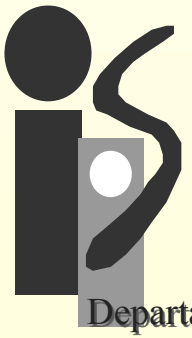
Truth Maintenance Systems

Razonamiento simbólico bajo incertidumbre
Razonamiento no monótono



Departamento de Informática e Ingeniería de Sistemas
C.P.S. Universidad de Zaragoza

Última revisión: Dic. 2009

- **Índice**
  - 1. Introducción al razonamiento no monótono
    - 1.1 La historia del asesino ABC
    - 1.2 Sistemas de razonamiento no monótono
    - 1.3 Aspectos clave de sistemas de razonamiento no monótono
  - 2. ¿Por qué utilizar un TMS?
  - 3. Cuestiones sobre implementación
    - 3.1 Razonamiento progresivo y regresivo
  - 4. Implementación: Búsqueda primero en profundidad
    - 4.1 Sistema mantenimiento de la verdad basado en justificaciones (JTMS)
      - `Sistema de dependencias de CLIPS`
    - 4.2 Sistema mantenimiento de la verdad basado en la lógica (LTMS)
  - 5. Implementación : Búsqueda en anchura (ATMS)

- **Bibliografía**

  - Giarratano and Riley. "Expert Systems. Principles and Programming". PWS Publishing Company, second Edition 1994.
  - CLIPS Reference Manual. Volume I - The Basic Programming Guide". Report JSC-25012 Software Thechnology Branch, Lyndon B. Johnson Space Center.
  - Elaine Rich, Kevin Knight. "Inteligencia Artificial" (cap. 7). Mc Graw Hill. Segunda Edición. 1994.
  - P.H. Wiston. "Inteligencia Artificial" (cap. 14). Addisson Wesley. Tercera Edición. 1992.

➢ Kenneth D. Forbus and Johan de Kleer "Building Problem Solvers" The MIT Press. 1993.

1.1 Razonamiento no monótono

- **Desafortunadamente, en muchos dominios de problemas no es posible crear modelos completos, consistentes e inalterables**
  - En este tema y en el de razonamiento bajo incertidumbre se describen técnicas de resolución de problemas con modelos incompletos e inciertos.
  - La historia del asesino ABC ilustra con claridad muchos aspectos fundamentales que estas técnicas deben proporciona [*The Web of Belief* de Quine y Ullian 1978]:

> *Sean Abbott, Babbitt y Cabot los sospechosos en un caso de asesinato. Abbott tiene una coartada en el registro de un respetable hotel de Albany. Babbitt también tiene una coartada, la de su cuñado, al cual estaba viendo en Brooklyn en el momento del crimen. Cabbott defiende también su coartada, asegurando que se encontraba viendo un campeonato de esquí en Catskills, pero sólo puede aportar su propio testimonio. Por lo tanto, creemos:*
>
> *(1) Que Abbot no cometó el crimen*
> *(2) Que Babbit no lo hizo*
> *(3) Que o Abbott o Babbitt o Cabot lo hizo*

1.1 Razonamiento no monótono

- **Desafortunadamente, en muchos dominios de problemas no es posible crear modelos completos, consistentes e inalterables**
  - En este tema y en el de razonamiento bajo incertidumbre se describen técnicas de resolución de problemas con modelos incompletos e inciertos.
  - La historia del asesino ABC ilustra con claridad muchos aspectos fundamentales que estas técnicas deben proporciona [*The Web of Belief* de Quine y Ullian 1978]:

> *Sean Abbott, Babbitt y Cabot los sospechosos en un caso de asesinato. Abbott tiene una coartada en el registro de un respetable hotel de Albany. Babbitt también tiene una coartada, la de su cuñado, al cual estaba viendo en Brooklyn en el momento del crimen. Cabbott defiende también su coartada, asegurando que se encontraba viendo un campeonato de esquí en Catskills, pero sólo puede aportar su propio testimonio. Por lo tanto, creemos:*
>
> *(1) Que Abbot no cometó el crimen*
> *(2) Que Babbit no lo hizo*
> *(3) Que o Abbott o Babbitt o Cabot lo hizo*

1.1 Razonamiento no monótono

- **Desafortunadamente, en muchos dominios de problemas no es posible crear modelos completos, consistentes e inalterables**
  - En este tema y en el de razonamiento bajo incertidumbre se describen técnicas de resolución de problemas con modelos incompletos e inciertos.
  - La historia del asesino ABC ilustra con claridad muchos aspectos fundamentales que estas técnicas deben proporciona [*The Web of Belief* de Quine y Ullian 1978]:

> *Sean Abbott, Babbitt y Cabot los sospechosos en un caso de asesinato. Abbott tiene una coartada en el registro de un respetable hotel de Albany. Babbitt también tiene una coartada, la de su cuñado, al cual estaba viendo en Brooklyn en el momento del crimen. Cabbott defiende también su coartada, asegurando que se encontraba viendo un campeonato de esquí en Catskills, pero sólo puede aportar su propio testimonio. Por lo tanto, creemos:*
>
> *(1) Que Abbot no cometó el crimen*
> *(2) Que Babbit no lo hizo*
> *(3) Que o Abbott o Babbitt o Cabot lo hizo*

1.1 Razonamiento no monótono

- **Desafortunadamente, en muchos dominios de problemas no es posible crear modelos completos, consistentes e inalterables**
  - En este tema y en el de razonamiento bajo incertidumbre se describen técnicas de resolución de problemas con modelos incompletos e inciertos.
  - La historia del asesino ABC ilustra con claridad muchos aspectos fundamentales que estas técnicas deben proporciona [*The Web of Belief* de Quine y Ullian 1978]:

> *Sean Abbott, Babbitt y Cabot los sospechosos en un caso de asesinato. Abbott tiene una coartada en el registro de un respetable hotel de Albany. Babbitt también tiene una coartada, la de su cuñado, al cual estaba viendo en Brooklyn en el momento del crimen. Cabbott defiende también su coartada, asegurando que se encontraba viendo un campeonato de esquí en Catskills, pero sólo puede aportar su propio testimonio. Por lo tanto, creemos:*
>
> *(1) Que Abbot no cometó el crimen*
> *(2) Que Babbit no lo hizo*
> *(3) Que o Abbott o Babbitt o Cabot lo hizo*

1.1 Razonamiento no monótono

- **Desafortunadamente, en muchos dominios de problemas no es posible crear modelos completos, consistentes e inalterables**
  - En este tema y en el de razonamiento bajo incertidumbre se describen técnicas de resolución de problemas con modelos incompletos e inciertos.
  - La historia del asesino ABC ilustra con claridad muchos aspectos fundamentales que estas técnicas deben proporciona [*The Web of Belief* de Quine y Ullian 1978]:

> *Sean Abbott, Babbitt y Cabot los sospechosos en un caso de asesinato. Abbott tiene una coartada en el registro de un respetable hotel de Albany. Babbitt también tiene una coartada, la de su cuñado, al cual estaba viendo en Brooklyn en el momento del crimen. Cabbott defiende también su coartada, asegurando que se encontraba viendo un campeonato de esquí en Catskills, pero sólo puede aportar su propio testimonio. Por lo tanto, creemos:*
>
> *(1) Que Abbot no cometó el crimen*
> *(2) Que Babbit no lo hizo*
> *(3) Que o Abbott o Babbitt o Cabot lo hizo*

El ejemplo ABC

- **En principio el sospechoso principal es Cabot, pero ...**

> *... Cabot puede posteriormente presentar pruebas de su coartada. Tuvo muy buena suerte y fue captado por las cámaras de TV en las gradas de la pista. De esta forma aparece una nueva creencia que es:*
>
> *(4) Cabot no lo hizo.*

- **Las creencias (1) a (4) son inconsistentes, por lo que una debe desestimarse. ¿Cuál es la evidencia más débil?**
  - La base de (1) en un hotel es buena, ya que se trata de un hotel prestigioso.
  - La base (2) es más débil, ya que puede ser que el cuñado de Babbitt esté mintiendo.
  - La base de (3) es doble: No existen signos de robo y sólo Abbott, Babbitt y Cabot parecen salir beneficiados con el asesinato.
  - La base de (4) es concluyente

El ejemplo ABC

- **En principio el sospechoso principal es Cabot, pero ...**

  ... Cabot puede posteriormente presentar pruebas de su coartada. Tuvo muy buena suerte y fue captado por las cámaras de TV en las gradas de la pista. De esta forma aparece una nueva creencia que es:

  (4) Cabot no lo hizo.

- **Las creencias (1) a (4) son inconsistentes, por lo que una debe desestimarse. ¿Cuál es la evidencia más débil?**
  - La base de (1) en un hotel es buena, ya que se trata de un hotel prestigioso.
  - La base (2) es más débil, ya que puede ser que el cuñado de Babbitt esté mintiendo.
  - La base de (3) es doble: No existen signos de robo y sólo Abbott, Babbitt y Cabot parecen salir beneficiados con el asesinato.
  - La base de (4) es concluyente

El ejemplo ABC

- **En principio el sospechoso principal es Cabot, pero ...**

> *... Cabot puede posteriormente presentar pruebas de su coartada. Tuvo muy buena suerte y fue captado por las cámaras de TV en las gradas de la pista. De esta forma aparece una nueva creencia que es:*
>
> *(4) Cabot no lo hizo.*

- **Las creencias (1) a (4) son inconsistentes, por lo que una debe desestimarse. ¿Cuál es la evidencia más débil?**
  - La base de (1) en un hotel es buena, ya que se trata de un hotel prestigioso.
  - La base (2) es más débil, ya que puede ser que el cuñado de Babbitt esté mintiendo.
  - La base de (3) es doble: No existen signos de robo y sólo Abbott, Babbitt y Cabot parecen salir beneficiados con el asesinato.
  - La base de (4) es concluyente

El ejemplo ABC

- **En principio el sospechoso principal es Cabot, pero ...**

> *... Cabot puede posteriormente presentar pruebas de su coartada. Tuvo muy buena suerte y fue captado por las cámaras de TV en las gradas de la pista. De esta forma aparece una nueva creencia que es:*
>
> *(4) Cabot no lo hizo.*

- **Las creencias (1) a (4) son inconsistentes, por lo que una debe desestimarse. ¿Cuál es la evidencia más débil?**
  - **La base** de (1) en un hotel es buena, ya que se trata de un hotel prestigioso.
  - **La base** (2) es más débil, ya que puede ser que el cuñado de Babbitt esté mintiendo.
  - **La base** de (3) es doble: No existen signos de robo y sólo Abbott, Babbitt y Cabot parecen salir beneficiados con el asesinato.
  - **La base** de (4) es concluyente

El ejemplo ABC

- **Las creencias (2) y (3) son los puntos débiles. Para resolver la inconsistencia se debe desestimar (2) o (3), es decir se incrimina a Babbitt o se amplia la lista con un nuevo sospechoso**
  - Obsérvese el progreso descendente de la revisión. Si se desestima (2), se debe revisar la creencia subyacente, y sin embargo provisional, de que el cuñado estaba diciendo la verdad y Babbitt estaba en Brooklyn. Si, en lugar de esto, se desestima (3), se tiene que revisar también la creencia subyacente de que nadie menos Abbott, Babbitt y Cabot sale beneficiado.
- **Se trata de evitar cierta arbitrariedad en este análisis.**
  - Se escogieron las creencias inconsistentes (1) a (4), y de acuerdo con ellas surgieron otras creencias en forma de creencias subyacentes: una creencia acerca del registro en el hotel, una creencia acerca del prestigio del hotel, una creencia sobre la televisión, y así sucesivamente. Se podrían listar docenas de creencias completas equivalentes a las anteriores, y a continuación se miraría si aparecen contradicciones, procediendo a restablecer la consistencia profundizando en ellas de diferentes maneras.

El ejemplo ABC

- **La organización de nuestro análisis hizo la tarea menos pesada:**
  - Nuestra atención se centro en cuatro creencias destacadas entre las cuales había que desechar una.
  - A continuación se colocaban las otras creencias por debajo de estas, como una ayuda para poder elegir cuál de las cuatro se desestimaba.
  - Cuando un conjunto de hechos llegue a un punto en el que aparezca una contradicción, se seleccionará el conjunto más pequeño de hechos que todavía mantenga la contradicción: por ejemplo (1) a (4). Al revisar y comparar la evidencia de las creencias del subconjunto, estas nos conducen de manera bastante sistemática a otras creencias del conjunto.
  - Al probar la evidencia, ¿dónde paramos al rastrear evidencias subyacentes?. En la práctica, cuando se ha restaurado la consistencia satisfactoriamente al sondearse las creencias que deben eliminarse.

1.2 Sistemas de razonamiento no monótono

- **Se han propuesto varios marcos lógicos y métodos computacionales para tratar con problemas donde surge conocimiento incierto, difuso y cambiante:**
  - El **razonamiento no monótono**, en el que se extienden los axiomas y/o las reglas para que sea posible razonar con información incompleta.
    - En estos sistemas una sentencia puede pensarse que es **CIERTA**, **FALSA** o **NINGUNA** de las dos.
  - El **razonamiento estadístico**, en el que se extiende la representación para permitir que algún tipo de **medida numérica sobre la certeza** (en lugar de simplemente CIERTO o FALSO) se pueda asociar a cada sentencia.

Lógica de predicados

- **Los sistemas convencionales de razonamiento, como la lógica de predicados, están diseñados para trabajar con información que cumple tres propiedades importantes:**
  - **La información es completa con respecto al dominio de interés**. Todos los hechos necesarios para resolver el problema o están presentes o pueden derivarse.
  - **La información es consistente**.
  - La única forma en que puede cambiar la información es que **se añadan nuevos hechos conforme estén disponibles**. Si estos hechos son consistentes con los demás hechos, ninguno de los hechos pertenecientes al conjunto de los que eran ciertos pueden refutarse. Esta propiedad se denomina **monotonía**.

1.3 Aspectos clave de sistemas de razonamiento no monótono

- **Para poder resolver problemas en los que no aparezcan alguna de las propiedades de un sistema de razonamiento monótono se deben tratar algunas aspectos:**
  - 1. ¿De qué forma puede extenderse la base de conocimiento para permitir inferencias realizadas tanto **sobre la base de una falta de conocimiento** como sobre una presencia del mismo?
    - Se deberían poder decir cosas como "Si no se tienen razones para sospechar que una persona cometió un crimen entonces asuma que no lo cometió" o "Si tienen razones para creer que alguien se ha puesto de acuerdo con sus parientes, entonces asuma que los parientes intentarán protegerle.
    - Es necesario tener clara la distinción
      - `Se sabe NOT P`
      - `No se sabe si P`
    - A una inferencia que dependa de la falta de una parte de conocimiento se la denomina **inferencia no monótona**, porque la adición de una nueva aserción puede invalidar una dependencia que dependía de la ausencia de esta aserción.

