VALIDEZ

TAUTOLOGÍA

El segundo concepto que necesitaremos es el de **validez**. Una sentencia es válida si es verdadera en *todos* los modelos. Por ejemplo, la sentencia $P \vee \neg P$ es una sentencia válida. Las sentencias válidas también se conocen como **tautologías**, son *necesariamente* verdaderas y por lo tanto vacías de significado. Como la sentencia *Verdadero* es verdadera en todos los modelos, toda sentencia válida es lógicamente equivalente a *Verdadero*.

TEOREMA DE LA DEDUCCIÓN

¿Qué utilidad tienen las sentencias válidas? De nuestra definición de implicación podemos derivar el **teorema de la deducción**, que ya se conocía por los Griegos antiguos:

Para cualquier sentencia α y β, $\alpha \models \beta$ si y sólo si la sentencia $(\alpha \Rightarrow \beta)$ es válida.

(En el Ejercicio 7.4 se pide demostrar una serie de aserciones.) Podemos pensar en el algoritmo de inferencia de la Figura 7.10 como en un proceso para averiguar la validez de $(BC \Rightarrow \alpha)$. A la inversa, cada sentencia que es una implicación válida representa una inferencia correcta.

SATISFACIBILIDAD

SATISFACE

El último concepto que necesitaremos es el de ***satisfacibilidad***. Una sentencia es satisfactoria si es verdadera para *algún* modelo. Por ejemplo, en la base de conocimiento ya mostrada, $(R_1 \wedge R_2 \wedge R_3 \wedge R_4 \wedge R_5)$ es *satisfacible* porque hay tres modelos en los que es verdadera, tal como se muestra en la Figura 7.9. Si una sentencia α es verdadera en un modelo m, entonces decimos que m **satisface** α, o que m **es un modelo de** α. La *satisfacibilidad* se puede averiguar enumerando los modelos posibles hasta que uno satisface la sentencia. La determinación de la *satisfacibilidad* de sentencias en lógica proposicional fue el primer problema que se demostró que era NP-completo.

Muchos problemas en las ciencias de la computación son en realidad problemas de *satisfacibilidad*. Por ejemplo, todos los problemas de satisfacción de restricciones del Capítulo 5 se preguntan esencialmente si un conjunto de restricciones se satisfacen dada una asignación. Con algunas transformaciones adecuadas, los problemas de búsqueda también se pueden resolver mediante *satisfacibilidad*. La validez y la *satisfacible* están íntimamente relacionadas: α es válida si y sólo si $\neg\alpha$ es *insatisfacible*; en contraposición, α es *satisfacible* si y sólo si $\neg\alpha$ no es válida.

$\alpha \models \beta$ si y sólo si la sentencia $(\alpha \wedge \neg\beta)$ es insatisfactoria.

REDUCTIO AD ABSURDUM

REFUTACIÓN

La demostración de β a partir de α averiguando la insatisfacibilidad de $(\alpha \wedge \neg\beta)$ se corresponde exactamente con la técnica de demostración en matemáticas de la *reductio ad absurdum* (que literalmente se traduce como «reducción al absurdo»). Esta técnica también se denomina demostración mediante **refutación** o demostración por **contradicción**. Asumimos que la sentencia β es falsa y observamos si se llega a una contradicción con las premisas en α. Dicha contradicción es justamente lo que queremos expresar cuando decimos que la sentencia $(\alpha \wedge \neg\beta)$ es *insatisfacible*.

7.5 Patrones de razonamiento en lógica proposicional

Esta sección cubre los patrones estándar de inferencia que se pueden aplicar para derivar cadenas de conclusiones que nos llevan al objetivo deseado. Estos patrones de infe-

REGLAS DE INFERENCIA

rencia se denominan **reglas de inferencia**. La regla más conocida es la llamada **Modus Ponens** que se escribe como sigue:

MODUS PONENS

$$\frac{\alpha \Rightarrow \beta, \quad \alpha}{\beta}$$

La notación nos dice que, cada vez que encontramos dos sentencias en la forma $\alpha \Rightarrow \beta$ y α, entonces la sentencia β puede ser inferida. Por ejemplo, si tenemos (*WumpusEnFrente* $\wedge$ *WumpusVivo*) $\Rightarrow$ *Disparar* y (*WumpusEnFrente* $\wedge$ *WumpusVivo*), entonces se puede inferir *Disparar*.

ELIMINACIÓN-$\wedge$

Otra regla de inferencia útil es la **Eliminación-$\wedge$**, que expresa que, de una conjunción se puede inferir cualquiera de sus conjuntores:

$$\frac{\alpha \wedge \beta}{\alpha}$$

Por ejemplo, de (*WumpusEnFrente* $\wedge$ *WumpusVivo*), se puede inferir *WumpusVivo*.

Teniendo en cuenta los posibles valores de verdad de α y β se puede observar fácilmente, de una sola vez, que el Modus Ponens y la Eliminación-$\wedge$ son reglas sólidas. Estas reglas se pueden utilizar sobre cualquier instancia en la que es aplicable, generando inferencias sólidas, sin la necesidad de enumerar todos los modelos.

