Lectura

Inteligencia Artificial

Texto tomado desde los libros, Introduction to Logic Programming de Michael Genesereth y Vinay K. Chaudhri y Introduction to Artificial Intelligence Second Edition 2018 de Wolfgang Ertel.

---

1 Aplicaciones de la programación lógica

La programación lógica se puede utilizar de manera fructífera en casi cualquier área de aplicación. Sin embargo, tiene un valor especial en áreas de aplicación caracterizadas por un gran número de definiciones, restricciones y reglas de acción, especialmente cuando esas definiciones, restricciones y reglas provienen de múltiples fuentes o donde cambian con frecuencia. Las siguientes son algunas áreas de aplicación en las que la programación lógica ha demostrado ser particularmente útil.

1. Sistemas de bases de datos. Al conceptualizar las tablas de bases de datos como conjuntos de oraciones simples, es posible utilizar lógica como soporte de los sistemas de bases de datos. Por ejemplo, se puede utilizar para definir vistas virtuales de datos en términos de tablas almacenadas explícitamente, se puede utilizar para codificar restricciones en bases de datos, se puede utilizar para especificar políticas de control de acceso y se puede utilizar para escribir reglas de actualización.

2. Hojas de cálculo lógicas/hojas de trabajo. Las hojas de cálculo lógicas (a veces denominadas hojas de trabajo) generalizan las hojas de cálculo tradicionales para incluir restricciones lógicas y fórmulas aritméticas tradicionales. Abundan los ejemplos de tales limitaciones. Por ejemplo, en la programación de aplicaciones, es posible que tengamos limitaciones de tiempo o restricciones sobre quién puede reservar habitaciones. En el ámbito de las reservas de viajes, es posible que tengamos restricciones para adultos y bebés. En las hojas de programas académicos, podríamos tener restricciones sobre la cantidad de cursos de diferentes tipos que los estudiantes deben tomar.

3. Integración de datos. Se puede utilizar para relacionar los conceptos en diferentes vocabularios y así permitir a los usuarios acceder a múltiples fuentes de datos heterogéneas de manera integrada, dando a cada usuario la ilusión de una única base de datos codificada en su propio vocabulario.

4. Administración de Empresas. La programación lógica tiene un valor especial para expresar e implementar reglas comerciales de varios tipos. Las reglas comerciales internas incluyen políticas empresariales (por ejemplo, aprobación de gastos) y flujo de trabajo (quién hace qué y cuándo). Las reglas comerciales externas incluyen los detalles de los contratos con otras empresas, las reglas de configuración y fijación de precios para los productos de la compañía, etc.

5. Derecho Computacional. El derecho computacional es la rama de la informática jurídica que se ocupa de la representación de reglas y regulaciones en forma computable. La codificación de leyes en forma computable permite el análisis legal automatizado y la creación de tecnología para poner ese análisis a disposición de los ciudadanos, los monitores, los encargados de hacer cumplir y los profesionales legales.

6. Juegos generales. Estos son sistemas capaces de aceptar descripciones de juegos arbitrarios en tiempo de ejecución y pueden usar tales descripciones para jugar de manera efectiva sin intervención humana. En otras palabras, no conocen las reglas hasta que comienzan los juegos. La programación lógica se usa ampliamente aquí, como la forma preferida de formalizar las descripciones de los juegos.

Pese a todas estas aplicaciones, hay algunas limitaciones en la IA lógica.

2 Limitaciones de la lógica

2.1 El problema del espacio de búsqueda

En una búsqueda de una prueba casi siempre hay muchas (dependiendo del cálculo, potencialmente infinitas) posibilidades para la aplicación de reglas de inferencia en cada paso. El resultado es el crecimiento del espacio de búsqueda. En el peor de los casos, se deben probar todas estas posibilidades para encontrar la prueba, lo que generalmente no es posible en un período de tiempo razonable.

