Propositional Logic

Logic is a language. We use this language all the time to make **statements** and **reason** about them. For example, you may have heard your parents and teachers saying the following things: "If you study well in BE and get good marks then you will get a good job". Also, "Once you have a good, well paying job, your life will be settled and happy". Implicit in these assertions is the fact that "if you study well in BE then your life will eventually be happy". Clearly, we unconsciously make such assertions and deductions in our day to day life but a robotic agent can not. Furthermore, if the set of assertions is large deducing another fact from it may be incredibly taxing and error prone too.

Logic provides us a formal machinery to concisely represent the aforementioned assertions (statements) and efficiently reason about them. For example we can write the statement "If you study well in BE and get good marks then you will get a good job" symbolically as $(SW \wedge GM) \Rightarrow GJ$ and the statement "Once you have a good well paying job, your life will be settled and happy"as $GJ \Rightarrow LH$ where the symbols SW, GM, GJ and LH have the usual meanings. From the set of assertions $\{(SW \wedge GM) \Rightarrow GJ, GJ \Rightarrow LH\}$ we can deduce the fact $SW \Rightarrow LH$ which means "if you study well in BE then your life will eventually be happy".

1 Syntax and Semantics

The syntax of propositional logic (PL) is defined over a countable set of proposition symbols $P = \{p_1, p_2, p_3, \cdots\}$ and propositional connectives $\{\neg, \vee, \wedge, \Rightarrow, \Leftrightarrow\}$. The set of all **well-formed formulas** (**wff**s) of propositional logic are defined inductively as the smallest set satisfying the following conditions:

- Every $p_i \in P$ is a wff, (such wffs are called **atomic formulas**)
- If α is a wff then $(\neg\alpha)$ is a wff,
- If α, β are wffs then so are $(\alpha \vee \beta), (\alpha \wedge \beta), (\alpha \Rightarrow \beta)$ and $(\alpha \Leftrightarrow \beta)$ and

- Nothing else is a wff.

Let Φ be the set of all wffs constructed from the set of propositional symbols P and the connectives $\{\neg, \vee, \wedge, \Rightarrow, \Leftrightarrow\}$ using the above rules. We use wff and formula interchangeably. Φ can alternatively be defined via Backus-Naur Form as follows:

$$\alpha, \beta \in \Phi ::= p_i \in P \mid (\neg\alpha) \mid (\alpha \vee \beta) \mid (\alpha \wedge \beta) \mid (\alpha \Rightarrow \beta) \mid (\alpha \Leftrightarrow \beta).$$

Note, that, where ever there is no confusion we write a propositional formula without the parentheses. So $\neg\alpha$ is the same as $(\neg\alpha)$, $(\alpha \vee \beta)$ is the same as $\alpha \vee \beta$ and so on. Furthermore, we may drop the subscript i from p_i where ever it is not needed and use p, q, r *etc.* to denote arbitrary proposition symbols.

A formula β which occurs in another formula α is called the **subformula** of α. (Can you define subformula as a relation over Φ?) Given α, we can inductively construct the set of all subformulas of α (Can you give the inductive definition?).

Exercise 1.1 *Let* $\alpha = \big((\neg((p_4 \wedge p_3) \Rightarrow p_1)) \wedge (\neg p_5)\big)$. *Find the set of all subformulas of* α.

We assume that wffs of PL can either be true or false but never both. This is quite a restrictive assumption with respect to our real experiences but simplifies the logical reasoning considerably.

The semantics of propositional logic is defined via maps defined over P called **valuations** $\nu : P \to \{T, F\}$. Every symbol in P gets exactly one of the truth values and ν can be extended inductively to the set of all wffs as follows:

- $$\nu(\neg\beta) = \begin{cases} T & \nu(\beta) = F \\ F & \nu(\beta) = T \end{cases}$$
- $$\nu(\alpha \vee \beta) = \begin{cases} T & \text{when } \nu(\alpha) = T \text{ or } \nu(\beta) = T \\ F & otherwise \end{cases}$$
- $$\nu(\alpha \wedge \beta) = \begin{cases} T & \text{when } \nu(\alpha) = T \text{ and } \nu(\beta) = T \\ F & otherwise \end{cases}$$
- $$\nu(\alpha \Rightarrow \beta) = \begin{cases} F & \text{when } \nu(\alpha) = T \text{ and } \nu(\beta) = F \\ T & otherwise \end{cases}$$

- $$\nu(\alpha \Leftrightarrow \beta) = \begin{cases} T & \text{when } \nu(\alpha) = \nu(\beta) \\ F & otherwise \end{cases}$$

We denote $\nu(\alpha) = T$ by $\nu \models \alpha$ (ν models α) and $\nu(\alpha) = F$ by $\nu \not\models \alpha$ (ν does not model α). $\models$ is actually a relation (**satisfiability relation**) defined over the set of all valuations over P and Φ.

Given a formula $\alpha \in \Phi$, we say that α is **satisfiable** if there exists a valuation $\nu : P \to \{T, F\}$ such that $\nu \models \alpha$. Similarly, we say that α is **unsatisfiable** if there **does not** exist a valuation $\nu : P \to \{T, F\}$ such that $\nu \models \alpha$. We say that α is **valid** if for every possible valuation $\nu : P \to \{T, F\}$, $\nu \models \alpha$.

A valid formula may also be called a **tautology** and an unsatisfiable formula a **contradiction**. For an arbitrary p, $(p \vee \neg p)$ is the simplest example of a valid tautology, whereas $(p \wedge \neg p)$ is the simplest example of a contradiction. They are, respectively, denoted by $\top$ and $\bot$.
Now, we define the satisfiability (validity) problem in propositional logic.

Definition 1.2 (Satisfiability Problem (PSAT)) *Given a wff α in Propositional Logic, does there exist a valuation ν such that $\nu \models \alpha$?*

In general, this problem can be solved as follows: Let P_α be the set of all propositional symbols (atomic formulas) occurring in α. Suppose $P_\alpha = \{p_1, p_2, \cdots, p_n\}$. Generate all possible valuations ν defined over P_α. There can be 2^n such valuations starting from $\nu_0(p_1) = \nu_0(p_2) = \cdots = \nu_0(p_n) = F$ to $\nu_{2^n-1}(p_1) = \nu_{2^n-1}(p_2) = \cdots = \nu_{2^n-1}(p_n) = T$. For each such valuation ν_j check whether $\nu_j \models \alpha$. If any ν_j models α then α is satisfiable otherwise it is unsatisfiable. Note, that, in the worst case we may need to consider all 2^n valuations to check satisfiability of α. That is why the satisfiability problem has an exponential time complexity in the worst case.

