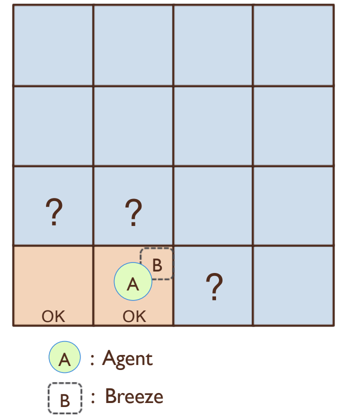
Tufts

Lecture 09:
Propositional Logic, II

Artificial Intelligence (CS 131)

1

Logic in the Wumpus World



- We can use PL to express the situation of the agent:
  - Let $P_{i,j}$ be true if there is a pit in location $[i, j]$
  - Let $B_{i,j}$ be true if there is a breeze in location $[i, j]$
  - The relevant KB is:

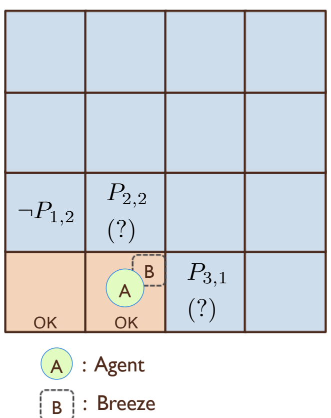
$$\{\neg P_{1,1}, \neg P_{2,1}, \neg B_{1,1}, B_{2,1}\}$$

- PL can also express Wumpus World rules like "Pits cause breezes in adjacent squares":

$$B_{1,1} \Leftrightarrow (P_{1,2} \lor P_{2,1})$$
$$B_{2,1} \Leftrightarrow (P_{1,1} \lor P_{2,2} \lor P_{3,1})$$

2

Inference in the Wumpus World



$$\{\neg P_{1,1}, \neg P_{2,1}, \neg B_{1,1}, B_{2,1}\}$$

$$B_{1,1} \Leftrightarrow (P_{1,2} \lor P_{2,1})$$
$$B_{2,1} \Leftrightarrow (P_{1,1} \lor P_{2,2} \lor P_{3,1})$$

- Based upon the knowledge base of basic propositional facts, and logical rules about how the world functions, we want to infer *new knowledge* about that world:

$$\{\neg P_{1,2}, (P_{2,2} \lor P_{3,1})\}$$

3

Inference Using Truth-Tables

Proposition Symbols							Sentences in the Knowledge Base				
$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	$\neg P_{1,1}$	$\neg B_{1,1}$	$B_{2,1}$	$B_{1,1} \Leftrightarrow (P_{1,2} \lor P_{2,1})$	$B_{2,1} \Leftrightarrow (P_{1,1} \lor (P_{2,2} \lor P_{3,1}))$
T	T	T	T	T	T	T	F	F	T	T	T
T	T	T	T	T	T	F	F	F	T	T	T
T	T	T	T	T	F	T	F	F	T	T	T
T	T	T	T	T	F	F	F	F	T	T	T
$\vdots$		$\vdots$		$\vdots$		$\vdots$		$\vdots$		$\vdots$	$\vdots$
F	T	F	F	F	T	T	T	T	T	T	T
F	T	F	F	F	T	F	T	T	T	T	T
F	T	F	F	F	F	T	T	T	T	T	T
$\vdots$		$\vdots$		$\vdots$		$\vdots$		$\vdots$		$\vdots$	$\vdots$
F	F	F	F	F	F	T	T	T	F	T	F
F	F	F	F	F	F	F	T	T	F	T	T

- We can use the full set of possible truth-values to check whether a sentence α is entailed by our knowledge base