Todas las equivalencias lógicas de la Figura 7.11 se pueden utilizar como reglas de inferencia. Por ejemplo, la equivalencia de la eliminación de la bicondicional nos lleva a las dos reglas de inferencia

$$\frac{\alpha \Leftrightarrow \beta}{(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)} \quad \text{y} \quad \frac{(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)}{\alpha \Leftrightarrow \beta}$$

Pero no todas las reglas de inferencia se pueden usar, como ésta, en ambas direcciones. Por ejemplo, no podemos utilizar el Modus Ponens en la dirección opuesta para obtener $\alpha \Rightarrow \beta$ y α a partir de β.

Veamos cómo se pueden usar estas reglas de inferencia y equivalencias en el mundo de *wumpus*. Comenzamos con la base de conocimiento conteniendo R_1 a R_5, y mostramos cómo demostrar $\neg H_{1,2}$, es decir, que no hay un hoyo en la casilla [1, 2]. Primero aplicamos la eliminación de la bicondicional a R_2 para obtener

$$R_6: \quad (B_{1,1} \Rightarrow (H_{1,2} \vee H_{2,1})) \wedge ((H_{1,2} \vee H_{2,1}) \Rightarrow \mathrm{B}_{1,1})$$

Entonces aplicamos la Eliminación-$\wedge$ a R_6 para obtener

$$R_7: \quad ((H_{1,2} \vee H_{2,1}) \Rightarrow B_{1,1})$$

Y por la equivalencia lógica de contraposición obtenemos

$$R_8: \quad (\neg B_{1,1} \Rightarrow \neg(H_{1,2} \vee H_{2,1}))$$

Ahora aplicamos el Modus Ponens con R_8 y la percepción R_4 (por ejemplo, $\neg \mathrm{B}_{1,1}$), para obtener

$$R_9: \quad \neg(H_{1,2} \vee H_{2,1})$$

Finalmente, aplicamos la ley de Morgan, obteniendo la conclusión

$$R_{10}: \neg H_{1,2} \wedge \neg H_{2,1}$$

Es decir, ni la casilla [1, 2] ni la [2, 1] contienen un hoyo.

PRUEBA

A la derivación que hemos realizado (una secuencia de aplicaciones de reglas de inferencia) se le denomina una **prueba** (o **demostración**). Obtener una prueba es muy semejante a encontrar una solución en un problema de búsqueda. De hecho, si la función sucesor se define para generar todas las aplicaciones posibles de las reglas de inferencia, entonces todos los algoritmos de búsqueda del Capítulo 3 se pueden utilizar para obtener una prueba. De esta manera, la búsqueda de pruebas es una alternativa a tener que enumerar los modelos. La búsqueda se puede realizar hacia delante a partir de la base de conocimiento inicial, aplicando las reglas de inferencia para derivar la sentencia objetivo, o hacia atrás, desde la sentencia objetivo, intentando encontrar una cadena de reglas de inferencia que nos lleven a la base de conocimiento inicial. Más adelante, en esta sección, veremos dos familias de algoritmos que utilizan estas técnicas.

El hecho de que la inferencia en lógica proposicional sea un problema NP-completo nos hace pensar que, en el peor de los casos, la búsqueda de pruebas va a ser no mucho más eficiente que la enumeración de modelos. Sin embargo, en muchos casos prácticos, *encontrar una prueba puede ser altamente eficiente simplemente porque el proceso puede ignorar las proposiciones irrelevantes, sin importar cuántas de éstas haya.* Por ejemplo, la prueba que hemos visto que nos llevaba a $\neg H_{1,2} \wedge \neg H_{2,1}$ no utiliza las proposiciones $B_{2,1}$, $H_{1,1}$, $H_{2,2}$ o $H_{3,1}$. Estas proposiciones se pueden ignorar porque la proposición objetivo $H_{1,2}$ sólo aparece en la sentencia R_4, y la otra proposición de R_4 sólo aparece también en R_2; por lo tanto, R_1, R_3 y R_5 no juegan ningún papel en la prueba. Sucedería lo mismo aunque añadiésemos un millón de sentencias a la base de conocimiento; por el otro lado, el algoritmo de la tabla de verdad, aunque sencillo, quedaría saturado por la explosión exponencial de los modelos.

MONÓTONO

Esta propiedad de los sistemas lógicos en realidad proviene de una característica mucho más fundamental, denominada **monótono**. La característica de monotonismo nos dice que el conjunto de sentencias implicadas sólo puede *aumentar* (pero no cambiar) al añadirse información a la base de conocimiento[10]. Para cualquier sentencia α y β,

$$\text{si} \quad BC \models \alpha \quad \text{entonces} \quad BC \wedge \beta \models \alpha$$

Por ejemplo, supongamos que la base de conocimiento contiene una aserción adicional β, que nos dice que hay exactamente ocho hoyos en el escenario. Este conocimiento podría ayudar al agente a obtener conclusiones *adicionales*, pero no puede invalidar ninguna conclusión α ya inferida (como la conclusión de que no hay un hoyo en la casilla [1, 2]). El monotonismo permite que las reglas de inferencia se puedan aplicar siempre que se hallen premisas aplicables en la base de conocimiento; la conclusión de la regla debe permanecer *sin hacer caso de qué más hay en la base de conocimiento.*

[10] Las lógicas **No Monótonas**, que violan la propiedad de monotonismo, modelan una característica propia del razonamiento humano: cambiar de opinión. Estas lógicas se verán en la Sección 10.7.

