دانشگاه پیام نور
مرکز آران و بیدگل

جزوه درس

هوش مصنوعی

گردآورنده:
مصطفی قبائی

مرجع:

1.Artifical Intelligence:A modern Approach
By:S.Russell and P.Norvig 2nd ed, 2003

بنام آنکه جان را فکرت آموخت

پیشگفتار

در ایـن جـزوه تـلاش شـده اسـت تمـام سرفـصل هـای درس پوشـش داده شـود. در مـورد ایـن جـزوه یادآوری این نکتـه مهـم، ضـروری اسـت کـه ایـن جـزوه جهـت کمـک بـه دانـشجویان در جهـت کـاهش یادداشـت بـرداری، ترسـیم شـکل هـا ، مثـال هـا در هنگـام تـدریس بـوده اسـت. بنـابر ایـن در کنـار ایـن جـزوه، هـر دانـشجو مطـابق بـا ذوق و سـلیقه خـود جـزوه ای دسـت نـویس ، حـاوی یادداشـت هـایی بـه منظور تکمیل و تفهیم این جزوه خواهد داشت.

این جزوه، ادعـای جـایگزینی مراجـع اصـلی ایـن درس را نـدارد بلکـه تنهـا خلاصـه ای از مطالـب مهـم از مراجع درس اسـت. در گـردآوری ایـن جـزوه از کتـاب ***هـوش مـصنوعی: رهیـافتی نـوین*** تـالیف راسـل و پیتر نورویگ استفاده شده است.

در اینجا لازم اسـت از تمـامی دانـشجویانی کـه اینجانـب را در تـدوین ایـن جـزوه یـاری نمـوده انـد کمـال تـشکر را داشـته باشـم. همچنـین در ایـن جـزوه احتمـال اشـتباه وجـود دارد، بـه ایـن جهـت در مطالعـه مطالب جزوه دقـت کـافی را داشـته باشـید و وجـود هرگونـه ایـراد، و همچنـین نظـرات و پیـشنهادات خـود را از طریق پست الکترونیکی mostafaghobaye@yahoo.com به اطلاع اینجانب برسانید.

مصطفی قبائی آرانی
پاییز ۸۹

تقدیم به:

دانشجویانی که عقلشان بر احساسشان غلبه می کند.
دانشجویانی که برای تحمل آرای دیگران تمرین می کنند.
دانشجویانی که برای چهل سال آینده خود برنامه دارند.
دانشجویانی که فرق هشت و هشت و یک دقیقه را می دانند.
دانشجویانی که رنگهای شاد خلقت را در ظاهر خود سپاس می گویند.
دانشجویانی که با محاسبه حروف اضافه سخن می گویند.
دانشجویانی که قاعده مند فکر می کنند.
دانشجویانی که برای هر سوالی چندین پاسخ متفاوت قائل اند.
دانشجویانی که عصبانیت خود را به تاخیر می اندازند.
دانشجویانی که شان را بر قدرت مقدم می شمارند.
دانشجویانی که در رفتار قابل پیش بینی اند.
دانشجویانی که برای افزایش قدرت کشور تامل می کنند.
دانشجویانی که دغدغه های وفای به عهد، آنها را از خواب بیدار می کند.

(دکتر محمود سریع القلم)

فصل اول: مقدمه

آنچه در این فصل خواهید آموخت:

- **رهیافت های هوش مصنوعی**
- **مبانی هوش مصنوعی**
- **تاریخچه هوش مصنوعی**

علل مطالعه هوش مصنوعی:

۱. یادگیری بیشتر در مورد خودمان

۲. استفاده و بهره برداری از موجودیت هایی که توسط هوش مصنوعی تولید می شوند.

سیستم های هوشمند:

۱. نرم افزاری: مثل نرم افزارهای هوشمند نظیر نرم افزارهای تشخیص چهره یا تشخیص صدا

۲. سخت افزاری: مثل ربات

افراد مختلف،دیدگاههای متفاوتی نسبت به هوش مصنوعی دارند و برای آنها دو سوال مطرح است:

۱. آیا بیشتر به رفتار اهمیت می دهید یا به تفکر و استدلال؟

۲. آیا در ساخت سیستم های هوشمند، انسان را الگو قرار می دهید یا بر اساس یک استاندارد ایده آل هوشمندی (منطقی بودن، عقلانیت) عمل می کنید؟

با توجه به این دو سوال، هوش مصنوعی به چهار رهیافت تقسیم می شود:

انسان گونه فکر کردن	منطقی فکر کردن
انسان گونه عمل کردن	منطقی عمل کردن

تفاوت انسانی بودن و منطقی بودن(عقلانیت): منطقی بودن یعنی یک سیستم بر اساس دانش خود بهترین کار ممکن را در یک لحظه انجام دهد در حالیکه انسان در هر لحظه نمی تواند بهترین کار ممکن را انجام دهد چون همه انسان ها کامل نیستند.

منطقی فکر کردن ۱. ۲.	مانند انسان فکر کردن ۳. ۴.
منطقی عمل کردن ۵. ۶.	مانند انسان عمل کردن ۷. ۸.

۱. مطالعه توانایی های ذهنی از طریق مدل های کامپیوتری (محاسباتی) .
۲. مطالعه محاسباتی که امکان مشاهده،درک و استدلال را فراهم می نماید.
۳. تلاش برای ساختن کامپیوترهایی که فکر کنند، ماشین هایی با قدرت تفکر و حس کامل
۴. خودکار سازی فعالیت هایی که با تفکر انسان در ارتباط هستند مثل یادگیری ، تصمیم گیری و حل مسئله
۵. هوش محاسباتی(AI)، شامل مطالعه عامل های هوشمند است.
۶. هوش مصنوعی با رفتار هوشمندانه مصنوعات دست بشر سر و کار دارد.
۷. هنر خلق ماشین هایی که توانایی عملیاتی داشته باشند که آن عملیات اگر بخواهد توسط انسان انجام شود نیاز به هوش دارد.
۸. مطالعه چگونگی ساخت کامپیوترهایی که کارها را در هر لحظه بهتر از انسان انجام می دهد.

از نظر تاریخی هر چهار رهیافت دنبال شده اند و هرکدام طرفداران مخصوص به خود را دارد ولی تنشی بین رهیافت های انسانی و رهیافت های منطقی وجود دارد به دو دلیل:

۱ نگرش مبتنی بر انسانی، جز علوم تجربی است که شامل فرضیات و تاثیر آن توسط تجربیات است.

۲.نگرش منطقی، ترکیبی از ریاضیات و مهندسی است که با منطق سر و کار دارد.

بررسی دقیق تر هر یک از این چهار رهیافت:

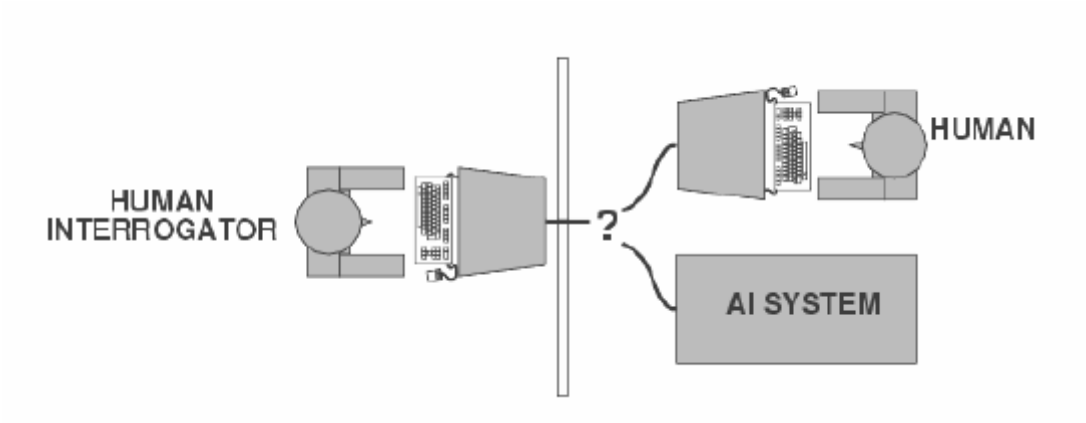
۱- مانند انسان عمل کردن[1](آزمون تورینگ):

اگر سیستمی بخواهد مانند انسان عمل کند باید از طریق تست تورینگ آزموده شود .تورینگ برای اولین بار تعریف علمی رضایت بخشی از هوش مصنوعی ارائه کرد یعنی برای تست سیستم هوشمند ، به جای پیشنهاد لیستی از پارامترها و سوالات که شاید هم بحث برانگیز باشد تست تورینگ را مطرح کرد.

تست تورینگ:

پرسش گر از طریق تله تایپ[2] یک سری سوالات را برای دو کامپیوتر مطرح می کند تا تشخیص بدهد در کدام کامپیوتر برنامه هوشمند و در کدام کامپیوتر انسان معمولی قرار دارد. اگر پرسش گر فریب بخورد و متوجه نشود آن گاه برنامه هوشمندی که طراحی کرده ایم با موفقیت از آزمون تورینگ بیرون می آید.

[1] Act like humans



کامپیوتری که بخواهد تست تورینگ را انجام دهد باید قابلیت های زیر را داشته باشد:

۱. پردازش زبان طبیعی[۳]: جهت توانایی در برقراری ارتباط موفقیت آمیز به زبان انگلیسی یا زبانهای انسانی دیگر

۲. بازنمایی دانش[۴] : به منظور ذخیره سازی آنچه که از قبل می داند و یا در حین آزمون آن را به دست می آورد.

۳. استدلال خودکار[۵] : جهت استفاده از اطلاعات ذخیره شده در پاسخگویی به سوالات و کسب نتایج جدید

۴. یادگیری ماشینی[۶] : تا خود را با شرایط تازه وفق دهد و الگوهارا کشف و برون ریزی کند.

تست تورینگ از ارتباط فیزیکی مستقیم بین کامپیوتر و پرسش گر اجتناب می کند زیرا شبیه سازی فیزیکی شرط هوشمندی نیست . تست کامل تورینگ، شامل یک سیگنال ورودی است که مصاحبه کننده از طریق آن توانایی ادراک اشیاء را دارد. به عبارت دیگر، اعمال فیزیکی ربات و انسان برای مصاحبه کننده به صورت یک سیگنال ویدئویی نشان داده می شود. بنابراین مشخصات فیزیکی آنها(مصاحبه شوندگان) مشخص نیست بلکه فقط عملکرد آنها در رابطه با اشیاء مورد محاسبه قرار می گیرد. لذا در تست جامع تورینگ به دو قابلیت زیر نیز نیاز است:

۵-بینایی کامپیوتر[۷]: برای درک اشیاء ۶- رباتیک[۸]: برای جابجایی و کنترل اشیاء

---

[2] Teletype
[3] Natural Language Process
[4] Knowledge representation
[5] Automated Reasoning
[6] Machine Learning

۲- مانند انسان فکر کردن[۹](مدل سازی شناختی):

اگر بخواهیم ادعا کنیم برنامه هوشمند ما مانند انسان فکر می کند ابتدا باید بررسی کنیم انسان چگونه فکر می کند برای این کار دو راه وجود دارد:

الف) از طریق درون گرایی: سعی در به دست آوردن طرز تفکر

اگر قادر به ایجاد تئوری دقیقی از ذهن باشیم آن گاه می توان این تئوری را به برنامه کامپیوتری تبدیل کرد. نویل و سیمون در سال ۱۹۶۱برنامه GPS[10] (حل کننده مسائل عمومی) را طراحی کردند آنها تنها به فکر حل کردن مسائل نبودند بلکه سعی کردند در حل یک مسئله خاص، مراحل استدلال برنامه خود را با مراحل تحلیل و استدلال انسان برای آن مساله، مقایسه و تطبیق دهند. در همین زمان محققان دیگری مانند وانگ فقط در پی پاسخ درست بوده اند بدون توجه به رفتار انسانی .

ب) از طریق تجارب روان شناسی:

علم شناختی[۱۱] یک علم میان رشته ای است که مدل های کامپیوتری هوش مصنوعی و فنون تجربی روانشناسی را ترکیب می کند تا بتواند تئوری های دقیقی از کارکرد ذهن دست آورد.

۲- منطقی فکر کردن[۱۲](رهیافت قوانین تفکر):

ارسطو از اولین کسانی بود که تلاش هایی را برای تدوین تفکر درست انجام داد قیاس صوری[۱۳]ارسطو معروف است این قیاس می گوید که با داشتن مقدمات درست، نتایج درست به دست می آید.

مثال از قیاس صوری:

مقدمات(مفروضات):

- سقراط یک انسان است
- همه انسان ها فانی هستند.

نتیجه: سقراط فانی است

---

[7] Vision
[8] Robotic
[9] Think like humans
[10] General problem solver
[11] Congnitive
[12] Think rationally
[13] Syllogism

این مجموع قوانین و یکسری قوانین دیگر باعث بوجود آمدن علم منطق شد. در سال ۱۹۶۵ برنامه هایی وجود داشتند که قادر بودند با وجود حافظه و زمان کافی شرحی از مسأله به زبان منطق را دریافت کنند و اگر راه حلی وجود داشته باشد آن را پیدا کنند. رهیافت منطقی فکر کردن با دو مشکل عمده روبرو بود:

۱. دریافت دانش غیررسمی و تبدیل آن به شکل رسمی توسط علائم منطقی کار ساده ای نیست مخصوصاً اگر درجه اطمینان دانش، کمتر از ٪۱۰۰ باشد
۲. تفاوت بزرگی بین قابل حل بودن مساله در تئوری و انجام آن در عمل وجود دارد یعنی مسائلی وجود دارند که با تعداد کمی فرضیات می تواند کامپیوتر را به بن بست محاسباتی بکشاند.

۴- منطقی عمل کردن[۱۴](رهیافت عامل منطقی):

اگر سیستمی بخواهد منطقی عمل کند لازمه اش این است که ابتدا منطقی فکر کند یعنی این رهیافت تمام رهیافت های قبلی را تحت پوشش خود قرار می دهد. بنابراین این رهیافت جامع ترین است. عامل های کامپیوتری ویژگی های دیگری از قبیل عملکرد خود مختار، ایستادگی در مدت زمان طولانی، وفق پیدا کردن، .... را نیز باید داشته باشند. در نگرش قوانین تفکر یا منطقی فکر کردن تاکید عمده بر روی استنتاج های صحیح بوده است در حالیکه تولید استنتاج های قسمتی از یک عامل منطقی است به عبارت دیگر عملکرد منطقی فقط شامل استنتاج درست نیست زیرا در بعضی موقعیت ها تصمیم صحیح و اثبات شدهای برای انجام وجود ندارد اما عامل باز هم باید عملی را انجام دهد حتی در بعضی موارد نمی توان بر اساس استنتاج تصمیم گیری نمود مثلا عقب کشیدن دست بدون تفکر از شی داغ، عکس العملی است که نسبت به عمل ناشی از تفکر انسانی موفق تر است. بنابراین ما هوش مصنوعی را از دیدگاه عامل عقلانی (رهیافت چهارم) بررسی می کنیم این کار دو مزیت دارد:

الف) اینکه رهیافت چهارم بسیار عمومی تر از قوانین تفکر هستند زیرا فکر کردن زیر مجموعه عمل کردن است یعنی برای منطقی عمل کردن ابتدا باید منطقی فکر کنیم(دلیل برتری بر رهیافت ۳)

ب) رهیافت چهارم در مقایسه با رهیافت های مبتنی بر تفکر و رفتار انسان بیشتر تابع پیشرفت علمی می باشد چون پیشرفت های علمی خیلی قانون پذیر تر از رهیافت هایی است که مبتنی بر تفکر یا رفتار انسان می

---

[14] Act rationally

باشندعلاوه بر این، رهیافت های مبتنی بر تفکر یا رفتار انسان پیچیده و ناشناخته و دور از دسترس ما هستند. (دلیل برتری بر رهیافت های ۱و۲)

در محیط های پیچیده رسیدن به عقلانیت کامل (رسیدن به منطق کامل و انجام اعمال صحیح) امکان پذیر نیست چون محاسبات زیادی را می برد بنابراین فرضیه سودمندی را می پذیریم و عقلانیت کامل حالت تئوریک و تحلیلی دارد عقلانیت محدود یعنی درست عمل کردن هنگامی که وقت کافی برای انجام محاسبات وجود داشته باشد.

زیربنای هوش مصنوعی:

زیربنای هوش مصنوعی یعنی رشته هایی که به نوعی با هوش مصنوعی در ارتباط بوده و باعث بوجود آمدن آن شده اند که آن رشته ها عبارت اند از:

- **فلسفه**

فلاسفه هوش مصنوعی را قابل تصور و انجام پذیر ساختند آنها این کار را با این فرض انجام می دهند که ذهن از برخی جهات مانند ماشینی است که بر اساس دانشی که به زبان داخلی خاصی بیان گردیده عمل می کند و این تفکر می تواند برای انتخاب اقداماتی که انجام خواهد شد استفاده شود.

- **ریاضیات**

ریاضیدانان، ابزارهای بکارگیری گزاره های منطقی قطعی و همچنین گزاره های غیر قطعی و احتمال را فراهم کردند این افراد همچنین اقدامات اولیه درک محاسبات و استدلال را در مورد الگوریتم ها انجام دادند.

- **اقتصاد**

اقتصاددانان، مسئله تصمیم گیری که نتایج مورد انتظار تصمیم گیرنده را بیشینه می کنند را به رسمیت درآوردند.

- **روانشناسی**

روانشناسان این فرضیه را پذیرفتند که می توان انسان ها و حیوان ها را ماشین های پردازش اطلاعات در نظر گرفت. زبان شناسان نشان دادندکه استفاده از زبان با این فرضیه تطابق دارد.

- **کامپیوتر**

مهندسین کامپیوتر ابزارهایی فراهم نمودند که با کاربردهای هوش مصنوعی را امکان پذیر می نماید حجم برنامه ها روبه افزایش است و بدون پیشرفت های عظیمی که در سرعت و حافظه صنعت کامپیوتر به وجود آمده است اجرای آنها ، امکان پذیر نمی باشد.

- **کنترل و Cybernetic**

نظریه کنترل با طراحی دستگاه ها سرو کار دارد که بر اساس بازخورد[۱۵] از محیط به صورت بهینه عمل می کنند . در ابتدا ابزارهای ریاضی نظریه کنترل کاملا با هوش مصنوعی تفاوت داشت اما این رشته ها در حال نزدیک شدن به یکدیگر هستند.

تاریخچه هوش مصنوعی:

۱. پیدایش هوش مصنوعی (۱۹۴۴-۱۹۵۶)
۲. اشتیاق اولیه،آرزوهای بزرگ (۱۹۵۲-۱۹۶۹)
۳. اندکی واقعیت(۱۹۶۶-۱۹۷۳)
۴. سیستم های مبتنی بر دانش:کلید قدرت؟ (۱۹۶۹-۱۹۷۹)
۵. تبدیل هوش مصنوعی به یک صنعت(۱۹۸۰ تاکنون)
۶. بازگشت به شبکه های عصبی (۱۹۸۶ تاکنون)
۷. تبدیل هوش مصنوعی به علم(۱۹۸۷ تاکنون)
۸. ظهور عاملهای هوشمند(۱۹۹۵ تاکنون)

۱۹۴۳: اولین کار جدی در حیطه ی هوش مصنوعی توسط مک کالو و والترپیتز انجام شد که آنها برای انجام کارهای خود از ۳ منبع استفاده کرده اند.

الف) دانش فیزیولوژی پایه و عملکرد نرون در مغز

ب) تحلیل رسمی منطق گزاره ها

ج) نظریه تورینگ در زمینه محاسبات

---

[15] Feed Back

راسل و وایپ هید، یک مدل عصبی مصنوعی پیشنهاد دادند که در آن هر عصب می توانست دو حالت off یا on داشته باشد. بنابراین هر تابع محاسباتی می تواند توسط شبکه ای از عصب های متصل به هم تشکیل گردد و هر تابع را می توان توسط ساختارهای ساده شبکه پیاده سازی کرد..

۱۹۵۰: آلن تورینگ اولین بار دید کاملی از هوش مصنوعی را در مقاله ای با عنوان «محاسبات ماشینی هوشمند» ارائه داد در این مقاله مفاهیمی مانند آزمون تورینگ – یادگیری ماشین و الگوریتم ژنتیک را مطرح کرد.

۱۹۵۱: هینسکی و ادموندز اولین کامپیوتر شبکه های عصبی را طراحی کرده اند. که SNARC نام داشت.

۱۹۵۲: آرتور ساموئل برنامه ای ساخت که یاد می گرفت بهتر از نویسنده اش بازی کند در نتیجه این فرضیه که کامپیوترها فقط کاری را انجام می دهند که به آن گفته می شود را نقض کرد چون یک کامپیوتر نتایج میانی را در خودش ذخیره می کرد و با عمل استدلال،کارهای خارق العاده ای انجام می داد.

۱۹۵۶: بزرگان و دانشمندان این رشته در کنفرانس دارت موث گرد هم آمدند و نام هوش مصنوعی را به جای عقلانیت محاسباتی(نام قدیمی هوش مصنوعی) انتخاب کردند.

۱۹۵۸: جان مک کارتی زبان لیسپ را که به بهترین زبان هوش مصنوعی تبدیل شد را تعریف کرد این زبان ، یک زبان سطح بالای قدیم است.

۱۹۶۳: برنامه SAINT توسط جیمز اسلاگل برای حل مسائل انتگرال گیری فرم بسته ارائه شد.

۱۹۶۷: برنامه Student که مسائل جبری داستان دار را حل می کرد.

۱۹۶۸: برنامهAnalogy توسط تام ایوانز طراحی شد مسئله ای بود که برای حل مسائل قیاس هندسی در آزمون هوش مورد استفاده قرار می گرفت.

۱۹۷۳-۱۹۶۶: کند شدن سیر تحقیقات هوش مصنوعی به سه دلیل:

الف) **پیچیده تر شدن الگوریتم های برنامه جدید مثل برنامه ترجمه متون،** ترجمه ماشینی متون ناموفق بود چون برخی از کلمات دارای چندین معنی بودند ثانیا ترجمه باید طبق زبان مقصد باشد. بنابراین ترجمه ای ناکارآمد به دست می آمد.

ب) **انجام ناپذیری بسیاری از مسائلی که سعی در حل آنها بود.** در اکثر برنامه های هوش مصنوعی برای حل مسائل، ترکیبات مختلف راه حل ها را انتخاب می کردیم تا زمانی که راه حل مسئله پیدا شود و این روش زمانی ممکن بود که تعداد فرضیات محدود بود ولی در اثبات مسائلی که فرضیات بیشتری داشتند با شکست مواجه می شدیم.

ج) **بکارگیری بعضی محدودیت ها روی ساختارهای اساسی مانند محدودیت نمایش پرسپترون دو ورودی**،پرسپترون، ساده ترین شبکه عصبی است و چیزهایی که می تواند یاد بگیرد را بازنمایی می کند ولی پرسپترون دو ورودی را نمی توان چنان آموزش داد که بتواند متفاوت بودن دو ورودی را تشخیص بدهد.

۱۹۷۹-۱۹۶۹: برنامه DENDRAL که وظیفه آن حل مسئله ساختار مولکولی یک ماده بر اساس اطلاعات فراهم شده از یک طیف نگار جرمی می باشد که ورودی برنامه شامل فرمول اولیه مولکول و طیف جرمی آن می باشد مثلا تمام زیر ملکول های $C_6H_{13}NO_2$ که جرم اتمی آنها برابر m=15 باشد را استخراج می کرد. نسخه اولیه برنامه تمام ساختارهای ممکن سازگار با فرمول را تولید می کرد که برای مولکول ها بزرگ دردسرساز بود چون باید تعداد زیر مولکول های زیادتری تولید می شد بنابراین به جای استفاده از یک نقطه بیشینه ، از دو نقطه بیشینهX_2,X_1 استفاده کردند تا تعداد حالت ها کاهش یابد. برنامه DENDRAL اولین سیستم خبره متمرکز دانش بود یعنی پایگاه دانش و کنترل در یک جا متمرکز بودند و مهارت سیستم از تعداد زیادی قواعد خاص ناشی می شود ولی سیستم های خبره دیگر مانند MYCIN بخش دانش و کنترل آنها از یکدیگر جدا بودند .

روش های ضعیف: روش هایی هستند که مبتنی بر یک جستجوی همه منظوره می باشند که قدم های اولیه یادگیری را بر می دارند اما تلاش در جهت یافتن راه حل های کامل را ندارند. برنامه DENDRAL از این رهیافت استفاده می کرد

برنامه MYCIN در زمینه پزشکی برای تشخیص عفونت های خونی و برای استفاده برخی پزشکان جوان توسط ۴۵۰ قاعده طراحی شده بود.

دو تفاوت MYCIN **و** DENDRAL:

۱. بر خلاف قواعد DENDRAL هیچ مدل نظری عمومی وجود نداشت که بتوان قواعد MYCIN را از آن استخراج کرد این قواعد باید از طریق مصاحبه مفصل با افراد خبره ، استخراج می شد. که خود این افراد خبره از کتابهای تخصصی و از قواعد سرانگشتی استفاده می کردند.

۲. این قواعد باید عدم قطعیتی که در دانش پزشکی وجود داشت را منعکس می ساخت بنابراین MYCIN از یک حساب عدم قطعیت به عنوان عوامل قطعیت استفاده می کرد.

تبدیل هوش مصنوعی به صنعت(۱۹۸۰تاکنون): اولین سیستم خبره تجاری موفق شرکت DEC کهR_1XCON نام داشت شروع به تنظیم سفارش های سیستم کامپیوتری بر اساس نیاز مشتری می کرد.

کاربردهای هوش مصنوعی (وضعیت فعلی هوش مصنوعی):

الف) **برنامه ریزی و زمان بندی خودمختار:** اولین برنامه خودمختارRemot Agent نام داشت که توسط ناسا برای کنترل فضاپیما طراحی شده بود و بر عملیات فضاپیما در حین اجرای طرح ها نظارت می کرد.

ب) **انجام بازی:** Deepblue نام اولین کامپیوتری بود که توانست با نتیجه ۳/۵ بر ۲/۵ بر قهرمان شطرنج جهان، گری کاسپارف غلبه کند.

ج) **کنترل خودمختار:** سیستم بینایی کامپیوتری ALVENN به منظور هدایت خودرو در طول مسیر آموزش دیده می شود.این سیستم بر روی یک خودرو نصب می شد که توسط دوربین های ویدئویی که روی ماشین نصب شده بود تصاویر ویدئویی را دریافت می کرد و حرکت مناسب را انجام می داد.

د) **تشخیص پزشکی:** مانند سیستم MYCIN

ه) **برنامه ریزی ترابری:** در دوران بحران خلیج فارس، امریکا از یک ابزار تحلیل و برنامه ریزی مجدد پویای DART استفاده کرد که برای انجام خودکار برنامه ریزی ترابری و زمان بندی در حمل و نقل استفاده می شد که تعداد مسافران، خودروها و مبدا و مقصد را به عنوان ورودی دریافت می کرد.

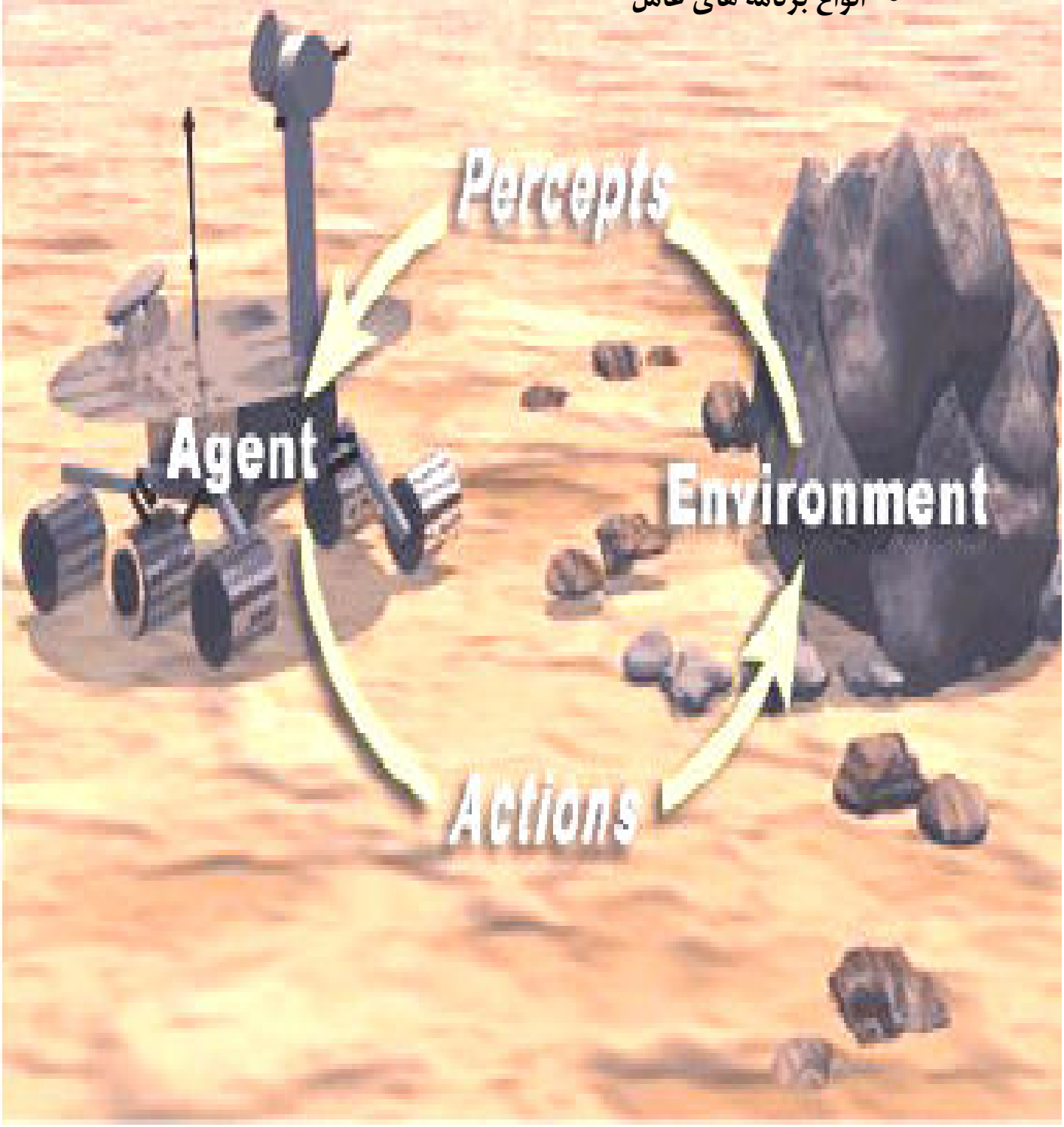
و) **رباتیک:** بسیاری از جراحان از رباتها، برای عمل جراحی استفاده می کردند.

ی) **درک زبان و حل مسئله:** Proverb نام برنامه کامپیوتری است که جداول کلمات متقاطع را بهتر از اکث انسانها حل می کرد.

فصل دوم: عاملهای هوشمند

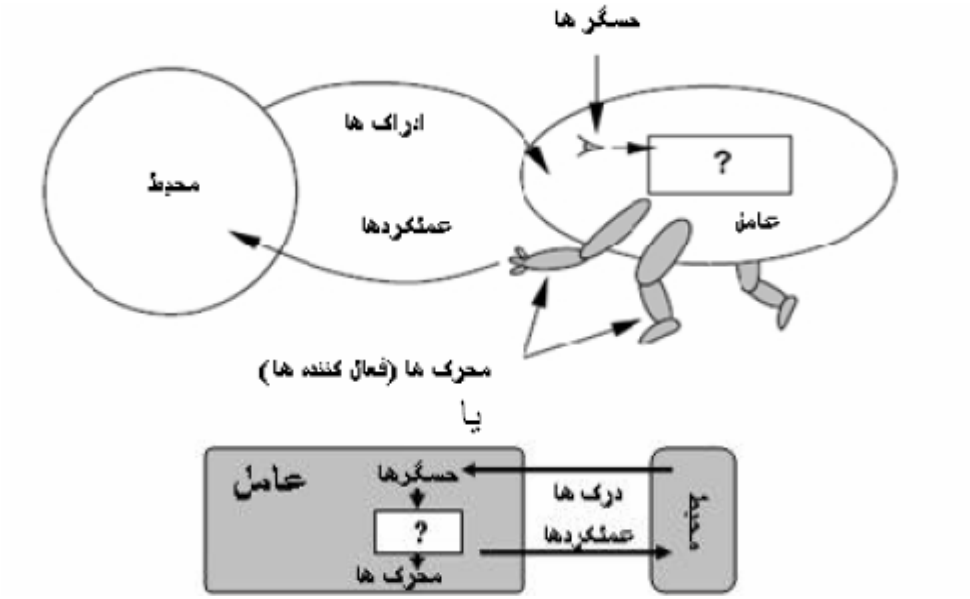
آنچه در این فصل خواهید آموخت:

- **عامل و مفاهیم مربوطه**
- **انواع محیط ها**
- **انواع برنامه های عامل**



عامل ها و محیط ها

عامل[۱۶]: هر چیزی است که محیط خود را از طریق حسگرها[۱۷] ادراک می کند و از طریق محرکها[۱۸] در آن محیط اقدامی انجام می دهد.



عامل ها از طریق حسگرها و اقدامگرها با محیط در تعامل هستند

مثال هایی از عامل ها:

عامل انسانی

۱. **sensor:** گوش، چشم، پوست، زبان، بینی،....

۲. **effector:** دست، پا،دهان، اندام‌های دیگر

عامل روباتیک

۱. **sensor:** دوربین، یابنده‌های مادون قرمز

۲. **effector:** موتور،چرخها،بازوها

عامل نرم افزاری:

۱.**effector:**صفحه کلید

۲-**effector**:صفحه نمایش

---

[16] agent
[17] sensor
[18] effector

ادراک[۱۹]: ورودی ها ی ادراکی عامل در هر لحظه

دنباله ادراکی[۲۰]: تاریخچه و سابقه کامل هر چیزی که عامل تاکنون دریافت کرده است.

انتخاب یک عمل توسط عامل در هر لحظه به کل دنباله ادراکاتی که تاکنون دریافت کرده است بستگی دارد.

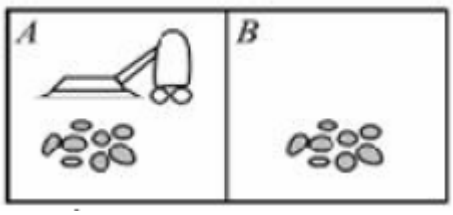
تابع عامل:

رفتار یک عامل توسط یک تابع عامل که هر رشته ادراکات ممکن را به یک عمل نگاشت می کند ، توصیف می شود تابع عامل را می توان به صورت جدولی درآورد که عامل را توصیف می نماید. اگر یک عامل را جهت انجام آزمایش در اختیار داشته باشیم ، می توانیم با آزمایش، تمام رشته ادراکات ممکن را ثبت و اقداماتی که عامل در پاسخ انجام می دهد در قالب یک جدول بسازیم . این جدول ویژگی های بیرونی عامل است. از دیدگاه درونی برای یک عامل مصنوعی تابع عامل به صورت یک برنامه عامل پیاده سازی می شود.تابع عامل یک توصیف انتزاعی ریاضی است ولی برنامه عامل یک پیاده سازی واقعی می باشد که در معماری در حال اجراست. به عبارت دیگر برنامه عامل، تابع عامل را تحقق می بخشد .مفهوم عامل ابزاری برای تحلیل سیستم هاست و نه یک توصیف انتزاعی که دنیا را به دو گروه عامل ها و غیر عامل تقسیم کند.

مثال:دنیای جاروبرقی

- محیط عامل، شامل دو فضای A,B می باشد که هر کدام می توانند تمیز یا کثیف باشند.
- **حسگرها:**حسگر کثیفی،حسگر تعیین محل----- **ادراکات:**[A,clean]
- **محرک ها:**چرخ ها،ابزارهای مکش------**عملکردها:**حرکت به چپ،راست،مکش،هیچ کار
- در این مثال ساده، ما می توانیم تمام رشته ادراکات ممکن و عملکردهای وابسته را لیست نماییم.این عامل،یک عامل مبتنی بر جدول نام دارد.

عملکرد	رشته ی ادراکی
Right	[A,Clean]
Suck	[A,Dirty]
Left	[B,Clean]
Suck	[B,Dirty]
Right	[A,Clean],[A,Clean]
Suck	[A,Clean],[A,Dirty]
N	N



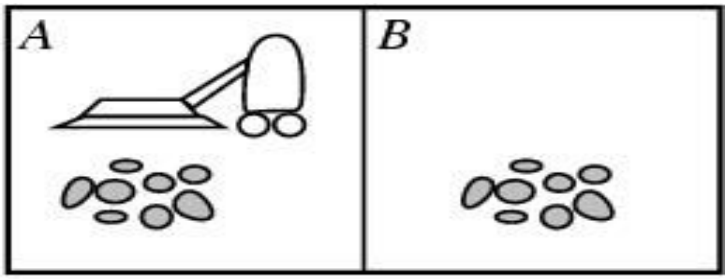
[19] percept
[20] Percept sequence

عامل ها چگونه عمل می کنند؟

یک عامل منطقی، عاملی است که کار درست انجام می دهد . کار درست، عملی است که باعث موفق تر شدن عامل می شود. برای اندازه گیری میزان موفقیت یک عامل از واژه ای بعنوان **معیار کارایی**[۲۱]استفاده می کنیم. بدیهی است که معیار کارایی یکسانی برای همه عامل ها وجود ندارد. معیار کارایی باید از دید **طراح عامل** تعیین شود . به عنوان مثال برای جاروبرقی معیارهای کارایی عبارتند از: میزان انرژی الکتریکی مصرف شده ، میزان زباله جمع شده در چند ساعت توسط جارو برقی و ....

زمان ارزیابی معیار کارایی اهمیت دارد . باید معیار کارایی را در دراز مدت تعیین کرد.

دنیای جارو برقی تنها با دو محل



- Environment: square A and B
- Percepts: [location and content] e.g. *[A, Dirty]*
- Actions: left, right, suck, and no-op

تفاوت میان منطقی بودن[۲۲] و عقل کل بودن[۲۳](همه چیز دانی)

باید بین منطقی بودن و عقل کل بودن تفاوت قائل شویم :

۱- یک عامل عقل کل نتیجه واقعی اقداماتش را می داند و بر این اساس عمل می کند ولی عقل کل بودن در واقعیت غیر ممکن است(شخص پیاده و هواپیما)

۲- منطقی بودن ، کارایی مورد انتظار را بیشینه می کند در حالیکه عقل کل بودن کارایی واقعی را بیشینه می کند و اگر ما انتظار داشته باشیم که یک عامل آنچه را که در عمل بهترین خواهد بود انجام دهد طراحی چنین عاملی غیر ممکن است .

---

[21] performance measure
[22] Rationality
[23] omniscience

۳- عقل کل بودن نیاز به یک دانش بی نهایت دارد ، ولی منطقی بودن یعنی در حدی که به عامل ، دانش تزریق کرده ایم از او انتظار داشته باشیم.

به طور خلاصه منطقی بودن یک عامل همیشه به چهارچیز وابسته است :

۱- معیار کارایی که در درجه موفقیت عامل را مشخص می کند

۲- مشاهداتی که عامل تاکنون بوسیله حسگرهایش دریافت کرده است ، این تاریخچه ادراکی کامل را رشته ادراکی می نامند .

۳- آنچه را که عامل درباره محیط می داند .

۴- اعمالی که عامل می تواند انجام دهد .

این موارد منجر به تعریف یک عامل منطقی ایده آل می شود .

تعریف عامل منطقی ایده آل:

"یک عامل منطقی ایده آل، به ازای هر رشته اداراکات ممکن باید اعمالی را بر اساس رشته اداراکی و دانش پیش زمینه ای که دارد، انجام دهد تا معیار کارایی اش را ماکزیمم کند ."

انجام اعمالی که منجر به، بدست آوردن اطلاعات مفید یا به عبارتی منجر به تغییر مشاهدات آینده می شود ، جمع آوری اطلاعات[۲۴] نامیده می شود که یک قسمت مهم از منطقی بودن است .

فاکتورهای لازم برای منطقی عمل کردن عبارت است از :

۱- جمع آوری اطلاعات و اکتشافات

۲- یادگیری سیستم با توجه به تجربیات خود

۳- خودمختاری که باید عامل داشته باشد

عامل های موفق، محاسبه تابع عامل را به سه دوره مختلف تقسیم می کنند:

۱-وقتی عامل طراحی می شود مقداری محاسبات توسط طراح عامل انجام می شود

۲- وقتی عامل در حال تصمیم گیری برای انجام عمل بعدی خود می باشد محاسبات بیشتری انجام می دهد.

۳- وقتی عامل از تجربیاتش یاد می گیرد ، محاسبات بیشتری برای تغییر رفتار خود انجام می دهد.

---

[24] Information gathering

خودمختاری[۲۵]:

رفتار یک عامل می تواند متکی بر پایه تجربه خود و دانش درونی بنا نهاده شود. اگر عامل فقط بر اساس دانش درونی(پیش زمینه) عمل کند و به ادراکات دریافت شده از محیط توجه نکند فاقد خود مختاری است بنابراین عامل باید خودمختار باشد یعنی تجربیاتش را نیز در نظر بگیرد.

در عمل به ندرت از ابتدا نیاز به خود مختاری کامل است . وقتی عامل تجربه ای ندارد و یا تجربه کمی دارد به صورت تصادفی عمل می کند ، چون از ابتدا هیچ ادارکی از محیط نگرفته، بنابراین همان گونه که خداوند برای تکامل حیوانات، به اندازه کافی واکنش های ذاتی قرار داده است تا بتوانند آنقدر زنده بمانند و سپس خودشان یاد بگیرند. به طریق مشابه، معقول خواهد بود که برای یک عامل هوش مصنوعی مقداری دانش اولیه برای عامل فراهم کنیم . پس از کسب تجربه کافی از محیطش، رفتار یک عامل عقلانی می تواند به طور موثر مستقل از دانش قبلی اش شود. بنابراین افزودن یادگیری، امکان طراحی یک عامل عقلانی ساده را می دهد تا در محیطهای گوناگون موفق باشد.

تعیین مشخصات محیط کار:

برای یک عامل ، اولین مرحله تعیین مشخصات محیط کار تا حد امکان به صورت کامل می باشد . مواردی مانند مقیاس کارایی (Performance measure)، محیط(Environment) ، اقدام گر ها (Actuator) و حسگرها (Sensor) تحت عنوان محیط کار (PEAS) توصیف می شوند.

- مثال: طراحی یک راننده تاکسی اتوماتیک
  - معیار کارآیی: امنیت، سرعت، راحتی، سود و ...
  - محیط: خیابان ها، افراد پیاده، مشتری ها و ...
  - اثر کننده ها: فرمان، شتاب دهنده، ترمزها، بوق، چراغ ها و ...
  - حسگرها: دوربین ها، حسگرهای صوتی (Sonar)، سرعت سنج، GPS، کیلومتر شمار، حسگرهای موتور، صفحه کلید، میکروفون و ...

- عامل: سیستم تشخیص پزشکی
  - معیار کارآیی: سلامتی بیمار، به حداقل رساندن هزینه و ...
  - محیط: بیمار، بیمارستان، کارمندان و ...
  - اثر کننده ها: صفحه نمایش (پرسش ها، آزمایش ها، تشخیص ها، مداوا)
  - حسگرها: صفحه کلید (دریافت علایم، یافته ها و پاسخ های بیمار)

[25] Autonomy

انواع محیط ها:

۱- کاملا رویت پذیر[۲۶] در مقابل نیمه رویت پذیر[۲۷] :

اگر حسگرهای یک عامل امکان دسترسی به وضعیت کامل محیط در هر لحظه از زمان را به عامل بدهند می گوییم محیط کاملا رویت پذیر است مانند : صفحه شطرنج و محیط پازل ۸

کار در محیط های کاملا رویت پذیر راحت است زیرا نیازی نیست که عامل هیچ حالتی را حفظ و ذخیره کند تا بتواند اتفاقاتی که در دنیا روی می دهد را ثبت کند. گاهی اوقات، عدم دقت حسگرها و یا نویز باعث می شود قسمتهایی از حالت ها در داده های حسگر حذف شوند اینگونه محیط ها، نیمه رویت پذیر هستند.

۲- قطعی[۲۸] در مقابل غیر قطعی (اتفاقی):

اگر بر اساس وضعیت فعلی و اقدامی که توسط عامل اجرا می شود وضعیت بعدی محیط به طور کامل تعیین شود گوییم که محیط قطعی است . در غیر این صورت اتفاقی است مثلا محیط دنیای جاروبرقی قطعی است ولی محیط رانندگی تاکسی اتفاقی است (ممکن است چراغ سبز شود یا بنزین تمام شود) . اگر یک محیط بدون توجه به اقدامات دیگر عامل ها قطعی باشد محیط را استراتژیک یا راهبردی گوییم مانند محیط بازی شطرنج

۳-مرحله ای[۲۹] در مقابل ترتیبی:

مرحله یا Episode شامل ادراک توسط عامل و آنگاه انجام یک اقدام می باشد . در محیط های مرحله ای انتخاب اقدام در هر مرحله تنها به خود آن مرحله بستگی دارد و در مراحل بعدی تاثیری ندارد بسیاری از کارهای دسته بندی از این نوع هستند مثل عاملی که قطعات معیوب را روی خط مونتاژ شناسایی می کند که بدون توجه به تصمیمات قبلی قطعه معیوب را شناسایی می کند .در محیط ترتیبی تصمیم فعلی می تواند بر تصمیمات بعدی تاثیر بگذارد مانند شطرنج و رانندگی تاکسی .

محیط های مرحله ای نسبت به ترتیبی ساده تر هستند چون عامل نیاز ندارد به جلوتر فکر کند.

---

[26] Fully observable
[27] Partially observable
[28] deterministic
[29] Episodic

۴-ایستا در مقابل پویا :

اگر در حالیکه عامل تعمق می کند محیط نیز تغییر کند می گوییم محیط برای آن عامل پویا است ، در غیر این صورت ایستاست .

هواشناسی و رانندگی تاکسی محیط های پویا هستند . در هواشناسی چون در حین تجزیه و تحلیل، دما تغییر می کند و در رانندگی ممکن است چراغ های راهنمایی در حین رانندگی تغییر کنند.

به محیطی که با گذشت زمان در حین سنجش شرایطش عوض نمی شود اما امتیاز کارایی آن فرق می کند محیط نیمه پویا[۳۰] گفته می شود. محیط شطرنج بدون ساعت، ایستا و محیط شطرنج با ساعت نیمه پویاست. جدول کلمات متقاطع ایستاست.

سوال: در کدامیک از محیط های زیر عامل نیازمند به حفظ وضعیت جاری نیست؟

✓ الف- محیط کاملا قابل مشاهده ب- قطعی ج- گسسته د-پویا

۵-گسسته[۳۱] در مقابل پیوسته:

اگر ادراکات و اعمال عامل را بتوان با اعداد گسسته بیان کرد محیط گسسته،در غیر اینصورت محیط پیوسته می باشد.بازی شطرنج یک محیط گسسته ولی رانندگی تاکسی یک محیط پیوسته است.

اگر تعداد محدود و مجزایی از درک-عمل ها(ایپیزود) داشته باشیم محیط گسسته و اگر تعداد نامحدود باشد محیط پیوسته است.

تعریف دیگر:اگر ادراکات یا اعمال یک عامل مقادیر پیوسته به خود بگیرد محیط پیوسته ولی اگر مجموعه مقادیر گسسته بگیرد محیط گسسته می باشد. به عنوان مثال در محیط رانندگی تاکسی زاویه های فرمان یا سرعت مقادیر پیوسته به خود میگیرند.

۶-تک عامله در مقابل چند عامله:

محیطی تک عامله است که فقط یک عامل در آن قرار دارد مانند محیطی که یک عامل در آن جدول کلمات متقاطع را حل می کند. در محیط های چند عامله،بیش از یک عامل در محیط قرار دارد. مانند محیط بازی شطرنج،که یک محیط دو عامله است.محیط های چند عامله به دو دسته محیط های چندعامله رقابتی و

---

[30] Semi dynamic
[31] Discrete

چندعامله مشارکتی تقسیم می شوند. در محیط چندعامله رقابتی،مقیاس کارایی عامل ها در تناقض با یکدیگر است مانند محیط بازی شطرنج ولی در چندعامله مشارکتی،مقیاس کارایی عامل ها در راستای یکدیگر است مانند محیط رانندگی تاکسی با مقیاس کارایی اجتناب از تصادف–در حالیکه اگر مقیاس کارایی در این محیط پارک کردن در یک محل باشد محیط چند عامله رقابتی می باشد.

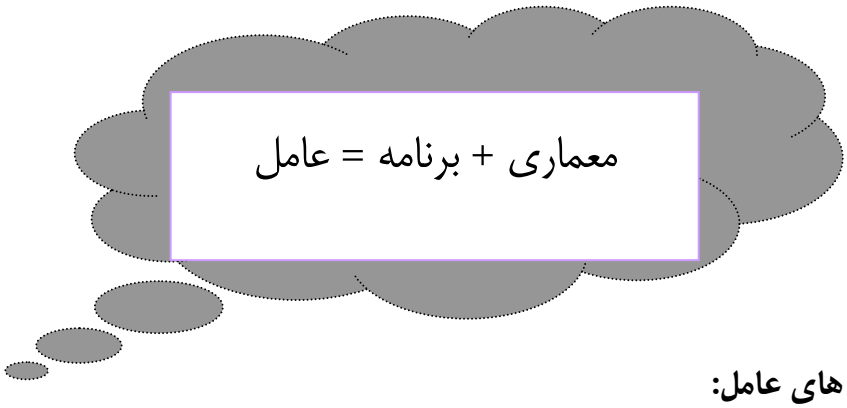
حالات نیمه رویت پذیر،اتفاقی،ترتیبی،پویا،پیوسته و چند عامله جزء مشکل ترین محیط ها می باشند مانند محیط رانندگی تاکسی و محیط سیستم تشخیص پزشکی.

مثال‌هایی از انواع محیط و ویژگی‌های آنها

محیط	قابل دسترسی	قطعی	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 تاکسی	NO	NO	NO	NO	NO
سیستم تشخیص پزشکی	NO	NO	NO	NO	NO
سیستم تحلیل تصویر	YES	YES	YES	Semi	NO
ربات جابجا کننده اشیاء	NO	NO	YES	NO	NO
کنترل‌کننده پالایشگاه	NO	NO	NO	NO	NO
آموزش‌دهنده انگلیسی با ارتباط متقابل	NO	NO	NO	NO	YES

ساختار عامل های هوشمند

وظیفه طراح هوش مصنوعی ، طراحی برنامه عامل است یعنی طراحی تابعی که رشته مشاهدات را به یک عمل نگاشت می کند. باید بین تابع عامل و برنامه عامل تفاوت قائل شویم . برنامه عامل،پیاده سازی تابع عامل می باشد که بر روی قسمتی از دستگاههای محاسباتی که معماری نامیده می شود اجرا خواهد شد. در ضمن،برنامه عامل،ادراک فعلی را به عنوان ورودی دریافت میکند در حالیکه تابع عامل کل تاریخچه ادراکات را به عنوان ورودی دریافت می کند.معماری ممکن است یک کامپیوتر یا یک سخت افزار خاص منظوره مانند پردازشگر تصاویر دوربین یا فیلتر کننده ورودی صدا باشد. به طور کلی معماری،. ابتدا مشاهدات و ادراکات را از حسگرها می گیرد و برای برنامه عامل قابل دسترس می کند ، سپس برنامه عامل را اجرا نموده و اعمال انتخاب شده را به محرک ها می رساند.ارتباط بین عامل،معماری و برنامه به صورت زیر می باشد.



انواع برنامه های عامل:

انواع گوناگون طراحی های ساده برنامه عامل وجود دارند که نوع اطلاعاتی را که صریحاً وجود دارند و در فرآیند تصمیم گیری استفاده می شوند را منعکس می سازند. این طراحی ها از نظر کارایی ، فشردگی و انعطاف پذیری با یکدیگر تفاوت دارند. طراحی مناسب برای هر عامل به طبیعت محیط بستگی دارد .

برنامه های عامل:

برنامه های عامل که معرفی خواهیم کرد همگی طرح کلی یا اسکلت یکسانی دارند.ادراک فعلی را به عنوان ورودی از حسگرها دریافت می کنند و یک عمل را به محرک ها بر می گردانند.

AGENT- BASE-PROGRAM

```
1- function SKELETON -AGENT
            (percept) returns action
2 - static:
3 -     memory, the agent memory of the world
4 - memory  ← Update -Memory(memory,percept)
5 - action     ← Choose -Best -Action(memory)
6 - memory  ← Update -Memory(memory,action)
7 -  return action
```

عاملهای مبتنی بر جدول[۳۲]:

برنامه های مبتنی بر جدول برای هر ادراک جدید فراخوانی می شوند وهر بار یک اقدام را بر می گرداند و با استفاده از ساختار داده خاصی مانند لیست پیوندی تاریخچه رشته ادراکات را ذخیره می کند. این روش برای عاملهایی که تعداد ادراکات آن کم و معلوم باشد مناسب است.

رویکرد مبتنی بر جدول در ساخت عامل به دلایل زیر محکوم به شکست است:

۱) جدول مورد نیاز یرای عامل ساده ای که تنها قادر به بازی شطرنج باشد 35^{100} سطر خواهد داشت.

۲) زمان بسیار طولانی لازم است تا طراح قادر به ساخت جدول باشد.

۳) عامل مبتنی بر جدول فاقد هرگونه خودمختاری است زیرا محاسبه بهترین عمل، کاملاً درونی صورت می گیرد و اگر چنانچه شرایط محیط به گونه ای غیر قابل پیش بینی تغییر کند، عامل شکست خواهد خورد.

۴) حتی اگر به عامل، روش یادگیری داده شود تا درجه ای از خودمختاری حاصل شود برای یادگیری مقدار صحیح از بین انبوه سطرهای جدول به زمان بی نهایت نیاز خواهد بود.

نکته: اگر p مجموعه ادراکات ممکن و T طول عمر عامل یعنی تعداد کل ادراکاتی که دریافت می کند باشد، جدول مرجع تعداد p^T داده خواهد داشت. (برای عامل جاروبرقی این فرمول برابر 4^T می شود).

---

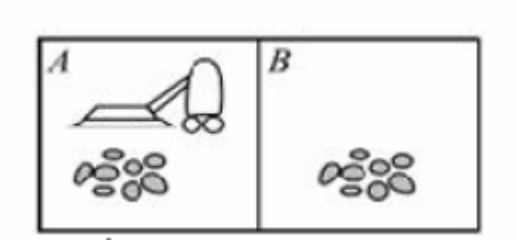
[32] Table-driven

با وجود تمام موارد فوق عامل مبتنی بر جدول آنچه را که می خواهیم انجام می دهد، یعنی نگاشت مورد نیاز عامل را پیاده سازی می کند. یک نمونه از برنامه عامل مبتنی بر جدول در زیر آمده است.

(1) Table - Driven - Agent

```
function TABLE -DRIVEN -AGENT (percept) return action
static:
    percepts, a sequence,initially empty
    table , a table,indexed by percept sequences, initially
            fully specified
append percept to the end of percepts
action← lookup (percepts,table)
return action
```

رشته ی ادراکی	عملکرد
[A,Clean]	Right
[A,Dirty]	Suck
[B,Clean]	Left
[B,Dirty]	Suck
[A,Clean],[A,Clean]	Right
[A,Clean],[A,Dirty]	Suck
⋮	⋮



چهار نوع اصلی برنامه عامل که اصول زیربنایی تقریبا هر سیستم هوشمندی را تشکیل می دهند عبارتند از:

- **عاملهای واکنشی ساده[۳۳]**
- **عاملهای واکنشی مبتنی بر مدل[34]**
- **عاملهای مبتنی برهدف[35]**
- **عاملهای مبتنی بر سودمندی[۳۶]**

---

[33] Simple reflex agent
[34] Model-based
[35] Goal-based

۱-عاملهای واکنشی ساده:

ساده ترین نوع عامل، عامل واکنشی ساده است. این عاملها اقدامات را بر اساس ادراک فعلی انتخاب می کنند و تاریخچه ادراکات را نادیده می گیرند یعنی حافظه ندارند. برای مثال عامل جاروبرقی یک عامل واکنشی ساده است زیرا تصمیم آن تنها بر اساس محل فعلی محیط و اینکه آیا آن مکان دارای زباله است یا نه، می باشد. به خاطر حذف سابقه ادراک برنامه عامل در مقایسه با جدول آن بسیار کوچک است. بعنوان مثال در برنامه عامل جاروبرقی در مقایسه با جدول آن تعداد حالات ممکن از 4^T به 4کاهش می یابد.

انتخاب عمل بر اساس یک سری قوانین شرط – عمل انجام می شوند.

قوانین شرط – عمل[۳۷] دو نوع هستند: اکتسابی و غریزی.

اکتسابی: ماشین جلویی چراغ زد، توقف می کنیم

غریزی: شی به چشم نزدیک شود چشمهای خود را ناخودآگاه می بندیم.

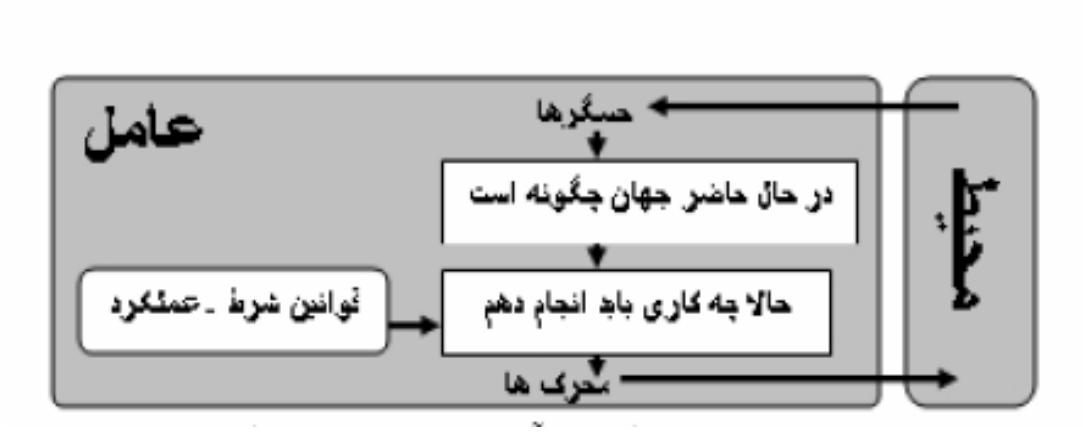
عاملهای واکنشی ساده این ویژگی قابل ستایش را دارند که ساده هستند ولی کاربرد و هوشمندی بسیار محدودی دارند. عامل واکنشی ساده تنها در صورتی کار می کند که بر اساس ادراک فعلی بتواند درست تصمیم بگیرد و این یعنی اینکه محیط کاملا رویت پذیر باشد. حتی اگر به مقدار اندکی عدم رویت پذیری وجود داشته باشد می تواند مشکلات بسیار جدی بروز کند. در مورد عاملهای واکنشی ساده که در محیط های نیمه روئیت پذیر کار می کنند حلقه های بی نهایت اغلب غیر قابل اجتناب هستند. اگر عامل بتواند اقداماتش را تصادفی کند گریز از حلقه های بی نهایت امکان پذیر است. برای مثال اگر عامل جاروبرقی تمیز را ادراک کند، برای انتخاب بین چپ و راست، سکه بیندازد که به طور متوسط در دو مرحله به مربع دیگر خواهد رسید. بنابر این یک عامل واکنشی ساده تصادفی شده ممکن است از یک عامل واکنشی ساده قطعی بهتر عمل کند. بنابراین در محیط های تک عاملی، تصادفی عمل کردن چندان منطقی نیست ولی در محیط های چند عامله عاقلانه است. برنامه عامل واکنشی ساده و ساختار آن در شکل زیر آمده است.

