Generalized Modus Ponens

This rule allows us to derive an implication...

\[
\begin{align*}
\text{True} & \quad \text{implies } p_i \\
p_1 \text{ and } \ldots \text{ and } p_i \text{ and } \ldots p_n & \quad \text{implies } q \\
p_1 \ldots p_{i-1} \text{ and } p_{i+1} \ldots p_n & \quad \text{implies } q
\end{align*}
\]

allows:

\[
\begin{align*}
a_1 \text{ and } \ldots a_i \text{ and } \ldots a_n & \quad \text{implies } p_i \\
p_1 \text{ and } \ldots p_j \text{ and } \ldots p_n & \quad \text{implies } q \\
p_1 \ldots p_{i-1} \text{ and } p_{i+1} \ldots p_n & \quad \text{and} \\
a_1 \text{ and } \ldots a_i \text{ and } \ldots a_n & \quad \text{implies } q
\end{align*}
\]

A benefit of this approach is that the reasoner now can have a single goal. To derive \(q\) from \((p_1 \text{ and } \ldots p_i \text{ and } \ldots p_n \quad \text{implies } q)\), use generalized modus ponens to

\[
\text{drive } p_1 \text{ and } p_2 \text{ and } \ldots p_n \quad \text{to} \quad \text{True}
\]

This is like recursion to a base case....
Generalized Modus Ponens

Many AI techniques are based on a predicate logic, extended in particular ways, using generalized *modus ponens* as the inference rule. It is simple to program and reasonably powerful.

The programming language Prolog is based on just this sort of logic.
The Catch...

But it’s not enough. Notice that I said “reasonably powerful”.

*Modus ponens* is sound and complete. It derives only true sentences, and it can derive any true sentence that a knowledge base of this form entails.

Notice that I said “of this form”.

*Modus ponens* works only for knowledge bases that contain **only** implications of positive literals.

Implications of positive literals are often called *Horn clauses*, after a logician who studied them deeply.

But disjunction (or) and negation (not) break the rule. And many legal sentences cannot be expressed in Horn clauses.
The Catch...

For example:

Eugene is a Hoosier, or the Eugene is crazy.

We can write this as a disjunction:

\( \text{isHoosier( Eugene ) or isCrazy( Eugene )} \)

We could use negation to convert this to implications:

\( (\text{not hoosier( Eugene )}) \Rightarrow \text{isCrazy( Eugene )} \)
\( (\text{not isCrazy( Eugene )}) \Rightarrow \text{hoosier( Eugene )} \)

Or we could use the semantics of implication to convert this to a clause:

\[
\text{True implies ( isHoosier( Eugene ) or isCrazy( Eugene ) )}
\]

But we cannot eliminate all the ors and all the nots at the same time.
The Catch, Part 2

So, Horn clause form is incomplete. That is, we cannot represent all sentences in this form.

But why is Horn clause form necessary?

Here is an example of how *modus ponens* breaks down if we use negation or disjunction.

Suppose that all classes at some university meet either Mon/Wed/Fri or Tue/Thu. The AI course meets at 2:30 PM in the afternoon, and Jane has volleyball practice Thursdays and Fridays at that time.

Can Jane take AI?
The Catch, Part 2

Of course not!

But *modus ponens* cannot figure that out! See:

True

implies ( TueThu( AI, 2:30 PM ) or MonWedFri( AI, 2:30 PM ) )

( TueThu( AI, 2:30 PM ) and busy( Thursday, 2:30 PM ) ) implies conflict( AI )

( MonWedFri( AI, 2:30 PM ) and busy( Friday, 2:30 PM ) ) implies conflict( AI )

True implies busy( Thursday, 2:30 PM )

True implies busy( Friday, 2:30 PM )
We can apply *modus ponens* to [2, 4] and [3, 5] to derive the following:

True **implies** (  
   TueThu( AI, 2:30 PM ) or  
   MonWedFri( AI, 2:30 PM )  )

TueThu( AI, 2:30 PM ) **implies** conflict( AI )

MonWedFri( AI, 2:30 PM ) **implies** conflict( AI )

But *modus ponens* cannot take us any farther, despite what’s “obvious” about those last two sentences.
Fixing the Catch

So, Horn clause form is incomplete. But, if we allow something non-Horn into our knowledge base, then modus ponens is incomplete.