Aspectos clave de sistemas de razonamiento no monótono

- 2. ¿De qué forma puede actualizarse correctamente la base de conocimiento cuando se añade un nuevo hecho (o cuando se elimina otro anterior)?
  - ¿Como puede encontrarse las demostraciones que quedan invalidadas por la adición de un hecho? La solución más usual consiste en tomar nota de las demostraciones, que llamaremos **justificaciones**.
  - Esto permite encontrar todas las justificaciones que dependen de la ausencia de un nuevo hecho y, por lo tanto, estas demostraciones pueden marcarse como inválidas.
  - Este mecanismo de almacenamiento también hace posible soportar un razonamiento convencional monótono en aquellos casos en que los axiomas puedan ocasionalmente eliminarse para reflejar cambios en el mundo que se desea modelar. Por ejemplo:
    - `Abbot está en la ciudad esta semana y puede testificar.`
    - `Si se espera a la semana siguiente puede que no esté en la ciudad.`

Aspectos clave de Sistemas de razonamiento no monótono

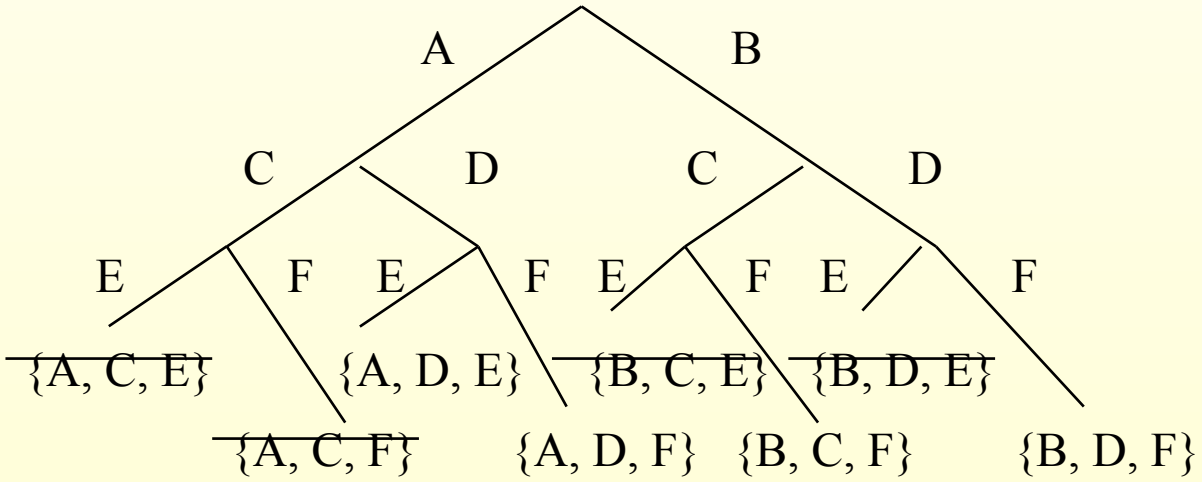
- 3. ¿Cómo puede usarse el conocimiento para ayudar a resolver **conflictos** que surgen cuando se pueden usar varias inferencias no monótonas inconsistentes?
  - Cuando las inferencias se basan tanto en la falta como en la presencia de conocimiento, las contradicciones surgen con mucha más frecuencia que en los sistemas lógicos convencionales, en los que las únicas contradicciones posibles dependen de hechos que se afirmaron explícitamente.
  - En los sistemas no monótonos con frecuencia existen partes de la base de conocimiento que son localmente consistentes pero mutuamente (globalmente) inconsistentes. Muchas de las técnicas para razonar de forma no monótona pueden definir las alternativas que pueden creerse, pero la mayoría de ellas no proporcionan la forma de elegir entre las opciones. Por lo tanto **se precisa de métodos adicionales (conocimiento del dominio/ experto) para resolver conflictos**.
    - Por ejemplo, tan pronto como se concluye que Abbott, Babbitt y Cabot afirman que no cometieron el crimen, y una vez concluido que alguno de ellos fue, se entra en una contradicción. En este caso se resuelve el conflicto encontrando la persona con la coartada más débil.

2. ¿Por qué utilizar un TMS?

- **1. Un resolutor de problemas debe identificar la responsabilidad de sus conclusiones para poder explicar como se deducen estas de las premisas.**
  - Ej. Eres un ingeniero que propone un nuevo diseño para un avión. Si tu jefe te dice que no funcionará no sabrás como continuar. Si por el contrario te dice que no habrá ningún material que soporte las tensiones que conllevan tu diseño tienes una idea de como de volver a rediseñar la solución.
- **2. Recuperarse de las inconsistencias**
  - En un mundo ideal todos las restricciones impuestas se podrían satisfacer. Si de nuevo tienes una forma de explicar porque es imposible una solución, esto te da ayudará a resolver el problema.

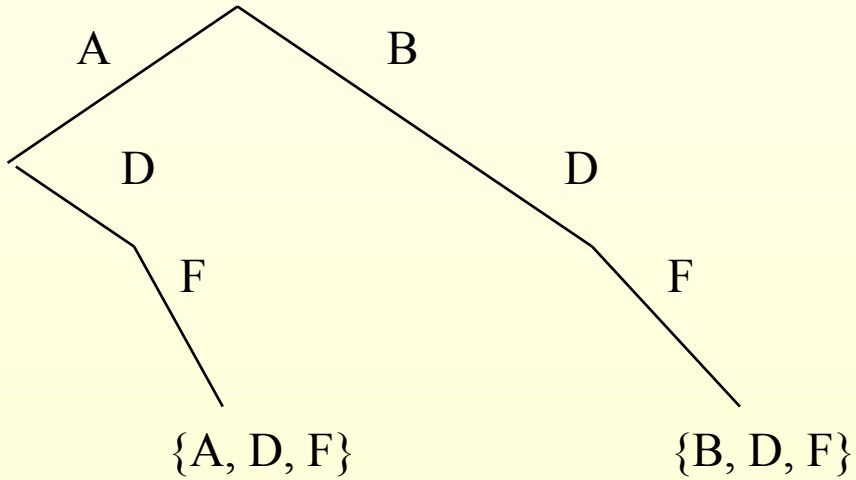
Memoria de inferencias

- **3. Mantener una memoria de inferencias**
  - La mayoría de los *resolutores* de problemas en IA buscan/exploran.
  - Con frecuencia se visitan los mismos espacios de búsqueda repetidamente.
    - P.E. Tenemos tres conjuntos de elecciones **A o B** (representando la estrategia en el diseño de un objeto), **C o D** (los materiales para construir el objeto) y **E o F** (heurísticas para elegir el tamaño de las piezas en el diseño). Suponer **A y C** son contradictorias, así como **B y E**



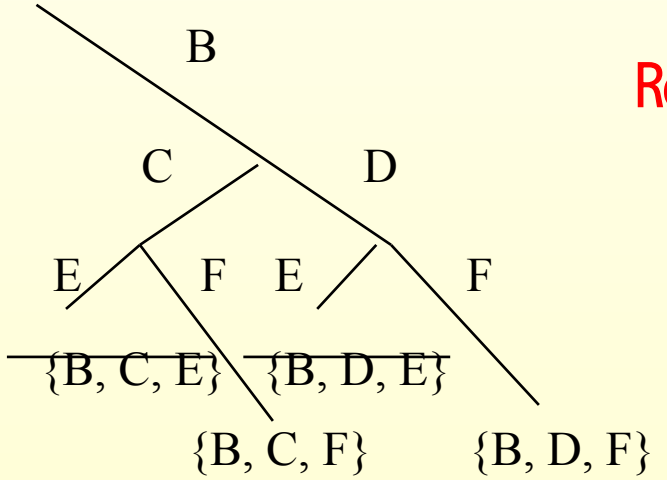
Evitar repetir cómputos

- Supongamos que D y F tienen un alto coste de cómputo (p.e. ejecutar un programa de análisis de elementos finitos), y que la búsqueda es en profundidad de izq. a der.
  - EL PROBLEMA es que cuando EL RETROCESO CRONOLOGICO (CHRONOLOGICAL BACKTRACKING) deja un contexto, toda la información del CONTEXTO SE PIERDE
  - NECESITAMOS:
    - Una **forma de memorizar el resultado**
    - Una forma de reconocer que el resultado del costoso cómputo no depende de A o B



Reconocer contradicciones

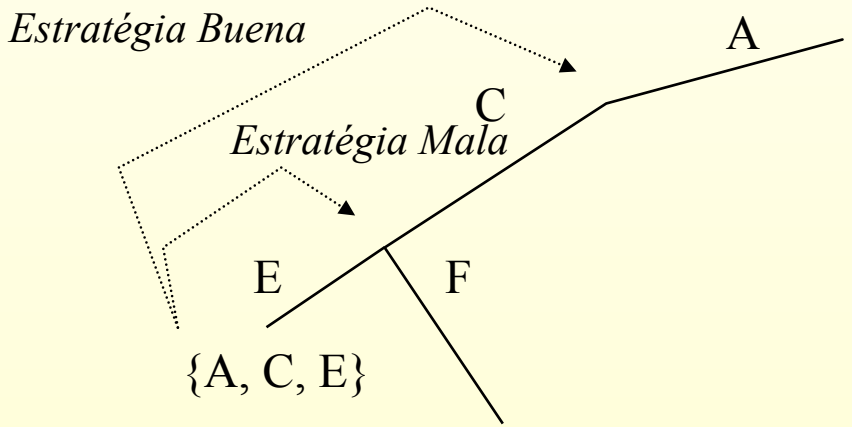
- El retroceso cronológico también gasta mucho tiempo redescubriendo contradicciones.
  - La búsqueda cronológica no reconocerá que B y E son contradictorios e intentará {B, D, E} después de haber intentado {B, C, E}
  - NECESITAMOS:
    - **Reconocer de qué elecciones depende una contradicción** y memorizarla para no volver a descender en el árbol sin chequear si el nuevo estado violará alguna inconsistencia previamente encontrada.



Registrar NoGoods

Guiar el retroceso

- **4. Guiar el retroceso**
  - Supongamos que la búsqueda detectó una inconsistencia mientras exploraba la solución {A, C, E}. La inconsistencia es causada por el hecho de elegir A y C que son incompatibles.
    - **NECESITAMOS reconocer las elecciones de las que depende una inconsistencia para retroceder a la elección más reciente que contribuye a la contradicción**.
      - `Esta estrategia se denomina retroceso dirigido por las dependencias (DEPENDENCY-DIRECTED BACKTRACKING).`



Suposiciones por defecto

- **5. Razonamiento por defecto**
  - Muchas aplicaciones requieren al *resolutor* de problemas hacer conclusiones basadas en información insuficiente
    - Con frecuencia suposiciones sobre un mundo cerrado (CLOSED-WORLD ASSUMPTIONS[1]) ayudan a restringir nuestras elecciones
      - `Si el coche no arranca suponemos que los únicos fallos posibles son el bujías o la batería. Si es otra cosa no sabríamos arreglarlo nosotros.`

1 The closed world assumption is the presumption that what is not currently known to be true is false

3.Cuestiones de Implementación

- **En todos los sistemas de razonamiento no monótono el proceso de razonamiento se separa en dos partes:**
  - El **resolutor de problema** que utiliza cualquier mecanismo que llegue a conclusiones conforme sean necesarias.
  - Un **sistema de mantenimiento de la verdad**, cuyo trabajo consiste en ir memorizando lo necesario para actualizar gradualmente el conocimiento conforme progresa la resolución del problema.
    - Las técnicas de localización de inferencias no monótonas para tratar los cambios de la base de conocimiento son:
      - **Primero en profundidad:** Se sigue un único camino, el más prometedor, hasta que surge alguna parte de conocimiento que fuerza a abandonar este camino por otro.
      - **Primero en anchura,** en dónde se consideran todas las posibilidades igual de prometedoras.

3.1 Razonamiento progresivo y regresivo

- **La resolución de un problema con conocimiento incierto puede hacerse mediante razonamiento hacia delante o mediante razonamiento hacia atrás.**
  - **Razonamiento hacia delante**: Las conclusiones que se derivan de forma no monótona se manipulan de la misma forma que las que se derivan de forma monótona.
    - Los sistemas de razonamiento no monótono que soportan este tipo de razonamiento permiten que las reglas estándar de encadenamiento hacia delante se extiendan con cláusulas ***a-no-ser-que,*** qué proporcionan la base del razonamiento por defecto.
    - El control (incluyendo la elección de la interpretación por defecto) se trata de la misma forma que todas las demás decisiones de control que realiza el sistema.

Razonamiento hacia atrás

- **Razonamiento hacia atrás** para determinar si alguna expresión P es cierta (o quizás para encontrar un conjunto de ligaduras de variables que hacen que sea cierto). Los sistemas de razonamiento no monótono que soportan este tipo de razonamiento pueden proporcionar alguna de estas características:
  - Que permita cláusulas por defecto (**a-no-ser-que**) en las reglas hacia atrás. Los conflictos entre ellas se resuelven con la misma estrategia de control utilizada en otros tipos de razonamiento.
  - Que soporte algún tipo de **debate** en el que se intente producir argumentos tanto en favor de P como en su contra. Para poder determinar que posibilidad es la más fuerte, se necesita algún tipo de conocimiento adicional aplicado a los argumentos.

Ejemplo razonamiento atrás

- **Dada la siguiente base de conocimiento, que emplea una estructura de control tipo PROLOG, si se hace la pregunta ¿Sospechosos(X)? ...**

```
Sospechoso(x) ← Se_beneficia(x)
                  A-NO-SER-QUE Coartada(x)

Coartada(x) ← En_algún_otro_sitio(x)

En_algún_otro_sitio(x) ← Registrado_en_hotel(x,y) y Lejos(y)
                         A-NO-SER-QUE Registro_falsificado(y)

Coartada(x) ← Defiende (x,y)
                A-NO-SER-QUE Miente(x)

En_algún_otro_sitio(x) ← Fotografía(x,y) y Lejos(y)

Contradicción ← TRUE
                  A-NO-SER-QUE ∃x: Sospechoso(x)
Se_beneficia(Abbott)
Se_beneficia(Babbitt)
Se_beneficia(Cabot)
```

Ejemplo razonamiento atrás

- El programa hará primero un intento con Abbott, que es sospechoso, basándose en lo que se sabe hasta ahora, por lo que el programa devolverá **Abbott** como respuesta.
- Si se añaden los hechos:

```
Registrado_en_hotel(Abbott, Albany)
Lejos(Albany)
```

  entonces el programa concluye que Abbott no es un sospechosos y tomaría en su lugar a Babbitt.
- **Considérese ahora la idea de un debate como alternativa a este enfoque**. En los sistemas de debate, se realiza un intento de encontrar múltiples respuestas. En el caso de las historia del asesinato ABC se consideraría a los tres sospechosos y después se intenta hacer algún tipo de elección entre los argumentos:
  - Por ejemplo, en este caso podríamos decidir entre: **1)** si es más probable mentir para defenderse a uno mismo que para defender a los demás y **2)** o si preferimos creer en el registro de un hotel que en la gente.