---

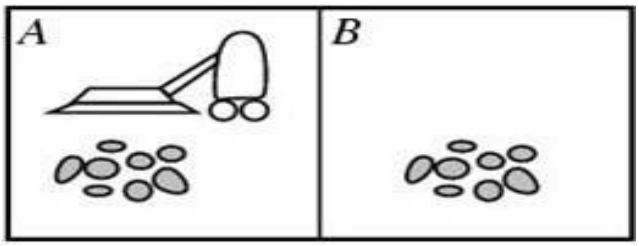
[36] Utility-based
[37] Condition-action Rule

```
function  SIMPLE _REFLEX_AGENT (percept) returns
                    action
static :
        Rules, a set of condition_action rules

state   <-- INTERPRET-INPUT (percept)
rule    <--  RULE-MATCH (state,rules)
action <-- RULE-ACTION [rule]
return action
```



مثالی از عامل واکنشی ساده:



function REFLEX-VACUUM-AGENT ([*location, status*])
return an action
if *status == Dirty* then return *Suck*
else if *location == A* then return *Right*
else if *location == B* then return *Left*

۲-عامل واکنشی مبتنی بر مدل:

همانطور که در روش قبل (عامل واکنشی ساده) اشاره شد گاهی اوقات محیط نیمه رویت پذیر است. موثر ترین روش برخورد عامل با محیط نیمه رویت پذیر نگه داشتن سوابق آن بخش از دنیا است که اکنون عامل نمی تواند آن را ببیند. این بدان معنی است که عامل باید به نوعی حالات داخلی را نگه داری کند که به تاریخچه ادراکات وابسته است. بهنگام سازی اطلاعات وضعیت داخلی همزمان با گذر زمان نیازمند دو نوع دانش کد شده در برنامه عامل است.

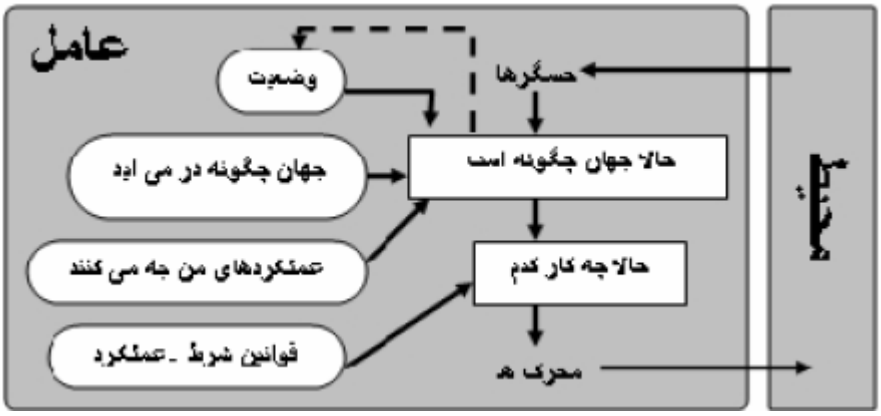
الف) نیازمند آن هستیم که برخی اطلاعات در باره چگونگی تغییر جهان، مستقل از عامل را داشته باشیم.

ب) نیازمند اطلاعات در مورد تاثیر اعمال خود عامل بر روی محیط می باشیم.

این دانش(دانش بخش ب) که درباره چگونگی عملکرد دنیا می باشد را مدل دنیا[۳۸] نامیده می شود. و عاملی که از چنین مدلی استفاده می کند مبتنی بر مدل نامیده می شود.

برنامه و دیاگرام عامل مبتنی بر مدل در شکل زیر آمده است:

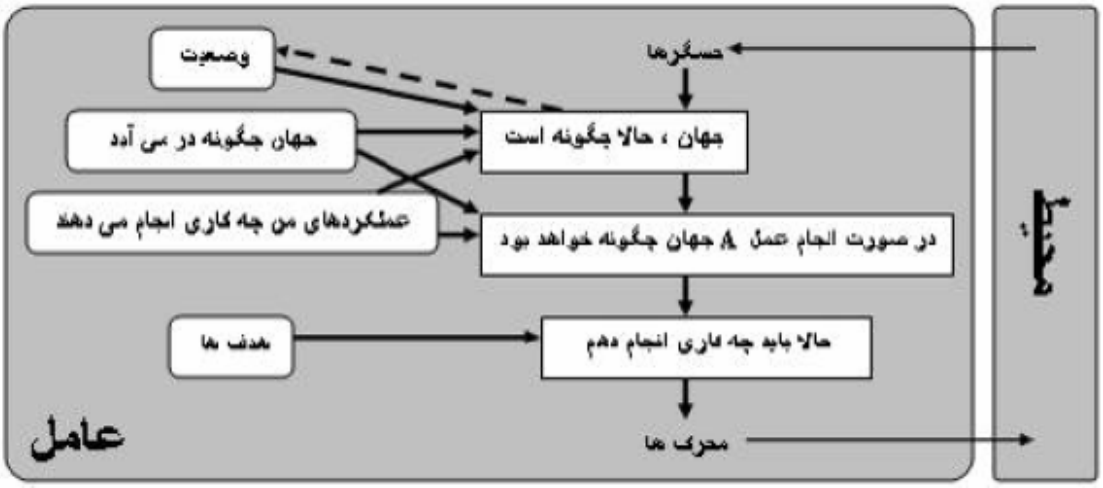
```
1 -  function  REFLEX_AGENT_WITH_STATE
                (percept) returns action
2 - static :
        State,a description of the current world state
        Rules,a a set of condition_action rules
3 - state   ← UPDATE-STATE  (state,percept)
4 - rule    ←RULE-MATCH (state,rules)
5 - action ← RULE-ACTION [rule]
6 - state   ←UPDATE-STATE (state,action)
7-  return action
```



[38] Model of the world

۳-عاملهای مبتنی بر هدف:

اطلاع از وضعیت فعلی محیط، همیشه برای تصمیم گیری در مورد اقدام بعدی کافی نیست. برای مثال در سر یک چهار راه تاکسی می تواند به چپ، راست، یا مستقیم برود. تصمیم درست به مقصد مسافر بستگی دارد. بنابراین همانگونه که عامل نیازمند دانستن وضعیت جاری و قبلی است نیاز به یک هدف نیز خواهد داشت تا تصمیم گیری های وی بر مبنای آن هدف جهت گیرد. رسیدن به هدف گاهی اوقات ساده و گاهی اوقات پیچیده است. در مواقعی که هدف ساده است، ارضای هدف بلافاصله بعد از انجام یک عمل نتیجه خواهد شد ولی در مواقعی که هدف پیچیده باشد، عامل باید دنباله اقدامات طولانی را برای رسیدن به هدف طی کند. در مواقع پیچیده جستجو[۳۹] و برنامه ریزی[۴۰] به یافتن دنباله ای از اعمال منجر خواهد شد.در مدل هدف گرا نمی توان از قوانین شرط – عمل گذشته تبعیت کرد بدین معنا که اگر فلان اتفاق افتاد آنگاه من چنان کنم. اگرچه بنظر می رسد عامل مبتنی بر هدف کمتر موثر باشد اما بسیار انعطاف پذیر است و می تواند خود را با شرایط محیطی وفق دهد و نیازی به دوباره نویسی قوانین شرط – عمل نیست.دیاگرام عامل مبتنی بر هدف در شکل زیر آمده است:



تفاوت عامل واکنشی و هدف گرا:

در طراحی عاملهای واکنشی، طراح برای حالات متفاوت عمل درست را پیش محاسبه می کند، ولی در عامل های هدف گرا عامل می تواند دانش خود را در مورد چگونگی واکنش، بهنگام سازی کنند.

---

[39] Search
[40] Planing

بنابراین این دو نوع برنامه فوق در موارد زیر با هم تفاوت دارند:

۱- برای عامل واکنشی ساده مجبور به دوباره نویسی تعداد زیادی قوانین شرط – عمل خواهیم بود.

۲- عامل هدف‌گرا نسبت به رسیدن به مقاصد متفاوت، انعطاف پذیر است

۳- به سادگی با تعیین یک هدف تازه، می توان عامل هدف گرا را به رفتاری تازه برسانیم.

۴-عاملهای مبتنی بر سودمندی:

اهداف به تنهایی برای ایجاد رفتاری با کیفیت و سودمندی بالا کافی نخواهد بود. برای مثال در رسیدن تاکسی به مقصد ممکن است دنباله زیادی از اعمال وجود داشته باشد تا به مقصد برسیم ولی بعضی از این مسیرها سریعتر، امن تر و ارزانتر از بقیه هستند. اهداف فقط بین حالات راضی و ناراضی تفاوت قائل می شوند و درباره اینکه یک حالت چقدر عامل را راضی می کند سخن نمی گویند. برای مقایسه بین حالت ها که میزان راضی بودن را تعیین می کند از تابع سودمندی استفاده می کنیم.

تابع سودمندی[۴۱]: یک حالت یا رشته ای از حالات را به یک عدد حقیقی که درجه رضایت نام دارد ،نگاشت می کند.

تعریف کامل تابع سودمندی امکان تصمیم گیری منطقی برای دو حالتی که اهداف ناکافی هستند را دارد.

الف)زمانیکه اهداف متناقض باشند تابع سودمندی می تواند موازنه خوبی برقرار کند. مثلا سرعت و ایمنی با هم در تناقض هستند.

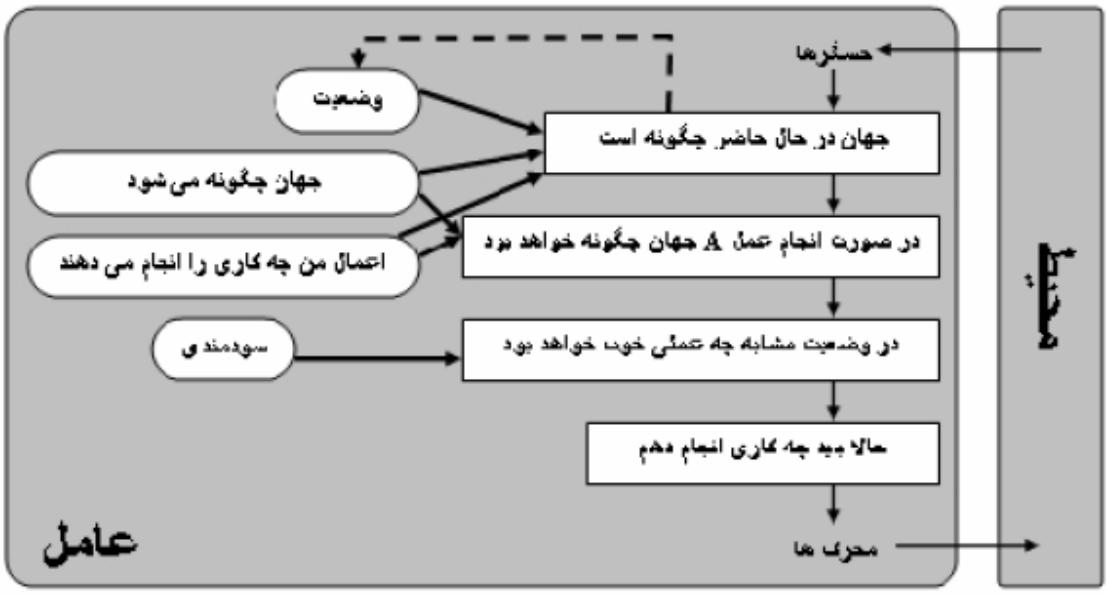
ب)زمانیکه چندین هدف وجود دارد عامل می تواند برای رسیدن به مقصد آنها را طی کند اما هیچ کدام از آنها با قطعیت قابل حصول نیست.

سودمندی روشی را فراهم می کند که در آن موفقیت عامل بر اساس اهمیت اهداف وزن دهی می شوند. مثلا یک تابع سودمندی یک تابع خطی است که در آن متغیرها اهداف می باشند و مقدار ضریب آنها بر اساس اهمیت اهداف تعیین می شوند.

---

[41] utility Function

دیاگرام عامل مبتنی بر سودمندی در شکل زیر آمده است:



تفاوت معیار کارایی و تابع سودمندی:

۱) معیار کارایی چگونگی و میزان موفقیت یک عامل را نشان می دهد ولی تابع سودمندی میزان سودمندی یک وضعیت از دنیا را از دیدگاه عامل بر می گرداند.

۲) معیار کارایی درجه موفقیت عامل است ولی تابع سودمندی درجه رضایت عامل است.

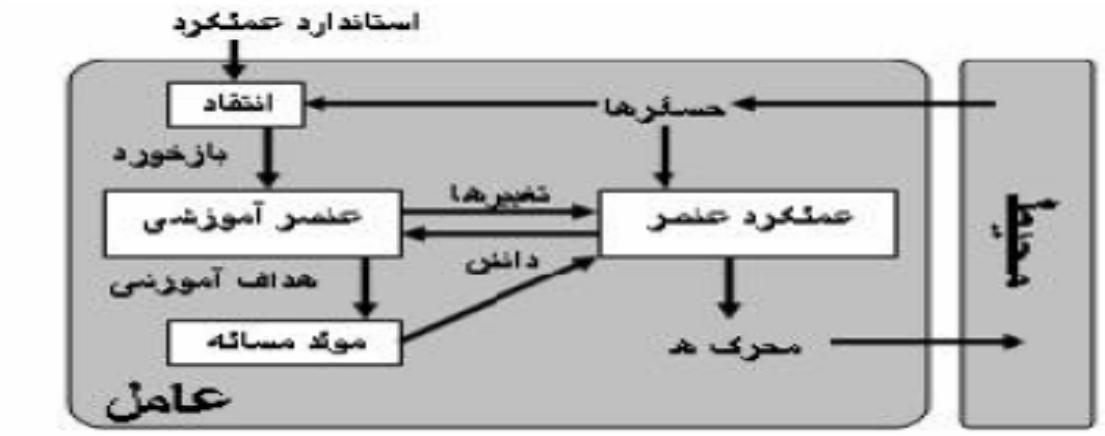
۳) معیار کارایی پارامتر مقایسه بین دو عامل است ولی تابع سودمندی پارامتر مقایسه بین دو وضعیت در رسیدن به هدف است.

عاملهای یادگیرنده :

عاملهای یادگیرنده شامل مولفه های عنصر کارایی، عنصر یاد گیرنده، مولد مسئله و منتقد می باشد.عنصر کارایی، مسئول انتخاب فعالیت های خارجی است. عنصر یادگیرنده، مسئول ایجاد بهبود هاست. منتقد، مسئول تولید بازخورد با توجه به استاندارد کارایی برای عنصر یادگیرنده است. مولد مسئله، مسئول پیشنهاد فعالیت هایی است که منجر به تجربیات آموزنده جدید می شود.

نکته: طراحی عنصر یادگیری بسیار به طراحی عنصر کارایی وابسته است.

دیاگرام یک عمل یادگیرنده در شکل زیر آمده است:



این چهار مولفه را بر روی مثال تاکسی خودکار بررسی می کنیم.

عنصر کارایی: عنصر کارایی شامل هر مجموعه از دانش ها در و روالها می باشد که تاکسی برای انتخاب اقدامات رانندگی اش در اختیار دارد. تاکسی با استفاده از این عنصر کارایی به جاده می رود و رانندگی می کند.

منتقد: منتقد دنیا را مشاهده می کند و اطلاعات را برای عنصر یادگیری می فرستد. برای مثال برای اینکه تاکسی بدون نگاه به عقب و راهنما زدن به سمت چپ می پیچد آنگاه منتقد شاهد کلمات زشت و زننده ایست که دیگر رانندگان آنرا به زبان می آورند.

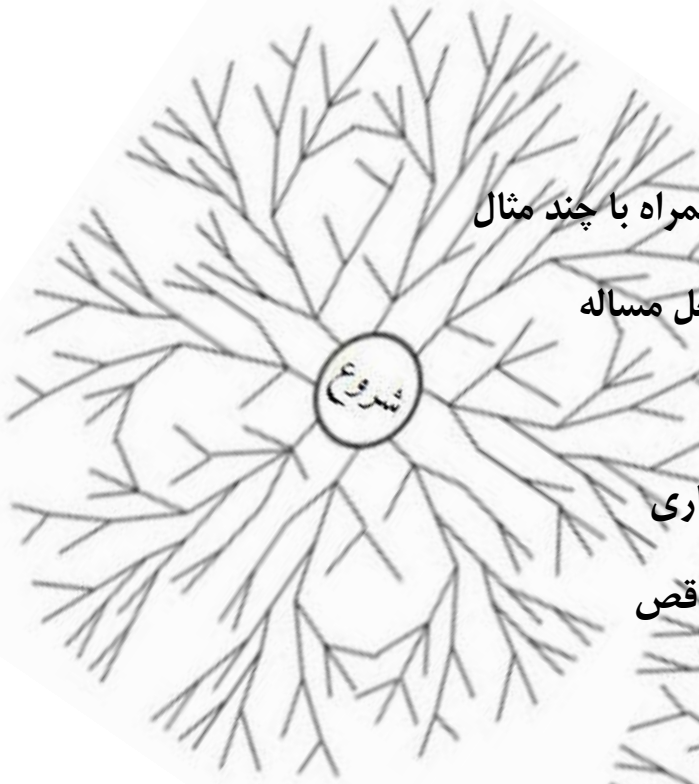
عنصر یادگیری:بر اساس این تجربه،عنصر یادگیری می تواند قاعده ای را تنظیم کند که بیانگر این است که در هنگام سبقت، راهنما بزند.عنصر کارایی با نصب این قاعده جدید ،تغییر خواهد کرد..

مولد مسئله:ممکن است زمینه های خاصی از رفتار را شناسایی کند که نیازمند اصلاح هستند مانند آزمایش هایی از قبیل امتحان ترمزها در جاده هایی با سطوح خشک و خیس.

فصل سوم: حل مسئله از طریق جستجو

آنچه در این فصل خواهید آموخت:

- **عامل های حل مساله**
- **فرموله بندی مساله همراه با چند مثال**
- **اندازه گیری کارایی حل مساله**
- **جستجوهای ناآگاهانه**
- **اجتناب از حالات تکراری**
- **جستجو با اطلاعات ناقص**



عامل‌های حل مسئله: نوعی از عامل‌های هدف‌گرا هستند که توسط یافتن ترتیب عملیات تصمیم می‌گیرند چه نوع عملی را انجام دهند تا به حالت مطلوب سوق پیدا کنند. مراحل زیر باید توسط یک عامل حل مسئله انجام شود:

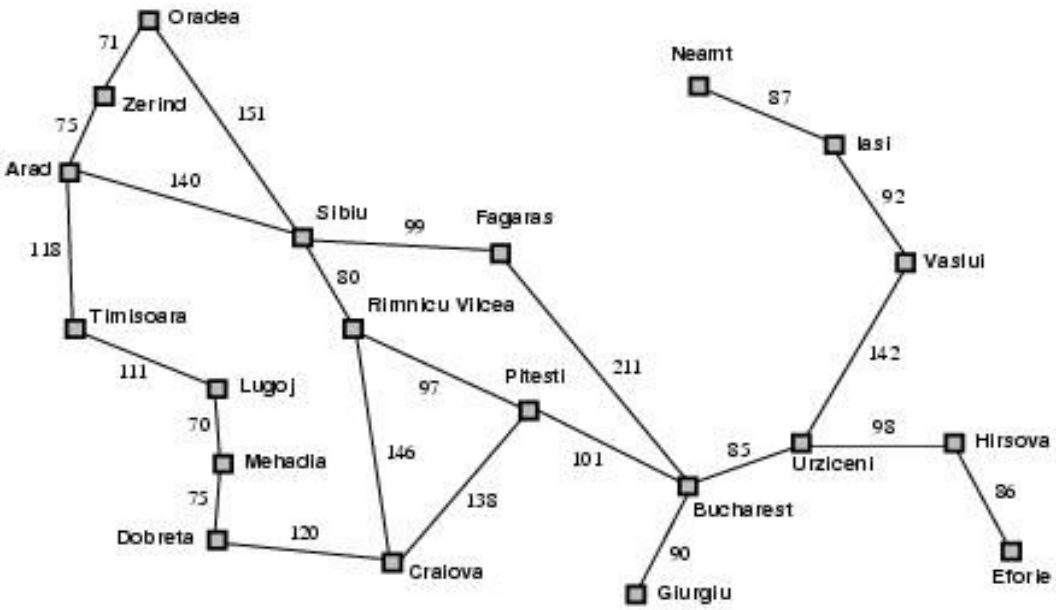
۱. **فرموله کردن (تدوین) هدف:**وضعیت‌های مطلوب نهایی کدامند.

۲. **فرموله کردن مسئله:** چه اقدامات و وضعیت‌هایی برای رسیدن به هدف موجود است.

۳. **جستجو:** در این مرحله عامل تصمیم می‌گیرد که چه رشته اعمالی می‌تواند وی را از حالت شروع به حالت هدف برساند. مسلما این رشته از اعمال از بین یک مجموعه اعمال ممکن انتخاب می‌شود، خروجی این مرحله یک راه‌حل[۴۲]است.

۴. **اجرا:** راه‌حلی که از مرحله قبل به دست آمده اجرا می‌شود.

مثال: نقشه رومانی:



صورت مسئله: رفتن از آراد به بخارست

فرموله کردن هدف: رسیدن به بخارست.

فرموله کردن مسئله:

- وضعیت‌ها: شهرهای مختلف
- فعالیت‌ها: حرکت بین شهرها

[42] solution

جستجو: دنباله‌ای از شهرها مثل آراد، سیبیو، فاگارس، بخارست. (جستجو با توجه به کم‌هزینه‌ترین مسیر)

تعریف مسئله: مجموعه‌ای از اطلاعات است که عامل از آنها برای این‌که چه عملی را انجام می‌دهد، استفاده می‌کند. یک مسئله با موارد زیر تعریف می‌شود:

۱. **حالت اولیه:** حالتی که عامل از آن شروع می‌کند. در مثال رومانی شهر آراد n(arad)

۲. **تابع جانشین:** توصیفی از حالت‌های ممکن که برای عامل مهیاست.

S(arad)={zerind-sibiu-timisoara}

۳. **فضای حالت:** مجموعه‌ای از حالت‌ها که از حالت اولیه می‌توان به آنها رسید. در مثال رومانی: کلیه شهرهایی که با شروع از آراد می‌توان به آنها رسید.

نکته: فضای حالت= حالت اولیه + تابع جانشین.

۴. **آزمون هدف:** تعیین می‌کند که آیا حالت خاصی حالت هدف است یا خیر.دو نوع هدف داریم:

- هدف صریح: در مثال رومانی رسیدن به بخارست.
- هدف انتزاعی (ضمنی): در مثال شطرنج رسیدن به حالت کیش و مات.

مسیر: دنباله‌ای از حالت‌ها که دنباله‌ای از فعالیت‌ها را به هم متصل می‌کند. در مثال رومانی: Arad, Sibiu, Fagaras یک مسیر است.

۵. **هزینه مسیر:** برای هر مسیر یک هزینه عددی در نظر می‌گیرد. در مثال رومانی: طول مسیر بین شهرها بر حسب کیلومتر.

راه حل مسئله: مسیری از حالت اولیه به حالت هدف است. راه‌حل بهینه کمترین هزینه را دارد.

نکته: فرآیند حذف جزییات از یک توصیف، تجرید[۴۳] یا انتزاع نام دارد. به عنوان نمونه در مثال رومانی یک سری از توصیفات در دنیای واقعی وجود دارند که به مسئله پیدا کردن یک مسیر به بخارست ربطی ندارد. مثلاً همراهان مسافر، منظره بیرون از پنجره، فاصله ایستگاه تا پمپ‌بنزین و ...

یک تجرید معتبر است اگر بتوانیم هر راه‌حل خلاصه را به یک راه‌حل درونیابی با جزئیات بیشتر بسط دهیم.

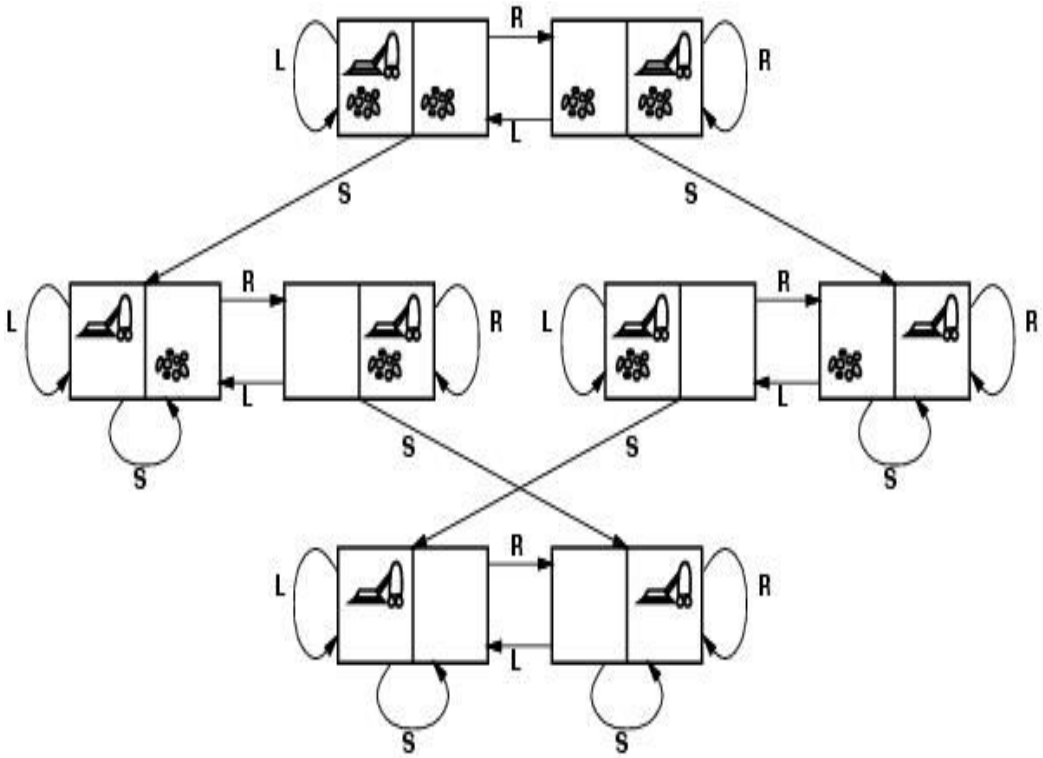
یک تجرید مفید است اگر هریک از اعمال نسبت به مسئله اصلی آسان‌تر انجام شوند.

---

[43] Abstraction

فرموله کردن چند مسئله نمونه:

مسئله دنیای جاروبرقی:



حالتها: دو مکان که هر یک ممکن است کثیف یا تمیز باشند.لذا ۸ = ۲^۲* ۲حالت در این جهان وجود دارد.

حالت اولیه: هر حالتی میتواند به عنوان حالت اولیه طراحی شود.

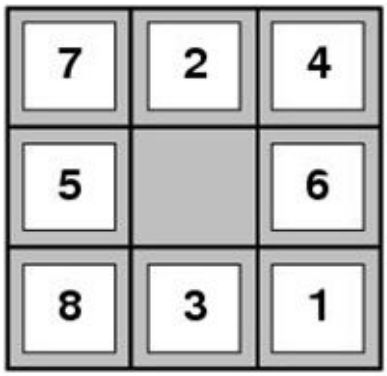
تابع جانشین: حالتهای معتبر از سه عملیات: راست، چپ، مکش.

آزمون هدف: تمیزی تمام مربعها.

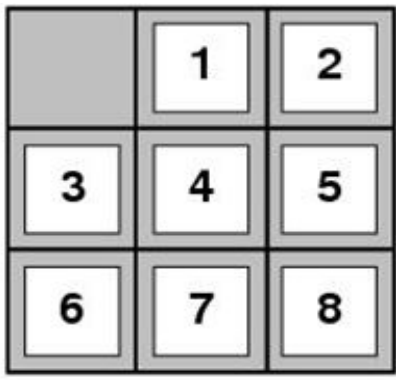
هزینه مسیر: تعداد مراحل در مسیر.

نکته: اگر محیطی n محل داشته باشد n*2^n حالت خواهد داشت.

مسئله معمای ۸:



Start State



Goal State

حالتها: مکان هر هشت خانه شماره دار و خانه خالی در یکی از ۹ خانه

حالت اولیه: هر حالتی را میتوان به عنوان حالت اولیه در نظر گرفت

تابع جانشین: حالتهای معتبر از چهار عمل، انتقال خانه خالی به چپ، راست، بالا یا پایین

آزمون هدف: بررسی می‌کند که حالتی که اعداد به ترتیب چیده شده‌اند (طبق شکل روبرو) رخ داده یا نه

هزینه مسیر: برابر با تعداد مراحل در مسیر.

مسئله ۸ وزیر:

هدف در این مسئله قرار دادن ۸ وزیر روی صفحه شطرنج است به نحوی که هیچ وزیری به دیگری حمله نکند.

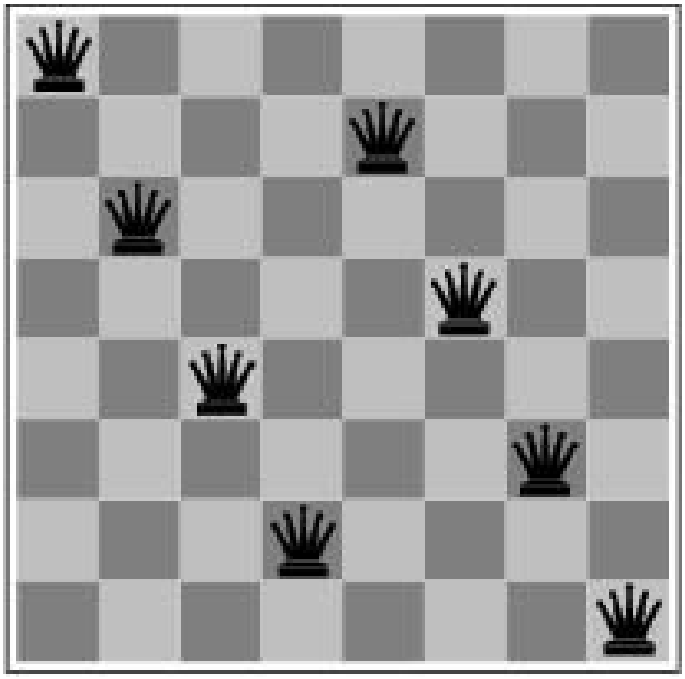
برای این مسئله دو نوع فرموله‌سازی وجود دارد:

۱. فرموله سازی افزایشی[۴۴] : که با صفحه خالی شروع شده و در هر عمل یک وزیر به صفحه اضافه می‌شود.

۲. فرموله سازی حالت کامل[۴۵]: با همه ۸ وزیر روی صفحه شروع می‌کند و در هر عمل یک وزیر به اطراف حرکت داده می‌شود.

---

[44] incremental formulation
[45] complete state formulation



فرمول بندی افزایشی

حالت‌ها: هر ترتیبی از ۰ تا ۸ وزیر در صفحه، یک حالت است

حالت اولیه: هیچ وزیری در صفحه نیست

تابع جانشین: وزیری را به خانه خالی اضافه می‌کند

آزمون هدف: ۸ وزیر در صفحه وجود دارند و هیچ کدام به یکدیگر گارد نمی‌گیرند

در این فرمول بندی باید ۱۴^۱۰*۳ دنباله ممکن بررسی می‌شود

فرمول بندی حالت کامل

حالت‌ها: چیدمان n وزیر $(0\leq n\leq 8)$، بطوری که در هر ستون از n ستون سمت چپ، یک وزیر قرار گیرد و هیچ دو وزیری بهم گارد نگیرند

حالت اولیه: با ۸ وزیر در صفحه شروع می‌شود

تابع جانشین: وزیری را در سمت چپ‌ترین ستون خالی قرار می‌دهد، بطوری که هیچ وزیری آن را گارد ندهد.

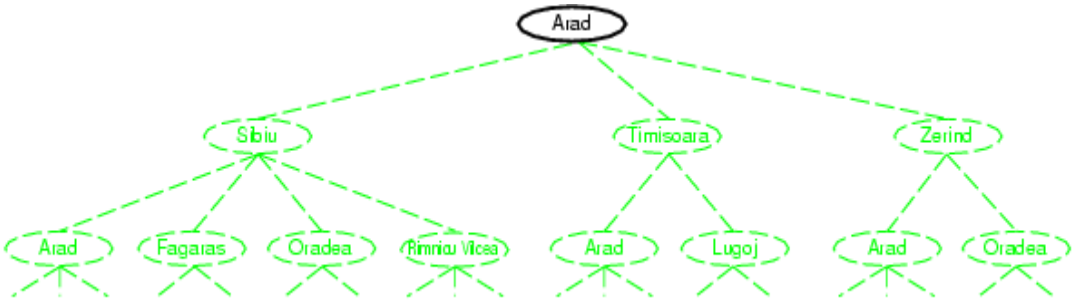
آزمون هدف: ۸ وزیر در صفحه وجود دارند و هیچ کدام به یکدیگر گارد نمی‌گیرند

این فرمول بندی فضای حالت را از ۱۴^۱۰*۳ به ۲۰۵۷ کاهش می‌دهد.

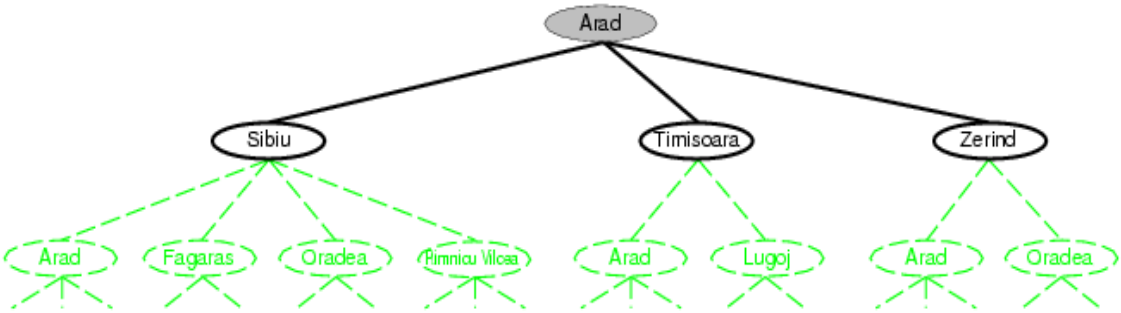
در فرموله سازی حالت کامل، فضای حالت از ۳*۱۰^۱۴ به ۲۰۵۷ کاهش می‌یابد و پیدا کردن راه‌حل در فضای حالت کامل راحت‌تر از افزایشی است. برای فرموله کردن مسئله ۱۰۰ وزیر در افزایشی ۱۰^۴۰۰ حالت و در روش فرموله سازی حالت کامل ۱۰^ ۵۲ برای پیدا کردن راه حل باید بررسی شود.

گسترش[۴۶]:

به معنی بکارگیری تابع مابعد برای یک حالت و تولید یک مجموعه جدید ازحالات می‌باشد.مجموعه نودهایی که تولید شده‌اند اما هنوز گسترش نیافته‌اند مجموعه حاشیه[۴۷] نامیده می‌شود و هر عنصر از مجموعه حاشیه یک نود برگ است یعنی نودی که هنوز هیچ مابعدی در درخت جستجو ندارد.



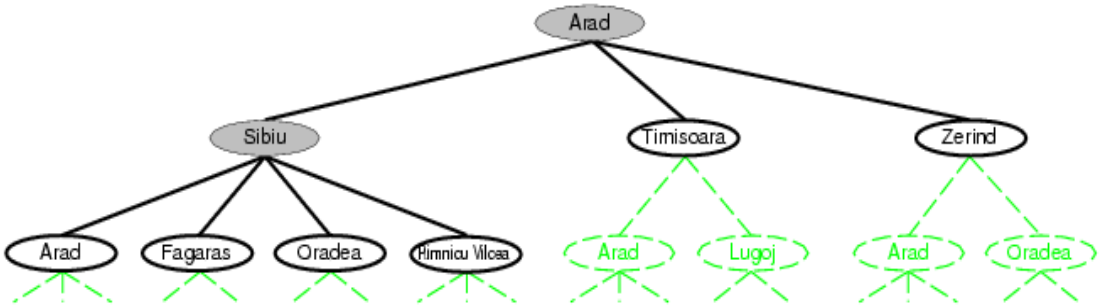
الف) حالت اولیه



(ب) پس از گسترش آراد

---

[46] expand
[47] fringe



(پ) پس از گسترش سیبیو

روش‌های مختلفی برای نمایش نودها وجود دارد در یکی از این روش ها فرض میشود یک نود شامل ساختاری با پنج جز می‌باشد:

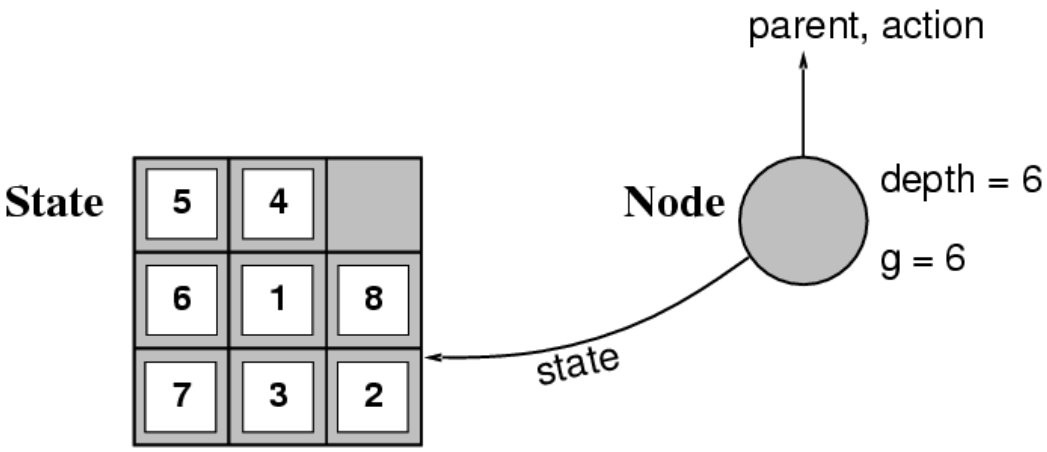
State: حالتی در فضای حالت که متناظر با نود مورد نظر است (حالت نود در فضای حالت).

Parent state: نودی از درخت جستجو که این نود را ایجاد کرده است.

Action: عملی که روی نود پدر برای تولید این نود به کار برده شده است.

Path cost: هزینه مسیر از حالت شروع تا این نود که با g(n) نمایش داده می‌شود.

Depth: تعداد اعمالی که در طول مسیر از حالت شروع تا این نود انجام شده است.



جستجو تابعی است که در هر مرحله یک نود را برای گسترش از مجموعه fringe انتخاب میکند بهترین روش برای پیاده سازی تابع جستجو استفاده از یک صفمی باشد. عملیات روی صف مورد نظر عبارتنداز :

- (Make-queue (element: با استفاده از عنصر یا عناصر داده شده، یک صف تولید می کند .
- (Empty ? (queue: در صورتی که هیچ عنصری در صف نباشد مقدار true بر می گرداند .
- (First (queue: اولین عنصر صف را بر می گرداند .
- (Remove-first (queue : اولین عنصر صف را بر می گرداند و آن را از صف حذف می کند .
- (Insert (element,queue: یک عنصر را وارد صف می کند و صف حاصل را بر می گرداند .
- (Insert-all (element,queue: یک مجموعه از عناصر را وارد صف می کند سپس صف تغییر یافته را بر می گرداند .

اندازه گیری کارایی حل مسئله :

خروجی الگوریتم جستجو راه حل[48] یا شکست[49] است . کارایی الگوریتم جستجو با چهار معیار زیر ارزیابی می شود :

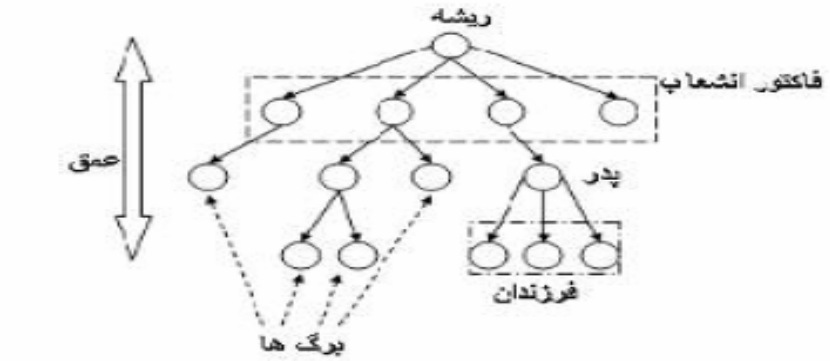
- **کامل بودن**[50] : آیا الگوریتم تضمین می کند که در صورت وجود راه حل، راه حل را پیدا کند؟
- **بهینگی**[51]: آیا الگوریتم تضمین می کند که از بین چندین راه حل،راه حل بهینه یا کم هزینه ترین را پیدا کند؟
- **پیچیدگی زمانی**[52] : چه مدت زمانی طول می کشد تا الگوریتم جستجو،راه حل راپیدا کند؟
- **پیچیدگی فضایی**[53] :الگوریتم جستجو برای پیدا کردن راه حل،چقدر حافظه نیاز دارد؟

پیچیدگی زمانی و مکانی بر اساس سه مفهوم زیر بیان می شود :

b : حداکثر فاکتور انشعاب درخت جستجو .

d : عمق کم هزینه ترین راه حل .

m : حداکثر عمق فضای حالت که می تواند بی نهایت نیز باشد .



[48] Solution
[49] Failure
[50] Completeness
[51] Optimally
[52] Time complexity
[53] Space complexity

معمولاً پیچیدگی زمانی بر اساس تعداد نودهای تولید شده در حین جستجو و پیچیدگی مکانی بر اساس ماکزیمم تعداد نوهای ذخیره شده در حافظه ا ندازه گیری می شود . برای ارزیابی کارایی یک الگوریتم جستجو می توان فقط هزینه جستجو که مجموع پیچیدگی زمانی و مکانی یا هزینه کل را در نظر گرفت که هزینه کل مجموع هزینه جستجو ومجموع هزینه راه حل می باشد .

نکته: هزینه کل=هزینه جستجو+هزینه مسیر

پس از تعریف یک مسئله و فرموله سازی آن باید راه حلی برای آن تشخیص داده شود که این راه حل بر اساس جستجو در فضای حالت پیدا خواهد شد.
استراتژی های جستجو از نظر اینکه آیا از تعداد مراحل یا هزینه مسیر از حالت جاری به حالت هدف اطلاعاتی دارند یا نه، به دو دسته تقسیم می شوند :
الف) جستجوهای نا آگاهانه[۵۴] : این دسته از الگوریتم های جستجو، هیچ اطلاعاتی غیر از تعریف مسئله در اختیار ندارند و فقط می توانند جانشین هایی را تولید و هدف را از غیر هدف تشخیص دهند که به آن جستجوهای کور کورانه[۵۵] نیز گفته می شود .
ب) جستجو های آگاهانه[۵۶] : این دسته از الگوریتم ها علاوه بر اطلاعات مسئله اطلاعات اضافی برای رسیدن به هدف در اختیار دارند که می توانند امید بخش تر بودن یک گره نسبت به گره دیگر را با توجه به اطلاعات اضافی تشخیص دهند که به این دسته جستجو ها، جستجو های اکتشافی[۵۷] نیز گفته می شود .
انواع جستجوهای نا آگاهانه عبارتند از:

- جستجوی اول-سطح[۵۸]
- جستجوی هزینه یکنواخت[59]
- جستجوی اول-عمق[60]
- جستجوی عمقی محدود شده[۶۱]
- جستجوی عمقی تکرار شونده[۶۲]
-

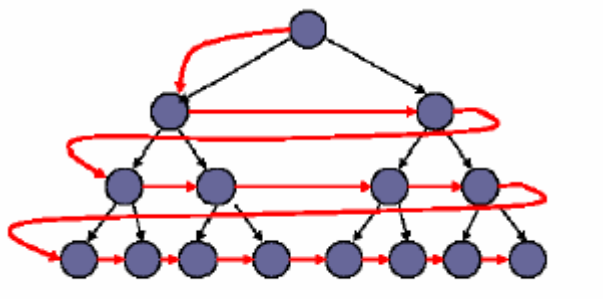
جستجوی اول-سطح :

در جستجوی اول سطح(bfs)، ابتدا نود ریشه گسترش یافته سپس همه مابعد های نود ریشه، و بعد از آن مابعد های آن و ... گسترش می یابد . بطور کلی نودهای یک سطح از درخت جستجو قبل از نود های سطح بعدی گسترش می یابند .

---

[54] Uninformed search
[55] blind
[56] Informed search
[57] heurestic
[58] Breadth-First-Search
[59] Uniform-Cost-Search
[60] Depth-First-search
[61] Limited-Depth-Search
[62] Iteration-Depth-Search

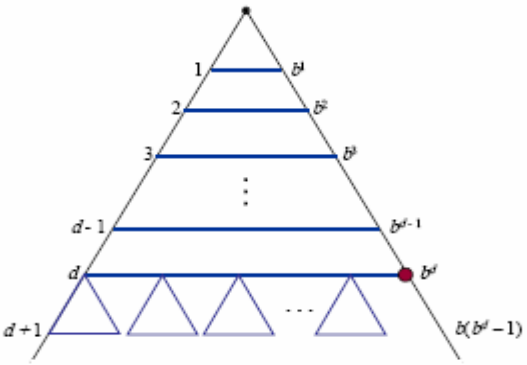
فراخـوانی(Tree-search(problem,FIFOqueue منجـر بـه اجـرای جـستجوی سـطح اول سـطح می شود.



شکل ۲-۲: جستجوی سطحی

صف FIFO ، مابعد های تولیـد شـده جدیـد را در انتهـای صـف قـرار مـی دهـد . بـدین ترتیـب نـود هـای کم عمق تر قبل از نود هـای عمیـق تـر گـسترش مـی یابنـد . اگـر کـم عمـق تـرین نـود هـدف عمـق در عمق متناهی d و فـاکتور انـشعاب b متنـاهی باشـد جـستجوی bfs حتمـا هـدف را بعـد از گـسترش همـه نود های کم عمق تر از d پیـدا مـی کنـد . لـذا ایـن روش کامـل اسـت اگـر تـابع هزینـه مـسیر یـک تـابع غیر کاهشی از عمـق نـود باشـد ایـن الگـوریتم بهینـه نیـز مـی باشـد. بـه عنـوان مثـال اگـر همـه اعمـال هزینه یکسان داشته باشند ایـن جـستجو هـدف را پیـدا میکنـد . لازم بـذکر اسـت ایـن الگـوریتم بـا وجـود کامل بودن همیشه کارا نیست . یعنی پیچیدگی مکانی و زمانی بسیار بالایی دارد .

پیچیدگی زمانی و حافظه جستجوی سطحی



$$b + b^2 + \ldots + b^d + b(b^d - 1) = O(b^{d+1})$$

هر نودی که تولید می شود باید در حافظه بمانـد زیـرا هـر نـود قـسمتی از مجموعـه حاشـیه اسـت یـا یـک جد برای نود های برگ است بنـابراین پیچیـدگی مکـانی نیـز ماننـد پیچیـدگی زمـانی اسـت .ویژگـی هـای الگوریتم جستجوی اول-سطح را در موارد زیر می توان خلاصه کرد:

- **کامل بودن:**بله
- **بهینگی:**بله(مشروط)در صـورتی بهینـه اسـت کـه هزینـه مـسیر تـابعی غیـر نزولـی از عمـق گـره باشد(مثل وقتی که اعمال هزینه یکسانی دارند)
- **پیچیدگی زمانی:** $O(b^{d+1})$
- **پیچیدگی فضا:** $O(b^{d+1})$

با توجه به اطلاعات موجود در جدول زیر، نیازمندیهای حافظه جستجوی اول-سطح بسیار بغرنج تر از پیچیدگی زمانی است . ۳۱ ساعت، زمان زیادی برای حل یک مسئله خیلی مهم با عمق هدف ۸ نیست اما تعداد کمی از کامپیو ترها دارای حافظه اصلی یک ترا بایتی هستند .علاوه بر آن نیازمندی های زمان هم یک فاکتور مهم می باشد . به عنوان مثال، اگر راه حل در عمق ۱۲ قرار داشته باشد آنگاه طبق فرضیات جدول زیر، جستجو ۳۵ سال طول خواهد کشید .

زمان و فضای لازم در جستجوی سطحی

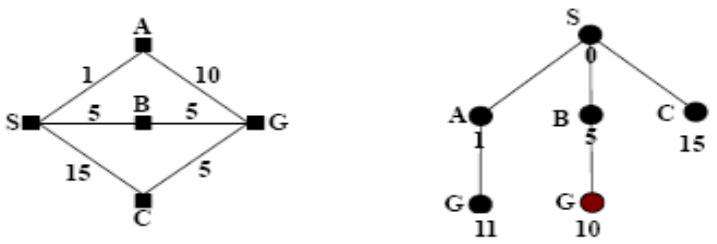
عمق	تعداد گره ها	زمان	حافظه
۲	۱۱۰۰	۱۱/۰ ثانیه	۱ مگا بایت
۴	۱۱۱۱۰۰	۱۱ ثانیه	۱۰۶ مگا بایت
۶	$۱۰^{۷}$	۱۹ دقیقه	۱۰ گیگا بایت
۸	$۱۰^{۹}$	۳۱ ساعت	۱ ترا بایت
۱۰	$۱۰^{۱۱}$	۱۲۹ روز	۱۰۱ ترا بایت
۱۲	$۱۰^{۱۳}$	۳۵ سال	۱۰ پتا بایت
۱۴	$۱۰^{۱۵}$	۳۵۲۳ سال	۱ هگزا بایت

- b =۱۰
- ۱۰۰۰۰ گره در هر ثانیه
- هر گره ۱۰۰۰ بایت

جستجوی هزینه یکنواخت:

جستجوی اول-سطح ، فقط وقتی هزینه همه اعمال یکسان است بهینه می باشد زیرا این روش جستجو کم عمق ترین نود را گسترش می دهد . الگوریتم جستجو با هزینه یکنواخت با هر تابع هزینه مسیری بهینه است این الگوریتم جستجو نودی را انتخاب می کند که کمترین g را دارد (g(n) هزینه رسیدن از حالت شروع به نود n)یعنی نود با کمترین هزینه مسیر را انتخاب می کند . توجه کنید که اگر هزینه همه اعمال یکسان باشد این روش جستجو با جستجوی اول سطح یکسان می شود . اگر این روش نودی را گسترش دهد که انجام یک عمل در آن با هزینه صفر منجر به برگشت به همان حالت شود مانند عمل NoOp در یک حلقه نامتناهی گیر می کند .

مثال: جستجوی هزینه یکنواخت



0: [S(0)]
1: [A(1), B(5), C(15)]
2: [B(5), G(11), C(15)]
3: [G(10), G(11), C(15)]
4: [G(11), C(15)]

اگـر هزینـه هـر عمـل بزرگتـر یـا مـساوی یـک ثابـت کوچـک مثـل ε باشـد کامـل بـودن ایـن جـستجو تضمین می شود همین شرط برای اطمینان از بهینگی نیزکافیست .

فرض کنید C^*کل هزینه از گـره آغـاز تـا هـدف باشـد و هـر عمـل حـد اقـل ε تـا هزینـه داشـته باشـد در بدترین حالت پیچیدگی مکانی وزمانی($O(b^{(C^*/\varepsilon)})$ می باشد .

نکته : اگر هر عملگر هزینه غیر منفی داشـته باشـد هزینـه یـک مـسیر بـا ادامـه آن کـاهش پیـدا نخواهـد کرد و الگوریتم میتواند کم هزینه تـرین مـسیر را پیـدا کنـد . اگـر بعـضی از عمـل هـا هزینـه منفـی داشـته باشند باید جستجویی از تمام گره ها صورت گیرد تا راه حل بهینه پیدا شود .

جستجوی عمقی[۶۳]:

این روش جستجو در هر مرتبه، عمیق ترین گره در مجموعه حاشیه(گره های گسترش نیافته) درخت جستجو را گسترش می دهد . اگر عمیق ترین نود فرزندی نداشته باشد جستجو به سمت عمیق ترین نودی که هنوز گسترش نیافته برمی گردد .این جستجو می تواند توسط الگوریتم Tree _search با یک صف LIFO پیاده سازی شود. علاوه بر استفاده از پیاده سازی از طریق Tree _search ، معمولا جستجوی اول عمق، با یک تابع بازگشتی پیاده سازی می شود .

جستجوی اول عمق به حافظه کمی نیاز دارد. این روش در هر لحظه یک مسیر از ریشه تا یک برگ را به همراه نودهای همزاد[۶۴] هر نود موجود در مسیر را در حافظه نگه می دارد . یک نود گسترش یافته به محض اینکه همه نوادگانش کاملا بررسی شدند از حافظه خارج می شود .

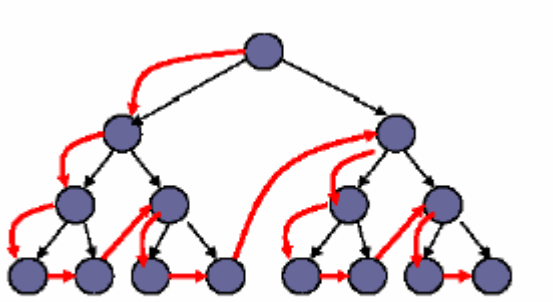
برای هر فضای حالت با فاکتور انشعاب b و عمق ماکزیمم m حستجوی اول عمق فقط به اندازه bm+1 نود حافظه نیاز دارد . نوع دیگر از جستجوی اول عمق ، جستجوی عقبگرد[۶۵] نامیده می شود که حافظه کمتری استفاده می کند .در این جستجو در هر لحظه به جای تولید همه فرزندان ، فقط یک فرزند تولید می شود . هر نود جزئی گسترش یافته به خاطر می سپارد که کدام فرزندانش باید بعدا گسترش یابند. در این حالت پیچیدگی حافظه o(m) می باشد .

اشکال جستجوی اول- عمق این است که ممکن است با یک انتخاب نادرست در یک مسیر خیلی طولانی یا حتی نامتناهی گیر کند در حالیکه یک راه حل، نزدیک به ریشه درخت وجود دارد اگر زیر درخت چپ دارای

---

[63] Depth First Search
[64] Sibling
[65] Back-tracking search

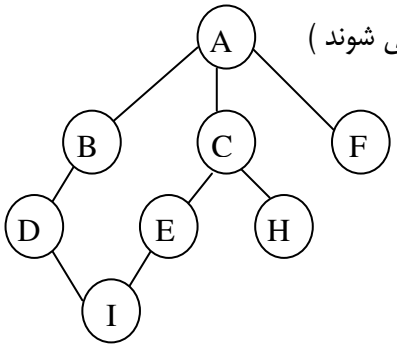
عمق نامحدود باشد و شامل هیچ راه حلی نباشد جستجوی اول- عمق هرگز تمام نمی شود. لذا این روش جستجو، با این فرضیات، کامل نیست . در بدترین حالت جستجوی اول-عمق همه نودهای درخت جستجو را تولید خواهد کرد. یعنی پیچیدگی زمانی آن $O(b^m)$ می باشد . یکی دیگر از مشکلات جستجوی اول-عمق حلقه بی پایان می باشد .مثالی از جستجوی اول-عمق در شکل زیر نشان داده شده است.



شکل ۲-۳: جستجوی عمقی

مثال :

اگر در گراف زیر جستجوی اول- عمق را از راس c شروع کنیم ترتیب رویت شدن گره ها از چپ به راست چگونه خواهد بود .(فرض کنید فرزندان یک گره بر اساس ترتیب حروف الفبا انتخاب می شوند)



C	H	E	A	F	B	D	I	

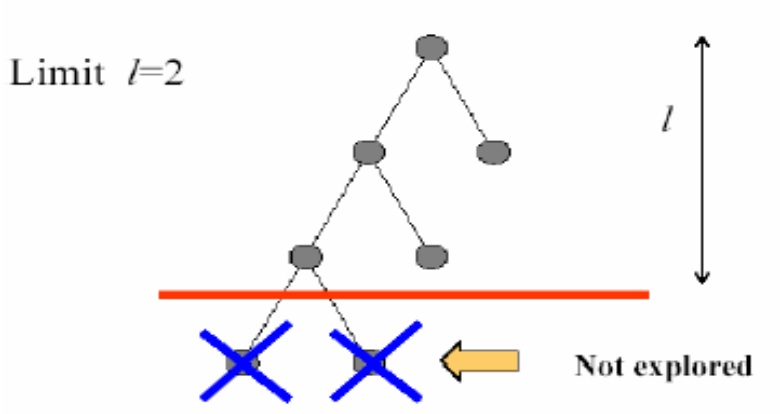
Out put : C A B D I F E H

جستجوی عمقی محدود شده[۶۶]:

مشکل درختهای نامحدود با مشخص کردن حدی برای عمق، حل می شود. وقتی درخت را به عمق L محدود کنیم یعنی فرض کرده ایم نودها در عمق L فرزندی ندارند این روش جستجوی عمقی محدود شده نامید ه میشود . اگر کم عمق ترین هدف بعد از عمق برش قرار گیرد یعنی L<d باشد این روش جستجو کامل نیست. اگر L>=d باشد جستجوی عمقی محدود شده کامل است اما همچنان بهینه نیست. پیچیدگی زمانی $O(b^l)$ و پیچیدگی مکانی O(bl) می باشد .جستجوی اول عمق نوع خاصی از جستجوی عمقی محدود شده با $L= \infty$ می باشد. گاهی اوقات عمق برش بر اساس دانش مسئله تعیین می شود به عنوان مثال در

[66] Depth Limited Search

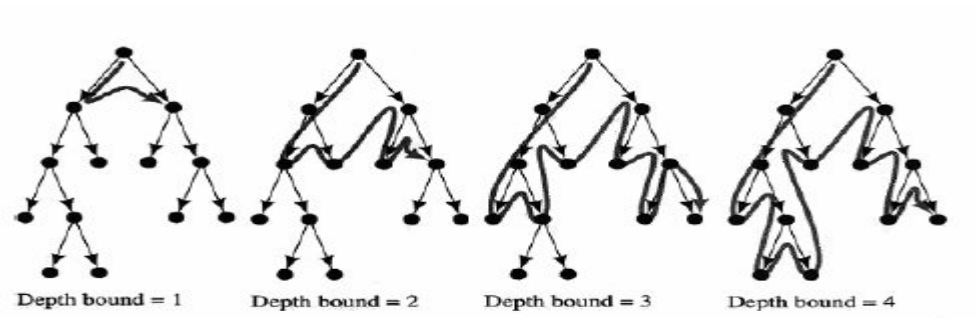
نقشه رومانی ۲۰ شهر وجود دارد. لذا اگر راه حلی وجودداشته باشد ، بیشترین طولی که می تواند داشته باشد ۱۹ است و L= 19 یک انتخاب ممکن است .



شکل ۲-۷: جستجوی عمقی محدود شده

جستجوی عمیق شونده تکراری[۶۷]:

این روش در واقع همان جستجوی عمقی محدود شونده است که برای یافتن بهترین عمق برش چندین بار اجرا می شود . جستجوی عمقی تکرار شونده این کار را با افزایش آهسته عمق برش انجام می دهد. ابتدا عمق ۱ سپس ۲ و .... تا یک هدف پیدا شود. عمق برش که به d می رسد هدف پیدا خواهد شود . این روش جستجو، جستجوی اول عمق و اول سطح را ترکیب می کند که مانند جستجوی اول سطح با فاکتور انشعاب متناهی کامل است و وقتی هزینه مسیر یک تابع غیر کاهش از عمق نود باشد بهینه است و پیچیدگی مکانی این روش مانند جستجوی عمقی خطی می باشد. پیچیدگی مکانی آن o(bd) است . زیرا در آخرین تکرار حداکثر عمق درخت d می باشد .