Si comparamos sistemas automatizados o sistemas de inferencia con matemáticos o expertos humanos que tienen experiencia en dominios especiales, hacemos observaciones interesantes. Por un lado, los matemáticos experimentados pueden demostrar teoremas que están lejos del alcance de los sistemas automatizados. Por otro lado, los sistemas automatizados realizan decenas de miles de inferencias por segundo. En contraste, un humano realiza tal vez una inferencia por segundo. Aunque los expertos humanos son mucho más lentos a nivel de objeto (es decir, en la realización de inferencias), aparentemente resuelven problemas difíciles mucho más rápido.

Hay varias razones para esto. Los humanos usamos cálculos intuitivos que funcionan en un nivel superior y, a menudo, llevamos a cabo muchas de las inferencias simples de un comprobador automatizado en un solo paso. Además, utilizamos lemas, es decir, fórmulas verdaderas derivadas de las que ya conocemos y, por lo tanto, no es necesario volver a probarlas cada vez. Mientras tanto, también hay máquinas de prueba que trabajan con tales métodos. Pero incluso ellas todavía no pueden competir con los expertos humanos.

Otra ventaja mucho más importante de nosotros, los humanos, es la intuición, sin la cual no podríamos resolver ningún problema difícil. El intento de formalizar la intuición causa problemas. La experiencia en proyectos de IA aplicada muestra que en dominios complejos como la medicina o las matemáticas, la mayoría de los expertos son incapaces de formular verbalmente este metaconocimiento intuitivo y mucho menos de formalizarlo. Por lo tanto, no podemos programar este conocimiento o integrarlo en cálculos en forma de `heurística`. Las heurísticas son métodos que en muchos casos pueden simplificar o acortar enormemente el camino hacia la meta, pero en algunos casos (generalmente raramente) pueden alargar mucho el camino hacia la meta. La búsqueda heurística es importante no solo para la lógica, sino en general para la resolución de problemas en la IA.

Un enfoque interesante, que se ha seguido aproximadamente desde 1990, es la aplicación de técnicas de aprendizaje automático al aprendizaje de heurísticas para dirigir la búsqueda de sistemas de inferencia. Un probador de resolución tiene, durante la búsqueda de una prueba, cientos o más posibilidades de pasos de resolución, pero solo unos pocos conducen a la meta. Sería ideal si el probador pudiera preguntarle a un sistema inteligente, cuales de dos cláusulas debería usar en el siguiente paso para encontrar rápidamente la prueba. Hay intentos de construir tales módulos de dirección de pruebas, que evalúan las diversas alternativas para el siguiente paso y luego eligen la alternativa con la mejor calificación. En el caso de la resolución, la calificación de las cláusulas disponibles podría calcularse mediante una función que calcula un valor en función del número de literales positivos, la complejidad de los términos, etc., para cada par de cláusulas resolubles.

¿Cómo se puede implementar esta función? Debido a que este conocimiento es `intuitivo`, el programador no está familiarizado con eso. En su lugar, uno intenta copiar la naturaleza y usa algoritmos de aprendizaje automático para aprender de pruebas exitosas. Los atributos de todos los pares de cláusulas que participan en los pasos de resolución satisfactorios se almacenan como positivos y los atributos de todas las resoluciones fallidas se almacenan como negativo. Luego, utilizando estos datos de entrenamiento

y un sistema de aprendizaje automático se genera un programa que puede calificar los pares de cláusulas de manera heurística.

Se sigue un enfoque diferente y más exitoso para mejorar el razonamiento matemático con sistemas interactivos que operan bajo el control del usuario. Aquí se podrían nombrar programas informáticos como Mathematica, Maple o Maxima, que pueden realizar automáticamente difíciles manipulaciones matemáticas simbólicas. La búsqueda de la prueba, sin embargo, se deja completamente al ser humano. Actualmente existen varios proyectos, como `Omega` y `MKM` para el desarrollo de sistemas de apoyo a los matemáticos durante las pruebas.

En resumen, se puede decir que, debido al problema del espacio de búsqueda, los probadores automáticos de hoy solo pueden probar teoremas relativamente simples en dominios especiales con pocos axiomas.

