Laporan Uji Coba Praktikum - Logical Agents

Inference --> Forward and backward chaining

Name : Abdan Hafidz NRP : 5054231021

Importing Libraries

In [106…

```
from utils import *
from logic import *
from notebook import psource
import agents as a
import warnings
warnings.filterwarnings('ignore')
warnings.simplefilter(action='ignore', category=FutureWarning)
```

Inference in Propositional Knowledge Base

In [107…

```
(P,Q) = symbols('P,Q')
```

Truth Table Enumeration

In [ ]:

```
psource(tt_check_all)
```

In this test we will prove several tautologies. First, we will test them in the form of a truth table and check suitability using Entails.

In [ ]:

```
psource(tt_entails)
```

In the first example we will test whether P & Q => Q is a tautology

In [110…

```
print("P    Q    P & Q    P & Q => Q")
for R in [True, False]:
    for S in [True, False]:
        print(f"{R}    {S}    {R and S}    { R if S & S else True}")
```

```
P    Q    P & Q    P & Q => Q
True     True    True    True
True     False    False    True
False     True    False    False
False     False    False    True
```

It can be seen that by testing the truth table we get a tautology, now we prove it using entails

In [111…

```
tt_entails(P & Q, Q)
```

Out[111]:

```
True
```

It is found that it is true that P & Q => Q is a tautology

We will also test P | Q is a tautology of P or Q is one of them

In [112…

```
print("P            Q        P | Q    P | Q => P          P | Q => Q")
for R in [True, False]:
    for S in [True, False]:
        print(f"{R}     {R}    {R or S}      { R if R or S else True }                    { S if R or S else True
}")
```

```
P             Q        P | Q   P | Q => P          P | Q => Q
True      True    True      True                    True
True      True    True      True                    False
False      False    True      False                    True
False      False    False      True                    True
```

It was found that P | Q has no tautology with P or Q. Likewise testing using Entails.

In [113…

```
tt_entails(P | Q, P)
```

Out[113]:

```
False
```

In [114…

```
tt_entails(P | Q, Q)
```

Out[114]:

```
False
```

Proof By Resolution (Decomposite Of Proposition)

We can handle a verification for certains logic's law

TABLE 6 Logical Equivalences.

Equivalence	*Name*
$p \wedge \mathbf{T} \equiv p$ $p \vee \mathbf{F} \equiv p$	Identity laws
$p \vee \mathbf{T} \equiv \mathbf{T}$ $p \wedge \mathbf{F} \equiv \mathbf{F}$	Domination laws
$p \vee p \equiv p$ $p \wedge p \equiv p$	Idempotent laws
$\neg(\neg p) \equiv p$	Double negation law
$p \vee q \equiv q \vee p$ $p \wedge q \equiv q \wedge p$	Commutative laws
$(p \vee q) \vee r \equiv p \vee (q \vee r)$ $(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$	Associative laws
$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$ $p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$	Distributive laws
$\neg(p \wedge q) \equiv \neg p \vee \neg q$ $\neg(p \vee q) \equiv \neg p \wedge \neg q$	De Morgan's laws
$p \vee (p \wedge q) \equiv p$ $p \wedge (p \vee q) \equiv p$	Absorption laws
$p \vee \neg p \equiv \mathbf{T}$ $p \wedge \neg p \equiv \mathbf{F}$	Negation laws

TABLE 7 Logical Equivalences Involving Conditional Statements.

$p \to q \equiv \neg p \vee q$

$p \to q \equiv \neg q \to \neg p$

$p \vee q \equiv \neg p \to q$

$p \wedge q \equiv \neg(p \to \neg q)$

$\neg(p \to q) \equiv p \wedge \neg q$

$(p \to q) \wedge (p \to r) \equiv p \to (q \wedge r)$

$(p \to r) \wedge (q \to r) \equiv (p \vee q) \to r$

$(p \to q) \vee (p \to r) \equiv p \to (q \vee r)$

$(p \to r) \vee (q \to r) \equiv (p \wedge q) \to r$

TABLE 8 Logical Equivalences Involving Biconditional Statements.

$p \leftrightarrow q \equiv (p \to q) \wedge (q \to p)$

$p \leftrightarrow q \equiv \neg p \leftrightarrow \neg q$

$p \leftrightarrow q \equiv (p \wedge q) \vee (\neg p \wedge \neg q)$

$\neg(p \leftrightarrow q) \equiv p \leftrightarrow \neg q$

It should be remembered that an equivalence is definitely a tautology, but not necessarily vice versa.