Exercise 1.3 *A formula α is a tautology if and only if $\neg\alpha$ is unsatisfiable.*

We can extend the satisfiability relation to set of formulas as follows. Let $\Delta = \{\alpha_1, \alpha_2, \cdots, \alpha_n\}$ be a set of propositional formulas. A valuation $\nu : P \to \{T, F\}$ models Δ (written as $\nu \models \Delta$) if $\nu \models \alpha_1$ and $\nu \models \alpha_2$ and $\cdots$ and $\nu \models \alpha_n$. Equivalently, $\nu \models \Delta$ iff $\nu \models (\alpha_1 \wedge \alpha_2 \wedge \cdots \wedge \alpha_n)$.

Definition 1.4 (Semantic Entailment) *Given $\Delta = \{\alpha_1, \alpha_2, \cdots, \alpha_n\}$ and another formula α, we say that α is a consequence of Δ (or Δ semantically entails α denoted by $\Delta \models \alpha$) if for every valuation ν, if $\nu \models \Delta$ then $\nu \models \alpha$.*

2 Equivalences and Normal Forms

- $\Rightarrow$ Equivalence:
$$(\alpha \Rightarrow \beta) \equiv (\neg\alpha \vee \beta)$$
- $\Leftrightarrow$ Equivalence:
$$(\alpha \Leftrightarrow \beta) \equiv (\neg\alpha \vee \beta) \wedge (\neg\beta \vee \alpha)$$
- **Idempotence**:
$$(\alpha \wedge \alpha) \equiv \alpha$$
$$(\alpha \vee \alpha) \equiv \alpha$$
- **Commutativity**:
$$(\alpha \wedge \beta) \equiv (\beta \wedge \alpha)$$
$$(\alpha \vee \beta) \equiv (\beta \vee \alpha)$$
- **Associativity**:
$$((\alpha \wedge \beta) \wedge \gamma) \equiv (\alpha \wedge (\beta \wedge \gamma))$$
$$((\alpha \vee \beta) \vee \gamma) \equiv (\alpha \vee (\beta \vee \gamma))$$
- **Absorption**:
$$(\alpha \vee (\alpha \wedge \beta) \equiv \alpha$$
$$(\alpha \wedge (\alpha \vee \beta) \equiv \alpha$$
- **Double Negation**:
$$(\neg(\neg\alpha)) \equiv \alpha$$
- **De Morgan's Law**:
$$\neg(\alpha \wedge \beta) \equiv \neg\alpha \vee \neg\beta$$
$$\neg(\alpha \vee \beta) \equiv \neg\alpha \wedge \neg\beta$$
- **Distributivity**:
$$((\alpha \wedge \beta) \vee \gamma) \equiv (\alpha \vee \gamma) \wedge (\beta \vee \gamma)$$
$$((\alpha \vee \beta) \wedge \gamma) \equiv (\alpha \wedge \gamma) \vee (\beta \wedge \gamma)$$

- **Tautology Laws**:

$$\top \wedge \alpha \equiv \alpha$$

$$\top \vee \alpha \equiv \top$$

$$\neg \top \equiv \bot$$

- **Unsatisfiability Laws**:

$$\bot \wedge \alpha \equiv \bot$$

$$\bot \vee \alpha \equiv \alpha$$

$$\neg \bot \equiv \top$$

Every wff in PL can be transformed, using the equivalences enumerated above, to an equivalent wff with a certain normal form. One of such normal forms is **negation normal form** (NNF).

Definition 2.1 *A formula α is in* **negation normal form** *if*

- *the negation is asserted only at the atomic formulas and*
- *contains only the following connectives* $\{\neg, \vee, \wedge\}$.

The following formulas are not in NNF.

- $\neg(\neg p)$, ($\neg$ is applied on $\neg p$ which not an atomic formula, but $\neg p$ is in NNF).
- $\neg(p \vee q)$, ($\neg$ is applied on $(p \vee q)$ which not an atomic formula, but $(\neg p \wedge \neg q)$ is in NNF, obtained using De Morgan's laws).
- $(p \Leftrightarrow (q \Rightarrow r))$, (not in NNF because of the presence of connectives $\Leftrightarrow$ and $\Rightarrow$).

$(p \Leftrightarrow (q \Rightarrow r))$ can be converted to its equivalent NNF formula using the various equivalences given above as follows:

- $(\neg p \vee (q \Rightarrow r)) \wedge (\neg(q \Rightarrow r) \vee p)$, Using $\Leftrightarrow$-equivalence
- $(\neg p \vee (\neg q \vee r)) \wedge (\neg(\neg q \vee r) \vee p)$, Using $\Rightarrow$-equivalence
- $(\neg p \vee (\neg q \vee r)) \wedge ((\neg\neg q \wedge \neg r) \vee p)$, Using De Morgan's law
- $(\neg p \vee (\neg q \vee r)) \wedge ((q \wedge \neg r) \vee p)$, Using Double negation.

Clearly, $(p \Leftrightarrow (q \Rightarrow r)) \equiv (\neg p \vee (\neg q \vee r)) \wedge ((q \wedge \neg r) \vee p)$ which is in NNF. Apart from NNF we consider other normal forms viz. **conjunctive normal form** and **disjunctive normal form**. First we define a **literal**.

Definition 2.2 (Literal) *A literal is an atomic formula or the negation of an atomic formula. For every $p \in P$, p and $\neg p$ are literals, the first one* **positive** *whereas the second one* **negative**.

Definition 2.3 (Conjunctive Normal Form) *A formula α is in conjunctive normal form (CNF) if it is a conjunction of disjunction of literals, e.g.,*

$$\alpha = \big(\bigwedge_{i=1}^{n}(\bigvee_{j=1}^{m_i} L_{i,j})\big), \quad L_{i,j} \in \{p_1, p_2, p_3 \cdots\} \cup \{\neg p_1, \neg p_2, \neg p_3, \cdots\}$$

$$\textit{where} \quad (\bigvee_{j=1}^{m_i} L_{i,j}) = L_{i,1} \vee L_{i,2} \vee \cdots \vee L_{i,m_i}$$

$$\textit{and} \quad \big(\bigwedge_{i=1}^{n}(\bigvee_{j=1}^{m_i} L_{i,j})\big) = (\bigvee_{j=1}^{m_1} L_{1,j}) \wedge (\bigvee_{j=1}^{m_2} L_{2,j}) \wedge \cdots \wedge (\bigvee_{j=1}^{m_n} L_{n,j})$$

