به وسیله برآورده کردن نیازهای گفته شده، LISP و PROLOG هـر دو دارای زبانهای برنامه نویسی غنی و کاملی هستند وقتی که این زبانها را فرا می گیـریم، دانشجو در ذهن و فکر دربارۀ روشهایی که آنها به وسـیله ویژگیهـای خـاص هـر زبان پشتیبانی می کنند، نیازها را نگه داری می کنند.

PROLOG

PROLOG یکی از بهترین نمونه و مثال یک زبان برنامه نویسی منطقی اسـت. یک برنامه منطقی دارای یک سری ویژگیهای قانون و منطق است . PROLOG از محاسبۀ اولیه استفاده می کند. در حقیقـت خـود ایـن نـام از برنامـه نویـسی PRO در LOGIC می آید یک مفسر برنامه را بر اساس یک منطق می نویـسد. ایده استفاده توصیفی محاسبه اولیه برای بیان خـصوصیات حـل مـسئله یکـی از محوریت های مشارکتPROLOG می باشد که برای علم کامپیوتر به طور کلی

و به طور اخص برای زبان برنامه نویسی هوشمند مورد استفاده قرار مـی گیرنـد. نفع اسفتاده از محاسبه اولیه برای برنامه نویسی شامل یک ساختار ظریف و ساده و قابل معنی می شود.

به دلیل همین خصوصیات است که PROLOG به عنوان یک محـرک اصـلی و مفید برای تحقیقاتی مثل موارد برنامه نویسی آزمایشی به عنوان یک کد، متغیـر کردن برنامه و طراحی ویـــژگیهـای زبان سطح بالا، مطرح است. PROLOG و دیگر زبانهای منطقی یک سبک برنامه نویسی مشخصی را دنبال می کنند که در آنها برنامه ها به صورت دستورات پشت سرهم و متوالی برای ایجاد یک الگوریتم، نوشته می شوند. این نوع برنامه اصولاً به کامپیوتر مـی گویـد کـه «چـه چیـزی درست است» و «چه چیزی باید صورت گیرد» و این به برنامه نویس اجـازه مـی دهد که بر روی حل مسئله به صورت یـک سـری خـصوصیات از یـک محـدوده تأکید کند تـا اینکـه بخواهـد بـه جزئیـات نوشـتاری سـطح پـائین سـاختارهای الگوریتمی برای بعد بپردازد.

اولین برنامه PROLOG در مارسی فرانسه در اوایل 1970 به عنـوان بخـشی از زبان معمول یک پروژه نوشته شد. تئوری نهفته در پشت ایـن زبـان در کارهـای کوالسکی،هیـز و دیگـران آورده شـده اسـت. عمـدۀ توسـعۀ PROLOG بـین

سالهای 1975 تا 1979 در بخش هوش مصنوعی دانشگاه ادینبورگ صورت گرفت.

در آنجا یک گروه مسئولیت کاربرد اولین PROLOG را به عهده داشتند که آقای David H.D مسئول آن بود. این گروه اولین PROLOG را ساخت که می توانست محاسبات کلی را انجام دهد. این محصول بر اساس سیستم DEC-10 ساخته شده بود و می توانست در مدهای توصیفی و مقایسه ای کارآئی داشته باشد.

مزیت این زبان به وسیله پروژه هایی که برای ارزیابی و گسترش قدرت بیان برنامه های منطقی نوشته شده اند، اثبات شده است.

بحث دربارۀ یک چنین کاربردهایی می تواند در سمینار و گردهمائی های مربوط به زبان برنامه نویسی هوش مصنوعی در سطح بین المللی مطرح شود.

LISP

LISP اولین بار به وسیله JACK MCCARTHY در اواخر دهه 1950 مطرح شد این زبان به عنوان یک مدل پیوسته محاسباتی بر اساس تئوری عملکرد مجدد،معرفی شد.

