# فصل دهم: یادگیری دسته قوانین

یکی از راحت‌ترین و قابل‌درک‌ترین فرم بیان فرضیه‌ها دسته قوانین if-then است. این فصل چندین الگوریتم برای یادگیری این دسته قوانین را بیان و بررسی می‌کند. یکی از مهم‌ترین حالات مسئله در یادگیری دسته قوانین با متغیرها horn clause درجه اول[۱] نامیده می‌شود، زیرا که دسته قوانین horn clase درجه اول را می‌توان به عنوان برنامه به زبان برنامه‌نویسی منطقی Prolog داد، یادگیری این دسته قوانین گاهی برنامه‌نویسی استقرایی منطقی[۲] (ILG) نیز نامیده می‌شود. این فصل چندین روش مختلف برای یادگیری دسته قوانین، شامل روش مبتنی بر وارون کردن عملگرهای نتیجه‌گیری[۳] ثابت‌کننده‌ی تئوری[۴] را بررسی می‌کند.

## ۱۰,۱ معرفی

در بسیاری از شرایط یادگیری تابع هدف استفاده از دسته قوانین if-then که با هم تابع یادگرفته شده را نشان می‌دهند بسیار مفید است. همان‌طور که در فصل ۳ نیز نشان داده شد، یکی از راه‌های یادگیری قوانین ساخت درخت تصمیم و تبدیل آن به دسته قوانین هم‌ارز است، به ازای هر برگ درخت یک قانون ساخته خواهد شد. روش دوم که در فصل ۹ نشان داده شد، استفاده از الگوریتم ژنتیک است، در این روش رشته بیت‌هایی متناسب با دسته قوانین ایجاد شده و از الگوریتم‌های ژنتیک برای جستجوی فضای فرضیه‌ای تعریفی استفاده می‌شود. در این فصل ما الگوریتم‌های مختلفی که مستقیماً دسته قوانین را یاد می‌گیرند و با الگوریتم‌های بالا در دو نکته کلیدی متفاوت‌اند را بررسی می‌کنیم. ابتدا اینکه این الگوریتم‌ها برای یادگیری دسته قوانین درجه اولی که متغیر دارند طراحی شده‌اند، اهمیت این نکته در قدرت بیان این قوانین نسبت به دسته

---

[۱] first-order horn clase
[۲] inductive logic programming
[۳] deductive operators
[۴] mechanical theorem provers

قوانین گزاره‌ای است. دوم اینکه الگوریتم‌های مطرح شده الگوریتم‌های ترتیبی[۱] هستند که قوانین را به ترتیب و برای کامل کردن دسته قوانین موجود یاد می‌گیرند.

برای مثال، دسته قوانین درجه اول زیر را که باهم مفهوم هدف Ancestor (جَد) را نشان می‌دهند در نظر بگیرید.

IF Parent(x,y) THEN Ancestor(x,y)

IF Parent(x,z)/\Ancestor(z,y) THEN Ancestor(x,y)

توجه داشته باشید که قوانین بالا تابعی بازگشتی را که به سختی می‌توان با درخت تصمیم‌گیری یا نمایش‌های گزاره‌ای[۲] نشان داد را نمایش می‌دهد. یکی از راه‌های پی بردن به قدرت نمایشی قوانین درجه اول توجه به هدف کلی زبان برنامه‌نویسی Prolog است. در Prolog برنامه‌ها دسته قوانین درجه اول، مشابه دو قانونی که در بالا ذکر شد، هستند (این نوع قوانین گاهی Horn clause نامیده می‌شوند). در واقع، قوانین بالا برنامه‌ای در زبان Prolog است که رابطه‌ی Ancestor را تشخیص می‌دهد. در این سبک، الگوریتم کلی ی یادگیری این گونه دسته قوانین را می‌توان برنامه‌نویسی خودکار Prolog از نمونه‌های آموزشی دانست. در این فصل چنین الگوریتم‌هایی که دسته قوانین را از نمونه‌های آموزشی یاد می‌گیرند بررسی خواهیم کرد.

در عمل، سیستم‌های یادگیری که از دسته قوانین درجه اول استفاده می‌کنند در مسائلی چون یادگیری قوانین شیمی در طیف‌سنج جرمی (Buchanan 1976; Lindsay 1980)، یادگیری ساختارهای جهش ژنتیکی (در رابطه با سرطان) (Srinivasan 1994) و یادگیری طراحی عناصر متناهی برای بررسی استرس در ساختارهای فیزیکی (Dolsak and Muggleton 1992) به کار رفته‌اند. در هر یک از این کاربردها، فرضیه‌ای که معرفی می‌شود، ادعاهایی است که به راحتی توسط دسته قوانین درجه اول بیان می‌شوند، و توصیف این ادعاها در بیان‌های گزاره‌ای بسیار سخت است.

در این فصل، ابتدا از الگوریتم‌هایی شروع خواهیم کرد که دسته قوانین گزاره‌ای را یاد می‌گیرند؛ دسته قوانین گزاره‌ای دسته قوانینی هستند که متغیر ندارند. الگوریتم‌هایی که برای جستجوی فضای فرضیه‌ای ی دسته قوانین فصلی در چنین شرایطی قابل‌درک‌اند. سپس الگوریتم‌هایی که دسته قوانین درجه اول یاد می‌گیرند را بررسی خواهیم کرد. در ادامه نیز دو روش کلی برای استقرا در برنامه‌نویسی منطقی و روابط اساسی بین استنباط استقرایی و استنتاجی را بررسی خواهیم کرد.

## ۱۰,۲ الگوریتم‌های ترتیبی

در این بخش خانواده‌ای از الگوریتم‌ها که دسته قوانین را بر اساس استراتژی یادگیری یک قانون و حذف داده‌های سازگار با آن قانون یاد می‌گیرند بررسی خواهیم کرد. چنین الگوریتم‌هایی، الگوریتم‌های ترتیبی نامیده می‌شوند. توضیح بیشتر اینکه، فرض کنید که یک زیر روال[۳] به نام Learn-one-rule داریم که دسته‌ای از نمونه‌های مثبت و منفی را دریافت کرده و قانونی را خروجی می‌دهد که اکثر نمونه‌های مثبت و تعداد بسیار کمی از نمونه‌های منفی را می‌پوشاند. می‌خواهیم که دقت این قانون خروجی حداکثر شود، لازم نیست پوشش قانون خروجی زیاد

---

[۱] sequential covering algorithms

[۲] propositional representation

[۳] subroutine

باشد. منظورمان از افزایش دقت قانون افزایش تعداد دسته‌بندی‌های درست آن است. با پذیرفتن پوشش کم، منظورمان این است که لازم نیست که قانون برای همه‌ی نمونه‌های آموزشی پیش‌بینی‌ای داشته باشد.

با داشتن زیر روال Learn-one-rule، یکی از ساده‌ترین راه‌ها، دادن همه‌ی نمونه‌های آموزشی موجود به زیر روال و حذف تمامی نمونه‌های مثبتی تحت پوشش قانون خروجی و تکرار این روال با بقیه‌ی نمونه‌هاست. این فرایند را می‌تواند آن‌قدر ادامه داد تا فصلی از قوانین که با هم تمامی نمونه‌های مثبت را می‌پوشانند به دست آید. در آخر نیز می‌توان قوانین را بررسی کرد تا هنگام دسته‌بندی نمونه‌های جدید اول قوانینی بررسی شوند که حداکثر دقت را دارند. یک حالت کلی از الگوریتم‌های ترتیبی در جدول ۱۰,۱ آورده شده است.

---

Sequential-Covering(Target_attribute,Attributes,Examples,Threshold)

- Learned_rules←{}
- Rule←Learn-One-Rule(Target_attribute,Attributes,Examples)
- تا زمانی که شرط Performance(Rule,Examples)>Threshold است حلقه‌ی زیر را ادامه بده
  - Learned_rules←Learned_rules+Rule
  - Example←Examples-{نمونه‌هایی که به درست توسط Rule دسته‌بندی می‌شوند}
  - Rule←Learn-one-rule(Target_attribute,Attributes,Examples)
- مرتب شده‌ی Learn_rules بر اساس Performance بر Learned_rules← Examples Learn_rules را برگردان

---

*جدول ۱۰,۱ الگوریتم ترتیبی برای یادگرفتن دسته قوانین فصلی.*
*Learn-one-rule قانونی را بر می‌گرداند که حداقل تعدادی از نمونه‌های Examples را بپوشاند. Performance زیر روالی است که کیفیت قانون را بررسی می‌کند. این الگوریتم تا زمانی که Performance کمتر از مقدار Threshold است به یادگیری قوانین ادامه خواهد داد.*

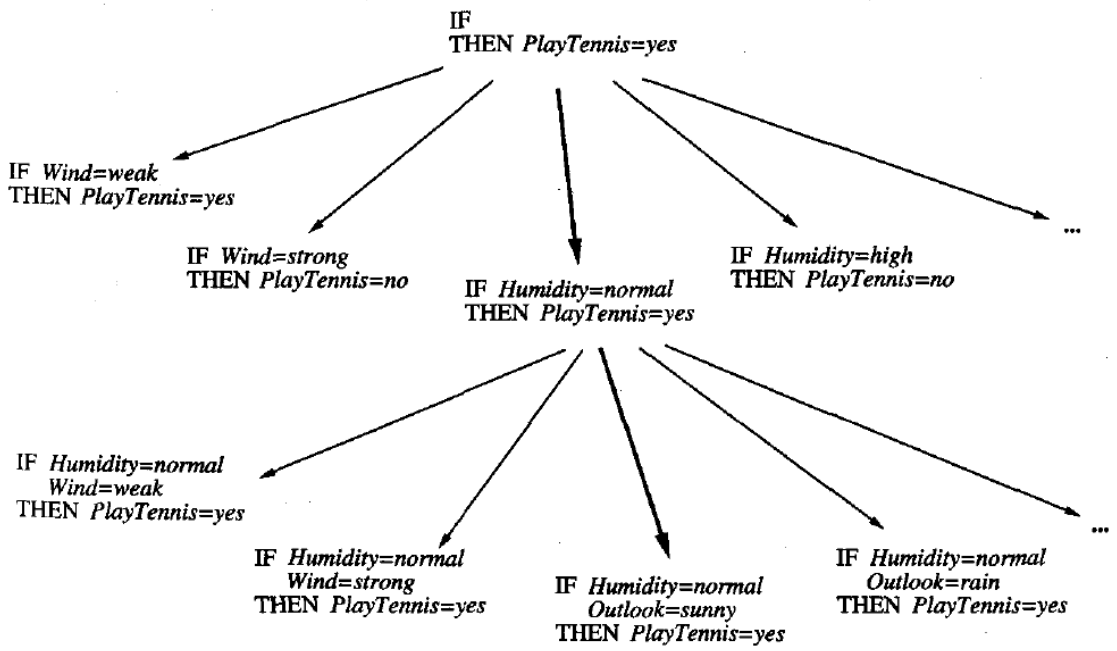
این الگوریتم ترتیبی یکی از متداول‌ترین روش‌های یادگیری دسته قوانین فصلی است. این روش مسئله‌ی یادگیری دسته قوانین فصلی را به سری‌ای از مسائل ساده‌تر تبدیل می‌کند، در هر یک از این مسائل ساده‌تر یک قانون عطفی یاد گرفته می‌شود. چون این روش، جستجویی حریصانه انجام می‌دهد، در یادگیری قوانین بدون مراجعه به مراحل قبلی تضمینی نیست که کوچک‌ترین دسته قوانین یا بهترین دسته قوانینی که نمونه‌های آموزشی را می‌پوشاند را پیدا کنیم.

Learn-one-rule را چگونه باید طراحی کنیم تا نیازمان در الگوریتم ترتیبی برطرف کند. الگوریتمی می‌خواهیم که قانونی را ایجاد کند که دقت بالایی داشته باشد، اما لازم نیست تمام نمونه‌های مثبت را بپوشاند. در این بخش الگوریتم‌های متفاوتی را به همراه تفاوت‌های مهمشان بررسی خواهیم کرد. در این بخش فقط به روش‌های گزاره‌ای می‌پردازیم و تعمیم این روش‌ها به horn clause های درجه اول را به بخش بعدی موکول می‌کنیم.

### ۱۰,۲,۱ جستجوی ستونی کلی به جزئی

یکی از راه‌های مؤثر تعریف Learn-one-rule سازمان‌دهی فضای فرضیه‌ها در همان شکل کلی الگوریتم ID3 است، با این تفاوت که در اینجا ما امیدوارکننده‌ترین شاخه‌ی درخت را در هر مرحله بررسی خواهیم کرد. همان‌طور که در درخت جستجوی شکل ۱۰,۱ نیز آمده است، جستجو قانونی شروع می‌شود که کلی‌ترین شروط را دارد (شرطی ندارد و تمامی نمونه‌های را می‌پوشاند)، سپس به صورت حریصانه با اضافه کردن ویژگی‌هایی که کارایی قانون را بر روی نمونه‌های آموزشی بیشترین افزایش می‌دهند ادامه پیدا خواهد کرد. این فرایند بعد از اضافه شدن

ویژگی دوم نیز به همین صورت پیش می‌رود، و این فرایند به همین ترتیب ادامه پیدا می‌کند. مشابه ID3 این فرایند نیز حریصانه با افزایش شروط ویژگی‌ها فرضیه را گسترش می‌دهد تا کارایی آن به حد آستانه‌ی قابل‌قبولی برسد. بر خلاف ID3 این تعریف Learn-one-rule فقط یک زیرشاخه در هر مرحله‌ی جستجو اضافه می‌شود، فقط یک شرط که جفتی از ویژگی و مقدارش است، اما در ID3 یک دسته زیرشاخه اضافه می‌شد که تمامی مقادیر ویژگی را مورد بررسی قرار می‌داد.




*شکل ۱۰,۱ جستجو در شروط در فرایند Learn-one-rule که از ترتیب کلی به جزئی استفاده می‌کند.*
*در هر مرحله تمامی شروطی که اضافه شدنشان ممکن است بررسی می‌شود. حکم قانون طوری انتخاب می‌شود تا نمونه‌هایی که در شروط قانون صدق می‌کنند را راضی کند. این شکل جستجوی ستونی با عمق ۱ را نشان می‌دهد.*

جستجوی کلی به جزئی‌ای که در بالا آورده شد جستجویی حریصانه و با عمق یک و بدون نگاه به مراحل قبلی[۱] بود. درست مثل تمامی جستجوهای حریصانه، خطر بهینه‌سازی جزئی در هر مرحله وجود دارد. برای کم کردن این خطر، می‌توانیم الگوریتم را طوری تغییر دهیم تا جستجویش ستونی[۲] شود؛ جستجویی که در آن به جای انتخاب بهترین گزینه لیستی k گزینه‌ی بهتر نگه داشته می‌شوند. در هر مرحله از پیشروی جستجو، خاص سازی برای تمامی این k گزینه‌ی مطرح ساخته و قوانین حاصل در مراحل بعدی نیز با اضافه کردن k گزینه‌ی بهتر خاص‌تر می‌شوند. جستجوی ستونی لیستی از امیدوارکننده‌ترین جایگزین‌های فرضیه‌ی فعلی (بهترین فرضیه) نگه‌داری می‌کند و میزان موفقیت آن‌ها در هر مرحله از جستجو در نظر گرفته می‌شود. این الگوریتم جستجوی کلی به جزئی ستونی در برنامه‌ی CN2 که توسط (Clark and Niblett 1989) توصیف شد به کار رفته است. این الگوریتم به طور کامل در جدول ۱۰,۲ آورده شده است.