Definition 2.4 (Disjunctive Normal Form) *A formula α is in disjunctive normal form (DNF) if it is a disjunction of conjunction of literals, e.g.,*

$$\alpha = \big(\bigvee_{i=1}^{n}(\bigwedge_{j=1}^{m_i} L_{i,j})\big), \quad L_{i,j} \in \{p_1, p_2, p_3 \cdots\} \cup \{\neg p_1, \neg p_2, \neg p_3, \cdots\}$$

$$\textit{where} \quad (\bigwedge_{j=1}^{m_i} L_{i,j}) = L_{i,1} \wedge L_{i,2} \wedge \cdots \wedge L_{i,m_i}$$

$$\textit{and} \quad \big(\bigvee_{i=1}^{n}(\bigwedge_{j=1}^{m_i} L_{i,j})\big) = (\bigwedge_{j=1}^{m_1} L_{1,j}) \vee (\bigwedge_{j=1}^{m_2} L_{2,j}) \vee \cdots \vee (\bigwedge_{j=1}^{m_n} L_{n,j})$$

Consider the formula $(p \Leftrightarrow (q \Rightarrow r))$ which we converted to NNF as

$$(\neg p \vee (\neg q \vee r)) \wedge ((q \wedge \neg r) \vee p).$$

This formula is not in CNF or DNF. We can convert to CNF using distribution of $\vee$ over $\wedge$ as follows:

$$(\neg p \vee (\neg q \vee r)) \wedge ((q \vee p) \wedge (\neg r \vee p))$$

Removing the redundant parentheses we get

$$(\neg p \vee \neg q \vee r) \wedge (q \vee p) \wedge (\neg r \vee p).$$

This formula contains three conjunctive subformulas (called **clauses**) $(\neg p \vee \neg q \vee r)$, $(q \vee p)$ and $(\neg r \vee p)$, each of which is a disjunction of literals. This formula can alternatively be written as a set of clauses

$$\{(\neg p \vee \neg q \vee r), (q \vee p), (\neg r \vee p)\}$$

where the commas are hidden $\wedge$'s. Each clause can, in turn, be written as a set of literals where the commas are hidden $\vee$'s.

$$(\neg p \vee \neg q \vee r) = \{\neg p, \neg q, r)\}$$

$$(q \vee p) = \{q, p\}$$

$$(\neg r \vee p) = \{\neg r, p\}$$

Therefore, the formula $(p \Leftrightarrow (q \Rightarrow r))$ can be represented in CNF as

$$\{\{\neg p, \neg q, r)\}, \{q, p\}, \{\neg r, p\}\}.$$

Exercise 2.5 *Convert $(p \Leftrightarrow (q \Rightarrow r))$ to its equivalent DNF.*

In general, a formula $\alpha = (\bigwedge_{i=1}^{n}(\bigvee_{j=1}^{m_i} L_{i,j}))$ in CNF can also be represented as

$$\{\{L_{1,1}, L_{1,2}, \cdots, L_{1,m_1}\}, \{L_{2,1}, L_{2,2}, \cdots, L_{2,m_2}\}, \cdots, \{L_{n,1}, L_{n,2}, \cdots, L_{n,m_n}\}\}$$

That is, the formula α has n clauses where for each i, $1 \leq i \leq n$, the ith clause has m_i literals.

Definition 2.6 (k-CNF) *A CNF formula α is in k-CNF, $k > 0$, if every clause in α has exactly k literals.*

2.1 Implication Normal Form

A 2-CNF formula α can be transformed to an equivalent formula in **implication normal form** by replacing each clause $C = \{L, L'\}$ in α with two implications $(\neg L \Rightarrow L')$ and $(\neg L' \Rightarrow L)$.

Let $\alpha = (\neg x \vee y) \wedge (y \vee z) \wedge (x \vee \neg z)$ be a formula in 2-CNF. It can equivalently be written as a set of set of literals $\alpha = \{\{\neg x, y\}, \{y, z\}, \{x, \neg z\}\}$. The INF of α is

$$(x \Rightarrow y) \wedge (\neg y \Rightarrow \neg x) \wedge (\neg y \Rightarrow z) \wedge (\neg z \Rightarrow y) \wedge (\neg x \Rightarrow \neg z) \wedge (z \Rightarrow x).$$

3 2-CNF Satisfiability

A CNF formula α is a 2-CNF formula if every clause in α contains exactly two literals. These are alternately known as Krom formulas after Melven R. Krom. Here are a few examples:

$$\beta_1 = (p_0 \vee p_2) \wedge (p_0 \vee \neg p_3) \wedge (p_1 \vee p_3) \wedge (\neg p_1 \vee \neg p_0)$$

$$\beta_2 = (\neg p_0 \vee p_1) \wedge (\neg p_1 \vee p_2) \wedge (p_0 \vee \neg p_2) \wedge (p_2 \vee p_1)$$

$$\beta_3 = (\neg p_1 \vee p_2) \wedge (p_1 \vee p_2) \wedge (p_3 \vee \neg p_2) \wedge (\neg p_3 \vee \neg p_2)$$

The linear 2-CNF satisfiability is as follows:
Input 2-CNF formula $\alpha = \{C_1, C_2, \cdots, C_n\}$

- Let P_α be the set of all atomic propositions occuring in α.
- Construct the implication graph of α, $IG_\alpha = (V, E)$ as follows:
  $V = \{p, \neg p \mid p \in P_\alpha\}$, $E = \{(\neg l, l'), (\neg l', l) \mid (l, l') \in \alpha\}$
- Find the set of all connected components of $IG_\alpha = \{CC_1, CC_2, \cdots CC_k\}$.
- If there exists a connected component CC such that there exists a $p \in P_\alpha$ and $p, \neg p \in CC$ then α is unsatisfiable.
- Otherwise α is satisfiable and an appropriate model $v : P_\alpha \to \{T, F\}$ is obtained as follows:
  - Construct the condensed graph $CG_\alpha = (V', E')$ by merging the nodes in the same connected component, that is

    $$V' = \{CC_1, CC_2, \cdots CC_k\}$$

    and

    $$E' = \{(CC, CC') \mid \exists p \in CC, p' \in CC', (p, p') \in E\}.$$

  - Topologically sort CG_α
  - Greedily assign F to the next CC_i in the topologically sorted sequence of CG_α. Assign T only when forced.

Exercise 3.1 *Using the above algorithm show that β_1 and β_2 are satisfiable whereas β_3 is not.*

4 Horn Formulas

A CNF formula α is a **Horn formula** if every clause in α contains at most one positive literal. For example consider the following two formulas:

$$\beta_1 = (p \vee \neg q) \wedge (\neg r \vee \neg p \vee s) \wedge (r) \wedge (\neg p \vee \neg q)$$

$$\beta_2 = (p \vee \neg q) \wedge (q \vee \neg p \vee s)$$

which can equivalently be written as

$$\beta_1 = \{\{p, \neg q\}, \{(\neg r, \neg p, s\}, \{r\} \wedge \{\neg p, \neg q\}\}$$

$$\beta_2 = \{\{p, \neg q)\}, \{q, \neg p, s\}\}$$

β_1 is a Horn formula as every clause has at most one positive literal, whereas β_2 is not a Horn formula as the second clause $(q \vee \neg p \vee s)$ has two positive literals q and s.