شکل ۲-۸: جستجوی عمقی تکرار شونده

[67] Iterative-Depth Search

در جستجوی عمقی تکرار شونده نودها در آخرین سطح یکبار، در سطح ماقبل آن دوبار، و .....تولید می شود .

بنابراین تعداد کل نودهای تولید شده عبارتست از :

پیچیدگی زمانی عمیق کننده تکراری

DLS ($l = 0$)	0
DLS ($l = 1$)	b^1
DLS ($l = 2$)	$b^1 + b^2$
.	.
.	.
.	.
DLS ($l = d - 1$)	$b^1 + b^2 + b^3 + \ldots + b^{d-1}$
DLS ($l = d$)	$b^1 + b^2 + b^3 + \ldots + b^{d-1} + b^d$

سربار $\leq N_{DLS\,(l=d)}$

$$N_{IDS} = db^1 + (d-1)b^2 + (d-2)b^3 + \ldots + 2b^{d-1} + b^d$$

کارآیی IDS

- تعداد گره های تولید شده توسط DLS در عمق d با فاکتور انشعاب b:

$N_{DLS} = b + b^2 + \ldots + b^{d-1} + b^d$

- تعداد گره های تولید شده توسط IDS در عمق d با فاکتور انشعاب b:

$N_{IDS} = db^1 + (d-1)\, b^2 + \ldots + 2b^{d-1} + 1b^d$

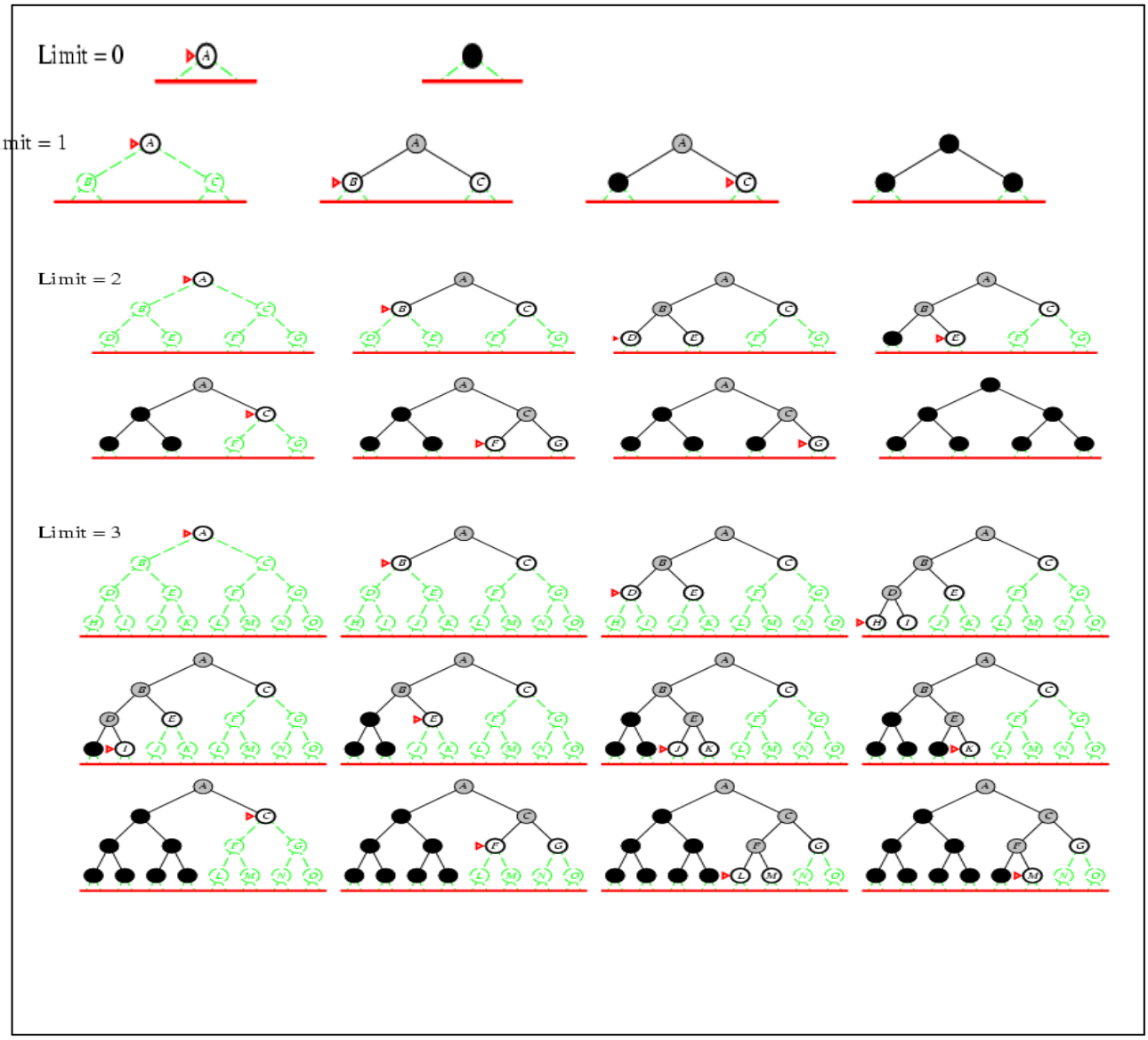
- اگر $b = 10$ و $d = 5$:

$N_{DLS} = 10 + 100 + 1{,}000 + 10{,}000 + 100{,}000 = 111{,}110$

$N_{IDS} = 50 + 400 + 3{,}000 + 20{,}000 + 100{,}000 = 123{,}450$

بنابراین جستجوی عمقی تکرارشونده علی رغم تولید تکراری حالات سریع تر از جستجوی اول سطح می باشد .به طور کلی اگر فضای حالت بزرگ و عمق راه حل نامشخص باشد در میان جستجوهای ناآگاهانه، جستجوی عمقی تکرار شونده ترجیح داده میشود اگر خصوصیت تکرار شونده این جستجو در جستجو با هزینه یکنواخت استفاده شود منجر به کاهش نیازمندیهای حافظه آن خواهد شد. البته در این مورد به جای محدودیت عمق از محدودیت مسیر استفاده می شود.

مثالی از جستجوی عمقی تکرار شونده در شکل زیر بیان شده است.

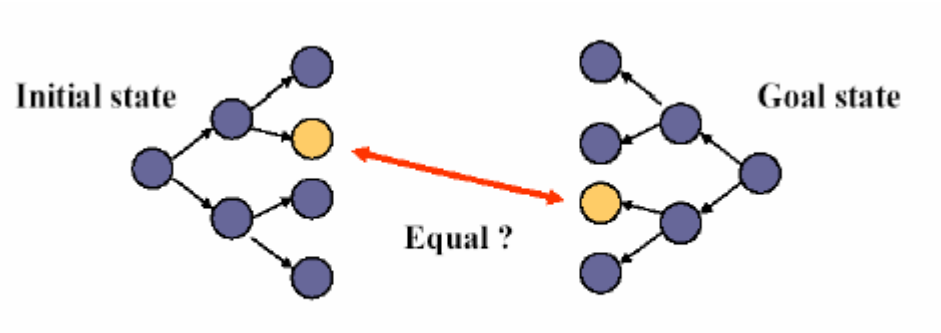


جستجوی دو طرفه[۶۸] :

ایده این روش جستجو از حالت شروع به سمت جلو و از حالت نهایی به سمت عقب می باشد بدین ترتیب که از حالت شروع عملگرها را مستقیم اعمال کرده و از حالت هدف نیز معکوس عملگرها را اعمال می کنیم و وقتی به حالت مشترک رسیدیم این مسیر جواب خواهد بود .

انگیزه به کارگیری این روش این خواهد بود که : $O(b^{d/2})=b^{d/2}+b^{d/2} <= b^{d}$

[68] Bidirectional search



شکل ۲-۹: جستجوی دو سویه

این جستجو بدین ترتیب پیاده سازی می شود که یکی از جستجوهای آن یا هر دو قبل از گسترش یک نود بررسی می کنند که آن نود در مجموعه حاشیه دیگری وجود دارد یا نه. اگر وجود داشت راه حل پیدا شده است بررسی وجود یک نود در مجموعه حاشیه جستجوی دیگری با استفاده از جدول درهم سازی[۶۹] در زمان ثابتی انجام می شود لذا پیچیدگی زمانی جستجوی دو طرفه $O(b^{d/2})$ است و پیچیدگی مکانی جستجوی دو طرفه نیز $O(b^{d/2})$ است . این میزان حافظه مهم ترین ضعف جستجوی دوطرفه است. اگر هر دو طرف این جستجو از جستجوی اول سطح استفاده کنند این جستجو کامل و اگر هزینه اعمال یکسان باشند این جستجو بهینه است اگر در دو طرف این الگوریتم، ترکیبات دیگری از جستجوها استفاده شود کامل بودن و بهینگی آن تعیین نمی شود.

معایب این روش (جست و جوی دو طرفه) :

۱- در این روش برای حرکت از سمت راست به عقب نیاز به معکوس عملگر می باشد،که این بدان معنا است که اگر در یک گره باشیم از چه گره ای می توان به این گره رسید این روش در مورد تمام مسئله ها امکان پذیر نیست چون بعضی از عملگرها را نمی توان معکوس اعمال کرد .

۲- در بعضی از موارد حالت هدف تعریف صریح و مشخصی ندارد و ضمنی است مانند بازی شطرنج که حالت هدف این است که حریف مات شود لذا حالت هدف صریحی وجود ندارد .

۳- نیاز به تصمیم گیری دارد که چه نوع جست و جویی در هر طرف انجام می شود .

---

[69] Hash table

مقایسه استراتژی‌های جستجوی ناآگاهانه

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 $l \geq d$	Yes	Yes

جدول ۲-۱: مقایسه روش‌های جستجوی ناآگاهانه

محدوده عمق انتخاب شده : l فاکتور انشعاب : b

عمیق ترین عمق مسئله : m عمق کم عمق ترین مسیر راه حل :d

اجتناب از حالت های تکراری :

حالت تکراری به معنای گسترش نودهایی است که چند لحظه پیش با آنها مواجه شده ایم و قبلا گسترش یافته اند این گسترش باعث اتلاف زمان و فضا می شود.مشکل حالات تکراری برای حالتی که فضای حالت آن یک درخت است و برای رسیدن به هر حالت فقط یک راه وجود دارد هرگز اتفاق نمی افتد . مثلا مسئله ۸وزیر به طوری که هر وزیر در سمت چپ ترین ستون خالی قرار گیرد بسیار کارا است، زیرا فقط یک مسیر برای رسیدن به هر نود وجود دارد، در برخی مسائل ، حالات تکراری غیر قابل اجتناب است ، این موضوع شامل مسائلی است که در آنها عملگرها قابل وارون شدن باشد مثل مسیریابی و معمای پازل ۸ ، که از این دسته اند .درخت جستجوی این مسائل،نامتناهی است، اما اگر بعضی حالات تکراری را حذف کنیم ، درخت جستجو متناهی می شود.وجود حالات تکراری ممکن است موجب غیر قابل حل شدن یک مسئله حل شدنی شود . سه راه حل برای مشکل راه حل تکراری وجود دارد :

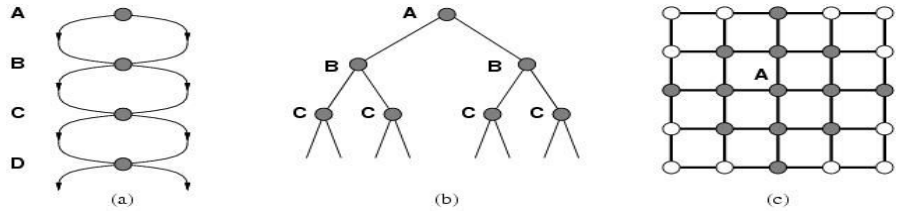
۱- به حالتی که هم اکنون آمده اید برنگردید .

۲- از ایجاد مسیرهای دوار بپرهیزید .

۳- حالتی را که قبلا تولید شده مجددا تولید نکنید چون این مسئله باعث می شود که هر حالت درحافظه نگهداری شود و پیچیده گی فضایی o(s) که s تمام حالات در درخت فضای حالت است

اگر یک الگوریتم هر حالتی را که تاکنون دیده است به خاطر آورد ، آنگاه می تواند گراف فضای حالت را به طور مستقیم جستجو کند . اگر به الگوریتم Tree-search یک ساختمان داده به نام closed که نودهای گسترش یافته را ذخیره می کند اضافه کنیم الگوریتم جدیدی به دست می آید که Graph- search نام دارد ، مجموعه نودهای گسترش نیافته در لیست open ذخیره می شوند ، و مجموعه نود های گسترش یافته در لیست close ذخیره می شوند .

اگر نود جاری با نودهای موجود در لیست close انطباق پیدا کرد ، به جای گسترش حذف می شود ، در مورد مسائلی با حالت تکراری زیاد Graph search کاراتر از Tree search است پیچیدگی زمانی و مکانیGraph-searchدر بدترین وضعیت تابع نمایی از سایز فضای حالت است که خیلی کوچک تر از $o(b^d)$ می باشد . توجه کنید که استفاده از لیست close ، سبب می شود که پیچیده گی مکانی جستجوی اول عمق و جستجوی عمقی تکرار شونده خطی نباشد . از آنجا که الگوریتم graphs همه نودها را در حافظه نگه می دارد ،بنابراین در برخی مسائل به خاطر محدودیت حافظه،اجرا نشدنی هستند .



شکل بالا، فضای حالتی را نشان می دهد که درخت جستجو ی به طور نمایی بزرگتری را تولید می کند .

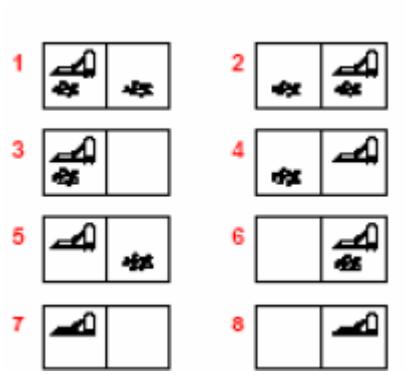
جستجو با اطلا عات ناقص[۷۰] :

برای جستجوهای نا آگاهانه ، فرض کردیم ،که محیط کاملا مشاهده پذیر و قطعی است ، و عامل نتیجه اعمالش را میداند و عامل میتواند دقیقا محاسبه کند کدام حالت از رشته ای از اعمالش ، نتیجه می شود ، و همیشه می داند در کدام حالت قرار دارد. اینگونه مسائل تک حالته نامیده می شوند. حال اگر دانش حالات و اعمال ، ناکافی باشند باعث به وجود آمدن ۳ نوع مسئله می شوند :

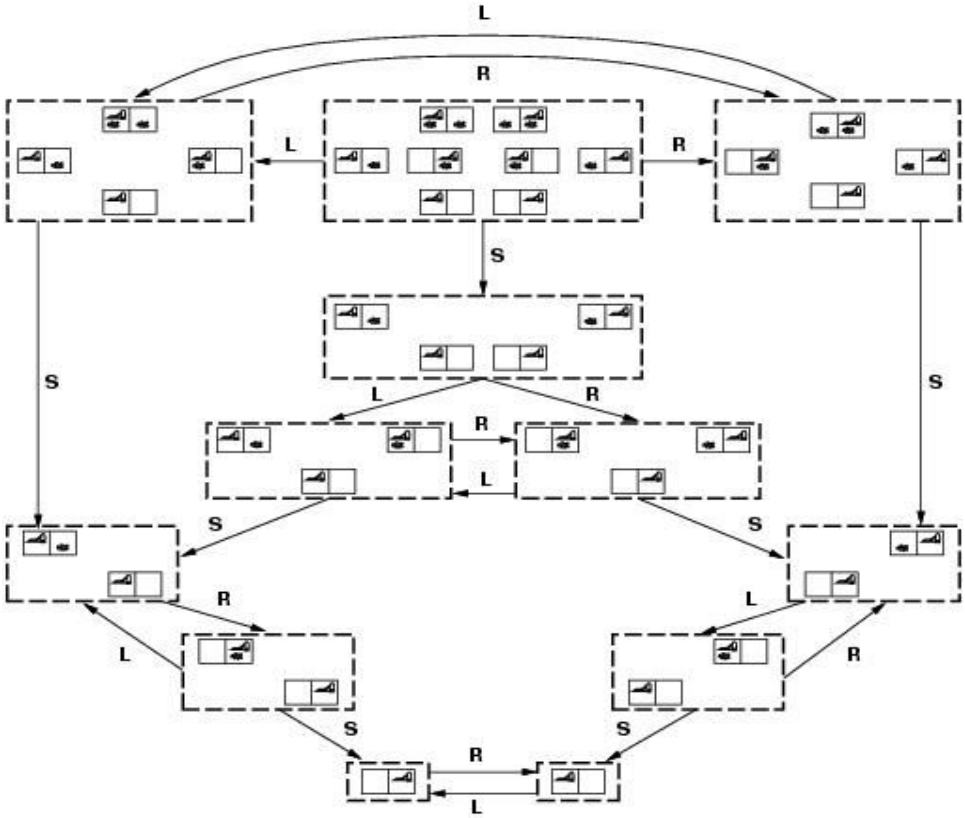
[70] Searching with partial information

۱-مسائل بدون حسگر[71] :

وقتی عامل حسگر نداشته باشد ، فقط می داند در یکی از حالات اولیه ی ممکن است و با انجام هر عمل به یکی از جندین حالت بعدی که حالت باور نامیده می شود منتقل می شود به عبارتی دیگر در اینگونه مسائل یک حالت به صورت مجموعه ای از حالات باور تعریف می شود لذا به این مسائل، چند حالته نیز گفته می شود.



شکل الف :فضای حالت برای دنیای معین بدون حسگر جارو برقی



شکل ب : فضای حالت باور برای دنیای معین بدون حسگر جارو برقی

[71] Sensorless problem

۲-مسائل اقتضائی[72] (احتمالی) :

اگر محیط پاره ای مشاهده پذیر باشد یا اعمال غیر قطعی باشند ، مشاهدات عامل بعد از هر عمل ، اطلاعات جدیدی را در اختیار می گذارد . هر مشاهده ای یک احتمال وقوع تعریف می کند ، که می تواند برای آن برنامه ریزی شود ، اگر عدم قطعیت به دلیل فعالیت های عامل دیگر ایجاد شود ، مسئله خصمانه نامیده می شود .

۳-مسائل اکتشافی[73] : اگر هم حالات و هم اعمال محیط(فضای حالت) ، ناشناخته باشند عامل باید به دنبال کشف آنها باشد، در واقع مسائل اکتشافی ، تعمیم یافته مسائل احتمالی است.

انواع مسأله

- قطعی، کاملا مشاهده پذیر ← مسائل تک – حالته
  - عامل دقیقا می داند در چه حالتی خواهد بود؛ راه حل یک دنباله می باشد.
- قطعی، مشاهده پذیر جزئی ← مسائل چند-حالته
  - ممکن است عامل ایده ای درباره اینکه کجاست نداشته باشد؛ راه حل یک دنباله است.
- غیر قطعی و/یا مشاهده پذیر جزئی ← مسائل احتمالی
  - ادراک اطلاعات جدیدی درباره حالت فعلی فراهم می کند.
  - در حین اجرا باید از حسگرها استفاده کند.
  - راه حل به صورت یک درخت
  - اغلب جستجو و اجرا به صورت یک در میان (interleave)
- فضای حالت ناشناخته ← مسائل اکتشافی (online)

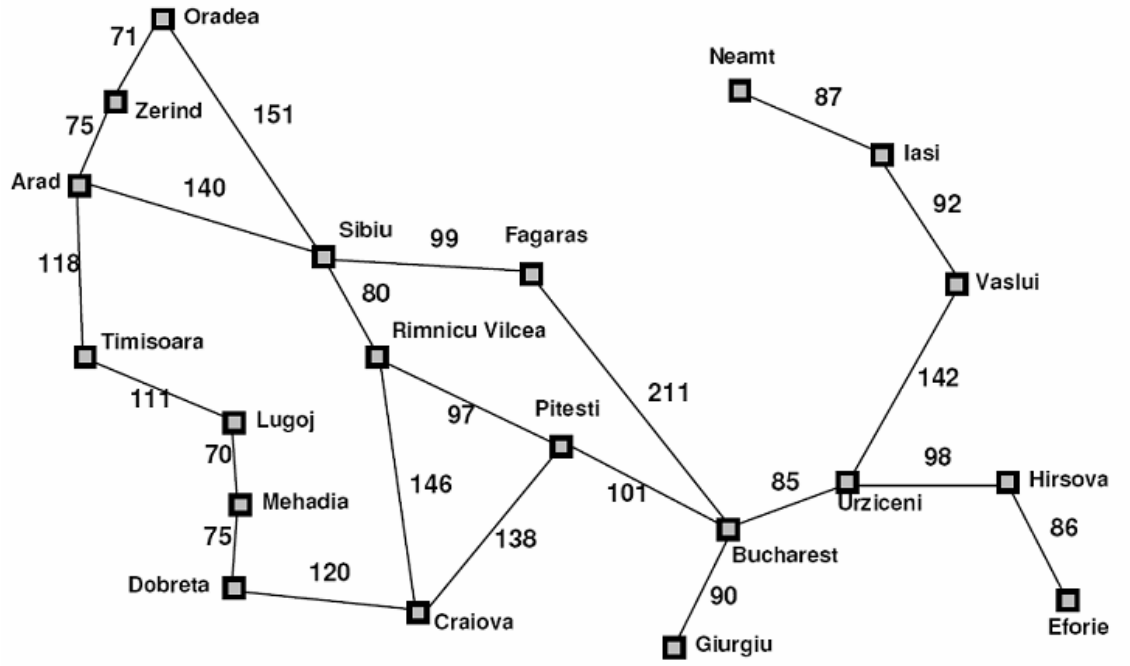
---

[72] Contingency
[73] Exploration problem

فصل چهارم: جستجو و اکتشاف آگاهانه

آنچه در این فصل خواهید آموخت:

- **جستجوی اول-بهترین**
  - **جستجوی حریصانه**
  - **جستجوی *A و خصوصیات آن**
- **جستجوهای حافظه محدود شده**
  - **جستجوی عمیق کننده تکراری *A(*IDA)**
  - **جستجوی *SMA**
  - **جستجوی اول بهترین بازگشتی(RBFS)**
- **توابع هیورستیک و مفاهیم مربوطه**
- **الگوریتم های جستجوهای محلی**
  - **جستجوی تپه نوردی**
  - **جستجوی شبیه سازی حرارت**
  - **جستجوی پرتو محلی**
  - **الگوریتم ژنتیک**



مقدمه:

همانطور که در فصل قبل دیدیم جستجوهای ناآگاهانه در بیشتر موارد ناکارا هستند زیرا در این الگوریتم ها،معیار انتخاب گره بعدی برای گسترش تنها به شماره سطخ آن بستگی دارد و از ساختار مسئله بهره نمی برند.الگوریتم های ناآگاهانه، درخت جستجو را به یک روش از پیش تعریف شده گسترش می دهند یعنی قابلیت تطبیق پذیری با آنچه که تاکنون در مسیر جستجو دریافت کرده اند و نیز حرکتی که می تواند خوب باشد را ندارند.در این فصل نشان می دهیم که چگونه یک الگوریتم جستجوی آگاهانه می تواند با کارایی بیشتر، راه حلها را پیدا کند.

جستجوی اول- بهترین[۷۴]:

شیوه ای که بر آن متمرکز می شویم جستجوی اول-بهترین نامیده می شود. جستجوی اول-بهترین مثالی از الگوریتم های Tree_search و Graph_search است که در آن ترتیب نودها برای گسترش بر اساس یک تابع ارزیابی[۷۵] f(n) می باشد.یعنی در هر مرحله نودی که کمترین مقدار ارزیابی را دارد برای گسترش انتخاب می شود . برای پیاده سازی این جستجو از طریق الگوریتم های عمومی Tree_search و Graph_search باید از صف اولویت که در آن نودها بر اساس مقدار تابع f(n) به ترتیب صعودی مرتب هستند استفاده شود. الگوریتم اول بهترین یک حالت کلی دارد و خانواده ی الگوریتم های اول بهترین در توابع ارزیابی با هم تفاوت دارند.می توان نشان داد جستجوی اول سطح با f(n)=Depth(n) وجستجوی هزینه ی یکسان با f(n)=g(n) حالت خاصی از جستجوی اول- بهترین هستند .جزء کلیدی در این الگوریتم ها تابع هیوریستیک[۷۶] یا اکتشافی می باشد که باh(n) نمایش داده می شود.تابع h(n) رایج ترین فرمی است که در آن دانش اضافی در مورد مسئله به الگوریتم جستجو اضافه می شود.تنها محدودیت تابع h(n) این است که اگر n گره هدف باشد آنگاه h(n)=0 می باشد.

h(n)=هزینه تخمینی از گره n تا گره هدف

---

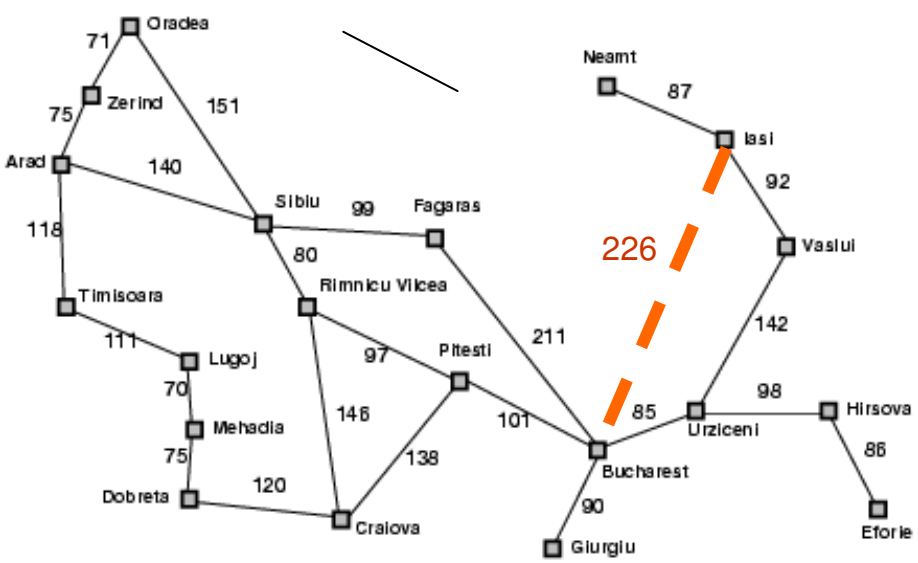
[74] Best-First Search
[75] Evaluation Function
[76] heuristic

```
function Best-First-Search(problem, Eval-FN)
                                    returns solution sequence
 nodes := Make-Queue(Make-Node(Initial-State(problem))
loop do
  if nodes is empty then return failure
   node := Remove-Front(nodes)
  if Goal-Test[problem] applied to State(node) succeeds
     then return node
   new-nodes := Expand(node, Operarors[problem],
                                Eval-FN)))
   nodes := Insert-by-Cost(new-nodes, Eval-FN(new-node))
end
```

جستجوی حریصانه[۷۷]:

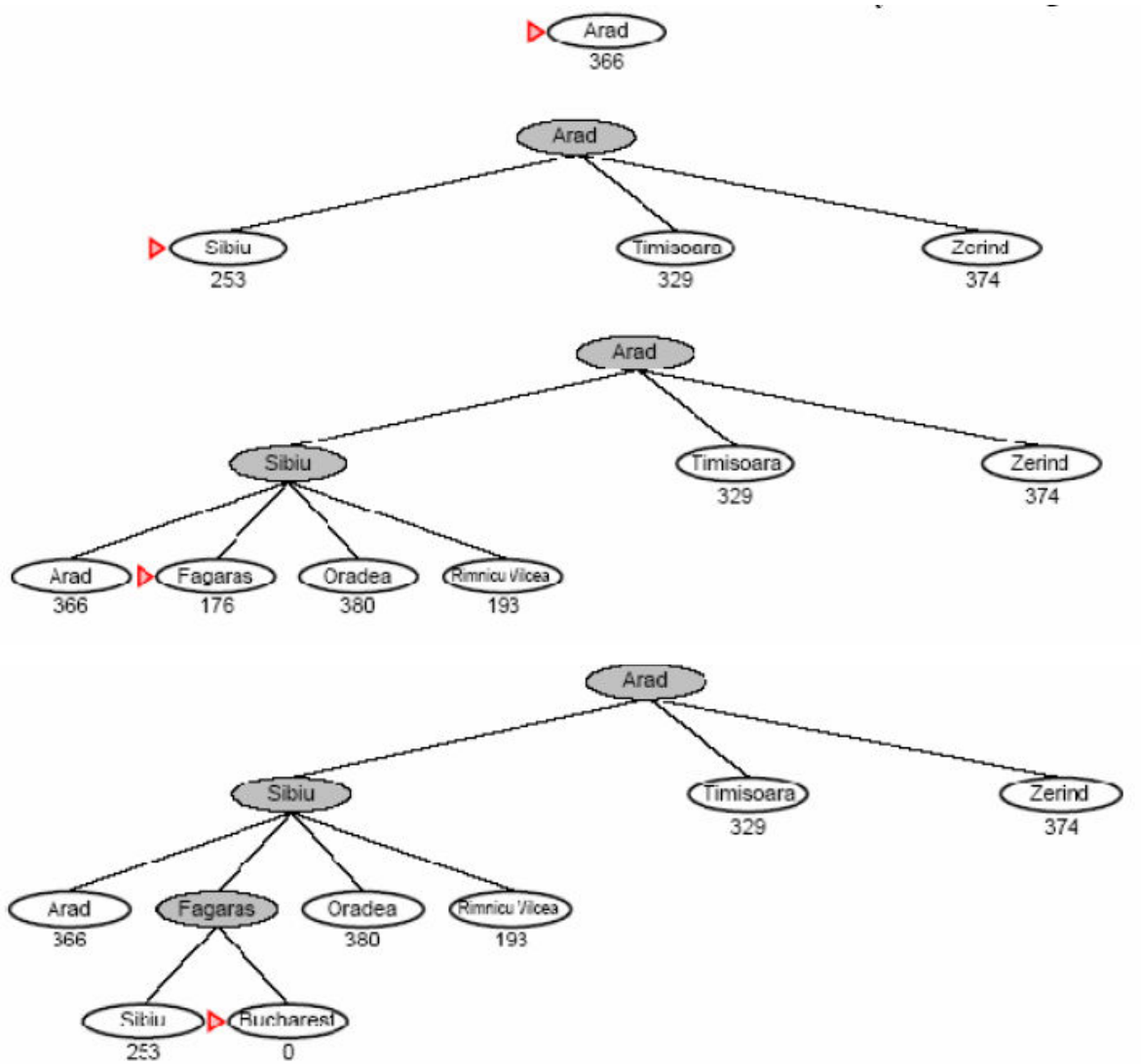
ایده اصلی در این روش به حداقل رساندن هزینه ی تخمین زده شده برای رسیدن به هدف می باشد بدین ترتیب که، گره ای که به هدف نزدیک تر باشد، ابتدا گسترش می یابد.تابعی که هزینه رسیدن از یک حالت به حالت هدف را تخمین می زند تابع اکتشافی نامیده می شود، وباحرفh نشان داده می شود.اکنون این روش جستجو را در مساله مسیریابی در رومانی با استفاده از هیورستیک خط مستقیم بررسی می کنیم.

$h_{SLD}(n)$ = فاصله مستقیم از n تا بخارست.



Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

[77] Greedy Search



جستجوی حریصانه ازلحاظ دنبال کردن یک مسیر ویژه در تمام طول راه به طرف هدف، مانند جستجوی اول عمق می باشد اما زمانی که به بن بست می رسد به سمت بالا بر می گردد.این جستجو بهینه و کامل نیست زیرا ممکن است مانند اول عمق در یک مسیر نامتناهی به سمت پایین شروع کند وهرگز برای بررسی بقیه نودها برنگردد . در بدترین وضعیت پیچدگی زمانی و مکانی برای جستجوی حریصانه $O(b^m)$است که m حداکثرعمق فضای جستجو است. جستجوی حریصانه تمام گره ها را در حافظه نگهداری می کند لذا پیچیدگی فضایی آن مشابه پیچیدگی زمانی آن می باشد. البته این پیچیدگی زمانی و مکانی با انتخاب یک تابع هیورستیک خوب به شدت کاهش می یابد.

جستجوی *A:

جستجوی حریصانه هزینه تخمین زده شده ی h(n) به سمت هدف را حداقل می کند و هزینه جستجو را کاهش می دهد اما نه کامل است نه بهینه،از طرف دیگر جستجو با هزینه یکسان، هزینه مسیر یعنی g(n) راحداقل می کند که هم کامل است هم بهینه ، اما در مواردی می تواند بی فایده باشد. برای دست یافتن به مزایای هر دو جستجو از ترکیب دو روش تحت عنوان*A استفاده می کنیم.

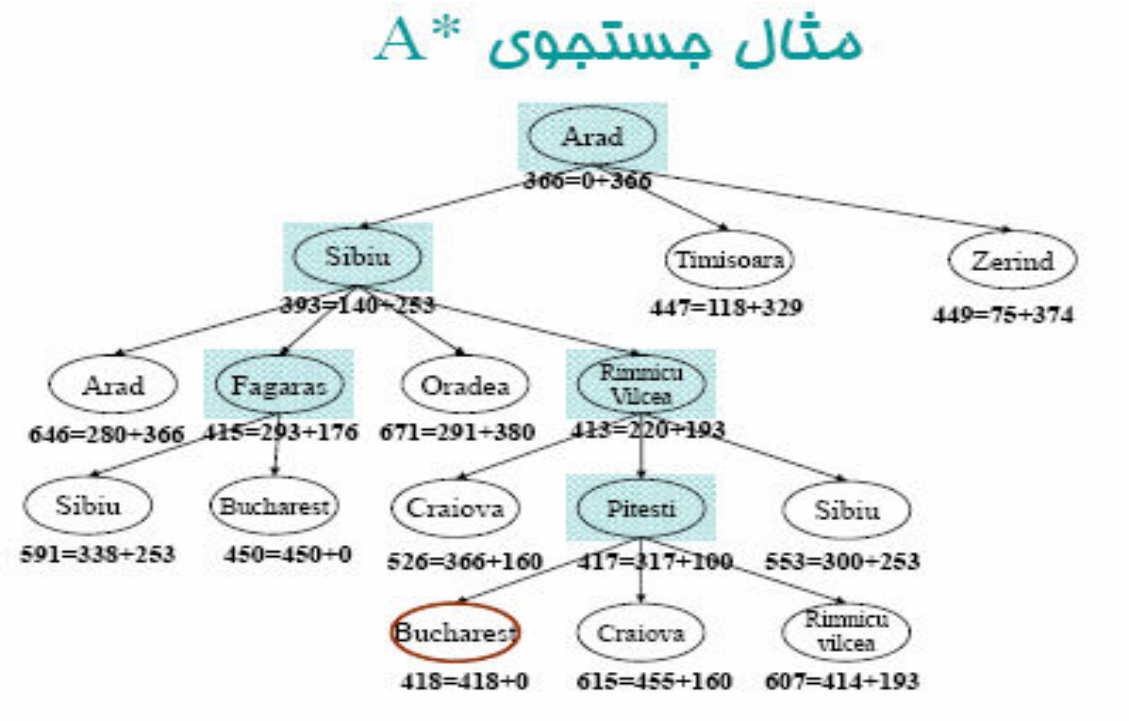
جستجوی *A

- ایده: از گسترش مسیرهایی که تاکنون مشخص شده پرهزینه می باشند، اجتناب کن.
- *A : ترکیب مزایای UCS و جستجوی حریصانه:
  - جستجوی حریصانه $h(n)$ را حداقل می کند، نه کامل و نه بهینه
  - جستجوی UCS، هزینه مسیر را حداقل می کند؛ کامل و بهینه است؛ می تواند بسیار زمانبر باشد.
- تابع ارزیابی

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

- $g(n)$: هزینه مسیر پیموده شده تا n
- $h(n)$: هزینه تخمینی ارزانترین مسیر راه حل از n تا هدف.
- $f(n)$: هزینه تخمینی ارزانترین راه حل که از n می گذرد.

شکل زیر جستجوی *A را برای مساله مسیریابی در کشور رومانی را نشان می دهد.



نکته:

اگر تابع هیوریستیک شرایط لازم را داشته باشد A* کامل وبهینه است.A* که از Graph_search استفاده می کند به شرطی بهینه است که h(n) سازگار[78] یا یکنواخت باشد و A* که از Tree_search استفاده می کند به شرطی بهینه است که h(n) قابل قبول[79] باشد.

هیورستیک قابل قبول:

تابع هیوریستیکی قابل قبول است که هرگز هزینه رسیدن به هدف را بیشتر از هزینه واقعی تخمین نزند.به عبارت دیگر،یک هیورستیک مانند h(n) قابل قبول است اگر برای هر گره n داشته باشیم: h(n)<=h*(n) که در این رابطه h*(n) هزینه واقعی برای رسیدن به هدف از گره n می باشد.از آنجایی که g(n) هزینه واقعی رسیدن به n است اگر h(n) قابل قبول باشد آنگاه f(n) نیز قابل قبول خواهد بود یعنی f(n) هرگز بیشتر از هزینه واقعی راه حل از طریق n تخمین نمی زند.

$$\left.\begin{array}{l} f(n)=h(n)+g(n) \\ \\ f^*(n)=h^*(n)+g^*(n) \end{array}\right\} \xrightarrow{h(n)<=h^*(n)} f(n)<=f^*(n)$$

هیوریستیک قابل قبول

- یک هیوریستیک مانند $h(n)$ قابل قبول است اگر برای هر گره n داشته باشیم: $h(n) \le h^*(n)$ که $h^*(n)$ هزینه واقعی برای رسیدن به هدف از گره n می باشد.
- یک هیوریستیک قابل قبول هرگز هزینه رسیدن به هدف را بیش از حد تخمین نمی زند، یعنی خوش بینانه است.
- مثال: هیوریستیک $h_{SLD}(n)$ (هیچگاه فاصله واقعی را بیش از حد تخمین نمی زند).
- قضیه: اگر $h(n)$ قابل قبول باشد، A* با استفاده از TREE-SEARCH بهینه است.

---

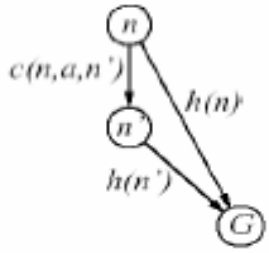
[78] Consistency
[79] admisible

هیورستیک سازگار(یکنوا):

تـابع h(n) سـازگار اسـت اگـر رابطـه h(n)<C(n,a,n')+h(n') برقـرار باشـد کـه در ایـن رابطـه، n' بـا اسـتفاده از عمـل a از حالـت n تولیـد شـده اسـت و C(n,a,n') هزینـه عمـل a را نـشان مـی دهـد ، ایـن نامساوی فرمی از نامساوی مثلثی اسـت کـه هـر ضـلع مثلـث بایـد کـوچکتر یـا مـساوی مجمـوع دو ضـلع دیگر باشد.

اثبات لم: سازگاری (Consistency)

- یک هیوریستیک **سازگار** است اگر : $h(n) \le c(n, a, n') + h(n')$
- اگر h سازگار باشد، داریم:



$$\begin{aligned} f(n') &= g(n') + h(n') \\ &= g(n) + c(n, a, n') + h(n') \\ &\ge g(n) + h(n) \\ &\ge f(n) \end{aligned}$$

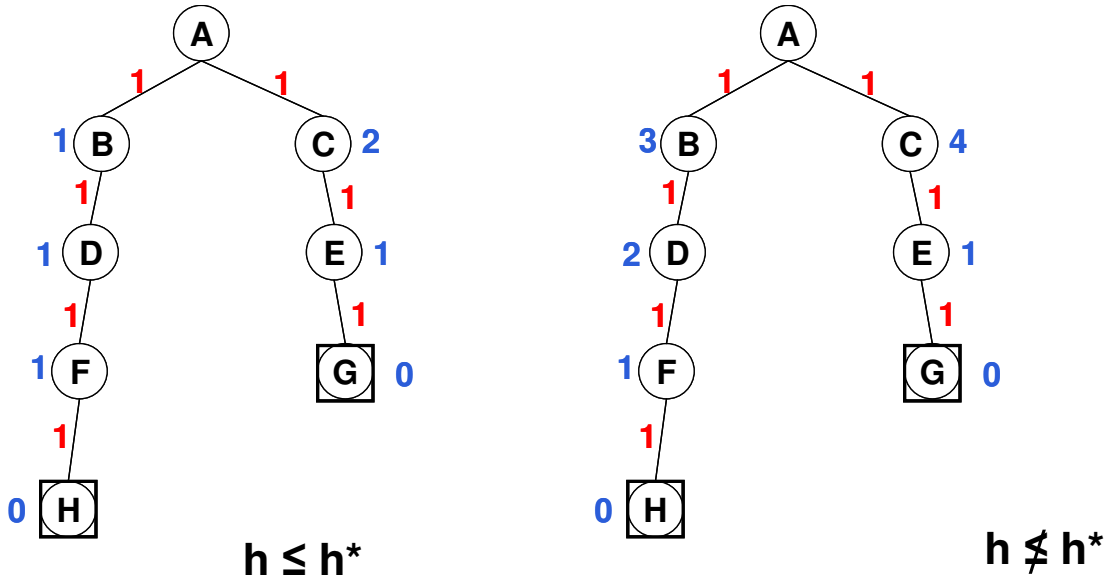
- یعنی، $f(n)$ در طول هر مسیری غیر کاهشی می باشد. (یکنوایی، monotonicity)
- **قضیه:** اگر $h(n)$ سازگار باشد، A* با استفاده از GRAPH-SEARCH بهینه است.

اگر تابع هیوریستیک یکنوا نباشد مـی تـوان آن را بـه نحـوی اصـلاح نمـود تـا یکنـوا گـردد.بـدین صـورت که هر گره جدیدی که تولید می شـود بایـد کنتـرل شـود کـه هزینـه f ایـن گـره از هزینـه f پـدزش کمتـر اسـت یـا نـه؟اگر کمتـر باشـد هزینـه f پـدر بـه جـای فرزنـد مـی نـشیند .ایـن معادلـه سـازی، معادلـه pathmax نامیـده مـی شـود کـه در ایـن معادلـه f(n')=max{f(n),g(n')+h(n')} ، n' فرزنـد n مـی باشد.

نکته: اگر c* هزینه مسیر راه حل بهینه باشد و h(n) قابل قبول باشد آنگاه:

- A* تمام گره ها با f(n)<c* را گسترش می دهد.
- A* بعضی از گره ها با f(n)=c* را گسترش می دهد.
- A* هیچ گره ای با f(n)>c* را گسترش نمی دهد.

نکته: اگر تابع h(n) قابل قبول باشد آنگاه A* بهینه خواهد بود یعنی هدف کم هزینه تر را پیدا خواهد کرد.برای درک این مطلب به مثال زیر توجه کنید:



هر الگوریتمی که تمامی گره ها را در نواحی بین ریشه وهدف بسط ندهد در معرض خطرگم کردن راه حل بهینه است اما A* برای هر تابع اکتشافی بهینگی دارد. A* در صورتی کامل نخواهدبود که گره های نامحدود زیادی با f(n)<c* وجود داشته باشد.تنها راهی که ممکن است گره های نا محدودی وجود داشته باشد این است که :

- گره ای با فاکتور انشعاب نامحدود وجود داشته باشد.
- مسیری با هزینه ی مسیر متناهی اما تعداد زیادی گره ی نامحدود در طول مسیر وجود داشته باشد.

در میان الگوریتم های بهینه، A* با هر تابع اکتشافی از نظر بهینگی کاراست یعنی هیچ الگوریتم دیگری تضمین نمی کند که تعداد کمتری گره نسبت به A* گسترش دهد.علت این است که هر الگوریتمی که تمام نودها با f(n)<c* را گسترش دهد در معرض خطراز دست دادن راه حل بهینه می باشد اگرچه جستجو A* کامل و بهینه و از نظر پیچیدگی کاراست اما تعداد نودهای موجود در کانتور هدف یک تابع نمایی از طول راه حل است.

شرط غیر نمایی بودن تعداد نودها عبارتنداز: $| h(n)-h^*(n) | <=O(\log(h^*(n)))$

تنها اشکال عمده ی A* زمان نمایی آن نیست از آنجائیکه این الگوریتم، تمام نودهای تولید شده را در حافظه نگهداری میکند بنابراین حافظه بسیار زیادی را مصرف خواهد کرد. به همین دلیل A* برای مسائل با فضای حالت بزرگ کارایی ندارد. الگوریتم هایی معرفی خواهیم کرد که بدون ازدست دادن کارایی و بهینگی بر مشکل فضای حالت بزرگ غلبه کنند مثل:SMA*,IDA*,RBFS.

خصوصیات A*

- کامل؟ بله، مگر اینکه تعدادی نامحدود گره با $f \leq f(G)$ وجود داشته باشد. A* در گراف های متناهی محلی (با فاکتور انشعاب محدود) کامل است به شرط آنکه هزینه تمام عملگرها مثبت باشد.
  - گره ای با فاکتور انشعاب نامحدود وجود داشته باشد.
  - مسیری با هزینه محدود اما با تعداد گره های نامحدود وجود داشته باشد.
- پیچیدگی زمانی؟ نمایی بر حسب [خطای نسبی h * طول راه حل]
  مگر اینکه خطا در تابع کشف کننده رشدی سریعتر از لگاریتم هزینه مسیر واقعی نداشته باشد، به زبان ریاضی:

$$| h(n) - h^*(n)| \leq O(\log h^*(n))$$

- پیچیدگی فضا؟؟ تمام گره ها را در حافظه نگه می دارد.
- بهینه؟؟ بله – نمی تواند f_{i+1} را گسترش دهد مگر f_i تمام شده باشد.
  - A* تمام گره ها با $f(n) < f^*$ را گسترش می دهد.
  - A* برخی از گره ها با $f(n) = f^*$ را گسترش می دهد.
  - A* گره ای با $f(n) > f^*$ را هرگز گسترش نمی دهد.
- A* دارای کارآیی بهینه (optimally efficient) می باشد!

کانتور[80]:

یک کانتور با برچسب L ، فضای بسته ای شامل تمام نودهایی است که مقدار f(n) آنها کمتر یا مساوی L است. کانتورها متحدالمرکزاند . کانتورها در جست وجو با هزینه یکسان در اطراف

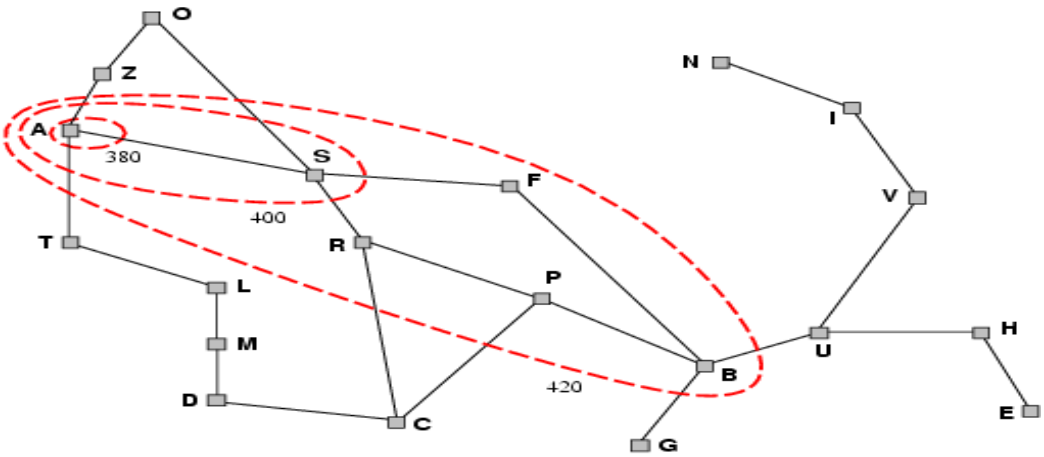
[80] contours

حالت اولیه ، دایره ای شکل اند و با توابع اکتشافی دقیقترنواحی به سمت حالت هدف کشیده می شوند ودر اطراف مسیر بهینه بیضی شکل میشوند.

جست و جوی اکتشافی با حافظه ی محدود(IDA*):

ساده ترین راه برای کاهش حافظه مورد نیاز A* ، استفاده از ایده ی موجود در جست و جوی عمقی تکرار شونده است . الگوریتم حاصل از اعمال ایده ی تکرار شونده ی عمقی در A* ، IDA* نام دارد. تفاوت IDA* با عمقی تکرار شونده در این است که در این الگوریتم به جای عمق ، محدودیت روی هزینه f قرار داده می شود.مقدار اولیه ی f-limit برابر مقدار f ریشه است. در هر تکرار گره هایی که f آنها کمتر از f-limit آن تکرار است ، گسترش می یابند . اگر در این تکرار هدف پیدا شد که کار تمام است ،در غیر این صورت کمترین مقدار f گره های گسترش نیافته در این تکرار ، جایگزین مقدار f-limit می شود و دوباره الگوریتم با مقدار جدید f-limit اجرا می گردد. این تکرارها ادامه می یابد تا زمانی که مقدار f-limit به گونه ای باشد که نود هدف نیز برای گسترش انتخاب شود. نود هدف در تکرار با f-limit=c* پیدا می شود.

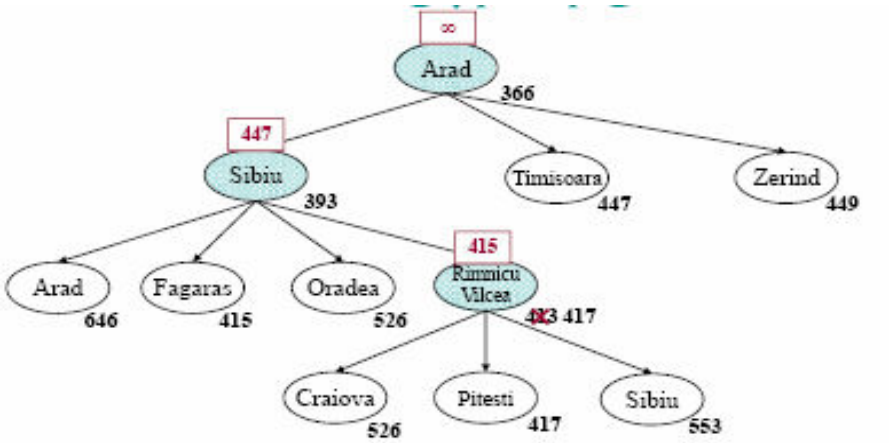
این الگوریتم کامل و بهینه است و در هر مرحله فقط گره هایی که f آنها کمتر از f-limit است در حافظه نگهداری می شوند. بنابراین از نظر پیچیدگی مکانی مانند جست و جوی عمقی، خطی است . البته در بدترین حالت $O(bc^*/\delta)$ می باشد که δ کمترین هزینه ی اعمال می باشد . IDA* برای مسا ئل با هزینه ی مرحله ای واحد مناسب است و نیاز به نگهداری صف مرتبی از گره ها ندارد . اما متأسفانه مانند نسخه ی تکرار شونده ی جست و جو با هزینه ی یکنواخت منجر به افزایش محاسباتش می شود.

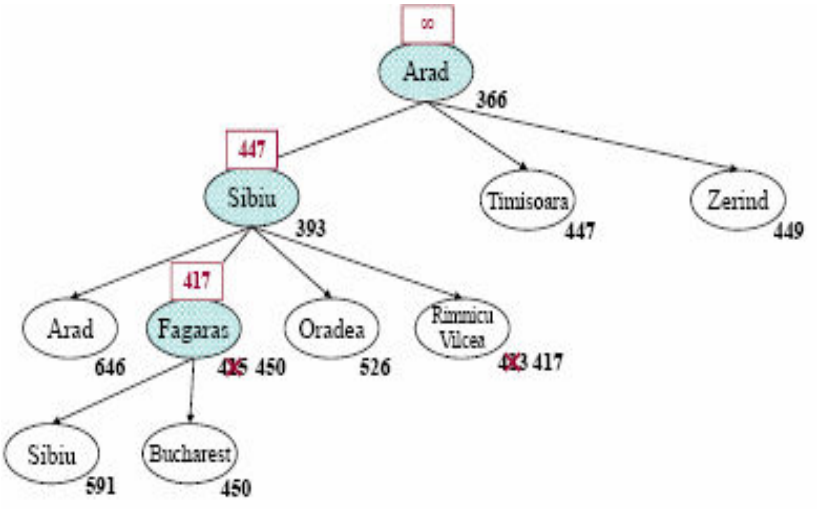


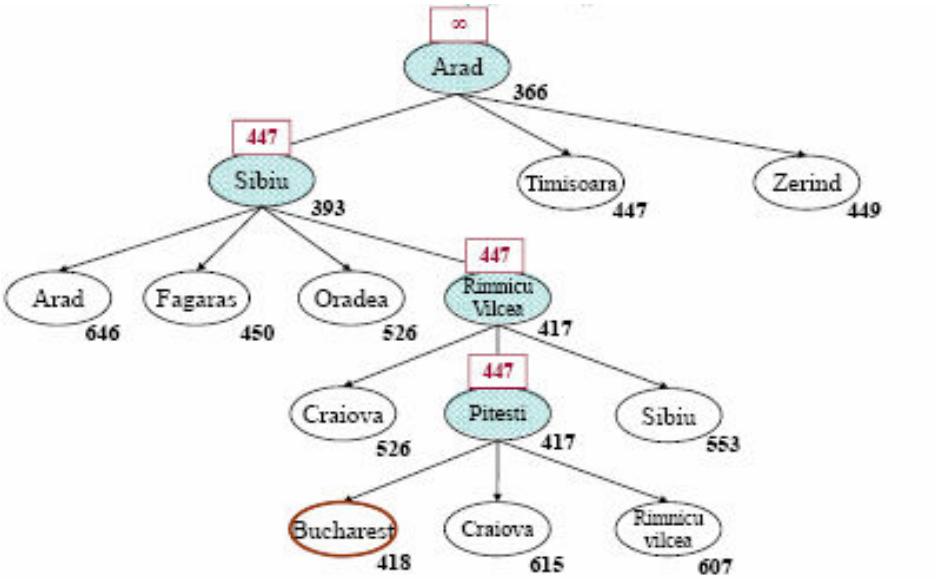
جست و جوی اولین- بهترین بازگشتی (RBFS):

این جست و جو یک الگوریتم بازگشتی ساده است که از جست و جوی اول بهترین تقلید می کند ، با این تفاوت که پیچیدگی مکانی آن خطی است . ساختار آن شبیه جست و جوی عمقی بازگشتی است . اما به جای آنکه دا ئماً مسیر فعلی را به سمت پایین ادامه دهد، مقدار f بهترین مسیر جانشین را از طریق اجداد گره فعلی نگهداری میکند . اگر f گره فعلی از این حد تجاوز کند ، الگوریتم به عقب بر می گردد تا مسیر جانشین را انتخاب نماید. در بازگشت به عقب این الگوریتم مقدار f مربوط به بهترین برگ از زیر درخت فراموش شده را به یاد می آورد و میتواند تصمیم بگیرد آیا این زیر درخت باید بعداً گسترش یابد یا خیر.

RBFS کمی از IDA* کاراتر است اما این الگوریتم نیز مشابه IDA* گره های تکراری تولید می کند.اگر تابع اکتشافی h(n) قابل قبول باشد ، RBFS همانند A* بهینه است. پیچیدگی مکانی آن تابع خطی O(bd) است . تعیین پیچیدگی زمانی آن دشوار است و به دقت تابع هیوریستیک و میزان تغییر بهترین مسیر در اثر گسترش گره ها بستگی دارد . IDA* و RBFS از حافظه اندکی استفاده می کنند که این مسأله می تواند به آنها آسیب برساند . IDA*در هر تکرار فقط یک عدد را نگهداری می کند که هزینه فعلی f است ولی RBFS اطلاعات بیشتری را نسبت به IDA* درحافظه نگهداری می کند. اگر حافظه ی زیادتری مهیا داشته باشد ، این دو الگوریتم راهی برای استفاده از آن ندارند بنابراین بهتر است الگوریتمی داشته باشیم که از کل حافظه موجود استفاده کند(مانند الگوریتم SMA*). شکل زیر جستجوی RBFS را روی مسئله مسیریابی در رومانی را نشان می دهد.







الگوریتم SMA*:

SMA* مانند A* بهترین برگ را گسترش می دهد تا حافظه پر شود. با پر شدن حافظه، بدون از بین بردن گره های قبلی نمی توان گره جدیدی اضافه کرد.زمانی که نیاز به تولید فرزند باشد و حافظه ای در اختیار الگوریتم نباشد ، نیاز به نوشتن مجدد بر روی حافظه است . برای انجام این امر،SMA* ، یک گره را حذف می کند و فرزند جدید از حافظه ی آن استفاده خواهد کرد. گره هایی که به این طریق حذف می شوند، گره های فراموش شده[٨١] نام دارند. در این حالت گره هایی

[81] Forgoten node

برای حذف شدن انتخاب می شوند که هزینه ی آنها بالاست . برای جلوگیری از جست وجوی مجدد زیردرخت هایی که از حافظه حذف شده اند ، در گره ی پدر آنها اطلاعاتی درباره ی کیفیت بهترین مسیر، درزیردرخت فراموش شده نگهداری می شود . بنابراین زمانی این زیردرخت ها دوباره تولید خواهند شد که ثابت شود سایر مسیرهای دیگر بدتر از مسیر فراموش شده هستند.

