Este texto está sendo construído para suprir, em parte, a falta de acesso gratuito a livro em português. São as minhas notas de aula da disciplina Lógica Para Computação. O objetivo é que auxilie você a acompanhar as aulas e permita uma consulta rápida, sem ter de retornar aos vídeos. Devido à velocidade da escrita, para tentar acompanhar o conteúdo das aulas, provavelmente terá erros.

Todo comentário é bem vindo!

Jânio Canuto

Motivação

Nesta disciplina vamos estudar, como o nome sugere, Lógica. Para ser mais exato, vamos estudar um pouco da lógica matemática. Com o intuito de deixar claro o que vamos estudar e como ela se relaciona com a computação, vamos iniciar com um breve histórico da lógica.

História

Lógica é uma disciplina trivial, não no sentido que damos à esta palavra atualmente mas, no sentido que era utilizado na Grécia antiga (em torno de 500 A.C.). Trivial é um assunto que faz parte da trivia, um conjunto de três disciplinas (gramática, retórica e lógica) consideradas fundamentais e que deveriam ser dominadas antes de iniciar o estudo do *quadrivium* (aritmética, geometria, música e astronomia).

Por que eram considerados tão importantes para a educação? Perceba que as três disciplinas triviais consistem no domínio da comunicação e do encadeamento de ideias. Originalmente, a lógica era estudada pelos Sofistas que empregavam-na principalmente em debates jurídicos. Eventualmente, procuraram criar um sistema de regras para determinar, sem dúvidas, quem ganhou um argumento, para isso foi desenvolvida a lógica.

Portanto, a lógica trata de um conjunto de regras para raciocinar e argumentar. Esse é um problema fundamental no desenvolvimento intelectual: distinguir o que é verdadeiro do que é falso, o que é certo do que é errado.

Originalmente, a lógica tratava de argumentos em linguagem natural. Por exemplo, ela era usada para determinar a validade de argumentos como:

“Todo homem é mortal.
Sócrates é um homem.
Logo, Sócrates é mortal.”

O problema é que a linguagem natural é propensa à ambiguidade, tanto de forma intrínseca quanto em termos de interpretação. Palavras como “manga” têm significados distintos dependendo do contexto em que se encontra. Certas frases podem ter significados muito distintos a depender da entonação dada pelo leitor. Você já deve ter passado por uma situação em que enviou uma mensagem a alguém e teve a mensagem mal interpretada devido ao tom com que a mensagem foi lida.

Além disso, a linguagem natural pode levar a paradoxos. Considere a frase “Esta frase é mentira”, se ela for verdadeira ela é mentira, se ela é mentira, então ela é verdadeira.

Estas observações levaram ao desenvolvimento das leis da lógica clássica:

1. Princípio da Identidade: uma coisa é idêntica a si mesma.
2. Princípio do Terceiro Excluído: uma coisa é ou não é, não há outra opção.
3. Princípio da Contradição: uma coisa não pode ser e não ser simultaneamente.

Estes princípios excluem a possibilidade de paradoxos e simplificam bastante o mundo, visto que só há dois estados possíveis para qualquer coisa.

Adicionalmente, percebeu-se que, mais que analisar cada argumento, era possível abstrair a estrutura da argumentação do contexto, dando origem à lógica simbólica. Considere o argumento apresentado anteriormente, ele tem uma forma geral que pode ser resumida em:

"Todo *X* é *Y*.
Z é *X*.
Logo, *Z* é *Y*."

Dadas que as duas primeiras sentenças são verdadeiras, a conclusão "*Z* é *Y*" é verdadeira para quaisquer afirmações *X*, *Y* e *Z* que obedeçam às leis da lógica clássica. Estas estruturas de raciocínio são objetos de estudo da Lógica, os chamados silogismos.

Em 1847, George Boole iniciou a formulação da lógica como uma linguagem matemática e as regras de inferência foram modeladas de modo similar à manipulação algébrica de expressões. Até o fim do século XIX a lógica simbólica e algébrica desenvolveu-se bastante, e se tornou bastante útil para resolver diversos problemas na matemática.

Lógica na Matemática

Uma prova matemática nada mais é que um argumento válido. Ou seja, uma sequência de idéias que chega a uma conclusão que é, sem dúvidas, correta. Conforme as provas tornaram-se mais elaboradas, paradoxos começaram a surgir, e a lógica algébrica encontrou o seu lugar na matemática, na formalização das provas. Em 1879, Frege propôs usar a lógica como uma linguagem para a matemática e, com isso, o rigor das provas aumentou bastante.

Com o rigor desta nova linguagem Cantor foi capaz de analisar o conceito do infinito de uma forma antes impossível, concluindo que existe uma hierarquia de infinitos, de tamanhos distintos (o infinito dos reais, por exemplo, é maior que o infinito dos naturais, já o dos racionais é do mesmo tamanho dos naturais). Chegando a uma definição para a comparação do tamanho de conjuntos infinitos: $|A| = |B|$ se, e somente se, existe uma função bijetora entre *A* e *B*.

Também por causa da formalização, Bertrand Russell percebeu que a teoria dos conjuntos continha paradoxos. Considere o conjunto $S = \{X \mid X \notin X\}$, se *S* não pertence a *S* então por definição *S* pertence a *S*. Isso levou a uma atualização da teoria de conjuntos em uma hierarquia de conjuntos.

Todo teorema matemático pode ser pensado como sendo uma sentença lógica que é sempre verdadeira. Daí David Hilbert levantou, no início do século XX, a importância de se obter um método (algoritmo) para a determinação automática e eficiente da validade de uma sentença lógica. Uma vez que a lógica é uma linguagem para a descrição do raciocínio e, para a matemática em particular, uma linguagem que descreve as provas, tal algoritmo seria um resolvedor automático de problemas ou um provador automático de teoremas.

Pouco tempo depois, os trabalhos de Gödel, Church e Turing mostraram que tal algoritmo não existe e, mesmo para os casos em que é possível resolver, nem sempre é possível fazê-lo de forma eficiente. Apesar destes resultados, a lógica continuou a se desenvolver, não mais como o fundamento definitivo e universal da matemática, mas como um ramo dela.

Lógica na Computação

Na computação a lógica encontrou uma nova casa. De fato, a ciência da computação inicia com os resultados de Church e Turing citados no fim da seção anterior. Os resultados dão origem à teoria da Computabilidade e, posteriormente, da Complexidade.

Além do fundamento teórico, todo *Hardware* utilizado nos computadores modernos é modelado e implementado utilizando as operações lógicas primitivas. O computador ENIAC era implementado em dígitos decimais, mas logo percebeu-se que era muito mais fácil trabalhar com lógica binária pois é mais simples criar circuitos para gerar e detectar de forma consistente apenas dois níveis de tensão ao invés de 10.

A lógica é uma linguagem formal que pode ser utilizada para descrever de forma precisa e pode ser utilizada como uma linguagem de programação. A linguagem SQL, para interface com bancos de dados, é essencialmente idêntica à lógica de predicados.

Outra área da computação que se utilizou bastante da lógica foi a Inteligência Artificial. Na década de 60, os sistemas especialistas, que modelam formalmente o conhecimento de especialistas em uma dada área, foram bastante desenvolvidos. Basicamente, trata-se de um conjunto de regras de decisão que tentam simular as ações tomadas por especialistas em um certo domínio.

Objeto de Estudo

Antes de definirmos o nosso objeto de estudo, vale uma breve explicação sobre as formas de raciocínio. Três formas de raciocínio são estudadas na lógica clássica: os raciocínios dedutivo, indutivo e abdutivo. Considerando que temos uma premissa, uma regra do tipo “se Premissa então Conclusão” e uma conclusão, podemos explicar as formas de raciocínio do seguinte modo:

1. Raciocínio dedutivo: dadas que a Premissa e Regra são verdadeiras, infere-se a Conclusão.
2. Raciocínio indutivo: a partir de exemplos de Premissa e Conclusão, infere-se a regra.
3. Raciocínio abdutivo: dadas a Regra e a Conclusão, infere-se a premissa.

Perceba que apenas o raciocínio dedutivo tem garantia de que o objeto inferido está correto.

No raciocínio indutivo, a base da Ciência, não há garantias de que a regra obtida é de fato o mecanismo que está provocando o efeito (conclusão) a partir da causa (premissa) observada. Considere a sequência 2, 10, 12, 16, 17, 18, X. Qual o valor de X? Esta é uma questão clássica em concursos e muitas pessoas aprenderam e se convenceram que a resposta correta é 200. Essa resposta é obtida através da inferência de que a regra de formação da sequência é “Número naturais, em ordem crescente, iniciados pela letra D”. Mas o fato é que, para qualquer que seja o valor de X, existe um polinômio de grau 6 que

passa por todos esses pontos. Portanto, o raciocínio indutivo não dá uma resposta definitiva para o problema.

Do mesmo modo, não há como garantir que o que é inferido pelo raciocínio abdutivo está correto. Considere a regra “Se chover, a grama molha” e a observação “a grama está molhada”. O raciocínio abdutivo conclui que choveu, mas não há nenhuma garantia disto, uma vez que a grama pode estar molhada por outro motivo.

Deste modo, na lógica matemática apenas o raciocínio dedutivo é utilizado. É ele a base das argumentações nesta área. Você pode se perguntar: “mas eu estudei o método de prova por indução, ele não é válido?”. Apesar do nome do método ser “indução”, o raciocínio é dedutivo.

Finalmente, o que vamos estudar nessa disciplina? Vamos abordar duas versões da lógica matemática clássica: a lógica proposicional e a lógica de predicados. Em ambos os casos vamos estudar a sintaxe e a semântica da linguagem, propriedades semânticas e técnicas para a verificação de tais propriedades. No fim do curso veremos uma introdução à programação lógica.

A todo momento vou tentar tratar da lógica como uma linguagem formal. Como tal, serve para descrever e, eventualmente, resolver problemas. Ou seja, uma linguagem de modelagem, tal qual o cálculo ou a álgebra. Bom, vamos lá!

Lógica Proposicional

A primeira linguagem lógica que vamos estudar é a proposicional, também chamada de Booleana ou lógica de ordem zero. Vamos começar definindo a linguagem a partir dos seus elementos mais básicos que são o alfabeto e a gramática.

Sintaxe

Alfabeto

O alfabeto da Lógica Proposicional é formado por quatro tipos de símbolos: símbolos proposicionais, símbolos de verdade, conectivos (ou operadores) lógicos e pontuação.

Como discutido no histórico, a lógica que estudaremos é simbólica. Deste modo, as afirmações são substituídas por *símbolos proposicionais* que fazem um papel semelhante às variáveis na álgebra. Neste texto utilizaremos como símbolos proposicionais qualquer coisa iniciada por letra maiúscula e que não contenha espaço, como por exemplo: A, X, H, F_a, S_1, Casa, TelaAzul. Apesar de ser formado por várias letras, TelaAzul é considerado um único símbolo proposicional e representa alguma afirmação que é verdadeira ou falsa.

Os símbolos de verdade representam as constantes desta linguagem que são tradicionalmente chamadas de *verdadeiro* e *falso*. Na prática, basta que sejam dois símbolos distintos que atendem algumas propriedades que veremos futuramente. No fim das contas, o que temos é um sistema binário (*i.e.*, que tem apenas dois valores). Neste texto vou utilizar 0 para representar o *falso* e 1 para representar o *verdadeiro*.

Os conectivos lógicos são funções, que têm significados que serão definidos posteriormente. Por hora, basta saber que utilizaremos os conectivos ¬, ∧, ∨, → e ↔.

Por fim, os símbolos de pontuação são os parênteses "(" e ")". Estes símbolos têm o mesmo papel dos parênteses na aritmética, que é a determinação da ordem de avaliação dos conectivos, evitando ambiguidades na leitura.

Alfabeto:

- *Símbolos de verdade*: 0 e 1.
- *Símbolos proposicionais*: P, Q, S, F_1, Casa, Cor, T, X, A_b, …
  (iniciado por maiúscula)
- *Conectivos (Operadores)*: ¬, ∧, ∨, → e ↔.
- *Símbolos de pontuação*: (e).

Fórmulas

A concatenação dos símbolos do alfabeto dão origem às fórmulas da lógica proposicional. No entanto, nem toda concatenação de símbolos dá origem a uma fórmula válida, assim como nem toda concatenação de letras forma palavras válidas em português e nem toda concatenação de símbolos forma uma fórmula aritmética válida.

A seguinte regra gramatical recursiva determina as fórmulas válidas da lógica proposicional:

Fórmula:

1. Todo símbolo de verdade é uma fórmula.
2. Todo símbolo proposicional é uma fórmula.
3. Se X é fórmula, então (¬X) é uma fórmula.
4. Se X e Y são fórmulas então (X □ Y) é uma fórmula, onde □ é $\wedge$, $\vee$, $\rightarrow$ ou $\leftrightarrow$.

Usando estas regras, uma quantidade infinita de fórmulas pode ser obtida. Perceba que, segundo estas regras, sempre que um conectivo é inserido na fórmula um par de parênteses também aparece.

Para evitar um excesso de parênteses podemos definir uma precedência de conectivos para permitir a omissão dos parênteses e garantir que não há ambiguidades. Na aritmética define-se que os operadores de multiplicação e divisão têm maior precedência que os operadores de soma e subtração, de tal modo que podemos escrever a fórmula ((3 x 4) + 5) como 3 x 4 + 5 sem ambiguidade de interpretação.

Neste texto utilizaremos a seguinte ordem de precedência: ¬, $\leftrightarrow$, $\rightarrow$, $\wedge$ e $\vee$. Assim, a fórmula $\neg X \wedge Y$ deve ser interpretada como $((\neg X) \wedge Y)$.

Ordem de Avaliação:

1. Parênteses.
2. ¬
3. $\leftrightarrow$
4. $\rightarrow$
5. $\wedge$
6. $\vee$
7. Da esquerda para a direita.

Perceba que as regras de formação e precedência independem da semântica, até agora só dissemos como montar as fórmulas sem dizer o que significam os operadores, é como se alguém te ensinasse a escrever palavras em uma língua desconhecida sem te dizer o significado.

Antes de partir para a semântica, vamos definir mais dois elementos sintáticos que podem vir a ser úteis futuramente.

Comprimento de Fórmula

O comprimento de uma fórmula X, denotado por |X|, pode ser definido a partir da definição de fórmula:

1. Se X é um símbolo proposicional ou de verdade, então |X| = 1.
2. Se X e Y são fórmulas, então:
    a. $|\neg X| = |X| + 1$.
    b. $|X \square Y| = |X| + |Y| + 1$, onde □ é $\wedge$, $\vee$, $\rightarrow$ ou $\leftrightarrow$.

De forma resumida, o comprimento da fórmula é igual ao número de símbolos proposicionais, de verdade e conectivos da fórmula, sem considerar a pontuação.

Exemplo: Qual o comprimento da fórmula $F = (A \rightarrow (B \vee C)) \wedge \neg B$?

Aplicando a definição recursiva:
$|F| = |(A \rightarrow (B \vee C)) \wedge \neg B|$
$= |(A \rightarrow (B \vee C))| + |\neg B| + 1$
$= (|A| + |B \vee C| + 1) + (|B| + 1) + 1$
$= ((1) + (|B| + |C| + 1) + 1) + ((1) + 1) + 1$
$= ((1) + ((1) + (1) + 1) + 1) + ((1) + 1) + 1$
$= 8$
Ou simplesmente contando o número de ocorrências de cada símbolo:
4 símbolos proposicionais + 4 conectivos = 8.

Subfórmula

Formalmente, se X é uma fórmula:

1. X é uma subfórmula de X.
2. Se $X = (\neg Y)$, então Y é subfórmula de X.
3. Se $X = (Y \square Z)$, então Y e Z são subfórmulas de X, onde $\square$ é $\wedge$, $\vee$, $\rightarrow$ ou $\leftrightarrow$.
4. Se Y é subfórmula de X, então toda subfórmula de Y também é subfórmula de X.

A definição de subfórmula é similar à definição de subconjunto. Existe uma relação direta entre a definição de comprimento e a de subfórmula, de modo que o número de subfórmulas é sempre menor ou igual ao comprimento. A cada passo recursivo da definição, pegamos um operador e dizemos que seus operandos são subfórmulas.

É importante notar que nem todo pedaço da fórmula é subfórmula, é particularmente fácil de cometer este erro quando os parênteses são omitidos.

Exemplo: Quais as subfórmulas de $F = A \rightarrow B \vee C$?
A primeira vista, é fácil achar que $B \vee C$ é uma subfórmula, mas não é o caso. Vamos reescrever esta fórmula com os parênteses que foram omitidos: $F = ((A \rightarrow B) \vee C)$. Daí fica claro que o $B \vee C$ não é subfórmula, pois tem um parêntese entre o símbolo B e o conectivo.
As subfórmulas são:

A própria fórmula:

1. $A \rightarrow B \vee C$

Os operandos da operação mais externa, que é o $\vee$:

2. $A \rightarrow B$
3. C

Os operandos da subfórmula 2:

4. A
5. B

Neste caso, temos o número de subformulas igual ao comprimento da fórmula.

Exemplo: Quais as subfórmulas de $F = A \rightarrow (A \vee B)$?
As subfórmulas são:

A própria fórmula:

1. $A \rightarrow (A \vee B)$

Os operandos da operação mais externa, que é o →:

2. A
3. A ∨ B

Os operandos da subfórmula 3:

4. A
5. B

Neste caso temos 5 subfórmulas, mas as subfórmulas 2 e 4 são idênticas, então o número de subformulas distintas é menor que o comprimento da fórmula.

Semântica

Semântica é a atribuição de significado a algo. No caso das fórmulas da lógica, existem duas interpretações possíveis, falso (0) ou verdadeiro (1).

Seguindo os conceitos discutidos no anteriormente, no histórico, os elementos da lógica têm dois valores possíveis (Princípio do Terceiro Excluído) e somente um destes valores (Princípio da Contradição).

Interpretação

A interpretação de uma fórmula lógica pode ser entendida como uma função cujo domínio são as fórmulas da lógica proposicional e o contradomínio é o conjunto dos símbolos de verdade, {0, 1}.

Qual o valor da fórmula $A_1 \wedge B \vee Coisa$? Bom, depende do significado de cada um dos elementos da fórmula. Qual o valor dos símbolos proposicionais A_1, B e Coisa? Como são interpretadas as operações ∧ e ∨? Até agora definimos apenas que o operador ∧ deve ser avaliado antes do operador ∨, mas como?

Todos estes elementos são definidos pela função Interpretação que defino agora.

Função Interpretação:

- Se F = P, onde P é um símbolo proposicional, I[F] = I[P] ∈ {0, 1}.
- Se F = 0, então I[F] = 0.
- Se F = 1, então I[F] = 1.
- Se F = ¬X, então
  - I[F] = I[¬X] = 1, se I[X] = 0;
  - I[F] = I[¬X] = 0, se I[X] = 1.
- Se F = X ∧ Y, então
  - I[F] = I[X ∧ Y] = 1, se I[X] = 1 e I[Y] = 1;
  - I[F] = I[X ∧ Y] = 0, se I[X] = 0 e/ou I[Y] = 0.
- Se F = X ∨ Y, então
  - I[F] = I[X ∨ Y] = 1, se I[X] = 1 e/ou I[Y] = 1;
  - I[F] = I[X ∨ Y] = 0, se I[X] = 0 e I[Y] = 0.
- Se F = X → Y, então
  - I[F] = I[X → Y] = 1, se I[X] = 0 e/ou I[Y] = 1;
  - I[F] = I[X → Y] = 0, se I[X] = 1 e I[Y] = 0.
- Se F = X ↔ Y, então

- I[F] = I[X ↔ Y] = 1, se I[X] = I[Y];
- I[F] = I[X ↔ Y] = 0, se I[X] ≠ I[Y].

O primeiro item apenas diz que a interpretação dos símbolos proposicionais têm como domínio o conjunto {0, 1}. O segundo e terceiro item definem os símbolos de verdade como constantes. Os itens que seguem definem cada um dos operadores lógicos.

Vamos agora dar nome a cada uma das operações definidas: ¬ é a negação (“não”), que inverte o valor lógico; ∧ é a conjunção (“e”), que só é verdade se todos os argumentos são verdadeiros; ∨ é a disjunção (“ou”), que só é falso se todos os argumentos são falsos; → é a implicação (“se … então”), que só é falso se o primeiro argumento é verdadeiro e o segundo é falso; e ↔ é a bi-implicação (“se, e somente se”), que é verdade se os argumentos têm o mesmo valor.

Na implicação é comum chamarmos o primeiro argumento de antecedente e o segundo de consequente. Podemos pensar nos operadores de conjunção e disjunção como condições que só são satisfeitas se todas as partes são satisfeitas ou se pelo menos uma das partes é satisfeita, respectivamente. Já a implicação e a bi-implicação são a base dos teoremas matemáticos.

A definição dada acima é uma possível forma de definir os operadores, mas como são funções de domínio finito, podemos também defini-los por enumeração. A definição por enumeração das fórmulas lógicas é comumente conhecida como “tabela verdade”. As tabelas abaixo são as tabelas verdades de cada um dos conectivos lógicos. Cada linha da tabela verdade corresponde a uma possível interpretação de X e Y.

Tabela Verdade da Negação

X	¬X
0	1
1	0

Tabela Verdade da Conjunção

X	Y	X ∧ Y
0	0	0
0	1	0
1	0	0
1	1	1

Tabela Verdade da Disjunção

X	Y	X ∨ Y
0	0	0
0	1	1

1	0	1
1	1	1

Tabela Verdade da Implicação

X	Y	$X \rightarrow Y$
0	0	1
0	1	1
1	0	0
1	1	1

Tabela Verdade da Bi-implicação

X	Y	$X \leftrightarrow Y$
0	0	1
0	1	0
1	0	0
1	1	1

Para interpretar uma fórmula, podemos aplicar a definição da função de interpretação ou a tabela verdade.

Exemplo: Dê as interpretações da fórmula $F = (\neg X \vee Y) \rightarrow (X \wedge Y)$.

Primeiramente, vamos utilizar a função de interpretação. Para ter um resultado, é necessário que a interpretação determine os valores das interpretações dos símbolos proposicionais.

Seja I_1 a interpretação tal que $I_1[X] = 0$ e $I_1[Y] = 0$.

Daí temos que F é uma implicação cujo primeiro argumento é $A = \neg X \vee Y$ e o segundo argumento é $B = X \wedge Y$.

$I_1[F] = I_1[A \rightarrow B] = 0$, se, e somente se, $I_1[A] = 1$ e $I_1[B] = 0$.

Seja $C = \neg X$, então $I_1[A] = I_1[C \vee Y] = 0$, se, e somente se, $I_1[C] = I_1[Y] = 0$.

Temos que $I_1[B] = I[X \wedge Y] = 1$ se, e somente se, $I_1[X] = I_1[Y] = 1$.

Da interpretação, sabemos que $I_1[X] = I_1[Y] = 0$, portanto $I_1[B] = 0$.

$I_1[C] = I_1[\neg X] = 1$, já que $I_1[X] = 0$.

Daí temos que $I_1[C] = 1$ e $I_1[Y] = 0$, logo $I_1[A] = 1$. Finalmente, temos que $I_1[F] = 0$.

Essa é uma das interpretações possíveis para esta fórmula, existem outras três, visto que cada interpretação atribui valores lógicos (binários) para os símbolos proposicionais e temos dois símbolos.

Utilizando a tabela verdade, a ideia é a mesma. A partir das interpretações das subformulas vamos obter a interpretação da fórmula como um todo.

X	Y	C = ¬X	A = C ∨ Y	B = X ∧ Y	F = A → B
0	0	1	1	0	0
0	1	1	1	0	0
1	0	0	0	0	1
1	1	0	1	1	1

A interpretação I_1, feita anteriormente, corresponde à primeira linha da tabela verdade acima, onde $I_1[X] = I_1[Y] = 0$.

Vamos discutir um pouco a semântica de cada um dos conectivos. A negação (¬) corresponde a uma inversão do valor lógico, é de certa forma equivalente ao sinal negativo na álgebra. A conjunção (∧) é verdadeira se todos os argumentos são verdadeiros. A disjunção (∨) é verdadeira se pelo menos um dos argumentos é verdadeiro, esse conectivo é comumente chamado de "ou", mas é importante perceber que é um "ou" exclusivo e não um "ou" exclusivo como normalmente utilizamos na língua portuguesa. A bi-implicação (↔) é verdade se os argumentos têm valores idênticos, semelhante ao comparador de igualdade na álgebra.

O operador de implicação (→) tem um significado que vale a pena um pouco mais de discussão porque difere um pouco a noção intuitiva que alguns de nós temos sobre a "implicação". Primeiro, perceba que implicação não é causalidade. A fórmula X → Y é verdadeira sempre que Y é verdadeira, não importa o valor ou o significado de X. Se Y é a afirmação "A terra é redonda", que vamos assumir como verdadeira, X pode ser qualquer coisa, relacionada ou não ao fato da terra ser redonda.

Em segundo lugar, o operador de implicação tem a estranha propriedade de ser verdadeiro quando o primeiro argumento é falso. "Se Falso então Y" é verdade independentemente do que seja Y. O motivo deste comportamento ser observado no operador tem a ver com o fato de não ser possível determinar se a consequência é verdadeira ou não a partir de uma afirmação falsa. Considere a afirmação X = "p é um número real maior que 10" e Y = "p^2 é um número real maior que 100". Se a primeira afirmação é verdadeira, a implicação só está correta se a segunda é verdadeira. Agora considere que a primeira afirmação é falsa. Se p é o número 2, p^2 é 4 e, neste caso, Y é falso. Agora considere que p é o número -11, neste caso p^2 vale 121. Temos então que X é falso e Y é verdadeiro. Então, a partir de uma premissa falsa, podemos obter uma consequência verdadeira ou falsa.

Exemplos:

a) $F_1 = (\neg A \vee \neg B) \rightarrow C$

A	B	C	D = ¬A	E = ¬B	F = D ∨ E	F_1 = F → C
0	0	0	1	1	1	0
0	0	1	1	1	1	1

0	1	0	1	0	1	0
0	1	1	1	0	1	1
1	0	0	0	1	1	0
1	0	1	0	1	1	1
1	1	0	0	0	0	1
1	1	1	0	0	0	1

b) $F_2 = (\neg A \wedge B) \rightarrow (C \vee A)$

A	B	C	D = ¬A	E = D ∧ B	F = C ∨ A	F_2 = E → F
0	0	0	1	0	0	1
0	0	1	1	0	1	1
0	1	0	1	1	0	0
0	1	1	1	1	1	1
1	0	0	0	0	1	1
1	0	1	0	0	1	1
1	1	0	0	0	1	1
1	1	1	0	0	1	1

c) $F_3 = A \vee \neg B \vee C$

A	B	C	D = ¬B	E = D ∨ C	F_3 = A ∨ E
0	0	0	1	1	1
0	0	1	1	1	1
0	1	0	0	0	0
0	1	1	0	1	1
1	0	0	1	1	1
1	0	1	1	1	1
1	1	0	0	0	1
1	1	1	0	1	1

Todas as tabelas verdade dos exemplos anteriores tinham 8 linhas. Quantas interpretações distintas uma fórmula pode ter?

Cada linha da tabela verdade corresponde a uma interpretação. A função de interpretação é completamente determinada com exceção do valor dos símbolos proposicionais.

Como cada símbolo proposicional tem apenas dois valores possíveis, o total de interpretações possíveis para uma fórmula com N variáveis é 2^N.

Existem outros operadores binários além dos 4 que adotamos neste texto? Com certeza! Quantos conectivos binários distintos podem ser definidos?

Cada conectivo pode ser visto como uma função que mapeia as entradas (operandos) em uma saída. Se o conectivo é binário, ele tem dois argumentos, portanto 4 interpretações possíveis. Cada mapeamento pode ser visto como uma função distinta, portanto temos $2^4 = 16$ mapeamentos possíveis, já que cada a tabela verdade tem 4 linhas e cada linha tem 2 valores possíveis.

Qual a diferença entre → e ⇒? E entre ↔ e ⇔?

Os operadores com traço duplo são normalmente encontrados em teoremas matemáticos e normalmente recebem o mesmo nome dos operadores lógicos com traço simples. No entanto, vamos pensar no significado destes símbolos em provas matemáticas.

Quando vemos um teorema do tipo X ⇒ Y, isto significa que se X é verdadeiro Y também é. Se X é falso não sabemos se Y é verdadeiro ou falso. Perceba que isso corresponde às linhas com resultado verdadeiro da tabela verdade do operador →.

Do mesmo modo, um teorema do tipo X ⇔ Y, significa que se X é verdadeiro Y é verdadeiro e se X é falso Y também é falso. Isso corresponde às conclusões que tiramos da tabela verdade do operador ↔ nos casos em que o resultado é verdadeiro.

Portanto, os operadores de traço duplo dizem que a relação é verdadeira. Ou seja, os casos falsos não podem ocorrer. Veremos isto com mais detalhes na próxima seção.

Atenção!
Alguns textos adotam os símbolos com traço duplo como sinônimos dos de traços simples. Como qualquer símbolo matemático, o significado é uma escolha que deve ser definida. Então se você estudar algum outro texto e achar que o significado não está condizente, procure verificar se as notações adotadas lá são as mesmas que adoto aqui.

Propriedades Semânticas

Aproveitando a discussão anterior, vamos definir algumas propriedades semânticas das fórmulas da lógica proposicional. As propriedades semânticas são nomes especiais a certos comportamentos observados na interpretação das fórmulas.

Defino agora algumas propriedades semânticas fundamentais no estudo da lógica:

Propriedades:

1. **Satisfatibilidade:**
   Uma fórmula F é *satisfatível,* ou *factível,* se, e somente se, existe pelo menos uma interpretação I tal que I[F] = 1.
2. **Validade:**

Uma fórmula F é uma *tautologia*, ou *válida*, se, e somente se, para toda interpretação I, I[F] = 1.

3. **Insatisfatibilidade:**
   Uma fórmula F é *contraditória*, ou *insatisfatível*, se, e somente se, para toda interpretação I, I[F] = 0.

Além destas propriedades, podemos também definir a satisfatibilidade de um conjunto de fórmulas:

- Um conjunto de fórmulas $\alpha = \{F_1, F_2, \ldots, F_N\}$ é satisfatível se, e somente se, existe uma interpretação I tal que $I[F_1] = I[F_2] = \ldots = I[F_N] = 1$. Neste caso podemos escrever $I[\alpha] = 1$.

A partir destas definições podemos definir formalmente os operadores semânticos descritos no fim da seção anterior:

- Dadas duas fórmulas, F e G, dizemos que F implica em G, denotado por $F \Rightarrow G$, se, e somente se, $F \rightarrow G$ é uma tautologia.
- Dadas duas fórmulas, F e G, dizemos que F equivale a G, denotado por $F \Leftrightarrow G$, se, e somente se, $F \leftrightarrow G$ é uma tautologia.

Como pode ser inferido por estas definições, as tautologias são fundamentais para a matemática. Do ponto de vista de funções, ou circuitos, as tautologias não são tão interessantes porque são funções que têm uma saída constante para qualquer entrada e, na prática, poderiam ser substituídas por uma constante.

Apesar disso, as tautologias também serão muito úteis em nosso estudo da lógica, em especial as equivalências. Perceba que se duas fórmulas são equivalentes elas podem ser substituídas uma pela outra. Do ponto de vista de funções, às vezes podemos encontrar expressões mais simples para nossas funções através do uso das equivalências. Do mesmo modo, circuitos digitais distintos que são representados pela mesma fórmula lógica são intercambiáveis.

Algumas equivalências são muito importantes e são a base para a manipulação algébrica das fórmulas lógicas. Pensando na fórmula lógica como uma função que mapeia o valor dos símbolos proposicionais no valor da fórmula, duas fórmulas equivalentes têm a mesma função e, por isso, são intercambiáveis. Qual a importância disso? Na eletrônica, fórmulas lógicas representam circuitos digitais e fórmulas equivalentes representam circuitos equivalentes, que podem ser substituídos sem prejuízo para o comportamento obtido. Para a computação, o mesmo ocorre, podemos obter expressões mais simples que têm o mesmo significado de expressões complexas.

Apresento a seguir alguns exemplos de tautologias importantes. A verificação da validade das tautologias apresentadas pode ser feita através do uso da tabela verdade e fica como exercício.

Equivalências e Implicações Importantes:

- $(A \vee \neg A) \Leftrightarrow 1$.

    - Esta afirmação corresponde ao princípio do terceiro excluído, uma fórmula lógica é “verdadeira” ou “falsa”, não havendo uma terceira possibilidade.
- $(A \land \neg A) \Leftrightarrow 0$.
    - Esta afirmação corresponde ao princípio da contradição, uma fórmula lógica não pode ser “verdadeira” e “falsa” simultaneamente.
- $(A \leftrightarrow B) \Leftrightarrow ((A \rightarrow B) \land (B \rightarrow A))$.
    - A bi-implicação é equivalente a implicação nos dois sentidos.
- $(A \leftrightarrow B) \Leftrightarrow ((A \land B) \lor (\neg A \land \neg B))$.
    - Pela definição da bi-implicação, ela é verdadeira se os dois argumentos são verdadeiros ou se ambos são falsos.
- $(A \rightarrow B) \Leftrightarrow (\neg B \rightarrow \neg A)$.
    - Este é o princípio da contraposição. A implicação não funciona no sentido inverso, ou seja, se B é verdade A não necessariamente é verdadeira. No entanto, se B é falso A também deve ser falso.
- $\neg(A \land B) \Leftrightarrow (\neg A \lor \neg B)$.
    - Esta é uma das leis de DeMorgan, que trata da aplicação da negação sobre a conjunção.
- $\neg(A \lor B) \Leftrightarrow (\neg A \land \neg B)$.
    - Esta é uma das leis de DeMorgan, que trata da aplicação da negação sobre a disjunção.
- $A \land (B \lor C) \Leftrightarrow ((A \land B) \lor (A \land C))$.
    - Distributividade do operador de conjunção sobre a disjunção.
- $A \lor (B \land C) \Leftrightarrow ((A \lor B) \land (A \lor C))$.
    - Distributividade do operador de disjunção sobre a conjunção.
- $(A \land B) \land C \Leftrightarrow A \land (B \land C)$
    - Associatividade do operador de conjunção.
- $(A \lor B) \lor C \Leftrightarrow A \lor (B \lor C)$
    - Associatividade do operador de disjunção.
- $(A \land 1) \Leftrightarrow A$.
    - Elemento neutro da conjunção.
- $(A \land 0) \Leftrightarrow 0$.
    - Elemento nulo da conjunção.
- $(A \lor 1) \Leftrightarrow 1$.
    - Elemento nulo da disjunção.
- $(A \lor 0) \Leftrightarrow A$.
    - Elemento neutro da disjunção.
- $(A \rightarrow 1) \Leftrightarrow 1$.
- $(A \rightarrow 0) \Leftrightarrow \neg A$.
- $(1 \rightarrow A) \Leftrightarrow A$.
- $(0 \rightarrow A) \Leftrightarrow 1$.
- $(A \leftrightarrow 1) \Leftrightarrow A$.
- $(A \leftrightarrow 0) \Leftrightarrow \neg A$.
- $(A \land B) \Rightarrow A$.
    - A conjunção implica nas suas partes.
- $(A \land B) \Rightarrow B$.
    - A conjunção implica nas suas partes.
- $A \Rightarrow (A \lor B)$.

    - A disjunção não implica em suas partes, mas as partes implicam na disjunção.
- $B \Rightarrow (A \vee B)$.
    - A disjunção não implica em suas partes, mas as partes implicam na disjunção.

Estas equivalências, e várias outras, nos permitem manipular algebricamente as fórmulas da lógica proposicional. É muito fácil verificar se as manipulações efetuadas estão corretas ou não verificando se a nova fórmula permanece equivalente àquela do passo anterior.

Exemplo: Considere a fórmula $F = ((A \rightarrow B) \wedge (B \rightarrow C)) \rightarrow (A \rightarrow C)$.
Vamos utilizar as equivalências vistas anteriormente para reescrever essa fórmula.

$((A \rightarrow B) \wedge (B \rightarrow C)) \rightarrow (A \rightarrow C)$
$\Leftrightarrow ((\neg A \vee B) \wedge (\neg B \vee C)) \rightarrow (\neg A \vee C)$
$\Leftrightarrow \neg((\neg A \vee B) \wedge (\neg B \vee C)) \vee (\neg A \vee C)$
$\Leftrightarrow (\neg(\neg A \vee B) \vee \neg(\neg B \vee C)) \vee (\neg A \vee C)$
$\Leftrightarrow (A \wedge \neg B) \vee (B \wedge \neg C) \vee \neg A \vee C$
$\Leftrightarrow (\neg A \vee (A \wedge \neg B)) \vee (C \vee (B \wedge \neg C))$
$\Leftrightarrow (\neg A \vee (A \wedge \neg B)) \vee (C \vee (B \wedge \neg C))$
$\Leftrightarrow ((\neg A \vee A) \wedge (\neg A \vee \neg B)) \vee ((C \vee B) \wedge (C \vee \neg C))$
$\Leftrightarrow (1 \wedge (\neg A \vee \neg B)) \vee ((C \vee B) \wedge 1)$
$\Leftrightarrow (\neg A \vee \neg B) \vee (C \vee B)$
$\Leftrightarrow \neg A \vee C \vee (B \vee \neg B)$
$\Leftrightarrow \neg A \vee C \vee 1$
$\Leftrightarrow 1$

Perceba que demonstramos através das manipulações algébricas que esta relação é uma tautologia, ou seja, $((A \rightarrow B) \wedge (B \rightarrow C)) \Rightarrow (A \rightarrow C)$. Esta também é uma relação importante, a transitividade da implicação.

Já vimos, da definição de equivalência, que $(A \Leftrightarrow B) \Leftrightarrow [(A \leftrightarrow B)$ é tautologia]. Mas na lógica, assim como na matemática, nem tudo que parece verdadeiro é verdadeiro. Considere a proposição $(A \Leftrightarrow B) \Leftrightarrow$ (A é tautologia $\Leftrightarrow$ B é tautologia). Será que ela é verdade?

Vamos dividir a equivalência em duas implicações:

1. $(A \Leftrightarrow B) \Rightarrow$ (A é tautologia $\Leftrightarrow$ B é tautologia);
2. (A é tautologia $\Leftrightarrow$ B é tautologia) $\Rightarrow (A \Leftrightarrow B)$.

Numa implicação semântica, se o antecedente é verdadeiro o consequente também deve ser. Na primeira implicação, o antecedente diz que A e B são equivalentes, ou seja, assumem sempre os mesmos valores. Portanto, se A é tautologia, B também será e vice-versa.

Na segunda implicação, considere que A e B não são tautologias, por exemplo $A = X \wedge Y$ e $B = X \vee Y$. Então, o antecedente da implicação é verdadeiro, já que é falso que A é tautologia e é falso que B é tautologia, como ambos são falsos, a equivalência é válida. No entanto, o consequente da implicação é falso, visto que A e B não são equivalentes.

Portanto, a primeira implicação é verdadeira e a segunda é falsa.

Métodos para Determinação de Propriedades Semânticas

Até agora temos apenas dois métodos para a determinação das propriedades semânticas, a tabela verdade e a manipulação algébrica utilizando as equivalências já conhecidas.

Considere a fórmula $H = S_1 \rightarrow (S_2 \rightarrow (S_3 \rightarrow (S_4 \rightarrow (S_5 \rightarrow (S_6 \rightarrow (S_7 \rightarrow (S_8 \rightarrow S_1)))))))$. Esta fórmula tem $2^8 = 256$ interpretações, ou seja, sua tabela verdade tem 256 linhas. Seria muito cansativo montar a tabela verdade para verificar se essa fórmula é uma tautologia.

Utilizando a manipulação algébrica podemos conseguir com uma certa facilidade essa demonstração.