4

Inference Using Truth-Tables

Proposition Symbols							Sentences in the Knowledge Base				
$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	$\neg P_{1,1}$	$\neg B_{1,1}$	$B_{2,1}$	$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$	$B_{2,1} \Leftrightarrow (P_{1,1} \vee (P_{2,2} \vee P_{3,1}))$
T	T	T	T	T	T	T	F	F	T	T	T
T	T	T	T	T	T	F	F	F	T	T	T
T	T	T	T	T	F	T	F	F	T	T	T
T	T	T	T	T	F	F	F	F	T	T	T
$\vdots$		$\vdots$		$\vdots$		$\vdots$		$\vdots$		$\vdots$	$\vdots$
F	T	F	F	F	T	T	T	T	T	T	T
F	T	F	F	F	T	F	T	T	T	T	T
F	T	F	F	F	F	T	T	T	T	T	T
$\vdots$		$\vdots$		$\vdots$		$\vdots$		$\vdots$		$\vdots$	$\vdots$
F	F	F	F	F	F	T	T	T	F	T	F
F	F	F	F	F	F	F	T	T	F	T	T

- Given the full set of truth values, we consider models/assignments for which *everything we know already* is true

Inference Using Truth-Tables

Proposition Symbols							Sentences in the Knowledge Base				
$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	$\neg P_{1,1}$	$\neg B_{1,1}$	$B_{2,1}$	$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$	$B_{2,1} \Leftrightarrow (P_{1,1} \vee (P_{2,2} \vee P_{3,1}))$
T	T	T	T	T	T	T	F	F	T	T	T
T	T	T	T	T	T	F	F	F	T	T	T
T	T	T	T	T	F	T	F	F	T	T	T
T	T	T	T	T	F	F	F	F	T	T	T
$\vdots$		$\vdots$		$\vdots$		$\vdots$		$\vdots$		$\vdots$	$\vdots$
F	T	F	F	F	T	T	T	T	T	T	T
F	T	F	F	F	T	F	T	T	T	T	T
F	T	F	F	F	F	T	T	T	T	T	T
$\vdots$		$\vdots$		$\vdots$		$\vdots$		$\vdots$		$\vdots$	$\vdots$
F	F	F	F	F	F	T	T	T	F	T	F
F	F	F	F	F	F	F	T	T	F	T	T

- Now, we can see that there *cannot* be a Pit at location $(1,2)$, since that sentence is false in every model consistent with what we already know

Inference Using Truth-Tables

Proposition Symbols							Sentences in the Knowledge Base				
$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	$\neg P_{1,1}$	$\neg B_{1,1}$	$B_{2,1}$	$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$	$B_{2,1} \Leftrightarrow (P_{1,1} \vee (P_{2,2} \vee P_{3,1}))$
T	T	T	T	T	T	T	F	F	T	T	T
T	T	T	T	T	T	F	F	F	T	T	T
T	T	T	T	T	F	T	F	F	T	T	T
T	T	T	T	T	F	F	F	F	T	T	T
$\vdots$		$\vdots$		$\vdots$		$\vdots$		$\vdots$		$\vdots$	$\vdots$
F	T	F	F	F	T	T	T	T	T	T	T
F	T	F	F	F	T	F	T	T	T	T	T
F	T	F	F	F	F	T	T	T	T	T	T
$\vdots$		$\vdots$		$\vdots$		$\vdots$		$\vdots$		$\vdots$	$\vdots$
F	F	F	F	F	F	T	T	T	F	T	F
F	F	F	F	F	F	F	T	T	F	T	T

- Similarly, we can see that there is a Pit either at $(2,2)$ or $(3,1)$, but we don't know which, since either one could be false (although *not both*)

Inference Techniques

- Inference is the procedure of deriving (i.e., generating) new sentences based on our existing knowledge base

$$KB \vdash_P \alpha \;\equiv\; \alpha \text{ can be derived from } KB \text{ using procedure } P$$

