Propositional Model Checking

Outline

I. Forward and Backward Chaining

II. Effective Propositional Model Checking

III. Agents Based on Propositional Logic

* Figures are from the textbook site unless a source is specifically cited.
I. Forward Chaining

**Question** $KB \models q$?

- Begins from positive literals (facts).
- If all the premises of an implications are known, then add its conclusion to $KB$ (as a new fact).
- Continues until $q$ is added or no further inferences can be made.
I. Forward Chaining

Question \( KB \models q? \)

- Begins from positive literals (facts).
- If all the premises of an implications are known, then add its conclusion to KB (as a new fact).
- Continues until \( q \) is added or no further inferences can be made.

```python
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 \leftarrow \text{a table, where } count[c] \text{ is initially the number of symbols in clause } c \text{'s premise}
            inferred \leftarrow \text{a table, where } inferred[s] \text{ is initially } false \text{ for all symbols}
            queue \leftarrow \text{a queue of symbols, initially symbols known to be true in } KB

    while queue is not empty do
        p \leftarrow \text{POP(queue)}
        if p = q then return true
        if inferred[p] = false then
            inferred[p] \leftarrow 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
```
Example of Forward Chaining

KB:

\[ P \Rightarrow Q \]
\[ L \land M \Rightarrow P \]
\[ B \land L \Rightarrow M \]
\[ A \land P \Rightarrow L \]
\[ A \land B \Rightarrow L \]
\[ A \]
\[ B \]
Example of Forward Chaining

KB:

\[ P \implies Q \]
\[ L \land M \implies P \]
\[ B \land L \implies M \]
\[ A \land P \implies L \]
\[ A \land B \implies L \]
\[ A \]
\[ B \]

AND-OR graph representation
Example of Forward Chaining

\[ KB: \]

\[
P \Rightarrow Q \\
L \land M \Rightarrow P \\
B \land L \Rightarrow M \\
A \land P \Rightarrow L \\
A \land B \Rightarrow L \\
A \\
B
\]

Q. \( KB \models Q? \)
$P \Rightarrow Q$
$L \land M \Rightarrow P$
$B \land L \Rightarrow M$
$A \land P \Rightarrow L$
$A \land B \Rightarrow L$
$A$
$B$
\[ P \Rightarrow Q \]
\[ L \land M \Rightarrow P \]
\[ B \land L \Rightarrow M \]
\[ A \land P \Rightarrow L \]
\[ A \land B \Rightarrow L \]

Diagram:
- Node labeled A
- Node labeled B
- Node labeled L
- Node labeled M
- Node labeled Q
- Arrows connecting nodes

Execution
Execution

\[ P \Rightarrow Q \]
\[ L \land M \Rightarrow P \]
\[ B \land L \Rightarrow M \]
\[ A \land P \Rightarrow L \]
\[ A \land B \Rightarrow L \]

\[ A \]
\[ B \]
Execution

\[ P \Rightarrow Q \]
\[ L \land M \Rightarrow P \]
\[ B \land L \Rightarrow M \]
\[ A \land P \Rightarrow L \]
\[ A \land B \Rightarrow L \]

A

B
\[ P \Rightarrow Q \]
\[ L \land M \Rightarrow P \]
\[ B \land L \Rightarrow M \]
\[ A \land P \Rightarrow L \]
\[ A \land B \Rightarrow L \]

\[ A \]
\[ B \]
Execution

\[ P \Rightarrow Q \]
\[ L \land M \Rightarrow P \]
\[ B \land L \Rightarrow M \]
\[ A \land P \Rightarrow L \]
\[ A \land B \Rightarrow L \]

\[ A \]
\[ B \]
Execution

\[ P \Rightarrow Q \]
\[ L \land M \Rightarrow P \]
\[ B \land L \Rightarrow M \]
\[ A \land P \Rightarrow L \]
\[ A \land B \Rightarrow L \]
\[ A \]
\[ B \]
$P \Rightarrow Q$

$L \land M \Rightarrow P$

$B \land L \Rightarrow M$

$A \land P \Rightarrow L$

$A \land B \Rightarrow L$

$L$

$A$

$B$
$P \Rightarrow Q$

$\text{Boxed: } L \land M \Rightarrow P$

$B \land L \Rightarrow M$

$A \land P \Rightarrow L$

$A \land B \Rightarrow L$

$A$

$B$
Execution

\[ P \Rightarrow Q \]

\[ L \land M \Rightarrow P \]

\[ B \land L \Rightarrow M \]

\[ A \land P \Rightarrow L \]

\[ A \land B \Rightarrow L \]

\[ A \]

\[ B \]
Execution

\[
P \Rightarrow Q
\]

\[
\begin{align*}
L \land M & \Rightarrow P \\
B \land L & \Rightarrow M \\
A \land P & \Rightarrow L \\
A \land B & \Rightarrow L \\
A & \\
B
\end{align*}
\]
Execution