$S_1 \rightarrow (S_2 \rightarrow (S_3 \rightarrow (S_4 \rightarrow (S_5 \rightarrow (S_6 \rightarrow (S_7 \rightarrow (S_8 \rightarrow S_1)))))))$

$\Leftrightarrow S_1 \rightarrow (S_2 \rightarrow (S_3 \rightarrow (S_4 \rightarrow (S_5 \rightarrow (S_6 \rightarrow (S_7 \rightarrow (\neg S_8 \vee S_1)))))))$

$\Leftrightarrow S_1 \rightarrow (S_2 \rightarrow (S_3 \rightarrow (S_4 \rightarrow (S_5 \rightarrow (S_6 \rightarrow (\neg S_7 \vee (\neg S_8 \vee S_1)))))))$

…

$\Leftrightarrow \neg S_1 \vee (\neg S_2 \vee (\neg S_3 \vee (\neg S_4 \vee (\neg S_5 \vee (\neg S_6 \vee (\neg S_7 \vee (\neg S_8 \vee S_1)))))))$

$\Leftrightarrow \neg S_1 \vee \neg S_2 \vee \neg S_3 \vee \neg S_4 \vee \neg S_5 \vee \neg S_6 \vee \neg S_7 \vee \neg S_8 \vee S_1$

$\Leftrightarrow (\neg S_1 \vee S_1) \vee \neg S_2 \vee \neg S_3 \vee \neg S_4 \vee \neg S_5 \vee \neg S_6 \vee \neg S_7 \vee \neg S_8$

$\Leftrightarrow 1 \vee \neg S_2 \vee \neg S_3 \vee \neg S_4 \vee \neg S_5 \vee \neg S_6 \vee \neg S_7 \vee \neg S_8$

$\Leftrightarrow 1$

De fato, verificamos que H é uma tautologia. No entanto, será que não temos outras formas de verificar isso? Verificar se uma fórmula é tautologia ou não é o mesmo que provar um teorema, então vamos utilizar os métodos de prova que conhecemos da matemática para demonstrar as propriedades semânticas (historicamente esse caminho foi inverso, ao utilizar a lógica para dar mais rigor às provas matemáticas surgiram os métodos de prova que conhecemos).

Prova Direta

A prova direta de tautologias e contradições corresponde à manipulação algébrica das fórmulas até a obtenção de uma fórmula atômica “1” ou “0”.

Prova por Enumeração

A prova por enumeração (ou exaustão, ou força bruta) corresponde à tabela verdade. Nesta tabela, todas as interpretações, ou seja, atribuições de valores às proposições são exploradas. No entanto, é possível evitar algumas interpretações se combinarmos a enumeração com a prova direta.

Método da Árvore Semântica

Como dito anteriormente, este método pode ser entendido como uma forma de enumeração mais eficiente (em alguns casos). No pior dos casos, a árvore semântica dá tanto trabalho quanto a tabela verdade.

A ideia da construção da árvore semântica é atribuir valores aos símbolos proposicionais um a um, e não todos de uma vez, como numa linha da tabela verdade. A fórmula é avaliada a cada atribuição realizada. Como na tabela verdade, todas as

possibilidades devem ser exploradas, mas não necessariamente todas as interpretações serão avaliadas.

Exemplo: Considere a fórmula F = ((A → B) ∧ (B → C)) → (A → C).

Passo 1:

- Escolha uma proposição e atribua um valor lógico: I[A] = 0.
- Avalie a fórmula utilizando a atribuição feita no passo anterior:
  - ((0 → B) ∧ (B → C)) → (0 → C)
    ⇔ (1 ∧ (B → C)) → 1
    ⇔ 1
- Conclusão: I[F] = 1 se I[A] = 0, independentemente dos valores de B e C.

Passo 2:

- Como todos os casos de B e C já foram considerados para A falso, vamos considerar agora o caso em que A é verdadeiro: I[A] = 1.
- Avalie a fórmula utilizando a atribuição feita no passo anterior:
  - ((1 → B) ∧ (B → C)) → (1 → C)
- Conclusão: I[F] depende dos valores de B e C.

Passo 2.1:

- Como no passo anterior não chegamos ao valor de I[F], vamos manter a atribuição do passo anterior e agora atribuir um valor para outra proposição: I[B] = 0
- Avalie a fórmula utilizando as atribuições anteriores:
  - ((1 → B) ∧ (B → C)) → (1 → C)
    ⇔ ((1 → 0) ∧ (0 → C)) → (1 → C)
    ⇔ (0 ∧ 1) → (1 → C)
    ⇔ 0 → (1 → C)
    ⇔ 1
- Conclusão: I[F] = 1.

Passo 2.2:

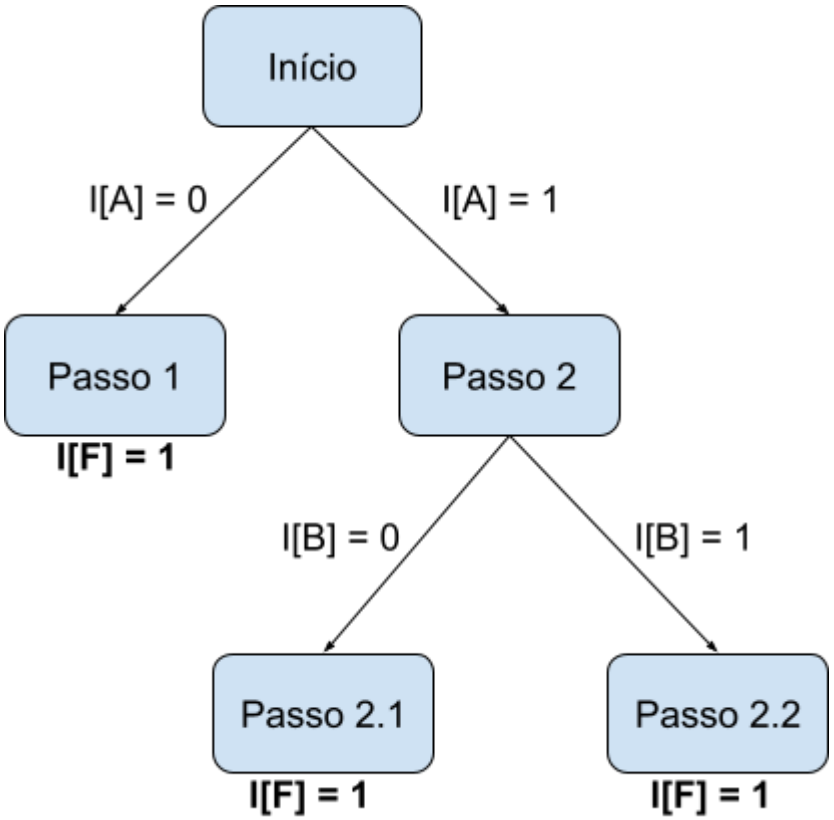
- No passo anterior chegamos a uma conclusão e agora vamos testar outra possibilidade para B mantendo o valor de A: I[B] = 1.
- Avalie a fórmula utilizando as atribuições anteriores:
  - ((1 → B) ∧ (B → C)) → (1 → C)
    ⇔ ((1 → 1) ∧ (1 → C)) → (1 → C)
    ⇔ (1 ∧ (1 → C)) → (1 → C)
    ⇔ (1 → C) → (1 → C)
    ⇔ 1
- Conclusão: I[F] = 1.

Perceba que todas as interpretações foram consideradas apesar de nem todas terem sido avaliadas. No passo 1 avaliamos que a fórmula é verdadeira se A é falso e qualquer valores de B e C, isso corresponde a quatro linhas da tabela verdade. No passo 2.1 avaliamos que a fórmula é verdadeira se A é verdadeiro e B é falso, independente do valor de C, correspondendo a duas linhas da tabela verdade. Finalmente, no passo 2.2 avaliamos que a fórmula é verdadeira se A e B são verdadeiras, independente do valor de C, correspondendo a mais duas linhas da tabela verdade.

Para facilitar a visualização, apresento a tabela verdade desta fórmula. As linhas em vermelho são determinadas no passo 1. As linhas em azul são determinadas no passo 2.1. As linhas em preto são determinadas no passo 2.2.

A	B	C	X = A → B	Y = B → C	Z = A → C	W = (X ∧ Y)	F = W → Z
0	0	0	1	1	1	1	1
0	0	1	1	1	1	1	1
0	1	0	1	0	1	0	1
0	1	1	1	1	1	1	1
1	0	0	0	1	0	0	1
1	0	1	0	1	1	0	1
1	1	0	1	0	0	0	1
1	1	1	1	1	1	1	1

Por que esse método é chamado de árvore semântica? Tem a ver com uma forma de representar os passos descritos anteriormente.



O estado “início” representa a fórmula antes de qualquer atribuição. O rótulo na seta que sai do “início” e vai para o “passo 1” indica a atribuição feita. Abaixo do “passo 1” está a conclusão obtida com a atribuição I[A] = 0. O mesmo raciocínio se aplica aos demais passos. No “passo 2” foi feita a atribuição I[A] = 1 mas nenhuma conclusão foi obtida, por isso, partem ramificações do “passo 2” para os demais passos, fazendo as atribuições indicadas nas setas. Em todos os “ramos” dessa árvore chegamos à conclusão que a fórmula é verdadeira, portanto ela é uma tautologia.

Prova por Contradição

Outra técnica muito utilizada para provar tautologias é a prova por contradição. Esta técnica de prova consiste em tomar uma direção oposta da que desejamos provar. Para provar uma tautologia, ao invés de perguntar “todos os casos são verdade?” perguntamos “existe algum caso falso?”.

Então consiste em assumir o oposto do que desejamos provar e mostramos que o oposto é impossível (contradição). Para provar tautologias, vamos assumir que a fórmula é falsa e verificar se isso é possível. Caso seja impossível ser falsa, concluímos que a fórmula é sempre verdadeira.

Mais uma vez, vamos usar a mesma fórmula dos exemplos anteriores como exemplo.

Exemplo: Considere a fórmula F = ((A → B) ∧ (B → C)) → (A → C).
Vamos assumir que esta fórmula é falsa, ou seja, I[F] = 0.
Temos então que I[((A → B) ∧ (B → C)) → (A → C)] = 0. Quais as condições para que isso ocorra? Devemos ter o antecedente verdadeiro e o consequente falso, ou seja:
I[((A → B) ∧ (B → C))] = 1 e I[(A → C)] = 0.
Para satisfazer a segunda condição, devemos novamente ter o antecedente verdadeiro e o consequente falso, ou seja I[A] = 1 e I[C] = 0.
Para satisfazer a primeira condição, devemos ter ambos verdadeiros, ou seja:
I[(A → B)] = 1 e I[(B → C)] = 1.
Para I[(A → B)] = 1 e I[A] = 1 (do argumento anterior), devemos ter I[B] = 1.
Para I[(B → C)] = 1 e I[C] = 0 (do argumento anterior), devemos ter I[B] = 0.
Como é impossível que B seja simultaneamente verdadeiro e falso, concluímos que é impossível que I[F] = 0. Logo, F é uma tautologia.

Atenção, para concluir que a fórmula é uma tautologia é necessário que a contradição ocorra em todas as tautologias. No exemplo anterior só há um caminho possível para que a fórmula seja falsa, mas nem sempre isso ocorre. Considere o próximo exemplo.

Exemplo: Considere a fórmula F = ((A ∧ B) ↔ A) ↔ (A → B).
Para que a fórmula seja falsa, I[F] = 0, há dois casos possíveis.
Caso 1: I[((A ∧ B) ↔ A)] = 1 e I[(A → B)] = 0.
Para termos I[(A → B)] = 0, devemos ter I[A] = 1 e I[B] = 0.

Com estes valores a primeira fórmula será avaliada como I[((1 ∧ 0) ↔ 1)] = I[(0 ↔ 1)] = 0.
O que é uma contradição pois precisávamos que essa condição fosse verdadeira.
Caso 2: I[((A ∧ B) ↔ A)] = 0 e I[(A → B)] = 1
Este caso vai se dividir em alguns subcasos porque para a primeira condição ser falsa há duas possibilidades.
Caso 2.1: I[(A ∧ B)] = 1 e I[A] = 0
Este caso já é uma contradição, pois se A é falso, a conjunção deve ser falsa.
Caso 2.2: I[(A ∧ B)] = 0 e I[A] = 1
Como já sabemos que I[A] = 1, concluímos que I[B] = 0 para satisfazer estas condições.
Com estas informações concluímos que I[(A → B)] = 0, que é uma contradição com a condição necessária no caso 2.

Neste exemplo, como há contradições em todos os casos, concluímos que a fórmula é uma tautologia e a equivalência é válida, ou seja, ((A ∧ B) ↔ A) ⇔ (A → B). No entanto, se há pelo menos um caso em que a contradição não ocorre, quer dizer que achamos uma (ou mais) interpretações que tornam a fórmula falsa, e concluímos que não é uma tautologia. Considere o próximo exemplo.

Exemplo: Considere a fórmula F = (A → B) ↔ (¬A → ¬B).
Há dois casos possíveis para a fórmula ser falsa.
Caso 1: I[A → B] = 1 e I[¬A → ¬B] = 0
Para que I[¬A → ¬B] = 0, devemos ter I[¬A] = 1 e I[¬B] = 0, ou seja, I[A] = 0 e I[B] = 1.
Com estes valores temos que a primeira condição é satisfeita, ou seja I[A → B] = 1. Portanto, com I[A] = 0 e I[B] = 1 a fórmula é falsa.
Caso 2: I[A → B] = 0 e I[¬A → ¬B] = 1
Para que I[A → B] = 0, devemos ter I[A] = 1 e I[B] = 0, que também não provoca uma contradição.

Apesar de nenhum caso ter contradição, não quer dizer que essa fórmula é sempre falsa. A tabela verdade desta fórmula é apresentada abaixo.

A	B	A → B	¬A → ¬B	(A → B) ↔ (¬A → ¬B)
0	0	1	1	1
0	1	1	0	0
1	0	0	1	0
1	1	1	1	1

Exemplo: Considere a fórmula F = A ↔ (A ∨ B).
Caso 1: I[A] = 1 e I[A ∨ B] = 0.

Este caso é uma contradição, pois se A é verdadeiro uma disjunção envolvendo A tem que ser verdadeira.
Caso 2: I[A] = 0 e I[A ∨ B] = 1.
Neste caso é possível satisfazer as duas condições, basta termos I[B] = 1. Daí concluímos que se A é falso e B é verdadeiro a fórmula é falsa.

A	B	A ∨ B	A ↔ (A ∨ B)
0	0	0	1
0	1	1	0
1	0	1	1
1	1	1	1

Este método é particularmente eficiente em fórmulas que envolvem implicações e disjunções, porque só há um caso em que uma fórmula desse tipo é falsa.

Indução Finita

Apesar de ser uma forma de prova amplamente utilizada na matemática, escolhi apresentá-la separadamente aqui porque é usada em uma situação levemente diferente das demais que vimos até agora. Nos demais casos as técnicas eram utilizadas para determinar a validade de uma determinada fórmula. A indução, em geral, é utilizada para determinar se uma certa afirmação é válida para todas as fórmulas da lógica.

Antes de enunciarmos o princípio da indução finita na lógica, vamos nos recordar do princípio da indução em outras situações. Muito provavelmente você já teve contato com a indução no contexto dos números naturais. Lá temos uma afirmação acerca dos números naturais, como por exemplo A(N): “a soma dos números naturais de 0 a N vale $(N^2 + N)/2$”, e para mostrar que tal afirmação é válida para qualquer número natural precisamos mostrar duas coisas: que a afirmação vale para o caso base ($N = 0$) e que se a afirmação vale para um certo valor de x de N, ela também vale para $x + 1$.

Mas por que essas duas provas são suficientes para mostrar que a afirmação é válida para todo o conjunto dos números naturais? A primeira prova, chamada de base da indução, nos garante que a afirmação é válida para $N = 0$, a partir daí, usando a segunda prova, podemos mostrar que vale para $N = 1$, e desse ponto podemos usar novamente a segunda prova para mostrar que vale para $N = 2$, e assim sucessivamente. A base e o passo de indução são os elementos que definem o conjunto dos naturais (o número 0 e a operação de sucessor) e, por isso, podemos construir uma prova para qualquer número natural usando estas duas afirmações.

Deste modo, nada nos impede de utilizar este raciocínio para definir outros conjuntos. Se nossa base de indução é 0 e o passo de indução é $x + 2$, ou invés de $x + 1$, vamos demonstrar que a afirmação é válida para todos os números pares. Se a base é 1 e o passo $2x$, vamos mostrar que vale para todas as potências de dois. Se a base de indução é 0 e nossos passos de indução são $x + 1$ e $x - 1$, então mostramos que a afirmação vale para todos os números inteiros.

Assim, para estender o raciocínio da indução para a lógica, basta que a base e o passo de indução definam todas as fórmulas da lógica proposicional. Já vimos que temos três casos de fórmulas básicas (atômicas) e cinco formas de combinar fórmulas para gerar novas fórmulas (os conectivos). Portanto, podemos enunciar o princípio da indução finita na lógica proposicional da seguinte forma:

Seja A(F) uma afirmação acerca de uma fórmula F da lógica proposicional. Para demonstrar que A(F) é válida para qualquer fórmula F da lógica proposicional, basta demonstrar que:

(Base de Indução):

- A(P), onde P é um símbolo proposicional, é válida;
- A(1) é válida;
- A(0) é válida;

(Passo de Indução):

- Se A(G) é válida, então A(¬G) é válida;
- Se A(G) e A(H) são válidas, então A(G $\wedge$ H) é válida;
- Se A(G) e A(H) são válidas, então A(G $\vee$ H) é válida;
- Se A(G) e A(H) são válidas, então A(G $\rightarrow$ H) é válida;
- Se A(G) e A(H) são válidas, então A(G $\leftrightarrow$ H) é válida;

Mais uma vez, perceba que as bases e passos de indução definem todas as fórmulas possíveis, conforme definido no capítulo sobre sintaxe. Muito trabalho, não é? Será que não conseguimos deixar esse processo mais simples? Para iniciar, perceba que não é realmente necessário mostrar que A(1) e A(0) são válidas, basta mostrar um dos dois casos, já que podemos definir 1 como ¬0 e vice-versa. E quanto aos demais casos?

Conjuntos Completos de Conectivos

Na definição utilizada da lógica proposicional, definimos um conjunto de cinco conectivos {¬, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$}, que não são todos os conectivos possíveis mas já discutimos que todos os outros podem ser definidos a partir destes cinco. Por isso, este conjunto de conectivos é chamado de conjunto completo, pois todas as operações lógicas podem ser definidas a partir deles.

Existem outros conjuntos além desse? Com certeza! Já vimos, através das equivalências, que é possível escrever uma bi-implicação utilizando duas implicações ((A $\leftrightarrow$ B) $\Leftrightarrow$ ((A $\rightarrow$ B) $\wedge$ (B $\rightarrow$ A))). Portanto, toda fórmula que contém bi-implicações pode ser reescrita de forma equivalente sem esse conectivo. Daí concluímos que o conjunto {¬, $\wedge$, $\vee$, $\rightarrow$} é um conjunto completo de conectivos.

Do mesmo modo, já vimos que a implicação pode ser escrita como uma disjunção ((A $\rightarrow$ B) $\Leftrightarrow$ (¬A $\vee$ B)). Por uma raciocínio análogo, o conectivo de implicação pode ser escrito em termos da disjunção e da negação, logo {¬, $\wedge$, $\vee$} é um conjunto completo de conectivos.

Utilizando as regras de DeMorgan podemos reescrever a conjunção em função de disjunções e negações ((A $\wedge$ B) $\Leftrightarrow$ ¬(¬A $\vee$ ¬B)) ou a disjunção em função de conjunções e negações ((A $\vee$ B) $\Leftrightarrow$ ¬(¬A $\wedge$ ¬B)). Utilizando estas relações chegamos à conclusão que os conjuntos {¬, $\vee$} e {¬, $\wedge$} também são conjuntos completos de conectivos.

Portanto, apenas dois conectivos são suficientes para definir todas as operações possíveis da lógica proposicional. Na verdade, se formos além, podemos utilizar conectivos não definidos originalmente para reduzir esse conjunto a um único conectivo.

Considere o operador $\barwedge$, definido através da seguinte tabela verdade:

A	B	A $\barwedge$ B
0	0	1
0	1	1
1	0	1
1	1	0

Perceba que A $\barwedge$ A $\Leftrightarrow$ ¬ A:

A	¬A	A $\barwedge$ A
0	1	1
1	0	0

e que ((A $\barwedge$ B) $\barwedge$ (A $\barwedge$ B)) $\Leftrightarrow$ (A $\wedge$ B):

A	B	A $\barwedge$ B	((A $\barwedge$ B) $\barwedge$ (A $\barwedge$ B))	(A $\wedge$ B)
0	0	1	0	0
0	1	1	0	0
1	0	1	0	0
1	1	0	1	1

Ou seja, o conjunto {$\barwedge$} é um conjunto completo de conectivos. Esse operador é comumente conhecido como *nand* (junção de *not* e *and*) pois (A $\barwedge$ B) $\Leftrightarrow$ ¬(A $\wedge$ B). Na eletrônica este operador é particularmente comum. Em circuitos lógicos programáveis era comum ter à disposição do usuário uma certa quantidade de operadores $\wedge$, $\vee$ e ¬ que podiam ser combinados de certas maneiras. No entanto, a decisão de qual conectivo ter mais ou menos sempre era um problema. Além disso, a produção de muitos componentes de um único tipo é mais barata que a produção de múltiplos componentes. Com o tempo, as portas *nand* se tornaram mais frequentes nos circuitos programáveis, pois elas podem assumir o papel de qualquer um dos outros operadores.

Vale ressaltar que, apesar de ser equivalente, do ponto de vista lógico, a uma conjunção seguida de uma negação, do ponto de vista da eletrônica, fazer um conectivo $\wedge$ e um $\barwedge$ tem o mesmo custo. Portanto, na eletrônica, uma conjunção seguida de uma negação é mais cara que um único *nand*.

Fechado esse parêntese sobre as implicações práticas dos conjuntos de conectivos completos, vamos às implicações teóricas. Esta seção iniciou-se após a pergunta se poderíamos simplificar a prova por indução para não ter de fazer tantos casos. O fato é que os passos de indução podem ser qualquer conjunto completo de conectivos lógicos. Na

definição dada na seção anterior utilizamos o conjunto original {¬, ∧, ∨, →, ↔}. Mas, como toda fórmula pode ser escrita utilizando exclusivamente os conectivos de um conjunto completo qualquer, poderíamos definir a indução lógica da seguinte forma:

Seja A(F) uma afirmação acerca de uma fórmula F da lógica proposicional. Para demonstrar que A(F) é válida para qualquer fórmula F da lógica proposicional, basta demonstrar que:
(Base de Indução):

- A(P), onde P é um símbolo proposicional, é válida;
- A(1) é válida;

(Passo de Indução):

- Se A(G) e A(H) são válidas, então A(G ⍤ H) é válida;

Fazendo uma analogia ao caso dos números naturais, poderíamos definir o conjunto dos naturais como sendo:

- 0 é um número natural;
- 1 é um número natural;
- Se *x* é um número natural, *x* + 2 também é.

Os dois primeiros casos são casos base e o último é o passo de indução. A partir do primeiro caso base mais o passo definimos todos os números pares e a partir do segundo caso base definimos todos os ímpares. É uma forma diferente, mais complicada até, mas que é equivalente à definição

- 0 é um número natural;
- Se *x* é um número natural, *x* + 1 também é.

Formas Normais

Além de podermos escrever as fórmulas da lógica proposicional utilizando certos conjuntos de conectivos, os conjuntos completos, também existem certas estruturas pré-definidas nas quais, para qualquer fórmula da lógica proposicional, existe uma fórmula equivalente utilizando esta estrutura. Estas estruturas são chamadas de formas normais.

Neste texto trabalharemos com duas formas normais, definidas como seguem:

1. **Forma Normal Disjuntiva (FND):**
Uma fórmula F está na forma normal disjuntiva (FND ou DNF - *Disjunctive Normal Form*), se é uma disjunção de conjunções de literais.

2. **Forma Normal Conjuntiva (FNC):**
Uma fórmula F está na forma normal conjuntiva (FNC ou CNF - Conjunctive Normal Form), se é uma conjunção de disjunções de literais.

Para entender as definições anteriores, precisamos definir o que são literais:

Literal:
Um literal, na lógica proposicional, é um símbolo proposicional ou sua negação.

Destas definições podemos concluir que fórmulas nas formas normais utilizam apenas o conjunto completo de conectivos {¬, ∧, ∨} e, além disso, o operador de negação

só aparece antes de símbolos proposicionais, nunca é aplicado a uma disjunção ou conjunção.

Obtenção das Formas Normais

Primeiramente, vamos obter as formas normais algebricamente, através das relações que já conhecemos. Os seguintes passos devem ser seguidos para obtenção de fórmulas na CNF ou DNF:

1. Substitua toda ocorrência de bi-implicação por implicações utilizando a equivalência $(A \leftrightarrow B) \Leftrightarrow ((A \rightarrow B) \wedge (B \rightarrow A))$.

2. Substitua toda ocorrência de implicação por disjunções utilizando a equivalência $(A \rightarrow B) \Leftrightarrow (\neg A \vee B)$.

Ao fim destes dois passos temos apenas símbolos do conjunto $\{\neg, \wedge, \vee\}$ na fórmula. Agora vamos fazer com que as negações ocorram apenas nos símbolos proposicionais.

3. Internalize as negações e remova as negações duplas utilizando repetidamente as leis de DeMorgan:
    a. $\neg(A \wedge B) \Leftrightarrow (\neg A \vee \neg B)$
    b. $\neg(A \vee B) \Leftrightarrow (\neg A \wedge \neg B)$
    c. $\neg\neg A \Leftrightarrow A$

Neste ponto, se você aplicou sempre que possível tais leis, temos uma fórmula onde as negações ocorrem apenas nos símbolos proposicionais. Agora vamos organizar a fórmula para uma conjunção de disjunções (CNF) ou disjunção de negações (DNF).

4. Para obter a fórmula no formato desejado, utilize as regras de distributividade:
    a. $A \vee (B \wedge C) \Leftrightarrow (A \vee B) \wedge (A \vee C)$
    b. $A \wedge (B \vee C) \Leftrightarrow (A \wedge B) \vee (A \wedge C)$

Exemplo: Considere a fórmula $F = (A \rightarrow B) \wedge C$

1. Não se aplica.
2. $F \Leftrightarrow (\neg A \vee B) \wedge C$

Neste momento a fórmula já está na forma normal conjuntiva, pois temos uma conjunção de disjunções, a primeira disjunção sendo $(\neg A \vee B)$ e a segunda sendo apenas o C. Para levar à forma normal disjuntiva, vamos utilizar a distributividade:

$F \Leftrightarrow (\neg A \vee B) \wedge C$
$\Leftrightarrow (\neg A \wedge C) \vee (B \wedge C)$

Exemplo: Considere a fórmula $G = (A \rightarrow B) \leftrightarrow (C \wedge A)$

1. $G \Leftrightarrow ((A \rightarrow B) \rightarrow (C \wedge A)) \wedge ((C \wedge A) \rightarrow (A \rightarrow B))$
2. Há 4 implicações:

a. $G \Leftrightarrow ((\neg A \vee B) \rightarrow (C \wedge A)) \wedge ((C \wedge A) \rightarrow (A \rightarrow B))$
b. $G \Leftrightarrow ((\neg A \vee B) \rightarrow (C \wedge A)) \wedge ((C \wedge A) \rightarrow (\neg A \vee B))$
c. $G \Leftrightarrow (\neg(\neg A \vee B) \vee (C \wedge A)) \wedge ((C \wedge A) \rightarrow (\neg A \vee B))$
d. $G \Leftrightarrow (\neg(\neg A \vee B) \vee (C \wedge A)) \wedge (\neg(C \wedge A) \vee (\neg A \vee B))$

3. Internalizando negações e removendo negações duplas:
   a. $G \Leftrightarrow ((\neg\neg A \wedge \neg B) \vee (C \wedge A)) \wedge (\neg(C \wedge A) \vee (\neg A \vee B))$
   b. $G \Leftrightarrow ((A \wedge \neg B) \vee (C \wedge A)) \wedge (\neg(C \wedge A) \vee (\neg A \vee B))$
   c. $G \Leftrightarrow ((A \wedge \neg B) \vee (C \wedge A)) \wedge ((\neg C \vee \neg A) \vee (\neg A \vee B))$
4. Levando para a forma normal conjuntiva:
   a. $G \Leftrightarrow ((A \wedge \neg B) \vee (C \wedge A)) \wedge (\neg C \vee \neg A \vee \neg A \vee B)$
   b. $G \Leftrightarrow ((A \wedge \neg B) \vee (C \wedge A)) \wedge (\neg C \vee \neg A \vee B)$
   c. $G \Leftrightarrow (((A \wedge \neg B) \vee C) \wedge ((A \wedge \neg B) \vee A)) \wedge (\neg C \vee \neg A \vee B)$
   d. $G \Leftrightarrow ((A \vee C) \wedge (\neg B \vee C)) \wedge ((A \vee A) \wedge (\neg B \vee A)) \wedge (\neg C \vee \neg A \vee B)$
   e. $G \Leftrightarrow (A \vee C) \wedge (\neg B \vee C) \wedge (A \vee A) \wedge (\neg B \vee A) \wedge (\neg C \vee \neg A \vee B)$
   f. $G \Leftrightarrow (A \vee C) \wedge (\neg B \vee C) \wedge A \wedge (\neg B \vee A) \wedge (\neg C \vee \neg A \vee B)$
5. Levando para a forma normal disjuntiva (vou partir do ponto 4.b):
   a. $G \Leftrightarrow ((A \wedge \neg B) \vee (C \wedge A)) \wedge (\neg C \vee \neg A \vee B)$
   b. $G \Leftrightarrow ((A \wedge \neg B) \wedge (\neg C \vee \neg A \vee B)) \vee ((C \wedge A) \wedge (\neg C \vee \neg A \vee B))$
   c. $G \Leftrightarrow (A \wedge \neg B \wedge \neg C) \vee (A \wedge \neg B \wedge \neg A) \vee (A \wedge \neg B \wedge B) \vee ((C \wedge A) \wedge (\neg C \vee \neg A \vee B))$
   d. $G \Leftrightarrow (A \wedge \neg B \wedge \neg C) \vee (C \wedge A \wedge \neg C) \vee (C \wedge A \wedge \neg A) \vee (C \wedge A \wedge B)$
   e. $G \Leftrightarrow (A \wedge \neg B \wedge \neg C) \vee (C \wedge A \wedge B)$

Parece trabalhoso, e às vezes realmente é, mas é um procedimento que pode ser automatizado por uma máquina. Nos passos 5.c e 5.d do exemplo anterior os itens marcados em vermelho são literais complementares na mesma conjunção e por isso aquelas conjunções foram simplificadas para 0, e desapareceram da fórmula.

Uma outra forma de se obter as formas normais conjuntiva e disjuntiva é a partir da tabela verdade. Vamos rever os mesmos exemplos:

Exemplo: Considere a fórmula $F = (A \rightarrow B) \wedge C$
Tabela Verdade:

A	B	C	$A \rightarrow B$	$(A \rightarrow B) \wedge C$
0	0	0	1	0
0	0	1	1	1
0	1	0	1	0
0	1	1	1	1
1	0	0	0	0
1	0	1	0	0

1	1	0	1	0
1	1	1	1	1

Uma conjunção só é verdadeira se todos os seus elementos são verdadeiros. Já uma disjunção, é verdadeira se pelo menos um dos seus elementos é verdadeiro. Deste modo, podemos obter uma fórmula na forma normal disjuntiva olhando para as linhas da tabela verdade onde o resultado é verdadeiro. Vamos usar cada uma das conjunções da FND para determinar uma linha da tabela verdade onde a fórmula é verdadeira.

Na linha 2, a fórmula é verdadeira com A e B falsos e C verdadeiro, a conjunção (¬A ∧ ¬B ∧ C) só é verdadeira nesta situação. Na linha 4, a fórmula é verdadeira com A falso e B e C verdadeiros, a conjunção (¬A ∧ B ∧ C) representa esta situação. Finalmente, a conjunção (A ∧ B ∧ C) só é verdadeira na situação presente na última linha da tabela.

Fazendo a disjunção destas três conjunções temos uma fórmula que é equivalente a F e está na forma normal disjuntiva:

F1 = (¬A ∧ ¬B ∧ C) ∨ (¬A ∧ B ∧ C) v (A ∧ B ∧ C)

e também deve ser equivalente à fórmula obtida através da manipulação algébrica:

F2 = (¬A ∧ C) ∨ (B ∧ C)

A conjunção (¬A ∧ C) é verdadeira se A é falso e C é verdadeiro, independente do valor de B, o que corresponde às duas primeiras conjunções de F1. Já a conjunção (B ∧ C) é verdade se B e C são verdadeiros, independente da interpretação de A, que corresponde às duas últimas conjunções de F1. Deste modo as duas fórmulas são equivalentes. Vamos verificar através da tabela verdade:

A	B	C	¬A∧¬B∧C	¬A∧B∧C	A∧B∧C	F1	¬A∧C	B∧C	F2
0	0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	1	1	0	1
0	1	0	0	0	0	0	0	0	0
0	1	1	0	1	0	1	1	1	1
1	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0
1	1	1	0	0	1	1	0	1	1

De forma análoga, uma disjunção só é falsa se todos os seus elementos são falsos. Já uma conjunção, é falsa se pelo menos um dos seus elementos é falso. Deste modo, podemos obter uma fórmula na forma normal conjuntiva olhando para as linhas da tabela verdade onde o resultado é falso (em vermelho). Vamos usar cada uma das disjunções da FNC para determinar uma linha da tabela verdade onde a fórmula é falsa.

Na linha 1, a fórmula é falsa com A, B e C falsos, a disjunção (A ∨ B ∨ C) só é falsa nesta situação. Na linha 3, a fórmula é falsa com A e C falsos e B verdadeiro, a disjunção (A ∨ ¬B ∨ C) representa esta situação. Repetindo o raciocínio, teremos as disjunções (¬A ∨ B ∨ C), (¬A ∨ B ∨ ¬C) e (¬A ∨ ¬B ∨ C) para as linhas 5, 6 e 7, respectivamente.

Fazendo a conjunção destas cinco disjunções temos uma fórmula que é equivalente a F e está na forma normal conjuntiva:

F3 = (A ∨ B ∨ C) ∧ (A ∨ ¬B ∨ C) ∧ (¬A ∨ B ∨ C) ∧ (¬A ∨ B ∨ ¬C) ∧ (¬A ∨ ¬B ∨ C)

e também deve ser equivalente à fórmula obtida através da manipulação algébrica:

F4 = (¬A ∨ B) ∧ C

C sozinho, em F4, é falso se C é falso, independente dos valores de A e B, isso corresponde a quatro das conjunções de F3. Mais uma vez, vamos verificar a tabela verdade:

A	B	C	¬A∨¬B∨C	A∨¬B∨C	¬A∨B∨C	A∨B∨C	¬A∨B∨¬C	F3	¬A ∨ B	F4
0	0	0	1	1	1	0	1	0	1	0
0	0	1	1	1	1	1	1	1	1	1
0	1	0	1	0	1	1	1	0	1	0
0	1	1	1	1	1	1	1	1	1	1
1	0	0	1	1	0	1	1	0	0	0
1	0	1	1	1	1	1	0	0	0	0
1	1	0	0	1	1	1	1	0	1	0
1	1	1	1	1	1	1	1	1	1	1

De modo geral, o procedimento pode ser resumido como segue. Para obter a FND a partir da tabela verdade vamos observar as linhas verdadeiras da tabela e escrever uma conjunção para cada linha. Se o símbolo é verdadeiro na linha, ele será um literal positivo (sem negação) na conjunção, se o símbolo é falso, ele será um literal negado.

Para obter a FNC a partir da tabela verdade vamos observar as linhas falsas da tabela e escrever uma disjunção para cada linha. Se o símbolo é verdadeiro na linha, ele será um literal negativo (negado) na disjunção, se o símbolo é falso, ele será um literal positivo.

No exemplo anterior, note que todas as fórmulas (F, F1, F2, F3 e F4) são equivalentes entre si. De modo geral, as fórmulas obtidas através da tabela verdade são mais longas. Utilizando o procedimento apresentado, numa fórmula com K símbolos proposicionais onde X interpretações são verdadeiras e Y são falsas, teremos KX literais na FND e KY literais na FNC.

Outro ponto importante é que é necessário criar a tabela verdade para utilizar este método, o que pode ser desgastante para fórmulas com muitos símbolos proposicionais. No entanto, é muito comum em problemas de eletrônica a necessidade de síntese de circuitos lógicos, onde sabemos o comportamento desejado para as entradas (a tabela verdade), mas não conhecemos a fórmula. Nestes casos o método apresentado aqui é utilizado para obtenção da fórmula e, posteriormente, os métodos algébricos são utilizados para simplificar a fórmula. A simplificação da fórmula é muito importante na eletrônica pois representa uma redução de custos.

Exemplo: Considere a fórmula G = (A → B) ↔ (C ∧ A)
Tabela Verdade:

A	B	C	A → B	A ∧ C	(A → B) ↔ (C ∧ A)
0	0	0	1	0	0
0	0	1	1	0	0
0	1	0	1	0	0
0	1	1	1	0	0
1	0	0	0	0	1
1	0	1	0	1	0
1	1	0	1	0	0
1	1	1	1	1	1

FND:
Só há dois casos positivos:
F1 = (A ∧ ¬B ∧ ¬C) ∨ (A ∧ B ∧ C)
Que foi a mesma fórmula obtida através da manipulação algébrica.

FNC:
Há seis casos falsos:
F2 = (A∨B∨C) ∧ (A∨B∨¬C) ∧ (A∨¬B∨C) ∧ (A∨¬B∨¬C) ∧ (¬A∨¬B ∨C) ∧ (¬A∨B∨¬C)
No caso algébrico obtivemos a fórmula:
F3 = (A ∨ C) ∧ (¬B ∨ C) ∧ A ∧ (¬B ∨ A) ∧ (¬C ∨ ¬A ∨ B)
A fórmula F3 é um pouco mais simples mas poderia ser melhor. Observe a tabela verdade e perceba que sempre que A é falso o resultado é falso, independente das outras proposições. Isto está explicitado em F3 pela proposição A sozinha. Isto quer dizer que o A sozinho pode substituir todas as disjunções onde o A não aparece negado.
F3 = (A ∨ C) ∧ (¬B ∨ C) ∧ A ∧ (¬B ∨ A) ∧ (¬C ∨ ¬A ∨ B)
F3 ⇔ (¬B ∨ C) ∧ A ∧ (¬C ∨ ¬A ∨ B)
Usando a mesma idéia:

F2 ⇔ A ∧ (¬A∨¬B ∨C) ∧ (¬A∨B∨¬C)

Por fim, mas não menos importante, perceba que este método de obtenção da FND e FNC a partir da tabela verdade são uma prova construtiva que, de fato, toda fórmula lógica tem um equivalente nas formas normais, uma vez que toda fórmula tem uma tabela verdade associada.

Sistemas de Dedução

Até agora estudamos a definição da linguagem da lógica proposicional e já temos as ferramentas necessárias para representar conhecimentos nessa linguagem. A próxima etapa é estudar as ferramentas para deduzir novos conhecimentos a partir do que já foi representado. Este é o mesmo processo que passamos ao programar, ao codificar um programa, estamos representando um conhecimento em uma linguagem formal e ao executar o programa estamos obtendo novos conhecimentos a partir das informações conhecidas (entradas + programa).

Vamos estudar três sistemas de dedução completamente sintáticos, ou seja, a inferência é realizada exclusivamente através da manipulação simbólica, sem que seja necessário o conhecimento do significado dos símbolos. Este processo puramente sintático é um dos objetos de estudo da lógica, que são os mecanismos ou estruturas de dedução.

Outra vantagem deste processo é que eles são adequados à implementação nos computadores digitais, que são máquinas puramente sintáticas.

Os três sistemas apresentados aqui são corretos, ou seja, toda prova obtida pelos sistemas são tautologias da lógica proposicional, e completos, ou seja, toda tautologia da lógica proposicional pode ser provada pelos sistemas. Não é nosso objetivo aqui, mas é possível demonstrar estas propriedades para os sistemas apresentados.

Além disso, os sistemas são consistentes, ou seja, se há uma prova de que H é uma tautologia, não é possível existir uma prova de que ¬H seja tautologia. Um conjunto de fórmulas □ é consistente se, e somente se, não existe fórmula F tal que F e ¬F podem ser deduzidos simultaneamente a partir de □. Aproveitando esta definição, podemos concluir que um conjunto de fórmulas é consistente se, e somente se, é satisfatível.

Sistema Axiomático

O primeiro sistema que vamos estudar é baseado em axiomas. Este sistema tem uma única regra de inferência e três axiomas. Como todo sistema axiomático, os axiomas são afirmações que assumimos que são verdadeiras, mas que não podem ser provadas através do próprio sistema.

