هوش مصنوعی

Artificial Intelligence

نام کتاب :

هوش مصنوعی رهیاتی نوین

مولف :

راسل و نورویگ

مترجم :

رامین رهنمون آناهیتا هماوندی

فصل اول

هوش مصنوعی

Artificial Intelligence

AI: به طور رسمی در سال ۱۹۵۶ مطرح شده است.

علل مطالعه AI:

AI سعی دارد تا موجودیت‌های هوشمند را درک کند. از این رو یکی از علل مطالعه آن یادگیری بیشتر در مورد خودمان است.

جالب و مفید بودن موجودیت‌های هوشمند .

AI چیست؟

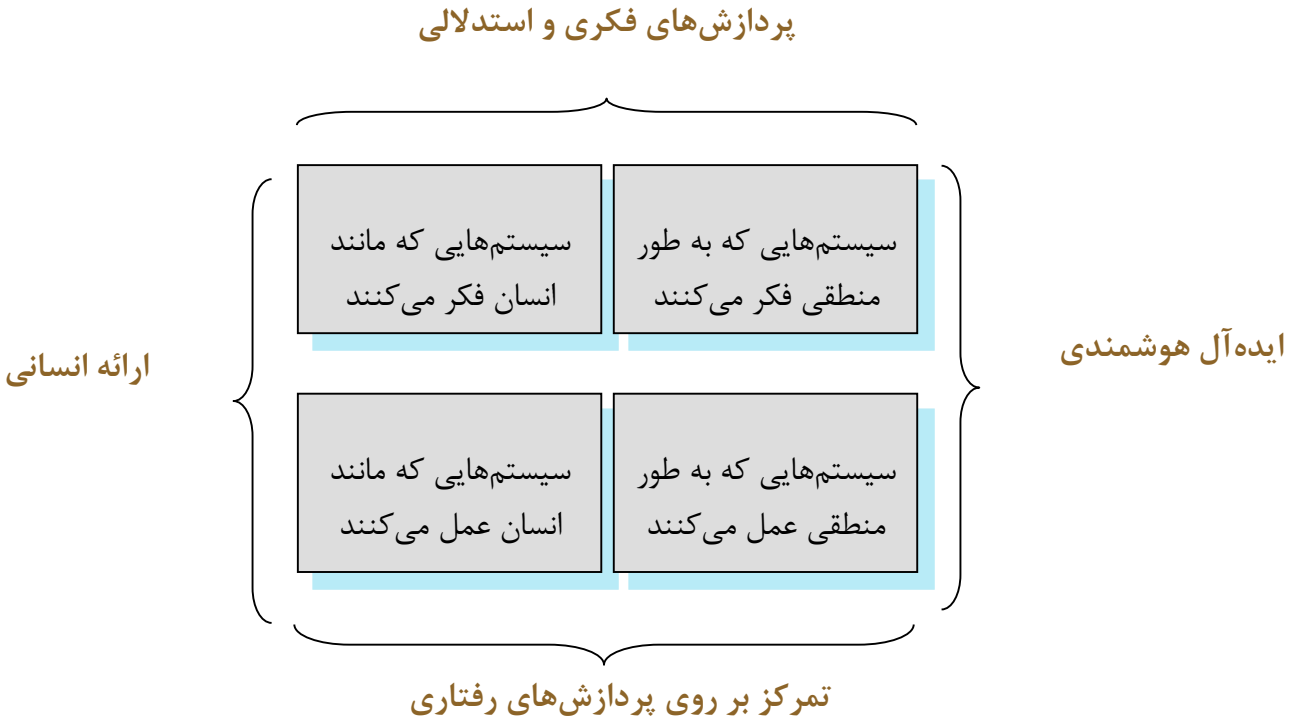
تعاریفی از AI که به چهار قسمت تقسیم شده‌اند:

پردازش فکری و استدلالی

پردازش رفتاری

ایده‌آل هوشمندی (منطقی بودن)

ارائه انسانی



۱– انسان گونه عمل کردن: رهیافت آزمون تورینگ

آزمونی از کامپیوتر به عمل آید، و آزمون گیرنده نتواند دریابد که در آن طرف انسان قرار دارد یا کامپیوتر.

برای این کار کامپیوتر باید قابلیت‌های زیر را داشته باشد:

- ❖ پردازش زبان طبیعی = محاوره
- ❖ بازنمایی دانش= ذخیره اطلاعات
- ❖ استدلال خودکار= استدلال و استخراج
- ❖ یادگیری ماشینی= کشف الگو و برون ریزی

تست تورینگ:

این آزمون از ارتباط فیزیکی مستقیم بین کامپیوتر و محقق اجتناب می‌کند.

به منظور قبول شدن در تست تورینگ کلی، کامپیوتر به موارد زیر احتیاج دارد:

- ❖ بینایی ماشین برای درک اشیاء

❖ روباتیک به منظور حرکت آنها

۲. انسانی فکر کردن–: رهیافت مدلسازی شناختی:

❖ چگونگی شناسایی عملکرد افکار انسان:

❖ ۱- درون گرایی

❖ ۲- تجارب روانشناسی

❖ علوم شناختی : مدل‌های کامپیوتر از AI و همچنین تکنیک‌های روانشناختی را گرد هم می‌آورد تا بتواند تئوری‌های دقیقی از کارکرد ذهن انسان به دست آورند.

۳. منطقی فکر کردن: قوانین رهیافت تفکر

❖ رمز «تفکر درست»: ارسطو سعی در کشف آن داشت.

❖ قیاس: از موضوعات مطرح شده توسط ارسطو می‌باشد، که الگوهایی برای ساختار توافقی ایجاد کرد که همواره نتایج صحیحی به اندازه مقدمات صحیح به دست می‌آورد.

❖ مثال: «سقراط انسان است، تمام انسان‌ها می‌میرند، پس سقراط خواهد مرد.»

دو مشکل عمده در این رسم منطق‌گرایی وجود دارد:

❖ تبدیل دانش غیر رسمی به شکل رسمی توسط اعلام، منطقی ساده نیست.

❖ تفاوت عمده‌ای بین قادر به حل مسئله بودن در اصول و انجام آن در عمل وجود دارد.

۴. منطقی عمل کردن: رهیافت عامل منطقی

❖ عامل: در اصل چیزی است که ابتدا درک می‌کند و سپس عمل می‌کند.

❖ در نگرش «قوانین تفکر» تأکید عمده بر روی استنتاج‌های صحیح بوده است.

❖ «مهارت‌های شناخت» که برای آزمون تورینگ موردنیاز است، برای انجام فعالیت‌های منطقی وجود دارند.

مزایای مطالعه AI به‌عنوان طراحی عامل منطقی:

❖ عمومی‌تر از رهیافت «قوانین تفکر»

❖ پیشرفت علمی، بسیار قانون‌پذیرتر از رهیافت‌هایی است که بر تفکر یا رفتار انسانی متکی هستند.

زیربنای هوش مصنوعی:

AI، از علوم مختلفی بهره می‌برد که از میان آنها علوم زیر مهم‌تر شناخته شده‌اند:

❖ علم فلسفه

❖ علم ریاضی

❖ علم روانشناسی

❖ علم زبان‌شناسی

❖ علم کامپیوتر

فلسفه: (۴۲۸ قبل از میلاد مسیح – تاکنون)

پایه‌های تفکر و فرهنگ غرب تشکیل شده است از: افلاطون، استادش سقراط، و شاگردش ارسطو.

قیاس: ارسطو، سیستمی غیررسمی از قیاس برای استدلال مناسب توسعه داد، امکان تولید نتایج، بر پایه فرضیات اولیه به طور مکانیکی وجود داشت.

در نظر گرفتن ذهن به‌عنوان سیستمی فیزیکی

رنه دکارت مدافع سرسخت قدرت استدلال بود؛ و همچنین طرفدار مکتب دوالیسم.

ماتریالیسم: در مقابل دوالیسم قرار دارد و معتقد است تمامی جهان مطابق قوانین فیزیکی عمل می‌کنند.

ویلهم لایبنیز:

❖ تبدیل موقعیت ماتریالیستی به نتایج منطقی

❖ ساخت ابزاری مکانیکی برای انجام عملیات منطقی

ایجاد منبع دانش:

فرانسیس بیکن، جنبش آزمون‌گرایان را آغاز کرد. و با شعار جان لاک مفهوم یافت:

«هیچ چیز قابل فهم نیست اگر ابتدا در حس نباشد.»

اصل استقرای امروزی، در حقیقت از کتاب دیوید هیوم نشأت می‌گیرد: "رسانه‌ای از طبیعت انسان"

برتراندراسل، پایه‌گذار پوزیوتیزم منطقی، ارائه‌دهندهٔ این تئوری بود که:

«قوانین عمومی توسط تکرار ارتباطات بین عناصر آنها به وجود می‌آیند.»

ارتباط بین دانش و عمل

اشیاء را با تحلیل، دسته‌بندی می‌کنیم و در اطراف آنها، کارکرد مورد نیازشان نوسان می‌نماید.

در این میان پایه سیستم‌مکاشفه‌ای GPS بنیان گذارده می‌شود.

ریاضیات (۸۰۰ C–تاکنون)

برای ارتباط فلسفه با دانش نظری، نیاز به فرمول‌سازی ریاضی در سه زمینه اصلی است:

❖ محاسبات

❖ منطق

❖ احتمالات

محاسبات:

نظریه اظهار محاسبات به عنوان الگوریتمی رسمی به خوارزمی برمی‌گردد، ریاضیدان عربی قرن نهم که نوشته‌های وی، جبر و تئوری اعداد عربی را به اروپا معرفی کرد.

منطق:

در این زمینه، دانشمندان زیادی بر چگونگی شکل‌گیری و هدایت آن، نقش داشته‌اند که به چند نفر از آنها اشاره می‌کنیم:

ارسطو: دانشمندی که بیشترین شکل‌گیری نگرش فلسفی منطق را به او نسبت می‌دهند.

جورج بول: یک زبان رسمی برای ساخت استنتاج منطقی ارائه داد.

FREGE: منطق مرتبه اول را به شکلی مطرح نمود که در بیشتر سیستم‌های نمایش دانش پایه استفاده می‌شود.

آلفرد تارسکی: تئوری چگونگی ارتباط بین اشیاء موجود در محیط منطقی، و اشیاء موجود در دنیای واقعی را ارائه نمود.

دیوید هیلبرت: ریاضیدان بزرگی بود که شهرت وی به دلیل مسائلی است که نتوانست حل کند.

راسل: قضیه کامل نبودن (incompleteness) را مطرح نمود.

تورینگ: ماشین تورینگ قادر به محاسبه هر تابع محاسبه‌پذیری است.

تئوری پیچیدگی:

انجام‌ناپذیری ، استحاله

استیون کوک و ریچارد کارپ: تئوری NP-completeness را مطرح کردند.

احتمالات:

گاردنیوی: اولین کسی بود که ایده احتمال را مطرح کرد.

پیر فرمت، پاسکال، برنولی، لاپلاس و دیگر دانشمندان بر رشد و توسعه این ایده تأثیر داشتند.

برنولی: دیدگاه «درجه باور» ذهنی را در مقایسه با نرخ نتایج عینی مطرح کرد.
بیس: قانونی برای بهنگام‌سازی احتمالات ذهنی را به وجود آورد.
نیومن و مورگنسترن: تئوری تصمیم‌گیری را آغاز کردند. و از ترکیب تئوری احتمال، و تئوری سودمندی حاصل می‌شود.

روانشناسی (۱۸۷۹– تاکنون):

هلمولتز: روشی علمی برای مطالعه بینایی انسان به کار برد؛ که این کتاب به عنوان مرجع بینایی فیزیولوژیک و حتی به‌عنوان «مهمترین رساله فیزیکی و روانشناختی بینایی انسان تا به امروز» شناخته می‌شود.
وندت: اولین آزمایشگاه روانشناسی تجربی را در دانشگاه لایپزیک راه‌اندازی کرد.
داتسون و تورن دایک: حرکت رفتارگرایی (behaviorism) را مطرح کردند.
اساس مشخصه روانشناسی شناختی(congnitive psychology)، این نگرش است که مغز دارنده و پردازش‌کننده اطلاعات است.
کریک، کتاب ماهیت بیان را منتشر کرد. و سه مرحله کلیدی را برای عامل مبتنی بر داشن معین کرد:

- ❖ محرک‌ها باید به شکل درونی تبدیل شوند.
- ❖ بازنمایی توسط پردازش‌های شناختی بازنمایی‌های داخلی جدیدی را مشتق کند.
- ❖ اینها دوباره به صورت عمل برگردند.

مهندسی کامپیوتر (۱۹۴۰- تاکنون)
برای پیشرفت هوش مصنوعی، به دو چیز احتیاج داریم:
هوش ، محصول مصنوعی
در این تقسیم‌بندی، کامپیوتر می‌تواند به عنوان محصول مصنوعی محسوب گردد.
Heath Robinson: اولین کامپیوتر مدرن عملیاتی بود که در سال ۱۹۴۰ توسط تیم آلن تورینگ به منظور کدگشایی پیام‌های آلمان‌ها ساخته شد.
Colossus: نام ماشین بعدی بود که تیوپ‌های مکنده در آن به کار برده شد.
Z-3: اولین کامپیوتر قابل برنامه‌ریزی که توسط کنراد زوس در ۱۹۴۱ اختراع شد.
اعداد با ممیز شناور و زبان Plankalkul نیز توسط زوس اختراع شدند.
ABC: اولین کامپیوتر الکترونیک در امریکا توسط جان آتاناسف و کلیفورد در دانشگاه ایالتی ایوا ساخته شد.
MARK I , II , III: توسط تیمی به رهبری هوراد ایکن در هاروارد توسعه داده شد.
ENIAC: اولین کامپیوتردیجیتال الکترونیک چند منظوره، توسط تیمی به سرپرستی ماچلی و اکرت در دانشگاه پنسیلوانیا ساخته شد.
IBM 701: اولین کامپیوتر سودآور، توسط ناتانیل روچتر در ۱۹۵۲ ساخته شد.
چارلز بابیج: طراحی ماشینی که جداول لگاریتمی را محاسبه کند.

- ❖ طراحی موتور آنالیتیکی
- ❖ طرح حافظه قابل‌آدرس‌دهی، برنامه ذخیره شده و پرش‌های شرطی

کار در زمینه AI منجر به ایده‌های بسیار متعددی شد که به علوم کامپیوتر برگشت؛ مانند:
اشتراک زمانی – مفسرهای دوسویه – نوع داده لیست پیوندی – مدیریت حافظه خودکار و برخی نکات کلیدی برنامه‌نویسی شیءگرا و محیط‌های توسعه برنامه مجتمع با واسط کاربر گرافیکی.

زبان‌شناسی (۱۹۷۵– تاکنون)

اسکینر در سال ۱۹۷۵ کتابی در زمینه رفتارگرایان برای یادگیری زبان، با نام «رفتار زبانی» منتشر کرد.
نوآم چامسکی بر اساس تئوری خودش یعنی ساختارهای ترکیبی، این کتاب را تجدید نظر و چاپ کرد. که به اندازه اصل کتاب شهرت پیدا کرد.
تئوری چامسکی بر اساس مدل‌های نحوی قرار دارد.
زبان‌شناسی مدرن و AI در یک زمان متولد شدند، بنابراین زبان‌شناسی نقش مهمی در رشد AI بازی نمی‌کند.
این دو دریک زمینه مشترک به نام
زبان‌شناسی محاسباتی(Computatioal linguistics) یا پردازش زبان طبیعی (natural language processing) بهم تنیده شده‌اند که در آن بر روی مسئله استفاده زبان تمرکز شده است.

تاریخچه هوش مصنوعی

پیدایش هوش مصنوعی (۱۹۴۳- ۱۹۵۶)
اشتیاق زودهنگام، آرزوهای بزرگ (۱۹۵۲-۱۹۶۹)
مقداری واقعیت (۱۹۷۴-۱۹۶۶)
سیستم‌های مبتنی بر دانش: کلید قدرت؟ (۱۹۶۹-۱۹۷۹)
بازگشت شبکه‌های عصبی (۱۹۸۶- تاکنون)
حوادث اخیر (۱۹۸۷- تاکنون)

پیدایش هوش مصنوعی

- اولین کار جدی در حیطه AI، توسط وارن مک‌کلود و والتر پیتز انجام شد.
- سه منبع استفاده شده توسط آنها:
  - دانش فیزیولوژی پایه و عملکرد نرون در مغز
  - تحلیل رسمی منطق گزاره‌ها متعلق به راسل و رایت هد
  - تئوری محاسبات تورینگ

در ۱۹۴۹ دونالد هب، قانون ساده بهنگام‌سازی برای تغییر تقویت اتصالات بین نرون‌ها را تعریف کرد که از طریق آن یادگیری میسر می‌گردد.
در زمانی که کلود شانون و آلن تورینگ، برنامه بازی شطرنج را نوشتند ، SNARC، اولین کامپیوتر شبکه عصبی در دانشگاه پرینستون توسط مینسکی و ادموندز ساخته شد.
این کامپیوتر، از ۳ هزار تیوپ مکشی و مکانیزم خلبانی خودکار اضافی که مربوط به بمب‌افکن‌های B24 می‌باشد برای شبیه‌سازی شبکه ۴۰ نرونی استفاده کرد.
محققین علاقمند به تئوری آتوماتا، شبکه‌های عصبی و مطالعه هوش، گرد یکدیگر جمع شدند و در کارگاهی در دورت موند مشغول فعالیت شدند. که در این میان نام هوش مصنوعی برای حیطه فعالیت آنها انتخاب شد.

اشتیاق زودهنگام، آرزوهای بزرگ (۱۹۵۲–۱۹۶۹)

فعالان در عرصه AI:
روچستو و تیمش در IBM
هربرت جلونتر: با ساخت Geometry Theorem Prover
آرتور ساموئل: ساخت برنامه برای بازی چکر
جان مک کارتی در MIT:
تعریف زبان لیسپ (Lisp) مهمترین زبان هوش مصنوعی
مفهوم اشتراک زمانی (time sharing)

نشر مقاله‌ای با عنوان "برنامه‌ها با حواس مشترک"
تشریح یک سیستم فرضی به نام Advice Taker ، که به اصول پایه بازنمایی معرفت و استدلال تجسم بخشید؛
کار بر روی سیستم برنامه‌ریزی سؤال-جواب
کار بر روی پروژه روبات‌های shakey
مینسکی: کار بر روی میکرو ورلدها و همکاری با مک‌کارتی، ولی بر سر اختلاف بر نگرش منطقی و ضدمنطقی کار تحقیقاتی خود را از هم جدا کردند.
مینسکی با گروهی از دانشجویان بر روی میکروورلدها کار کرد که برخی از آنها عبارتند از:
جیمز اسلاگل، SAINT، قادر به حل مسائل انتگرال‌گیری فرم بسته
اوانز: ANALOGY، حل مسائل مشابهت هندسی در تست‌های هوش
رافائل: SIR: پاسخ به قضایای پرسشی جملات ورودی
بابرو: STUDENT: حل مسائل داستانی جبر

مقداری واقعیت (۱۹۶۶–۱۹۷۴)

مشکلات تقریباً تمام پروژه‌ها تحقیقی AI وقتی پدیدار می‌شدند که مسائل گسترده‌تری برای حل توسط آنها مطرح می‌شد:
برنامه‌های اولیه اغلب دارای دانش محدود یا فاقد دانش در مورد موضوع کار بودند.
انجام ناپذیری بسیاری از مسائل
به دلیل اعمال برخی محدودیت‌های پایه‌ای بر روی ساختار پایه مورد استفاده برای تولید رفتار هوشمند

سیستم‌های مبتنی بر دانش: کلید قدرت؟ (۱۹۶۹–۱۹۷۹)

روش‌های ضعیف: مبتنی بر یک جستجوی همه‌منظوره می‌باشند که قدم‌های اولیه یادگیری را برمی‌دارند اما تلاشی در جهت یافتن راه‌حل‌های کامل ندارند.
به این دلیل که اطلاعات ضعیفی را در مورد دامنه فعالیت خود به کار می‌برند.
پس برای حل مسائل دشوار، تقریباً جواب را از قبل باید بدانیم.
برنامه DENDRAL از برنامه‌هایی است که از این رهیافت استفاده می‌کند.
اهمیت برنامه DENDRAL در این بود که اولین سیستم موفق با دانش غنی بود، یعنی تبحر سیستم بر پایه تعداد بسیار زیادی قانون ایجاد شده بود. سیستم‌های بعدی ایده اصلی رهیافت Advice taker مک کارتی را دنبال می‌کردند یعنی جداسازی دانش (در شکل قوانین) و مؤلفه استدلال.

MYCIN نسبت به DENDRAL دو تفاوت عمده دارد:

برخلاف قوانین DENDRAL، هیچ مدل تئوری‌وار عمومی برای آنکه قوانین MYCIN استنتاج شود، وجود نداشت.
قوانین می‌بایست عدم قطعیت مربوط به دانش پزشکی را منعکس می‌کرد.

AI به یک صنعت تبدیل می‌شود (۱۹۸۰–۱۹۸۸)

RI: اولین سیستم خبره تجاری موفق از شرکت DEC که سودآوری زیادی را برای شرکت بهمراه داشت.
پروژه «نسل پنجم»: این پروژه ژاپنی به منظور ساخت کامپیوترهای هوشمندی که پرولوگ را به جای کد ماشین اجرا می‌کردند، انجام شد.
شرکت‌های دیگر جهان از جمله میکروالکترونیک، MCC، لیسپ ماشین، تگزاس اینسترومنت، سمبولیکس، زیراکس و غیره در ساخت ایستگاه‌های کاری بهینه شده در این عرصه فعالیت داشتند.

بازگشت شبکه‌های عصبی:

دانشمندان فعال در این عرصه:

هاپ فیلد: که به آنالیز خواص ذخیره‌سازی و بهینه‌سازی شبکه‌ها پرداخت.
راسل هارت و هینتون: مطالعه مدل‌های شبکه عصبی را ادامه دادند.
بریسون و هو: الگوریتم یادگیری انتشار به عقب را مجدداً مطرح کردند.

حوادث اخیر:

رهیافت HMM: رهیافت غالب در سال‌های اخیر می‌باشد که توسط مایکف به وجود آمده است.
این رهیافت از دو جنبه زیر حائز اهمیت است:
مبتنی بر نظریه ریاضی محض است.
طی فرایندی با یادگیری گروه عظیمی از داده گفتار واقعی خود را بهبود می‌بخشد.
برنامه‌ریزی: در دهه ۷۰ فقط برای میکروواردها مناسب بودند، اکنون برای زمانبندی کار در کارخانه‌ها و مأموریت‌های فضایی استفاده می‌شوند.
بیان شبکه باور: استدلال کارا را در مورد ترکیب رویدادهای غیرمنطقی ممکن ساخت.
ایده سیستم‌های خبره فرماتیو توسط کار جوداپیر و اردیک هوروتیز و دیوید هکرمن مطرح شد:
"سیستم‌هایی که مطابق قوانین تئوری تصمیم‌گیری به طور منطقی عمل می‌کنند و سعی ندارند که تبحر انسانی را تقلید کنند."