Horn formulas can be equivalently rewritten as implications. For example, $\beta_1 = (p \vee \neg q) \wedge (\neg r \vee \neg p \vee s) \wedge (r) \wedge (\neg p \vee \neg q)$ can be rewritten as

$$(q \Rightarrow p) \wedge (r \wedge p \Rightarrow s) \wedge (\top \Rightarrow r) \wedge (p \wedge q \Rightarrow \bot).$$

Note, r is equivalent to $\bot \vee r$ which is, in turn, equivalent to $\neg\top \vee r$, which is equivalent to $\top \Rightarrow r$. Similarly, the clause $(\neg p \vee \neg q)$ can be equivalently transformed to an implication using unsatisfiability laws as follows:

$$(\neg p \vee \neg q) \equiv (\neg p \vee \neg q \vee \bot) \equiv (p \wedge q \Rightarrow \bot)$$

The clauses in Horn formulas can be distinguished to be of three types:

(FACT) A clause with a single positive literal, for example (r) in β_1.

(RULE) A clause with one or more negative literals along with one positive literal, for example $(\neg r \vee \neg p \vee s)$ in β_1.

(GOAL) A clause with only negative literals, for example $(\neg p \vee \neg q)$ in β_1.

4.1 Satisfiability Algorithm for Horn formulas

In order to check the satisfiability of a given Horn formula α we use an integer array *mark* which has an entry for every atomic formula p occurring in α.

Input: A Horn formula α written as set of implications

1. Initialize $mark$ array to 0 for every atomic formula p in P_α;
2. For every clause $\top \Rightarrow p$ in α, assign $mark[p] = 1$;
3. **while** (there is a clause C in α of the form $p_1 \wedge \cdots \wedge p_k \Rightarrow q$ or $p_1 \wedge \cdots \wedge p_k \Rightarrow \bot$, $k \geq 1$, $mark[p_1] = \cdots = mark[p_k] = 1$, $mark[q] = 0$) **do**
   - 3.1 **if** C is a fact **then** $mark[q] = 1$;
   - 3.2 **else**(C is a goal) output α "unsatisfiable"; HALT;
4. output α "satisfiable"; HALT;

The valuation ν which satisfies the formula α can be extracted from $mark$ array as follows: For any $p \in P_\alpha$

$$\nu(p) = \begin{cases} T & mark(p) = 1 \\ F & mark(p) = 0 \end{cases}$$

Exercise 4.1 *Apply the above marking algorithm on the following Horn formula and check whether it is satisfiable.*

$$\{(q \Rightarrow p), (r \wedge p \Rightarrow s), (\top \Rightarrow r), (p \wedge q \Rightarrow \bot)\}.$$

Algorithm for Sematic Entailment

Let $\Delta = \{\alpha_1, \alpha_2, \cdots, \alpha_n\}$ be a set of wffs and α be another wff. How do we check whether Δ entails α? As already discussed, $\Delta \models \alpha$ if and only if $\delta = \alpha_1 \wedge \alpha_2 \wedge \cdots \wedge \alpha_n \wedge \neg\alpha$ is unsatisfiable. Therefore, semantic entailment can be checked instead be checking whether a wff is unsatisfiable, using any of the techniques which we already know. This may be explained using an example.

Let $\Delta = \{\neg s \wedge c, w \Rightarrow s, \neg w \Rightarrow t, t \Rightarrow h\}$ and $\alpha = h$. We would like to know whether $\Delta \models h$? In order to check this, we contruct another wff $\delta = \neg s \wedge c \wedge w \Rightarrow s \wedge \neg w \Rightarrow t \wedge t \Rightarrow h \wedge \neg h$. If δ is unsatisfiable then we can assert that $\Delta \models h$ else (if δ is satisfiable) $\Delta \not\models h$.

It turns out that δ is a 2-CNF formula. So, we can check the unsatisfiability/satisfiablity of δ by constructing the corresponding implication graph and looking into its connected components.

The implication graph in Figure 4.1 (corresponding to the formula $\delta = \neg s \wedge c \wedge w \Rightarrow s \wedge \neg w \Rightarrow t \wedge t \Rightarrow h \wedge \neg h$), has paths between t and $\neg t$ as well as between h and $\neg h$. This means that δ is unsatisfiable. Which, in turn, means that $\Delta \models h$.

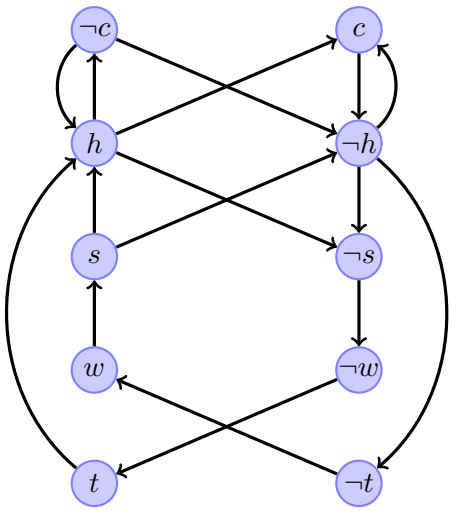

Figure 1: Implication Graph for $\delta = \neg s \wedge c \wedge w \Rightarrow s \wedge \neg w \Rightarrow t \wedge t \Rightarrow h \wedge \neg h$

Exercise 4.2 *For each of the the following cases check whether* $\Delta \models \alpha$.

1. $\Delta = \{p \vee q, p \Rightarrow r, q \Rightarrow r\}$, $\alpha = r$.
2. $\Delta = \{(p \Rightarrow q) \Rightarrow q, (p \Rightarrow p) \Rightarrow r, (r \Rightarrow s) \Rightarrow \neg(s \Rightarrow q)\}$, $\alpha = r$.
3. $\Delta = \{p \Rightarrow q, \neg q \Rightarrow r\}$, $\alpha = \neg q \Rightarrow \neg r$.
4. $\Delta = \{(p \Rightarrow q) \Rightarrow (r \Rightarrow s)\}$, $\alpha = (p \Rightarrow s) \Rightarrow (r \Rightarrow q)$.
5. $\Delta = \{(q \vee r) \Rightarrow s, p \Rightarrow q, p\}$, $\alpha = s$.
6. $\Delta = \{q \Rightarrow (r \vee s), p \Rightarrow q, p, \neg r\}$, $\alpha = s$.
7. $\Delta = \{(q \wedge r) \Rightarrow s, p \Rightarrow q, p, p \Rightarrow r\}$, $\alpha = s$

5 Deduction in Propositional Logic

There are broadly two ways to describe meanings of assertions in Propositional Logic. One is through models, which we have already seen. That is, given a wff α is it true for some model (satisfiable), all models (valid), no models (contradiction)? These models may correspond to the states of the world for which the assertion α has been made. This view has profound applications, in particular verification of circuits and programs.