Soundness of forward chaining: every inference is an application of Modus Ponens.
Execution

Soundness of forward chaining: every inference is an application of Modus Ponens.
Completeness: every entailed atomic sentences will be derived.
Backward Chaining

- If $q$ is true, no work is needed.
- Otherwise, finds implications in the KB whose conclusion is $q$.
- If all the premises of one of these implications can be proved true (recursively by backward chaining), then $q$ is true.
Backward Chaining

- If $q$ is true, no work is needed.
- Otherwise, finds implications in the KB whose conclusion is $q$.
- If all the premises of one of these implications can be proved true (recursively by backward chaining), then $q$ is true.

\[
P \Rightarrow Q \\
L \land M \Rightarrow P \\
B \land L \Rightarrow M \\
A \land P \Rightarrow L \\
A \land B \Rightarrow L \\
A \\
B
\]
Backward Chaining

- If $q$ is true, no work is needed.
- Otherwise, finds implications in the KB whose conclusion is $q$.
- If all the premises of one of these implications can be proved true (recursively by backward chaining), then $q$ is true.

Q. $KB \models Q$?

\[ P \Rightarrow Q \]
\[ L \land M \Rightarrow P \]
\[ B \land L \Rightarrow M \]
\[ A \land P \Rightarrow L \]
\[ A \land B \Rightarrow L \]
\[ A \]
\[ B \]
Backward Chaining

- If $q$ is true, no work is needed.
- Otherwise, finds implications in the KB whose conclusion is $q$.
- If all the premises of one of these implications can be proved true (recursively by backward chaining), then $q$ is true.

Q. $KB \models Q$?

\[
\begin{align*}
P & \Rightarrow Q \\
L \land M & \Rightarrow P \\
B \land L & \Rightarrow M \\
A \land P & \Rightarrow L \\
A \land B & \Rightarrow L \\
A & \\
B & 
\end{align*}
\]
Backward Chaining

- If $q$ is true, no work is needed.
- Otherwise, finds implications in the KB whose conclusion is $q$.
- If all the premises of one of these implications can be proved true (recursively by backward chaining), then $q$ is true.

Q. $KB \vDash Q$?

$$
P \Rightarrow Q
L \land M \Rightarrow P
B \land L \Rightarrow M
A \land P \Rightarrow L
A \land B \Rightarrow L
A
B$$
Backward Chaining

- If \( q \) is true, no work is needed.
- Otherwise, finds implications in the KB whose conclusion is \( q \).
- If all the premises of one of these implications can be proved true (recursively by backward chaining), then \( q \) is true.

Q. \( KB \models Q? \)

\[
\begin{align*}
P & \Rightarrow Q \\
L \land M & \Rightarrow P \\
B \land L & \Rightarrow M \\
A \land P & \Rightarrow L \\
A \land B & \Rightarrow L \\
A & \\
B &
\end{align*}
\]
Backward Chaining

- If $q$ is true, no work is needed.
- Otherwise, finds implications in the KB whose conclusion is $q$.
- If all the premises of one of these implications can be proved true (recursively by backward chaining), then $q$ is true.

Q. $KB \models Q$?

\[
\begin{align*}
P & \Rightarrow Q \\
L \land M & \Rightarrow P \\
B \land L & \Rightarrow M \\
A \land P & \Rightarrow L \\
A \land B & \Rightarrow L \\
A & \\
B & 
\end{align*}
\]
Backward Chaining

- If $q$ is true, no work is needed.
- Otherwise, finds implications in the KB whose conclusion is $q$.
- If all the premises of one of these implications can be proved true (recursively by backward chaining), then $q$ is true.

**Q.** $KB \models Q$?

\[
\begin{align*}
P & \Rightarrow Q \\
L \land M & \Rightarrow P \\
B \land L & \Rightarrow M \\
A \land P & \Rightarrow L \\
A \land B & \Rightarrow L \\
A & \\
B &
\end{align*}
\]
Backward Chaining

- If \( q \) is true, no work is needed.
- Otherwise, finds implications in the KB whose conclusion is \( q \).
- If all the premises of one of these implications can be proved true (recursively by backward chaining), then \( q \) is true.

**Q.** \( KB \models Q ? \)

\[
\begin{align*}
P & \implies Q \\
L \land M & \implies P \\
B \land L & \implies M \\
A \land P & \implies L \\
A \land B & \implies L \\
A & \\
B &
\end{align*}
\]
Backward Chaining

- If $q$ is true, no work is needed.
- Otherwise, finds implications in the KB whose conclusion is $q$.
- If all the premises of one of these implications can be proved true (recursively by backward chaining), then $q$ is true.