- Procedure P is sound if it only ever leads to sentences that are **entailed** by the knowledge base:

$$KB \vdash_P \alpha \;\Rightarrow\; KB \models \alpha$$

- Procedure P is complete if it can generate *all* sentences that are entailed by the knowledge base:

$$KB \models \alpha \;\Rightarrow\; KB \vdash_P \alpha$$

Effective Inference Procedures

- Enumerating all truth tables to derive new sentences can be automated (algorithm: Figure 7.10, page 221 of R&N text)
- This procedure *is* sound and complete; however, it is ineffective due to complexity:
  - For n propositional symbols, the full truth-table has 2^n rows
  - If it took only a nanosecond to check each row, then checking:

10 symbols	$= 0.000001s$		
20 symbols	$= 0.0015s$		
30 symbols	$= 1.1s$		
40 symbols	$= 1100s$	$=$	18.3 minutes
50 symbols	$= 1,125,900s$	$=$	13 days
60 symbols	$= 12 \times 10^9 s$	$=$	36.6 years
$\vdots$			
100 symbols	$= 1.27 \times 10^{21} s$	$=$	40.2 *trillion* years

- For Breeze and Pit only, (4 × 4) Wumpus World has 32 symbols
- For Breeze, Stench, Glitter, Pit, Gold, Wumpus, we already get 96!

A Special Case: Definite/Horn Form

- In some cases, we can represent *KB* in a special *restricted* PL syntax
- A definite clause α is either (1) a literal: either P_i or $\neg P_i$, for some propositional symbol P_i, or (2) a sentence of the special form:

Tail $\boxed{P_i \wedge \cdots \wedge P_j} \Rightarrow \boxed{P_k}$ Head

- Definite clauses are equivalent to disjunctions with *exactly one* positive symbol

$$P_1 \wedge P_2 \wedge \cdots \wedge P_j \Rightarrow P_k \equiv \neg(P_1 \wedge P_2 \wedge \cdots P_j) \vee P_k$$
$$\equiv \neg P_1 \vee \neg P_2 \vee \cdots \vee \neg P_j \vee P_k$$

- A slightly more relaxed definition: Horn clause, which has *at most one* positive symbol; both of the following are Horn clauses, but only the *first* is definite

$$\neg P_1 \vee \neg P_2 \vee \neg P_3 \vee P_4$$
$$\neg P_1 \vee \neg P_2 \vee \neg P_3 \vee \neg P_4$$

A Special Case: Definite/Horn Form

- If our knowledge base is made up only of definite clauses, then we can repeatedly apply the rule of Modus Ponens (*MP*):

$$\frac{\alpha_1 \quad \alpha_2 \quad \cdots \quad \alpha_n \quad (\alpha_1 \wedge \alpha_2 \wedge \cdots \wedge \alpha_n) \Rightarrow \beta}{\beta}$$

- This can be written in disjunctive form as well:

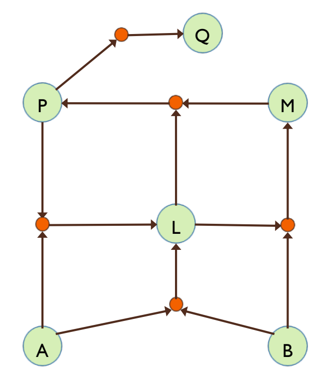
$$\frac{\alpha_1 \quad \alpha_2 \quad \cdots \quad \alpha_n \quad \neg\alpha_1 \vee \neg\alpha_2 \vee \cdots \vee \neg\alpha_n \vee \beta}{\beta}$$

- This rule gives us a sound and complete inference procedure that can be employed in simple linear-time algorithms

Forward Chaining with Definite KBs

- When we have a set of definite clauses, we can represent the structure of our knowledge base graphically

KB		
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		



Forward Chaining with Definite KBs

Image source: Russell & Norvig (2021)

```
function PL-FC-ENTAILS?(KB, q) returns true or false
  inputs: KB, the knowledge base, a set of propositional definite clauses
          q, the query, a proposition symbol
  count ← a table, where count[c] is initially the number of symbols in clause c's premise
  inferred ← a table, where inferred[s] is initially false for all symbols
  queue ← a queue of symbols, initially symbols known to be true in KB

  while queue is not empty do
     p ← POP(queue)
     if p = q then return true
     if inferred[p] = false then
        inferred[p] ← true
        for each clause c in KB where p is in c.PREMISE do
           decrement count[c]
           if count[c] = 0 then add c.CONCLUSION to queue
  return false
```