What do we do now?

We need an inference rule that handles disjunctions and negations. If we find one, then we will be able to handle any expression...

Why? Because we can use the semantics of implication to convert our Horn clauses into disjunctions with negations.

\[
\text{childOf}(x, y) \textbf{ and } \text{likes}(y, \text{Basketball}) \\
\textbf{implies} \\
\text{likes}(x, \text{Basketball})
\]

\[
\text{not} [ \text{childOf}(x, y) \textbf{ and } \text{likes}(y, \text{Basketball}) ] \\
\textbf{or} \\
\text{likes}(x, \text{Basketball})
\]
Toward Fixing the Catch

And then:

\[ \text{not } [ \text{childOf}(x, y) \text{ and } \text{likes}(y, \text{Basketball}) ] \]
\[ \text{or} \]
\[ \text{likes}(x, \text{Basketball}) \]

BECOMES

\[ [ \text{not } \text{childOf}(x, y) ] \]
\[ \text{or} \]
\[ [ \text{not } \text{likes}(y, \text{Basketball}) ] \]
\[ \text{or} \]
\[ \text{likes}(x, \text{Basketball}) \]

More generally,

\[ p_1 \text{ and } p_2 \ldots \text{ and } p_n \text{ implies } q \]

becomes:

\[ (\text{not } p_1) \text{ or } (\text{not } p_2) \ldots \text{ or } (\text{not } p_n) \text{ or } q \]
An Example: Converting Sentences into Clause Form

1. isHoosier( Eugene )
2. ∀x isHoosier(x) implies likes( x, Basketball )
3. ∀xy childOf( x, y ) and likes( y, Basketball ) implies likes( x, Basketball )
4. ∀x likes( x, Basketball ) implies likes( x, March )
5. daughter( Ellen, Eugene )
6. ∀xy daughter( x, y ) implies childOf( x, y )

1. isHoosier( Eugene )
2. not isHoosier(x) or likes( x, Basketball )
3. [ not childOf( x, y ) ]
   or [ not likes( y, Basketball ) ]
   or likes( x, Basketball )
4. [ not likes( x, Basketball ) ] or likes( x, March )
5. daughter( Ellen, Eugene )
6. [ not daughter( x, y ) ] or childOf( x, y )
We’re Almost There...

Now we are able to write any sentence in the predicate logic, using only the connectives not and or.

- Any sentence that uses implies can be converted using the rule: \((p \implies q) \equiv (\neg p) \lor q\)

- Any sentence that uses and can be rewritten as separate sentences!

This form is called “clause form”, or disjunctive normal form.

So, now we can write our sentences in a new, complete form. If only we had an inference rule that worked on disjunctions and negations, we would be able to infer any true sentence.

The semantics of the or connective lead us to a new inference rule:
The Solution: Resolution!

The semantics of or lead us to a new inference rule:

\[
\begin{align*}
    p & \quad \text{or} \quad \text{disjunction}_1 \\
    \lnot p & \quad \text{or} \quad \text{disjunction}_2 \\
    \text{disjunction}_1 & \quad \text{or} \quad \text{disjunction}_2
\end{align*}
\]

We call this inference rule resolution because it resolves the case analysis that is required whenever one sentence asserts \( p \) and another asserts \( \lnot p \).

Let us consider our new knowledge base:

1. \( \text{isHoosier( Eugene )} \)
2. \( \lnot \text{isHoosier}(x) \quad \text{or} \quad \text{likes}( x, \text{Basketball}) \)
3. \( [ \lnot \text{childOf}( x, y ) ] \quad \text{or} \quad [ \lnot \text{likes}( y, \text{Basketball} ) ] \quad \text{or} \quad \text{likes}( x, \text{Basketball} ) \)
4. \( [ \lnot \text{likes}( x, \text{Basketball} ) ] \quad \text{or} \quad \text{likes}( x, \text{March} ) \)
5. \( \text{daughter}( \text{Ellen, Eugene} ) \)
6. \( [ \lnot \text{daughter}( x, y ) ] \quad \text{or} \quad \text{childOf}( x, y ) \)

Can we infer that Ellen likes March?
Yes, But How?