---

Learn-one-rule(Target_attribute,Attributes,Examples,k)

این زیر روال قانونی که تعدادی از نمونه‌های Examples را پوشش می‌دهد خروجی می‌دهد. در این جستجو زیر روال از جستجوی ستونی

---

[۱] backtrack
[۲] Beam search

حریصانه برای پیدا کردن بهترین قانون کمک می‌گیرد، این جستجو توسط معیار Performance کنترل می‌شود.

- Best_hypothesis را با مقدار اولیه‌ی Ø مقداردهی اولیه کن.
- Candidate_hypothesis را با مقدار اولیه‌ی {Best_hypothesis} مقداردهی اولیه کن.
- تا زمانی که مجموعه‌ی Candidate_hypotheses تهی نیست حلقه‌ی زیر را ادامه بده

۱. Candidate_hypotheses خاص‌تری بعدی را ایجاد کن

- مجموعه‌ی تمامی قیود به فرم (a=v) ←All_constrains
- که در آن a عضوی از مجموعه‌ی Attributes است و v نیز مقدار ممکنی از a است که در مجموعه‌ی Examples فعلی ظاهر شده
  - برای تمامی h های Candidate_hypotheses و برای تمامی c های All_constrains
    - با اضافه کردن c خاص سازی از h ایجاد کن
- هر فرضیه‌ی تکراری، ناسازگار یا کاملاً اختصاصی را از New_candidate_hypotheses حذف کن.

۲. Best_ hypothesis را تغییر بده

- برای تمامی h های New_candidate_hypotheses حلقه‌ی زیر را اجرا کن
  - اگر (Performance(h,Examples,Target_attribute)> Performance(Best_hypothesis,Examples,Target_attribute))
    آنگاه Best_hypothesis←h

۳. Candidate_ hypotheses را تغییر بده

- K بهترین عضو مجموعه‌ی New_candidate_hypotheses را بر اساس معیار Performance Candidate_ hypotheses←
- قانونی به فرم زیر را خروجی بده

"اگر Best_hypothesis آنگاه prediction"

که در این رابطه prediction پرتکرارترین مقدار Target_attribute در میان Examples هایی است که در Best_hypothesis صدق می‌کنند.

Performance(h,Examples,Target_attribute)

- زیرمجموعه‌ای از Examples که با h سازگار است ←h_Examples
- مقدار Entropy(h_Examples)– را برگردان، آنتروپی بر اساس مجموعه‌ی Target_attribute محاسبه می‌گردد.

*جدول ۱۰,۲ یکی از تعریف‌های Learn-one-rule جستجویی کلی به جزئی و ستونی است.*
*شرط فرضیه‌ی فعلی با متغیر Candidate_hypotheses مشخص می‌شود. این الگوریتم مشابه الگوریتمی است که در برنامه‌ی CN2 به کار رفته است (Clark and Niblett 1989).*

بیایید نکاتی که در مورد الگوریتم Learn-one-rule در جدول ۱۰,۲ آورده شد را با دقت بیشتری بررسی کنیم. ابتدا اینکه توجه داشته باشید که هر فرضیه که در حلقه‌ی اصلی در نظر گرفته می‌شود عطفی از شروط ویژگی مقدار است. هر یک از این فرضیه‌های عطفی نظیر یک دسته

شروط ممکن برای یادگیری‌اند که با آنتروپی نمونه‌هایی که می‌پوشاند اندازه‌گیری می‌شود. جستجو مرحله به مرحله فرضیه‌های خاص‌تر را در نظر می‌گیرد تا به کلی‌ترین فرضیه برسد که تمامی ویژگی‌های ممکن را داشته باشد. قانون خروجی الگوریتم قانونی است که در این جستجو بیشترین Performance را داراست، میزان خاصی قانون در این انتخاب هیچ تأثیری ندارد. حکم قانون خروجی در مرحله‌ی انتهایی الگوریتم مشخص می‌شود، بعد از این که شروط توسط متغیر Best_hypothesis مشخص شد. حکم قانون خروجی متداول‌ترین مقدار ویژگی هدف در نمونه‌های تحت پوشش آن تعیین می‌شود. نکته‌ی آخر این که توجه داشته باشید با اینکه جستجوی ستونی خطر قانون‌های موضعی بهینه را کمتر می‌کند اما هنوز خطر از بین نخواهد رفت. با این وجود، حتی زمانی که قوانین موضعی بهینه‌اند، الگوریتم ترتیبی می‌تواند دسته قوانینی را یاد بگیرد که با هم نمونه‌های آموزشی را بپوشانند، زیرا که الگوریتم متناوباً از زیرروال Learn-one-rule استفاده می‌کند.

### ۱۰,۲,۲ نسخه‌ها

الگوریتم ترتیبی، با استفاده از الگوریتم Learn-one-rule دسته قوانینی را یاد می‌گیرد که نمونه‌های آموزشی را پوشش دهند. نسخه‌های متفاوتی از این روش مورد بحث و بررسی قرار گرفته است. برای مثال، در بعضی شرایط ممکن است لازم باشد طوری برنامه‌ریزی شود که فقط قوانینی که نمونه‌های مثبت را می‌پوشاند یاد بگیریم و نمونه‌هایی که در هیچ‌یک از قوانین صدق نمی‌کند را به طور پیش‌فرض منفی دسته‌بندی کنیم. برای مثال، برای یادگیری تابع هدفی مثل "زنان بارداری که دوقلو باردارند" یکی از چنین شرایط است. در این حالت، نسبت نمونه‌های مثبت به کل جمعیت کم است، بنابراین قوانین خلاصه‌تر و مفهوم‌تر خواهند بود اگر فقط قوانین برای دسته‌بندی نمونه‌های مثبت در نظر گرفته شود و بقیه‌ی نمونه‌ها به طور پیش‌فرض منفی دسته‌بندی شوند. این روش مشابه negative-as-failure در زبان Prolog است که در آن اگر نتوان ثابت کرد که یک نمونه مثبت است به طور پیش‌فرض منفی فرض می‌شود. برای یادگیری چنین قوانینی که فقط یک مقدار ویژگی هدف را تعیین می‌کنند، الگوریتم Learn-one-rule را می‌توان طوری تغییر داد که پارامتری اضافی داشته باشد که مقدار پیش‌بینی مقدار هدف مورد توجه را مشخص کند. جستجوی ستونی کلی به جزئی به همان شکل قبلی دست‌نخورده باقی می‌ماند و فقط تعریف زیر روال Performance که فرضیه‌ها را ارزیابی می‌کرد تغییر می‌کند. توجه داشته باشید که تعریف Performance به عنوان آنتروپی در این وضعیت دیگر مناسب نیست، زیرا که این تعریف به قانون‌هایی که تعداد زیادی نمونه‌ی منفی را می‌پوشانند نیز مثل قوانینی که تعداد زیادی نمونه‌ی مثبت را می‌پوشانند مقدار زیادی نسبت می‌دهد. در چنین شرایطی استفاده از نسبت نمونه‌های مثبت پوشانده شده به کل نمونه‌های پوشانده شده‌ی فرضیه معیاری مناسب‌تر خواهد بود.

تغییرات دیگری نیز ممکن است، این نسخه‌های تغییریافته در خانواده‌ای از الگوریتم‌هایی با نام AQ، (Michalski , 1969, Michalski et al. 1986) قرار می‌گیرند، این خانواده از الگوریتم‌ها قبل از الگوریتم CN2 به وجود آمده و تمامی نتایج بالا بر اساس این خانواده از الگوریتم‌ها به دست آمده است. الگوریتم‌های AQ نیز فصلی از قوانین عطفی را یاد می‌گیرند، با این وجود، AQ و الگوریتم بیان شده در بسیاری از جهات متفاوت‌اند. ابتدا اینکه الگوریتم پوششی[۳] ی AQ با الگوریتم ترتیبی فرق دارد، زیرا که صراحتاً به دنبال قوانینی می‌گردد که مقدار هدف خاصی را پوشش می‌دهند، و برای هر مقدار هدف یک دسته قوانین فصلی را یاد می‌گیرد. دوم اینکه الگوریتم‌های AQ برای یادگیری تک قوانین از روشی متفاوت با Learn-one-rule استفاده می‌کنند. مشابه قبل این زیر روال نیز از جستجوی ستونی کلی به جزئی استفاده می‌کند با این تفاوت که برای تمرکز این جستجو از یک نمونه‌ی مثبت استفاده می‌شود. در کل، این زیر روال فقط ویژگی‌هایی را در نظر می‌گیرد که با نمونه‌ی مثبت سازگارند و با این فرض برای تعمیم از جستجوی کلی به جزئی استفاده می‌کند. در هر مرحله الگوریتم قانونی سازگار با یکی از نمونه‌های مثبت غیر پوشش داده شده یاد می‌گیرد تا مشابه قبل جستجو را برای پیدا کردن فصلی از قوانین عطفی ادامه دهد.

---

[۳] covering

## ۱۰,۳ یادگیری دسته قوانین: خلاصه

الگوریتم ترتیبی بالا و الگوریتم یادگیری درختی فصل ۳ متدهای متفاوت ممکنی برای یادگیری دسته قوانین هستند. این قسمت جنبه‌های مختلف فضای طراحی این گونه الگوریتم‌ها (الگوریتم‌های یادگیری دسته قوانین) را بررسی خواهد کرد.

ابتدا اینکه الگوریتم‌های ترتیبی در هر مرحله فقط یک قانون را یاد می‌گیرند و نمونه‌هایی را که توسط آن قانون پوشانده می‌شوند حذف کرده و همین فرایند را با بقیه‌ی نمونه‌ها ادامه می‌دهند. در مقابل، الگوریتم‌های یادگیری درختی مثل ID3 کل دسته قوانین را با هم و در یک جستجو برای درختی قابل‌قبول یاد می‌گیرند. به همین دلیل، الگوریتم‌هایی چون ID3 را الگوریتم‌های پوشش همزمان[۴] می‌نامند، در مقابل الگوریتم‌های ترتیبی مثل CN2. کدام یک از این نوع الگوریتم‌ها ارجحیت دارد؟ تفاوت کلیدی در قدم‌های ابتدایی جستجوی آن‌هاست. در هر مرحله‌ی ID3 بین تمامی ویژگی‌های ممکن، ویژگی‌ها با استفاده از تقسیم‌بندی‌ای که انجام می‌دهند انتخاب می‌شوند. در مقابل CN2 در بین جفت ویژگی مقدارها، ویژگی‌ها را با استفاده از زیرمجموعه‌ای از داده‌ها که می‌پوشانند انتخاب می‌کند. یکی از راه‌های درک این تفاوت توجه به تعداد انتخاب‌های خاصی است که این دو الگوریتم انجام می‌دهند تا دسته قوانین مساوی‌ای را یاد بگیرند. در یادگیری n قانون k ویژگی تست ممکن برای شروط دارد، الگوریتم‌های ترتیبی k.n مرحله جستجو انجام می‌دهند و در هر مرحله نیز انتخابی مستقل برای معلوم کردن شرط قانون انجام می‌دهند. در مقابل، الگوریتم‌های پوشش همزمان تعداد بسیار کمتری انتخاب مستقل انجام می‌دهند زیرا که هر انتخاب ویژگی در گره‌ای از درخت متناسب با معلوم کردن آن ویژگی برای تعداد زیادی از قوانین است. به عبارت دیگر، اگر گره‌ای از درخت ویژگی‌ای که m مقدار ممکن دارد را بررسی کند، انتخاب این ویژگی برای آن گره‌ی درخت هم‌ارز انتخاب این ویژگی برای m قانون نظیر آن است (تمرین ۱۰,۱). بنابراین، الگوریتم‌های ترتیبی مثل CN2 نسبت به الگوریتم‌های پوشش همزمان مثل ID3 تعداد بیشتری انتخاب مستقل انجام می‌دهند. با این وجود، این سؤال همچنان بدون جواب باقی است: کدام روش ارجحیت دارد؟ اگر داده‌های موجود به اندازه‌ی کافی زیاد باشد تا بتوان تعداد زیاد انتخاب‌های مستقل الگوریتم ترتیبی را برطرف کند الگوریتم ترتیبی بهتر است، در مقابل اگر داده‌های موجود کم باشد بهتر است که انتخاب ویژگی‌ها بین شروط مشترک باشد و الگوریتم‌های پوشش همزمان مفیدتر خواهد بود. جنبه‌ی با اهمیت دیگر، این است که، آیا اینکه قوانین ویژگی‌های مشابهی را تست کنند برای ما مطلوب است؟ در الگوریتم‌های پوشش همزمان مثل یادگیری درختی چنین کاری انجام می‌شود. اما در الگوریتم‌های ترتیبی نیازی به چنین کاری نیست.

جنبه دوم تفاوت این دو نوع الگوریتم نحوه‌ی کنترل جستجویشان در Learn-one-rule است. در الگوریتمی که در بالا توضیح دادیم، این جستجو از فرضیه‌های کلی‌تر به جزئی‌تر انجام می‌شود. در دیگر الگوریتم‌های مطرح شده (مثل Find-S که در فصل ۲ بود) این جستجو از جزئی به کلی انجام می‌شود. یکی از مزیت‌های جستجوی کلی به جزئی در این است که کلی‌ترین فرضیه منحصربه‌فرد است، اما جزئی‌ترین فرضیه‌ها در اکثر فضاهای فرضیه‌ای منحصربه‌فرد نیستند (به تعداد نمونه‌ها خاص‌ترین فرضیه وجود دارد). با وجود تعداد زیادی خاص‌ترین فرضیه معلوم نیست که جستجو را باید از کدام فرضیه شروع کرد. یکی از برنامه‌هایی که از جستجوی جزئی به کلی استفاده می‌کند، Golem (Muggleton and Feng 1990) است که این مشکل را با انتخاب چندین نمونه به طور تصادفی و شروع با آن‌ها حل می‌کند. سپس بهترین فرضیه در میان این فرضیه‌های تصادفی انتخاب می‌شود.

---

[۴] simultaneous covering

جنبه سوم تفاوت در این است که Learn-one-rule جستجویی آزمون‌وخطایی[۵] در فرضیه‌های با قاعده دارد یا به عبارت دیگر کنترل نمونه‌ای[۶] است و تک نمونه‌های آموزشی تعمیم آن را کنترل می‌کنند. الگوریتم‌های جستجوی متداول کنترل نمونه‌ای شامل Find-S، Candidate-Elimination (در فصل ۲)، AQ، و الگوریتم Cigol که پیش‌تر در همین فصل بررسی شد هستند. در هر یک از این الگوریتم‌ها تولید یا تغییر فرضیه با بررسی یک تک نمونه‌ی آموزشی انجام می‌شود، و انتظار می‌رود که فرضیه‌ی حاصل کارایی بهتری برای آن تک نمونه داشته باشد. این نوع الگوریتم‌ها با الگوریتم Learn-one-rule آزمون‌وخطایی که در جدول ۱۰,۲ آمد متفاوت‌اند، در این الگوریتم فرضیه‌های موفق فقط بر اساس زبان[۷] بیان فرضیه‌ها به وجود می‌آیند و فقط زمانی به داده‌های آموزشی رجوع می‌شود که می‌خواهیم با استفاده از کارایی بر روی کل نمونه‌های آموزشی، بین این فرضیه‌های ممکن فرضیه‌ای را انتخاب کنیم. یکی از مزیت‌های مهم روش آزمون‌وخطا این است که هر انتخاب در جستجو بر اساس کارایی است که بر پایه‌ی تعداد زیادی نمونه تعریف می‌شود، بنابراین اثر داده‌های خطادار مینیمم می‌شود، اما در مقابل در الگوریتم‌های کنترل نمونه‌ای که فرضیه‌ها بر اساس تک نمونه‌های آموزشی تغییر می‌کنند خطر اشتباه بر اساس یک نمونه‌ی خطای آموزشی بسیار زیاد است و الگوریتم در مقابل داده‌های خطادار کاملاً آسیب‌پذیر است.