Os três axiomas deste sistema são os seguintes:

- A_1 = ¬(X ∨ X) ∨ X
- A_2 = ¬X ∨ (Y ∨ X)
- A_3 = ¬(¬X ∨ Y) ∨ (¬(Z ∨ X) ∨ (Y ∨ Z))
-

Os axiomas são definidos apenas para o alfabeto {¬, ∨}. Por isso, vamos considerar além dos axiomas as seguintes equivalências como válidas para poder tratar todos os conectivos da lógica proposicional que adotamos.

- $(X \rightarrow Y) \Leftrightarrow (\neg X \vee Y)$
- $(X \leftrightarrow Y) \Leftrightarrow ((X \rightarrow Y) \wedge (Y \rightarrow X))$
- $(X \wedge Y) \Leftrightarrow \neg(\neg X \vee \neg Y)$
-

Deste modo, os axiomas podem ser reescritos, equivalentemente, de outras formas, como por exemplo:

- $A_1 = (X \vee X) \rightarrow X$
- $A_2 = X \rightarrow (Y \vee X)$
- $A_3 = (X \rightarrow Y) \rightarrow ((Z \vee X) \rightarrow (Y \vee Z))$
-

Como dito anteriormente, estes axiomas não podem ser provados utilizando o próprio sistema axiomático. No entanto, é relativamente fácil aceitar que eles são verdadeiros ou, excepcionalmente, podemos usar os métodos de prova dos capítulos anteriores para demonstrar que estas fórmulas são tautologias.

- $A_1 = \neg(X \vee X) \vee X \Leftrightarrow \neg X \vee X \Leftrightarrow 1$

- $A_2 = \neg X \vee (Y \vee X) \Leftrightarrow (\neg X \vee X) \vee Y \Leftrightarrow 1 \vee Y \Leftrightarrow 1$

- $A_3 = \neg(\neg X \vee Y) \vee (\neg(Z \vee X) \vee (Y \vee Z))$
  $A_3 \Leftrightarrow (X \wedge \neg Y) \vee (\neg Z \wedge \neg X) \vee (Y \vee Z)$
  $A_3 \Leftrightarrow ((X \vee \neg Z) \wedge (X \vee \neg X) \wedge (\neg Y \vee \neg Z) \wedge (\neg Y \vee \neg X)) \vee (Y \vee Z)$
  $A_3 \Leftrightarrow ((X \vee \neg Z) \wedge (\neg Y \vee \neg Z) \wedge (\neg Y \vee \neg X)) \vee (Y \vee Z)$
  $A_3 \Leftrightarrow (X \vee \neg Z \vee Y \vee Z) \wedge (\neg Y \vee \neg Z \vee Y \vee Z) \wedge (\neg Y \vee \neg X \vee Y \vee Z)$
  $A_3 \Leftrightarrow 1 \wedge 1 \wedge 1 \Leftrightarrow 1$

Utilizando as equivalências já conhecidas dos capítulos anteriores, chegamos à conclusão que todos os axiomas são tautologias. Além dos três axiomas e das três equivalências, o mecanismo que será utilizado para a dedução de conhecimentos é conhecido como *modus ponens* e pode ser enunciado como segue:

- $X \wedge (X \rightarrow Y) \Rightarrow Y$

Ou seja, sabendo que X é verdade e que $(X \rightarrow Y)$ também é, deduza que Y é verdade. Apesar de serem listados apenas 3 axiomas e uma regra de inferências, na verdade eles são infinitos visto que os elementos X, Y e Z nestas expressões são quaisquer fórmulas da lógica proposicional.

Por fim, vamos definir o mecanismo de prova no sistema axiomático:

Seja F uma fórmula e um conjunto de fórmulas, denominadas de hipóteses. Uma prova de F a partir de , no sistema axiomático, é uma sequência de fórmulas $F_1, F_2, \ldots, F_n$, onde se tem

- $F = F_n$

E para todo *i* tal que $1 \leq i \leq n$,

- F_i é um axioma ou,
- $F_i \in \square$ ou,
- F_i é deduzida de F_j e F_k, utilizando a regra *modus ponens*, onde $j,k < i$. Neste caso, necessariamente $F_k = F_j \rightarrow F_i$.

Se a prova existe, dizemos que F é consequência lógica de □. Ou, que □ implica em F. Denotaremos esta afirmação por $\square \vDash F$ ou $\square \Rightarrow F$. Se o conjunto de hipóteses é vazio, ou seja, se F pode ser provado apenas a partir dos axiomas, dizemos que F é um teorema do sistema axiomático e denotaremos por $\vDash F$.

A seguir apresento alguns exemplos de provas utilizando o sistema axiomático. Perceba que nestes exemplos utilizaremos apenas o sistema (3 axiomas, 3 equivalências e *modus ponens*), nenhuma informação anteriormente conhecida será utilizada.

Exemplo 1: Sejam A, B e C três fórmulas. Tem-se que:
$((A \rightarrow B) \wedge (C \vee A)) \Rightarrow (B \vee C)$
Prova:

1.	$(A \rightarrow B)$	[hipótese]
2.	$(A \rightarrow B) \rightarrow ((C \vee A) \rightarrow (B \vee C))$	[A_3 com X = A, Y = B e Z = C]
3.	$(C \vee A) \rightarrow (B \vee C)$	[*modus ponens* de 1. e 2.]
4.	$(C \vee A)$	[hipótese]
5.	$(B \vee C)$	[*modus ponens* de 3. e 4.]

A partir das hipóteses chegamos à fórmula desejada. Perceba que a cada passo o procedimento de prova foi obedecido, ou é uma hipótese, ou é um axioma ou é a aplicação do *modus ponens* sobre passos anteriores. Na coluna da direita indicamos como foi obtida a afirmação daquela linha. Deste ponto em diante vamos indicar o *modus ponens* de X e Y por MP(X, Y).

Exemplo 2: Tem-se que $\vdash (P \vee \neg P)$
Prova:

1.	$((P \vee P) \rightarrow P) \rightarrow ((\neg P \vee (P \vee P)) \rightarrow (P \vee \neg P))$	[A_3 com X=$(P \vee P)$, Y=P e Z=¬P]
2.	$(P \vee P) \rightarrow P$	[A_1 com X=P]
3.	$(\neg P \vee (P \vee P)) \rightarrow (P \vee \neg P)$	[MP(1, 2)]
4.	$P \rightarrow (P \vee P)$	[A_2 com X=P e Y=P]
5.	$\neg P \vee (P \vee P)$	[equivalência]
6.	$(P \vee \neg P)$	[MP(3, 5)]

Exemplo 3: Tem-se que $\vdash (P \rightarrow \neg\neg P)$
Prova:

1.	$(X \vee \neg X)$	[Do Ex. 2]
2.	$(\neg P \vee \neg\neg P)$	[De 1. com X = ¬P]

3. $(P \rightarrow \neg\neg P)$ [equivalência]

Exemplo 4: Tem-se que $\vdash (\neg\neg P \rightarrow P)$

Prova:

1. $(\neg P \rightarrow \neg\neg\neg P) \rightarrow ((P \vee \neg P) \rightarrow (\neg\neg\neg P \vee P))$ [A_3 com X=¬P, Y=¬¬¬P e Z=P]
2. $(\neg P \rightarrow \neg\neg\neg P)$ [Do Ex. 3]
3. $(P \vee \neg P) \rightarrow (\neg\neg\neg P \vee P)$ [MP(1, 2)]
4. $(P \vee \neg P)$ [Do Ex. 2]
5. $(\neg\neg\neg P \vee P)$ [MP(4, 3)]
6. $(\neg\neg P \rightarrow P)$ [equivalência]

Exemplo 5: Tem-se que $\vdash (P \rightarrow P)$

Prova:

1. $(P \rightarrow \neg\neg P)$ [Do Ex. 3]
2. $(P \rightarrow \neg\neg P) \rightarrow ((P \vee P) \rightarrow (\neg\neg P \vee P))$ [A_3 com X = P, Y = ¬¬P e Z = P]
3. $(P \vee P) \rightarrow (\neg\neg P \vee P)$ [MP(1, 2)]
4. $(P \rightarrow \neg\neg P) \rightarrow ((\neg\neg P \vee P) \rightarrow (\neg\neg P \vee \neg\neg P))$ [A_3 com X=P, Y=¬¬P e Z=¬¬P]
5. $(\neg\neg P \vee P) \rightarrow (\neg\neg P \vee \neg\neg P)$ [MP(1,4)]
6. $\neg P \vee (P \vee P)$ [A_2 com X = P e Y = P]
7. $(\neg\neg P \vee P) \vee \neg P$ [Ex. 1 sobre 3. e 6.]
8. $(\neg P \rightarrow \neg\neg\neg P)$ [Do Ex. 3]
9. $\neg\neg\neg P \vee (\neg\neg P \vee P)$ [Ex. 1 sobre 8. e 7.]
10. $(\neg\neg P \vee \neg\neg P) \vee \neg\neg\neg P$ [Ex. 1 sobre 5. e 9.]
11. $(\neg\neg\neg P \rightarrow \neg P)$ [Do Ex. 4]
12. $\neg P \vee (\neg\neg P \vee \neg\neg P)$ [Ex. 1 sobre 11. e 10.]
13. $(\neg\neg P \vee \neg\neg P) \rightarrow \neg\neg P$ [A_1 com X = ¬¬P]
14. $\neg\neg P \vee \neg P$ [Ex. 1 sobre 13. e 12.]
15. $(\neg\neg P \rightarrow P)$ [Do Ex. 4]
16. $\neg\neg\neg P \vee \neg\neg P$ [De 14.]
17. $P \vee \neg\neg\neg P$ [Ex. 1 sobre 15. e 16.]
18. $\neg P \vee P$ [Ex. 1 sobre 11. e 17.]
19. $(P \rightarrow P)$ [equivalência]

Perceba que a prova utilizando o sistema axiomático nem sempre é óbvia. Devido à grande quantidade de opções que temos para as substituições e ações a cada passo, a busca pelo caminho correto pode ser muito longa. Neste exemplo, utilizamos também os resultados dos exemplos anteriores para simplificar o procedimento.

Exemplo 6: Tem-se que $(A \vee B) \Rightarrow (B \vee A)$

Prova:

1. $(P \rightarrow P)$ [Ex. 5]

2. $(B \rightarrow B)$ [De 1.]
3. $(B \rightarrow B) \rightarrow ((A \vee B) \rightarrow (B \vee A))$ [A_3 com X=B, Y=B e Z=A]
4. $(A \vee B) \rightarrow (B \vee A)$ [MP(2, 3)]
5. $(A \vee B)$ [hipótese]
6. $(B \vee A)$ [MP(5, 4)]

Exemplo 7: Tem-se que $((A \rightarrow B) \wedge (B \rightarrow C)) \Rightarrow (A \rightarrow C)$
Prova:

1. $(\neg A \vee B)$ [hipótese]
2. $(B \rightarrow C)$ [hipótese]
3. $C \vee \neg A$ [Ex. 1 sobre 2. e 1.]
4. $\neg A \vee C$ [Do Ex. 6]
5. $(A \rightarrow C)$ [equivalência]

Exemplo 8: Tem-se que $((A \rightarrow C) \wedge (B \rightarrow C)) \Rightarrow ((A \vee B) \rightarrow C)$
Prova:

1. $(B \rightarrow C)$ [hipótese]
2. $(B \rightarrow C) \rightarrow ((A \vee B) \rightarrow (C \vee A))$ [A_3 com X=B, Y=C e Z=A]
3. $(A \vee B) \rightarrow (C \vee A)$ [MP(1, 2)]
4. $(A \rightarrow C)$ [hipótese]
5. $(A \rightarrow C) \rightarrow ((C \vee A) \rightarrow (C \vee C))$ [A_3 com X=A, Y=C e Z=C]
6. $(C \vee A) \rightarrow (C \vee C)$ [MP(4, 5)]
7. $(A \vee B) \rightarrow (C \vee C)$ [Ex. 7 sobre 3. e 6.]
8. $(C \vee C) \rightarrow C$ [A_1 com X = C]
9. $(A \vee B) \rightarrow C$ [Ex. 7 sobre 7. e 8.]

Exemplo 9: Tem-se que $((A \rightarrow B) \wedge (\neg A \rightarrow B)) \Rightarrow B$
Prova:

1. $(A \rightarrow B)$ [hipótese]
2. $(\neg A \rightarrow B)$ [hipótese]
3. $(A \vee \neg A) \rightarrow B$ [Ex. 8 sobre 1. e 2.]
4. $(A \vee \neg A)$ [Do Ex. 2]
5. B [MP(4, 3)]

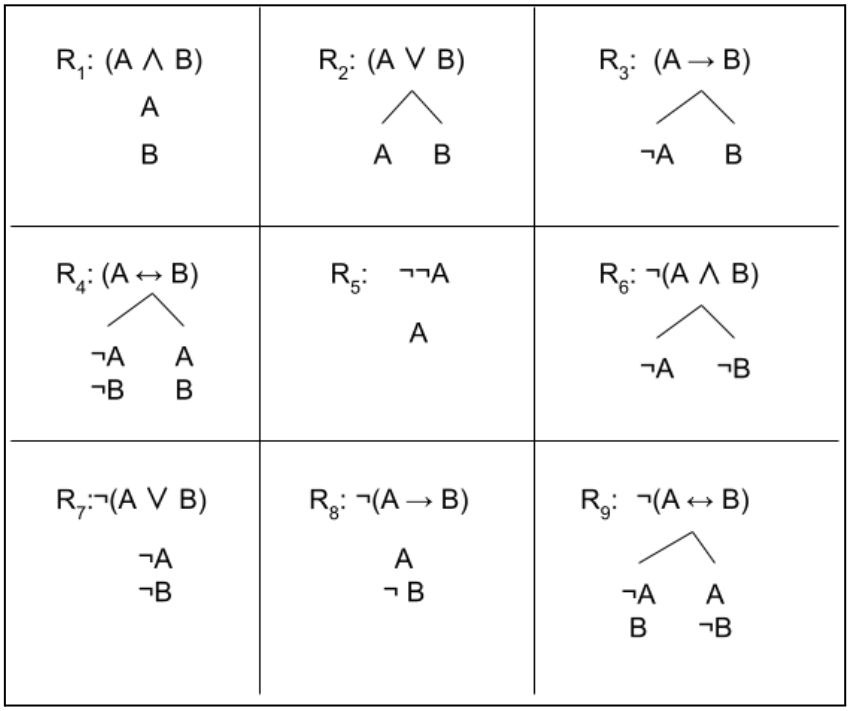
Bom, vamos parar por aqui, o fato é que qualquer teorema da lógica proposicional pode ser provado utilizando este sistema.

Como dito anteriormente, o grande problema deste sistema é a dificuldade de uma aplicação eficiente do método. Por isso, na prática, quase nunca vamos utilizá-lo. Nas próximas seções veremos sistemas mais eficientes.

Sistema de *Tableaux* Semânticos

Os dois métodos que veremos a partir de agora são na verdade formas de determinar a insatisfatibilidade de fórmulas, mas que também podemos utilizar para verificar tautologias, como veremos adiante.

Os *tableaux* semânticos são construídos a partir de nove regras de inferência, ou de reescrita, descritas a seguir



As regras R_1 e R_2 são as regras fundamentais, que definem a estrutura do *tableau* (sem *x* porque está no singular). A regra R_1 indica que fórmulas em sequência são interpretadas como uma conjunção e a regra R_2 indica que fórmulas em ramos distintos (em paralelo) são interpretadas como uma disjunção.

Todas as demais regras são equivalências já conhecidas, utilizando as construções definidas pelas primeiras duas regras. R_3 é a equivalência $(A \rightarrow B) \Leftrightarrow (\neg A \vee B)$, R_4 é a equivalência $(A \leftrightarrow B) \Leftrightarrow ((\neg A \wedge \neg B) \vee (A \wedge B))$, R_5 é a equivalência $\neg\neg A \Leftrightarrow A$, R_6 é a equivalência $\neg(A \wedge B) \Leftrightarrow (\neg A \vee \neg B)$, R_7 é a equivalência $\neg(A \vee B) \Leftrightarrow (\neg A \wedge \neg B)$, R_8 é a equivalência $\neg(A \rightarrow B) \Leftrightarrow (A \wedge \neg B)$ e, finalmente, R_9 é a equivalência $\neg(A \leftrightarrow B) \Leftrightarrow ((\neg A \wedge B) \vee (A \wedge \neg B))$.

As regras cobrem todos os tipos de fórmulas possíveis na lógica proposicional. As regras R_1 a R_4 tratam de fórmulas com os conectivos binários e as regras R_6 a R_9 tratam de fórmulas com os conectivos negados. A fórmula R_5 remove negações duplas.

Construção do *Tableau*

A construção do *tableau* semântico associado à fórmula F é feita da seguinte forma:

1. Inicie com a fórmula F.
2. Enquanto houver fórmula não utilizada:
    a. Escolha uma fórmula não utilizada.

b. Aplique a regra adequada e reescreva a fórmula na parte inferior do *tableau*.

Atenção: *se a fórmula escolhida ocorre antes de uma bifurcação, a equivalência deve ser escrita em todas as bifurcações abaixo dela.*

Uma heurística que normalmente vai poupar trabalho na construção dos *tableaux* é: sempre que possível, utilize primeiro as regras que não provocam bifurcações.

Para deixar mais claro, vamos fazer a construção de alguns tableaux.

Exemplo 1: $(A \vee B) \wedge (A \wedge \neg B)$

1.	$(A \vee B) \wedge (A \wedge \neg B)$		F
2.	$A \vee B$		$[R_1(1)]$
3.	$A \wedge \neg B$		$[R_1(1)]$
	↙	↘	
4.	A	B	$[R_2(2)]$
5.	A	A	$[R_1(3)]$
6.	$\neg B$	$\neg B$	$[R_1(3)]$

Neste caso, para ilustrar a construção, fizemos a bifurcação antes. As linhas 2 e 3 foram resultado da aplicação da regra 1 na primeira linha. As fórmulas da linha 4 foram obtidas a partir da aplicação da regra 2 sobre a fórmula da linha 2. As fórmulas das linhas 5 e 6 oriundam da aplicação da regra 1 sobre a fórmula na linha 3. Neste caso, perceba que o resultado da aplicação da regra 1 sobre a fórmula na linha 3 foi repetida dois dois lados da bifurcação que ocorre na linha 4.

Após a aplicação destas regras, temos apenas literais.

Exemplo 2: $(A \rightarrow B) \wedge \neg(A \vee B) \wedge \neg(C \rightarrow A)$

1.	$(A \rightarrow B) \wedge \neg(A \vee B) \wedge \neg(C \rightarrow A)$	F
2.	$A \rightarrow B$	$[R_1(1)]$
3.	$\neg(A \vee B)$	$[R_1(1)]$
4.	$\neg(C \rightarrow A)$	$[R_1(1)]$
5.	$\neg A$	$[R_7(3)]$

6.		¬B	$[R_7(3)]$
7.		C	$[R_8(4)]$
8.		¬A	$[R_8(4)]$
	↙	↘	
9.	¬A	B	$[R_3(2)]$

Interpretação do Tableau

O tableau associado à uma fórmula F é uma árvore que tem como raiz (início) a fórmula F e, ao aplicar as regras de inferência do tableau, pode se ramificar em diversos caminhos. Lembre que a regra R_1, da conjunção, diz que fórmulas em sequência devem ser satisfeitas simultaneamente.

Portanto, ao observar um ramo da árvore, ou seja, um caminho que parte da raiz e vai até uma extremidade (folha), temos uma sequência de fórmulas que devem ser satisfeitas simultaneamente para que a fórmula seja verdadeira. Ramos distintos representam formas distintas de satisfazer F.

Nos dois exemplos anteriores as árvores criadas têm dois ramos. Vamos rever os tableaux anteriores e interpretar as informações contidas nele.

Exemplo 3: Considere o seguinte *tableau*

(A ∨ B) ∧ (A ∧ ¬B)	
A ∨ B	
A ∧ ¬B	
↙	↘
A	B
A	A
¬B	¬B

Este tableau tem dois ramos, o ramo da esquerda é composto pelas fórmulas
(A ∨ B) ∧ (A ∧ ¬B), (A ∨ B), (A ∧ ¬B), A, A, ¬B.

O ramo da direita é composto pelas fórmulas
(A ∨ B) ∧ (A ∧ ¬B), (A ∨ B), (A ∧ ¬B), B, A, ¬B.

Até a terceira fórmula, antes da bifurcação, os dois ramos são idênticos. O que cada um dos ramos nos diz é que todas estas fórmulas devem ser satisfeitas simultaneamente. O

processo de inferência decompõe as fórmulas até literais. No primeiro ramo temos os literais A, A e ¬B, no segundo ramo temos os literais B, A, ¬B.

Do primeiro ramo, temos que se A é verdadeiro e ¬B é verdadeiro, ou seja, B é falso, a fórmula é satisfatível.

No segundo ramo temos que A deve ser verdadeiro e B deve ser verdadeiro e falso simultaneamente, o que é uma contradição. Portanto, através deste ramo não é possível satisfazer F.

Daí concluímos que F é satisfatível e que para satisfazê-la devemos ter A verdadeiro e B falso.

Exemplo 4: Considere o *tableau*

$(A \rightarrow B) \wedge \neg(A \vee B) \wedge \neg(C \rightarrow A)$

$A \rightarrow B$

$\neg(A \vee B)$

$\neg(C \rightarrow A)$

$\neg A$

$\neg B$

C

$\neg A$

↙ ↘

$\neg A$ B

Novamente temos dois ramos. Adiantando o processo, o ramo da esquerda tem os literais ¬A, ¬B, C, ¬A e ¬A, enquanto o ramo da direita tem os literais ¬A, ¬B, C, ¬A e B.

O ramo da esquerda é uma forma de satisfazer a fórmula F, ele diz que A deve ser falso, B deve ser falso e C deve ser verdadeiro. O ramo da direita não é satisfatível, pois temos os literais complementares, B e ¬B.

Também não é difícil de verificar que esta conclusão está correta, considere a seguinte manipulação da fórmula dada:

$(A \rightarrow B) \wedge \neg(A \vee B) \wedge \neg(C \rightarrow A) \Leftrightarrow (\neg A \vee B) \wedge \neg A \wedge \neg B \wedge C$

Na conjunção temos os literais ¬A, ¬B e C. Com ¬A, satisfazemos a primeira disjunção.

Concluímos. portanto, que ao observar os literais em um ramo do tableau temos duas possibilidades:

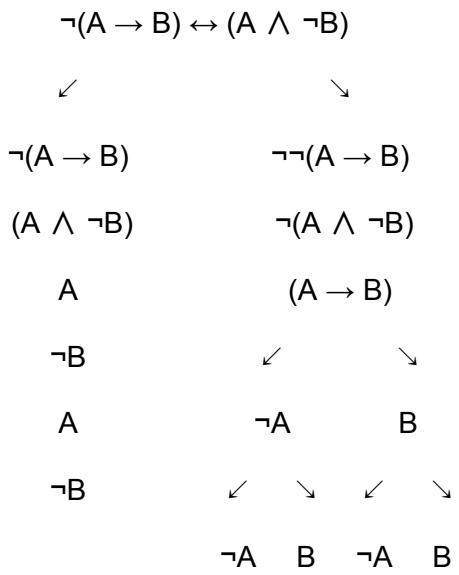
1. Se não há literais complementares, a fórmula é satisfatível; e obtemos também as interpretações necessárias para satisfazê-la.
2. Se há literais complementares, este ramo não é satisfatível. Este tipo de ramo é denominado **ramo fechado do *tableau***.

Se todos os ramos do *tableau* associado a F são fechados, concluímos que F é insatisfatível, ou seja, é sempre falsa. Atenção, o fato de todos os ramos serem abertos não quer dizer que a fórmula é sempre verdade, basta considerar, como contra-exemplo, o tableau associado a (A ∨ B), que tem todos os ramos abertos mas não é uma tautologia.

Portanto, o tableau é um método sintático que nos permite determinar se uma fórmula é satisfatível ou não.

Uma consequência da construção do tableau é que podemos facilmente obter uma representação na forma normal disjuntiva da fórmula dada. Para tal, os literais presentes em cada ramo aberto formarão uma conjunção e a fórmula final será uma disjunção destas conjunções.

Exemplo 5: Considere a fórmula F = ¬(A → B) ↔ (A ∧ ¬B)
Tableau associado a F:



O tableau tem 5 ramos. O ramo mais à esquerda tem os literais A e ¬B, que diz que a fórmula é verdadeira se A é verdadeiro e B é falso. Tomando a primeira bifurcação à direita e as duas seguintes à esquerda, temos o literal ¬A, que diz que a fórmula é verdade se A é falso. Nos dois ramos seguintes, tomando a primeira bifurcação à direita e depois esquerda-direita ou direita-esquerda, temos os literais ¬A e B. Finalmente, tomando todas as bifurcações à direita temos um ramo que tem apenas o literal B. Nenhum dos ramos é fechado.

Como nenhum ramo é fechado, a seguinte fórmula na FND é equivalente a F:
(A ∧ ¬B) ∨ ¬A ∨ (¬A ∧ B) ∨ (¬A ∧ B) ∨ B, onde cada conjunção corresponde a um ramo. Esta fórmula pode ser simplificada facilmente para (A ∧ ¬B) ∨ ¬A ∨ (¬A ∧ B) ∨ B,

já que o terceiro e quarto ramos geram a mesma conjunção. E mais ainda para (A ∧ ¬B) ∨ ¬A ∨ B, já que ¬A cobre a interpretação (¬A ∧ B).

A fórmula do exemplo anterior é uma tautologia, de fato, ela corresponde à regra R_8 do tableau. Mas já vimos que o fato de todos os ramos serem abertos não implica na fórmula ser uma tautologia. Como podemos usar o *tableau* para provar que F é uma tautologia?

Prova por Tableau

Já vimos que a análise do *tableau* associado a uma fórmula nos permite determinar se a fórmula é satisfatível ou não. Uma fórmula insatisfatível é uma fórmula que é sempre falsa. Logo, a negação dessa fórmula é sempre verdadeira, ou seja, uma tautologia.

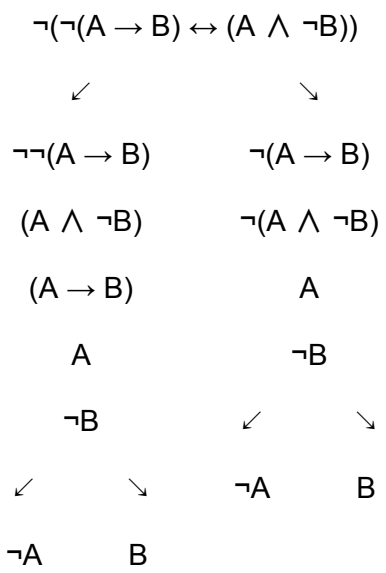
Utilizando este raciocínio, podemos usar o seguinte procedimento para provar a validade de uma fórmula utilizando *tableau*:

Dada uma fórmula F, para determinar se F é tautologia faça:

1. Construa o tableau associado a ¬F.
2. Se o tableau associado a ¬F é fechado:
    a. concluímos que ¬F é insatisfatível;
    b. logo, F é tautologia.

Dizemos que F é um teorema do sistema de tableaux. Como no caso do sistema axiomático, se Γ é um conjunto de hipóteses Γ = $\{H_1, H_2, \ldots, H_n\}$, dizemos que o conjunto implica em F ou que F é consequência lógica de Γ se existe uma prova de $(H_1 \wedge H_2 \wedge \ldots \wedge H_n) \rightarrow F$.

Exemplo 6: Considere a fórmula F = ¬(A → B) ↔ (A ∧ ¬B)
Tableau de ¬F:



Todos os ramos desse tableau são fechados. No primeiro e terceiro ramos temos a ocorrência dos literais complementares A e ¬A. No segundo e quarto ramos temos os literais complementares B e ¬B.

Portanto, a fórmula é uma tautologia, ou seja, $\neg(A \rightarrow B) \Leftrightarrow (A \wedge \neg B)$.

Exemplo 7: Considere o conjunto de hipóteses □ = {H_1, H_2, H_3, H_4, H_5} tal que:

$H_1 = A$

$H_2 = B$

$H_3 = B \rightarrow C$

$H_4 = C \rightarrow D$

$H_5 = (A \wedge D) \rightarrow \neg E$

Podemos concluir que □ $\vdash \neg E$?

Faremos a verificação utilizando tableau. Precisamos verificar se há uma prova de $F = (A \wedge B \wedge (B \rightarrow C) \wedge (C \rightarrow D) \wedge ((A \wedge D) \rightarrow \neg E)) \rightarrow \neg E$.

Portanto, vamos verificar se o tableau associado a ¬F é fechado:

$\neg((A \wedge B \wedge (B \rightarrow C) \wedge (C \rightarrow D) \wedge ((A \wedge D) \rightarrow \neg E)) \rightarrow \neg E)$

$A \wedge B \wedge (B \rightarrow C) \wedge (C \rightarrow D) \wedge ((A \wedge D) \rightarrow \neg E)$

$\neg\neg E$

E

A

B

$B \rightarrow C$

$C \rightarrow D$

$(A \wedge D) \rightarrow \neg E$

↙ ↘

$\neg B$ | C

fechado

↙ ↘

$\neg C$ | D

fechado

↙ ↘

$\neg(A \wedge D)$ | $\neg E$

fechado

↙ ↘

$\neg A$ | $\neg D$

fechado *fechado*

Como todos os ramos são fechados, concluímos que ¬E é uma consequência lógica de □. Neste caso, perceba que encerramos a expansão do ramo assim que verificamos que ele é fechado, pois tudo que descende deste ponto será fechado também. Após a primeira bifurcação, por exemplo, verificamos que ¬B, do lado esquerdo, é complementar a B, na sexta linha, portanto este ramo é fechado. Qualquer ramo que ocorresse abaixo deste ponto também seria fechado pois conteria os literais complementares B e ¬B

Isso encerra o conteúdo acerca deste sistema de dedução. Na próxima seção vamos apresentar o último dos sistemas de dedução que estudaremos na lógica proposicional.

Sistema de Resolução

O último sistema que estudaremos é chamado de método da resolução e é o mais recente dos métodos, desenvolvido na década de 60. Este é, provavelmente, o método mais comumente implementado em computadores e é a base de linguagens de programação lógica, como o PROLOG.

Este método foi desenvolvido para fórmulas na forma normal conjuntiva, especificamente, e, por isso, vamos iniciar o estudo através de uma representação particular para fórmulas na FNC.

Representação Clausal (Notação de Conjuntos)

Já vimos que as fórmulas representadas na FNC são uma conjunção de disjunções. Cada elemento da conjunção de uma fórmula na FNC é chamado de cláusula.

Exemplo: Considere a fórmula $F = (P \rightarrow Q) \wedge ((P \wedge Q) \rightarrow R) \wedge \neg R$.

Primeiro, vamos escrevê-la na FNC:

$F = (P \rightarrow Q) \wedge ((P \wedge Q) \rightarrow R) \wedge \neg R \Leftrightarrow (\neg P \vee Q) \wedge (\neg(P \wedge Q) \vee R) \wedge \neg R$

$\Leftrightarrow (\neg P \vee Q) \wedge (\neg P \vee \neg Q \vee R) \wedge \neg R$

Dizemos que esta fórmula tem três cláusulas:

$C_1 = (\neg P \vee Q)$

$C_2 = (\neg P \vee \neg Q \vee R)$

$C_3 = \neg R$

A notação de conjuntos é uma forma de representar as fórmulas na FNC omitindo os conectivos de conjunção e disjunção. Nesta notação, cada cláusula é representada por um conjunto de literais e a fórmula como um todo é um conjunto de cláusulas.

Exemplo: Considere as cláusulas do exemplo anterior

$C_1 = (\neg P \vee Q)$

$C_2 = (\neg P \vee \neg Q \vee R)$
$C_3 = \neg R$
Na notação de conjuntos temos
$C_1 = \{\neg P, Q\}$
$C_2 = \{\neg P, \neg Q, R\}$
$C_3 = \{\neg R\}$
A fórmula como um todo é representada na notação de conjuntos da seguinte forma:
$F = \{\{\neg P, Q\}, \{\neg P, \neg Q, R\}, \{\neg R\}\}$

Esta notação, além de omitir os conectivos, apresenta algumas propriedades da lógica. Os conjuntos {A, B} e {B, A} são idênticos e, de fato, sabemos que a ordem dos fatores não afeta o resultado da conjunção ou da disjunção. Além disso, conjuntos não permitem a presença de elementos repetidos, portanto a fórmula $(A \vee B \vee A)$ é representada pelo conjunto {A, B}, sem repetição do literal A. Mais uma vez, isso é válido na lógica, uma vez que $(A \vee A) \Leftrightarrow (A \wedge A) \Leftrightarrow A$.

Tendo esta nova notação, vamos definir a única regra de inferência da resolução nestes termos.

Expansão por Resolução

O sistema de resolução tem uma única regra de inferência, que é definida da seguinte forma:

Regra de Resolução:
Dadas duas cláusulas, C_1 e C_2, tais que o literal X pertence a C_1 e seu complementar, $\neg X$, ocorre em C_2, a resolução de C_1 e C_2 é dada por:

$$R(C_1, C_2) = (C_1 \cup C_2) \setminus \{X, \neg X\}$$

onde a operação $A \setminus B$ é a diferença de conjuntos (remove de A os elementos de B).

Atenção! *Alguns textos definem a resolução para quaisquer pares de cláusulas e não apenas aqueles com literais complementares. Nestes casos a resolução é apenas a união das cláusulas. Para efeitos práticos, isto é irrelevante, veremos que a regra de resolução só é usada quando há exatamente um par de literais complementares entre as cláusulas.*

Em outras palavras, a resolução de duas cláusulas é o conjunto de todos os seus literais com a remoção de um par de literais complementares.

Em termos da lógica, o que esta regra de inferência diz é que se duas cláusulas C_1 e C_2 são verdadeiras então a cláusula $R(C_1, C_2)$ também é. Ou seja, para satisfazer C_1 e C_2 é necessário (mas não suficiente) que $R(C_1, C_2)$ seja verdadeira. A seguinte fórmula resume este parágrafo: $(C_1 \wedge C_2) \Rightarrow R(C_1, C_2)$.

É importante observar que apenas um par de literais complementares é removido, pela definição dada acima, este é um erro comum na aplicação desta regra. A seguir apresentamos alguns exemplos de aplicação da resolução:

Exemplo: Considere as cláusulas
$C_1 = \{P\}$, $C_2 = \{\neg P\}$, $C_3 = \{\neg P, Q\}$, $C_4 = \{P, R\}$ e $C_5 = \{P, \neg Q\}$.

Entre as cláusulas C_1 e C_3 temos o literal P ocorrendo em C_1 e ¬P ocorrendo em C_3. Portanto é possível fazer a resolução dessas duas cláusulas:

$R(C_1, C_3) = (\{P\} \cup \{\neg P, Q\}) \setminus \{P, \neg P\} = \{P, \neg P, Q\} \setminus \{P, \neg P\} = \{Q\}$

Entre as cláusulas C_3 e C_4 também ocorre o literal P e seu complementar. Daí temos:

$R(C_4, C_3) = (\{P, R\} \cup \{\neg P, Q\}) \setminus \{P, \neg P\} = \{P, R, \neg P, Q\} \setminus \{P, \neg P\} = \{Q, R\}$

Entre as cláusulas C_1 e C_2 também ocorre o literal P e seu complementar. A resolução é, portanto,

$R(C_1, C_2) = (\{P\} \cup \{\neg P\}) \setminus \{P, \neg P\} = \{P, \neg P\} \setminus \{P, \neg P\} = \{\ \}$

A cláusula vazia representa o símbolo de verdade 0, pois as fórmulas P e ¬P não podem ser satisfeitas simultaneamente.

Finalmente, entre as cláusulas C_3 e C_5 temos dois pares de literais complementares, P e Q. Nestes casos é muito comum a ocorrência do seguinte erro na aplicação do método da resolução:

$R(C_3, C_5) = \{\ \}$, isto está errado!

Perceba que a definição da resolução remove apenas um par de literais complementares, portanto existem duas resoluções possíveis para estas cláusulas, que apresentamos passo a passo abaixo

Removendo P:

$R(C_3, C_5) = (\{\neg P, Q\} \cup \{P, \neg Q\}) \setminus \{P, \neg P\} = \{P, \neg P, Q, \neg Q\} \setminus \{P, \neg P\} = \{Q, \neg Q\}$

Removendo Q:

$R(C_3, C_5) = (\{\neg P, Q\} \cup \{P, \neg Q\}) \setminus \{Q, \neg Q\} = \{P, \neg P, Q, \neg Q\} \setminus \{Q, \neg Q\} = \{P, \neg P\}$

Nos dois casos temos uma tautologia, uma vez que {P, ¬P} é uma disjunção de literais complementares $(P \vee \neg P) \Leftrightarrow 1$. O mesmo vale para {Q, ¬Q}. Então as duas resoluções são equivalentes. *Observe que sempre que há mais de um par de literais complementares teremos como resultado da resolução uma tautologia, pois vão restar um ou mais pares de literais na disjunção resultante.*

Agora que temos a regra de inferência, podemos definir como é realizada a expansão por resolução de uma fórmula.

Dada uma fórmula F, na notação clausal, com cláusulas $C_1, C_2, \ldots, C_n$, a expansão por resolução de F é um conjunto de cláusulas tais que:

- As primeiras n cláusulas da expansão são as cláusulas de F.
- A cláusula C_k, com $k > n$, é obtida pela aplicação da regra da resolução sobre C_x e C_y desde que:
  - $x,y < k$, ou seja, as cláusulas usadas na resolução já ocorreram na expansão.
  - $R(C_x, C_y) \neq C_i \ \forall \ i < k$, ou seja, a cláusula resultante não ocorreu na expansão.
  - Há exatamente um par de literais complementares entre C_x e C_y.

A expansão se encerra quando não for mais possível criar novas cláusulas, nos termos descritos anteriormente, ou quando a cláusula vazia é produzida.

Exemplo: Considere a fórmula $F = (A \vee B) \wedge (A \wedge \neg B)$.

F já está na CNF, então vamos apenas escrevê-la na notação clausal:

$F = \{\{A, B\}, \{A\}, \{\neg B\}\}$

A expansão por resolução de F é:

1. {A, B}
2. {A}
3. {¬B}

Perceba que não há nenhum outro passo a ser realizado pois as únicas cláusulas com literais complementares são C_1 e C_3, mas o resultado é {A}, que já ocorreu anteriormente, em C_2.

Exemplo: Considere a fórmula G = (A → B) ∧ (B → C) ∧ (C → A).

Convertendo para a CNF:

G = (A → B) ∧ (B → C) ∧ (C → A) ⇔ (¬A ∨ B) ∧ (¬B ∨ C) ∧ (¬C ∨ A)

Na notação clausal:

G = {{¬A, B}, {¬B, C}, {¬C, A}}

A expansão por resolução de G é:

1. {¬A, B}
2. {¬B, C}
3. {¬C, A}
4. {¬A, C} R(1, 2)
5. {¬C, B} R(1, 3)
6. {¬B, A} R(2, 3)

Nenhuma outra inferência é possível pois as cláusulas que ainda não foram resolvidas, como R(3, 4), R(2, 5) e R(1, 6), possuem dois pares de literais complementares e não devem ser resolvidas.

Exemplo: Considere a fórmula H = {{A, B}, {A, ¬B}, {¬A, B}, {¬A, ¬B}}

A expansão por resolução de H é:

1. {A, B}
2. {A, ¬B}
3. {¬A, B}
4. {¬A, ¬B}
5. {A} R(1,2)
6. {B} R(1,3)
7. {¬B} R(2,4)
8. {¬A} R(3,4)
9. { } R(5,8)

Como a cláusula vazia foi encontrada, a expansão encerra.

Vimos até agora como fazer a expansão por resolução e como interpretar a regra de inferência deste método, vamos avaliar agora como a expansão pode ser entendida.

Interpretação da Expansão

Como já dito, a regra de inferência da resolução indica condições necessárias para que duas cláusulas sejam satisfeitas simultaneamente. Portanto, ao encontrar uma cláusula

vazia podemos concluir que a fórmula é insatisfatível, já que há um par de cláusulas que não pode ser satisfeito. Como a fórmula é uma conjunção de cláusulas, a existência de um par insatisfatível implica que toda a fórmula é insatisfatível.

A inexistência da cláusula vazia indica que a fórmula é satisfatível. No entanto, ao contrário do tableau, não necessariamente chegaremos até literais na expansão por resolução. Isso quer dizer que descobriremos apenas algumas condições necessárias para a satisfatibilidade, mas não todas. Eventualmente, se literais são alcançados, isso indica que tais literais devem ser verdade para a fórmula ser verdadeira, mas não necessariamente é a única forma dela ser verdade.