Using resolution as our inference rule, we conclude:

Ellen likes March.

by constructing a proof by contradiction...

To derive the sentence: \( \text{likes}(\text{Ellen}, \text{March}) \)

We build a proof by contradiction in this way:

1. Assume that our goal sentence is false.
   \[ \text{not \ likes}(\text{Ellen}, \text{March}) \]

2. Try to show that the goal sentence being false causes a contradiction.
   ... try to infer FALSE ...

If assuming that the goal sentence is false causes a contradiction, then it must not be false. Everything else in our knowledge base is true (or so we assume), and so the cause of the contradiction is our negation.
An Example Proof by Contradiction

\[
\begin{align*}
& \text{p or disjunction}_1 \\
& \text{[not p] or disjunction}_2 \\
\end{align*}
\]

\[
\begin{align*}
\text{disjunction}_1 & \text{ or disjunction}_2
\end{align*}
\]

1. isHoosier(Eugene)
2. [not isHoosier(x)] or likes(x, Basketball)
3. [not childOf(x, y)] or [not likes(y, Basketball)] or likes(x, Basketball)
4. [not likes(x, Basketball)] or likes(x, March)
5. daughter(Ellen, Eugene)
6. [not daughter(x, y)] or childOf(x, y)
7. [not likes(Ellen, March)] ASSUMPTION

8. [not likes(Ellen, Basketball)]
9. [not childOf(Ellen, y)] or [not likes(y, Basketball)]
A. [not daughter(Ellen, y)] or
   [not likes(y, Basketball)]
B. [not likes(Eugene, Basketball)]
C. [not isHoosier(Eugene)]
13. FALSE

Since assuming that Ellen does not like March cause a contradiction, then it must follow from the knowledge base that Ellen does like March.
But Can Jane Take AI?

Big deal! you might say. We could prove that with modus ponens. Can resolution do something modus ponens couldn’t?

Use resolution to show that our good friend Jane cannot take AI from this knowledge base:

True implies
    ( TueThu( AI, 2:30 PM ) or
     MonWedFri( AI, 2:30 PM ) )

( TueThu( AI, 2:30 PM ) and
busy( Thursday, 2:30 PM ) ) implies
    conflict( AI )

( MonWedFri( AI, 2:30 PM ) and
busy( Friday, 2:30 PM ) ) implies
    conflict( AI )

True implies
    busy( Thursday, 2:30 PM )

True implies
    busy( Friday, 2:30 PM )

First, convert these sentences to clause form. Then, assume Jane can take AI...
Poor Jane

1. TueThu( AI, 2:30 PM ) or MonWedFri( AI, 2:30 PM )
2. [ not TueThu( AI, 2:30 PM ) ] or
   [ not busy( Thursday, 2:30 PM ) ] or conflict( AI )
3. [ not MonWedFri( AI, 2:30 PM ) ] or
   [ not busy( Friday, 2:30 PM ) ] or conflict( AI )
4. busy( Thursday, 2:30 PM )
5. busy( Friday, 2:30 PM )

6. not conflict( AI ) (not conflict( AI )

7. not TueThu( AI, 2:30 PM ) ] or
   [ not busy( Thursday, 2:30 PM ) ] (2, 6)
8. MonWedFri( AI, 2:30 PM ) or
   [ not busy( Thursday, 2:30 PM ) ] (1, 7)
9. MonWedFri( AI, 2:30 PM ) (4, 8)
A. [ not busy( Friday, 2:30 PM ) ] or
   conflict( AI ) (3, 9)
B. conflict( AI ) (5, A)
C. FALSE (6, B)

So, resolution can handle cases that generalized modus ponens can’t!