تفاوت چهارم دو روش این است که آیا و چگونه فرضیه‌ها هرس می‌شوند؟ همان‌طور که می‌دانید در یادگیری درختی، احتمال این وجود دارد که دسته قوانینی پیدا کنیم که با نمونه‌های آموزشی خیلی خوب سازگار باشد اما روی کل نمونه‌ها ضعیف عمل کند. یکی از روش‌های برطرف کردن این مشکل هرس هر قانون بعد از یادگیری از نمونه‌های آموزشی است. در کل، شروط قوانینی که حذفشان باعث افزایش کارایی قوانین بر روی یک دسته‌ی هرس، که مجزا از داده‌های آموزشی است، می‌شود باید هرس شوند. بحث کامل روی این موضوع را در قسمت ۳,۷,۱,۲ انجام دادیم.

جنبه آخر تفاوت بین این دو روش، نحوه‌ی تعریف روال Performance که برای کنترل جستجو در Learn-one-rule استفاده می‌شود است. توابع ارزیابی مختلفی را می‌توان مورداستفاده قرار داد. چندین نمونه از متداول‌ترین توابع ارزیابی در زیر آورده شده‌اند:

- تکرار نسبی[۸]. اگر n تعداد نمونه‌هایی باشد که با قانون تطابق دارند و $n_c$ تعداد نمونه‌هایی باشد که قانون درست دسته‌بندی می‌کند، ارزیابی تکرار نسبی قانون کسر زیر خواهد بود:

$$\frac{n_c}{n}$$

  تکرار نسبی در ارزیابی قوانین در AQ مورداستفاده قرار گرفته است.

- تخمین m دقت[۹]. این تخمین دقت به سمت دقتی پیش‌فرض برای قوانین بایاس شده است. از این روش زمانی که تعداد داده‌ها کم است و دقت قوانین باید بر اساس مجموعه‌ی نمونه‌های کوچکی تخمین زده شود استفاده می‌شود. اگر n و $n_c$ به ترتیب تعداد نمونه‌های سازگار و درست دسته‌بندی شده بر اساس قانون مورد نظر باشند و p نیز احتمال اولیه‌ی نمونه‌ی تصادفی باشد و اگر m یک وزن باشد، خواهیم داشت که دقت تخمین m، میانگین وزن‌دار تکرار نسبی و احتمال اولیه خواهد بود.

---

[۵] generate then test
[۶] example driven
[۷] syntax
[۸] relative frequency
[۹] m-estimate of accuracy

$$\frac{n_c + mp}{n + m}$$

توجه دارید که اگر m صفر باشد این کسر همان تکرار نسبی خواهد بود. با افزایش مقدار m تعداد نمونه‌هایی که برای تغییر احتمال اولیه‌ی p لازم است افزایش خواهد یافت. معیار m میانگین توسط (Cestnik and Bratko 1991) پیشنهاد شد و در نسخه‌های مختلف CN2 نیز به کار می‌رود. همچنین این تخمین در دسته‌بندی کننده ساده‌ی بیز به کار می‌رود.

- آنتروپی[10]. این معیار در تعریف Performance در الگوریتم جدول ۱۰,۲ استفاده شده است. اگر S مجموعه‌ای از نمونه‌ها باشد که با شروط قانون تطابق دارد، آنتروپی یکدستی تابع هدف را در این مجموعه از نمونه‌ها اندازه‌گیری می‌کند. ما از منفی آنتروپی استفاده می‌کنیم تا قوانین بهتر امتیاز بیشتری داشته باشند.

$$-Entropy(S) = \sum_{i=1}^{c} p_i \log_2 p_i$$

در این رابطه c تعداد مقادیر ممکن تابع هدف است و $p_i$ نیز نسبتی از S است که ویژگی هدف i امین مقدارش را می‌گیرد. ترکیبی از آنتروپی و یک تست آماری در الگوریتم CN2 مورداستفاده قرار گرفته است (Clark and Niblett 1989). همچنین آنتروپی هسته‌ی تابع بهره‌ی اطلاعات در بسیاری از الگوریتم‌های یادگیری درختی است.

## ۱۰,٤ یادگیری قوانین درجه اول

در قسمت قبل، الگوریتم‌هایی که دسته قوانین گزاره‌ای را یاد می‌گرفتند (دسته قوانینی که هیچ متغیری نداشتند) را مورد بحث قرار دادیم. در این بخش، به یادگیری قوانینی که متغیر دارند خواهیم پرداخت و یادگیری horn clause های درجه اول را در حالت کلی بررسی می‌کنیم. یکی از انگیزه‌های توجه به چنین قوانینی قدرت بیان آن‌ها نسبت به قوانین گزاره‌ای است. یادگیری استقرایی قوانین یا تئوری‌های درجه اول گاهی برنامه‌نویسی استقرایی منطقی[11] (ILP) نامیده می‌شود زیرا که می‌توان به این فرایند به دید روش ایجاد خودکار برنامه‌های Prolog از نمونه‌ها نگاه کرد. Prolog هدفی کلی است، زبان برنامه‌نویسی معادل تورینگ که در آن برنامه‌ها با دسته horn clause ها مشخص می‌شوند.

### ۱۰,٤,۱ horn clause های درجه اول

برای مشاهده‌ی مزیت‌های استفاده از نمایش درجه اول به جای نمایش گزاره‌ای (بدون متغیر)، کار یادگیری مفهوم ساده‌ی Daughter(x,y) را که بر روی زوج افراد x و y تعریف می‌شود را در نظر بگیرید. زمانی که x دختر y است مقدار Daughter(x,y) درست است و در غیر این صورت مقدار آن غلت است. فرض کنید که برای هر فرد در داده‌های موجود با ویژگی‌های Name، Father، Mother، Male و Female توصیف می‌شود. بنابراین، هر نمونه‌ی آموزشی توصیفی از دو فرد با ویژگی‌هایشان و مقدار ویژگی هدف Daughter خواهد بود. برای مثال، در زیر یک نمونه‌ی مثبت آورده شده است:

$$< Name_1 = Sharon, Mother_1 = Louise, Father_1 = Bob,$$

---

[10] entropy

[11] inductive logic programming

$$Male_1 = False, Female_1 = True, Name_2 = Bob,$$

$$Mother_2 = Nora, Father_2 = Victor, Male_2 = True,$$

$$Female_2 = Fales, Daughter_{1,2} = True >$$

در این نمونه زیرنویس هر ویژگی مشخص می‌کند که ویژگی کدام یک از دو فرد توصیف می‌شود. حال اگر تعدادی نمونه‌ی آموزشی برای مفهوم هدف $Daughter_{1,2}$ جمع کنیم و آن‌ها را به یک یادگیر گزاره‌ای مثل CN2 یا C4.5 بدهیم قانون‌های خروجی مثل قانون زیر خواهند بود:

$$IF \quad (Father_1 = Bob) \wedge (Name_2 = Bob) \wedge (Female_1 = True)$$

$$THEN \quad Daughter_{1,2} = True$$

با وجود اینکه این قانون درست است اما بسیار خاص است، حتی اگر در دسته‌بندی نمونه‌های جدید به کار رود کاربردش بسیار کم خواهد بود. مشکل اینجاست که نمایش گزاره‌ای هیچ راه کلی‌ای برای توصیف روابط بین مقادیر ویژگی‌ها ندارد. در مقابل، برنامه‌ای که از قوانین درجه اول استفاده می‌کند می‌تواند قانون کلی زیر را یاد بگیرد:

$$IF \quad Father(y, x) \wedge Female(y), \quad THEN \quad Daughter(x, y)$$

در این رابطه x و y متغیرهایی هستند که هر فردی می‌توانند باشند.

Horn clause های درجه اول همچنین می‌توانند متغیرهایی در شروط داشته باشند که در نمایش گزاره‌ای ممکن نیست. برای مثال، یک قانون برای GrandDaughter می‌تواند قانون زیر باشد:

$$IF \quad Father(y, z) \wedge Mother(z, x), Female(y)$$

$$THEN \quad GrandDaughter(x, y)$$

توجه داشته باشید که متغیر z در این رابطه پدر y است که در حکم قانون نیامده است. زمانی که یک متغیر فقط در شرط یک قانون ظاهر می‌شود بدین معناست که قانون به وجود آن وابسته است؛ به عبارت دیگر، حکم قانون تا زمانی درست است که حداقل یک نمونه وجود داشته باشد که در شرط قانون در مکان z صدق کند.

همچنین در این نوع نمایش استفاده از خود حکم و ایجاد قوانین بازگشتی ممکن است. برای مثال، دو قانونی که در ابتدای همین فصل آورده شده‌اند با هم ویژگی Ancestor(x,y) را بیان می‌کردند. متدهای یادگیری ILP مثل متدهایی که در زیر توضیح خواهیم داد اثبات شده که می‌توانند پهنای گسترده‌ای از توابع بازگشتی ساده (مثل تابع Ancestor و توابعی که برای ترتیب کردن عناصر یک لیست ، پاک کردن یک عضو خاص یا ترکیب کردن دو لیست به کار می‌روند) را یاد بگیرند.

### ۱۰,٤,۲ واژگان

قبل از شروع بحث در مورد الگوریتم‌های یادگیری دسته قوانین horn clause، بیایید ابتدا بعضی واژگان اساسی منطق را معرفی کنیم. تمامی اصطلاحات[۱]، ترکیبی از ثابت‌ها[۲] (مثلاً Bob و Louise)، متغیرها[۳] (مثل x و y) ، نمادهای گزاره‌ای[۴] (مثل Married و Greater_Than) و نمادهای توابع[۵] (مثل age) هستند. تفاوت گزاره‌ها و توابع این است که گزاره‌ها یکی از دو مقدار True و False را می‌گیرند اما توابع می‌توانند هر ثابتی را به عنوان مقدار داشته باشند. ثابت‌ها را با حروف بزرگ و متغیرها را با حروف کوچک نشان خواهیم داد. همچنین توابع را با حروف کوچک و گزاره‌ها را با حروف بزرگ نشان خواهیم داد.

با این نشانه‌گذاری می‌توان اصطلاح به فرم ذیل ساخت: یک جمله[۶] مساوی یک ثابت، یا متغیر یا هر تابعی از یک جمله باشد (مثل Bob، x، age(Bob)). یک عبارت[۷] یک گزاره یا نقیض آن است (مثل Married(Bob,Louise) یا ¬Greater_Than(age(Sue),20)). اگر یک عبارت علامت نقیض (¬) داشته باشد به آن عبارت منفی می‌گوییم، در غیر این صورت به آن عبارت مثبت می‌گوییم.

یک بند[۸] فصلی از عبارات است که در آن فرض می‌شود تمامی متغیرها معلوم فرض می‌شوند. یک horn clause شامل حداکثر یک عبارت مثبت است،

$$H \vee \neg \mathrm{L}_1 \vee \ldots \vee \neg \mathrm{L}_\mathrm{n}$$

در این رابطه H عبارت مثبت و $\neg L_1 \ldots \neg L_n$ عباراتی منفی هستند. چون داریم $(B \vee \neg A) = (B \leftarrow A)$ و $\neg(B \wedge A) = (B \vee \neg A)$، پس horn clause بالا را می‌توان به صورت مشابه به شکل زیر نشان داد.

$$H \leftarrow (\mathrm{L}_1 \wedge \ldots \wedge L_n)$$

که در نمادگذاری قبلی معادل با عبارت زیر است.

$$IF \mathrm{L}_1 \wedge \ldots \wedge L_n \; THEN \; H$$

بدون توجه به نحوه‌ی نمایش horn clause شروط قانون $L_1 \wedge \ldots \wedge L_n$ بدنه‌ی قانون[۹] یا مقدم[۱۰] نامیده می‌شوند. عبارت H حکم را تشکیل می‌دهد که سر قانون[۱۱] یا نتیجه[۱۲] نامیده می‌شود. برای راحتی کار، این تعاریف در جدول ۱۰,۳ گردآوری شده اما باز در موقع مواجهه با تعاریف دیگر آن‌ها را بیان می‌کنیم.

---

۱ Expression
۲ constant
۳ variable
۴ predicate symbol
۵ Function symbols
۶ term
۷ literal
۸ clause
۹ clause body
۱۰ antecedents

هر عبارت خوش‌فرم[۱۳] از ثابت‌ها (مثل Joe, 23 ,Mary)، متغیرها (مثل x)، گزاره‌ها (مثل، Female در (Female(Mary) و توابع (مثل age در (age(Mary) تشکیل شده است. (گزاره=predicate)

جمله[۱۴] هر ثابت، متغیر یا تابعی است که بر روی جمله‌ای دیگر اعمال شده است. (مثل (Mary, age(Mary), x, age(x).

عبارت[۱۵] هر گزاره یا عکس گزاره‌ای که به مجموعه‌ای از جمله‌ها اعمال می‌شود است. (مثل، Female(Mary), (¬Female(x), Greater_than(age(Mary,20)

عبارت ؟ (ground literal) ---

عبارت منفی (negative literal) عبارتی است که پیشوند منفی دارد (مثل، (¬Female(Joe).

عبارت مثبت (positive literal) عبارتی است که علامت منفی نداشته باشد (مثل، (Female(Mary).

--- (clause) فصلی از عبارات به فرم $M_1 \vee \dots M_n$ است که متغیرهای آن در کل گزاره‌هایی هستند.

(horn clause) ها به فرم زیر بیان می‌شوند

$$H \leftarrow (L_1 \wedge \dots \wedge L_n)$$

در این رابطه $H, L_1, \dots, L_n$ همگی عبارات مثبتی هستند. H سر horn clause یا حکم (consequent) نامیده می‌شود. عطف عبارات $L_1 \wedge \dots \wedge L_n$ نیز بدنه یا شرط horn clause نامیده می‌شود.

برای تمامی عبارات A و B عبارت (A←B) معادل (A∧¬B) است و ¬(A∧B) نیز معادل (¬A∨¬B). بنابراین، horn clause ها را می‌توان به فرم معادل فصلی نوشت

$$H \vee \neg L_1 \vee \dots \vee \neg L_n$$

یک جانشینی (substitution) تابعی است که متغیرها را با جملاتی جایگزین می‌کند. برای مثال، جانشینی {x/3,y/z} متغیر x را با جمله‌ی 3 و متغیر y را با جمله‌ی z جایگزین می‌کند.

---

برای عبارت L و جایگزینی $\theta$ از $L\theta$ برای حاصل اعمال $\theta$ به L استفاده می‌کنیم.

جانشینی یکتا کننده‌ی[16] دو عبارت $L_1$ و $L_2$ جانشینی $\theta$ است که داشته باشیم $L_1\theta = L_2\theta$.

---

[۱۱] clause head

[۱۲] consequent

[۱۳] well-formed

[۱۴] term

[۱۵] literal

[۱۶] unifying substitution

جدول ۱۰,۳ تعاریف اولیه‌ی منطق درجه اول *(first order logic)*.