Resolución

Hemos argumentado que las reglas de inferencia vistas hasta aquí son *sólidas*, pero no hemos visto la cuestión acerca de lo *completo* de los algoritmos de inferencia que las utilizan. Los algoritmos de búsqueda como el de búsqueda en profundidad iterativa (página 87) son completos en el sentido de que éstos encontrarán cualquier objetivo alcanzable, pero si las reglas de inferencia no son adecuadas, entonces el objetivo no es alcanzable; no existe una prueba que utilice sólo esas reglas de inferencia. Por ejemplo, si suprimimos la regla de eliminación de la bicondicional la prueba de la sección anterior no avanzaría. En esta sección se introduce una regla de inferencia sencilla, la **resolución**, que nos lleva a un algoritmo de inferencia completo cuando se empareja a un algoritmo de búsqueda completo.

Comenzaremos utilizando una versión sencilla de la resolución aplicada al mundo de *wumpus*. Consideremos los pasos que nos llevaban a la Figura 7.4(a): el agente vuelve de la casilla [2, 1] a la [1, 1] y entonces va a la casilla [1, 2], donde percibe un hedor, pero no percibe una corriente de aire. Ahora añadimos los siguientes hechos a la base de conocimiento:

$$R_{11}: \quad \neg B_{1,2}$$
$$R_{12}: \quad B_{1,2} \Leftrightarrow (H_{1,1} \vee H_{2,2} \vee H_{1,3})$$

Mediante el mismo proceso que nos llevó antes a R_{10}, podemos derivar que no hay ningún hoyo en la casilla [2, 2] o en la [1, 3] (recuerde que se sabe que en la casilla no había ninguna percepción de hoyos):

$$R_{13}: \quad \neg \mathrm{H}_{2,2}$$
$$R_{14}: \quad \neg \mathrm{H}_{1,3}$$

También podemos aplicar la eliminación de la bicondicional a la R_3, seguido del Modus Ponens con la R_5, para obtener el hecho de que puede haber un hoyo en la casilla [1, 1], la [2, 2] o la [3, 1]:

$$R_{15}: \quad \mathrm{H}_{1,1} \vee \mathrm{H}_{2,2} \vee \mathrm{H}_{3,1}$$

Ahora viene la primera aplicación de la regla de resolución: el literal $\neg \mathrm{H}_{2,2}$ de la R_{13} *se resuelve con* el literal $\mathrm{H}_{2,2}$ de la R_{15}, dando el *resolvente*

$$R_{16}: \quad \mathrm{H}_{1,1} \vee \mathrm{H}_{3,1}$$

En lenguaje natural: si hay un hoyo en la casilla [1, 1], o en la [2, 2], o en la [3, 1], y no hay ninguno en la [2, 2], entonces hay uno en la [1, 1] o en la [3, 1]. De forma parecida, el literal $\neg \mathrm{H}_{1,1}$ de la R_1 se resuelve con el literal $\mathrm{H}_{1,1}$ de la R_{16}, dando

$$R_{17}: \quad \mathrm{H}_{3,1}$$

En lenguaje natural: si hay un hoyo en la casilla [1, 1] o en la [3, 1], y no hay ninguno en la [1, 1], entonces hay uno en la [3, 1]. Los últimos dos pasos de inferencia son ejemplo de la regla de inferencia de **resolución unitaria**,

RESOLUCIÓN UNITARIA

$$\frac{\ell_1 \vee \cdots \vee \ell_k, \quad m}{\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k}$$

en donde cada ℓ es un literal y ℓ_i y m son **literales complementarios** (por ejemplo, uno es la negación del otro). Así, la resolución unitaria toma una **cláusula** (una disyunción de literales) y un literal para producir una nueva cláusula. Fíjese en que un literal se puede ver como una disyunción con un solo literal, conocido como **cláusula unitaria**.

LITERALES COMPLEMENTARIOS

CLÁUSULA

CLÁUSULA UNITARIA

La regla de resolución unitaria se puede generalizar a la regla general de **resolución**,

$$\frac{\ell_1 \vee \cdots \vee \ell_k, \quad m_1 \vee \cdots \vee m_n}{\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n}$$

donde ℓ_i y m_j son literales complementarios. Si sólo tratáramos con cláusulas de longitud dos, podríamos escribir la regla así

$$\frac{\ell_1 \vee \ell_2, \quad \neg\ell_2 \vee \ell_3}{\ell_1 \vee \ell_3}$$

Es decir, la resolución toma dos cláusulas y genera una cláusula nueva con los literales de las dos cláusulas originales *menos* los literales complementarios. Por ejemplo, tendríamos

$$\frac{P_{1,1} \vee P_{3,1}, \quad \neg P_{1,1} \vee \neg P_{2,2}}{P_{3,1} \vee \neg P_{2,2}}$$

Hay otro aspecto técnico relacionado con la regla de resolución: la cláusula resultante debería contener sólo una copia de cada literal[11]. Se le llama **factorización** al proceso de eliminar las copias múltiples de los literales. Por ejemplo, si resolvemos $(A \vee B)$ con $(A \vee \neg B)$ obtenemos $(A \vee A)$, que se reduce a A.