شرایط کنونی:

برخی از سیستم‌هایی موجود در جهان که از هوش مصنوعی استفاده می‌کنند:
HITECH: اولین برنامه کامپیوتری که موفق به شکست استاد بزرگ شطرنج جهان، آرنولد دنکر شده است.
PEGASUS: یک برنامه درک گفتار که سؤالات کاربر را جواب می‌دهد و تمامی برنامه‌های مسافرتی شخص را با یک برنامه‌ریزی درست، مقرون به صرفه می‌کند.
MARVEL: سیستم خبره‌ای که داده‌های ارسالی از سفینه فضایی را تحلیل نموده و در صورت بروز مشکلات جدی، پیغام هشدار به تحلیلگران می‌دهد.

فصل دوم

عامل های هوشمند

عامل:

به هر چیزی اطلاق می‌شود، که قادر به درک محیط پیرامون خود از طریق حس‌گرها(sensor) و اثرگذاری بر روی محیط از طریق اثرکننده‌ها (effector) باشد.

عامل نرم‌افزاری:

عامل نرم‌افزاری رشته‌های بیتی را به عنوان درک محیط و عمل، کدگذاری می‌کند.

عوامل انسانی

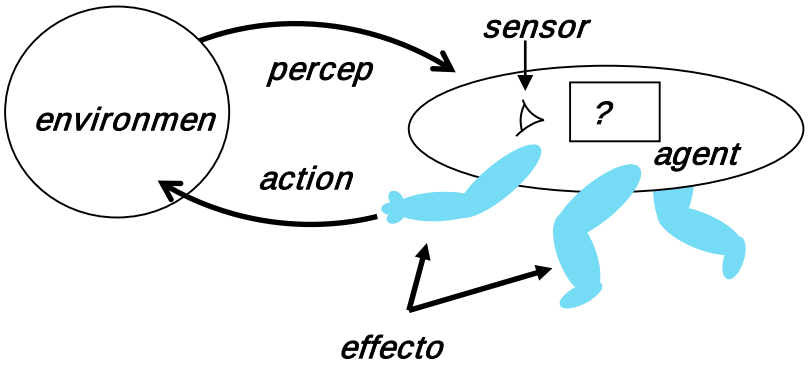
حس کردن: گوش، چشم، دیگر ارگان‌ها

اثرگذاری: دست، پا، بینی، اندام‌های دیگر

عوامل روباتیک

حس کردن: دوربین، یابنده‌های مادون قرمز

اثرگذاری: موتور



عامل‌ها چگونه باید عمل کنند؟

عامل منطقی: چیزی است که کار درست انجام می‌دهد.

عمل درست: آن است که باعث موفق‌ترین عامل گردد.

کارایی: چگونگی موفقیت یک عامل را تعیین می‌کند.

تفاوت میان منطقی بودن و دانش کل (omniscience):

عامل دانای کل معنی خروجی واقعی اعمال خود را دانسته و بر پایه آن عمل می‌کند اما دانش کل در واقعیت غیرممکن است.

اگر معین کنیم که هر عامل هوشمند همواره باید همان کاری را انجام دهد که در عمل مناسب است، هیچگاه نمی‌توان عاملی را طراحی نمود که این مشخصات را مرتفع سازد.

آن چه در هر زمانی منطقی است به چهار چیز وابسته است:

- معیار کارایی که درجه موفقیت را تعیین می‌کند.
- هر چیزی که تا کنون عامل، ادراک نموده است. ما این تاریخچه کامل ادراکی را دنباله ادراکی می‌نامیم.
- آنچه که عامل درباره محیط خود می‌داند.
- اعمالی که عامل می‌تواند صورت دهد.

رفتار عامل وابسته به دنباله ادراکی تا حال است.

عامل را باید به‌عنوان ابزاری برای تحلیل سیستم‌ها قلمداد کرد؛ نه شخصیتی مطلق که جهان را به دو بخش عامل و غیرعامل‌ها تقسیم می‌کند.

نگاشت ایده‌آل از دنباله‌های ادراکی به عملیات

هر عامل خاصی را به وسیله جدولی توصیف می‌کنیم، که در آن عمل آن در پاسخ به هر دنباله ادراکی قرار می‌گیرد.
این بدان معنی نیست که ما جدول خاصی با یک ورودی برای هر دنباله ادراک ممکنی تولید کنیم. می‌توان مشخصات نگاشت را بدون شمارش خسته‌کننده آنها انجام داد.

مثال: تابع ریشه دوم

دنباله ادراکی: دنباله‌ای از کلیدهای زده شده

نگاشت ایده‌آل:برای مقادیر مثبت X نشان داده شده توسط ادراک، Z نیز مثبت باشد و عمل مناسب نمایش نشان داده شود.

خودمختاری:

در اینجا تعریف عامل باید کامل‌تر شود و بخش دانش درونی به آن اضافه می‌گردد.
رفتار عامل می‌تواند متکی بر دو پایه تجربه خود و دانش درونی بنا نهاده شود.
این رفتار، در ساخت عامل برای شرایط محیطی خاص که در آن عمل خواهد کرد، استفاده می‌شود.
سیستم به وسعتی خود مختار است که رفتار آن بر اساس تجربه خودش تعیین می‌کند. زمانی که عامل فاقد تجربه و یا کم تجربه است، مسلماً تصادفی عمل خواهد کرد، مگر آنکه طرح کمک‌هایی به آن داده باشد.
عامل هوشمند واقعاً خود مختار باید قادر به عمل موفقیت‌آمیز در دامنه وسیعی از محیط‌ها باشد و البته باید زمان کافی برای تطبیق نیز به آن داده شود.

ساختار عامل‌های هوشمند

وظیفه هوش مصنوعی طراحی برنامه عامل است؛
این طراحی شامل تابعی است که نگاشت عامل از ادراک به عملیات را پیاده سازی می‌کند.
معماری: فرض می‌کنیم برنامه عامل بر روی نوعی ابزار محاسبه‌گر اجرا می‌گردد که آن را معماری می‌نامیم.
برنامه عامل، باید توسط معماری قابل پذیرش و اجرا باشد.
عموماً، معماری ادراک از طریق حس‌گرها را برای برنامه آماده ساخته، برنامه را اجرا نموده و اعمال انتخابی برنامه را به عمل‌کننده‌های سیستم منتقل می‌کند.
ارتباط بین عامل‌ها، معماری‌ها و برنامه‌ها را می‌توان به صورت ذیل جمع بندی نمود:
برنامه+ معماری= عامل
در اینجا مسئله تمایز بین محیط واقعی و مصنوعی مطرح می‌شود؛ اما

مسأله اصلی، پیچیدگی مابین:

ارتباط رفتار عامل، دنباله ادراکی تولید شده بوسیله محیط، و اهدافی که عامل قصد حصول آن را دارد، است.
مشهورترین محیط مصنوعی، محیط تست تورینگ (turing) است.

برنامه‌های عامل:

تشابهات عامل‌های هوشمند:

دریافت ادراک محیطی
تولید اعمال لازم

دو نکته در مورد شالوده برنامه قابل ذکر هستند:

برنامه عامل تنها یک درک از شرایط محیطی واحد را به عنوان ورودی دریافت می‌کند.
هدف یا معیار کارایی بخشی از برنامه شالوده نخواهد بود.

چرا تنها به پاسخ‌ها نگاه نمی‌کنیم؟

جدول مراجعه باید بر پایه حفظ کامل دنباله ادراکی در حافظه عمل نموده و از آن برای ایندکس‌سازی داخل جدول استفاده کند.

جدول عامل نوع راننده تاکسی

جنبه‌های مختلف یک عمل، انواع مختلف برنامه‌های عامل را پیشنهاد خواهد کرد.

نوع عامل	ادراکات	عملیات	اهداف	محیط
راننده تاکسی	دوربین‌ها، سرعت سنج، GPS، Sonar میکروفون	راهنمایی کردن، شتاب‌دهنده، ترمز، صحبت با مسافر	ایمنی، سرعت، قانونمندی، راحتی، افزایش سودمندی	جاده، پیاده‌رو، ترافیک، مشتری

برای مثال، ۴ عامل را مورد بررسی قرار می دهیم:

- عامل‌های واکنشی ساده
- عامل‌هایی که اثرات دنیا را حفظ می‌کنند
- عامل‌های هدف‌گرا
- عامل‌های سودمند

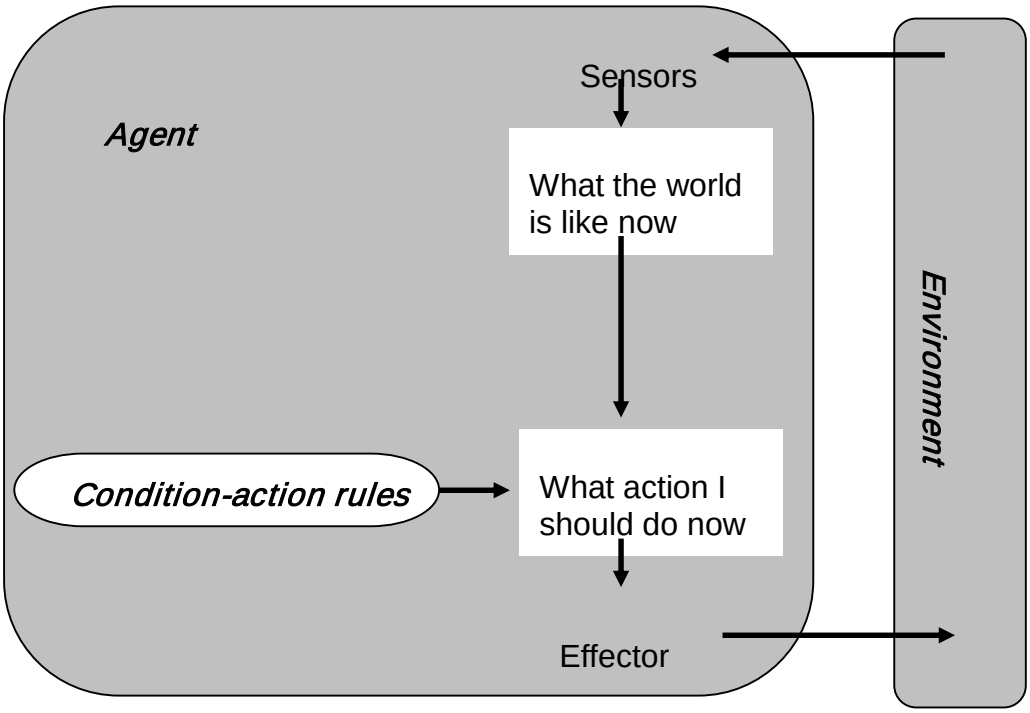
عامل‌های واکنشی ساده

در اینجا جدول رجوع باید مورد توجه قرار گرفته و فیلدهای مختلف آن توسط اطلاعات ورودی پر شود.

اتصالاتی (واکنش‌هایی) وجود دارند که انسان‌ها بسیاری از آنها را دارا بوده:

برخی از آنها قابل یادگیری و برخی دیگر غریزی است.

دیاگرام شماتیک از عامل ساده واکنشی



مربع مستطیل: نشان‌دهنده وضعیت داخلی جاری فرایند تصمیم‌گیری عامل

بیضی: نشان‌دهنده وضعیت اطلاعات پس‌زمینه

عامل‌هایی که اثرات دنیا را حفظ می‌کنند

از آنجایی ناشی می‌شود که حسگرها نمی‌توانند دسترسی کامل به وضعیت دنیا را به وجود آورند.

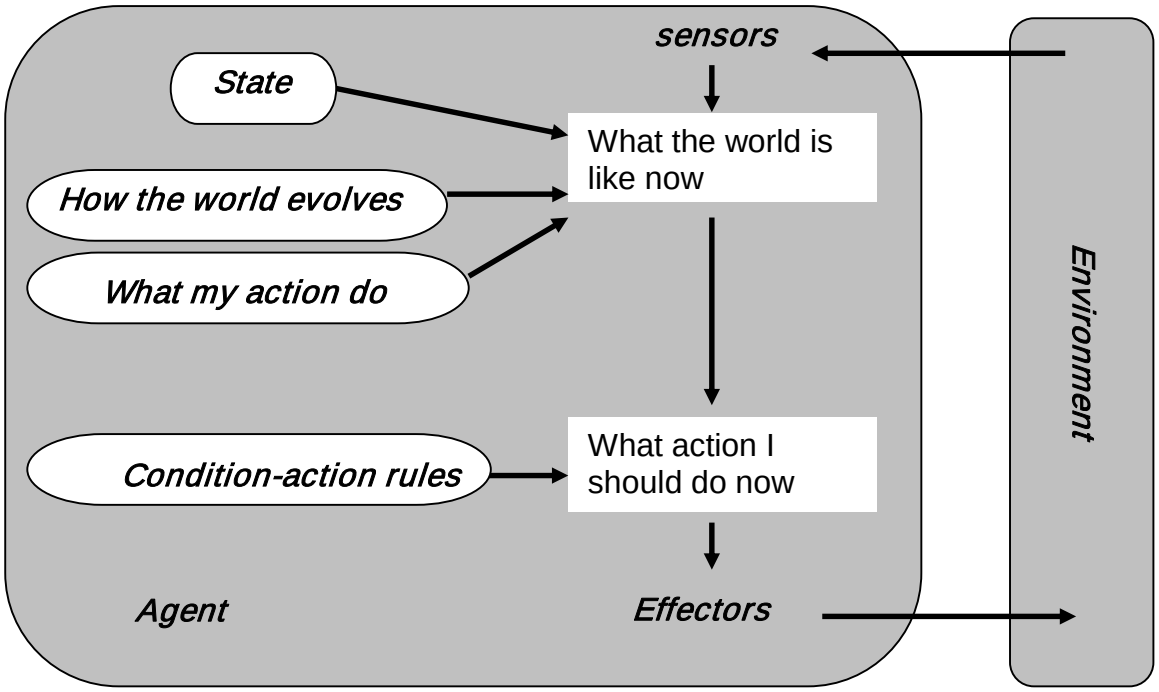
در چنین شرایطی، عامل ممکن است نیازمند دستکاری برخی اطلاعات وضعیت داخلی باشد تا از طریق آن تمایز بین وضعیت‌های دنیا که در ظاهر ورودی ادراکی یکسانی می‌کنند ولی در واقع معنی کاملاً متفاوتی دارند را میسر سازد.

بهنگام‌سازی اطلاعات وضعیت داخلی همزمان با گذر زمان نیازمند دو نوع دانش کد شده در برنامه عامل است.

اول: نیازمند آنیم که برخی اطلاعات درباره چگونگی تغییر جهان مستقل از عامل را داشته باشیم.

دوم: نیازمند اطلاعات درباره اعمال خود هستیم که بر روی دنیا اثرگذار است.

عامل واکنشی با حالت داخلی



عامل‌های هدف گرا:

دانستن درباره وضعیت کنونی محیط همواره برای تصمیم‌گیری عمل نمی‌تواند کافی باشد.

به همان گونه که عامل نیازمند شرح وضعیت جاری است، به نوعی نیازمند اطلاعات هدف(goal) می‌باشد که توضیح موقعیت مطلوب است.

برنامه عامل می‌تواند این اطلاعات را با اطلاعاتی درباره نتایج اعمال ممکن (همانند اطلاعاتی که در عامل واکنش برای بهنگام‌سازی وضعیت داخلی استفاده شد) ترکیب نموده تا اعمال مناسب را برای دسترسی به هدف انتخاب نماید.

در مواقعی ساده است: که رضایت از هدف بلافاصله از عمل واحد تولید گردد.

در مواقعی پیچیده است: که عامل باید دنباله‌های طولانی را در نظرگرفته تا راهی برای دستیابی به هدف پیدا کند.

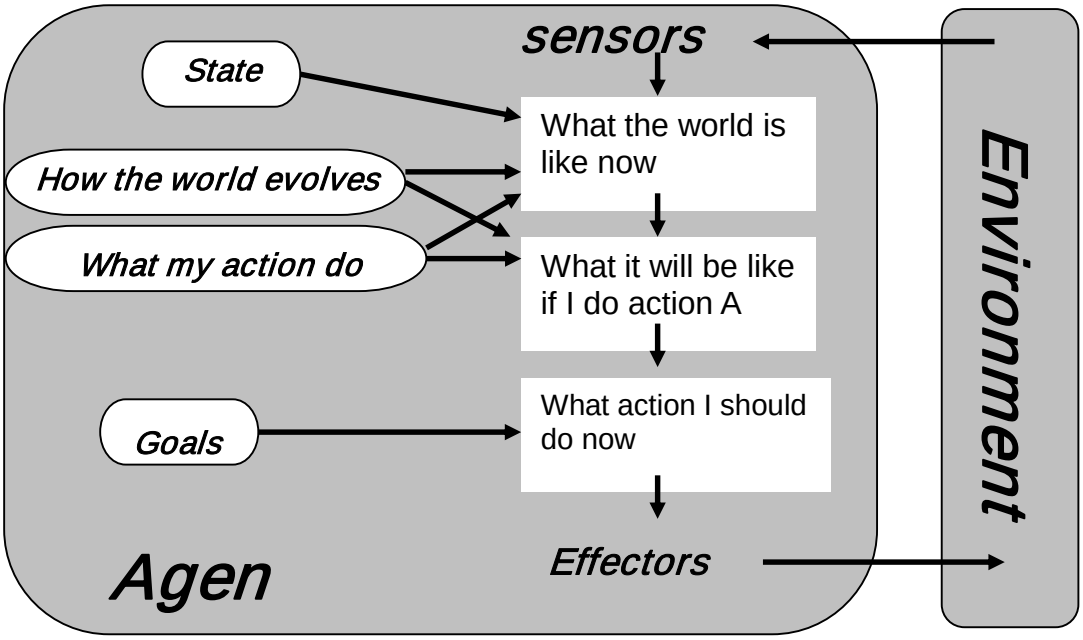
در مواقع پیچیده، جستجو و برنامه‌ریزی به یافتن دنباله اعمال منجر خواهند شد.

تفاوت عامل‌های واکنشی و هدف‌گرا:

در طراحی عامل‌های واکنشی طراح برای حالات متفاوت عملی درست را پیش محاسبه می‌کند. در عامل‌های هدف‌گرا، عامل می‌تواند دانش خود را در مورد چگونگی واکنش بهنگام سازد.

۱. برای عامل واکنشی ما مجبور به دوباره نویسی تعداد زیادی قوانین شرط –عمل خواهیم بود.
۲. عامل هدف‌گرا نسبت به رسیدن به مقاصد متفاوت انعطاف پذیر است.
۳. بسادگی با تعیین یک هدف تازه، می‌توانیم عامل هدف‌گرا را به رفتار تازه برسانیم.

عاملی با اهداف دقیق



عامل‌های سودمند:

اهداف به تنهایی برای تولید رفتار با کیفیت بالا کافی نیستند.

ملاک کارایی عومی باید مقایسه‌ای بین وضعیت‌های دنیای متفاوت (یا دنباله حالات) را بر پایه چگونگی رضایت عامل در صورت حصول هدف بدهد.

بنابراین اگر یک وضعیت دنیا به دیگری ترجیح داده می‌شود، آنگاه آن برای عامل سودمندتر خواهد بود

سودمندی: تابعی است که یک وضعیت را به عدد حقیقی نگاشت می‌دهد، که درجه رضایت مربوط را تشریح می‌کند.

مشخصات کامل تابع سودمندی امکان تصمیم‌گیری منطقی را برای دو نوع حالتی که هدف مشکل دارد، اجازه می‌دهد.

۱. زمانی که اهداف متناقض وجود دارند.

۲. زمانی که چندین هدف دارند که عامل می‌تواند آنها را هدف قرار دهد و هیچکدام از آنها با قطعیت قابل حصول نیست.

ارتباط بین عامل و محیط: اعمال بوسیله عامل بر محیط انجام می‌شود، که خود ادراک عامل را مهیا می‌سازد.

خواص محیط:

- ❖ قابل دسترسی در مقابل غیر دسترسی
- ❖ قطعی در برابر غیر قطعی
- ❖ اپیزودیک در مقابل غیراپیزودیک
- ❖ ایستا در مقابل پویا
- ❖ گسسته در مقابل پیوسته

قابل دسترسی در مقابل غیرقابل دسترسی

محیط قابل دسترسی: محیطی که عامل آن توسط ابزار حس‌کننده‌اش امکان دسترسی به وضعیت کامل محیط را داشته باشد.

محیط قابل دسترسی راحت است، زیرا عامل نیازمند دستکاری هیچ وضعیت داخلی برای حفظ دنیا را نخواهد داشت.

قطعی در مقابل غیر قطعی

محیط قطعی: محیطی است که اگر وضعیت بعدی محیط بوسیله وضعیت کنونی و اعمالی که با عامل‌ها انتخاب گردد، تعیین شود.

بهتر است به قطعی یا غیر قطعی بودن محیط از دیدگاه عامل نگاه کنیم.

اپیزودیک در مقابل غیر اپیزودیک

- ❖ محیط اپیزودیک (episodic)، تجربه عامل به اپیزودهایی تقسیم می‌گردد.
- ❖ هر اپیزود شامل درک و عمل عامل است.
- ❖ کیفیت اعمال آن تنها به خود اپیزود وابسته است.
- ❖ محیط‌های اپیزودی بسیار ساده‌ترند زیرا عامل نباید به جلوتر فکر کند.

ایستا در مقابل پویا

محیط پویا: محیطی که در حین سنجیدن عامل تغییر می‌کند.

محیط نیمه‌پویا: محیطی که با گذر زمان تغییر نمی‌کند اما امتیاز کارایی تغییر می‌کند.

محیط‌های ایستا برای کار ساده هستند زیرا عامل نیاز به نگاه‌کردن به دنیا در حین تصمیم‌گیری عملی نداشته و همچنین در مورد گذر زمان نیز نگران نمی‌باشد.

گسسته در مقابل پیوسته

محیط گسسته: اگر تعداد محدود و مجزا از ادراک و اعمال بوضوح تعریف شده باشد.

- بازی شطرنج گسسته است.
- رانندگی تاکسی پیوسته است.

سخت‌ترین حالت در بین حالات موجود برای محیط:

غیر قابل دسترسی، غیر اپیزودیک، پویا و پیوسته

مثال‌هایی از انواع محیط و ویژگی‌های آنها

محیط	قابل دسترسي	قطعي	اپیزودیک	ایستا	گسسته
شطرنج به همراه ساعت	YES	YES	NO	Semi	YES
شطرنج بدون ساعت	YES	YES	NO	YES	YES
پوکر	NO	NO	NO	YES	YES
تخته نرد	YES	NO	NO	YES	YES
راندن تاکسي	NO	NO	NO	NO	NO
سيستم تشخيص پزشكي	NO	NO	NO	NO	NO
سيستم تحليل تصوير	YES	YES	YES	Semi	NO
ربات جابجا کننده اشیاء	NO	NO	YES	NO	NO
کنترل‌کننده پالایشگاه	NO	NO	NO	NO	NO
آموزش‌دهنده انگليسي با ارتباط متقابل	NO	NO	NO	NO	YES

برنامه‌های محیط

شبیه‌ساز یک یا چند عامل را به عنوان ورودی گرفته و بگونه‌ای عمل می‌کند که هر عامل ادراک درست و نتیجه بازگشتی عمل خود را بدست آورد.