Além disso, como a resolução é uma consequência das cláusulas anteriores, a conjunção de todas as cláusulas da expansão é equivalente à fórmula original.

Exemplo: Considere a fórmula I = {{A, B}, {¬A, B}, {A, C}}.

A expansão por resolução de I é:

1. {A, B}
2. {¬A, B}
3. {A, C}
4. {B} R(1,2)
5. {B, C} R(2,3)

A partir desta expansão podemos concluir que I é satisfatível e que B deve ser verdadeiro. Lembre que esta condição é necessária mas não é suficiente. Para I ser satisfeita, B deve ser verdade, mas não basta B ser verdade para que I seja verdadeira. Vamos avaliar a tabela verdade desta fórmula:

A	B	C	(A ∨ B) ∧ (¬A ∨ B) ∧ (A ∨ C)
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Nas linhas em vermelho, onde a fórmula é verdadeira, temos que B é verdadeiro. No entanto, há um caso, marcado em azul, em que B é verdade mas I não é.

A conjunção de todas as cláusulas da expansão é equivalente à fórmula original, ou seja:

$$I \Leftrightarrow (A \vee B) \wedge (\neg A \vee B) \wedge (A \vee C) \wedge B \wedge (B \vee C)$$

⇔ B ∧ (A ∨ C)

Todas as outras cláusulas sumiram porque contém B, e daí automaticamente são verdade se B é verdadeiro. Note que, na fórmula resultante, B precisa ser verdadeiro mas não é a única condição para que I seja verdade. Além disso, é necessário que A ou C sejam verdade.

Exemplo: Considere a fórmula H = {{A, B}, {A, ¬B}, {¬A, B}, {¬A, ¬B}}

A expansão por resolução de H é:

1. {A, B}
2. {A, ¬B}
3. {¬A, B}
4. {¬A, ¬B}
5. {A} R(1,2)
6. {B} R(1,3)
7. {¬B} R(2,4)
8. {¬A} R(3,4)
9. { } R(5,8)

Como a cláusula vazia foi encontrada, a expansão encerra e concluímos que esta fórmula não pode ser satisfeita. A cláusula 5 indica que A deve ser verdadeiro para que as cláusulas 1 e 2 sejam satisfeitas. Já a cláusula 8 indica que A deve ser falso para que as cláusulas 3 e 4 sejam satisfeitas. Logo, não é possível satisfazer todas estas cláusulas ao mesmo tempo.

Como curiosidade, este é um caso que é fácil de perceber que é insatisfatível apenas olhando para a fórmula. Na FNC, cada disjunção determina uma ou mais interpretações que são falsas. Como temos apenas dois símbolos proposicionais, existem apenas quatro interpretações. Como há quatro cláusulas distintas, cada cláusula elimina uma interpretação. Vamos observar a tabela verdade de H para verificar isto.

A	B	(A ∨ B)	(A ∨ ¬B)	(¬A ∨ B)	(¬A ∨ ¬B)	H
0	0	0	1	1	1	0
0	1	1	0	1	1	0
1	0	1	1	0	1	0
1	1	1	1	1	0	0

Cada cláusula é falsa em uma das linhas da tabela verdade.

Prova por Resolução

Assim como os *tableaux* semânticos, o método da resolução nos permite verificar se uma fórmula é insatisfatível ou não. Utilizando o mesmo raciocínio da seção anterior,

podemos utilizar o seguinte procedimento para provar a validade de uma fórmula utilizando resolução:

Dada uma fórmula F, para determinar se F é tautologia faça:

1. Escreva ¬F na forma clausal.
2. Faça a expansão por resolução de ¬F.
3. Se a expansão por resolução de ¬F contém a cláusula vazia:
    a. concluímos que ¬F é insatisfatível;
    b. logo, F é tautologia.

Dizemos que F é um teorema do sistema de resolução. Como no caso do sistema axiomático, se □ é um conjunto de hipóteses, □ = {H_1, H_2, …, H_n}, dizemos que o conjunto implica em F ou que F é consequência lógica de □ se existe uma prova de (H_1 ∧ H_2 ∧ … ∧ H_n) → F.

Vamos repetir os mesmos exemplos da seção de *tableaux*, agora utilizando o método da resolução.

Exemplo: Considere a fórmula F = ¬(A → B) ↔ (A ∧ ¬B)

Vamos converter ¬F para a FNC:

¬F ⇔ ¬(¬(A → B) ↔ (A ∧ ¬B))
⇔ ¬((¬(A → B) → (A ∧ ¬B)) ∧ ((A ∧ ¬B) → ¬(A → B)))
⇔ ¬((¬(¬A ∨ B) → (A ∧ ¬B)) ∧ ((A ∧ ¬B) → ¬(¬A ∨ B)))
⇔ ¬((¬¬(¬A ∨ B) ∨ (A ∧ ¬B)) ∧ (¬(A ∧ ¬B) ∨ ¬(¬A ∨ B)))
⇔ ¬(((¬A ∨ B) ∨ (A ∧ ¬B)) ∧ ((¬A ∨ B) ∨ (A ∧ ¬B)))
⇔ ¬((¬A ∨ B) ∨ (A ∧ ¬B))
⇔ ¬(¬A ∨ B) ∧ ¬(A ∧ ¬B)
⇔ A ∧ ¬B ∧ (¬A ∨ B)

Finalmente, a expansão por resolução de ¬F é:

1. {A}
2. {¬B}
3. {¬A, B}
4. {B} R(1,3)
5. { } R(2,4)

Daí concluímos que F é uma tautologia.

Exemplo: Considere o conjunto de hipóteses □ = {H_1, H_2, H_3, H_4, H_5} tal que:

H_1 = A
H_2 = B
H_3 = B → C
H_4 = C → D
H_5 = (A ∧ D) → ¬E

Podemos concluir que □ ⊢ ¬E?

Devemos demonstrar que há uma prova de
F = (A ∧ B ∧ (B → C) ∧ (C → D) ∧ ((A ∧ D) → ¬E)) → ¬E.

Primeiro vamos converter ¬ para a FNC:

¬F ⇔ ¬((A ∧ B ∧ (B → C) ∧ (C → D) ∧ ((A ∧ D) → ¬E)) → ¬E)
⇔ A ∧ B ∧ (B → C) ∧ (C → D) ∧ ((A ∧ D) → ¬E) ∧ ¬¬E

⇔ A ∧ B ∧ (¬B ∨ C) ∧ (¬C ∨ D) ∧ (¬(A ∧ D) ∨ ¬E) ∧ E
⇔ A ∧ B ∧ (¬B ∨ C) ∧ (¬C ∨ D) ∧ (¬A ∨ ¬D ∨ ¬E) ∧ E

A expansão por resolução de ¬F é:

1. {A}
2. {B}
3. {¬B, C}
4. {¬C, D}
5. {¬A, ¬D, ¬E]
6. {E}
7. {¬D, ¬E} R(1, 5)
8. {¬D} R(6, 7)
9. {C} R(2, 3)
10. {D} R(4, 9)
11. { } R(8, 10)

Como a expansão é fechada, concluímos que □ ⇒ ¬E.

Lógica de Predicados

Essa versão da lógica, também conhecida como lógica de primeira ordem, que estudaremos a partir de agora, pode ser entendida como uma extensão da lógica proposicional, que vimos até então.

A primeira pergunta que surge é: qual a necessidade de estender a lógica proposicional? Note que a lógica proposicional, apesar de nos permitir a representação de diversos problemas, vem com algumas restrições em sua capacidade de representação.

Primeiro, considere um tabuleiro cujas posições são nomeadas por pares ordenados. A posição (x, y) é aquela que está na x-ésima linha, a partir do topo, e na y-ésima coluna, a partir da esquerda. Na figura abaixo temos um exemplo de um tabuleiro de tamanho 3 x 3.

(1,1)	(1,2)	(1,3)
(2,1)	(2,2)	(2,3)
(3,1)	(3,2)	(3,3)

Considere a seguinte afirmação acerca do movimento de uma peça neste tabuleiro: “Uma peça na casa (x, y) alcança a casa (x, y + 1) em um passo para a direita, desde que y seja menor que 3”. Para representar esta única afirmação na lógica proposicional precisaríamos de uma relação para cada par de casas horizontalmente adjacentes, por exemplo “Uma peça na casa (1,1) alcança a casa (1,2) em um passo para a direita” seria uma das afirmações necessárias. Todas estas relações estão resumidas na primeira frase através do uso de *variáveis* (x e y) e de *funções* (y + 1). Na lógica proposicional as afirmações são determinadas e estáticas, e podem ser interpretadas como verdadeiras ou falsas. A afirmação genérica proposta tem uma interpretação que depende do valor atribuído às variáveis x e y.

Considere agora a afirmação: “Para quaisquer números naturais x e y, se x é maior que y, existe z, tal que $x > z \geq y$”. Mais uma vez temos uma regra genérica que não pode ser representada pela lógica proposicional. Além disso, neste caso o domínio é infinito, e não apenas um tabuleiro de tamanho limitado, de modo que não podemos obter uma representação equivalente mesmo utilizando um conjunto extenso de fórmulas.

Resumindo, são dois os principais motivos pelos quais a extensão da lógica proposicional é interessante: (1) aumentar a capacidade de representação da lógica e (2) obter representações mais concisas em alguns casos.

De modo similar ao que fizemos com a lógica proposicional, vamos rever a sintaxe, a semântica e os sistemas de dedução da lógica, estendendo-os.

Sintaxe

A sintaxe da lógica de predicados é muito similar à sintaxe da lógica proposicional. Veremos aqui quais as principais diferenças entre os dois.

Alfabeto

Os elementos centrais da linguagem lógica são as afirmações, que na lógica proposicional eram estáticas e chamadas de “proposições”, dando origem ao nome da linguagem. Como discutido brevemente nos parágrafos anteriores, a idéia central da lógica de predicados é a extensão da definição dessas afirmações, passando a ser possível fazer afirmações cuja interpretação depende de certos valores atribuídos a partes da afirmação, através das variáveis. Portanto, os predicados são funções que têm como contradomínio o conjunto de símbolos de verdade e são os elementos essenciais desta linguagem.

Já que os predicados são funções, é necessário a ocorrência de variáveis e desejável a possibilidade de utilizar funções nos argumentos destes predicados. Além disso, já vimos que na definição das relações genéricas dadas como exemplo anteriormente, utilizamos expressões como “para quaisquer *x* e *y*” e “existe *z*”. Estas expressões são quantificadoras das variáveis e também farão parte da nossa linguagem.

Deste modo, o alfabeto da lógica de predicados é constituído por:

- Símbolos de verdade: 0 e 1.
- Predicados: p, q, r, teste, …
    - Os predicados sem argumentos são as *proposições*.
- Variáveis: X, Y, Z, Coisa, …
- Funções: f, g, h, menorQue, …
    - As funções sem argumentos são as *constantes*.
- Conectivos: ¬, $\wedge$, $\vee$, $\rightarrow$ e $\leftrightarrow$.
- Símbolos de pontuação: (e).
- Quantificadores: $\forall$ e $\exists$.

Os símbolos de verdade, conectivos e pontuação são os mesmos da lógica proposicional. Neste texto, os predicados e funções serão representados por cadeias de caracteres iniciadas por letras minúsculas enquanto as variáveis serão representadas por cadeias de caracteres iniciadas por letras maiúsculas.

O quantificador $\forall$ é lido como “para todo” e é chamado de quantificador universal, enquanto o quantificador $\exists$ é lido como “existe” e é chamado de quantificador existencial.

Fórmulas

A definição de fórmula é muito similar àquela utilizada na lógica proposicional:

- Todo átomo é uma fórmula.
- Se F é uma fórmula, (¬F) também é.
- Se F e G são fórmulas, também são:
    - (F $\wedge$ G)
    - (F $\vee$ G)
    - (F $\rightarrow$ G)
    - (F $\leftrightarrow$ G)
- Se F é uma fórmula e X é uma variável, também são fórmulas:
    - ($\forall$X) F)
    - ($\exists$X) F)

Em relação ao que tínhamos antes, apenas a última regra é novidade: “a aplicação de um quantificador sobre uma fórmula também é uma fórmula”. No entanto, há uma pequena diferença quanto à definição de “átomo”. Anteriormente os átomos eram os

símbolos de verdade e símbolos proposicionais, agora, como os predicados tomaram o lugar das proposições, definimos os átomos como segue.

São átomos:

- Símbolos de verdade
- Predicados

Aproveitando o assunto, podemos afirmar que as definições de literal e das formas normais permanecem inalteradas. Relembrando, literais são átomos ou suas negações e a forma normal conjuntiva (FNC) é a conjunção de disjunções de literais.

Pela definição de fórmula dada, percebemos que as variáveis e funções só podem ocorrer como argumentos de predicados, funções e quantificadores e não como fórmulas por si só. Portanto, deve ficar claro pelo contexto se um certo símbolo é um predicado ou uma função.

Também vamos aproveitar e atualizar nossa lista de precedência, incluindo os quantificadores $\forall$ e $\exists$ após o operador de negação, na lista de precedência. Deste modo, a fórmula $(\forall X)p(X) \rightarrow r(Y)$ é equivalente à fórmula $(((\forall X)p(X)) \rightarrow r(Y))$.

Comprimento de Fórmula

O comprimento de uma fórmula é dado pelo número de ocorrências de átomos, quantificadores e conectivos. Ou, de forma mais formal, se F é uma fórmula, o comprimento de F, denotado por |F|, é definido como:

- Se F é um átomo, então $|F| = 1$.
- Se $F = \neg G$, então $|F| = 1 + |G|$.
- Se $F = (G \wedge H)$, então $|F| = 1 + |G| + |H|$.
- Se $F = (G \vee H)$, então $|F| = 1 + |G| + |H|$.
- Se $F = (G \rightarrow H)$, então $|F| = 1 + |G| + |H|$.
- Se $F = (G \leftrightarrow H)$, então $|F| = 1 + |G| + |H|$.
- Se $F = (\forall X)\ G$, então $|F| = 1 + |G|$.
- Se $F = (\exists X)\ G$, então $|F| = 1 + |G|$.

É importante salientar que o comprimento da fórmula independe do número de argumentos que os predicados apresentam. Um predicado com qualquer quantidade de argumentos soma uma unidade ao comprimento da fórmula.

Exemplo: $F = ((\forall X)\ (\exists Y)\ p(X, Y) \rightarrow (\exists Z)\ \neg q(Z)) \wedge r(Y)$

Temos a ocorrência de 3 quantificadores, 3 predicados e 3 conectivos, totalizando 9. Vamos verificar utilizando a definição formal.

$|F| = 1 + |(\forall X)\ (\exists Y)\ p(X, Y) \rightarrow (\exists Z)\ \neg q(Z)| + |r(Y)|$
$|F| = 1 + (1 + |(\forall X)\ (\exists Y)\ p(X, Y)| + |(\exists Z)\ \neg q(Z)|) + 1$
$|F| = 1 + (1 + (1 + |(\exists Y)\ p(X, Y)|) + (1 + |\neg q(Z)|)) + 1$
$|F| = 1 + (1 + (1 + (1 + |p(X, Y)|)) + (1 + (1 + |q(Z)|))) + 1$
$|F| = 1 + (1 + (1 + (1 + 1)) + (1 + (1 + 1))) + 1$
$|F| = 1 + (1 + (1 + 2) + (1 + 2)) + 1$
$|F| = 1 + (1 + 3 + 3) + 1$
$|F| = 1 + 7 + 1$
$|F| = 9$

Subfórmula

Se X é uma fórmula:

1. X é uma subfórmula de X.
2. Se X = (¬Y), então Y é subfórmula de X.
3. Se X = (Y□Z), então Y e Z são subfórmulas de X, onde □ é $\wedge$, $\vee$, $\rightarrow$ ou $\leftrightarrow$.
4. Se X = (($\forall$ V) Y), então Y é subfórmula de X.
5. Se X = (($\exists$ V) Y), então Y é subfórmula de X.
6. Se Y é subfórmula de X, então toda subfórmula de Y também é subfórmula de X.

Apenas os casos que tratam dos quantificadores são novos, em relação à definição de subfórmula para a lógica proposicional.

Semântica

Já vimos, para a lógica proposicional, que a interpretação é uma função cujo domínio são as fórmulas lógicas e o contradomínio são os símbolos de verdade. Mais uma vez, vamos estender essa definição para a linguagem dos predicados.

Interpretação

A função de interpretação, já conhecida, define como são interpretados os conectivos lógicos e deixa livre a interpretação dada aos símbolos proposicionais, vindo daí a construção da tabela-verdade que exploramos na lógica proposicional.

Para interpretar uma fórmula do tipo F = p(X, f(X)) é necessário saber qual o significado do predicado *p*, da fórmula *f* e qual o valor da variável *X*. Além disso, na presença de quantificadores, por exemplo, na fórmula G = ($\exists$ X) p(X, f(X)), é necessário saber qual o domínio da interpretação, para saber quais valores de X devem ser considerados.

Considere os seguintes exemplos.

Exemplo: F = p(X, Y)
Sabendo que p é o predicado que é verdadeiro quando X é maior que Y, só conseguimos decidir se a fórmula F é verdadeira ou falsa após saber o valor atribuído às variáveis X e Y. Saber apenas uma delas não é suficiente.

Exemplo: G = ($\forall$ X) p(X, Y)
Sabendo que p é o predicado que é verdadeiro quando X é maior que Y, só conseguimos decidir se a fórmula F é verdadeira ou falsa após saber o valor atribuído à variável Y e também qual o domínio de valores que devemos considerar para X.

Portanto, uma interpretação deve definir o domínio da interpretação, os significados dos predicados e funções e o valor das variáveis não quantificadas. Apenas com todas

estas informações somos capazes de decidir o significado de uma dada fórmula da lógica de predicados.

Se F é uma fórmula, a definição da interpretação dos símbolos de verdade e dos conectivos lógicos não é alterada:

- Se F = 0, então I[F] = 0.
- Se F = 1, então I[F] = 1.
- Se F = ¬X, então
  - I[F] = I[¬X] = 1, se I[X] = 0;
  - I[F] = I[¬X] = 0, se I[X] = 1.
- Se F = X ∧ Y, então
  - I[F] = I[X ∧ Y] = 1, se I[X] = 1 e I[Y] = 1;
  - I[F] = I[X ∧ Y] = 0, se I[X] = 0 e/ou I[Y] = 0.
- Se F = X ∨ Y, então
  - I[F] = I[X ∨ Y] = 1, se I[X] = 1 e/ou I[Y] = 1;
  - I[F] = I[X ∨ Y] = 0, se I[X] = 0 e I[Y] = 0.
- Se F = X → Y, então
  - I[F] = I[X → Y] = 1, se I[X] = 0 e/ou I[Y] = 1;
  - I[F] = I[X → Y] = 0, se I[X] = 1 e I[Y] = 0.
- Se F = X ↔ Y, então
  - I[F] = I[X ↔ Y] = 1, se I[X] = I[Y];
  - I[F] = I[X ↔ Y] = 0, se I[X] ≠ I[Y].

Para variáveis, funções e predicados, as seguintes regras devem ser observadas. Se I é uma interpretação sobre o domínio $\mathscr{D}$, temos que:

- Se X é uma variável, I[X] ∈ $\mathscr{D}$.
- Se f é uma função com *n* argumentos (a_1, … a_n), I[f(a_1, …, a_n)] ∈ $\mathscr{D}$.
- Se p é um predicado com *n* argumentos (a_1, … a_n), I[p(a_1, …, a_n)] ∈ {0, 1}.

Estes são os elementos livres para interpretação. Do ponto de vista da matemática, a única diferença entre as funções e os predicados é o contradomínio. Como as fórmulas lógicas devem ser avaliadas como verdadeiras ou falsas, apenas os predicados podem ser utilizados como átomos.

Uma forma interessante de entender esta diferença é que os predicados são perguntas cuja resposta é sempre “sim” ou “não” enquanto as funções podem usar qualquer “objeto” como resposta.

Seja p(X, Y) o predicado que é verdadeiro se X é o dobro de Y. Ao interpretar a fórmula p(6, 3) obtemos como resposta “verdadeiro” e ao avaliar a fórmula p(5, 3) obtemos como resposta “falso”. No primeiro caso, é como se estivéssemos perguntando “6 é o dobro de 3?” e no segundo caso estamos perguntando “5 é o dobro de 3?”.

Já as funções nos permitem perguntas abertas. Seja f(X) a função que retorna o dobro de X. Ao interpretar f(3) obtemos “6” como resposta. Essa interpretação seria equivalente à pergunta “quem é o dobro de 3?”

Pela definição da interpretação, as funções são mapeamentos do tipo $\mathscr{D}^n \mapsto \mathscr{D}$ enquanto os predicados são mapeamentos do tipo $\mathscr{D}^n \mapsto \{0, 1\}$. Por isso, algumas funções não podem ser definidas para certos domínios.

Considere uma interpretação sobre o domínio dos números naturais. Neste caso não é possível definir a função d(X, Y), onde d retorna a divisão de X por Y, porque neste caso o mapeamento seria do tipo $\mathbb{N}^2 \mapsto \mathbb{Q}$ e não $\mathbb{N}^2 \mapsto \mathbb{N}$, como esperado.

As variáveis e funções são os únicos elementos da fórmula que têm um contradomínio não binário. Estes elementos do alfabeto da lógica de predicados são chamados de *termos*.

E quanto aos quantificadores? Primeiro devemos entender como deve ser lida uma fórmula com quantificadores. A fórmula $(\forall X)$ F deve ser lida como “Para todo valor do domínio que for colocado no local da variável X, a fórmula F é verdadeira” e a fórmula $(\exists X)$ F deve ser lida como “Existe pelo menos um valor do domínio tal que, se for colocado no local da variável X, a fórmula F é verdadeira”. Assim sendo, podemos definir as regras de interpretação dos quantificadores como segue.

- $I[(\forall X)\ F] = 1 \Leftrightarrow \forall\ a \in \mathscr{D}, \langle X \leftarrow a\rangle I[F] = 1$
- $I[(\forall X)\ F] = 0 \Leftrightarrow \exists\ a \in \mathscr{D}; \langle X \leftarrow a\rangle I[F] = 0$
- $I[(\exists X)\ F] = 1 \Leftrightarrow \exists\ a \in \mathscr{D}; \langle X \leftarrow a\rangle I[F] = 1$
- $I[(\exists X)\ F] = 0 \Leftrightarrow \forall\ a \in \mathscr{D}, \langle X \leftarrow a\rangle I[F] = 0$

A notação $\langle x \leftarrow a\rangle I[H]$ diz que a variável X deve ser substituída pelo valor a na interpretação de H. Os casos positivos são apenas uma representação simbólica das frases enunciadas no parágrafo anterior. Vamos avaliar os casos negativos.

$I[(\forall X)\ F] = 0$ indica que “É falso que F é verdade para qualquer valor do domínio”. Ora, isso é o mesmo que dizer que “Há pelo menos um valor do domínio que torna F falsa” que é a frase representada simbolicamente nas regras acima. De modo similar, $I[(\exists X)\ F] = 0$ indica que “É falso que há um valor do domínio que torna F verdade”. Logo, concluímos que “Para qualquer valor do domínio, F é falso”, que é a frase representada simbolicamente nas regras apresentadas.

Escopo de Quantificadores

Um ponto importante para a interpretação das fórmulas é saber se uma certa variável está quantificada ou não. Por exemplo, na fórmula $(\forall X)\ p(X) \rightarrow q(X)$, a variável X em q(X) está quantificada ou não?

Para responder esta pergunta precisamos definir qual o escopo dos quantificadores, ou seja, até onde ele alcança.

O escopo de um quantificador é a sub-fórmula imediatamente à direita dele. Ou seja, o quantificador só se aplica às ocorrências da variável quantificada que aparecem dentro do seu escopo. Se uma mesma variável está quantificada de formas diferentes e está no escopo de mais de um quantificador, apenas o quantificador mais próximo (escopo mais interno) é considerado.

Exemplo: Na fórmula $(\forall X)\ p(X) \rightarrow q(X)$, apenas a primeira ocorrência de X está quantificada, a ocorrência de X em q(X) está fora do escopo de $(\forall X)$.

Exemplo: Na fórmula $(\forall X)\ (p(X) \rightarrow (\exists X)\ q(X))$, o escopo do quantificador $(\forall X)$ é a subfórmula $(p(X) \rightarrow (\exists X)\ q(X))$, enquanto o escopo de $(\exists X)$ é q(X). A ocorrência de X em p(X) está apenas no escopo do quantificador universal. A ocorrência de X em r(X) está no escopo dos dois quantificadores e, por isso, apenas o existencial é considerado. Esta fórmula poderia ser escrita, equivalentemente, como $(\forall X)\ p(X) \rightarrow (\exists X)\ q(X)$.

Uma ocorrência de variável que está fora do escopo de qualquer quantificador é uma ocorrência livre da variável. Dizemos que os símbolos livres de uma fórmula são os símbolos de funções, os símbolos de predicados e as variáveis que ocorrem livre, todos eles precisam ser definidos pela interpretação, para que seja possível interpretar a fórmula como um todo.

Exemplos de Interpretações de Fórmulas

Em todos os exemplos que seguem, considere uma interpretação sobre o domínio dos naturais na qual $I[p(X, Y)] = 1 \Leftrightarrow$ "$I[X] < I[Y]$", $I[X] = 2$ e $I[Y] = 3$.

Exemplo: $F_1 = p(X, Y)$

$I[F_1] = I[p(X, Y] = 1 \quad \Leftrightarrow$ "$I[X] < I[Y]$"
$\Leftrightarrow$ "$2 < 3$"

Que é verdade, pois conhecemos a função "<". Isso é recorrente na lógica de predicados, a necessidade do uso de conhecimentos externos, neste caso da matemática, para a determinação da interpretação de fórmulas.

Note que é impossível criarmos uma tabela verdade para esta fórmula. A tabela verdade deve listar todas as interpretações possíveis, mas neste caso não é possível sequer enumerar todas as possibilidades de X e Y dentro do domínio escolhido, já que é infinito.

Exemplo: $F_2 = (\forall X)\ p(X, Y)$

Neste caso a variável X não está mais livre, de modo que o valor "2" definido na interpretação será ignorado. Vamos avaliar esta fórmula de dois modos, primeiro assumindo que ela é verdadeira.

$I[F_2] = I[(\forall X)\ p(X, Y)] = 1 \quad \Leftrightarrow\ \forall\ a \in \mathbb{N}, \langle X \leftarrow a \rangle I[p(X, Y)] = 1$
$\Leftrightarrow\ \forall\ a \in \mathbb{N}, \langle X \leftarrow a \rangle I[X] < I[Y]$
$\Leftrightarrow\ \forall\ a \in \mathbb{N}, \langle X \leftarrow a \rangle I[X] < 3$
$\Leftrightarrow\ \forall\ a \in \mathbb{N}, a < 3$

Esta última afirmação é falsa, nem todo número natural é menor que 3. Daí concluímos que a suposição inicial, que que $I[F_2] = 1$, é falsa, ou seja, $I[F_2] = 0$.

Agora vamos supor que a fórmula é falsa. Pelo resultado anterior, devemos obter um resultado que não é contraditório.

$I[F_2] = I[(\forall X)\ p(X, Y)] = 0 \quad \Leftrightarrow\ \exists\ a \in \mathbb{N}; \langle X \leftarrow a \rangle I[p(X, Y)] = 0$
$\Leftrightarrow\ \exists\ a \in \mathbb{N}; a < 3$ é falso
$\Leftrightarrow\ \exists\ a \in \mathbb{N}; a \geq 3$

Como esperado, não há contradições, de fato existe um número natural maior ou igual a 3, e a suposição inicial é verdadeira.

Exemplo: $F_3 = (\forall X)\,(\exists Y)\,p(X, Y)$

Neste caso as duas variáveis estão quantificadas, então os valores definidos pela interpretação serão ignorados.

$I[F_3] = I[(\forall X)\,(\exists Y)\,p(X, Y)] = 0 \quad \Leftrightarrow\ \exists\ a \in \mathbb{N};\ \langle X \leftarrow a\rangle I[(\exists Y)\,p(X, Y)] = 0$

$\Leftrightarrow\ \exists\ a \in \mathbb{N};\ \forall\ b \in \mathbb{N},\ \langle X \leftarrow a\rangle\langle Y \leftarrow b\rangle\ I[p(X, Y)] = 0$

$\Leftrightarrow\ \exists\ a \in \mathbb{N};\ \forall\ b \in \mathbb{N},\ a < b$ é falso

$\Leftrightarrow\ \exists\ a \in \mathbb{N};\ \forall\ b \in \mathbb{N},\ a \geq b$

Esta afirmação é falsa, pois diz que há um número natural “a” que é maior que todos os outros números naturais. No entanto, sabemos que os números naturais não têm um limite superior. Portanto, concluímos que nossa suposição inicial estava errada, logo $I[F_3] = 1$.

Exemplo: $F_4 = (\exists Y)\,(\forall X)\,p(X, Y)$

$I[F_4] = I[(\exists Y)\,(\forall X)\,p(X, Y)] = 0 \quad \Leftrightarrow\ \forall\ a \in \mathbb{N},\ \langle Y \leftarrow a\rangle I[(\forall X)\,p(X, Y)] = 0$

$\Leftrightarrow\ \forall\ a \in \mathbb{N},\ \exists\ b \in \mathbb{N};\ \langle Y \leftarrow a\rangle\langle X \leftarrow b\rangle\ I[p(X, Y)] = 0$

$\Leftrightarrow\ \forall\ a \in \mathbb{N},\ \exists\ b \in \mathbb{N};\ b < a$ é falso

$\Leftrightarrow\ \forall\ a \in \mathbb{N},\ \exists\ b \in \mathbb{N};\ b \geq a$

Esta afirmação é verdadeira, pois está sendo dito que para qualquer número natural existe um outro maior ou igual a ele. Então, como suposto, $I[F_4] = 0$.

Os dois últimos exemplos deixam claro que a ordem dos quantificadores é importante para a interpretação da fórmula. Podemos pensar na ordem dos quantificadores como uma sequência de escolhas que são feitas para os valores das variáveis.

No exemplo 3 foi dito que “para qualquer X existe um Y tal que p(X, Y) é verdade”. Neste caso o valor de Y é escolhido depois do valor de X e não necessariamente o mesmo valor de Y deve ser escolhido para todos os X.

No exemplo 4 foi dito que “existe um Y tal que, para qualquer X, p(X, Y) é verdade”. Neste caso o valor de Y é escolhido primeiro e ele é o mesmo para todos os X que são escolhidos depois.

Exemplo: $F_5 = (\forall X)\,(\exists Y)\,p(X, Y) \rightarrow p(X, Y)$

Note que as ocorrências de X e Y em p(X, Y) no consequente da implicação não estão quantificadas.

$I[F_5] = I[(\forall X)\,(\exists Y)\,p(X, Y) \rightarrow p(X, Y)] = 1 \Leftrightarrow I[(\forall X)\,(\exists Y)\,p(X, Y)] = 0$ e/ou $I[p(X, Y)] = 1$

No primeiro exemplo vimos que I[p(X, Y)] = 1, portanto concluímos que $I[F_5] = 1$.

Exemplo: $F_6 = (\forall X)((\exists Y)\, p(X, Y) \rightarrow p(X, Y))$

O quantificador universal está aplicado às duas ocorrências de X, neste caso. Podemos reescrever esta fórmula como:
$F_6 \Leftrightarrow (\forall X)(\exists Y)\, p(X, Y) \rightarrow (\forall X)\, p(X, Y)$

Dos exemplos anteriores, já vimos que o antecedente da implicação é verdadeiro e o consequente da implicação é falso. Da regra de interpretação do operador de implicação, concluímos que $I[F_6] = 0$.

Fazendo diretamente, pelas regras de interpretação, para exercitar, temos:
$I[F_6] = 0 \Leftrightarrow \exists\ a \in \mathbb{N}; \langle X \leftarrow a\rangle I[(\exists Y)\, p(X, Y) \rightarrow p(X, Y)] = 0$
$\Leftrightarrow \exists\ a \in \mathbb{N}; \langle X \leftarrow a\rangle I[(\exists Y)\, p(X, Y)] = 1$ e $\langle X \leftarrow a\rangle I[p(X, Y)] = 0$
$\Leftrightarrow \exists\ a \in \mathbb{N}; \exists\ b \in \mathbb{N}; \langle X \leftarrow a\rangle\langle Y \leftarrow b\rangle I[p(X, Y)] = 1$ e $\langle X \leftarrow a\rangle I[p(X, Y)] = 0$
$\Leftrightarrow \exists\ a \in \mathbb{N}; \exists\ b \in \mathbb{N}; a < b$ e $a \geq 3$
$\Leftrightarrow \exists\ a \in \mathbb{N}; \exists\ b \in \mathbb{N}; 3 \leq a < b$
Como a última afirmação é verdadeira, ou seja, existem números naturais tais que o primeiro é maior ou igual a 3 e o segundo é maior que o primeiro. Concluímos que a suposição estava correta, ou seja, $I[F_6] = 0$.

Exemplo: $F_7 = (\forall X)(\exists Y)(p(X, Y) \rightarrow p(X, Y))$

Vamos supor que esta afirmação seja falsa.

$I[F_7] = 0 \Leftrightarrow \exists\ a \in \mathbb{N}; \langle X \leftarrow a\rangle I[(\exists Y)(p(X, Y) \rightarrow p(X, Y))] = 0$
$\Leftrightarrow \exists\ a \in \mathbb{N}; \forall\ b \in \mathbb{N}, \langle X \leftarrow a\rangle\langle Y \leftarrow b\rangle I[(p(X, Y) \rightarrow p(X, Y))] = 0$
$\Leftrightarrow \exists\ a \in \mathbb{N}; \forall\ b \in \mathbb{N}, \langle X \leftarrow a\rangle\langle Y \leftarrow b\rangle I[(p(X, Y)] = 1$ e $\langle X \leftarrow a\rangle\langle Y \leftarrow b\rangle I[p(X, Y)] = 0$
A última afirmação é falsa, pois é impossível uma mesma interpretação de uma mesma fórmula ser verdadeira e falsa simultaneamente. Portanto, concluímos que a fórmula F_7 é verdadeira.
Neste caso $I[F_7]$ é verdade para qualquer interpretação, pois não foi necessário utilizar nenhum elemento livre, nem mesmo o domínio, para determinar que $I[F_7] = 1$.

Propriedades Semânticas

As propriedades semânticas são as mesmas da lógica proposicional.

- Uma fórmula F é *satisfatível,* ou *factível,* se, e somente se, existe pelo menos uma interpretação I tal que I[F] = 1.
- Uma fórmula F é uma *tautologia*, ou *válida*, se, e somente se, para toda interpretação I, I[F] = 1.

- Uma fórmula F é *contraditória*, ou *insatisfatível*, se, e somente se, para toda interpretação I, I[F] = 0.

Apesar de serem as mesmas, é importante perceber que ao passo que a lógica de predicados nos dá mais liberdade e capacidade de representação, esse aumento de poder vem com um custo na dificuldade da verificação das propriedades semânticas.

Lembre que para o uso efetivo da lógica como uma ferramenta de dedução de conhecimento, para a solução de problemas ou prova de teoremas, é fundamental a verificação de tais propriedades.

Já foi mencionado, na seção anterior, que não possuímos mais as tabelas verdade, pois é impossível listar todas as interpretações possíveis para uma fórmula, visto que são infinitas. As árvores semânticas também não são adequadas pois sofrem do mesmo problema da tabela verdade. Resta-nos usar a definição de interpretação e realizar algumas provas de forma direta ou por contradição, como vínhamos fazendo nos exemplos anteriores.

Para demonstrar que uma fórmula é *satisfatível* ou que *não é tautologia*, basta encontrar uma interpretação verdadeira ou falsa, respectivamente. O problema está na prova de *insatisfatibilidade* e *tautologias*, pois nestes casos precisamos demonstrar que não existe qualquer interpretação que torne a fórmula verdadeira ou falsa, respectivamente.

No restante desta seção apresento exemplos de provas das propriedades semânticas. Alguns dos resultados obtidos nestes exemplos serão utilizados posteriormente, estude-os com atenção.

Determinação de Propriedades Semânticas

Vamos começar retomando os exemplos da seção anterior para os quais obtivemos interpretação falsa e verificar se há alguma interpretação que os tornam verdadeiros.

Exemplo: $F_1 = (\forall X)\ p(X, Y)$

Na seção anterior, vimos que para uma interpretação sobre o domínio dos naturais onde I[Y] = 3 e I[p(X, Y)] = 1 ⇔ I[X] < I[Y], esta fórmula seria interpretada como falsa. No entanto, isso não quer dizer que ela seja falsa para qualquer interpretação.
Considere agora uma interpretação J sobre os naturais onde J[Y] = 0 e J[p(X, Y)] = 1 ⇔ J[X] ≥ J[Y].

$$
\begin{aligned}
J[F_1] = J[(\forall X)\ p(X, Y)] = 1 \quad &\Leftrightarrow \forall\ a \in \mathbb{N}, \langle X \leftarrow a \rangle J[p(X, Y)] = 1 \\
&\Leftrightarrow \forall\ a \in \mathbb{N}, \langle X \leftarrow a \rangle J[X] \geq J[Y] \\
&\Leftrightarrow \forall\ a \in \mathbb{N}, \langle X \leftarrow a \rangle J[X] \geq 0 \\
&\Leftrightarrow \forall\ a \in \mathbb{N}, a \geq 0
\end{aligned}
$$

Esta última afirmação é verdadeira, todo número natural é maior ou igual a 0. Daí concluímos que a suposição inicial, que que $J[F_1] = 1$, está correta.

Como eu encontrei esta interpretação? Para a interpretação I, na seção anterior, chegamos à conclusão que $I[(\forall X)\ p(X, Y)] = 1 \Leftrightarrow \forall\ a \in \mathbb{N}, a < 3$. Sabemos que a afirmação do lado direito da equivalência é falsa para os números naturais. Nessa afirmação, o comparador “menor que” e o valor “3” foram determinados pela interpretação I. O que fiz foi buscar uma

afirmação sabidamente verdadeira, com a mesma estrutura, alterando apenas as partes definidas por I. Assim, J trocou “menor que” por “maior ou igual a” e “3” por “0”.
Este é um exemplo das dificuldades que enfrentamos ao realizar provas na lógica de predicados. Se eu não conseguisse encontrar esta interpretação, isto quer dizer que a fórmula é insatisfatível ou que eu não procurei o suficiente?

Exemplo: $F_2 = (\exists Y)\,(\forall X)\,p(X, Y)$

Na seção anterior, para uma interpretação I sobre os naturais onde $I[p(X, Y)] = 1 \Leftrightarrow I[X] < I[Y]$, chegamos à conclusão que $I[(\exists Y)\,(\forall X)\,p(X, Y)] = 0 \Leftrightarrow \forall\, a \in \mathbb{N},\ \exists\ b \in \mathbb{N};\ b \geq a$. Esta afirmação é verdadeira, pois está sendo dito que para qualquer número natural existe um outro maior ou igual a ele. Então, como suposto, $I[F_4] = 0$. Outra forma é interpretar diretamente o que está escrito na fórmula. Na interpretação dada, $(\exists Y)\,(\forall X)\,p(X, Y)$ significa “existe um número natural que é maior que todos os outros”, que sabemos que é falso.
Utilizando a mesma técnica do exemplo anterior, precisamos que a afirmação do lado direito da equivalência seja falsa. Supondo que vamos manter o domínio, o único elemento livre é a definição do predicado. Seja J uma interpretação sobre os naturais tal que, $J[p(X, Y)] = 1 \Leftrightarrow J[X] \geq J[Y]$. Neste caso a fórmula $(\exists Y)\,(\forall X)\,p(X, Y)$ significa “existe um número natural que é menor ou igual a todos os outros”, que sabemos que é verdade. Vamos verificar:

$$
\begin{aligned}
J[F_2] = J[(\exists Y)\,(\forall X)\,p(X, Y)] = 0 &\Leftrightarrow \forall\, a \in \mathbb{N}, \langle Y \leftarrow a\rangle J[(\forall X)\,p(X, Y)] = 0 \\
&\Leftrightarrow \forall\, a \in \mathbb{N},\ \exists\ b \in \mathbb{N};\ \langle Y \leftarrow a\rangle\langle X \leftarrow b\rangle\, J[p(X, Y)] = 0 \\
&\Leftrightarrow \forall\, a \in \mathbb{N},\ \exists\ b \in \mathbb{N};\ b \geq a \text{ é falso} \\
&\Leftrightarrow \forall\, a \in \mathbb{N},\ \exists\ b \in \mathbb{N};\ b < a
\end{aligned}
$$

A afirmação final é falsa pois se *a* é igual a 0, não existe número *b* menor que *a* dentre os naturais. Como a afirmação final é falsa, concluímos que é falso que $J[F_2] = 0$, logo $J[F_2] = 1$.

