TT-ENTAILS: Inference by Enumeration in Propositional Logic

CSE 4308/5360 – Artificial Intelligence I
Vassilis Athitsos
University of Texas at Arlington
Entailment and Inference

• To say that a knowledge base $\textbf{KB}$ entails a statement $\textbf{alpha}$ means simply that:

$\textbf{KB} \Rightarrow \textbf{alpha}.$

• Alternative (but equivalent) definition: a knowledge base $\textbf{KB}$ entails a statement $\textbf{alpha}$ if and only if:
  – In every world where $\textbf{KB}$ is true, $\textbf{alpha}$ is also true.

• This definition is compatible with our intuition.
  – $\textbf{KB} \Rightarrow \textbf{alpha}$ means that if $\textbf{KB}$ is true (thus, if we live in a world where $\textbf{KB}$ is true) then $\textbf{alpha}$ is true.
  – If there exists a world where $\textbf{KB}$ is true and $\textbf{alpha}$ is false, then clearly we cannot say that $\textbf{KB} \Rightarrow \textbf{alpha}.$
Worlds in Propositional Logic

- A knowledge base **KB** entails a statement **alpha** if and only if:
  - In every world where **KB** is true, **alpha** is also true.
- In the above definition, we use the term “world”.
- What is a “world” in propositional logic?
- Equivalent definitions of the term “world”:
  - A world is a row in the truth table.
  - A world is an assignment of boolean values to all symbols.
Inference by Enumeration

• In propositional logic, to determine if $KB$ entails $\alpha$, we apply an algorithm called “inference by enumeration”.

• Inference by enumeration is a “smoking-gun” algorithm.

• What is a “smoking gun” algorithm?
  – You ask a question to the algorithm.
  – The algorithm does a loop, searching for a smoking gun.
  – If it finds a smoking gun, it returns one answer.
  – If it finds no smoking gun, it returns another answer.
Inference by Enumeration

- Inference by enumeration as a “smoking gun” algorithm:
  - You ask a question to the algorithm. What question?
  - The algorithm does a loop, searching for a smoking gun. What would be a smoking gun?
  - If it finds a smoking gun, it returns one answer. What answer?
  - If it finds no smoking gun, it returns another answer. What answer?
Inference by Enumeration

• Inference by enumeration as a “smoking gun” algorithm:
  – You ask a question to the algorithm. What question?
    • Does KB entail alpha?
  – The algorithm does a loop, searching for a smoking gun. What would be a smoking gun?
    – If it finds a smoking gun, it returns one answer. What answer?
    – If it finds no smoking gun, it returns another answer. What answer?
Inference by Enumeration

• Inference by enumeration as a “smoking gun” algorithm:
  – You ask a question to the algorithm. What question?
    • Does KB entail alpha?
  – The algorithm does a loop, searching for a smoking gun. What would be a smoking gun?
    • A row in the truth table where KB is true and alpha is false.
  – If it finds a smoking gun, it returns one answer. What answer?
  – If it finds no smoking gun, it returns another answer. What answer?
Inference by Enumeration

• Inference by enumeration as a “smoking gun” algorithm:
  – You ask a question to the algorithm. What question?
    • Does KB entail alpha?
  – The algorithm does a loop, searching for a smoking gun. What would be a smoking gun?
    • A row in the truth table where KB is true and alpha is false.
  – If it finds a smoking gun, it returns one answer. What answer?
    • False (KB does NOT entail alpha).
  – If it finds no smoking gun, it returns another answer. What answer?
Inference by Enumeration

• Inference by enumeration as a “smoking gun” algorithm:
  – You ask a question to the algorithm. What question?
    • Does KB entail alpha?
  – The algorithm does a loop, searching for a smoking gun. What would be a smoking gun?
    • A row in the truth table where KB is true and alpha is false.
  – If it finds a smoking gun, it returns one answer. What answer?
    • False (KB does NOT entail alpha).
  – If it finds no smoking gun, it returns another answer. What answer?
    • True (KB entails alpha).
Inference by Enumeration

• Inference by enumeration algorithm:
  – For each row R in the truth table:
    • If KB is true in R and alpha is false in R, return false.
  – Return true.

• This is what you will have to implement in a subsequent assignment (assignment 4 or 5).

• Seems pretty simple, a three-line piece of pseudocode.

• Problem:
Inference by Enumeration

• Inference by enumeration algorithm:
  – For each row R in the truth table:
    • If $\textbf{KB}$ is true in R and $\textbf{alpha}$ is false in R, return false.
  – Return true.

• This is what you will have to implement in a subsequent assignment (assignment 4 or 5).
• Seems pretty simple, a three-line piece of pseudocode.
• Problem: how do you loop through rows of the truth table?
Inference by Enumeration

• Inference by enumeration algorithm:
  – For each row R in the truth table:
    • If KB is true in R and alpha is false in R, return false.
  – Return true.

• This is what you will have to implement in a subsequent assignment (assignment 4 or 5).

• Seems pretty simple, a three-line piece of pseudocode.

• Problem: how do you loop through rows of the truth table?
  – Answer: the TT-Entails pseudocode.
The TT-Entails Pseudocode

Boolean TT-Entails?(LogicalExpression KB, LogicalExpression alpha)
    List symbols1 = ExtractSymbols(KB);
    List symbols2 = ExtractSymbols(alpha);
    List symbols = concatenate(symbols1, symbols2);
    return TT-Check-All(KB, alpha, symbols, [ ])

Boolean TT-Check-All(LogicalExpression KB, LogicalExpression alpha,
    List symbols, Map model)
    if Empty?(symbols):
        if PL-True?(KB,model) then return PL-True?(alpha, model)
        else return true
    else:
        P = First(symbols);
        rest = Rest(symbols);
        return TT-Check-All(KB, alpha, rest, Extend(P, true, model)) and
            TT-Check-All(KB, alpha, rest, Extend(P, false, model))
Making Sense of TT-Entails

• Contrast the TT-Entails pseudocode to the previous pseudocode we saw for inference by enumeration:
  – For each row R in the truth table:
    • If KB is true in R and alpha is false in R, return false.
  – Return true.

• The two pseudocodes look very different.

• However, they do EXACTLY THE SAME THING.
  – It typically takes an entire lecture to convince people of that.

• TT-Entails provides a specific (but complicated) way to loop through rows in the truth table.
What We Need to Specify

Boolean TT-Entails? (LogicalExpression KB, LogicalExpression alpha)
  List symbols1 = ExtractSymbols(KB);
  List symbols2 = ExtractSymbols(alpha);
  List symbols = concatenate(symbols1, symbols2);
  return TT-Check-All(KB, alpha, symbols, [ ])

Boolean TT-Check-All(LogicalExpression KB, LogicalExpression alpha, 
  List symbols, Map model)
  if Empty?(symbols):
    if PL-True?(KB, model) then return PL-True?(alpha, model)
    else return true
  else:
    P = First(symbols);
    rest = Rest(symbols);
    return TT-Check-All(KB, alpha, rest, Extend(P, true, model)) and 
    TT-Check-All(KB, alpha, rest, Extend(P, false, model))

• To understand TT-Entails, we must understand the items in red.
• Hard ones: LogicalExpression, ExtractSymbols, PL-True?
• Easy ones: concatenate, First, Rest, Extend
Implementing a Logical Expression

Boolean TT-Entails?(LogicalExpression KB, LogicalExpression alpha)

• We will treat all these terms as synonyms and use interchangeably:
  – (Logical) expression.
  – (Logical) statement.
  – Sentence.