## ۱۰,۵ یادگیری دسته قوانین درجه اول: FOIL

الگوریتم‌های مختلفی برای یادگیری دسته قوانین درجه اول یا همان horn clause ها ارائه شده است. در این بخش برنامه‌ای به نام FOIL (Quinlar 1990) که روشی بسیار مشابه الگوریتم‌های ترتیبی و الگوریتم‌های Learn-one-rule قسمت قبل است را توضیح می‌دهیم. در واقع برنامه‌ی FOIL تعمیم طبیعی الگوریتم‌های قبلی به نمایش قوانین درجه اول است. رسماً، فرضیه‌های یادگرفته شده‌ی FOIL دسته قوانین درجه اولی هستند که هر قانون یک horn clause است. فقط دو تفاوت وجود دارد. اول اینکه قوانین یادگرفته شده‌ی FOIL محدودکننده‌تر از قوانین کلی horn clause هستند، زیرا که عبارات آن‌ها نمی‌توانند شامل توابع نمادی باشند (این باعث می‌شود تا پیچیدگی جستجو فضای فرضیه‌ای کمتر شود). دوم اینکه قوانین FOIL از قوانین horn clause شامل‌ترند[۱۷] زیرا که قوانین FOIL می‌توانند در بدنه قانون عبارت منفی نیز داشته باشند. از FOIL برای مسائل زمینه‌های مختلف استفاده شده است. برای مثال، از آن برای یادگیری حالت بازگشتی الگوریتم Quicksort و یادگیری تمییز دادن چینش‌های قانونی و غیرقانونی صفحه‌ی شطرنج استفاده شده است.

الگوریتم FOIL در جدول ۱۰,۴ به طور خلاصه بیان شده است. توجه دارید که حلقه‌ی خارجی مشابه نسخه‌ای از الگوریتم‌های ترتیبی‌ای که قبلاً درباره‌ی آن بحث کردیم است؛ این حلقه در هر اجرا یک قانون جدید یاد می‌گیرد و نمونه‌های مثبتی را که توسط قانون پوشانده می‌شوند حذف خواهد کرد. حلقه‌ی داخلی متناسب با نسخه‌ای از الگوریتم Learn-one-rule است که برای نمایش قوانین درجه اول تعمیم داده شده است. همچنین توجه دارید که تفاوت‌های بسیار کم و کوچکی میان FOIL و الگوریتم‌های قبلی وجود دارد. در کل، FOIL بر خلاف الگوریتم قبلی که دنبال قوانینی که مقدار تابع هدف را درست یا غلت دسته‌بندی می‌کنند تنها به دنبال قوانینی می‌گردد که مقدار تابع هدف را درست (True) پیش‌بینی می‌کنند. همچنین جستجوی FOIL بیشتر hillclimbing است تا جستجوی ستونی (به طور معادل، از ستونی با پهنای یک برای جستجو استفاده می‌کند).

---

FOIL(Target_predicate,Predicates,Examples)

Pos ← تمامی نمونه‌هایی که برایشان Target_predicate درست است.
Neg ← تمامی نمونه‌هایی که برایشان Target_predicate غلت است.
Learned_rules ← {}
تا زمانی که Pos تهی نیست حلقه‌ی زیر را ادامه بده
قانون جدید NewRule را یاد بگیر

NewRule ← قانونی که Target_predicate را بدون شرط پیش‌بینی می‌کند
Neg ← NewRuleNeg
تا زمانی که NewRuleNeg تهی نیست حلقه‌ی زیر را ادامه بده
عبارتی جدید برای خاص سازی به NewRule اضافه کن

---

[۱۷] expressive

Candidate_literals ← عبارات کاندید برای NewRule را با Predicates ایجاد کن.

$$\underset{L \in \text{Candidate\_literals}}{\text{argmax}} \; Foil\_Gain(L, NewRule) \rightarrow \text{Best\_literal}$$

Best_literal را به شروط NewRule اضافه کن.

NewRuleNeg ← زیرمجموعه‌ای از NewRuleNeg که شروط NewRule را راضی می‌کند.

Learned_rules+NewRule ← Learned_rules

Pos ← {اعضایی از Pos که توسط NewRule پوشانده می‌شوند}-Pos .

Learned_rules را خروجی بده.

*جدول ۱۰,۴ الگوریتم پایه‌ای FOIL.*

*متد خاصی برای ساخت Candidate_literals و تعریف FOIL_Gain به کاربرده می‌شود که در متن آورده شده است. الگوریتم پایه‌ای را می‌توان با کمی تغییر با داده‌های خطادار سازگار کرد، در متن به آن تغییرات اشاره شده است.*

جستجوی فضای فرضیه‌ای FOIL با بررسی سلسله مراتبی[۱۸] بهتر درک می‌شود. هر تکرار حلقه‌ی بیرونی FOIL یک قانون به مجموعه‌ی فصلی فرضیه Learned_rules می‌افزاید. اثر هر قانون جدید در کلی‌تر کردن فرضیه فصلی فعلی است (اندازه‌ی مجموعه‌ی نمونه‌هایی که مثبت دسته‌بندی می‌کند را افزایش می‌دهد). با این نگاه، روش جستجو کلی به جزئی در میان فضای فرضیه‌ای و با شروع از خاص‌ترین فرضیه فصلی و خروج از الگوریتم در زمانی که تمام نمونه‌ها را پوشاند خواهد بود. حلقه‌ی داخلی جستجویی بهتر[۱۹] برای تعیین دقیق تعریف هر قانون جدید انجام می‌دهد. این حلقه‌ی داخلی فضای فرضیه‌ای دومی که فصل عبارت‌هاست را برای پیدا کردن فصلی که فرض‌های قانون را تشکیل دهد جستجو می‌کند. با این فضای فرضیه‌ای، این حلقه از روشی کلی به جزئی، hillclimbing و با شروع از کلی‌ترین فرضیه‌ی ممکن (بدون هیچ شرطی) و افزایش عبارات به آن برای خاص‌تر کردن قانون برای پرهیز از پوشاندن نمونه‌های منفی عمل می‌کند.

دو تفاوت اساسی میان FOIL و الگوریتم‌های ترتیبی و Learn-one-rule که قبلاً بررسی کردیم وجود دارد. این تفاوت‌ها که ناشی از قوانین درجه اول‌اند به شرح زیرند:

در جستجوی کلی به جزئی برای یادگیری قوانین جدید، FOIL از مراحلی دیگر برای ایجاد خاص سازی‌های ممکن استفاده می‌کند.این تفاوت ناشی از آن است که شروط قانون می‌توانند متغیر نیز داشته باشند.

FOIL از معیار کارایی‌ای به نام Foil_Gain استفاده می‌کند در حالی که Learn-one-rule که در جدول ۱۰,۲ نیز آمده بود از معیار آنتروپی استفاده می‌کرد. این تفاوت ناشی از آن است که FOIL فقط به دنبال قوانینی می‌گردد که نمونه‌ها را مثبت دسته‌بندی کنند.

دو قسمت بعدی این تفاوت‌ها را به طور دقیق‌تر بررسی می‌کنند.

### ۱۰,۵,۱ ایجاد خاص سازی‌های ممکن FOIL

برای ایجاد خاص سازی‌ای از قانون فعلی، FOIL مجموعه‌ای از عبارات جدید را ایجاد می‌کند، هر یک از این عبارات ممکن است به تنهایی به شروط قانون اضافه شوند. به عبارت دقیق‌تر، فرض کنید که قانون فعلی به فرم زیر باشد،

[۱۸] hierarchically

[۱۹] finer-grained

$$P(x_1, x_2, \dots, x_k) \leftarrow L_1 \dots L_n$$

در این رابطه $L_1 \dots L_n$ عبارات تشکیل‌دهنده‌ی شرط قانون فعلی و $P(x_1, x_2, \dots, x_k)$ نیز عبارت سر قانون یا حکم[۲۰] است. FOIL خاص سازی‌های ممکن این قانون را با در نظر گرفتن عبارتی جدید مثل $L_{n+1}$ که به یکی از فرم‌های زیر است:

$Q(v_1, \dots, v_r)$، که در آن Q گزاره‌ی دلخواهی از Predicates و $v_i$ ها نیز متغیرهای جدید یا موجود در قانون هستند. حداقل یکی از $v_i$ ها در ایجاد عبارت باید در قانون موجود باشد.

$Equal(x_j, x_k)$، که در آن $x_j$ و $x_k$ متغیرهای موجود در قانون هستند.

قرینه‌ی یکی از دو عبارت بالا.

برای تصور، یادگیری قوانین را برای عبارت هدف GrandDaughter(x,y) بر اساس گزاره‌های Father (predicator) و Female در نظر بگیرید. جستجوی کلی به جزئی FOIL با کلی‌ترین قانون ممکن شروع می‌شود،

$$\text{GrandDaughter}(x, y) \leftarrow$$

که بدین معناست که GrandDaughter(x,y) برای هر x و y مقدار درست دارد. برای خاص سازی این قانون اولیه، فرایند بالا عبارات زیر را به عنوان خاص‌تر سازی‌های ممکن شرط قانون در نظر می‌گیرد: Equal(x,y)، Female(x)، Female(y)، Father(y,x)، Father(x,y)، Father(x,z)، Father(y,z)، Father(z,x)، Father(z,y) و نقیض عبارات مذکور (مثل $\neg\text{Equal}(x, y)$). توجه دارید که z متغیری جدید است در حالی که x و y در قانون فعلی موجودند.

حال فرض کنید که بین عبارات مذکور، FOIL حریصانه عبارت Father(y,z) که ما را به خاص‌ترین عبارت ممکن می‌برد را انتخاب کند،

$$\text{GrandDaughter}(x, y) \leftarrow \text{Father}(y, z)$$

در کلی سازی عبارات ممکن برای خاص‌تر کردن این قانون، FOIL تمامی عبارات ذکر شده در مرحله‌ی قبل به اضافه‌ی عبارات Female(z)، Equal(z,x)، Father(z,w)، Father(w,z) و نقیضشان را در نظر خواهد گرفت. این عبارات جدید به خاطر اضافه شدن متغیر z در مرحله‌ی قبل به مجموعه‌ی عبارات ممکن اضافه می‌شوند. به همین خاطر متغیر جدید w نیز در نظر گرفته خواهد شد.

اگر FOIL در این مرحله عبارت Father(z,x) و در مرحله‌ی بعدی Female(y) را انتخاب کند نتیجه قانون زیر خواهد بود، که فقط نمونه‌های مثبت را می‌پوشاند و متعاقباً جستجو را متوقف می‌کند.

$$\text{GrandDaughter}(x, y) \leftarrow \text{Father}(y, z) \wedge Father(z, x) \wedge Female(y)$$

در این لحظه، FOIL تمامی نمونه‌های مثبت پوشانده شده توسط این قانون را حذف خواهد کرد. اگر نمونه‌ی مثبت باقی بماند، جستجوی کلی به جزئی برای یافتن قانونی دیگر از سر گرفته خواهد شد.

---

[۲۰] postcondition

### ۱۰,۵,۲ کنترل جستجو در FOIL

برای انتخاب بهترین عبارت میان عبارات ممکن تولیدی در هر مرحله FOIL از معیاری از کارایی قانون بر روی داده‌های آموزشی استفاده می‌کند. در این کار، تمامی ترکیب‌های ممکن متغیرهای قانون فعلی در نظر گرفته می‌شود. برای تصور این فرایند، دوباره مثالی یادگیری GrandDaughter(x,y) را در نظر بگیرید. فرض کنید که داده‌های آموزشی مجموعه‌ی ساده‌ی زیر باشد و از قرارداد[۲۱] P(x,y) استفاده می‌کنیم (بخوانید "P ی x، y است").

GrandDaughter(Victor,Sharon)  Fahter(Sharon,Bob)  Father(Tom,Bob)

Female(Sharon)  Father(Bob,Victor)

بیایید در اینجا برای سادگی کار فرض کنیم که برای تخمین متغیر GrandDaughter فقط از متغیرهای GrandDaughter، Father و Female بر روی ثابت‌های Victor، Sharon، Bob و Tom استفاده می‌کنیم، متغیرهایی که در بالا نیامده‌اند غلت فرض شده‌اند (بدین معنا که انگار عبارات ¬GrandDaughter(Tom,Bob)، ¬GrandDaughter(Victor,Victor) و ... در بالا آمده‌اند.).

برای انتخاب بهترین خاص کننده‌ی قانون فعلی، FOIL تمامی ترکیب‌های ممکن ثابت‌های داده‌های آموزشی را در نظر می‌گیرد. برای مثال، در مرحله‌ی اول قانون به شکل زیر خواهد بود،

$$\text{GrandDaughter}(x, y) \leftarrow$$

در این قانون متغیرهای x و y لازم نیست هیچ شرط خاصی داشته باشند و ممکن است هر ترکیب چهار ثابت Victor، Sharon، Bob و Tom باشند. در اینجا از نمایش {x/Bob, y/Sharon} برای نمایش یک ترکیب (binding) خاص استفاده می‌کنیم، که به هر متغیر یک مقدار ثابت نسبت می‌دهد. چون ۴ ثابت وجود دارند بنابراین برای مقدار اولیه‌ی قانون ۱۶ حالت ترکیب ممکن است. ترکیب {x/Victor, y/Sharon} یک نمونه‌ی مثبت است زیرا که داده‌های آموزشی GrandDaughter(Victor,Shron) را شامل می‌شود. ۱۵ ترکیب دیگر ممکن (مثل {x/Bob, y/Tom}) مدارک منفی (negative evidence) در قانون مثال فعلی هستند زیرا که هیچ ادعایی (assertion) برای درستی آن‌ها در داده‌های آموزشی موجود نیست.

در هر مرحله قانون بر اساس این مجموعه‌ی ترکیب‌های[۲۲] مثبت و منفی ارزیابی می‌شود، با این فرض که قوانینی که ترکیب‌های مثبت بیشتر و ترکیب‌های منفی کمتری را بپوشانند ارجحیت بیشتری دارند. با اضافه شدن عبارات بیشتر به قانون، مجموعه‌ی ترکیب‌ها نیز تغییر خواهد کرد. توجه داشته باشید که اگر عبارتی که متغیر جدیدی را تعریف می‌کند به قانون اضافه شود، تعداد ترکیب‌های قانون در طول افزایش خواهد یافت (برای مثال، اگر Father(y,z) به قانون بالا اضافه شود، ترکیب اولیه‌ی {x/Victor, y/Sharon} به {x/Victor, y/Sharon, z/Bob} که طول بیشتری دارد تبدیل خواهد شد. همچنین توجه دارید که اگر امکان داشته باشد که متغیر جدید با ثابت‌های مختلفی جفت[۲۳] شود، تعداد ترکیب‌های قانون جدید بیشتر از تعداد ترکیب‌های قانون قبلی خواهد بود.

---

[۲۱] convention
[۲۲] binding
[۲۳] bind

تابع ارزیابی FOIL برای تخمین کارایی اضافه کردن عبارت جدید بر اساس ترکیب‌های مثبت و منفی پوشش داده شده قبل و بعد از اضافه کردن عبارت جدید عمل می‌کند. به عبارت دقیق‌تر، قانونی مثل R و عبارت ممکن L که به بدنه‌ی آن اضافه می‌شود را در نظر بگیرید، اگر R′ قانون جدید حاصل از اضافه کردن L به R باشد، مقدار Foil_Gain(L,R) که برای اضافه کردن L به R تعریف می‌شود به صورت زیر تعریف می‌شود،

$$Foil\_Gain(L,R) \equiv t\left(\log_2 \frac{p_1}{p_1+n_1} - \log_2 \frac{p_0}{p_0+n_0}\right) \qquad (10.1)$$

در این رابطه $p_0$ و $n_0$ به ترتیب تعداد ترکیب‌های مثبت و منفی قانون R و $p_1$ و $n_1$ به ترتیب تعداد ترکیب‌های مثبت و منفی قانون R′ هستند. t نیز تعداد نمونه‌های مثبتی است که توسط قانون R پوشانده می‌شد و با اضافه کردن عبارت L هنوز پوشانده می‌شود. هنگامی که متغیری جدید با اضافه کردن L به R اضافه می‌شود، ترکیب‌های قبلی پوشانده شده، زمانی پوشانده شده در باقی می‌مانند که ترکیبی نظیر آن‌ها در R′ صدق کند.