شبیه‌ساز محیط را بر اساس اعمال و دیگر فرآیندهای پویای محیط بهنگام می‌سازد.

محیط با وضعیت آغازین و تابع بهنگام‌سازی تعریف می‌گردد.

فصل سوم
حل مسائل توسط جستجو

یک نوع عامل هدفگرا، عامل حل مسئله نامیده می‌شود.
عامل‌های حل مسئله توسط یافتن ترتیب عملیات تصمیم می‌گیرند که چه انجام دهند تا آنها را به حالت‌های مطلوب سوق دهد.

عامل‌های حل مسئله

عامل‌های هوشمند به طریقی عمل می‌کنند که محیط مستقیماً به داخل دنباله حالت‌هایی وارد شود که معیار کارآرایی را افزایش می‌دهند.
عملیات به گونه‌ای ساده‌سازی می‌شوند که عامل قادر باشد تا هدفی را قبول کرده و به آن برسد.
الگوریتم جستجو مسئله‌ای را به عنوان ورودی دریافت نموده و راه‌حلی را به صورت دنباله عملیات بر می‌گرداند.
فاز اجرایی: مرحله‌ای است که در آن زمان، راه‌حلی پیدا می‌شود و عملیات پیشنهادی می‌توانند انجام شوند.
به طور ساده برای طرح یک عامل مراحل «فرموله‌سازی، جستجو، اجرا» را در نظر می‌گیریم.
پس از فرموله‌سازی یک هدف و یک مسئله برای حل عامل،

۱. رویه جستجویی را برای حل آن مسئله فراخوانی می‌کند.
۲. از راه حل برای راهنمایی عملیاتش استفاده می‌کند و هرآنچه که راه حل پیشنهاد می‌کند را انجام می‌دهد.
۳. آن مرحله را از دنباله حذف می‌کند.
۴. زمانی که راه‌حل اجرا شد، عامل هدف جدیدی را پیدا می‌کند.

چهار نوع اساسی از مسائل وجود دارند:

- مسائل تک حالته (Single-state)
- مسائل چند حالته (Multiple-state)
- مسائل احتمالی (Contingency)
- مسائل اکتشافی (Exploration)

دانش و انواع مسئله دنیای مکش (جاروبرقی):

اگر دنیا حاوی دو محل باشد:
هر محل ممکن است که شامل خاک باشد و یا نباشد و عامل ممکن است که در یک محل یا دیگر محل‌ها باشد؛ که دارای هشت حالت متفاوت خواهد بود.
هدف تمیز کردن تمام خاک‌هاست که در اینجا معادل با مجموعه حالت {۸و ۷} است.

مدل‌های مختلف برای مسئله جاروبرقی:

- مدل تک حالته:

حس‌گرهای عامل به آن اطلاعات کافی می‌دهند تا وضعیت دقیق مشخص شود. (دنیا قابل دسترسی است). عامل می‌تواند محاسبه کند که کدام وضعیت پس از هر دنباله از عملیات قرار خواهد گرفت.

- مدل چند حالته:

عامل تمام اثرهای عملیاتش را می‌داند اما دسترسی به حالت دنیا را محدود کرده است.
زمانی که دنیا تماماً قابل دسترسی نیست عامل باید در مورد مجموعه حالت‌هایی که ممکن است به آن برسد استدلال کند.

- مدل احتمالی:

با این مدل حل مسئله، حس‌گرهایی را در طول فاز اجرایی نیاز داریم. عامل اکنون باید تمام درخت عملیاتی را بر خلاف دنباله عملیاتی منفرد، محاسبه کند. که به طور کلی هر شاخه درخت، با یک امکان احتمالی که از آن ناشی می‌شود، بررسی می‌شود.

مدل اکتشافی:

عاملی که هیچ اطلاعاتی در مورد اثرات عملیاتش ندارد.

در این حالت، عامل باید تجربه کند و به تدریج کشف کند که چه عملیاتی باید انجام شود و چه وضعیت‌هایی وجود دارند.

این روش یک نوع جستجو است.

اگر عامل نجات یابد، «نقشه‌ای» از محیط را یاد می‌گیرد که می‌تواند مسائل بعدی را حل کند.

مسائل و راه‌حل‌های خوب تعریف شده

مسئله: در واقع مجموعه‌ای از اطلاعات است که عامل از آنها برای تصمیم‌گیری در مورد اینکه چه کاری انجام دهد، استفاده می‌کند.

عناصر اولیه تعریف یک مسئله، وضعیتها عملیات هستند.

برای تعریف یک مسئله موارد زیر نیاز داریم:

- ✓ وضعیت آغازین (initial state) که عامل خودش از بودن در آن آگاه است.
- ✓ مجموعه‌ای از عملیات ممکن، که برای عامل قابل دسترسی باشد.
- ✓ آزمون هدف (goal test)، که عامل می‌تواند در یک تعریف وضعیت منفرد آن را تقاضا کند تا تعیین گردد که آن حالت، وضعیت هدف است یا خیر.
- ✓ تابع هزینه مسیر، تابعی است که برای هر مسیر، هزینه‌ای را در نظر می‌گیرد؛ و با حرف g مشخص می‌شود.

هزینه یک سفر= مجموع هزینه‌های عملیات اختصاصی در طول مسیر

برای حل مسئله چند حالته، فقط به یک اصلاح جزئی نیاز داریم:

یک مسئله شامل:

❖ یک مجموعه حالت اولیه

❖ مجموعه‌ای از عملگرهای ویژه برای هر عمل به گونه‌ای که از هر وضعیت داده شده مجموعه‌ای حالات رسیده شده و یک آزمون هدف و تابع هزینه مسیر را معین کند.

یک عملگر:توسط اجتماع نتایج اعمال عملگر در هر وضعیت مجموعه، به کار برده می‌شود.

یک مسیر: مجموعه حالات را مرتبط می‌کند.

یک راه حل: مسیری است که به مجموعه‌ای از حالات که تمام آنها، وضعیت هدف هستند، سوق می‌دهند.

اندازه‌گیری کارایی حل مسئله:

کارایی یک جستجو، حداقل از سه طریق می‌تواند اندازه‌گیری شود:

❖ آیا این جستجو راه حلی پیدا می‌کند؟

❖ آیا راه حلی مناسبی است؟

❖ هزینه جستجو از نظر زمانی و حافظه مورد نیاز برای یافتن راه حل چیست؟

مجموع هزینه جستجو= هزینه مسیر + هزینه جستجو

عامل باید تصمیم بگیرد که چه منابعی را فدای جستجو و چه منابعی را صرف اجرا کند.

انتخاب حالات و عملیات

هنر واقعی حل مسئله، تصمیم‌گیری در مورد این است که چه چیزهایی در تعریف حالات و عملگرها باید به حساب آورده شوند و چه چیزهایی باید کنار گذاشته شوند.

انتزاع:

فرآیند حذف جزئیات از یک بارنمایی انتزاع (abstraction) نامیده می‌شود.

همانگونه که تعریف را خلاصه می‌کنیم می‌بایست عملیات را نیز خلاصه نمائیم.

انتزاع به این دلیل مفید است، که انجام هر کدام از عملیات آسانتر از مسئله اصلی است.

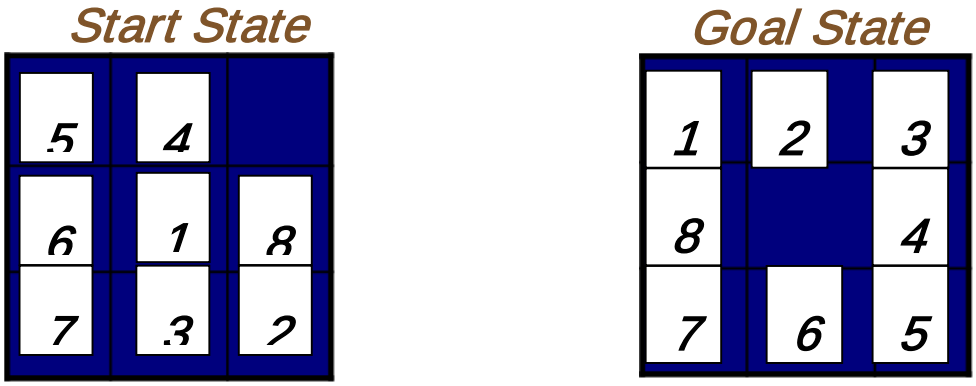
انتخاب یک انتزاع خوب از این رو شامل حذف تا حد ممکن می‌شود تا زمانی که عملیات خلاصه شده برای انجام آسان باشند.

مسائل نمونه: مسائل اسباب‌بازی

معمای ۸:

معمای ۸ نمونه‌ای است شامل یک صفحهٔ ۳*۳ با ۸ مربع شماره دار در یک صفحه خالی.

هر مربع که مجاور خانه خالی است. می‌تواند به درون آن خانه برود. هدف رسیدن به ساختاری است که در سمت راست شکل نشان داده شده است. نکته مهم این است که بجای اینکه بگوییم «مربع شماره ۴ را به داخل فضای خالی حرکت بده» بهتر است بگوییم «فضای خالی جایش را با مربع سمت چپش عوض کند.»



حالت‌ها: توصیف وضعیت مکان هر ۸ مربع را در یکی از ۶ خانهٔ صفحه مشخص می‌کند. برای کارایی بیشتر، بهتر است که فضاهای خالی نیز ذکر شود.

عملگرها: فضای خالی به چپ، راست، بالا و پائین حرکت کند.

آزمون هدف: وضعیت با ساختار هدف مطابقت می‌کند.

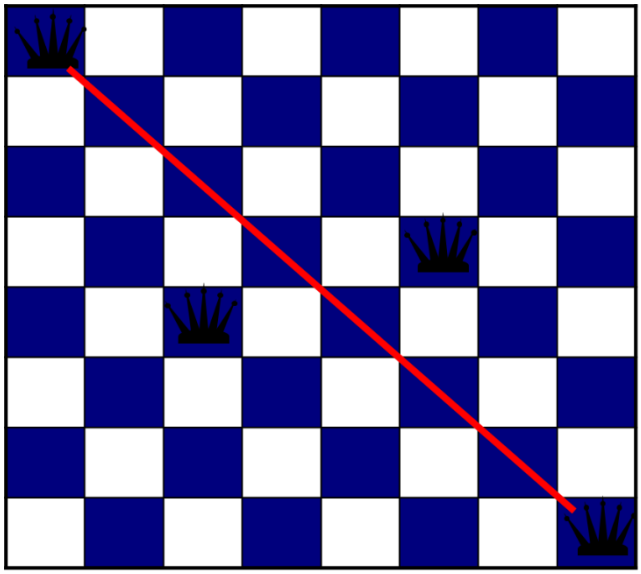
هزینه مسیر: هر قدم ارزش ۱ دارد، بنابراین هزینه مسیر همان طول مسیر است.

مسئله ۸ وزیر:

هدف از مسئله ۸ وزیر، قرار دادن ۸ وزیر بر روی صفحه شطرنج به صورتی است که هیچ وزیری نتواند به دیگری حمله کند.

دو نوع بیان ریاضی اصلی وجود دارد بیان افزایشی که با جایگزینی وزیرها، به صورت یکی یکی کار می‌کند و دیگری بیان وضعیت کامل که با تمام ۸ وزیر روی صفحه شروع می‌کند و آنها را حرکت می‌دهد.

در این فرمول ما ۶۴ امکان داریم.



بنابراین ما تست هدف و هزینه مسیر را به صورت زیر خواهیم داشت:

آزمون هدف: ۸ وزیر روی صفحه، که با هم برخورد ندارند.

هزینه مسیر: صفر.

حالات: ترتیب از صفر تا ۸ وزیر بدون هیچ برخورد.

عملگرها: یک وزیر را در خالی‌ترین ستون سمت چپ جایگزین کنید که هیچ برخوردی با بقیه نداشته باشد.

Cryptarithmetic :

در مسائل کریپتاریتمتیک، حروف به جای ارقام می‌نشینند و هدف یافتن جایگزینی از اعداد برای حروف است که مجموع نتیجه از نظر ریاضی درست باشد. معمولاً هر حرف باید به جای یک رقم مختلف بنشینند.

مثال:

```
FORTY    29786
+ TEN    +  850
+ TEN    +  850
----------  ----------
SIXTY    31486
```

F=2, O=9, R=7, etc.

یک فرمول ساده:

حالات: یک معمای Cryptarithmetic با چند حروف جایگزین شده توسط ارقام.

عملگرها: وقوع یک حروف را با یک رقم جایگزین کنید که قبلاً در معما ظاهر نشده باشد.

آزمون هدف: معما فقط شامل ارقام است و یک مجموع صحیح را بر می‌گرداند.

هزینه مسیر: صفر- تمام راه حل‌های صحیح است.

می‌خواهیم که از تبدیل جایگزینی‌های مشابه اجتناب کنیم:

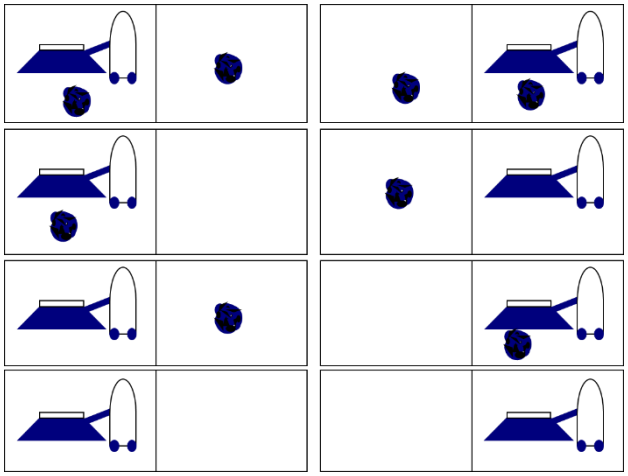
- ✓ قبول یک ترتیب ثابت مانند ترتیب الفبایی.
- ✓ هر کدام که بیشترین محدودیت جایگزینی را دارد، انتخاب کنیم؛ یعنی حرفی که کمترین امکان مجاز را دارند، محدودیت‌های معما را می‌دهد.

دنیای مکش:

مسئله تک حالته: عامل از جای خودش اطلاع دارد و تمام مکان‌های آلوده را می‌شناسد و دستگاه مکنده ما درست کار می‌کند.

حالات: یکی از ۸ حالت نشان داده شده.

عملگرها: حرکت به چپ، حرکت به راست، عمل مکش.
آزمون هدف: هیچ خاکی در چهار گوش‌ها نباشد.
هزینه مسیر: هر عمل ارزش ۱ دارد.



مسئله چند حالته: عامل دارای حسگر نمی‌باشد.
مجموعه وضعیت‌ها : زیر مجموعه‌ای از حالات.
عملگرها: حرکت به چپ، حرکت به راست، عمل مکش.
آزمون هدف: تمام حالات در مجموعه حالت‌ها فاقد خاک باشند.
هزینه مسیر: هر عمل هزینه ۱ دارد.

مسئله کشیش‌ها و آدمخوارها:

سه کشیش و سه آدم خوار در یک طرف رودخانه قرار دارند و هم چنین قایقی که قادر است یک یا دو نفر را حمل کند. راهی را بیابید که هر نفر به سمت دیگر رودخانه برود، بدون آنکه تعداد کشیش‌ها در یکجا کمتر از آدم خوارها شود.
حالات: یک حالت شامل یک دنبالهٔ مرتب شده از عدد است که تعداد کشیش‌ها، تعداد آدمخوارها و محل قایق در ساحلی از رودخانه که از آنجا مسئله شروع شده را نمایش می‌دهد.
عملگرها: از هر حالت، عملگرهای ممکن یک کشیش، یک آدمخوار، دو کشیش، دو آدمخوار، یا یکی از هر کدام را در قایق جا می‌دهند.
آزمون هدف: رسیدن به حالت(۰و ۰ و ۰).
هزینه مسیر: تعداد دفعات عبور از رودخانه.

مسائل دنیای واقعی

مسیریابی:

الگوریتم‌های مسیر یابی کاربردهای زیادی دراند، مانند مسیریابی در شبکه‌های کامپیوتری، سیستم‌های خودکار مسافرتی و سیستم‌های برنامه‌نویسی مسافرتی هوایی.

مسائل فروشنده دوره گرد و تور :

مسئله فروشنده دوره گرد مسئله مشهوری است که در آن هر شهر حداقل یکبار باید ملاقات شود هدف یافتن کوتاهترین مسیر است.
علاوه بر مکان عامل، هر حالت باید مجموعه شهرهایی را که عامل ملاقات کرده، نگه دارد.
علاوه بر برنامه‌ریزی صفر برای فروشنده دوره‌گرد، این الگوریتم‌ها برای اعمالی نظیر برنامه‌ریزی حرکات مته خوردکار سوراخ‌کننده برد مدار استفاده می‌شود.

طرح VISI :

ابزار طراحی کمکی کامپیوتری در هر فازی از پردازش استفاده می‌شود دو وظیفه بسیار مشکل عبارتند از:

Channel routing
Cell layout

که بعد از اینکه ارتباطات و اتصالات مدار کامل شد، این دو قسمت انجام می‌شوند.

هدف طراحی مداری روی تراشه است که کمترین مساحت و طول اتصالات و بیشترین سرعت را داشته باشد.

هدف قرار دادن سلول‌ها روی تراشه به گونه‌ای است که آنها روی هم قرار نگیرند و بنابراین فضایی نیز برای سیم‌های ارتباطی وجود دارد که باید بین سلول‌ها قرار گیرند.

کانال‌یابی، مسیر ویژه‌ای را برای هر سیم که از فواصل بین سلول‌ها استفاده می‌کند، پیدا می‌کند.

هدایت ربات:

یک ربات می‌تواند در یک فضای پیوسته با یک مجموعه نامحدودی از حالات و عملیات ممکن حرکت کند.

ربات‌های واقعی باید قابلیت تصحیح اشتباهات را در خواندن حسگرها و کنترل موتور داشته باشند.

خط تولید خودکار:

در مسائل سرهم‌بندی، مشکل یافتن قانونی است که تکه‌های چند شیئی را جمع کند. اگر ترتیب نادرست انتخاب شود، راهی نیست که بتوان قسمت‌های بعدی را بدون از نو انجام دادن قسمت‌های قبلی، اضافه کرد.

کنترل یک مرحله در دنباله، یک مسئله جستجوی پیچیدهٔ هندسی است که ارتباط نزدیکی با هدایت ربات دارد. از این رو تولید مابعدهای مجاز گران‌ترین قسمت دنباله سرهم‌بندی است و استفاده از الگوریتم‌های آگاهانه برای کاهش جستجو، ضروری است.

جستجو برای راه‌حل:

نگهداری و گسترش یک مجموعه از دنباله‌های راه حل ناتمام.

جستجوی حالت‌های موجود و یافتن راه‌حل بنا بر اصل جستجو.

تولید دنباله‌های عمل:

فرایند گسترش حالت: فرایندی که از طریق تولید مجموعه جدیدی از حالات، عملگرها در حالت جاری را به کار گرفته، و نتیجتاً حالت هدف را در مجموعه وارد می‌کند.

اصل جستجو: انتخاب یک حالت و کنار گذاشتن بقیه برای بعد، زمانی که اولین انتخاب به حل مسئله منجر نشود.

ریشه درخت جستجو: یک گره جستجو است که با حالت اولیه مطابقت دارد.

گره‌های برگی درخت: حالاتی هستند که دارای فرزندی در درخت نیستند.

ساختارهای داده برای درخت‌های جستجو:

گره به عنوان یک ساختار داده با پنج قسمت به شرح زیر است:

- وضعیتی که گره در فضای حالات دارا می‌باشند.
- گره‌ای که در جستجوی درخت، گره جدیدی را تولید کرده است (گره والد).
- عملگری که برای تولید گره به کار رفته است.
- تعداد گره‌های مسیر، از ریشه تا گره موردنظر (عمق گره).
- هزینه مسیر، از حالت اولیه تا گره.

تفاوت بین گره‌ها و حالت‌ها:

گره‌ها عمق و والد دارند؛ در صورتی که حالت‌ها شامل چنین چیزهایی نیستند.

استراتژی جستجو:

استراتژی‌ها باید دارای ۴ معیار زیر باشند:

- ❖ کامل بودن
- ❖ پیچیدگی زمانی
- ❖ پیچیدگی فضا
- ❖ بهینگی

ما ۶ استراتژی را بررسی خواهیم کرد:

- جستجوی سطحی
- جستجوی با هزینه یکسان
- جستجوی عمقی
- جستجوی عمقی محدود شده
- جستجوی عمیق‌کننده تکراری
- جستجوی دوطرفه
- **جستجوی سطحی:**
- در این استراتژی که بسیار سیستماتیک است، ابتدا گره ریشه، و سپس تمام گره‌های دیگر گسترش داده می‌شوند.
- به عبارت کلی‌تر، تمام گره‌های عمیق d، قبل از گره‌های عمیق d+1 گسترش داده می‌شوند.
- **مزایا:**
- جستجوی سطحی، کامل و بهینه می‌باشد زیرا هزینه مسیر، یک تابع کاهش‌نیابنده از عمق گره است.
- **معایب:**
- مرتبه زمانی $O(bd)$ می باشد که نمایی است.
- نیاز به حافظه زیاد.

جستجوی با هزینه یکسان:

در این استراتژی، در شرایط عمومی، اولین راه حل، ارزان‌ترین راه نیز هست.

اگر هزینه مسیر توسط تابع $g(n)$ اندازه‌گیری شود، در این صورت جستجوی سطحی همان جستجوی با هزینه یکسان است با:

$$g(n)=DEPTH(n)$$

جستجوی عمقی:

این استراتژی، یکی از گره‌ها را در پائین‌ترین سطح درخت بسط می‌دهد؛ اما اگر به نتیجه نرسید، به سراغ گره‌هایی در سطوح کم عمیق‌تر می‌رود.

مزایا:

این جستجو، نیاز به حافظه نسبتاً کمی فقط برای ذخیره مسیر واحدی از ریشه به یک گره برگی، و گره‌های باقی‌مانده بسط داده نشده دارد.

پیچیدگی زمانی $O(bm)$ می‌باشد. به طوریکه b فاکتور انشعاب فضای حالت، و m حداکثر عمق درخت باشد.

معایب:

اگر مسیری را اشتباه طی کند، هنگام پائین رفتن گیر می‌کند.

جستجوی عمقی نه کامل و نه بهینه است.

در درخت‌های با عمق نامحدود و بزرگ این استراتژی کار نمی‌کند.