FACTORIZACIÓN

La *solidez* de la regla de resolución se puede ver fácilmente si consideramos el literal ℓ_i. Si ℓ_i es verdadero, entonces m_j es falso, y de aquí $m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n$ debe ser verdadero, porque se da $m_1 \vee \cdots \vee m_n$. Si ℓ_i es falso, entonces $\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k$ debe ser verdadero, porque se da $\ell_1 \vee \cdots \vee \ell_k$. Entonces l_i es o bien verdadero o bien falso, y así, se obtiene una de las dos conclusiones, exactamente tal cómo establece la regla de resolución.

Lo que es más sorprendente de la regla de resolución es que crea la base para una familia de procedimientos de inferencia *completos*. *Cualquier algoritmo de búsqueda completo, aplicando sólo la regla de resolución, puede derivar cualquier conclusión implicada por cualquier base de conocimiento en lógica proposicional*. Pero hay una advertencia: la resolución es completa en un sentido muy especializado. Dado que A sea

[11] Si una cláusula se ve como un conjunto de literales, entonces esta restricción se respeta de forma automática. Utilizar la notación de conjuntos para representar cláusulas hace que la regla de resolución sea más clara, con el coste de introducir una notación adicional.

verdadero, no podemos utilizar la resolución para generar de forma automática la consecuencia $A \vee B$. Sin embargo, podemos utilizar la resolución para responder a la pregunta de si $A \vee B$ es verdadero. Este hecho se denomina **completitud de la resolución**, que indica que la resolución se puede utilizar siempre para confirmar o refutar una sentencia, pero no se puede usar para enumerar sentencias verdaderas. En las dos siguientes secciones explicamos cómo la resolución lleva a cabo este proceso.

COMPLETITUD DE LA RESOLUCIÓN

Forma normal conjuntiva

La regla de resolución sólo se puede aplicar a disyunciones de literales, por lo tanto, sería muy importante que la base de conocimiento y las preguntas a ésta estén formadas por disyunciones. Entonces, ¿cómo nos lleva esto a un procedimiento de inferencia completa para toda la lógica proposicional? La respuesta es que *toda sentencia en lógica proposicional es equivalente lógicamente a una conjunción de disyunciones de literales*. Una sentencia representada mediante una conjunción de disyunciones de literales se dice que está en **forma normal conjuntiva** o **FNC**. Que lo consideraremos bastante útil más tarde, al tratar la reducida familia de sentencias ***k*-FNC**. Una sentencia k-FNC tiene exactamente k literales por cláusula:

FORMA NORMAL CONJUNTIVA

K-FNC

$$(\ell_{1,1} \vee \ldots \vee \ell_{1,k}) \wedge \ldots \wedge (\ell_{n,1} \vee \ldots \vee \ell_{n,k})$$

De manera que se puede transformar cada sentencia en una sentencia de tipo 3-FNC, la cual tiene un conjunto de modelos equivalente.

Mejor que demostrar estas afirmaciones (*véase* el Ejercicio 7.10), vamos a describir un procedimiento de conversión muy sencillo. Vamos a ilustrar el procedimiento con la conversión de R_2, la sentencia $B_{1,1} \Leftrightarrow (H_{1,2} \vee H_{2,1})$, a FNC. Los pasos a seguir son los siguientes:

1. Eliminar $\Leftrightarrow$, sustituyendo $\alpha \Leftrightarrow \beta$ por $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$.

$$(B_{1,1} \Rightarrow (H_{1,2} \vee H_{2,1})) \wedge ((H_{1,2} \vee H_{2,1}) \Rightarrow B_{1,1})$$

2. Eliminar $\Rightarrow$, sustituyendo $\alpha \Rightarrow \beta$ por $\neg\alpha \vee \beta$.

$$(\neg B_{1,1} \vee H_{1,2} \vee H_{2,1}) \wedge (\neg(H_{1,2} \vee H_{2,1}) \vee B_{1,1})$$

3. Una FNC requiere que la $\neg$ se aplique sólo a los literales, por lo tanto, debemos «anidar las $\neg$» mediante la aplicación reiterada de las siguientes equivalencias (sacadas de la Figura 7.11).

$$\neg(\neg\alpha) \equiv \alpha \text{ (eliminación de la doble negación)}$$
$$\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta) \text{ (de Morgan)}$$
$$\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta) \text{ (de Morgan)}$$

En el ejemplo, sólo necesitamos una aplicación de la última regla:

$$(\neg B_{1,1} \vee H_{1,2} \vee H_{2,1}) \wedge ((\neg H_{1,2} \wedge \neg H_{2,1}) \vee B_{1,1})$$

4. Ahora tenemos una sentencia que tiene una $\wedge$ con operadores de $\vee$ anidados, aplicados a literales y a una $\wedge$ anidada. Aplicamos la ley de distributividad de la Figura 7.11, distribuyendo la $\vee$ sobre la $\wedge$ cuando nos es posible.

$$(\neg B_{1,1} \vee H_{1,2} \vee H_{2,1}) \wedge (\neg H_{1,2} \vee B_{1,1}) \wedge (\neg H_{2,1} \vee \mathrm{B}_{1,1})$$

La sentencia inicial ahora está en FNC, una conjunción con tres cláusulas. Es más difícil de leer pero se puede utilizar como entrada en el procedimiento de resolución.