• Is it reasonable that KB is the same data type as alpha?
  – After all, KB is a **set** of statements, whereas alpha is a single statement.
Implementing a Logical Expression

We will treat all these terms as synonyms and use interchangeably:

- (Logical) expression.
- (Logical) statement.
- Sentence.

Is it reasonable that KB is the same data type as alpha?
- After all, KB is a set of statements, whereas alpha is a single statement.

Yes! KB is simply the conjunction of all the statements it contains.
Implementing a Logical Expression

Boolean $\text{TT-Entails?(} \text{LogicalExpression} \ \text{KB}, \text{LogicalExpression} \ \text{alpha})$

- What is a good data structure for a sentence in propositional logic?
Implementing a Logical Expression

What is a good data structure for a sentence in propositional logic?

Answer: a tree.

As an aside, trees are the typical choice for representing content specified in some language, such as:

- Logical statements, expressed in some logical language.
- Programs, expressed in a programming language (compilers convert them to tree representations in the process of compiling them).
- Text written in a natural language, such as English (ever heard of a parse tree?).
Trees

• How do we implement a tree as a class?
Trees

• How do we implement a tree as a class?
• A tree is a recursive data structure.
• There is no difference between the data structure for a tree and the data structure for a node in the tree.
• A tree is represented by its root.
Trees

class Tree
{
    Whatever content; // Content of the node (root of the tree)
    Tree[] children;   // Children of the node.
}

• Again, remember that there is no distinction (in programming) between the data type for a tree and the data type for a node.
• A tree is simply its root.
• The root knows about its children, which know about their children, ...
• Children can be implemented as an array or a list.
class Tree
{
    Whatever content; // Content of the node (root of the tree)
    Tree[] children;   // Children of the node.
}

• The tree is a recursive data structure.
  – Overall, programming with logic will be a big exercise in recursion.
• What is recursive about it?
• Answer: it is a data type that uses itself in its definition.
  – The children of the tree are an array (or list) of trees.
In any recursive definition, we must identify the base case(s).

What is the base case here?
class Tree
{
    Whatever content; // Content of the node (root of the tree)
    Tree[] children;   // Children of the node.
}

• In any recursive definition, we must identify the base case(s).
• What is the base case here?
  – A leaf.
• How is a leaf represented?
class Tree
{
    Whatever content; // Content of the node (root of the tree)
    Tree[] children; // Children of the node.
}

• In any recursive definition, we must identify the base case(s).
• What is the base case here?
  – A leaf.
• How is a leaf represented?
  – No children (empty array or list of children, or you can make children a NULL pointer).
Logical Expressions as Trees

• What are the building blocks of sentences in propositional logic?
Logical Expressions as Trees

• What are the building blocks of sentences in propositional logic?
  – Symbols.
  – Connectives.

• To figure out how to represent logical expressions as trees, we should start with the base case.

• Which logical expressions are leaves?
Logical Expressions as Trees

• What are the building blocks of sentences in propositional logic?
  – Symbols.
  – Connectives.

• To figure out how to represent logical expressions as trees, we should start with the base case.

• Which logical expressions are leafs?
  – Symbols.

• Then, how do we represent a connective?
Logical Expressions as Trees

• What are the building blocks of sentences in propositional logic?
  – Symbols.
  – Connectives.

• To figure out how to represent logical expressions as trees, we should start with the base case.

• Which logical expressions are leafs?
  – Symbols.

• Then, how do we represent a connective?

• A connective is a tree node.
  – The sentences it connects are the children of the node.
Example

• How do we translate this statement in English?

B12 <=> (P11 OR P22 OR P13)
Example

• How do we translate this statement in English?

\[ B_{12} \iff (P_{11} \lor P_{22} \lor P_{13}) \]

• There is a breeze at square 1,2 if and only if there is:
a pit at square 1,1, or a pit at square 2,2, or a pit at square 1,3.
Example

B12 <=> (P11 OR P22 OR P13)

• How do we represent this statement as a tree?
• What goes to the root? What is the top-level connective?
Example

B12 <=> (P11 OR P22 OR P13)

• How do we represent this statement as a tree?
• What goes to the root? What is the top-level connective?
• The root contains connective <=> as content.
• What are the children of the root?
Example

B12 <=> (P11  OR  P22 OR P13)

• How do we represent this statement as a tree?
• What goes to the root? What is the top-level connective?
• The root contains connective <=> as content.
• What are the children of the root?
  – B12 on the left.
  – (P11  OR  P22 OR P13) on the right.
• How do we represent (P11  OR  P22 OR P13)?
Example

B12 $\iff (P_{11} \ OR \ P_{22} \ OR \ P_{13})$

- How do we represent this statement as a tree?
- What goes to the root? What is the top-level connective?
- The root contains connective $\iff$ as content.
- What are the children of the root?
  - B12 on the left.
  - $(P_{11} \ OR \ P_{22} \ OR \ P_{13})$ on the right.
- How do we represent $(P_{11} \ OR \ P_{22} \ OR \ P_{13})$?
  - It is a node with:
    - OR as the connective
- How do we represent B12, P11, P22, P13?
Example

B12 <=> (P11 OR P22 OR P13)

• How do we represent this statement as a tree?
• What goes to the root? What is the top-level connective?
• The root contains connective <=> as content.
• What are the children of the root?
  – B12 on the left.
  – (P11 OR P22 OR P13) on the right.
• How do we represent (P11 OR P22 OR P13)?
• It is a node with:
  – OR as the connective
• How do we represent B12, P11, P22, P13?
  – They are leaf nodes.
Example

\[ \text{B12 } \iff (\text{P11 OR P22 OR P13}) \]

- The above statement becomes:

\[ \iff \]

\[ \begin{align*}
\text{B12} & \quad \text{OR} \\
\text{P11} & \quad \text{P22} \quad \text{P13}
\end{align*} \]
The LogicalExpression Class

• This is the Tree class from a previous slide:

```java
class Tree {
    Whatever content; // Content of the node (root of the tree)
    Tree[] children;  // Children of the node.
}
```

• How can we convert this to represent logical expressions?
The LogicalExpression Class

• This is the Tree class from a previous slide:
  
  ```java
class Tree {
    Whatever content; // Content of the node (root of the tree)
    Tree[] children; // Children of the node.
  }
```

• How can we convert this to represent logical expressions?
  
  ```java
class LogicalExpression {
    String symbol;
    String connective;
    LogicalExpression[] children;
  }
```
The LogicalExpression Class

class LogicalExpression
{
    String symbol;
    String connective;
    LogicalExpression[] children;
}

• The **symbol** and **connective** member variables are the content of the tree node.
  – Of course, in practice, one of the two has to be NULL. The node cannot contain both a symbol and a connective.

• If the sentence is a symbol, then:
  – symbol = ???
  – connective = ???
  – children = ???
The LogicalExpression Class

class LogicalExpression
{
    String symbol;
    String connective;
    LogicalExpression[] children;
}

• The **symbol** and **connective** member variables are the content of the tree node.
  – Of course, in practice, one of the two has to be NULL. The node cannot contain both a symbol and a connective.

• If the sentence is a symbol, then:
  – symbol = the symbol (very straightforward)
  – connective = NULL
  – children = NULL (or empty array, whatever you prefer).
class LogicalExpression
{
    String symbol;
    String connective;
    LogicalExpression[] children;
}

• The **symbol** and **connective** member variables are the content of the tree node.
  – Of course, in practice, one of the two has to be NULL. The node cannot contain both a symbol and a connective.