Exemplo: $F_3 = (\forall X)\,(\exists Y)\,p(X, Y) \rightarrow (\forall X)\,p(X, Y)$

Dos exemplos anteriores, já vimos que o consequente da implicação é verdadeiro para uma interpretação J sobre os naturais tal que $J[Y] = 0$ e $J[p(X, Y)] = 1 \Leftrightarrow J[X] \geq J[Y]$. Daí, concluímos, pela definição de interpretação do conectivo de implicação, que $I[F_3] = 1$.

Exemplo: $F_4 = \neg(\forall X)\,H \leftrightarrow (\exists X)\,\neg H$

Ao invés de iniciar com uma interpretação e depois ajustar os elementos livres para satisfazer ou não as afirmações, vamos iniciar explorando a interpretação da fórmula em uma interpretação abstrata I.

Pela definição de interpretação do conectivo de bi-implicação, temos que $I[F_4] = 1 \Leftrightarrow I[\neg(\forall X)\, H] = I[(\exists X)\, \neg H]$. Primeiro, vamos considerar que $I[(\exists X)\, \neg H] = 1$.

$$\begin{aligned} I[(\exists X)\, \neg H] = 1 &\Leftrightarrow \exists\ a \in \mathscr{D}, \langle X \leftarrow a\rangle I[\neg H] = 1 \\ &\Leftrightarrow \exists\ a \in \mathscr{D}, \langle X \leftarrow a\rangle I[H] = 0 \\ &\Leftrightarrow I[(\forall X)\, H] = 0 \\ &\Leftrightarrow I[\neg(\forall X)\, H] = 1 \end{aligned}$$

Daí concluímos que a fórmula do lado esquerdo é verdade se, e somente se, a do lado direito também é. Como as fórmulas só têm dois valores possíveis, podemos concluir daí que a fórmula do lado esquerdo é falsa se, e somente se, a do lado direito também é. Mas, neste exemplo excepcionalmente, vamos verificar o caso em que $I[(\exists X)\, \neg H] = 0$.

$$\begin{aligned} I[(\exists X)\, \neg H] = 0 &\Leftrightarrow \forall\ a \in \mathscr{D}, \langle X \leftarrow a\rangle I[\neg H] = 0 \\ &\Leftrightarrow \forall\ a \in \mathscr{D}, \langle X \leftarrow a\rangle I[H] = 1 \\ &\Leftrightarrow I[(\forall X)\, H] = 1 \\ &\Leftrightarrow I[\neg(\forall X)\, H] = 0 \end{aligned}$$

Como esperado, o lado esquerdo da bi-implicação é falso se, e somente se, o lado direito também é. Daí concluímos que $I[\neg(\forall X)\, H] = I[(\exists X)\, \neg H]$ e $I[F_4] = 1$. Observe que em nenhum momento foi necessário definir a interpretação I, de modo que esta afirmação é verdadeira para qualquer interpretação, ou seja, F_4 é uma tautologia.

Vamos refazer esta prova utilizando o método da contradição. Suponha que $I[F_4] = 0$. Para isso, devemos ter $I[\neg(\forall X)\, H] \neq I[(\exists X)\, \neg H]$. Dos desenvolvimentos anteriores, temos que:

$I[\neg(\forall X)\, H] = 1 \Leftrightarrow \exists\ a \in \mathscr{D}, \langle X \leftarrow a\rangle I[H] = 0$

e

$I[(\exists X)\, \neg H] = 0 \Leftrightarrow \forall\ a \in \mathscr{D}, \langle X \leftarrow a\rangle I[H] = 1$

Então, para que F_4 seja falsa, é necessário que H seja verdade para qualquer valor do domínio e, ao mesmo tempo, exista um valor que torna H falso. Isto é uma contradição, de modo que é impossível tornar F_4 falso.

Exemplo: $F_5 = \neg(\exists X)\, H \leftrightarrow (\forall X)\, \neg H$

Como no exemplo anterior, temos que $I[F_5] = 1 \Leftrightarrow I[\neg(\exists X)\, H] = I[(\forall X)\, \neg H]$. Vamos verificar apenas o caso em que $I[\neg(\exists X)\, H] = 1$.

$$\begin{aligned} I[\neg(\exists X)\, H] = 1 &\Leftrightarrow I[(\exists X)\, H] = 0 \\ &\Leftrightarrow \forall\ a \in \mathscr{D}, \langle X \leftarrow a\rangle I[H] = 0 \\ &\Leftrightarrow \forall\ a \in \mathscr{D}, \langle X \leftarrow a\rangle I[\neg H] = 1 \\ &\Leftrightarrow I[(\forall X)\, \neg H] = 1 \end{aligned}$$

Daí concluímos que $I[\neg(\exists X)\, H] = I[(\forall X)\, \neg H]$ e $I[F_5] = 1$ para qualquer interpretação. Ou seja, F_5 é uma tautologia.

As tautologias apresentadas nos dois últimos exemplos apresentam uma relação de equivalência entre os quantificadores, de modo que um pode ser definido em termos do outro e, num alfabeto reduzido da lógica de predicados, apenas um deles seria necessário para definir qualquer fórmula. Então, para relembrar, provamos as seguintes relações de equivalência:

- $\neg(\forall X)\, H \Leftrightarrow (\exists X)\, \neg H$
- $\neg(\exists X)\, H \Leftrightarrow (\forall X)\, \neg H$

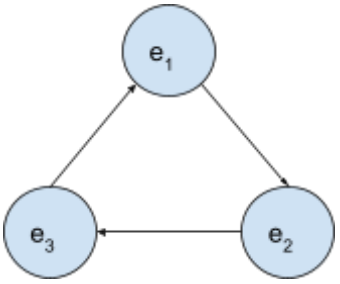
Exemplo: $F_6 = (\forall X)\,(\exists Y)\, p(X, Y) \rightarrow (\exists Y)\,(\forall X)\, p(X, Y)$

Vamos verificar se é possível termos $I[F_6] = 0$. Pela definição da interpretação da implicação, isso só é possível se tivermos simultaneamente $I[(\forall X)\,(\exists Y)\, p(X, Y)] = 1$ e $I[(\exists Y)\,(\forall X)\, p(X, Y)] = 0$.

$I[(\forall X)\,(\exists Y)\, p(X, Y)] = 1 \quad \Leftrightarrow \forall\, a \in \mathscr{D}, \langle X \leftarrow a\rangle I[(\exists Y)\, p(X, Y)] = 1$
$\Leftrightarrow \forall\, a \in \mathscr{D}, \exists\, b \in \mathscr{D}; \langle X \leftarrow a\rangle\langle Y \leftarrow b\rangle I[p(X, Y)] = 1$

$I[(\exists Y)\,(\forall X)\, p(X, Y)] = 0 \quad \Leftrightarrow \forall\, d \in \mathscr{D}, \langle Y \leftarrow d\rangle I[(\forall X)\, p(X, Y)] = 0$
$\Leftrightarrow \forall\, d \in \mathscr{D}, \exists\, c \in \mathscr{D}; \langle Y \leftarrow d\rangle\langle X \leftarrow c\rangle I[p(X, Y)] = 0$

A primeira afirmação diz que para qualquer valor que colocarmos no primeiro argumento, é possível escolher um valor do segundo argumento que satisfaz p. A segunda afirmação diz que para todo valor do segundo argumento é possível escolher um valor para o primeiro argumento que torna p falso. Já vimos um exemplo de interpretação que satisfaz as duas condições na seção anterior.
Uma técnica que pode ajudar a encontrar interpretações que satisfazem tais restrições, para predicados com dois argumentos, especificamente, é utilizar um domínio qualquer e representar o predicado através de setas que conectam os elementos do domínio. Se há uma seta do elemento e_1 para o elemento e_2, então $I[p(e_1, e_2)] = 1$, caso não haja seta, $I[p(e_1, e_2)] = 0$. Considere o domínio $\mathscr{D} = \{e_1, e_2, e_3\}$ e o predicado p definido pelas conexões da figura abaixo:



Todo elemento do domínio aponta seta para alguém, o que satisfaz a primeira afirmação. Todo elemento do domínio não recebe seta de alguém, o que satisfaz a segunda afirmação. Logo, F_6 pode ser falso, e não é uma tautologia.

Exemplo: $F_7 = (\exists Y)\,(\forall X)\, p(X, Y) \rightarrow (\forall X)\,(\exists Y)\, p(X, Y)$

Novamente, vamos verificar se é possível termos $I[F_7] = 0$. Para tal, devemos ter $I[(\forall X)(\exists Y)\, p(X, Y)] = 0$ e $I[(\exists Y)(\forall X)\, p(X, Y)] = 1$.

$I[(\forall X)(\exists Y)\, p(X, Y)] = 0 \quad \Leftrightarrow \exists\, a \in \mathscr{D}, \langle X \leftarrow a\rangle I[(\exists Y)\, p(X, Y)] = 0$
$\Leftrightarrow \exists\, a \in \mathscr{D}, \forall\, b \in \mathscr{D}; \langle X \leftarrow a\rangle\langle Y \leftarrow b\rangle I[p(X, Y)] = 0$

$I[(\exists Y)(\forall X)\, p(X, Y)] = 1 \quad \Leftrightarrow \exists\, d \in \mathscr{D}, \langle Y \leftarrow d\rangle I[(\forall X)\, p(X, Y)] = 1$
$\Leftrightarrow \exists\, d \in \mathscr{D}, \forall\, c \in \mathscr{D}; \langle Y \leftarrow d\rangle\langle X \leftarrow c\rangle I[p(X, Y)] = 1$

A primeira afirmação diz que há um valor, a, do domínio que se for colocado no primeiro argumento, p(a, Y) é falso, para qualquer valor de Y. A segunda afirmação diz que há um valor, d, do domínio que se for colocado no segundo argumento, p(X, d) é verdadeiro, para qualquer valor de X. Estas frases não podem ser satisfeitas simultaneamente, pois p(a, d) deveria ser simultaneamente verdadeiro e falso para que ambas fossem satisfeitas.
Daí, concluímos que é impossível ter uma interpretação I tal que $I[F_7] = 0$, ou seja, $I[F_7] = 1$ para qualquer interpretação, sendo F_7 uma tautologia.
Foi provado então que $(\exists Y)(\forall X)\, p(X, Y) \Rightarrow (\forall X)(\exists Y)\, p(X, Y)$.

Exemplo: $F_8 = (\forall X)\, p(X) \rightarrow p(a)$

Para $I[F_8] = 0$, devemos ter $I[(\forall X)\, p(X)] = 1$ e $I[p(a)] = 0$. $I[(\forall X)\, p(X)] = 1$ quer dizer que qualquer valor do domínio satisfaz p. $I[p(a)] = 0$ diz que há um certo valor do domínio, atribuído à constante a, que torna p falso. As duas coisas não podem ocorrer simultaneamente, portanto F_8 é uma tautologia.
Outro caminho é mostrado a seguir
$I[(\forall X)\, p(X)] = 1 \quad \Leftrightarrow \forall\, d \in \mathscr{D}, \langle X \leftarrow d\rangle I[p(X)] = 1$
$\Rightarrow I[p(a)] = 1$
Concluímos que se $I[(\forall X)\, p(X)] = 1$, então $I[p(a)] = 1$. Esta é a definição de implicação. Deste exemplo concluímos que $(\forall X)\, p(X) \Rightarrow p(a)$, onde a é uma constante qualquer.
Pelo contrapositivo da implicação, se p(a) é falso, $(\forall X)\, p(X)$ também é falso e daí, se p(a) é insatisfatível, $(\forall X)\, p(X)$ também é.
Esta implicação pode ser escrita de forma mais genérica como
$(\forall X)\, H \Rightarrow H\langle X \leftarrow t\rangle$
A fórmula $H\langle X \leftarrow t\rangle$ é a fórmula obtida a partir de H pela substituição de todas as ocorrências livres de X em H pelo termo t (variável ou função). Dizemos que $\langle X \leftarrow t\rangle$ é uma substituição segura se toda variável de t ocorre livre em $H\langle X \leftarrow t\rangle$.

Exemplo: $F_9 = (\exists X)\, p(X) \rightarrow p(a)$

a é uma constante e, como tal, assume um dos valores do domínio. O fato de existir um valor de X que satisfaz p não garante que este valor seja o mesmo que a constante a assume. Deste modo, a afirmação não seria uma tautologia, pois poderíamos ter o antecedente verdadeiro e o consequente falso.

No entanto, vamos incluir uma condição de que a é uma constante cujo valor satisfaz p. Neste caso a implicação é válida pois estamos afirmando que a é um valor que torna o consequente verdadeiro.

$I[(\exists X)\, p(X)] = 1 \quad \Leftrightarrow \exists\, d \in \mathscr{D}, \langle X \leftarrow d\rangle I[p(X)] = 1$
$\Rightarrow I[p(a)] = 1$, se a é um dos valores que satisfaz p.

Podemos pensar nas constantes como apelidos dados aos elementos do domínio. O que queremos dizer neste exemplo é que $(\exists X)\, p(X) \Rightarrow p(a)$ se a é uma constante escolhida para nomear um dos valores do domínio que satisfaz p. Não sabemos quem é esse valor mas, pelo antecedente da implicação, sabemos que ele existe.

Exemplo: Seja H uma fórmula onde X não ocorre livre e G uma fórmula qualquer. Nestas condições, $(\forall X)\,(H \vee G) \Leftrightarrow (H \vee (\forall X)\, G)$.

Seja I uma interpretação tal que $I[(\forall X)\,(H \vee G)] = 1$. Temos que:
$I[(\forall X)\,(H \vee G)] = 1 \quad \Leftrightarrow \forall\, a \in \mathscr{D}, \langle X \leftarrow a\rangle I[H \vee G] = 1$
$\Leftrightarrow \forall\, a \in \mathscr{D}, \langle X \leftarrow a\rangle I[H] = 1$ e/ou $\forall\, a \in \mathscr{D}, \langle X \leftarrow a\rangle I[G] = 1$
Como X não ocorre livre em H, temos que $\forall\, a \in \mathscr{D}, \langle X \leftarrow a\rangle I[H] = 1 \Leftrightarrow I[H] = 1$. Logo,
$I[(\forall X)\,(H \vee G)] = 1 \quad \Leftrightarrow I[H] = 1$ e/ou $\forall\, a \in \mathscr{D}, \langle X \leftarrow a\rangle I[G] = 1$
$\Leftrightarrow I[H] = 1$ e/ou $I[(\forall X)\, G] = 1$
$\Leftrightarrow I[H \vee (\forall X)\, G] = 1$

Exemplo: {H é tautologia} ⇒ {$(\forall X)$ H é tautologia}

Como H é tautologia, para toda interpretação temos que I[H] = 1. Daí temos que
$\forall\, a \in \mathscr{D}, \langle X \leftarrow a\rangle I[H] = 1 \Leftrightarrow I[(\forall X)\, H] = 1$
E isso vale para toda interpretação. Portanto, se H é tautologia, $(\forall X)$ H também é.

Indução Finita

Não vou me estender no assunto, mas é importante lembrar que o princípio da indução finita também pode ser usado na lógica de predicados. A definição do princípio da indução para a lógica de predicado pode ser dada como segue.

Seja A(F) uma afirmação acerca de uma fórmula F da lógica de predicados. Para demonstrar que A(F) é válida para qualquer fórmula F da lógica de predicados, basta demonstrar que:
(Base de Indução):

- $A(p(t_1, \ldots, t_n)$, onde p é um predicado, é válida;
- A(1) é válida;
- A(0) é válida;

(Passo de Indução):

- Se A(G) é válida, então A(¬G) é válida;
- Se A(G) e A(H) são válidas, então A(G $\wedge$ H) é válida;
- Se A(G) e A(H) são válidas, então A(G $\vee$ H) é válida;
- Se A(G) e A(H) são válidas, então A(G $\rightarrow$ H) é válida;
- Se A(G) e A(H) são válidas, então A(G $\leftrightarrow$ H) é válida;
- Se A(G) é válida, então A(($\forall$X) G) é válida;
- Se A(G) é válida, então A(($\exists$X) G) é válida;

De forma resumida, a base de indução mostra que a proposição é válida para todos os átomos (fórmulas de comprimento unitário). E o passo de indução trata de todas as formas de construir fórmulas de comprimento maior. Este conjunto de regras pode ser resumido para qualquer conjunto completo de conectivos. Já vimos que podemos escrever um quantificador em função do outro, então a base e passos de indução para a lógica de predicados podem ser apenas os seguintes.

(Base de Indução):
- A(p(t_1, …, t_n), onde p é um predicado, é válida;
- A(1) é válida;

(Passo de Indução):
- Se A(G) é válida, então A(¬G) é válida;
- Se A(G) e A(H) são válidas, então A(G $\wedge$ H) é válida;
- Se A(G) é válida, então A(($\forall$X) G) é válida;

Sistemas de Dedução

Nesta seção vamos estender para a lógica de predicados os três sistemas de dedução apresentados para a lógica proposicional.

Sistema Axiomático

Na lógica de predicados o sistema axiomático sofre dos mesmos problemas que o sistema axiomático na lógica proposicional, é extremamente difícil obter uma implementação eficiente deste método. Por isso, apresentarei o sistema brevemente e não me alongarei muito nos exemplos.

Os axiomas deste sistema são os mesmos da lógica proposicional acrescidos de mais dois, cuja validade já foi demonstrada nos exemplos anteriores.

- A_1 = (H $\vee$ H) $\rightarrow$ H
- A_2 = H $\rightarrow$ (G $\vee$ H)
- A_3 = (H $\rightarrow$ G) $\rightarrow$ ((F $\vee$ H) $\rightarrow$ (G $\vee$ F))
- A_4 = ($\forall$X) H $\rightarrow$ H$\langle$X $\leftarrow$ t$\rangle$, onde $\langle$X $\leftarrow$ t$\rangle$ é uma substituição segura.
- A_5 = ($\forall$X) (H $\vee$ G) $\rightarrow$ (H $\vee$ ($\forall$X) G), onde X não ocorre livre em H.

Vamos considerar além dos axiomas as seguintes equivalências como válidas para poder tratar todas as fórmulas da lógica de predicados. A validade da nova equivalência, E_4, também foi demonstrada nos exemplos anteriores.

- $E_1 = (X \rightarrow Y) \Leftrightarrow (\neg X \vee Y)$
- $E_2 = (X \leftrightarrow Y) \Leftrightarrow ((X \rightarrow Y) \wedge (Y \rightarrow X))$
- $E_3 = (X \wedge Y) \Leftrightarrow \neg(\neg X \vee \neg Y)$
- $E_4 = (\exists X)\, H \Leftrightarrow \neg(\forall X)\, \neg H$

Além dos cinco axiomas e das quatro equivalências, dois mecanismos de dedução serão utilizados, o *modus ponens* e a *generalização,* que podem ser enunciados, respectivamente como segue:

- $H \wedge (H \rightarrow G) \Rightarrow G$
- $H \Rightarrow (\forall X)\, H$

A validade da generalização já foi demonstrada nos exemplos anteriores. Por fim, vamos definir o mecanismo de prova no sistema axiomático:

Seja F uma fórmula e □ um conjunto de fórmulas, denominadas de hipóteses. Uma prova de F a partir de □, no sistema axiomático, é uma sequência de fórmulas $F_1, F_2, \ldots, F_n$, onde se tem

- $F = F_n$
  e, para todo *i* tal que $1 \leq i \leq n$,
- F_i é um axioma ou,
- $F_i \in$ □ ou,
- F_i é deduzida de F_j e F_k, utilizando a regra *modus ponens*, onde $j < i$ e $k < i$, ou
- F_i é deduzida de F_j, utilizando a regra de *generalização*, onde $j < i$.

Se a prova existe, dizemos que F é consequência lógica de □. Ou, que □ implica em F. Denotaremos esta afirmação por □ $\models$ F ou □ $\Rightarrow$ F. Se o conjunto de hipóteses é vazio, ou seja, se F pode ser provado apenas a partir dos axiomas, dizemos que F é um teorema do sistema axiomático e denotaremos por $\models$ F.

Exemplo: Se X não ocorre livre em A, então $(A \rightarrow B) \Rightarrow (A \rightarrow (\forall X)\, B)$
Prova:

1. $(\neg A \vee B)$ [hipótese]
2. $(\forall X)\, (\neg A \vee B)$ [generalização de 1.]
3. $(\forall X)\, (\neg A \vee B) \rightarrow (\neg A \vee (\forall X)\, B)$ [A_5]
4. $(\neg A \vee (\forall X)\, B)$ [MP(2, 3)]
5. $(A \rightarrow (\forall X)\, B)$ [reescrita de 4.]

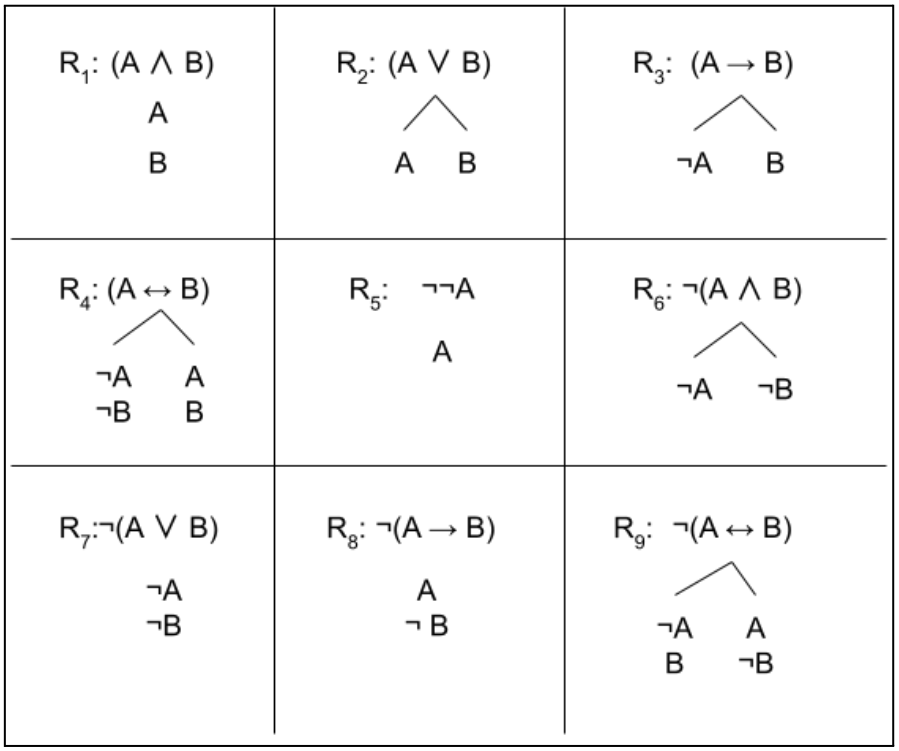
Exemplo: $(\forall X)\, (\forall Y)\, A \Rightarrow (\forall Y)\, (\forall X)\, A$
Prova:

1. $(\forall X)\, (\forall Y)\, A$ [hipótese]
2. $(\forall X)\, (\forall Y)\, A \rightarrow (\forall Y)\, A$ [A_4 com $\{X \leftarrow X\}$]
3. $(\forall Y)\, A$ [MP(1, 2)]
4. $(\forall Y)\, A \rightarrow A$ [A_4 com $\{Y \leftarrow Y\}$]

5. A [MP(3, 4)]
6. $(\forall X)$ A [generalização de 5.]
7. $(\forall Y)(\forall X)$ A [generalização de 6.]

Sistema de *Tableaux* Semânticos

Assim como ocorreu para o sistema axiomático, vamos apenas estender o sistema que já conhecíamos para a lógica proposicional. O novo sistema deve valer para as fórmulas da lógica proposicional, de tal modo que todas as regras já vistas anteriormente continuam valendo:



Como discutido anteriormente, a essência do *tableau* consiste numa decomposição das fórmulas até os literais. Fórmulas que ocorrem em um mesmo ramo do *tableau* precisam ser satisfeitas simultaneamente para que a fórmula na raiz seja satisfeita, enquanto os diferentes ramos representam alternativas para satisfazer a fórmula raiz.

Deste modo, se um ramo apresenta uma contradição (literais complementares) ele não é capaz de satisfazer a fórmula. Se todos os ramos apresentam contradições, ou seja, são fechados, dizemos que a fórmula raiz é insatisfatível.

As nove regras que herdamos da lógica proposicional são capazes de tratar todos os conectivos lógicos, mas as fórmulas da lógica de predicados podem conter também quantificadores. Portanto, precisamos de quatro novas regras para tratar estes elementos, duas para a negação dos quantificadores e duas para os quantificadores não negados.

Nos capítulos anteriores já vimos duas equivalências que nos permitem reescrever quantificadores negados em função do outro, uma forma de internalizar as negações. Estas equivalências serão duas das novas regras:

R_{10}: $\neg(\forall X)\ A$	R_{11}: $\neg(\exists X)\ A$
$(\exists X)\ \neg A$	$(\forall X)\ \neg A$

Também já vimos duas implicações que nos permitem remover um quantificador e trocá-lo por uma constante:

R_{12}: $(\forall X)\ A$	R_{13}: $(\exists X)\ A$
$A\{X \leftarrow c\}$	$A\{X \leftarrow c\}$, onde c é uma constante nova.

No caso dos quantificadores universais, a constante usada na substituição é livre, enquanto no caso dos quantificadores existenciais esta constante deve ser nova no ramo em questão. Esta restrição se dá porque estamos dando um nome para um valor que sabemos que existe e satisfaz A, mas não sabemos qual é. Se usássemos um nome que já ocorreu correríamos o risco de cometer um erro, ao usar um nome novo esse risco não existe. Considere o exemplo que segue:

Exemplo: $(\exists X)$ p(a, X).

A fórmula diz que há um valor para o segundo argumento do predicado p que, utilizado em conjunto com o valor a no primeiro argumento, satisfaz p. Não podemos concluir que p(a, a) é verdade pois nada garante que o mesmo valor utilizado no primeiro argumento também satisfaz no segundo argumento. No entanto, p(a, b) é verdade, pois sabemos que existe um valor, que decidimos chamar de b, esse valor pode ser igual a a ou não, mas não temos como afirmar isso a partir do que foi exposto.

Vamos considerar uma interpretação específica. Seja I uma interpretação sobre os números naturais tal que I[p(X, Y)]: “X é menor que Y” e I[a] = 2. Nesse caso a interpretação da fórmula I[$(\exists X)$ p(a, X)] = “$\exists\ b \in \mathbb{N}$; 2 < b”. Ou seja, há um valor natural maior que 2, o que é verdade. Perceba que o valor desconhecido foi chamado de b porque se fosse chamado de *a* a interpretação seria “2 < 2”, que é falsa.

Construção do *Tableau*

A construção do *tableau* semântico associado à fórmula F é feita de forma semelhante à construção do tableau para a lógica proposicional:

1. Inicie com a fórmula F.
2. Escolha uma fórmula que já ocorreu no *tableau*.
    a. Aplique a regra adequada e reescreva a fórmula na base do *tableau*.

Uma heurística que normalmente vai poupar trabalho na construção dos *tableaux* é, sempre que possível, utilizar primeiro as regras que não provocam bifurcações.

No caso da lógica proposicional, foi imposta uma regra de que cada fórmula só seria utilizada uma vez, e para a maioria dos casos isso é verdade. As regras R_1 a R_{11} são equivalências e o resultado da aplicação destas regras é único, portanto não faz sentido

expandir mais de uma vez as fórmulas que utilizam estas regras, o resultado não estará errado, mas é um esforço que não trará nenhuma informação extra.

Já as regras R_{12} e R_{13} podem produzir resultados distintos a cada aplicação, uma vez que a substituição é livre (a menos da restrição no caso existencial). Portanto, às vezes pode ser necessário aplicar tais regras mais de uma vez à mesma fórmula.

Outra heurística importante vem da restrição presente na regra da eliminação do quantificador existencial. Perceba que as duas regras de eliminação de quantificadores são idênticas a menos da restrição. Por isso, é sempre interessante, quando houver opção, eliminar os existenciais (que têm restrições) antes dos universais. Ou seja, aplicar a R_{13} antes de R_{12}.

Prova por Tableau

Já vimos que a análise do tableau associado a uma fórmula nos permite determinar se a fórmula é satisfatível ou não. Uma fórmula insatisfatível é uma fórmula que é sempre falsa. Logo, a negação dessa fórmula é sempre verdadeira, ou seja, uma tautologia.

Utilizando este raciocínio, podemos usar o seguinte procedimento para provar a validade de uma fórmula utilizando tableau:

Dada uma fórmula F, para determinar se F é tautologia faça:

3. Construa um tableau associado a ¬F.
4. Se há tableau associado a ¬F é fechado:
    a. concluímos que ¬F é insatisfatível;
    b. logo, F é tautologia.

Dizemos que F é um teorema do sistema de tableaux. Como no caso do sistema axiomático, se Γ é um conjunto de hipóteses $\Gamma = \{H_1, H_2, \ldots, H_n\}$, dizemos que o conjunto implica em F ou que F é consequência lógica de Γ se existe uma prova de $(H_1 \wedge H_2 \wedge \ldots \wedge H_n) \rightarrow F$.

Há uma diferença fundamental entre o uso do tableau na lógica de predicados e na lógica proposicional. Na lógica de predicados dizemos que há uma prova se há pelo menos um tableau fechado associado a ¬F enquanto na lógica proposicional todo tableau associado ¬F é fechado se F é tautologia.

Isso quer dizer que há a possibilidade de obtermos tableaux abertos mesmo em casos onde a fórmula ¬F é insatisfatível. Essa possibilidade vem das regras de eliminação dos quantificadores e, por isso, não ocorria na lógica proposicional. Na seção que segue veremos diversos exemplos de aplicação dos tableaux semânticos para a determinação de propriedades semânticas de fórmulas da lógica de predicados.

Exemplos de Uso dos Tableaux

Nestes exemplos devem ficar mais claras a origem das observações feitas até aqui acerca das diferenças entre os tableaux na lógica proposicional e de predicados.

Exemplo: $H_1 = (\forall X)\,(\forall Y)\, p(X, Y) \rightarrow p(a, a)$ é tautologia?

Para saber se esta fórmula é tautologia devemos fazer o tableau de $\neg H_1$ e verificar se ele é fechado, ou seja, devemos saber se $\neg H_1$ é insatisfatível.

$$\neg H_1 = \neg((\forall X)\,(\forall Y)\, p(X, Y) \rightarrow p(a, a))$$

$$(\forall X)\,(\forall Y)\,p(X, Y)$$

$$\neg p(a, a)$$

$$(\forall Y)\,p(a, Y)$$

$$p(a, a)$$
fechado

No primeiro passo aplicamos a regra R_8 a $\neg H_1$. No segundo passo aplicamos a regra R_{12} a $(\forall X)\,(\forall Y)\,p(X, Y)$ com $\{X \leftarrow a\}$ e depois aplicamos a regra R_{12} em $(\forall Y)\,p(a, Y)$ com $\{Y \leftarrow a\}$. Desse modo obtivemos um tableau fechado. Perceba que qualquer outro par de substituições escolhido levaria a um tableau aberto.
Por isso, a substituição deve ser guiada para provocar a contradição com algum literal já existente. Neste caso, tínhamos $\neg p(a, a)$ e, por isso, foi escolhido substituir X e Y por a no predicado p(X, Y). De modo geral, como nosso objetivo é encontrar contradições, é frequente a reutilização de constantes que já ocorreram quando a regra R_{12} é aplicada.

Exemplo: p(a,a) é consequência lógica de $(\exists X)\,(\exists Y)\,p(X,Y)$?
A pergunta pode ser reescrita como $H_2 = (\exists X)\,(\exists Y)\,p(X,Y) \rightarrow p(a, a)$ é tautologia? Daí aplicamos um raciocínio similar ao do caso anterior:

$$\neg H_2 = \neg((\exists X)\,(\exists Y)\,p(X, Y) \rightarrow p(a, a))$$

$$(\exists X)\,(\exists Y)\,p(X, Y)$$

$$\neg p(a, a)$$

$$(\exists Y)\,p(b, Y)$$

$$p(b, c)$$
aberto

No primeiro passo aplicamos a regra R_8 a $\neg H_2$. No segundo passo aplicamos a regra R_{13} a $(\exists X)\,(\exists Y)\,p(X, Y)$ com $\{X \leftarrow b\}$ e depois aplicamos a regra R_{13} em $(\exists Y)\,p(b, Y)$ com $\{Y \leftarrow c\}$. Desse modo obtivemos um tableau aberto. Neste caso, pelas restrições da regra de eliminação do quantificador existencial, não pudemos reutilizar a constante a ao substituir X e não pudemos utilizar nem a nem b ao substituir Y.
Na lógica de proposicional poderíamos concluir imediatamente que $\neg H_2$ é satisfatível, ou seja, que H_2 não é tautologia. Mas na lógica de predicados precisamos antes ter certeza de que não há qualquer outro tableau associado a $\neg H_2$ que seja fechado. Perceba que, neste exemplo, só seria possível chegar a um tableau fechado se o predicado p(X, Y), na linha 2 puder ser transformado no predicado $\neg p(a, a)$, presente na linha 3.
No entanto, devido às restrições de R_{13}, nunca poderemos substituir X ou Y por a, daí, qualquer tableau associado a $\neg H_2$ será aberto e de fato p(a, a) não é consequência lógica de $(\exists X)\,(\exists Y)\,p(X, Y)$.

Vamos aproveitar este exemplo para interpretar um pouco o que está escrito na expansão. Ao aplicar R_8 em $\neg H_2$ concluímos que para H_2 ser falso devemos ter $(\exists X)\ (\exists Y)\ p(X, Y)$ verdadeiro (linha 2) e p(a, a) falso (linha 3). Na continuação da expansão chegamos até p(b, c), que podemos entender como se existem X e Y que satisfazem p(X, Y), sabemos que p(b, c) é verdade, mas não podemos garantir que os valores de a, b e c são idênticos.

Exemplo: $H_3 = \neg((\forall X)\ (p(X) \wedge q(x)) \rightarrow (\forall X)\ p(x))$ é satisfatível?
Para responder essa pergunta, basta fazer a expansão de H_3

$$H_3 = \neg((\forall X)\ (p(X) \wedge q(x)) \rightarrow (\forall X)\ p(x))$$

$$(\forall X)\ (p(X) \wedge q(x))$$

$$\neg(\forall X)\ p(X)$$

$$(\exists X)\ \neg p(X)$$

$$p(a) \wedge q(a)$$

$$p(a)$$

$$q(a)$$

$$\neg p(b)$$

aberto

Na linha 5, obtivemos $(p(a) \wedge q(a))$ através da aplicação de R_{12} na fórmula da linha 2. Depois disso, obtivemos $\neg p(b)$ na última linha através da aplicação de R_{13} na fórmula da linha 4. Neste caso tivemos de usar uma constante diferente de a porque a constante *a* já havia ocorrido anteriormente. Deste modo chegamos a um tableau aberto.
Mas o que aconteceria se fizéssemos as escolhas numa ordem distinta:

$$H_3 = \neg((\forall X)\ (p(X) \wedge q(x)) \rightarrow (\forall X)\ p(x))$$

$$(\forall X)\ (p(X) \wedge q(x))$$

$$\neg(\forall X)\ p(X)$$

$$(\exists X)\ \neg p(X)$$

$$\neg p(a)$$

$$p(a) \wedge q(a)$$

$$p(a)$$

q(a)
fechado

Agora, obtivemos ¬p(a) na linha 5 através da aplicação de R_{13} à linha 4 e depois obtivemos (p(a) ∧ q(a)) na linha 6 através de aplicação de R_{12} à linha 2. Agora temos uma contradição entre p(a) e ¬p(a) e o tableau é fechado. Como há um tableau fechado associado a H_3, concluímos que H_3 é instatisfatível.

Como o objetivo é encontrar a contradição, é sempre vantajoso utilizar R_{13} antes de R_{12} pois se R_{13} é utilizada depois não conseguiremos obter contradições, já que ela nos obrigará a utilizar constantes distintas das que já existem.

Exemplo: (∃X) (p(X) → q(X)) é equivalente a (∀X) p(X) → (∃X) q(X) ?
Para responder isso temos que verificar se a relação

$$H_4 = (\exists X)\,(p(X) \rightarrow q(X)) \leftrightarrow ((\forall X)\,p(X) \rightarrow (\exists X)\,q(X))$$

é uma tautologia. Poderíamos fazer através de duas verificações de implicação, mas vamos fazê-la diretamente utilizando tableau.

$$\neg H_4 = \neg((\exists X)\,(p(X) \rightarrow q(X)) \leftrightarrow ((\forall X)\,p(X) \rightarrow (\exists X)\,q(X)))$$

(∃X) (p(X) → q(X))		¬(∃X) (p(X) → q(X))	
¬((∀X) p(X) → (∃X) q(X))		(∀X) p(X) → (∃X) q(X)	
(∀X) p(X)		(∀X) ¬(p(X) → q(X))	
¬(∃X) q(X)		¬(∀X) p(X)	(∃X) q(X)
(∀X) ¬q(X)		(∃X) ¬p(X)	q(a)
p(a) → q(a)		¬p(a)	¬(p(a) → q(a))
¬p(a)	q(a)	¬(p(a) → q(a))	p(a)
p(a) *fechado*	p(a)	p(a)	¬q(a) *fechado*
	¬q(a) *fechado*	¬q(a) *fechado*	

Como todos os ramos são fechados, concluímos que as duas fórmulas são equivalentes. Neste caso, todas as vezes que aplicamos R_{13} fizemos substituições {X ← a}. Não há violação da regra, pela repetição da constante, porque as substituições ocorrem em ramos distintos do tableau.

Exemplo: $(\forall X)\,(p(X) \wedge q(X)) \leftrightarrow ((\forall X)\,p(X) \wedge (\forall X)\,q(X))$ é tautologia?

Vamos verificar esta afirmação através de dois tableaux. Considere $H_5 = (\forall X)\,(p(X) \wedge q(X))$ e $H_6 = ((\forall X)\,p(X) \wedge (\forall X)\,q(X))$. Sabemos que $H_5 \leftrightarrow H_6 \Leftrightarrow (H_5 \rightarrow H_6) \wedge (H_6 \rightarrow H_5)$, daí temos que $H_5 \leftrightarrow H_6$ só é tautologia se $H_5 \rightarrow H_6$ e $H_6 \rightarrow H_5$ são tautologias. Em outras palavras, H_5 equivale a H_6 se H_5 implica em H_6 e H_6 implica em H_5.

Parte 1: $H_5 \Rightarrow H_6$?

$$\neg(H_5 \rightarrow H_6)$$

$$H_5 = (\forall X)\,(p(X) \wedge q(X))$$

$$\neg H_6 = \neg((\forall X)\,p(X) \wedge (\forall X)\,q(X))$$

$\neg(\forall X)\,p(X)$	$\neg(\forall X)\,q(X)$
$(\exists X)\,\neg p(X)$	$(\exists X)\,\neg q(X)$
$\neg p(a)$	$\neg q(a)$
$p(a) \wedge q(a)$	$p(a) \wedge q(a)$
$p(a)$	$p(a)$
$q(a)$	$q(a)$
fechado	*fechado*

Portanto, concluímos que $H_5 \Rightarrow H_6$.

Parte 2: $H_6 \Rightarrow H_5$?

$$\neg(H_6 \rightarrow H_5)$$

$$H_6 = (\forall X)\,p(X) \wedge (\forall X)\,q(X)$$

$$\neg H_5 = \neg(\forall X)\,(p(X) \wedge q(X))$$

$$(\exists X)\,\neg(p(X) \wedge q(X))$$

$$(\forall X)\,p(X)$$

$$(\forall X)\,q(X)$$

$$\neg(p(a) \wedge q(a))$$

$\neg p(a)$	$\neg q(a)$
$p(a)$	$p(a)$

fechado

q(a)

fechado

Concluímos daí que $H_6 \Rightarrow H_5$. Logo, $H_5 \Leftrightarrow H_6$.

Exemplo: $H_7 = (\exists X)\,(p(X) \wedge q(X))$ e $H_8 = (\exists X)\,p(X) \wedge (\exists X)\,q(X)$ são equivalentes?