Q. $KB \models Q$?

\[
\begin{align*}
    P & \Rightarrow Q \\
    L \land M & \Rightarrow P \\
    B \land L & \Rightarrow M \\
    A \land P & \Rightarrow L \\
    A \land B & \Rightarrow L \\
    A & \\
    B &
\end{align*}
\]
Backward Chaining

- If \( q \) is true, no work is needed.
- Otherwise, finds implications in the KB whose conclusion is \( q \).
- If all the premises of one of these implications can be proved true (recursively by backward chaining), then \( q \) is true.

**Q.** \( KB \models Q? \)

\[
\begin{align*}
P & \Rightarrow Q \\
L \land M & \Rightarrow P \\
B \land L & \Rightarrow M \\
A \land P & \Rightarrow L \\
A \land B & \Rightarrow L
\end{align*}
\]
Backward Chaining

- If $q$ is true, no work is needed.
- Otherwise, finds implications in the KB whose conclusion is $q$.
- If all the premises of one of these implications can be proved true (recursively by backward chaining), then $q$ is true.

Q. $KB \models Q$?

$$
\begin{align*}
P &\Rightarrow Q \\
L \land M &\Rightarrow P \\
B \land L &\Rightarrow M \\
A \land P &\Rightarrow L \\
A \land B &\Rightarrow L \\
A & \\
B
\end{align*}
$$
Backward Chaining

- If \( q \) is true, no work is needed.
- Otherwise, finds implications in the KB whose conclusion is \( q \).
- If all the premises of one of these implications can be proved true (recursively by backward chaining), then \( q \) is true.

Q. \( KB \models Q \)?
Backward Chaining

- If $q$ is true, no work is needed.
- Otherwise, finds implications in the KB whose conclusion is $q$.
- If all the premises of one of these implications can be proved true (recursively by backward chaining), then $q$ is true.

Q. $KB \models Q$?

\[
\begin{align*}
P & \Rightarrow Q \\
L \land M & \Rightarrow P \\
B \land L & \Rightarrow M \\
A \land P & \Rightarrow L \\
A \land B & \Rightarrow L
\end{align*}
\]
Backward Chaining

- If \( q \) is true, no work is needed.
- Otherwise, finds implications in the KB whose conclusion is \( q \).
- If all the premises of one of these implications can be proved true (recursively by backward chaining), then \( q \) is true.

Q. \( KB \models Q? \)

\[
\begin{align*}
P & \Rightarrow Q \\
L \land M & \Rightarrow P \\
B \land L & \Rightarrow M \\
A \land P & \Rightarrow L \\
A \land B & \Rightarrow L \\
A \\
B
\end{align*}
\]
Backward Chaining

- If \( q \) is true, no work is needed.
- Otherwise, finds implications in the KB whose conclusion is \( q \).
- If all the premises of one of these implications can be proved true (recursively by backward chaining), then \( q \) is true.

Q. \( KB \models Q? \)

\[
\begin{align*}
P & \Rightarrow Q \\
L \land M & \Rightarrow P \\
B \land L & \Rightarrow M \\
A \land P & \Rightarrow L \\
A \land B & \Rightarrow L \\
A \\
B
\end{align*}
\]

AND-OR graph search!
Forward vs. Backward Chaining

♦ Applicable range
  • Prove the entailment of a single proposition symbol
  • KB consists of definite clauses only.

\[
\text{Either } P \text{ or } P_1 \land P_2 \land \cdots P_k \Rightarrow Q
\]

♦ Forward chaining is *data-driven*, automatic, unconscious processing.

♦ It may perform a lot of work that is irrelevant to the goal.

♦ Backward chaining is *goal-driven*, and appropriate for problem solving.

♦ It may run in time sublinear in the size of KB, since it touches only relevant facts.
II. Effective Propositional Model Checking

$KB \models \beta$ if and only if $KB \land \neg \beta$ is unsatisfiable.
II. Effective Propositional Model Checking

KB ⊨ β if and only if KB ∧ ¬β is unsatisfiable.

One sentence in propositional logic (PL)
II. Effective Propositional Model Checking

\[ KB \models \beta \text{ if and only if } KB \land \neg \beta \text{ is unsatisfiable.} \]

One sentence in propositional logic (PL)

**Satisfiability problem** Is a sentence \( s \) in PL satisfiable?
II. Effective Propositional Model Checking

\( KB \models \beta \text{ if and only if } KB \land \neg \beta \text{ is unsatisfiable.} \)

One sentence in propositional logic (PL)

**Satisfiability problem**  Is a sentence \( s \) in PL satisfiable?

- Cast the problem as one of constraint satisfaction.

Many combinatorial problems in computer science can be reduced to checking the satisfiability of a propositional sentence.
II. Effective Propositional Model Checking

\( KB \models \beta \text{ if and only if } KB \land \neg \beta \text{ is unsatisfiable.} \)

One sentence in propositional logic (PL)

**Satisfiability problem**  Is a sentence \( s \) in PL satisfiable?

- Cast the problem as one of constraint satisfaction.