Ejemplo razonamiento adelante

- **Las reglas hacia atrás funcionan de la manera descrita, siempre que estén presentes los hechos que se necesitan cuando se invocan las reglas. Pero si se comienza con la situación inicial planteada, y se concluye que Abbott es el sospechoso y, más tarde, se nos dice que estuvo registrado en un hotel de Albany, las reglas hacia atrás no podrán detectar que algo ha cambiado.**
  - Para lograr que el comportamiento del sistema se vea influenciado por los datos es necesario utilizar reglas hacia adelante.
  - Se puede pensar en un sistema que aproveche las ventajas de razonar adelante y atrás:
    - Se podría utilizar razonamiento hacia atrás cuyo objetivo fuese encontrar un sospechoso y
    - Se podría utilizar razonamiento hacia adelante de forma que se activarían las reglas como consecuencia de la aparición de nuevos hechos que se consideraran relevantes para la tarea de encontrar un sospechoso.

Ejemplo razonamiento adelante

Si: Se_beneficia(x)
A-NO-SER-QUE Coartada(x)
entonces Sospechoso(x)
Si: En_algún_otro_sitio(x)
entonces Coartada(x)
Si: Registrado_en_hotel(x,y) y Lejos(y)
A-NO-SER-QUE Registro_falsificado(y)
entonces En_algún_otro_sitio(x)
Si: Defiende (x,y)
A-NO-SER-QUE Miente(x)
entonces Coartada(x)
Si: Fotografía(x,y) y Lejos(y)
entonces En_algún_otro_sitio(x)
Si: Cierto
A-NO-SER-QUE ∃x: Sospechoso(x)
entonces Contradicción()

Se_beneficia(Abbott)
Se_beneficia(Babbitt)
Se_beneficia(Cabot)

4. Implementación: primero en profundidad

- **El motor de inferencia puede examinar las consecuencias de un sólo conjunto de suposiciones cada vez.**
- **Si se quiere usar una vuelta atrás dirigida por dependencias, es necesario realizar las siguientes acciones:**
  - Asociar a cada nodo una o más justificaciones.
    - Cada **justificación se corresponde con un proceso de derivación que lleva al nodo**.
    - Cada justificación debe contener una **lista con todos los nodos de los que depende** la derivación.
  - Proporcionar un mecanismo que cuando se produzca una contradicción entre el nodo y su justificación, genere el **conjunto de suposiciones malas (NoGoods)** que están detrás de la justificación. Este conjunto de suposiciones malas será el mínimo tal que si se elimina algún elemento, la justificación se invalida y el nodo deja de ser creíble.
  - Proporcionar el **mecanismo que propague el resultado de la retirada de una suposición**.

4.1 TMS basado en justificaciones

- **En un sistema de mantenimiento de la verdad (TMS) basado en justificaciones (JTMS), el TMS no conoce nada sobre la estructura de las aserciones en sí mismas.**
  - El único papel del JTMS es servir como libro de anotaciones para un sistema de resolución de problemas.
  - El resolutor de problemas proporciona al JTMS las aserciones y las dependencias entre las aserciones.
- **Ejemplo del asesino ABC**
  - Inicialmente se creía que Abbott era el principal asesino porque se beneficiaba y no tenía coartada. Nuestra razón para esta creencia es no monótona porque puede cambiar y se basa en la regla:

```
Si: Se_beneficia(x)
    A-NO-SER-QUE Coartada(x)
entonces: Sospechoso(x)
```

  - ¿Cómo se representa una creencia, y como se fuerza el cambio de la creencia?

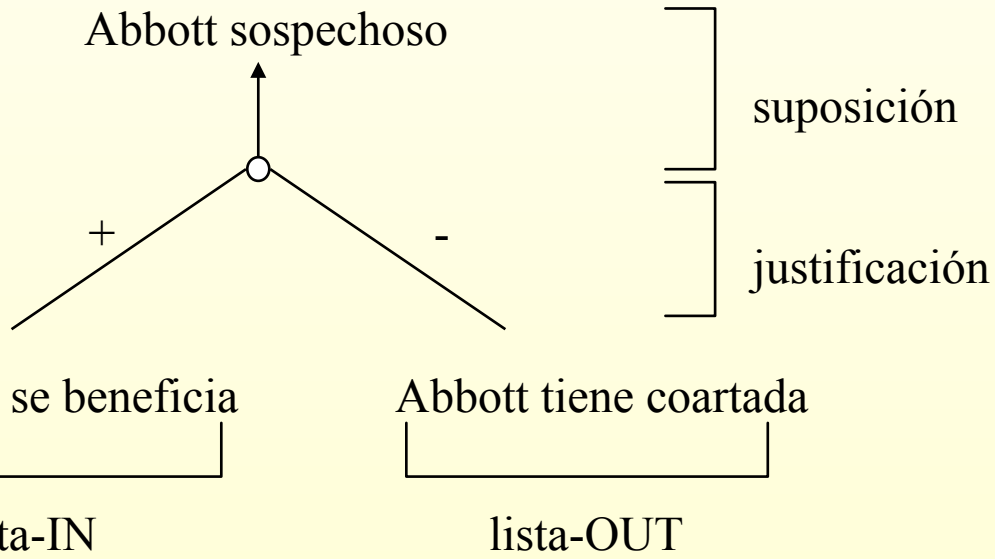
Representación y cambio de una creencia

- **Con reglas se puede hacer *ad hoc*, pero se requiere un desarrollo en el que seconstruyeran cuidadosamente las reglas para cada posible cambio .**
  - Regla que indica que si Abbott tiene coartada se borra Sospechoso(Abbott)
  - Regla que indica que si Abbott pierde coartada se incluye Sospechoso (Abbott)
  - Esta técnica se vuelve rápidamente intratable
- **Una red de dependencias TMS proporciona una forma independiente del dominio para representar y cambiar creencias de forma consistente.**

Si: Se_beneficia(x)

A-NO-SER-QUE Coartada(x)

entonces: Sospechoso(x)

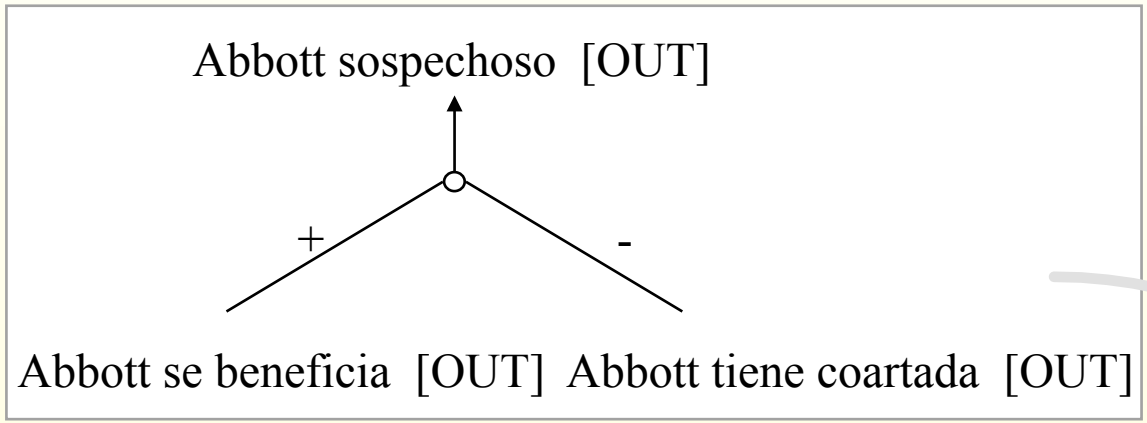


Red de dependencias

- **Las aserciones de una red de dependencias (nodos) TMS se creen si tienen una justificación válida:**
  - Se cree en cada aserción de la lista-IN
  - No se cree en ninguna de la lista-OUT
- **Una justificación es no monótona si su lista-OUT no está vacía o, recursivamente, si las aserciones de su lista-IN tienen justificaciones no monótonas. En caso contrario es monótona.**
- **Si se cree en la aserción representada por el nodo, se etiqueta IN**
- **Si no existe una buena razón para creer en la aserción se etiqueta OUT.**
- **La tarea de etiquetado de un JTMS consiste en etiquetar cada nodo de acuerdo a dos criterios:**
  - Consistencia
  - La buena fundamentación

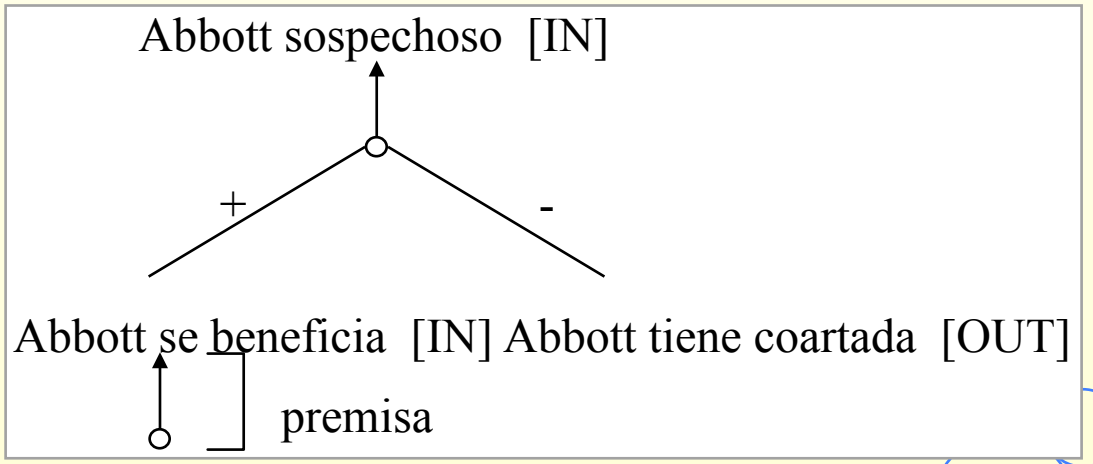
Etiquetado consistente

- **Una justificación es válida si cada nodo de su lista-IN tiene etiqueta IN y cada nodo de su lista-OUT tiene etiqueta OUT**