Another view is via deduction, deriving a (possibly unknown assertion) α from known (and valid) assertions (Δ). This is denoted by $\Delta \vdash \alpha$. It can be shown that if Δ contains only valid wffs (tautologies) then α is also valid (a tautology).

Furthermore, it has been shown that the two schemes of deduction and entailment coincide.

Theorem 5.1 (Completeness Theorem) *Let Δ be a set of wffs and α be an arbitrary wff. Then, $\Delta \models \alpha$ if and only if $\Delta \vdash \alpha$.*

Deduction should be familiar to the reader from the study of plane geometry in high school, where we learn to start with certain **hypotheses** (the "givens"), and to prove a **conclusion** by a sequence of steps, each of which follows from previous steps by one of a limited number of reasons, called **inference rules**.

Unfortunately, discovering a **deductive proof** for a tautology is difficult. It is an example of an "inherently intractable" problem, in the NP-hard class. Thus we cannot expect to find deductive proofs except by luck or by exhaustive search. In the next section, we shall discuss resolution proofs, which appear to be a good heuristic for finding proofs, although in the worst case this technique, like all others, must take exponential time.

5.1 Deductive Proof

Suppose we are given certain wffs $\alpha_1, \alpha_2, \cdots, \alpha_k$ as hypotheses, and we wish to draw a conclusion in the form of another wff α. In general, neither the conclusion nor any of the hypotheses will be tautologies, but what we want to show is that $\alpha_1 \wedge \alpha_2 \wedge \cdots \wedge \alpha_k \Rightarrow \alpha$ is a tautology. That is, we want to show that if we assume $\alpha_1, \alpha_2, \cdots, \alpha_k$ are true, then it follows that α is true.

One way to show it is to construct its truth table and test whether it has value 1 in each row–the routine test for a tautology. However, that may not be sufficient for two reasons.

- Tautology testing becomes infeasible if there are too many variables in the formula.
- More importantly, while tautology testing works for propositional logic, it cannot be used to test tautologies in more complex logical systems, such as predicate logic.

We can often show that $(\alpha_1 \wedge \alpha_2 \wedge \cdots \wedge \alpha_k) \Rightarrow \alpha$ is a tautology by presenting a deductive proof. A deductive proof is a sequence of lines, each of which is

either a given hypothesis or is constructed from one or more previous lines by a rule of inference. If the last wff is α, then we say that we have **a proof of** α **from** $\alpha_1, \alpha_2, \cdots, \alpha_k$.

There can be as many rules of inference as we might use. The only requirement is that if an inference rule allows us to write expression β as a line whenever expressions $\beta_1, \beta_2, \cdots, \beta_n$ are lines, then $(\beta_1 \wedge \beta_2 \wedge \cdots \wedge \beta_n) \Rightarrow \beta$ must be a tautology.

Rules of Inference

$$\frac{\alpha \qquad (\alpha \Rightarrow \beta)}{\beta} \text{ Modus Ponens}$$

$$\frac{\neg\beta \qquad (\alpha \Rightarrow \beta)}{\neg\alpha} \text{ Modus Tolens}$$

$$\frac{(\alpha \Rightarrow \beta) \qquad (\beta \Rightarrow \gamma)}{(\alpha \Rightarrow \gamma)} \text{ Hypothetic Syllogism}$$

$$\frac{\alpha}{(\alpha \vee \beta)} \text{ Addition}$$

$$\frac{(\alpha \wedge \beta)}{\alpha} \text{ Simplification}$$

$$\frac{\alpha \quad \beta}{(\alpha \wedge \beta)} \text{ Conjunction}$$

$$\frac{(\alpha \vee \gamma) \quad (\beta \vee \neg\gamma)}{(\alpha \vee \beta)} \text{ Resolution}$$

It turns out that if we use hypotheses and conclusion as clauses (CNF), we can use resolution as the **only** inference rule to build proofs. This is the topic of the next section.

Exercise 5.2 *For each of the the following cases check whether* $\Delta \vdash \alpha$.

1. $\Delta = \{p \vee q, p \Rightarrow r, q \Rightarrow r\}$, $\alpha = r$.
2. $\Delta = \{(p \Rightarrow q) \Rightarrow q, (p \Rightarrow p) \Rightarrow r, (r \Rightarrow s) \Rightarrow \neg(s \Rightarrow q)\}$, $\alpha = r$.
3. $\Delta = \{p \Rightarrow q, \neg q \Rightarrow r\}$, $\alpha = \neg q \Rightarrow \neg r$.
4. $\Delta = \{(p \Rightarrow q) \Rightarrow (r \Rightarrow s)\}$, $\alpha = (p \Rightarrow s) \Rightarrow (r \Rightarrow q)$.
5. $\Delta = \{(q \vee r) \Rightarrow s, p \Rightarrow q, p\}$, $\alpha = s$.
6. $\Delta = \{q \Rightarrow (r \vee s), p \Rightarrow q, p, \neg r\}$, $\alpha = s$.
7. $\Delta = \{(q \wedge r) \Rightarrow s, p \Rightarrow q, p, p \Rightarrow r\}$, $\alpha = s$

6 Resolution

Resolution (along with Resolution refutation) is a technique to solve the problem of semantic entailment. Given a set of wffs Δ and another wff α we can check whether Δ entails α (denoted by $\Delta \models \alpha$) using the syntactic transformation scheme of resolution.

Resolution is applied on sets of disjunctive clauses. A single application of resolution on a pair of (disjunctive) clauses C_1, C_2 computes another disjunctive clause $R = (C_1 - \{L\}) \cup (C_2 - \{\bar{L}\})$ where $L \in C_1, \bar{L} \in C_2$ and

$$\bar{L} = \begin{cases} \neg p & L = p \\ p & L = \neg p. \end{cases}$$

R is called the **resolvent** and L is the literal **resolved upon**. If $C_1 = \{L\}$ and $C_2 = \{\bar{L}\}$, then the resolvent $R = \emptyset$, is the empty clause, denoted by $\Box$.