تابع Foil_Gain تفسیری مستقیم از تئوری اطلاعات است. بنا به تئوری اطلاعات، $-\log_2 \frac{p_0}{p_0+n_0}$ مینیمم تعداد بیت‌های لازم برای کد سازی دسته‌بندی یک ترکیب دلخواه از ترکیب‌های پوشانده شده توسط قانون R است. به طور مشابه، $-\log_2 \frac{p_1}{p_1+n_1}$ نیز تعداد بیت‌های لازم برای کد سازی یک ترکیب دلخواه از ترکیب‌های قانون R′ است. از آنجایی که t تعداد ترکیب‌های پوشانده شده‌ی مشترک R و R′ است، Foil_Gain(L,R) را می‌توان به عنوان میزان کاهش تعداد بیت‌های لازم برای کدسازی تمامی دسته‌بندی‌های ممکن ترکیب‌های R دانست.

### ۱۰,۵,۳ یادگیری دسته قوانین بازگشتی[۲۴]

در بحث بالا از احتمال اینکه عبارت‌های اضافه شده به بدنه‌ی قانون خود گزاره‌ی (predicate) هدف باشند صرف‌نظر کردیم، مثل عبارت سر قانون. با این وجود، اگر متغیر هدف را به مجموعه‌ی گزاره‌ها اضافه کنیم، FOIL با توجه به آن عبارات ممکن را تشکیل می‌دهد. همین خاصیت به FOIL اجازه می‌دهد تا توانایی یادگیری قوانین بازگشتی را داشته باشد، قوانینی که از گزاره‌های سر و بدنه‌ی قانون با هم استفاده می‌کنند. برای مثال، قانون زیر را که تعریفی بازگشتی از رابطه‌ی Ancestor است را در نظر بگیرید.

IF Parent(x,y) THEN Ancestor(x,y)

IF Parent(x,z)/\Ancestor(z,y) THEN Ancestor(x,y)

با داشتن مجموعه‌ای از نمونه‌های آموزشی را می‌توان با روشی مشابه یادگیری GrandDaughter یاد گرفت. توجه دارید که قانون دوم از جمله قوانینی است که به ذات درون جستجوی FOIL محسوب می‌شود، Ancestor در مجموعه‌ی گزاره‌ها[۲۵] که از آن عبارات جدید قانون ایجاد می‌شود در نظر گرفته خواهد شد. البته اینکه این قانون یادگرفته می‌شود یا خیر کاملاً به عبارت‌های دیگر ممکن و اینکه آیا می‌توانند در جستجوی حریصانه‌ی FOIL امتیاز بیشتری داشته باشند بستگی دارد. (Cameron-Jones and Quinlar 1993) در مورد نمونه‌های

---

[۲۴] recursive

[۲۵] Predicates

بسیاری از کاربردهای موفق FOIL در یادگیری قوانین بازگشتی بحث کرده‌اند. آن‌ها همچنین در مورد مباحث دیگر FOIL چون چگونگی اجتناب از ایجاد حلقه‌های بی‌نهایت در قوانین بحث‌هایی انجام داده‌اند.

### ۱۰,۵,٤ خلاصه‌ی FOIL

به طور خلاصه، FOIL الگوریتم‌های ترتیبی CN2 را برای کنترل یادگیری دسته قوانین درجه اول مشابه horn clause ها تامیم می‌دهد. برای یادگیری هر قانون و در هر مرحله‌ی اضافه کردن عبارت به شروط قانون، FOIL جستجویی کلی به جزئی انجام می‌دهد. عبارات جدید اضافه شده به قانون ممکن است در شروط قانون یا حکم قانون موجود باشند، و یا حتی ممکن است متغیرهای جدیدی به قانون اضافه کنند. در هر مرحله، از تابع Foil_Gain که در رابطه‌ی ۱۰,۱ آمده برای انتخاب بین عبارات جدید ممکن استفاده می‌شود. اگر عبارات جدید بتوانند گزاره‌ی هدف را در بربگیرند، FOIL می‌تواند، اصولاً دسته قوانین بازگشتی را نیز یاد بگیرد. تا جایی که پیچیدگی مانع از ایجاد قوانین با حلقه بی‌نهایت شود اثبات شده FOIL می‌تواند با موفقیت دسته قوانین بازگشتی را یاد بگیرد.

اگر داده‌های آموزشی بدون خطا باشند، FOIL ممکن است آن‌قدر اضافه کردن قوانین را ادامه دهد تا دیگر هیچ نمونه‌ی منفی پوشانده شده‌ای وجود نداشته باشد. برای کنترل داده‌های خطادار، جستجو تا زمانی که یک شرط بین دقت، پوشانندگی و پیچیدگی برقرار شود ادامه پیدا خواهد کرد. FOIL برای جلوگیری از رشد بیش از اندازه‌ی قوانین از روش کوتاه‌ترین توضیح استفاده می‌کند، روشی که در آن عبارات جدید فقط زمانی اضافه می‌شوند که طول توضیح آن‌ها از طول توضیح داده‌های آموزشی‌ای که توجیه می‌کنند کمتر باشد. جزئیات این استراتژی در (Quinlar 1990) آمده است. علاوه بر آن، FOIL قوانینی که یاد می‌گیرد را نیز بعد از یادگیری هرس می‌کند، این روش هرس کردن مشابه هرس کردن درخت‌های تصمیم‌گیری است (فصل ۳).

## ۱۰,٦ استقرا به عنوان استنتاج وارونه

روش دوم و کاملاً متفاوت برای برنامه‌نویسی استقرایی بر اساس مشاهدات ساده‌ای است که نشان می‌دهد استقرا (induction) فقط وارون استنتاج (deduction) است! در کل، یادگیری ماشین ساختن تئوری‌هایی است که برای داده‌های مشاهده شده توضیح می‌آورند. با داشتن مجموعه‌ی داده‌های D و دانش قبلی B (background knowledge)، یادگیری را می‌توان پیدا کردن فرضیه‌هایی مثل h تعریف کرد که h و B بتوانند با هم دلیلی برای D بیاورند. به عبارت دقیق‌تر، مثل همیشه فرض کنید که داده‌های آموزشی D که مجموعه‌ای از نمونه‌های آموزشی به فرم $< x_i, f(x_i) >$ است در اختیار است. در اینجا $x_i$ نشان‌دهنده‌ی i امین نمونه‌ی آموزشی و $f(x_i)$ مقدار هدف آن است. یادگیری، مسئله‌ی پیدا کردن فرضیه‌ای مثل h است که بتوان دسته‌بندی $f(x_i)$ برای $x_i$ را از h و مشخصات $x_i$ و دانش قبلی B نتیجه گیری کرد.

$$(\forall < x_i, f(x_i) > \in D)(B \wedge h \wedge x_i) \vdash f(x_i) \qquad (10.2)$$

عبارت $X \vdash Y$ را بخوانید "Y را می‌توان از X نتیجه گرفت (follows deductively)" یا "X موجب Y (entails) می‌شود". رابطه‌ی ۱۰,۲ محدودیت‌هایی که لازم است که h داشته باشد تا بتوان از B و h و $x_i$، $f(x_i)$ را نتیجه گرفت توصیف می‌کند.

برای مثال، مسئله‌ای را در نظر بگیرید که در آن مفهوم هدف "جفت افرادی مثل <u,v> است که در آن v فرزند u است"، و آن را با Child(u,v) نشان می‌دهیم. فرض کنید که فقط یک نمونه‌ی مثبت به ما داده می‌شود Child(Bob, Sharon)، و در آن نمونه‌ها به

صورت Male(Bob)، Female(Sharon) و Father(Sharon,Bob) توصیف شده‌اند. علاوه بر آن فرض کنید که دانش قبلی داریم که $Parent(u,v) \leftarrow Father(u,v)$. می‌توان وضعیت عبارات رابطه‌ی ۱۰،۲ را به صورت زیر مشخص کرد:

$$x_i: Male(Bob), Female(Sharon), Father(Sharon, Bob)$$

$$f(x_i): \quad Child(Bob, Sharon)$$

$$B: \quad Parent(u,v) \leftarrow Father(u,v)$$

دو فرضیه از فرضیه‌های ممکنی که در رابطه‌ی $(B \wedge h \wedge x_i) \vdash f(x_i)$ صدق می‌کنند در زیر آورده شده‌اند،

$$h_1: Child(u,v) \leftarrow Father(v,u)$$
$$h_2: Child(u,v) \leftarrow Parent(v,u)$$

توجه دارید که عبارت هدف Child(Bob, Sharon) بدون استفاده از B و تنها از $h_1 \wedge x_i$ نتیجه گرفته می‌شود. اما در مورد فرضیه‌ی $h_2$، اوضاع متفاوت است، عبارت هدف Child(Bob, Sharon) از $B \wedge h_2 \wedge x_i$ نتیجه‌گیری خواهد شد و نمی‌توان آن را از $h_2 \wedge x_i$ به تنهایی نتیجه گرفت. این مثال، نقش دانش قبلی در گسترش مجموعه‌ی فرضیه‌های قابل‌قبول برای یک مجموعه‌ی معلوم از داده‌های آموزشی را مشخص می‌کند. همچنین، نشان می‌دهد که چگونه متغیرهای جدید (مثل Parent) را می‌توان به فرضیه‌ها (مثل $h_2$) حتی زمانی که متغیر در توصیف نمونه‌ی $x_i$ دخیل نیست معرفی کرد. این فرایند افزودن (augmenting) مجموعه‌ای از متغیرها بر اساس دانش قبلی استقرای ساختاری (constructive induction) نامیده می‌شود.

این اهمیت رابطه‌ی (۱۰،۲) است که مسئله‌ی یادگیری را در محیط استنتاج استقرایی و منطق بیان می‌کند. در بحث ما این منطق، منطق گزاره‌ای و منطق درجه اول خواهد بود و الگوریتم‌های راحت الدرک برای اتوماتیک کردن استنتاج خواهد بود. جالب است که ایجاد عکس این فرایندها برای اتوماتیک کردن فرایند تعمیم استقرایی ممکن است. به نظر می‌رسد، این دید که استقرا را می‌توان با عکس کردن استنتاج به دست آورد را اولین بار در قرن نوزدهم W. S. Jevons مشاهده کرد، وی می‌نویسد:

استقرا در حقیقت عکس عمل استنتاج است و نمی‌توان بدون این ارتباط وجودش را تصور کرد، بنابراین سؤال اهمیت پیش نخواهد آمد. چه کسی فکر خواهد کرد که جمع یا تفریق در ریاضیات مهم‌تر است؟ اما در سهولت تفاوت زیادی بین یک عمل و عکسش وجود دارد؛ ... باید در نظر گرفته شود که انجام عمل استقرا بسیار سخت‌تر و پیچیده‌تر از عمل استنتاج است ... (Jevons 1874)

در ادامه‌ی این فصل به استقرا به دید عکس استنتاج نگاه خواهیم کرد. در اینجا مسئله‌ی اصلی و مورد توجه طراحی عکس عملگرهای نتیجه‌گیری (entailment operators) است. یکی از عملگرهای عکس نتیجه‌گیری، O(B, D) است که نمونه‌های آموزشی $D = \{< x_i, f(x_i) >\}$ و دانش قبلی B را دریافت کرده و فرضیه‌ای خروجی می‌دهد که در رابطه‌ی ۱۰،۲ صدق کند.

$$O(B,D) = h \; such \; that \; (\forall < x_i, f(x_i) > \in D)(B \wedge h \wedge x_i) \vdash f(x_i)$$

البته در کل، ممکن است فرضیه‌های مختلفی برای h وجود داشته باشد که $(\forall < x_i, f(x_i) > \in D)(B \wedge h \wedge x_i) \vdash f(x_i)$. یکی از روش‌های ILP برای انتخاب میان چنین فرضیه‌هایی استفاده از قانون کوتاه‌ترین توضیح است (برای اطلاعات بیشتر به بخش ۶،۶ مراجعه کنید).

روش‌های جذاب دیگر برای فرمولی کردن کار یادگیری به عنوان پیدا کردن فرضیه‌ای مثل h که بتواند در رابطه‌ی $(\forall < x_i, f(x_i) > \in D)(B \wedge h \wedge x_i) \vdash f(x_i)$ صدق کند وجود دارد.

این فرمولی کردن تعریف‌های معمول یادگیری مفاهیم را به عنوان پیدا کردن مفهومی کلی که با مجموعه‌ای از نمونه‌های آموزشی مطابقت داشته باشد را شامل می‌شود (این تعریف معمول مشابه حالتی است که هیچ دانش قبلی‌ای نداشته باشیم.).
یکی کردن نماد دانش قبلی B، با این فرمول تعریف غنی‌تری از تناسب یک فرضیه به ما می‌دهد. تا به حال، تناسب فرضیه (مثل شبکه‌ای عصبی) را فقط وابسته به مشخصات فرضیه و داده‌ها و مستقل از محیط مطالعه فرض می‌کردیم. در مقابل، این فرمول اجازه می‌دهد تا اطلاعات فضای عمل که با دانش قبلی B مشخص می‌شود را جزو تعریف "تناسب" (fit) قرار دهیم. در کل، h با نمونه‌ای مثل $< x_i, f(x_i) >$ متناسب است هر گاه بتوان $f(x_i)$ را از $B \wedge h \wedge x_i$ نتیجه گرفت.
با تأثیر دادن دانش قبلی B، این فرمول متدهای یادگیری‌ای را می‌طلبد که به جای جستجوی کورکورانه‌ی فضای فرضیه‌ای از دانش قبلی برای هدایت جستجوی h استفاده کنند. فرایند دقت عکس (inverse resolution) که در قسمت‌های بعدی توضیح داده می‌شود از دانش قبلی برای این کار استفاده خواهد کرد.
به طور همزمان، تحقیقات بر روی استقرای برنامه‌نویسی منطقی نشان می‌دهد که این فرمولی کردن با چندین مشکل کاربردی روبروست:

الزامات لازم برای $(\forall < x_i, f(x_i) > \in D)(B \wedge h \wedge x_i) \vdash f(x_i)$ به طور طبیعی با داده‌های آموزشی خطادار سازگار نیست. مشکل اینجاست که این عبارت اجازه‌ی وجود خطای احتمالی در داده‌های مشاهده شده نمونه‌های $x_i$ یا مقدار تابع هدفشان $f(x_i)$ را به ما نمی‌دهد. وجود چنین خطاهایی ممکن است محدودیت‌های متناقض در h ایجاد کند. متأسفانه، اکثر محیط‌های منطقی رسمی با دادن مجموعه‌ای متناقض از ادعاها (assertion) کارایی خود را در تمیز دادن مقدار واقعی و غیرواقعی از دست می‌دهند.
زبان منطقی درجه اول آن‌قدر شامل است که تعداد فرضیه‌هایی که در عبارت $(\forall < x_i, f(x_i) > \in D)(B \wedge h \wedge x_i) \vdash f(x_i)$ صدق می‌کنند بسیار زیاد است که جستجو میان این فضای فرضیه‌ها در حالت کلی رام نشدنی (intractable) است. اکثر تحقیقات فعلی در فرم‌های محدود قوانین درجه اول بررسی می‌شوند یا از دانش قبلی درجه دوم نیز استفاده می‌گردد تا بتوان فضای فرضیه‌ای را برای جستجو ساده‌تر کرد.
بر خلاف این تصور که دانش قبلی باید به جستجو برای یک فرضیه کمک کند، در اکثر سیستم‌های ILP (شامل تمامی سیستم‌های بحث شده در این فصل) پیچیدگی جستجو فضای فرضیه‌ای با افزایش دانش قبلی افزایش می‌یابد. (با این وجود، برای الگوریتم‌هایی که دانش قبلی پیچیدگی جستجوی فضای فرضیه‌ای را کم می‌کنند به فصل ۱۱ و ۱۲ مراجعه کنید)
در قسمت بعدی، یکی از روش‌های کلی عکس عملگر نتیجه‌گیری را که با عکس کردن استنتاج فرضیه‌هایی ایجاد می‌کند را بررسی خواهیم کرد.