2.2 Decidibilidad e incompletitud

La lógica de predicados de primer orden proporciona una herramienta poderosa para la representación del conocimiento y el razonamiento. Sabemos que hay probadores de teoremas y cálculos correctos y completos. Cuando se prueba un teorema, es decir, un enunciado verdadero, dicho probador es muy útil porque, debido a que está completo, uno sabe después de un tiempo finito que el enunciado realmente es verdadero. ¿Qué pasa si la afirmación no es cierta? El teorema de completitud no responde a esta pregunta. Específicamente, no existe un proceso que pueda probar o refutar cualquier fórmula de $PL1$ en tiempo finito, ya que se sostiene que:

Teorema: El conjunto de fórmulas válidas en la lógica de predicados de primer orden es semidecidible.

Este teorema implica que hay programas (probadores de teoremas) que, dada una fórmula verdadera (válida) como entrada, determinan su verdad en un tiempo finito. Sin embargo, si la fórmula no es válida, puede suceder que el probador nunca se detenga. La lógica proposicional es decidible porque el método de las tabla de verdad proporciona todos los modelos de una fórmula en un tiempo finito. Evidentemente, la lógica de predicados con cuantificadores y símbolos de funciones anidadas es un lenguaje demasiado poderoso para ser decidible.

Por otro lado, la lógica de predicados no es lo suficientemente poderosa para muchos propósitos. A menudo, uno desea hacer declaraciones sobre conjuntos de predicados o funciones. Esto no funciona en PL1 porque solo conoce cuantificadores para variables, pero no para predicados o funciones.

Kurt Gödel demostró, poco después de su teorema de completitud para PL1, que la completitud se pierde si ampliamos PL1 aunque sea mínimamente para construir una lógica de orden superior. Una lógica de primer orden solo puede cuantificar sobre variables. Una lógica de segundo orden también puede cuantificar sobre fórmulas de primer orden y una lógica de tercer orden puede cuantificar sobre fórmulas de segundo orden. Incluso agregar solo el axioma de inducción para los números naturales hace que la lógica sea incompleta. El enunciado `Si un predicado p(n) se cumple para n, entonces p(n + 1)` también se cumple, o

$$\forall\ p\ p(n) \Rightarrow p(n+1)$$

es una proposición de segundo orden porque cuantifica sobre un predicado. Gödel demostró el siguiente teorema:

Teorema de incompletitud de Gödel. Todo sistema de axiomas para los números naturales con suma y multiplicación (aritmética) está incompleto. Es decir, hay afirmaciones verdaderas en aritmética que no son probables.

La demostración de Gödel funciona con lo que se llama Gödelización, en la que cada fórmula aritmética se codifica como un número. Se obtiene un número único de Gödel . La Gödelización ahora se usa para formular la proposición:

```
F = No soy demostrable
```

en el lenguaje de la aritmética. Esta fórmula es cierta por la siguiente razón. Supongamos que F es falso. Entonces podemos demostrar F y por tanto, demostrar que F no es demostrable. Ésta es una contradicción.

Por tanto, F es verdadera y por tanto, no demostrable.

El trasfondo de este teorema es que las teorías matemáticas (sistemas de axiomas) y, más en general, los lenguajes se vuelven incompletos si el lenguaje se vuelve demasiado poderoso. Un ejemplo similar es la teoría de conjuntos. Este lenguaje es tan poderoso que se pueden formular paradojas con él. Sin embargo, esto no significa que las lógicas de orden superior sean totalmente inadecuadas para los métodos formales. Ciertamente, existen sistemas formales así como también probadores de lógicas de orden superior.