Un algoritmo de resolución

Los procedimientos de inferencia basados en la resolución trabajan utilizando el principio de prueba mediante contradicción que vimos al final de la Sección 7.4. Es decir, para demostrar que $BC \models \alpha$, demostramos que $(BC \wedge \neg\alpha)$ es *insatisfacible*. Lo hacemos demostrando una contradicción.

En la Figura 7.12 se muestra un algoritmo de resolución. Primero se convierte $(BC \wedge \neg\alpha)$ a FNC. Entonces, se aplica la regla de resolución a las cláusulas obtenidas. Cada par que contiene literales complementarios se resuelve para generar una nueva cláusula, que se añade al conjunto de cláusulas si no estaba ya presente. El proceso continúa hasta que sucede una de estas dos cosas:

- No hay nuevas cláusulas que se puedan añadir, en cuyo caso α no implica β, o
- Se deriva la cláusula vacía de una aplicación de la regla de resolución, en cuyo caso α implica β.

La cláusula vacía (una disyunción sin disyuntores) es equivalente a *Falso* porque una disyunción es verdadera sólo si al menos uno de sus disyuntores es verdadero. Otra forma de ver que la cláusula vacía representa una contradicción es observar que se presenta sólo si se resuelven dos cláusulas unitarias complementarias, tales como P y ¬P.

```
función RESOLUCIÓN-LP(BC, α) devuelve verdadero o falso
  entradas: BC, la base de conocimiento, una sentencia en lógica proposicional
            α, la petición, una sentencia en lógica proposicional

  cláusulas ← el conjunto de cláusulas de BC ∧ ¬α en representación FNC
  nueva ← { }
  bucle hacer
    para cada C_i, C_j en cláusulas hacer
      resolventes ← RESUELVE-LP(C_i, C_j)
      si resolventes contiene la cláusula vacía entonces devolver verdadero
      nueva ← nueva ∪ resolventes
    si nueva ⊆ cláusulas entonces devolver falso
    cláusulas ← cláusulas ∪ nueva
```

Figura 7.12 Un algoritmo sencillo de resolución para la lógica proposicional. La función RESUELVE-LP devuelve el conjunto de todas las cláusulas posibles que se obtienen de resolver las dos entradas.

Ahora podemos aplicar el procedimiento de resolución a una inferencia sencilla del mundo de *wumpus*. Cuando el agente está en la casilla [1, 1] no percibe ninguna brisa, por lo tanto no puede haber hoyos en las casillas vecinas. Las sentencias relevantes en la base de conocimiento son

$$BC = R_2 \wedge R_4 = (B_{1,1} \Leftrightarrow (H_{1,2} \vee H_{2,1})) \wedge \neg B_{1,1}$$

y deseamos demostrar α, es decir $\neg H_{1,2}$. Cuando convertimos $(BC \wedge \neg\alpha)$ a FNC obtenemos las cláusulas que se muestran en la fila superior de la Figura 7.13. La segunda fila en la figura muestra todas las cláusulas obtenidas resolviendo parejas de la primera fila. Entonces, cuando $H_{1,2}$ se resuelve con $\neg H_{1,2}$ obtenemos la cláusula vacía, representada mediante un cuadrado pequeño. Una revisión de la Figura 7.13 nos revela que muchos pasos de resolución no nos sirven de nada. Por ejemplo, la cláusula $B_{1,1} \vee \neg B_{1,1} \vee H_{1,2}$ es equivalente a *Verdadero* $\vee H_{1,2}$, que es también equivalente a *Verdadero*. Deducir que *Verdadero* es verdadero no nos es muy útil. Por lo tanto, se puede descartar cualquier cláusula que contenga dos literales complementarios.

Completitud de la resolución

Para concluir con nuestro debate acerca de la resolución, ahora vamos a demostrar por qué es completo el procedimiento RESOLUCIÓN-LP. Para hacerlo nos vendrá bien introducir el concepto de **cierre de la resolución** $CR(S)$ del conjunto de cláusulas S, que es el conjunto de todas las cláusulas derivables, obtenidas mediante la aplicación repetida de la regla de resolución a las cláusulas de S o a las derivadas de éstas. El cierre de la resolución es lo que calcula el procedimiento RESOLUCIÓN-LP y asigna como valor final a la variable *cláusulas*. Es fácil ver que $CR(S)$ debe ser finito, porque sólo hay un conjunto finito de las diferentes cláusulas que se pueden generar a partir del conjunto de símbolos $P_1, \ldots, P_k$ que aparecen en S, (fíjese que esto no sería cierto si no aplicáramos el procedimiento de factorización, que elimina las copias múltiples de un literal). Por eso, el procedimiento RESOLUCIÓN-LP siempre termina.