جستجوی عمقی محدود شده:

این استراتژی، برای رهایی از دامی که جستجوی عمقی در آن گرفتار می‌شد، از یک برش استفاده می‌کند.
جستجوی عمقی محدود شده کامل است اما بهینه نیست.
زمان و پیچیدگی فضای جستجوی عمقی محدودشده، مشابه جستجوی عمقی است. این جستجو پیچیدگی زمانی O(bL) و فضای O(bL) را خواهد داشت، که L محدودهٔ عمق است.
در یک درخت جستجوی نمایی، تقریباً تمام گره‌ها در سطح پائین هستند، بنابراین موردی ندارد که سطوح بالایی چندین مرتبه بسط داده شوند. تعداد بسط‌ها در یک جستجوی عمقی محدود شده با عمق d و فاکتور انشعاب b به قرار زیر است:

1+b+b2+...+bd-2+bd-1+bd

جستجوی عمیق‌کننده تکراری:

قسمت دشوار جستجوی عمقی محدود شده، انتخاب یک محدودهٔ خوب است.
اگر محدودهٔ عمق بهتری را پیدا کنیم، این محدوده، ما را به سوی جستجوی کاراتری سوق می‌دهد. اما برای بیشتر مسائل، محدودهٔ عمقی مناسب را تا زمانی که مسئله حل نشده است، نمی‌شناسیم.
جستجوی عمیق‌کنندهٔ تکراری استراتژی است که نظریه انتخاب بهترین محدودهٔ عمقی، توسط امتحان تمام محدودهٔ مسیرهای ممکن را یادآوری می‌کند.

مزایا:

ترکیبی از مزایای جستجوی سطحی و عمقی را دارد.
این جستجو مانند جستجوی سطحی کامل و بهینه است، اما فقط مزیت درخواست حافظه اندک را از جستجوی عمقی دارد.
مرتبه بسط حالات مشابه جستجوی سطحی است، به جز اینکه بعضی حالات چند بار بسط داده می‌شوند.
در جستجوی عمیق‌کننده تکراری، گره‌های سطوح پائینی یک بار بسط داده می‌شوند، آنهایی که یک سطح بالاتر قرار دارند دوبار بسط داده می‌شوند و الی‌آخر تا به ریشه درخت جستجو برسد، که d+1 بار بسط داده می‌شوند.

بنابراین مجموع دفعات بسط در این جستجو عبارتست از:

(d+1)1+(d)b+(d-1)b2+...+3bd-2+2bd-1+1bd

پیچیدگی زمانی این جستجو هنوز O(bd) است، و پیچیدگی فضا O(bd) است.
در حالت کلی، عمیق‌کننده تکراری، روش جستجوی برتری است؛ زمانی که فضای جستجوی بزرگی وجود دارد و عمق راه حل نیز مجهول است.

جستجوی دوطرفه:

ایده جستجوی دوطرفه در واقع شبیه‌سازی جستجویی به سمت جلو از حالت اولیه و به سمت عقب از هدف است و زمانی که این دو جستجو به هم برسند، متوقف می‌شود.
برای پیاده‌سازی الگوریتم سؤالات زیر باید پاسخ داده شوند:
۱- سؤال اصلی این است که، جستجو از سمت هدف به چه معنی است؟ ماقبل‌های (predeccessors) یک گره n را گره‌هایی درنظر می‌گیریم که n مابعد (successor) آنها باشد. جستجو به سمت عقب بدین معناست که تولید ماقبل‌ها از گرهٔ هدف آغاز شود.
۲- زمانی که تمام عملگرها، قابل وارونه‌شدن باشند، مجموعه ماقبل‌ها و مابعدها یکسان هستند.
۳- چه کار می‌توان کرد زمانی که هدف‌های متفاوتی وجود داشته باشد؟ اگر لیست صریحی از حالت‌های هدف وجود داشته باشد، می‌توانیم یک تابع ماقبل برای مجموعه حالت تقاضا کنیم در حالیکه تابع مابعد یا (جانشین) در جستجوی مسائل چندوضعیته به کار می‌رود.

۴- باید یک راه موثر برای کنترل هر گره جدید وجود داشته باشد تا متوجه شویم که آیا این گره قبلاً در درخت جستجو توسط جستجوی طرف دیگر، ظاهر شده است یا خیر.
۵- نیاز داریم که تصمیم بگیریم که چه نوع جستجویی در هر نیمه قصد انجام دارد.

مقایسه استراتژی‌های جستجو:

ارزیابی استراتژی‌های جستجو. b فاکتور انشعاب، d عمل پاسخ، m ماکزیمم عمق درخت جستجو، l محدودیت عمق است.

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Time	b^d	b^d	b^m	b^l	b^d	$b^{d/2}$
Space	b^d	b^d	bm	bl	bd	$b^{d/2}$
Optimal?	Yes	Yes	No	No	Yes	Yes
Complete	Yes	Yes	No	Yes, if	Yes	Yes

اجتناب از حالات تکراری:

برای مسائل زیادی، حالات تکراری غیرقابل اجتناب هستند. این شامل تمام مسائلی می‌شود که عملگرها قابل وارونه شدن باشند، مانند مسائل مسیریابی و کشیش‌ها و آدمخوارها.

سه راه برای حل مشکل حالات تکراری برای مقابله با افزایش مرتبه و سرریزی فشار کار کامپیوتر وجود دارد:

به حالتی که هم اکنون از آن آمده‌اید، برنگردید. داشتن تابع بسط (یا مجموعه عملگرها) از تولید مابعدهایی که مشابه حالتی هستند که در آنجا نیز والدین این گره‌ها وجود دارند، جلوگیری می‌کند.

۱. از ایجاد مسیرهای دوار بپرهیزید. داشتن تابع بسط (یا مجموعه عملگرها) از تولید مابعدهای یک گره که مشابه اجداد آن گره است، جلوگیری می‌کند.

۲. حالتی را که قبلاً تولید شده است، مجدداً تولید نکنید. این مسئله باعث می‌شود که هر حالت در حافظه نگهداری شود، پیچیدگی فضایی O(bd) داشته باشد. بهتر است که به O(s) توجه کنید که s تعداد کل حالات در فضای حالت ورودی است.

جستجوی ارضاء محدودیت (Constraint Satisfaction Problem):

نوع خاصی از مسئله است که CSP، حالات توسط مقادیر مجموعه‌ای از متغیرها تعریف می‌شوند و آزمون هدف مجموعه‌ای از محدودیت‌ها را به آنها اختصاص می‌دهد که متغیر ملزم به پیروی از آنها هستند.

CSPها می‌توانند توسط الگوریتم‌های جستجوی geneal-purpose حل شوند، اما به علت ساختار خاص آنها، الگوریتم‌هایی صرفاً برای CSPهایی طرح می‌شوند که از الگوریتم‌های عمومی کارآیی بهتری دارند.

محدودیت‌ها به گونه‌های مختلفی ظاهر می‌شوند.

- ❖ محدودیت‌های یکتا
- ❖ محدودیت‌های دودویی
- ❖ محدودیت‌های مطلق

❖ محدودیت‌های اولویت‌دار

✓ در CSPهای گسسته که دامنه‌های آن محدود هستند، محدودیت‌ها می‌توانند به سادگی توسط شمردن ترکیبات مجاز مقادیر نمایش داده شوند.

✓ با استفاده از یک شماره‌گذاری، هر CSP گسسته می‌تواند به یک CSP دودویی تبدیل شود.

چطور یک الگوریتم جستجوی همه منظوره را در یک CSP به کار ببریم:

در حالتی که تمام متغیرها، تعیین نشده‌اند:

عملگرها مقداری را به یک متغیر از مجموعه مقادیر ممکن، نسبت می‌دهند.

آزمون هدف تمام متغیرها کنترل می‌کند که آیا مقدار گرفته‌اند و تمام محدودیت‌ها از بین رفته‌اند یا خیر.

- توجه کنید که حداکثر عمق درخت جستجو در n و تعداد متغیرها و تمام راه‌حل‌ها در عمق n هستند.
- جستجوی عمقی روی یک CSP زمان جستجو را تلف می‌کند زمانی که محدودیت‌ها قبلاً مختلف شده باشند.

فصل چهارم

روش‌های جستجو آگاهانه

جستجوی بهترین:

این استراتژی به این صورت بیان می‌شود که در یک درخت، زمانی که گره‌ها مرتب می‌شوند، گره‌ای که بهترین ارزیابی را داشته باشد، قبل از دیگر گره‌ها بسط داده می‌شود.

هدف: یافتن راه‌حل‌های کم‌هزینه است، این الگوریتم‌ها عموماً از تعدادی معیار تخمین برای هزینه راه‌حل‌ها استفاده می‌کنند و سعی بر حداقل کردن آنها دارند.

حداقل هزینه تخمین زده شده برای رسیدن به هدف: جستجوی حریصانه

یکی از ساده‌ترین استراتژی‌های جستجوی بهترین، به حداقل رساندن هزینه تخمین زده شده برای رسیدن به هدف است. بدین صورت که حالت گره‌ای که به حالت هدف نزدیک‌تر است، ابتدا بسط داده می‌شود.

تابع کشف‌کننده: هزینه رسیدن به هدف از یک حالت ویژه می‌تواند تخمین زده شود اما دقیقاً تعیین نمی‌شود. تابعی که چنین هزینه‌هایی را محاسبه می‌کند تابع کشف‌کننده h نامیده می‌شود.

جستجوی حریصانه: جستجوی بهترین که h را به منظور انتخاب گره بعدی برای بسط استفاده می‌کند، جستجوی حریصانه (greedy search) نامیده می‌شود.

ویژگی‌های جستجوی حریصانه:

- ❖ جستجوی حریصانه از لحاظ دنبال کردن یک مسیر ویژه در تمام طول راه به طرف هدف، مانند جستجوی عمقی است، اما زمانی که به بن‌بست می‌رسد، برمی‌گردد.
- ❖ این جستجو بهینه نیست و ناکامل است.
- ❖ پیچیدگی زمانی در بدترین حالت برای جستجوی حریصانه $O(bm)$، که m حداکثر عمق فضای جستجو است.
- ❖ جستجوی حریصانه تمام گره‌ها را در حافظه نگه می‌دارد، بنابراین پیچیدگی فضای آن مشابه پیچیدگی زمانی آن است.
- ❖ میزان کاهش پیچیدگی به مسئله و کیفیت تابع h بستگی دارد.

حداقل‌سازی مجموع هزینه مسیر: جستجوی A*

جستجو با هزینه یکسان، هزینه مسیر، $g(n)$ را نیز حداقل می‌کند.

با ترکیب دو تابع ارزیابی داریم:

$$f(n) = g(n) + h(n)$$

$g(n)$: هزینه مسیر از گره آغازین به گره n را به ما می‌دهد.

$h(n)$: هزینه تخمین زده شده از ارزانترین مسیر از n به هدف است

و ما داریم:

هزینه تخمین زده شده ارزانترین راه حل از طریق $n = f(n)$

کشف‌کنندگی قابل قبول:

تابع hای را که هزینه‌ای بیش از تخمین برای رسیدن به هدف نداشته باشد، یک کشف‌کنندگی قابل قبول (admissible heuristic) گویند.

جستجوی *A:

جستجوی بهترین که f به عنوان تابع ارزیاب و یک تابع h قابل قبول استفاده می‌کند، به عنوان جستجوی *A شناخته می‌شود.

رفتار جستجوی *A

نگاهی گذرا به اثبات کامل و بهینه بودن *A:

مشاهده مقدماتی:

تقریباً تمام کشف‌کنندگی‌های مجاز دارای این ویژگی هستند که در طول هر مسیری از ریشه، هزینه f هرگز کاهش پیدا نمی‌کند.

این خاصیت برای کشف‌کنندگی، خاصیت یکنوایی (monotonicity) گفته می‌شود.

اگر یکنوا نباشد، با ایجاد یک اصلاح جزئی آن را یکنوا می‌کنیم.

بنابراین هر گره جدیدی که تولید می‌شود، باید کنترل کنیم که آیا هزینهٔ f این گره از هزینه f پدرش کمتر است یا خیر.

اگر کمتر باشد، هزینهٔ f پدر به جای فرزند می‌نشیند:

بنابراین:

f همیشه در طول هر مسیری از ریشه غیرکاهشی خواهد بود، مشروط بر اینکه h امکان‌پذیر باشد.

h*(n): هزینه واقعی رسیدن از n به هدف است.

در استفاده عملی، خطاها با هزینه مسیر متناسب هستند، و سرانجام رشد نمایی هر کامپیوتر را تسخیر می‌کند. البته، استفاده از یک کشف‌کنندگی خوب هنوز باعث صرفه‌جویی زیادی نسبت به جستجوی ناآگاهانه می‌شود.

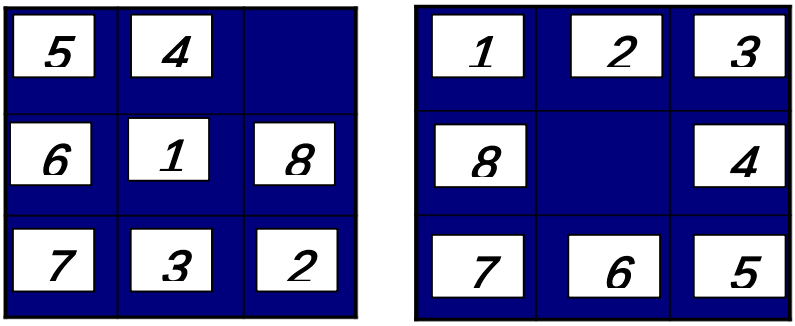
A* معمولاً قبل از اینکه دچار کمبود زمان شود، دچار کمبود فضا می‌شود. زیرا این جستجو تمام گره‌های تولید شده را در حافظه ذخیره می‌کند.

توابع کشف‌کننده:

مسئله ۸ را بررسی می‌کنیم:

معمای ۸ یکی از مسائل اولیه کشف‌کنندگی بود.

هدف: لغزاندن چهارخانه‌ها به طور افقی یا عمودی به طرف فضای خالی است تا زمانی که ساختار کلی مطابق با هدف (goal) باشد.



اگر خواستار یافتن راه‌حل‌های کوتاه باشیم، به یک تابع کشف‌کننده نیاز داریم که هرگز در تعداد مراحل به هدف اغراق نکند. در اینجا ما دو کاندید داریم:

h1 = تعداد چهارخانه‌هایی که در مکان‌های نادرست هستند. h1 یک کشف‌کننده مجاز است، زیرا واضح است که هر چهارخانه‌ای که خارج از مکان درست باشد حداقل یکبار باید جابجا شود.

h2 = مجموع فواصل چهارخانه‌ها از مکان‌های هدف صحیحشان است. فاصله‌ای که ما حساب می‌کنیم، مجموع فواصل عمودی و افقی است که بعضی وقتها city block distance و یا Manhattan distance نامیده می‌شود.

اثر صحت کشف‌کنندگی بر کارایی:

یک راه برای تشخیص کیفیت کشف‌کنندگی فاکتور انشعاب مؤثر b* است. اگر مجموع تعداد گره‌های بسط داده شده توسط A*برای یک مسئله ویژه N باشد و عمق راه حل d، سپس b* فاکتور انشعابی است که یک درخت یکنواخت با عمق d خواهد داشت تا گره‌های N را نگهدارد. بنابراین:

$$N=1+ b^*+(b^*)2...+(b^*)d$$

- معمولاً فاکتور انشعاب مؤثر که توسط کشف‌کنندگی نمایش داده می‌شود، مقدار ثابتی دارد.

- یک کشف‌کنندگی خوب طراحی شده، *b در حدود ۱ دارد.

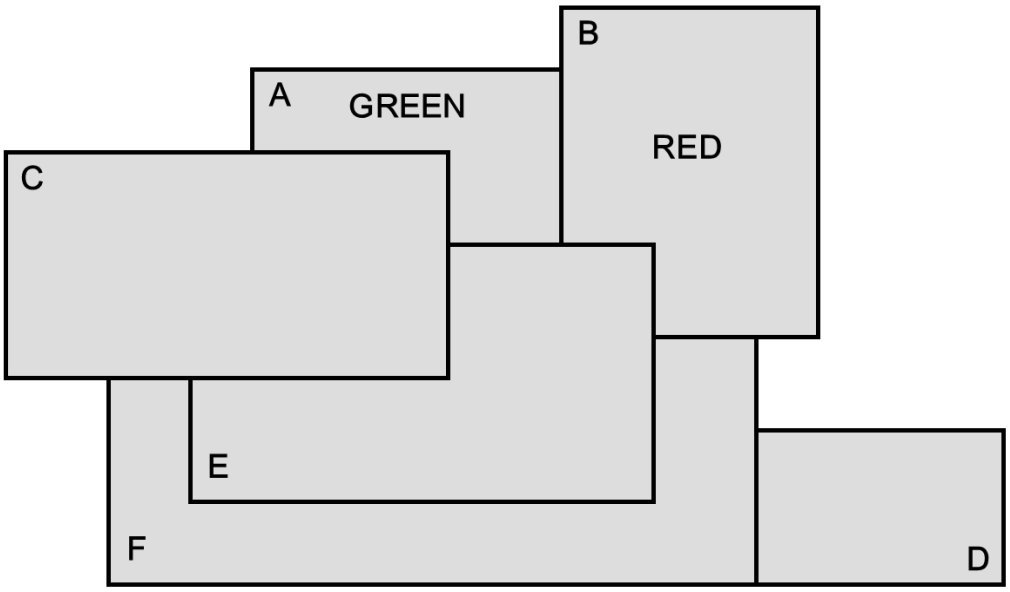
کشف‌کننده‌ها برای مسائل ارضا محدودیت:

مسئله ارضاء محدودیت شامل یک سری از متغیرهایی است که ,ویژگی‌های زیر را دارا هستند:

- ✓ می‌توانند مقادیری را از دامنهٔ داده شده دریافت کنند.
- ✓ با یک سری از محدودیت‌ها، ویژگی‌های راه حل را مشخص کنند.

رنگ‌آمیزی نقشه نمونه‌ای از این کشف‌کننده‌هاست:

هدف رنگ‌آمیزی نقشه، اجتناب از رنگ‌آمیزی مشابه دو کشور همسایه است.



ما حداکثر از سه رنگ (قرمز، آبی، سبز) می‌توانیم استفاده کنیم:

اگر رنگ سبز را برای کشور A، قرمز را برای B، انتخاب کنیم، کشور E باید آبی باشد. در این صورت ما ناچاریم که C را به رنگ قرمز درآوریم و F سبز باشد. رنگ‌آمیزی D با رنگ آبی یا قرمز، بستگی به راه‌حل دارد. در این حالت مسئله بدون هیچگونه جستجویی حل می‌شود.

جستجوی *SMA:

الگوریتم *SMA، حافظه محدود *A ساده شده (Simplified-Memory-BoundedA*) می‌باشد.

این الگوریتم، قادر است تا از تمام حافظه موجود برای اجرای جستجو استفاده کند. استفاده از حافظه بیشتر کارایی جستجو را وسعت می‌بخشد. می‌توان همیشه از فضای اضافی صرفنظر کرد.

SMA* دارای خواص زیر است:

- ❖ می‌تواند از تمام حافظه قابل دسترس استفاده کند.
- ❖ از حالات تکراری تا جایی که حافظه اجازه می‌دهد، جلوگیری می‌کند.
- ❖ این الگوریتم کامل است به شرط آنکه حافظه برای ذخیره کم عمق‌ترین مسیر راه حل کافی باشد.
- ❖ این الگوریتم بهینه است، اگر حافظه کافی برای ذخیره کم‌عمق‌ترین مسیر راه‌حل کافی باشد. بعلاوه بهترین راه‌حلی را برمی‌گرداند که بتواند با حافظه موجود مطابقت داشته باشد.
- ❖ زمانی که حافظه موجود برای درخت جستجوی کامل کافی باشد، جستجو Optimally efficient است.

طراحی *SMAساده است.

- زمانی که نیاز به تولید فرزند داشته باشد ولی حافظه‌ای نداشته باشد، نیاز به ساختن فضا بر روی صف دارد. برای انجام این امر، یک گره را حذف می‌کند. گره‌هایی که به این طریق از صف حذف می‌شوند، گره‌های فراموش‌شده یا (forgotten nodes) نامیده می‌شوند.

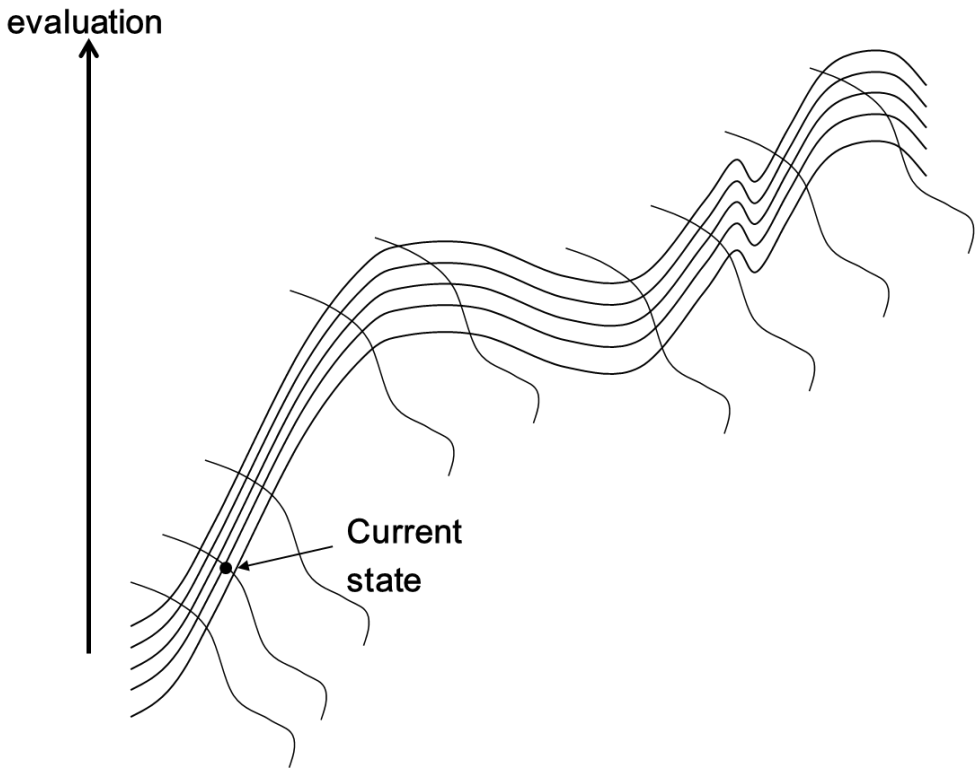
▪ برای اجتناب از جستجوی مجدد زیردرخت‌هایی که از حافظه حذف شده‌اند، در گره‌های اجدادی، اطلاعاتی در مورد کیفیت بهترین مسیر در زیر درخت فراموش شده، نگهداری می‌شود.

الگوریتم‌های اصلاح تکراری

▪ بهترین راه برای فهم الگوریتم‌های اصلاح تکراری درنظر داشتن تمام حالاتی است که روی سطح یک دورنمایی در معرض دید قرار داده شده است. ارتفاع هر نقطه در دورنما مطابق با تابع ارزیاب حالت آن نقطه است. ایده اصلاح تکراری، حرکت کردن در اطراف دورنما و سعی بر یافتن قله‌های مرتفع است، که همانا راه‌حل‌های بهینه هستند.