• If the sentence is NOT a symbol, then:
  – symbol = ???
  – connective = ???
  – children = ???
The LogicalExpression Class

class LogicalExpression {

    String symbol;
    String connective;
    LogicalExpression[] children;

}

• The symbol and connective member variables are the content of the tree node.
  – Of course, in practice, one of the two has to be NULL. The node cannot contain both a symbol and a connective.

• If the sentence is NOT a symbol, then:
  – symbol = NULL
  – connective = the top-level connective
  – children = the sentences that the connective connects.
At this point, we have established what the LogicalExpression data type is.
Next task: implementing ExtractSymbols.

Boolean TT-Entails?(LogicalExpression KB, LogicalExpression alpha)
  List symbols1 = ExtractSymbols(KB);
  List symbols2 = ExtractSymbols(alpha);
  List symbols = concatenate(symbols1, symbols2);
  return TT-Check-All(KB, alpha, symbols, [ ])

Boolean TT-Check-All(LogicalExpression KB, LogicalExpression alpha, List symbols, Map model)
  if Empty?(symbols):
    if PL-True?(KB,model) then return PL-True?(alpha, model)
    else return true
  else:
    P = First(symbols);
    rest = Rest(symbols);
    return TT-Check-All(KB, alpha, rest, Extend(P, true, model)) and
    TT-Check-All(KB, alpha, rest, Extend(P, false, model))
ExtractSymbols

• Next task: implementing ExtractSymbols.
• Arguments: ???
• Return type: ???
• Task: ???
ExtractSymbols

• Next task: implementing ExtractSymbols.
• Arguments: one argument, sentence, of type LogicalExpression.
  – In other words, it is a sentence (or statement) in propositional logic.
• Return type: ???
• Task: ???
ExtractSymbols

• Next task: implementing ExtractSymbols.
• Arguments: one argument, sentence, of type LogicalExpression.
  – In other words, it is a sentence (or statement) in propositional logic.
• Return type: a list of strings.
• Task: ???
ExtractSymbols

- Next task: implementing ExtractSymbols.
- Arguments: one argument, sentence, of type LogicalExpression.
  - In other words, it is a sentence (or statement) in propositional logic.
- Return type: a list of strings.
- Task: Return the list of all symbols appearing in sentence.
List<string> ExtractSymbols(LogicalExpression sentence)
    List result = empty list;
    if (sentence.symbol != NULL):
        ???
    else:
        ???
List<string> ExtractSymbols(LogicalExpression sentence)
    List result = empty list;
    if (sentence.symbol != NULL):
        result.add(sentence.symbol);
    else:
        ???
This is a classic example of how recursion makes life simple.

A few lines of pseudocode.

They translate into a few lines in real code.

They can process an arbitrarily long and complicated sentence in propositional logic.
ExtractSymbols

```csharp
List<string> ExtractSymbols(LogicalExpression sentence)
{
    List result = empty list;
    if (sentence.symbol != NULL):
        result.add(sentence.symbol);
    else:
        for each child in sentence.children:
            result = concatenate(result, ExtractSymbols(child));
    return result;
}
```

- What is the base case?
ExtractSymbols

List<string> ExtractSymbols(LogicalExpression sentence)
    List result = empty list;
    if (sentence.symbol != NULL):
        result.add(sentence.symbol);
    else:
        for each child in sentence.children:
            result = concatenate(result, ExtractSymbols(child));
    return result;

• What is the base case?
  – A sentence that is a symbol.

• How is it handled?
What is the base case?
- A sentence that is a symbol.

How is it handled?
- We return a list of that symbol.

Why do we need to return a list, and not just the symbol?
What is the base case?
  – A sentence that is a symbol.

How is it handled?
  – We return a list of that symbol.

Why do we need to return a list, and not just the symbol?
  – Because ExtractSymbols returns a list of strings.

```
List<string> ExtractSymbols(LogicalExpression sentence)
{
    List result = empty list;
    if (sentence.symbol != NULL):
        result.add(sentence.symbol);
    else:
        for each child in sentence.children:
            result = concatenate(result, ExtractSymbols(child));
    return result;
}
```
List<string> ExtractSymbols(LogicalExpression sentence)
    List result = empty list;
    if (sentence.symbol != NULL):
        result.add(sentence.symbol);
    else:
        for each child in sentence.children:
            result = concatenate(result, ExtractSymbols(child));
    return result;

• What is the recursive case?
What is the recursive case?
- A sentence that is NOT a symbol.

How is it handled?
ExtractSymbols

```csharp
List<string> ExtractSymbols(LogicalExpression sentence) {
    List result = empty list;
    if (sentence.symbol != NULL):
        result.add(sentence.symbol);
    else:
        for each child in sentence.children:
            result = concatenate(result, ExtractSymbols(child));
    return result;
}
```

- What is the recursive case?
  - A sentence that is NOT a symbol.

- How is it handled?
  - We get the results of calling ExtractSymbols on all the children.
  - We concatenate those results.
Back to TT-Entails

Boolean TT-Entails?(LogicalExpression KB, LogicalExpression alpha)
List symbols1 = ExtractSymbols(KB);
List symbols2 = ExtractSymbols(alpha);
List symbols = concatenate(symbols1, symbols2);
return TT-Check-All(KB, alpha, symbols, [ ])

Boolean TT-Check-All(LogicalExpression KB, LogicalExpression alpha, List symbols, Map model)
if Empty?(symbols):
    if PL-True?(KB, model) then return PL-True?(alpha, model)
    else return true
else:
    P = First(symbols);
    rest = Rest(symbols);
    return TT-Check-All(KB, alpha, rest, Extend(P, true, model)) and
    TT-Check-All(KB, alpha, rest, Extend(P, false, model))

• Next task: implementing PL-True?
PL-True?

- Next task: implementing PL-True?.
- Arguments:
  - LogicalExpression sentence.
  - Map<String, Boolean> model.
- Return type: Boolean.
- Task: determine if sentence is true or false in the row of the truth table represented by model.
- From the above, it should be clear that model represents a row in the truth table.
- What kind of data structure is good for that?
PL-True?

- Next task: implementing PL-True?.
- Arguments:
  - LogicalExpression sentence.
  - Map<String, Boolean> model.
- Return type: Boolean.
- Task: determine if sentence is true or false in the row of the truth table represented by model.
- From the above, it should be clear that model represents a row in the truth table.
- What kind of data structure is good for model?
- Any kind of association map (like dictionaries in Python, HashMaps in Java, stl::map in C++) that can map symbols to boolean values.
Boolean PL-True?(LogicalExpression sentence, Map model)
   if sentence.symbol != NULL:
      ???
Boolean PL-True?(LogicalExpression sentence, Map model)
    if sentence.symbol != NULL:
        return model[sentence.symbol];
    else ???
Boolean PL-True?(LogicalExpression sentence, Map model)
    if sentence.symbol != NULL:
        return model[sentence.symbol];
    else if sentence.connective == "and":
        ???
Boolean PL-True?(LogicalExpression sentence, Map model)
if sentence.symbol != NULL:
    return model[sentence.symbol];
else if sentence.connective == "and":
    for each child in sentence.children:
        if (PL-True(child, model) == false):
            return false;
    return true;
else if sentence.connective == "or":
    ???
Boolean PL-True?(LogicalExpression sentence, Map model)
    if sentence.symbol != NULL:
        return model[sentence.symbol];

    else if sentence.connective == "and":
        for each child in sentence.children:
            if (PL-True(child, model) == false):
                return false;
        return true;

    else if sentence.connective == "or":
        for each child in sentence.children:
            if (PL-True(child, model) == true):
                return true;
        return false;
Boolean PL-True?(LogicalExpression sentence, Map model)
   if sentence.symbol != NULL:
      return model[sentence.symbol];
   else if sentence.connective == "and":
      for each child in sentence.children:
         if (PL-True(child, model) == false):
            return false;
      return true;
   else if sentence.connective == "or":
      for each child in sentence.children:
         if (PL-True(child, model) == true):
            return true;
      return false;

and so on.