Definition 6.1 *Let S be a set of clauses. Then $Res(S)$ can be defined as*

$$Res(S) = S \cup \{R \mid R \text{ is a resolvent of any two clauses } C_1, C_2 \in S\}.$$

Furthermore, define

$$Res^0(S) = S$$

$$Res^{k+1}(S) = Res(Res^k(S)) \text{ for } k \geq 0,$$

and finally, let

$$Res^*(S) = \bigcup_{k \geq 0} Res^k(S).$$

Resolution Algorithm to check $\Delta \models \alpha$

Input: Set of wffs Δ, wff α

1. Convert α to clausal form;
2. Convert all the wffs in Δ to clausal form and put them together in S;
3. **repeat**
   - 3.1 $S' = S$;
   - 3.2 $S = Res(S)$;
4. **until**($\alpha \in S$ or $S' = S$);
5. If $\alpha \in S$ then $\Delta \models \alpha$ else $\Delta \not\models \alpha$;

Exercise 6.2 *Use resolution algorithm to check whether*

$$\{s \Rightarrow (p \vee q), (q \Rightarrow r), \neg p, (p \vee q \vee r \vee s)\} \models r.$$

Essentially, the above resolution algorithm computes $Res^*(S)$, where S is the set of clauses corresponding to Δ. It turns out that resolution is sound but not complete. Even though $\{(p \wedge q)\}$ entails $(p \vee q)$, $(p \vee q) \notin Res^*(\{p, q\})$. We can modify the above algorithm to make it complete as follows:

Resolution Refutation Algorithm to check $\Delta \models \alpha$

Input: Set of wffs Δ, wff α

1. Convert $\neg\alpha$ to clausal form;
2. Convert all the wffs in Δ to clausal form and put them together along with $\neg\alpha$ in S;
3. **repeat**
   - 3.1 $S' = S$;
   - 3.2 $S = Res(S)$;
4. **until**($\Box \in S$ or $S' = S$);
5. If $\Box \in S$ then $\Delta \models \alpha$ else $\Delta \not\models \alpha$;

Exercise 6.3 *Use Resolution refutation algorithm to check whether*

$$\{(p \wedge q)\} \models (p \vee q)$$

Here are a few more exercises.

Exercise 6.4 *Use resolution or resolution refutation to show the following or otherwise.*

1. $\vdash (p \wedge q) \Rightarrow p$
2. $\{p\} \vdash q \Rightarrow (p \wedge q)$
3. $\{p\} \vdash (p \Rightarrow q) \Rightarrow q$
4. $\{(p \Rightarrow r) \wedge (q \Rightarrow r)\} \vdash (p \wedge q) \Rightarrow r$
5. $\{p \Rightarrow (q \Rightarrow r), p \Rightarrow q\} \vdash p \Rightarrow r$

6. $\{p \Rightarrow (q \wedge r)\} \models (p \Rightarrow q) \wedge (p \Rightarrow r)$
7. $\{r, p \Rightarrow (r \Rightarrow q)\} \vdash p \Rightarrow (q \wedge r)$
8. $\{p \Rightarrow (q \vee r), \neg q, \neg r\} \vdash \neg p$
9. $\{p \Rightarrow q, s \Rightarrow t, p \vee s\} \vdash q \wedge t$
10. $\{(c \wedge n) \Rightarrow t, h \wedge \neg s, h \wedge \neg(s \vee c) \Rightarrow p\} \vdash (n \wedge \neg t) \Rightarrow p$

7 Forward and Backward Chaining in Propositional Logic

We have already discussed Horn formulas—-CNF formulas which contain **at most one positive literal** in each clause. We have also seen that Horn clauses can be of three types: facts (clauses of the type $\top \Rightarrow p$), goals (clauses of the kind $(p_1 \wedge p_2 \wedge \cdots) \Rightarrow \bot$) and rules (clause of the kind $(p_1 \wedge p_2 \wedge \cdots) \Rightarrow p$). Horn clauses, like $(p_1 \wedge p_2 \wedge \cdots) \Rightarrow p$, (i.e., the rules) are also called **definite clauses**. The literal p on the right side of the implication is called the **head** of the clause and the literals on the left are called the **body** of the clause.

Inference with definite Horn clauses can be done through **forward chaining** and **backward chaining** algorithms in **linear** time.

We shall first look at forward chaining algorithm. Consider the knowledge base of Horn clauses

$$KB = \{P \Rightarrow Q, L \wedge M \Rightarrow P, B \wedge L \Rightarrow M, A \wedge P \Rightarrow L, A \wedge B \Rightarrow L, A, B\}.$$

The task is, can we infer Q from KB? That is, whether $KB \vdash Q$. Here, Q is called **query**. The best way to understand the forward chaining algorithm is by drawing an **AND-OR graph** for the knowledge base KB.

There are two types of nodes in the AND-OR graph, circles and squares. For each literal that occurs in the KB we have a circle in the graph and a square for each clause except those which have no body. Those clauses in the KB which have no body (the facts) are represented as leaves in the graph, with no incoming edges. The other clauses in the KB are represented by squares in the graph. There is an outgoing edge from a square to the literal at the head of the clause, whereas there are incoming edges into the square from every literal in the body of the clause.

Consider the clause $L \wedge M \Rightarrow P$. The square corresponding to this clause has incoming edges from L and M and an outgoing edge to P. This means

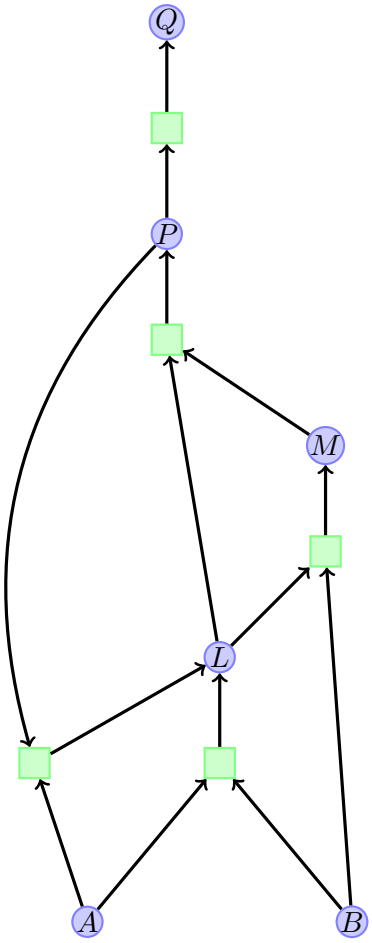

Figure 2: AND-OR Graph for the Knowledge Base

that, in order for P to be true it is necessary that both L and M are true. The leaves A and B are always true. The AND-OR graph of KB gives the truth dependencies among the literals occurring in KB as constrained by the clauses.

Given KB, we construct the AND-OR graph as follows. We use two mark arrays, one for the literals in KB and other for clauses.