خواص SMA*

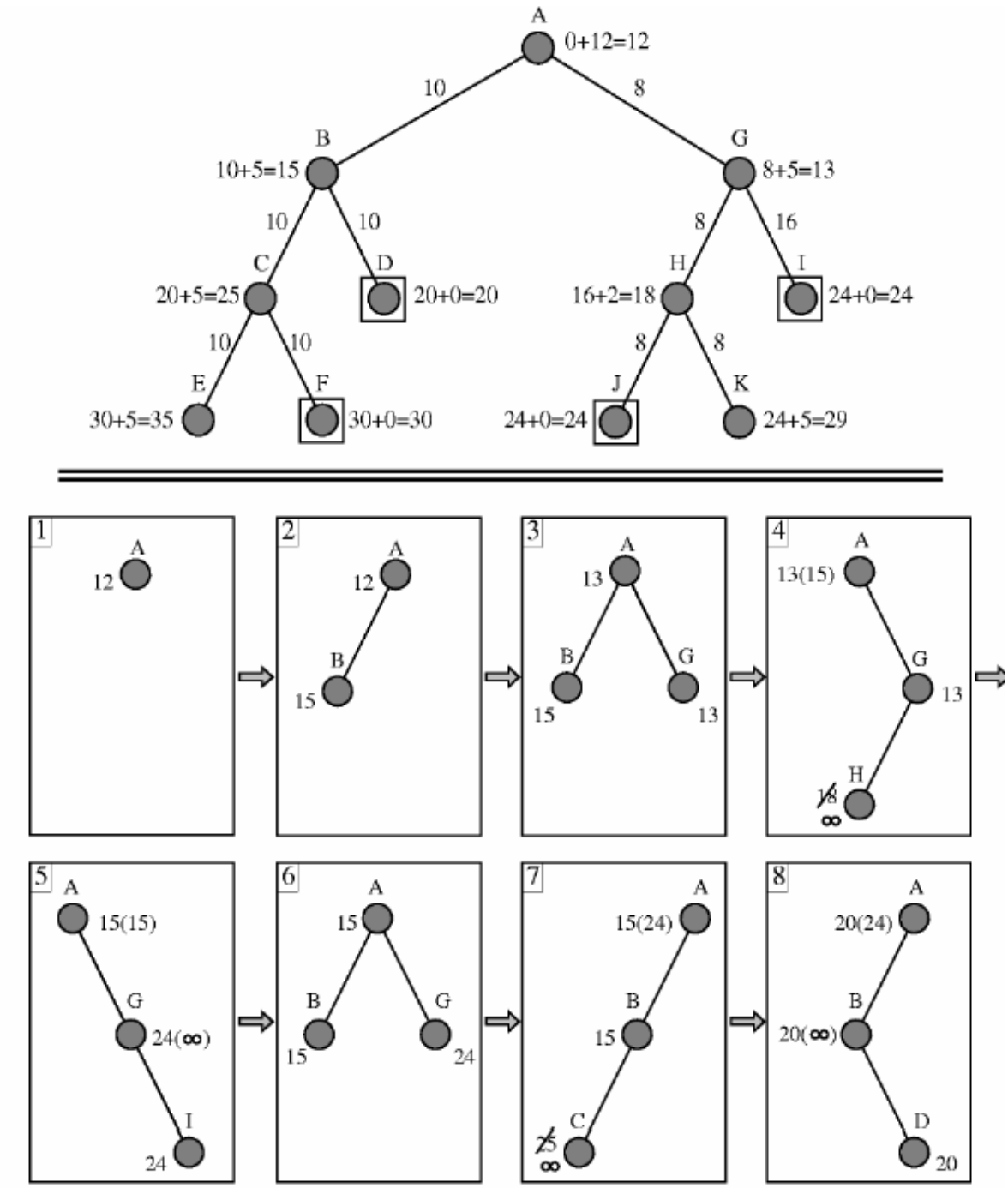
- می تواند از تمام حافظه قابل دسترس استفاده ببرد.
- از حالات تکراری تا جایی که حافظه اجازه می دهد جلوگیری می کند .
- این الگوریتم کامل است به شرط آنکه حافظه کافی، برای ذخیره ی کم عمق ترین مسیر راه حل وجود داشته باشد .
- این الگوریتم بهینه است به شرط آنکه حافظه کافی برای ذخیره ی کم هزینه ترین مسیر (هدف با کمترین f) وجود داشته باشد .
- زمانی که حافظه ی موجود برای جست وجوی درخت کافی باشد، جست وجوی SMA* بهینه ی کاراست.

SMA* بهترین الگوریتم همه منظوره برای یافتن راه حل های بهینه است .اگر مقدار f تمام برگ ها یکسان باشد ، باید الگوریتم یک گره را هم برای گسترش و هم برای حذف انتخاب کند . SMA* این مسأله را با گسترش بهترین برگ جدید و حذف بهترین برگ قدیمی حل می کند .گاهی اوقات ممکن است SMA* مجبور شود دائماً بین مجموعه ای از مسیرهای حل کاندید تغییر وضع دهد ، در حالی که بخش کوچکی از هر کدام در حافظه جای می شود .

نکته : ترتیب الگوریتم های حافظه محدود شده از نظر پیچیدگی حافظه:

IDA* < SMA*<RBFS

شکل زیر مثالی از الگوریتمSMA* با ۳ خانه حافظه را نشان می دهد



شکل ۳-۳: مراحل الگوریتم *SMA روی یک مثال

توابع هیورستیک:

در این بخش، توابع اکتشافی معمای ۸ بررسی می شود تا با ماهیت این توابع آشنا شوید.معمای ۸ یکی از اولین مسائل جستجوی اکتشافی بود.میانگین هزینه راه حل برای یک نمونه تصادفی معمای۸، ۲۲ مرحله می باشدکه متوسط فاکتور انشعاب برابر ۳ می باشد بنابراین جستجوی ۲۲ مرحله ای تقریباً b^d که در اینجا برابر 3^{22} حالت دارد که تعداد حالات بسیار زیادی است و با

انتخـاب یـک تـابع اکتـشافی مناسـب مـی تـوان مراحـل جـستجو را کـاهش داد.اگـر بخـواهیم بـا A* کوتاهترین راه حل هـا را بیابیم،بـه تـابعی اکتـشافی نیـار داریـم کـه تعـداد مراحـل را اضـافه تخمـین نزنـد یعنی قابل قبول باشد. دو تابع هیورستیک رایج برای معمای ۸ عبارتند از:

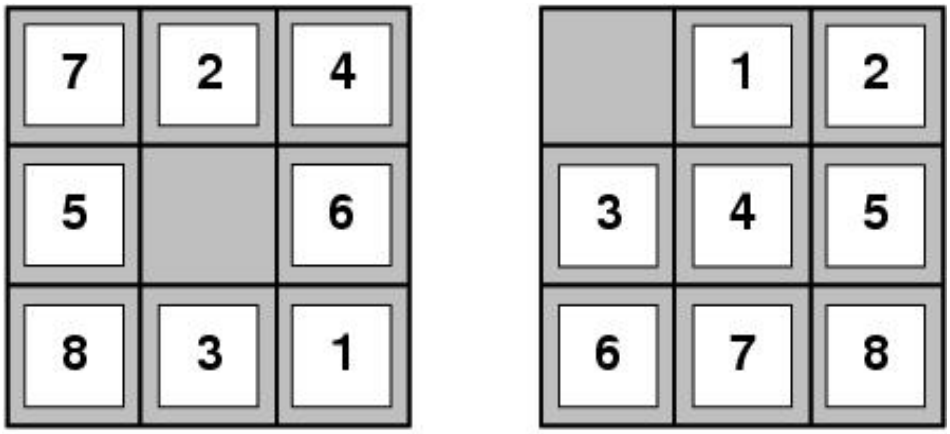
h1= تعداد خانـه هـایی کـه در مکـان هـای نادرسـت قـرار دارنـد.ایـن تابع،هیورسـتیک قابـل قبـولی اسـت زیرا بدیهی است هر خانه که در جای نامناسبی قرار دارد،حداقل یکبار باید جابجا شود.

h2=مجموع فواصل خانه ها از مکانهـای صـحیح آنهـا. آز آنجائیکـه خانـه هـا در امتـداد قطـر جابجـا نمـی شوند، فاصله ای که محاسبه مـی شـود،مجموع فواصـل افقـی و عمـودی اسـت.ایـن فاصـله گـاهی فاصـله بلوک شهر یا فاصله مانهاتان[۸۲] نامیـده مـی شـود. ایـن تـابع نیـز قابـل قبـول اسـت زیـرا بـا هـر جابجـایی یک خانه، یک مرحله به هدف نز دیک تر می شود.

شکل زیر مثالی از معمـای ۸ را نـشان مـی دهـد کـه مقـدار توابـع هیورسـتیکh1وh2 بـرای آنهـا محاسـبه شده است:

h1=8

h2=3+1+2+2+2+3+3+2=18

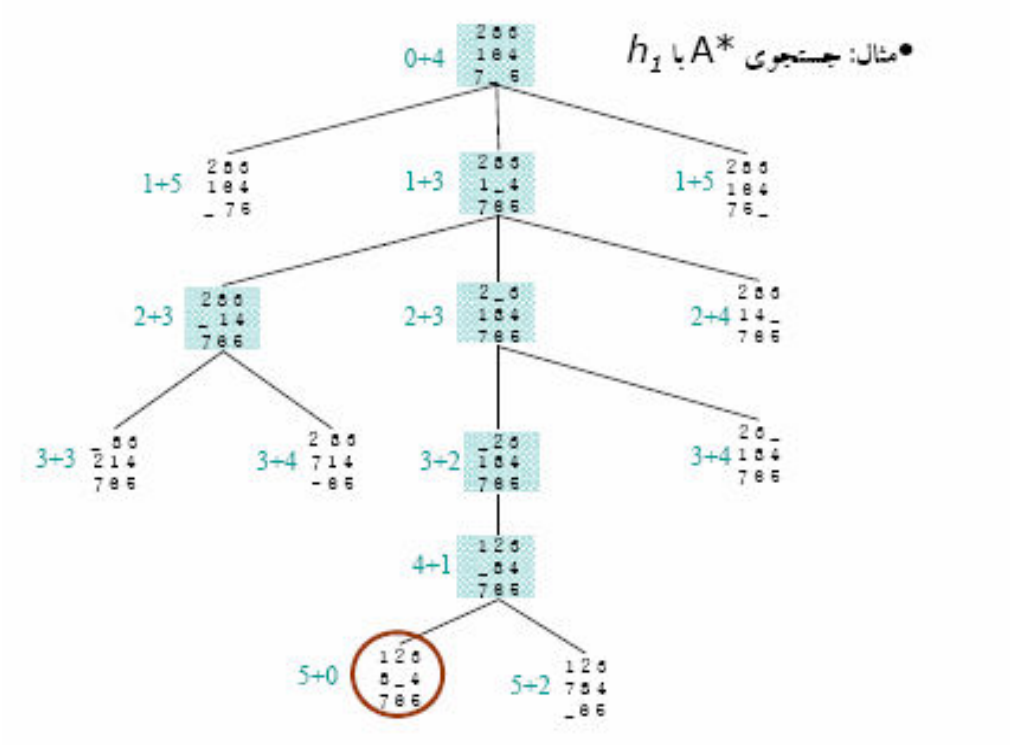


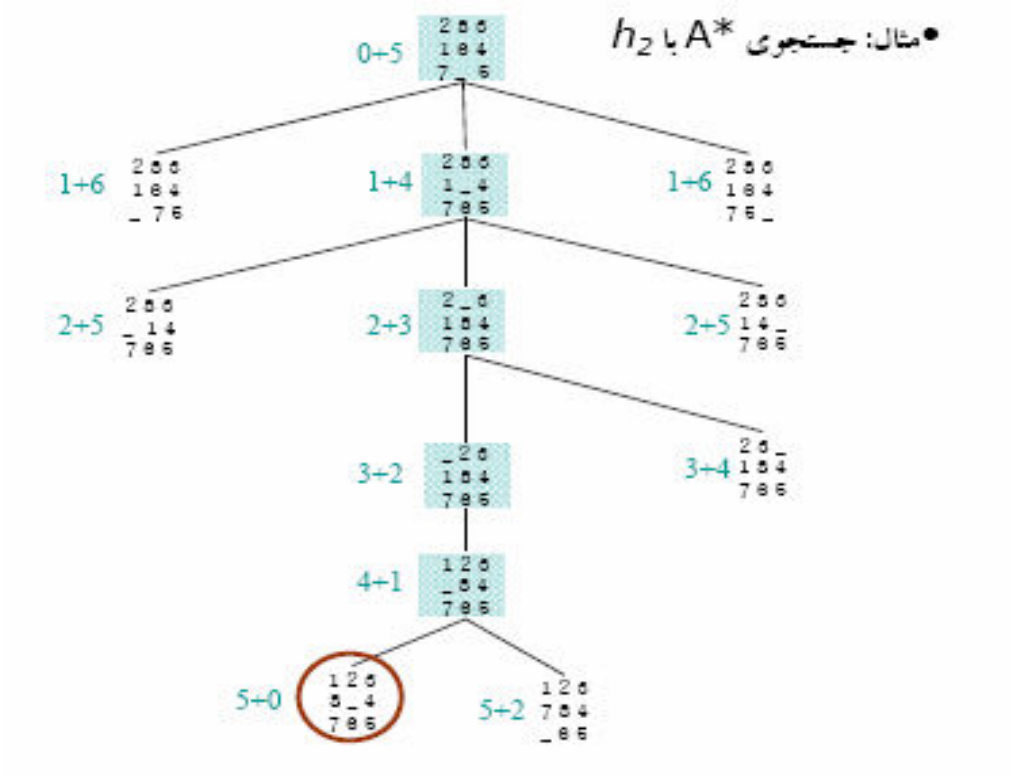
Start State Goal State

همانطور که انتظـار داریـم هـیچ کـدام از ایـن برآوردهـا، هزینـه واقعـی راه حـل نیـست،بلکه هزینـه واقعـی ۲۶ است.

[82] manhattan

در شکل زیر نمونه ای از مسئله معمای ۸ یکبار با تابع هیورستیک h1 و یکبار با h2 حل شده است.





اثر کیفیت تابع هیورستیک بر کارایی:

یک روش تعین کیفیت تابع هیورستیک،فاکتور انشعاب مؤثر[۸۳] است که با b* نشان داده می شود.اگر تعداد کره های گسترش یافته توسط روال جستجو N باشد و عمق راه حل d باشد آنگاه b* به صورت زیر محاسبه می شود.

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

- **مثال:** اگر A* راه حلی را در عمق ۵ با استفاده از ۵۲ گره پیدا کند، فاکتور انشعاب موثر را محاسبه کنید.

پاسخ:

$$53 = 1 + b^* + (b^*)^2 + \ldots (b^*)^5 \rightarrow b^* = 1.92$$

بنابراین اندازه گیری b* روی مجموعه ای کوچک می تواند مفید بودن یا نبودن یک تابع هیوریستیک را در حالت کلی مشخص کند. مقدار b* برای تابع اکتشافی خوب طراحی شده، نزدیک به 1 است. فاکتور انشعاب مؤثر برای مسا ئل بزرگ وسخت معمولا ثابت است . بنابراین در یک هیوریستیک هر چه فاکتور انشعاب مؤثر به 1 نزدیکتر باشد آ ن هیوریستیک کیفیت کشف کنندگی بیشتری دارد.

تسلط[۸۴]:

اگر به ازای هر n ، h2 (n) >= h1(n) (با فرض اینکه h2,h1 قابل قبول باشند) آنگاه h2 بر h1 تسلط دارد; یعنی کاراتر می باشد و برای جست و جو مناسب تر است. تعداد گره هایی که A* با به کارگیری h2 گسترش می دهد هرگز بیشتر از تعداد گره هایی که A* با بکارگیری h1 گسترش می دهد نخواهد بود یعنی هر گره ای که به وسیله ی A* با h2 گسترش می یابد قطعا به وسیله ی A* با h1 نیز گسترش می یابد. علاوه بر این A* با h1 ممکن است منجر به گسترش گره های دیگری بشود. لذا همیشه بهتراست از تابع اکتشافی قابل قبول با مقادیر بزرگتر استفاده کرد ؛ به شرطی که زمان محاسبه ی مقدار آن تابع هیوریستیک در هر نود خیلی زیاد نباشد.

---

[83] Effective Branch Factor
[84] Dominance

مقایسه جستجو های A* (h1) ، A*(h2) ، IDS برای حل صد نمونه مسئله تصادفی معمای ۸:

برای تست توابع اکتشافی h2 و h1 صد نمونه مسئله تصادفی پازل ۸ با عمق های مختلف از ۲ تا ۲۴ تولید شده است. و به کمک جست و جوی عمقی تکرار شونده و جست وجوی درختی A* با استفاده از h2, h1 حل شده اند . شکل زیر مقایسه ای بین هزینه ی جست وجو یعنی تعداد گره های گسترش یافته و فاکتور انشعاب مؤثر در این جستجوها را نشان می دهد. با توجه به نتایج حاصل از این جدول، می توان نتیجه گرفت که h2 بهتر از h1 است و A* آگاهانه خیلی بهتر از IDS ناآگاهانه است.

مقایسه بین هزینه جستجو و فاکتور انشعاب موثر

	Search Cost			Effective Branching Factor		
d	IDS	$A^*(h_1)$	$A^*(h_2)$	IDS	$A^*(h_1)$	$A^*(h_2)$
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6,384	39	25	2.80	1.33	1.24
10	47,127	93	39	2.79	1.38	1.22
12	364,404	227	73	2.78	1.42	1.24
14	3,473,941	539	113	2.83	1.44	1.23
16	-	1,301	211	-	1.45	1.25
18	-	3,056	363	-	1.46	1.26
20	-	7,276	679	-	1.47	1.27
22	-	18,094	1,219	-	1.48	1.28
24	-	39,135	1,641	-	1.48	1.26

مسا ئل تعدیل شده[85]:

مسا ئل تعدیل شده مسا ئلی هستند که بعضی از محدودیت های عملگر های آن حذف شده باشد به عبارت دیگر مسئله ساده شده باشد. این مسائل با محدودیت کمتری بر روی عملگرها مواجه هستند.اگر قوانین معمای ۸ طوری تغییر کند که هر خانه بتواند در هر جایی قرار گیرد به جای اینکه فقط به خانه ی همجوار خالی برود آنگاه h1 کوتاه ترین راه حل را پیدا می کند. و اگر هر خانه بتواند به یک مربع در هر جهت منتقل شود حتی در مربع اشغال شده، آنگاه h2 نیزکوتاهترین راه حل را پیدا می کند .

[85] Relaxed problem

اصل معمای۸ را در نظر بگیرید:

«خانه A می تواند به خانه B برود اگر A هم جوار افقی یا عمودی B باشد و B خالی باشد.»

با حذف یک یا دو شرط از اصل فوق، می توان سه مسئله راحت تولید کرد:

- خانه A می تواند به خانه ی B برود اگرA هم جوارB باشد.
- خانه A می تواند به خانه ی B برود اگرB خالی باشد.
- خانه A می تواند به خانه B منتقل شود.

معمولاً حل یک مسئله تعدیل شده یک تخمین یا تا بع کشف کننده ی خوبی برای حل مسئله اصلی خواهد بود.هزینه راه حل بهینه یک مساله راحت،بیشتر از هزینه راه حل بهینه در مسئله اصلی نخواهد بود. برنامه ای به نام ABSOLVER با استفاده از روش مسئله راحت و تکنیک هایی دیگر، توابع اکتشا فی را به طور خود کار تولید می کند. این برنامه توابع هیور یستیک جدیدی را برای معمای ۸ تولید کرد که از توابع قبلی مفیدتر است .این برنامه، همچنین، اولین تابع اکتشافی مفید را برای معمای روبیک پیدا کرده است .

یکی از مشکلات تولید توابع اکتشافی جدید، یافتن بهترین تابع هیوریستیک است اگر مجمو ای از توابع اکتشافی قابل قبول h1,h2,…,hn را داشته باشیم و هیچکدام بر دیگری تسلط نداشته باشند انگاه بهترین تابع هیوریستیک به صورت زیر است: $h(n) = \max(h_1(n),..., h_k(n))$

این تابع اکتشافی مرکب از توابع اکتشافی h1,h2,…,hn با کیفیت تر می باشند از انجائیکه همه ی توابع قابل قبول است h نیز قابل قبول است در این حالت h بر تمام توابع مذکور تسلط دارد وکاراتر است. روش دیگر برای ابداع یک کشف کننده ی خوب استفاده از اطلاعات آماری است که این اطلاعات می تواند توسط اجرای جستجو روی تعدادی مسائل، جمع آوری شود. مانند ۱۰۰ مسئله که ساختار معمای۸ را داشته باشند و به طور تصادفی انتخاب شوند و در نهایت روی نتایج اجرای نمونه های مختلف تحلیل های آماری مثل میانگین،میانه،واریانس و... گرفته شود. ایده ی بانک اطلاعاتی الگو، ذخیره ی هزینه ای واقعی راه حل های هر ترکیب ممکن از زیر مسئله را ممکن می سازد گاهی اوقات می توان برای ابداع یک تابع اکتشافی جدید از ترکیب خطی چند تابع استفاده کرد . به عنوان مثال تابع خطی H (n) = C1 x1(n) +C2 x2(n) که در آ ن

x1(n) تعـداد خانـه هـا بـا مکـان نادرسـت و x2(n) تعـداد جفـت هـایی از خانـه هـای همجـوار کـه درحالـت هـدف نیـز همجـوار یکدیگرنـد. مقـادیر ثابـتc1,c2 بـرای تطـابق بـا هزینـه ی واقعـی تنظـیم خواهند شد.

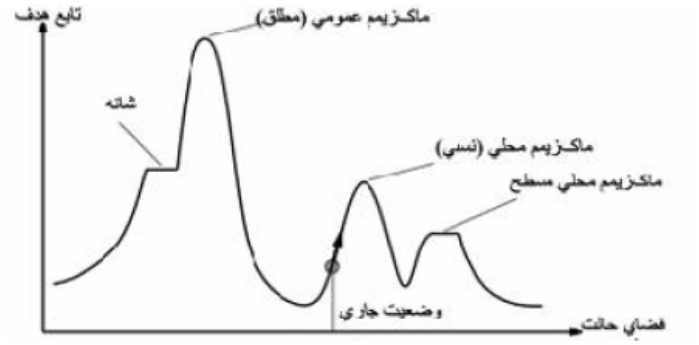
الگوریتم های جست وجوی محلی:

تاکنون الگوریتم های جستجویی که بررسـی کـردیم طـوری طراحـی شـده بودنـد کـه فـضای جـست وجـو را به طور سیستماتیک(قدم به قدم) مورد بررسـی قـرار مـی دادنـد و در آنهـا تـا رسـیدن بـه هـدف یـک یـا چند مسیر نگه داری می شد و مسیر رسیدن به هدف راه حل مسئله را تشکیل می داد.

در الگوریتم های جست و جو ی محلی مـسیر رسـیدن بـه هـدف مهـم نیـست بـه عنـوان مثـال در مـسئله ی۸ وزیر پیکر بندی نهایی مهـم اسـت و ترتیـب اضـافه شـدن وزرا مهـم نمـی باشـد. ایـن نـوع مـسا ئـل شـامل کاربردهـای زیـادی از قبیـل : طراحـی مـدارهای مجتمـعVLSI ، زمـان بنـدی کارها،چیـدمان دستگاه ها در کف کارخانه و غیره مـی باشـد.اگـر چـه الگـوریتم هـای جـست وجـوی محلـی سیـستمانیک نیستند ولی دو مزیت عمده دارند:

- اینکه از حافظه کمی استفاده می کنند.
- در فضاهای حالت بزرگ ونامتنـا هـی کـه الگـوریتم هـای سیـستماتیک مناسـب نیـستند راه حـل هایی خوبی پیدا می کنند .

الگوریتم های جست وجوی محلی علاوه بـر یـافتن هـدف، بـرای حـل مـسا ئـل بهینـه سـازی نیـز مفیـد هستند. در این مسا ئل مقصود یـافتن بهتـرین حالـت بـر اسـاس یـک تـابع هـدف[۸۶] مـی باشـد بـرای درک جست جوی محلی به سطح فضا ی حالت زیر توجه کنید:



[86] Objective function

سطح فضای حالت فوق،هم دارای مکان(که با حالت مشخص می شود) و هم دارای ارزیابی(که با مقدار تابع هدف تعریف می شود) می باشد. اگر ارزیابی متناظر با هزینه باشد هدف رسیدن به پایین ترین دره یا مینیمم مطلق است. اگر ارتفاع متناظر با تابع هدف باشد، هدف یافتن بلندترین قله یا ماکزیمم مطلق است.الگوریتم های جستجوی محلی این سطح را کاوش می کنند. یک الگوریتم جستجوی محلی کامل در صورت وجود هدف آن را می یابد و یک الگوریتم جستجوی محلی بهینه،همواره ماکزیمم یا مینیمم مطلق را می یابد.

ایده ی کلی الگوریتم های جست وجو ی محلی این است که وقتی از یک حا لت قبلی به حا لت فعلی برسیم دیگر حا لت قبلی را فراموش می کنیم. لذا در این مسا ئل درخت نداریم بلکه تنها یک وضعیت فعلی داریم که خودش شامل تمام اطلاعات مورد نیاز مسئله است. در این الگو ریتم ها از یک حا لت قانونی شروع کرده و سپس با انجام تغییراتی در آ ن سعی در اصلاح کیفیت آن ساختار داریم. الگوریتم های جستجوی محلی که در این بخش بررسی خواهیم کرد عبارتند از:۱-تپه نوردی۲-شبیه سازی حرارت۳-جستجوی پرتو محلی۴-الگوریتم ژنتیک

جستجوی تپه نوردی[۸۷] :

این الگوریتم درجهت افزایش مقدارارزش تابع ارزیاب حرکت می کندیعنی به طرف بالای تپه. وقتی به قله رسید یعنی جایی که هیچ همسایه ای ازآن بلندترنیست خاتمه می یابد. الگوریتم تپه نوردی فقط به همسایگان حالت جاری نگاه می کند. تپه نوردی گاهی جستجوی محلی حریصانه نیزنامیده می شود، زیرا بدون اینکه به آینده فکرکند بهترین همسایه راانتخاب می کند. اگردرالگوریتم تپه نوردی به دنبال مینیمم محلی یاکاهش هزینه باشیم الگوریتم، کاهش گرادیان نامیده می شود.

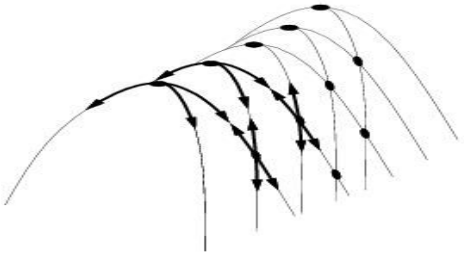
«مانند بالا رفتن از کوه اورست در مه غلیظ با ضعف حافظه»

```
function HILL-CLIMBING(problem) returns a state that is a local maximum
   inputs: problem, a problem
   local variables: current, a node
                    neighbor, a node
   current ← MAKE-NODE(INITIAL-STATE[problem])
   loop do
      neighbor ← a highest-valued successor of current
      if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
      current ← neighbor
```

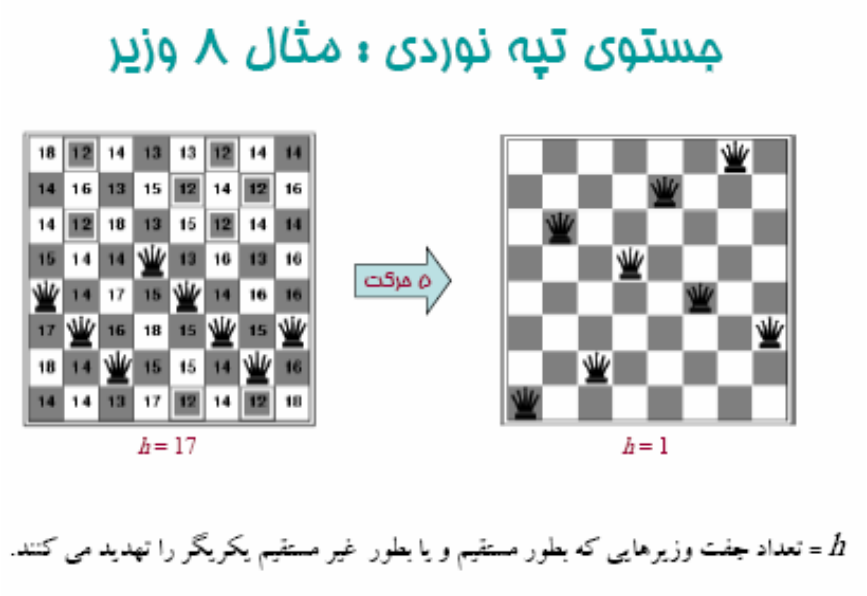
[87] Hill climbing

الگوریتم تپه نوردی به دلایل زیرمتوقف می شود:

- **ماکزیمم محلی**[88]:قله ای است که ازهمه همسایگانش بلندترمی باشداماازماکزیمم مطلق کوتاهتراست. (درمسئله ۸ وزیر،۸ وزیربدون برخورد ماکزیمم مطلق است ولی ۷وزیربدون برخوردویکی برخوردی ماکزیمم محلی است.)
- **دماغه ها**[89]:دماغه(تیغه) منجربه رشته ای ازماکزیمم های محلی می شودکه عبورازآن توسط الگوریتم های حریصانه کاردشواری است .(در مساله ۸ وزیر، چندحالت با۷ وزیربدون برخوردویکی برخوردی ایجادمی شود.)



- **فلات**[90]:ناحیه ازسطح فضای حالت است که درآن مقدارتابع ارزیابی یکسان است .دونوع فلات وجوددارد:ماکزیمم محلی صاف که درآن هیچ راهی به سمت بالاوجودنداردو شانه[91] که ازطریق آن می توان به بالاتررفت وپیشروی کرد.جستجوی تپه نوردی ممکن است نتواندرفلات راهش راپیداکند.



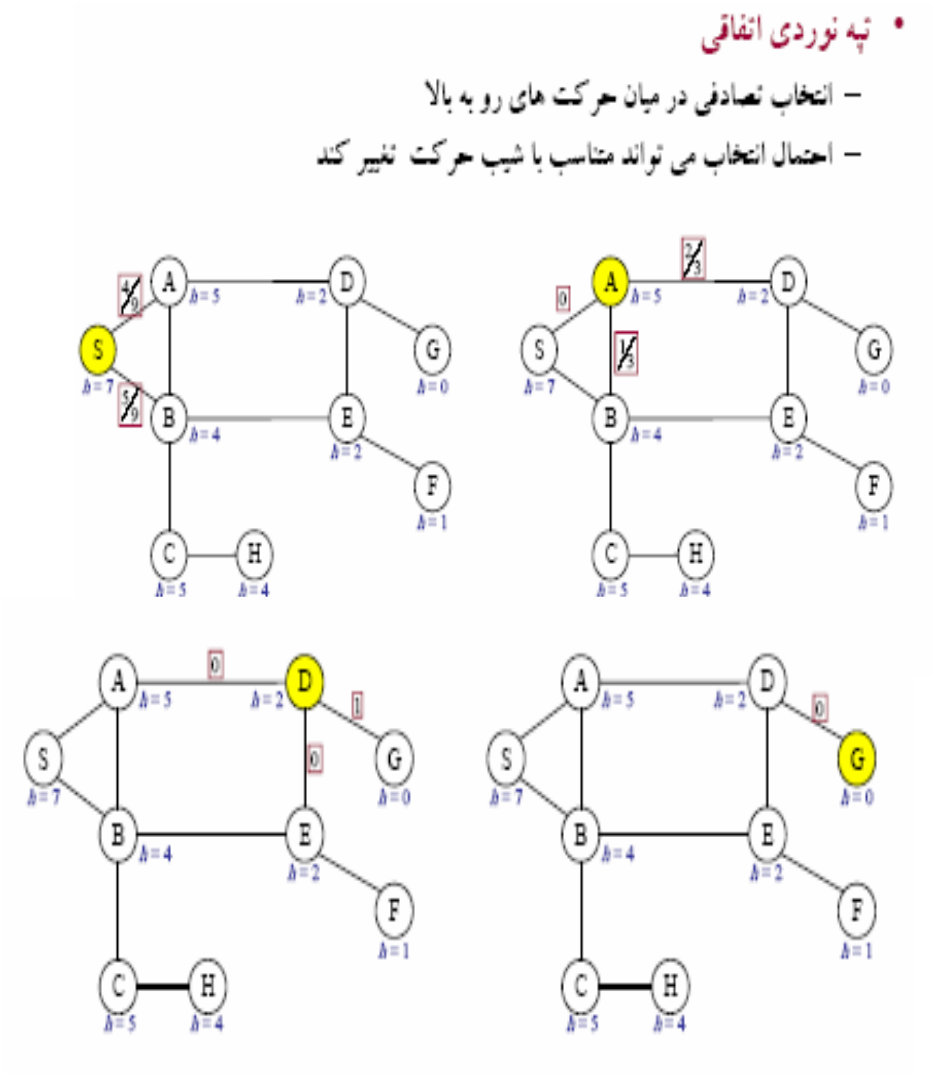
---

[88] Local maxima
[89] Ridge
[90] plateux
[91] shoulder

فرم های مختلفی از تپه نوردی مطرح شده است.الگوریتم تپه نوردی که مطرح شد تپه نوردی با تندترین شیب نیز نامیده می شود. این الگوریتم بسیارسریع عمل می کند. درمواردی که موفق است به طورمتوسط بعداز۴مرحله به جواب می رسد و وقتی ناموفق است بعداز۳مرحله متوقف می شود.و برای فضای حالتی با ۱۷ میلیون حالت مناسب است.انواع دیگر تپه نوردی عبارتند از:

- **تپه نوردی تصادفی[۹۲] :**

درهرمرحله ازمیان حرکتهای روبه بالای تپه یکی رابه طورتصادفی انتخاب می کند. احتمال انتخاب بامیزان شیب حرکت روبه بالا تغییرمی کند.این روش نسبت به روش تپه نوردی باتندترین شیب آهسته ترهمگرا می شود،اما دربعضی فضا های حالت، راه حل بهتری رامی یابد.

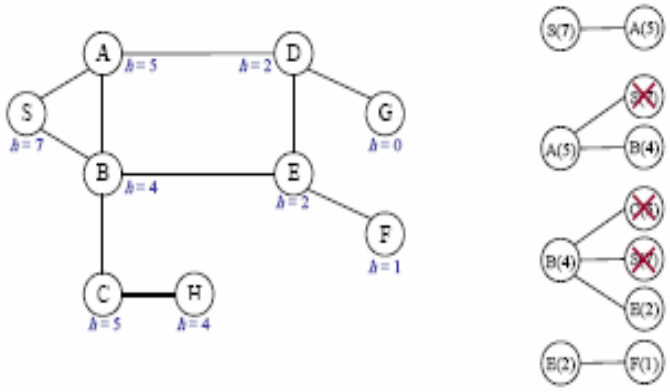


[92] Stochastic hill climbing

- **تپه نوردی اولین انتخاب [۹۳]:**

تپه نـوردی تـصادفی رابـه ایـن صـورت پیـاده سـازی مـی کنـد کـه بطـور تـصادفی مابعدهاراتولیـدمی کند تابالاخره یکـی ازآنهـا بهترازحالـت جـاری باشـد.ایـن روش بـرای مـسائلی کـه دارای مابعـدهای زیادی هستند مناسب است.

- **تپه نوردی اولین انتخاب**
  - همان تپه نوردی اتفاقی که حالت های بعدی را به طور تصادفی تولید می کند تا یکی از آنها بهتر از حالت فعلی باشد



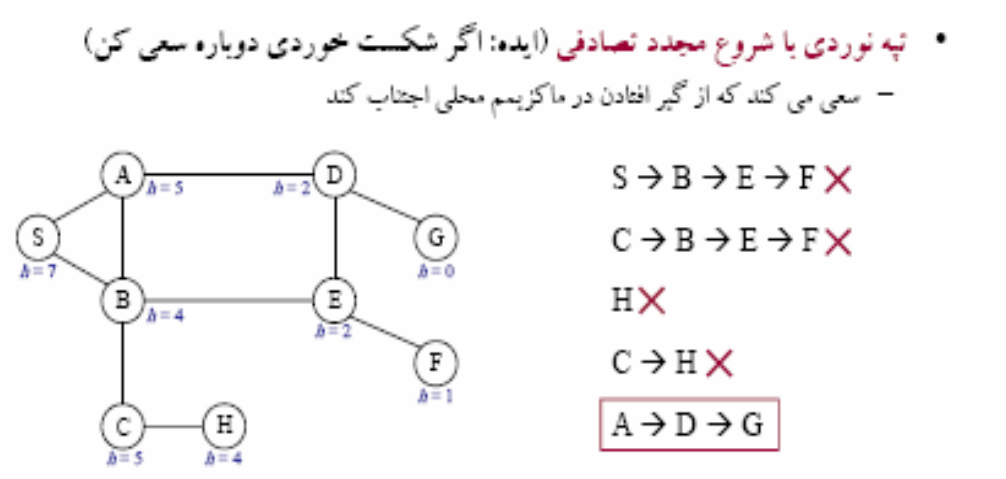
الگوریتمهای تپه نـوردی اشـاره شـده فـوق(تـصادفی-اولـین انتخـاب-تنـدترین شـیب) کامـل نیـستند،چون درماکزیمم محلی متوقف می شوند ودر ماکزیمم مطلق متوقف نمی شوند.

تپه نوردی باشروع مجددتصادفی[۹۴] :

ایـن روش معتقداسـت اگرموفـق نـشدید دوبـاره سـعی کنیـد. ایـن روش یـک سـری ازجـستجوهای تپـه نوردی راشامل می شود که ازحالت اولیـه تـصادفی شـروع و بارسـیدن بـه حالـت هـدف متوقـف مـی شـود. بااحتمال نزدیک بـه یـک ایـن روش کامـل است،زیراسـرانجام حالـت هـدف بـه عنـوان حالـت اولیـه تولیـد می شود.روش تپـه نـوردی باشـروع تـصادفی مجـددبرای مـسئله ۸- وزیر،مناسـبترین مـی باشـد. موفقیـت تپه نوردی بـه شـکل سـطح حالـت بـستگی دارد. اگرتعـداد مـاکزیمم هـای محلـی وفلاتهـا کـم باشـد تپـه نوردی باشروع مجدد تصادفی سریعا راه حل خوبی پیدا می کند.

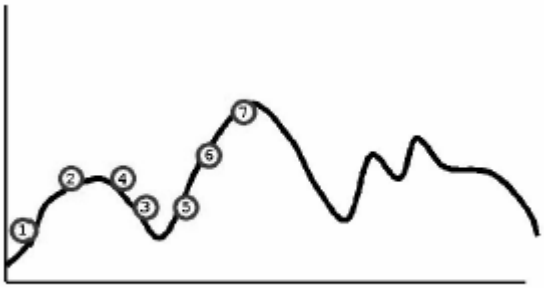
---

[93] First-choice hill climbing
[94] Random restart hill climbing



الگوریتم جست و جوی شبیه سازی حرارت:

الگوریتم تپه نوردی که اجازه ی حرکت رو به پایین یعنی به طرف حالات با ارتفاع کمتر یا هزینه ی بالاتر را نمی دهد کامل نیست، زیرا ممکن است در یک ماکزیمم محلی متوقف شود. در طرف مقابل، یک حرکت کاملا تصادفی یعنی انتخاب ما بعد ها به صورت تصادفی کامل است، اما کارا نیست. بنابر این تپه نوردی را با حرکت تصادفی ترکیب می کنیم تا هر دو ویژگی کامل بودن و کارایی را داشته باشد. شبیه سازی حرارت چنین الگوریتمی است . برای درک این الگوریتم، الگوریتم کاهش گرادیان را در نظر بگیرید و رفتن توپ پینگ پنگ به عمیق ترین شکاف در یک سطح ناصاف را در نظر بگیرید. اگر توپ بچرخد فقط در مینیمم محلی قرار می گیرد، اگر سطح را تکان بدهیم می توانیم توپ را از مینیمم محلی خارج کنیم، راه حل این است که سطح به اندازه ی کافی تکان داده شود تا توپ از مینیمم محلی خارج شود ، اما نباید طوری تکان داده شود که از مینیمم مطلق خارج شود. الگوریتم شبیه سازی حرارت با تکان شدید (دمای زیاد) شروع می کند و به تدریج شدت تکان دادن را کاهش می دهد (به سمت دمای پایین می رود). تفاوت تپه نوردی و شبیه سازی حرارت در این است که در تپه نوردی بهترین مابعد انتخاب می شود ولی در شبیه سازی حرارت مابعد ها به طور تصادفی انتخاب می شوند.الگوریتم شبیه سازی حرارت در طراحی مدارات VLSI ،زمان بندی کارخانه ها و مسائل بهینه سازی در مقیاس بزرگ، نسبت به تپه نوردی مؤثرتر است.



شکل ۳-۷: روند کلی آنالینگ شبیه‌سازی‌شده

جست و جوی پرتو محلی[95]:

در روش های قبلی مانند تپه نوردی و شبیه سازی حرارت فقط یک گره در حافظه نگه داشته می شد ولی در الگوریتم جستجوی پرتو محلی به جای یک گره، k گره در حافظه نگه داری می شود. این الگوریتم با k حالت که به طور تصادفی تولید شده اند، شروع می کند . در هر مرحله تمام ما بعد های k حالت را تولید می کند. اگر یکی از آنها هدف باشد الگوریتم متوقف می شود در غیر اینصورت k حالت از بهترین ما بعدها انتخاب شده و این عمل تکرار می شود.

تفاوت جست و جوی پرتو محلی با تپه نوردی شروع تصادفی مجدد در این است که در تپه نوردی شروع تصادفی مجدد هر فرایند جستجو مستقل از بقیه اجرا می شود. در حالیکه در الگوریتم های جستجوی پرتو محلی اطلاعات مفید بین k فرایند جستجوی موازی مبادله می شود و وابستگی بین آنها وجود دارد جستجوی پرتو تصادفی به جای انتخاب k حالت از بهترین مابعد ها ، k حالت را به طور تصادفی از میان مابعد ها انتخاب می کند ، به طوریکه انتخاب یک مابعد یک تابع تصادفی از میزان ارزش آن مابعد می باشد.

گام‌های جستجوی پرتوی محلی

۱- شروع جستجو با k حالت تصادفی

۲- ایجاد همه حالت‌های بعدی k حالت

۳- نگهداری k تا بهترین از حالت‌های ایجاد شده

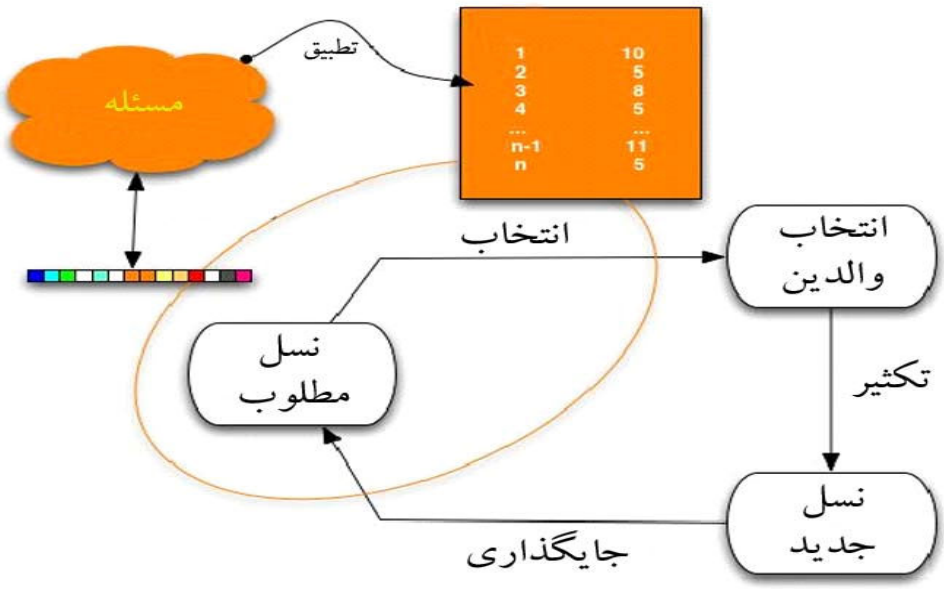
۴- بازگشت به گام دوم

---

[95] Local beam serach

الگوریتم ژنتیک[۹۶]:

الگوریتم ژنتیک نوعی از جستجوی پرتو اتفاقی است که در آن ما بعدها از ترکیب دو حالت والد(به جای تغییر یک حالت) تولید می شوند. الگوریتم ژنتیک مانند جستجوی پرتو محلی با مجموعه ای از k حالت تصادفی شروع می کند که این مجموعه جمعیت[۹۷] نام دارد. هر حالت یا فرد،به صورت رشته ای از تعداد متناهی الفبا(یا مجوعه ای از صفر و یک ها) نمایش داده می شود. در مساله ۸-وزیر حالت باید مکان ۸ وزیر را مشخص کند و هر وزیر باید در ستونی شامل ۸ مربع قرار گیرد. حالت ها در الگوریتم ژنتیک،کروموزوم نیز نامیده می شوند و هر کروموزوم از چندین ژن تشکیل شده است.الگوریتم ژنتیک یک الگوریتم تکاملی برای جست و جو در فضای وسیع است. رویه ی کلی الگوریتم ژنتیک به شرح ذیل می باشد:



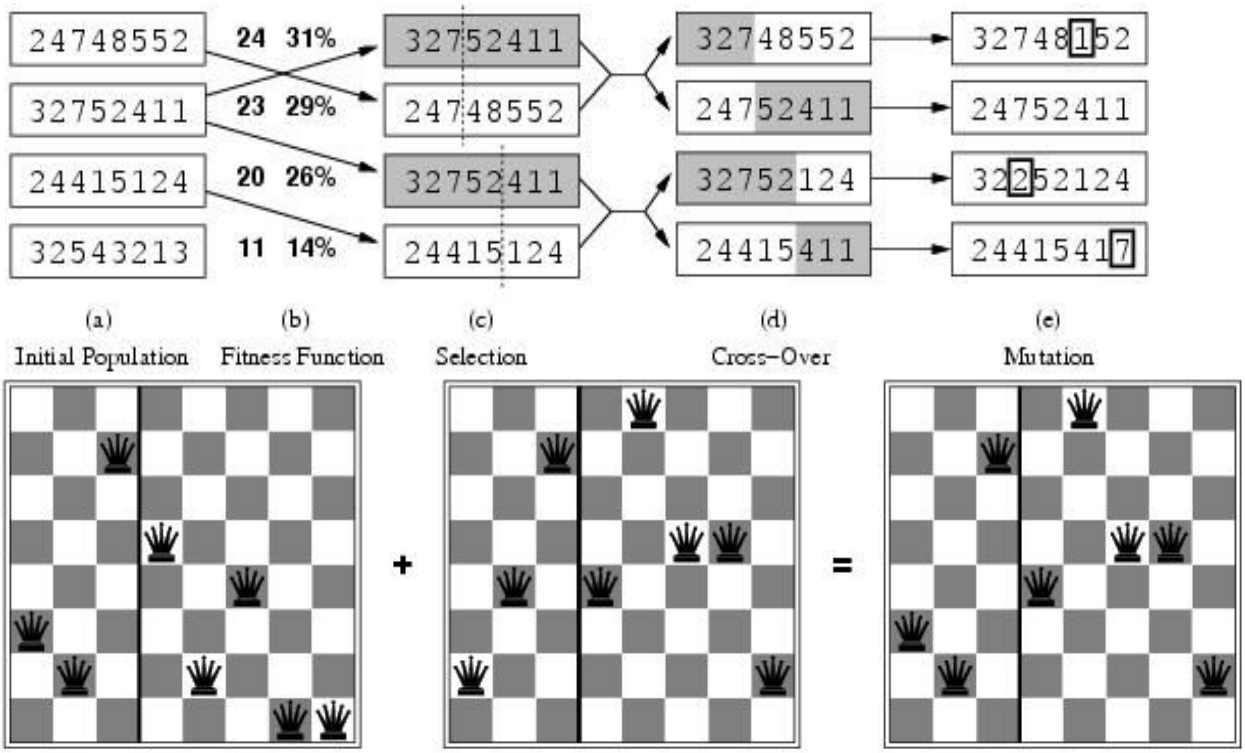
- **تولید جمعیت:** یک جمعیت مجموعه ای از کروموزوم ها ست که هر کروموزوم نمایانگر یک راه حل ممکن است. جمعیت اولیه می تواند توسط الگوریتم های مکاشفه ای به دست می آید.
- **ارزیابی کروموزوم ها:** به هر کروموزوم یک ارزش انتساب داده می شود . هدف جست و جوی ژنتیک پیدا کردن ارزش بهینه ی کروموزوم می باشد.

---

[96] Genetic Algorithm
[97] population

- **عملیات تبادل[۹۸] و جهش[۹۹]:** عملیات تبادل یک جفت از کروموزوم ها را به طور تصادفی انتخاب کرده سپس یک نقطه ی تصادفی از کروموزوم اول را انتخاب می کند ، بعد در هر دو کروموزوم از آن نقطه تا انتهای کروموزوم را با هم جابه جا می کند. عملیات جهش به طور تصادفی یک کروموزوم را انتخاب کرده و سپس یک ژن که معادل انتقال وزیر به ستون دیگر در ۸ وزیر است را در درون آن کروموزوم به طور تصادفی تغییر می دهد.
- در نهایت کروموزوم های جدید به دست آمده از جمعیت دوباره ارزیابی می شوند . تا اینجا یک مرحله از الگوریتم ژنتیک به پایان می رسد . این مرحله از الگوریتم یا به هدف از قبل تعیین شده میرسد یا دوباره از بین این جمعیت ، جمعیت تصادفی از کروموزوم ها انتخاب و الگوریتم تکرار می شود.

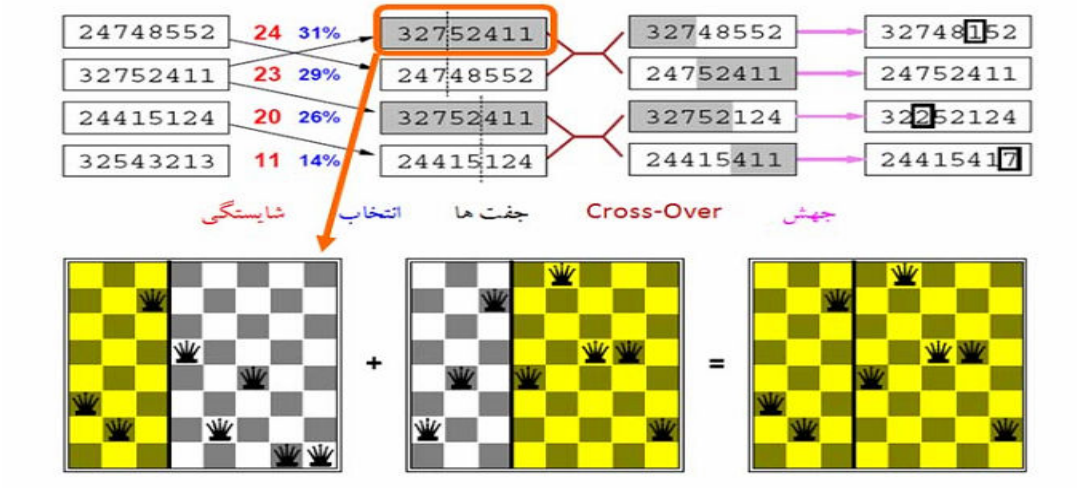
شکل زیر مثالی از الگوریتم ژنتیک برای حل مساله ۸-وزیر را نشان می دهد.



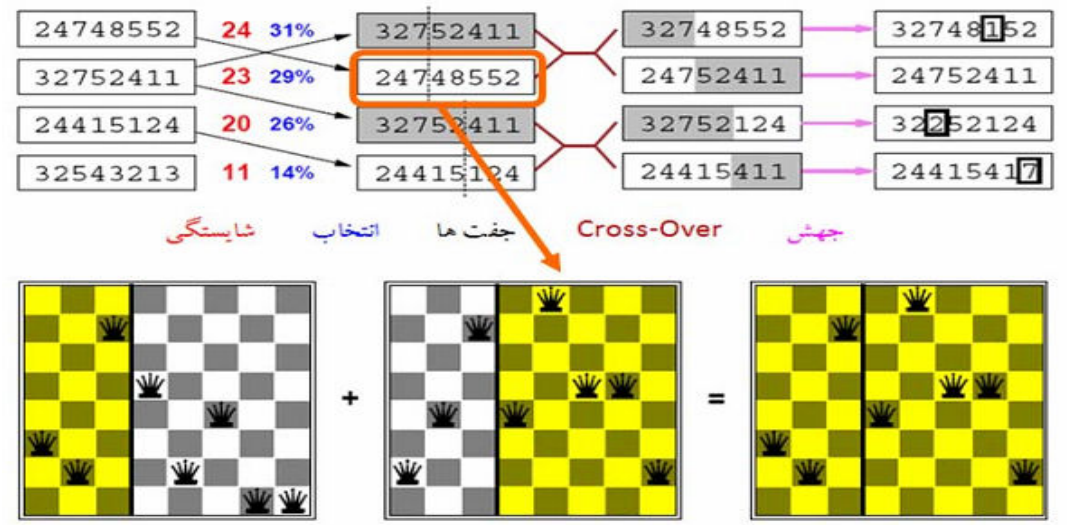
---

[98] Cross-over
[99] mutation

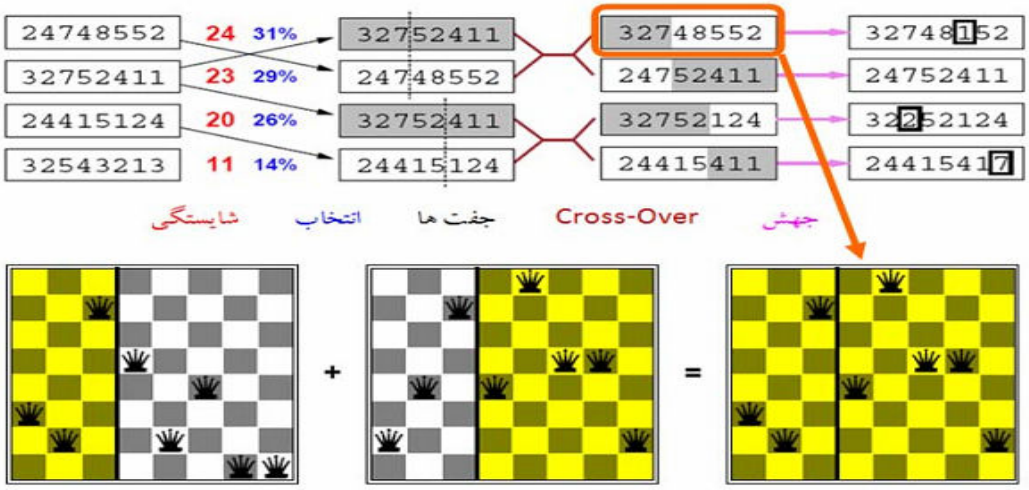
گام بعدی :

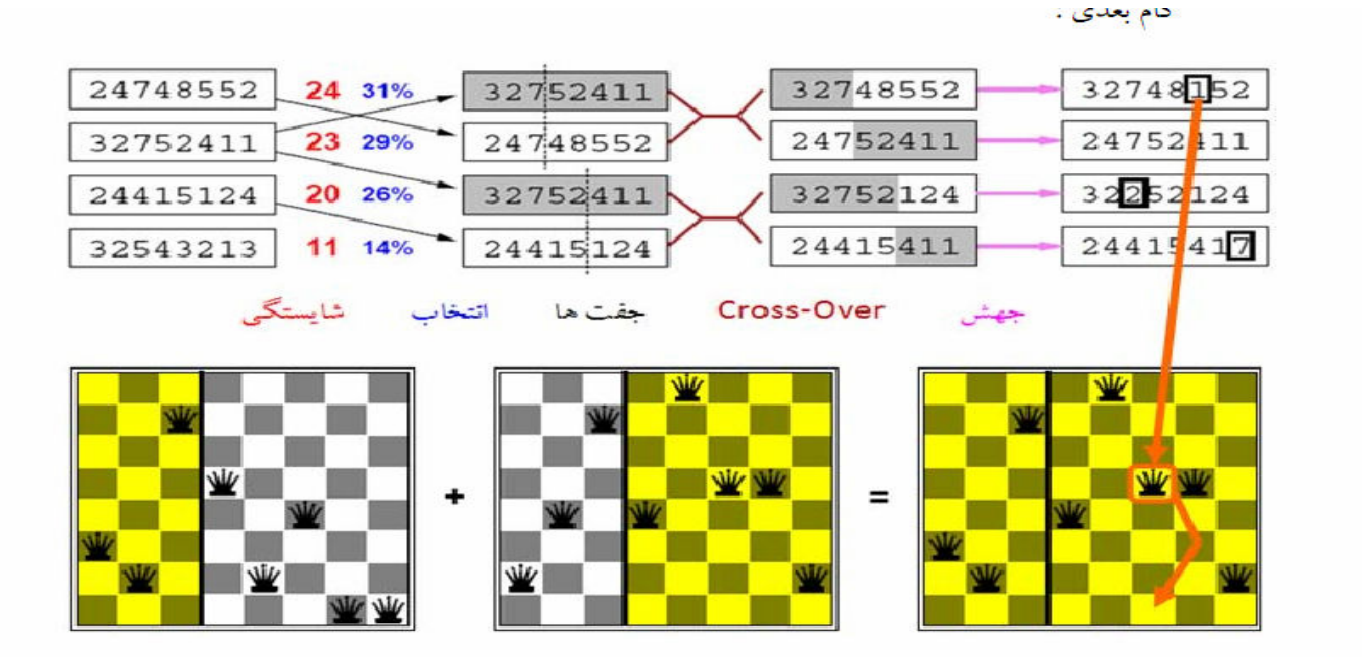


گام بعدی .



گام بعدی :

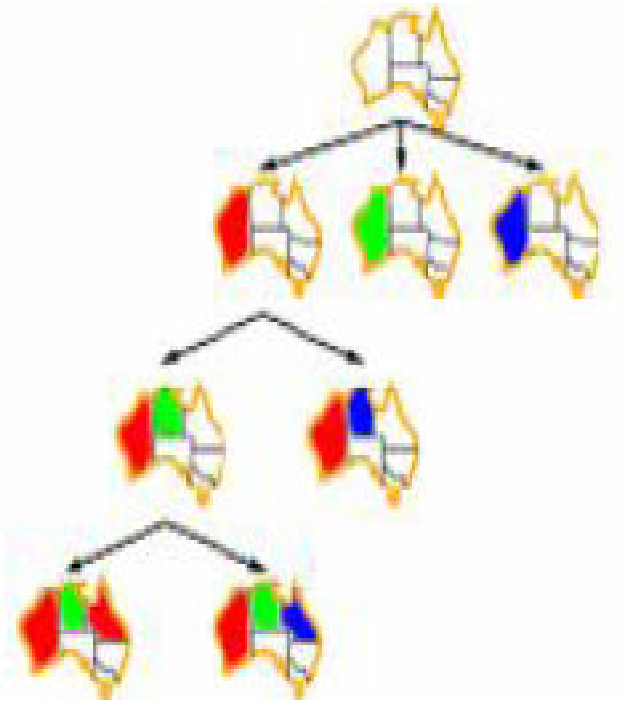



گام بعدی .
24748552
32752411
24415124
32543213
24 31%
23 29%
20 26%
11 14%
32752411
24748552
32752411
24415124
32748552
24752411
32752124
24415411
32748152
24752411
32252124
24415417
شایستگی
انتخاب
جفت ها
Cross-Over
جهش

فصل پنجم: مسائل ارضای محدودیت(CSP)

آنچه در این فصل خواهید آموخت:

- **تعریف مسائل ارضای محدودیت و چند مثال**
- **جستجوی عقبگرد برای CSP**
- **کنترل روبه جلو(Forward Checking)**
- **استفاده از هیورستیک ها برای حل مسائل CSP**
- **جستجوی محلی برای مسائل ارضای محدودیت**

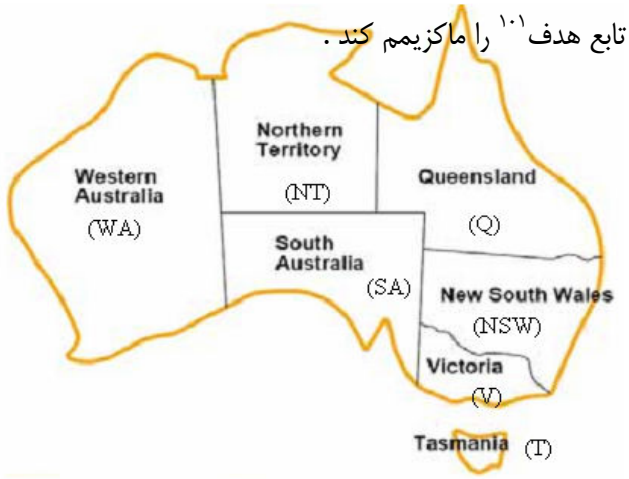


مسائل ارضای محدودیت(CSP[100]):

مسائل ارضای محدودیت یا **CSP** توسط مجموعه ای از متغیرهای $x_1, x_2, ..., x_n$ ومجموعه ای از محدودیتهای $c_1, c_2, ..., c_n$ تعریف می شوند هر متغیر x_i دارای دامنه ی نا تهی D_i از مقادیر ممکن می باشد هر محدودیت c_i شامل تعدادی زیر مجموعه از متغیر هاست به طوری که آن محدودیت مقادیر ترکیبات مجاز این زیر مجموعه ها را مشخص می کند. هر حالت با انتساب مقادیری به چند متغیر یا تمام آنها تعریف می شود بنابراین در حالت اولیه هیچ یک از متغیر ها مقدار ندارند .