• How do we handle "if"?
Boolean PL-True?(LogicalExpression sentence, Map model)
    if sentence.symbol != NULL:
        return model[sentence.symbol];
    else if sentence.connective == "and":
        for each child in sentence.children:
            if (PL-True(child, model) == false):
                return false;
        return true;
    else if sentence.connective == "or":
        for each child in sentence.children:
            if (PL-True(child, model) == true):
                return true;
        return false;
and so on.

• How do we handle "if"?
• left = sentence.children[0];
• right = sentence.children[1];
• Return false if:
  – PL-True(left, model) == true
  – PL-True(right, model) == false
• Return true otherwise.
Boolean PL-True?(LogicalExpression sentence, 
Map model)
    if sentence.symbol != NULL:
        return model[sentence.symbol];
    else if sentence.connective == "and":
        for each child in sentence.children:
            if (PL-True(child, model) == false):
                return false;
        return true;
    else if sentence.connective == "or":
        for each child in sentence.children:
            if (PL-True(child, model) == true):
                return true;
        return false;
and so on.

- How do we handle "iff"?
Boolean PL-True?(LogicalExpression sentence, Map model)
    if sentence.symbol != NULL:
        return model[sentence.symbol];
    else if sentence.connective == "and":
        for each child in sentence.children:
            if (PL-True(child, model) == false):
                return false;
        return true;
    else if sentence.connective == "or":
        for each child in sentence.children:
            if (PL-True(child, model) == true):
                return true;
        return false;

and so on.

• How do we handle "iff"?
• left = sentence.children[0];
• right = sentence.children[1]
• Return true if:
  – PL-True(left, model) and PL-True(right, model) return the same thing.
• Return false otherwise.
Boolean PL-True?(LogicalExpression sentence, Map model)
  if sentence.symbol != NULL:
    return model[sentence.symbol];

  else if sentence.connective == "and":
    for each child in sentence.children:
      if (PL-True(child, model) == false):
        return false;
      return true;

  else if sentence.connective == "or":
    for each child in sentence.children:
      if (PL-True(child, model) == true):
        return true;
      return false;

and so on.

• How do we handle "not"?
• child = sentence.children[0];
• Return the opposite of what PL-True?(child, model) returns.
Back to TT-Entails

Boolean TT-Entails?(LogicalExpression KB, LogicalExpression alpha)
  List symbols1 = ExtractSymbols(KB);
  List symbols2 = ExtractSymbols(alpha);
  List symbols = concatenate(symbols1, symbols2);
  return TT-Check-All(KB, alpha, symbols, [ ])

Boolean TT-Check-All(LogicalExpression KB, LogicalExpression alpha,
  List symbols, Map model)
  if Empty?(symbols):
    if PL-True?(KB,model) then return PL-True?(alpha, model)
    else return true
  else:
    P = First(symbols);
    rest = Rest(symbols);
    return TT-Check-All(KB, alpha, rest, Extend(P, true, model)) and
    TT-Check-All(KB, alpha, rest, Extend(P, false, model))

• Next task: understanding what TT-Entails actually does.
TT-Entails goes through the truth table.

- If it finds a smoking gun (any row where KB is true and alpha is false), it returns false.
- If it goes through all the rows in the truth table and does NOT find a smoking gun, it returns true.
- Clearly, the actual work is done by TT-Check-All.
- So, to understand TT-Entails, we must understand TT-Check-All.

Boolean TT-Entails?( LogicalExpression KB, LogicalExpression alpha)
List symbols1 = ExtractSymbols(KB);
List symbols2 = ExtractSymbols(alpha);
List symbols = concatenate(symbols1, symbols2);
return TT-Check-All(KB, alpha, symbols, [ ])

75
TT-Check-All

Boolean TT-Entails?(LogicalExpression KB, LogicalExpression alpha)
  List symbols1 = ExtractSymbols(KB);
  List symbols2 = ExtractSymbols(alpha);
  List symbols = concatenate(symbols1, symbols2);
  return TT-Check-All(KB, alpha, symbols, [ ])

Boolean TT-Check-All(LogicalExpression KB, LogicalExpression alpha, 
  List symbols, Map model)

• TT-Check-All arguments:
  – KB and alpha, the same arguments as in TT-Entails.
  – model: this is an association map, mapping symbols to true/false 
    values.
  – symbols: this is a list of symbols.
TT-Check-All: Model and Symbols

Boolean TT-Entails?(LogicalExpression KB, LogicalExpression alpha)
  List symbols1 = ExtractSymbols(KB);
  List symbols2 = ExtractSymbols(alpha);
  List symbols = concatenate(symbols1, symbols2);
  return TT-Check-All(KB, alpha, symbols, [ ])

Boolean TT-Check-All(LogicalExpression KB, LogicalExpression alpha,
  List symbols, Map model)

• What is the value of **model** when TT-Check-All gets called from TT-Entails?
  – It is empty. In other words, it does not map any symbol to a value.

• What is the value of **symbols** when TT-Check-All gets called from TT-Entails?
  – It is the list of all symbols appearing in KB or appearing in alpha.
TT-Check-All

Boolean TT-Check-All(LogicalExpression KB, LogicalExpression alpha, List symbols, Map model)

if Empty?(symbols):
    if PL-True?(KB,model) then return PL-True?(alpha, model)
    else return true
else:
    P = First(symbols);
    rest = Rest(symbols);
    return TT-Check-All(KB, alpha, rest, Extend(P, true, model)) and
    TT-Check-All(KB, alpha, rest, Extend(P, false, model))

• As you see, TT-Check-All calls itself.
  – Each time it calls itself, a symbol is removed from symbols, and it is assigned a value in model.
• Overall, argument symbols is the list of symbols that still do not have a value in model.
• The model is empty initially, and gets extended at each recursive call.
What TT-Check-All Does

The task of TT-Check-All is to look for a smoking gun in all rows of the truth table compatible with its model.
  - If it finds a smoking gun, it returns false.
  - Otherwise, it returns true.

When is a row of the truth table compatible with the model?
### What TT-Check-All Does

The task of TT-Check-All is to look for a smoking gun in all rows of the truth table **compatible with its model**.
- If it finds a smoking gun, it returns false.
- Otherwise, it returns true.

When is a row of the truth table compatible with the model?
- A row of the truth table can also be represented as a Map. It assigns values to symbols.
- If the assignments in the row of the truth table do not contradict any assignments in the model, then the row is compatible with the model.

```plaintext
Boolean TT-Entails?(LogicalExpression KB, LogicalExpression alpha)
    List symbols1 = ExtractSymbols(KB);
    List symbols2 = ExtractSymbols(alpha);
    List symbols = concatenate(symbols1, symbols2);
    return TT-Check-All(KB, alpha, symbols, [])

Boolean TT-Check-All(LogicalExpression KB, LogicalExpression alpha,
    List symbols, Map model)
```
What TT-Check-All Does

The task of TT-Check-All is to look for a smoking gun in all rows of the truth table compatible with its model. 
- If it finds a smoking gun (a row where KB is true and alpha is false), it returns false.
- Otherwise, it returns true.