در مقالات اولیهٔ مک کارتی (1960) اهداف خود را مشخص می کند: ایجاد یک زبان سمبولیک تا یک زبان محاسباتی. ایجاد زبانی که بتوان از آن به عنوان یک مدل محاسباتی بر اساس تئوری عملکرد مجدد استفاده کرد و از آن بتوان برای تعریف دقیق یک ساختار و تعریف زبانی استفاده کرد.

گر چه LISP یکی از قدیمی ترین زبانهای محاسباتی است که هنوز فعال است، ولی دقت کافی در برنامه نویسی و طراحی توسعه باعث شده که این یک زبان برنامه نویسی فعال باقی بما ند.

در حقیقت این مدل برنامه نویسی طوری مؤثر بوده است، که تعدادی از دیگر زبانها بر اساس عملکرد برنامه نویسی آن واقع شده اند مثل FP ، ML و SCHEME .

این لیست اساس برنامه ها و ساختارهای اطلاعاتی در LISP است، LISP خلاصه شده نام پروسه LIS است. این برنامه یک سری لیست های عملکردی درون ساختاری دارد.

LISP به برنامه نویس قدرت کامل برای اتصال به ساختارهای اطلاعاتی را می دهد.

اصولاً LISP یک زبان کامل است که دارای عملکردها ولیست های لازمه برای

توصیف عملکردهای جدید، تشخیص تناسب و ارزیابی معانی می باشد.

تنها هدف کنترل برنامه بازگشت و شرایط منحصر به فرد است. عملکردهای کامل تر هنگامی که لا زم باشد در قالب این اصول تعریف می شوند. در طی زمان بهترین عملکردها به عنوان بخشی از زبان می شوند. پروسه توسعهٔ زبان به وسیلهٔ اضافه کردن عملکردهای جدید موجب توسعه محورهای زیادی از LISP می شوند که اغلب شامل صدها عملکرد بخصوص برای ایجاد اطلاعات کنترل برنامه، خروجی و ورودی، Edit کردن عملکردهای LISP می شوند.

این ارتباطات محرکه ای هستند که به وسیله LISP از یک مدل ساده و ظریف به یک مدل قوی و غنی و عملکردی برای ساخت سیستم های نرم افزاری بزرگ، تبدیل می شود.

یکی ازمهم ترین برنامه های مرتبط با LISP برنامه SCHEME می باشد که یک تفکر دوباره دربارهٔ زبان در آن وجود دارد که به وسیله توسعه AI وبرای آموزش اصول مفاهیم علم کامپیوتر مورد استفاده قرار می گیرند.

7. *برنامه نویسی شیء گرا*

برخلاف برنامه LISP و PROLOG ،برنامه شیء گرا ریشه در مهندسی نرم

افزار دارد. اولین بار در سال 1970 توسعه یافته که به وسیله Alan Kay این تحقیقات صورت گرفته است.

ساخت ایده ها از محرک، که زبان نروژی تظاهر می کند در سال 1960 و مقاله Symour در استفاده از LOGO برای آموزش کودکان، صورت پذیرفته است.

استفاده از Dyna book برای اولین بار به عنوان یک کامپیوتر، که افرادی به غیر متخصصان علم کامپیوتر با آن سروکار داشتند.

به دلیل اینکه کاربر افراد معمولی بودند سیستم عملکرد و کاربرد نرم افزار نباید تکنیکی می بود و به سادگی قابل تشخیص بود. راه حل آنها برای این مسئله یک مداخلهٔ گرافیکی است با استفاده از منوها و آیکون های گرافیکی و اشاره گرها، یک موس یا یک سری برنامه ها برای ادیت کردن، داده ها می باشد.

دخالت کاربر در طراحی یک notebook متأثر از طراحی کاربرها برای یک سری کامپیوترهای تخصصی مثل سیستم های به کارگیری کامپیوتر شخصی مثل مکینتاش، ماکروسافت و محل های مربوط به ویندوز می باشد.