## ۱۰,۷ دقت عکس

متدی کلی‌ای برای اتوماتیک کردن استنتاج قانون دقت (resolution rule) است که توسط (Robinson 1965) پیشنهاد شد. قانون دقت قانونی کامل برای استنتاج در منطق درجه اول است. بنابراین، جای این سؤال هست که بپرسیم آیا می‌توان قانون دقت را وارون کرد تا

یک عکس عملگر نتیجه‌گیری به دست آید؟ جواب آری است، وارون این قانون عملگری است که پایه‌ی برنامه‌ی Cigol را تشکیل می‌دهد که توسط (Muggleton and Buntine 1988) ارائه شد.

بهتر است که قانون دقت را در فرم گزاره‌ای معرفی کنیم تا برای تعمیم بر روی نمایش درجه اول آماده باشد. اگر L یک عبارت گزاره‌ای دلخواه باشد و P و R نیز دو حکم گزاره‌ای دلخواه باشند قانون دقت به صورت زیر خواهد بود،

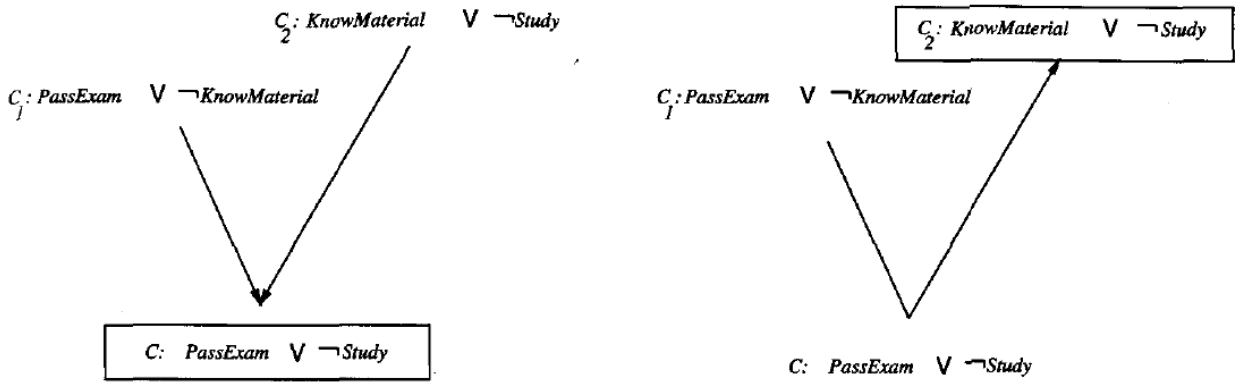
$$P \quad \vee \quad L$$

$$\neg L \quad \vee \quad R$$

$$P \quad \vee \quad R$$

آن را باید به صورت روبرو خواند: با داشتن دو حکم (clause) بالایی، حکم پایین نتیجه‌گیری می‌شود. با داشتن دو ادعـای $P \vee L$ و $\neg L \vee R$ واضح است که یکی از دو عبارت L یا ¬L غلت است، بنابراین حکم $P \vee R$ قانون دقت درست خواهد بود.

فرم کلی گزاره‌ای عملگر دقت در جدول ۱۰,۵ آورده شده است. با داشتن دو عبارت $C_1$ و $C_2$ عملگر دقـت ابتـدا عبـارتی مثـل L را مشـخص می‌کند که عبارتی مثبت در یکی از عبارت و عبارتی منفی در عبارت دیگر است. سپس نتایج فرمول بالا را می‌کشد. برای مثال، عملکرد عملگـر دقت را در شکل سمت راست شکل ۱۰,۲ در نظر بگیرید. با داشتن $C_1$ و $C_2$، مرحله‌ی اول فرایند عبـارت $L = \neg KnowMaterial$ را مشخص می‌کند که در $C_1$ آمده و عکسش $\neg(\neg KnowMaterial) = KnowMaterial$ در $C_2$ آمـده اسـت. بنـابراین نتیجـه حکمی است که از ترکیب این دو عبارت به وجود می آید $C_1 - \{L\} = PassExam$ و $C_2 - \{\neg L\} = \neg Study$ است. به عنوان مثالی دیگر، نتیجه ی اعمال قانون دقت را به دو عبارت $C_1 = A \vee B \vee C \vee \neg D$ و $C_2 = \neg B \vee E \vee F$ را کـه $A \vee C \vee \neg D \vee E \vee F$ است.




*شکل ۱۰,۲ در سمت چپ کاربردی از (استنتاج) قانون دقت برای به دست آوردن C از $C_1$ و $C_2$ آمده است.*
*در سمت راست کاربردی از (استقرا) عکس آن عمل و به دست آوردن $C_2$ از C و $C_1$ آورده شده است.*

با داشتن عبارات $C_1$ و $C_2$ عبارتی مثل L از حکم $C_1$ پیدا کن که $\neg L$ در $C_2$ آمده باشد.

C را با استفاده از تمامی عبارات $C_1$ و $C_2$ به غیر از $L$ و $\neg L$ تشکیل بده. به عبارت دقیق‌تر، مجموعه عبارات C را می‌توان از رابطه‌ی زیر به دست آورد،

$$C = (C_1 - \{L\}) \cup (C_2 - \{\neg L\})$$

در اینجا $\cup$ نشان‌دهنده‌ی اجتماع مجموعه‌ها و "-" نشان‌دهنده‌ی اختلاف مجموعه‌هاست.

*جدول ۱۰،۵ عملگر دقت (فرم گزاره‌ای).*

*با داشتن دو حکم $C_1$ و $C_2$ می‌توان با استفاده از عملگر دقت Cای را پیدا کرد که $C_1 \wedge C_2 \vdash C$.*

وارون کردن عملگر دقت برای ساخت یک عکس عملگر نتیجه‌گیری $O(C, C_1)$ که اعمال استقرایی انجام می‌دهد بسیار ساده است. در کل، عکس عملگر نتیجه‌گیری باید بتواند یکی از حکم‌های اولیه $C_2$ را با بازگشایی C (resolve) و دیگر حکم $C_1$ به دست آورد. مثالی را فرض کنید که در آن حکم نهایی $C = A \vee B$ و حکم اولیه $C_1 = B \vee D$. چگونه می‌توان حکمی مثل $C_2$ را پیدا کرد که $C_1 \wedge C_2 \vdash C$؟ اول اینکه باید توجه داشت که طبق تعریف عملگر دقت هر عبارتی که در C هست ولی در $C_1$ نیست حتما در $C_2$ موجود خواهد بود. در مثال ما، این نشان می‌دهد که $C_2$ حتماً عبارت A را خواهد داشت. دوم اینکه عبارتی که در $C_1$ وجود دارد اما در C وجود ندارد توسط قانون دقت حذف شده است، بنابراین عکس آن باید در $C_2$ حضور داشته باشد. در مثال ما، این نشان می‌دهد که $C_2$ باید حتماً عبارت ¬D را داشته باشد. بنابراین خواهیم داشت که $C_2 = A \vee \neg D$. واضح است که از ترکیب دو قانون $C_1$ و $C_2$ به دست آمده C نتیجه گرفته خواهد شد.

با معلوم بودن حکم‌های C و $C_1$ عبارتی مثل L را پیدا کن که در $C_1$ موجود باشد اما در C موجود نباشد.

حکم دوم مثل $C_2$ را که عبارات زیر را شامل می‌شود را تشکیل بده

$$C_2 = \big(C - (C_1 - \{L\})\big) \cup \{\neg L\}$$

*جدول ۱۰،۶ عکس عملگر دقت (فرم گزاره‌ای).*

*با داشتن دو حکم C و $C_1$ می‌توان حکمی مثل $C_2$ را پیدا کرد که $C_1 \wedge C_2 \vdash C$.*

توجه دارید که در مثال بالا جواب دیگری برای $C_2$ وجود دارد. در کل، $C_2$ را می‌توان به صورت خاص‌تر $A \vee \neg D \vee B$ دانست. تفاوت میان این جواب و جواب اول در این است که در اینجا عبارتی که در $C_1$ ظاهر شده بود را به $C_2$ اضافه کرده‌ایم. نکته‌ی کلی اینجا این است که قانون عکس دقت قطعی نیست، در کل ممکن است بیش از یک $C_2$ وجود داشته باشد که بتوان از $C_1$ و $C_2$، C را نتیجه گرفت. روش ما در انتخاب بین جواب‌های مختلف ارجح دانستن حکم‌هایی است که طول توضیح کمتری دارند، یا به طور مشابه، فرض می‌کنیم $C_2$ هیچ عبارتی مشترک با $C_1$ ندارد. اگر این بایاس به سمت حکم‌های کوتاه‌تر را با قانون عکس دقت ترکیب کنیم الگوریتم جدول ۱۰،۶ به دست خواهد آمد.

حال می‌توانیم بر اساس عکس عملگر نتیجه‌گیری الگوریتم‌های یادگیری قانونی مثل دقت معکوس را طراحی کنیم. در کل، الگوریتم یادگیری می‌تواند از عکس نتیجه‌گیری برای ساختن فرضیه‌هایی که بتوانند به همراه دانش قبلی داده‌های آموزشی را نتیجه دهند تشکیل دهند. یکی از استراتژی‌های ممکن استفاده از الگوریتم‌های ترتیبی برای یادگیری حلقه‌ای دسته‌ای از horn clause با این روش است. در هر حلقه، الگوریتم نمونه‌ی آموزشی‌ای مثل $< x_i, f(x_i) >$ را که هنوز پوشانده نشده است انتخاب می‌کند. سپس از قانون عکس دقت برای ایجاد فرضیه‌ای ممکن مثل $h_i$ که در رابطه‌ی $(B \wedge h_i \wedge x_i) \vdash f(x_i)$ صدق کند استفاده می‌شود، در اینجا B دانش قبلی به علاوه‌ی تمامی قوانین قبلی یادگرفته شده است. توجه دارید که این جستجویی بر پایه نمونه‌هاست (example-driven)، زیرا که هر فرضیه ممکن برای پوشاندن یک نمونه‌ی خاص ایجاد شده است. البته اگر چندین فرضیه موجود باشد (که یک نمونه را بپوشانند) فرضیه‌ای ارجح‌تر خواهد بود که

دقت بهتری بر روی دیگر نمونه‌های پوشانده شده‌اش داشته باشد. برنامه‌ی Cigol از عکس دقت به علاوه‌ی نوعی الگوریتم ترتیبی بـه همـراه تعامل با کاربر برای به دست آوردن نمونه‌های آموزشی بیشتر برای به دست آوردن راهنمایی در جستجوی میـان فضـای وسـیع مراحـل اسـتقرا استفاده می‌کند. با این وجود Cigol بیشتر از قوانین درجه اول به جای نمایش گزاره‌ای استفاده می‌کند. در زیر تعمـیم قـانون دقـت لـازم بـرای سازگاری با نمایش درجه اول را توصیف خواهیم کرد.

### ۱۰,۷,۱ دقت در نمایش درجه اول

قانون دقت به راحتی به قوانین درجه اول تعمیم پیدا می‌کند. مشابه حالت گزاره‌ای، این فرایند دو حکم به برای ورودی دریافت کرده و یک حکم خروجی می‌دهد. تفاوت کلیدی با حالت گزاره‌ای در این است که این فرایند در نمایش درجه اول بر اساس یکتا کردن (unifying) جانشینی‌ها (substitution) انجام می‌شود.

هر نگاشت (mapping) از متغیرها به جملات را جانشینی می‌نامیم. برای مثال، جانشینی $\theta = \{x/Bob, y/z\}$ در متغیر x مقدار عبارت Bob و در متغیر y مقدار عبارت z را قرار می‌دهد. از نماد $W\theta$ برای نمایش نتیجه‌ی اعمال جانشینی $\theta$ در عبارت W استفاده می‌کنیم. بـرای مثال، اگر L عبارت Father(x, Bill) باشد و $\theta$ همان جانشینی مثال قبلی باشد داریم، LΘ=Father(x, Bill).

زمـانی کـه مـی‌گـوییم $\theta$ جانشـینی‌ای یکتاسـت کـه بـرای دو عبـارت $L_1$ و $L_2$ داشـته باشـیم $L_1\theta = L_2\theta$. بـرای مثـال، اگـر $L_1 = Father(x, y)$ و $L_2 = Father(Bill, z)$ باشد و $\theta = \{x/Bob, y/z\}$، θ یک جانشینی یکتا برای $L_1$ و $L_2$ است زیرا کـه $L_1\theta = L_2\theta = Father(Bill, y)$. اهمیت جانشینی یکتا در این است که در فرم گزاره‌ای دقت، می‌توان با پیدا کردن عبارتی مثل L که L در $C_1$ و $\neg L$ نیز در $C_2$ باشد نتیجه ی دو حکم $C_1$ و $C_2$ را تعیین کرد. در دسته قوانین درجه اول، این تعمیم پیدا کردن عبـارتی مثـل $L_1$ از $C_1$ و $L_2$ از $C_2$ است که بتوان جانشینی‌ای مثل θ پیدا کرد که برای $L_1$ و $\neg L_2$ یکتا باشد ($L_1\theta = \neg L_2\theta$).

قانون دقت سپس حکم C را از رابطه‌ی زیر تشکیل می‌دهد.

$$C = (C_1 - \{L_1\})\theta \cup (C_2 - \{L_2\})\theta \qquad (10.3)$$

حالت کلی عبارت دقت در جدول ۱۰,۷ آمده است. برای تصور، فرض کنیـد کـه $C_1 = White(x) \leftarrow Swan(x)$ و بـا فـرض اینکـه $C_2 = Swan(Fred)$. عبارت $C_1 = White(x) \vee \neg Swan(x)$ به دست خواهد آمد. حال می‌توان قانون دقت را اعمـال کـرد. بنابراین، نتیجه‌ی C از ترکیب دو عبارت $(C_1 - \{L_1\})\theta = White(Fred)$ و $(C_2 - \{L_2\})\theta = \emptyset$ یا C=White(Fred).

---

عبارتی مثل $L_1$ از $C_1$ و $L_2$ از $C_2$ و جانشینی θ را پیدا کن که $L_1\theta = \neg L_2\theta$.