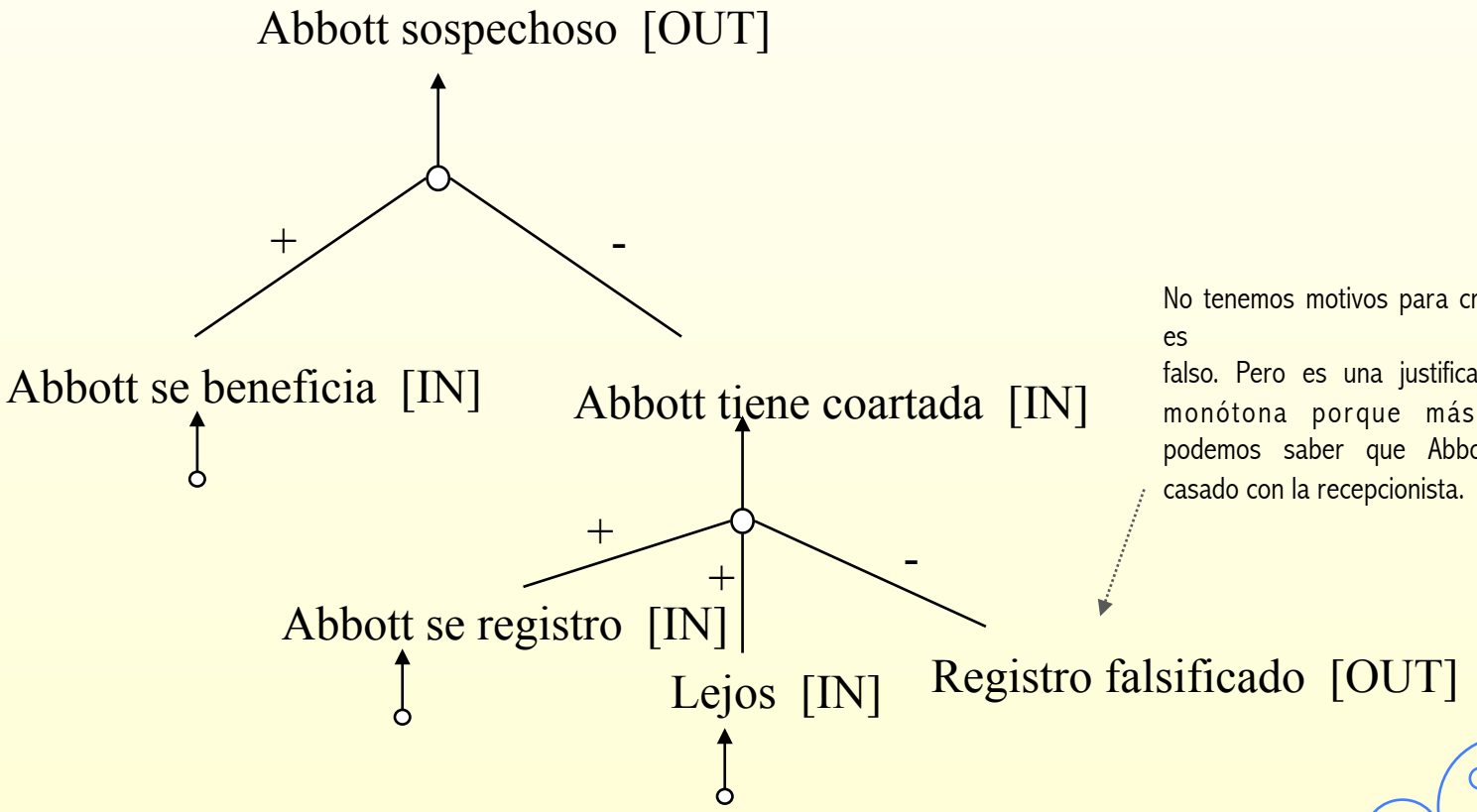
Falta la premisa
Abbott es beneficiario

o Una justificación de premisa siempre es válida



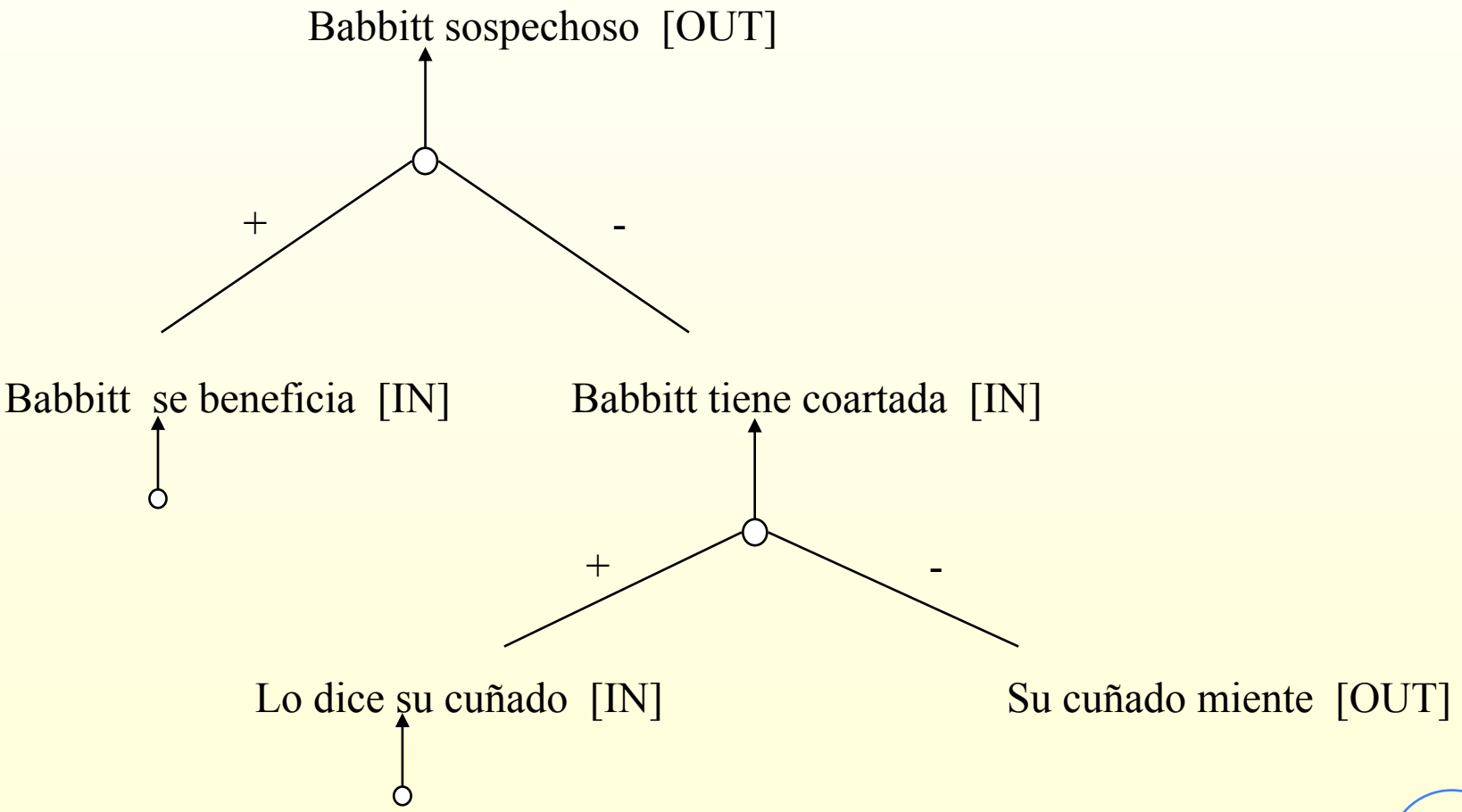
Cambio de etiquetado

- El estado inicial sobre el caso del asesinato es que Abbott es el principal sospechoso. Posteriormente, el detective establece que Abbott se encontraba en la lista del registro de un buen Hotel. Asumiendo reglas hacia adelante, el cambio de etiquetado es:



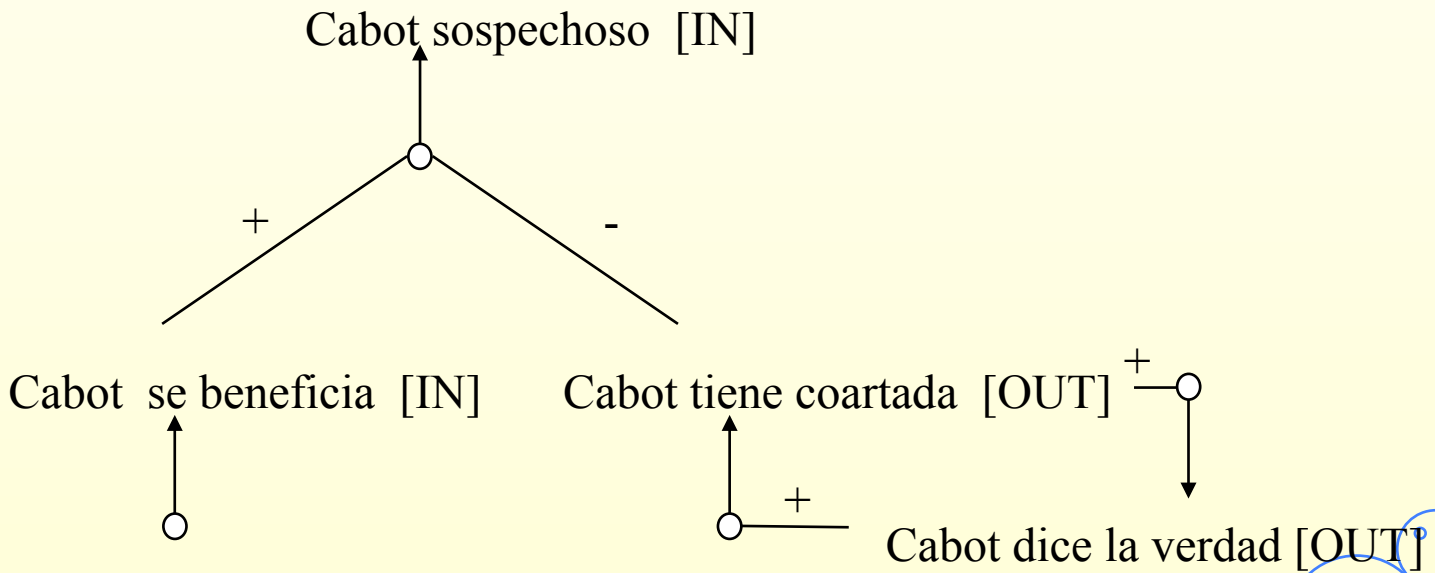
Etiquetado consistente

- Etiquetado para Babbitt:



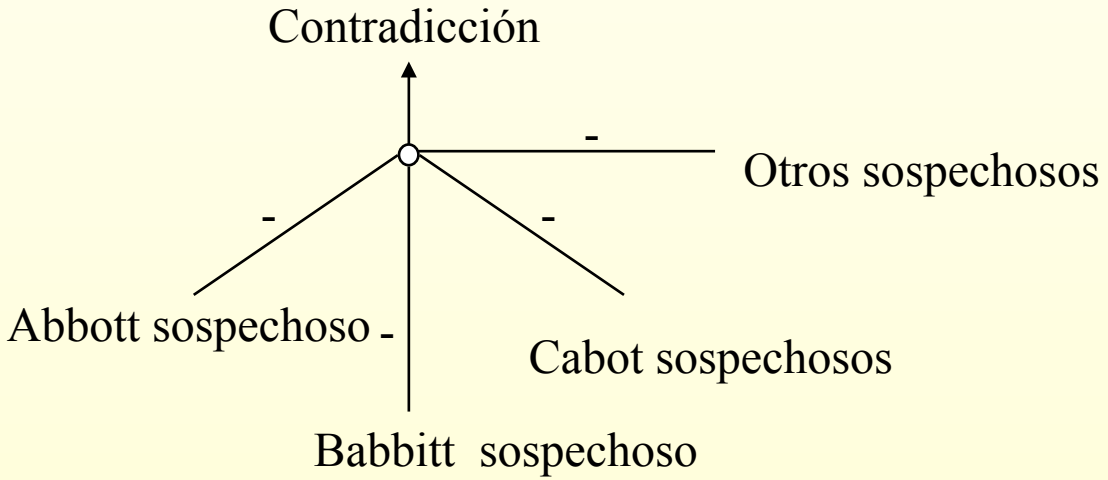
Buena fundamentación

- **El etiquetado de una red de dependencias debe tener una buena cimentación**
  - Una cadena de justificaciones no debe depender en sí mismos de los nodos que soportan.
  - Ejemplo: Inicialmente, el único soporte de que Cabot estuvo en una competición de esquí es que Cabot este diciendo la verdad sobre que estuvo allí.
  - Un TMS debe desarmar estas argumentaciones, etiquetando OUT cadenas no rotas de enlaces positivos.



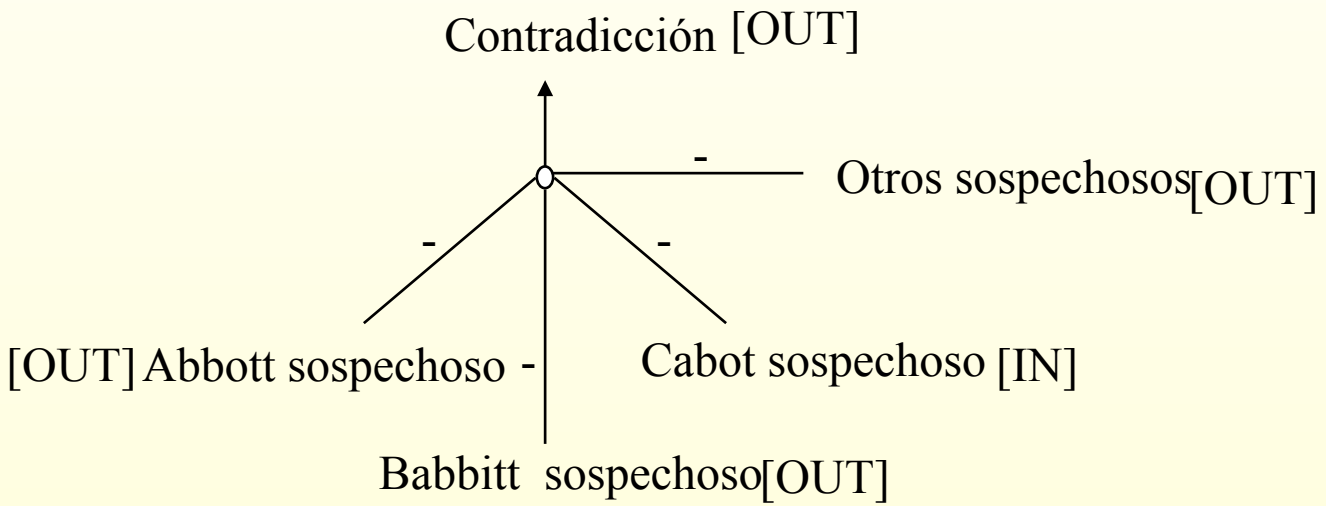
Contradicciones

- **Además de un etiquetado consistente y bien fundamentado un JTMS debe resolver las contradicciones.**
  - En un JTMS un nodo contradictorio no representa una contradicción lógica, sino un estado de la base de conocimiento explícitamente catalogado como no deseado.
  - En nuestro ejemplo, hay contradicción si no tenemos al menos un sospechoso.



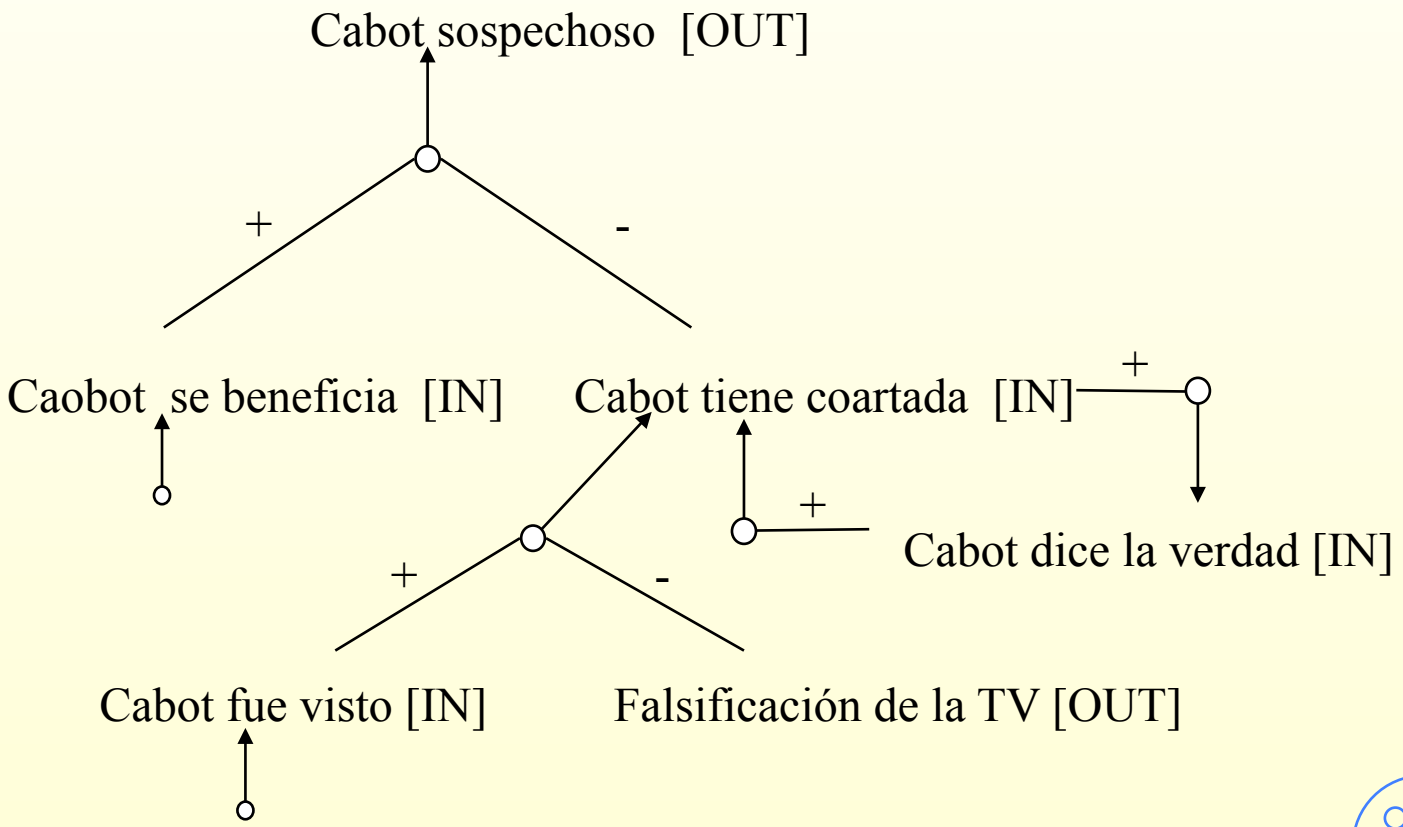
Contradicciones en ABC

- Inicialmente Cabot es el sospechoso, porque estuvo en una competición de esquí sin testigos. Por lo tanto no hay contradicción.



Contradicciones ABC

- Posteriormente se sabe que Cabot fue grabado por la televisión en el campeonato. Por lo tanto ahora **contradicción** pasa a etiquetarse con **IN**.