در یک برنامه small talk ،همه چیز در قالب هدف و یک ساختار قابل محاسبه مرک و قراردادی مطرح می شود. اهداف نه فقط شامل انواع اطلاعات برای محاسبه بلکه شامل انواع روشهای لازم برای محاسبه حالت و وضعیت هدف

نیز می شوند.

ارزشهای یک هدف به صورت کلاس ها بیان می شود. اهداف ممکن است اهداف طبقه بندی شده که توصیف کنندۀ تمامی مواد یک نمونه باشد و بیانگر نوع ذات و توصیف تما می موارد یا مواردی که بیانگر یک عضو واحد هستند را شامل شود.

وقتی مواردی از یک نوع اطلاعات به وسیله اهداف توصیف می شود این موارد ذاتاً دارای نوع توصیف و روشهای توصیفی از عملگرهایشان می باشند،برای شکل دادن یک عملیات بر روی یک هدف، یک پیام به سمت هدف فرستاده شده که حاصل روش مناسبی می باشد. به عنوان مثال، اضافه کردن 3 و 4 پیام 4+ به سمت شیء 3 فرستاده می شود و 3 پاسخ می دهد می شود 7 .

به وسیله ایجاد انواع ترکیب اطلاعات و عمل بر روی آنها در یک عمل واحد مربوط به هدف، small talk از کد Modular (پیمانه ای) توسعه و نوع کاربرد برای عناصر اطلاعات و کد مربوط به تکثیر آنها، پشتیبانی می کند.

به دلیل اینکه اهداف small talk در قالب یک کلاس شبکه ای همراه با اهداف کاملاً ویژه که بخشی از تمامی روش های کاملاً کلی است ، بسیار ساده است که یک ساختار جدید برنامه ای توصیف کنیم که عملاً با اهداف موجود در

برنامه همراه باشد. بنابراین یک برنامه اصولاً می تواند قدرت کامل یک سیستم باشد که شامل گرافیک،بازنگری و ارتباط است.

علاوه بر این روش های توسعه نرم افزاری مثل ارائه اطلاعات و زبانهای نهفته، فشار بر اپراتور و استفاده از کدها از طریق یک گروه اصلی و زبانهای نهفته در قالب یک مدل رایج پشتیبانی می شوند.

زبانهای شیءگرا همراه با بسیاری از خصوصیات مندرج در یک کلاس اطلاعاتی، شامل کلاس اصلی و توانایی پاسخ در ساختار اطلاعات می شود به همین دلایل زبانهای شیءگرا در برنامه نویسی AI استفاده می شوند.

محیطهای هیبرید

نیاز به برنامه نویسی اطلاعاتی موجب توسعهٔ تعدادی برنامه نویسی و تکنیک های زبان، شامل سیستم های تولید،قوانین و کلاس شیءگرا می شود.

یک سیستم هیبرید بیانگر نمونه های چند منظوره در قالب یک محیط برنامه نویسی خاص می باشد. گر چه محیطهای هیبرید متفاوت می باشد. ولی عموماً شامل خصوصیات ذیل می شوند.

1-نمایش شیء گرا از محدوده اشیاء

یک چنین سیستم هایی ذاتاً ویژگیهای کلاس را پشتیبانی می کنند و اغلب

شامل یک مکانیسم انتقال پیام برای عکس العمل هدف می باشند.

2-قوانین نمایش اطلاعات neuristic

گر چه چهارچوب اهداف به معنی توصیف طبقه بندی اهداف، می باشند. قوانین به عنوان عمدهٔ نظر توصیف مسائل اطلاعاتی می باشند.

ساختار ..... if .... then مناسب شیوهٔ تخصص انسانی است که بیانگر پروسهٔ تصمیم گیر است. قوانین دریافت اطلاعاتی از اهداف را دارند که با استفاده از یک زبان که مستقیماً در چارچوب یک هدف می خواند و می نویسد و یا به وسیلهٔ استفاده از پیام که مستقیماً وارد هدف می شود صورت می پذیرد.