Novamente, vamos fazer a prova em duas partes.
Parte 1: $H_7 \Rightarrow H_8$?

$$\neg(H_7 \rightarrow H_8)$$

$$H_7 = (\exists X)\,(p(X) \wedge q(X))$$

$$\neg H_8 = \neg((\exists X)\,p(X) \wedge (\exists X)\,q(X))$$

$$p(a) \wedge q(a)$$

$$p(a)$$

$$q(a)$$

↙	↘
$\neg(\exists X)\,p(X)$	$\neg(\exists X)\,q(X)$
$(\forall X)\,\neg p(X)$	$(\forall X)\,\neg q(X)$
$\neg p(a)$	$\neg q(a)$
fechado	*fechado*

Logo, $H_7 \Rightarrow H_8$.

Parte 2: $H_8 \Rightarrow H_7$?

$$\neg(H_8 \rightarrow H_7)$$

$$H_8 = (\exists X)\,p(X) \wedge (\exists X)\,q(X)$$

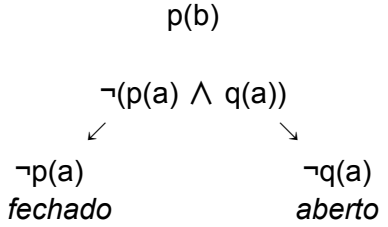
$$\neg H_7 = \neg(\exists X)\,(p(X) \wedge q(X))$$

$$(\forall X)\,\neg(p(X) \wedge q(X))$$

$$(\exists X)\,p(X)$$

$$(\exists X)\,q(X)$$

$$p(a)$$



Neste caso, não chegamos a um tableau fechado. Não importa em qual ordem façamos as substituições, pelo menos um dos ramos permanecerá aberto pois cada um dos existenciais é substituído por uma constante distinta e temos apenas uma constante que é inserida pelo universal. Portanto, H_8 não implica em H_7.

Vamos avaliar o significado das fórmulas H_7 e H_8 para tentar entender porque a implicação só vale em um sentido. $H_7 = (\exists X)\,(p(X) \wedge q(X))$ diz que há um valor do domínio que satisfaz a conjunção de p e q, ou seja, esse valor satisfaz simultaneamente os dois predicados.

$H_8 = (\exists X)\,p(X) \wedge (\exists X)\,q(X)$ também diz que os dois predicados podem ser satisfeitos simultaneamente, mas há uma diferença fundamental, em H_8 não é dito que é o mesmo valor que satisfaz ambos. Apesar da variável X ocorrer nos dois predicados em H_8, perceba que elas estão no escopo de quantificadores distintos e, para todos os efeitos, podem ser tratadas como variáveis distintas, ou seja, $(\exists X)\,p(X) \wedge (\exists X)\,q(X) \Leftrightarrow (\exists X)\,p(X) \wedge (\exists Y)\,q(Y)$.

Então H_7 diz que há um único valor que satisfaz os dois predicados enquanto H_8 diz que há valores que satisfazem os predicados sem especificar se eles são idênticos ou não. Por isso, H_8 não implica em H_7.

Para os exemplos que seguem considere conjunto de premissas $\mathscr{P} = \{P_1, P_2, P_3, P_4, P_5\}$ onde:

$P_1 = (\forall X)\,(\forall Y)\,(s(X, Y) \rightarrow m(X, Y))$
$P_2 = (\forall X)\,(\forall Y)\,(\forall Z)\,((s(Z, Y) \wedge m(X, Z)) \rightarrow m(X, Y))$
$P_3 = s(a, b)$
$P_4 = s(b, c)$
$P_5 = s(c, d)$

As premissas P_3 a P_5 são afirmações acerca do predicado s. As premissas P_1 e P_2 podem ser entendidas como uma definição recursiva do predicado m em função dos predicados m e s.

Exemplo: m(a, b) é consequência lógica de $\mathscr{P}$?
Essa pergunta pode ser entendida como "$\mathscr{P} \rightarrow$ m(a, b) é tautologia?", ou ainda "$(P_1 \wedge P_2 \wedge P_3 \wedge P_4 \wedge P_5) \rightarrow$ m(a, b) é tautologia?". Vamos utilizar o tableau para fazer esta prova, lembre que devemos começar com a negação do que queremos provar:

$$\neg((P_1 \wedge P_2 \wedge P_3 \wedge P_4 \wedge P_5) \rightarrow m(a, b))$$

$$(P_1 \wedge P_2 \wedge P_3 \wedge P_4 \wedge P_5)$$

$$\neg m(a, b)$$

$$P_1 = (\forall X)\,(\forall Y)\,(s(X, Y) \rightarrow m(X, Y))$$

$$P_2 = (\forall X)\,(\forall Y)\,(\forall Z)\,((s(Z, Y) \wedge m(X, Z)) \rightarrow m(X, Y))$$

$$P_3 = s(a, b)$$

$$P_4 = s(b, c)$$

$$P_5 = s(c, d)$$

$$(\forall Y)\,(s(a, Y) \rightarrow m(a, Y))$$

$$s(a, b) \rightarrow m(a, b)$$

$$\swarrow \qquad \searrow$$

$$\neg s(a, b) \qquad m(a, b)$$

fechado *fechado*

Concluímos então que m(a, b) é consequência lógica de $\mathscr{P}$.

Perceba que em todos os casos que tentamos provar se ($\mathscr{P} \rightarrow$ X) é tautologia o tableau começará com a lista das premissas seguido de X negado. Nos próximos exemplos aproveitaremos isso para reduzir o tamanho do tableau descrito. Além disso, nas fórmulas que têm múltiplos quantificadores universais, vamos fazer as substituições simultaneamente, já que a substituição do quantificador universal é livre.

Exemplo: m(a, c) é consequência lógica de $\mathscr{P}$?
Vamos fazer o tableau aproveitando as observações feitas no exemplo anterior:

$$\neg(\mathscr{P} \rightarrow m(a, c))$$

$$P_1 = (\forall X)\,(\forall Y)\,(s(X, Y) \rightarrow m(X, Y))$$

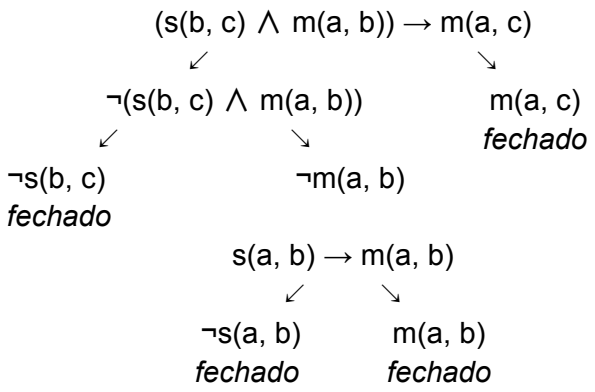
$$P_2 = (\forall X)\,(\forall Y)\,(\forall Z)\,((s(Z, Y) \wedge m(X, Z)) \rightarrow m(X, Y))$$

$$P_3 = s(a, b)$$

$$P_4 = s(b, c)$$

$$P_5 = s(c, d)$$

$$\neg m(a, c)$$



Concluímos que m(a, c) também é consequência lógica de $\mathscr{P}$.

Exemplo: m(a, d) é consequência lógica de $\mathscr{P}$?

$$\neg(\mathscr{P} \rightarrow m(a, d))$$

$$P_1 = (\forall X)\,(\forall Y)\,(s(X, Y) \rightarrow m(X, Y))$$

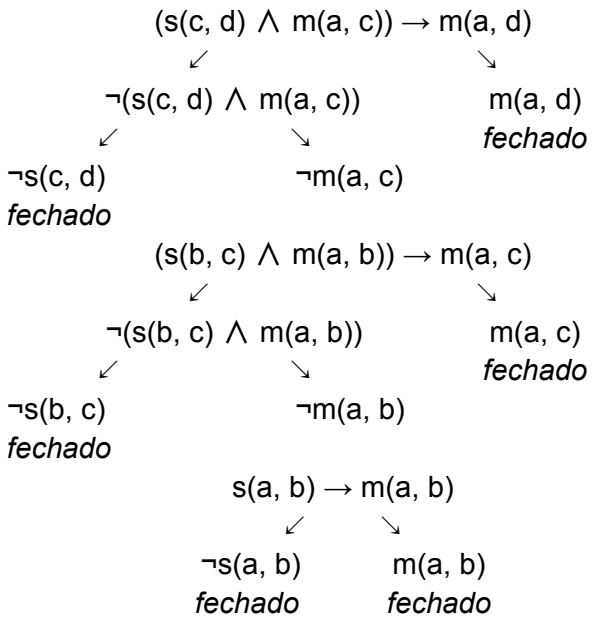
$$P_2 = (\forall X)\,(\forall Y)\,(\forall Z)\,((s(Z, Y) \wedge m(X, Z)) \rightarrow m(X, Y))$$

$$P_3 = s(a, b)$$

$$P_4 = s(b, c)$$

$$P_5 = s(c, d)$$

$$\neg m(a, d)$$



Concluímos que m(a, d) é consequência lógica de $\mathscr{P}$. Neste caso utilizamos a regra P_2 duas vezes com duas substituições distintas. No primeiro caso foi usado {X ← a, Y ← d, Z ← c} e no segundo {X ← a, Y ← c, Z ← b}. Este exemplo deixa claro que às vezes é necessário expandir a mesma fórmula mais de uma vez, sem isso não seria possível chegar ao tableau fechado.

Exemplo: m(a, e) é consequência lógica de $\mathscr{P}$?

$$\neg(\mathscr{P} \rightarrow m(a, e))$$

$$P_1 = (\forall X)\,(\forall Y)\,(s(X, Y) \rightarrow m(X, Y))$$

$$P_2 = (\forall X)\,(\forall Y)\,(\forall Z)\,((s(Z, Y) \wedge m(X, Z)) \rightarrow m(X, Y))$$

$$P_3 = s(a, b)$$

$$P_4 = s(b, c)$$

$$P_5 = s(c, d)$$

$$\neg m(a, e)$$

Antes de continuar o desenvolvimento deste exemplo, vamos parar e pensar um pouco nas possibilidades. Se o conjunto de premissas for insatisfatível, qualquer coisa pode ser provada a partir deles. Este conjunto é satisfatível pois qualquer desenvolvimento dele vai aparecer m(X, Y) em algum ramo, e nunca vai ocorrer m(X, Y) negativo, de modo que pelo menos um ramo estará sempre aberto.
Assim, concluímos que se for possível chegar à cláusula vazia é necessário usar a cláusula referente ao que queremos provar (¬m(a, e)).
Existem duas formas de se obter m(a, e) positivo:
Utilizando P_1 com {X ← a, Y ← e}, desse modo vai aparecer ¬s(a, e) em um ramo, e esse literal não ocorre positivo, portanto este ramo permanecerá aberto.
Utilizando P_2 com a mesma substituição teremos ¬s(Z, e) e ¬m(a, Z), onde Z será substituído por alguma constante. O ramo que tem ¬s(Z, e) permanecerá aberto porque não tem como conseguir o predicado s positivo com a constante e no segundo argumento.
Portanto, é impossível chegar à cláusula vazia e concluímos que m(a, e) não é consequência lógica de $\mathscr{P}$. Isso quer dizer que m(a, e) é falso? Não necessariamente, isso quer dizer apenas que a partir das premissas dadas não podemos provar m(a, e).
Imagine uma interpretação em que s(X, Y) significa “a letra Y sucede a letra X no alfabeto” e m(X, Y) significa “X ocorre antes de Y no alfabeto”. Neste caso m(a, e) seria verdade, pelo que conhecemos do alfabeto, mas não podemos provar a partir das informações que foram dadas. Foi dito apenas que b sucede a, c sucede b e d sucede c.

Exemplo: $(\exists X)\ m(X, c)$ é consequência lógica de $\mathscr{P}$?

Essa pergunta é mais genérica que os exemplos anteriores. Nos casos dos exemplos 7 a 10 a pergunta era definida, cuja resposta era sim ou não, agora temos uma pergunta aberta "existe algum valor que se for colocado no primeiro argumento do predicado m e utilizando c como segundo argumento satisfaz m?" ou, utilizando a interpretação sugerida no exemplo anterior "existe alguma letra que vem antes de c no alfabeto?". Sabemos que a resposta a esta pergunta é "sim", mas falta verificar se podemos provar isso a partir das premissas apresentadas.

$$\neg(\mathscr{P} \rightarrow (\exists X)\ m(X, c))$$

$$P_1 = (\forall X)\ (\forall Y)\ (s(X, Y) \rightarrow m(X, Y))$$

$$P_2 = (\forall X)\ (\forall Y)\ (\forall Z)\ ((s(Z, Y) \wedge m(X, Z)) \rightarrow m(X, Y))$$

$$P_3 = s(a, b)$$

$$P_4 = s(b, c)$$

$$P_5 = s(c, d)$$

$$\neg(\exists X)\ m(X, c)$$

$$(\forall X)\ \neg m(X, c)$$

$$\neg m(b, c)$$

$$s(b, c) \rightarrow m(b, c)$$

$\swarrow$	$\searrow$
$\neg s(b, c)$	$m(b, c)$
fechado	*fechado*

Na linha 9 substituímos X por b na fórmula da linha 8 e chegamos a um tableau fechado. De onde concluímos que a resposta da pergunta é "sim", existe um valor de X que satisfaz m(X, c). Além disso, se prestarmos atenção na expansão do tableau, vemos que a substituição que satisfaz é $\{X \leftarrow b\}$, ou seja, m(b, c) é consequência lógica de $\mathscr{P}$.
Do exemplo 8, também vimos um tableau fechado que mostra que m(a, c) é consequência lógica de $\mathscr{P}$. Perceba que se fizéssemos a substituição $\{X \leftarrow a\}$ também chegaríamos a um tableau fechado, semelhante àquele do exemplo 8. Pensando na interpretação sugerida no exemplo 10, a pergunta feita era "existe alguma letra que vem antes de c?" e a partir das premissas concluímos que a e b vêm antes de c, pois são as únicas substituições que levam a um tableau fechado.
Existem outras letras que vêm antes de c? Talvez, mas não podemos provar a partir das informações fornecidas.

Exemplo: $(\exists X)\,(\exists Y)\,m(X, Y)$ é consequência lógica de $\mathscr{P}$?

Neste caso estamos perguntando se há algum par de valores que satisfaz o predicado m.

$$\neg(\mathscr{P} \rightarrow (\exists X)\,(\exists Y)\,m(X, Y))$$

$$P_1 = (\forall X)\,(\forall Y)\,(s(X, Y) \rightarrow m(X, Y))$$

$$P_2 = (\forall X)\,(\forall Y)\,(\forall Z)\,((s(Z, Y) \wedge m(X, Z)) \rightarrow m(X, Y))$$

$$P_3 = s(a, b)$$

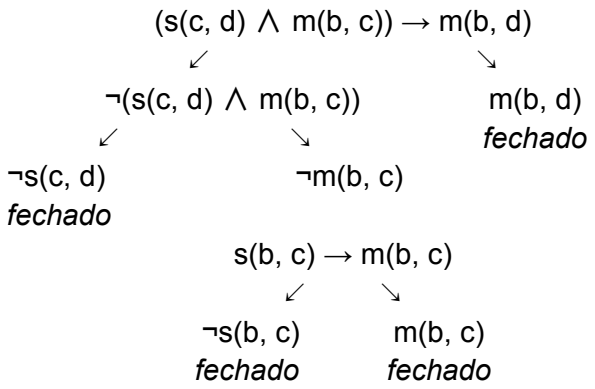
$$P_4 = s(b, c)$$

$$P_5 = s(c, d)$$

$$\neg(\exists X)\,(\exists Y)\,m(X, Y)$$

$$(\forall X)\,(\forall Y)\,\neg m(X, Y)$$

$$\neg m(b, d)$$



Neste caso fizemos a substituição $\{X \leftarrow b, Y \leftarrow d\}$ e chegamos a um tableau fechado, de modo que concluímos que o par (b, d) satisfaz o predicado m. Existem vários outros pares que satisfazem esse predicado, tais como (a, b), (a, c), (a, d), (b, c) e (c, d), todos levam a um tableau fechado utilizando as premissas fornecidas.

Sistema de Resolução

Assim como fizemos para os demais sistemas, vamos agora fazer uma extensão do sistema de resolução para a lógica de predicados.

Essencialmente, o sistema é análogo ao tableau no sentido de que busca contradições para determinar a insatisfatibilidade de fórmulas lógicas. Como a forma clausal representa uma conjunção de disjunções, podemos interpretar a expansão por resolução como uma espécie de tableau de ramo único. Ao contrário do tableau, a regra utilizada na resolução não gera uma fórmula equivalente àquela de um passo anterior, e sim uma

fórmula que é consequência lógica da fórmula anterior, de tal modo que a conjunção de todas as cláusulas que ocorrem na expansão é equivalente à fórmula original. Ou seja, se $X \Rightarrow Y$, então $(X \wedge Y) \Leftrightarrow X$. Temos muitas formas de entender essa afirmação, uma delas é que se as informações Y podem ser deduzidas a partir de X, acrescentar a informação Y ao conjunto de informações X não vai mudar nada do que X já dizia.

Já vimos que o sistema de resolução é definido para fórmulas em um formato específico, chamado de forma clausal ou notação de conjuntos. Nosso primeiro passo, portanto, é obter a forma clausal para a lógica de predicados.

Definição (Forma Clausal na Lógica de Predicados):

Sejam $L_1, \ldots, L_N$ literais da lógica de predicados. A fórmula na notação de conjuntos

$$\{\{L_1, \ldots, L_k\}, \{L_{k+1}, \ldots, L_i\}, \ldots, \{L_{i+1}, \ldots, L_N\}\}$$

equivale à fórmula

$$\forall^*((L_1 \vee \ldots \vee L_k) \wedge (L_{k+1} \vee \ldots \vee L_i) \wedge \ldots \wedge (L_{i+1} \vee \ldots \vee L_N))$$

onde $\forall^*$ indica que todas as variáveis que ocorrem na fórmula estão quantificadas universalmente.

A menos do quantificador universal sobre toda a fórmula, a definição da forma clausal é a mesma da lógica proposicional. Os conjunto internos são chamados de cláusulas, e representam disjunções de literais, enquanto o conjunto externo representa a conjunção destas cláusulas.

Nem toda fórmula da lógica de predicados está nesse formato, às vezes os quantificadores ocorrem dentro da fórmula, e não à esquerda. Além disso, há também o quantificador existencial, que existe na lógica de predicados mas não ocorre na forma clausal. Precisamos então de procedimentos para obter uma fórmula no formato adequado a partir de um fórmula qualquer da lógica de predicados, que é o que faremos nas seções que seguem.

Forma Prenex

Dizemos que uma fórmula da lógica de predicados está na forma Prenex quando todos os quantificadores ocorrem no início da fórmula, ou seja, é uma fórmula H da forma:

$$H = (Q_1X_1)\ (Q_2X_2)\ \ldots\ (Q_NX_N)\ F$$

onde Q_i são quantificadores, X_i são variáveis e F é uma fórmula sem quantificadores. Além disso, se F está na FNC dizemos que H está na forma Prenex-FNC.

Das seções anteriores, já conhecemos algumas equivalências que podem ser utilizadas para chegar à forma Prenex. Considere o seguinte procedimento:

Dada uma fórmula A da lógica de predicados, faça:

1. Utilize a equivalência $(A \leftrightarrow B) \Leftrightarrow (A \rightarrow B) \wedge (B \rightarrow A)$ para reescrever todas as bi-implicações
2. Utilize a equivalência $(A \rightarrow B) \Leftrightarrow (\neg A \vee B)$ para reescrever todas as implicações.

Depois destes dois passos, a fórmula terá apenas conectivos $\wedge$, $\vee$ e $\neg$ e, possivelmente, quantificadores. Vamos agora internalizar as negações, para tal, utilize recursivamente as seguintes equivalências até que as negações ocorram apenas sobre átomos:

3. a. $\neg(A \wedge B) \Leftrightarrow (\neg A \vee \neg B)$
3. b. $\neg(A \vee B) \Leftrightarrow (\neg A \wedge \neg B)$
3. c. $\neg(\forall X)\, A \Leftrightarrow (\exists X)\, \neg A$
3. d. $\neg(\exists X)\, A \Leftrightarrow (\forall X)\, \neg A$

A menos das duas últimas regras, estes eram os passos que seguiríamos para a obtenção de uma fórmula na forma FNC. Após todos estes passos temos uma fórmula que utiliza apenas os conectivos $\wedge$, $\vee$ e $\neg$ e, além disso, as negações ocorrem apenas sobre átomos.

Vamos ver agora como externar os quantificadores. Já conhecemos algumas equivalências que podem ser utilizadas para tal:

- Se X não ocorre em B, $((QX)\ A \square (QX)\ B) \Leftrightarrow (QX)\ (A \square B)$, onde Q é um quantificador e $\square$ é $\wedge$ ou $\vee$.
- Se X não ocorre em B e Y não ocorre em A, $((Q_1X)\ A \square (Q_2Y)\ B) \Leftrightarrow (Q_1X)\ (Q_2Y)\ (A \square B)$, onde Q_i são quantificadores e $\square$ é $\wedge$ ou $\vee$. A ordem dos quantificadores pode ser invertida sem qualquer prejuízo para o significado da fórmula.

Nestes casos, estamos estendendo o escopo dos quantificadores aproveitando-nos do fato de que a variável não ocorre na outra fórmula e, portanto, não afetará o significado da mesma. Além disso, vimos que:

- $((\forall X)\, A \wedge (\forall X)\, B) \Leftrightarrow (\forall X)\, (A \wedge B)$
- $((\exists X)\, A \vee (\exists X)\, B) \Leftrightarrow (\exists X)\, (A \vee B)$

Mas o que fazer quando vemos fórmulas como $(\exists X)\, p(X) \wedge (\exists X)\, q(X)$? Já vimos no exemplo 6 da seção anterior que esta fórmula não é equivalente a $(\exists X)\, (p(X) \wedge q(X))$, ou seja, não podemos simplesmente estender o escopo do quantificador sobre as duas fórmulas porque isto altera o significado da fórmula, como já discutido antes.

Vamos nos utilizar de um conceito fundamental da lógica de predicados, o escopo dos quantificadores. Já discutimos que, para todos os efeitos, a fórmula $(\exists X)\, p(X) \wedge (\exists X)\, q(X)$ é equivalente a $(\exists Y)\, p(Y) \wedge (\exists Z)\, q(Z)$ porque, apesar do mesmo nome de variável ter sido utilizado na primeira fórmula, como esta variável ocorre no escopo de quantificadores distintos, ela pode muito bem ter qualquer outro nome. Em ambos os casos a fórmula diz que “Há um valor do domínio que satisfaz o predicado p e há um valor do domínio que satisfaz o predicado q” sem dizer, em momento algum, que é o mesmo valor que satisfaz os dois predicados.

Desta discussão, concluímos que o nome utilizado para variáveis em escopos distintos pode ser alterado. Precisamos, no entanto, tomar cuidado para não causar novos problemas ao fazer essa alteração. Imagine a fórmula $(\exists X)\, p(X, Y) \wedge (\exists X)\, q(X, Y)$, neste caso não podemos dizer que ela é equivalente a $(\exists Y)\, p(X, Y) \wedge (\exists Z)\, q(X, Y)$ porque a variável Y ocorria livre em p e agora passou a ser quantificada. Para evitar tais problemas, as substituições de variáveis devem ser seguras, ou seja, não devem alterar o significado da fórmula. Do exposto, vamos definir a seguinte regra de renomeação de variáveis:

R_0: Dada uma fórmula H com variáveis $X_1, \ldots, X_N$ quantificadas e $Y_1, \ldots, Y_M$ livres. Substitua cada uma das ocorrências das variáveis X_i por variáveis Z_i tais que $Z_i \neq Z_j$ se $i \neq j$ e Z_i de todos os X_i e Y_i.

Após a aplicação de R_0 não há mais problemas na extensão do escopo de qualquer quantificador e podemos simplesmente escrever todos os quantificadores à esquerda da fórmula, na ordem em que eles ocorrem na fórmula original. Após o passo 6, no procedimento que estávamos descrevendo, podemos continuar:

4. Aplique a regra de renomeação de variáveis, R_0.
5. Estenda o escopo dos quantificadores sobre toda a fórmula.

Neste ponto já temos uma fórmula que está na forma Prenex e é equivalente à fórmula inicial, $A \Leftrightarrow (Q_1X_1)\ (Q_2X_2)\ \ldots\ (Q_NX_N)\ F$. Agora, para obter a forma Prenex-FNC basta fazer:

6. Aplique recursivamente a distributividade sobre F até que F esteja na FNC.

Ex.: Seja $A = (\forall X)\ p(X) \wedge ((\forall X)\ q(X) \rightarrow (\exists Y)\ r(X, Y, Z))$. Obtenha uma fórmula equivalente a A que está na forma Prenex-FNC.

Vamos aplicar o procedimento descrito nesta seção:

1. Não há bi-implicações, este passo não se aplica.
2. Reescrita da implicação:
   $A \Leftrightarrow (\forall X)\ p(X) \wedge (\neg(\forall X)\ q(X) \vee (\exists Y)\ r(X, Y, Z))$
3. Internalização das negações:
   $A \Leftrightarrow (\forall X)\ p(X) \wedge ((\exists X)\ \neg q(X) \vee (\exists Y)\ r(X, Y, Z))$
4. Renomeação de variáveis:
   $A \Leftrightarrow (\forall S)\ p(S) \wedge ((\exists T)\ \neg q(T) \vee (\exists U)\ r(X, U, Z))$
5. Externalização dos quantificadores:
   $A \Leftrightarrow (\forall S)\ (\exists T)\ (\exists U)\ (p(S) \wedge (\neg q(T) \vee r(X, U, Z)))$
6. Não se aplica, pois a fórmula após os quantificadores já está na FNC.

É importante notar que a fórmula resultante é equivalente à fórmula original e que ela não é única. Já sabemos que ordem dos fatores em uma conjunção ou disjunção não afeta o resultado, portanto, a partir do passo 4 poderíamos escrever:

$$A \Leftrightarrow (\forall S)\ p(S) \wedge ((\exists T)\ \neg q(T) \vee (\exists U)\ r(X, U, Z)) \Leftrightarrow ((\exists T)\ \neg q(T) \vee (\exists U)\ r(X, U, Z)) \wedge (\forall S)\ p(S)$$

E daí teríamos que:

$$A \Leftrightarrow (\exists T)(\exists U)(\forall S)\ ((\neg q(T) \vee r(X, U, Z)) \wedge p(S)) \Leftrightarrow (\exists T)(\exists U)(\forall S)\ (p(S) \wedge (\neg q(T) \vee r(X, U, Z))),$$

que é essencialmente a mesma fórmula de antes com a ordem de alguns quantificadores trocadas. No entanto, nem toda permutação da ordem dos quantificadores é equivalente. Neste exemplo, particularmente, não há qualquer manipulação da fórmula que faça com que o quantificador universal ocorra entre os quantificadores existenciais.

Uma parte do nosso problema está sanado. Já temos uma fórmula com quantificadores à esquerda e que a parte sem quantificadores está na FNC. No entanto, do modo que a forma clausal foi definida, todas as variáveis devem ser quantificadas

universalmente. Então, não podemos ter variáveis livres ou quantificadores existenciais. Trataremos destes problemas na próxima seção.

Skolemização

Qual a interpretação da fórmula p(X)? Já sabemos que isso depende da interpretação, de qual o significado do predicado p e qual o valor da variável X, que neste caso está livre e deve ser determinada pela interpretação.

No entanto, se p(X) é satisfatível, (∃X) p(X) também é. Pois o primeiro implica no segundo. Deste modo, concluímos também que se (∃X) p(X) é insatisfatível, p(X) também é.

Estamos preparando a fórmula para ser utilizada com o método da resolução, que serve para demonstrar se uma fórmula é satisfatível ou não.

Assim, se desejamos saber se uma fórmula é satisfatível ou não, podemos aplicar o quantificador existencial a todas as variáveis livres da fórmula sem prejuízo para o resultado obtido. Então, para resolver o problema das variáveis livres, basta aplicar o fecho existencial à fórmula e o resultado da aplicação do método da resolução (ou do tableau semântico) estará correto.

No entanto, é bom lembrar que p(X) não é equivalente a (∃X) p(X), a relação que temos é que p(X) ⇒ (∃X) p(X) e daí, ¬(∃X) p(X) ⇒ ¬p(X). Apenas a propriedade de satisfatibilidade é idêntica para as duas fórmulas.

Agora, só falta resolvermos o problema dos quantificadores existenciais. Para resolver essa questão vamos utilizar uma estratégia similar à regra de eliminação de quantificadores existenciais (R_{13}) adotada nos tableaux semânticos.

Nos tableaus, fizemos a eliminação do quantificador existencial através da introdução de uma constante nova. Isso sempre era possível porque no tableau, ao aplicar a regra de eliminação, o quantificador existencial ocorre no início da fórmula. Já foi discutido que a ordem dos quantificadores afeta o significado da fórmula, (∀X) (∃Y) p(X, Y), por exemplo, não é equivalente a (∃Y) (∀X) p(X, Y).

Assim como no caso da introdução de quantificadores existenciais para tratar das variáveis livres, a fórmula obtida através das regras de eliminação de quantificadores não são equivalentes às fórmulas originais mas são equisatisfatíveis, ou seja, a fórmula obtida é consequência lógica da primeira.

No problema atual devemos eliminar todos os quantificadores existenciais e eles nem sempre vão ocorrer no início da fórmula. Vamos tratar alguns casos específicos para entender o procedimento que será descrito em breve.

Ex. 1: H_1 = (∃X) p(X)

Utilizando as regras do tableau semântico já sabemos que H_1 ⇒ p(a), onde a é uma constante. Se p(a) é insatisfatível, H_1 também é.

Ex. 2: H_2 = (∃X) (∃Y) p(X, Y)

Como no caso anterior, sabemos que H_2 ⇒ p(a, b), onde a e b são constantes.

Ex. 3: H_3 = (∃X) (∀Y) p(X, Y)

Aplicando mais uma vez a regra do tableau, temos que $H_3 \Rightarrow (\forall Y)\ p(a, Y)$, onde a é uma constante.

Ex. 4: $H_4 = (\forall Y)\ (\exists X)\ p(X, Y)$

Neste caso não podemos aplicar a regra de eliminação do tableau pois há um quantificador universal antes do existencial. Note que $(\forall Y)\ p(a, Y)$ não é consequência lógica de H_4. H_4 diz que para toda escolha do segundo argumento há uma escolha possível para o primeiro argumento tal que p é satisfeito. $(\forall Y)\ p(a, Y)$ diz que há uma certa constante que se colocada no primeiro argumento, qualquer valor do segundo argumento satisfaz p. Estas duas afirmações são distintas, porque H_4 não diz que é sempre o mesmo valor de X que deve ser escolhido, mas existe um valor de X para cada escolha de Y, ou seja, a escolha de X depende da escolha que foi feita para Y. Deste modo, o que a fórmula nos diz é que há uma função que escolhe um valor para o primeiro argumento a partir da escolha feita para o segundo, em termos de fórmulas temos $H_4 \Rightarrow (\forall Y)\ p(f(Y), Y)$.

Ex. 5: $H_5 = (\forall X)\ (\forall Y)\ (\exists Z)\ p(X, Y, Z)$

Mais uma vez, a escolha de Z depende da escolha que foi feita anteriormente para X e Y, ou seja, a fórmula nos diz que há uma função de X e Y que determina o valor do terceiro argumento: $H_5 \Rightarrow (\forall X)\ (\forall Y)\ p(X, Y, f(X, Y))$.

Ex. 6: $H_6 = (\forall X)\ (\exists Z)\ (\forall Y)\ p(X, Y, Z)$

Neste caso, apesar de termos os mesmos quantificadores do exemplo anterior, note que a escolha de Z depende apenas da escolha de X, portanto podemos escrever $H_6 \Rightarrow (\forall X)\ (\forall Y)\ p(X, Y, f(X))$.

Ex. 7: $H_7 = (\forall X)\ (\exists Y)\ (\exists Z)\ p(X, Y, Z)$

Tanto a escolha de Y quanto a escolha de Z dependem da escolha que foi feita para X. Seguindo o raciocínio anterior temos que $H_7 \Rightarrow (\forall X)\ p(X, f(X), g(X))$. É importante notar que devemos utilizar funções distintas para substituir Y e Z. Assim como no caso da substituição das constantes, onde não podemos garantir que a mesma constante vai funcionar nas duas posições, não podemos garantir que a mesma função pode ser usada para escolher os valores de Y e Z. Como sabemos apenas que tais funções existem, vamos adotar nomes distintos para as funções.

Este procedimento de eliminação dos quantificadores existenciais é chamado de Skolemização (em homenagem ao matemático norueguês Thoralf Skolem). O procedimento, dada uma fórmula na forma Prenex, pode ser descrito como segue:

1. Identifique o quantificador existencial mais interno da fórmula. Vamos supor que seja $(\exists X)$.
2. Identifique todas as variáveis cujos quantificadores universais ocorrem à esquerda de $(\exists X)$. Suponhamos que sejam $\{X_1, X_2, \ldots, X_S\}$.
3. Elimine $(\exists X)$.
4. Substitua toda ocorrência de X por $f(X_1, X_2, \ldots, X_S\}$, onde f é um novo símbolo de função.

5. Se ainda há quantificador existencial na fórmula, volte para o passo 1.

A fórmula obtida após esse processo não é equivalente à fórmula original, mas é consequência lógica dela. De tudo que já discutimos até agora, deve estar claro que se a fórmula Skolemizada for insatisfatível, a fórmula original também é.

Ex: Faça a Skolemização de

$$H = (\exists S)(\forall T)(\forall U)(\exists V)(\forall W)(\exists X)(\forall Y)(p(S, V, X) \land (q(T, S) \lor r(U, W, Y)))$$

O quantificador existencial mais interno é $(\exists X)$, e antes dele temos T, U e W quantificadas universalmente. Daí vamos substituir X por uma função destas variáveis:

$$H \Rightarrow (\exists S)(\forall T)(\forall U)(\exists V)(\forall W)(\forall Y)(p(S, V, f(S, T, U)) \land (q(T, S) \lor r(U, W, Y)))$$

Agora, o quantificador existencial mais interno é $(\exists V)$, antes dele temos T e U quantificadas universalmente. Vamos substituir V por uma nova função de T e U:

$$H \Rightarrow (\exists S)(\forall T)(\forall U)(\forall W)(\forall Y)(p(S, g(T, U), f(T, U, W)) \land (q(T, S) \lor r(U, W, Y)))$$

Por fim, o único quantificador existencial que sobrou é $(\exists S)$. Antes de $(\exists S)$ não há qualquer variável quantificada universalmente, portanto S será substituída por uma função de nenhum argumento, ou seja, uma constante.

$$H \Rightarrow (\forall T)(\forall U)(\forall W)(\forall Y)(p(a, g(T, U), f(T, U, W)) \land (q(T, a) \lor r(U, W, Y)))$$

ou simplesmente:

$$H \Rightarrow \forall^* (p(a, g(T, U), f(T, U, W)) \land (q(T, a) \lor r(U, W, Y)))$$

ou ainda, na notação de conjuntos

$$H \Rightarrow \{\{p(a, g(T, U), f(T, U, W))\}, \{q(T, a), r(U, W, Y)\}\}$$

Unificação

Neste momento já temos as fórmulas na forma clausal, como o método da resolução exige, então basta aplicar a regra da resolução que já conhecemos, não é? Não exatamente.

A regra da resolução para a lógica proposicional identifica um par de literais complementares de duas cláusulas e os remove, ficando com uma cláusula composta pelos demais literais, $(A \lor X) \land (B \lor \neg X) \Rightarrow (A \lor B)$.

No entanto, considere as cláusulas $\{p(X)\}$ e $\{\neg p(Y)\}$. Do ponto de vista sintático, não há literais complementares entre elas, pois p(X) não é o complemento de ¬p(Y), elas diferem, entre si, em mais do que a negação. Apesar de não serem complementares, vamos considerar o significado destas cláusulas. $\{p(X)\} \Leftrightarrow (\forall X)\, p(X)$ e $\{\neg p(Y)\} \Leftrightarrow (\forall Y)\, \neg p(Y)$, a primeira fórmula diz que o predicado p é verdade para todos os valores do domínio, enquanto a segunda diz que o predicado p é falso para todos os valores do domínio. Estas duas afirmações são contraditórias entre si e, obviamente, não podem ser satisfeitas

simultaneamente. Do mesmo modo, {p(X)} e {¬p(a)} apresentam informações contraditórias, pois a segunda diz que há um certo valor do domínio, chamado de a, que não satisfaz p.

No primeiro caso, se fizéssemos a substituição {X ← Y} as duas cláusulas seriam complementares, de forma similar, com a substituição {X ← a} nas cláusulas do segundo exemplo também teríamos cláusulas complementares.

Estes exemplos serviram para provocar a ideia de que, mais do que procurar elementos complementares, o método da resolução busca contradições. No caso da lógica proposicional, contradição é sinônimo de complementaridade sintática. Para a lógica de predicados, a contradição pode surgir de outras formas por causa da presença das variáveis. Portanto, vamos precisar de um passo extra na resolução, que é verificar se os literais são contraditórios.

A verificação desta contradição semântica pode ser feita através de um artifício sintático, que é chamado de unificação. A unificação de um conjunto de expressões da lógica de predicados consiste na identificação de uma substituição que, se aplicada às fórmulas do conjunto, as tornam idênticas, ou seja, o conjunto torna-se unitário.

Uma substituição é um conjunto $\Theta = \{X_1 \leftarrow t_1, X_2 \leftarrow t_2, \ldots, X_N \leftarrow t_N\}$ onde os X_i são variáveis e os t_i são termos (variáveis ou funções) tais que $t_i \neq X_i$ e, se $i \neq j$ então $X_i \neq X_j$, ou seja, cada variável só é substituída uma vez e não é permitido a substituição de uma variável por ela mesma. A substituição vazia, $\Theta = \{\ \}$, não afeta a fórmula.

Se S é um conjunto de expressões e Θ é uma substituição, a aplicação de Θ em S, denotada por SΘ, gera um novo conjunto de expressões onde cada ocorrência de X_i em S é trocada t_i. É importante observar que as substituições definidas por Θ são feitas na expressão original, e não de forma sequencial.

Ex: Seja $\Theta = \{Y \leftarrow X, X \leftarrow W, W \leftarrow a\}$ e S = p(X, Y, W).
SΘ = p(W, X, a), onde a ocorrência de X em S virou W, a ocorrência de Y virou X e a ocorrência de W virou a.
Se a aplicação das substituições definidas por Θ fosse sequencial obteríamos um resultado diferente.
p(X, Y, W) {Y ← X}
= p(X, X, W) {X ← W}
= p(W, W, W) {W ← a}
= p(a, a, a)

Aproveitando este tópico da aplicação sequencial de substituições,será interessante para nós a definição de uma composição de substituições tal que $S(\Theta_{12}) = (S\Theta_1)\Theta_2$, ou seja, Θ_{12} é uma única substituição que equivale à aplicação sequencial das substituições Θ_1 e Θ_2.

Vamos ver alguns exemplos de substituições sequenciais para observar os possíveis problemas que podem ocorrer.

Ex. 1: $\Theta_1 = \{Y \leftarrow W\}$, $\Theta_2 = \{W \leftarrow a\}$ e S = p(Y, W).

$(S\Theta_1)\Theta_2 = p(W,W)\Theta_2 = p(a, a)$.

Se definíssemos Θ como a união de Θ_1 e Θ_2 o resultado não seria equivalente, pois $\Theta = \Theta_1 \cup \Theta_2 = \{Y \leftarrow W, W \leftarrow a\}$ e SΘ = p(W, a), que é diferente do que obtivemos pela aplicação sequencial.

Isso ocorre porque ao aplicar a segunda substituição, ela transforma o resultado obtido pela primeira. Então, para obter um resultado igual deveríamos fazer a união de Θ_2 com a aplicação de Θ_2 sobre os termos de Θ_1, ou seja $\Theta_{12} = \{Y \leftarrow W\Theta_2, W \leftarrow a\} = \{Y \leftarrow a, W \leftarrow a\}$. Agora sim temos $S\Theta_{12} = (S\Theta_1)\Theta_2$.

Ex. 2: $\Theta_3 = \{X \leftarrow Y\}$ e $\Theta_4 = \{Y \leftarrow X\}$.