For definite clauses that are equivalent to conditionals, count how many propositions make up its tail

- For example, the following clause has *count* = 3:

$$\neg P_1 \lor \neg P_2 \lor \neg P_3 \lor P_4 \;\equiv\; (P_1 \land P_2 \land P_3) \Rightarrow P_4$$

Artificial Intelligence (CS 452) 13

13

Forward Chaining with Definite KBs


```
function PL-FC-ENTAILS?(KB, q) returns true or false
  inputs: KB, the knowledge base, a set of propositional definite clauses
          q, the query, a proposition symbol
  count ← a table, where count[c] is initially the number of symbols in clause c's premise
  inferred ← a table, where inferred[s] is initially false for all symbols
  queue ← a queue of symbols, initially symbols known to be true in KB

  while queue is not empty do
     p ← POP(queue)
     if p = q then return true
     if inferred[p] = false then
        inferred[p] ← true
        for each clause c in KB where p is in c.PREMISE do
           decrement count[c]
           if count[c] = 0 then add c.CONCLUSION to queue
  return false
```


Premise elimination: if we see part of the tail, decrement *count*

Modus Ponens: if we have seen *all* parts of tail, infer head

Artificial Intelligence (CS 452) 14

14

Backwards Chaining

- A similar algorithm works *backward* from the thing we want to prove, *q*, checking if it is already in KB
  - If not, then we check for a Horn clause that has q as its head
  - If so, we repeat the process for all premises in the tail
  - Terminate (and fail) if any premise can't be proved, else succeed
- Both FC and BC are linear-time algorithms:
  - By keeping track of whether we have used a premise before (either to "fire" a clause in FC or as an object of proof in BC), we visit each propositional symbol at most once
  - For n sentences in our *KB*, the algorithms run in $O(n)$
  - BC can often do even better, avoiding unnecessary work

Thursday, 06 Oct. 2011 Artificial Intelligence (CS 452) 15

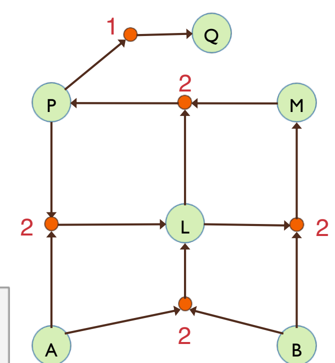
15

Forward Chaining with Definite KBs, 01

- Initial *count* set to # premises in tail of conditionals

KB
$P \Rightarrow Q$
$L \land M \Rightarrow P$
$B \land L \Rightarrow M$
$A \land P \Rightarrow L$
$A \land B \Rightarrow L$
A
B

queue : basic propositions *already* known



Thursday, 06 Oct. 2011 Artificial Intelligence (CS 452) 16

16

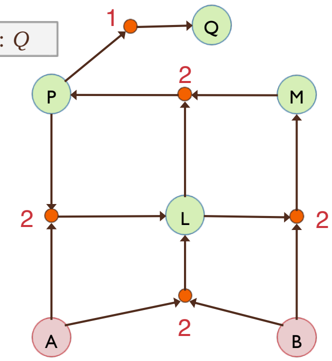
Forward Chaining with Definite KBs, 02

- Suppose we are trying to infer Q : we begin to "fire" clauses using Modus Ponens rules, decrementing *count*