▪ الگوریتم‌های اصلاح تکراری معمولاً اثر حالت جاری را فقط حفظ می‌کنند، و توجهی فراتر از همسایگی آن حالت ندارند.

الگوریتم‌های اصلاح تکراری سعی بر یافتن قله‌هایی بروی سطح حالات دارند، جائی که ارتفاع توسط تابع ارزیابی تعریف می‌شود.



<u>این الگوریتم‌ها به دو گره اصلی تقسیم می‌شوند.</u>

- ❖ الگوریتم‌های تپه‌نوردی (Hill-climbing)
- ❖ Simulated annealing

۱- الگوریتم‌های جستجوی تپه‌نوردی (Hill-climbing)

یک اصلاح خوب این است زمانی که بیش از یک فرزند خوب برای انتخاب وجود دارد، الگوریتم بتواند به طور تصادفی از میان آنها یکی را انتخاب کند.

این سیاست ساده، سه زیان عمده دارد:

❖ Local Maxima: یک ماکزیمم محلی، برخلاف ماکزیمم عمومی، قله‌ای است که پائین‌تر از بلندترین قله درفضای حالت است. زمانی که روی ماکزیمم محلی هستیم، الگوریتم توقف خواهد نمود. اگرچه راه حل نیز ممکن است دور از انتظار باشد.

❖ Plateaux: یک فلات محوطه‌ای از فضای حالت است که تابع ارزیاب یکنواخت باشد. جستجو یک قدم تصادفی را برخواهد داشت.

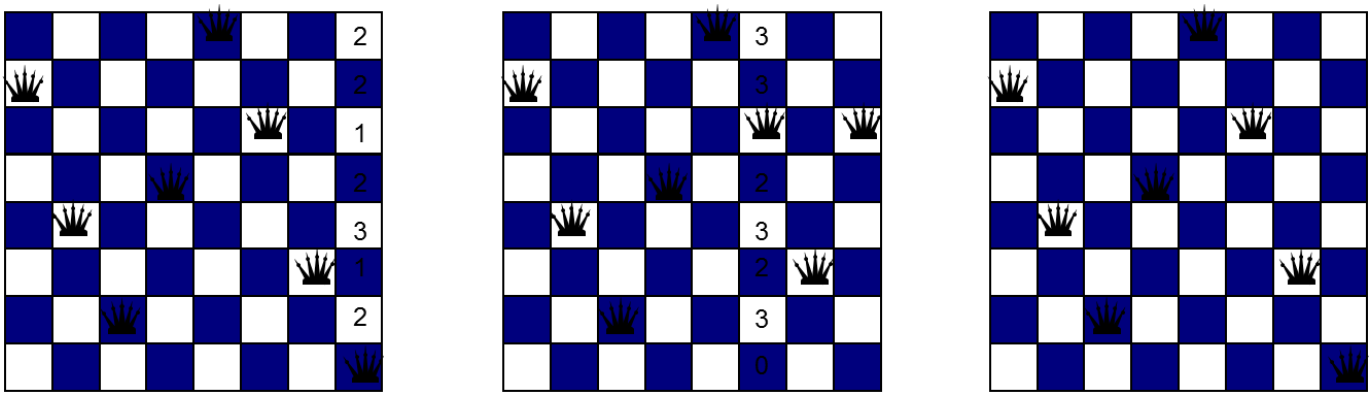
❖ Ridges: نوک کوه، دارای لبه‌های سراشیب است. بنابراین جستجو به بالای نوک کوه به آسانی می‌رسد، اما بعد با ملایمت به سمت قله می‌رود. مگر اینکه عملگرهایی موجود باشند که مستقیماً به سمت بالای نوک کوه حرکت کنند. جستجو ممکن است از لبه‌ای به لبه دیگر نوسان داشته باشد و پیشرفت کمی را حاصل شود.

در هر مورد، الگوریتم به نقطه‌ای می‌رسد که هیچ پیشرفتی نیست. اگر این اتفاق بیفتد، تنها کار ممکن برای انجام دادن آغاز مجدد از نقطه شروع دیگری دوباره آغاز می‌شود.

موفقیت hill-climbing خیلی به ظاهر فضای حالت «سطح» بستگی دارد: اگر فقط ماکزیمم‌های محلی کمی وجود داشته باشد، تپه‌نوردی با شروع تصادفی خیلی سریع راه‌حل خوبی را پیدا خواهد کرد.

۲- Simulated annealing

در این گروه از الگوریتم‌ها به جای شروع دوباره به طور تصادفی زمانی که در یک ماکزیمم محلی گیر افتادیم، می‌توانیم اجازه دهیم که جستجو چند قدم به طرف پائین بردارد تا از ماکزیمم محلی فرار کند.



راه‌حل دو مرحله‌ای برای مسئله ۸ وزیر با استفاده از حداقل برخوردها را نشان می‌دهد. در هر مرحله یک وزیر به منظور تعیین مجدد ستون انتخاب می‌شود. تعداد برخوردها (در این مورد، تعداد وزیرهای حمله‌کننده) در هر چهارخانه نشان داده شده است. الگوریتم وزیر را به چهارخانه‌ای که حداقل برخورد را داشته باشد، برای از بین بردن تصادفی برخوردها، حرکت می‌دهد.

اگر حرکت واقعاً شرایط را بهبود بخشد، آن حرکت همیشه اجرا می‌شود.

پارامترهای مؤثر به شرح زیر می‌باشند:

۱- ΔE: چگونگی ارزیابی.

۲- T: تعیین احتمال.

الگوریتم شباهت صریحی با annealing (پردازشی که به طور آهسته مایعی را تا زمانی که یخ ببندد سرد می‌کند)، گسترش یافته است. مقدار تابع مطابق با انرژی ورودی اتم‌های ماده است، و T با دما مطابقت دارد. جدول میزان دما را در جایی که پائین آمده است، تعیین می‌کند.

کاربردها در مسائل ارضا محدودیت

مسائل ارضاء محدودیت (CSP)، می‌توانند توسط روش‌های اصلاح تکراری با استفاده از موارد زیر حل شوند.

❖ مقدار دادن به تمام متغیرها.

❖ به کاربردن عملگرهای تغییر به منظور حرکت دادن ساختار به طرف یک راه‌حل.

الگوریتم‌هایی که CSPها را حل می‌کنند، روش‌های تصحیح کشف‌کنندگی، نامیده می‌شوند، زیرا آنها تناقضات را در ساختار جاری مسئله اصلاح می‌کنند.

در انتخاب مقدار جدید برای یک متغیر، واضح‌ترین کشف‌کنندگی انتخاب مقداری است که کمترین مقدار تناقضات را با دیگر متغیرها نتیجه دهد، که همان کشف‌کنندگی مینیمم تناقضات است.

فصل ۵
تئوری بازی

بازی‌ها در نقش مسائل جستجو

رقابت انتزاعی، که در بازی‌های صفحه‌ای دیده می‌شود، موجب شده تا تئوری بازی جزء تحقیقات AI قرار بگیرد.
وضعیت بازی برای بازنمایی آسان است و عامل‌ها معمولاً به تعداد کمی از عملیات محدود می‌شوند.
دلایلی که محققین قدیم، شطرنج را به‌عنوان موضوعی در AI برگزیدند:

- ❖ بازی شطرنج کامپیوتری اثباتی بر وجود ماشینی است که اعمال هوشمندانه‌ای را انجام می‌دهند.
- ❖ سادگی قوانین
- ❖ وضعیت دنیا کاملاً برای برنامه شناخته شده است. (بازنمایی بازی به عنوان یک جستجو از طریق فضای موقعیت‌های ممکن بازی، ساده است.)
- ❖ پیچیدگی بازی‌ها، به طور کامل نوعی از عدم قطعیت را معرفی می‌کنند.
- ❖ عدم قطعیت به علت وجود اطلاعات گم شده رخ نمی‌دهد، اما به علت اینکه فرد زمانی برای محاسبه دقیق نتایج حرکت ندارد عدم قطعیت بوجود می‌آید.
- ❖ در این مورد، فرد بر اساس تجربیات گذشته می‌تواند بهترین حدس را بزند.
- ❖ **تصمیمات کامل در بازی‌های دونفره:**
- ❖ مورد کلی از یک بازی با دو بازیکن را در نظر می‌گیریم که آن را MIN,MAX می‌نامیم.

یک بازی به طور رسمی می‌تواند به عنوان نوعی از مسئله جستجو به همراه قسمت‌های زیر تعریف شود:

- ❖ حالت اولیه شامل مکان صفحه وتعیین نوبت حرکت هر بازیکن است.
- ❖ مجموعه‌ای از عملگرها که حرکات صحیح را که بازیکن می‌تواند انجام دهد، تعیین می‌کند.
- ❖ آزمون پایانی زمان بازی را تعیین می‌کند. حالاتی را که بازی درآنها به پایان رسیده است حالات پایانی نامیده می‌شوند.
- ❖ تابع سودمندی (تابع امتیاز payoff) که مقدار عددی برای نتیجه بازی را تعیین می‌کند.

اگر به آن به عنوان یک مسئله جستجو نگاه شود، جستجو برای دنباله‌ای از حرکات که منتهی به حالت پایانی می‌شد (مطابق با تابع سودمندی)، و سپس پیشروی و ساخت اولین حرکت در دنباله بود.
اما حرکات MIN غیر قابل پیش‌بینی است؛
بنابراین:
MAX باید استراتژی‌ای را بیابد که به یک حالت پایانی برنده بدون توجه به عملکرد MIN منجر شود، که این استراتژی شامل حرکات درست برای MAX برای هر حرکت ممکن از MIN می‌باشد.
الگوریتم MINMAX به منظور تعیین استراتژی بهینه برای MAX طراحی شده است و از این رو می‌توان بهترین حرکت را تصمیم‌گیری کرد. الگوریتم شامل ۵ مرحله است:

۱. تولید درخت کامل بازی، تمام راه تا مراحل پایانی
۲. درخواست تابع سودمندی برای هر حالت پایانی به منظور بدست آوردن مقدارش.
۳. از سودمندی حالات پایانی به منظور تعیین سودمندی گره‌ها یک مرحله بالاتر دردرخت جستجو استفاده کنید.
۴. بررسی مقادیر را از گره‌های برگی تا ریشه، یک لایه در هر لحظه، ادامه دهید.
۵. احتمالاً مقادیر به بالای درخت می‌رسند، MAX حرکتی را انتخاب می‌کند که به بالاترین مقدار منتهی می‌شود.

اگر:

m: حداکثر عمق درخت،

b: تعداد حرکات قانونی در هر نقطه،

آنگاه:

زمان پیچیدگی الگوریتم minimax، $O(bm)$ است.

الگوریتم یک جستجو عمقی است.

تصمیمات ناقص:

الگوریتم minimax فرض می‌کند که برنامه زمان لازم برای جستجوی تمامی راه‌های ممکن وضعیت‌های پایانی را دارد که این فرض معمولاً عملی نیست.

الگوریتم مینی‌ماکس، به دو راه تغییر یابد:

- ❖ تابع سودمندی با تابع ارزیابی EVAL جایگزین شود.
- ❖ آزمون پایانی با آزمون قطع CUTOFF-TEST جایگزین گردد.

تابع ارزیابی:

تابع ارزیابی تخمینی از سودمندی مورد انتظار بازی را ازموقعیت داده شده برمی‌گرداند.

واضح است که ارائه یک برنامه بازی بی نهایت به کیفیت تابع ارزیابی بستگی دارد.

چگونه به طور دقیق کیفیت را می‌توان اندازه گرفت؟

۱. تابع ارزیابی با تابع سودمندی در مورد حالت پایانی باید به توافق برسند.

۲. نباید زیاد طول بکشد! (اگر پیچیدگی را محدود نکنیم minimax به عنوان یک زیربرنامه فراخوانی می‌شود و مقدار دقیق وضعیت محاسبه می‌شود.) از این رو، معامله‌ای بین صحت تابع ارزیابی و هزینه زمان آن وجود دارد.

۳. تابع ارزیابی باید به درستی شانس‌های واقعی برای برد را منعکس کند.

تابع ارزیابی فرض می‌گیرد که مقدار هر مهره می‌تواند به طور مستقل از دیگر مهره‌ها روی صفحه قضاوت شود. این نوع از تابع ارزیابی، تابع خطی وزن دار نامیده می‌شود.

این تابع می‌تواند به صورتی ذکر شود که Wها وزن ها هستند و fها اعداد هر نوع مهره روی صفحه خواهند بود.

قطع جستجو:

صریح‌ترین رهیافت برای کنترل میزان جستجو قراردادن محدودیتی برای داشتن یک عمق ثابت است، بنابراین تست قطع برای تمام گره‌ها در زیر عمق d موفق می‌شود. عمق طوری انتخاب می‌شود که میزان زمان استفاده شده از آنچه که قوانین بازی اجازه می‌دهد تجاوز نکند.

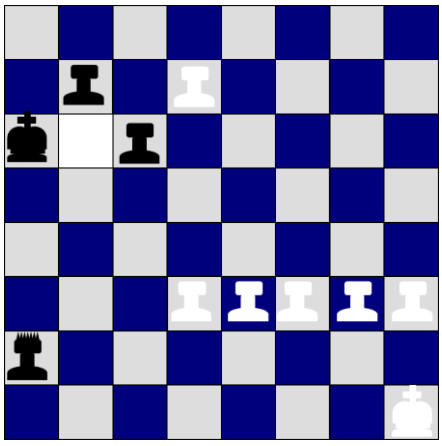
زمانی که، وقت تمام می‌شود، برنامه حرکت انتخابی توسط عمیق‌ترین جستجوی کامل شده را برمی‌گرداند.

به دلیل تخمینی بودن توابع ارزیابی این رهیافتها می‌توانند نتایج ناخوشایندی را به همراه داشته باشند.

تابع ارزیابی فقط باید برای مواقعی به کاربرده شود که خاموش هستند، یعنی اینکه تفاوتهای چشم‌گیر در مقدار، در آینده نزدیک بعید به نظر می‌رسد.

این جستجوی فوق العاده **جستجوی خاموش** نامیده می‌شود.

مسئله افقی (horizonproblem): رخ سیاه مانع از حرکت وزیر سفید به حالت افقی شده است و این موقعیت به نفع سیاه است. در حالی که برگ برنده در دست سفید است.



هرس آلفا-بتا:

هرس درخت جستجو:

پردازش حذف شاخه‌ای از درخت جستجو، با در نظر داشتن و بدون آزمایش، هرس درخت جستجو نامیده می‌شود.

زمانی که این تکنیک برای یک درخت minimax استاندارد، به کار برده می‌شود، حرکت مشابهی همانطور که minimax انجام می‌داد، برمی‌گرداند؛ اما شاخه‌هایی که در تصمیم نهایی دخالتی ندارند را هرس می‌کند.

جستجوی minimax عمقی است، بنابراین، در هر لحظه، باید گره‌هایی در نظر گرفته شوند که در طول یک مسیر مجزا در درخت هستند.

α مقدار بهترین انتخابی باشد که تا کنون در طول مسیر برای MAX پیدا شده است. و β مقدار بهترین (به طور مثال، پایین‌ترین مقدار) انتخابی باشد که در طول مسیر تا این لحظه برای MIN پیدا شده است.

درخت جستجوی آلفا-بتا:

این درخت، مقدار α و β را همچنانکه جلو می‌رود، به روز درمی‌آورد، و زیر درخت را هرس می‌کند (فراخوانی بازگشتی را قطع می‌کند) به محض اینکه معلوم می‌شود که این زیر درخت بدتر از مقدار α یا β جاری است.

مزایای هرس آلفا-بتا

مزایای آلفا-بتا به مرتبه‌ای که در آن گره‌های فرزندی آزمایش شده‌اند، برمی‌گردد.

پیچیدگی O(b/log b)d می‌باشد.

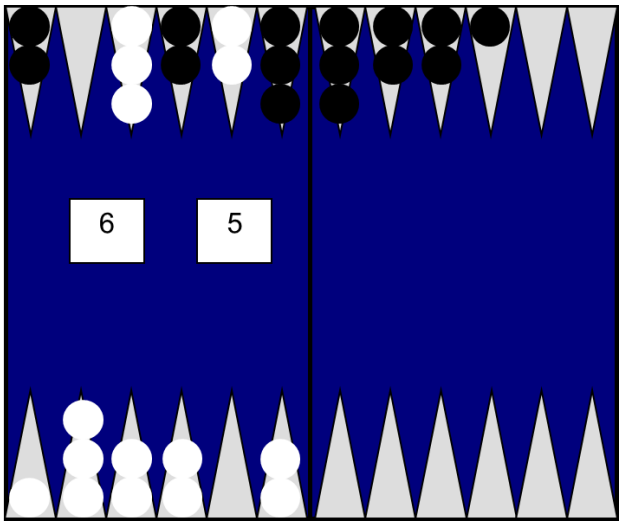
۱) در عمل، یک تابع ساده مرتب‌کننده شما را به نتیجه بهترین حالت بر خلاف نتیجه تصادفی سوق می‌دهد.

۲) رهیافت مشهور دیگر انجام جستجوی عمیق‌کننده تکراری و استفاده از مقادیر backed-up از یک تکرار برای تعیین ترتیب جانشین‌ها در تکرار بعدی است.

نتایج بازی‌ها نیز قابل ملاحظه هستند (و در حقیقت، مسائل جستجو در حالت کلی) و باید به صورت یک مدل درخت مطلوب فرض شوند تا نتایجشان را به دست آورند.

بازی‌هایی که شامل عنصر شانس هستند:

تخته نرد یک بازی عمومی است که شانس و مهارت را با هم ترکیب می‌کند.



تاس‌های سفید ۵-۶، چهار حرکت زیر را می‌تواند انجام دهد:

(۱۶-۱۰ و ۱۰-۵) و (۲۴-۱۹ و ۱۱-۵) و (۱۱-۵ و ۱۰-۵) و (۱۶-۱۱ و ۱۱-۵)

درخت بازی در تخته نرد باید شامل گره‌های شانس برای گره‌های MIN و MAX باشد.

مرحله بعدی فهم چگونگی ساخت تصمیمات صحیح است.

محاسبه مقادیر انتظاری گره‌ها، صریح است. برای گره‌های پایانی، از تابع سودمندی مانند بازیهای قطعی استفاده می‌کنیم.

با پیشروی در درخت جستجو به اندازه یک مرحله، به یک گره شانس برخورد می‌کنیم.
اگر ما فرض کنیم که $S(C,d_i)$ مجموعه موقعیت‌های تولید شده توسط اعمال حرکات قانونی برای پرتاب $P(d_i)$ در موقعیت C باشد، می‌تواند مقدار expectimax از C را با استفاده از فرمول زیر محاسبه نمود:

$$\text{Expectimax}(c)=\sum_i P(d_i)\ \max_{s\varepsilon} S(c,d_i)\ (\text{utility}(s))$$

این فرمول، سودمندی مورد انتظار در موقعیت c را با فرض بهترین بازی ارائه می‌دهد.

ارزیابی موقعیت در بازی‌ها با گره‌های شانس:

حضور گره‌های شانس بدین معناست که باید در مورد آنچه که به معنای مقادیر ارزیابی است، دقیق بود.
اگر ما تغییری را در مقیاس مقادیر ارزیابی ایجاد کنیم، برنامه در مجموع به طور متفاوت رفتار می‌کند.

پیچیدگی:

بدلیل اینکه expectiminimax تمام دنباله‌های پرتاب تاس را در نظر می‌گیرد، زمانی معادل $O(bmnm)$ می‌برد، که n تعداد پرتابهای محدود است.
مزیت آلفا- بتا، با داشتن بهترین بازی نادیده گرفتن پیشرفت‌ها در آینده است که احتمال وقوعشان کم است.
در بازیهای به همراه تاس، دنباله‌های محتملی از حرکات وجود ندارد، چون برای آن حرکاتی که باید انجام بگیرند، ابتدا تاس باید به روش درستی پرتاب شود تا آن حرکات منطقی شوند.
اگر بگوئیم که تمام مقادیر سودمندی بین ۱+ و ۱- هستند، سپس مقدار گره‌های برگی محدود می‌شوند و در عوض ما می‌توانیم حد بالایی روی مقدار گره شانسی بدون توجه به فرزندانش قرار دهیم.

فصل ششم

عامل‌هاییکه به طور منطقی استدلال می‌کنند

معرفی طراحی پایه‌ای برای یک عامل مبتنی بر دانش:

رهیافت مبتنی بر دانش روش قدرتمندی از ساخت برنامه عامل است. هدف آن پیاده‌سازی نمایی از عامل است که بتواند به عنوان دانش در مورد دنیای آنها و استدلال در مورد گونه‌هایی ممکن از رفتار آنها به کار می‌رود.

عامل‌های مبتنی بر دانش قادرند که:

۱. وظایف جدید را به صورت اهداف تعریف شده صریح قبول کنند.
۲. آنها می‌توانند به سرعت توسط گفتن یا یادگیری دانش جدید درمورد حیطه، به رقابت برسند.
۳. آنها می‌توانند خود شانس را با تغییرات محیط، توسط به روز در آوردن دانش مربوطه، تطبیق دهند.

عامل مبتنی بر دانش به موارد زیر نیاز دارد:

۱) چه چیزهایی را بداند؟
۲) وضعیت جاری دنیا؟
۳) چطور توسط ادراک به خواص نادیده دنیا رجوع کند؟
۴) چطور دنیا زمان را می‌گشاید؟
۵) عامل به چیزی می‌خواهد برسد؟
۶) فعالیت‌هایی که در شرایط مختلف انجام می‌دهد چیست؟

بخش مرکزی عامل مبتنی بر دانش پایگاه دانش (knowledge base) آن، یا KB است.

پایگاه دانش: مجموعه‌ای از نمایش حقایق در مورد نیاز است.

جمله: هر نمایش اختصاصی یک جمله (sentence) نامیده می‌شود.

جملات: جملات در یک زبانی که زبان بازنمایی دانش (knowledge representation) نامیده می‌شود، بیان می‌شوند.

ASK: به منظور افزودن جملات جدید به پایگاه دانش به کار برده می‌شود.

TELL: به منظور پرسش اینکه چه چیزهایی شناخته شده است.

تشخیص اینکه چه چیزی باید پس از TELLed به KB دنبال شود، مسئولیت مکانیزمی به نام استنتاج (inference) است، که قسمت مهم دیگر عامل مبتنی بر دانش را تشکیل می‌دهد.

هر زمان که برنامه دانش صدا زده می‌شود، دو عمل انجام می‌شود:

۱. به پایگاه دانش گفته می‌شود (TELL) که چه دریافت کرده است.
۲. از پایگاه دانش سؤال می‌شود (ASK) که چه عملی باید انجام شود.

در فرآیند پاسخ به این پرسش، استدلال منطقی برای اثبات اینکه کدام عمل بهتر از بقیه است استفاده می‌شود و دانسته‌های عامل و اهداف آن مشخص می‌شوند.