What should model be equal to, to make TT-Check-All search through the entire truth table?
What TT-Check-All Does

• The task of TT-Check-All is to look for a smoking gun in all rows of the truth table **compatible with its model**.
  – If it finds a smoking gun (a row where KB is true and alpha is false), it returns false.
  – Otherwise, it returns true.

• What should model be equal to, to make TT-Check-All search through the entire truth table?
  – An empty model is compatible with every row in the truth table.
What TT-Check-All Does

Boolean TT-Entails?(LogicalExpression KB, LogicalExpression alpha)
    List symbols1 = ExtractSymbols(KB);
    List symbols2 = ExtractSymbols(alpha);
    List symbols = concatenate(symbols1, symbols2);
    return TT-Check-All(KB, alpha, symbols, [ ]) 

Boolean TT-Check-All(LogicalExpression KB, LogicalExpression alpha,
    List symbols, Map model)

- The task of TT-Check-All is to look for a smoking gun in all rows of the truth table **compatible with its model**.
- When it is first called from TT-Entails, TT-Check-All needs to check the entire truth table.
- Therefore, the **model** argument in that first call is the empty model, shown as [].
- Why? Because the empty model is compatible with all rows in the truth table.
What TT-Check-All Does

Boolean TT-Check-All(LogicalExpression KB, LogicalExpression alpha, List symbols, Map model)

if Empty?(symbols):
    if PL-True?(KB, model) then return PL-True?(alpha, model)
    else return true
else:
    P = First(symbols);
    rest = Rest(symbols);
    return TT-Check-All(KB, alpha, rest, Extend(P, true, model)) and
    TT-Check-All(KB, alpha, rest, Extend(P, false, model))

• Let’s look at what TT-Check-All does when called from TT-Entails:
  – The **model** argument is empty.
  – The **symbols** argument contains all symbols.

• What will Empty?(symbols) return?
What TT-Check-All Does

Boolean TT-Check-All(LogicalExpression KB, LogicalExpression alpha, List symbols, Map model)
  if Empty?(symbols):
    if PL-True?(KB,model) then return PL-True?(alpha, model)
    else return true
  else:
    P = First(symbols);
    rest = Rest(symbols);
    return TT-Check-All(KB, alpha, rest, Extend(P, true, model)) and
    TT-Check-All(KB, alpha, rest, Extend(P, false, model))

• Let’s look at what TT-Check-All does when called from TT-Entails:
  – The **model** argument is empty.
  – The **symbols** argument contains all symbols.

• What will Empty?(symbols) return? False.

• So, we move on to the else part.
What TT-Check-All Does

Boolean TT-Check-All(LogicalExpression KB, LogicalExpression alpha, List symbols, Map model)

if Empty?(symbols):
    if PL-True?(KB,model) then return PL-True?(alpha, model)
    else return true
else:
    P = First(symbols);
    rest = Rest(symbols);
    return TT-Check-All(KB, alpha, rest, Extend(P, true, model)) and
    TT-Check-All(KB, alpha, rest, Extend(P, false, model))

• Let’s look at what TT-Check-All does when called from TT-Entails:
  – The model argument is empty.
  – The symbols argument contains all symbols.
• P is set equal to one of the symbols (the first in the list, it does not really matter which one).
• Rest is set equal to the rest of the symbols (all symbols except for P).
What TT-Check-All Does

Boolean TT-Check-All(LogicalExpression KB, LogicalExpression alpha, List symbols, Map model)

if Empty?(symbols):
    if PL-True?(KB, model) then return PL-True?(alpha, model)
    else return true
else:
    P = First(symbols);
    rest = Rest(symbols);
    return TT-Check-All(KB, alpha, rest, Extend(P, true, model)) and 
    TT-Check-All(KB, alpha, rest, Extend(P, false, model))

• TT-Check-All calls itself twice.
What TT-Check-All Does

Boolean TT-Check-All(LogicalExpression KB, LogicalExpression alpha, List symbols, Map model)

  if Empty?(symbols):
    if PL-True?(KB,model) then return PL-True?(alpha, model)
    else return true
  else:
    P = First(symbols);
    rest = Rest(symbols);
    return TT-Check-All(KB, alpha, rest, Extend(P, true, model)) and
    TT-Check-All(KB, alpha, rest, Extend(P, false, model))

• In the first call:
  – For the third argument (symbols), it uses value rest. So, it passes all the symbols except for the first one (which was assigned to P).
  – For the fourth argument (model), it passes an extended model, which includes the assignments of the previous model, plus a new assignment: ??
What TT-Check-All Does

Boolean TT-Check-All(LogicalExpression KB, LogicalExpression alpha, List symbols, Map model)
    if Empty?(symbols):
        if PL-True?(KB,model) then return PL-True?(alpha, model)
        else return true
    else:
        P = First(symbols);
        rest = Rest(symbols);
        return TT-Check-All(KB, alpha, rest, Extend(P, true, model)) and 
         TT-Check-All(KB, alpha, rest, Extend(P, false, model))

• In the first call:
  – For the third argument (symbols), it uses value rest. So, it passes all the symbols except for the first one (which was assigned to P).
  – For the fourth argument (model), it passes an extended model, which includes the assignments of the previous model, plus a new assignment: P=true.
What TT-Check-All Does

Boolean TT-Check-All(LogicalExpression KB, LogicalExpression alpha, List symbols, Map model)

if Empty?(symbols):
    if PL-True?(KB,model) then return PL-True?(alpha, model)
    else return true
else:
    P = First(symbols);
    rest = Rest(symbols);
    return TT-Check-All(KB, alpha, rest, Extend(P, true, model)) and 
    TT-Check-All(KB, alpha, rest, Extend(P, false, model))

• In the second call:
  – For the third argument (symbols), it uses value rest (same as in the first call).
  – For the fourth argument (model), it passes an extended model, which includes the assignments of the previous model, plus a new assignment: P=false.
  – Note that in the first call P is set to true, in the second call P is set to false.
Prefix Notation

• To see how TT-Entails actually works, we should do an example.
• To make notation look like that of assignment 4, we will use prefix notation.
  – Instead of writing $A \text{ and } B \text{ and } C$ we will write (and $A B C$).
  – Instead of writing $A \text{ or } B \text{ or } C$ we will write (or $A B C$).
  – Instead of writing $\text{not } A$ we will write (not $A$).
  – Instead of writing $A \Rightarrow B$ we will write (if $A B$).
  – Instead of writing $A \Leftrightarrow B$ we will write (iff $A B$).
• In prefix notation, to write a statement that has a connective:
  – We write a left parenthesis.
  – We write the connective.
  – We write the statements that the connective connects.
  – We write a right parenthesis.
• Example: (and M_1_2 S_1_1 (not (or M_1_3 M_1_4))))
An Example

Boolean TT-Entails?(LogicalExpression KB, LogicalExpression alpha)
- List symbols1 = ExtractSymbols(KB);
- List symbols2 = ExtractSymbols(alpha);
- List symbols = concatenate(symbols1, symbols2);
- return TT-Check-All(KB, alpha, symbols, [ ])