  Many combinatorial problems in computer science can be reduced to checking the satisfiability of a propositional sentence.

- Complete backtracking search (DPLL algorithm)

- Incomplete local search (WALKSAT algorithm)
DPLL Algorithm

Davis, Putnam, Logemann, and Loveland (1960, 1962)

With enhancements, modern solvers can handle a problem with a multiple of \(10^7\) variables.
DPLL Algorithm

Davis, Putnam, Logemann, and Loveland (1960, 1962)
With enhancements, modern solvers can handle a problem with a multiple of $10^7$ variables.

```plaintext
function DPLL-SATISFIABLE?(s) returns true or false
inputs: s, a sentence in propositional logic

clauses ← the set of clauses in the CNF representation of s
symbols ← a list of the proposition symbols in s
return DPLL(clauses, symbols, {})  

function DPLL(clauses, symbols, model) returns true or false

if every clause in clauses is true in model then return true
if some clause in clauses is false in model then return false
P, value ← FIND-PURE-SYMBOL(symbols, clauses, model)
if P is non-null then return DPLL(clauses, symbols − P, model ∪ {P=value})
P, value ← FIND-UNIT-CLAUSE(clauses, model)
if P is non-null then return DPLL(clauses, symbols − P, model ∪ {P=value})
P ← FIRST(symbols); rest ← REST(symbols)
return DPLL(clauses, rest, model ∪ {P=true}) or
       DPLL(clauses, rest, model ∪ {P=false}))
```
DPLL Algorithm

Davis, Putnam, Logemann, and Loveland (1960, 1962)

With enhancements, modern solvers can handle a problem with a multiple of $10^7$ variables.

Early termination: a clause is true if any of its literals is true. E.g., $A \lor \neg B \lor \neg C$ is true if $A$ is true (regardless of the values assigned to $B$ and $C$).

```python
function DPLL-SATISFIABLE?(s) returns true or false
inputs: s, a sentence in propositional logic

clauses ← the set of clauses in the CNF representation of s
symbols ← a list of the proposition symbols in s
return DPLL(clauses, symbols, { })

function DPLL(clauses, symbols, model) returns true or false

if every clause in clauses is true in model then return true
if some clause in clauses is false in model then return false

P, value ← FIND-PURE-SYMBOL(symbols, clauses, model)
if P is non-null then return DPLL(clauses, symbols − P, model ∪ {P=value})

P, value ← FIND-UNIT-CLAUSE(clauses, model)
if P is non-null then return DPLL(clauses, symbols − P, model ∪ {P=value})

P ← FIRST(symbols); rest ← REST(symbols)
return DPLL(clauses, rest, model ∪ {P=true}) or
    DPLL(clauses, rest, model ∪ {P=false})
```
DPLL Algorithm

Davis, Putnam, Logemann, and Loveland (1960, 1962)

With enhancements, modern solvers can handle a problem with a multiple of $10^7$ variables.

```
function DPLL-SATISFIABLE?(s) returns true or false
    inputs: s, a sentence in propositional logic
    clauses ← the set of clauses in the CNF representation of s
    symbols ← a list of the proposition symbols in s
    return DPLL(clauses, symbols, { })

function DPLL(clauses, symbols, model) returns true or false
    if every clause in clauses is true in model then return true
    if some clause in clauses is false in model then return false
    P, value ← FIND-PURE-SYMBOL(symbols, clauses, model)
    if P is non-null then return DPLL(clauses, symbols − P, model ∪ {P=value})
    P, value ← FIND-UNIT-CLAUSE(clauses, model)
    if P is non-null then return DPLL(clauses, symbols − P, model ∪ {P=value})
    P ← FIRST(symbols); rest ← REST(symbols)
    return DPLL(clauses, rest, model ∪ {P=true}) or DPLL(clauses, rest, model ∪ {P=false})
```

**Early termination**: a clause is true if any of its literals is true. E.g., $A \lor \neg B \lor \neg C$ is true if $A$ is true (regardless of the values assigned to $B$ and $C$).