Contradicción en un JTMS

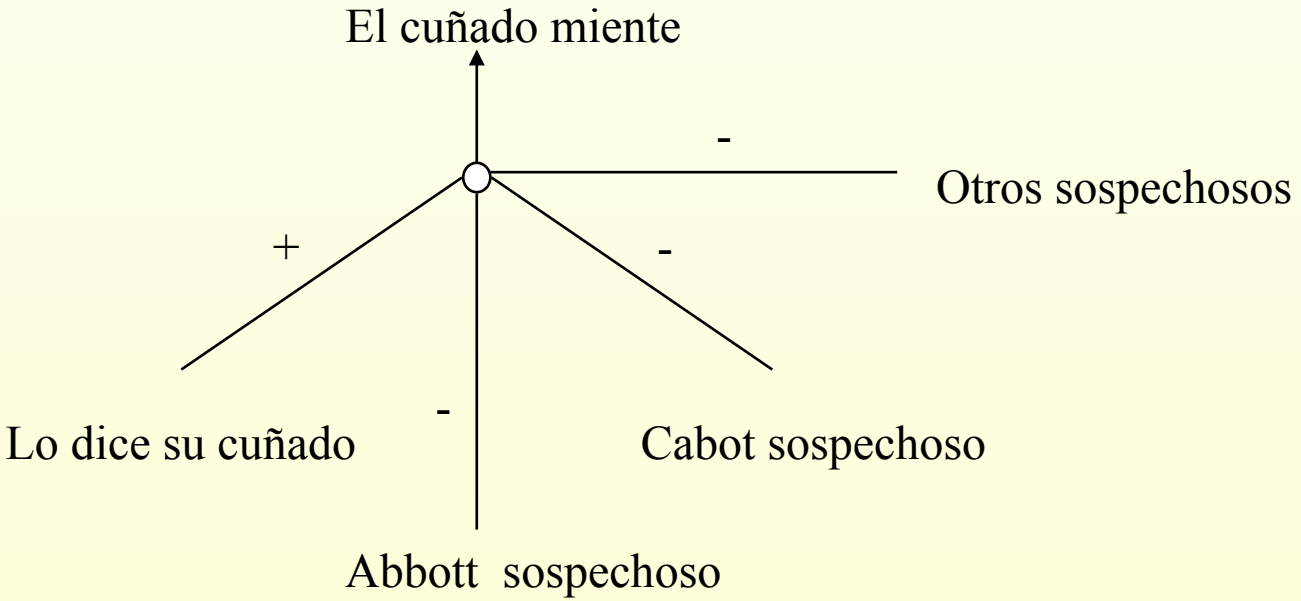
- **Una vez detectada una contradicción el JTMS debe presentar las alternativas para lograr que la contradicción vuelva a ser OUT**
  - En un JTMS puede hacerse que un nodo sea OUT consiguiendo que todas sus justificaciones sean inválidas.
    - Las justificaciones monótonas no pueden invalidarse sin retractar explícitamente de las aserciones hechas en la red.
    - Las justificaciones no monótonas pueden invalidarse afirmando algunos hechos que es necesario que no existan para la justificación. A estas aserciones con justificaciones no monótonas se las llama ***suposiciones***.
      - Una suposición se puede retirar haciendo que algún elemento de su lista-OUT sea IN (o recursivamente en algún elemento de la lista-OUT de la justificación de algún elemento en su lista-IN).
    - Pueden existir muchas suposiciones de este tipo a lo largo de una red grande de dependencias, pero la red nos da una forma de identificar aquellas que son relevantes para cada contradicción.
      - Se pueden utilizar los enlaces de dependencias para determinar un árbol Y/O de suposiciones candidatas a retirar.
      - Se retira mediante la justificación de otras creencias.

Eliminación contradicción del caso ABC

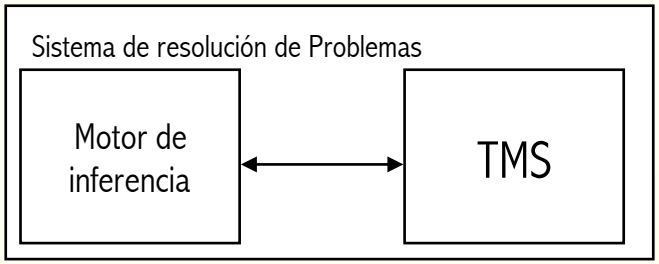
- La contradicción es en si misma una suposición, siempre y cuando su justificación sea válida. Se puede retirar:
  - Creyendo que hay otros sospechosos
  - Ceer de nuevo que alguno de los personajes es sospechoso.
    - Para creer esto último hay que poner en tela de juicio sus coartadas:
      - El registro del hotel falsificado
      - El cuñado de Babbitt mintió
      - Se trucaron las imagenes de TV
  - El algoritmo de vuelta atrás dirigido por dependencias nos informa por lo tanto de que existe un árbol O con cuatro nodos. **¿Qué deberíamos hacer?**
    - **Un JTMS no da la respuesta**. Esta elección es problema del resolutor de problemas que creó las dependencias en primer lugar.
    - Un JTMS también puede poseer algoritmos que creen justificaciones utilizando razonamiento abductivo:
      - Se cree que su cuñado miente, sólo porque sino hay contradicción. Esta suposición se pude retirar posteriormente. Por ejemplo, si aparece un pariente lejano.

Justificación abductiva

- Ejemplo de justificación abductiva para la creencia de que el cuñado de Babbitt mintió. Si llegamos a creer que Abbott o Cabot son sospechosos, o encontramos un pariente lejano, o de alguna forma creemos que el cuñado de Babbitt no dijo en realidad que Babbitt estaba en su casa, entonces la justificación para esta mentira se vuelve inválida.



Acciones de un TMS



- **Un TMS debe realizar las siguiente acciones**
  - Etiquetado consistente
  - Resolución de contradicciones
    - Nos muestra las suposiciones alternativas que se pueden hacer para eliminar una contradicción.
- **Un TMS no realiza (las tiene que realizar el programa de resolución de pro blemas):**
  - Aplicación de reglas para derivar conclusiones
  - Creación de justificaciones para los resultados de aplicar las reglas (aunque las justificaciones se crean como parte de la resolución de contradicciones).
  - Elección entre las distintas formas alternativas de resolver una contradicción.
  - Detección de contradicciones.

4.1.1 TMS en CLIPS

- **CLIPS permite especificar que la existencia de un hecho depende de la existencia de otro hecho o grupo de hechos. Las condiciones *logical* en la LHS de la regla son el mecanismo que ofrece CLIPS para el mantenimiento de la verdad:**
  - Por ejemplo, la siguiente regla indica que los bomberos deben utilizar la máscara de oxígeno si hay un fuego que produce humos nocivos

```
CLIPS>
(defrule humos-nocivos-presentes
     (emergencia (tipo fuego))
     (humos-nocivos-presentes)
 =>
  (assert
     (utiliza-mascara-oxigeno)))
CLIPS> (reset)
CLIPS> (watch facts)
CLIPS>
(assert (emergencia (tipo fuego))
        (humos-nocivos-presentes)))
==> f-1 (emergencia (tipo fuego))
==> f-2 (humos-nocivos-presentes)
<Fact-2>
CLIPS> (run)
==> f-3 (utiliza-mascara-oxigeno)
CLIPS> (retract 1 2)
<== f-1 (emergencia (tipo fuego))
<== f-2 (humos-nocivos-presentes)
CLIPS> (facts)
f-0     (initial-fact)
f-3     (utiliza-mascara-oxigeno)
For a total of 2 facts.
```

CLIPS: logical

```
CLIPS>
(defrule humos-nocivos-presentes
 (logical
   (emergencia (tipo fuego))
    (humos-nocivos-presentes))
 =>
 (assert
   (utiliza-mascara-oxigeno)))
CLIPS> (reset)
==> f-0     (initial-fact)
CLIPS>
(assert
  (emergencia (tipo fuego))
  (humos-nocivos-presentes))
==> f-1 (emergencia (tipo fuego))
==> f-2 (humos-nocivos-presentes)
<Fact-2>
```

```
CLIPS> (run)
==> f-3 (utiliza-mascara-
   oxigeno)
CLIPS> (retract 1)
<== f-1 (emergencia (tipo
   fuego))
<== f-3 (utiliza-mascara-
   oxigeno)
```

- `utiliza-mascara-oxigeno` recibe el **soporte lógico** de los hechos `emergencia` y `humos-nocivos-presentes`.
- `utiliza-mascara-oxigeno` es un hecho **dependiente** de `emergencia` y `humos-nocivos-presentes`.
- `emergencia` y `humos-nocivos-presentes` son **dependencias** de `utiliza-mascara-oxigeno` .
- **`logical` además de crear dependencias entre grupos de hechos, tiene un `and` implícito.**

CLIPS: logical

- `logical` **no tienen porque incluir todos los patrones de la LHS de la reglas**
  - **Si debe incluir el primero**, y no debe haber patrones sin `logical` entre patrones que tienen `logical`.

```
(defrule ok
   (logical (a))
   (logical (b))
   (c)
   =>
   (assert (d)))
```

```
(defrule not-ok-2
   (a)
   (logical (b))
   (logical (c))
   =>
   (assert (d)))
```

```
(defrule not-ok-1
   (logical (a))
   (b)
   (logical (c))
   =>
   (assert (d)))
```

```
(defrule not-ok-3
   (or (a)
       (logical (b)))
   (logical (c))
   =>
   (assert (d)))
```

- También es posible hacer que **los hechos dependan de la no existencia de hechos utilizando** not **delante de los patrones**. Incluso se pueden utilizar dentro de logical expresiones más completas utilizando exists, forall o combinaciones complejas.

Soportes lógicos de un hecho

- **Normalmente añadir a la lista de hechos un hecho que ya existe no tiene efecto. Pero un hecho dependiente lógicamente que es derivado de más de una fuente, no se elimina de la memoria de trabajo hasta que desaparecen todas sus soportes lógicos.**

```
CLIPS> (defrule rule1
            (logical (a))
            (logical (b))
            (c)
           => (assert (g) (h)))
CLIPS> (defrule rule2
            (logical (d))
            (logical (e))
            (f)
      =>   (assert (g) (h)))

CLIPS> (assert (a) (b) (c) (d) (e) (f))
==> f-1     (a)
==> f-2     (b)
==> f-3     (c)
==> Activation 0      rule1: f-1,f-2,f-3
==> f-4     (d)
==> f-5     (e)
==> f-6     (f)
==> Activation 0      rule2: f-4,f-5,f-6
```

```
CLIPS> (run)
FIRE    1 rule2: f-4,f-5,f-6     ;1ª regla añade soporte lógico
==> f-7     (g)
==> f-8     (h)
FIRE    2 rule1: f-1,f-2,f-3     ;2ª regla añade más soporte
CLIPS> (retract 1)
<== f-1     (a)
; Saca 1er soporte de (g) and (h)
CLIPS> (assert (h))
FALSE ; (h) sin soporte condicional
FALSE
CLIPS>(retract 4)
<== f-4     (d) ) ; Saca segundo soporte de (g)
<== f-7     (g)   ; (g) ya no tiene soporte
```

Soportes lógicos de un hecho

```
CLIPS>
(defrule humos-nocivos-presentes
 (logical
    (humos-nocivos-presentes))
    (emergencia  (tipo fuego))
 =>
 (assert
   (utiliza-mascara-oxigeno)))

(defrule extintores-de-gas-en-uso
   (logical
      (extintores-de-gas-en-uso))
      (emergencia  (tipo fuego))
 =>
(assert
   (utiliza-mascara-oxigeno)))
CLIPS> (reset)
==> f-0     (initial-fact)
```

```
CLIPS> (assert
(emergencia (tipo fuego))
(humos-nocivos-presentes)
(extintores-de-gas-en-uso))
==> f-1 (emergencia (tipo fuego))
==> f-2 (humos-nocivos-presentes)
==> f-3 (extintores-de-gas-en-uso)
<Fact-3>
CLIPS> (run)
FIRE 1 extintores-de-gas-en-uso: f-3,f-1
==> f-4    (utiliza-mascara-oxigeno)
FIRE 2 humos-nocivos-presentes: f-2,f-1
CLIPS> (retract 2)
<== f-2    (humos-nocivos-presentes)
CLIPS> (retract 3)
<== f-3    (extintores-de-gas-en-uso)
<== f-4    (utiliza-mascara-oxigeno)
CLIPS>
```

Instrucciones útiles

- `dependents` **y** `dependencies` **muestran dependencias entre hechos**

```
CLIPS> (facts)
f-0 (initial-fact)
f-1 (emergencia (tipo fuego))
f-2 (humos-nocivos-presentes)
f-3 (extintores-de-gas-en-uso)
f-4  (utiliza-mascara-oxigeno)
For a total of 5 facts.
CLIPS> (dependents 1)
None
CLIPS> (dependents 2)
f-4
CLIPS> (dependents 3)
f-4
CLIPS> (dependents 4)
None.
```

```
CLIPS> (dependencies 1)
None
CLIPS> (dependencies 2)
None
CLIPS> (dependencies 3)
None
CLIPS> (dependencies 4)
f-2
f-3
CLIPS>
```

exists

- **exists**
  - Un patrón **se reconoce si existe al menos un hecho**, sin mirar el número de hechos reconocidos.

```
CLIPS>(defrule alerta-por-emergencia
            (emergencia)
      =>
      (printout t
       "Emergencia: Alerta "crlf)
      (assert (operador-alerta)))
CLIPS>(reset)
CLIPS>
 (assert (emergencia (tipo fuego)))
<Fact-0>
CLIPS>
(assert (emergencia (tipo inundacion)))
<Fact-1>
CLIPS> (run)
Emergencia: Alerta
Emergencia: Alerta
```

```
CLIPS>(defrule alerta-por-emergencia
            (exists (emergencia))
      =>
      (printout t
       "Emergencia: Alerta "crlf)
      (assert (operador-alerta)))
CLIPS>(reset)
CLIPS>
 (assert (emergencia (tipo fuego)))
<Fact-0>
CLIPS>
(assert (emergencia (tipo inundacion)))
<Fact-1>
CLIPS> (run)
Emergencia: Alerta
```

forall

- **forall**
  - Permite que un reconocimiento se base en un conjunto de precondiciones que se satisfacen para cada ocurrencia de otro patrón.

```
(deftemplate emergencia (slot tipo)
            (slot localizacion))
(deftemplate cuadrilla (slot nombre)
             (slot localizacion))
(deftemplate evacuado (slot edificio))
(defrule todos-los-fuegos-atendidos
  (forall (emergencia (tipo fuego)  (localizacion ?lugar))
    (cuadrilla (localizacion ?lugar))
    (evacuado (edificio ?lugar)))
  =>
   (printout t
"Todos los edificios con fuego " crlf
"han sido evacuados y " crlf
"tienen bomberos en el sitio " crlf))
```

forall

```
CLIPS> (reset)
==> Activation 0
    todos-los-fuegos-atendidos: f-0,
CLIPS> (assert (emergencia (tipo fuego)
          (localizacion edificio-11)))
<== Activation 0
    todos-los-fuegos-atendidos: f-0,
<Fact-1>
CLIPS> (assert
    (evacuado (edificio edificio-11)))
<Fact-2>
CLIPS> (assert (cuadrilla (nombre A)
          (localizacion edificio-11)))
==> Activation 0
    todos-los-fuegos-atendidos: f-0,
<Fact-3>
CLIPS> (assert (cuadrilla (nombre B)
          (localizacion edificio-1)))
<Fact-4>
CLIPS> (assert (emergencia (tipo fuego)
          (localizacion edificio-1)))
<== Activation 0
    todos-los-fuegos-atendidos: f-0,
<Fact-5>
CLIPS> (assert
     (evacuado (edificio edificio-1)))
==> Activation 0
    todos-los-fuegos-atendidos: f-0,
<Fact-6>
CLIPS> (retract 4)
<== Activation 0
   todos-los-fuegos-atendidos: f-0,
CLIPS> (retract 5)
==> Activation 0
  todos-los-fuegos-atendidos: f-0,
CLIPS> (run)
Todos los edificios con fuego
han sido evacuados y
tienen bomberos en el sitio
```

4.2 TMS Basado en la Lógica

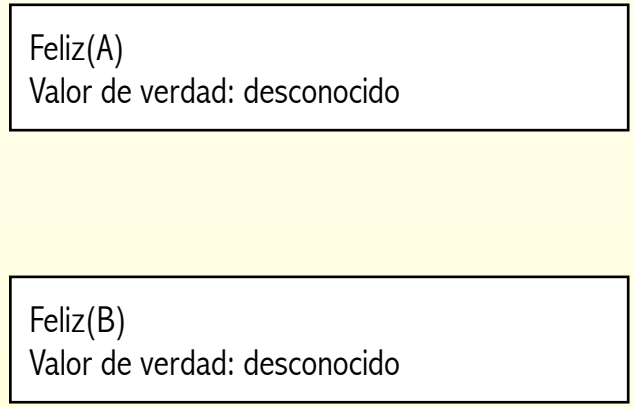
- **Un sistema de mantenimiento de la verdad basado en la lógica (LTMS) [McAllester 1980] es muy similar a un JTMS. Pero se diferencia en un aspecto importante:**
  - Un JTMS trata los nodos de la red como átomos, lo que significa que no hay relaciones entre ellos excepto las que se sitúan explícitamente en las justificaciones. Un JTMS no tiene problemas en etiquetar simultáneamente a P y Not(P) como IN.
  - En un LTMS se detectan contradicciones de este tipo de forma automática.