Boolean TT-Check-All(LogicalExpression KB, LogicalExpression alpha, List symbols, Map model)

- Example KB:
  (iff B_1_1 (or P_1_2 P_2_1))
  B_1_1

- Example alpha:
  (not P_1_2)

- What is the knowledge base saying in English?
An Example

Boolean TT-Entails?(LogicalExpression KB, LogicalExpression alpha)
List symbols1 = ExtractSymbols(KB);
List symbols2 = ExtractSymbols(alpha);
List symbols = concatenate(symbols1, symbols2);
return TT-Check-All(KB, alpha, symbols, [ ])

Boolean TT-Check-All(LogicalExpression KB, LogicalExpression alpha,
List symbols, Map model)

• Example KB:
  (iff B_1_1 (or P_1_2 P_2_1))
  B_1_1
• Example alpha:
  (not P_1_2)

• What is the knowledge base saying in English?
  – There is a breeze at square (1, 1) if and only if: there is a pit at (1, 2), or there is a pit at (2, 1).
  – There is a breeze at square (1, 1).
Boolean TT-Entails?(LogicalExpression KB, LogicalExpression alpha)
List symbols1 = ExtractSymbols(KB);
List symbols2 = ExtractSymbols(alpha);
List symbols = concatenate(symbols1, symbols2);
return TT-Check-All(KB, alpha, symbols, [ ])

Boolean TT-Check-All(LogicalExpression KB, LogicalExpression alpha, List symbols, Map model)

• Example KB:
  (iff B_1_1 (or P_1_2 P_2_1))
  B_1_1

• Example alpha:
  (not P_1_2)

• What is the knowledge base saying in English?
  – There is a breeze at square (1, 1) if and only if: there is a pit at (1, 2), or there is a pit at (2, 1).
  – There is a breeze at square (1, 1).

• What does alpha say in English?
An Example

Boolean TT-Entails?(LogicalExpression KB, LogicalExpression alpha)
   List symbols1 = ExtractSymbols(KB);
   List symbols2 = ExtractSymbols(alpha);
   List symbols = concatenate(symbols1, symbols2);
   return TT-Check-All(KB, alpha, symbols, [ ])

Boolean TT-Check-All(LogicalExpression KB, LogicalExpression alpha,
   List symbols, Map model)

- **Example KB:**
  (iff B_1_1 (or P_1_2 P_2_1))
  B_1_1

- **Example alpha:**
  (not P_1_2)

- **What is the knowledge base saying in English?**
  
  - There is a breeze at square (1, 1) if and only if: there is a pit at (1, 2), or there is a pit at (2, 1).
  
  - There is a breeze at square (1, 1).

- **What does alpha say in English?**
  
  - There is no pit at (1, 2).
Boolean TT-Entails?(LogicalExpression KB, LogicalExpression alpha)
List symbols1 = ExtractSymbols(KB);
List symbols2 = ExtractSymbols(alpha);
List symbols = concatenate(symbols1, symbols2);
return TT-Check-All(KB, alpha, symbols, [ ])

Boolean TT-Check-All(LogicalExpression KB, LogicalExpression alpha, List symbols, Map model)

• Example KB:
  (iff B_1_1 (or P_1_2 P_2_1))
  B_1_1
• Example alpha:
  (not P_1_2)

• If we call TT-Entails? with these arguments, what are we asking the computer?
An Example

Boolean TT-Entails?(LogicalExpression KB, LogicalExpression alpha)
List symbols1 = ExtractSymbols(KB);
List symbols2 = ExtractSymbols(alpha);
List symbols = concatenate(symbols1, symbols2);
return TT-Check-All(KB, alpha, symbols, [ ])

Boolean TT-Check-All(LogicalExpression KB, LogicalExpression alpha, 
List symbols, Map model)

• Example KB:
  (iff B_1_1 (or P_1_2 P_2_1))
  B_1_1
• Example alpha:
  (not P_1_2)

• If we call TT-Entails? with these arguments, what are we asking the computer?
  – Given this knowledge base, can we infer that there is no pit at square (1,2)?

• Let’s see how TT-Entails works on this question.
An Example

Boolean TT-Entails?(LogicalExpression KB, LogicalExpression alpha)
List symbols1 = ExtractSymbols(KB);
List symbols2 = ExtractSymbols(alpha);
List symbols = concatenate(symbols1, symbols2);
return TT-Check-All(KB, alpha, symbols, [ ])

Boolean TT-Check-All(LogicalExpression KB, LogicalExpression alpha, List symbols, Map model)

• Example KB:
  (iff B_1_1 (or P_1_2 P_2_1))
  B_1_1
• Example alpha:
  (not P_1_2)

• symbols1 = ???
An Example

Boolean TT-Entails?(LogicalExpression KB, LogicalExpression alpha)
List symbols1 = ExtractSymbols(KB);
List symbols2 = ExtractSymbols(alpha);
List symbols = concatenate(symbols1, symbols2);
return TT-Check-All(KB, alpha, symbols, [ ])

Boolean TT-Check-All(LogicalExpression KB, LogicalExpression alpha, List symbols, Map model)

• Example KB:
  (iff B_1_1 (or P_1_2 P_2_1))
  B_1_1
• Example alpha:
  (not P_1_2)

• symbols1 = [B_1_1, P_1_2, P_2_1]
Boolean TT-Entails?(LogicalExpression KB, LogicalExpression alpha)
List symbols1 = ExtractSymbols(KB);
List symbols2 = ExtractSymbols(alpha);
List symbols = concatenate(symbols1, symbols2);
return TT-Check-All(KB, alpha, symbols, [ ])

Boolean TT-Check-All(LogicalExpression KB, LogicalExpression alpha,
List symbols, Map model)

• Example KB:
  (iff B_1_1 (or P_1_2 P_2_1))
  B_1_1
• Example alpha:
  (not P_1_2)

• symbols1 = [B_1_1, P_1_2, P_2_1]
• symbols2 = ???
An Example

Boolean TT-Entails?(LogicalExpression KB, LogicalExpression alpha)
List symbols1 = ExtractSymbols(KB);
List symbols2 = ExtractSymbols(alpha);
List symbols = concatenate(symbols1, symbols2);
return TT-Check-All(KB, alpha, symbols, [ ])

Boolean TT-Check-All(LogicalExpression KB, LogicalExpression alpha, List symbols, Map model)

• Example KB:
  (iff B_1_1 (or P_1_2 P_2_1))
  B_1_1
• Example alpha:
  (not P_1_2)

• symbols1 = [B_1_1, P_1_2, P_2_1]
• symbols2 = [P_1_2]
An Example

Boolean TT-Entails?(LogicalExpression KB, LogicalExpression alpha)
List symbols1 = ExtractSymbols(KB);
List symbols2 = ExtractSymbols(alpha);
List symbols = concatenate(symbols1, symbols2);
return TT-Check-All(KB, alpha, symbols, [ ])

Boolean TT-Check-All(LogicalExpression KB, LogicalExpression alpha, List symbols, Map model)

• Example KB:
  (iff B_1_1 (or P_1_2 P_2_1))
  B_1_1

• Example alpha:
  (not P_1_2)

• symbols1 = [B_1_1, P_1_2, P_2_1]
• symbols2 = [P_1_2]
• symbols = ???
An Example