So we can carry out revalidation with an additional step which may be optional with a tautology test using Entails, apart from proving resolution to_cnf

As an example we will prove whether it is really equivalent for P => Q with ~P | Q

In [115…
```
to_cnf(P |'==>'| Q)
```

Out[115]:
```
(Q | ~P)
```

Likewise with entails testing

In [116…
```
tt_entails(P |'==>'| Q, Q | ~P)
```

Out[116]:
```
True
```

It is found that this is an equivalence

Testing for De Morgan's laws

In [117…
```
to_cnf(~(P & Q))
```

Out[117]:
```
(~P | ~Q)
```

In [118…
```
to_cnf(~(P | Q))
```

Out[118]:
```
(~P & ~Q)
```

Next we will test the remaining logical equivalence from Table 7 above

In [119…
```
to_cnf(~Q |'==>'| ~P)
```

Out[119]:
```
(~P | Q)
```

Remember that ~P | Q is equivalent to P => Q

In [120...
```
to_cnf(~P |'==>' | Q)
```

Out[120]:
```
(Q | P)
```

In [121...
```
to_cnf(~ (P |'==>' | ~Q))
```

Out[121]:
```
(Q & P)
```

In [122...
```
to_cnf(~(P | '==>' | Q))
```

Out[122]:
```
(~Q & P)
```

In [123...
```
R = Symbol('R')
to_cnf(P | '==>' | (Q & R) )
```

Out[123]:
```
((Q | ~P) & (R | ~P))
```

In [124...
```
to_cnf(P | '==>' | (Q | R) )
```

Out[124]:
```
(Q | R | ~P)
```

In [125...
```
to_cnf( (P | Q) | '==>' | (R) )
```

Out[125]:
```
((~P | R) & (~Q | R))
```

In [126...
```
to_cnf( (P & Q) | '==>' | (R) )
```

Out[126]:
```
(R | ~P | ~Q)
```

It can be seen that all tests in table 7 are appropriate

Now we test for a Biimplication P <=> Q

In [127…

```
to_cnf( P  | '<=>' | Q )
```

Out[127]:

```
((P | ~Q) & (Q | ~P))
```

Propotitional Knowledge Base & Proof Resolution

We can add satisfying value for each value / statement that can conquered the evaluation result (True / False).

For example we can create Ponnens & Tollens as Knowledge Base *Modus Ponnens*

$$\begin{array}{l} \quad p \\ \quad p \rightarrow q \\ \hline \therefore q \end{array}$$

Modus Tollens

$$\frac{p \Rightarrow q, \neg q}{\therefore \neg p}$$

In [ ]:

```
psource(pl_resolution)
```

In [129…

```
kb = PropKB()
```

For example we can provide for P => Q , P satisfying the implication will give us the conclusion Q

In [130…

```
kb.tell(P|'==>'|Q)
```

In [131…

```
kb.tell(P)
```

In [132…

```
kb.ask_if_true(P)
pl_resolution(kb, P)
```

Out[132]:

```
True
```

Also for ~Q will give us the conclusion ~P

In [133…

```
kb.retract(P)
kb.tell(~Q)
```

In [134…

```
# kb.ask_if_true(~Q)
pl_resolution(kb, ~P)
```

Out[134]:

```
True
```

Forward and Backward Chaining

Untuk Langkah Forward and Backward Chaining kita dapat mengujinya dengan salah satu sample yaitu hukum Silogisme.

Hukum Silogisme menyatakan bahwa untuk kedua Implikasi

P =>Q

Q => R

:. P => R

We can make clauses

In [ ]:
```
psource(PropDefiniteKB.clauses_with_premise)
psource(pl_fc_entails)
```

In [136…
```
clauses = ['P ==> Q',
           'Q ==> R']
```

We will now tell this information to our knowledge base.

In [137…
```
definite_clauses_KB = PropDefiniteKB()
for clause in clauses:
    definite_clauses_KB.tell(expr(clause))
```

By applying the implication, if we add P as valid, then the implication P => Q will be fulfilled and make Q true

In [138…
```
clauses.append('P')
definite_clauses_KB.tell(expr('P'))
```

In [139…
```
pl_fc_entails(definite_clauses_KB, expr('Q'))
```

Out[139]:
```
True
```

The consequence of Q being true is that the implication Q => R will be fulfilled so that R is true

In [140…

```
pl_fc_entails(definite_clauses_KB, expr('R'))
```

Out[140]:

```
True
```

So we can simplify the result of P being true and R being true

In [141…

```
tt_entails( (P |'==>'| Q) & (Q |'==>'| R), P |'==>'| R)
```

Out[141]:

```
True
```