می‌توانیم یک عامل مبتنی بر دانش را در سه سطح تعریف کنیم:

۱. سطح دانش knowledge level یا سطح epistemological که خلاصه‌ترین سطح است؛ می‌توانیم عامل را توسط گفتن اینکه عامل چه می‌داند، تعریف نماییم.
۲. سطح منطقی logical level سطحی است که دانش به صورت جملات رمزگذاری می‌شود.
۳. سطح پیاده سازی Implementation Level سطحی است که در معماری عامل اجرا می‌شود و بازنمایی‌های فیزیکی از جملات سطح منطقی، در این سطح وجود دارد.

انتخاب پیاده‌سازی در کارآیی بهتر عامل بسیار اهمیت دارد، اما به سطح منطقی و سطح دانش مربوط نمی‌شود.

دنیای WUMPUS:

مشابه دنیای مکش، دنیای Wumpus شبکه‌ای از مربع است که توسط دیوارهایی احاطه شده‌اند، که هر مربع می‌تواند شامل عامل‌ها و اشیاء باشد.
وظیفه عامل یافتن طلا و بازگشتن به نقطه شروع و بالا رفتن از غار است.
برای مشخص نمودن وظیفه عامل، ادراکات، عملیات و اهداف آن را باید مشخص کنیم. در دنیای Wumpus، اینها به صورت زیر هستند:

- از مربعی که شامل Wumpus است و مربع‌های مجاور (نه قطری) عامل بوی بدی را دریافت می‌کند .
- در مربعهایی که مستقیماً مجاور با چاله‌ها هستند، عامل نسیمی را دریافت می‌کند.
- در مربعی که طلا وجود دارد، عامل یک درخششی را درک می‌کند.
- زمانی که یک عامل به داخل دیواره قدم بر می‌دارد، ضربه‌ای را دریافت می‌کند.
- زمانی که Wumpus کشته می‌شود، فریادی سر می‌دهد که هر جایی از غار شنیده می‌شود.
- ادراکات به عامل به صورت لیستی از پنج سیمبول داده می‌شود.
- مانند دنیای مکش، عملایتی برای جلو رفتن، چرخیدن 90 به سمت چپ، چرخیدن ۹۰ به سمت راست وجود دارد.
- عامل نابود خواهد شد زمانی که وارد یک مربع شامل سیاده چاله و یا کی Wumpus زنده می‌شود.
- هدف عامل یافتن طلا و برگرداندن آن به خانه شروع با سرعت تمام است، بدون آنکه کشته شود.

بازنمایی، استدلال و منطق:

بازنمایی و استدلال با همدیگر، عملکرد یک عامل مبتنی بر دانش را حمایت خواهند کرد.
بازنمایی دانش (knowledge representation) دانش را در فرم حل شدنی کامپیوتر مطرح می‌سازد، که به عامل‌ها کمک می‌کند تا ارائه بهتری داشته باشند.

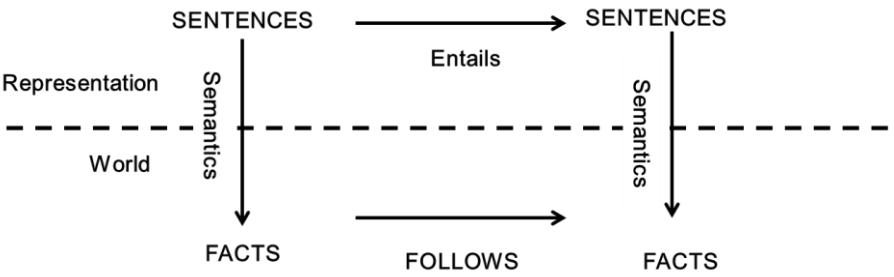
زبان بازنمایی دانش متوسط دو خاصیت تعریف می‌شود:

نحو (Syntax): یک زبان ساختاری ممکن برای تشکیل جملات را ایجاد می‌کند.
بازنمایی واقعی در داخل کامپیوتر: هر جمله توسط یک ساختار فیزیکی یا خاصیت فیزیکی قسمتی از عامل پیاده‌سازی می‌شود.
معنی (Semantic): تعیین می‌کند که حقایق موجود در دنیا به چه جملاتی نسبت داده شوند.
با Semanticها، می‌توانیم بگوییم زمانی که ساختار ویژه با یک عامل وجود دارد، عامل به جملات مربوطه، اعتقاد دارد.
معنی‌های زبان تعیین می‌کند که حقایق به کدام جملات مربوط می‌شوند.

تفاوت بین حقایق و بازنمایی‌های آنها:

حقایق قسمتی از دنیای واقعی را تشکیل می‌دهند، اما بازنمایی‌های آنها باید به صورتی کد شوند که بتواند به طور فیزیکی در یک عامل ذخیره شود.
جملات قسمتی از ساختار فیزیکی عامل هستند و استدلال باید پردازشی از ایجاد ساختار جدید فیزیکی از نمونه‌های قدیمی‌تر باشد.

استدلال مطلوب باید این اطمینان را حاصل کند که ساختار جدید حقایقی را بازنمایی می‌کند که از حقایقی که ساختار قدیمی ایجاد کرده بود، پیروی کنند.



استلزام:

ارتباط بین حقایقی که دنباله رو یکدیگر هستند را نشان می‌دهد.

در علائم ریاضی، ارتباط استلزام بین یک پایگاه دانش KB و یک جمله a به صورت «KB مستلزم a است» تلفظ می‌شود و به صورت KB |= a نوشته می‌شود.

رویه استنتاج می‌تواند یکی از دو عامل ذیل را انجام دهد:

۱. با داشتن پایگاه دانش KB می‌تواند جملات تازه‌ای از a تولید کند که مفهوم آن استلزام توسط KB باشد.

۲. یا با داشتن یک پایگاه دانش KB و جمله a دیگری، این رویه می‌تواند گزارش دهد که a توسط KB مستلزم شده است یا خیر.

رویه استنتاج i می‌تواند توسط جملاتی که آنها را مشتق می‌کند، تعریف شود. اگر i بتواند a را از KB مشتق کند، منطق‌دان می‌تواند بنویسید: KB |_i a که خوانده می‌شود «آلفا از KB توسط i مشتق شده است یا «i مشتق می‌کند آلفا از KB».

ثبت عملیات رویه استنتاج صحیح، اثبات (Proof) نامیده می‌شود.

کلید استنتاج صحیح:

داشتن مراحل استنتاج است که به جملات مورد عمل قرار گرفته، توجه داشته باشد.

بازنمایی:

زبان‌های برنامه‌نویسی (مانند C یا پاسکال یا Lips) برای تعریف الگوریتم‌ها مناسب هستند و بین ساختارهای داده پیوستگی ایجاد می‌کنند.

زبان‌های طبیعی بیش‌تر محتاج محاوره بر خلاف بازنمایی هستند.

مزایا و معایب زبان طبیعی:

زبان طبیعی راهی خوب برای سخنگو است تا مخاطب را متوجه منظور خود سازد؛ اما اغلب این تقسیم دانش بدون بازنمایی صریح خود دانش انجام می‌شود. زبان‌های طبیعی هم چنین از ابهامات رنج می‌برند، مانند عبارت «سگ‌ها و گربه‌های کوچک»، روشن نیست که آیا سگ‌ها نیز کوچک هستند یا خیر.

یک زبان بازنمایی خوب می‌بایست:

مزایای زبان‌های طبیعی و رسمی را با هم داشته باشد.

پرمعنی و رسا باشد.

دقیق و غیر مبهم

مستقل از متن

قابل استنتاج

معانی:

یک جمله خودش به تنهای معنایی ندارد.
می‌توان زبانی را تعریف نمود که در آن هر جمله یک تفسیر اختیاری داشته باشد. اما در عمل تمام زبان‌های بازنمایی ارتباط سیستماتیکی بین جملات اعمال می‌کنند.

صدق‌پذیری:

یک جمله معتبر (Valid) یا لزوماً صحیح است اگر و فقط اگر تحت تمام تفسیرهای ممکن در تمام دنیای ممکن، بدون توجه از آنچه که تصور می‌شد که معنا دهد و بدون توجه به حالت آن مطلب در کل، تعریف شده باشد.
یک جمله صدق‌پذیر (satisfiable) است اگر و فقط اگر تفسیری در دنیایی برای صحت آن وجود داشته باشد. جمله در خانه Wumpus [1,2] وجود دارد " Satisfiable است زیرا امکان دارد که Wumpus در آن خانه باشد، حتی اگر چنین اتفاقی نیفتاده باشد، جمله‌ای که صدق‌پذیر نباشد صدق ناپذیر (unsatisfiable) است.
جملات خود تناقضی صدق‌ناپذیر هستند، اگر تناقض به معنای سیمبول‌ها بستگی نداشته باشد.

استنتاج در کامپیوترها:

معتبر بودن و صدق ناپذیری به قابلیت کامپیوتری که استدلال می‌کند، بستگی دارد.

کامپیوترها از دور نقطه ضعف رنج می‌برند:

کامپیوتر لزوماً تفسیری را که شما برای جملات در پایگاه دانش به کار می‌بردید، نمی‌داند.
چیزی در مورد دنیا نمی‌داند به جز آنچه که در پایگاه دانش ظاهر می‌شود.
چیزی که استنتاج رسمی را قدرت می‌بخشد، نبودن محدودیت بر روی پیچیدگی جملاتی است که کامپیوتر باید آنها را مورد عمل قرار دهد.
بزرگترین چیز در مورد استنتاج رسمی، قابلیت آن برای بدست آوردن نتایج صحیح است حتی زمانی که کامپیوتر اطلاعی از تفسیر استفاده شده توسط شما نداشته باشد.
کامپیوتر فقط نتایج معتبر را گزارش می‌کند، که بایست بدون توجه به تفسیر شما، صحیح باشد.

منطق شامل موارد زیر می‌شود:

۱- یک سیستم رسمی برای تعریف حالتهای مطلب که شامل:
الف- نحو (syntax) زبان، که روش درست کردن جملات را شرح می‌دهد.
ب- معانی (semantic) زبان، که محدودیت‌های سیستماتیکی را روی چگونگی ارتباط جملات با حالات موضوع قرار می‌دهند.
۲- تئوری اثبات- مجموعه‌ای از قوانین برای استنباط استلزامی یک سری از جملات.
ما روی دو نوع منطق تمرکز خواهیم کرد:

- منطق بولین یا گزاره‌ای،
- منطق مرتبه اول (دقیق تر بگوییم، حساب گزاره مرتبه اول با تساوی.)

✓ در منطق گزاره‌ای سیمبول‌ها تمام گزاره‌ها را بازنمایی می‌کنند.
✓ سیمبولهای گزاره‌ای می‌توانند با استفاه از ربط‌دهنده‌های بولین (Boolin connevtives) جملات را با معناهای پیچیده‌ترین تولید کنند.
منطق مرتبه اول با بازنمایی دنیاهایی به نام اشیاء (objects) و گزاره ها روی اشیاء (به عنوان مثال، خواص اشیاء یا ارتباط بین اشیاء)، به خوبی استفاده از ربط دهنده ها و سورها (quantifiers)، به جملات اجازه می‌دهند تا در مورد چیزی در دنیا به سرعت نوشته شوند.
در منطق مرتبه اول گزاره‌ای یک جمله یک حقیقت را بیان می‌کند و عامل باور دارد که جمله صحیح است، یا جمله نادرست است یا قادر نیست تا از راه دیگری تنیجه‌گیری کند.

سیستم‌هایی که مبتنی بر منطق شولا (Fuzzy) هستند، می‌توانند در جایی از اعتقاد را در یک جمله داشته باشند و هم‌چنین به درجات حقیقت نیز اجاره دهند: یک حقیقت نیازی به درست یا نادرست بودن در دنیا ندارد، اما می تواند تا یک میزانی صحت داشته باشد.

منطق گزاره‌ای: یک منطق بسیار ساده:

علائم منطق گزاره‌ای:

ثابتهای منطقی (true, False)

علائم گزاره‌ای: Q, P

رابطهای $\neg, \vee, \wedge, \Rightarrow, \Leftrightarrow$

پرانتز ()

تمام جملات توسط قرار دادن این علائم با هم و با استفاده از قوانین زیر، ساخته می‌شوند:

- ثابتهای منطقی (true, False) خودشان جمله محسوب می‌شوند.
- علامات گزاره‌ای نظیر Q, P هر کدام به تنهایی یک جمله هستند.
- پرانتزهای اطراف یک عبارت، آن عبارت را تبدیل به یک جمله واحد می‌سازند مثل $(P \wedge Q)$.
- یک جمله می‌تواند توسط ترکیب جملات ساده‌تر با یکی از پنج رابط منطقی ایجاد می‌شود.

روش رفع ابهام منطق گزاره‌ای بسیار شبیه عبارت ریاضی است.

معانی:

یک سیمبول گزاره‌ای می‌تواند آنچه که خواست شما است، معنی بدهد. یعنی اینکه، تفسیر آن هر حقیقت اختیاری می‌تواند باشد.

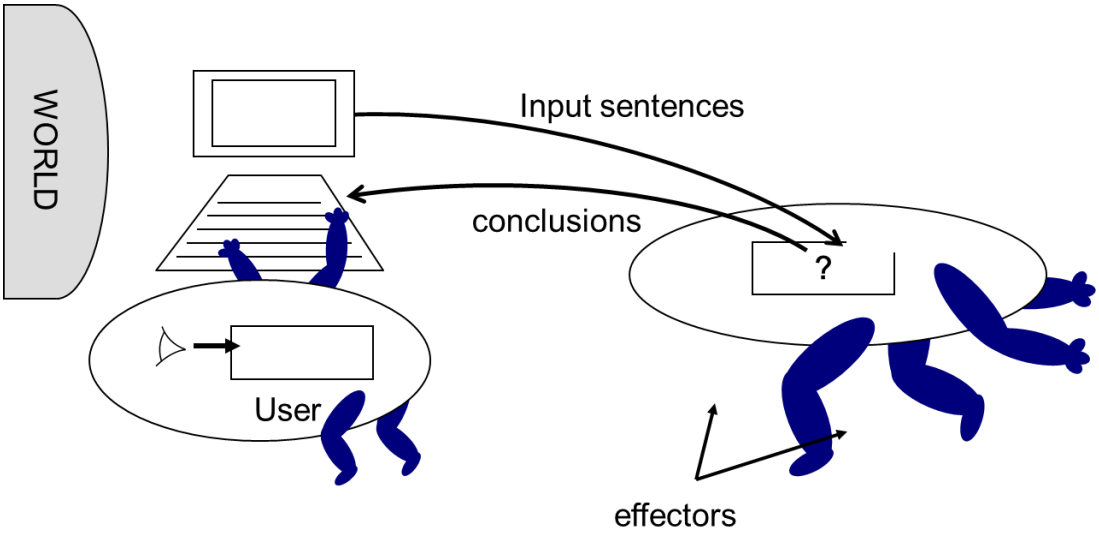
یک جمله پیچیده، معنایی مرکب از معناهای هر قسمت از جمله را دارد، هر رابط می‌تواند به عنوان یک تابع تصور شود.

اعتبار و استنتاج:

جدول درستی برای تعریف رابطها و برای کنترل جملات معتبر به کار می‌رود.

ماشین هیچ ایده‌ای از معنای نتایج ندارد، کاربر می‌تواند نتایج را بخواند و از تفسیر خود برای سیمبولهای گزاره‌ای به معنای نتیجه پی ببرد.

وجود یک یک سیستم استدلال ضروری است تا قادر باشد، نتایجی را استخراج کند که از مقدمه‌ها، بدون توجه به دنیا که اولویت رجوع جملات را مشخص می‌کند، پیروی کنند.



جملات اغلب به دنیایی رجوع می‌کنند که عامل دسترسی مستقلی به آن نداشته باشد.

مدل‌ها Models:

دنیایی که در آن جمله‌ای تحت تفسیری ویژه، درست باشد. یک مدل (Model) از آن جمله نامیده می‌شود.

مدل‌ها در منطق بسیار حائز اهمیت هستند زیرا، دوباره استلزام را مطرح می‌کنند، جمله α توسط یک پایگاه دانش KB مستلزم می‌شود، اگر مدل‌های KB تمام مدلهای α باشند. سپس زمانی که KB درست باشد، α نیز درست خواهد بود.

«دنیاهای واقعی» متفاوت بسیاری وجود دارند که مقادیر درستی مشابهی برای آن سیمبول‌ها دارند. تنها تقاضایی که برای کامل شدن تصفیه لازم است، درستی یا نادرستی هر سیمبول گزاره‌ای در هر دنیا است

قوانین استنتاج برای منطق گزاره‌ای:

پردازشی که توسط هر کدام از آنها، صحت یک استنباط از طریق جداول درستی بدست آمده است، می‌توند به کلاس‌های استنتاج‌ها گسترش داده شود.

نمونه‌های مطمئنی از استنتاج‌ها وجود دارند. که بیشتر و بیش‌تر بوجود می‌آیند، و صحت آنها می‌تواند یکبار برای همیشه نشان داده شوند.

زمانی که یک قانون پیاده شد، می‌توان به منظور ساخت استنتاج‌ها بدون ساخت جداول درستی، استفاده شود.

$\frac{\alpha}{\beta}$ ⟵ این جمله نیست، اما یک قانون استنتاج است.

یک قانون استنتاج زمانی درست است اگر نتیجه آن در تمام موارد درست باشد و مقدم‌ها نیز درست باشند.

یک اثبات منطقی شامل دنباله‌ای از کاربردهای قوانین استنتاج است که ابتدا با جمله‌های موجود در KB آغاز می‌شود، و منجر به تولید جمله‌ای می‌شود که اثبات را پایان می‌دهد.

یکنوایی:

استفاده قوانین استنتاج به منظور یافتن نتیجه از یک پایگاه دانش، به طور صریح مبتنی بر خواص عمومی منطقهای قطعی (شامل گزاره‌ای و منطق مرتبه اول) است که یکنوایی (monotonicity) نامیده می‌شود.

می‌توانیم خواص یکنوایی منطق را به طور زیر شرح دهیم:

$$if\ KB_1 \models_{\alpha}\ then\ (KB_1 \cup KB_2) \models \alpha$$

منطق مرتبه اول وگزاره‌ای دراین حالت، یکنوا هستند.

تئوری احتمال، یکنوا نیست.

کلاس مفیدی از جملات برای زمانی که رویه استنتاجی با زمان چند جمله‌ای وجود دارد که این کلاس جملات هورن (Horn sentences) نامیده می‌شود. یک جمله هورن فرمی به صورت زیر دارد:

$$P_1 \wedge P_2 \ \ldots \wedge P_n \Rightarrow Q$$

که P_i و Q اتمهای خنثی هستند. دو مورد مهم وجود دارد: اول، زمانی که Q ثابت False است.

ما به جمله‌ای می‌رسیم که برابر است با:

$$\neg P_1 \vee \ldots \vee \neg P_n$$

دوم اینکه، زمانی که n=1 و P_1=True ما به True=>Q می‌رسیم که برابر است با جمله اتمی Q.

منطق گزاره‌ای به ما اجازه می‌دهد که به تمام نکات مهم درمورد منطق و چگونگی استفاده از آن به منظور ارائه استنتاج که نهایتاً به عملیات تبدیل می‌شود، برسیم. اما منطق گزاره‌ای بسیار ضعیف است.

مشکل کند شدن رویه استنتاج:

۱) مشکل فقط نوشتن این قوانین نیست بلکه تعداد زیاد آنها، باعث مشکل می‌شود.

۲) مشکل دیگر، روبرو شدن با تغییرات محیط است. ما جزیی از عامل استدلال کننده را در یک مکان و زمان ویژه نشان دادیم، و تمام گزاره‌ها در پایگاه دانش در آن زمان خاص، درست بودند. اما در حالت کلی، دنیا هر لحظه در حال تغییر است.

اندازه یک جدول درستی 2^n است. که n تعداد سیمبولهای گزاره‌ای در پایگاه دانش است.

برای اجتناب از سردرگمی، ما به سیمبول‌های گزاره‌ای متفاوتی، برای تشخیص مکان عامل در هر مرحله نیازداریم.

۱- ما نمی‌دانیم که بازی چه مدت طول خواهد کشید، بنابراین نمی‌دانیم که چه تعداد از این گزاره‌های وابسته به زمان، نیاز داریم.

۲- اکنون باید برگردیم و حالتهای وابسته به زمان از هر قانون را بنویسیم.

فصل هفتم

منطق مرتبه اول

منطق گزاره‌ای هستی شناسی بسیارمحدودی دارد و فقط برای دنیایی که شامل حقایق باشد، تعهد قبول می‌کند و این امر بازنمایی مسائل ساده را نیز مشکل ساخته است.

منطق مرتبه اول (First-Order_logic) تعهدات هستی شناسانه قوی‌تری را نسبت به منطق گزاره‌ای ایجاد می‌کند.

اجزایی که در این منطق وجود دارند:

اشیاء(Objects): مردم، خانه‌ها، اعداد، تئوریها، رنگها، بازیهای بیس‌بال، جنگلها، کشورها...

روابط(Relations): برادرِ، بزرگتر از، داخل، قسمتی از، رنگ ...دارد، بدهکار است، اتفاق افتاد بعد از...

خواص(Properties): قرمز، گرد: غیرواقعی، رسمی...

توابع(Functions): پدرِ، بهترین دوست، یکی بیشتر از، نوبت سوم...

ما ادعا نمی‌کنیم که دنیا واقعاً از اشیاء و روابط بین آنها ساخته شده است، بلکه این جداسازی به ما کمک می‌کند با بهتر در مورد دنیا قضاوت کنیم.

- ❖ منطق مرتبه اول قادر است تا حقایقی را در مورد تمام اشیاء جهان بیان دارد.
- ❖ اگرچه منطق مرتبه اول، موجودیت اشیاء و روابط آنها را ممکن می‌سازد، اما هیچ تعهد هستی‌شناسی را برای چیزهایی مثل طبقات، زمان و حوادث قبول نمی‌کند.
- ❖ منطق مرتبه اول از این نظر جهانی است که قادر است تا هر چیزی را که قابل برنامه‌ریزی باشد، بیان کند.

نحو و معانی:

منطق مرتبه اول جملاتی دارد، اما همچنین واژه‌هایی term نیز دارد که اشیاء را بازنمایی می‌کنند.

سیمبول‌های ثابت، متغیرها و سیمبول‌های تابع برای ساخت واژه‌ها استفاده می‌شوند، و کمیت‌سنجها و سیمبولهای گزاره‌ای برای ساخت جملات به کار برده می‌شوند.

تعریف دقیق هر عنصر به صورت زیر است:

سیمبولهای ثابت (Constant Symbols):

یک تفسیر می‌بایست معین کند که کدام شیء توسط کدام سیمبول ثابت در اشیاء ارجاع داده می‌شود.

هر سیمبول ثابت، دقیقاً به اسم یک شیء نامگذاری می‌شود، اما تمام اشیاء نیازی به داشتن نام ندارند و بعضی از آنها می‌توانند چند اسم داشته باشند.