2.3 El pingüino volador

Con un ejemplo sencillo demostraremos un problema fundamental de lógica y posibles enfoques de solución. Dadas las declaraciones:

1. Tweety es un pingüino
2. Los pingüinos son pájaros
3. Los pájaros pueden volar

Formalizado en PL1, la base de conocimientos KB da como resultado:

$$\begin{aligned} &\text{pingüino(tweety)} \\ \text{pingüino(x)} &\Rightarrow \text{pájaro(x)} \\ \text{pájaro(x)} &\Rightarrow \text{vuela(x)} \end{aligned}$$

A partir de esto, se puede derivar `vuela(tweety)`. Evidentemente, la formalización de los atributos de vuelo de los pingüinos es insuficiente. Probamos la declaración adicional `los pingüinos no pueden volar`, que es:

$$\text{pingüino} \Rightarrow \neg\text{vuela(x)}$$

De ahí se puede derivar ¬vuela(tweety). Pero `volar(tweety)` sigue siendo cierto. Por tanto, la base de conocimientos es inconsistente. Aquí notamos una característica importante de la lógica, a saber, la monotonía. Aunque declaramos explícitamente que los pingüinos no pueden volar, aún se puede derivar lo contrario.

Definición: Una lógica se llama monótona si para una base de conocimiento arbitraria KB y una fórmula arbitraria ϕ, el conjunto de fórmulas derivables de KB es un subconjunto de las fórmulas derivables de $KB \cup \phi$.

Si se amplía un conjunto de fórmulas, entonces, después de la ampliación, todavía se pueden probar todos los enunciados previamente derivables y potencialmente también se pueden probar enunciados adicionales. El conjunto de enunciados demostrables crece así monótonamente cuando se amplía el conjunto de fórmulas. Para nuestro ejemplo, esto significa que la extensión de la base de conocimientos nunca conducirá a nuestro objetivo. Por lo tanto, modificamos KB reemplazando la declaración obviamente falsa, `(todas) las aves pueden volar` por la declaración más exacta, `(todas) las aves excepto los pingüinos pueden volar` y obtenemos como KB_2 las siguientes cláusulas:

$$\begin{aligned} &\text{pingüino(tweety)} \\ &\text{pingüino(x)} \Rightarrow \text{pájaro(x)} \\ &\text{pájaro(x)} \wedge \neg\text{pingüino(x)} \Rightarrow \text{vuela(x)} \\ &\text{pingüino(x)} \Rightarrow \neg\text{vuela(x)} \end{aligned}$$

Podemos derivar ¬vuela(tweety) pero no `vuela(tweety)`, porque para eso necesitaríamos ¬pingüino(x), que, sin embargo, no es derivable. Mientras solo haya pingüinos en este mundo, reina la paz. Sin embargo, todo pájaro normal causa problemas de inmediato. Deseamos agregar el cuervo Abraxas y obtenemos KB_3:

cuervo(tweety)
cuervo(x) $\Rightarrow$ pájaro(x)
pingüino(tweety)
pingüino(x) $\Rightarrow$ pájaro(x)
pájaro(x) $\wedge$ $\neg$pingüino(x) $\Rightarrow$ vuela(x)
pingüino(x) $\Rightarrow$ $\neg$vuela(x)

No podemos decir nada sobre los atributos de vuelo de Abraxas porque olvidamos formular que los cuervos no son pingüinos. Por lo tanto, ampliamos KB_3 a KB_4:

cuervo(tweety)
cuervo(x) $\Rightarrow$ pájaro(x)
cuervo(x) $\Rightarrow$ $\neg$pingüino(x)
pingüino(tweety)
pingüino(x) $\Rightarrow$ pájaro(x)
pájaro(x) $\wedge$ $\neg$pingüino(x) $\Rightarrow$ vuela(x)
pingüino(x) $\Rightarrow$ $\neg$vuela(x)

El hecho de que los cuervos no son pingüinos, que es evidente para los humanos, debe agregarse aquí explícitamente. Por lo tanto, para la construcción de una base de conocimientos con los aproximadamente 9.800 tipos de aves en todo el mundo, debe especificarse para cada tipo de ave (excepto los pingüinos) que no es miembro de los pingüinos. Debemos proceder de manera análoga para todas las demás excepciones, como el avestruz.