Boolean TT-Entails?(LogicalExpression KB, LogicalExpression alpha)
  List symbols1 = ExtractSymbols(KB);
  List symbols2 = ExtractSymbols(alpha);
  List symbols = concatenate(symbols1, symbols2);
  return TT-Check-All(KB, alpha, symbols, [ ])

Boolean TT-Check-All(LogicalExpression KB, LogicalExpression alpha, List symbols, Map model)

- Example KB:
  (iff B_1_1 (or P_1_2 P_2_1))
  B_1_1
- Example alpha:
  (not P_1_2)

- symbols1 = [B_1_1, P_1_2, P_2_1]
- symbols2 = [P_1_2]
- symbols = [B_1_1, P_1_2, P_2_1]
An Example

Boolean TT-Entails?(LogicalExpression KB, LogicalExpression alpha)
List symbols1 = ExtractSymbols(KB);
List symbols2 = ExtractSymbols(alpha);
List symbols = concatenate(symbols1, symbols2);
return TT-Check-All(KB, alpha, symbols, [ ])

Boolean TT-Check-All(LogicalExpression KB, LogicalExpression alpha,
List symbols, Map model)

• Example KB:
  (iff B_1_1 (or P_1_2 P_2_1))
  B_1_1
• Example alpha:
  (not P_1_2)

• symbols1 = [B_1_1, P_1_2, P_2_1]
• symbols2 = [P_1_2]
• symbols = [B_1_1, P_1_2, P_2_1]
• We now call TT-Check-All.
An Example

Boolean TT-Check-All(KB, alpha, symbols, model)
  if Empty?(symbols):
    if PL-True?(KB,model) then return PL-True?(alpha, model)
    else return true
  else:
    P = First(symbols);
    rest = Rest(symbols);
    return TT-Check-All(KB, alpha, rest, Extend(P, true, model)) and
    TT-Check-All(KB, alpha, rest, Extend(P, false, model))

- Example KB:
  (iff B_1_1 (or P_1_2 P_2_1))
  B_1_1
- Example alpha:
  (not P_1_2)

- symbols = ???
- model = ???
An Example

Boolean TT-Check-All(KB, alpha, symbols, model)
   if Empty?(symbols):
      if PL-True?(KB,model) then return PL-True?(alpha, model)
      else return true
   else:
      P = First(symbols);
      rest = Rest(symbols);
      return TT-Check-All(KB, alpha, rest, Extend(P, true, model)) and
      TT-Check-All(KB, alpha, rest, Extend(P, false, model))

• Example KB:
  (iff B_1_1 (or P_1_2 P_2_1))
  B_1_1
• Example alpha:
  (not P_1_2)

• symbols = [B_1_1, P_1_2, P_2_1]
• model = [] (the empty model)
An Example

Boolean TT-Check-All(KB, alpha, symbols, model)
  if Empty?(symbols):
    if PL-True?(KB,model) then return PL-True?(alpha, model)
    else return true
  else:
    P = First(symbols);
    rest = Rest(symbols);
    return TT-Check-All(KB, alpha, rest, Extend(P, true, model)) and
    TT-Check-All(KB, alpha, rest, Extend(P, false, model))

• Example KB:
  (iff B_1_1 (or P_1_2 P_2_1))
  B_1_1
• Example alpha:
  (not P_1_2)

• symbols = [B_1_1, P_1_2, P_2_1]
• model = [] (the empty model)
• What is the task of this call to TT-Check-All?
  Which rows of the truth table should it check?
An Example

```plaintext
Boolean TT-Check-All(KB, alpha, symbols, model)
    if Empty?(symbols):
        if PL-True?(KB,model) then return PL-True?(alpha, model)
        else return true
    else:
        P = First(symbols);
        rest = Rest(symbols);
        return TT-Check-All(KB, alpha, rest, Extend(P, true, model)) and
        TT-Check-All(KB, alpha, rest, Extend(P, false, model))
```

- Example KB:
  (iff B_1_1 (or P_1_2 P_2_1))
  B_1_1
- Example alpha:
  (not P_1_2)
- symbols = [B_1_1, P_1_2, P_2_1]
- model = [] (the empty model)
- What is the task of this call to TT-Check-All?
  - Which rows of the truth table should it check?
  - All rows (since the model is empty).
An Example

Boolean TT-Check-All(KB, alpha, symbols, model)
   if Empty?(symbols):
      if PL-True?(KB,model) then return PL-True?(alpha, model)
      else return true
   else:
      P = First(symbols);
      rest = Rest(symbols);
      return TT-Check-All(KB, alpha, rest, Extend(P, true, model)) and
      TT-Check-All(KB, alpha, rest, Extend(P, false, model))

- Example KB:  
  (iff B_1_1 (or P_1_2 P_2_1)) 
  B_1_1 
- Example alpha:  
  (not P_1_2) 
- symbols = [B_1_1, P_1_2, P_2_1] 
- model = [] (the empty model) 
- if Empty?(symbols):  
  - What happens here?
An Example

Boolean TT-Check-All(KB, alpha, symbols, model)

if Empty?(symbols):
    if PL-True?(KB, model) then return PL-True?(alpha, model)
    else return true
else:
    P = First(symbols);
    rest = Rest(symbols);
    return TT-Check-All(KB, alpha, rest, Extend(P, true, model)) and
    TT-Check-All(KB, alpha, rest, Extend(P, false, model))

- Example KB: (iff B_1_1 (or P_1_2 P_2_1))
  B_1_1
- Example alpha: (not P_1_2)
- symbols = [B_1_1, P_1_2, P_2_1]
- model = [] (the empty model)
- if Empty?(symbols):
  - What happens here?
  - The condition is false, we move to the else part.
An Example

Boolean TT-Check-All(KB, alpha, symbols, model)
if Empty?(symbols):
    if PL-True?(KB,model) then return PL-True?(alpha, model)
    else return true
else:
    P = First(symbols);
    rest = Rest(symbols);
    return TT-Check-All(KB, alpha, rest, Extend(P, true, model)) and
    TT-Check-All(KB, alpha, rest, Extend(P, false, model))

• Example KB:
  (iff B_1_1 (or P_1_2 P_2_1))
  B_1_1
• Example alpha:
  (not P_1_2)

• symbols = [B_1_1, P_1_2, P_2_1]
• model = [] (the empty model)
• P = ???
An Example

Boolean TT-Check-All(KB, alpha, symbols, model)
   if Empty?(symbols):
      if PL-True?(KB, model) then return PL-True?(alpha, model)
      else return true
   else:
      P = First(symbols);
      rest = Rest(symbols);
      return TT-Check-All(KB, alpha, rest, Extend(P, true, model)) and
           TT-Check-All(KB, alpha, rest, Extend(P, false, model))

• Example KB:
  (iff B_1_1 (or P_1_2 P_2_1))
  B_1_1
• Example alpha:
  (not P_1_2)

• symbols = [B_1_1, P_1_2, P_2_1]
• model = [] (the empty model)
• P = B_1_1
An Example

Boolean TT-Check-All(KB, alpha, symbols, model)
   if Empty?(symbols):
      if PL-True?(KB,model) then return PL-True?(alpha, model)
      else return true
   else:
      P = First(symbols);
      rest = Rest(symbols);
      return TT-Check-All(KB, alpha, rest, Extend(P, true, model)) and
      TT-Check-All(KB, alpha, rest, Extend(P, false, model))

• Example KB:
  (iff B_1_1 (or P_1_2 P_2_1))
  B_1_1
• Example alpha:
  (not P_1_2)