CIERRE DE LA RESOLUCIÓN

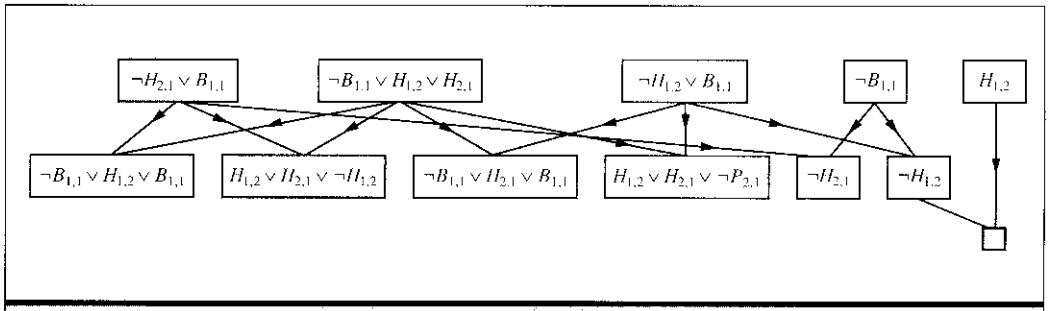


Figura 7.13 Aplicación parcial de RESOLUCIÓN-LP a una inferencia sencilla en el mundo de wumpus. Se observa que $\neg H_{1,2}$ se sigue de las cláusulas 3.ª y 4.ª de la fila superior.

TEOREMA FUNDAMENTAL DE LA RESOLUCIÓN

El teorema de la completitud para la resolución en lógica proposicional se denomina **teorema fundamental de la resolución**:

> Si un conjunto de cláusulas es *insatisfacible*, entonces el cierre de la resolución de esas cláusulas contiene la cláusula vacía.

Vamos a probar este teorema demostrando su contraposición: si el cierre $CR(S)$ *no* contiene la cláusula vacía, entonces S es *satisfacible*. De hecho, podemos construir un modelo de S con los valores de verdad adecuados para $P_1, \ldots, P_k$. El procedimiento de construcción es como sigue:

Para i de 1 a k,

- Si hay una cláusula en $CR(S)$ que contenga el literal $\neg P_i$, tal que todos los demás literales de la cláusula sean falsos bajo la asignación escogida para $P_1, \ldots, P_{i-1}$, entonces asignar a P_i el valor de *falso*.
- En otro caso, asignar a P_i el valor de *verdadero*.

Queda por demostrar que esta asignación a $P_1, \ldots, P_k$ es un modelo de S, a condición de que $CR(S)$ se cierre bajo la resolución y no contenga la cláusula vacía. Esta demostración se deja como ejercicio.

Encadenamiento hacia delante y hacia atrás

La completitud de la resolución hace que ésta sea un método de inferencia muy importante. Sin embargo, en muchos casos prácticos no se necesita todo el poder de la resolución. Las bases de conocimiento en el mundo real a menudo contienen sólo cláusulas, de un tipo restringido, denominadas **cláusulas de Horn**. Una cláusula de Horn es una disyunción de literales de los cuales, *como mucho uno es positivo*. Por ejemplo, la cláusula $(\neg L_{1,1} \vee \neg Brisa \vee B_{1,1})$, en donde $L_{1,1}$ representa que el agente está en la casilla [1, 1], es una cláusula de Horn, mientras que la cláusula $(\neg B_{1,1} \vee H_{1,2} \vee H_{2,1})$ no lo es.

CLÁUSULAS DE HORN

La restricción de que haya sólo un literal positivo puede parecer algo arbitraria y sin interés, pero realmente es muy importante, debido a tres razones:

1. Cada cláusula de Horn se puede escribir como una implicación cuya premisa sea una conjunción de literales positivos y cuya conclusión sea un único literal positivo. (*Véase* el Ejercicio 7.12.) Por ejemplo, la cláusula de Horn $(\neg L_{1,1} \vee \neg Brisa \vee B_{1,1})$ se puede rescribir como la implicación $(L_{1,1} \wedge Brisa) \Rightarrow B_{1,1}$. La sentencia es más fácil de leer en la última representación: ésta dice que si el agente está en la casilla [1, 1] y percibe una brisa, entonces la casilla [1, 1] tiene una corriente de aire. La gente encuentra más fácil esta forma de leer y escribir sentencias para muchos dominios del conocimiento.