Para cada objeto en la base de conocimiento, además de sus atributos, se deben enumerar todos los atributos que no tiene. Para resolver este problema, se han desarrollado diversas formas de lógica no monótona, que permiten eliminar conocimientos (fórmulas) de la base de conocimientos. Bajo el nombre de lógica por defecto, se han desarrollado lógicas que permiten asignar atributos a los objetos que son válidos siempre que no se disponga de otras reglas. En el ejemplo de Tweety, la regla que los `pájaros pueden volar` sería una regla predeterminada. A pesar de un gran esfuerzo, estas lógicas en la actualidad, debido a problemas semánticos y prácticos, no han tenido éxito.

La monotonía puede resultar especialmente inconveniente en problemas de planificación complejos en los que el mundo puede cambiar. Si, por ejemplo, una casa azul se pinta de rojo, luego se pinta de rojo. Una base de conocimientos como:

color(casa, azúl)
pintar(casa, rojo)
pintar(x,y) $\Rightarrow$ color(x,y)

lleva a la conclusión de que, después de pintar, la casa es roja y azul. El problema que surge aquí en la planificación se conoce como el problema del marco (frame problem).

Un enfoque interesante para modelar problemas como el ejemplo de Tweety es la teoría de la probabilidad. La afirmación `todos los pájaros pueden volar` es falsa. Una afirmación como `casi todos los pájaros pueden volar` es correcta. Esta afirmación se vuelve más exacta si damos una probabilidad de que `los pájaros pueden volar`. Esto conduce a la lógica probabilística, que en la actualidad representa una subárea importante de la IA y una herramienta importante para modelar la incertidumbre.

2.4 Modelado de la incertidumbre

La lógica de dos valores solo puede y debe modelar circunstancias en las que existen valores verdaderos, falsos y no hay otros valores de verdad. Para muchas tareas del razonamiento cotidiano, la lógica de dos valores no es lo suficientemente expresiva. La regla:

$$\text{pájaro(x)} \Rightarrow \text{vuela(x)}$$

es cierta para casi todas las aves, pero para algunas es falsa. Como ya se mencionó, trabajar con probabilidades permite una formulación exacta de la incertidumbre. La afirmación 99% `de todas las aves pueden volar` se puede formalizar con la expresión:

$$P(\text{pájaro(x)} \Rightarrow \text{vuela(x)}) = 0.99$$

Es mejor trabajar con probabilidades condicionales como:

$$P(\text{vuela} \mid \text{pájaro}) = 0.99.$$

Con la ayuda de las redes bayesianas, también se pueden modelar aplicaciones complejas con muchas variables.

Se necesita un modelo diferente para la declaración `el clima es agradable`. Aquí a menudo no tiene sentido hablar en términos de verdadero y falso. La variable `clima` no debe modelarse como binaria, sino continuamente con valores. Por ejemplo, en el intervalo $[0, 1]$, `clima` = 0.7 significa `el clima es bastante agradable`. La lógica difusa se desarrolló para este tipo de variable continua (difusa).

La teoría de la probabilidad también ofrece la posibilidad de realizar afirmaciones sobre la probabilidad de variables continuas. Una declaración en el informe meteorológico como `hay una alta probabilidad de que llueva` podría, por ejemplo, formularse exactamente como una densidad de probabilidad de la forma:

$$P(rainfall = X) = Y$$

Esta representación muy general e incluso visualizable de ambos tipos de incertidumbre que hemos comentado, junto con la estadística inductiva y la teoría de las redes bayesianas, permite, en principio, dar respuesta a consultas probabilísticas arbitrarias. La teoría de la probabilidad y la lógica difusa no son directamente comparables a la lógica de predicados porque no permiten variables o cuantificadores. Por tanto, pueden verse como extensiones de la lógica proposicional, como se muestra en la siguiente tabla:

Formalismo	Número de valores verdaderos	Probabilidades Expresables
Lógica Proposicional	2	-
Lógica difusa	∞	-
Lógica probabilística discreta	n	Si
Lógica probabilística continua	∞	Si