Special Topics Lecture:

Why empty KB is TRUE and empty Clause is FALSE

by Rick Lathrop

Notation used in this Special Topics lecture

- Prefix notation (or extended Polish notation)
  - The operator appears first, followed by its arguments
  - (AND A B C) in **prefix** notation is (A AND B AND C) in **infix** notation
  - (OR A B C) in **prefix** notation is (A OR B OR C) in **infix** notation
  - Prefix notation enables an easy parser — it looks at the first element of each list and dispatches args to a handler for that operator token.
  - Lisp and related languages define their entire syntax in prefix notation
  - See https://en.wikipedia.org/wiki/Polish_notation
    - Prefix allows any arity, delimited by (...); strict Polish = fixed arity

- Conjunctive Normal Form (CNF) — the AND of ORs
  - KB = {AND (OR literal-1 literal-2 ...) (OR literal-3 literal-4 ...) ...}

- Drop ANDs and ORs — we know where they are (clausal notation)
  - KB = { (literal-1 literal-2 ...) (literal-3 literal-4 ...) ...}
  - In this lecture, KB uses {...} brackets, clauses use (...) parentheses
  - Here, the empty KB { } always means {AND} with no clauses
  - Here, the empty clause () always means (OR) with no literals

Side Trip: Functions AND, OR, and identity values

```
function AND(arglist) returns a truth-value
   return ANDOR(arglist, TRUE)
/* Think of AND as by default TRUE, but args may make it FALSE */

function OR(arglist) returns a truth-value
   return ANDOR(arglist, FALSE)
/* Think of OR as by default FALSE, but args may make it TRUE */

function ANDOR(arglist, identityvalue) returns a truth-value
   /* identityvalue is TRUE for AND, and is FALSE for OR. */
   if (arglist == NIL)
      then return identityvalue
   if ( FIRST(arglist) == NOT(identityvalue) )
      then return NOT(identityvalue)
   return ANDOR( REST(arglist), identityvalue )
```

So: AND() evaluates to TRUE and OR() evaluates to FALSE!

Side Trip: We only need one logical connective.
(Note: AND, OR, NOT are "syntactic sugar" in logic.)

Both NAND and NOR are logically complete.

- **NAND is also called the "Sheffer stroke"**
- **NOR is also called "Pierce's arrow"**

(NOT A) = (NAND A TRUE) = (NOR A FALSE)

(AND A B) = (NAND TRUE (NAND A B))
= (NOR (NOR A FALSE) (NOR B FALSE))

(OR A B) = (NAND (NAND A TRUE) (NAND B TRUE))
=(NOR FALSE (NOR A B))

This fact is exploited by, e.g., VLSI semiconductor fabrication, which often provide a single NAND/NOR gate for efficiency.

Review: KB |= S means |= (KB ⇒ S)

- KB |= S is read "KB entails S."
  - Means "S is true in every world (model) in which KB is true."
- KB |= S is equivalent to |= (KB ⇒ S)
  - |= (KB ⇒ S) means "(KB ⇒ S) is true in every world (i.e., is valid)."
  - |= (KB ⇒ S) means TRUE |= (KB ⇒ S) means { } |= (KB ⇒ S)
- And so: { } |= S is equivalent to |= ({ } ⇒ S)
- So what does ({ } ⇒ S) mean?
  - Means "TRUE implies S."
  - Means "S is valid."
  - In Horn form, means "S is a fact." p. 256 (3rd ed.; p. 281, 2nd ed.)
- **Why does { } mean TRUE here,**
  but () means FALSE in resolution proofs?

Review: (TRUE $\Rightarrow$ S) means "S is a fact."

- By convention,
  - The null conjunct is "syntactic sugar" for TRUE (see above slides).
  - The null disjunct is "syntactic sugar" for FALSE (see following slides).
  - **Each is assigned the truth value of its identity element.**
    - For conjuncts, TRUE is the identity: (A $\wedge$ TRUE) $\equiv$ A
    - For disjuncts, FALSE is the identity: (A $\vee$ FALSE) $\equiv$ A

- A KB is the conjunction of all of its sentences.
  - So we see that { } is the null conjunct and means TRUE.
  - Better way to think of it: **{ }** does not **exclude** any worlds (models) because nothing **falsifies** the dominant connective **AND**.

- In Conjunctive Normal Form each clause is a disjunct.
  - So we see that ( ) is the null disjunct and means FALSE.
  - Better way to think of it: **()** does not **include** any worlds (models) because nothing **satisfies** the dominant connective **OR**.