**Pure symbol**: a symbol appearing always positive or always negative in all clauses. E.g., $A$ and $B$ are pure in $A \lor \neg B$, $\neg B \lor \neg C$, $C \lor A$ while $C$ is not pure. Assignment $A ← true$ will reduce the set to $\neg B \lor \neg C$, enabling $C$ to become a pure symbol.
DPLL Algorithm

Davis, Putnam, Logemann, and Loveland (1960, 1962)
With enhancements, modern solvers can handle a problem with a multiple of $10^7$ variables.

function DPLL-SATISFIABLE?(s) returns true or false
inputs: s, a sentence in propositional logic

clauses ← the set of clauses in the CNF representation of s
symbols ← a list of the proposition symbols in s
return DPLL(clauses, symbols, { })

function DPLL(clauses, symbols, model) returns true or false

if every clause in clauses is true in model then return true
if some clause in clauses is false in model then return false

P, value ← FIND-PURE-SYMBOL(symbols, clauses, model)
if P is non-null then return DPLL(clauses, symbols - P, model ∪ {P=value})

P, value ← FIND-UNIT-CLAUSE(clauses, model)
if P is non-null then return DPLL(clauses, symbols - P, model ∪ {P=value})

P ← FIRST(symbols); rest ← REST(symbols)
return DPLL(clauses, rest, model ∪ {P=true}) or
DPLL(clauses, rest, model ∪ {P=false})

Early termination: a clause is true if any of its literals is true. E.g., $A \lor \neg B \lor \neg C$ is true if $A$ is true (regardless of the values assigned to $B$ and $C$).

Pure symbol: a symbol appearing always positive or always negative in all clauses. E.g., $A$ and $B$ are pure in $A \lor \neg B \lor \neg C$, $C \lor A$ while $C$ is not pure. Assignment $A \leftarrow$ true will reduce the set to $\neg B \lor \neg C$, enabling $C$ to become a pure symbol.
DPLL Algorithm

Davis, Putnam, Logemann, and Loveland (1960, 1962)
With enhancements, modern solvers can handle a problem with a multiple of $10^7$ variables.

**Pure symbol:** a symbol appearing always positive or always negative in all clauses. E.g., $A \lor \neg B \lor \neg C$ is true if $A$ is true (regardless of the values assigned to $B$ and $C$).

**Unit clause propagation** on a clause in which all literals but one are assigned false. E.g., $\neg B \lor \neg C$ simplifies to the unit clause $\neg C$ if $B = true$.

**Early termination:** a clause is true if any of its literals is true. E.g., $A \lor \neg B \lor \neg C$ is true if $A$ is true (regardless of the values assigned to $B$ and $C$).

```python
function DPLL-SATISFIABLE?(s) returns true or false
    inputs: s, a sentence in propositional logic
    clauses ← the set of clauses in the CNF representation of s
    symbols ← a list of the proposition symbols in s
    return DPLL(clauses, symbols, {})

function DPLL(clauses, symbols, model) returns true or false
    if every clause in clauses is true in model then return true
    if some clause in clauses is false in model then return false
    P, value ← FIND-PURE-SYMBOL(symbols, clauses, model)
    if P is non-null then return DPLL(clauses, symbols – P, model ∪ {P=value})
    P, value ← FIND-UNIT-CLAUSE(clauses, model)
    if P is non-null then return DPLL(clauses, symbols – P, model ∪ {P=value})
    P ← FIRST(symbols); rest ← REST(symbols)
    return DPLL(clauses, rest, model ∪ {P=true}) or DPLL(clauses, rest, model ∪ {P=false})
```

Truth value to assign to $P$
Dealing with pure symbols only

Dealing with unit clauses only

DPLL(\ldots,\{P = true\})

DPLL(\ldots,\{P = false\})
Recursion Tree

\[ \text{DPLL(\ldots, \{P = \text{true}\})} \]
\[ \Downarrow \]
\[ \text{Dealing with pure symbols only} \]

\[ \vdots \]
\[ \text{DPLL(\ldots, \{P = \text{false}\})} \]
\[ \Downarrow \]
\[ \text{Dealing with unit clauses only} \]

Can be handled within one call using iterations.
Local Search Algorithms

- Take steps in the space of complete assignments, flipping the truth value of one symbol at a time.

- Use an evaluation that counts the number of unsatisfied clauses.

- Escape local minima using various forms of randomness.

- Find a good balance between greediness and randomness.
The WALKSAT Algorithm

function WALKSAT(clauses, p, max_flips) returns a satisfying model or failure
inputs: clauses, a set of clauses in propositional logic

    p, the probability of choosing to do a “random walk” move, typically around 0.5
    max_flips, number of value flips allowed before giving up

model ← a random assignment of true/false to the symbols in clauses
for each $i = 1$ to max_flips do
    if model satisfies clauses then return model
    clause ← a randomly selected clause from clauses that is false in model
    if RANDOM(0, 1) ≤ p then
        flip the value in model of a randomly selected symbol from clause
    else flip whichever symbol in clause maximizes the number of satisfied clauses
return failure
The WALKSAT Algorithm

function WALKSAT(clauses, p, max_flips) returns a satisfying model or failure
inputs: clauses, a set of clauses in propositional logic
        p, the probability of choosing to do a “random walk” move, typically around 0.5
        max_flips, number of value flips allowed before giving up