- Initially, we consider facts in KB, for every fact (clause) $c = p$, we mark p as well as c and construct a leaf in the graph.
- Then, we look at each clause in turn. For every clause $c = (p_1 \wedge \cdots \wedge p_n) \Rightarrow p$, if $p_1, \cdots, p_n$ are all marked then mark p (if is not already marked) as well as c (which should not be already marked).
- Also, we construct a node for the clause c and a node p (if it is not already there) and appropriate edges, from p_i's to c and c to p.
- If we finish with all literals and clauses marked then AND-OR graph has been successfully constructed. We ignore those clauses in the KB which are not marked by the routine. Clearly, if there are no facts in KB then the AND-OR graph would be empty.
- In order to decide $KB \vdash^? Q$ it is sufficient to check whether Q is a node in the AND-OR graph of KB, which is true in the present case.

It is easy to see that forward chaining is **sound**: every inference is essentially an application of Modus Ponens. Forward chaining is also **complete**: every entailed atomic formula can be derived using forward chaining algorithm.

The backward-chaining algorithm works backward from the query Q. We don't construct the whole AND-OR graph as in the previous case, but only the relevant parts, which are needed to decide the truth of Q.

If Q is a fact in KB then $KB \vdash Q$ holds trivially.

Otherwise, we find implications in KB which have Q at their head. If all the premises of one of those implication can be proved true by backward chaining then $KB \vdash Q$ holds.

Let $c_1, c_2, \cdots, c_k$ be the clauses in KB which have Q as the consequent. Non deterministically choose a c from these set of clauses. Suppose $c = p_1 \wedge \cdots \wedge p_k \rightarrow p$. If $KB \vdash p_i$ for each $1 \leq i \leq k$, then $KB \vdash Q$.

Often, the cost of backward chaining is **much less** than linear in the size of KB, because it touches only relevant facts.

8 Propositional Logic and Knowledge-based Agents

Human beings **know** things and do **reasoning**. Knowledge and reasoning are also important for artificial agents because they enable successful behaviours that would be hard to achieve otherwise. We have already seen that knowledge of action outcomes enables problem-solving agents to perform well in complex environments. Knowledge-based agents can benefit from knowledge expressed in general forms, combining and recombining information for different purposes.

Knowledge and reasoning also plays important role in dealing with partially observable environments. A knowledge-based agent can combine general knowledge with current percepts to infer hidden aspects of the current state prior to selecting actions. For example, a physician diagnoses a patient by combining his knowledge learnt from books, his teachers and his own experiences with the symptoms he observes. He may know that if a patient has high fever as well as stiff neck then he has meningitis. Meningitis is not directly observable but the symptoms, high fever and stiff neck, are.

Knowledge based reasoning is also flexible. The agent can accept new tasks in the form of explicitly described goals, learn new knowledge about the environment and reason appropriately.

8.1 Knowledge based Agent

The central component of a knowledge-based agent is its **knowledge base** or **KB**. Informally, a knowledge base is a set of sentences. Each sentence is expressed in a language called a knowledge representation language and represents some assertion about the world. In the present case Propositional Logic is the language we use to represent knowledge of an agent.

There must be a way to add new sentences to the knowledge base and a way to query what is known. The standard names for these tasks are **TELL** and **ASK**, respectively. Both tasks may involve **inference**—-that is, deriving new sentences from old. In **logical agents**—-knowledge-based agent which use logic for representation of knowledge and reasoning about it—-inference must obey the fundamental requirement that when one **ASK**s a question of the knowledge base, the answer should follow from what has been **TELL**ed to the knowledge base previously.

Figure 3 shows the outline of a knowledge-based agent. Like all normal agents, it takes a percept as input and returns an action as output. The agent maintains a knowledge base, KB, which may initially contain some **background knowledge**.

```
function KB-AGENT (percept) returns an action
KB, a knowledge base
t, a counter, initially 0

TELL(KB, MAKE-PERCEPT-SENTENCE(percept,t))
action<-ASK(KB, MAKE-ACTION-QUERY(t))
TELL(KB, MAKE-ACTION-SENTENCE(action,t))

t<-t+1

return action
```

Figure 3: A generic knowledge-based agent

Each time the agent program is called, it does two things. First, it **TELL**s the knowledge base what it perceives. Second, it **ASK**s the knowledge base what action it should perform. Once the action is chosen, the agent records its choice with **TELL** and executes the action.

The details of the representation language (in this case PL) are hidden inside two functions that implement the interface between the sensors and the actuators and the core representation and reasoning system.

MAKE-PERCEPT-SEQUENCE takes a percept and a time and returns a sentence asserting that the agent perceived the percept at the given time.

MAKE-ACTION-QUERY takes time as input and returns a sentence that asks what action should be performed at that time.

The details of the inference mechanisms are hidden inside **TELL** and **ASK**.

8.2 The WUMPUS World

A simple example of knowledge based agent is the agent navigating the wumpus world, devised by Michael Genesereth. The **wumpus world** is a cave consisting of rooms connected by passageways. Lurking somewhere within the cave is the **wumpus**, a beast that eats anyone who enters the room. The wumpus can be **shot** by the agent, but the agent has *only one arrow*. Some rooms contain bottomless **pits** that will trap anyone who wanders into the room (except for wumpus, which is too big to fall in). The only mitigating feature of living in this environment is the possibility of finding a heap of **gold**.

S		B	PIT
W	B S G	PIT	B
S		B	
START	B	PIT	B

Figure 4: A Typical Wumpus World. The agent is at the bottom left corner

A sample world is shown in Figure 4. The precise description of the task environment is given by the following PEAS description:

- **performance measure:** $+1000$ for picking up the goal, -1000 for falling into a pit or being eaten by the wumpus, -1 for each action taken and -10 for using up the arrow.
- **Environment:** A 4×4 grid of rooms labelled by $[i, j]$, where $1 \leq i, j \leq 4$. The agent always starts in the square labelled $[1, 1]$, facing to the right. The locations of the gold and wumpus are chosen randomly, with a uniform distribution, from the squares other than the start square. Additionally, each square, other than the start can be pit, with probability 0.2.
- **Actuators:** The agent can move forward, turn left by 90 degrees, or turn right by 90 degrees. Moving forward has no effect if there is a wall in front of the agent. The agent dies a miserable death if it enters a square containing a pit or a live wumpus.

  The action *Grab* can be used to pick up an object that is in the same square as the agent. The action *Shoot* can be used to shoot an arrow in a straight line in the direction the agent is facing. The arrow continues until it hits the wumpus or hits a wall. The agent has only one arrow so only the first *Shoot* action has any effect.
- **Sensors:** The agent has five sensors, each of which gives a single bit of information:
  - In the square containing the wumpus and in the directly adjacent squares will perceive a *stench* (S).