Demostración mediante propagación de restricciones

- **Implementación de un LTMS mediante combinación de la lógica con la propagación de restricciones:**
  - Las redes de propagación de verdad son como el ejemplo del circuito electrónico. la diferencia es que las puertas con conectivos lógicos `&`, `V`, `¬`, `=>` que aceptan argumentos y producen un juicio falso o verdad.
  - Las redes de propagación de verdad facilitan la tarea de mantener un registro de lo que sucede conforme se van haciendo o eliminando suposiciones.
  - Las redes de propagación de verdad permiten ver con mayor facilidad la forma en que el retrocesos no cronológico, dirigido por la dependencia, puede ayudar a corregir contradicciones que se suscitan cuando las suposiciones son incompatibles.

Ejemplo

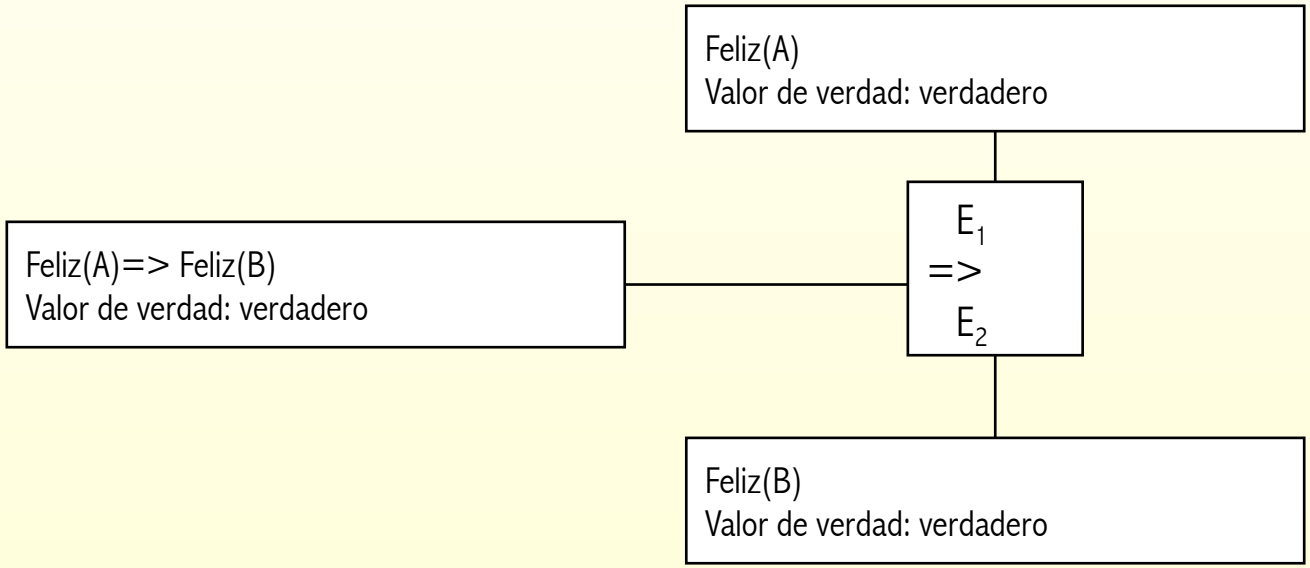
- **Considérese dos conocidos A y B, tal que a B le gusta A, de modo que si A es feliz, se concluye que B es feliz.**
  - ¿Cómo se pude utilizar la propagación de restricciones para hacer deducciones sobre quién es feliz?
    - Se necesita crear una caja de verdad para cada expresión y subexpresión lógicas que puede ser falsa o verdadera. Por ejemplo `Feliz(A)=>Feliz(B), F(A), F(B).`



✴ Inicialmente Feliz(A) tiene valor desconocido, pero si pasa a verdadero es de esperar que, por propagación de restricciones, esta expresión junto con Feliz(A)=> Feliz(B) limiten el valor de verdad de Feliz(B) a verdadero.

Red de propagación de verdad

- En general, cada conectivo lógico mantienen unido un par de subexpresiones y determina cómo los valores de verdad de éstas restringen el valor de verdad de su combinación.
  - Se precisa una caja de propagación de la verdad para cada conectivo lógico que aparezca en las expresiones. Ejemplo:



Expresión de restricción

- También es necesario saber cómo cada tipo de caja de propagación de verdad impone la restricción. De esta forma un valor determinado para una caja de verdad contribuye a determinar el valor de otras
  - Por ejemplo: E_1 => E_2 tiene las siguientes consecuencias lógicas:
    - ¬ E_1V E_2V¬(E_1 => E_2)
    - E_1V (E_1 => E_2)
    - ¬ E_2V (E_1 => E_2)
  - El valor de al menos un literal de cada expresión debe ser cierto. Para resaltar este punto podemos escribir las expresiones de la siguiente forma (se denominan **expresión de restricción**):
    - Requiere E_1 es falsa o E_2 es verdadera o (E_1 => E_2) es falsa
    - Requiere E_1 es verdadera o (E_1 => E_2) es verdadera
    - Requiere E_2 es falsa o (E_1 => E_2) es verdadera
  - Se requiere que al menos una de las subexpresiones de cada línea debe tener el valor estipulado de verdadero o falso. Por lo tanto si ninguna de las expresiones, excepto una, tienen el valor estipulado, se requiere que la restante lo tenga.

Implementación red de propagación

- Se pueden implementar las restricciones asociadas con las cajas de propagación de verdad mediante **demonios** que se ejecuten cuando se escribe el valor de alguna de las cajas de verdad conectadas a la caja de propagación de verdad.

- **Una red de propagación de verdad es una red de propagación de valores donde**
  - Las cajas de verdad contienen valores de verdad. Cada caja de verdad corresponde a un literal o a una expresión.
  - Las cajas de propagación de verdad imponen restricciones lógicas mediante demonios que se ejecutan al actualizar los valores de las cajas de verdad.
  - Por ejemplo, el código del demonio asociado a la actualización en cualquier terminal de una caja de verdad implicación que captura la primera expresión:
    - si E_1 es verdadera y E_2 es falsa escribe falso en la caja de la implicación.
    - sino si E_1 es verdadera y la implicación también escribe verdadera en la caja E_2
    - sino si E_2 es falsa y la implicación verdadera escribe falsa en la caja E_1

Obtención expresiones de restricción

- **¿Cómo se obtienen las expresiones de restricción?**
  - La tabla de verdad de la implicación es

E_1	E_2	$E_1 => E_2$
verdadera	falsa	falsa
falsa	falsa	verdadera
falsa	verdadera	verdadera
verdadera	verdadera	verdadera

  - Pero existen 2^3 posibles combinaciones de valores verdaderos y falsos. En consecuencia estas combinaciones deberían estar prohibidas:

E_1	E_2	$E_1 => E_2$
verdadera	falsa	verdadera
falsa	falsa	falsa
falsa	verdadera	falsa
verdadera	verdadera	falsa

Obtención expresiones de restricción

- Si expresamos lo mismo de otra manera, lo siguiente debe ser verdadero,pues cada línea excluye una de las cuatro combinaciones prohibidas:

  $\neg (E_1 \& \neg E_2 \& (E_1 => E_2)) \& \neg (\neg E_1 \& \neg E_2 \& \neg (E_1 => E_2))$
  $\& \neg (\neg E_1 \& E_2 \& \neg (E_1 => E_2)) \& \neg (E_1 \& E_2 \& \neg (E_1 => E_2))$

- Poniendo las negaciones de afuera adentro, se pueden reescribir las expresiones de esta forma:

  $(\neg E_1 \, V \, E_2 \, V \neg (E_1 => E_2)) \& (E_1 \; V \; E_2 \; V (E_1 => E_2))$
  $\& \; (E_1 \, V \neg E_2 \, V (E_1 => E_2)) \& \; (\neg E_1 \, V \neg E_2 \, V (E_1 => E_2))$

- Ahora podemos simplificar mediante la combinación de la segunda y tercera líneas, y de la tercera y cuarta, lo que produce la siguiente expresión:

  $(\neg E_1 \, V \, E_2 \, V \neg (E_1 => E_2))$
  $\& \; (E_1 \, V (E_1 => E_2)) \& \; (\neg E_2 \, V (E_1 => E_2))$

Expresiones de restricción

- **Se puede hacer lo mismo para el resto de operadores &, V, ¬ y quedarían las siguientes expresiones de restricción:**
  - Para las cajas de verdad **&**
    - Requiere E_1 es falsa o E_2 es falsa o (E_1 & E_2) es verdadera
    - Requiere E_1 es verdadera o (E_1 & E_2) es falsa
    - Requiere E_2 es verdadera o (E_1 & E_2) es falsa
  - Para las cajas de verdad **V**
    - Requiere E_1 es verdadera o E_2 es verdadera o (E_1 V E_2) es falsa
    - Requiere E_1 es falsa o (E_1 & E_2) es verdadera
    - Requiere E_2 es falsa o (E_1 & E_2) es verdadera
  - Para las cajas de verdad **¬**
    - Requiere E_1 es verdadera o ¬ E_1 es verdadera
    - Requiere E_1 es falsa o ¬ E_1 es falsa

La propagación de verdad puede establecer justificaciones

- **En el ejemplo si** A **es feliz,** B **también lo es. Ahora ampliamos el ejemplo con dos conocidos más** C **y** D**. A** C **le gusta** B**, de modo que si** B **es feliz, también lo es** C**. Finalmente, se supone que** A **y** D **son felices. Por lo tanto se tienen las siguientes suposiciones:**

```
Feliz(A) => Feliz(B)
Feliz(B) => Feliz(C)
Feliz(A)
Feliz(D)
```

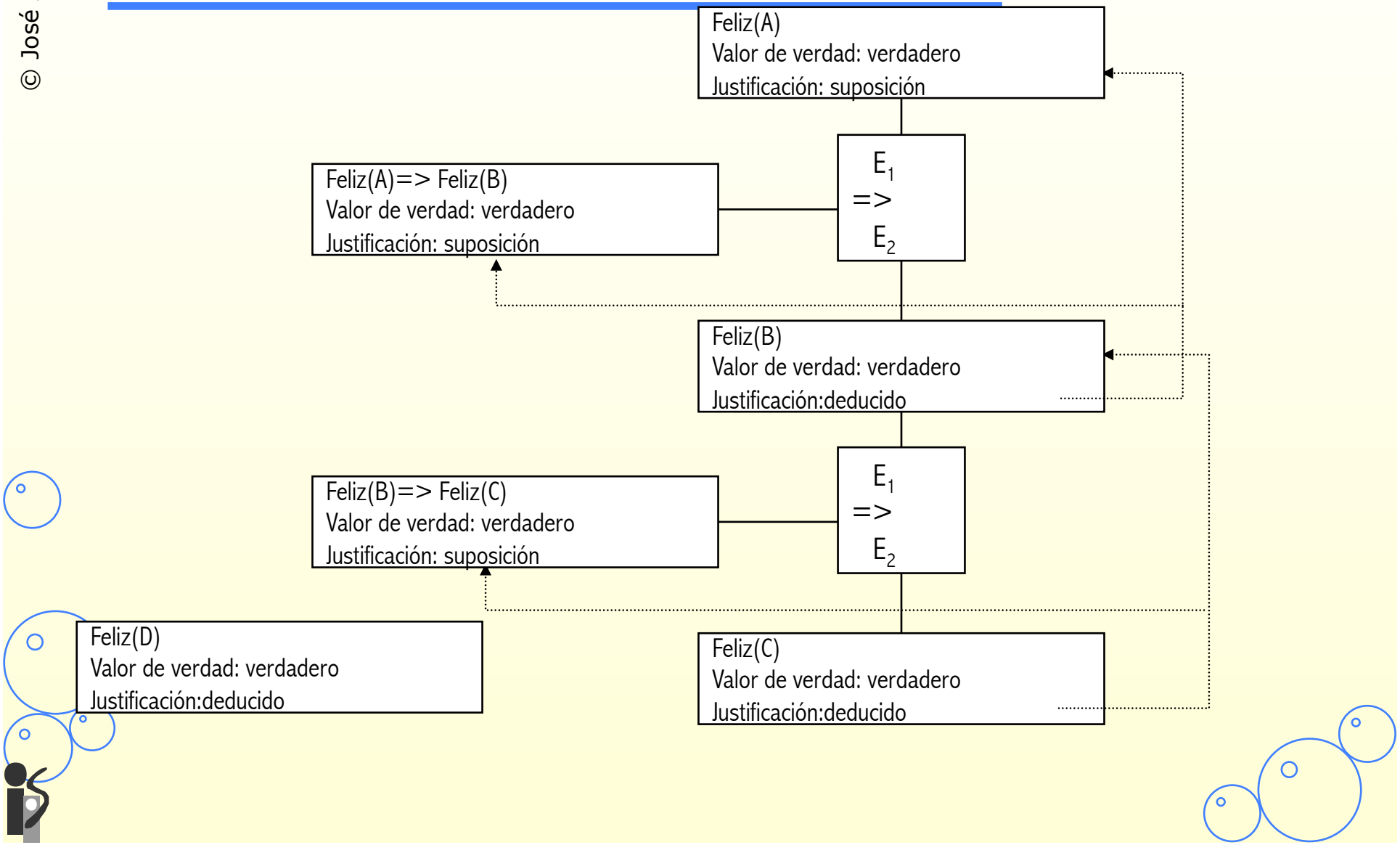
Ejemplo propagación

- Estas cuatro expresiones producen cuatro cajas de verdad y dos de propagación de verdad. Las primeras suposiciones, cada una con => , proporcionan las expresiones de restricción que determinan el comportamiento de las cajas de propagación de verdad:
  - `Requiere Feliz(A) es falsa o Feliz(B) es verdadera o (Feliz(A) => Feliz(B)) es falsa`
  - `Requiere Feliz(A) es verdadera o (Feliz(A) => Feliz (B)) es verdadera`
  - `Requiere Feliz(B) es falsa o (Feliz(A) => Feliz(B)) es verdadera`
  - `Requiere Feliz(B) es falsa o Feliz(C) es verdadera o (Feliz(B) => Feliz(C)) es falsa`
  - `Requiere Feliz(B) es verdadera o (Feliz(B) => Feliz (C)) es verdadera`
  - `Requiere Feliz(C) es falsa o (Feliz(B) => Feliz(C)) es verdadera`
- En este punto, las cajas de propagación de verdad llegan a la conclusión de que `Feliz(B)` y `Feliz(C)` son verdaderas.