سیمبولهای گزاره (Predicate Symbols):

یک تفسیر معین می‌کند که یک سیمبول گزاره به یک رابطه ویژه درمدل رجوع می‌کند.

سیمبولهای تابع (Function Symbols):

بعضی از روابط تابع هستند، بدین معنا که هر شیئ دقیقاً به شیئ دیگری توسط رابطه رجوع می‌کند.

انتخاب **ثابت، گزاره،** و **سیمبولهای تابع** به کلی به **کاربرد** بستگی دارد.

ترم‌ها (Terms):

یک ترم، یک عبارت منطقی است که به یک شیئ رجوع می‌کند.

معانی رسمی ترم‌ها بسیار صریح است. تفسیر، یک رابطه تابعی ارجاع داده شده توسط سیمبول تابع، و اشیاء ارجاع داده شده توسط واژه‌ها را اختصاص می‌دهد که آرگومان‌هایش هستند. از این رو، تمام ترم به شیئ رجوع می‌کند که به عنوان (n+1) امین مدخل در آن tuple در رابطه‌ای که اولین n عنصر آن اشیاء ارجاع شده توسط آرگومان‌ها هستند، ظاهر می‌شود.

جملات اتمی (Atomic sentences):

می‌توانیم با استفاده از ترم‌هایی برای ارجاع به اشیاء و گزاره‌هایی برای ارجاع به روابط، جملات اتمی به وجود آوریم، که حقایق را پایه‌گذاری می‌کنند.

یک جمله اتمی از یک **سیمبول گزاره‌ای** تشکیل یافته و توسط یک **لیست پرانتز از واژه‌ها** دنبال می‌شود.

یک جمله اتمی درست است **اگر** رابطه ارجاع شده توسط سیمبول گزاره با اشیاء ارجاع شده توسط آرگومان‌ها مطابقت داشته باشد.

رابطه در صورتی صحت دارد که tuple اشیاء در رابطه باشد.

حقیقت یک جمله بنابراین هم به تفسیر و هم به دنیا بستگی دارد.

جملات پیچیده:

ما می‌توانیم از رابط‌های منطقی برای تشکیل جملات پیچیده‌تر فقط در محاسبات گزاره‌ای استفاده کنیم.

معانی جملات که با استفاده از رابطهای منطقی فرم گرفته‌اند، ازلحاظ گزاره‌ای با آن یکسان هستند.

سورها (Quantifires):

زمانی که ما منطقی در اختیار داریم که شامل اشیاء است، طبیعی است که ذکر خواص کلی اشیاء را بر شمارش اشیاء توسط نام ترجیح می‌دهیم. سورها به ما اجازه این کار را می‌دهند.

منطق مرتبه اول دو سور استاندارد دارد:

- عمومی (universal)
- وجودی (existential)

سور عمومی: (Universal Quantification)

$\forall$ معمولاً به معنی «برای تمام» است.

شما یک جمله را می‌توانید به صورت $\forall x P$ که P یک عبارت منطقی است تصور کنید. و P معادل با ترکیب عطفی $(\wedge)$ تمام جملات حاصل شده توسط جانشینی نام یک شیئ برای متغیر x هرجا که در P ظاهر شود، است.

سور وجودی (Existential):

$\exists$ به صورت «وجود دارد...» تلفظ می‌شود. درحالت کلی $\exists x P$ زمانی درست است که P برای بعضی از اشیاء در دنیا درست باشد. بنابراین می‌تواند به عنوان معادلی برای ترکیب فصلی جملات بدست آمده توسط جانشینی اسم یک اشیاء برای متغیر x، تصور شود.

بنابراین، یک جمله شرطی با سور وجودی در دنیایی شامل هر شیئ که مقدم آن ترکیب شرطی نادرست باشد، درست است. از این رو همچنین جملاتی اصلاً چیزی برای گفتن ندارند.

سورهای لانه‌ای (Nested Quantifiers):

x , y معادل با x و y است

ترتیب سورها بسیار مهم است. اگر ما آنها را در پرانتز قرار دهیم روشن‌تر می‌شود.

در حالت کلی، $x(\ y \ P(x,y))$ جمله دلخواهی است که شامل x,y می‌باشد. می‌گوید که هر شیئی در دنیا یک خاصیت ویژه‌ای دارد، و آن خاصیت به چند شیئی توسط رابطه p مربوط می‌شود.

از طرف دیگر، $x(\ y \ P(x,y))$ می‌گوید که در دنیا شیئی وجود دارد که خاصیت ویژه‌ای دارد و خاصیت توسط p به هر شیئی در دنیا مربوط می‌شود

مشکل اساسی زمانی بوجود میآید ، که دو سور با یک متغیر استفاده می‌شوند.

قانون این است که متغیر به داخلی‌ترین سور که آن را بیان می‌کند، پس این متغیر ارتباطی با دیگر سورها نخواهد داشت.

ارتباط بین و

در واقع دو سور وجودی و عمومی از طریق تناقض با هم در ارتباط هستند.
بدلیل اینکه در واقع رابط عاطفی در دنیای اشیاء است و رابط فصلی است، تعجب آور نخواهد بود که آنها از قوانین دمورگان پیروی کنند. قوانین دمورگان در ارتباط با جملات سوری به شرح زیر است:

$$\forall x \neg P \equiv \neg \exists x P \qquad \neg P \wedge \neg Q \equiv \neg (P \vee Q)$$
$$\neg \forall x P \equiv \exists x \neg P \qquad \neg (P \wedge Q) \equiv \neg P \vee \neg Q$$
$$\forall x P \equiv \neg \exists x \neg P \qquad P \wedge Q \equiv \neg (\neg P \vee \neg Q)$$
$$\exists x P \equiv \neg \forall x \neg P \qquad P \vee Q \equiv \neg (\neg P \wedge \neg Q)$$

برای اهداف AI، **محتوا** و از این رو **قابلیت خواندن جملات** مهم هستند.

بنابراین:

ما هر دو سور را نگه می‌داریم.

تساوی (Equality):

به غیر از گزاره‌ها و ترم‌هایی که قبلاً به آنها اشاره می‌توانیم از سیمبول تساوی (equality symbol) برای ساختن عباراتی که دو ترم به شیئی مشابه رجوع کنند، استفاده می‌کنیم.

سیمبول تساوی : می‌تواند به منظور شرح خواص یک تابع داده شده، استفاده شود. این سمبول هم چنین می‌تواند با علامت نقیض برای نشان دادن عدم تشابه دو شیئی استفاده شود.

توسعه‌ها و تمایزات نگارشی:

سه نوع از روشهای که روی منطق مرتبه اول اعمال می‌شود:

۱- منطق مرتبه بالاتر

۲-۱ عبارات تابعی و گزاره‌ای با استفاده از عملگر λ

۲-۲ سور یکتایی

۲-۳ عملگر یکتایی

۳- انواع علائم

منطق مرتبه بالاتر:

- ما را قادر می‌سازد تا بتوانیم کیفیت روابط و توابع اشیاء را به خوبی تعیین کنیم.
- قدرت معنا دارتری نسبت به منطق اول دارد.

عبارات تابعی و گزاره‌ای با استفاده از عملگر λ :

- اغلب مفید است که توابع و گزاره‌های پیچیده را از قسمت های ساده‌تری تشکیل دهیم.
- عملگر λ مرسوم است که برای این منظور استفاده شود.
- این λ– expression می‌تواند برای آرگومان‌ها نیز به کار برده شود تا به یک ترم منطقی منتهی شود.

برای مثال گزارهٔ «از جنیست متفاوت و از آدرس مشابه هستند.» را می‌تواند به صورت زیر نوشت:

$$\lambda x, y \; Gender(x) \neq gender(y) \wedge Address(x) = Address(y)$$

سور یکتایی:

راه دقیقی برای گفتن اینکه یک شیئی منحصر به فرد یک گزاره را قانع می‌کند، وجود ندارد. بعضی از مؤلفان علامت ! x King(x) را استفاده می‌کنند.

جمله بالا بدین معناست که «یک شیئی منحصر به فرد x وجود دارد که King(x) را قانع می‌کند «یا غیر رسمی تر بگوییم» دقیقاً یک King وجود دارد.

عملگر یکتایی:

برای مفهوم یکتایی استفاده می‌کنیم.

علامت ! xp(x) عموماً برای بازنمایی مستقیم شیئی مورد نظر استفاده می‌شود.

انواع علائم:

تعدادی از علائم رایج در منطق مرتبه اول:

Syntax item	This book	Others
Negation (not)	$\neg P$	$\sim P\ \overline{P}$
Conjunction (and)	$P \wedge Q$	$P \& Q\ \ P.Q\ \ PQ\ \ P,Q$
Disjunction (or)	$P \vee Q$	$P \mid Q\ \ P;Q\ \ P+Q$
Implication (if)	$P \Rightarrow Q$	$P \rightarrow Q\ \ P \supset Q$
Equivalence (iff)	$P \Leftrightarrow Q$	$P \equiv Q\ \ P \leftrightarrow Q$
Universal (all)	$\forall x\, P(x)$	$(\forall x)\ P(x) \wedge\ xP(x)\ \ P(x)$
Existential (exists)	$\exists x\, P(x)$	$(\exists x)\ P(x) \vee\ xP(x)\ \ P(Skolem_i)$
Relation	$R(x, y)$	$(Rxy)\ \ Rxy\ \ xRy$

استفاده از منطق مرتبه اول:

- دامنه Kinship
- اصل موضوعات، تعاریف و قضایا
- دامنه مجموعه‌ها
- علائم خاص برای مجموعه‌ها، لیست‌ها و محاسبات
- طرح پرسش و گرفتن پاسخ

عامل‌های منطق برای دنیای Wumpus:

ما معماری سه عامل را در نظر می‌گیریم:

۱) عامل‌های (reflex) که فقط ادراکات و عملیاتشان رامطابق هم طبقه‌بندی می‌کنند.

۲) عامل‌های مبتنی بر مدل (model-based) که بازنمایی داخلی از دنیا را تشکیل می‌دهند و از آن برای عملکردشان استفاده می‌کنند.

۳) عامل‌های مبتنی بر هدف goal-based که اهداف را صورت می‌دهند و سعی دارند تا به آنها برسند. (عامل‌های مبتنی بر هدف معمولاً عامل‌های مبتنی بر مدل نیز هستند.)

عامل واکنشی ساده:

ساده‌ترین نوع ممکن عامل، قوانینی دارد که مستقیماً ادراکات را به عملیات مرتبط می‌سازد. این قوانین مشابه واکنش یا غرایز هستند.

محدودیت‌های عامل‌های واکنشی ساده:

- وجود مسائلی که باید به عامل از طریق بازنمایی دنیا فهمانده شود.
- عامل‌های واکنشی نمی‌توانند از حلقه‌های نامحدود اجتناب ورزند.

بازنمایی تغییر در دنیا:

در طراحی عامل، تمام ادراکات به پایگاه دانش اضافه می‌شود، و در اصل تاریخچه ادراک تمام آن چیزهایی است که در مورد دنیا باید دانسته شود. اگر ما قوانینی داشته باشیم که به گذشته به همان خوبی زمان جاری رجوع کنند، می‌توانیم قابلیت‌های یک عامل را برای یافتن جایی که عملکرد بهینه دارد، افزایش دهیم.
هر سیستمی که تصمیماتی را بر پایه ادراکات گذشته می‌گیرد، می‌تواند برای استفاده مجدد از جملاتی در مورد حالت جاری، دوباره نوشته شود، به شرط اینکه این جملات به محض رسیدن هر درک تازه‌ای و در عمل تازه‌ای که انجام می‌شود، به روز درآورده شود.
قوانینی که روش‌هایی در آن دنیا می‌تواند تغییر کند (تغییر نکند) را تعریف می‌کنند، قوانین diachronic نامیده می‌شوند که از زبان یونانی به معنای «سرتاسر زمان» برگرفته شده است. بازنمایی تغییرات یکی از مهم‌ترین حیطه‌ها در بازنمایی دانش است.
ساده‌ترین راه برای کنار آمدن با تغییرات، **تغییر پایگاه دانش** است.
یک عامل می‌تواند در فضای گذشته و حالات ممکن آینده، به جستجو بپردازد، و هر حالت توسط پایگاه دانش متفاوتی بازنمایی می‌شود.
در اصل، بازنمایی موقعیت و عملیات تفاوتی با بازنمایی اشیاء واقعی یا روابط واقعی ندارد.
ما نیاز داریم که در مورد اشیاء و روابط مناسب، تصمیم‌گیری کنیم و سپس قضایایی در رابطه با آنها بنویسیم.

محاسبه موقعیت:

محاسبه موقعیت (Situation Calculus) روش خاصی برای تعریف تغییرات در منطق مرتبه اول است.
تصوری که از دنیا می‌شود، آن را به صورت دنباله‌ای از موقعیت‌ها در نظر می‌گیرد، که هر کدام از آنها یک "snapshot" از حالت دنیا است.

استنتاج خواص پنهانی دنیا:

زمانی که عامل بتواند تشخیص دهد که کجا قرار دارد، می‌تواند کیفیت‌ها را با محل، به جای موقعیت تطبیق دهد.

قوانین همزمان:

قضایایی را که ما برای تسخیر اطلاعات ضروری برای این استنباط‌ها خواهیم داشت، قوانین همزمان (Synchronic) نامیده می‌شوند، زیرا آنها خواص حالت یک دنیا را به دیگر خواص حالت دنیای مشابه، مربوط می‌کنند.
دو نوع اصلی از قوانین همزمان وجود دارند:

قوانین Causal:

قوانین سببی جهت مفروض شده علت را در دنیا منعکس می‌کنند: بعضی از خواص پنهانی دنیا، ادراکات مطمئنی را برای تولید شدن باعث می‌شوند.

۲) قوانین تشخیصی (Diagnostic rules):

قوانین تشخیصی مستقیماً دلالت بر حضور خواص پنهان شده از اطلاعات مبتنی بر ادراک دارند.
اگرچه قوانین تشخیصی به نظر می‌آید که اطلاعات مطلوبی را مستقیماً تولید کنند، خیلی حیله‌گیرانه است، اطمینان داشته باشیم که آنها قوی‌ترین نتایج ممکن را از اطلاعات موجود به دست می‌آورند.
مهم‌ترین مسئله برای به خاطر سپردن این است که اگر قضایا به درستی و کمال، روش عملکرد دنیا و روشی که ادراکات تولید می‌شوند را تعریف کنند، رویه استنتاج به درستی قوی‌ترین شرح ممکن از حالت دنیا با ادراکات داده شده را استخراج خواهد کرد.

اولویت بین عملیات:

تغییرات عقاید عامل در مورد بعضی از چهره‌های دنیا نیاز به تغییرات در قوانینی که با دیگر چهره‌ها سروکار دارند، دارد.

- ✓ عامل ما به سادگی توسط پرسش برای رسیدن به چیزی متفاوت، می‌تواند دو مرتبه برنامه‌ریزی شود.

✓ اهداف، مطلوب بودن حالات حاصل را بدون توجه به روش به دست آمدن آنها توضیح می‌دهند.

اولین قدم، شرح مطلوبیت خود عملیات (action)، و ترک ماشین برای انتخاب بهترین عمل است.

از یک مقیاس ساده استفاده می‌کنیم:

عملیات می‌توانند عالی، خوب، متوسط ، ریسکی و یا مهلک باشند.

عامل همیشه باید یک عمل فوق‌العاده‌ای را در صورت یافتن انجام دهد؛ در غیر اینصورت، یک عمل خوب در غیر اینصورت، یک عمل متوسط، و یک عمل ریسک‌دار اگر تمام قبلی‌ها شکست بخورند.

سیستم مقدار عملیاتی:

سیستمی که حاوی قوانینی از این نوع است یک سیستم مقدار عملیاتی (action-value) نامیده می‌شود.

➢ توجه کنید که قوانین به آنچه که واقعاً عملیات انجام می‌دهند، رجوع نمی‌کنند، فقط به مطلوب بودن آنها توجه دارند.

به سوی یک عامل هدفدار:

حضور یک هدف دقیق به عامل اجازه می‌دهد تا دنباله‌ای از عملیاتی که منجر رسیدن به هدف می‌شوند را پیدا کند.

حداقل سه روش برای یافتن چنین دنباله‌ای وجود دارد:

۱) استنتاج
۲) جستجو
۳) برنامه‌ریزی

استنتاج:

نوشتن قضایایی که به ما اجازه ASK از KB را برای دنباله‌ای از عملیات بدهد که ضمانت رسیدن به هدف را به طور امن بکند، چندان مشکل نیست.

مشکلات این روش:

- برای دنیاهای بزرگ، تقاضاهای محاسباتی بسیار زیاد است.
- مشکل تشخیص راه‌حل‌های خوب از راه‌حل‌های بیهوده وجود دارد.

جستجو:

ما می‌توانیم از رویه جستجوی سطحی برای یافتن مسیری به هدف استفاده کنیم. این از عامل درخواست می‌کند تا دانش خود را به صورت مجموعه‌ای از عملگرها درآورد، و بازنمایی حالات را دنبال کند، بنابراین الگوریتم جستجو می‌تواند به کار برده شود.

برنامه ریزی:

شامل استفاده از سیستم‌های استدلال خاصی می‌شود که برای استدلال در مورد عملیات طراحی شده‌اند.

فصل هشتم

استنتاج در منطق مرتبه اول

قوانین استنتاج مربوط به سورها:

قوانین استنتاج برای منطق گزاره‌ای:

Modus Ponens
And – Elimination
And – Introduction
Or – Introduction
Resolution

سه قانون استنتاجی جدید:

۱- حذف سور عمومی (Universal Elimination):

برای هر جمله a متغیر v، و ترم زمینی g داریم:

$$\frac{\forall v, a}{SUBST(\{v/g\}, a)}$$

۲- حذف سور وجودی:

برای هر جمله a، متغیرv ، و سیمبول ثابت k که جای دیگری از پایگاه دانش ظاهر نشده است، داریم:

$$\frac{\exists v, a}{SUBST(\{v/K\}, a)}$$

۳- (Existential Introduction):

برای هر جمله a، متغیر v که در a واقع نباشد، و ترم زمینی gکه در a واقع نشود داریم:

$$\frac{a}{\exists v \ SUBST(\{g/v\}, a)}$$

می‌توان این قوانین را با استفاده از:

یک جمله با سور عمومی به عنوان ترکیب عطفی تمام مقداردهی‌های ممکن آن، و تعریف یک جمله با سور وجودی به عنوان ترکیب فصلی تمام مقداردهی‌های ممکن آن، کنترل کرد.

کاربرد قوانین استنتاج، در واقع پرسشی از مطابقت نمونه‌های پیش‌فرضیات آنها با جملات موجود در KB و سپس افزودن نمونه‌های جدید آنهاست.

اگر ما فرایند یافتن اثبات را به عنوان یک پردازش جستجو فرموله‌سازی کنیم، پس واضح است که اثبات همان راه حل مسئله جستجو است و روشن است که باید برنامه‌ای هوشمند برای یافتن اثبات بدون دنبال کردن هر گونه مسیر نادرست موجود باشد.

Modus Ponens **تعمیم یافته**:

- فرم Canonical
- یکسان‌سازی (Unificaiton)

فرم Canonical**:**

تمام جملات موجود در پایگاه دانش باید به صورتی باشند که با یکی از پیش‌فرضیات قانون Modus Ponens مطابقت داشته بشاند، فرم Canonical برای Modus Ponens متضمن این نکته است که هر جمله در پایگاه دانش از چه نوع اتمی یا شرطی با یک ترکیب عطفی از جملات اتمی در طرف چپ و یک اتم منفرد در طرف راست باید باشد.

ما جملات را به جملات Horn زمانی تبدیل می‌کنیم که ابتدا وارد پایگاه دانش، با استفاده از حذف سور وجودی و حذف And شده باشند.

یکسان‌سازی (Unificaiton):

وظیفه روتین یکسان‌ساز Unify، گرفتن دو جمله اتمی p، q و برگرداندن یک جانشین که p، q را مشابه هم خواهد ساخت، است. (اگر چنین جانشینی موجود نباشد، Unify، fail برمی‌گرداند.)

$$\text{UNIFY}(p, q) = \theta \text{ , } \text{SUBST}(\theta, p) = \text{SUBST}(\theta, q)$$

UNIFY، عمومی‌ترین یکسان‌ساز (Most General Unifier) یا (MGU) را برمی‌گرداند، که جانشینی است که کمترین تعهد را در قبل محدودسازی متغیرها دارد.

زنجیره‌سازی به جلو و عقب (Forward AND Backward Chaining):

زنجیره‌سازی به جلو (forward chaining):

قانون Modus Ponens تعمیم یافته به دو صورت استفاده می‌شود. می‌توانیم با جملات موجود در پایگاه دانش شروع کنیم و نتایج جدیدی را که می‌توانند استنباط‌های بیشتری را بسازند، تولید کنیم. این روش زنجیره‌سازی به جلو نامیده می‌شود.

این روش زمانی استفاده می‌شود که حقیقت جدیدی به پایگاه داده ما اضافه شده باشد و خواسته باشیم نتایج آن را تولید کنیم.

زنجیره‌سازی به عقب (Backward Chaining):

می‌توانیم با چیزی که قصد اثباتش را داریم آغاز کنیم و جملات شرطی را پیدا کنیم که به ما اجازه بدهند نتیجه را از آنها استنتاج کنیم، و سپس سعی در ایجاد پیش‌فرضیات آنها داشته باشیم.

این روش زمانی استفاده می‌شود که هدفی برای اثبات وجود داشته باشد.

الگوریتم زنجیره‌سازی به جلو:

زنجیره‌سازی به جلو توسط افزودن یک حقیقت جدید p به پایگاه دانش، فعال می‌شود و می‌تواند به عنوان قسمتی از پردازش TELL برای مثال، همکاری داشته باشد. در اینجا ایده، یافتن تمام ترکیبات شرطی است که P را به‌عنوان پیش‌فرض داشته باشد، سپس اگر بقیه پیش‌فرضیات برقرار باشند، می‌توانیم نتیجه ترکیب شرطی را به پایگاه دانش توسط راه‌اندازی استنتاج‌های بعدی اضافه کنیم.

ما به ایدهٔ ترکیب (Composition) جانشینی نیز نیاز داریم.

$COMPOSE(\theta_1, \theta_2)$ جانشینی است که اثر آن با اثر اعمال هر جانشینی به نوبت، برابر است. زیرا:

$$SUBST(COMPOSE(\theta_1, \theta_2), P) = SUBST(\theta_1, P)$$

زنجیره‌سازی به جلو، تصویری تدریجی از شرایط در حالی که داده‌های جدید وارد می‌شوند، می‌سازد.

پردازش‌های استنتاجی آن مستقیماً با حل مسئله ویژه در ارتباط نیستند،

به همین دلیل رویهٔ data-driven یا data-directed نامیده می‌شود.