- **انتساب سازگار :** انتسابی است که هیچ محدودیتی را نقض نمی کند .
- **انتساب کامل :** انتسابی است که هر متغیری در آن باشد .
- **راه حل :** راه حل ها در مسئله CSP انتسابهای سازگار و کامل می باشند یعنی یک راه حل شامل تمام متغیر هاست علاوه بر آن تمام محدودیت ها را نیز برآورده می کند بعضی از CSPها به راه حلهایی نیاز دارند که تابع هدف[101] را ماکزیمم کند .

مثال :



مجموعه ای از متغیر ها : $\{WA, NT, SA, Q, NSW, V, T\}$

مجموعه ای از مقادیر(دامنه) : { آبی (B)، سبز (G)،قرمز(R)}

محدودیت ها : دو استان همجوار همرنگ نباشند

$c_1 : WA \neq NT, c_2 : WA \neq SA, c_3 : NSW \neq V, \ldots$

---

[100] constraint satisfaction problem
[101] Objective function

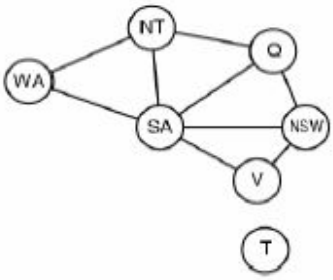
WA,NT={(R,B)(R,G),(B,R),(B,G),(G,R),(G,B)}

حالت:{ WA=قرمز, NT= سبز , T=سبز,V=قرمز,NSW=سبز,Q=قرمز,SA=آبی }

این حالت هم سازگار و هم کامل است پس یک راه حل است .

گراف محدودیت : یالها : محدودیت ها گره ها : متغیر ها

- گراف محدودیت:
  - گره ها متغیرها را نشان می دهند.
  - یالهای گراف محدودیت ها ی بین متغیرها را نشان می دهند.

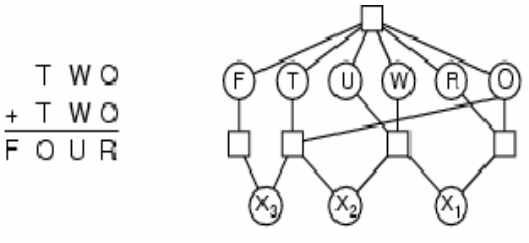


مسئله رمزنگاری[۱۰۲] (معمای ریاضیات) :

هدف در مسئله معمای ریاضیات این است که در آن هر حرف یک رقم متفاوت از سایرین به خود بگیرد به طوریکه جمع ارقام انتساب یافته صحیح می باشد

```
FORTY      29786
+ TEN      + 850
+ TEN      + 850
----------  ----------  {F=2, O=9, R=7, T=8,Y=6,E=5,N=0,S=3,I=1,X=4}
 SIXTY     31486
```

مثال :



[102] Cryptography

متغیرها : $\{T,W,O,F,U,R,x_1,x_2,x_3\}$ دامنه : $\{0,1,2,...,9\}$

محدودیت ها : متغیر ها مقادیر متفاوت داشته باشند . $T \neq F \neq W \neq O \neq U \neq R$

All different $\{T,W,O,F,U,R\}$

محدودیت Alldifferent را می توان به محدودیت های دو دویی تبدیل کرد به طور کلی اگر متغیر های کافی برای یک مسئله CSP در نظر گرفته شود هر محدودیت مرتبه ی بالا می تواند به محدودیت های دودویی تبدیل شود .

مسئله رمزنگاری را می توان به صورت زیر فرموله سازی کرد:

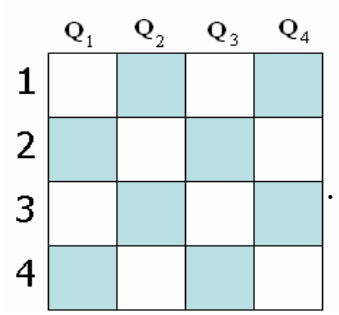
حالات : یک معمای رمزنگاری با چند حروف جایگزین شده توسط ارقام .

عملگر ها : وقوع هر حرف را با یک رقم جایگزین کنید که قبلاً در معما ظاهر نشده است .

آزمون هدف : معما فقط شامل ارقام است ویک مجموعه از اعداد صحیح برمی گرداند.

هزینه ی مسیر : صفر.

مسئله ۴-وزیر :



متغیر ها : $\{Q_1,Q_2,Q_3,Q_4\}$ دامنه : $\{۴،۳،۲،۱\}$

محدودیت ها : وزیر ها نمی توانند در یک سطر یا ستون یا قطر قرار گیرند .

Q_1,Q_2 برای : $\{(۱،۳)(۱،۴)(۲،۴)(۳،۱)(۴،۱)(۴،۲)\}$

مزایای بیان مسئله به صورت CSP عبارت اند از :

- به دلیل نمایش استاندارد حالت ها (مجموعه ی متغیرهایی با مقادیرشان) می توان تابع Successor و آزمون هدف را به شکل کلی نوشت به طوریکه برای هر CSP قابل اعمال باشد .
- می توان هیوریستیک های کلی و کارایی ایجاد نمود که نیاز به تخصص اضافی در دامنه ی خاص مسئله نداشته باشند .

نکته:یک مسئله CSP می تواند با استفاده از فرموله سازی افزایشی مانند یک مسئله ی جستجوی استاندارد ارائه شود.

حالت اولیه : انتساب خالی که در آن هیچ متغیری مقدار ندارد .

تابع ما بعد : یک مقدار می تواند به هر متغیر فاقد مقدار نسبت داده شود اگر با متغیر هایی که قبلاً مقدار گرفته اند تضاد نداشته باشد .

آزمون هدف : آیا انتساب فعلی کامل است یا نه ؟

هزینه ی مسیر : یک هزینه ی ثابت برای هر مرحله .

نکته: هر راه حل یک انتساب کامل است لذا اگر در مسئله n متغیر وجود داشته باشد راه حل در عمق n خواهد بود و درخت جستجو دارای عمق n می باشد بنابراین در بین جستجوها ، جستجوی عمقی، مناسب ترین برای حل یک مسئله CSP می باشد البته اگر از فرموله سازی حالت کامل استفاده کنیم الگوریتم های جستجوی محلی نیز می توانند برای مسائل CSP مفید باشند .

انواع مسائل CSP:

۱) **گسسته و متناهی** : ساده ترین نوع مسائل CSP شامل متغیرهای گسسته با دامنه های متناهی می باشد مانند مسئله رنگ آمیزی نقشه ، مسئله ۸ وزیر ، مسئله رمزنگاری ریاضی ، CSPهای بولین . اگر در یک مسئله CSP حداکثر اندازه دامنه هر متغیر برابر d باشد و Nتعداد متغییرها باشد آنگاه تعداد انتسابهای کامل $O(d^N)$ میباشد .

۲) **گسسته ونامتناهی:** متغیر های گسسته ممکن است دامنه های نامتناهی نیز داشته باشند مانند مجموعه ای از اعداد صحیح یا ر شته ها . به عنوان مثال در زمانبندی کارها ، تاریخ شروع هر کار یک متغیر است و مقادیر ممکن آنها اعداد صحیح می باشند در مسائل با دامنه های نامتناهی نمی توان محدودیت هارا با شمارش تمام ترکیبات مجاز مقادیر تعریف کرد به جای آن باید از زبان محدودیت استفاده شودبه عنوان مثال اگر job_1 که ۵ روز طول میکشد باید قبل از job_2 انجام شود به یک زبان محدودیت از نامساوی های جبری مثل $startjob_1 + 5 \le startjob_2$ نیاز است . علاوه براین، چنین

محـدودیتهایی را نمـی تـوان بـا شـمارش تمـام انتـسابهای ممکـن حـل کـرد زیـرا تعـداد آنهـا نامتناهی است .

۳) **پیوسـته** : در دنیـای واقعـی مـسائل ارضـای محـدودیت بـا دامنـه هـای پیوسـته بـسیار متـداول هـستند مـثلاً زمانبنـدی آزمایـشات روی تلـسکوپ فـضایی هابـل بـه مـشاهدات زمـانی دقیقـی نیاز دارد . شروع وپایان هر مـشاهده متغیـر هـای پیوسـته ای هـستند کـه بایـد از قـوانین نجـومی تبعیـت کننـد . معروفتـرین دسـته از CSPهـای بـا دامنـه ی پیوسـته، مـسائل برنامـه نویـسی خطی هستند که در آنها محـدودیت هـا بایـد بـه صـورت نامـساوی هـای خطـی باشـند کـه یـک ناحیه ی محدب ایجاد کننـد . مـسائل برنامـه نویـسی خطـی مـی توانـد در زمـان چنـد جملـه ای بر اساس تعداد متغیر ها حل شود .

انواع محدودیت ها :

الف) **یکانی**[۱۰۳] **:** روی یک متغیر باشد . $NT \neq \text{Red}$

ب) **دوگانی**[۱۰۴]**:** محدودیت روی دو متغیر باشد . $WA \neq NT$

ج) **مرتبه بالاتر**[۱۰۵] **:** محدودیت روی سه یا بیشتر متغیر باشد مانند مسئله رمزنگاری ریاضی

$$T \neq F \neq W \neq O \neq U \neq R$$

محدودیت هایی که ذکر شد محدودیت های کامل یا مطلق[۱۰۶] نامیده می شوند که نقض یک محـدودیت مطلق به معنای حذف یک راه حل بالقوه می باشد .

محدودیت های اولویت دار[۱۰۷] **:** در این محدودیت ها یک راه حل بر سایر راه حلها ترجیح داده می شود به عنوان مثال در برنامه زمانی دانشگاه پروفسور **x** ترجیح می دهد صـبح تـدریس کنـد در حالیکـه پروفـسور **y** ترجیح می دهد بعد ازظهر تدریس کند اگر جدول زمانی به گونه ای تنظیم شود کـه پروفـسور **x** در سـاعت ۲ بعد از ظهر تدریس کند یک راه حل محسوب می شود اما راه حل بهینه نیست. محـدودیت هـای اولویـت دار می توانند به صورت هزینه در انتساب هر یک از متغیر ها اعمال شوند .

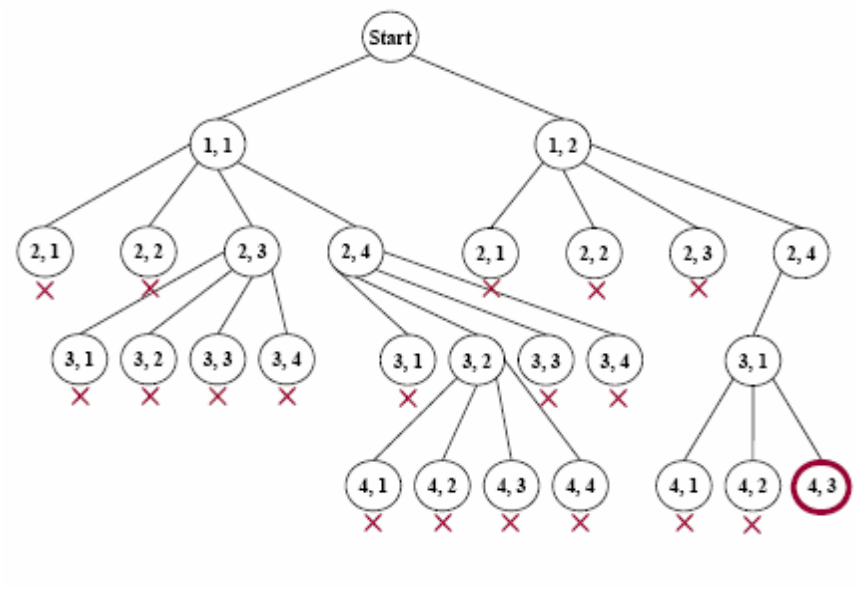
---

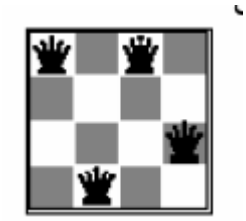
[103] unary
[104] binary
[105] High order
[106] Absolute
[107] preference constraints

نکته: مسائل زیر برای حل به روش CSP مناسبند : مسئله زمانبندی ، ۸ وزیر ، حل جدول کلمات متقاطع ، رنگ آمیزی نقشه ، معمای ریاضی (رمزنگاری) ، زمانبندی امتحانات ، چینش ماژولها روی یک تراشه و مسائل دنیای واقعی از قبیل : مسائل انتسابی مانند اینکه چه کسی چه کلاسی را درس می دهد ، مسائل زمانبندی حمل ونقل و زمانبندی کارخانه .

جستجوی عقبگرد[108] برای CSP: اصطلاح جستجوی عقبگرد برای جستجوی عمقی به کار میرود که در هر سطح یک متغیر را مقدار میدهد و وقتی مقدار معتبری برای انتساب به یک متغیر وجود نداشته باشد به عقب برمی گردد . جستجوی عقبگرد یک الگوریتم ناآگاهانه است که برای حل مسائل بزرگ ناکارامد است .به مثال زیر توجه کنید .

مثال:حل مساله ۴-وزیر به روش عقبگرد

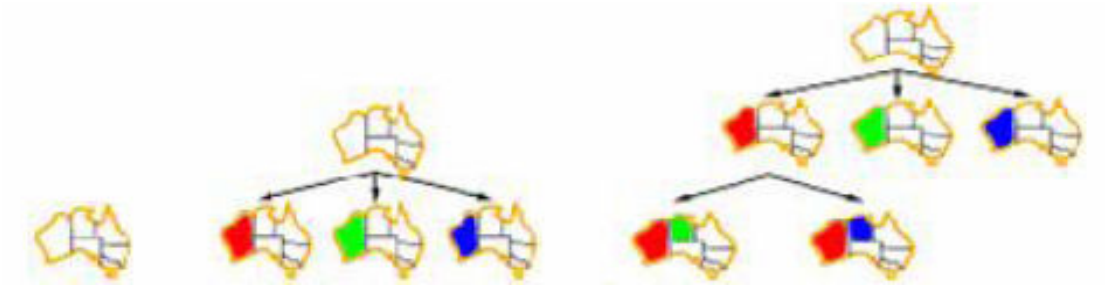


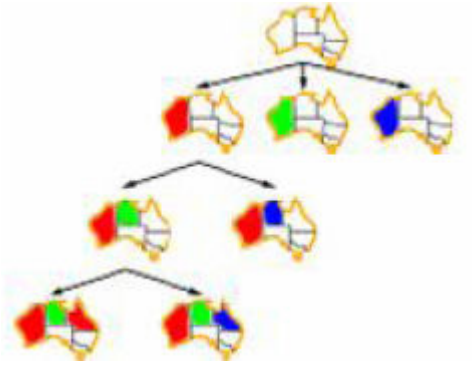


راه حل: {(۱،۲)(۲،۴)(۳،۱)(۴،۳)}

[108] Back Tracking Search

مثال:حل مساله رنگ آمیزی نقشه به روش عقبگرد





ترتیب انتخاب متغیر ها :

الف) **مقادیر باقیمانده کمینه**[109] **(MRV)یا محدودیت ترین متغیر**[110] **یا اولین شکست**[111]

ایده : متغیری را انتخاب کن که کمترین مقادیر معتبر را دارد .

ب) **اکتشاف درجه ای :**

ایده : متغییر با بیشترین درجه اول انتخاب میشود .

ج) **اکتشاف مقدار با کمترین محدودیت**[112]

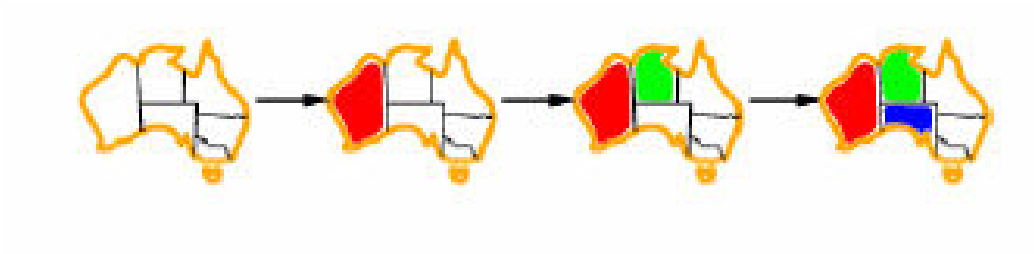
ایده : متغیری انتخاب می شود که کمترین مقادیر را از متغیرهای باقیمانده حذف میکند .

الف) **MRV**: ایده این روش این است که متغیری را انتخاب میکند که کمترین مقادیر معتبر را دارد یا به عبارتی محدودترین متغیر را انتخاب میکند . محدودترین متغیر ، متغیری است که

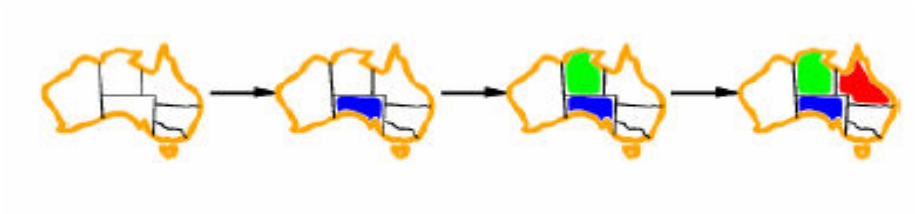
---

[109] Minimum Remaining Value
[110] Most Constraint Value
[111] first failure
[112] least constraint value

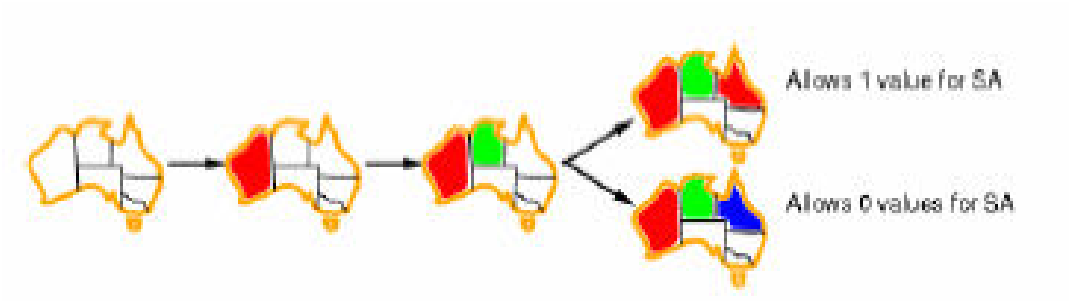
کـوچکترین دامنـه را دارد در ایـن صـورت متغیـری انتخـاب مـی شـودکه بـه احتمـال زیـاد بـا شکـست مواجه شده ودرخت جستجو با هرس مواجه خواهدشد .



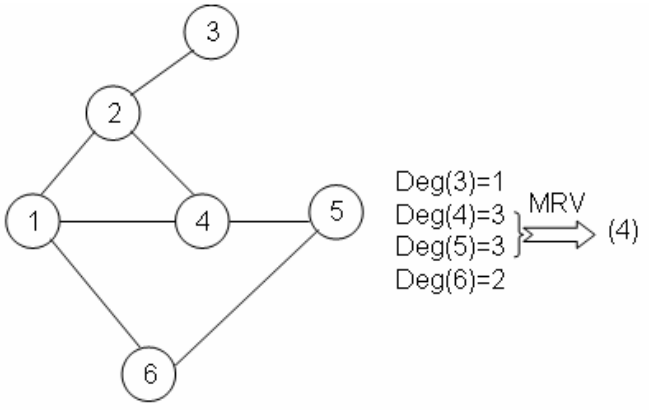
ب) **اکتشاف درجه ای**: سـعی میکنـد فـاکتور انـشعاب را در انتخـاب آینـده کـم کنـد متغیـری انتخـاب مـی شـود کـه بیـشترین محـدودیت را روی متغیـر هـای انتـساب نیافتـه اعمـال میکنـد ایـن هیوریـستیک برای انتخاب اولین متغیر جهت انتساب مقدار مناسب است .



ج) **اکتشاف مقدار با کمتـرین محـدودیت** : ایـن روش مقـداری را تـرجیح مـی دهـد کـه در گـراف محـدودیت ، متغیرهـای همـسایه بـه نـدرت آنـرا انتخـاب مـی کننـد و سـعی در ایجـاد بیـشترین قابلیـت انعطاف برای انتساب بعدی متغیر هـا دارد ایـده ی ایـن روش بـرای یـک متغیـر ایـن اسـت کـه مقـداری را که کمترین محـدودیت را ایجـاد مـی کنـد انتخـاب مـی کنـد یـا مقـدار را کـه کمتـرین مقـادیر را از متغیـر های باقیمانده حذف می کند .



مثال: در صورتی که بخواهیم با استفاده از روش CSP گراف مقابل را با سه رنگ **R,G,B** رنگ آمیزی کنیم بعد از رنگ آمیزی رئوس ۱و ۲ کدام راس رنگ آمیزی خواهد شد ؟ (مهندسی کامپیوتر–۸۷)



بررسی پیشرو[113] (FC)

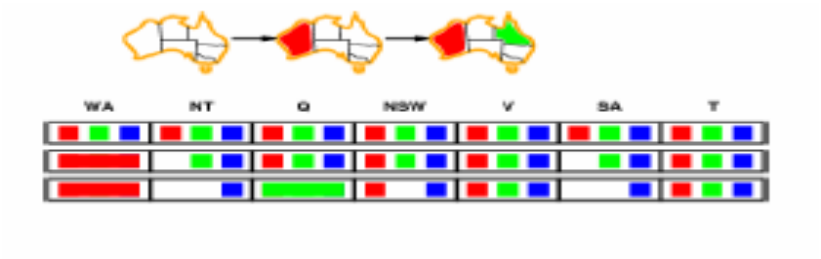
ایده:وقتی متغیری مقدار گرفت آن مقدار از همسایگانش حذف می شود .

الگوریتم هنگامی خاتمه می یابد که برای یک متغیر هیچ مقدار مجاز باقی نمانده باشد .

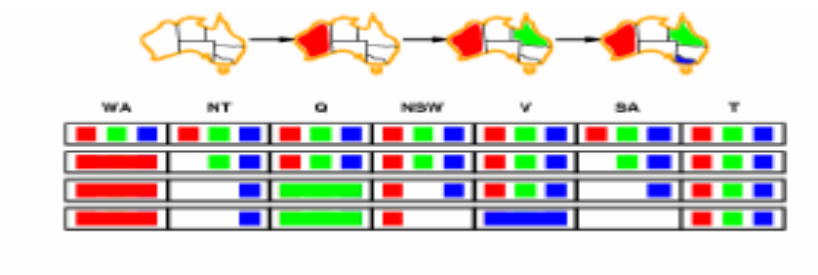
گام اول:

گام دوم:

گام سوم:



گام چهارم:



---

[113] Forward Checking

به متغیری رسیدیم که هیچ مقدار مجازی برایش باقی نمانده(متغیر **V**) پس الگوریتم خاتمه می یابد .

جدول **FC** :

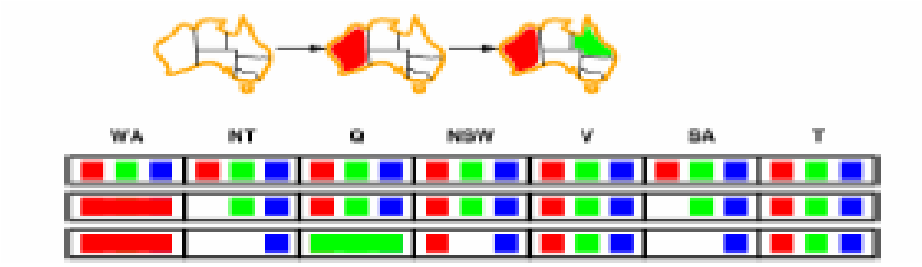
	WA	NT	Q	NSW	V	SA	T
Initial domains	R G B	R G B	R G B	R G B	R G B	R G B	R G B
After WA=red	R	G B	R G B	R G B	R G B	G B	R G B
After Q=green	R	B	G	R B	R G B	B	R G B
After V=blue	R	B	G	R	B		R G B

وقتی انتساب به X صورت می گیرد فرایند بررسی پیشرو متغیرهای بدون انتساب مثل y را در نظر می گیرد که از طریق یک محدودیت به X متصل است و هر مقداری را که با مقدار انتخاب شده برای **x** برابر است از دامنه **y** حذف می کنیم . زمانی که یک متغیر هیچ مقدار مجاز باقیمانده ندارد جستجو خاتمه می یابد پس از انتساب آبی=V، دامنه SA خالی می شود لذا بررسی پیشرو تشخیص می دهد که انتساب« **WA=R , Q=G , V=B**» با محددیت های مساله سازگار نیست و الگوریتم به سرعت به عقب بر می گردد. این روش جستجو، تناقض را سریع تر از جستجوی عقب گرد ساده پیدا می کند .

پخش (انتشار) محدودیت :

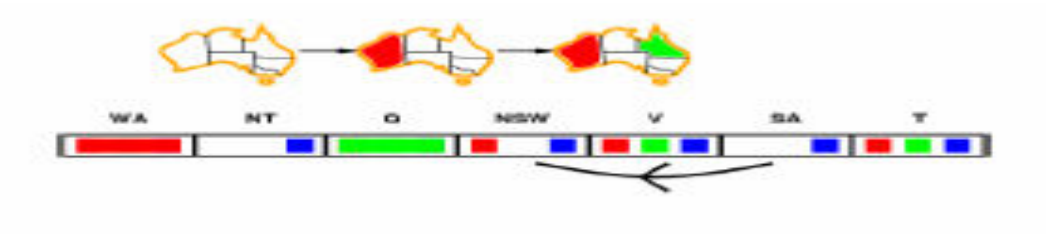
اگر چه بررسی پیشرو بسیاری از ناسازگاری ها را کشف می کند اما همه آنها را نمی تواند تشخیص بدهد . برای مثال در جدول FC سطر سوم وقتی WA=R و Q=G است SA, NT فقط می توانند آبی باشد. اما این دو متغیر همجوارند و نمی توانند رنگ یکسانی داشته باشند. جستجوی بررسی پیشرو این مورد را به عنوان ناسازگاری تشخیص نمی دهد .

بنابراین بررسی پیشرو محدودیت ها را از متغیرهای انتساب یافته به متغییرهای انتساب نیافته منتشر می کند اما تمام شکست ها را نمی تواند در زودترین زمان ممکن تشخیص بدهد. پخش محدودیت یعنی تاثیر محدودیت روی یک متغیر بر سایر متغیرها.انتشار محدودیت به طور مکرر بر محدودیت ها به طور محلی تاکید دارد. در مثال قبل ابتدا اطلاعات محدودیت از **G,WA** به ترتیب به **NT,SA** پخش شد.

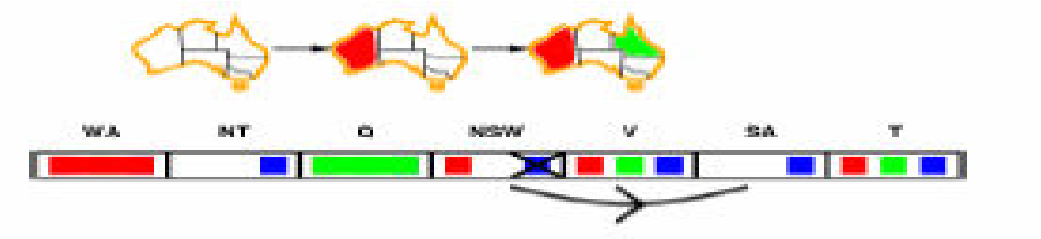


سازگاری یال[114] : ایده سازگاری یال یک روش صحیح برای پخش محدودیت ها فراهم می کند و در نتیجه بسیار قوی تر از جستجوی بررسی پیشرو است . منظور از ARC یال جهت دار در گراف محدودیت است . یال $x \rightarrow y$ سازگار است اگر و فقط اگر به ازای هر مقدار **X** ، بعضی مقادیر مجاز برای **y** موجود باشد .

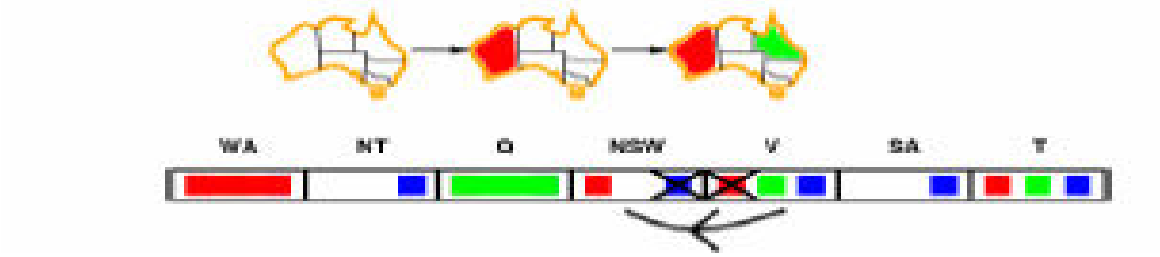
SA→NSW سازگار است اگر SA=blue و NSW=red.



NSW→SA سازگار می شود با حذف blue از NSW.

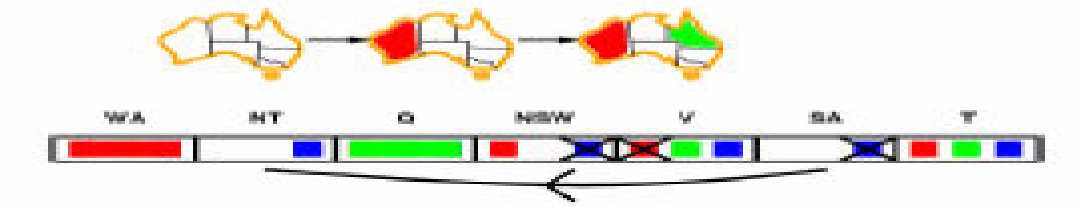


V→NSW سازگار می شود با حذف blue از NSW و red از V.



[114] Arc consistency

SA→NT سازگار می شود با حذف blue از NSW,SA و red از V.



بررسی Arc consistency می تواند به صورت یک مرحله پیش پردازش قبل از شروع جستجو انجام شود یا به صورت یک مرحله پخش مانند بررسی پیشرو پس از هر انتساب در حین جستجو که به آن روش MAC[115] گفته می شود انجام شود . در هر موقع این فرایند باید به صورت مکرر انجام گیرد تا هیچ ناسازگاری باقی نماند زیرا هر زمان که به منظور رفع ناسازگاری یال یک مقدار از دامنه متغیری حذف می شود ممکن است در یال هایی که به آن متغیر اشاره می کنند ناسازگاری جدید ایجاد شود . پیچیدگی زمانی Arc consistency در بدترین حالت $O(n^2 d^3)$ میباشد که در آن d تعداد مقادیر دامنه و n تعداد متغیرهاست . اگر چه روش سازگاری یال نسبت به بررسی پیشرو هزینه ی بیشتری دارد اما مفیدتر است. زیرا تناقض(عدم موفقیت) را زودتر از روش بررسی پیشرو تشخیص می دهد.

سازگاری k(K- consistency): سازگاری یال تمام ناسازگاری های ممکن را نمی تواند تشخیص دهد مثلا انتسابWA=R , NSW=R ناسازگار است ولی سازگاری یال نمی تواند آن را تشخیص دهد بنابراین به کمک سازگاری K شکل قوی تری از پخش محدودیت را می تواند تعریف کرد .

تعریف K- consistence : یک مساله ارضای محدودیت K- consistence است اگر برای هر k-1 متغیر و برای هر انتساب سازگار آن متغیرها همیشه یک مقدار سازگار برای هر **K** امین متغیر وجود داشته باشد. به عنوان مثال معنای 1-consistence این است که هر متغیر با خودش سازگار است این سازگاری Node consistence نامیده می شود .2-consistence همان سازی یال است. معنای 3-consistence این است که هر جفت از متغیرهای همجوار می توانند به سومین متغیر همسایه گسترش یابند این سازگاری ، سازگاری مسیر یاPath consistence نامیده می شوند. یک گراف قویاً K- consistence است هرگاه -1,

[115] Maintaining Arc Consistency

2-consistence consistence و ..... k-2consistence وk-1consistence باشد اگر گراف قویاً K-consistence باشد می توان مساله را بدون عقب گرد حل کرد که پیچیدگی زمانی آن O(nd) است

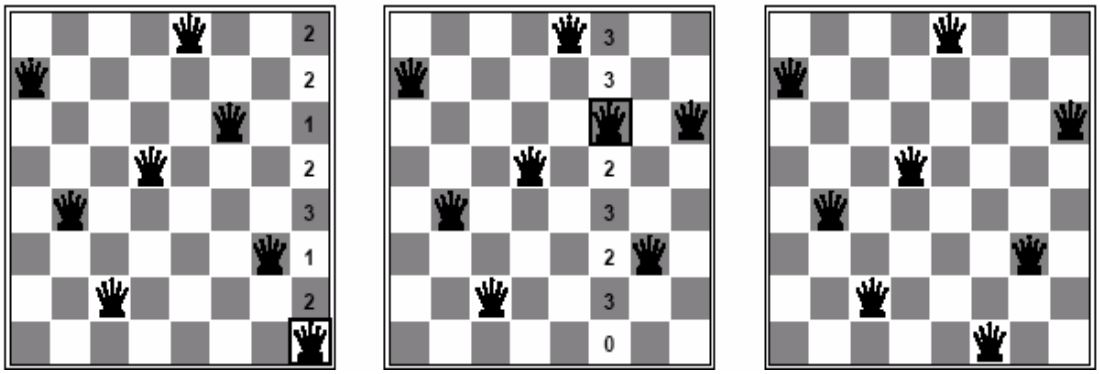
ترتیب تشخیص تناقض:

عقب گرد < FC < ARC consistence < K- consistence < k- consistence قویاً

جستجوی محلی برای CSP :

الگوریتم های جستجوی محلی مانند تپه نوردی و simulated annealing ،بسیاری ازمسائل CSP را به طورکارایی حل می کنند. آنها از فرموله سازی حا لت کامل استفاده می کنند یعنی تمام متغیرها باید دارای مقدار باشد . حالت اولیه به همه متغیرها مقداری را نسبت می دهد و تابع مابعد در هر مرحله مقدار یک متغیر را تغییر می دهد . به عنوان مثال در مسئله۸- وزیر حالت اولیه آن یک ترکیب تصادفی از مکان هشت وزیر در هشت ستون است و تابع مابعد یک وزیر را انتخاب نموده و به جای دیگری در همان ستون انتقال می دهد . در انتخاب مقدار جدید برای یک متغیر، **هیوریستیک مینیمم تناقضات**[۱۱۶]، بهترین هیوریستیک است این تابع مقداری را انتخاب می کند که کمترین برخورد را با سایر متغیرها ایجاد می کند یعنی مقداری را انتخاب کن که کمترین تعداد محدودیت ها را نقض کند . زمان اجرای **هیورستیک مینیمم تناقضات**،مستقل از اندازه مسئله خواهد بود.

هیورستیک مینیمم تناقضات برای حل مسائل سخت مانند زمان بندی مشاهدات تلسکوپ هابل به کار می رود مزیت دیگر جستجوی محلی این است که می تواند در صورت تغییر در مسئله تنظیمات را به صورت آنلاین انجام دهد.

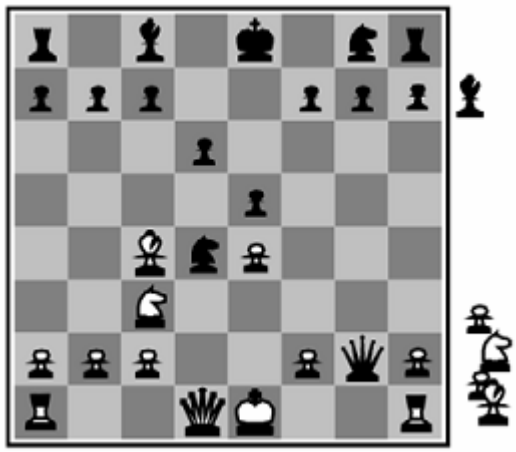


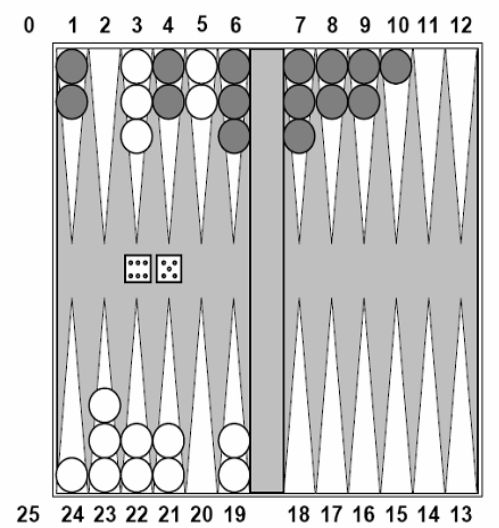
[116] Min-conflicts Heuristic

فصل ششم: جستجوی رقابتی

آنچه در این فصل خواهید آموخت:

- **بازی ها و دلایل مطالعه آنها**
- **الگوریتم Mini-Max**
- **بازی های چند نفره**
- **هرس آلفا-بتا**
- **تصمیمات بلادرنگ ناقص**
- **بازیهایی که حاوی عنصر شانس هستند**





مقدمه:

بازی ها حالتی از محیط های چند عامله هستند در محیط های چند عامله، هر عامل باید اعمال سایر عامل ها و چگونگی تاثیر آن ها را مورد توجه قرار دهد به تفاوت بین محیط های چند عامله رقابتی و چند عامله همکار توجه داشته باشید محیط های رقابتی که در آن اهداف عامل در تناقض با یکدیگر است منجر به مسایل جست و جو ی خصمانه می شود این مسایل اغلب به عنوان بازی ها شناخته می شوند. نظریه ریاضیات بازی یک شاخه از علم اقتصاد است در اکثر بازی ها دو عامل وجود دارند که اعمال آن ها به صورت یک در میان انجام می شوند در اکثر بازی ها دو عامل وجود دارند که اعمال آن ها به صورت یک در میان انجام می شوند و مقادیر سودمندی در حالات انتهای بازی، مساوی و مخالف هم می باشند برای مثال اگر بازیکن یک بازی شطرنج را ببرد ۱+ و اگر ببازد ۱- به آن تخصیص داده می شود.

دلایل مطالعه بازی ها عبارتند از:

- قابلیت هوشمندی انسان ها را به کار می گیرند.
- به دلیل ماهیت انتزاعی بازی ها
- حالت بازی ها را به راحتی می توان نشان داد و عامل های معمولا به مجموعه ای کوچکی از اعمال محدود هستند که نتایج آن ها با قوانین دقیق تعریف شده اند.

انواع بازی ها:

شطرنج : بازی ای است که بر روی یک صفحه انجام می شود و برای دو بازیگر است که شانزده مهره را با توجّه به قوانینی معین حرکت می دهند . هدف مات کردن شاه طرف مقابل می باشد .

بازی چکرز : صفحه ی بازی چکرز برای دو نفر می باشد که هر نفر دارای دوازده مهره می باشد ؛ هدف پریدن و دستگیر نمودن مهره های حریف می باشد .

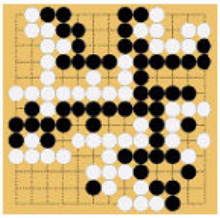
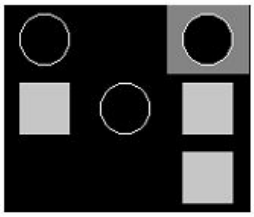
بازی go : بازی ای بر روی صفحه برای دو بازیگر می باشد که شمارنده هایی را روی یک شبکه قرار می دهند ، هدف محاصره کردن و سپس دستگیر کردن شمارنده های حریف می باشد . [۱]

تیک - تاک - تو کور : بازی تیک – تاک – تو دارای دو بازیگر می باشد . برای هر دو بازیگر هدف این است که اوّلین کسی باشند که سه شی ء همانند را در یک ردیف ، ستون یا قطر قرار می دهند . صفحه ی این بازی دارای یک شبکه ی سه در سه می باشد . بنابراین دارای نه خانه می باشد . [۲]

تخته نرد : بازی ای بر روی صفحه است و دو بازیگر دارد . هر بازیگر دارای پانزده مهره می باشد که آن ها را در بیست و چهار خانه ی مثلثی شکل با توجّه به عدد ظاهر شده روی دو تاس حرکت می دهد . [۳]

پل : یک بازی کارتی حقّه ای برای چهار نفر می باشد که این چهار نفر به صورت دو گروه دو نفری با هم بازی می کنند و در هر طرف هر گروه مقابل هم می نشیند . بازی پل دارای دو مرحله می باشد : پیشنهاد و بازی . [۴]

پوکر : بازی ای کارتی می باشد ، محبوب ترین بازی از یک دسته از بازی ها به نام بازی های همچشمی می باشد که در آن بازیگران با کارت هایی کاملاً پنهان یا اندکی پنهان روی یک چیزی شرط بندی می کنند ، سپس چیزی که شرط بندی روی آن انجام شده است به بازیگر یا بازیگران باقی مانده ای که دارای بهترین ترکیب کارت ها هستند جایزه داده می شود . [۱] در زیر تصویر صفحه ی چند بازی ای که هم اکنون معرّفی نمودیم مشاهده می نمایید :

نام بازی	چکرز	go	تیک – تاک – تو کور	تخته نرد
تصویر صفحه				

	قطعی	**شانسی**
اطّلاعات کامل	شطرنج[۲]،بازی چکرز[۳]،go	تخته نرد[۴]
اطّلاعات ناقص	تیک تا تو کور[۵]	پل[۶]،پوکر[۷]

الگوریتم Min-Max:

در این قسمت بازی های دو نفره را با دو بازیکن min و max در نظر می گیریم ابتدا max حرکت می کند و سپس نوبت به حریف یعنی min می رسد به همین گونه ادامه می یابد تا بازی تمام شود در انتهای بازی امتیازات به بازیکن تخصیص داده می شود یک بازی می تواند به صورت نوعی مسئله جست وجو به صورت زیر فرموله سازی شود:

حالت اولیه: شامل موقعیت صفحه بازی و مشخص می شود که نوبت کدامیک از بازیکنان است.

تابع جانشین: لیستی از(Move,State)را برمی گرداند که هر کدام نشانگر یک حرکت قانونی و حالت نتیجه باشد.

آزمون پایانی: این آزمون مشخص می کند که آیا بازی تمام شده است یا خیر.حالاتی که بازی در آن ها به پایانی نامیده می شود.

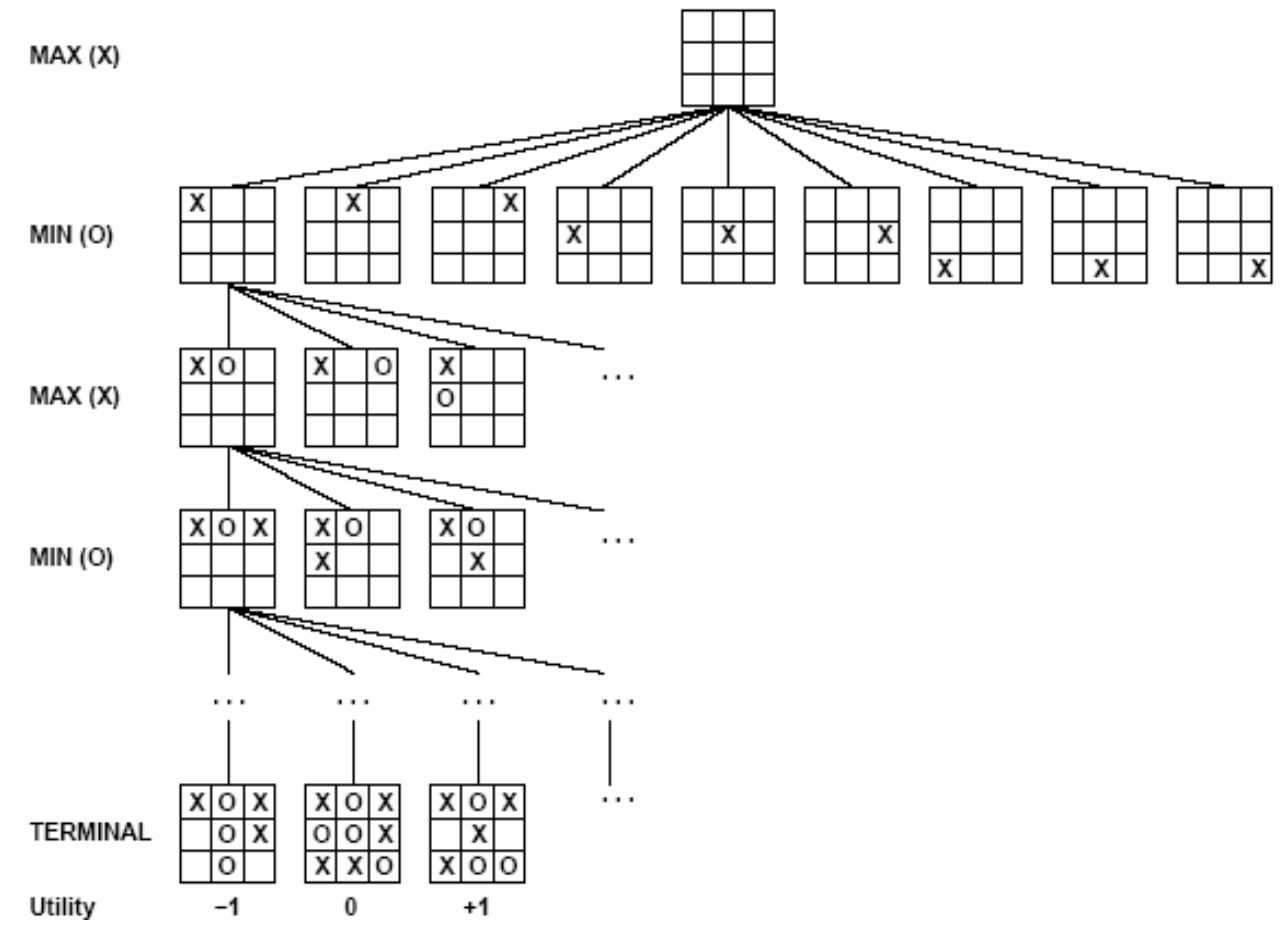
تابع سودمندی[۱۱۷]: این تابع به حالت پایانی یک عدد نسبت می دهد مثلا در شطرنج برد عدد ۱+ و برای باخت ۱- و برای مساوی صفر است در بعضی بازی ها نتایج تنوع وسیعی دارند مثلا در بازی تخته نرد،نتایج از ۱۹۲+ تا ۱۹۲- تغییر می کند این تابع ،تابع هدف یا تابع امتیاز نیز نامیده می شود.

یک نمونه بازی (tic-tac-toe):

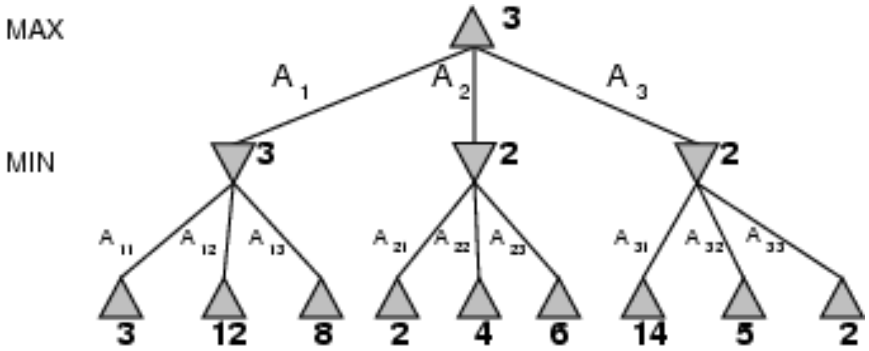
حالت اولیه وحرکات قانونی بازیکنان، درخت بازی را تشکیل می دهند.در شکل زیر قسمتی از درخت بازی tic-tac-toe را نشان می دهد در حالت اولیه بازیکن max اولین حرکت ممکن را می تواند انجام دهد مهره x متعلق به max وo متعلق به min است.بازی به نوبت بین max و min ادامه می یابد اگر بازیکن سه مهره ی خود را در ردیف عمودی،افقی و قطری قرار داده باشد یا اینکه صفحه پر شده باشد بازی تمام می شود عدد نسبت داده شده به هر برگ در درخت بازی میزان سودمندی آن حالت پایانی را از دیدگاه max برمی گرداند مقادیر بزرگ برای max خوب و برای بازیکن min بد است.

---

[117] Utility function



مثال:



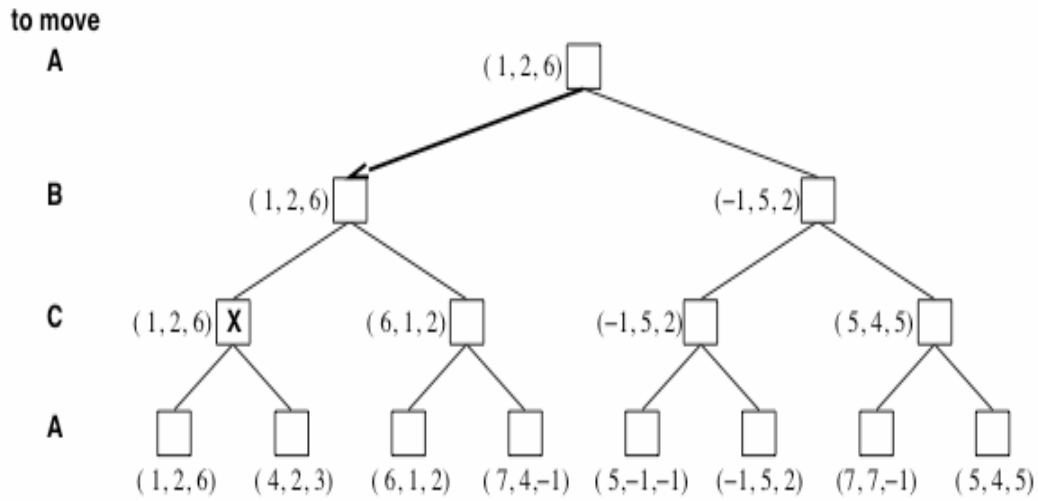
$$\text{MINMAX-VALUE}(n) = \begin{cases} \text{UTILITY}(n) & \text{if } n \text{ is a terminal state} \\ \max_{s \in Successors(n)} \text{MINIMAX-VALUE}(s) & \text{if } n \text{ is a MAX node} \\ \min_{s \in Successors(n)} \text{MINIMAX-VALUE}(s) & \text{if } n \text{ is a MIN node} \end{cases}$$

الگوریتم minimax در واقع ،یک جست و جوی اول عمق در درخت بازی است،اگر ماکزیمم عمق درخت m و تعداد b حرکات قانونی در هر حالت وجود داشته باشد پیچیدگی زمانی $O(b^m)$ خواهد بود.

اگر الگوریتم تمام فرزندان را با هم ایجاد کند،پیچیدگی مکانی O(bm) خواهد بود و اگر در هر لحظه فقط یک فرزند ایجاد کند،پیچیدگی مکانی O(m) خواهد بود الگوریتم minimax در صورت محدود بودن درخت، کامل و بهینه است.

بازی های چند نفره:

اکنون می خواهیم الگوریتم min- max را به بازی های چند نفره گسترش دهیم در ابتدا باید به جای هر مقدار برای هر برگ، یک بردار از مقادیر برای هر برگ در نظر بگیریم برای مثال در یک بازی سه نفره با سه بازیکن AوBوC، بردار $(v_A\ v_B\ v_C)$ به هر نود نسبت داده می شود در یک حالت پایانی، این بردار، سودمندی آن حالت را برای هر بازیکن نشان می دهد،ساده ترین روش پیاده سازی تابع سودمندی به فرمی است که به جای یک مقدار، یک بردار برمی گرداند بازی های چند نفره معمولا با موضوع اتحاد[۱۱۸] بین بازیکنان روبرو هستند اتحاد در ابتدای بازی ایجاد می شود و با پیشرفت بازی شکسته می شود به عنوان مثال فرض کنید احتمال پیروزی AوB در مقابل C ضعیف است بنابراین بهتر است این دو علیه C متحد شده و به جای حمله به یکدیگر، به C حمله کنند در غیر این صورت C به تنهایی آنها را شکست می دهد. البته به تدریج که C ضعیف می شود اتحاد بین AوB شکسته می شود.

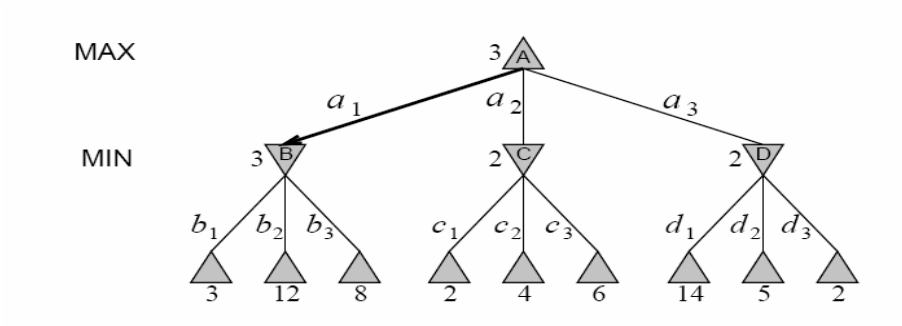


[118] Alliance

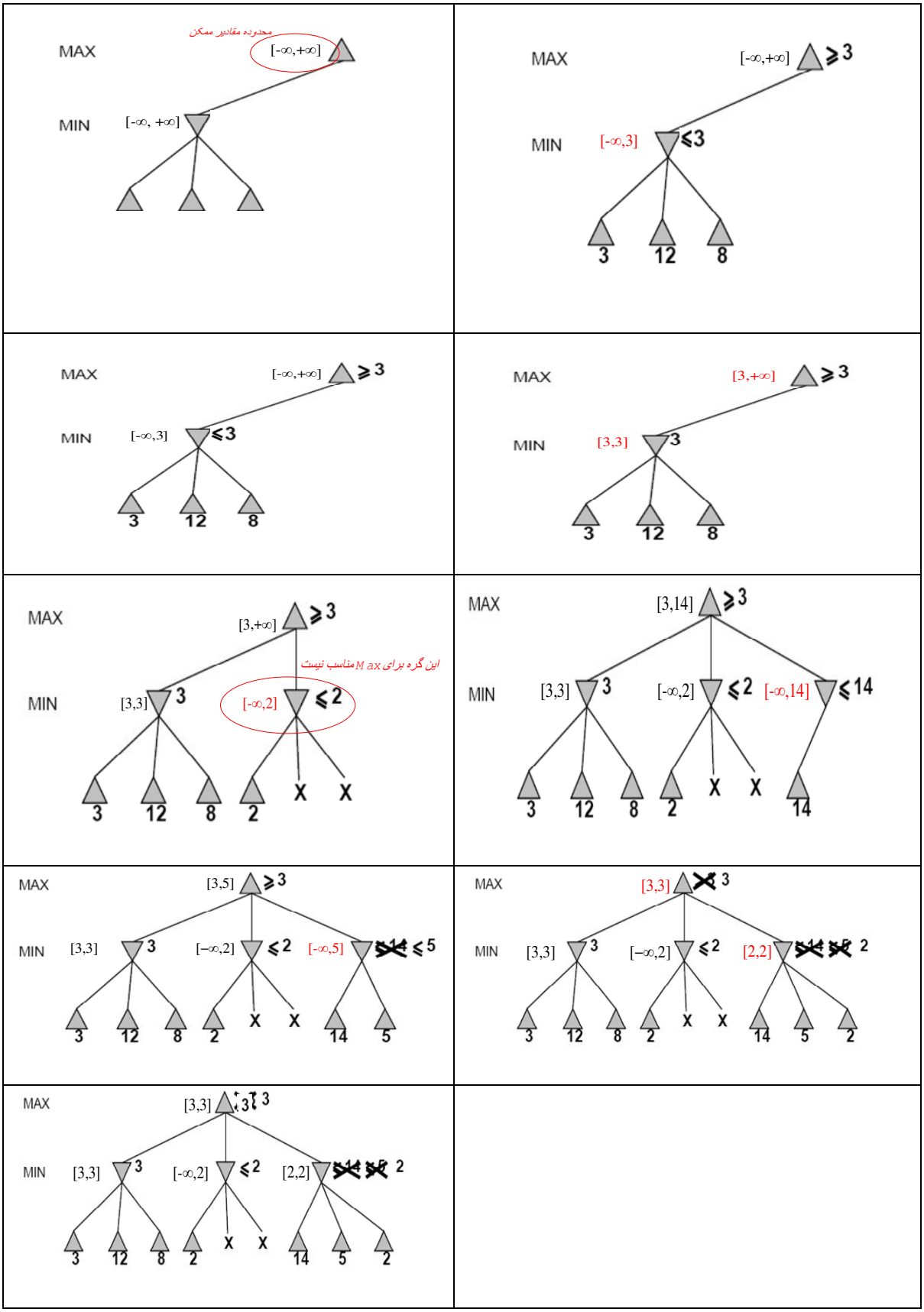
مثال:بازی سه نفره زیر را در نظر بگیرید بازیکن A با استفاده از الگوریتم minimax کدام عمل را انجام خواهد داد؟(IT85)

هرس آلفا-بتا (α-β):

مشکل جستجوی minimax این است که تعداد حالات بازی که این جستجو باید بررسی کند بر حسب تعداد حرکات نمایی است .متأسفانه نمی توان این رشد نمایی را کاهش داد بلکه درخت را می توان به طور کارایی به نصف کاهش داد زیرا تصمیم صحیح بدون بررسی همه گره های درخت جستجو امکان پذیر است.پردازش حذف شاخه ای از درخت جستجو بدون ایجاد آن شاخه، هرس نامیده می شود.هرس استفاده شده در الگوریتم minimax هرس α-β نامیده می شود.الگوریتم minimax استاندارد و هرس α-β حرکت یکسانی را به عنوان نتیجه برمی گردانند زیرا هرس آلفا-بتا شاخه هایی که در نتیجه نهایی دخالت ندارند را هرس می کند. α وβ مقادیر موقتی هستند . α مقداری است که بازیکن max تاکنون دریافت کرده و کمتر از این نخواهد شد و β مقداری است که بازیکن min دریافت کرده و بیشتر از این نخواهد شد.به خاطر داشته باشید که الگوریتم minimax یک الگوریتم عمقی است و در نتیجه ما مجبوریم در هر زمان فقط نودهای یک مسیر از درخت را در نظر بگیریم.تأثیر هرس α-β به ترتیب بررسی فرزندان (مابعدها) بستگی دارد.



درخت بازی A



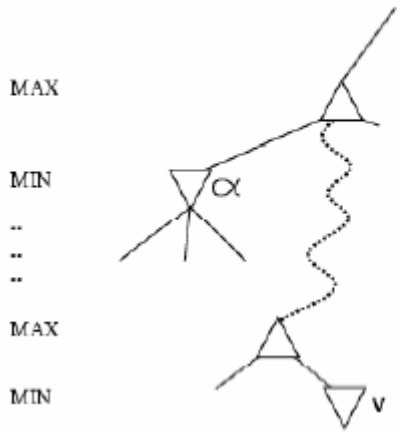
مراحل اتخاذ تصمیم بهینه در مورد درخت بازی A

بنابراین اگر در ابتدا فرزندی که بهتر است مورد بررسی قرار گیرد مطلوب تر است.اگر فرض کنیم چنین کاری می تواند انجام شودیعنی همیشه بهترین فرزند، اول بررسی شود این برنامه به جای بررسی $O(b^m)$ کافی است $O(b^{m/2})$ را بررسی کند. اگر به جای مرتب سازی فرزندان آنها را به طور تصادفی انتخاب کنیم تعداد نودهای مورد بررسی $O(b^{3m/4})$ خواهد بود .در بازی ها نیز نودهای تکراری ممکن است بوجود بیایند بنابراین بهتر است مقدار ارزیابی نودها را در یک جدول درهم ساز ذخیره کنیم تا در صورت دوباره مواجه شدن با این نود مقدار ارزیابی آن محاسبه نشود..جدول درهم سازی موقعیت های مشاهده شده، جدول جابجایی نیز نامیده می شود.این جدول معادل لیست closed در الگوریتم Graph search است.