KB

$$\begin{aligned} P &\Rightarrow Q \\ L \wedge M &\Rightarrow P \\ B \wedge L &\Rightarrow M \\ A \wedge P &\Rightarrow L \\ A \wedge B &\Rightarrow L \\ &A \\ &B \end{aligned}$$

query : Q



17

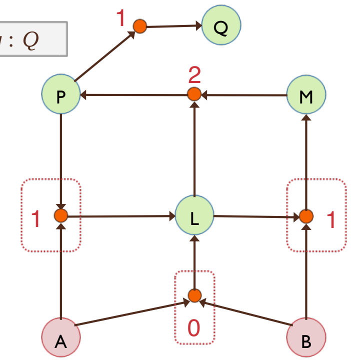
Forward Chaining with Definite KBs, 03

- Known facts in the queue lead to decremented *count*

KB

$$\begin{aligned} P &\Rightarrow Q \\ L \wedge M &\Rightarrow P \\ B \wedge L &\Rightarrow M \\ A \wedge P &\Rightarrow L \\ A \wedge B &\Rightarrow L \\ &A \\ &B \end{aligned}$$

query : Q



18

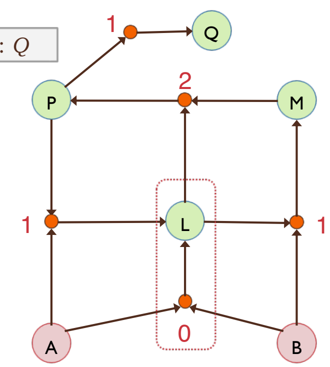
Forward Chaining with Definite KBs, 04

- Whenever *count* reaches 0, we add the associated head of the conditional to the queue

KB

$$\begin{aligned} P &\Rightarrow Q \\ L \wedge M &\Rightarrow P \\ B \wedge L &\Rightarrow M \\ A \wedge P &\Rightarrow L \\ A \wedge B &\Rightarrow L \\ &A \\ &B \\ &L \end{aligned}$$

query : Q



19

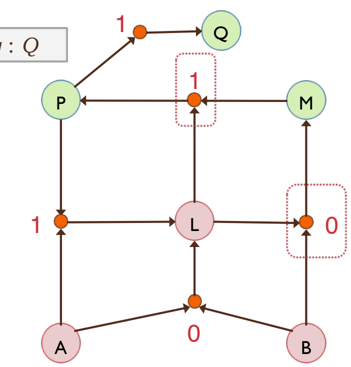
Forward Chaining with Definite KBs, 05

- We repeat the process so long as something is in the queue, and we have not yet arrived at our query

KB

$$\begin{aligned} P &\Rightarrow Q \\ L \wedge M &\Rightarrow P \\ B \wedge L &\Rightarrow M \\ A \wedge P &\Rightarrow L \\ A \wedge B &\Rightarrow L \\ &A \\ &B \\ &L \end{aligned}$$

query : Q



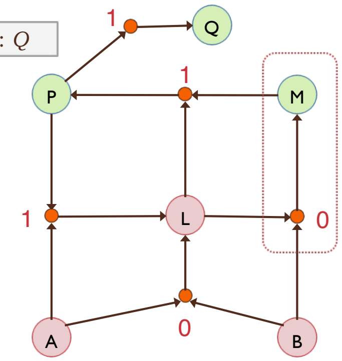
20

Forward Chaining with Definite KBs, 06

- We repeat the process so long as something is in the queue, and we have not yet arrived at our query

KB
$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B
L
$\boxed{M}$

$query : Q$



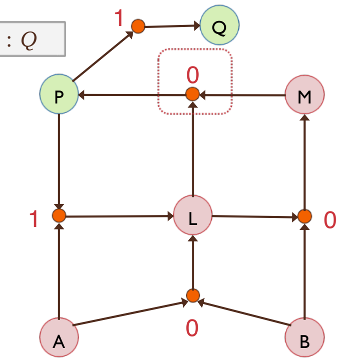
21

Forward Chaining with Definite KBs, 07

- We repeat the process so long as something is in the queue, and we have not yet arrived at our query