نتیجه‌ی C را با اجتماع تمامی عبارات $C_1\theta$ و $C_2\theta$ به جز $L_1\theta$ و $\neg L_2\theta$ را تشکیل بده. به عبارت دقیق‌تر مجموعه‌ی عباراتی که در C خواهند بود به صورت زیرند،

$$C = (C_1 - \{L_1\})\theta \cup (C_2 - \{L_2\})\theta$$

---

*جدول ۱۰,۷ قانون دقت (فرم درجه اول)*

### ۱۰,۷,۲ وارون کردن دقت: حالت درجه اول

به صورت تحلیلی می‌توان مشتق عکس قانون دقت را با دست‌کاری رابطه‌ی ۱۰,۳ که تعریف قانون دقت است به دست آورد. ابتدا توجه داشته باشید که $\theta$ در رابطه‌ی ۱۰,۳ را می‌توان به صورت یکتا به $\theta_1$ و $\theta_2$ تجزیه کرد که $\theta = \theta_1\theta_2$، در این رابطه $\theta_1$ شامل تمامی جانشینی‌های مربوط به متغیرهای $C_1$ و $\theta_2$ شامل تمامی جانشینی‌های مربوط به متغیرهای $C_2$ می‌شود. این تجزیه برای این عملی است، که $C_1$ و $C_2$ همیشه با متغیرهای خالص (distinct) شروع می‌شوند (زیرا که این دو به صورت جهانی جملات مستقل‌اند-- ). با این تجزیه $\theta$ می‌توان رابطه‌ی ۱۰,۳ را به صورت زیر بازنویسی کرد،

$$C = (C_1 - \{L_1\})\theta_1 \cup (C_2 - \{L_2\})\theta_2$$

همیشه توجه داشته باشید که علامت "-" در اینجا به معنای اختلاف مجموعه‌هاست. حال اگر عکس عملگر دقت فقط حکم‌های $C_2$ ی را که اشتراکی با $C_1$ ندارند خروجی دهد (طبق قانون کوتاه‌ترین طول توضیح) می‌توان عبارت بالا را به صورت زیر بازنویسی کرد،

$$C - (C_1 - \{L_1\})\theta_1 = (C_2 - \{L_2\})\theta_2$$

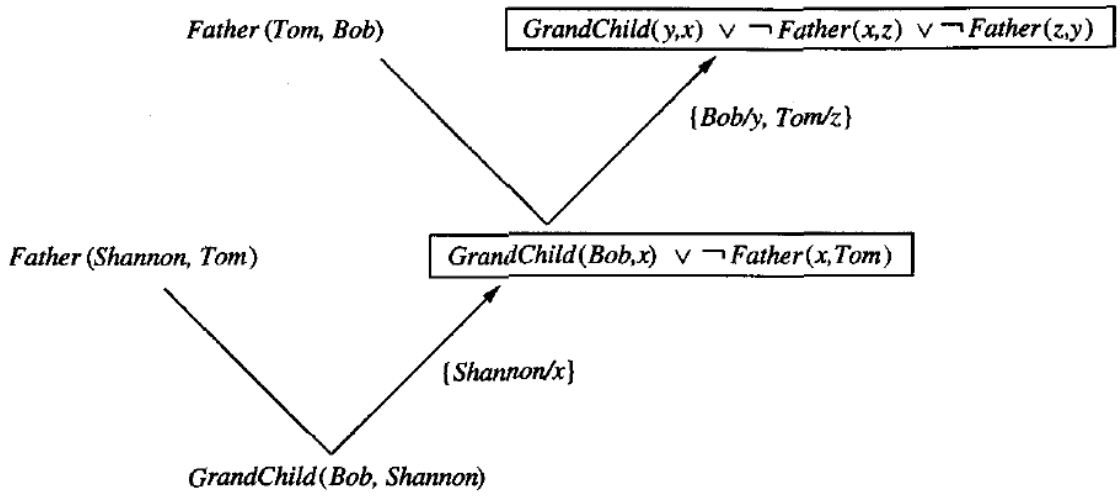
حال بالاخره از این حقیقت که طبق تعریف داریم $L_2 = \neg L_1\theta_1\theta_2^{-1}$ استفاده کرده و $C_2$ را برای به دست می آوریم،

**عکس دقت:**

$$C_2 = (C - (C_1 - \{L_1\})\Theta_1)\Theta_2^{-1} \cup \{\neg L_1\theta_1\theta_2^{-1}\} \qquad (10.4)$$

رابطه‌ی ۱۰,۴ عکس قانون دقت را برای منطق درجه اول نشان می‌دهد. مشابه فرم گزاره‌ای، عملگر عکس نتیجه‌گیری غیرقطعی (nondeterministic) است. در کل، در اعمال این عملگر باید در حالت کلی انتخاب‌های مختلف برای $C_1$ برای حل و جانشینی یکتای $\Theta_1$ و $\Theta_2$ وجود دارد. هر یک از این انتخاب‌ها ممکن است به جوابی خاص برای $C_2$ ختم می‌گردد.

شکل ۱۰,۳ اعمال چندمرحله‌ای عکس قانون دقت را برای یک مثال ساده نشان می‌دهد. در این شکل، هدف یادگیری قوانینی برای GrandChild(y,x) با داشتن داده‌های آموزشی D=GrandChild(Bob,Shannon) و دانش قبلی B={Father(Shannon,Tom), Father(Tom,Bob)} است. به بالاترین مرحله‌ی درخت عکس قانون دقت در شکل ۱۰,۳ توجه کنید. در اینجا، حکم C را برای نمونه‌ی آموزشی GrandChild(Bob, Shannon) را بررسی کرده و حکم $C_1 = Father(Shannon, Tom)$ را از دانش قبلی بر می‌گزینند. برای اعمال عکس عملگر دقت فقط یک انتخاب، Father(Shannon,Tom)، برای $L_1$ وجود دارد. فرض کنید که عکس جانشینی‌ها را به صورت $\Theta_1^{-1} = \{\}$ و $\Theta_2^{-1} = \{Shannon/x\}$. در این حالت، حکم نتیجه‌ی $C_2$ از ترکیب حکم $(C - (C_1 - \{L_1\})\Theta_1)\Theta_2^{-1} = (C\Theta_1)\Theta_2^{-1} =$ $GrandChild(Bob, x)$ و حکم $\neg Father(x, Tom) = \{\neg L_1\theta_1\theta_2^{-1}\}$. بنابراین نتیجه‌ی حکم GrandChild(Bob,Shannon)∨¬Father(x,Tom) یا به طور مترادف (GrandChild(Bob,x) ←Father(x,Tom)) خواهد بود. توجه داشته باشید که این قانون کلی به اضافه‌ی $C_1$ نمونه‌ی آموزشی GrandChild(Bob,Shannon) را نتیجه خواهد داد.




*شکل ۱۰٫۳ یک فرایند عکس دقت چندمرحله‌ای.*
*در این مثال، حکم‌های درون مستطیل‌ها حاصل از مراحل استنتاجی هستند. برای هر مرحله، C حکمی است که در بالا نشان داده شده، $C_1$ حکم سمت چپ و $C_2$ حکم داخل مستطیل در راست است. در هر دو مرحله‌ی استنتاجی، $\theta_1$ جانشینی تهی {} است، و $\theta_2^{-1}$ نیز جانشینی نشان داده شده در زیر $C_2$ است. توجه دارید که نتیجه نهایی (مستطیل گوشه‌ی بالا و راست) فرم جایگزین GrandChild(y,x)←Father(x,y)∧Father(z,y) است.*

به طور مشابه، این حکم نتیجه‌گیری شده را می‌توان به عنوان نتیجه‌ی C برای مرحله‌ی دوم عکس دقت به کار برد، مشابه شکل ۱۰٫۳. در هر مرحله، توجه دارید که چندین حالت نتیجه‌ی ممکن وجود دارد، این نتایج به انتخاب جانشینی وابسته‌اند (تمرین ۱۰٫۷). در مثال شکل ۱۰٫۳، مجموعه‌ی خاصی از انتخاب‌ها به حکم راضی‌کننده‌ی GrandChild(y,x)←Father(x,y)∧Father(z,y) می‌رسد.

### ۱۰٫۷٫۳ خلاصه‌ی وارون کردن دقت

به طور خلاصه، عکس دقت روشی کلی برای ایجاد اتوماتیک فرضیه‌ی h است که در رابطه‌ی $(B \wedge h \wedge x_i) \vdash f(x_i)$ صدق کند. این کار با عکس قانون کلی دقت (که در رابطه‌ی ۱۰٫۳ آمده) صورت می‌گیرد. با شروع از بازگشایی قانون و حل آن برای حکم $C_2$، عکس قانون دقت در رابطه‌ی ۱۰٫۴ به دست می آید.

با داشتن مجموعه‌ای از حکم‌ها، ممکن است با بکار بردن عکس قانون دقت چندین فرضیه به دست آید. توجه دارید که عکس قانون دقت این مزیت را دارد که فقط فرضیه‌هایی که در $(B \wedge h \wedge x_i) \vdash f(x_i)$ صدق می‌کنند را ایجاد می‌کند. در مقابل جستجوی آزمون‌وخطایی FOIL در هر مرحله از جستجو تعداد زیادی فرضیه که بعضی‌شان این شرط را ارضا نمی‌کنند ایجاد می‌کند. سپس FOIL از داده‌های D برای انتخاب بین این فرضیه‌ها استفاده خواهد کرد. با دانستن این تفاوت، انتظار می‌رود که جستجوی مبتنی بر عکس قانون دقت متمرکزتر و کاراتر باشد. با این وجود، این نتیجه‌گیری همیشه برقرار نیست. یکی از دلایل این است که عکس عملگر دقت فقط کسر کوچکی از داده‌های موجود را در هر مرحله از ایجاد فرضیه‌اش مدنظر قرار می‌دهد، در حالی که، FOIL تمامی داده‌های ممکن را برای انتخاب میان فرضیه‌های نحوی (syntactical) ایجاد شده مدنظر قرار می‌دهد. تفاوت‌های بین استراتژی‌هایی که از عکس استنتاج استفاده می‌کنند و استراتژی‌هایی که از جستجوی آزمون‌وخطایی استفاده می‌کنند همچنان موضوعی تحقیقاتی است. (Srinivasan et al. 1995) آزمایشی را که در آن این دو روش مقایسه می‌شوند مطرح می‌کند.

### ۱۰,۷,٤ تعمیم، نتیجه‌گیری $\Theta$ و نتیجه‌گیری

قسمت قبلی به ارتباط بین استقرا و عکس نتیجه‌گیری پرداخت. با در نظر داشتن اسرار قبلی در استفاده از ترتیب کلی‌تری برای سازمان‌دهی جستجوی فضای فرضیه‌ای، بد نیست که رابطه‌ی بین روابط کلی‌تری و عکس نتیجه‌گیری را مشاهده کنیم. برای درک این رابطه، تعاریف زیر را در نظر بگیرید.

رابطه‌ی کلی‌تری. در فصل ۲ رابطه‌ی کلی‌تر یا مساوی بودن ($\geq_g$) را به فرم زیر تعریف کردیم: برای دو تابع منطقی مقدار $h_j(x)$ و $h_k(x)$ زمانی می‌گوییم $h_j(x) \geq_g h_k(x)$ اگر و تنها اگر داشته باشیم که $(\forall x)h_k(x) \rightarrow h_j(x)$. این رابطه‌ی $\geq_g$ در بسیاری از الگوریتم‌های یادگیری برای کنترل جستجوی فضای فرضیه‌ای به کار می‌رود.

نتیجه‌گیری $\Theta$ ($\Theta$-subsumption).دو حکم $C_j$ و $C_k$ را در نظر بگیرید که هر دو به فرم $HVL_1 \vee \ldots \vee L_n$ هستند و H نیز عبارتی مثبت و $L_i$ نیز عبارات دلخواهی هستند. اگر و فقط اگر جانشینی‌ای مثل $\Theta$ یافت شود که $C_i\Theta \subseteq C_k$، آنگاه حکم $C_k$ زمانی نتیجه‌گیری $\Theta$ ی $C_j$ است. این تعریف را (Plotkin 1970) انجام داده است.

نتیجه‌گیری (Entailment). دو حکم $C_j$ و $C_k$ را در نظر بگیرید. اگر و فقط اگر $C_k$ را از $C_j$ استنتاج کرد (می‌نویسیم $C_j \vdash C_k$)، آنگاه $C_j$ از $C_k$ نتیجه‌گیری می‌شود.

رابطه‌ی بین سه تعریف بالا چیست؟ ابتدا بیایید تعریف $\geq_g$ را با نمادگذاری درجه اول مشابه دو تعریف دیگر بازنویسی کنیم. اگر h(x) فرضیه‌ی منطقی مقدار برای مفهوم هدف c(x)، که در آن h(x) با عطفی از عبارات بیان شده، آنگاه می‌توان فرضیه را با حکم زیر بازنویسی کرد،

$$c(x) \leftarrow h(x)$$

در اینجا نیز از تفسیر Prolog مبنی بر اینکه اگر نتوان اثبات کرد که x نمونه‌ای مثبت است، منفی دسته‌بندی خواهد شد استفاده می‌کنیم. بنابراین، می‌توان مشاهده کرد که تعریف قبلی‌مان از $\geq_g$ به شروط یا بدنه‌ی Horn clause ها اعمال می‌شود. حکم ضمنی Horn clause میز مفهوم هدف c(x) است.

رابطه‌ی این تعریف $\geq_g$ با تعریف نتیجه‌گیری $\Theta$ چیست؟ توجه دارید که اگر $h_1 \geq_g h_2$، آنگاه حکم $C_1: c(x) \leftarrow h_1(x)$ از $C_2: c(x) \leftarrow h_2(x)$ نتیجه‌گیری $\Theta$ می‌شود. علاوه بر این، نتیجه‌گیری $\Theta$ حتی هنگامی که سر حکم‌ها متفاوت‌اند درست است. برای مثال، حکم A در مثال زیر حکم B را نتیجه $\Theta$ می‌دهد:

$$A: Mother(x, y) \leftarrow Father(x, z) \wedge Spouse(z, y)$$

$$B: Mother(x, Louise) \leftarrow Father(x, Bob) \wedge Spouse(Bob, y) \wedge Female(x)$$

زیرا که $A\Theta \subseteq B$ اگر $\Theta$={y/Louise,z/Bob}. تفاوت کلیدی در این است که $\geq_g$ به طور ضمنی فرض می‌کند که سر دو حکم یکی هستند، در حالی که نتیجه‌گیری $\Theta$ برای حکم‌هایی که سرهای متفاوتی دارند نیز درست است.

بالاخره اینکه، نتیجه‌گیری $\Theta$ حالت خاصی از نتیجه‌گیری است. بدین معنا که اگر حکم A از حکم B نتیجه‌گیری $\Theta$ شود خواهیم داشت که $A \vdash B$. با این وجود، می‌توان حکم‌های A و B را پیدا کرد که $A \vdash B$ اما B از A نتیجه‌گیری $\Theta$ نشود. برای مثال زوج مرتب حکم‌های زیر را در نظر بگیرید:

$$A: Elephant(father\_of(x)) \leftarrow Elephant(x)$$

B: Elephant(father_of(father_of(y))) ← Elephant(y)

در اینجا تابع father_of(x) تابعی است که بر روی افراد تعریف شده و پدر x را بر می‌گرداند. توجه دارید که B را می‌توان از A اثبات کرد اما جانشینی Θ وجود ندارد که نتیجه‌گیری Θ ی A، B شود.

همان‌طور که در این مثال‌ها نیز نشان داده شد، نمادگذاری قبلی کلی‌تر بودن حالت خاصی از نتیجه‌گیری Θ است که خود نیز حالت خاصی از نتیجه‌گیری است. بنابراین جستجوی فضای فرضیه‌ای با کلی‌تر یا خاص‌تر کردن فرضیه‌ها محدودتر از جستجو با عملگرهای عکس نتیجه‌گیری است. متأسفانه، نمادگذاری متوسط نتیجه‌گیری Θ، نمادگذاری راحتی را ایجاد می‌کند که در بین نمادگذاری قبلی‌مان در رابطه‌ی کلی‌تری و نمادگذاری نتیجه‌گیری است.

### Prolog ۱۰,۷,۵

با وجود اینکه عکس دقت متدی فریبنده برای ایجاد فرضیه‌های ممکن است، در عمل می‌تواند به راحتی به انفجاری از فرضیه‌های ممکن تبدیل شود. روش جایگزین دیگر استفاده از عکس نتیجه‌گیری برای ایجاد تک خاص‌ترین فرضیه است که با دانش قبلی داده‌های آموزشی را نتیجه بدهند. این خاص‌ترین فرضیه را می‌توان برای محدود کردن جستجوی کلی‌تری در فضای فرضیه‌ای مشابه فضای فرضیه‌ای FOIL به کاربرد، با این شرط اضافه که فقط فرضیه‌های کلی‌تر از این مرز در نظر گرفته خواهند شد. این روش در سیستم Prolog مورداستفاده قرار گرفته است، الگوریتم Prolog به طور خلاصه در زیر آورده شده:

کاربر با استفاده از زبان عبارات درجه اول محدود شده فضای فرضیه‌ای H را مشخص می‌کند. محدودیت‌های با "نحوه‌ی تعریف" ایجاد می‌شود که به کاربر امکان مشخص کردن پیش‌بینی و نمادهای تابع و نوع و فرم آرگومان‌های هرکدام را می‌دهد ---.