CLÁUSULAS POSITIVAS

CABEZA

CUERPO

HECHO

Las cláusulas de Horn como ésta, con *exactamente* un literal positivo, se denominan **cláusulas positivas**. El literal positivo se denomina **cabeza**, y la disyunción de literales negativos **cuerpo** de la cláusula. Una cláusula positiva que no tiene literales negativos simplemente aserta una proposición dada, que algunas veces se le denomina **hecho**. Las cláusulas positivas forman la base de la

programación lógica, que se verá en el Capítulo 9. Una cláusula de Horn *sin* literales positivos se puede escribir como una implicación cuya conclusión es el literal *Falso*. Por ejemplo, la cláusula $(\neg W_{1,1} \vee \neg W_{1,2})$ (el *wumpus* no puede estar en la casilla [1, 1] y la [1, 2] a la vez) es equivalente a $W_{1,1} \wedge W_{1,2} \Rightarrow$ *Falso*. A este tipo de sentencias se las llama **restricciones de integridad** en el mundo de las bases de datos, donde se utilizan para indicar errores entre los datos. En los algoritmos siguientes asumimos, para simplificar, que la base de conocimiento sólo contiene cláusulas positivas y que no dispone de restricciones de integridad. Entonces decimos que estas bases de conocimiento están en forma de Horn.

RESTRICCIONES DE INTEGRIDAD

2. La inferencia con cláusulas de Horn se puede realizar mediante los algoritmos de **encadenamiento hacia delante** y de **encadenamiento hacia atrás**, que en breve explicaremos. Ambos algoritmos son muy naturales, en el sentido de que los pasos de inferencia son obvios y fáciles de seguir por las personas.

ENCADENAMIENTO HACIA DELANTE

ENCADENAMIENTO HACIA ATRÁS

3. Averiguar si hay o no implicación con las cláusulas de Horn se puede realizar en un tiempo que es *lineal* respecto al tamaño de la base de conocimiento.

Este último hecho es una grata sorpresa. Esto significa que la inferencia lógica es un proceso barato para muchas bases de conocimiento en lógica proposicional que se encuentran en el mundo real.

El algoritmo de encadenamiento hacia delante ¿IMPLICACIÓN-EHD-LP?(BC, q) determina si un símbolo proposicional q (la petición) se deduce de una base de conocimiento compuesta por cláusulas de Horn. El algoritmo comienza a partir de los hechos conocidos (literales positivos) de la base de conocimiento. Si todas las premisas de una implicación se conocen, entonces la conclusión se añade al conjunto de hechos conocidos. Por ejemplo, si $L_{1,1}$ y *Brisa* se conocen y $(L_{1,1} \wedge Brisa) \Rightarrow B_{1,1}$ está en la base de conocimiento, entonces se puede añadir $B_{1,1}$ a ésta. Este proceso continúa hasta que la petición q es añadida o hasta que no se pueden realizar más inferencias. En la Figura 7.14 se muestra el algoritmo detallado. El principal punto a recordar es que el algoritmo se ejecuta en tiempo lineal.

La mejor manera de entender el algoritmo es mediante un ejemplo y un diagrama. La Figura 7.15(a) muestra una base de conocimiento sencilla con cláusulas de Horn, en donde A y B se conocen como hechos. La Figura 7.15(b) muestra la misma base de conocimiento representada mediante un **grafo Y-O**. En los grafos Y-O múltiples enlaces se juntan mediante un arco para indicar una disyunción (cualquier enlace se puede probar). Es fácil ver cómo el encadenamiento hacia delante trabaja sobre el grafo. Se seleccionan los hechos conocidos (aquí A y B) y la inferencia se propaga hacia arriba tanto como se pueda. Siempre que aparece una conjunción, la propagación se para hasta que todos los conjuntores sean conocidos para seguir a continuación. Se anima al lector a que desarrolle el proceso en detalle a partir del ejemplo.

GRAFO Y-O

Es fácil descubrir que el encadenamiento hacia delante es un proceso **sólido**: cada inferencia es esencialmente una aplicación del Modus Ponens. El encadenamiento hacia delante también es **completo**: cada sentencia atómica implicada será derivada. La forma más fácil de verlo es considerando el estado final de la tabla *inferido* (después de que el algoritmo encuentra un **punto fijo** a partir del cual no es posible realizar nuevas inferencias).

PUNTO FIJO

```
función ¿IMPLICACIÓN-EHD-LP?(BC, q) devuelve verdadero o falso
  entradas: BC, la base de conocimiento, un conjunto de cláusulas de Horn en Lógica Proposicional
            q, la petición, un símbolo proposicional
  variables locales: cuenta, una tabla ordenada por cláusula, inicializada al número de cláusulas
                     inferido, una tabla, ordenada por símbolo, cada entrada inicializada a falso
                     agenda, una lista de símbolos, inicializada con los símbolos de la BC que se
                     sabe que son verdaderos

  mientras agenda no esté vacía hacer
    p ← POP(agenda)
    a menos que inferido[p] hacer
      inferido[p] ← verdadero
      para cada cláusula de Horn c en la que aparezca la premisa p hacer
        reducir cuenta[c]
        si cuenta[c] = 0 entoncer hacer
          si CABEZA[c] = q entonces devolver verdadero
          PUSH(CABEZA[c], agenda)
  devolver falso
```

Figura 7.14 El algoritmo de encadenamiento hacia delante para la lógica proposicional. La variable *agenda* almacena la pista de los símbolos que se saben son verdaderos pero no han sido «procesados» todavía. La tabla *cuenta* guarda la pista de las premisas de cada implicación que aún son desconocidas. Siempre que se procesa un símbolo p de la agenda la cuenta se reduce en uno para cada implicación en la que aparece la premisa p. (Este proceso se puede realizar en un tiempo constante si la *BC* se ordena de forma adecuada.) Si la cuenta llega a cero, todas las premisas de la implicación son conocidas, y por tanto, la conclusión se puede añadir a la agenda. Por último, necesitamos guardar la pista de qué símbolos han sido procesados; no se necesita añadir un símbolo inferido si ha sido procesado previamente. Este proceso nos evita un trabajo redundante, y también nos prevé de los bucles infinitos que podrían causarse por implicaciones tales como $P \Rightarrow Q$ y $Q \Rightarrow P$.