• symbols = [B_1_1, P_1_2, P_2_1]
• model = [] (the empty model)
• P = B_1_1
• rest = ???
An Example

Boolean TT-Check-All(KB, alpha, symbols, model)
if Empty?(symbols):
    if PL-True?(KB,model) then return PL-True?(alpha, model)
    else return true
else:
    P = First(symbols);
    rest = Rest(symbols);
    return TT-Check-All(KB, alpha, rest, Extend(P, true, model)) and
    TT-Check-All(KB, alpha, rest, Extend(P, false, model))

• Example KB:
  (iff B_1_1 (or P_1_2 P_2_1))
  B_1_1
• Example alpha:
  (not P_1_2)

• symbols = [B_1_1, P_1_2, P_2_1]
• model = [] (the empty model)
• P = B_1_1
• rest = [P_1_2, P_2_1]
An Example

Boolean TT-Check-All(KB, alpha, symbols, model)
  if Empty?(symbols):
    if PL-True?(KB,model) then return PL-True?(alpha, model)
    else return true
  else:
    P = First(symbols);
    rest = Rest(symbols);
    return TT-Check-All(KB, alpha, rest, Extend(P , true, model)) and
    TT-Check-All(KB, alpha, rest, Extend(P, false, model))

- Example KB: (iff B_1_1 (or P_1_2 P_2_1))
  B_1_1
- Example alpha: (not P_1_2)
- symbols = [B_1_1, P_1_2, P_2_1]
- model = [] (the empty model)
- P = B_1_1
- rest = [P_1_2, P_2_1]
- How can we interpret this return statement?
When the model is empty, the job of TT-Check-All is to search the entire truth table for a smoking gun (a counterexample, where KB is true and alpha is false).

TT-Check-All divides this task into two:
- First function call: Check all rows in the truth table where B_1_1 = true.
- Second function call: Check all rows in the truth table where B_1_1 = false.

The final result is an AND of the results of the two function calls.
- If none of the two calls finds a smoking gun, we return true.
TT-Check-All(KB, alpha,
symbols = [B_1_1, P_1_2, P_2_1],
model = [ ])

TT-Check-All(KB, alpha,
symbols = [P_1_2, P_2_1],
model = [B_1_1 = T])

TT-Check-All(KB, alpha,
symbols = [P_2_1],
model = [B_1_1 = T, 
P_1_2 = T])

TT-Check-All(KB, alpha,
symbols = [P_1_2, P_2_1],
model = [B_1_1 = F])

TT-Check-All(KB, alpha,
symbols = [P_2_1],
model = [B_1_1 = T, 
P_1_2 = F])
TT-Check-All(KB, alpha, symbols = [B_1_1, P_1_2, P_2_1], model = [])

TT-Check-All(KB, alpha, symbols = [P_1_2, P_2_1], model = [B_1_1 = T])

TT-Check-All(KB, alpha, symbols = [P_1_2, P_2_1], model = [B_1_1 = F])

TT-Check-All(KB, alpha, symbols = ???, model = ??? )

TT-Check-All(KB, alpha, symbols = ???, model = ??? )
TT-Check-All(KB, alpha,
symbols = [B_1_1, P_1_2, P_2_1]
model = [ ])

TT-Check-All(KB, alpha,
symbols = [P_1_2, P_2_1],
model = [B_1_1 = T])

TT-Check-All(KB, alpha,
symbols = [P_1_2, P_2_1],
model = [B_1_1 = F])

TT-Check-All(KB, alpha,
symbols = [P_2_1],
model = [B_1_1 = F, P_1_2 = T])

TT-Check-All(KB, alpha,
symbols = [P_2_1],
model = [B_1_1 = F, P_1_2 = F])
TT-Check-All(KB, alpha,
symbols = [B_1_1, P_1_2, P_2_1],
model = [ ])

TT-Check-All(KB, alpha,
symbols = [P_1_2, P_2_1],
model = [B_1_1 = T])

TT-Check-All(KB, alpha,
symbols = [P_2_1],
model = [B_1_1 = F, P_1_2 = T])

TT-Check-All(KB, alpha,
symbols = [P_2_1],
model = [B_1_1 = F, P_1_2 = F])

TT-Check-All(KB, alpha,
symbols = [B_1_1, P_1_2, P_2_1],
model = [B_1_1 = F])

TT-Check-All(KB, alpha,
symbols = [B_1_1, P_1_2, P_2_1],
model = [B_1_1 = T])

TT-Check-All(KB, alpha,
symbols = [B_1_1, P_1_2, P_2_1],
model = [B_1_1 = F, P_1_2 = T])

TT-Check-All(KB, alpha,
symbols = [B_1_1, P_1_2, P_2_1],
model = [B_1_1 = F, P_1_2 = F])
TT-Check-All(KB, alpha, symbols = [B_1_1, P_1_2, P_2_1], model = [ ])

TT-Check-All(KB, alpha, symbols = [P_1_2, P_2_1], model = [B_1_1 = T])

TT-Check-All(KB, alpha, symbols = [P_2_1], model = [B_1_1 = F, P_1_2 = T])

TT-Check-All(KB, alpha, symbols = [P_2_1], model = [B_1_1 = F, P_1_2 = F])

TT-Check-All(KB, alpha, symbols = [], model = [B_1_1 = F, P_1_2 = T, P_2_1 = T])

TT-Check-All(KB, alpha, symbols = [], model = [B_1_1 = F, P_1_2 = T, P_2_1 = F])
Entire tree of recursive calls to TT-Check-All
At the top level:
Initial call to TT-Check-All
from TT-Entails.
Empty model
At the second level:
Two function calls. Model assigns value to B_1_1.
At the third level:
Four function calls.
Model assigns values to $B_{1\_1}$, $P_{1\_2}$.
At the fourth level:
Eight function calls.
Base case: Model assigns values to all symbols.
How does the return value at the top level depend on the eight return values at the fourth level?
The top level returns true if and only if all eight calls at the bottom level return true.
What do the bottom levels return?

Boolean TT-Check-All(KB, alpha, symbols, model)
  if Empty?(symbols):
    if PL-True?(KB, model) then return PL-True?(alpha, model)
    else return true
  else:
    ...

- Example KB: ???
  (iff B_1_1 (or P_1_2 P_2_1))
  B_1_1
- Example alpha: ???
  (not P_1_2)
- TT-Check-All returns ???
What do the bottom levels return?

Boolean TT-Check-All(KB, alpha, symbols, model)
  if Empty?(symbols):
    if PL-True?(KB,model) then return PL-True?(alpha, model)
    else return true
  else:
    ...

Example KB: true
  (iff B_1_1 (or P_1_2 P_2_1))
  B_1_1

Example alpha: false
  (not P_1_2)

TT-Check-All returns false.

Since this call to TT-Check-All returns false, what can you tell about the return value of TT-Entails in this example?
What do the bottom levels return?

Boolean TT-Check-All(KB, alpha, symbols, model)
  if Empty?(symbols):
    if PL-True?(KB, model) then return PL-True?(alpha, model)
    else return true
  else:
    ...

• Example KB: true
  (iff B_1_1 (or P_1_2 P_2_1))
  B_1_1

• Example alpha: false
  (not P_1_2)

• TT-Check-All returns false.

Since this call to TT-Check-All returns false, what can you tell about the return value of TT-Entails in this example?

TT-Entails will return false.
If any call at the bottom level returns false, TT-Entails returns false.
If all calls at the bottom level return true, TT-Entails returns true.