بنابراین به طور خلاصه در هرس α-β داریم:

- ممکن است یک گره یا زیردرخت به طور کامل هرس شود.
- ممکن است درخت به طور کامل بررسی شود و هرسی صورت نگیرد.
- الگوریتم α-β در یک بازه زمانی ثابت تقریباً دو برابر minimax کارایی دارد.
- هنگامی ارزش α یا β به پدرش اختصاص می یابد که فرزند دیگری نداشته باشددر سایر حالات این مقدار موقتی است.
- موثر بودن هرس بستگی به ترتیب مقادیر فرزندان یک گره بستگی دارد.اگر در Min بودیم ترتیب صعودی بهتر است و اگر درMax بودیم ترتیب نزولی بهتر است.

وجه تسمیه



- آلفا مقدار بهترین انتخاب (یعنی بالاترین مقدار) یافته شده تا کنون در هر نقطه انتخاب در طول مسیر برای *max* می باشد
- اگر v بدتر از آلفا باشد، *max* از آن اجتناب می کند
  ← آن شاخه را هرس می کند
- بتا نیز برای *min* مانند آلفا تعریف می شود

مثال:اگر در درخت های زیر با هرس$\alpha-\beta$ پیمایش شوند کدام گره ها ملاقات نخواهند شد؟

تصمیمات بلادرنگ ناقص:

الگوریتم minmax کل درخت بازی را جستجومی کند درحالیکه هرسα-β قسمت وسیعی از درخت را هرس می کند با این وجود الگوریتم هرسα-β باز هم باید تارسیدن به حالت پایانی ادامه داده ودر نتیجه قسمت وسیعی ازدرخت کامل را بررسی کند.شانون پیشنهاد کرد عمق درخت بازی محدود شودوبه جای تابع سودمندی ازتابع ارزیابی[119] اکتشافی برای نودها استفاده شود.بعبارت دیگر در الگوریتم minmax وهرسα-β باید تغییرات زیرصورت بگیرد:

- تابع سودمندی با تابع ارزیابی اکتشافی جایگزین شود.این تابع ارزیابی اکتشافی تخمینی ازسودمندی یک واقعیت را بر می گرداند.
- آزمون پایانی را با آزمون برش[120] جایگزین کنیم.این آزمون مشخص می کندکه چه موقع تابع ارزیابی فراخوانی شود.

توابع ارزیابی:

توابع ارزیابی، تخمینی از سودمندی مورد انتظار بازی از موقعیت داده شده را بر می گردانند. کارایی یک برنامه بازی به کیفیت این تابع ارزیابی بستگی دارد.تابع ارزیابی باید به سه دلیل مناسب باشد:

- مقادیری که تابع ارزیابی به حالات پایانی نسبت می دهدبا مقادیری که تابع سودمندی به آنها نسبت می دهد متناسب باشد
- محاسبات تابع ارزیابی نباید زمان زیادی مصرف کند.
- در نودهای غیر پایانی این تابع باید به درستی شانس های واقعی برد را منعکس کند

بسیاری از توابع ارزیابی برای هر خصوصیت در یک حالت ترکیبات عددی متفاوتی را در نظر می گیرند وسپس آنها را با هم جمع نموده وبعنوان یک مقدار برمی گردانند.برای مثال در بازی شطرنج سرباز

---

[119] Evaluation function
[120] Cut-off test

ارزش۱,اسب یا فیل ۲وقلعه ۵ است.این مقادیر جمع شده ومقدار ارزیابی یک موقعیت را مشخص مـی کنـد. از نظر ریاضیاتی این نوع تابع ، تابع خطی وزن دار نامیده می شود.

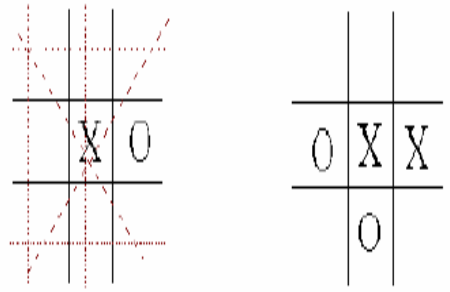
$$Eval(s)=w_1f_1(s)+w_2f_2(s)+.....+w_nf_n(s)=\sum f_i(s)$$

که در این تابع، f_i ها خصوصیات موجود در یک موقعیت وw_iوزن ها را نشان می دهد.مثلا"در مورد شطرنج f_i می تواند تعدادمهره های روی صفحه و w_i ها ارزش هرکدام باشد.

مثال: تابع ارزیابی برای بازی دوز

$e(n)$ =
If n is win for Max, $+\infty$, If n is win for Min, $-\infty$

Else
(Possible number of rows, columns and diagonals available to Max)
–
(possible number of rows, columns and diagonals available to Min)



$e(n) = 6 - 4 = 2$ $e(n) = 4 - 3 = 1$

قطع جست وجو:

راحت ترین راه برای کنترل میزان جستجو، قرار دادن محدودیت روی عمق اسـت.میـزان عمـق مـوردنظر بـا استفاده از محدودیت های زمانی بازی مشخص می شود. بنابراین تست قطع برای تمام گره ها در زیر عمق d موفق خواهد بود.عمق طوری انتخاب می شودکه میزان زمان استفاده شده ازآنچه قوانین بازی اجازه می دهـد تجاوز نکند.زمانی که وقت بازی تمام می شود برنامه حرکت انتخابی توسط عمیق ترین جستجوی کامل شـده را بر می گرداند.به دلیل تخمینی بودن توابع ارزیابی، این رهیافت، نتایج ناخوشایندی را به دنبال خواهد داشت. تابع ارزیابی باید فقط برای حالاتی بکار برده شوند که ساکن هستند.

حالت ساکن[۱۲۱] :

فرض کنید که حالت x در عمق برش d قرار دارد و Eval(x)=n باشد اگر عمق برش درخت را افزایش دهیم آنگاه x دیگر در عمق برش نیست ودارای مقدارminimax است .در این صـورت اگـر مقـدار minimax وm تفاوت چشمگیری نداشته باشندحالت x ساکن است.موقعیت های غیـر سـاکن بایـد بـسط داده شـوند تـا بـه موقعیت ساکن برسیم.این جستجوی اضافی جستجوی ساکن نامیده می شوند.

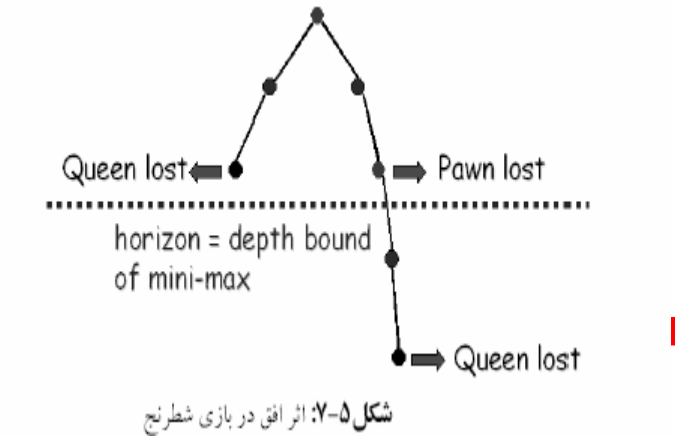
[121] Quiescent

علاوه بر روش هرس α-β روش های زیر نیز برای بهبود الگوریتم minmax بکار برده می شود:

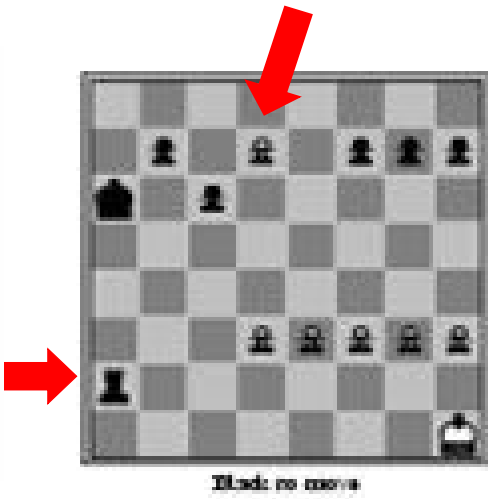
۱)انتظار برای سکون ۲)جستجوی ثانویه ۳)هرس رو به جلو ۴)عمیق شونده تکراری

اثرافق[۱۲۲]:

حالتی است که در آن انجام حرکتی را که برای حریف بسیار سودمند است مدتی به تعویق می اندازد.در حالیکه حریف بالاخره آن حرکت را انجام خواهد داد.به عنوان مثال در یک بازی شطرنج یک نفر با کیش دادن های متوالی خوردن مهره خود را به تعویق می اندازد.



شکل۵-۷: اثر افق در بازی شطرنج

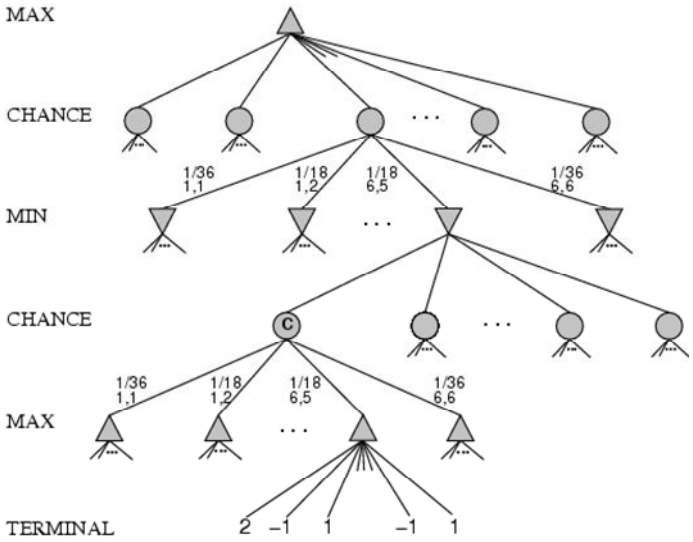


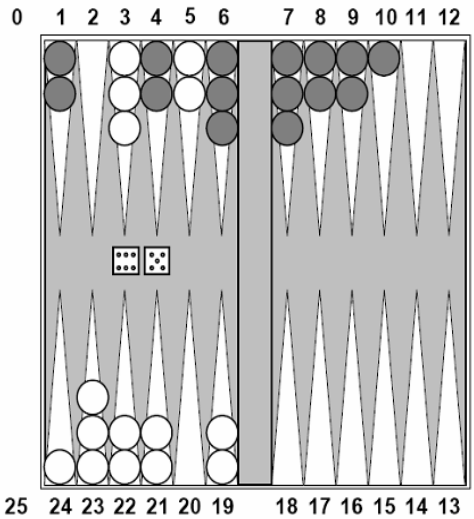
بازی هایی که دارای عامل شانس هستند:

درزندگی واقعی بر خلاف شطرنج حوادث غیر قابل پیش بینی زیادی وجود دارند که مارا در شرایط غافلگیرانه ای قرار می دهند.بازی های زیادی غیر قابل پیش بینی بودن را توسط یک عنصر تصادفی مانند پرتاپ تاس یا سکه نشان می دهند.تخته نرد یک بازی است که شانس ومهارت را ترکیب می کند.تاس ها در ابتدای بازی توسط بازیکنی که نوبتش است ریخته می شود تا مجموعه حرکاتی که قابل انجام هستند مشخص شوند.درخت بازی تخته نرد علاوه بر گره هایmin وmax باید شامل گره های شانس نیز باشد.

---

[122] Horizon Effect





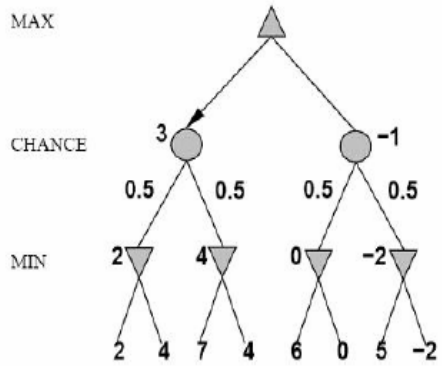
دوایر نشان دهنده ی گره های شانس هستند .شاخه هایی که از هر گره شانس خارج شده اند, مقادیر مختلف تاس ها را مشخص می کنند.هر کدام با شانسی که دارند برچسب می خورند.برای دو تاس ۳۶ حالت وجود دارد که در آن ۲۱ حالت متفاوت قابل استخراج است مثلاً (۱،۳) و(۳،۱) را یک حالت در نظر می گیریم واحتمال آن ۲/۳۶ است.

مرحله بعدی این است که چگونه بفهمیم تصمیم گیری صحیحی داشته باشیم؟اگرچه در اینجا مقدار minmaxقطعی وجود ندارد, در عوض می توانیم میانگین یا مقدار مورد انتظار[۱۲۳] رامحاسبه کنیم. این مقدار مورد انتظار، کلیه ترکیبات تاس ها را در نظر می گیرد.بنابراین مقدارminmaxدر بازی های قطعی رابه مقدار مورد انتظاردر بازی های شامل عنصر شانس عمومیت می دهیم..نودهایmaxوmin وپایانی مانند قبل محاسبه می شود ولی مقدار عنصر شانس به صورت زیر محاسبه می شود:

روش مینی‌ماکس مورد انتظار (Expected MiniMax)

$$v = \sum_{chance\ nodes} P(n) \times \text{Minimax}(n)$$

$$3 = 0.5 \times 4 + 0.5 \times 2$$



شکل۵-۸: درخت بازی برای روش مینی ماکس مورد انتظار

پیچیدگی زمانی

$O(b^m n^m)$,

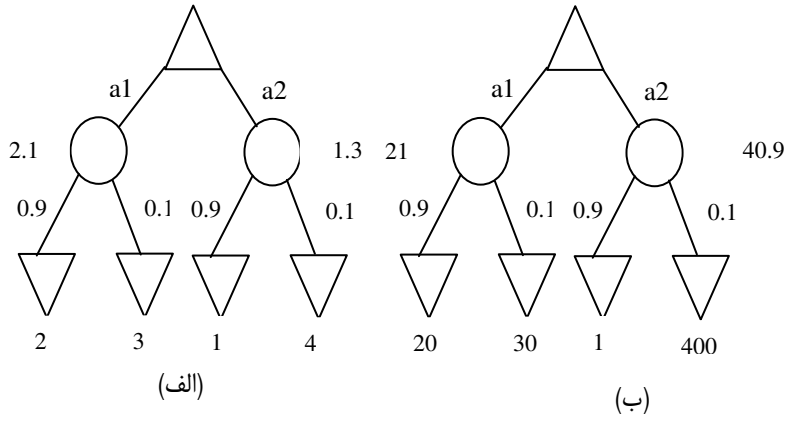
n: تعداد رویدادهای متفاوت

[۱۲۳] Expected

پیچیدگی زمـانی الگـوریتم Expect_ minmax(x) $O(b^m n^m)$ مـی باشـدکه در آن، n تعـداد پرتـاب هـای موجود,mحداکثر عمق درخت و b فاکتور انشعاب می باشد.

ارزیابی موقعیت در بازیها با عنصر شانس:

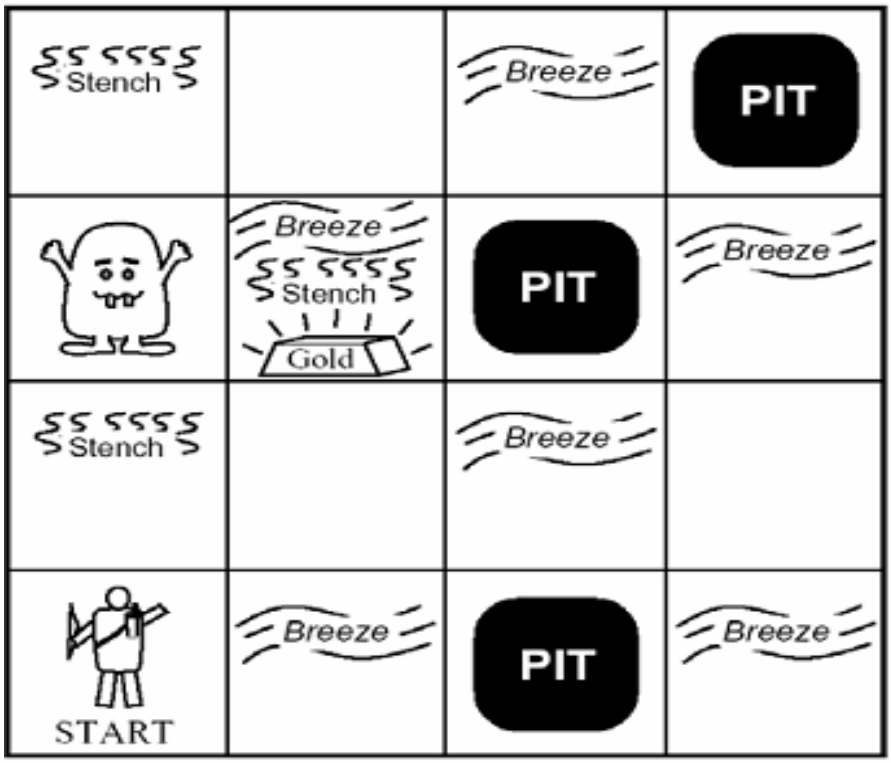
در الگوریتم minmax هرانتقال با حفظ مرتبه ارزش برگ ها، تاثیری در انتخاب حرکت ندارد. یعنی می تـوان مقادیر ۱،۲،۳،۴ یا مقادیر ۱،۲۰،۳۰،۴۰۰ را استفاده نمود و تصمیم مشابهی گرفت. اما با وجود گره های شانس، این خاصیت حفظ نمی شود.در شکل الف با مقادیر برگی ۱،۲،۳،۴ حرکت a1 بهترین است در حالیکه در شکل ب با مقادیر برگی ۱،۲۰ ۳۰،۴۰۰ حرکت a2 بهترین است بنابراین در الگوریتم minmaxاستاندارد،انتقالبا حفظ ارزش برگ ها تاثیری در انتخاب حرکت ندارد ولی اگرعنصر شانس چاشنی کار شـود انتقـال بـا حفـظ مرتبـه کارساز نیست.



فصل هفتم: عامل های منطقی

آنچه در این فصل خواهید آموخت:

- **عامل های مبتنی بر دانش**
- **محیط Wumpus**
- **منطق-مدل ها و استلزام ها**
- **منطق گزاره ای**
- **هم ارزی،اعتبار و صدق پذیری**
- **الگوهای استدلال در منطق گزاره ای**
- **الگوریتم Resuoltion**
- **الگوریتم زنجیره روبه جلو و زنجیره روبه عقب**



عامل مبتنی بر دانش:

جزء اصلی عامل مبتنی بر دانش پایگاه دانش[۱۲۴](KB)است . یک پایگاه دانش مجموعه ای از جملات در یک زبان رسمی است. هرجمله در یک زبان به نام زبان بازنمایی دانش[۱۲۵](مثلا منطق) بیان می شود و ادعایی را در مورد دنیا باز نمایی می کند. باید راهی برای اضافه کردن جملات به پایگاه دانش(TELL) وراهی برای پاسخ به سؤال پرسیده شده از پایگاه دانش (ASK) وجود داشته باشد که هر دوی این کار ممکن است شامل عمل استنتاج نیز باشد .استنتاج یعنی جملات جدیدی از جملات قدیمی بدست می آیند . نتایج حاصل،باید از KB پیروی کنند. پیروی[۱۲۶] یعنی انجام فرایند استنتاج تحت مقررات خاص. هر وقت برنامه عامل فراخوانی می شود ۳ کار انجام می شود:

- به پایگاه دانش می گوید چه چیزی را از محیط توسط سنسورهایش دریافت کرده است.

Tell(KB,Make-Percept-Sentence(percept,t))

- از پایگاه دانش سؤال می کند چه عملی را باید انجام دهد.

action ← Ask(KB,Make-Action-Query(t))

- عامل عمل انتخاب شده را با Tell ذخیره کرده وآن را اجرا می کند.

Tell(KB,Make-Action-Sentence(action,t))

```
function KB-AGENT(percept) returns an action
   static: KB, a knowledge base
           t, a counter, initially 0, indicating time

   TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
   action ← ASK(KB, MAKE-ACTION-QUERY(t))
   TELL(KB, MAKE-ACTION-SENTENCE(action, t))
   t ← t + 1
   return action
```

۷-۱ یک عامل عمومی مبتنی بر دانش

[124] Knowledge Base
[125] Knowledge Representation
[126] follow

عامل مبتنی بر دانش خیلی شبیه به عاملهایی با حالت درونی می باشد . عاملها دردو سطح متفاوت تعریف می شوند:

- **سطح دانش:** در این سطح مشخص می شود عامل چه چیزی می داند و چه اهدافی دارد بدون توجه به جزئیات پیاده سازی
- **سطح پیاده سازی:** در این سطح، ساختمان داده ی اطلاعات پایگاه دانش و الگوریتم هایی که بر روی آن کار می کنند نشان داده می شود.

دنیای وامپوز:

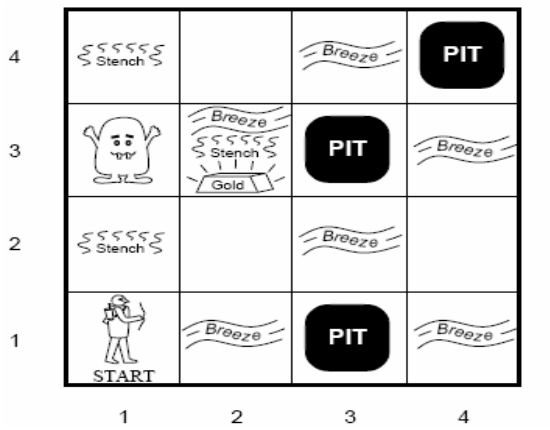
قبل از اینکه به مفاهیم و الگوریتم های این فصل بپردازیم بهتر است مثالی را معرفی کرده و سپس مباحث خود را روی آن بیان کنیم.همان طور که در فصل ۲ اشاره کردیم برای هر عامل باید مشخصات محیط کار(PEAS) مشخص شود.مشخصات محیط کار دنیای وامپوز به صورت زیر است:

معیار کارایی : ۱۰۰۰+ : انتخاب طلا، ۱۰۰۰- :افتادن در گودال یا خورده شدن ، ۱- : هر مرحله ، ۱۰- : استفاده از تیر .

محیط: بوی تعفن در مربعهای همجوار wumpus ، نسیم در مربعهای همجوار گودال ، درخشش در مربع حاوی طلا، کشته شدن wumpus با شلیک در صورت مقابله ، تیر فقط مستقیم عمل می کند، برداشتن و انداختن طلا.

حسگرها: بوی تعفن، نسیم، تابش، ضربه، جیغ زدن

محرکها: گردش به چپ، گردش به راست، جلو رفتن، برداشتن، انداختن، شلیک کردن



ویژگیهای محیطی دنیای وامپوز:

قابل مشاهده کامل: خیر, فقط ادراک محلی امکان پذیر است.

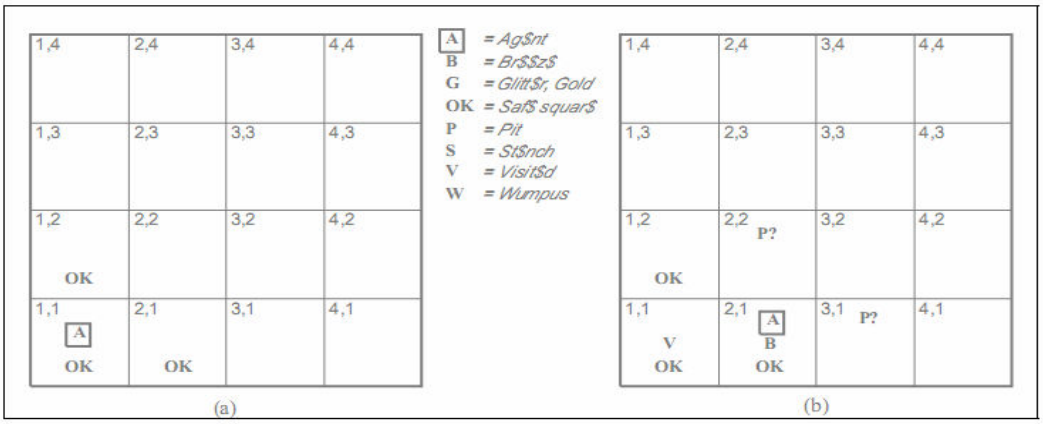
قطعی: بله، وضعیت بعدی دقیقا مشخص است.

رویدادی: خیر، ترتیبی از فعالیتهای مرتبط باهم است.

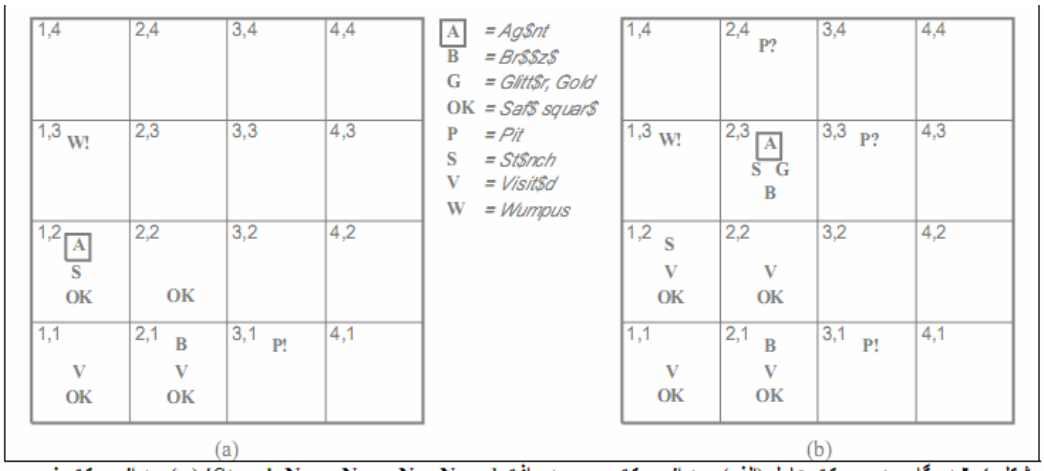
ایستا: بله, wumpus و گودالها حرکت ندارند

تک عامله: بلهwumpus در اصل یک خصوصیت طبیعی محیط است و یک عامل مستقل محسوب نمی شود. **گسسته:** بله

فرایند اکتشاف در دنیای وامپوز:



شکل ۳-۶ اولین گام برداشته شده توسط عامل در دنیاي wumpus (الف) وضعیت اولیه، بعد از دریافت [None,None,None,None,None] (ب) بعد از یك حرکت و دریافت [None,Breeze,None,None,None]



شکل ٤-٦ دو گام بعدي حرکت عامل (الف) بعد از حرکت سوم و دریافت [Stench,None,None,NonNone] (ب) بعد از حرکت پنجم و دریافت [Stench,Breeze,Glitter,None,None]

منطق[۱۲۷]:

منطق یک زبان رسمی برای بازنمایی دانش است بطوریکه می توان از آن نتیجه گیری هایی نمود. قبلاً اشاره کردیم که پایگاه دانش شامل مجموعه ای از جملات است . این جملات بر اساس نحو[۱۲۸] زبان بازنمایی دانش، نشان داده می شوند. این زبان همه جملاتی را که خوش فرم[۱۲۹] هستند را نشان می دهد. یک منطق باید معانی[۱۳۰] زبان را نیز تعریف کند. آنچه معانی باید انجام دهند مشخص کردن معنای هر جمله است. معانی زبان، درستی یک جمله را در هر دنیای ممکن مشخص می کنند.در منطق گزاره ای و منطق مرتبه اول هر جمله در هر دنیای ممکن درست یا نادرست است و حالت مابینی نداریم ولی در منطق فازی یک حالت مابین وجود دارد. زبانهای محدودیت ها، منطق ها هستند وحل محدودیت ها فرمی از استنتاج منطقی است . به جای دنیای ممکن از اصطلاح مدل[۱۳۱] استفاده می کنیم . مدل تعریف دیگری نیز دارد. وقتی می گوییمm مدل جمله αاست معنایش این است که جمله α در مدلm درست است.مدلها تجرید های ریاضی هستند وهرکدام درستی یا نادرستی جمله مرتبط را مشخص می کنند.

مثال: $\alpha : x+y=4$ $m1: x=0, y=4$ $m2: x=4, y=0$

$m3: x=3, y=1$ $m4: x=1, y=3$ $m5: x=2, y=2$

در این مثال مدلها ترکیبات ممکن اعدادی است که به x,y نسبت داده می شوند پس مدل ۲ مفهوم دارد:

- هریک از ترکیبات ممکن بر اساس تعداد گزاره های موجود، یک مدل نامیده می شود .
- به دنیایی که جمله αدر آن درست باشد مدل جمله ی α گفته می شود.

	p	q
M1	F	F
M2	F	T
M3	T	F
M4	T	T

نکته: برای n تا گزاره 2^n تا مدل خواهیم داشت.

---

[127] Logic
[128] Syntax
[129] well form
[130] Semantic
[131] Model

استلزام[۱۳۲]: گاهی اوقات ممکن است بخواهیم جملات جدید وتازه ای را که الزاماً صحیح هستند تولید و یا استفاده کنیم در حالیکه جملات قدیمی نیز صحیح هستند. چنین ارتباطی بین جملات استلزام نامیده می شود.

α ⊨β: α مستلزم β است اگروفقط اگر در هر مدلی که αدرست باشد βنیز درست باشد.

(مدلهایβ ⊆ مدلهایα)

a ⊨ b:

- جمله a استلزام جمله b است.
- جمله a جمله b را ایجاد میکنداگر و فقط اگر، در هر مدلی که a درست است، b نیز درست است
- اگر a درست باشد، b نیز درست است
- درستی b در درستی a نهفته است

پایگاه دانش می تواند به عنوان یک جمله در نظر گرفته شود ومی گوییم یک جمله مستلزم پایگاه دانش است اگر و فقط اگر αدر تمام دنیاهایی که در آن KB درست است، درست باشد.

KB ⊨ α : در هر مدلی که KB درست است αنیز درست است(مدلهایα⊆ مدلهای KB)

استلزام (entailment)

- استلزام بدین معناست که چیزی از چیز دیگری پیروی می کند:

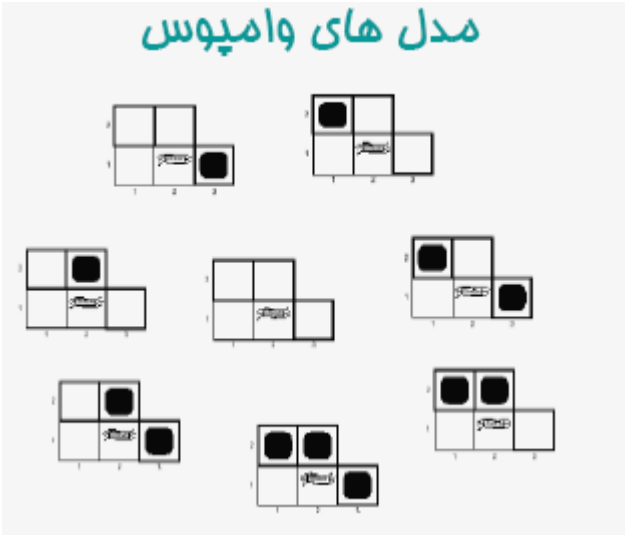
$KB \models \alpha$

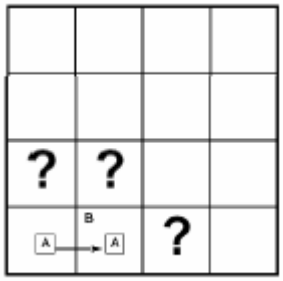
- پایگاه دانش KB مستلزم جمله α است، اگر و فقط اگر
- α در تمام دنیاهایی که در آن KB درست است، درست باشد.
- مثال: $x + y = 4$ مستلزم $4 = x + y$ می باشد.
- استلزام رابطه ایست که بین ساختار جملات (syntax) و بر مبنای معنای جملات تعریف می شود.

---

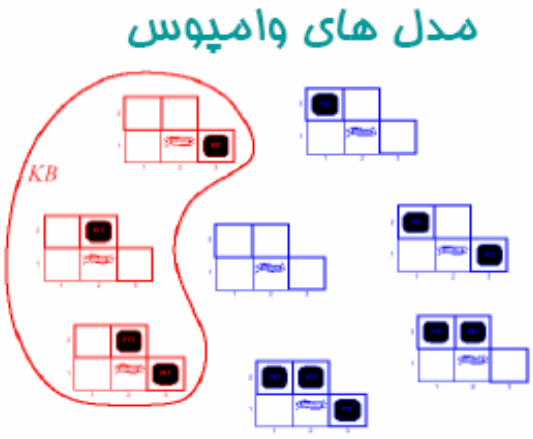
[132] Entaiment

برای درک بهتر مفهوم استلزام به مثالی از دنیای وامپوز توجه فرمائید:



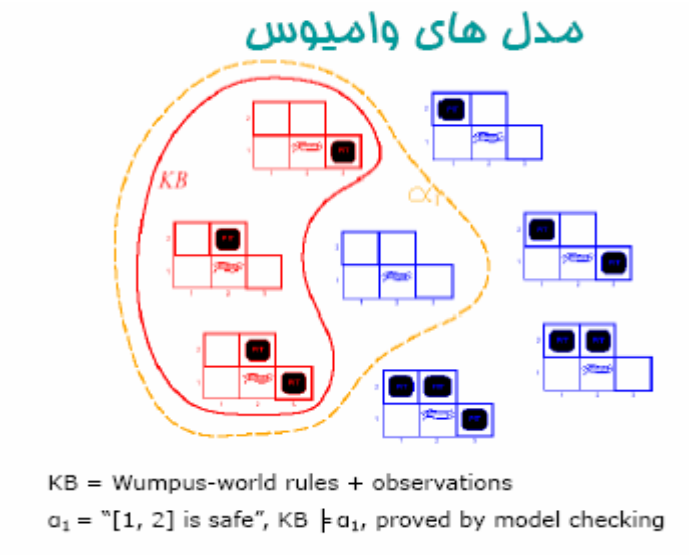


عامل در خانه[1,1] چیزی دریافت نکرده است و در خانه [2,1] نسیم دریافت کرده است.این مشاهدات با دانش عامل در مورد قوانین بازی وامپوز ترکیب شده و به KB اضافه می شوند . عامل می داند که در یکی از خانه های [3,1],[2,2],[1,2] چاله وجود دارد .هرکدام از این ۳ خانه ممکن است چاله داشته باشد یا نداشته باشد بنابراین 2^3یعنی ۸ مدل ممکن وجود دارد . روش بررسی(شمارش) مدل[۱۳۳] تمام مدلهای ممکن را شمارش می کند وبررسی می کند که آیا در تمام مدلهایی که KB درست است αنیز درست می باشد؟

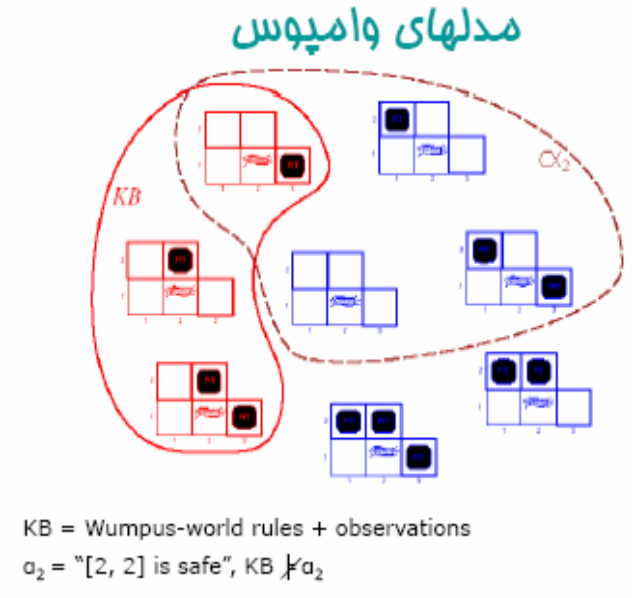


KB = Wumpus-world rules + observations

[133] Model checking



KB = Wumpus-world rules + observations
α_1 = "[1, 2] is safe", KB $\models \alpha_1$, proved by model checking



KB = Wumpus-world rules + observations
α_2 = "[2, 2] is safe", KB $\not\models \alpha_2$

برای درک هرچه بهتر و تمایز مفهوم استنتاج و استلزام می توان مجموعه همه جملات KB را بعنوان انبار کاه و α را به عنوان سوزن در نظر گرفت. استلزام مثل سوزن در این انبار کاه و استنتاج مانند یافتن سوزن در این انبار کاه است. بعبارت دیگر الگوریتم استنتاج i بتواند α را از پایگاه دانش بدست آورد(مشتق کند) می نویسیم: $KB \vdash_i \alpha$

صحت[۱۳۴]: اگر یک الگوریتم استنتاج فقط جملات استلزام یافته را بدست آورد(مشتق کند) الگوریتم استنتاج صحیح گفته می شود. صحت یک خصوصیت مطلوب است یک الگوریتم استنتاج غیر صحیح از کشف سوزن هایی که وجود ندارند خبر می دهد لذا روش شمارش مدل یک رویه ی استنتاج صحیح است. شمارش مدل هنگامی کار می کند که فضای مدل متناهی باشد ودر ریاضیات که معنای مدل نامتناهی است کار نمی کند.

کامل بودن[۱۳۵] : یک الگوریتم استنتاج کامل است اگر بتواند همه جملاتی که استلزام می شود را بدست آورد. برای انبار کاه وقتی که متناهی است به نظر می رسد با بررسی سیستماتیک می توان تصمیم گرفت که آیا سوزن در انبار کاه وجود دارد یا خیر. کامل بودن نیز یک خصوصیت

---

[134] Soundness
[135] completeness

مطلوب است ولی بیشتر پایگاههـای دانـش انبـار نتـایج نـا متنـاهی دارنـد. در منطـق مرتبـه اول یـک رویـه استنتاج کامل و صحیح وجود دارد .

- $KB \vdash_i \alpha$ = جمله α بوسیله رویه i از KB قابل اشتقاق می باشد.
- نتایج KB مانند یک انبار کاه می باشد، و α مانند یک سوزن
- استلزام = سوزن در انبار کاه؛ استنتاج = یافتن سوزن
- صحت (soundness): رویه i صحیح است اگر

$$KB \vdash_i \alpha \Rightarrow KB \models \alpha$$

- کامل بودن (completeness): رویه استنتاج i کامل است اگر

$$KB \models \alpha \Rightarrow KB \vdash_i \alpha$$

- مثال: در منطق مرتبه اول (First Order Logic) یک رویه استنتاج کامل و صحیح وجود دارد.

منطق گزاره ای:

منطق، مجموعه قواعدی است که به کمک آن می توان معتبر بودن یک استدلال را تشخیص داد.

گـزاره[۱۳۶]:جملـه خبـری اسـت کـه میتوانـد ارزش درسـت یـا نادرسـت داشـته باشـد.گـزاره هـارا بـا نمـاد p,q,r,... نـشان مـی دهـیم. گـزاره هـا دو نـوع هـستند : سـاده و مرکـب. گـزاره سـاده قابـل تجزیـه بـه گزاره های کوچکتر نیست ولی گزاره مرکـب از چنـد گـزاره سـاده بـه کمـک روابـط ترکیبـی تـشکیل شـده است.

روابط ترکیبی:

- **نقیض(not,~):** عملگری است که ارزش گزاره را نقض می کند.
- **ترکیـب عطفـی(and,^):** ایـن عملگـر دو عملونـدی اسـت وفقـط زمـانی True اسـت که هر دو عملوند آن Trueباشند.
- **ترکیـب فـصل(or,∨):**ایـن عملگـر دو عملونـدی اسـت وفقـط زمـانی True اسـت کـه حداقل یکی از گزاره ها True باشد.

---

[136] Propositional

- **ترکیب شرطی(⇒):** زمانی False است که مقدم آن True و تالی آن False باشد.
- **ترکیب دو شرطی(⇔) :** برروی دو گزاره عمل می کند و ارزش آن هنگامی که هر دو گزاره هم ارزش باشند درست است.

جدول صحت :(معنای منطق گزاره ها)

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
false	false	true	false	false	true	true
false	true	true	false	true	true	false
true	false	false	false	true	false	false
true	true	false	true	true	true	true

گرامر منطق گزاره ها(نحو منطق گزاره ها)

sentence→Atomic sentence| Complex sentence

Atomic sentence→True|false|Symbol

Symbol→p|q|r|…

Complex sentence→~sentence |(sentence ∧ sentence)

|(sentence ∨ sentence)

| (sentence ⇒ sentence)

| (sentence ⇔ sentence)

روش انتفـاع مقـدم: هرگـاه در یـک گـزاره شـرطی طـرف اول نادرسـت باشـد بـدون توجـه بـه طـرف دوم، ارزش کل گزاره درست خواهد بود.

مثال : $\phi \subseteq A$ ، $x \in \phi \to x \in A$ مثال: اگر 5 زوج باشد ← سام باهوش است.

روش صحت تالی: هرگـاه در یـک گـزاره شـرطی طـرف اول(تـالی) صـحیح باشـد طـرف دوم ترکیـب شرطی صحیح خواهد بود.

- یـک گـزاره مرکـب را همیـشه درسـت[۱۳۷] مـی گـوییم هرگـاه مـستقل از ارزش گـزاره هـای تشکیل دهنده آن، همیشه درست باشد.

مثال: $p \to p$ یا $p \vee \sim p$

p	$\sim p$	$p \vee \sim p$
F	T	T
T	F	F

- یک گزاره مرکـب را همیـشه نادرسـت مـی گـوییم هرگـاه مـستقل از ارزش گـزاره هـای تـشکیل دهنده آن، همیشه نادرست باشد.

مثال: $p \wedge \sim p$

p	$\sim p$	$p \wedge \sim p$
F	T	F
T	F	F

- دو گزاره مرکب را هـم ارز مـی گـوییم هرگـاه ارزش آنهـا در کلیـه حالـت هـا(مـدل هـا) یکـسان باشد. $p \equiv Q$

[137] tatulogy

مثال) $\sim(p \wedge q) \equiv \sim p \vee \sim q$

p	q	$p \wedge q$	$\sim(p \wedge q)$
F	F	F	T
F	T	F	T
T	F	F	T
T	T	T	F

p	q	$\sim p$	$\sim q$	$\sim p \vee \sim q$
F	F	T	T	T
F	T	T	F	T
T	F	F	T	T
T	T	F	F	F

هم ارزی های مهم:

$$\left.\begin{array}{l} 1)\, p \vee q \equiv q \vee p \\ 2)\, p \wedge q \equiv q \wedge p \end{array}\right\} \text{جابجایی}$$

$$\left.\begin{array}{l} 3)\, p \vee (q \vee r) \equiv (p \vee q) \vee r \\ 4)\, p \wedge (q \wedge r) \equiv (p \wedge q) \wedge r \end{array}\right\} \text{شرکت پذیری}$$

$$\left.\begin{array}{l} 5)\, p \vee (q \wedge r) \equiv (p \vee q) \vee (p \vee r) \\ 6)\, p \wedge (q \vee r) \equiv (p \wedge q) \wedge (p \wedge r) \end{array}\right\} \text{توزیع پذیری:}$$

$$\left.\begin{array}{l} 7)\, p \wedge (p \vee q) \equiv p \\ 8)\, p \vee (p \wedge q) \equiv p \end{array}\right\} \text{جذب :}$$

$$\left.\begin{array}{l} 9)\, p \wedge (\sim p \vee q) \equiv p \wedge q \\ 10)\, p \vee (\sim p \wedge q) \equiv p \vee q \end{array}\right\} \text{شبه جذب}$$

$$\left.\begin{array}{l} 11)\, \sim(p \vee q) \equiv \sim p \wedge \sim q \\ 12)\, \sim(p \wedge q) \equiv \sim p \vee \sim q \end{array}\right\} \text{دمورگان}$$

$13)\, p \vee F \equiv p \quad 14)\, p \vee T \equiv T \quad 15)\, p \wedge F \equiv F \quad 16)\, p \wedge T \equiv p \quad 17)\, p \vee \sim p \equiv T$

$18)\, p \wedge \sim p \equiv F \quad 19)\, p \to q \equiv \sim p \vee q \quad 20)\, p \to q \equiv \sim q \to \sim p \quad 21)\, \sim(p \to q) \equiv p \wedge \sim q$

$22)\, p \leftrightarrow q \equiv (p \to q) \wedge (q \to p) \quad 23)\, \sim(p \leftrightarrow q) \equiv \sim p \leftrightarrow q \equiv p \leftrightarrow \sim q$

مثال) درستی هم ارزی زیر را بررسی کنید.

$[(p \to q) \wedge r] \to p \equiv r \to p$

جواب:

$[(\sim p \vee q) \wedge r] \to p \equiv \sim[(\sim p \vee q) \wedge r] \vee p \equiv [(p \wedge \sim q) \vee \sim r] \vee p \equiv [(p \wedge \sim q \vee p] \vee \sim r$
$\equiv p \vee \sim r \equiv \sim r \vee p \equiv r \to p$

گزاره نما: عبارتی شامل یک یا چند متغییر که متغییرها از مجموعه خاصی عضو می پذیرند را گزاره نما می گویند(به اعضای این مجموعه ، مجموعه ی جهانی می گوییم) .گزاره نما را با $p_x, q_x, \ldots$ نمایش می دهیم.

تک متغییره : $p_x : x+4>7 \quad x \in R$

دو متغییره : $p_{x,y} : x^2+y^2=16 \quad x, y \in R$

سور ها: علائمی هستند که یک گزاره نما را به یک گزاره تبدیل می کنند.

الف) $\forall$**(سور عمومی):** به ازای همه ی مقادر باید درست باشد.

ب) $\exists$**(سور و جودی):** به ازای بعضی از مقادیر باید درست باشد.

ج) $\exists$**(سور صفر):**به ازای هیچ مقدار باید درست باشد.

د) $\exists!$**(سور انحصاری):** به ازای یک مقدار باید درست باشد.

$\forall x \in R,\ x^2>0 \qquad \forall x \in R,\ x^2 \geq 0 \qquad \forall z \in \not\subset,\ z^2 \geq 0 \qquad \forall x \in z,\ x^2-9=0 \Rightarrow x=\pm 3$

$\exists!\ x \in z,\ x-1=2 \Rightarrow x=3$

ترکیب سورها:

$\forall x, \forall y\ p(x,y) \quad \forall x, \exists y\ p(x,y) \quad \exists x, \forall y\ p(x,y) \quad \exists x, \exists y\ p(x,y)$

نقیض سورها: $(\exists x, px) \equiv \forall x, \sim px \qquad (\forall x, px) \equiv \exists x, \sim px$

مثال)

$(\forall x \in R,\ \exists y \in N : xy>4) \equiv \exists x \in R,\ \forall y \in N : xy \leq 4$

$(\forall x, x \in A, \rightarrow x \in) \equiv \exists x, x \in A \wedge x \notin B$

استنتاج[۱۳۸] : فرض کنید که گزاره های $p_1 \wedge p_2 \wedge p_3 \ldots p_n$ گزاره های درست باشند. چناچه از درستی این گزا ره ها، درستی گزاره ی q نتیجه شود در این صورت q را یک استلزام منطقی از گزاره های $p_1 \wedge p_2 \wedge p_3 \ldots p_n$ گویند و این استنتاج به صورت زیر نشان داده می شود.

$$\begin{array}{l} \left.\begin{array}{l} p_1 \\ p_2 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ p_n \end{array}\right\} \\ \hline \therefore\ q \end{array}$$

Inference [138]

قوانین استنتاج:

الف)قانون انتزاع: $\begin{array}{l} p \to q \\ \underline{p} \\ \therefore p \end{array}$

ب) قانون نقیض انتزاع : $\begin{array}{l} p \to q \\ \underline{\sim q} \\ \therefore \sim p \end{array}$

ج) قیاس : $\begin{array}{l} p \to q \\ \underline{q \to r} \\ \therefore p \to r \end{array}$

د)رفع مولفه: $\begin{array}{l} p \wedge q \\ \underline{\sim p} \\ \therefore q \end{array}$

ه) حذف عاطف: $\dfrac{p \wedge q}{\therefore q}$ یا $\dfrac{p \wedge q}{\therefore p}$

و)ادخال فاصل: $\dfrac{q}{\therefore p \wedge q}$ یا $\dfrac{p}{\therefore p \wedge q}$

مثال: درستی استنتاج های زیر را بررسی کنید:

مثال۱)

جواب:

$\begin{array}{l} q \to r \\ \underline{\sim r} \\ \therefore \sim q \end{array}$ $\quad$ $\begin{array}{l} p \vee q \\ \underline{\sim q} \\ \therefore p \end{array}$

۱)

$\begin{array}{l} q \to r \\ p \vee q \\ \underline{\sim r} \\ \therefore p \end{array}$

مثال۲)

جواب:

$\begin{array}{l} s \to q \\ \underline{s} \\ \therefore q \end{array} \Rightarrow \begin{array}{l} \sim r \to \sim q \\ \underline{q} \\ \therefore \sim(\sim r) \equiv r \end{array} \Rightarrow \begin{array}{l} r \to \sim p \\ \underline{r} \\ \therefore \sim p \end{array}$

$\begin{array}{l} s \to q \\ \sim r \to \sim q \\ r \to \sim p \\ \underline{s} \\ \therefore \sim p \end{array}$

مثال ۳)

جواب:

$\begin{array}{l} r \to \sim p \\ \underline{\sim p \to s} \\ \therefore r \to s \end{array} \Rightarrow \begin{array}{l} p \vee q \\ \underline{\sim q} \\ \therefore p \end{array} \Rightarrow \begin{array}{l} p \to s \\ \underline{p} \\ \therefore s \end{array}$

$\begin{array}{l} p \vee q \\ r \to \sim p \\ \sim p \to s \\ \underline{\sim q} \\ \therefore s \end{array}$

روش برهان خلف: در این روش فرض می کنیم که حکم برقرار نیست و آن را در KB قرار می دهیم. عمل استنتاج را انجام می دهیم اگر به تناقض رسیدیم پس نقیض حکم نادرست بوده و خود حکم درست است.

$\begin{array}{l} \sim p \to q \\ q \to r \\ \underline{\sim r} \\ \therefore p \end{array}$

مثال)پایگاه دانش زیر را با منطق گزاره ها بازنمایی کنید سپس عمل استنتاج را انجام دهید.

«اگر گروه نوازندگان نمی توانست موسیقی را اجرا کند یا از احضار به موقع پذیرایی نمی شد آنگاه ضیافت سال نو لغو می گردید و آقای احمدی عصبانی می شد. اگر این ضیافت لغو می گردید آنگاه می بایستی مبالغ پرداخت شده تحویل داده می شد هیچ تحویلی انجام نشد بنابراین گروه نوازندگان توانست موسیقی را اجرا کند.»

P : گروه نوازندگان موسیقی را اجرا کند

r : لغو شدن ضیافت سال نو

q : از حضار به موقع پذیرایی شود

s : عصبانی شدن آقای احمدی

t : مبالغ پرداخت شده تحویل داده شوند.

$$\frac{\begin{array}{l}(\sim p \vee \sim q) \rightarrow (r \wedge s)\\ r \rightarrow t\\ \sim t\end{array}}{\therefore p} \qquad \frac{\begin{array}{l}r \rightarrow t\\ \sim t\end{array}}{\therefore \sim r} \Rightarrow \frac{\sim r}{\sim r \vee \sim s} \Rightarrow \frac{\begin{array}{l}\sim (p \wedge q) \rightarrow (r \wedge s)\\ \sim p \vee \sim s\end{array}}{\therefore p \wedge q} \Rightarrow \frac{p \wedge q}{\therefore p}$$

هم ارزی[۱۳۹] **:** دو جمله هم ارز منطقی می باشند اگر و فقط اگر در مدل های یکسانی نتایج مشابه داشته باشند.

$$\alpha \equiv \beta \quad Iff \quad \alpha \models \beta \quad and \quad \beta \models \alpha$$

- دو جمله هم ارز منطقی می باشند، اگر و فقط اگر هر دو در مدلهای یکسانی درست باشند.
- $\alpha \equiv \beta$ iff $\alpha \models \beta$ and $\beta \models \alpha$

$(\alpha \wedge \beta)$	$\equiv (\beta \wedge \alpha)$	commutativity of $\wedge$
$(\alpha \vee \beta)$	$\equiv (\beta \vee \alpha)$	commutativity of $\vee$
$((\alpha \wedge \beta) \wedge \gamma)$	$\equiv (\alpha \wedge (\beta \wedge \gamma))$	associativity of $\wedge$
$((\alpha \vee \beta) \vee \gamma)$	$\equiv (\alpha \vee (\beta \vee \gamma))$	associativity of $\vee$
$\neg(\neg\alpha)$	$\equiv \alpha$	double-negation elimination
$(\alpha \Rightarrow \beta)$	$\equiv (\neg\beta \Rightarrow \neg\alpha)$	contraposition
$(\alpha \Rightarrow \beta)$	$\equiv (\neg\alpha \vee \beta)$	implication elimination
$(\alpha \Leftrightarrow \beta)$	$\equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha))$	biconditional elimination
$\neg(\alpha \wedge \beta)$	$\equiv (\neg\alpha \vee \neg\beta)$	de Morgan
$\neg(\alpha \vee \beta)$	$\equiv (\neg\alpha \wedge \neg\beta)$	de Morgan
$(\alpha \wedge (\beta \vee \gamma))$	$\equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma))$	distributivity of $\wedge$ over $\vee$
$(\alpha \vee (\beta \wedge \gamma))$	$\equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma))$	distributivity of $\vee$ over $\wedge$

[139] Equivalence

معتبر بودن[۱۴۰] **:** یک جمله معتبر است اگر در تمام مدل ها درست باشد (جمله همیشه راستگو)

ارضا شدنی[۱۴۱]**:** یک جمله ارضا شدنی است اگر در ربعضی (حداقل یک) مدل ها درست باشد.یک جمله ارضا نشدنی است اگر در هیچ مدلی درست نباشد.

اعتبار و صدق پذیری

- یک جمله معتبر (valid) است اگر در تمام مدل ها درست باشد
  - مثال: True، $A \vee \sim A$، $A \Rightarrow A$، $(A \wedge (A \Rightarrow B) \Rightarrow B)$
- ارتباط معتبر بودن با استنتاج:

$KB \models \alpha$ iff $(KB \Rightarrow \alpha)$ is ***valid***

- یک جمله صدق پذیر (satisfiable) اگر در بعضی از مدل ها درست باشد
  - مثال: $A \vee B$
- یک جمله صدق ناپذیر است اگر در هیچ مدلی درست نباشد
  - مثال: $A \wedge \sim A$
- ارتباط صدق پذیری با استنتاج:

$KB \models \alpha$ iff $(KB \wedge \sim\alpha)$ is ***unsatisfiabe***

اگر جمله α در مدل m درست باشد گوییم مدل m جمله α را ارضا می کند یا m مدلی از α است. ارضا شدن می تواند به صورت شمارش مدل های ممکن برای یافتن مدلی که جمله را ارضا می کند انجام شود. تعیین ارضا شدنی یک جمله در منطق گزاره ای اولین مسئله ای بود که ثابت شد.

نکته : اعتبار و ارضا شدن به یکدگیر وابسته هستند. α معتبر است اگر و فقط اگر ~α ارضا نشدنی باشد α ارضا شدنی است اگر و فقط اگر ~α معتبر نباشد.

الگوهای استدلال در منطق گزاره ای:

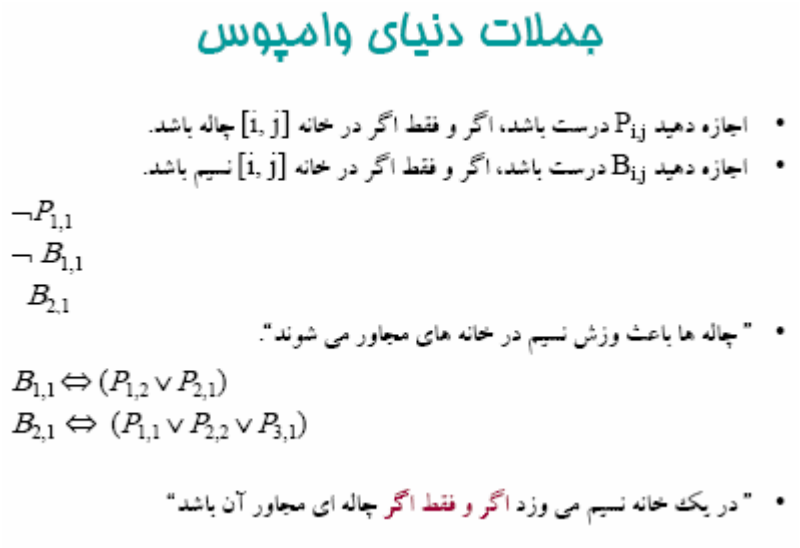
در این بخش به معرفی الگوهای استاندارد استنتاج می پردازیم. این الگوها برای مشتق کردن زنجیره ای از نتایج برای رسیدن به هدف استفاده می شوند. به الگوهای استنتاج، قوانین استنتاج نیز گفته می شود.

۱- قیاس استثنایی (Modus Ponenes) : $\frac{\alpha \Rightarrow \beta, \alpha}{\beta}$ ۲- حذف and : $\frac{\alpha \wedge \beta}{\alpha}$ یا $\frac{\alpha \wedge \beta}{\beta}$

---

[140] Validity
[141] Satisfaibility

این قوانین در هر مثال خاص مورد استفاده قرار می گیرند و استنتاج های صحیحی را بدون نیاز به شمارش مدل ها تولید می کنند. علاوه بر این دو قانون، تمام هم ارزی های منطقی گفته شده می توانند بعنوان قوانین استنتاج به کار گرفته شوند.روش های اثبات به دو دسته تقسیم می شوند. اثبات از طریق قوانین استنتاج و اثبات از طریق جدول صحت. این دو روش را روی مثال دنیای وامپوز بررسی می کنیم.



$$\begin{cases} R_1 \sim P_{1,1} \\ R_2: \beta_{1,1} \Leftrightarrow (P_{1,1} \vee P_{2,1}) \\ R_3: \beta_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1}) \\ R_4 \sim \beta_{1,1} \\ \underline{R_5 \sim \beta_{2,1}} \\ \therefore \sim P_{1,2} \end{cases}$$

اثبات از طریق قوانین استنتاج:

جواب:

اثبات: به این اشتقاق ها یعنی بکار گرفتن دنباله ای از قوانین استنتاج، اثبات گفته می شود. پیدا کردن اثبات ها شبیه یافتن راه حل در مسئله جستجو ست در حقیقت علمگرها(تابع تولید کننده بعدی) همان قوانین استنتاج هستند و می توان در الگوریتم های جستجو از آنها استفاده کرد.