3-پشتیبانی از انواع روش های جستجو

بیشتر سیستم های پشتیبانی از جستجوی اولیه و انتهایی حمایت می کنند عموماً بیان یک هدف در قالب جستجویی ،تغییر به سمت عقب می باشد . علاوه بر این یک واقعیت تازه دربارهٔ حافظه کارکرد ممکن است ایجاد علت های اولیه از قوانین کند که به وسیله این واقعیت جدید پشتیبانی می شوند.

4-توصیف دامنهٔ کاربرد عملکرد متقابل و تأثیرات جانبی

یک demon فرآیندی است که به وسیله عملکردهای جانبی بعضی از

اعمال مشخص می شود. یک نمونه از استفاده demon کنترل در یک سیستم زمانی است که بیانگر دوره ای در مانیتور یک چاپگر و یا دیگر وسایل می باشد. demon به وسیله یک زمان مشخص می شود. محیطهای AI این ایده را توسعه می دهند و باعث ایجاد demon می شوند که هنگامی که هدف تولید یا توصیف شود به کار می آیند.

چنین demonهایی برای به زمان نگه داشتن یک نشانگر در پاسخ به تغییر مقدار مورد استفاده قرار می گیرند. Demon های مهم و مطرح اصولاً دارای مقادیری متغیر می باشد که هنگامی که ارزش متغیر تغییر کند demon خوانده شده و وقتی که این اتفاق افتاد demon ایجاد و خلق می شود که این وقتی اتفاق می افتد که یک مقدار خلق شده باشد و ارزش ها در قالب گرافیکی فعال می شوند که این فعالیت می تواند متغیر باشد.

5-تداخلگرهای گرافیکی

اینها شامل یک طیفی از امکاناتی می باشند که اجازه تداوم و دنبال کردن موارد را می دهند. به عنوان مثال نشانگرهای گرافیکی می توانند ساختار قوانین یک اصل اطلاعاتی را به صورت یک درخت توصیف دهند. یکی از مهمترین خصوصیات محیطهای هیبرید،توانائی اتصال با استفاده از demon می باشد که

به صورت یک نشانگر گرافیکی متصل به شیء و هدف می باشد. که این موجب عملکرد گرافیکی برای بیان زمان واقعی نشانگر می باشد که در حقیقت بیشتر محیط ها دارای یک پشتیبانی سطح بالای از داده های گرافیکی می باشند.

6-اجتناب از زبانهای زیرین

روشهایی که در قالب یک زمان خاص یا پاسخگو می باشند به وسیله محیط و یا اغلب اوقات LISP و PROLOG یا حتی و یا پاسکال توصیف شده اند که این موجب توصیف طیفی فرآیند اطلات و هم چنین یک برنامه اطلاعاتی که طیف وسیعی از زبانهای که شکل دهنده هندسی، جهت ها و سنسورها و یا دیگر عملکردهایی که به صورت بهتری در قالب روشهای الگوریتمی به کار گرفته می شود را شامل می شود.

7-توانائی ترجمه اطلاعات جهت اجرای سریعتر یا تحویل روی یک ماشین کوچکتر

وقتی که برنامه شیء گرا کامل شد. یک محیط کامل و توسعه یافته اغلب ، بلندی است که به تدریج افول می کند و پائین می آید بیشتر محیط های مدرن AI اجازه کاربرد سریعتر و ساده تر را که اغلب کوچکتر و ارزانتر است را در یک

ماشین ترجمه ایجاد می کنند.

8. یک نمونه هیبرید

بسیاری از نمونه های مطلوب اصولاً از طریق اشیا، ارتباطات و کنش و واکنش متقابل بین آنها واقع یم شود در شکل زیر یک نمونه اتصال به وسیله باطری همراه با یک سوئیچ برای یک لامپ (شکل 364) در نظر گرفته شده است.