Prolog از الگوریتمی ترتیبی برای یادگیری دسته عباراتی از H که داده‌ها را می‌پوشانند کمک می‌گیرد. برای هر نمونه‌ی $< x_i, f(x_i) >$ H درون $h_i$ که هنوز توسط عبارات یادگرفته شده پوشانده نشده، این سیستم ابتدا به دنبال خاص‌ترین فرضیه‌ی $(B \wedge h_i \wedge x_i) \vdash f(x_i)$ می‌گردد که داشته باشیم). به عبارت دقیق‌تر، این سیستم فرضیه را با محاسبه‌ی خاص‌ترین فرضیه‌ها در میان فرضیه‌هایی که از آن‌ها $f(x_i)$ با k بار استفاده از قانون دقت نتیجه گرفته می‌شود تخمین می‌زند (k پارامتری است که توسط کاربر تعیین می‌شود).

Prolog از جستجویی کلی به جزئی در فضای فرضیه‌ای محدود بین کلی‌ترین فرضیه‌ی ممکن و مرز خاص‌ترین $h_i$، که در مرحله‌ی ۲ محاسبه شد، استفاده می‌کند. در این مجموعه‌ی فرضیه‌ها، این سیستم به دنبال فرضیه‌ای است که کمترین طول توضیح (که با تعداد عبارات سنجیده می‌شود) را داشته باشد. این بخش از جستجو با روش ابتکاری A*-like که هرس بدون ریسک حذف خاص‌ترین فرضیه را ممکن می‌سازد انجام می‌گیرد.

جزئیات الگوریتم Prolog در (Muggleton 1992,1995) آورده شده است.

## ۱۰,۸ خلاصه و منابع برای مطالعه‌ی بیشتر

نکات اصلی این فصل شامل موارد زیر می‌شود:

الگوریتم‌های پوشش ترتیبی فصلی از قوانین را با یادگیری یک قانون دقیق و سپس حذف نمونه‌های مثبت پوشش داده شده توسط این قانون و تکرار این فرایند تا اینکه تمامی نمونه‌های مثبت پوشش داده شوند یاد می‌گیرند. این الگوریتم کارا و حریص برای یادگیری دسته قوانین گزینه‌ای در مقابل یادگیری درختی بالا به پایین مثل الگوریتم ID3 است که می‌توان آن را به صورت پوشش همزمان (در مقابل پوشش ترتیبی) دانست.

در الگوریتم‌های پوشش ترتیبی متدهای مختلفی را برای یادگیری یک قانون ارائه شده است. این متدها در استراتژی جستجو برای بررسی فضای شروط ممکن قانون متفاوت‌اند. یکی از روش‌های متداول، که در برنامه‌ی CN2 نیز به کار رفته، استفاده از جستجوی ستونی و کلی به جزئی است. در این روش قوانین خواص تر ایجاد و مورد بررسی قرار می‌گیرند تا قانونی به اندازه‌ی کافی دقیق پیدا گردد. روش‌های جستجوی فرضیه‌ای جزئی به کلی دیگر از جستجویی مبتنی بر نمونه‌ها به جای آزمون‌وخطا کمک می‌گیرند و از معیارهای آماری مختلف برای دقت قانون در کنترل جستجو کمک می‌گیرند.

مجموعه قوانین درجه اول (قوانینی که متغیر نیز دارند) نمایش شاملی دارند. برای مثل، زبان برنامه‌نویسی Prolog برنامه‌ها را در حالت کلی با استفاده از مجموعه‌ای از horn clause های درجه اول نشان می‌دهد. برای همین گاهی به مسئله‌ی یادگیری horn clause های درجه اول مسئله‌ی برنامه‌نویسی منطقی استقرایی می‌گویند.

یکی از روش‌های یادگیری دسته قوانین درجه اول تعمیم الگوریتم پوشش ترتیبی CN2 از نمایش گزاره‌ای به نمایش درجه اول است. این روش در برنامه‌ی FOIL به کار رفته است، این برنامه دسته قوانین درجه اول، شامل جمله قوانین بازگشتی، را یاد می‌گیرد. روش دیگری برای یادگیری قوانین درجه اول بر اساس این مشاهده است که استقرا عکس استنتاج است. به عبارت دیگر، مسئله‌ی استقرا پیدا کردن فرضیه مثل h است که

$$(\forall < x_i, f(x_i) > \in D)(B \wedge h \wedge x_i) \vdash f(x_i)$$

در این رابطه B دانش قبلی کلی است، $x_1 \ldots x_n$ توصیف نمونه‌های درون داده‌های آموزشی D هستند و $f(x_1) \ldots f(x_n)$ نیز مقادیر هدف نمونه‌های آموزشی هستند.

با دنبال کردن دیدگاه استقرا به عنوان عکس استنتاج، بعضی برنامه‌ها جستجویی برای فرضیه‌ها با استفاده از عملگری که عملگر معروف استنتاج را عکس می‌کند انجام می‌دهند. برای مثال Cigol از عکس دقت، عملگری که عکس عملگر استنتاج دقت متداول در اثبات قضایای مکانیکی است، استفاده می‌کند. Progol نیز استراتژی عکس استنتاج را با استراتژی جستجوی فضای فرضیه‌ای کلی به جزئی ترکیب می‌کند.

کارهای اولیه بر روی یادگیری نسبی توضیحات شامل برنامه‌ی معروف Winston (1970) برای یادگیری توضیحات به فرم شبکه برای مفاهیمی چون رئیس ("arch")، کار Banerji (1964,1969) و سری کارهای Michalski بر روی برنامه‌های AQ (Michalski 1960; Michalski et al. 1986) می‌شود. کارهای Michalski از جمله اولیه کارهایی است که استفاده از نمایش منطقی را در یادگیری بررسی کرد. تعریف Plotkin (1970) از جایگزینی θ رابطه‌ی اولیه‌ی بین استقرا و استنتاج را ثابت کرد. همچنین Vere (1975) یادگیری نمایش منطقی را مورد بررسی قرار داد، برنامه‌ی META-DENDRAL ی Buchanan توضیحات نسبی مربوطه‌ی طیف جرمی ساختارهای مولکولی --- را یادگرفته. این برنامه به طور موفقیت‌آمیز در کشف قوانین مفید ای که متعاقباً در کتب شیمی منتشر شده است. الگوریتم Candidate-Elimination vertion space ی Mitchell در رابطه‌های مشابه ساختارهای شیمی به کار رفته است.

با رواج زبان Prolog در اواسط دهه‌ی 1980، تحقیقات به سمت یادگیری توضیحات نسبی بیان شده با دسته Horn clause ها رفت. کارهای اولیه بر روی horn clause ها شامل برنامه‌ی MIS از Shapiro (1983) و MARVIN از Sammut and Banerji

(1986) می‌شود. الگوریتم FOIL از Quinlan (1990) که در فصل نیز مورد بررسی قرار گرفت، به سرعت با الگوریتم‌های جستجوی کلی به جزئی برای قوانین درجه اول شامل MFOIL از Dzeroski (1991)، FOCL از Pazzani et al. (1991)، CLAUDIEN از De Raedt and Bruynooghe (1993) و MARKUS از Grobelnik (1992). الگوریتم FOCL در فصل ۱۲ مورد بررسی قرار گرفته است.

خط دیگر تحقیقی از یادگیری horn clause با عکس عمل استنتاج توسط Muggleton and Buntine (1988) پیشنهاد شد که بر اساس ایده‌های مشابه Sammut and Banerji (1986) و Muggleton (1987) ایجاد شده بود. کارهای اخیر در این حوضه بر دیگر استراتژی‌های جستجو و متدهای تمرکز فضای فرضیه برای ایجاد یادگیری مهار شده است. برای مثال، Kietz and Wrobel (1992) از قانون schemata در برنامه‌ی RDT خود که فرم بیان در نظر گرفته شده در طول یادگیری را محدود می‌کند استفاده کرده‌اند، Muggleton and Feng (1992) نیز محدودیت نمایش درجه اول را به ij-determinate را بررسی کرده‌اند. Cohen (1994) برنامه‌ی کلی GRENDEL را که مجموعه‌ای از توضیحات محض را درباره‌ی زبان توصیف بدنه‌ی قانون دریافت کرده و به کاربر اجازه می‌دهد تا فضای فرضیه‌ای را به طور دلخواه محدود کند را مورد بحث قرار داده است.

Lavrac and Dzeroski (1994) کتابی ساده‌ای را درباره‌ی برنامه‌نویسی استقرایی منطقی ارائه می‌کنند. دیگر کتب می‌توان به Bergadano and Gunetti (1995)، Morik et al. (1993)، Muggleton (1992,1995b) اشاره کرد. فصل مربوطه‌ی Wrobel (1996) نیز دید خوبی در این زمینه ارائه می‌کند. Bratko and Muggleton (1995) خلاصه‌ی تعدادی از کاربردهای اخیر ILP در مسائل کاربردی مهم را مطرح می‌کنند. مجموعه‌ای از کارگاه‌های ILP سالانه منبع خوبی از تحقیقات اخیر در این زمینه ارائه می‌کنند (به De Raedt (1996) رجوع کنید).

## تمرینات

۱۰,۱ الگوریتمی پوشش ترتیبی مشابه CN2 و الگوریتمی پوشش همزمان مثل ID3 را در نظر بگیرید. هر دو الگوریتم برای یادگیری مفهوم هدف تعریف شده روی نمونه‌های توصیفی با عطف n متغیر منطقی به کار می‌روند. اگر ID3 درخت تصمیم متقارنی با عمق d را یاد بگیرد این درخت $2^d - 1$ گره تصمیم مستقل خواهد داشت، و بنابراین $2^d - 1$ انتخاب در هنگام ساخت فرضیه خروجی انجام می‌دهد. چه تعداد قانون برای نمایش چنین درختی به عنوان فصل دسته قوانین لازم است؟ هر یک از این قوانین چندین شرط خواهند داشت؟ یک الگوریتم پوشش ترتیبی چه تعداد انتخاب مستقل باید انجام دهد تا همین دسته قوانین را ایجاد کند؟ فکر می‌کنید کدام سیستم با داده‌های آموزشی یکی، بیشتر تمایل به overfit خواهد داشت؟

۱۰,۲ الگوریتم learn-one-rule در جدول ۱۰,۲ را چنان تغییر دهدی که بتواند قوانینی که شروطشان مقایسه‌ی ویژگی‌ها با اعداد حقیقی است را نیز یاد بگیرد (مثل temperature>42). الگوریتم خود را با تغییراتی که باید به جدول ۱۰,۲ اعمال شود مشخص کنید. راهنمایی: به همین تعمیم در یادگیری درختی توجه کنید.

۱۰,۳ الگوریتم learn-one-rule در جدول ۱۰,۲ را چنان تغییر دهه‌ی که بتواند قوانینی که شروطشان عضویت در مجموعه‌ی معلومی است را نیز یاد بگیرد (مثل $\text{nationality} \in \{Canadian, Brazilian\}$). الگوریتم تغییر یافته‌ی شما باید تمامی فرضیه‌های ممکن شامل چنین زیرمجموعه‌هایی را جستجو کند. الگوریتم خود را با تغییراتی که باید به جدول ۱۰,۲ اعمال شود مشخص کنید.

۱۰,۴ استفاده از learn-one-rule در ایجاد استراتژی جستجوی فضای فرضیه‌ای را در نظر بگیرید. در کل، ویژگی‌های جستجوی زیـر را در نظر بگیرید:

(a) آزمون‌وخطا در مقابل کنترل شده با داده‌ها

(b) کلی به جزئی در مقابل جزئی به کلی

(c) پوشش ترتیبی در مقابل پوشش همزمان

در مورد مزیت‌های انتخاب‌های الگوریتم جدول ۱۰,۱ و ۱۰,۲ بحث کنید. برای هر یک از این سه ویژگی استراتژی جستجو، تأثیر مثبت یا منفی آن را بحث کنید.

۱۰,۵ از قانون عکس دقت در فرم گزاره‌ای بر روی گزاره‌های $C = A \vee B$ و $C_1 = A \vee B \vee G$ اعمال کنید. حـداقل دو جـواب بـرای $C_2$ ارائه کنید.

۱۰,۶ از قانون عکس دقت در فرم گزاره‌ای بر روی گزاره‌های $C = R(B,x) \vee P(x,A)$ و $C_1 = S(B,y) \vee R(z,x)$ اعمال کنیـد. حداقل چهار جواب برای $C_2$ ارائه کنید. A و B ثابت و x و y متغیر هستند.

۱۰,۷ اولین مرحله‌ی عکس قانون دقت را در شکل ۱۰,۳ را در نظر بگیرید. حداقل دو حاصل مختلف ـــــ

۱۰,۸ روابط تعریفی مسئله‌ی استقرا در این فصل را در نظر بگیرید:

$$(\forall < x_i, f(x_i) > \in D)(B \wedge h \wedge x_i) \vdash f(x_i)$$

حال تعریف قبلی‌مان از بایاس استقرایی در فصل ۲ را نیز در نظر بگیرید، رابطه‌ی ۲,۱. در آنجا $B_{bias}$ را با رابطه‌ی زیر تعریف کردیم:

$$(\forall x_i \in X)(B_{bias} \wedge D \wedge x_i) \vdash L(x_i, D)$$

در این رابطه $L(x_i, D)$ دسته‌بندی‌ای است که یادگیر برای نمونه‌ی $x_i$ بعد از مشاهده‌ی داده‌های آزمایشی D انجام می‌دهد و X کل فضـای نمونه‌ای است. توجه دارید که رابطه‌ی اول قصد دارد که فرضیه‌ای را که یادگیر علاقه به پیدا کردنش را دارد توصیف کند در حالی کـه رابطـه‌ی دوم قصد دارد خط‌مشی یادگیر را برای تعمیم فرای داده‌های آموزشی را تعیین کند. یادگیری ایجاد کنید که بایـاس اسـتقرایی‌اش دقیقـاً مشـابه دانش قبلی موجود B باشد.

## فرهنگ لغات تخصصی فصل (فارسی به انگلیسی)

| | |
|---|---|
| horn clause درجه اول | first-order horn clase |
| برنامه‌نویسی منطقی | inductive logic programming (ILG) |
| برنامه‌نویسی استقرایی منطقی | inductive logic programming |
| پوشش همزمان | simultaneous covering |
| تعریف | Implementing |

| | |
|---|---|
| تکرار نسبی | relative frequency |
| ثابت‌کننده‌ی تئوری | mechanical theorem provers |
| جستجوی ستونی | beam search |
| حکم | Clause, postcondition |
| دسته قوانین | sets of rules |
| دقت | Resolution |
| روش‌های گزاره‌ای | propositional representation |
| زبان | Syntax |
| زیر روال | Subroutine |
| شرط | Precondition |
| عبارت | Literal |
| عملگرهای نتیجه‌گیری | deductive operator |
| قابلیت بیان | Expressive |
| قوی‌تر در بیان | Expressive |
| کارایی | Performance |
| کنترل نمونه‌ای | example driven |
| الگوریتم پوششی | covering |
| الگوریتم‌های ترتیبی | sequential covering algorithms |
| ممکن | candidate |
| منطق درجه اول | first order logic |
| نگاه به مرحله‌های قبلی | backtrack |
| نمادهای گزاره‌ای | predicate symbol |
| نمایش‌های گزاره‌ای | propositional representation |