خاصیت یکنوایی[۱۴۲] :

معنای یکنوایی این است که وقتی مقدم های مناسبی مانند β در پایگاه دانش پیدا شوند قوانین استنتاج به کار گرفته می شوند و نتایج جدیدی صرف نظر از آنچه در پایگاه دانش قرار دارد به KB اضافه می شود.(اگر $KB \models \alpha$ آنگاه $(KB, \beta \models \alpha$

[142] Monotonocity

اثبات با استفاده از جدول صحت:

می دانیم که هدف استنتاج این است که برای جمله ای مثل α تصمیم بگیرد آیا $KB \models \alpha$ را یا خیر. اولین الگوریتم استنتاج ، پیاده سازی مستقیم تعریف استلزام است. این الگوریتم عبارتست از شمارش مدل ها و بررسی اینکه آیا در هر مدلی که KB درست است α نیز درست است. در منطق گزاره ای مدل ها شامل انتساب True یا False به هر سمبل گزاره ای است الگوریتم TT-Entails یک الگوریتم کامل و صحیح است. کامل است زیر برای هر KB ،αعمل می کند و همیشه خاتمه می یابد چون تعداد مدل ها متناهی است. این الگوریتم صحیح است زیرا مستقیما تعریف استلزام را پیاده سازی می کند اگر KB و α حاوی n سمبل گزاره ای باشند آنگاه 2^n مدل ممکن وجود دارد. لذا بررسی این مدل ها پیچیدگی نمایی $O(2^n)$ را خواهد داشت .این الگوریتم پیچیدگی فضایی O(n) دارد زیرا به صورت عمقی شمارش می کند.

استفاده از جدول درستی برای استنتاج

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	KB	α_1
false	*false*	*false*	*false*	*false*	*false*	*false*	*false*	*true*
false	*false*	*false*	*false*	*false*	*false*	*true*	*false*	*true*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
false	*true*	*false*	*false*	*false*	*false*	*false*	*false*	*true*
false	*true*	*false*	*false*	*false*	*false*	*true*	*true*	*true*
false	*true*	*false*	*false*	*false*	*true*	*false*	*true*	*true*
false	*true*	*false*	*false*	*false*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*true*	*false*	*false*	*false*	*true*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
true	*true*	*true*	*true*	*true*	*true*	*true*	*false*	*false*

استنتاج بوسیله شمارش

- شمارش تمام مدل ها به روش اول – عمق صحیح و کامل است

```
function TT-ENTAILS?(KB, α) returns true or false
    symbols ← a list of the proposition symbols in KB and α
    return TT-CHECK-ALL(KB, α, symbols, [])

function TT-CHECK-ALL(KB, α, symbols, model) returns true or false
    if EMPTY?( symbols) then
        if PL-TRUE(KB, model) then return PL-TRUE(α, model)
        else return true
    else do
        P ← FIRST( symbols); rest ← REST( symbols)
        return TT-CHECK-ALL(KB, α, rest, EXTEND(P, true, model)) and
                TT-CHECK-ALL(KB, α, rest, EXTEND(P, false, model))
```

برای n سمبول $O(2^n)$

تحلیل[۱۴۳] :

ایده این روش بر اساس برهان خلف استوار است. به جای اینکه ثابت کند گزاره ای مانند q درست است ثابت می کند ~q نمی تواند درست باشد .بدین منظور، فرضیات اولیه را با نقیض هدف ترکیب می کند و عبارات جدیدی تولید می کند این عمل را آنقدر انجام می دهد تا به تناقض برسد وقتی به تناقض رسید می گوید این نقیض هدف بود که باعث شد به هدف نرسیم بنابراین نقیض هدف نادرست و خود هدف درست است. قانون استنتاج Resolution در ترکیب با هر الگوریتم جستجوی کامل، منجر به الگوریتم استنتاج کامل می شود.

قانون Resolution واحد، یک عبارت و یک لیترال را گرفته، عبارت دیگری تولید میکند.

$$\frac{l_1 \vee \ldots \vee l_k, m}{l_1 \vee \ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots \vee l_k}$$

قانون Resolution واحد را میتوان به قانون Resolution کامل، تعمیم داد:

$$\frac{l_1 \vee \ldots \vee l_k, m_1 \vee \ldots \vee m_n}{l_1 \vee \ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots \vee l_k \vee m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n}$$

فرم های نرمال:

از آنجائیکه الگوریتم های استنتاج روی فرم خاصی از جملات،عمل استنتاج را انجام می دهند لذا قبل از شرح هر الگوریتم استنتاج،فرم مربوط به آن الگوریتم را معرفی می کنیم.روش تحلیل بر روی جملات به فرم CNF و روش های زنجیره روبه جلو و زنجره رو به عقب بر روی جملات به فرم Horn عمل استنتاج را انجام می دهند.

فرم CNF[144]:

اگر یک فرمول را بر حسب ضرب حاصل جمع ها بیان کنیم یک فرم CNFحاصل می شود.

ترکیب فصلی متغییرها که درآن همه ی متغییرها یا نقیض آن ها ظاهر شده است را ماکسترم می گوییم.مثلاً ماکسترم های p,q عبارتند از:

$$\underbrace{\bar{p} \vee \bar{q}}_{M_0}, \underbrace{\bar{p} \vee q}_{M_1}, \underbrace{p \vee \bar{q}}_{M_2}, \underbrace{p \vee q}_{M_3}$$

---

[143] Resolution
[144] Conjuction

مثال) $Q=(p\to R)\wedge(p\Leftrightarrow q)$

p	q	R	$(p\to R)$	$(p\Leftrightarrow q)$	Q
F	F	F	T	T	T
F	F	T	T	T	T
F	T	F	T	F	F
F	T	T	T	F	F
T	F	F	F	F	F
T	F	T	T	F	F
T	T	F	F	F	F
T	T	T	T	T	T

$$\sim Q=(\bar{p}\wedge q\wedge\bar{R})\vee(\bar{p}\wedge q\wedge R)\vee(p\wedge\bar{q}\wedge\bar{R})\vee(p\wedge\bar{q}\wedge R)\vee(p\wedge q\wedge\bar{R})$$
$$Q=(p\vee\bar{q}\vee R)\wedge(p\vee\bar{q}\vee\bar{R})\wedge(\bar{p}\vee q\vee R)\wedge(\bar{p}\vee q\vee\bar{R})\wedge(\bar{p}\vee\bar{q}\vee R)$$

فرم DNF[145]:

اگر یک فرمول را بر حسب جمع حاصلضرب ها بنویسیم DNF حاصل می شود.

ترکیب عطفی متغییرها که در آن همه ی متغییر ها یا نقیض آن ها ظاهر شده باشد را مینترم می گوییم. مثلاً مینترم های p,q عبارت است از :

$\underline{\quad}$ $\underline{\quad}$ $\underline{\quad}$ $\underline{\quad}$

m_0 m_1 m_2 m_3

مثال۱:

p	q	$p\to q$
F	F	T
F	T	T
T	F	F
T	T	T

$$p\to q\equiv(p\wedge q)\vee(p\wedge q)\vee(p\wedge q)=\sum m(0,1,3)$$

مثال۲:

p	q	$p\vee q$
F	F	F
F	T	T
T	F	T
T	T	T

$$p\vee q\equiv(\bar{p}\wedge q)\vee(p\wedge\bar{q})\vee(p\wedge q)=\sum m(1,2,3)$$

[145] Conjuction

فرم نرمال عطفی:

قانون Resolution فقط به لیترال های ترکیب فصلی اعمال می شود لذا در بکار گیری آن فقط به پایگاه دانش و پرسش هایی شامل این ترکیبات فصلی نیاز است هر جمله درمنطق گزاره ها با یک ترکیب عطفی از ترکیبات فصلی لیترال هم ارز است. جمله ای که به صورت یک ترکیب عطفی از ترکیبات فصلی لیترال ها بیان می شود فرم نرمال عطفی یا CNF گفته می شود .

روش های بدست آوردن CNF:
- ۱-حذف ترکیبات شرطی
- ۲-کاهش دامنه نفی توسط دمورگان
- ۳-توزیع پذیری

به مثال زیر در دنیای ومپوز دقت کنید:

$$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$$

۱) $[B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})] \wedge [(P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1}]$
$[\neg B_{1,1} \vee (P_{1,2} \vee P_{2,1})] \wedge [\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1}]$

۲) $[\neg B_{1,1} \vee (P_{1,2} \vee P_{2,1})] \wedge [(\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1}]$

۳) $[\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}] \wedge [(B_{1,1} \vee P_{1,2}) \wedge (B_{1,1} \vee \neg P_{2,1})]$
$\equiv [(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (B_{1,1} \vee \neg P_{1,2}) \wedge (B_{1,1} \vee \neg P_{2,1})]$

الگوریتم Resolution:

۱- برای اینکه نشان دهیم KB|=α, مشخص می کنیم (KB ^ ┐α) ارضا نشدنی است.

۲- ابتدا (KB ^ ┐α) را به CNF تبدیل می کنیم.

۳- سپس قانون Resolution به عبارات کوچک حاصل اعمال می شود.

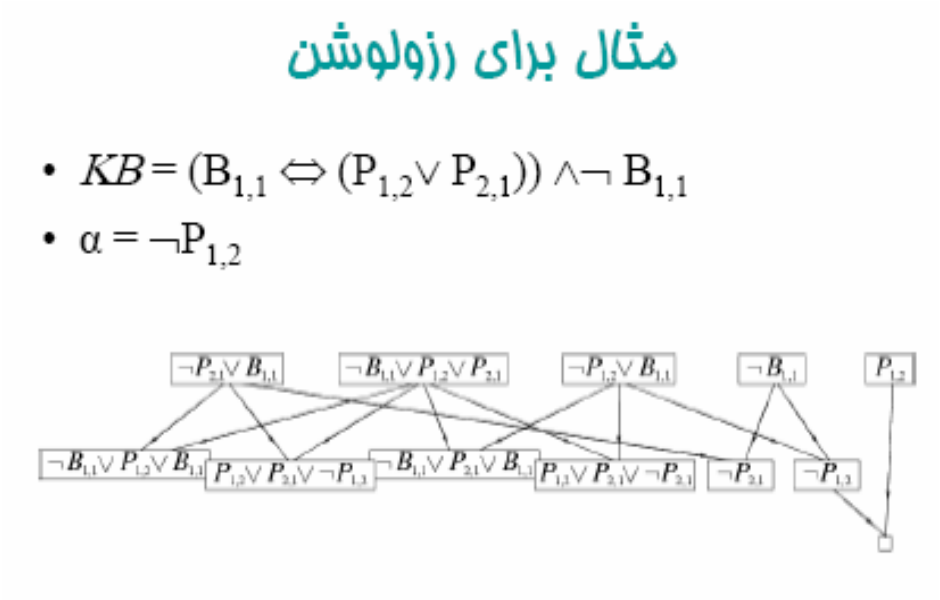
٤- هر جفتی که شامل لیترالهای مکمل باشد، Resolution می شود تا عبارت جدیدی ایجاد گردد.

٥- اگر این عبارت قبلاً در مجموعه نباشد، به آن اضافه می شود.

٦- این فرایند تا محقق شدن یکی از شروط زیر ادامه می یابد:

الـف- هـیچ عبـارت دیگـری وجـود نداشـته باشـد کـه بتوانـد اضـافه شـود. در ایـن صـورت، KB جمله α را نتیجه نمی دهد

ب- کـاربرد قـانون Resolution، عبـارت تهـی را ایجـاد مـی کنـد. در ایـن صـورت، KB جملـه α را نتیجه می دهد.



عبارات هورن[۱۴۶]:

ترکیب فصلی لیترالهایی که حـداکثر یکـی از آنهـا مثبـت باشـد را عبـارات هـورن مـی گـوئیم..محـدود شدن به یک لیترال مثبت ممکن است جالب نباشد اما به سه دلیل زیر مهم است:

- هر عبارت هورن را مـی تـوان بـه صـورت یـک اسـتلزام(ترکیـب شـرطی) نوشـت کـه مقـدم آن ترکیـب عطفـی لیترالهـای مثبـت و تـالی آن یـک لیتـرال مثبـت اسـت. بـه آن نـوع از عبارات هورن که فقط یک لیتـرال مثبـت دارنـد، عبـارات معـین[۱۴۷] نامیـده مـی شـوند.لیتـرال مثبت را رأس و لیترالهـای منفـی را بدنـه عبـارت گوینـد.عبـارت معینـی کـه فاقـد لیترالهـای منفـی باشـد حقیقـت[۱۴۸] نـام دارد.عبـارات معـین اسـاس برنامـه نویـسی منطقـی را مـی سـازند.برخـی جمـلات هـورن، لیتـرال مثبـت ندارنـد ایـن نـوع جمـلات در دنیـای بانـک

---

[146] Horn clause
[147] Define clause
[148] fact

اطلاعـاتی محـدودیت هـای جـامعیتی نامیـده مـی شـود. در الگـوریتم هـای زنجیـره رو بـه جلو و زنجیره رو بـه عقـب بـرای سـهولت فـرض مـی شـود کـه پایگـاه دانـش فقـط شـامل فراکرد های معین می باشد و محدودیت های جامعیتی وجود ندارد .

عبارات هورن و ترکیب شرطی متناظر با آنها در مثال زیر آورده شده است:

$A \wedge B \wedge C \Rightarrow D$ ○ $\quad$ $\neg A \vee \neg B \vee \neg C \vee D$ ○

$A \wedge B \Rightarrow False$ ○ $\quad$ $\neg A \vee \neg B$ ○

نمونه ای از محدودیت جامعیتی و ترکیب شری متناظر با آن در زیر آورده شده است:

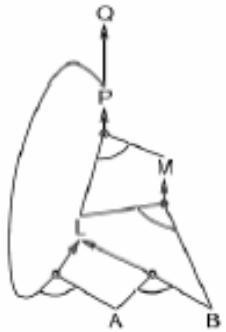
$\neg W_{1,1} \vee W_{1,2} \vee false \equiv W_{1,1} \vee W_{1,2} \Rightarrow false$ $\quad$ $\neg W_{1,1} \vee W_{1,2}$(مثال

- الگوریتم زنجیره سـاز رو بـه جلـو و زنجیـره سـاز رو بـه عقـب ، عمـل اسـتنتاج را فقـط روی جملات هورن انجام می دهند.
- تـصمیم گیـری در مـورد اسـتلزام بـا جمـلات هـورن در زمـان خطـی بـر حـسب انـدازه KB انجام می شود.

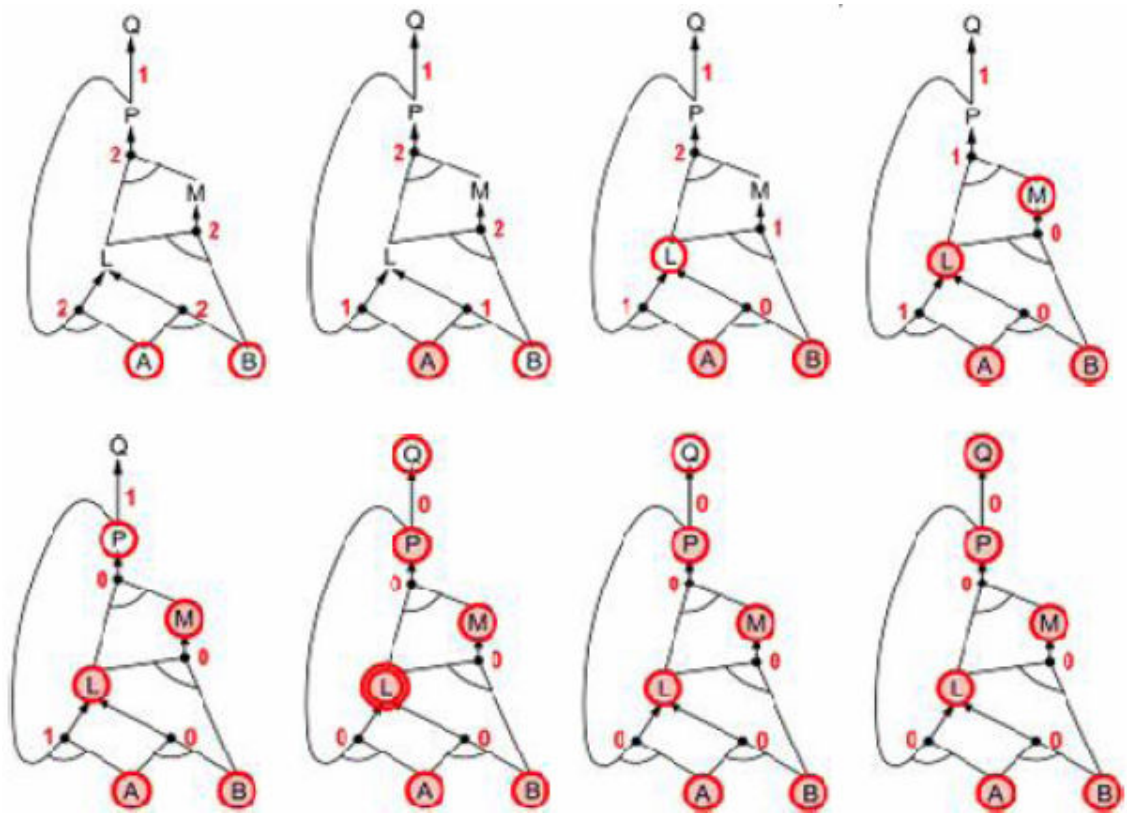
زنجیره ساز رو به جلو[۱۴۹](FC):

بـرای اینکـه ثابـت کنـیم گـزاره ای ماننـد q از KB مـشتق مـی شـود(بدسـت مـی آیـد) بایـد از فرضـیات اولیه شروع کنیم و هر ترکیـب شـرطی کـه مقـدم آن درسـت باشـد تـالی آن بـه KB اضـافه مـی شـود تـا هنگامیکه یا نتیجه بدست آید یا امکان استنتاج وجود نداشته باشد.

$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B



[149] Forward chaining

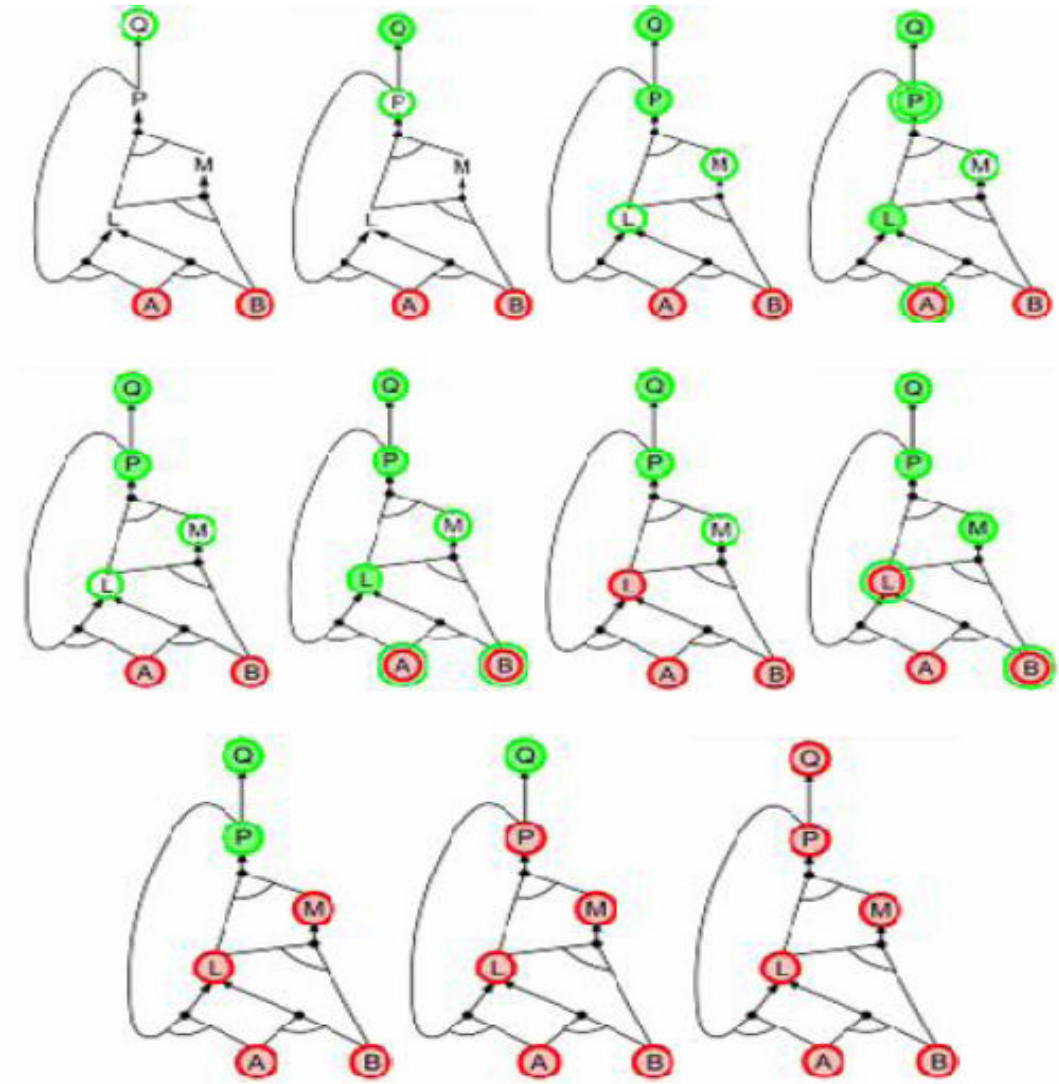


الگوریتم زنجیره ساز رو بـه جلـو از فرضـیات شـروع و بـه هـدف مـی رسـد و مبتنـی بـر داده[۱۵۰] مـی باشـد الگـوریتم زنجیـره سـاز رو بـه جلـو تعیـین مـی کنـد آیـا پرسـش گـزارهٔ q توسـط پایگـاه دانـش جمـلات هورن مـشتق مـی شـود یـا خیـر؟ پیچیـدگی زمـانی الگـوریتم زنجیـره سـاز رو بـه جلـو، خطـی اسـت.FC یـگ الگـوریتم صـحیح اسـت زیـرا اسـتنتاج هـای انجـام شـده در آن کـاربردی از قـانون انتـزاع اسـت . زنجیره ساز رو به جلو در KB به فـرم هـورن کامـل اسـت زیـرا تمـام جمـلات اتمـی را مـشتق مـی کنـد . زنجیـره سـاز رو بـه جلـو نمونـه ای از مفهـوم کلـی داده گراسـت یعنـی اسـتدلالی کـه در آن داده هـا مشخص و ثابت هستند.

[150] Data-driven

زنجیره ساز رو به عقب[۱۵۱](BC):

این الگوریتم از جمله ی پرسش شروع می کند و به عقب بر می گردد اگر پرسش q درست باشد کار تمام است و گر نه الگوریتم، جملات شرطی را در پایگاه دانش می یابد که تالی آن ها q باشد اگر بتوان ثابت کرد تمام مقدم های یکی از آن جملات شرطی درست است آن گاه می توان نتیجه گرفت q درست است. این الگوریتم نوعی استدلال هدف گراست یعنی برای پاسخ به پرسش هایی مثل اکنون به چه چیزی باید جواب بدهم؟ کلید هایم کجاهستند؟ و... مفیداست. اغلب هزینه این الگوریتم بر حسب اندازه KB کمتر از هزینه خطی است چون زنجیره ساز رو به عقب فقط با حقایق مرتبط برخورد می کند.



[151] Backward chaining

مقایسه دو روش FC,BC :

FC بر مبنای داده است ولی BC بر مبنای هدف است در FC ممکن است کار های بسیاری را انجام دهیم که به هدف مربوط نمی شود ولی در BC فقط با حقایق مرتبط با هدف، برخورد میشود و به همین دلیل پیچیدگی BC می تواند بسیار بهتر از خطی نسبت اندازه ی KB باشد چون موارد بیهوده را بسط نمی دهد.

کاربرد زنجیره ای رو به جلو و زنجیره ای رو به عقب:

- در سیستم های انتگرال گیری از راه تجزیه به انتگرال های ساده تر، از استنتاج رو به جلو استفاده می شود چون نقطهٔ آغازین، مشخص و ثابت است.
- در سیستم های خودکار اثبات قضایا بهتر است از استدلال رو به عقب استفاده شود چون فاکتور انشعاب آن ها از طرف مقدم می تواند بسیار زیاد باشد.
- در سیستم های تشخیص پزشکی از استنتاج رو به عقب استفاده می شود چون با کاربری سروکار داریم که از آن پرسش می کنیم.
- برای سیستم های چیدمان اشیا بهتر است از زنجیره ساز رو به جلو استفاده شود چرا که هدف معینی ندارد .

به طور خلاصه زنجیره ساز رو به جلو در پیشگیری، طراحی و کنترل مورد استفاده قرار می گیرد و زنجیره ساز رو به عقب درتشخیص مورد استفاده قرار می گیرد.

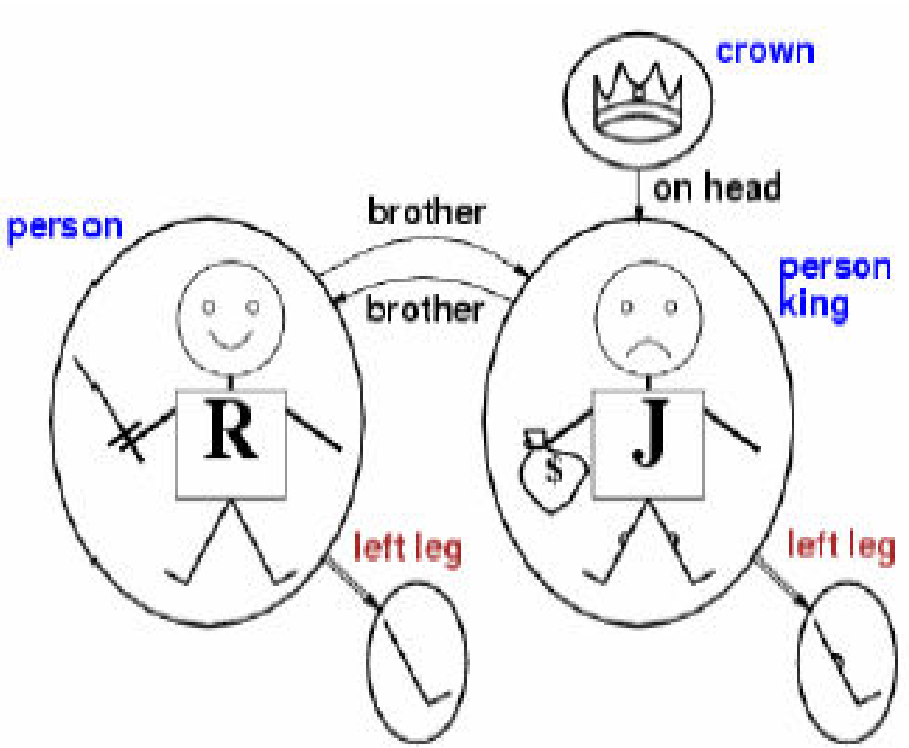
مقایسه دو روش

- FC:
  - بر مبنای داده (data driven)
  - ممکن است کارهای بسیاری انجام دهد که به هدف مربوط نمی شوند
- BC:
  - بر مبنای هدف (goal driven)
  - پیچیدگی BC می تواند بسیار بهتر از خطی نسبت به اندازه KB باشد.

فصل هشتم: منطق مرتبه اول

آنچه در این فصل خواهید آموخت:

- **چرا منطق مرتبه اول(FOL)؟**
- **انواع منطق**
- **بررسی ساختار(نحو) و معانی جملات در FOL**
- **معرفی چند دامنه نمونه با ستفاده از FOL**
- **مهندسی دانش**



مروری بر ویژگی های منطق گزاره ها :

- منطق گزاره ها یک شیوه توصیفی است که در آن دانش و استنتاج از یکدیگر مجزا هستند و استنتاج کاملا مستقل از دامنه است .
- .منطق گزاره ها یک زبان اعلانی است زیرا معنای آن بر پایه یک رابطه صحیح بین جملات و دنیاهای ممکن قرار دارد . علاوه بر این، منطق گزاره ها دارای قدرت بیان کافی برای اداره کردن اطلاعات جزئی با استفاده از ترکیب های فصلی و نقیض است .
- یکی دیگر از ویژگی های منطق گزاره ای، قابلیت ترکیب است که در زبانهای بازنمایی دانش مفید می باشد . در یک زبان ترکیبی معنای یک جمله تابعی از معنای اجزای آن است .
- در منطق گزاره ای، معنا مستقل از متن است ، برخلاف زبانهای طبیعی که معنای جملات وابسته به متن است .
- منطق گزاره ای دارای معنای ترکیبی-اعلانی است که مستقل از محتوا و غیر مبهم است و قدرت بیان بسیار محدودی در نمایش دانش دارد .

منطق مرتبه اول[۱۵۲](FOL)

این منطق، اساس منطق گزاره ها را پذیرفته ولی می خواهد بر اساس منطق گزاره ها ، منطق گویاتری بسازد که ایده های بازنمائی را از زبانهای طبیعی می گیرد و از اشکالات آن اجتناب می کند . وقتی به نحو زبان طبیعی توجه می کنیم درآن موارد زیر را خواهیم داشت :

۱- اشیاء[۱۵۳] : برخی از عناصر ، اسم ها هستند که نشان دهنده ی اشیاء می باشند . دنیا از یکسری اشیایی تشکیل شده که توسط خصوصیاتشان از یکدیگر متمایز می شوند ، مثل رنگ ها ، اعداد ، کشورها و...

۲ – رابطه ها[۱۵۴] : عناصر دیگر ، فعل ها یا عبارات فعلی هستند که رابطه بین اشیاء را نمایش می دهند . مثل رابطه زوج بودن ، اول بودن برای اشیاء از نوع اعداد و ...

انواع رابطه ها :

- رابطه های یکانی: prim(5)
- رابطه دوتایی: brother (richard , jack)
- توابع: plus (2,3)=5 (دو شیء میگیرد و خروجی آن یک شیء دیگر است.)

مثال : برای جملات داده شده، اشیا، روابط و توابع را مشخص کنید:

One plus two equals three

(یک بعلاوه دوبرابر سه)

Object : one , two , three (اشیاء)

Relation : equals (رابطه)

Function : plus (تابع)

---

[152] First Order Logic

[153] Object

[154] Relation

Square niegboring the wampus smelly
(مربع های مجاور وامپوز بو می دهند.)
(اشیاء) Object : square , wampus
(رابطه) Relation : niegboring
(تابع) Function: smelly

انواع منطق:

انـواع منطـق را از دو دیـدگاه هـستی شناسـی و حقیقـت شناسـی بررسـی مـی کنـیم.بنـابراین بهتـر اسـت ابتدا این دو مفهوم بررسی شوند.

- **هستی شناسی**[۱۵۵]

تفـاوت عمـده ی بـین منطـق مرتبـه اول و منطـق گـزاره هـا بـه هـستی شناسـی برمـی گـردد . هـستی شناسی یک زبان یعنی آنچه این زبان در مـورد یـک حقیقـت (مـسئله) فـرض مـی کنـد . بـه عنـوان مثـال منطـق گـزاره ای فـرض مـی کنـد حقـایقی وجـود دارنـد کـه ممکـن اسـت رخ بدهنـد یـا ندهنـد . منطـق مرتبه اول فرض می کند کـه دنیـا شـامل اشـیاء و روابـط بـین آنهاسـت کـه ممکـن اسـت ایـن رابطـه هـا برقـرار باشـد یـا نباشـد . منطـق هـای خـاص منظـوره هـستی شناسـی بیـشتری دارد . بـه عنـوان مثـال منطق موقتی فرض می کند که حقایق در زمانهای خاص رخ می دهد .

- **حقیقت شناسی**[۱۵۶]

حقیقـت شناسـی یعنـی حـالات ممکـن دانـشی کـه منطـق در مـورد یـک حقیقـت فـرض مـی کنـد . در منطق گزاره ای و منطق مرتبـه اول هـر جملـه یـک حقیقـت را بازنمـایی مـی کنـد و عامـل مـی توانـد آن حقیقت را درست یا نادرست بدانـد یـا نظـری نداشـته باشـد . بنـابراین، ایـن دو منطـق در مـورد هـر جملـه سه حالت را در نظر می گیرنـد . سیـستمی کـه از نظریـه احتمـال اسـتفاده مـی کنـد درجـه ای از اعتقـاد را در نظر مـی گیـرد کـه از صـفر تـا یـک تغییـر مـی کنـد . حقـایق در منطـق فـازی درجـه ای از درسـتی را نـشان مـی دهنـد . درجـه ای از اعتقـاد در منطـق احتمـالی بـا درجـه ای از درسـتی در منطـق فـازی متفاوت است .

Language	Ontology	Epistemology
Propositional Logic	facts	true/false/unknown
First Order Logic	facts, objects, relations	true/false/unknown
Temporal Logic	facts, objects, relations, times	true/false/unknown
Probability theory	facts	degree of belief∈[0, 1]
Fuzzy logic	degree of truth ∈[0, 1]	known interval value

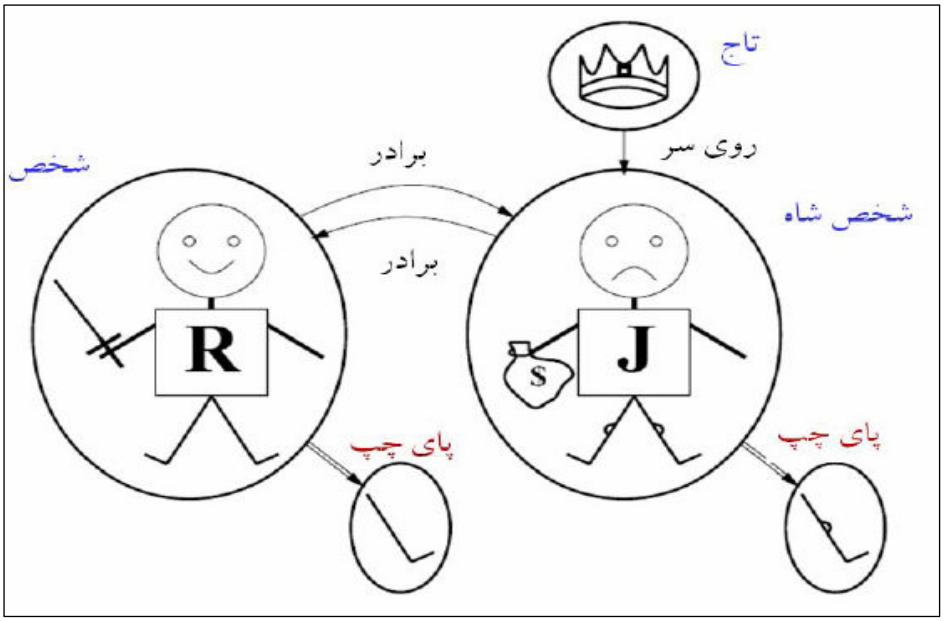
155 Ontology
156 Epistemology

زبان	هستی شناسی (آنچه در جهان است)	حقیقت شناسی (اعتقادات عامل راجع به یک حقیقت)
منطق گزاره ها	حقایق	درست / نادرست / نامشخص
FOL	حقایق / اشیاء / روابط	درست / نادرست / نامشخص
منطق موقتی	حقایق / اشیاء / روابط / زمان	درست / نادرست / نامشخص
نظریه احتمال	حقایق	درجه ای از اعتقادات بین [۱و۰]
منطق فازی	حقایق با درجه درستی متعلق به [۱و۰]	در فاصله ای معین

جدول مربوط به زبانهای رسمی و تعهدات هستی شناسی و حقیقت شناسی آنها

مدل های منطق مرتبه اول:

در منطق مرتبه اول،مدل ها دارای اشیاء هستند. در حقیقت ،مدلها شامل اشیاء و روابط میان آنها می باشند . دامنه یک مدل، مجموعه ای از اشیایی است که در آن مدل وجود دارند.این اشیاء عناصر دامنه نامیده می شوند. مدلی از منطق مرتبه اول را در شکل زیر مشاهده می کنید. مدل شکل زیر ۵ شیء دارد . اشیاء موجود در این مدل عبارتند از : ریچارد ، جان ، تاج ، پای چپ جان ، پای چپ ریچارد. اشیاء موجود در یک مدل ممکن است به شکل های مختلفی با یکدیگر ارتباط داشته باشند . رابطه مجموعه ای از چندتایی هایی از اشیاء مرتبط با هم است . نمونه هایی از روابط و توابع موجود در این مدل را در زیر مشاهده می کنید:



Brother : { <Richard , john> و <John , Richard>} (رابطه برادری)

On head : {<Crown , Khing John>} (رابطه بر سر بودن تاج)

رابطه های دودوئی در این مدل :

- Brother (John , Richard)
- Brother (Richard , john)
- On-head (Crown , Khing John)

رابطه های یکانی در این مدل :

- Person(John)
- Person(Richard)
- King(John)
- King(Richard)
- Be-crown (Crown)

توابع موجود در این مدل:

- Left_leg_of(John)
- Left_leg_of(Richard)
- Length(Left_leg_of(John))

نحو منطق مرتبه اول :

عناصر اصلی نحو منطق مرتبه اول،سمبل هایی است که نشان دهنده اشیاء،رابطه ها و توابع هستند. سمبل ها در منطق مرتبه اول به سه دسته تقسیم می شوند . این سمبل ها با حروف بزرگ شروع می شوند :

- سمبل های ثابت : برای اشاره به اشیاء به کار می روند . (مثل John , Richard)
- سمبل های مسند : رابطه ها را نشان می دهند. (مثل Brother , Person , King)
- سمبل های تابع : توابع را نشان می دهند . (مثل Left_leg_of)

در منطق مرتبه اول تعداد مدل ها و تعداد متغیر ها ممکن است بی نهایت باشد . روش بررسی(شمارش)مدل، که در منطق گزاره ها به درستی کار می کند در منطق مرتبه اول قابل استفاده نیست زیرا امکان شمارش تمام مدل های ممکن وجود ندارد .

گرامر منطق مرتبه اول :

```
Sentence → AtomicSentence
           | Sentence Connective Sentence
           | Quantifier Variable, ... Sentence
           | ~ Sentence
           | ( Sentence )

AtomicSentence → Predicate(Term, ...) | Term = Term

Term → Function(Term, ...)
       | Constant
       | Variable
```

Connective → ⇒ | ∧ | ∨ | ⇔
Quantifier → ∀ | ∃
Constant → A | X_1 | John | ...
Variable → a | x | ...
Predicate → Before | HasColor | ...
Function → MotherOf | LeftLegOf

: Term

هر ترم یک عبارت منطقی است که به یک شیء اشاره می کند . ترم ها سه نوع هستند :

- **ثابت ها :** (مثل ... , John , B , A) که با حروف بزرگ شروع می شوند.
- **متغیرها :** (مثل... , b , x , a) که با حروف کوچک شروع می شوند .
- **توابع :** هر ترم پیچیده نیز شامل یک سمبل تابع و چند ترم در داخل پرانتز به عنوان آرگومان های آن تابع است.این نکته را به خاطر داشته باشید که خروجی یک تابع، یک شیء است ولی خروجی یک رابطه T یا F است.در زیر به چند نمونه از توابع اشاره شده است.

Brother (John), Length (Left_Leg_of(John)), Plus(2,3)

جملات اتمیک :

با ترکیب ترم ها که به اشیاء اشاره می کنند و مسند ها که به روابط اشاره می کنند ، جملات اتمیک به دست می آیند. جملات اتمیک حقایقی را بیان می کنند که شامل مسند(فعل جمله) همراه با ترم هایی داخل پرانتز هستند و نتیجه آن T یا F است. در زیر به چند نمونه از جملات اتمیک اشاره شده است.

Brother (John , Richard)
Morried [Father(Richard) , Mother(John)]
<u>Brother</u> [John , <u>Brother</u>(John)]
Larg_than [Length(Left_Leg_Of(John)), Length(Left_Leg_Of(Richard))]

جملات پیچیده :

با ترکیب جملات اتمیک و روابط منطقی می توان جملات پیچیده تری ساخت . روابط منطقی عبارتند از: ∧ ,∨ ,⇒ ,~ ,⇔ . در زیر به چند نمونه از جملات پیچیده اشاره شده است.

Brother(John , Richard) ∧ Brother(Richard , John)
King(Richard) ∨ king(John)
~ King(Richard) → King(John)
Older(John , 30) → ~Younger(John , 30)
~ Brother(John , peter)

سورها[۱۵۷] :

در منطق مرتبه اول به جای شمارش اشیاء با استفاده از نام آنها، می توان خواص همه یا برخی از اشیاء را بازنمایی کرد . این امر با استفاده از سورها امکان پذیر است . منطق مرتبه اول

[157] Quantifier

دارای دو سور استاندارد به نامهای سور وجودی و سور عمومی می باشد . سورها کمک می کنند تا به جای شمارش اشیاء از طریق نام آنها ، خواص کلکسیونی از اشیاء را بیان کنیم .

سورعمومی[۱۵۸] :

بیان قوانین کلی در منطق گزاره ها دشوار است ولی در منطق مرتبه اول با استفاده از سور عمومی این کار به آسانی امکان پذیر است .

سورهای عمومی مطابق شکل روبروتعریف می شوند : < جمله > < متغیر > $\forall$

مثال: $\forall x$, King (x) $\Rightarrow$ person(x) (همه پادشاهان شخص هستند)

$\forall x$, P(x) : این جمله در یک مدل معادل جمله $P(k1) \cap p(k2) \cap \ldots p(kn)$ است که k ها عناصر دامنه مدل هستند و در آن p(x) یک عبارت منطقی است که بیان می کند p برای هر شیء x درست است .

King (John) $\Rightarrow$ Person (John)
King (Richard) $\Rightarrow$ Person (Richard)
King (Left_Leg_Of(John)) $\Rightarrow$ Person (Left_Leg_Of(John))
King (Left_Leg_Of(k)) $\Rightarrow$ Person (Left_Leg_Of(R))
King (Crown) $\Rightarrow$ Person (Crown)

ارزش نهایی تمام جملات فوق T (صحیح) است . (با استفاده از روش انتفاع مقدم یعنی اگر مقدم نادرست باشد ، بدون توجه به طرف دوم ، طرف دوم درست است، درستی جملات دوم تا چهارم مشخص می شود.)

سورعمومی

$\forall \langle Variables \rangle \langle Sentence \rangle$

Everyone at the class is smart

$\forall x\ at(x, Class) \Rightarrow Smart(x)$

- سور عمومی برابر است با ترکیب عطفی تمام جملاتی که با جایگذاری متغیرها در Sentence بدست می آیند.

$at(KingJohn, Class) \Rightarrow Smart(KingJohn)$
$\wedge\ at(Richard, Class) \Rightarrow Smart(Richard)$
$\wedge\ at(Class, Class) \Rightarrow Smart(Class)$
$\wedge$

سور وجودی[۱۵۹] :

یک جمله با سور عمومی حقیقتی را در مورد همه اشیاء بیان می کند در حالی که سور وجودی حقیقتی را در مورد بعضی از اشیاء بین می کند .

سورهای وجودی مطابق شکل روبروتعریف می شوند : < جمله > < متغیر > $\exists$

مثال : $\exists x$, Crown(x) $\wedge$ On Head(x , john)

---

[158] Universial Quantifier
[159] Existential Quantifier

$\exists x\ ,\ p(x)$: به این معناست که p برای حداقل یک شیء درست باشد . به عبارت دیگر، این جمله ، معادل جمله فصلی $p(k1) \vee p(k2) \vee \ldots p(kn)$ است .

سوروجودی

$\exists \langle Variables \rangle \langle Sentence \rangle$

Someone at class is smart

$\exists x\ at(x, class) \wedge Smart(x)$

- سور وجودی برابر است با ترکیب **فصلی** تمام جملاتی که با جایگذاری متغیرها در Sentence بدست می آیند.

$$
\begin{aligned}
& at(KingJohn, Class) \wedge Smart(KingJohn) \\
\vee\ & at(Richard, Class) \wedge Smart(Richard) \\
\vee\ & at(Class, Class) \wedge Smart(Class)
\end{aligned}
$$

خصوصیات سورها :

به نظر می رسد در استفاده از سور عمومی ، رابط مورد استفاده ترکیب شرطی و در استفاده از سور وجودی ، رابط مورد استفاده ترکیب عطفی مناسب باشد . بکارگیری ترکیب عطفی به عنوان رابط اصلی همراه با سور عمومی منجر به ادعایی بسیار قوی و بکارگیری ترکیب شرطی به عنوان رابط اصلی همراه سور وجودی منجر به ادعایی بسیار ضعیف می شود . به مثال های زیر توجه کنید:

$\forall x\ ,\ King(x) \Rightarrow Person(x)$: هر کسی پادشاه است آنگاه یک شخص است

$\forall x\ ,\ King(x) \wedge Person(x)$: همه افراد پادشاه هستند (ادعایی قوی)

$\exists \mathbf{x}\ ,\ King(x) \wedge Person(x)$: وجود دارد شخصی که پادشاه است

$\exists \mathbf{x}\ ,\ King(x) \Rightarrow Person(x)$: (ادعایی ضعیف)اگر وجود داشته باشد پادشاهی ،آنگاه شخص خواهد بود

یک اشتباه متداول

- سور عمومی اغلب با ترکیب شرطی و سور وجودی اغلب با ترکیب عطفی بکار می رود

$\forall x\ Human(x) \Rightarrow Mortal(x)$

$\exists x\ Sister(x, Spot) \wedge Cat(x)$

- مثال : جمله زیر به معنای " هر کسی در کلاس است و هر کسی باهوش است " می باشد

$\forall x\ at(x, Class) \wedge Smart(x)$

ویا جمله زیر:

$\exists x\ at(x, Class) \Rightarrow Smart(x)$

مثال:جملات زیر را با توجه به نکته ای که در بالا به آن اشاره شد در فرم منطق مرتبه اول بازنمایی کنید.(فرض کنید که $p(x)$ معرف سیاستمدار بودنx و$q(x)$ معرف نادرست بودن x باشد.)

برخی سیاستمداران نادرست هستند : $\exists x , p(x) \wedge q(x)$

هیچ سیاستمداری نادرست نیست : $\forall x , p(x) \Rightarrow \sim q(x)$

تمامی سیاستمداران نادرست نیستند : $\exists x , p(x) \wedge \sim q(x)$

تمامی سیاستمداران نادرست هستند : $\forall x , p(x) \Rightarrow q(x)$

سورهای تو در تو[160]:

با استفاده از چند سور می توانیم جملات پیچیده تری بسازیم. در ساده ترین حالت،سورها از یک نوع هستند.به مثال های زیر توجه کنید.

$\forall x \forall y \ \text{Brother}(x , y) \Rightarrow \text{Sibling}(x , y)$
$\forall x,y \ \text{Sibling}(x , y) \leftrightarrow \text{Sibling}(y , x)$
(Sibling : همزاد بودن)

در دو جمله فوق، چون سوره از یک نوع هستند جابجایی سورها تاثیری در معنای جمله ندارد.ولی اگر سورهای تودرتو بایکدیگر متفاوت باشند معنای جمله با جابجایی سورها تحت تاثیر قرار خواهد گرفت.

$\forall x \exists y \neq \exists y \forall x$
$\exists x \exists y = \exists y \exists x$
$\forall x \forall y \equiv \forall y \forall x$

حداقل یک شخص وجود دارد که همه اشخاص را دوست دارد : $\exists x \forall y \ \text{Loves}(x , y)$

همه افراد،حداقل یک نفر را دوست دارند : $\forall y \exists x \ \text{Loves}(x , y)$

به مثال دیگری در زمینه ریاضیات توجه فرمائید.

$\forall x \in /N , \exists y \in /Z , x+y=0$ T
$\exists y \in /Z , \forall x \in /N , x+y=0$ F

ارتباط بین سورها : سور عمومی و سور وجودی از طریق نقیض با یکدیگر ارتباط دارند .

$\sim (\exists x \ p) \equiv \forall x , \sim p$ $\sim (\forall x \ p) \equiv \exists x , \sim p$

$\exists x \ p \equiv \sim\forall x , \sim p$ $\forall x \ p \equiv \sim\exists x , \sim p$

به مثال های زیر در این زمینه توجه فرمائید:

همه افراد، بستنی را دوست دارند : $\forall x \ \text{Likes}(x , \text{Icecream})$

وجود ندارد کسی که بستنی دوست نداشته باشد : $\sim \exists x , \sim \text{Likes}(x , \text{Icecream})$

افرادی وجود دارند که گل کلم دوست دارند : $\exists x \ \text{Likes}(x , \text{Broccoli})$

هیچ کس نیست که گل کلم دوست نداشته باشد : $\sim\forall x \ \sim\text{Likes}(x , \text{Broccoli})$

---

[160] nested quantifiers

ارتباط بین سورها

- سور عمومی و وجودی از طریق تناقض با یکدیگر در ارتباط هستند:

$$\neg(\forall x P) \equiv \exists x \neg P$$
$$\neg(\exists x P) \equiv \forall x \neg P$$
$$\forall x P \equiv \neg\exists x \neg P$$
$$\exists x P \equiv \neg\forall x \neg P$$

- مثال: دوگانی سور

$$\forall x\ Likes(x, IceCream) \qquad \neg\exists x\ \neg Likes(x, IceCream)$$
$$\exists x\ Likes(x, Broccoli) \qquad \neg\forall x\ \neg Likes(x, Broccoli)$$

مفهوم تساوی[۱۶۱] **:** در منطق مرتبه اول به غیر از به کارگیری مسندها برای ساختن جملات اتمی، راه دیگری نیز وجود دارد . ما میتوانیم از علامت تساوی (=) برای ساختن جملاتی که در آن دو ترم به شیء یکسانی اشاره می کنند، استفاده کنیم .به عنوان مثال در جمله Father(John)= peter ، شیءای که Father(John) به آن اشاره می کند با شیءای که peter به آن اشاره می کند یکسان است.

همچنین ،از علامت = ،می توان همراه با سمبل نقیض برای تاکید بر مساوی نبودن دو شیء استفاده کرد .

$\exists x,y$ Brother(x , Richard) $\wedge$ Brother(y , Richard) $\wedge$ ~(x=y)

"ریچارد حداقل دو برادر دارد "

ادعاها و پرسش ها در منطق مرتبه اول :

جملات با استفاده از Tell به پایگاه دانش اضافه می شوند . چنین جملاتی را **ادعا**[۱۶۲] می گوییم .

Tell (KB , King(John))

Tell (KB , $\forall$x King(x) $\Rightarrow$ Person(x))

با استفاده از Ask سوالاتی از پایگاه دانش در مورد درستی جملات پرسیده می شود . سوالاتی که توسط Ask از پایگاه دانش پرسیده می شود ، **درخواست**[۱۶۳] یا **هدف**[۱۶۴] نام دارد .

Ask(KB , King(Richard))

Ask(KB , Person(John))

دامنه ی خویشاوندی :

اشیاء : افراد

رابطه یکانی : Male , Female

رابطه دودوئی : Parent , Husband , Married , Brother

تابع : Father , mother

---

[161] equality
[162] assertion
[163] Query
[164] Goal

به جملات زیر در دامنه خویشاوندی توجه فرمائید:

$\forall$ m,c Mother(c) $\Leftrightarrow$ Female (m) $\wedge$ Parent(m,c)

مادر هر شخصی، والد مونث آن شخص است .

$\forall$ w,h Husband(h,w) $\Leftrightarrow$ Male(h) $\wedge$ Spouse (h,w)

شوهر هر فرد ، همسر مذکر آن فرد است .

$\forall$x Male(x) $\Leftrightarrow$ ~Female (x)

مذکر و مونث بودن مولفه های متضادند .

$\forall$ p,c Parent(p,c) $\Leftrightarrow$ Child(c,p)

والدین و فرزند رابطه معکوس دارند.

$\forall$ g,c Grandparent(g,c) $\Leftrightarrow$ $\exists$p parent(g,p) $\wedge$ parent(p,c)

مادربزرگ یا پدربزرگ هر فرد ، والدِ والد آن فرد است .

$\forall$ x,y Sibling(x,y) $\Leftrightarrow$ $\exists$p parent(p,x) $\wedge$ parent(p,y) $\wedge$ (x$\neq$y)

همزاد[۱۶۵] یک فرد ، فرزند دیگر والدین آن فرد است .

دامنه مجموعه ها :

مسند دودوئی : $x \in S$, $S1 \subseteq S2$

مسند یکانی : Set(s) (یک موجودیت را می گیرد و می گوید مجموعه هست یا نه؟)

توابع دودوئی : $S1 \cap S2$ ، $S1 \cup S2$

ثابت : { } (مجموعه تهی)

{x | s} : مجموعه حاصل از افزودن x به s

به جملات زیر در دامنه مجموعه ها توجه فرمائید:

$\forall$ **S1,S2** $S1 \subseteq S2 \Leftrightarrow (\forall x \;\; x \in S1 \rightarrow x \in S2)$

یک مجموعه ، زیرمجموعه دیگری است اگر و فقط اگر تمام اعضای مجموعه اول عضو مجموعه دوم هم باشند .

$\forall$ **S1,S2** $(S1 = S2) \Leftrightarrow (S1 \subseteq S2 \wedge S2 \subseteq S1)$

دو مجوعه مساوی هستند اگر و فقط اگر هر کدام زیر مجموعه دیگری باشد.

$\forall$ **x,S1,S2** $x \in (S1 \cap S2) \Leftrightarrow (x \in S1 \wedge x \in S2)$

یک شی عضو اشتراک دو مجموعه است اگر و فقط اگر عضو هر دو مجموعه باشد.

$\forall$ **x,S1,S2** $x \in (S1 \cup S2) \Leftrightarrow (x \in S1 \vee x \in S2)$

یک شی عضو اجتماع دو مجموعه است اگر و فقط اگر عضو یکی از دو مجموعه باشد.

$\forall$ **S**,Set(s) $\Leftrightarrow$ **(S={})** $\vee$ **(**$\exists$ **x , S2** Set(s2) $\wedge$ **(s=** {x | s2}) **)**

مجموعه ها یا یک مجوعه تهی است یا با اضافه کردن یک عنصر به یک مجموعه دیگر ساخته می شوند.

$\forall$ **x,S** **x**$\in$**S** $\Leftrightarrow$ **S=** { **x** | **S** }

افزودن یک عنصر که قبلا در مجموعه وجود داشته باشد ، تاثیری ندارد .

---

[165] Sibling

مهندسی دانش[۱۶۶] :

به فرآیند کلی ساخت پایگاه دانش ، فرآیند مهندسی دانش گفته می شود . یک مهندس دانش کسی است که حوزه یا دامنه خاصی را بررسی می کند و یاد می گیرد چه مفاهیمی در آن دامنه مهم هستند .یک فرآیند مهندسی شامل مراحل زیر است:

۱- مشخص نمودن وظیفه

مهندس دانش باید بازه ای از پرسش ها را که برای پایگاه دانش قابل دسترس است توصیف کند و انواع حقایقی که برای یک مسئله خاص وجود دارند را تعیین نماید.

۲- جمع آوری دانش مربوطه

مهندس دانش ممکن است در دامنه یا حوزه ای خاص خبره باشد یا از افراد خبره برای استخراج اطلاعات استفاده کند.این فراین کسب دانش[۱۶۷] نامیده می شود. دز این مرحله دانش به طور رسمی بازنمایی نمی شود و ایده این مرحله درک حوزه پایگاه دانش و درک اینکه دامنه چگونه کار می کند است.

۳- تصمیم گیری در مورد واژه نامه مسندها ، توابع و ثابت ها

یعنی ترجمه مفاهیم مهم در سطح دامنه به اسامی در سطح منطق. به عنوان مثال آیا چاله ها باید به وسیله اشیا یا به وسیله مسندهای یکانی بازنمایی شوند

۴- کدگذاری دانش عمومی در مورد دامنه

مهندس دانش باید اصول مربوط به تمام ترم های واژه نامه را بنویسد. بدین ترتیب معنای ترم ها مشخص خواهند شد.

۴- کدگذاری توصیف یک نمونه مسئله خاص

این مرحله شامل نوشتن جملات اتمی در مورد نمونه هایی از مفاهیمی است که به عنوان بخشی از هستی شناسی هستند.

۵- اعمال پرسش ها به رویه استنتاج و دریافت پاسخ

۶- اشکال زدایی پایگاه دانش

اگر یک اصل در یک پایگاه دانش نباشد آنگاه جواب برخی از پردازش ها وجود نخواهد داشت که باید اشکال زدایی صورت گیرد.فقدان اصول و یا وجود اصول ضعیف را می توان با توجه به مکان هایی که زنجیره استدلال به طور غیر منتظره متوقف می شود مشخص کرد.

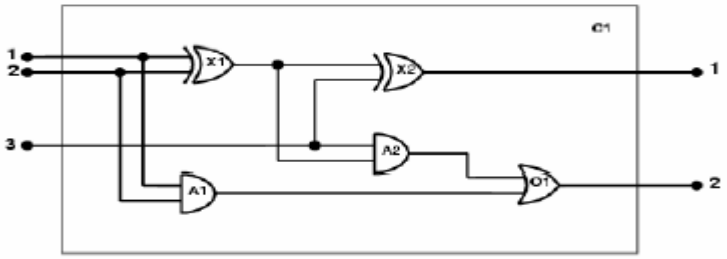
برای درک بهتر مراحل فوق،این هفت مرحله را در یک مثال(حوزه مدارهای الکترونیکی) بسط می دهیم.