الگوریتم زنجیره‌سازی به عقب:

زنجیره‌سازی به عقب به منظور یافتن تمام پاسخ‌ها برای سؤال طرح شده، به وجود آمده است. بنابراین زنجیره‌سازی به عقب، وظیفه‌ای که از رویه ASK خواسته شده را انجام می‌دهد. الگوریتم زنجیره‌سازی به عقب BACK-CHAIN ابتدا توسط کنترل درمی‌یابد که آیا پاسخ‌ها مستقیماً از جملات پایگاه دانش، تولید می‌شوند یا خیر. سپس تمام ترکیبات شرطی که نتایجشان با پرسش (query) مطابقت دارد را پیدا می‌کند و سعی دارد تا پیش‌فرض‌های آن ترکیبات شرطی را توسط زنجیره‌سازی به عقب ایجادکند.

اگر پیش‌فرض، یک ترکیب عطفی باشد، سپس BACK-CHAIN ترکیبات عطفی را عطف به عطف پردازش می‌کند، تا یکسان‌ساز را برای تمام پیش‌فرض بسازد.

کامل بودن Completeness:

تصور کنید که ما پایگاه دانش زیر را در اختیار داریم:

$$\forall x \quad P(x) \Rightarrow Q(x)$$
$$\forall x \quad \neg P(x) \Rightarrow R(x)$$
$$\forall x \quad Q(x) \Rightarrow S(x)$$
$$\forall x \quad R(x) \Rightarrow S(x)$$

سپس ما می‌خواهیم که S(A) را نتیجه بگیریم، S(A) درست است،

اگر Q(A)یا R(A) درست باشد، و یکی از آنها باید درست باشد زیرا:

یا P(A) یا ¬ P(A) درست است.

متأسفانه، زنجیره‌سازی با Modus Ponens نمی‌تواند S(A) را نتیجه بگیرد.

مشکل این است که $\forall x \neg P(x) \Rightarrow R(x)$ نمی‌تواند به صورت Horn دربیاید، و از این رو توسط Modus Ponens نمی‌تواند استفاده شود.

این بدان معنی است که رویه اثباتی که از Modus Ponens استفاده می‌کند ناکامل (incomplete) است:

جملاتی که در پایگاه دانش مستلزم شده‌اند ولی رویه نمی‌تواند آنها را استنتاج کند.

پرسش در مورد وجود رویه‌های اثبات کامل بحثی است که ارتباط مستقیم با ریاضیات دارد. اگر یک رویه اثبات کامل بتواند برای عبارات ریاضی پیدا شود، دو چیز دنبال می‌شود:

- تمام مفروضات می‌توانند به طور مکانیکی ایجاد شوند.
- تمام ریاضیات می‌توانند به عنوان نتیجهٔ منطقی مجموعه‌ای از اصل موضوع‌های پایه‌ای ایجاد شوند.

یک رویه اثبات کامل برای منطق مرتبه اول ارزش بسیاری در AI دارد:

- نظریه‌های عملی در رابطه با پیچیدگی کامپیوتری.
- فعال ساختن یک ماشین برای حل هر گونه مسئله که در زبان می‌تواند قرار داده شود.

قضیه کامل بودن گودل نشان داد که، برای منطق مرتبه اول، هر جمله‌ای که توسط مجموعه جملات دیگری مستلزم شود می‌تواند از آن مجموعه اثبات شود. بنابراین می‌توانیم قوانین استنتاجی را که به یک رویه اثبات کامل R اجازه می‌دهد، پیدا کنیم:

$$if \ \ KB \models then \ KB \vdash_R \alpha$$

این قضیه کامل بودن مشابه این است که بگوییم رویه‌ای برای یافتن سوزنی در یک پشته کاه وجود دارد و این ادعای بیهوده نیست زیرا جملات با سود عمومی و سیمبول‌های تابع لانه‌ای دلخواهی در پشته‌های کاه با اندازه نامحدود، استفاده می‌شوند.

گودل نشان داد که رویه اثباتی وجود دارد اما هیچ رویه‌ای را ذکر نکرد.

استلزام در منطق مرتبه اول، نیمه تصمیم‌پذیر (Semidecidable) بنابراین می‌توانیم نشان دهیم که جملات از پیش‌فرضیات تبعیت می‌کنند، اما همیشه نمی‌توانیم نشان دهیم که آنها از پیش‌فرضیات تبعیت نمی‌کنند.

به‌عنوان یک فرضیه، سازگاری (consistency) مجموعه جملات (سؤالی در مورد وجود راه‌حلی برای تبدیل تمام جملات به جملات درست) نیز نیمه تصمیم‌پذیر است.

Resolution: یک رویه استنتاج کامل

از دو ترکیب شرطی می‌توانیم ترکیب سومی را مشتق کنیم که پیش‌فرض اولی را به نتیجه دومی متصل می‌کند.
Modus Ponens به ما اجازه استخراج ترکیب شرطی جدید را نمی‌دهد و فقط نتایج اتمی را استخراج می‌کند. از این رو قانون resolution قدرتمندتر از Modus Ponens است.

قانون استنتاج resolution:

در فرم ساده قانون resolution، پیش‌فرضیات دارای دقیقاً دو ترکیب فصلی هستند. ما می‌توانیم این قانون را برای دو ترکیب فصلی به هر طولی وسعت بخشیم، که اگر یکی از قسمت‌های ترکیب فصلی در یک clause(Pj) با نقیض قسمت دیگر ترکیب فصلی (qk) یکسان باشند، سپس ترکیب فصلی از تمام قسمت‌ها استنتاج می‌شود بغیر از آن دو:

Resolution تعمیم یافته (ترکیبات فصلی)

Resolution تعمیمی یافته (ترکیبات شرطی)

Resolutionتعمیم یافته (ترکیبات فصلی):

برای Pi و qi فرضی که UNIFY (Pj ¬ qk)=θ

$$\frac{P_1 \vee \ldots P_j \ldots \vee P_m \qquad q_1 \vee \ldots q_k \ldots \vee q_n}{SUBST(\theta,(P_1 \vee \ldots P_{j-1} \vee P_{j+1} \ldots p_m \vee q_1 \vee \ldots q_{k-1} \vee q_{k+1} \ldots q_n))}$$

معادلاً، می‌توانیم این عبارت را به صورت ترکیب شرطی بنویسیم.

Resolution تعمیم یافته (ترکیبات شرطی):

برای اتم‌های Pi و qi و ri و si که

UNIFY (Pj , qk)=θ

$$\frac{P_1 \wedge \ldots P_j \ldots \wedge P_{n1} \Rightarrow r_1 \vee \ldots r_{n2} \qquad s_1 \wedge \ldots \wedge s_{n3} \Rightarrow q_1 \vee \ldots q_k \ldots \vee q_{n4}}{SUBST(\theta,(P_1 \wedge \ldots P_{j-1} \vee P_{j+1} \wedge p_{n1} \wedge s_1 \wedge \ldots s_{n3} \Rightarrow r_1 \vee \ldots r_{n2} \vee q_1 \vee \ldots q_{k-1} \vee q_{k+1} \vee \ldots \vee q_{n4}))}$$

فرم‌های Canonical برای resolution:

در نسخه اولیه قانون resolution، هر جمله یک ترکیب فصلی از حروف فرضی است.

تمام ترکیبات فصلی در KB فرض شده‌اند که در یک ترکیب عطفی صریح (مانند یک KBمعمولی) به هم متصل شده‌اند، بنابراین این فرم، فرم نرمال عطفی Conjunctive normal form (CNF) نامیده می‌شود.

اگرچه هر جمله به تنهایی یک ترکیب فصلی است.

در صورت ثانویه قانون resolution، هر جمله یک ترکیب شرطی با یک ترکیب عطفی از اتم‌ها در سمت چپ و یک ترکیب فصلی از اتم‌ها در طرف راست است.

این حالت، فرم نرمال شرطی implicative normal form (INF) نامیده می‌شود.

هر مجموعه از جملات می‌توانند به دو فرم ترجمه شوند. فرم نرمال عطفی رایج‌تر است، اما فرم نرمال شرطی طبیعی‌تر به نظر می‌آید.

Resolution تعمیمی از Modus Ponens است.

فرم نرمال شرطی رایج‌تر از فرم Horn است، به دلیل اینکه طرف سمت راست می‌تواند یک ترکیب شرطی باشد و نه فقط یک اتم تنها.

Modus Ponens قابلیت ترکیب اتم‌ها با یک ترکیب شرطی را به منظور استخراج نتیجه به صورتی دارد که resolution قادر به انجام آن نیست.

زنجیره‌سازی با resolution قدرتمندتر از زنجیره‌سازی با Modus Ponens است، اما هنوز کامل نیست.

برهان خلف:

رویه استنتاج کاملی که از resolution استفاده می‌کند برهان خلف (refutation) نامیده می‌شود و هم‌چنین به عنوان اثبات توسط تناقض (proof by contradiction) و reduction and absurdum) شناخته شده است.

تبدیل به فرم نرمال:

- ✓ هر جمله مرتبه اولی می‌تواند به صورت فرم نرمال شرطی (یا عطفی) دربیاید.
- ✓ از یک مجموعه از جملات به فرم نرمال می‌توانیم اثبات کنیم که یک جمله نرمال از مجموعه پیروی خواهد کرد.

رویه‌ای برای تبدیل به فرم نرمال:

۱) حذف ترکیب شرطی:

می‌توان تمام ترکیبات شرطی را با معادل فصلی جایگزین نمود.

۲) حذف ¬:

نقیض فقط برای فرم نرمال عطفی مجاز است، و برای تمام فرم‌های نرمال شرطی قدغن است.

۳) استاندارد کردن متغیرها:

این عمل بعداً از ایجاد ابهام زمان حذف سورها جلوگیری می‌کند.

۴) انتقال سورها به سمت چپ:

۵) Skolemize:

Skolemization پردازشی است که در آن تمام سورهای وجودی حذف می‌شوند.

۶) توزیع ∧ بر ∨ :

۷) ترکیبات فصلی و عطفی لانه‌ای مسطح شده:

در این مورد، جمله به فرم نرمال عطفی (CNF)است.

۸) تبدیل ترکیبات فصلی به ترکیب شرطی:

برخورد با مسئله تساوی:

یکسان‌سازی یک تست نحوی مبتنی بر ظاهر ترم‌های آرگومانی است و تست صحیح معنایی مبتنی بر اشیایی که نمایش می‌دهند، نیست.

دو روش برای انجام این امر:

۱) بدیهی نمودن تساوی به وسیله ذکر خواص آن:

باید ذکر شود که تساوی، انعطاف‌پذیر، متقارن و (متعدی) است.

۲) استفاده از یک قانون استنتاج از یک قانون استنتاج:

می‌توانیم قانون استنتاج را به صورت زیر تعریف کنیم:

Demodulation: برای تمام ترم‌های x,y,z که UNIFY (x,y) = θ

$$\frac{x = y, (\ldots z \ldots)}{(\ldots SUBST(\theta, y) \ldots)}$$

استراتژی‌های Resolution:

۴ استراتژی که برای راهنمایی جستجو به سمت یک اثبات استفاده می‌شوند، را بررسی خواهیم کرد:

۱) **Unit preference**:

در اینجا ما سعی بر تولید جمله کوتاهی به صورت True => False داریم.

این استراتژی یک کشف‌کننده مفید است که می‌تواند با دیگر استراتژی‌ها ترکیب شود.

۲) مجموعه Support

هر resolution جمله‌ای را از مجموعه Support با جمله دیگری ترکیب می‌کند و نتیجه را به مجموعه Support اضافه می‌کند. اگر مجموعه Support به نسبت تمام پایگاه دانش کوچک باشد، فضای جستجو را قطع خواهد کرد.

یک انتخاب بد برای مجموعه Support الگوریتم را ناکامل خواهد ساخت.

استراتژی مجموعه Support دارای این مزیت است که درخت‌های اثباتی تولید می‌کند که اغلب برای درک افراد آسان هستند، زیرا آنها هدف‌گرا هستند.

۳) Resolution ورودی:

در استراتژی resolution ورودی هر resolution یکی از جملات ورودی را (از KB یا query) با جمله دیگر ترکیب می‌کند.

در پایگاه‌های دانش Horn، Modus Ponens نوعی از استراتژی resolution ورودی است، زیرا یک ترکیب شرطی از KB اصلی را با دیگر جملات ترکیب می‌کند. از این رو شگفتی‌آور نخواهد بود که resolution ورودی برای پایگاه‌های دانشی که به صورت Horn هستند، کامل باشد اما در حالت کلی ناکامل است.

۴) Subsumption:

متد Subsumption تمام جملاتی که توسط یک جمله موجود در KB، Subsume می‌شوند، را حذف می‌کند.

Subsumption به نگهداری KB به صورت کوچک کمک می‌کند، و در نتیجه فضای جستجو را کوچک می‌سازد.

فصل نهم

برنامه‌ریزی

تفاوت عامل برنامه‌ریزی با عامل حل مسئله در سه چیز است:

بازنمایی اهداف، حالات و عملیات

استفاده از بازنمایی‌های منطقی و صریح برنامه‌ریز را قادر می‌سازد تا سنجش عامل را معقولانه هدایت کند.

عامل برنامه‌ریزی همچنین در روش بازنمایی و جستجو برای راه‌حل‌ها نیز تفاوت دارد.

یک عامل ساده برنامه‌ریزی:

زمانی که حالت دنیا قابل دسترسی است، عامل می‌تواند از ادراکات تولید شده توسط محیط استفاده کرده و مدل کامل و صحیحی از حالت دنیای جاری بسازد. سپس، با داشتن هدف، می‌تواند الگوریتم برنامه‌ریزی مناسبی را برای تولید برنامه عمل فراخوانی کند. عامل سپس می‌تواند در طی مراحل برنامه، هر لحظه یک عمل را اجرا کند.

عامل با محیط از طریق یک روش حداقل در عمل است و از ادراکاتش برای شرح حالت اولیه استفاده می‌کند و از این رو هدف اولیه را دنبال می‌کند؛ اما به سادگی توانسته مراحل برنامه را تشکیل بدهد.

از حل مسئله به برنامه‌ریزی:

برنامه‌ریزی و **حل مسئله** موضعات متفاوتی هستند زیرا در بازنمایی اهداف و حالات و عملیات و هم چنین بازنمایی ساختار دنباله‌های عملیاتی متفاوت عمل می‌کنند.

عناصر اولیه یک حل مسئله مبتنی بر جستجو:

- ❖ بازنمایی عملیات.
- ❖ بازنمایی حالات.
- ❖ بازنمایی اهداف.
- ❖ بازنمایی برنامه‌ها.

بازنمایی عملیات:

عملیات توسط برنامه‌هایی که شرح حالت مابعد را تولید می‌کنند، تعریف می‌شود.

بازنمایی حالات:

در حل مسئله، شرح کامل حالت اولیه داده شده است و عملیات توسط برنامه‌ای که شرح کامل حالت را تولید می‌کنند، بازنمایی می‌شوند.

بنابراین:

تمام بازنمایی‌های حالت، کامل هستند.

بازنمایی اهداف:

تنها دانشی که عامل در مورد هدف در اختیار دارد، **تست هدف** و **تابع کشف‌کننده** است. هر دو اینها بر روی حالت‌ها اعمال می‌شوند تا مطلوبیت آنها مورد ارزیابی قرار گیرد.

بازنمایی برنامه‌ها:

در حل مسئله یک راه‌حل دنباله‌ای از عملیات است. در طول تشکیل راه‌حل‌ها، الگوریتم‌های جستجو فقط دنباله‌های پیوسته عملیات را که از حالت اولیه آغاز می‌شوند (یا در مورد جستجوی دوطرفه، خاتمه دادن به حالت هدف) در نظر می‌گیرند.

حال ببینیم چطور این تصمیمات بر روی قابلیت عامل تأثیر می‌گذارند، تا مسئله ساده زیر را حل کنند:

«یک لیتر شیر و یک خوشه موز و یک مته چندسرعته را بخر.»

حالت اولیه: عامل در خانه است اما بدون هیچ یک از اشیاء موردنظر.

عملگر: تمام کارهایی که عامل قادر به انجام آن است.

تابع کشف‌کننده: تعداد چیزهایی که هنوز به دست آورده نشده‌اند.

اولین ایده کلیدی در ورای برنامه‌ریزی:

«بسط دادن» بازنمایی حالات، اهداف و عملیات است. الگوریتم‌های برنامه‌ریزی از تعاریفی به زبان‌های رسمی استفاده می‌کنند که معمولاً منطق مرتبه اول و یا زیرمجموعه‌ای از آن است.

حالات و اهداف توسط مجموعه‌هایی از جملات بازنمایی می‌شوند و عملیات توسط شرح پیش‌شرط‌ها و تأثیرات منطقی بازنمایی می‌شوند که برنامه‌ریزی را قادر می‌سازد تا ارتباطات بین حالات و عملیات را هدایت کند.

دومین ایده کلیدی در ورای برنامه‌ریزی:

این است که برنامه‌ریز آزاد است تا عملیات را به برنامه هر زمان که لازم باشد، اضافه کند. هرچند که دنباله افزایشی در حالت اولیه وجود داشته باشد.

هیچ الزامی بر وجود ارتباط بین مرتبه برنامه ریزی و مرتبه اجرا نیست. با ساختن تصمیمات «مشخص» و «مهم» در ابتدا، برنامه‌ریزی می‌تواند فاکتور انشعاب را برای انتخاب‌های بعدی و نیاز به پی‌جویی به عقب را برای تصمیمات اختیاری کاهش دهد.

سومین ایده کلیدی در ورای برنامه‌ریزی:

این است که بیشتر بخش‌های دنیا مستقل از دیگر بخش‌ها هستند.و این امر داشتن یک هدف عطفی را ممکن می‌سازد و می‌توان آن را با یک استراتژی تقسیم و غلبه حل نمود.

الگوریتم‌های تقسیم و غلبه مؤثر هستند؛ زیرا تقریباً همیشه حل چندین زیرمسئله کوچک آسان‌تر از یک مسئله بزرگ است. بهر حال تقسیم و غلبه در مواردی که هزینه ترکیب راه‌حل‌های زیرمسائل زیاد باشد، با شکست مواجه می‌شود. بسیاری از معماها دارای این خاصیت هستند.

دلیل اینکه معماها «گول‌زننده» هستند، این است که قرار دادن زیربرنامه‌ها کنار هم کار دشواری است.

فصل دهم

عدم قطعیت

مسئله‌ای که با منطق مرتبه اول و بنابراین با رهیافت عامل منطق‌گرا وجود دارد این است که:
عامل‌ها اغلب هیچگاه دسترسی کامل به تمام حقیقت درباره محیط خود را ندارند.

سؤالات بسیار مهمی وجود دارند که عامل نمی‌تواند پاسخ طبقه‌بندی شده به آن را بیابد. بنابراین باید تحت **عدم قطعیت (uncertainity)** عمل کند.

عدم قطعیت به علت کامل نبودن، و
عدم صحت درک عامل از خواص محیط ناشی می‌شود.

مسئله کیفیت:

قوانین بسیاری در دامنه کامل نیستند؛ زیرا:
۱) شرایط بسیار زیادی باید دقیقاً شمارش شوند،
یا
۲) برخی از شرایط ناشناخته هستند.

برخورد با دانش غیرقطعی:

ابزار اصلی ما برای کنار آمدن با درجات باور، تئوری احتمالات خواهد بود که درجه عددی از باور را بین ۰ و ۱ به جملات اختصاص می‌دهد.

احتمالات روشی از خلاصه‌سازی عدم قطعیت را به وجود می‌آورد که از تنبلی و جهل ما ناشی می‌شوند.
احتمال ۰ برای باور مبهمی که دارای جملات نادرست است، و
احتمال ۱ برای باور مبهمی که دارای جمله درست است، تخصیص داده می‌شود.
جمله در حقیقت خودش هم درست و هم نادرست است.
مهم است توجه داشته باشیم که **درجه باور** با **درجه درستی** متفاوت است.
تئوری احتمالات تعهد ontological را همانند منطق ایجاد می‌کند، که حقایق در دنیا هم وجود دارند و هم ندارند.
درجه درستی که با درجه باور در تضاد است، موضوع منطق فازی است.
در منطق مرتبه اول و گزاره‌ای، جمله بسته به تعبیر و دنیا، درست یا نادرست خواهد بود و زمانی درست است که حقیقتی را که به آن رجوع می‌کند، موضوع اصلی باشد.
عبارات احتمالی کاملاً مشابه این نوع معناها نیستند. به آن علت است که احتمالاتی که عامل به یک گزاره تخصیص می‌دهد به ادراکاتی که تا آن لحظه دریافت کرده است بستگی دارد.
در بحث استدلال غیرقطعی، ما آن را **شاهد (evidence)** می نامیم.
همانطور که وضعیت استلزام زمانی که جملات بیشتری به پایگاه دانش اضافه می‌شوند تغییر می‌کند، احتمالات نیز در صورت وجود شواهد بیشتر، تغییر خواهند کرد.
تمام عبارات احتمالی باید شواهدی را با توجه به اینکه کدام احتمال تشخیص داده شده است، تعیین کنند. همانگونه که عامل ادراکات جدیدی را دریافت می‌کند، ارزیابی‌های احتمالی به منظور انعکاس شاهد جدیدی، به روز درآورده می‌شوند.

عدم قطعیت و تصمیمات عقلانی:

حضور عدم قطعیت روش‌های تصمیم‌گیری عامل را تغییر داده است. عامل منطقی عموماً هدف واحدی دارد و هر برنامه‌ای که امکان رسیدن به آن قطعی است را اجرا می‌کند. یک عمل می‌تواند انتخاب و یا رد شود، چه به هدف برسد و چه نرسد و بدون توجه به آنچه که دیگر عملیات انجام می‌دهند.

تئوری سودمندی:

این تئوری این گونه بیان می‌شود:

هر وضعیت درجه‌ای از فایده یا سودمندی را برای یک عامل دارد وعامل به حالاتی با سودمندی بالاتر رجوع خواهد کرد. سودمندی یک حالت به عاملی وابسته است که مفروضاتش توسط تابع سودمندی بازنمایی شده است.
تئوری سودمندی رعایت حال دیگران را نیز می‌کند. برای یک عامل کاملاً منطقی است که سودمندی بالاتر را به وضعیتی اختصاص دهد که عامل خودش از آن رنج ولی دیگران منفعت می‌برند.
مفروضات که به عنوان سودمندی‌ها، مطرح شدند با احتمالات در تئوری عمومی تصمیمات عقلانی که تئوری تصمیم‌گیری نامیده می‌شود، ترکیب می‌شوند:

Dicision theory=probalility+utility theory

ایده اساسی در مورد تئوری تصمیم‌گیری این است که یک عامل منطقی است

اگر و فقط اگر

عملی را که منتهی به بالاترین سودمندی می‌شود، انتخاب کند.
این اصل سودمندی مورد انتظار ماکزیمم(MEU) نامیده می‌شود.
احتمالات و سودمندی‌ها در ارزیابی یک عمل توسط توزین سودمندی یک نتیجه ویژه و با احتمالی که پدید آورده است، ترکیب می‌شوند.

طراحی برای یک عامل تصمیم‌گیری نظری:

ساختار عاملی که از تئوری تصمیم‌گیری برای انتخاب عملیات استفاده می‌کند، در سطح انتزاعی با عامل منطقی یکسان است.