KB
$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B
L
$\boxed{M}$

$query : Q$



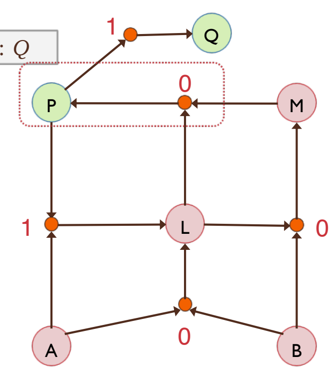
22

Forward Chaining with Definite KBs, 08

- We repeat the process so long as something is in the queue, and we have not yet arrived at our query

KB
$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B
L
M
$\boxed{P}$

$query : Q$



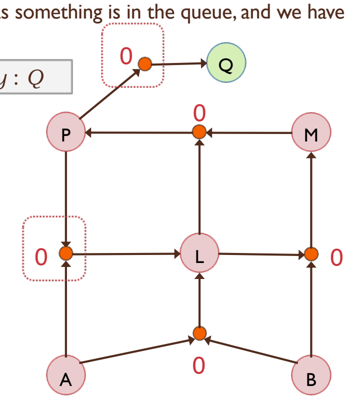
23

Forward Chaining with Definite KBs, 09

- We repeat the process so long as something is in the queue, and we have not yet arrived at our query

KB
$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B
L
M
$\boxed{P}$

$query : Q$



24

Forward Chaining with Definite KBs, 10

- We conclude when either every count is 0, or query is in the queue

KB

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B

L

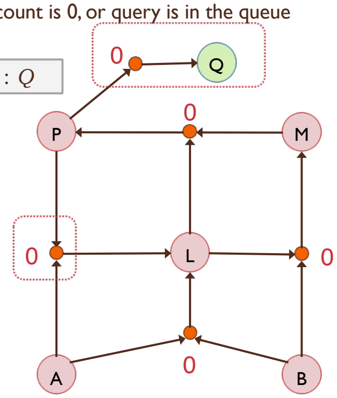
M

P

Q

$query : Q$

SUCCESS



A General Algorithm: Resolution

- While definite/Horn clauses cannot express everything in PL, we can put *any* sentence into Conjunctive Normal Form (CNF): a conjunction where each clause is a disjunction of positive/negative literals, e.g.:

$$(\neg P_1 \vee P_2) \wedge (P_3 \vee P_4 \vee \neg P_5) \wedge P_6$$

- For disjunctive clauses, resolution is a sound & complete inference rule:

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

where l_i and m_j are complementary—i.e., one is the negation of the other:

$$\frac{P_{1,3} \vee P_{2,2} \qquad \neg P_{2,2}}{P_{1,3}} \qquad \frac{\neg P_{2,4} \vee P_{1,1} \qquad P_{3,3} \vee P_{2,4}}{P_{1,1} \vee P_{3,3}}$$

Converting to CNF

- *Every* sentence of PL can be written in CNF by a simple set of conversions. For example:

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

1. Eliminate $\Leftrightarrow$, replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$:

$$(S_{1,1} \Rightarrow (\neg P_{1,2} \vee \neg P_{2,1})) \wedge ((\neg P_{1,2} \vee \neg P_{2,1}) \Rightarrow S_{1,1})$$

2. Eliminate $\Rightarrow$, replacing $\alpha \Rightarrow \beta$ with $\neg\alpha \vee \beta$:

$$(\neg S_{1,1} \vee \neg P_{1,2} \vee \neg P_{2,1}) \wedge (\neg(\neg P_{1,2} \vee \neg P_{2,1}) \vee S_{1,1})$$

3. Convert $\neg(\alpha \vee \beta)$ to $(\neg\alpha \wedge \neg\beta)$, by de Morgan's rules:

$$(\neg S_{1,1} \vee \neg P_{1,2} \vee \neg P_{2,1}) \wedge ((\neg\neg P_{1,2} \wedge \neg\neg P_{2,1}) \vee S_{1,1})$$