model ← a random assignment of true/false to the symbols in clauses
for each \( i = 1 \) to max_flips do
    if model satisfies clauses then return model
    clause ← a randomly selected clause from clauses that is false in model
    if RANDOM(0, 1) < \( p \) then
        flip the value in model of a randomly selected symbol from clause
    else flip whichever symbol in clause maximizes the number of satisfied clauses
return failure
The WALKSAT Algorithm

function WALKSAT(clauses, p, max_flips) returns a satisfying model or failure
inputs: clauses, a set of clauses in propositional logic
p, the probability of choosing to do a “random walk” move, typically around 0.5
max_flips, number of value flips allowed before giving up

model ← a random assignment of true/false to the symbols in clauses
for each i = 1 to max_flips do
    if model satisfies clauses then return model
    clause ← a randomly selected clause from clauses that is false in model
    if RANDOM(0, 1) ≤ p then
        flip the value in model of a randomly selected symbol from clause
    else flip whichever symbol in clause maximizes the number of satisfied clauses
return failure
III. Agent Based on Propositional Logic

- Write down a complete **logical model** of the effects of action.
- How **logical inference** can be used by an agent?.
- How to keep track of the world without resorting to **inference** history?
- How to use **logical inference** to construct plans based on the KB?

Knowledge base (KB):

- general knowledge about how the world works
- percept sentences obtained in a particular world
Current State in the Wumpus World

### Axioms:

- \( \neg P_{1,1} \)  
- \( \neg W_{1,1} \)  
- \( B_{1,1} \iff (P_{1,2} \lor P_{2,1}) \)  
- \( S_{1,1} \iff (W_{1,2} \lor W_{2,1}) \)  

...  

// 16 rules of this type  
// 16

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>START</strong></td>
<td><strong>Breeze</strong></td>
<td><strong>PIT</strong></td>
<td><strong>Breeze</strong></td>
</tr>
<tr>
<td>1</td>
<td><strong>Breeze</strong></td>
<td><strong>Pit</strong></td>
<td><strong>Breeze</strong></td>
</tr>
<tr>
<td>2</td>
<td><strong>Stench</strong></td>
<td><strong>Breeze</strong></td>
<td><strong>Stench</strong></td>
</tr>
<tr>
<td>3</td>
<td><strong>Wumpus</strong></td>
<td><strong>Breeze</strong></td>
<td><strong>Stench</strong></td>
</tr>
<tr>
<td>4</td>
<td><strong>Stench</strong></td>
<td><strong>Breeze</strong></td>
<td><strong>Pit</strong></td>
</tr>
</tbody>
</table>

- \( P_{x,y} = \text{true} \) if there is a pit in \([x, y]\).
- \( W_{x,y} = \text{true} \) if there is a wumpus in \([x, y]\), dead or alive.
- \( B_{x,y} = \text{true} \) if the agent perceives a breeze in \([x, y]\).
- \( S_{x,y} = \text{true} \) if the agent perceives a stench in \([x, y]\).
Current State in the Wumpus World

Axioms:

\[ \neg P_{1,1}, \neg W_{1,1} \]
\[ B_{1,1} \iff (P_{1,2} \lor P_{2,1}) \quad // 16 \text{ rules of this type} \]
\[ S_{1,1} \iff (W_{1,2} \lor W_{2,1}) \quad // 16 \]
\[ \ldots \]

\[ \text{Exactly one wumpus} \]
\[ W_{1,1} \lor W_{1,2} \lor \cdots \lor W_{4,3} \lor W_{4,4} \quad // \geq 1 \text{ wumpus} \]
\[ \neg W_{i,j} \lor \neg W_{k,l} \quad 1 \leq i, j, k, l \leq 4 \text{ and } (i, j) \neq (k, l) \]
\[ \quad // \leq 1 \text{ Wumpus;} \]
\[ \quad // \frac{16 \times 15}{2} = 120 \text{ rules} \]

- \( P_{x,y} = true \) if there is a pit in \([x, y]\).
- \( W_{x,y} = true \) if there is a wumpus in \([x, y]\), dead or alive.
- \( B_{x,y} = true \) if the agent perceives a breeze in \([x, y]\).
- \( S_{x,y} = true \) if the agent perceives a stench in \([x, y]\).
Representing Percepts

A percept asserts something only about the current time.

$Stench^4$: the agent senses stench at time step 4 (in square $A$).
$\neg Stench^3$: the agent senses no stench at time step 3 (in square $B$).
Representing Percepts

A percept asserts something only about the current time.

\[ Stench^4: \text{ the agent senses stench at time step 4 (in square } A) \]
\[ \neg Stench^3: \text{ the agent senses no stench at time step 3 (in square } B) \]

Associate propositions with time steps for aspects of the world that changes over time.

\[ L^0_{1,1}: \text{ the agent is in square } [1, 1] \text{ at time step 0.} \]
Representing Perceps

A percept asserts something only about the current time.