$(p(X, Y)\Theta_3)\Theta_4 = p(Y, Y)\Theta_4 = p(X, X)$

Usando a mesma técnica do exemplo anterior, definiremos $\Theta_{34} = \{X \leftarrow Y\Theta_4, Y \leftarrow X\} = \{X \leftarrow X, Y \leftarrow X\}$. No entanto, a substituição $\{X \leftarrow X\}$ não tem qualquer efeito sobre a fórmula e, conforme a definição de substituição dada acima, não deve aparecer no conjunto. Portanto, a substituição obtida é $\Theta_{34} = \{Y \leftarrow X\}$.

Ex. 3: $\Theta_5 = \{X \leftarrow a\}$ e $\Theta_6 = \{X \leftarrow b\}$.

$(p(X)\Theta_5)\Theta_6 = p(a)\Theta_6 = p(a)$.

Neste caso, como as duas substituições são sobre a variável X, a segunda acaba não ocorrendo pois a primeira já eliminou o X. Não podemos definir Θ_{56} como $\{X \leftarrow a, X \leftarrow b\}$ pois não é permitido ter duas substituições da mesma variável no mesmo conjunto. Neste caso devemos eliminar as substituições de X que vieram da segunda substituição ficando apenas com $\Theta_{56} = \{X \leftarrow a\}$.

Podemos agora definir um procedimento para obtenção da composição de substituições. Dadas as substituições $\Theta_1 = \{X_1 \leftarrow t_1, \ldots, X_N \leftarrow t_N\}$ e $\Theta_2 = \{Y_1 \leftarrow s_1, \ldots, Y_M \leftarrow s_M\}$, a composição Θ_{12} tal que $S\Theta_{12} = (S\Theta_1)\Theta_2$ é obtida da seguinte forma:

1. $\Theta_{12} = \{X_1 \leftarrow t_1\Theta_2, \ldots, X_N \leftarrow t_N\Theta_2, Y_1 \leftarrow s_1, \ldots, Y_M \leftarrow s_M\}$.
2. Retire de Θ_{12} as substituições $Y_i \leftarrow s_i$ tais que Y_i é igual a algum X_j, $1 \leq j \leq N$.
3. Retire de Θ_{12} as substituições $X_i \leftarrow t_i\Theta_2$ onde $X_i = t_i\Theta_2$.

Tendo definido a substituição, vamos considerar um exemplo mais complexo.

Ex.: Sejam $S_1 = p(f(X), Y, X)$ e $S_2 = p(Z, g(Z), a)$. Existe uma substituição que unifica as expressões S_1 e S_2? Que substituição é essa?
Não vamos discutir ainda como essa substituição foi obtida, mas perceba que $\Theta = \{X \leftarrow a, Z \leftarrow f(a), Y \leftarrow g(f(a))\}$ é uma resposta para a pergunta. Pois $S_1\Theta = p(f(a), g(f(a), a) = S_2\Theta$.

Ex.: Sejam $S_3 = p(X, Y)$ e $S_4 = p(W, a)$. Existe uma substituição que unifica as expressões S_3 e S_4? Qual?
Neste caso é mais fácil achar a substituição. $\Theta_1 = \{X \leftarrow W, Y \leftarrow a\}$ é uma substituição que responde às perguntas. No entanto, $\Theta_2 = \{X \leftarrow a, W \leftarrow a, Y \leftarrow a\}$ também seria uma substituição válida. Existe alguma preferência entre essas duas substituições?

A substituição Θ que unifica um conjunto de expressões é chamada de unificador do conjunto, nos exemplos anteriores encontramos unificadores das expressões dadas. Quando há mais de um unificador, digamos Θ_1 e Θ_2, para o mesmo conjunto, dizemos que Θ_1 é mais geral que Θ_2 se $\Theta_2 = \Theta_1\Theta_3$, onde Θ_3 é uma substituição.O unificador, Θ, mais

geral de um conjunto é aquele que, dado qualquer unificador □, podemos escrever □ = Θφ, onde φ é uma substituição.

Deste modo, no exemplo anterior, Θ_1 é mais geral que Θ_2, e é preferível, pois impõe menos restrições. Vamos definir um procedimento para achar o unificador mais geral de um conjunto de expressões.

Algoritmo da unificação: Seja S um conjunto de expressões da lógica de predicados.

1. Faça k = 0 e $\Theta_k = \{\}$.
2. Se $S\Theta_k$ está unificado, pare, Θ_k é o unificador mais geral de S. Caso contrário, varra simultaneamente as expressões de $S\Theta_k$ da esquerda para a direita e determine o conjunto D_k formado pela primeira subexpressão distinta entre as expressões de $S\Theta_k$.
3. Se existe em D_k uma variável X e um termo t, tal que X não ocorre em t, faça $\Theta_{k+1} = \Theta_k\{X \leftarrow t\}$, $k \leftarrow k + 1$ e volte para o passo 2. Caso contrário, pare, S não é unificável.

Vamos aplicar o algoritmo em algumas expressões para entender melhor seu funcionamento.

Ex.: S = {p(a), p(b)}.

1. k = 0, $\Theta_0 = \{\}$.
2. $S\Theta_0 = S = \{p(a), p(b)\}$. Olhando simultaneamente as duas expressões vemos que ambas começam com p, seguidas da abertura do parêntese e depois temos uma diferença a na primeira expressão e b na segunda, portanto $D_0 = \{a, b\}$.
3. Não há em D_0 uma variável, portanto o conjunto S não é unificável.

Ex.: S = {p(X, Y), p(W, a)}.

1. k = 0, $\Theta_0 = \{\}$.
2. $S\Theta_0 = S = \{p(X, Y), p(W, a)\}$. Olhando simultaneamente as duas expressões vemos que ambas começam com p, seguidas da abertura do parêntese e depois temos uma diferença X na primeira expressão e W na segunda, portanto $D_0 = \{X, W\}$.
3. Há em D_0 uma variável (X) e um termo (W) no qual esta variável não ocorre.
4. $\Theta_1 = \Theta_0\{X \leftarrow W\} = \{X \leftarrow W\}$.
5. k = 1.
6. $S\Theta_1 = \{p(W, Y), p(W, a)\}$. A diferença entre as duas expressões é $D_1 = \{Y, a\}$.
7. Há em D_1 uma variável (Y) e um termo (a) no qual esta variável não ocorre.
8. $\Theta_2 = \Theta_1\{Y \leftarrow a\} = \{X \leftarrow W, Y \leftarrow a\}$.
9. k = 2.
10. $S\Theta_2 = \{p(W, a)\}$. Como S está unificado, Θ_2 é um unificador mais geral de S.

Ex.: S = {p(X), p(f(X))}.

1. k = 0, $\Theta_0 = \{\}$.
2. $S\Theta_0 = S = \{p(X), p(f(X))\}$. Olhando simultaneamente as duas expressões vemos que ambas começam com p, seguidas da abertura do parêntese e depois temos uma diferença X na primeira expressão e f na segunda, como devemos pegar a sub-expressão que inicia neste ponto, temos $D_0 = \{X, f(X)\}$. f sozinho não é uma

subexpressão válida, pois é uma função sem os seus argumentos, daí f(X) entra no conjunto D_0.
3. Há em D_0 uma variável (X) e um termo (f(X)). No entanto, X ocorre em f(X) e, portanto, este conjunto não é unificável.

Vamos supor, por um instante, que este teste de ocorrência não fosse realizado. Vamos seguir fazendo a substituição {X ← f(X)} em um desenvolvimento hipotético.

a. $\Theta_1 = \Theta_0\{X \leftarrow f(X)\} = \{X \leftarrow f(X)\}$.
b. k = 1.
c. $S\Theta_1 = \{p(f(X)), p(f(f(X)))\}$. Até o primeiro f as duas expressões são idênticas, depois temos X no argumento de f na primeira expressão e f(X) no argumento de f na segunda, daí $D_1 = \{X, f(X)\}$.
d. $\Theta_2 = \Theta_1\{X \leftarrow f(X)\} = \{X \leftarrow f(f(X))\}$.
e. k = 2.
f. $S\Theta_2 = \{p(f(f(X))), p(f(f(f(X))))\}$. Até o segundo f as duas expressões são idênticas, daí teremos $D_2 = \{X, f(X)\}$.
g. …

Neste ponto você já deve ter percebido que este procedimento não vai acabar nunca, pois o conjunto de diferenças nunca é alterado. Daí a importância da verificação de ocorrência da variável no termo que a substitui.

Ex.: S = {p(f(X), Y, X), p(Z, g(Z), a)}

1. k = 0, $\Theta_0 = \{\ \}$.
2. $S\Theta_0 = S = \{p(f(X), Y, X), p(Z, g(Z), a)\}$. Temos $D_0 = \{f(X), Z\}$.
3. Há em D_0 uma variável (X) e um termo (f(X)) no qual esta variável não ocorre.
4. $\Theta_1 = \Theta_0\{Z \leftarrow f(X)\} = \{Z \leftarrow f(X)\}$.
5. k = 1.
6. $S\Theta_1 = \{p(f(X), Y, X), p(f(X), g(f(X)), a)\}$. Temos $D_1 = \{Y, g(f(X))\}$.
7. Há em D_1 uma variável (Y) e um termo (g(f(X))) no qual esta variável não ocorre.
8. $\Theta_2 = \Theta_1\{Y \leftarrow g(f(X))\} = \{Z \leftarrow f(X), Y \leftarrow g(f(X))\}$.
9. k = 2.
10. $S\Theta_2 = \{p(f(X), g(f(X)), X), p(f(X), g(f(X)), a)\}$. Temos $D_2 = \{X, a\}$.
11. Há em D_2 uma variável (X) e um termo (a) no qual esta variável não ocorre.
12. $\Theta_3 = \Theta_2\{X \leftarrow a\} = \{Z \leftarrow f(a), Y \leftarrow g(f(a)), X \leftarrow a\}$.
13. k = 3.
14. $S\Theta_3 = \{p(f(a), g(f(a)), a)\}$. Como S está unificado, Θ_3 é um unificador mais geral de S.

Agora que temos o método de unificação definido, podemos finalmente enunciar a regra da resolução para a lógica de predicados. O algoritmo da unificação nos dá um instrumento para tornar diversas expressões sintaticamente idênticas. É importante salientar que a expressão unificada não é necessariamente equivalente às expressões originais, mas é uma consequência lógica comum a todas elas. Isto vem do fato que a variável é o termo mais genérico, ao substituir a variável por outros termos obtemos expressões com mais restrições se alguma das substituições troca uma variável por uma função. A expressão unificada é uma espécie de interseção do que está sendo dito pelas expressões originais, é

um caso que faz parte de todas as afirmações. {p(X, Y)} diz que p é verdade para todo X e Y enquanto {p(f(Y), a)} diz que p é verdade para todos os valores retornados por uma certa função f e a constante a. A expressão unificada, p(f(a), a) diz que p é verdade se a está no segundo argumento e se o primeiro argumento é o retorno da função f com a como entrada. Este caso em particular é comum às duas afirmações {p(X, Y)} e {p(f(Y), a)}.

Regra da Resolução

Vamos utilizar o artifício da unificação para poder enunciar a regra da resolução de forma mais genérica. Se C_1 e C_2 são cláusulas tais que L_1 ocorre em C_1 e $\neg L_2$ ocorre em C_2, e L_1 e L_2 são unificáveis pela substituição Θ, obtida pelo algoritmo de unificação, dizemos que o resolvente de C_1 e C_2 é dado por $R(C_1, C_2) = (C_1\Theta \cup C_2\Theta) \setminus \{L_1\Theta, \neg L_2\Theta\}$.

Deste modo, especificamos a situação na qual há uma contradição entre L_1 e $\neg L_2$ através da substituição Θ e seguimos com a conclusão de que neste caso, após a substituição, a união de C_1 e C_2 deve ser verdade, como era concluído na lógica proposicional.

Ou seja, a extensão usa a unificação para identificar a contradição e pode muito bem ser aplicada à lógica proposicional pois lá não há a ocorrência de variáveis e toda substituição será vazia.

Expansão e Prova por Resolução

A expansão por resolução se dá de forma análoga ao que fazíamos na lógica proposicional, primeiro escrevemos a fórmula na forma clausal, listamos as cláusulas e em seguida aplicamos repetidamente a regra da resolução até que não seja possível criar mais cláusulas ou que a cláusula vazia seja atingida. Quando a cláusula vazia é atingida, dizemos que a resolução é fechada e concluímos que a fórmula é insatisfatível.

Assim como antes, para obter uma prova por resolução basta negar a fórmula antes de iniciar o processo de resolução. Daí, se ¬H é insatisfatível, concluímos que H é tautologia.

Ex.: H = {{p(f(X), Y, X), ¬q(a, Y, S)}, {¬p(Z, g(Z), W)}, {q(X, g(f(a)), c)}} é satisfatível?

Expansão:

1. {p(f(X), Y, X), ¬q(a, Y, S)}
2. {¬p(Z, g(Z), W)}
3. {q(X, g(f(a)), c)}
4. {¬q(a, g(f(W), S)} R(1, 2), Θ = {X ← W, Z ← f(W), Y ← g(f(W))}
5. { } R(3, 4), Θ = {X ← a, W ← a, S ← c}

Como a cláusula vazia foi atingida, concluímos que H é insatisfatível. Para obter a fórmula da linha 4 fizemos a resolução das cláusulas nas linhas 1 e 2, através da unificação de p(f(X), Y, X) e ¬p(Z, g(Z), W) com a substituição Θ = {X ← W, Z ← f(W), Y ← g(f(W))}. Perceba que a substituição foi aplicada também sobre ¬q(a, Y, S), que resta da primeira cláusula após a eliminação dos literais complementares unificados. A cláusula vazia foi obtida da resolução das cláusulas das linhas 3 e 4 utilizando a substituição {X ← a, W ← a, S ← c}.

Nos tableaux para a lógica de predicados vimos que havia uma diferença fundamental em relação à lógica proposicional, pois podem haver tableaux abertos associados a fórmulas insatisfatíveis, esta possibilidade vinha da ordem em que as regras de eliminação dos quantificadores eram aplicadas. Na resolução, temos apenas uma regra, então a expansão por resolução é única (a menos da ordem das cláusulas) para uma mesma forma na forma clausal. Isto quer dizer que não temos que ter a mesma preocupação que tivemos nos tableaux? Não exatamente. Considere o exemplo que segue.

Ex.: G = $(\forall X)\ (p(X) \rightarrow r(X))$ é consequência lógica de H = $(\exists X)\ p(X) \rightarrow (\forall X)\ r(X)$? Para tal, $(H \rightarrow G)$ deve ser tautologia, ou, equivalentemente $\neg(H \rightarrow G)$ deve ser insatisfatível.

Vamos começar fazendo escrevendo $F = \neg(H \rightarrow G)$ na forma clausal:
$F = \neg(((\exists X)\ p(X) \rightarrow (\forall X)\ r(X)) \rightarrow (\forall X)\ (p(X) \rightarrow r(X)))$
$\Leftrightarrow ((\exists X)\ p(X) \rightarrow (\forall X)\ r(X)) \wedge \neg(\forall X)\ (p(X) \rightarrow r(X))$
$\Leftrightarrow ((\exists X)\ p(X) \vee (\forall X)\ r(X)) \wedge (\exists X)\ \neg(p(X) \rightarrow r(X))$
$\Leftrightarrow ((\forall X)\ \neg p(X) \vee (\forall X)\ r(X)) \wedge (\exists X)\ (p(X) \wedge \neg r(X))$
Fazendo a renomeação das variáveis:
$F \Leftrightarrow ((\forall X)\ \neg p(X) \vee (\forall Y)\ r(Y)) \wedge (\exists Z)\ (p(Z) \wedge \neg r(Z))$
Externalizando os quantificadores:
$F \Leftrightarrow (\forall X)\ (\forall Y)\ (\exists Z)\ ((\neg p(X) \vee r(Y)) \wedge p(Z) \wedge \neg r(Z))$
Após a skolemização:
{{¬p(X), r(Y)}, {p(f(X, Y))}, {¬r(f(X, Y))}}

Expansão por resolução:

1. {¬p(X), r(Y)}
2. {p(f(X, Y))}
3. {¬r(f(X, Y))}

Não é possível aplicar a resolução a nenhum par de cláusulas e temos uma resolução aberta.

Isto quer dizer que H não implica em G? Não necessariamente, apesar da expansão de fato ser única para uma dada na fórmula na forma clausal, existe mais de uma forma Prenex-FNC equivalente a F e isso pode gerar uma forma clausal distinta.
No passo de externalização dos quantificadores, poderíamos escrever:
$F \Leftrightarrow (\exists Z)\ (\forall X)\ (\forall Y)\ ((\neg p(X) \vee r(Y)) \wedge p(Z) \wedge \neg r(Z))$
Após a skolemização teremos:
{{¬p(X), r(Y)}, {p(a)}, {¬r(a)}}

Expansão por resolução:

1. {¬p(X), r(Y)}
2. {p(a)}
3. {¬r(a)}
4. {r(Y)} R(1, 2), $\Theta = \{X \leftarrow a\}$
5. { } R(3, 4), $\Theta = \{Y \leftarrow a\}$

E chegamos a uma expansão fechada. Assim como no caso dos tableaux, como há uma expansão por resolução fechada, concluímos que G é consequência lógica de H.

Portanto, atenção, o fato de haver mais de uma forma Prenex associada a uma mesma fórmula, só podemos afirmar que uma fórmula é satisfatível se a resolução de todas as suas formas prenex são abertas. Equivalentemente, dizemos que uma fórmula é insatisfatível se há pelo menos uma resolução fechada associada a ela.

Nos tableaux, vimos que é melhor fazer a remoção dos quantificadores existenciais antes dos universais. Aqui na resolução a mesma regra se aplica. Como podemos fazer isso? Basta escolher a forma prenex que tem os quantificadores existenciais mais à esquerda, deste modo a skolemização vai impor menos restrições e a unificação tem mais chances de ocorrer.

Ex.: H = $(\exists X)\, p(X) \rightarrow (\forall X)\, r(X)$ é consequência lógica de G = $(\forall X)\, (p(X) \rightarrow r(X))$? Para tal, $(G \rightarrow H)$ deve ser tautologia, ou, equivalentemente $\neg(G \rightarrow H)$ deve ser insatisfatível.

Vamos começar fazendo escrevendo $F = \neg(G \rightarrow H)$ na forma clausal:
$F = \neg((\forall X)\, (p(X) \rightarrow r(X)) \rightarrow ((\exists X)\, p(X) \rightarrow (\forall X)\, r(X)))$
$\Leftrightarrow (\forall X)\, (p(X) \rightarrow r(X)) \wedge \neg((\exists X)\, p(X) \rightarrow (\forall X)\, r(X))$
$\Leftrightarrow (\forall X)\, (\neg p(X) \vee r(X)) \wedge (\exists X)\, p(X) \wedge \neg(\forall X)\, r(X)$
$\Leftrightarrow (\forall X)\, (\neg p(X) \vee r(X)) \wedge (\exists X)\, p(X) \wedge (\exists X)\, \neg r(X)$
Fazendo a renomeação das variáveis:
$F \Leftrightarrow (\forall X)\, ((\neg p(X) \vee r(X)) \wedge (\exists Y)\, p(Y) \wedge (\exists Z)\, \neg r(Z))$
Externalizando os quantificadores temos quatro possibilidades:
$P_1 = (\forall X)\, (\exists Y)\, (\exists Z)\, ((\neg p(X) \vee r(X)) \wedge p(Y) \wedge \neg r(Z))$
$P_2 = (\exists Y)\, (\forall X)\, (\exists Z)\, ((\neg p(X) \vee r(X)) \wedge p(Y) \wedge \neg r(Z))$
$P_3 = (\exists Z)\, (\forall X)\, (\exists Y)\, ((\neg p(X) \vee r(X)) \wedge p(Y) \wedge \neg r(Z))$
$P_4 = (\exists Y)\, (\exists Z)\, (\forall X)\, ((\neg p(X) \vee r(X)) \wedge p(Y) \wedge \neg r(Z))$

Na primeira possibilidade os existenciais estão após o universal, na segunda e terceira o universal está entre os existenciais e na quarta o universal vem por último. A possibilidade 4 é a mais provável de chegar à cláusula vazia, mas vamos fazer todas elas.

Após a skolemização teremos :
$P_1 \Rightarrow \{\{\neg p(X), r(X)\}, \{p(f(X))\}, \{\neg r(g(X))\}\}$
$P_2 \Rightarrow \{\{\neg p(X), r(X)\}, \{p(a)\}, \{\neg r(g(X))\}\}$
$P_3 \Rightarrow \{\{\neg p(X), r(X)\}, \{p(f(X))\}, \{\neg r(a)\}\}$
$P_4 \Rightarrow \{\{\neg p(X), r(X)\}, \{p(a)\}, \{\neg r(b)\}\}$

As expansões por resolução são:

	P_1	P_2	P_3	P_4
1	$\{\neg p(X), r(X)\}$	$\{\neg p(X), r(X)\}$	$\{\neg p(X), r(X)\}$	$\{\neg p(X), r(X)\}$
2	$\{p(f(X))\}$	$\{p(a)\}$	$\{p(f(X))\}$	$\{p(a)\}$
3	$\{\neg r(g(X))\}$	$\{\neg r(g(X))\}$	$\{\neg r(b)\}$	$\{\neg r(b)\}$
4		$\{r(a)\}$ R(1,2) $\{X \leftarrow a\}$	$\{\neg p(a)\}$ R(1,3) $\{X \leftarrow a\}$	$\{r(a)\}$ R(1,2) $\{X \leftarrow a\}$
5				$\{\neg p(b)\}$ R(1,3) $\{X \leftarrow b\}$

Como em nenhuma das expansões é possível chegar à cláusula vazia, chegamos à conclusão que de fato H não é consequência lógica de G.

Programação Lógica

A expressão “programação lógica” trata do uso da lógica como linguagem de programação de computadores. De forma mais solta, programação lógica pode ser entendida como a utilização da lógica para a solução de problemas. Uma ideia fundamental na programação lógica é que um algoritmo é formado por dois elementos disjuntos, a lógica e o controle. O controle consiste em como obter a solução enquanto o componente lógico corresponde à descrição do problema que deve ser solucionado.

Os sistemas de dedução que estudamos, mais especificamente o sistema de resolução, é o controle, e faz parte da implementação da linguagem. Cabe ao programador escrever apenas o componente lógico, ou seja, fazer a especificação ou modelagem do problema através da linguagem da lógica de predicados. A programação lógica é mais próxima da programação funcional que das linguagens imperativas, já que ambas fazem parte do paradigma declarativo.

Uma linguagem puramente lógica funcionaria da forma que foi descrita acima e é o que já vínhamos fazendo durante todo o texto, usando lógica para representar problemas e resolvendo tais problemas através do uso dos sistemas de dedução apresentados. Considere o exemplo que segue.

Ex. 1: Considere as seguintes afirmações:

- A_1: O pai de André é Carlos.
- A_2: O pai de Breno é Carlos.
- A_3: O pai de Carlos é Daniel.
- A_4: O pai de Daniel é Edward.
- A_5: Se P é pai de F, F é descendente de P.
- A_6: Se F é descendente de P, todo descendente de F também é descendente de P.

Podemos classificar estas afirmações em dois tipos, as quatro primeiras são fatos acerca da relação de paternidade entre certas pessoas e as duas últimas são regras que definem a relação de descendência entre quaisquer pessoas. Como poderíamos representar estas afirmações utilizando a lógica de predicados?

Vamos definir um predicado chamado pai, com dois argumentos, para representar os fatos das três primeiras afirmações. Este predicado será tal que I[pai(X, Y)] = “o pai de X é Y”. Também vamos definir constantes para os indivíduos especificados, I[a] = “André”, I[b] = “Breno”, I[c] = “Carlos”, I[d] = “Daniel”, I[e] = “Edward”. Daí temos:

- A_1 = pai(a, c)
- A_2 = pai(b, c)
- A_3 = pai(c, d)
- A_4 = pai(d, e)

A relação de descendência também tem dois argumentos, vamos definir o predicado como tal que I[desc(X, Y)] = “X é descendente de Y”. Daí podemos escrever as duas últimas afirmações como:

- $A_5 = (\forall F)\,(\forall P)\,(pai(F, P) \rightarrow desc(F, P))$

- $A_6 = (\forall F)\,(\forall X)\,(\forall P)\,((desc(F, P) \wedge desc(X, F)) \rightarrow desc(X, P))$

A fórmula A_5 diz que se F é filho de P então F é descendente de P, seja lá quem são F e P. A fórmula A_6 diz que se F é descendente de P e X é descendente de F, então X é descendente de P, para todos F, X e P.

No exemplo anterior temos a modelagem do conhecimento através da lógica, que é parte do programa lógico. Mas, onde está o problema a ser resolvido? O que falta agora é uma pergunta acerca das coisas que sabemos. O programa lógico é formado por duas partes, a base de conhecimento (ou *knowledge base,* ou KB, ou premissas, ou hipóteses) e uma consulta (ou pergunta). Que perguntas podemos fazer? Qualquer uma, mas nem todas terão resposta.

Lembre que a programação lógica consiste na aplicação dos sistemas de dedução da lógica às fórmulas que representam o problema. Portanto, só pode ser respondido aquilo que pode ser deduzido a partir das fórmulas. No exemplo anterior, que temos informações acerca de uma família, poderíamos perguntar "Edward é avô de Carlos?" e, pelo nosso conhecimento acerca das informações familiares, sabemos que a resposta é sim. No entanto, uma dedução lógica utilizando as informações dadas chegaria a uma resposta negativa, pois não foi definido o que é um "avô". De modo similar, se perguntássemos se "André é filho de Carlos" a resposta seria não, pois não foi descrito o que é um "filho".

A maioria das perguntas universais, tais como "todo mundo tem um descendente?", também terá uma resposta negativa, a menos que a pergunta faça parte da base de conhecimento. As perguntas que fazemos na verdade têm um preâmbulo implícito "A partir deste conjunto de fórmulas, podemos concluir que…". Então só terá resposta positiva aquilo que pode ser deduzido a partir da base de conhecimento e a resposta negativa indica que não é possível provar algo, não necessariamente que algo é falso. É muito importante ter isso em mente quando se utiliza a programação lógica.

No fim, restam duas perguntas que podemos fazer à nossa base de conhecimento: (a) perguntas definidas, cuja resposta é sim ou não. (b) perguntas existenciais, cuja resposta consiste na atribuição de valor a uma variável.

Nos exemplos que seguem, vamos fazer algumas perguntas considerando a base de conhecimento do exemplo 1, $KB_1 = \{A_1, A_2, A_3, A_4, A_5, A_6\}$.

Ex. 2: Dado KB_1, podemos concluir que André é descendente de Daniel?
A afirmação "André é descendente de Daniel" pode ser escrita, utilizando a interpretação já definida, como desc(a, d). Então o que desejamos provar é que $(KB_1 \rightarrow desc(a, d))$ é uma tautologia ou, equivalentemente, que $\neg(KB_1 \rightarrow desc(a, d))$ é insatisfatível.
Já sabemos que $\neg(KB_1 \rightarrow desc(a, d)) \Leftrightarrow (KB_1 \wedge \neg desc(a, d))$. Como KB_1 é uma conjunção de fórmulas, todas na forma Prenex, podemos simplesmente escrever cada elemento da conjunção na FNC.

$KB_1 =$ $pai(a, c) \wedge pai(b, c) \wedge pai(c, d) \wedge pai(d, e) \wedge$
$(\forall F)\,(\forall P)\,(pai(F, P) \rightarrow desc(F, P)) \wedge$
$(\forall F)\,(\forall X)\,(\forall P)\,((desc(F, P) \wedge desc(X, F)) \rightarrow desc(X, P))$

$KB_1 \Leftrightarrow pai(a, c) \wedge pai(b, c) \wedge pai(c, d) \wedge pai(d, e) \wedge$
$\forall^*\,(\neg pai(F, P) \vee desc(F, P)) \wedge$
$\forall^*\,(\neg desc(F, P) \vee \neg desc(X, F) \vee desc(X, P))$

Na forma clausal, fazendo a renomeação de variáveis:

$KB_1 \Rightarrow$ {{pai(a, c)}, {pai(b, c)}, {pai(c, d)}, {pai(d, e)}, {¬pai(A, B), desc(A, B)},
{¬desc(F, P), ¬desc(X, F), desc(X, P)}}

Para aplicação da resolução, nossas cláusulas iniciais são as cláusulas de KB_1 seguidas da negação da coisa que desejamos saber se é consequência lógica de KB_1.

Resolução:

1. {pai(a, c)}
2. {pai(b, c)}
3. {pai(c, d)}
4. {pai(d, e)}
5. {¬pai(A, B), desc(A, B)}
6. {¬desc(F, P), ¬desc(X, F), desc(X, P)}
7. {¬desc(a, d)}
8. {¬desc(F, d), ¬desc(a, F)} R(6, 7) {X ← a, P ← d}
9. {¬desc(B, d), ¬pai(a, B)} R(5, 8) {F ← B, A ← a}
10. {¬desc(c, d)} R(1, 9) {B ← c}
11. {¬pai(c, d)} R(5, 10) {A ← c, B ← d}
12. { } R(3, 11)

E chegamos à conclusão de que André é descendente de Daniel.

Ex. 3: Dado KB_1, existe alguém que é descendente de Carlos?
A afirmação “Existe alguém que é descendente de Carlos” é escrita como ($\exists$ D) desc(D, c). Aproveitando o desenvolvimento do exemplo anterior, temos que mostrar se ($KB_1 \wedge \neg(\exists D)$ desc(D, c)) $\Leftrightarrow$ ($KB_1 \wedge (\forall D)$ ¬desc(D, c)) é insatisfatível. Como já temos KB_1 na forma clausal, do exemplo anterior, e a pergunta já está na forma adequada, vamos seguir com a resolução.

Resolução:

1. {pai(a, c)}
2. {pai(b, c)}
3. {pai(c, d)}
4. {pai(d, e)}
5. {¬pai(A, B), desc(A, B)}
6. {¬desc(F, P), ¬desc(X, F), desc(X, P)}
7. {¬desc(D, c)}
8. {¬pai(D, c)} R(5, 7) {A ← D, B ← c}
9. { } R(1, 8) {D ← a}

Então temos uma resposta afirmativa, “Sim, podemos provar que há alguém que é descendente de Carlos”. Além disso, se prestarmos atenção nas substituições realizadas, podemos também afirmar que “a” (André) é descendente de Carlos. Isto porque a variável D, da pergunta, foi substituída por a. Isto quer dizer que André é o único descendente de Carlos? Não, também seria possível chegar a cláusula vazia fazendo a resolução de 2 e 8, utilizando a substituição {D ← b}, de onde concluiríamos que Breno é descendente de Carlos.

Ex. 4: Dado KB_1, existe um par de pessoas que são irmãos?
Como discutido anteriormente, esta consulta não faz sentido porque o significado de “irmão” não foi definido na nossa base de conhecimento, portanto nunca vamos conseguir provar a partir dela. No entanto, podemos fazer essa pergunta utilizando nossos conhecimentos. Poderíamos perguntas se há um par de indivíduos que compartilham o mesmo pai, o que poderíamos escrever, através de fórmulas, da seguinte forma $\exists * (pai(S, U) \wedge pai(T, U))$.
A negação desta fórmula é $\forall * \neg(pai(S, U) \wedge pai(T, U)) \Leftrightarrow \forall * (\neg pai(S, U) \vee \neg pai(T, U))$.
Portanto, se $(KB_1 \wedge \forall *(\neg pai(S, U) \vee \neg pai(T, U)))$ for insatisfatível, concluímos que na base há um par de indivíduos (S e T) que compartilham o mesmo pai (U).

Resolução:

1. {pai(a, c)}
2. {pai(b, c)}
3. {pai(c, d)}
4. {pai(d, e)}
5. {¬pai(A, B), desc(A, B)}
6. {¬desc(F, P), ¬desc(X, F), desc(X, P)}
7. {¬pai(S, U), ¬pai(T, U)}
8. {¬pai(S, c)} R(1, 7) {T ← a, U ← c}
9. { } R(2, 8) {S ← b}

Daí concluímos que André e Breno têm o mesmo pai, Carlos. Há outras formas de chegar a cláusula vazia? Estranhamente, sim. Perceba que podemos fazer a resolução de 1 e 8, utilizando a substituição {S ← a}, e também chegaríamos a cláusula vazia. Isto quer dizer que André é irmão dele mesmo? Sim e não. Pelo que sabemos da vida, isso não é verdade, mas segundo nossa definição de irmão a resposta está correta. Em nossa definição não dissemos que S e T devem ter valores distintos, dissemos apenas que são variáveis e, como tais, podem assumir qualquer valor.

Os exemplos que passaram servem para levantarmos alguns pontos acerca da programação lógica. Primeiramente, é fundamental lembrar que toda resposta obtida é restrita às hipóteses assumidas. Se assumirmos hipóteses erradas, podemos chegar a conclusões erradas. Além disso, uma resposta negativa diz apenas que não podemos provar o que foi pedido a partir das premissas. Este é um aspecto geral da lógica.

A programação puramente lógica consiste, como já foi dito, no que foi feito até aqui, a aplicação dos sistemas de dedução para a obtenção de provas. No entanto, este método levanta alguns problemas quando utilizado na prática. No exemplo 3, vimos que há duas respostas possíveis “André” e “Breno”, suponha que duas pessoas fazem a resolução e cada um chega a uma resposta diferente, como comparar os resultados? É possível, mas seria mais interessante uma uniformização da resposta obtida.

Estas resoluções poderiam ser bem mais longas, se você prestar atenção, em todos os passos da resolução eu escolhi envolver a última cláusula, e há um motivo para isso, que já foi discutido anteriormente. Se fosse possível chegar à cláusula vazia sem utilizar a última cláusula, isso quer dizer que nossa base de conhecimentos é inconsistente e daí qualquer coisa poderia ser provada a partir dela. Ao escolher envolver sempre a última cláusula eu estou assumindo que a base de conhecimentos é consistente.

A cláusula {¬desc(F, P), ¬desc(X, F), desc(X, P)} permite a criação de uma resolução infinita. Imagine que a cláusula da pergunta é {¬desc(A, B)} (“Há alguém que é descendente de alguém?”). Ao resolver as duas cláusulas, removemos o literal positivo e ficamos com dois literais negativos. Estes dois literais negativos por sua vez podem ser resolvidos com a mesma cláusula dando origem a quatro literais negativos, e assim sucessivamente, sem nunca chegar a uma resposta.

Prolog

Prolog é uma linguagem de programação, muitas vezes utilizada como sinônimo de programação lógica, apesar de ser uma implementação específica e, como veremos em breve, não ser puramente lógica.

Os programas em Prolog são a base de conhecimento (KB), à qual podemos fazer consultas (*queries*). Em Prolog, as bases de conhecimento são formadas por cláusulas de Horn definidas, ou seja, cláusulas que contêm exatamente um literal positivo. Quando a cláusula é unitária, ela deve ser uma afirmação, já que só tem um literal e ele deve ser positivo, chamamos essas cláusulas de *fatos* e quando não é unitária ela será da forma $\{\neg L_1, \neg L_2, \ldots, \neg L_k, L_{k+1}\} \Leftrightarrow \forall^* (\neg L_1 \vee \neg L_2 \vee \ldots \vee \neg L_k \vee L_{k+1}) \Leftrightarrow \forall^* (\neg(L_1 \wedge L_2 \wedge \ldots \wedge L_k) \vee L_{k+1}) \Leftrightarrow \forall^* ((L_1 \wedge L_2 \wedge \ldots \wedge L_k) \rightarrow L_{k+1})$. Ou seja, as cláusulas não unitárias são implicações ou *regras* onde os literais negativos são as condições ou antecedentes e o literal positivo é o consequente. Portanto, toda base de conhecimento em Prolog é composta de fatos e implicações.

As consultas, por sua vez, são cláusulas sem literais positivos. Isto se dá porque já vimos que ¬(KB → consulta) ⇔ KB ∧ ¬consulta, ou seja, a consulta aparece negada.

Como há apenas literais negados na cláusula de consulta, podemos pensar em cada literal como um objetivo a ser atingido. As cláusulas de KB têm exatamente um literal positivo, então, ao fazer a resolução da consulta com uma cláusula de KB, uma nova cláusula contendo apenas literais negativos é gerada, ou seja, uma nova consulta.

Ex: Nos exemplos anteriores tínhamos a cláusula A = {¬desc(F, P), ¬desc(X, F), desc(X, P)} e ao fazer a pergunta “André é descendente de Daniel?” surgiu a cláusula de consulta C_0 = {¬desc(a, d)}. A resolução de A e C_0 é C_1 = {¬desc(F, d), ¬desc(a, F)}.
Podemos interpretar isto da seguinte forma, C_0 pergunta se André é descendente de Daniel e A diz que X é descendente de P se F é descendente de P e X é descendente de F. Ao identificar X com André e P com Daniel, chega-se à conclusão que André é descendente de Daniel se há uma certa pessoa, F, que é descendente de Daniel e André e descendente desta pessoa, que é o que está escrito em C_1.

A resolução em Prolog é feita da seguinte forma. Suponha que a base de conhecimento é KB = $\{P_1, P_2, \ldots P_N\}$, onde P_i são cláusulas de Horn definidas e a consulta é $C_0 = \{\neg L_1, \neg L_2, \ldots, \neg L_M\}$, onde L_i são literais.

1. Faça i = 0.
2. Faça k(i) = 1
3. Verifique se o primeiro literal de C_i pode ser unificado com o literal positivo de uma variação de $P_{k(i)}$ (uma variação de uma cláusula é a mesma cláusula com novos nomes para as suas variáveis). Se sim, vá para o passo 4, se não, para o passo 6.
4. Faça $C_{i+1} = R(P_{k(i)}, C_i)$, onde os literais negativos de $P_{k(i)}$ ocorrem no início de C_{i+1}.

5. Se C_{i+1} é a cláusula vazia, pare, a resposta é positiva, retorne as substituições das variáveis da consulta. Senão, incremente i e volte para o passo 2.
6. Se k(i) < N, incremente k(i) e volte para o passo 3. Se não, vá para o passo 7.
7. Se i > 0, decremente i, incremente k(i) e volte para o passo 3. Senão, pare, a resposta é negativa.

Este método de resolução garante que as respostas são sempre obtidas na mesma ordem para uma dada base de conhecimento e consulta. No entanto, deixamos de ter a programação puramente lógica porque agora a ordem em que escrevemos o nosso programa vai afetar o resultado obtido. As cláusulas da base de conhecimentos e os literais da cláusula de consulta são avaliados sequencialmente, na ordem em que foram escritos. Quando não é possível chegar à cláusula vazia, o programa retorna à última unificação realizada e tenta uma alternativa. Se não há mais alternativas, a resolução para e concluímos que não é possível chegar à cláusula vazia. Este retorno a um estado anterior é conhecido como *backtracking*.

Em Prolog, consideramos que cada cláusula da base de conhecimento está quantificada universalmente, de modo que o escopo de cada variável é a própria cláusula. De qualquer modo, como uma "cópia" (variação) da cláusula é usada para a resolução, não teremos este problema. É importante lembrar que a cada etapa da resolução as variáveis da cláusula da base são atualizadas.

Ex: KB = {
1. {p(Y, a), ¬s(a, Y), ¬r(Z, a)},
2. {p(W, c), ¬q(c, Z), ¬r(Z, W)},
3. {q(c, c)},
4. {r(c, b)},
5. {r(a, a)},
6. {s(X, Y), ¬t(X, g(Y))},
7. {s(X, Y), ¬u(h(X), Y)},
8. {u(h(a), b)}
}

e C_0 = {¬p(b, X)}

Vamos aplicar o procedimento descrito acima.