4. Eliminate any double negations:

$$(\neg S_{1,1} \vee \neg P_{1,2} \vee \neg P_{2,1}) \wedge ((P_{1,2} \wedge P_{2,1}) \vee S_{1,1})$$

5. Apply distributive laws ($\vee$ over $\wedge$):

$$(\neg S_{1,1} \vee \neg P_{1,2} \vee \neg P_{2,1}) \wedge (P_{1,2} \vee S_{1,1}) \wedge (P_{2,1} \vee S_{1,1})$$

The Resolution Algorithm

Image source: Russell & Norvig (2021)

- We can use repeated applications of resolution to show $(KB \wedge \neg\alpha)$ is unsatisfiable, where α is some query we are interested in (NB: negated)
  - If achieve empty set of clauses, this shows that the KB and the query $\neg\alpha$ cannot *both* be true at the same time; by definition, this means that KB entails α
  - A form of proof by contradiction

function PL-RESOLUTION(KB, α) **returns** *true* or *false*
  inputs: KB, the knowledge base, a sentence in propositional logic
    α, the query, a sentence in propositional logic

  clauses ← the set of clauses in the CNF representation of $KB \wedge \neg\alpha$
  new ← { }
  while *true* **do**
    for each pair of clauses C_i, C_j **in** *clauses* **do**
      resolvents ← PL-RESOLVE(C_i, C_j)
      if *resolvents* contains the empty clause **then return** *true*
      new ← *new* ∪ *resolvents*
    if *new* ⊆ *clauses* **then return** *false*
    clauses ← *clauses* ∪ *new*

The basic resolution rule, applied as many times as appropriate

No entailment if we ever run out of new things to resolve

Using Resolution

Suppose we want to prove: $\neg P_{1,2}$ using KB: $(B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})) \wedge \neg B_{1,1}$

Convert KB to CNF: $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge \neg B_{1,1}$

Equivalent to clauses: $\{(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}), (\neg P_{1,2} \vee B_{1,1}), (\neg P_{1,2} \vee B_{1,1}), \neg B_{1,1}\}$

Add in the negation of what we want to prove: $\neg\neg P_{1,2} \equiv P_{1,2}$

$\underbrace{\boxed{\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}} \; \boxed{\neg P_{1,2} \vee B_{1,1}} \; \boxed{\neg P_{1,2} \vee B_{1,1}} \; \boxed{\neg B_{1,1}}}_{\text{Original } KB} \; \boxed{P_{1,2}}$

Original *KB*

Negation of what we want to prove

Using Resolution

- Resolution rule will not *always* be *productive*: it can sometimes generate consequences that are not really helpful to our proof:

$\boxed{\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}} \; \boxed{\neg P_{1,2} \vee B_{1,1}} \; \boxed{\neg P_{1,2} \vee B_{1,1}} \; \boxed{\neg B_{1,1}} \; \boxed{P_{1,2}}$

$\boxed{\neg B_{1,1} \vee P_{2,1} \vee B_{1,1}}$

Using Resolution

- However, it will also eventually generate *everything* that is entailed by *KB* (a complete procedure):

$\boxed{\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}} \; \boxed{\neg P_{1,2} \vee B_{1,1}} \; \boxed{\neg P_{1,2} \vee B_{1,1}} \; \boxed{\neg B_{1,1}} \; \boxed{P_{1,2}}$

$\boxed{\neg B_{1,1} \vee P_{2,1} \vee B_{1,1}}$

$\boxed{\neg P_{1,2}}$

Empty Clause: implies a contradiction (something *cannot* be satisfied), and so $\neg P_{1,2}$ is entailed by *KB*

$\boxed{\emptyset}$

Resolution as an Algorithm

- Resolution is a sound and complete inference method
- However, while *often* effective, we must live with the fact that in the worst case it can run in exponential time
  - Given n propositional symbols, we may need 2^n applications of basic resolution rule to reach a stopping condition
- This *seems to be* a necessary fact for any general-purpose PL inference procedure: it is well-known that the PL entailment/provability question is NP-complete