Enlaces de justificación

- **Conforme se lleva a cabo la propagación de verdad, resulta fácil para las cajas de propagación de verdad mantener un registro de la forma en que los valores de verdad se determinan.**
  - En el registro, los **enlaces de justificación** van de cada uno de los valores de verdad a los valores de caja de verdad de la cual dependen los primeros.

Enlaces de justificación



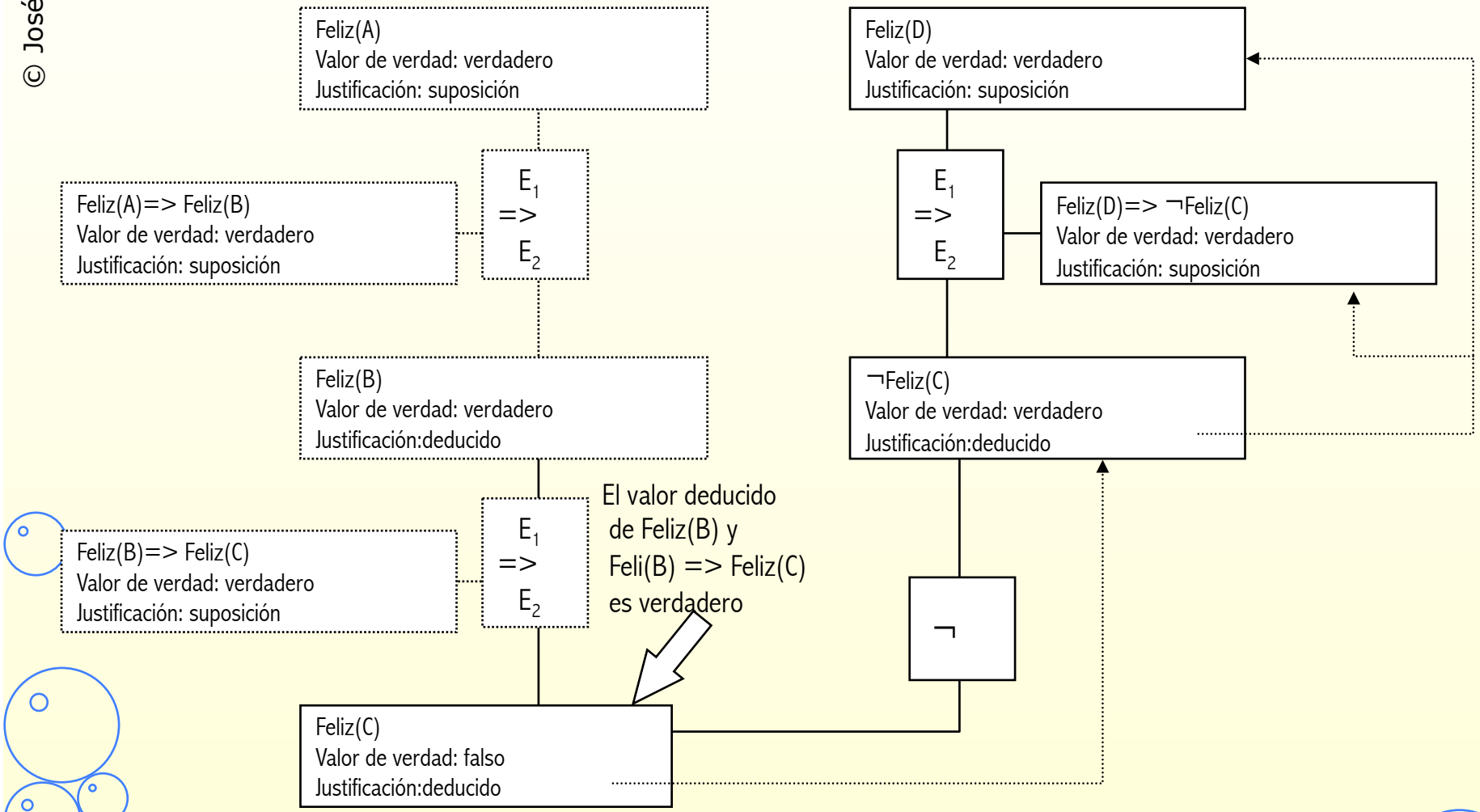
Cambio de opinión

- **Ahora se supone de manera tentativa, que a** C **no le gusta** D**, de modo que si** D **es feliz,** C **no lo es:**

```
Feliz(A) => Feliz(B)
Feliz(B) => Feliz(C)
Feliz(A)
Feliz(D)
Feliz(D) => ¬Feliz(C)
```

  - Esta nueva expresión produce dos cajas de verdad, y dos cajas de propagación de verdad adicionales. Las expresiones de restricción para las cajas de propagación son:
    - Requiere Feliz(D) es falsa o ¬ Feliz(C) es verdadera o (Feliz(D) => ¬ Feliz(C)) es falsa
    - Requiere Feliz(D) es verdadera o (Feliz(D) => ¬ Feliz(C)) es verdadera
    - Requiere ¬ Feliz(C) es falsa o (Feliz(D) => ¬ Feliz(C)) es verdadera
    - Requiere Feliz(C) es verdadera o ¬ Feliz(C)) es verdadera
    - Requiere Feliz(C) es falsa o ¬ Feliz(C)) es falsa

Ejemplo contradicción



Rastreo de las suposiciones

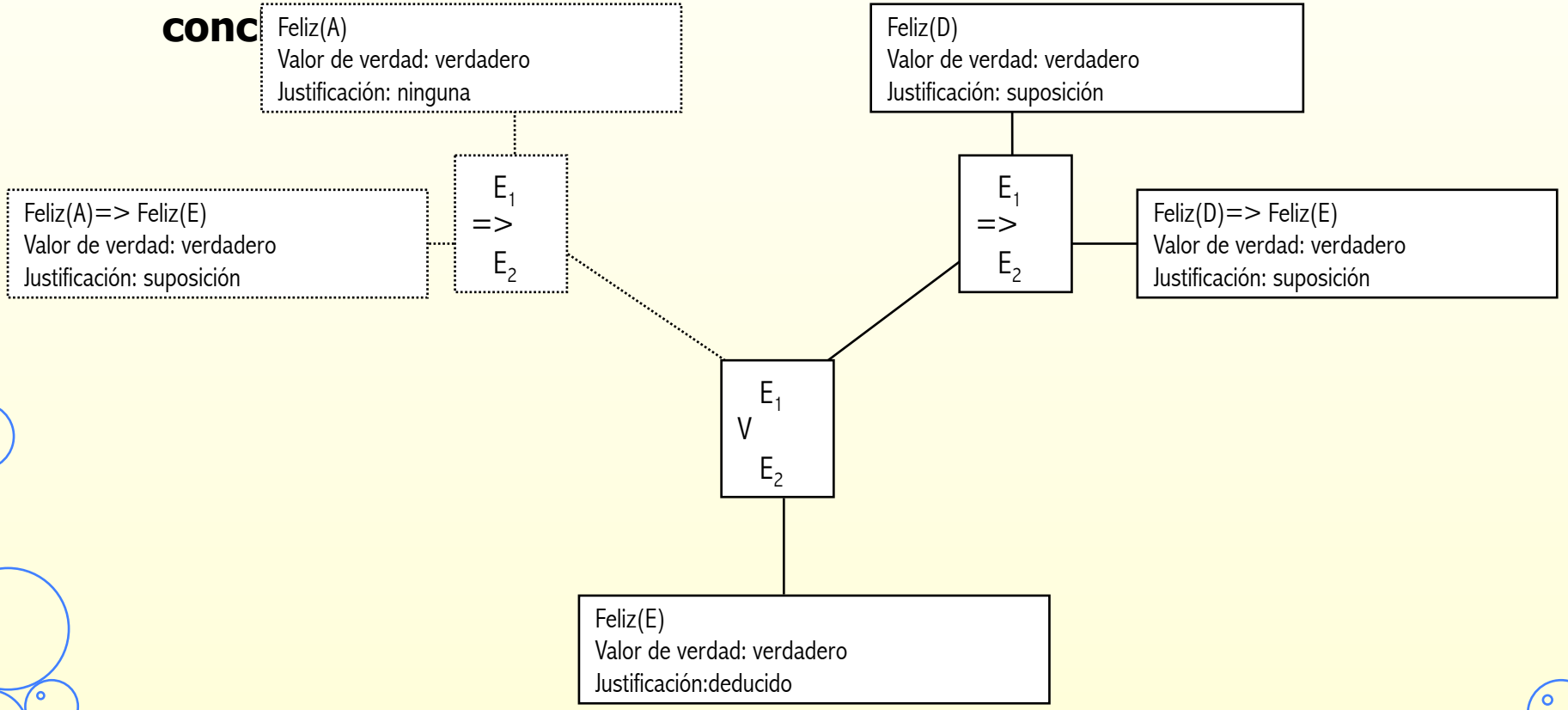
- **Cuando algo anda mal, es posible el rastreo, dirigido por las dependencias, las suposiciones que conducen a la contradicción**
  - En el ejemplo, al efectuar el rastreo desde el valor verdadero de C nos lleva a tres suposiciones:

    `Feliz(A) => Feliz(B)`
    `Feliz(B) => Feliz(C)`
    `Feliz(A)`
  - Al hacer el rastreo desde el valor falso de C llegamos a dos suposiciones:

    `Feliz(D) =>` ¬Feliz(C)
    `Feliz(D)`
  - Una vez identificadas todas las contribuciones a la contradicción, es el momento de considerar lo que se debe hacer. Se debe elegir cual es la suposición a eliminar.

Más de una justificación

- **Habrá casos en que más de un argumento respalde una conclusión, y la eliminación de uno de ellos no afecta en nada la conc**



Límites

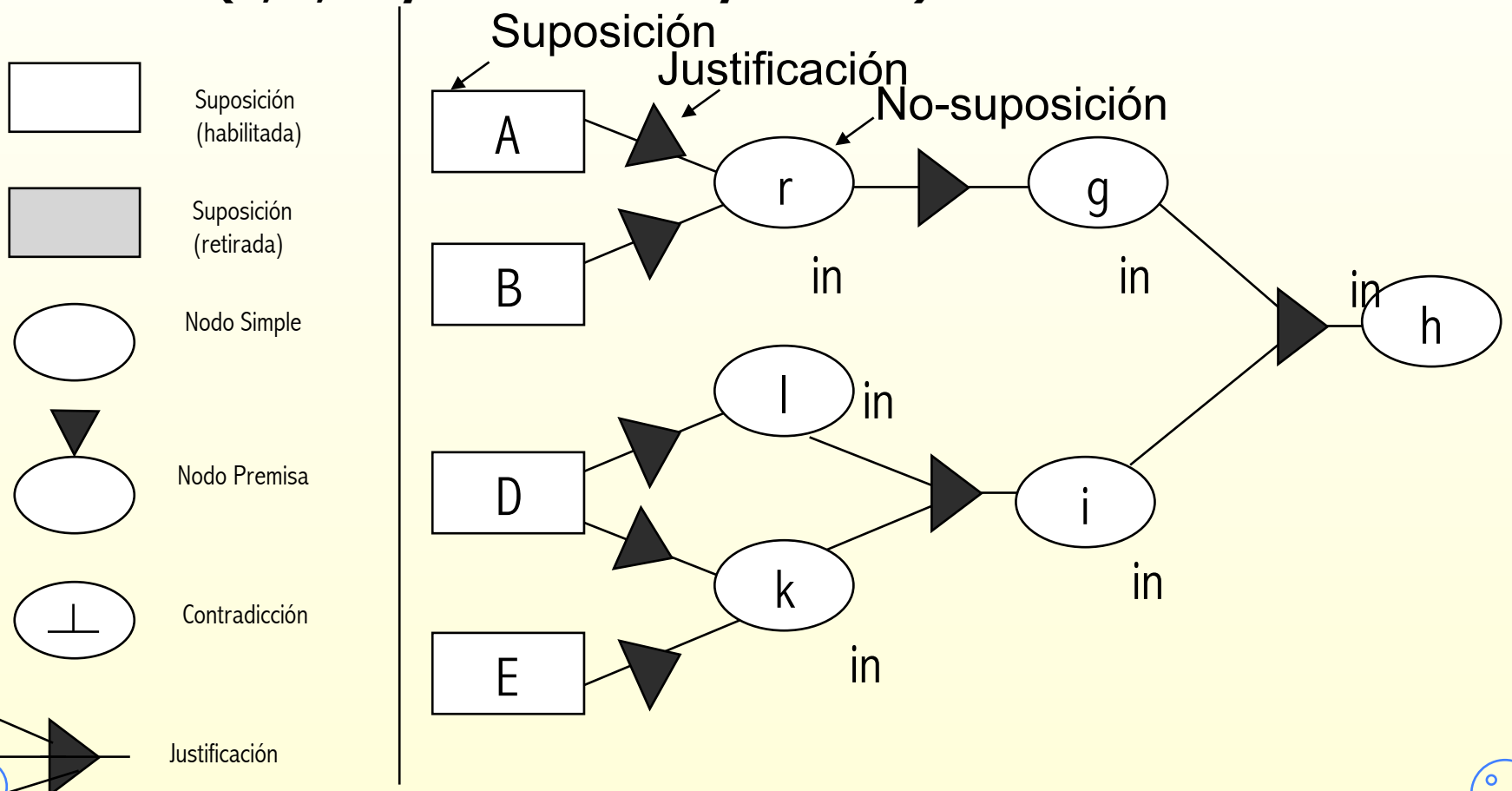
- **La propagación de verdad tiene límites:**
  - Sólo trata con el cálculo proposicional. No se pueden utilizar para tratar expresiones con variables.
  - No puede probar todas las expresiones verdaderas. No es un procedimiento de prueba completo.