لامپ، باطری و سوئیچ ممکن است هر کدام به وسیله کلاسها بیان شوند که بیانگر ویژگیهای باطری، سوئیچ و لامپ باشد . مشخصه های الکترونیکی شکل 2 ممکن است به عنوان موارد بخصوصی از این کلاس های کلی بیان می شوند. توجه داشته باشید که نمونه ها دارای مقادیر نمونه ای مربوط به کلاس خاص شیء مربوط به خود می شوند به عنوان مثال اگر سوئیچ 1 در حالت خاموش قرار گیرد. قسمت کنترل که مربوط می شود به لامپ 1 تحت تأثیر قرار خواهد گرفت که این موارد در شکل زیر نشان داده شده اند.

یک قانون ممکن است در اینجا به وجود بیاید که :
اگر نور وارد AND نشود، سوئیچ AND را بسته و باطری درست است بنابراین باید به قسمتی که ممکن است آسیب دیده باشد مراجعه کرد.

در نمایش هیبرید قوانین دارای ویژگیهایی هستند که بیشتر بیانگر مقدار اهمیت

کلاسها و اشیاء می باشند.که در شکل 3 بـه آن اشـاره شـده اسـت. ایـن قـانون ممکن است به عنوان بخشی از قانون اولیه سیـستم در تـلاش بـرای بـه جریـان انداختن این مدار باشد که در جای دیگر برای راه انـدازی سـوئیچ کنتـرل بـرای حالات متغیر است.

9. انتخاب زبان کاربردی

همانگونه که هوش مصنوعی به مرحله رشد می رسـد وقابلیـت هـای خـود را در طیـف وسـیعی از مـسائل کـاربردی بـه اثبـات مـی رسـاند اعتمـاد بـه LISP و PROLOG نیز مدنظر می باشد، موارد مربوط به توسعهٔ نرم افزاری، همانند نیاز به تداخلگرها به صورت ساده وآسان همراه با یک کد منطقی تا استفاده از AI در Moduleهای کوچکتر و یا بزرگتر در برنامه ها و نیاز به ایجاد توسعه اسـتاندارد متأثر از مشتریان دولتی و یا گروهی موجب توسـعهٔ سیـستم هـای AI در انـواع زبانهای مثل C ، C^{++} , Java و Smalltalk شده است.

کـــه زبـــانهای LISP و PROLOG کـار خـود را در محـدودهٔ توسـعه و Prototype سازی سیستم های AI در صنعت و دانشگاهها دنبال می کنند.

یـــک اطـــلاعات و دانش کاربردی مربوط به ایـن زبانهـا بـه عنـوان بخـشی از

مهارت هر برنامه نویس AI می باشد. علاوه بر این، این زبانها به عنوان زمینه ای برای بسیاری از این خصوصیات می باشند که در ادامه همکاری با زبانهای برنامه نویسی جدید می باشند.

احتمالاً بهترین نمونه از این زبانها Java میباشد که متناسب با استفاده اولویت دینامیکی اش، دارای مدیریت حافظه اتوماتیک و دیگر خصوصیاتی است که در زبانهای که ترجمه شده وجود دارد به نظر می رسد که دیگر زبانهای برنامه نویسی برای رسیدن به حد مطلوب از استانداردهای این زبانها استفاده می کنند.

هم چنانکه این تکامل صورت می پذیرد و ادامه می یابد دانش مربوط به LISP و PROLOG یا Small talk و روش های برنامه نویس قادرند تنها از نظر مقدار توسعه یابند.

بنابراین، از اینکه از یکی از این زبانهای AI استفاده کنیم یا خود را در برنامه نویسی با زبانهای C^{++} و C و Java یا یکی از زبانهای رقیب پیدا کنیم راضی و قانع خواهیم بود.