---

[166] Knowledge Engineering
[167] Knowledge acquisition

دامنه مدارهای الکتریکی

- جمع کننده یک بیتی



دامنه مدارهای الکتریکی

۱. مشخص نمودن وظیفه
- آیا مدار واقعا به درستی جمع می کند؟ (وارسی مدار)

۲. جمع آوری دانش مربوطه
- ترکیب سیم ها و گیت ها؛ انواع گیت ها (AND, OR, XOR, NOT)
- دانش نامربوط: اندازه، شکل، رنگ، قیمت گیت ها

۳. تصمیم گیری در مورد لغات
- راه های مختلف:

$Type(X_1) = XOR$

$Type(X_1, XOR)$
$XOR(X_1)$

دامنه مدارهای الکتریکی

۴. کد نمودن دانش عمومی دامنه

$-\forall t_1,t_2\ Connected(t_1, t_2) \Rightarrow Signal(t_1) = Signal(t_2)$
$-\forall t\ Signal(t) = 1 \vee Signal(t) = 0$

$-1 \neq 0$

$-\forall t_1,t_2\ Connected(t_1, t_2) \Rightarrow Connected(t_2, t_1)$

$-\forall g\ Type(g) = OR \Rightarrow Signal(Out(1,g)) = 1 \Leftrightarrow \exists n\ Signal(In(n,g)) = 1$

$-\forall g\ Type(g) = AND \Rightarrow Signal(Out(1,g)) = 0 \Leftrightarrow \exists n\ Signal(In(n,g)) = 0$

$-\forall g\ Type(g) = XOR \Rightarrow Signal(Out(1,g)) = 1 \Leftrightarrow Signal(In(1,g)) \neq Signal(In(2,g))$

دامنه مدارهای الکتریکی

۵. کد نمودن دانش مربوط به یک نمونه مساله خاص

$Type(X_1) = XOR$ $Type(X_2) = XOR$
$Type(A_1) = AND$ $Type(A_2) = AND$
$Type(O_1) = OR$

$Connected(Out(1,X_1),In(1,X_2))$ $Connected(In(1,C_1),In(1,X_1))$
$Connected(Out(1,X_1),In(2,A_2))$ $Connected(In(1,C_1),In(1,A_1))$
$Connected(Out(1,A_2),In(1,O_1))$ $Connected(In(2,C_1),In(2,X_1))$
$Connected(Out(1,A_1),In(2,O_1))$ $Connected(In(2,C_1),In(2,A_1))$
$Connected(Out(1,X_2),Out(1,C_1))$ $Connected(In(3,C_1),In(2,X_2))$
$Connected(Out(1,O_1),Out(2,C_1))$ $Connected(In(3,C_1),In(1,A_2))$

دامنه مدارهای الکتریکی

۶. اعمال پرس و جو بر رویه استنتاج

چه ترکیبی از ورودی ها باعث می شوک که اولین خروجی مدار C_1 (بیت جمع) به صفر و خروجی دوم مدار C_1 (بیت نقلی) به یک تبدیل شود؟

$$\exists i_1,i_2,i_3\ Signal(In(1,C_1)) = i_1 \wedge Signal(In(2,C_1)) = i_2 \wedge Signal(In(3,C_1)) = i_3 \wedge Signal(Out(1,C_1)) = 0 \wedge Signal(Out(2,C_1)) = 1$$

پاسخ های ممکن:

$\{i_1/1,\ i_2/1,\ i_3/0\}$,
$\{i_1/1,\ i_2/0,\ i_3/1\}$,
$\{i_1/0,\ i_2/1,\ i_3/1\}$

دامنه مدارهای الکتریکی

۶. اعمال پرس و جو بر رویه استنتاج

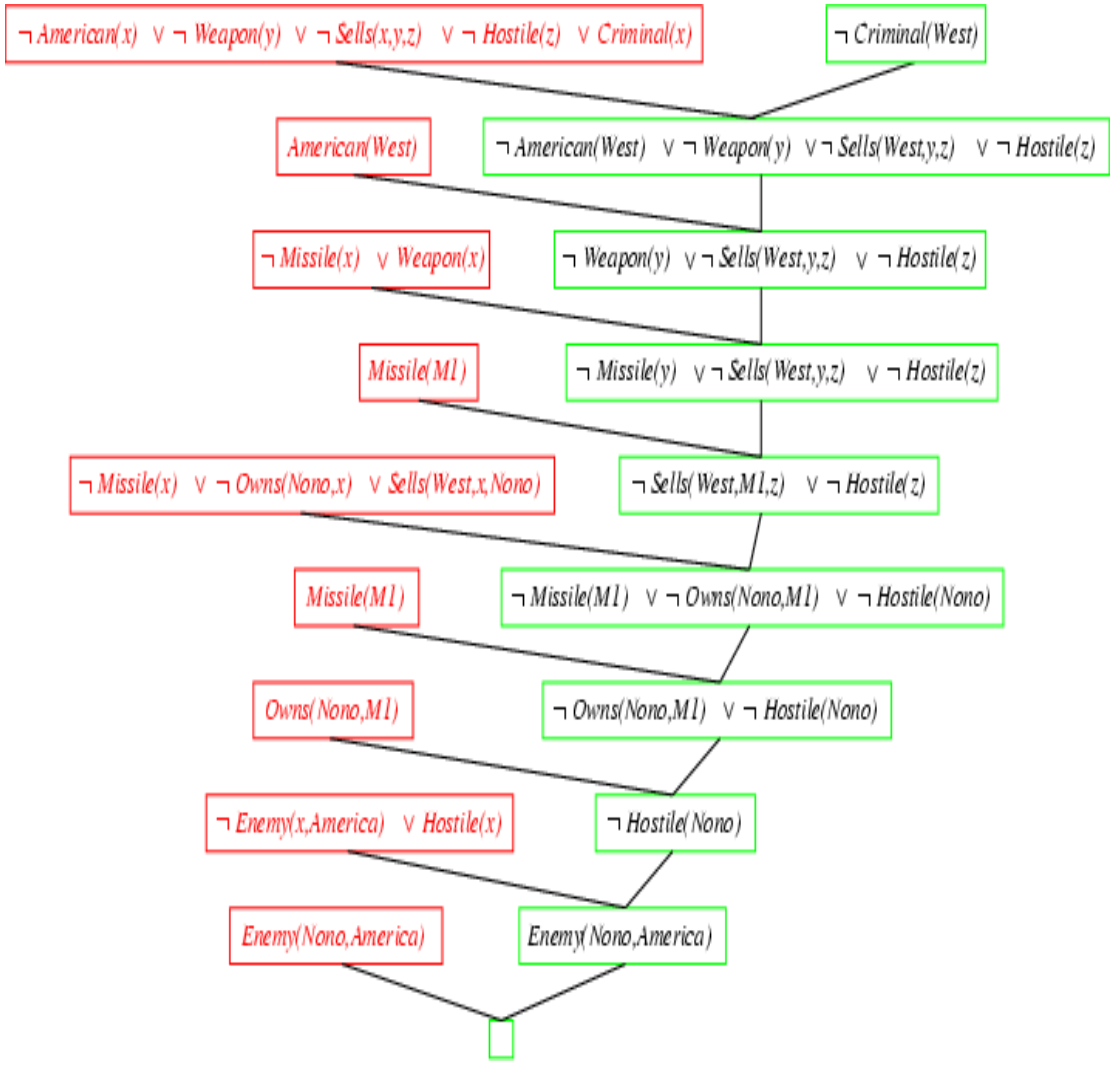
$$\exists i_1,i_2,i_3,o_1,\ o_2\ Signal(In(1,C_1)) = i_1 \wedge Signal(In(2,C_1)) = i_2 \wedge Signal(In(3,C_1)) = i_3 \wedge Signal(Out(1,C_1)) = o^1 \wedge Signal(Out(2,C_1)) = o_2$$

پاسخ: جدول ورودی/خروجی کامل که می تواند برای بررسی مدار استفاده شود.

فصل نهم: استدلال در منطق مرتبه اول

آنچه در این فصل خواهید آموخت:

- **تقلیل استنتاج مرتبه اول به استنتاج گزاره ای**
- **یکسان سازی(Unification)**
- **قانون انتزاع تعمیم یافته(GMP)**
- **زنجیره استنتاج روبه جلو**
- **زنجیره استنتاج روبه عقب**
- **Resolution**



مقدمه:

در این فصل ایده های مربوط به سیستم های استنتاجی منطقی مدرن را معرفی می کنیم . از قوانین استنتاج ساده شروع می کنیم که این قوانین می توانند به جملاتی با انواع سورها اعمال شوند و آنها را به شکل جملات بدون سور تبدیل کنند. این قوانین منجر به این ایده می شود که استنتاج مرتبه اول می تواند با تبدیل پایگاه دانش به جملات منطق گزاره ای و استفاده از منطق گزاره ای انجام شود. (تقلیل به استنتاج گزاره ای) معمولاً استدلال با جملات مرتبه ی اول با استفاده ازResolutionراه حل ناکارآمدتری نسبت به استدلال با عبارات متناهی با استفاده از روبه جلو و زنجیره روبه عقب می باشد .

قانون حذف سور عمومی:

این قانون می گوید که ما می توانیم هر جمله حاصل از جایگزینی متغیر بایک ترم زمینی را استنتاج کنیم . برای نوشتن این قانون استنتاج به تابع جانشینی نیاز داریم . مقدار تابع subst(θ,α) برابر با نتیجه جانشینی θدرجمله α است . لذا قانون حذف سور عمومی به صورت زیر می باشد : به ازای هر متغیر v و جمله ی α , در جمله ی α به جای متغیرv , term زمینی g را جایگزین کن .

نمونه سازی عمومی (UI)

- هر نمونه از یک جمله دارای سور عمومی، توسط آن جمله قابل استلزام است.

$$\frac{\forall v\, \alpha}{\mathrm{Subst}(\{v/g\}, \alpha)}$$

- مثال: $\forall x\ King(x) \wedge Greedy(x) \Rightarrow Evil(x)$

$King(John) \wedge Greedy(John) \Rightarrow Evil(John)$
$King(Richard) \wedge Greedy(Richard) \Rightarrow Evil(Richard)$
$King(Father(John)) \wedge Greedy(Father(John)) \Rightarrow Evil(Father(John))$
.
.
.

: Subst({v/g}, α)

اگر در جمله ای که به فرم منطق مرتبه ی اول است برای متغیر های آن جمله، جایگزینی پیدا کنیم به این عمل جایگزینیاSubstitution گفته می شود که در آنv یک متغیر و α جمله ای به فرم منطق مرتبه اول و g یک ترم زمینی[۱۶۸] می باشد.

[168] ground term

قانون حذف سور وجودی:

قانون حذف سور وجودی کمی پیچیده تر است . برای هر جمله α و متغیر v و سمبل k که تاکنون در پایگاه دانش ظاهر نشده داریم : به ازای برخی متغیر v و جمله ی α , در جمله ی α به جای متغیر v , ثابت k را که تاکنون در پایگاه دانش ظاهر نشده جایگزین کن .

نمونه سازی وجودی (EI)

- برای هر جمله α، متغیر v و سیمبول ثابت k که در جای دیگری از پایگاه دانش ظاهر نشده باشد:

$$\frac{\exists v\, \alpha}{\mathrm{Subst}(\{v/k\}, \alpha)}$$

- مثال: $\exists x\ Crown(x) \wedge OnHead(x, John)$

$$Crown(C_1) \wedge OnHead(C_1, John)$$

به شرطی که C_1 یک سیمبول جدید باشد، که به آن ثابت اسکولم می گویند.

جمله با سور وجودی می گوید که یک یا چند شیء وجود دارد که شرطی را برآورده می کند و فرآیند نمونه سازی (حذف) فقط نامی را برای آن اشیاء انتخاب می کند که این نام جدید یک ثابت اسکولم نامیده می شود . حذف سور وجودی حالت خاصی از یک فرایند کلی به نام skolemization است.

نکته: نمونه سازی یا حذف سورعمومی می تواند چندین بار اعمال شود تا نتایج محتلفی تولید کند ولی نمونه سازی یا حذف سور وجودی فقط یک بار می تواند اعمال شود و سپس جمله دارای سور وجودی حذف می گردد.

تقلیل به استنتاج گزاره ای:

وقتی قوانین برای استنتاج جملات غیر سوری از جملات سوری در اختیار داریم می توانیم استنتاج مرتبه اول را به استنتاج گزاره ای تقلیل دهیم. ایده این است که سور وجودی می تواند با یک بار نمونه سازی جایگزین شود و یک سور عمومی می تواند توسط مجموعه ای از تمام نمونه سازهای ممکن جایگزین شود. به مثال زیر توجه کنید:

تقلیل به استنتاج گزاره ای

- فرض کنید که KB فقط شامل جملات زیر باشد:

$\forall x\ King(x) \wedge Greedy(x) \Rightarrow Evil(x)$
King(John)
Greedy(John)
Brother(Richard,John)

- با نمونه سازی جمله دارای سور عمومی به تمام طرق ممکن، داریم:

King(John) ∧ Greedy(John) ⇒ Evil(John)
King(Richard) ∧ Greedy(Richard) ⇒ Evil(Richard)
King(John)
Greedy(John)
Brother(Richard,John)

- KB جدید گزاره ای سازی شده است. سیمبول های گزاره ای عبارتند از:

King(John), Greedy(John), Evil(John), King(Richard), etc

این تکنیک گزاره سازی نام دارد . بنابراین هر پایگاه دانش در منطق مرتبه اول[۱۶۹] می تواند به گونه ای گزاره سازی شود که استلزام را حفظ کند ، یعنی یک جمله توسط KB جدید قابل استلزام است اگر و فقط اگر توسط KB اصلی قابل استلزام باشد. بنابراین برای تقلیل به استنتاج گزاره ای کافیست پایگاه دانش و پرسش را گزاره سازی کنیم و با استفاده از الگوریتم های استنتاج منطق گزاره ای مانند Resolution و زنجیره روبه جلو یا عقب نتیجه را برگردانیم. یک مسئله در این تکنیک گزاره سازی رخ می دهد و آن این است که وقتی KB شامل سمبل های تابع باشد مجموعه جانشین های ترم زمینی نامتناهی است. به عنوان مثال اگر پایگاه دانش دارای تابع Father باشد،ممکن است جملات تودرتوی نامتناهی مثل Father(Father(Father(John))) ساخته شود. الگوریتم های گزاره ای ما با مجموعه های بزرگ و نامتناهی جملات دچار مشکل می شوند.برای رفع این مشکل از تئوری Herbrond استفاده می کنیم.

تئوری Herbrond:

اگر جمله α توسط پایگاه دانش منطق مرتبه اول قابل استلزام باشد آنگاه این جمله توسط زیر مجموعه ی محدودی از پایگاه دانش گزاره ای سازی شده قابل استلزام است.

تئوری چرچ و تورینگ:

استلزام در منطق مرتبه اول نیمه تصمیم پذیر است . یعنی الگوریتم هایی وجود دارند که جملات استنتاج شده را اثبات می کنند اما الگوریتم هایی وجود ندارند که در مورد جملات غیر استنتاج شده با قطعیت اعلام کنند که آنها واقعا اثبات نمی شوند.

نکته: یکی دیگر از مشکلات گزاره ای سازی نمودن این است که جملات نا مربوط زیادی تولید می گردد،به مثال زیر توجه کنید:

مشکلات گزاره ای سازی نمودن

- به نظر می رسد که گزاره ای سازی کردن جملات نامربوط زیادی تولید می کند. برای مثال از جملات زیر:

$\forall x\ King(x) \wedge Greedy(x) \Rightarrow Evil(x)$

King(John)

$\forall y\ Greedy(y)$

Brother(Richard,John)

- بدیهی است که Evil(John). اما گزاره ای سازی کردن حقایق زیادی مانند Greedy(Richard) تولید می کند که نامربوط می باشند

[169] First Order Logical

قانون انتزاع تعمیم یافته[170] GMP :

به ازای جملات pi,p'i,q یک مجموعه جانشینی θ که subst(θ,pi)=subst(θ,p'i) . برای هر i رابطه ی زیر برقرار است:

Generalized Modus Ponens (GMP)

$$\frac{p_1', p_2', \ldots, p_n', (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{q\theta} \quad \text{where } p_i'\theta = p_i\theta \text{ for all } i$$

p_1' is *King(John)* p_1 is *King(x)*
p_2' is *Greedy(y)* p_2 is *Greedy(x)*
θ is {x/John,y/John} q is *Evil(x)*
q θ is *Evil(John)*

- GMP با پایگاه دانشی از فراکردهای معین (دقیقاً یک لیترال مثبت) کار می کند
- فرض می شود که تمام متغیرها دارای سور عمومی هستند.

به راحتی مـی تـوان نـشان داد کـه GMP یـک قـانون اسـتنتاج صـحیح اسـت.GMP شـکل ارتقـاء یافتـه قانون انتـزاع اسـت . زیـرا قـانون MP در منطـق گـزاره ای بـه منطـق مرتبـه اول ارتقـاء مـی یابـد. امتیـاز مهم قانون استنتاج GMP نسبت به گزاره سـازی ایـن اسـت کـه ایـن قـوانین فقـط جانـشین هـایی را کـه لازم است انجام می دهند .GMP با پایگـاه دانـشی از فراکـرد هـای معـین یعنـی عبـاراتی کـه دقیقـا یـک لیترال مثبت دارند کار می کند.

یکسان سازی[171]:

به عمل جایگزینی متغیر ها به جای متغیرها ،توابع به جای متغیرها و ثابتها به جای متغیرها بطوریکه گزاره های یکسان تولید شود،یکسان سازی گفته می شود. به عبارت دیگر داریم: اگر

- Unify(α,β) = θ if αθ = βθ

p	q	θ
Knows(John,x)	Knows(John,Jane)	{x/Jane}}
Knows(John,x)	Knows(y,OJ)	{x/OJ,y/John}}
Knows(John,x)	Knows(y,Mother(y))	{y/John,x/Mother(John)}}
Knows(John,x)	Knows(x,OJ)	{fail}

استاندارد سازی متغیرها: مثلاً Knows(z, OJ)

[170] General Modus Ponens
[171] Unification

قوانین استنتاج ارتقا یافته به یافتن جانشین هایی نیاز دارند که باعث می شوند عبارات منطقی مختلف یکسان به نظر برسند . این فآیند یکسان سازی نام دارد و جزء کلیدی الگوریتم های استنتاج مرتبه اول می باشد. الگوریتم unify دو جمله را دریافت کرده و یک یکسان ساز θ را در صورت وجود بر می گرداند.مشکل Fail هنگامی اتفاق می افتد که دو جمله از متغیر یکسانی استفاده کنند که با استاندارد سازی یعنی تغییر نام متغیر در یکی از جملات مشکل Fail برطرف خواهد شد.

- برای یکسازی سازی Knows(John, x) و Knows(y, z) دو جانشینی وجود دارد یا

1) θ = {y/John, x/z }
2) θ = {y/John, x/John, z/John}

- اولین یکسان ساز عمومی تر از دومی می باشد
- فقط یک عمومی ترین یکسان ساز (MGU) وجود دارد که منحصر به فرد می باشد.

MGU = { y/John, x/z }

نکته:تعداد یکسان سازها برای دو جمله می تواند صفر،یک یا بیش از یکی باشد.

نکات تکمیلی در مورد یکسان سازی:

- مسندهای غیر همنام قابل یکسان سازی نخواهند بود. **f(x,y)≠g(A,B)**
- دو ثابت غیر همنام قابل یکسان سازی نیستند. **f(A,x)≠f(B,x)**
- در یکسان سازی روی زوج عبارات شامل یک متغیر با همان متغیر یکسان سازی نمی تواند انجام بگیرد. به عنوان مثال در عبارت{ f(x,x) , f(g(x),g(x))} یا { P(g(x),x), P(x,g(x)} اگر بخواهیم جایگزینی {x/g(x)} را بپذیریم باید طبق الگوریتمg(x) را جایگزین x کنیم که این جایگزینی شاملx موجود در خود g(x) نیز می باشد و یک نوع فراخوانی بازگشتی بی نهایت به وجود خواهدآمد. **x/g(g(g(g……)))**

۱-کدام زوج از عبارات زیر قابل یکسان سازی هستند؟

الف) { **P(f(x),y)**, **P(y,f(x))** }

ب) { **P(f(x),x)**, **P(x,x)** }

ج) { **P(x,y)**, **P(y,f(x))** }

د) { **P(f(x),x)**, **P(y,f(y))** }

۲-کدام زوج از عبارات زیر قابل یکسان سازی هستند؟

الف) $\begin{cases} Q(x, g(w,t)) \\ Q(g(A,B),y) \end{cases}$

ب) $\begin{cases} Q(x,g(y,y)) \\ Q(A,g(B,C)) \end{cases}$

ج) $\begin{cases} Q(x,y) \\ Q(g(x,y),B) \end{cases}$

د) $\begin{cases} Q(B,g(x,y) \\ Q(g(x,y),B) \end{cases}$

فراکردهای[172] معین منطق مرتبه اول :

فراکردهای معین،ترکیبات فصلی لیترال هایی است که دقیقاً یکی از آنها مثبت باشد . یک فراکرد معین می تواند به صورت یک جمله اتمی یا یک جمله شرطی که مقدم آن ترکیب عطفی لیترال های مثبت و تالی آن یک لیترال مثبت است نشان داده شود . برخلاف لیترال های گزاره ای، لیترال های مرتبه اول می تواند شامل متغیر نیز باشد . در این مورد متغیرها دارای سور عمومی هستند و سورهای عمومی در فراکردهای معین مرتبه اول حذف می شوند . فراکردهای معین، فرم متعارف برای استفاده از قانون GMP است .به چند نمونه از فراکردهای معین توجه فرمائید.

King(x) ∧Greedy(x) ⇒ Evil(x)
King(John)
Greedy(y)

پایگاه دانش نمونه:

قبل از اینکه به معرفی الگوریتم های استنتاج در منطق مرتبه اول بپردازیم لازم است با یک پایگاه دانش نمونه آشنا شویم و جملات موجود در آن را به منطق مرتبه اول تبدیل کنیم سپس الگوریتم های خود را روی این پایگاه دانش اجرا خواهیم کرد.

«قانون می گوید که این یک جنایت است که یک آمریکایی به ملت های دشمن اسلحه بفروشد. کشور Nono دشمن آمریکا می باشد و دارای تعدادی موشک می باشد و تمام این موشک ها توسط کلنل وست که یک سرهنگ آمریکایی است به این کشور فروخته شده است.»

با توجه به جملات موجود در پایگاه دانش، ثابت کنید که West مجرم است.

- The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American.
-
- Prove that Colonel West is a criminal

[172] Define clause

... برای یک آمریکایی این جنایت است که به ملل دشمن اسلحه بفروشد :

$$American(x) \wedge Weapon(y) \wedge Sells(x,y,z) \wedge Hostile(z) \Rightarrow Criminal(x)$$

کشور نونو دارای تعدادی موشک می باشد ... . توجه کنید :

$$\exists x Owns(Nono,x) \wedge Missle(x) : Missle(M_1), Owns(Nono,M_1)$$

... تمام این موشک ها توسط کلنل وست فروخته شده اند .

$$\forall x Missle(x) \wedge Owns(Nono,x) \Rightarrow Sells(West,x,Nono)$$

موشک ها اسلحه هستند : $Weapon(x) \Rightarrow Missle(x)$

$$Enemy(x,America) \Rightarrow Hostile(x)$$

کشور نونو یک دشمن آمریکا است : $Enemy(Nono,America)$

... it is a crime for an American to sell weapons to hostile nations:

$American(x) \wedge Weapon(y) \wedge Sells(x,y,z) \wedge Hostile(z) \Rightarrow Criminal(x)$

Nono … has some missiles, i.e., ∃x Owns(Nono,x) ∧ Missile(x):

$Owns(Nono,M1)$ and $Missile(M1)$

… all of its missiles were sold to it by Colonel West

$Missile(x) \wedge Owns(Nono,x) \Rightarrow Sells(West,x,Nono)$

Missiles are weapons:

$Missile(x) \Rightarrow Weapon(x)$

An enemy of America counts as "hostile":

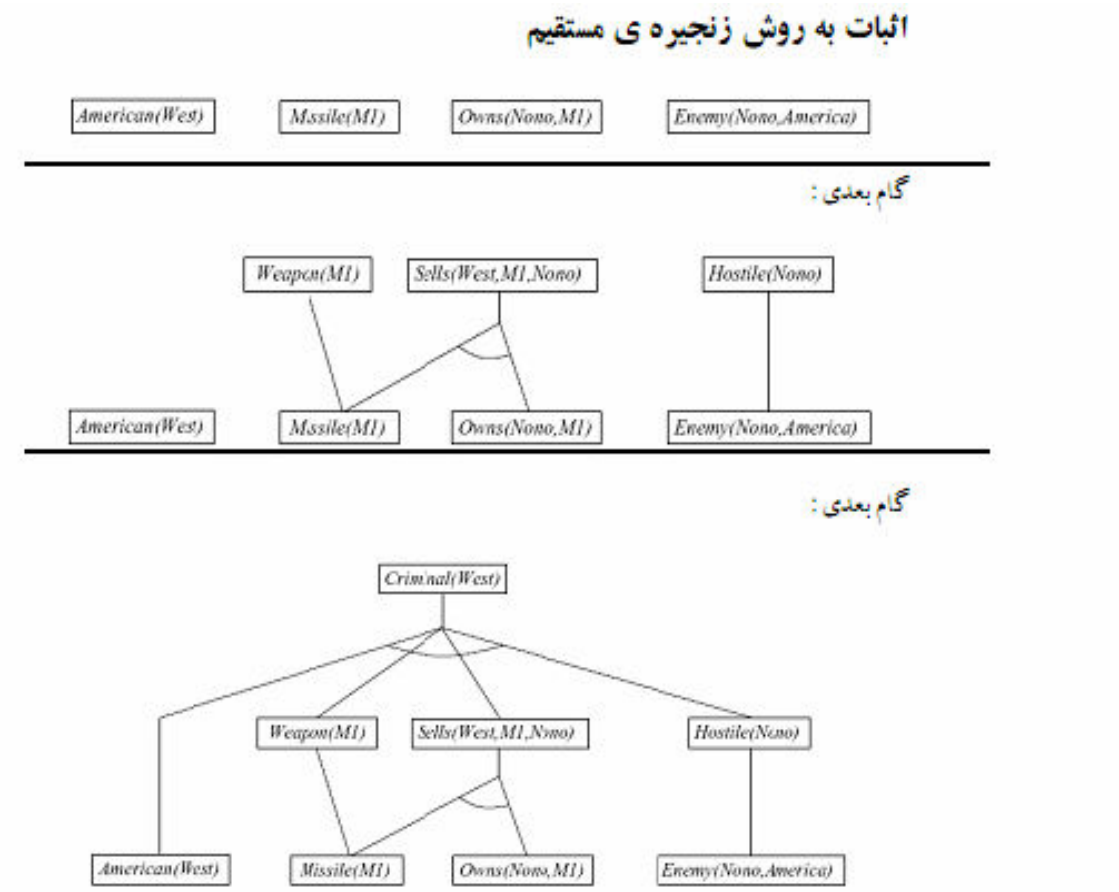
$Enemy(x,America) \Rightarrow Hostile(x)$

West, who is American …

$American(West)$

الگوریتم زنجیره ساز روبه جلو :

این الگوریتم برای فراکردهای معین (کلاز هورن) کار می کند . در این الگوریتم ابتدا ، از جملات اتمی موجود در پایگاه دانش شروع می کنیم . اگر تمام مقدمات یک جمله شرطی درست باشد نتیجه آن جمله شرطی به حقایق موجود در KB اضافه می شود . این فرایند ادامه می یابد تا وقتی که پرسش q به KB اضافه شود یا استنتاج دیگری ممکن نباشد این الگوریتم استنتاج،برای سیستم هایی مفید است که استنتاج هایی در پاسخ به اطلاعات دریافتی جدید انجام می دهند. بنابراین ساخت پایگاه دانش فقط با استفاده از فراکردهای معین ارزشمند است زیرا می توان از هزینه Resolution اجتناب کرد .



پایگاه دانش Datalog :

پایگاه دانشی که شامل مجموعه ای از فراکردهای معین مرتبه اول فاقد سمبل تابع باشد را پایگاه دانشlog Data می گوییم.(مانند پایگاه دانش معرفی شده فوق) . عدم وجود سمبل تابع،عمل استنتاج را آسان تر می کند. الگوریتم زنجیره سازی به جلو برای فراکردهای معین مرتبه اول کامل و صحیح است . همچنین الگوریتم زنجیره ساز به جلو برای Datalog با تعداد تکرارهای محدود متوقف می شود ولی در حالت کلی اگر پایگاه دانش مستلزم جمله α نباشد ممکن است متوقف نشود و این مشکل اجتناب ناپذیر است .

زنجیره ساز به جلو افزایشی :

در این الگوریتم در تکرار t، فقط قوانینی را چک می کنیم که مقدم آن شامل یک عطف مثل P_i باشد که با حقیقت P'_i که در تکرار t-1 استنتاج شده است یکسان سازی شود سپس در مرحله تطبیق قانون، P_i را با P'_i جایگزین کرده و اجازه می دهد که سایر عطف های قانون با حقایقی از تکرارهای قبلی تطبیق یابد که خودعمل تطبیق می تواند پر هزینه باشد. بوسیله شاخص بندی پایگاه دانش اجازه داده می شود که حقایق شناخته شده در زمان O(1) بازیابی شوند . استنتاج روبه جلو به طور گسترده ای در پایگاه داده های استنتاجی بکار گرفته می شود . پس به طور خلاصه زنجیره سازی روبه جلو افزایشی یعنی اینکه در تکرار K ام نیاز به تطبیق قانونی که هیچ کدام از شرایطش در تکرار K-1 اضافه شده است نیست.

الگوریتم زنجیره ساز روبه عقب :

این الگوریتم از هدف به سمت عقب برمی گردد تا به حقایق شناخته شده (جملات اتمیKB) که اثبات جمله هدف را پشتیبانی می کنند برسد . این الگوریتم با لیستی به نام Goals که اهداف در خود نگه می دارد و در ابتدا فقط شامل پرسش یا هدف است فراخوانی می شود . لیست اهداف را می توان به عنوان یک پشته در نظر گرفت که تمام عناصر موجود در لیست Goals باید ارضا شوند تا شاخه جاری با موفقیت اثبات شود . الگوریتم زنجیره ساز روبه عقب،یک الگوریتم اول عمق بازگشتی است و نیازمندی های حافظه برای این الگوریتم بر حسب سایز اثبات جمله خطی است ((O(n)).

زنجیره ساز روبه عقب برخلاف زنجیره سازی روبه جلو از دو مشکل زیر رنج می برد :

الف- **ناکامل بودن :** به دلیل وجود حلقه ها . **راه حل :** مقایسه هدف فعلی با تمام اهداف موجود در پشته

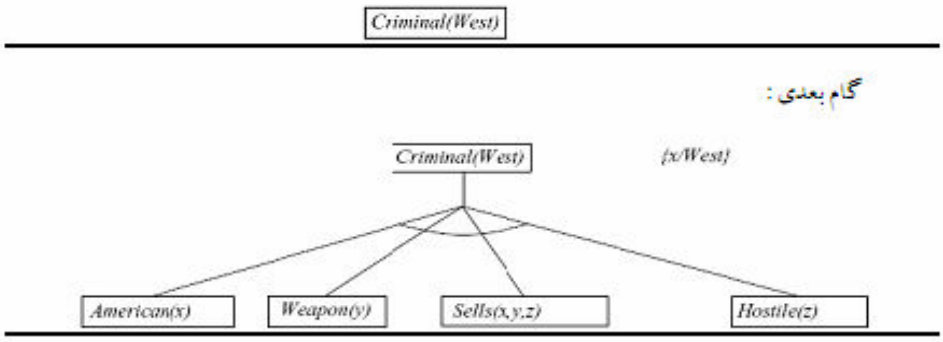
ب- **ناکارا بودن:** به دلیل زیر هدف های تکراری . **راه حل:** ذخیره نتایج قبلی با هدف حافظه بیشتر که نوعی Memorization است انجام می گیرد .

Memorization در سیستم زنجیره سازی روبه عقب یعنی اینکه راه حل های اهداف فرعی[۱۷۳] (هدف اصلی همان پرسش است و اهداف فرعی جملاتی هستند که در لیستGoals قرار می گیرد و باید اثبات شوند) در حافظه کش نگهداری می شوند و با رسیدن مجدد به آن اهداف فرعی به جای تکراری محاسبات قبلی، از همان راه حل های ذخیره شده درکش استفاده می شود . الگوریتم زنجیره ساز روبه عقب، نوعی استدلال هدف گراست که به طور گسترده ای در برنامه نویسی منطقی و زبان پرولوگ بکار گرفته می شود .

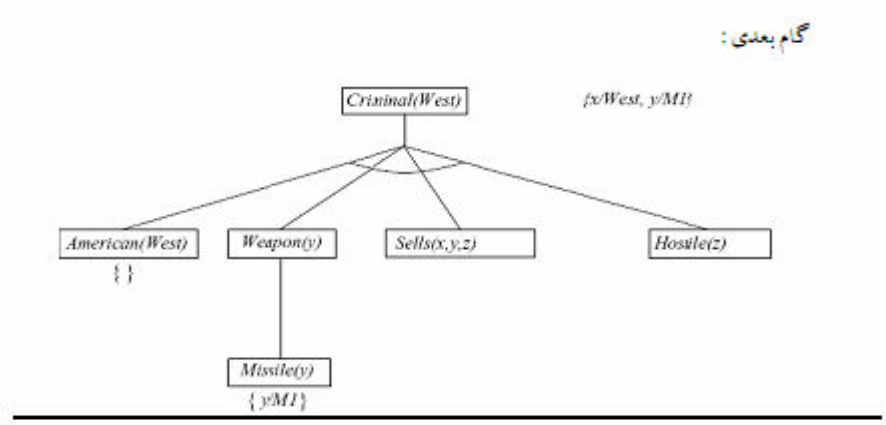
---

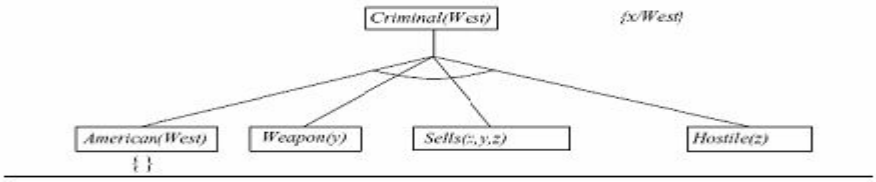
[173] Sub-goal

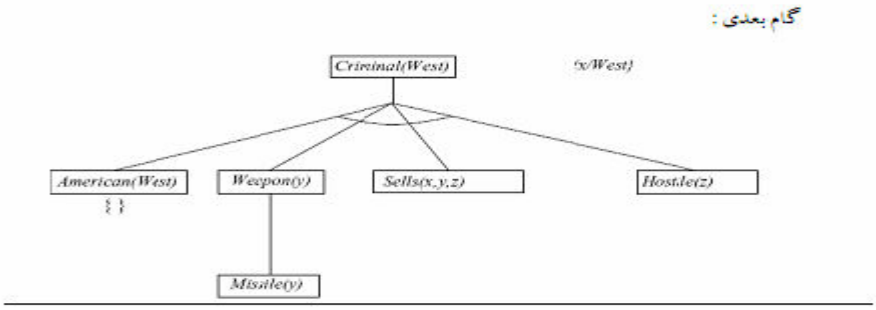
مثال زنجیره ی معکوس

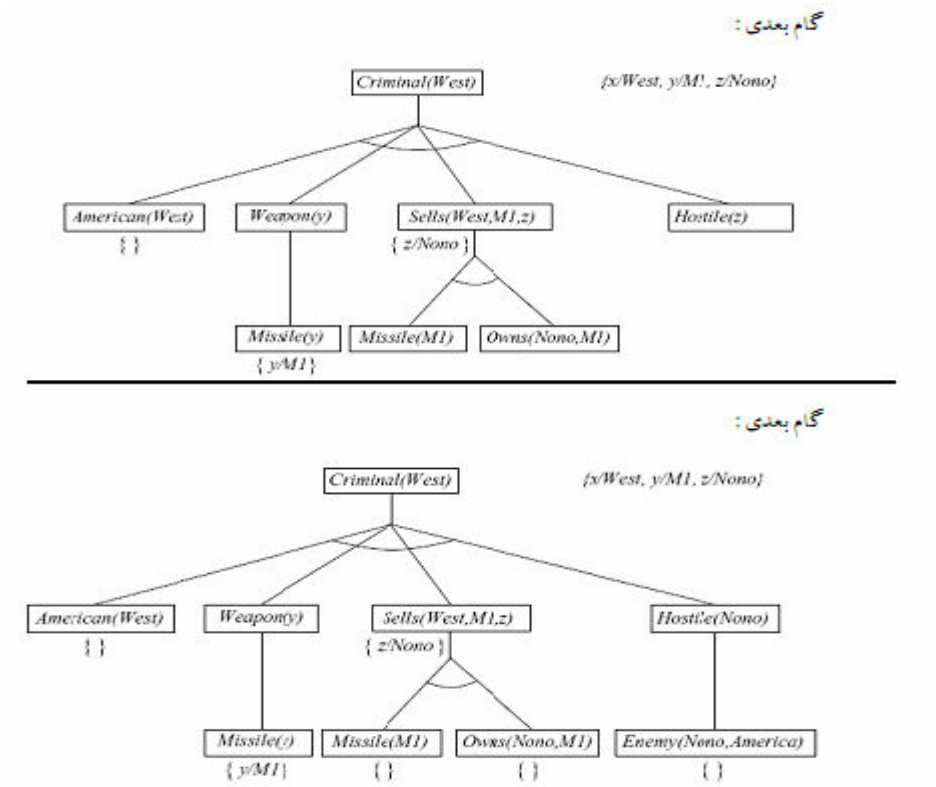


گام بعدی :









برنامه نویسی پرولوگ (منطقی) :

زبان پرولوگ[۱۷۴] متعلق به گروهی از زبان های برنامه نویسی است که زبان برنامه نویسی منطقی نام دارد .این زبان بر مبنای منطق مرتبه اول بنا نهاده شده است . برنامه نویسی منطقی به بازنمایی ایده اعلانی بسیار نزدیک است. ایده اعلانی عبارتست از اینکه سیستم باید با نمایش دانش در یکم زبان رسمی شناخته شود و مسایل آن باید با اجرای فرایندهای استنتاج بر روی آن دانش ، حل شوند. این ایده در معادله Robert Kawalsi به صورت زیراست .در این زبان رابطه زیر برقرار است :

Algorithm = Logic+Control

برنامه های پرولوگ مجموعه ای از فراکردهای معین هستند که با فرم منطق مرتبه اول استاندارد، کمی متفاوت است .

- پرولوگ از حروف بزرگ برای متغیرها و حروف کوچک برای ثابت ها استفاده می کند.
- برای نوشتن یک فراکرد ابتدا رأس و بعد بدنه ی آن نوشته می شود .
- به جای $\Rightarrow$ از $-$: استفاده می شود.
- لیترال های موجود در بدنه با ، از هم جدا می شوند .
- انتهای جمله با . مشخص می شود .

---

[174] Programing in logic

مثال :

American (x) ^ weapons (y) ^ sells (x ,y , z) ^ Hostile (z) ⇒ Criminal
Prolog : Criminal (x) : - american (x) , weapons (y) , sells (x) , Hostile (z) .

پرولوگ برای استنتاج، از روش زنجیره سازی روبه عقب روی فراکردهای Horn استفاده می کند و به طور گسترده ای در ژاپن و امریکا استفاده شده است مخصوصاً در پروژه نسل پنجم کامپیوترها . یکی از قابلیت های قابل توجه در زبان پرولوگ پردازش لیست هاست . لیست دنباله ای مرتبی از عناصر با هر طول متناهی است عناصر می توانند ثابت ، متغیر یا زیر لیست باشند . هر لیست در زبان پرولوگ بین دو علامت [] قرار می گیرد . عنصر اول هر لیست را head و باقیمانده را tail می گویند .

برنامه نویسی منطقی: پرولوگ

- Algorithm = Logic + Control
- اصول: عقبگرد با فراکردهای هورن- به طور گسترده ای در اروپا و ژاپن استفاده شده است (در پروژه نسل پنجم کامپیوترها)
- برنامه = مجموعه ای از فراکردها
- فراکرد =

```
head :- literal1, ... literaln.
```

- مثال:

```
criminal(X) :- american(X), weapon(Y), sells(X,Y,Z), hostile(Z).
```

- اضافه کردن یک لیست به انتهای لیست دیگر برای تولید یک لیست جدید (عمل Append)

```
append([],Y,Y).
append([X|L],Y,[X|Z]) :- append(L,Y,Z).
```

- کوئری:

```
append(A,B,[1,2]) ?
```

- پاسخ ها:

```
A=[]      B=[1,2]

A=[1]     B=[2]

A=[1,2]   B=[]
```

مثال های پرولوگ

- جستجوی اول - عمق از یک حالت شروع X :

```
dfs(X) :- goal(X) .
dfs(X): - successor(X,S),dfs(S).
```

نیازی به حلقه روی S وجود ندارد : successor برای همه موفق است .

اضافه نمودن دو لیست برای تولید کردن سومی :

```
append([],Y,Y).
append([X|L],Y,[X|Z]) : - append(L,Y,Z).
```

سوال : ? append(A,B,[1,2]) - جواب ها :

A=[] B=[1,2]

A=[1] B=[2]

A=[1,2] B=[]

نکاتی دیگر در مورد پرولوگ:

- هر برنامه شامل دنباله ای از جملات است که به طور ضمنی به هم عطف شده اند .
- تمامی متغیرها به طور ضمنی دارای سور عمومی هستند .
- تمام استنتاج ها به طور زنجیره سازی روبه عقب با الگوریتم جستجوی اول عمق صورت می گیرد یعنی در صورت رسیدن به بن بست برای اثبات یک جمله پرولوگ، به آخرین مرحله ما قبل برگشته و تلاش می کند راه حل های دیگر را آزمایش کند به این کار Back traking گفته می شود .

دستور Cut :

یکی از دستورات بسیار مهم در زبان پرولوگ دستور cut یا برش است . این دستور به زبان پرولوگ اجازه می دهد که مکانیزم Back traking را از محل وقوع برش متوقف کرده تا زمان جستجو کاهش پیدا کند .

دو دلیل اصلی استفاده از دستور برش عبارتند از :

- برنامه شما سریع تر عمل خواهد کرد چون زیرهدف های متعددی را بررسی نخواهد کرد .
- برنامه شما حافظه کمتری مصرف خواهد کرد .

موارد استفاده از دستور Cut :

- زمانی که می خواهیم به پرولوگ بگوییم که قانون درست را برای هدف معین یافته است . بنابراین cutمی گوید که اگر این مسیر را ادامه دهی به هدف درست می رسی و نیاز به بازگشت به عقب نمی باشد .
- زمانیکه می خواهیم به پرولوگ بگوییم ادامه دادن برای تطبیق با هدف بی فایده است و دیگر این مسیر را ادامه ندهداز این دستور استفاده می شود. .

- زمانی که می خواهیم پاسخ های بعدی ممکن را تولید نکنیم دستور cut به پرولوگ می گوید که دیگر به دنبال پاسخ بعدی نباشد .

به مثال زیر توجه فرمایید :

P:-a,!b. $a \wedge b \rightarrow P$

P:-c. $c \rightarrow P$

تحلیل[۱۷۵] :

تحلیل برای جملات مرتبه اول، نسخه ارتقا یافته تحلیل در منطق گزاره ای است . در منطق گزاره ای دو لیترال مکمل یکدیگر هستنداگر یکی نقیض دیگری باشد اما در منطق مرتبه اول دو لیترال مکمل یکدیگرنداگر یکی از آنها نقیض دیگری یکسان ساز شود .قانون Resolution در منطق مرتبه اول به شکل زیر می باشد:

- رزولوشن در منطق مرتبه اول

$$\frac{l_1 \vee \cdots \vee l_k, \qquad m_1 \vee \cdots \vee m_n}{(l_1 \vee \cdots \vee l_{i-1} \vee l_{i+1} \vee \cdots \vee l_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n)\theta}$$

که $\mathrm{Unify}(l_i, \neg m_j) = \theta$

برای مثال ،

$$\frac{\neg Rich(x) \vee Unhappy(x) \quad Rich(Ken)}{Unhappy(Ken)} \qquad \text{with } \theta = \{x/Ken\}$$

برای استفاده از Resolution در منطق مرتبه اول لازم است جملات به فرم نرمال عطفی CNF تبدیل شوند . فرم نرمال عطفی، شامل ترکیبات فصلی از لیترال هاست .

مراحل تبدیل به CNF :

- حذف ترکیب شرطی
- انتقال ~ به داخل عبارات
- استاندارد کردن متغیرها (تغییر نام)
- فرایندحذف سور وجودی[۱۷۶]

---

[175] Resolution
[176] Skolemization

- حذف سور عمومی و انتقال سورها به سمت چپ ترین
- توزیع ^ روی v

تبدیل به صورت نرمال CNF - هر کس که همه ی حیوانات را دوست می دارد کسی او را دوست می دارد :

$$\forall x[\forall y Animal(y) \Rightarrow Loves(x,y)] \Rightarrow [\exists y Loves(y,x)]$$

۱. استلزام[1] ها را حذف کنید

$$\forall x[\neg\forall y \neg Animal(y) \vee Loves(x,y)] \vee [\exists y Loves(y,x)]$$

۲. ¬ را به درون حرکت دهید :

$$\neg\forall x, p \equiv \exists x \neg p, \neg\exists x, p \equiv \forall x \neg p :$$

$$\forall x[\exists y \neg(\neg Animal(y) \vee Loves(x,y))] \vee [\exists y Loves(y,x)]$$

$$\forall x[\exists y \neg\neg Animal(y) \wedge \neg Loves(x,y)] \vee [\exists y Loves(y,x)]$$

$$\forall x[\exists y Animal(y) \wedge \neg Loves(x,y)] \vee [\exists y Loves(y,x)]$$

۳. متغیر های استاندارد : هر کمیت ، باید از یک متغیر جدا استفاده نماید ؛

$$\forall x[\exists y Animal(y) \wedge \neg Loves(x,y)] \vee [\exists z Loves(z,x)]$$

۴. عبارت های وجودی را اسکلمایز نمایید

$$\forall x[Animal(F(x)) \wedge \neg Loves(x,F(x))] \vee Loves(G(x),x)$$

۵. سور های عمومی را رها نمایید :

$$[Animal(F(x)) \wedge \neg Loves(x,F(x))] \vee Loves(G(x),x)$$

۶. ∧ را روی ∨ توزیع نمایید :

$$[Animal(F(x)) \vee Loves(G(x),x)] \wedge [\neg Loves(x,F(x)) \vee Loves(G(x),x)]$$

استاندارد سازی متغییرها:

استاندارد سازی به این معناست که هر سور باید از یک متغیر استفاده کند و یک سور نمی تواند بیش از یک متغیر استفاده کنند . بنابراین برای جملاتی مثل [∀ x, p(x)] v [∃ x , p(x)] که دو سور از متغیرهای همنام استفاده می کنند باید نام یکی از متغیرها را تغییر نام بدهیم . بدین ترتیب وقتی بعدا سورها حذف می شوند تناقضی پیش نخواهد آمد . تغییر نام های مشابه با توجه به حوزه عملکرد سورها صورت می گیرد هر سور یک حوزه عملکرد دارد که با) یا]شروع و با (یا [پایان می یابد .

$$\forall x\ [\exists y\ Animal(y) \wedge \neg Loves(x,y)] \vee [\exists z\ Loves(z,x)]$$

نکته :

اگر متغیری در حوزه سور عمومی یا وجودی باشد محدود شده و اگر درحوزه هیچ سوری نباشد متغیر آزاد[۱۷۷] نام دارد.

مثال :x موجود در q آزاد و x موجود در p محدود شده است.

$\exists x[p(x,y)] \Rightarrow q(x)$

فرایند حذف سور وجودی(اسکولمایز) :

فرض کنید $\exists x , w(x)$ یک عبارت خوش فرم یا بخش بزرگتری از عبارت خوش فرم باشد .

- اگر $\exists x$ در حوزه هیچ سور عمومی نباشد مقدار ثابتی مثل C را انتخاب و به جای x در $w(x)$ قرار میدهیم .
- اگر x داخل حوزه سورهای عمومی $\exists[\forall_{x1}\forall_{x2}.....\forall_{xn}]$ باشد تابعی همانند F را انتخاب کرده و حاصل به صورت $F(x_1,.....xn)$ w خواهد بود .

به چند مثال در این زمینه توجه فرمائید.

$\exists x,p(x,y)$
$\forall x, \exists y\ p(x,y)$
$\exists x\forall y\forall z\exists u\forall v\exists w\ p(x,y,z,u,v,w)$

به چند نمونه تست در این زمینه توجه فرمائید:

۱-کدام گزینه بیانگر فرم clause (CNF) جمله مقابل است ؟ (مهندسی کامپیوتر- ۸۱)

$[\forall x, P(x) \rightarrow Q(x)] \rightarrow R(a)$

۲-کدام گزینه نقیض عبارت مقابل است ؟ (مهندسی کامپیوتر- ۸۱)

$\exists y\forall x\forall z[P(x) \wedge Q(x,y) \rightarrow R(x,y,z)]$

۳-کدام یک از موارد زیر فرم clause جمله مقابل است ؟ (IT-۸۴)

$[\forall x, S(x,R) \rightarrow L(x,G)] \rightarrow H(R)$

[177] free

مثال در مورد Resolution :

فرض کنید KB زیر را داریم :

الف) هر سگی یا انسانی را دوست دارد یا از گربه متنفر است .

ب) Rover یک سگ است .

ج) Rover گربه ها را دوست دارد .

حکم : اثبات کنید بعضی از سگها هستند که گربه ها را دوست دارند .

H(x) : متنفر بودن از گربه ها توسط x.

D(x) : معرف سگ بودن x.

L(x) : معرف دوست داشتن فردی توسط x .

الف) $\forall x\ D(x) \longrightarrow L(x) \vee H(x)$

ب) $D(Rover)$

ج) $\sim H(Rover)$

حکم: $\exists x\ (D(x) \wedge L(x))$

CNF موارد بالا :

الف) $\sim D(x) \vee L(x) \vee H(x)$

ب) $D(Rover)$

ج) $\sim H(Rover)$

حکم : $D(y) \wedge L(y)$

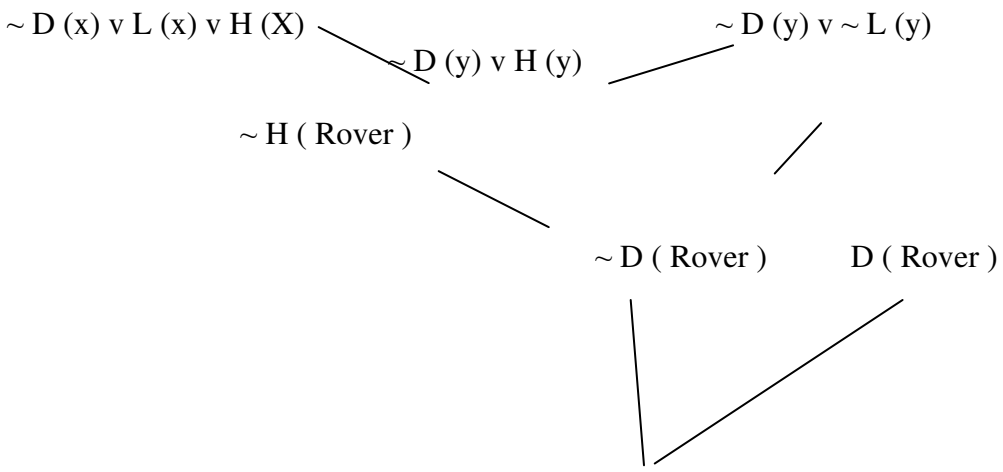
نقیض حکم : $\sim D(y) \vee \sim L(y)$

الگوریتم Resolution :

۱- ابتدا فرضیات و نقیض حکم را به صورت گزاره های منطق مرتبه اول بیان می کنیم .

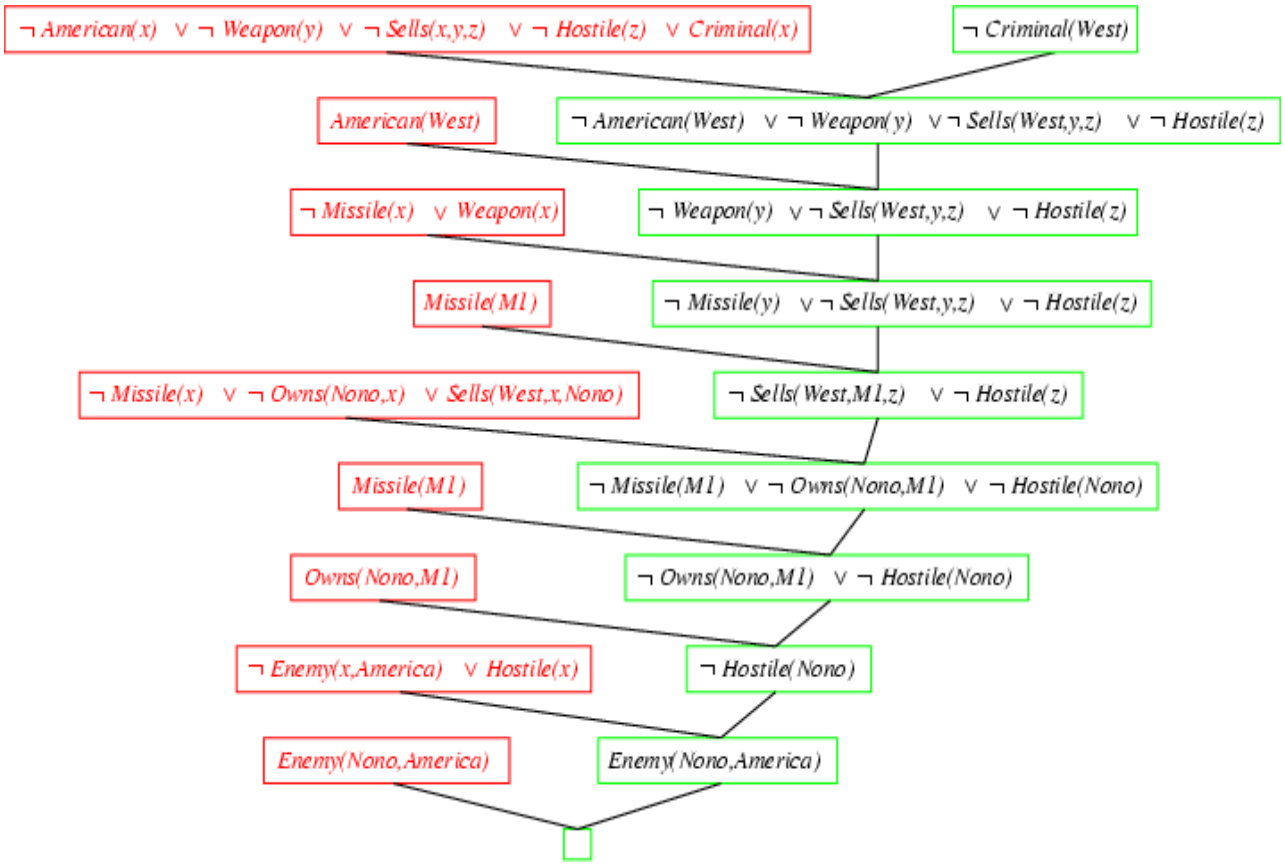
۲- فرضیات و نقیض حکم را با روش تبدیل گزاره به CNF, به CNF های معادل تبدیل می کنیم .

۳- قانون Resolution را به این مجموعه تا زمانی اعمال می کنیم تا به تناقض برسیم .



به تناقض رسیدیم

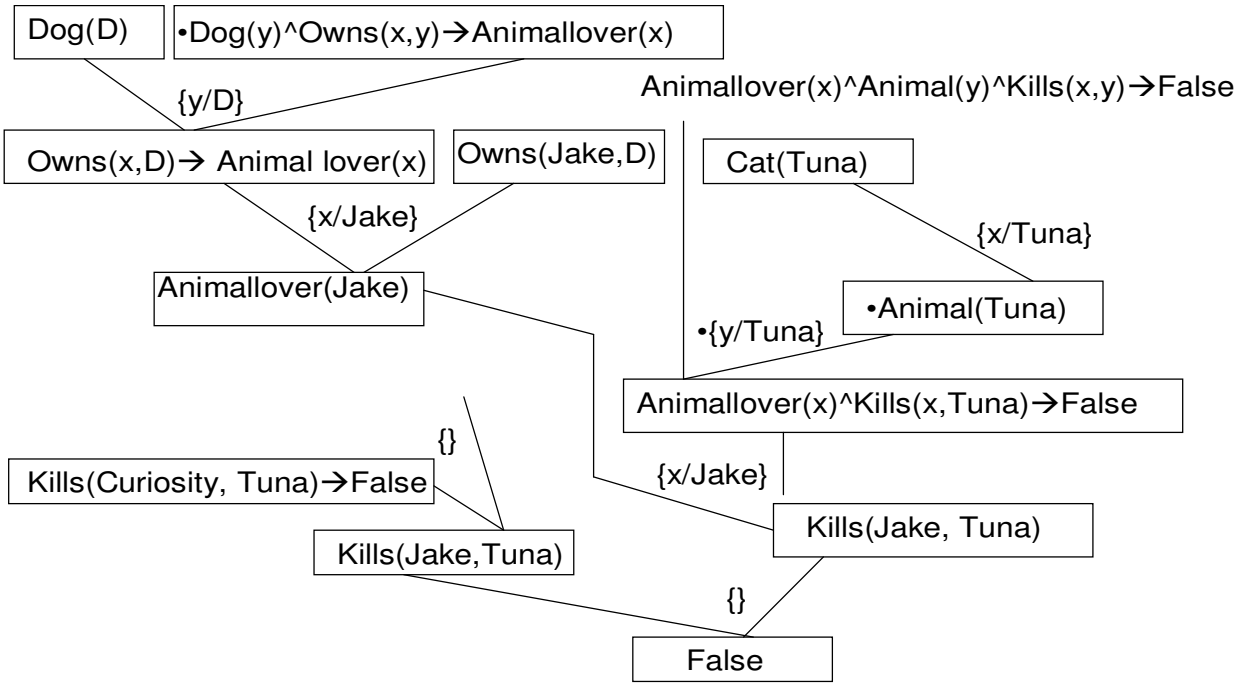
اثبات مجرم بودن West توسط Resolution:



مثال jake,Tuna به وسیله Resolution:

- Jake owns a dog .
- Every dog owner is an animal lover .
- No animal lover kills an animal .
- Either Jake or Curiosity killed the cat , who is named Tuna .
- Did Curiosity kill the cat ?

- A. ∃x Dog (x) ^ Owns (Jake ,x)
- B. ∀x [∃y Dog(y) ^ Owns(x,y)→Animal lover(x)]
- C. ∀x Animallover(x)→∀x Animal(y)→~Kills(x,y)
- D. Kills(Jake, Tuna) v Kills(Curiosity, Tuna)
- E. Cat (Tuna)
- F. ∀x Cat(x) → Animal(x)
- A1. Dog(D)
- A2. Owns(Jake,D)
- B. Dog(y) ^ Owns(x,y)→ Animal lover(x)
- C. Animal lover(x) ^ Animal(y) ^ Kills(x,y)→False
- D. Kills(Jake, Tuna) v Kills(Curiosity, Tuna)
- E. Cat(Tuna)
- F. Cat(x) → Animal(x)



استراتژی های تحلیل :

کاربرد پشت سرهم قانون Resolution در صورت وجود اثباتی را خواهد یافت اما کارایی آن ممکن است خوب نباشد. برای بهبودی کارایی این روش، استراتژی های زیر پیشنهاد می گردد :

- **اولویت(ترجیح) واحد[۱۷۸]:**

این روش ترجیح می دهد هنگامی از قانون Resolution استفاده شود که یکی از جملات فقط یک لیترال(جملاتی که فقط یک لیترال دارند Unit clause نامیده می شوند) باشد.ایده موجود در این روش این است که ما سعی داریم فراکرد خالی ایجاد کنیم لذا بهتر است به استنتاج هایی اولویت داده شود که عبارات کوچکتری دارند.بکارگیری Resolution، برای یک جمله مانند P و هر جمله دیگری مانند P v Q v R همیشه منجر به تولید فراکرد کوچکتری می شود. اولویت واحد،فرم محدودی از Resolution است که در آن، هر مرحله از Resolution بایدحاوی یک عبارت واحد(Unit clause)باشد.این استراتژی در حالت کلی کامل نیست اما برای پایگاه دانش به فرم هورن کامل است.

- **مجموعه پشتیبان[۱۷۹] :**

تقدم هایی که در استفاده ازResolutionترتیب قایل می شوند،مفید هستند اما در حالت کلی بهتر است بعضی از Resolution ها حذف شوند. روش مجموعه پشتیبان این کار را انجام می دهد.این روش با شناسایی زیر مجموعه ای از جملات به نام مجموعه پشتیبان شروع می کند. هر Resolution،جمله ای از مجموعه پشتیبان را با جمله دیگر آن، ترکیب می کند و نتیجه را به مجموعه پشتیبان اضافه می کند اگر مجموعه پشتیبان نسبت به کل پایگاه دانش کوچک باشد فضای جستجو بسیار کوچک خواهد شد . در این شیوه،یک انتخاب نامناسب برای مجموعه پشتیبان، الگوریتم را ناکامل خواهد کرد. استراتژی مجموعه پشتیبان دارای این مزیت است که درخت های اثباتی را تولید می کند که اغلب برای درک افراد آسان می باشد زیرا این روش هدف گرا می باشد.

- **تحلیل ورودی[۱۸۰] :**

در این استراتژی، هر Resolution، یکی از جملات ورودی (از KB یا پرسش) را با یک جمله دیگر ترکیب می کند. اثبات مجرم بودن West که داراری شکل محوری است از این روش استفاده می کند. در پایگاههای دانش هورن، قانون Modus Ponens (انتزاع) نوعی از استراتژی تحلیل ورودی است زیرا یک ترکیب شرطی از KB اصلی را با دیگر جملات ترکیب می کند .بنابراین تعجبی ندارد که تحلیل ورودی برای پایگاههای دانشی که به صورت هورن هستند کامل باشد اما در حالت کلی ناکامل است .

---

[178] unit preference
[179] set of support
[180] Input Resolution

- **شمول[181]:**

این روش تمام جملاتی را که حالت خاصی از یک جمله موجود در KB هستند، از KB حذف می کند.به عنوان مثال،اگر جمله P(x) در KB موجود باشد،نیاز به افزودن P(A) نیست. این روش به نگهداری KB، به صورت کوچک کمک می کند و در نتیجه فضای جستجو را کاهش می دهد.

[181] subsumption