$Stench^4$: the agent senses stench at time step 4 (in square $A$).
$\neg Stench^3$: the agent senses no stench at time step 3 (in square $B$).

Associate propositions with time steps for aspects of the world that changes over time.

$L^0_{1,1}$: the agent is in square $[1, 1]$ at time step 0.

For any time step $t$ and any square $[x, y]$,
Representing Percects

A percept asserts something only about the current time.

\[ \text{Stench}^4: \text{ the agent senses stench at time step 4 (in square } A). \]
\[ \neg \text{Stench}^3: \text{ the agent senses no stench at time step 3 (in square } B). \]

Associate propositions with time steps for aspects of the world that changes over time.

\[ L_{1,1}^0: \text{ the agent is in square } [1, 1] \text{ at time step 0.} \]

For any time step \( t \) and any square \([x, y]\),

\[ L_{x,y}^t \Rightarrow (Breeze^t \Leftrightarrow B_{x,y}) \]
\[ L_{x,y}^t \Rightarrow (Stench^t \Leftrightarrow S_{x,y}) \]
Representing Percepts

A percept asserts something only about the current time.

\textbf{Stench}^4:  the agent senses stench at time step 4 (in square A).
\textbf{¬Stench}^3:  the agent senses no stench at time step 3 (in square B).

Associate propositions with time steps for aspects of the world that changes over time.

\(L^0_{1,1}:\)  the agent is in square \([1, 1]\) at time step 0.

For any time step \(t\) and any square \([x, y]\),

\(L^t_{x,y} \Rightarrow (Breeze^t \iff B_{x,y})\)
\(L^t_{x,y} \Rightarrow (Stench^t \iff S_{x,y})\)
Describing a Transition Model

$Forward^t$: the agent executes the forward action at time $t$. 
Describing a Transition Model

$Forward^t$: the agent executes the forward action at time $t$.

**Effect axioms** specify outcome of an action at the next time step.

$$L_{1,1}^0 \land FacingEast^0 \land Forward^0 \Rightarrow (L_{2,1}^1 \land \neg L_{1,1}^1)$$

// if the agent is at [1,1] facing east at time 0 and goes forward,
// the result is that the agent is in [2,1] and no longer in [1,1].
Describing a Transition Model

*Forward*: the agent executes the forward action at time $t$.

**Effect axioms** specify outcome of an action at the next time step.

\[ L_{1,1}^0 \land \text{FacingEast}^0 \land \text{Forward}^0 \Rightarrow (L_{2,1}^1 \land \lnot L_{1,1}^1) \]

// if the agent is at [1,1] facing east at time 0 and goes forward,
// the result is that the agent is in [2,1] and no longer in [1,1].

**Frame axioms** assert all the proportions that remain the same.
Describing a Transition Model

\[ \text{Forward}^t : \text{the agent executes the forward action at time } t. \]

**Effect axioms** specify outcome of an action at the next time step.

\[ L_{1,1}^0 \land \text{FacingEast}^0 \land \text{Forward}^0 \Rightarrow (L_{2,1}^1 \land \neg L_{1,1}^1) \]

// if the agent is at [1,1] facing east at time 0 and goes forward, // the result is that the agent is in [2,1] and no longer in [1,1].

**Frame axioms** assert all the proportions that remain the same.

\[ \text{Forward}^t \Rightarrow (\text{HaveArrow}^t \iff \text{HaveArrow}^{t+1}) \]
\[ \text{Forward}^t \Rightarrow (\text{WumpusAlive}^t \iff \text{WumpusAlive}^{t+1}) \]
Describing a Transition Model

Forward\textsuperscript{t}: the agent executes the forward action at time \(t\).

**Effect axioms** specify outcome of an action at the next time step.

\[ L_{1,1}^{0} \land \text{FacingEast}^{0} \land \text{Forward}^{0} \Rightarrow (L_{2,1}^{1} \land \neg L_{1,1}^{1}) \]

// if the agent is at [1,1] facing east at time 0 and goes forward, // the result is that the agent is in [2,1] and no longer in [1,1].

**Frame axioms** assert all the proportions that remain the same.

\[ \text{Forward}^{t} \Rightarrow (\text{HaveArrow}^{t} \Leftrightarrow \text{HaveArrow}^{t+1}) \]
\[ \text{Forward}^{t} \Rightarrow (\text{WumpusAlive}^{t} \Leftrightarrow \text{WumpusAlive}^{t+1}) \]

\(O(mn)\) frame axioms for \(m\) actions and \(n\) fluents
Axioms for Successor States

**Successor-state axiom**, one for every fluent $F$, states that
- either the action at $t$ causes $F$ to be true at $t + 1$,
- or $F$ was already true at $t$ and the action does not cause it to be false.

$$F^{t+1} \iff \text{ActionCauses}F^t \lor (F^t \land \neg \text{ActionCausesNot}F^t)$$
Axioms for Successor States