5 Búsqueda en anchura

- **En un JTMS, en cualquier momento se pueden cambiar las suposiciones reetiquetando los nodos de una red de dependencia. Esto puede llevar un trabajo considerable.**

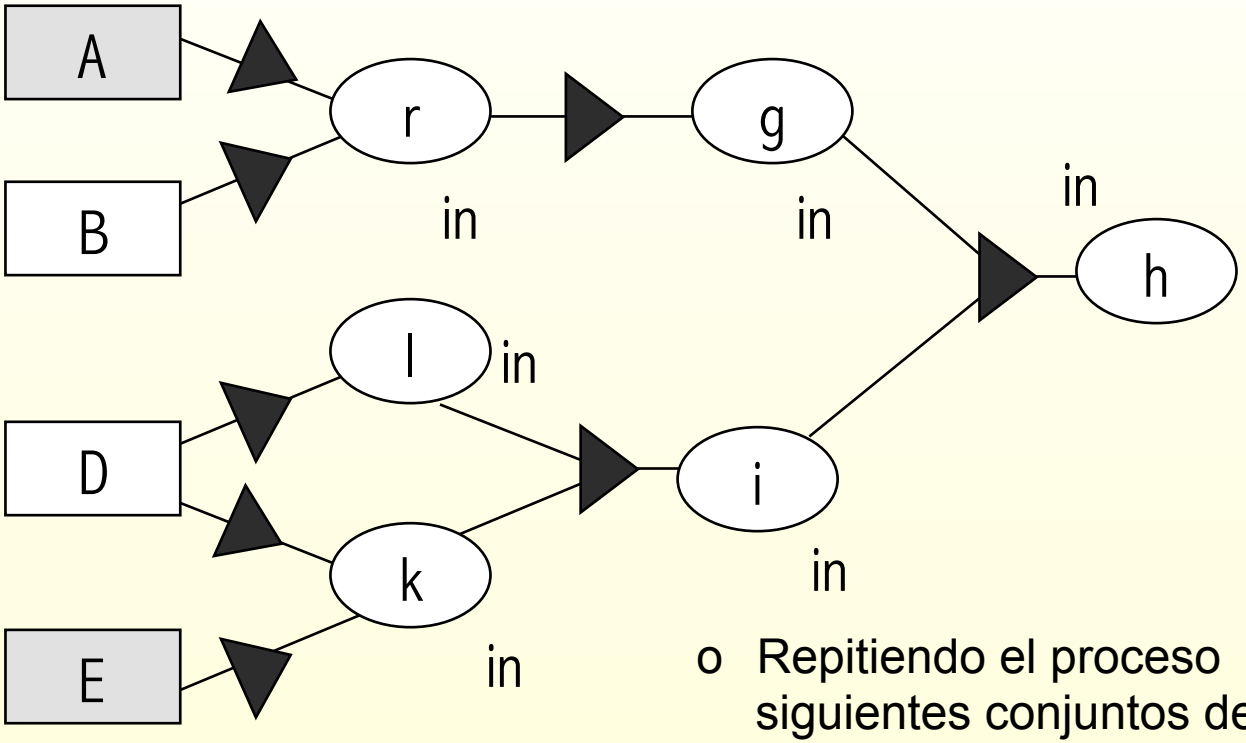
Etiquetas ATMS

- **JTMS (A, B, D y E habilitadas y h es in)**



Etiquetas ATMS

- **JTMS (A y E deshabilitadas y h es in)**



- Repitiendo el proceso h es in con los siguientes conjuntos de suposiciones:

{A, D} {A, B, D} {A, D, E} {A, B, D, E}

{B, D}, {B, D, E}

Etiquetas ATMS

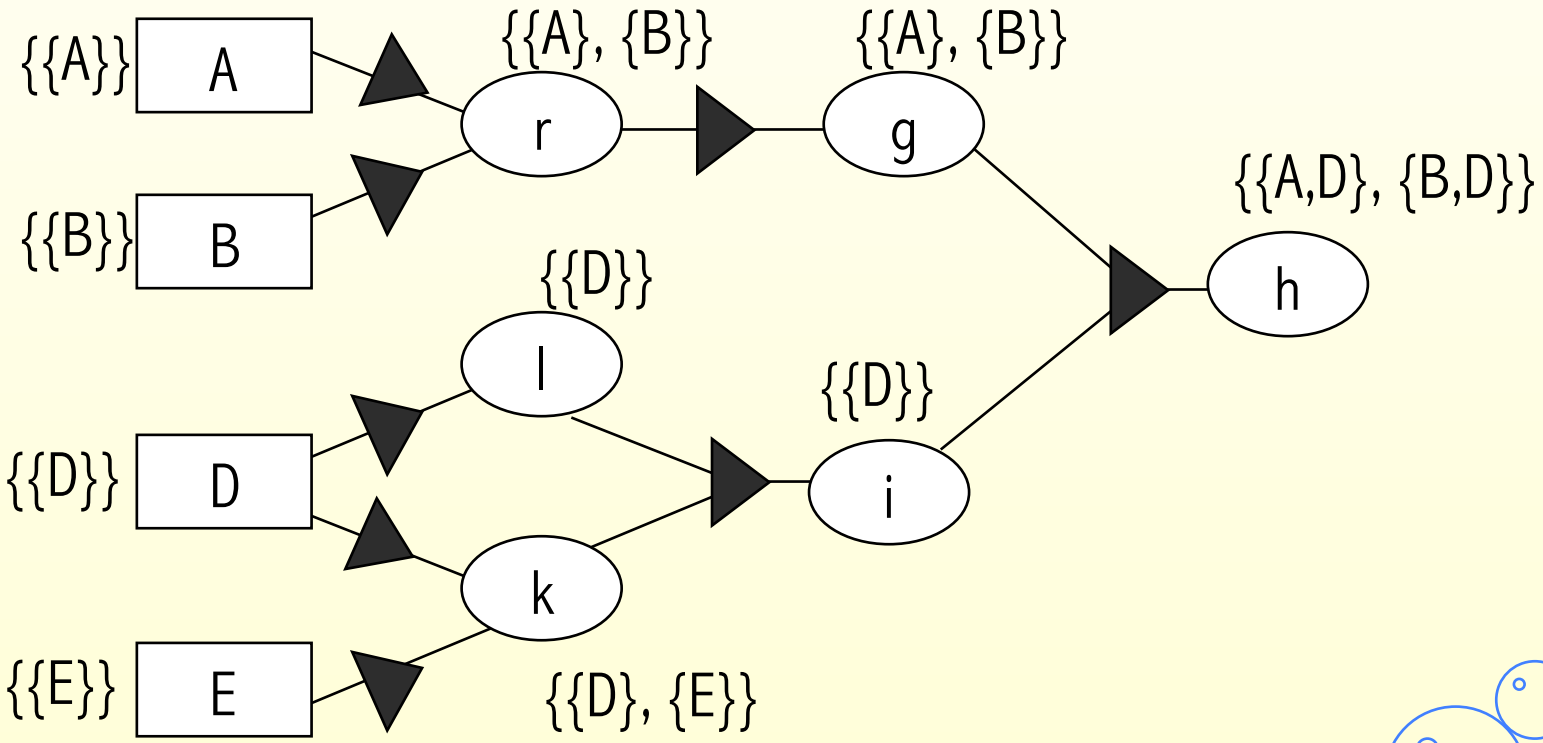
- **Denominamos**
  - E**ntorno** a un conjunto de suposiciones
  - Si averiguamos que un conjunto de aserciones A es inconsistente, lo denominamos conjunto **no-bueno** (**no-good** set), y cualquier conjunto B de aserciones que contienen A también será inconsistente.
  - Un entorno **consistente** es aquel que no es nogood.
  - El **contexto** de un entorno es el conjunto de nodos etiquetados **in** en el entorno.
- **Una forma de responder a preguntas sobre si un nodo es** :IN **en un entorno es registrar en la etiqueta del nodo todos los entornos consistentes en los que el nodo es** :IN**.**

Etiquetas ATMS

- Puesto que todos los entornos están simultáneamente disponibles, no es necesario añadir o eliminar sentencias, ni por lo tanto el reetiquetado de la red. Sólo se precisa cambiar de entorno.
- El problema de los Sistemas de mantenimiento de la verdad basados en suposiciones (ATMS, Assumption-Based Truth Maintenance) es que dadas n suposiciones, hay 2^n entornos.
  - Si un nodo es :IN en cualquier en todos los entornos tendremos su etiqueta contiene 2^n entornos.

Etiquetas ATMS

- Dos ideas hacen la idea utilizable:
  - Si un nodo se deduce de un entorno, se deducirá de cualquiera que contenga al primero. Por lo tanto la etiqueta del nodo h seria {{A,D} {B,D}}
  - Los nogoods no interesan, por lo que no se incluyen en la etiqueta.



Etiquetas ATMS

- Observación:
  - La etiqueta vacía <d,{}> indica que d no esta en ningún entorno. Las contradicciones tienen la etiqueta vacia
  - La etiqueta entorno vacío <p,{{}}> indica que p está en todos los entorno (es una premisa).

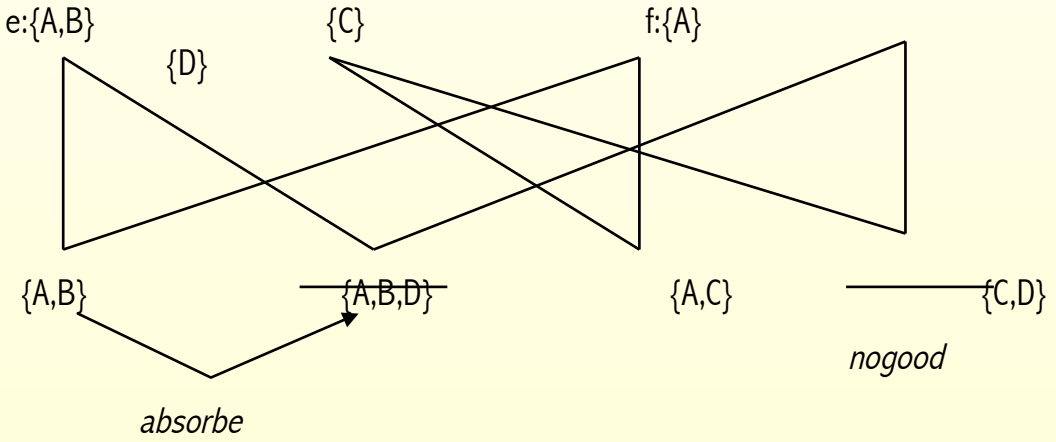
Algoritmo de etiquetado

- 1. Calcula una etiqueta tentativa

  $L' \mp \{U\ e_i \mid e_i\ L_{ik}\}$ dónde L_{ik} es la etiqueta del i-ésimo nodo de la k-ésima justificación para el nodo n.

- 2. Se saca de L' todos los nodos **nogood** y los entornos absorbidos por otros en L'

Ejemplo: <e, {{A,B} {C}}>, <f, {{A}{D}}>, nogood{C,D}



Utilización de un ATMS

- **En un ATMS se mantienen en paralelo varios caminos alternativos. La vuelta atrás se evita a expensas del mantenimiento de múltiples** entornos, **cada uno de los cuales se corresponde con un conjunto de suposiciones consistentes**.
  - El universo de entornos consistentes va podándose conforme se detectan contradicciones.
  - Los entornos consistentes que quedan se usan para etiquetar las suposiciones, de forma que indiquen los entornos en los que cada aserción tiene una justificación.

Ejemplo ATMS

- **Contamos con las siguientes suposiciones:**
  - A1. El registro del hotel se falsificó.
  - A2. El registro del hotel no se falsificó
  - A3. El cuñado de Babbitt mintió.
  - A4. El cuñado de Babbitt no mintió
  - A5. Cabot mintió.
  - A6. Cabot no mintió.
  - A7. A, B, C los únicos sospechosos.
  - 8. A, B, C no son los únicos sospechosos

Nodos	Justificaciones	Etiquetas de los nodos	
[1] El registro no se falsificó	{A2}	{A2}	{A2}
[2] Abbott en el hotel	[1] -> [2]	{A2}	{A2}
[3] El cuñado no mintió	{A4}	{A4}	{A4}
[4] Babbitt se apoya en su cuñado	[3] -> [4]	{A4}	{A4}
[5]Cabot no mintió	{A6}	{A6}	{A6}
[6] Cabot en el campeon. de esquí	[5] -> [6]	{A6}	{A6}
[7]A, B, C únicos sospechosos	{A7}	{A7}	{A7}
[8]Abbott principal sospechoso	[7] ^[13] ^[14]->[8]	{A7, A4, A6}	{A7, A4, A6}
[9]Babbitt principal sospechoso	[7] ^[12] ^[14]->[9]	{A7, A2, A6}	{A7, A2, A6}
[10]Cabot principal sospechoso	[7] ^[12] ^[13]->[10]	{A7, A2, A4}	{A7, A2, A4}

Ejemplo ATMS

Nodos	Justificaciones		Etiquetas de los nodos
[11]A,B,C no son los únicos sospechosos	{A8}	{A8}	{A8}
[12] Abbott no es el principal sospechoso	[2] -> [12]	{A2}	{A2}, {A8}
	[11] -> [12]	{A8}	
	[9] -> [12]	{A7, A2, A6}	
	[10] -> [12]	{A7, A2, A4}	
[13] Babbitt no es el principal sospechoso	[4] -> [13]	{A4}	{A4}, {A8}
	[11] -> [13]	{A8}	
	[8] -> [13]	{A7, A4, A6}	
	[10] -> [13]	{A7, A4, A2}	
[14] Cabot no es el principal sospechoso	[6] -> [14]	{A6}	{A6},{A8}
	[11] -> [14]	{A8}	
	[8] -> [14]	{A7, A4, A6}	
	[9] -> [14]	{A7, A2, A6}	

Ejemplo ATMS

- Se supone que el programa de resolución de problemas crea los nodos 1 a 14. El programa también puede indicar la siguiente contradicción marcando como no bueno:
  - A, B, C son los únicos sospechosos; A, B, C no son los únicos sopechosos: {A7, A8} -> Se generan las etiquetas mostradas.
  - Observaciones
    - `Nodo12: {A2} absorbe a la tercera y cuarta etiqueta`
    - `nodo 8:Las etiquetas resultan de la unión de las de los nodos 7, 13 y 14. <7,{{A7}}>, <13,{{A4},{A8}}>, <14,{{A6},{A8}}>`
      - {A7, ~~A4, A6}, {A7, A4, A8}, {A7, A8,~~ A6}, {A7, A8}
  - Si ahora el programa etiqueta como no bueno {A2} (El registro del hotel no se falsificó), se deben eliminar las etiquetas de los nodos 1, 2, 9, 10 y 12.
    - `En este momento el único nodo sospechosos con una etiqueta es el nodo 8. "Abbot principal sospechoso",`
    - `Pero el nodo 12 tiene todavía una etiqueta {A8} (derivada de A, B, C no son los únicos sospechosos).`
  - Es necesario un procesos de elección entre estos nodos.