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

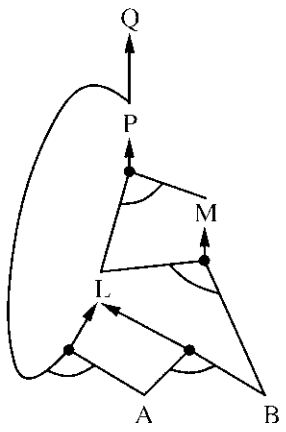
$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B

(a)



(b)

Figura 7.15 (a) Una base de conocimiento sencilla con cláusulas de Horn. (b) Su correspondiente grafo Y-O.

La tabla contiene el valor *verdadero* para cada símbolo inferido en el proceso, y el valor *falso* para los demás símbolos. Podemos interpretar la tabla como un modelo lógico, más aún, *cada cláusula positiva de la BC original es verdadera en este modelo.* Para ver esto asumamos lo opuesto, en concreto, que alguna cláusula $a_1 \wedge \ldots \wedge a_k \Rightarrow b$ sea falsa en el modelo. Entonces $a_1 \wedge \ldots \wedge a_k$ debe ser verdadero en el modelo y b debe ser falso. ¡Pero esto contradice nuestra asunción de que el algoritmo ha encontrado un punto fijo! Por lo tanto, podemos concluir que el conjunto de sentencias atómicas inferidas hasta el punto fijo define un modelo de la *BC* original. Además, cualquier sentencia atómica q que se implica de la *BC* debe ser cierta en todos los modelos y en este modelo en particular. Por lo tanto, cada sentencia implicada q debe ser inferida por el algoritmo.

DIRIGIDO POR LOS DATOS

El encadenamiento hacia delante es un ejemplo del concepto general de razonamiento **dirigido por los datos**, es decir, un razonamiento en el que el foco de atención parte de los datos conocidos. Este razonamiento se puede utilizar en un agente para derivar conclusiones a partir de percepciones recibidas, a menudo, sin la necesidad de una petición concreta. Por ejemplo, el agente de *wumpus* podría DECIR sus percepciones a la base de conocimiento utilizando un algoritmo de encadenamiento hacia delante de tipo incremental, en el que los hechos se pueden añadir a la agenda para iniciar nuevas inferencias. A las personas, a medida que les llega nueva información, se les activa una gran cantidad de razonamiento dirigido por los datos. Por ejemplo, si estoy en casa y oigo que comienza a llover, podría sucederme que la merienda quede cancelada. Con todo esto, no será muy probable que el pétalo diecisieteavo de la rosa más alta del jardín de mi vecino se haya mojado. Las personas llevan a cabo un encadenamiento hacia delante con un control cuidadoso, a fin de no hundirse en consecuencias irrelevantes.

El algoritmo de encadenamiento hacia atrás, tal como sugiere su nombre, trabaja hacia atrás a partir de la petición. Si se sabe que la petición q es verdadera, entonces no se requiere realizar ningún trabajo. En el otro caso, el algoritmo encuentra aquellas implicaciones de la base de conocimiento de las que se concluye q. Si se puede probar que todas las premisas de una de esas implicaciones son verdaderas (mediante en encadenamiento hacia atrás), entonces q es verdadera. Cuando se aplica a la petición Q de la Figura 7.15, el algoritmo retrocede hacia abajo por el grafo hasta que encuentra un conjunto de hechos conocidos que forma la base de la demostración. El algoritmo detallado se deja como ejercicio. Al igual que en el encadenamiento hacia delante, una implementación eficiente se ejecuta en tiempo lineal.

RAZONAMIENTO DIRIGIDO POR EL OBJETIVO

El encadenamiento hacia atrás es un tipo de **razonamiento dirigido por el objetivo**. Este tipo de razonamiento es útil para responder a peticiones tales como «¿Qué debo hacer ahora?» y «¿Dónde están mis llaves?» A menudo, el coste del encadenamiento hacia atrás es *mucho menor* que el orden lineal respecto al tamaño de la base de conocimiento, porque el proceso sólo trabaja con los hechos relevantes. Por lo general, un agente debería repartir su trabajo entre el razonamiento hacia delante y el razonamiento hacia atrás, limitando el razonamiento hacia delante a la generación de los hechos que sea probable que sean relevantes para las peticiones, y éstas se resolverán mediante el encadenamiento hacia atrás.