- In the squares directly adjacent to a pit, the agent will perceive a *breeze* (B).
- In the square where the gold is, the agent will perceive a *glitter* (G) .
- If the agent walks into a wall, it will perceive a *bump*.
- When the wumpus is killed, it emits a *scream* which can be perceived anywhere in the cave.

The percepts will be given to the agent in the form of a list of five symbols; for example, if there is a stench and a breeze, but no glitter, bump, or scream, the agent will receive the percept $[Stench, Breeze, None, None, None]$.

The principal **difficulty** with the agent is its **initial ignorance of the environment configuration**. The only knowledge initially it has is about the **rules of the environment**, for eg., if there is a breeze in the given square then there is a pit in at least one of the adjacent square, and that it is in $[1,1]$ and $[1,1]$ is a safe square. Note that, this initial knowledge can be written as propositional logic formulas. The initial state of agent along with its safety can be framed in PL as follows:

$$In([1,1]) \wedge OK([1,1]).$$

A sample rule, for the square $[i,j]$ somewhere in the middle, can be written as follows:

$$Breeze([i,j]) \Leftrightarrow Pit([i-1,j]) \vee Pit([i,j-1]) \vee Pit([i+1,j]) \vee Pit([i,j+1]).$$

The corresponding rule for a square in the corner, say $[1,4]$, would be different:

$$Breeze([1,4]) \Leftrightarrow Pit([1,3]) \vee Pit([2,4]).$$

The state of agent's knowledge at the beginning is show in Figure 5. We do not mention the rules of the environment in the figure, they are implicit and static, at least for this example. A in square $[1,1]$ means $In([1,1])$ holds, whereas OK in the same square means $OK([1,1])$ is true.

The (knowledge-based) agent has to explore the environment, gather information (collect more such PL formulas in the knowledge base) about it and look for the gold while evading himself from pits and the wumpus.

The first percept is $[None, None, None, None, None]$, that is, the following PL formula holds:

$$\neg Stench([1,1]) \neg Breeze([1,1]) \wedge \neg Glitter([1,1]) \wedge \neg Bump([1,1]) \wedge \neg Scream([1,1]).$$

Figure 5: Knowledge of Wumpus world agent initially.

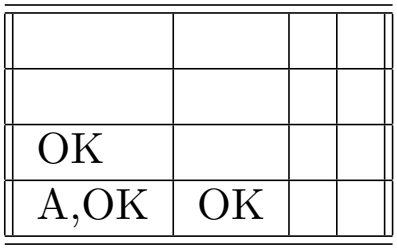

Figure 6: Knowledge of Wumpus world agent after the first percept

Using the information provided by the percept and rules of the environment we can infer (using anyone of the inference techniques of PL we have learnt) that the neighbouring squares are safe. Thus, we can add the following PL formula to the knowledge base of the agent:

$$OK([1,2]) \wedge OK([2,1]).$$

The state of agent's knowledge at this point can be shown as in Figure 6. A cautious agent will move only into a square that it knows is OK. Let us suppose that the agent decides to move to $[2,1]$. Consequently, the knowledge base is modified as follows: Remove $In([1,1])$ from and add $In([2,1]) \wedge Visited([1,1])$ to the knowledge base. The state of agent's knowledge after this change is shown in the Figure 7. The agent detects a breeze in $[2,1]$, i.e., $Breeze([2,1])$ holds, so there must be a pit in a neighbouring square, i.e, $Pit([1,1]) \vee Pit([2,2]) \vee Pit([3,1])$ must hold. But, we already know $[1,1]$ is safe, i.e. $\neg Pit([1,1])$, so there must be a pit either in $[2,2]$ or $[3,1]$, i.e, $Pit([2,2]) \vee Pit([3,1])$ holds. At this point, the agent does not have enough knowledge to decide (infer) with certainty where exactly the pit is, i.e., which of the PL formulas $Pit([2,2])$ or $Pit([3,1])$ holds. So the prudent agent turns back and goes to $[1,2]$ via $[1,1]$. The state of agent's knowledge after this

OK			
V,OK	A,OK		

Figure 7: Knowledge of Wumpus world agent after the first movement

A,OK			
V,OK	V,B,OK		

Figure 8: Knowledge of Wumpus world agent after the second movement

movement is shown in the Figure 9.

The new percept in $[1,2]$ is $[Stench, None, None, None, None]$, which means, $Stench([1,2])$ holds. Using the rules of the environment we can infer $Wumpus([1,1]) \vee Pit([2,2]) \vee Pit([1,3])$, i.e., there must be the dreaded wumpus in one of the neighbouring squares. But the wumpus can not be in $[1,1]$, i.e., $Wumpus([1,1])$ can not hold. Furthermore, $Wumpus([2,2])$ can not hold otherwise $Stench([2,1])$ would hold, which is not true. Therefore, $Wumpus([1,3])$ should hold, i.e., wumpus must be in the square $[1,3]$. Also, $Pit([2,2])$ can not hold otherwise $Breeze([1,2])$ would hold, which is not true. Thus, the square $[2,2]$ must be safe, i.e., $OK([2,2])$ holds. The state of knowledge after this percept and series of inferences is given in Figure 9. As $[2,2]$ is safe, the agent moves to this square.

In $[2,2]$ the percept is $[None, None, None, None, None]$, i.e., both $Stench([2,2])$ and $Breeze([2,2])$ do not hold. This means both the squares $[2,3]$ and $[3,2]$ are safe. The knowledge of the agent in $[2,2]$ is given in the Figure 10.

Suppose, the agent now moves to the square $[3,2]$. the percept in $[3,2]$ is $[Stench, Breeze, Glitter, None, None]$. The knowledge of the agent in square $[3,2]$ is given in the Figure 11. The agent detects a glitter, so it should grab the gold and the game ends.

W			
A,OK,S	OK		
V,OK	V,B,OK		

Figure 9: Knowledge of Wumpus world agent in $[1, 2]$

W	OK		
V,OK,S	A,OK	OK	
V,OK	V,B,OK		

Figure 10: Knowledge of Wumpus world agent in $[2, 2]$

W	A,B,S,G,OK		
V,OK,S	V,OK	OK	
V,OK	V,B,OK		

Figure 11: Knowledge of Wumpus world agent in $[3, 2]$

Thus, we have seen that we can describe the knowledge and reasoning of an agent in wumpus world using PL. The initial knowledge and the rules of the environment are represented as PL formulas. Then, the agent explores the environment, gets more knowledge from the percepts which he converts to PL formulas and adds to his knowledge base. He also infers non-observable facts about the environment using his knowledge base which he continually augments. Furthermore, using his knowledge he decides about which squares are safe and accordingly moves to such squares, i.e, takes action.