**Successor-state axiom**, one for every fluent $F$, states that

- either the action at $t$ causes $F$ to be true at $t + 1$,
- or $F$ was already true at $t$ and the action does not cause it to be false.

$$F^{t+1} \iff \text{ActionCauses}F^t \lor (F^t \land \neg \text{ActionCausesNot}F^t)$$

$$\text{HaveArrow}^{t+1} \iff (\text{HaveArrow}^t \land \neg \text{Shoot}^t)$$
**Axioms for Successor States**

*Successor-state axiom*, one for every fluent $F$, states that
- either the action at $t$ causes $F$ to be true at $t+1$,
- or $F$ was already true at $t$ and the action does not cause it to be false.

$$ F^{t+1} \iff ActionCausesF^t \lor (F^t \land \neg ActionCausesNotF^t) $$

$$ HaveArrow^{t+1} \iff (HaveArrow^t \land \neg Shoot^t) $$

// no action, e.g., for reloading
**Axioms for Successor States**

*Successor-state axiom*, one for every fluent $F$, states that

- either the action at $t$ causes $F$ to be true at $t + 1$,
- or $F$ was already true at $t$ and the action does not cause it to be false.

\[ F^{t+1} \iff \text{Action Causes } F^t \lor (F^t \land \neg \text{Action Causes Not } F^t) \]

\[ \text{Have Arrow}^{t+1} \iff (\text{Have Arrow}^t \land \neg \text{Shoot}^t) \]

// no action, e.g., for reloading

\[ L_{1,1}^{t+1} \iff (L_{1,1}^t \land (\neg \text{Forward}^t \lor \text{Bump}^{t+1})) \lor (L_{1,2}^t \land (\text{Facing South}^t \lor \text{Forward}^t)) \]
\[ \lor (L_{2,1}^t \land (\text{Facing West}^t \lor \text{Forward}^t)) \]
Axioms for Successor States

**Successor-state axiom**, one for every fluent \( F \), states that
- either the action at \( t \) causes \( F \) to be true at \( t + 1 \),
- or \( F \) was already true at \( t \) and the action does not cause it to be false.

\[
F^{t+1} \iff \text{ActionCauses}\!\!F^t \lor (F^t \land \neg\text{ActionCausesNot}\!\!F^t)
\]

\[
\text{HaveArrow}^{t+1} \iff (\text{HaveArrow}^t \land \neg\text{Shoot}^t)
\]

// no action, e.g., for reloading

\[
L_{1,1}^{t+1} \iff (L_{1,1}^t \land (\neg\text{Forward}^t \lor \text{Bump}^{t+1})) \lor (L_{1,2}^t \land (\text{FacingSouth}^t \lor \text{Forward}^t)) \lor (L_{2,1}^t \land (\text{FacingWest}^t \lor \text{Forward}^t))
\]

**Square-OK axiom** asserts that a square is free of a pit or live Wumpus.

\[
\text{OK}^{t}_{x,y} \iff \neg P_{x,y} \land \neg(W_{x,y} \land \text{WumpusAlive}^t)
\]
Initial Percepts and Actions

¬Stench\(^0\) \& ¬Breeze\(^0\) \& ¬Glitter\(^0\) \& ¬Bump\(^0\) \& ¬Scream\(^0\); Forward\(^0\)
¬Stench\(^1\) \& Breeze\(^1\) \& ¬Glitter\(^1\) \& ¬Bump\(^1\) \& ¬Scream\(^1\); TurnRight\(^1\)
¬Stench\(^2\) \& Breeze\(^2\) \& ¬Glitter\(^2\) \& ¬Bump\(^2\) \& ¬Scream\(^2\); TurnRight\(^2\)
¬Stench\(^3\) \& Breeze\(^3\) \& ¬Glitter\(^3\) \& ¬Bump\(^3\) \& ¬Scream\(^3\); Forward\(^3\)
¬Stench\(^4\) \& ¬Breeze\(^4\) \& ¬Glitter\(^4\) \& ¬Bump\(^4\) \& ¬Scream\(^4\); TurnRight\(^4\)
¬Stench\(^5\) \& ¬Breeze\(^5\) \& ¬Glitter\(^5\) \& ¬Bump\(^5\) \& ¬Scream\(^5\); Forward\(^5\)
Stench\(^6\) \& ¬Breeze\(^6\) \& ¬Glitter\(^6\) \& ¬Bump\(^6\) \& ¬Scream\(^6\)

Query the knowledge base:

\[ \text{ASK}(KB, L_{1,2}^6) = true \]
\[ \text{ASK}(KB, W_{1,3}) = true \]
\[ \text{ASK}(KB, P_{3,1}) = true \]
\[ \text{ASK}(KB, OK_{2,2}^6) = true \]

// the square [2,2] is OK to move into.