- i = 0; k(0) = 1
- Podemos resolver C_0 com 1 usando a substituição {X ← a, Y ← b}. C_1 = {¬s(a, b), ¬r(Z, a)}.
- i = 1; k(1) = 1
- Não podemos unificar o primeiro literal de C_1 com 1. k(1) = 2.
- Não podemos unificar o primeiro literal de C_1 com 2. k(1) = 3.
- Não podemos unificar o primeiro literal de C_1 com 3. k(1) = 4.
- Não podemos unificar o primeiro literal de C_1 com 4. k(1) = 5.
- Não podemos unificar o primeiro literal de C_1 com 5. k(1) = 6.
- Podemos unificar o primeiro literal de C_1 com uma variação de 6: {s(X_1, Y_1), ¬t(X_1, g(Y_1))}. O índice é usado para deixar claro que não estamos falando do mesmo X ou Y da primeira substituição. A substituição usada é {X_1 ← a, Y_1 ← b}. C_2 = {¬t(a, g(b)), ¬r(Z, a)}.
- i = 2; k(2) = 1

- Não podemos unificar o primeiro literal de C_2 com 1. k(2) = 2.
- Não podemos unificar o primeiro literal de C_2 com 2. k(2) = 3.
- Não podemos unificar o primeiro literal de C_2 com 3. k(2) = 4.
- Não podemos unificar o primeiro literal de C_2 com 4. k(2) = 5.
- Não podemos unificar o primeiro literal de C_2 com 5. k(2) = 6.
- Não podemos unificar o primeiro literal de C_2 com 6. k(2) = 7.
- Não podemos unificar o primeiro literal de C_2 com 7. k(2) = 8.
- Não podemos unificar o primeiro literal de C_2 com 8.
- Como k(2) não é menor que 8, decrementamos i, e incrementamos k(i).
- i = 1; k(1) = 7
- Podemos unificar o primeiro literal de C_1 com uma variação de 7: $\{s(X_2, Y_2), \neg u(h(X_2), Y_2)\}$. A substituição usada é $\{X_2 \leftarrow a, Y_2 \leftarrow b\}$. $C_2 = \{\neg u(h(a), b), \neg r(Z, a)\}$.
- i = 2; k(2) = 1
- Não podemos unificar o primeiro literal de C_2 com 1. k(2) = 2.
- Não podemos unificar o primeiro literal de C_2 com 2. k(2) = 3.
- Não podemos unificar o primeiro literal de C_2 com 3. k(2) = 4.
- Não podemos unificar o primeiro literal de C_2 com 4. k(2) = 5.
- Não podemos unificar o primeiro literal de C_2 com 5. k(2) = 6.
- Não podemos unificar o primeiro literal de C_2 com 6. k(2) = 7.
- Não podemos unificar o primeiro literal de C_2 com 7. k(2) = 8.
- Podemos unificar o primeiro literal de C_2 com 8. $C_3 = \{\neg r(Z, a)\}$.
- i = 3; k(3) = 1
- Não podemos unificar o primeiro literal de C_3 com 1. k(3) = 2.
- Não podemos unificar o primeiro literal de C_3 com 2. k(3) = 3.
- Não podemos unificar o primeiro literal de C_3 com 3. k(3) = 4.
- Não podemos unificar o primeiro literal de C_3 com 4. k(3) = 5.
- Podemos unificar o primeiro literal de C_3 com 5. A substituição usada é $\{Z \leftarrow a\}$. $C_4 = \{\ \}$.
- Como a cláusula é vazia: sim, a substituição é $\{X \leftarrow a\}$

A pergunta feita era se havia algum valor do segundo argumento que, quando utilizado em conjunto com b no primeiro argumento, satisfaz o predicado p, dada a base de conhecimento. A resposta é sim, e p(b, a) é consequência lógica da base de conhecimentos.

Este procedimento costuma ser representado através de uma árvore cujos nós são as cláusulas de consulta e as transições são rotuladas com a cláusula da base e a substituição utilizada.

$$C_0 = \{\neg p(b, X)\}$$

$$\downarrow 1. \{X \leftarrow a, Y \leftarrow b\}$$

$$C_1 = \{\neg s(a, b), \neg r(Z, a)\}$$

$$6. \{\{X_1 \leftarrow a, Y_1 \leftarrow b\} \swarrow \qquad\qquad \searrow 7. \{X_2 \leftarrow a, Y_2 \leftarrow b\}$$

$C_2 = \{\neg t(a, g(b)), \neg r(Z, a)\}$ $C_2 = \{\neg u(h(a), b), \neg r(Z, a)\}$

ramo aberto

↓ 8.

$C_3 = \{\neg r(Z, a)\}$

↓ 5. $\{Z \leftarrow a\}$

$C_4 = \{\ \}$

Este procedimento pode ser facilmente modificado para retornar outras respostas, bastando que, ao invés de parar ao chegar na cláusula vazia, seja dada a opção de incrementar k(i) e voltar para o passo 3, ou seja, fazer o *backtracking* para tentar outras alternativas.

Ex: Vamos continuar o desenvolvimento do exemplo anterior dando a possibilidade de continuar buscando novas respostas depois de achar a cláusula vazia.

Tínhamos, k(0) = 1, k(1) = 7, k(2) = 8, k(3) = 5 e i =3. Os k(i) nos dizem qual a última cláusula da base que investigamos para aquela cláusula objetivo e i diz qual a cláusula atual, a profundidade na árvore de busca. Vamos continuar incrementando k(3) e voltando para o passo 3.

- k(3) = 6.
- Não podemos unificar o primeiro literal de C_3 com 6. k(3) = 7.
- Não podemos unificar o primeiro literal de C_3 com 7. k(3) = 8.
- Não podemos unificar o primeiro literal de C_3 com 8.
- Como k(3) não é menor que 8, decrementamos i, e incrementamos k(i).
- i = 2; k(2) = 9
- Como k(2) não é menor que 8, decrementamos i, e incrementamos k(i).
- i = 1; k(1) = 8
- Não podemos unificar o primeiro literal de C_1 com 8.
- Como k(1) não é menor que 8, decrementamos i, e incrementamos k(i).
- i = 0; k(0) = 2
- Podemos unificar o primeiro literal de C_0 com uma variação de 2: $\{p(W, c), \neg q(c, Z_1), \neg r(Z_1, W)\}$. A substituição usada é $\{X \leftarrow c, W \leftarrow b\}$. $C_1 = \{\neg q(c, Z_1), \neg r(Z_1, b)\}$.
- i = 1; k(1) = 1
- Não podemos unificar o primeiro literal de C_1 com 1. k(1) = 2.
- Não podemos unificar o primeiro literal de C_1 com 2. k(1) = 3.
- Podemos unificar o primeiro literal de C_1 com 3. A substituição usada é $\{Z_1 \leftarrow c\}$. $C_2 = \{\neg r(c, b)\}$.
- i = 2; k(2) = 1
- Não podemos unificar o primeiro literal de C_2 com 1. k(2) = 2.
- Não podemos unificar o primeiro literal de C_2 com 2. k(2) = 3.
- Não podemos unificar o primeiro literal de C_2 com 3. k(2) = 4.
- Podemos unificar o primeiro literal de C_2 com 4. $C_3 = \{\ \}$.
- Como a cláusula é vazia: sim, a substituição é $\{X \leftarrow c\}$.

Achamos uma nova resposta, vamos continuar o procedimento, incrementando k(2).

- k(2) = 5
- Não podemos unificar o primeiro literal de C_2 com 5. k(2) = 6.
- Não podemos unificar o primeiro literal de C_2 com 6. k(2) = 7.
- Não podemos unificar o primeiro literal de C_2 com 7. k(2) = 8.
- Não podemos unificar o primeiro literal de C_2 com 8.
- Como k(2) não é menor que 8, decrementamos i, e incrementamos k(i).
- i = 1; k(1) = 4
- Não podemos unificar o primeiro literal de C_1 com 4. k(1) = 5.
- Não podemos unificar o primeiro literal de C_1 com 5. k(1) = 6.
- Não podemos unificar o primeiro literal de C_1 com 6. k(1) = 7.
- Não podemos unificar o primeiro literal de C_1 com 7. k(1) = 8.
- Não podemos unificar o primeiro literal de C_1 com 8.
- Como k(1) não é menor que 8, decrementamos i, e incrementamos k(i).
- i = 0; k(0) = 3
- Não podemos unificar o primeiro literal de C_0 com 3. k(0) = 4.
- Não podemos unificar o primeiro literal de C_0 com 4. k(0) = 5.
- Não podemos unificar o primeiro literal de C_0 com 5. k(0) = 6.
- Não podemos unificar o primeiro literal de C_0 com 6. k(0) = 7.
- Não podemos unificar o primeiro literal de C_0 com 7. k(0) = 8.
- Não podemos unificar o primeiro literal de C_0 com 8.
- Como k(0) não é menor que 8, pare, esgotamos todas as possibilidades.

Quando chegamos em k(0) = 8, já tentamos unificar a consulta com todas as cláusulas da base, por isso não há mais tentativas. A árvore abaixo ilustra o procedimento de busca completo realizado.

$C_0 = \{\neg p(b, X)\}$

1. $\{X \leftarrow a, Y \leftarrow b\}$ ↙ ↘ 2. $\{X \leftarrow c, W \leftarrow b\}$

$C_1 = \{\neg s(a, b), \neg r(Z, a)\}$ $C_1 = \{\neg q(c, Z_1), \neg r(Z_1, b)\}$

6. $\{\{X_1 \leftarrow a, Y_1 \leftarrow b\}$ ↙ ↘ 7. $\{X_2 \leftarrow a, Y_2 \leftarrow b\}$ ↓ 3. $\{Z_1 \leftarrow c\}$

$C_2 = \{\neg t(a, g(b)), \neg r(Z, a)\}$ $C_2 = \{\neg u(h(a), b), \neg r(Z, a)\}$ $C_2 = \{\neg r(c, b)\}$

ramo aberto

↓ 8. ↓ 4.

$C_3 = \{\neg r(Z, a)\}$ $C_3 = \{\ \}$

↓ 5. $\{Z \leftarrow a\}$

$C_4 = \{\ \}$

Agora já sabemos como a resolução é realizada no Prolog. Neste texto, especificamente, os exemplos e testes relativos ao Prolog serão realizados segundo a implementação SWI (disponível em swi-prolog.org).

Sintaxe

Já vimos que os programas em Prolog consistem de duas partes, a base de conhecimento, formada por cláusulas de Horn definidas e as consultas, formadas apenas por literais negativos. Vamos ver agora os elementos sintáticos que nos permitem escrever programas nesta linguagem.

Termos

No jargão da linguagem, os programas são compostos por *termos*, que podem ser *átomos*, *números, variáveis* ou *estruturas (termos complexos)*. É importante salientar que as palavras *termos* e *átomos* têm significados distintos em Prolog e na lógica em geral.

Átomos são compostos por letras, números e símbolos e devem iniciar com uma letra minúscula. Se o nome do átomo estiver entre aspas simples (`'`) ele pode iniciar com qualquer caractere. Os átomos são os nomes de predicados (*functors*) ou constantes.

Números são constantes, não podem ser usados como nome de predicados, especiais por adotarem a nomenclatura usualmente adotada para representação numérica em computador. A constante 1a não é válida, pois inicia com um dígito e não com uma letra minúscula, mas a constante 1 é válida, pois é apenas um número. Os números podem ser formado apenas por dígitos, precedidos ou não do sinal negativo, (inteiros: `-1`, `0`, `-23`, `43`), números com parte fracionária, precedidos ou não do sinal negativo (reais: `0.2`, `-0.33`, `0.47`) ou um desses casos seguidos de uma potência inteira de 10 (`2e2`, `-1.3e-2`, `5e-2`). Não é permitida a omissão do dígito antes do ponto (.3 não é válido, por exemplo) e não é permitido o uso de potências não inteiras de 10 (2e3.4 não é válido, por exemplo). Apesar

de ser permitido o uso de números, para o Prolog eles são apenas símbolos não têm significado de quantidade, a menos que isto seja dito ao Prolog, como veremos futuramente.

Variáveis são compostas por letras, números e símbolos e devem iniciar com uma letra maiúscula ou o sublinhado (*underscore*: _). A variável cujo nome é apenas _ é especial e é chamada de *variável anônima* esta variável especial é usada quando o resultado da unificação não nos interessa a consulta {¬p(X)} retorna o valor de X que deve ser utilizado para satisfazer p, se isso for possível, a consulta {¬p(_)} apenas diz se há um valor que satisfaz p sem retornar qual é esse valor.

As estruturas combinam os elementos anteriores e são utilizadas para definir predicados. Uma estrutura tem um nome de predicado (*functor*) seguido dos argumentos, separados por vírgulas, entre parênteses. Os argumentos, por sua vez, são qualquer termo Prolog, inclusive outra estrutura. Os predicados com mesmo nome, mas com quantidades distintas de argumentos são considerados predicados distintos. São exemplos de estruturas: `pai(a, b)`, `depois('André', 'Carlos')`, `menor(X, 2)`, `menor(Y)`. Nos exemplos, `'André'` é uma constante, pois está entre aspas simples. Ao se referir aos predicados, o Prolog utiliza a notação `/N` para indicar o número de argumentos predicados. Se o prolog diz que não existe predicado `p /2`, por exemplo, ele indica que não foi definido um predicado cujo nome é p e que tem 2 argumentos.

Programas

Utilizando os termos e alguns conectivos podemos escrever as cláusulas do programa Prolog. Existem 3 conectivos definidos na linguagem prolog. A conjunção, representada pela vírgula (`,`), a disjunção, representada pelo ponto-e-vírgula (`;`) e a implicação, representada por dois pontos e hífen (`:-`). A implicação, em Prolog, é escrita de forma reversa ao que estamos acostumados na lógica, o consequente vem antes do conectivo e o antecedente depois. A fórmula lógica ((p(X) ∧ r(Y)) → q(X, Y)) é representada em Prolog como `q(X, Y) :- p(X), r(Y).`

Toda cláusula em Prolog deve ser finalizada com um ponto (`.`). É esse caractere que indica o fim da definição da cláusula.

De posse destes dados, já podemos escrever qualquer base de conhecimento, lembrando sempre que as cláusulas da base de conhecimento são cláusulas de Horn definidas e terão, portanto, apenas um literal positivo. Os *fatos* são cláusulas formadas apenas por termos, as *regras* são implicações onde o literal positivo aparece do lado esquerdo (antes do `:-`).

Lembre que o Prolog compara o primeiro literal da cláusula de consulta com o literal positivo de cada cláusula da base de conhecimento, esse é um dos motivos pelo qual a implicação é escrita de forma reversa no Prolog. Assim o literal positivo está sempre no início da cláusula e facilita a comparação.

Apesar de ser definido na linguagem, a disjunção não é necessária para a definição da base de conhecimentos. Lembre que (A ∨ B) → C pode ser escrito como (A → C) ∧ (B → C). Portanto, a cláusula `p(X) :- q(X) ; r(X).` pode ser escrita através de duas cláusulas: `p(X) :- q(X). p(X) :- r(X).`

Ex: A base de conhecimento do último exemplo,
KB = {{p(Y, a), ¬s(a, Y), ¬r(Z, a)}, {p(W, c), ¬q(c, Z), ¬r(Z, W)}, {q(c, c)}, {r(c, b)}, {r(a, a)},

{s(X, Y), ¬t(X, g(Y))}, {s(X, Y), ¬u(h(X), Y)}, {u(h(a), b)}}

pode ser escrita em Prolog como:

```
p(Y, a) :- s(a, Y), r(Z, a).
p(W, c) :- q(c, Z), r(Z, W).
q(c, c).
r(c, b).
r(a, a).
s(X, Y) :- t(X, g(Y)).
s(X, Y) :- u(h(X), Y).
u(h(a), b).
```

Como escrever a cláusula de consulta se não temos o conectivo de negação? Para responder essa pergunta precisamos falar um pouco sobre o uso do Prolog. Ao utilizar a linguagem, escrevemos e carregamos uma base de conhecimentos e o *prompt* da linguagem espera uma consulta. A consulta é indicada por um conectivo especial formado por interrogação e hífen (`?-`). Este conectivo é uma espécie de implicação que espera apenas os antecedentes após o hífen. Do ponto de vista lógico, podemos imaginar que é uma implicação cujo consequente é o símbolo de verdade 0, de tal modo que temos:
`?- p(X), q(Y)` ⇔ (p(X) ∧ q(Y)) → 0 ⇔ ¬(p(X) ∧ q(Y)) ∨ 0 ⇔ ¬p(X) ∨ ¬q(Y).

Ou seja, os literais escritos após `?-` são os literais negados da cláusula de consulta. Como a consulta deve ser uma fórmula única, pode ser interessante o uso do conectivo de disjunção em consultas. Não é possível saber se há um valor que satisfaz p ou q sem uso da disjunção na consulta.

Ex.: No exemplo anterior, queríamos saber se havia algum X que satisfazia p(b, X) e a cláusula de consulta era {¬p(b, X)}. Em Prolog essa consulta é escrita como `?- p(b, X).`

Há uma outra utilidade do conectivo de disjunção. Ao encontrar uma cláusula vazia o Prolog exibe a unificação utilizada e aguarda uma interação do usuário. Se o usuário digitar o operador de disjunção (`;`), a busca por uma outra solução continua, como fizemos no exemplo anterior.

Unificação

Um operador fundamental do Prolog é o operador de unificação (`=`). Como já vimos, a unificação é a operação fundamental para a execução do algoritmo de resolução. São as substituições resultantes da unificação que determinam as respostas apresentadas pelo Prolog.

Dado o programa `p(a).` a consulta `?- p(X).` dá como resultado `X = a`. Esta resposta dada pelo Prolog deve ser interpretada como “Há um valor de X que satisfaz p(X), este valor é obtido unificando X com a”.

Ao fazer a consulta `?- X = Y.`, o Prolog tenta unificar as variáveis `X` e `Y`. Se `X` ainda não foi instanciada (unificada) e `Y` foi instanciada para qualquer termo, então `X` e `Y` podem ser unificados e `X` passa a assumir o valor de `Y`.

Constantes só podem ser unificadas com elas mesmas. Portanto, se `X` e `Y` já foram instanciados, `X = Y` só é verdade se tanto `X` quanto `Y` assumem o mesmo valor.

Estruturas são unificadas se possuem o mesmo nome, a mesma quantidade de argumentos e cada um dos seus argumentos podem ser unificados.

Ex.: Os termos em negrito são a resposta dada pelo Prolog. Na minha implementação, respostas positivas que não necessitam de substituição de variáveis retornam apenas `true` enquanto respostas negativas retornam `false`. Em algumas implementações o retorno pode ser `yes` ou `no`, ao invés de `true` e `false`.

```
?- X = a.
```
X = a.

Isso ocorre porque X é uma variável e a é uma constante. A unificação é possível e a substituição é {X ← a}.

```
?- a = a.
```
true.

Porque os dois operandos da unificação são constantes idênticas. Neste caso nenhuma substituição de variável foi realizada.

```
?- a = b.
```
false.

Porque os operandos são constantes distintas.

```
?- X = a, X = Y.
```
X = Y, Y = a.

Há dois literais na consulta, o primeiro é satisfeito unificando `X` com a constante `a`. Como estamos na mesma cláusula, a variável `X` do segundo literal vale agora `a`, e daí `Y` é instanciado com o mesmo valor. A resposta do Prolog diz que `X` e `Y` são unificados (compartilham o mesmo valor) e que o valor de `Y` é `a`.

```
?- X = a, X = b.
```
false.

Depois que o primeiro objetivo é satisfeito, `X` é unificado com `a`. A partir deste ponto o segundo objetivo não pode ser satisfeito porque agora `X` vale `a` e não pode ser unificado com `b`.

```
?- X = a, Y = a, X = Y.
```
X = a, Y = a.

Tanto `X` quanto `Y` são instanciados com `a` depois que os dois primeiros objetivos são satisfeitos, daí `X` e `Y` podem ser unificados pois ambos têm o mesmo valor.

```
?- X = a, Y = b, X = Y.
```
false.

`X` e `Y` são instanciados para valores distintos depois que os dois primeiros objetivos são satisfeitos, daí `X` e `Y` não podem ser unificados. E como o terceiro objetivo falha, a conjunção dos objetivos como um todo falha.

```
?- p(f(X), Y, X) = p(Z, g(Z), a).
```

X = a, Y = g(f(a)), Z = f(a).

Este exemplo já foi visto em outro ponto deste texto, quando o algoritmo de unificação foi apresentado. Primeiramente o Prolog verifica que os dois predicados a serem unificados têm o mesmo nome e o mesmo número de argumentos. Passado esse teste, é verificado se é possível unificar todos os argumentos, ou seja, é realizada a consulta `?- f(X) = Z, Y = g(Z), X = a.` Caso seja possível, as substituições são exibidas, como neste exemplo.

```
?- 'gato' = gato.
true.
```

Para átomos que iniciam com letra minúscula é indiferente para o Prolog se ele está entre aspas simples ou não.

```
?- '2' = 2.
false.
```

Neste caso não é unificável porque `'2'` é um átomo e `2` é um número.

```
?- 'Casa' = Casa.
Casa = 'Casa'.
```

Neste caso a unificação é possível mas o retorno não foi simplesmente true. Isto ocorre porque `Casa`, sem aspas, é uma variável e com aspas é um átomo. O que o Prolog diz é que a variável `Casa` deve ser instanciada para o valor `'Casa'` para que os termos sejam idênticos.

Atenção, ao contrário da unificação que vimos ao estudar o método da resolução, a unificação do Prolog não faz o teste de ocorrência, ou seja, ele permite a unificação de uma variável com um termo que contém esta variável. O comportamento obtido ao tentar fazer a consulta `?- X = f(X)` depende da implementação do Prolog que você está utilizando.

Em algumas implementações há um estouro de memória, outras percebem que a recursão não tem fim e retornam `X = f(f(f(f(...))))`. Na implementação que utilizo o retorno é simplesmente `X = f(X)`.

A ausência do teste de ocorrência é uma escolha visando a eficiência da computação. Como a unificação é feita diversas vezes, há um ganho de desempenho ao pular o teste de ocorrência. O Prolog confia que o programador (nós) não vamos apresentar uma situação patológica como esta para ele.

Se você realmente deseja fazer a unificação com o teste de ocorrência, há o predicado nativo `unify_with_occurs_check \2`. Como indicado, este predicado tem dois argumentos e ele faz a unificação dos argumentos utilizando o teste de ocorrência. Como esperado, a consulta `?- unify_with_occurs_check(X, f(X))` retorna `false`.

Programas em Prolog dependem fortemente da unificação e é bom entender bem como este procedimento funciona. Ao contrário de outras linguagens de programação, o Prolog não possui entrada e saídas. Temos uma consulta e a única forma de passar valores entre predicados é através da unificação de variáveis.

Ex.: Considere uma base de conhecimento que tem apenas o seguinte fato:
`h(r(p(X1, Y), p(X2, Y))).`

Para o Prolog, o que está sendo dito é que o predicado `h` é verdadeiro se o seu argumento é uma função `r` que, por sua vez, possui dois argumentos `p`, cada um deles com dois argumentos tais o segundo valor é igual nos dois `p`, e o primeiro argumento é livre. Vamos dar uma interpretação mais natural para esta cláusula.
Seja `h /1` o predicado que é verdadeiro se seu argumento é uma reta horizontal.
Seja `r /2` uma reta, definida através de dois pontos.
Seja `p /2` um ponto, definido através dos seus valores de abscissa e ordenada.
Deste modo, o que está sendo dito na base é que uma reta é horizontal se ela passa por dois pontos com a mesma ordenada.

Através da unificação podemos obter algumas informações interessantes a partir deste predicado.

```
?- h(r(p(1, 2), p(1, 3))).
false.
```

A reta que passa pelos pontos (1, 2) e (1, 3) é horizontal? não.

```
?- h(r(p(1, 2), p(0, X))).
X = 2.
```

Se uma reta passa pelo ponto (1, 2), qual deve ser sua ordenada na abscissa 0 para que ela seja horizontal? 2.

```
?- h(r(p(1, 2), X)).
X = p(_5062, 2).
```

Se uma reta passa pelo ponto (1, 2), em qual outro ponto ela deve passar para que seja horizontal? Na resposta do Prolog, o primeiro argumento de p inicia com um sublinhado, portanto é uma variável. O que está sendo dito é que deve ser um ponto cuja ordenada é 2 e cuja abscissa tem qualquer valor. O número 5062 após o sublinhado não tem qualquer significado especial e é gerado aleatoriamente pelo Prolog.

```
?- h(X).
X = r(p(_5036, _5038), p(_5042, _5038).
```

Como deve ser uma reta para que seja horizontal? A resposta é que ela deve passar por dois pontos e os argumentos dos pontos são variáveis com nomes aleatórios. No entanto, perceba que o segundo argumento dos dois `p` são idênticos, indicando que eles devem ter o mesmo valor.

Para todos os efeitos, os valores atribuídos às variáveis `X1` e `X2`, na definição da base de conhecimento, não têm qualquer utilidade. A única coisa que interessa é que os dois pontos têm a mesma ordenada. Poderíamos escrever, o mesmo predicado de forma equivalente utilizando variáveis anônimas.

```
h(r(p(_, Y), p(_, Y))).
```

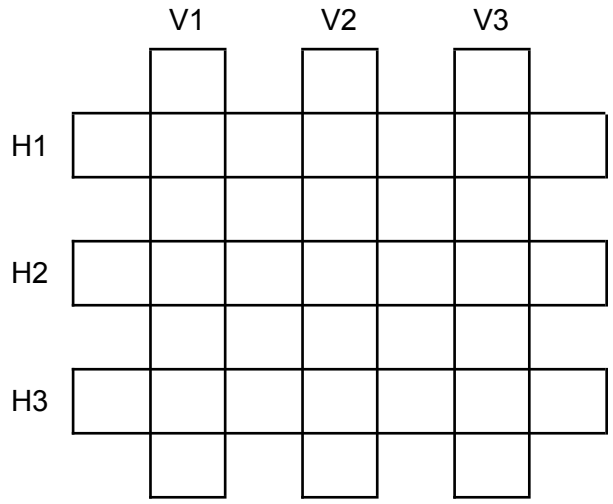
Dizemos apenas que o valor da abscissa não interessa e todas as consultas realizadas anteriormente teriam o mesmo resultado.

Ex.: Considere a seguinte base de conhecimentos:

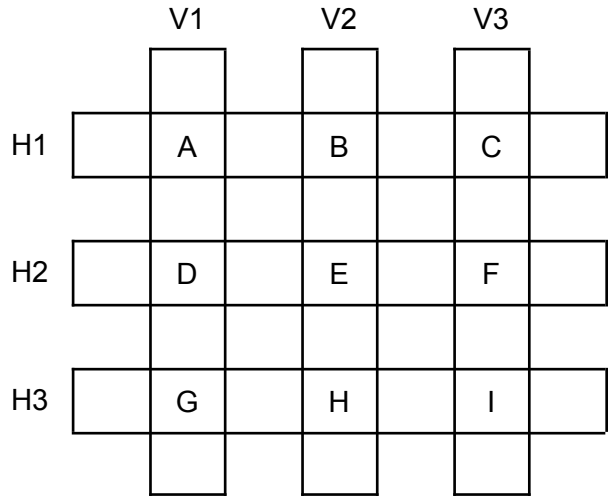
```
p(astante, a,s,t,a,n,t,e).
p(cobalto, c,o,b,a,l,t,o).
p(pistola, p,i,s,t,o,l,a).
p(astoria, a,s,t,o,r,i,a).
p(baratto, b,a,r,a,t,t,o).
```

```
p(statale, s,t,a,t,a,l,e).
```

Os predicados p têm 8 argumentos, o primeiro é uma palavra de 7 letras e os argumentos seguintes são cada uma de suas letras, na ordem em que são escritas.
Considere o seguinte tabuleiro:



Temos que preencher o tabuleiro com palavras na vertical (de cima para baixo) e na horizontal (da esquerda para a direita), de tal modo que cada letra ocupe um quadrado.
Vamos definir uma consulta que determina quais das palavras da base podem ser usadas para preencher o tabuleiro nas posições horizontais (H1, H2 e H3) e verticais (V1, V2 e V3).
Quais as restrições que temos? A palavra deve constar na base e devemos respeitar os cruzamentos das palavras, a quarta letra de V2 deve ser idêntica a quarta letra de H2, por exemplo. Fora dos cruzamentos, não há qualquer restrição. Vamos redesenhar o tabuleiro dando nome aos pontos de cruzamento.



A consulta `?- p(H1,_,A,_,B,_,C,_)` unifica `H1` com uma palavra da base e a segunda letra de `H1` com a variável `A`, a quarta com a variável `B` e a sexta com a variável `C`. Qualquer palavra da base pode ser unificada com `H1`.

De modo similar, a consulta `?- p(H1,_,A,_,B,_,C,_), p(H2,_,D,_,E,_,F,_), p(H3,_,G,_,H,_,I,_)` unifica três palavras da base com `H1`, `H2` e `H3` e suas segunda, quarta e sexta letras, respectivamente com as variáveis `A`, `B`, `C`, `D`, `E`, `F`, `G`, `H` e `I`. Como não há cruzamentos entre as palavras horizontais, quaisquer palavras da base podem ser unificadas com `H1`, `H2` e `H3`.

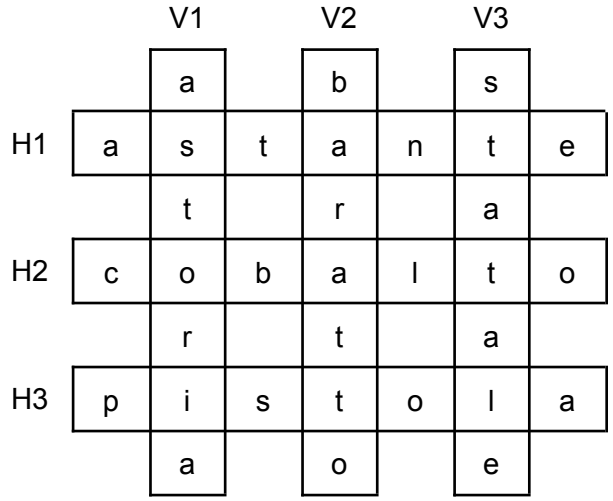
Agora vamos introduzir as palavras verticais, tomando o cuidado de respeitar os cruzamentos. A segunda, quarta e sexta letras da palavra `V1` devem ter os mesmos valores que foram atribuídos a `A`, `D` e `G`. `B`, `E` e `H` para `V2` e `C`, `F` e `I` para `V3`.

Portanto, a consulta que atende a todas as restrições é:

```
?- p(H1,_,A,_,B,_,C,_), p(H2,_,D,_,E,_,F,_), p(H3,_,G,_,H,_,I,_),
p(V1,_,A,_,D,_,G,_), p(V2,_,B,_,E,_,H,_), p(V3,_,C,_,F,_,I,_).
```

```
A = s, B = E, E = a, C = F, F = H, H = t, D = o, G = i, I = l,
H1 = astante, H2 = cobalto, H3 = pistola,
V1 = astoria, V2 = baratto, V3 = statale.
```

A resposta dada pelo Prolog diz que ele achou uma unificação que satisfaz simultaneamente todos os objetivos. De fato, a solução dada está correta:



Apesar de correta, a solução obtida também nos retornou diversas unificações que não nos interessavam, que são os valores das letras nos pontos de cruzamento. Se desejarmos que o Prolog exiba apenas as palavras, podemos acrescentar à nossa base de conhecimento um predicado que é verdadeiro se a solução existe e retorna os valores das palavras.

```
solucao(S,T,U,X,Y,Z) :- p(H1,_,A,_,B,_,C,_), p(H2,_,D,_,E,_,F,_),
                         p(H3,_,G,_,H,_,I,_), p(V1,_,A,_,D,_,G,_),
                         p(V2,_,B,_,E,_,H,_), p(V3,_,C,_,F,_,I,_),
                         S = H1, T = H2, U = H3,
                         X = V1, Y = V2, Z = V3.
```

O predicado `solucao /6` é definido através da consulta criada anteriormente. As unificações ao final fazem com que as variáveis dos argumentos de `solucao` sejam identificados com as palavras encontradas.

Apesar de correto, estas unificações são desnecessárias porque tudo que elas dizem é que alguns pares de variáveis compartilham o mesmo valor. Portanto, poderíamos escrever este predicado simplesmente como

```
solucao(H1,H2,H3,V1,V2,V3) :-   p(H1,_,A,_,B,_,C,_),
                                p(H2,_,D,_,E,_,F,_),
                                p(H3,_,G,_,H,_,I,_),
                                p(V1,_,A,_,D,_,G,_),
                                p(V2,_,B,_,E,_,H,_),
                                p(V3,_,C,_,F,_,I,_).
```

Agora podemos usar este predicado para obter apenas as informações que nos interessam:

```
?- solucao(H1, H2, H3, V1, V2, V3).
H1 = astante, H2 = cobalto, H3 = pistola,
V1 = astoria, V2 = baratto, V3 = statale.
```

Além do operador de unificação o Prolog também define um operador inverso, uma espécie de negação da unificação. O operador `\=` é falso quando o operador `=` é verdadeiro e vice-versa. Ao contrário do operador de unificação, a resposta do operador `\=` é sempre verdadeiro ou falso, sendo falso quando a unificação é possível. Portanto, a consulta `?- X \= Y` retorna `false`.

Além das operações de unificação e seu inverso, também existe a operação de igualdade e seu inverso, respectivamente `==` e `\==`. Estes operadores não fazem a unificação, eles apenas testam a igualdade entre termos. Deste modo, a forma com que as variáveis são tratadas são diferentes. Na unificação, se X ou Y são variáveis não instanciadas, `?- X = Y` é sempre verdadeiro. Na igualdade, se X ou Y são variáveis não instanciadas `?- X == Y` é sempre falso. Duas variáveis só são iguais se foram instanciadas para o mesmo valor ou se foram unificadas entre si anteriormente. Ou seja, `?- X = Y, X == Y` retorna `X = Y`, o que indica que os dois objetivos foram satisfeitos. O retorno é decorrente da unificação feita no primeiro literal. Sempre que o operador de `==` é verdadeiro o operador `\==` é falso e vice-versa.

Os operadores de diferença (`\==`) e impossibilidade de unificação (`\=`) podem ser usados para impor restrições sobre os valores de variáveis.

Ex.: Considere o seguinte programa em Prolog:

```
pai('Andre', 'Carlos').
pai('Breno', 'Carlos').
irmao(X, Y) :- pai(X, Z), pai(Y, Z).
```

Esse programa define dois fatos, que Carlos é pai de Andre e Breno, e uma regra que diz que duas pessoas são irmãos se têm o mesmo pai. Tudo parece razoável, vamos fazer uma consulta para saber se há um par de irmãos na base.

```
?- irmao(X, Y).
X = Y, Y = 'Andre'
```

A resposta diz que Andre é irmão dele mesmo. Do ponto de vista do que foi escrito no programa lógico, a resposta está correta, porque não foi dito que X e Y devem ter valores distintos. Podemos resolver este problema reescrevendo a regra que define a relação de irmãos como `irmao(X, Y) :- pai(X, Z), pai(Y, Z), X \= Y` ou `irmao(X, Y) :- pai(X, Z), pai(Y, Z), X \== Y.`

Deste modo, só será possível dizer que `X` e `Y` são irmãos se eles têm o mesmo pai e têm valores distintos. No entanto, é fundamental que o teste de diferença seja a última das cláusulas. O Prolog avalia os literais na ordem em que foram escritos, se o teste vier no início, antes das variáveis serem instanciadas, ele não terá qualquer efeito porque `X \= Y` é sempre falso se uma das variáveis não foi instanciada e `X \== Y` é sempre verdadeiro se uma das variáveis não foi instanciada. Neste sentido, o Prolog não é uma linguagem puramente lógica, porque a ordem dos operandos em uma conjunção não deveria afetar o resultado.
Após a modificação sugerida, a mesma consulta passa a retornar `X = 'Andre', Y = 'Breno'`. Após retornar um resultado, o Prolog espera uma interação do usuário, se for digitado o símbolo de disjunção (`;`), uma nova solução é buscada. Podemos interpretar isso como "a solução é essa ***ou*** …".
Neste exemplo, solicitando novas soluções obtemos as seguintes respostas:

```
?- irmao(X, Y).
X = 'Andre', Y = 'Breno';
X = 'Breno', Y = 'Andre';
false.
```

Cada linha é uma resposta, a primeira nos diz que Andre é irmão de Breno, a segunda nos diz que Breno é irmão de Andre. Ao tentar buscar uma nova resposta, nenhuma é encontrada, por isso a terceira linha é `false`.

Aspectos Procedurais

Apesar de ser baseada na lógica, a utilização da linguagem Prolog tem certos aspectos procedurais que precisam ser levados em conta quando escrevemos nossos programas. Nas seções que seguem abordaremos alguns destes casos.

Recursão

A recursão é um elemento fundamental dos programas em Prolog. A recursão ocorre quando um predicado é definido em termos dele mesmo. Considere o seguinte exemplo, o mesmo que vimos antes de apresentar a sintaxe do Prolog.

Ex.: O predicado `pai(X, Y)` deve ser entendido como "o pai de `X` é `Y`", o predicado `desc(X, Y)` significa que "`X` é descendente de `Y`.

1. `pai(a, c).`
2. `pai(b, c).`
3. `pai(c, d).`
4. `pai(d, e).`
5. `desc(X, Y) :- pai(X, Y).`
6. `desc(X, Y) :- pai(Z, Y), desc(X, Z).`

Perceba que na linha 6 o predicado `desc(X, Y)` é verdadeiro se `desc(X, Z)` é verdadeiro e `pai(Z, Y)` também. Ou seja, X é descendente de Y se Y é pai de um certo Z e X é descendente desse Z. Na linha 5 temos o caso base desta recursão, X é descendente de Y se Y é pai de X, o primeiro nível de descendência.

Já fizemos anteriormente, mas vamos repetir como é feito o processo de inferência nesta base de conhecimento. Lembre-se que o Prolog avalia as cláusulas na ordem em que foram escritas e sempre tenta fazer a unificação do objetivo mais à esquerda da cláusula de consulta, além disso, toda resolução é feita com uma variação da cláusula de programa. Vamos consultar quem são os descendentes de `e`.

P_1	`?- desc(X, e).`	Resolução com 5.
P_2	`?- pai(X, e).`	Resolução com 4.
P_3	`?- .`	`X = d;` Volta para P_1 e resolve com 6.
P_4	`?- pai(Z, e), desc(X, Z).`	Resolução com 4.
P_5	`?- desc(X, d).`	Resolução com 5.
P_6	`?- pai(X, d).`	Resolução com 3.
P_7	`?- .`	`X = c;` Volta para P_5 e resolve com 6.
P_8	`?- pai(Z, d), desc(X, Z).`	Resolve com 3.
P_9	`?- desc(X, c).`	Resolve com 5.
P_{10}	`?- pai(X, c).`	Resolve com 1.
P_{11}	`?- .`	`X = a;` Volta para P_{10} e resolve com 2.
P_{12}	`?- .`	`X = b;` Volta para P_9 e resolve com 6.
P_{13}	`?- pai(Z, c), desc(X, Z).`	Resolve com 1.
P_{14}	`?- desc(X, a).`	Resolve com 5.
P_{15}	`?- pai(X, a).`	Falha. Volta para P_{14} e resolve com 6.
P_{16}	`?- pai(Z, a), desc(X, Z).`	Falha. Volta para P_{13} e resolve com 2.
P_{17}	`?- desc(X, b).`	Resolve com 5.
P_{18}	`?- pai(X, b).`	Falha. Volta para P_{17} e resolve com 6.
P_{19}	`?- pai(Z, b), desc(X, Z).`	Falha. Não há alternativas. Encerra.

Funciona exatamente como o exemplo dado no início da seção de programação lógica, antes de apresentarmos o Prolog. No entanto, o verdadeiro objetivo desta seção não é dizer que a recursão existe, mas alertar para uma diferença entre a lógica pura e o Prolog.

Do ponto de vista lógico, a ordem dos operandos em uma conjunção não afetam o significado da fórmula e, pelo menos do ponto de vista teórico, é possível chegar às mesmas conclusões através do processo de resolução. No Prolog, por causa da forma como o procedimento de resolução é realizado, é preciso tomar cuidado com a ordem na qual escrevemos as cláusulas e também na ordem dos literais das cláusulas.

Ex.: Vamos repetir o programa do exemplo anterior invertendo a ordem das duas últimas cláusulas.

1. `pai(a, c).`
2. `pai(b, c).`
3. `pai(c, d).`
4. `pai(d, e).`
5. `desc(X, Y) :- pai(Z, Y), desc(X, Z).`
6. `desc(X, Y) :- pai(X, Y).`

Fazendo a mesma consulta do exemplo anterior, obtemos as mesmas respostas porém em uma ordem diferente:

```
?- desc(X, e).
X = a; X = b; X = c; X = d.
```

Agora vamos pegar a cláusula 5 e inverter a ordem dos literais.

1. `pai(a, c).`
2. `pai(b, c).`
3. `pai(c, d).`
4. `pai(d, e).`
5. `desc(X, Y) :- desc(X, Z), pai(Z, Y).`
6. `desc(X, Y) :- pai(X, Y).`

Do ponto de vista lógico, tudo permanece igual. Vamos fazer uma consulta:

```
?- desc(X, e).
ERROR: Out of local stack
```

Nenhum resultado foi dado e, além disso, acabou a memória disponível. Por que isso aconteceu? Vamos realizar o procedimento de resolução passo a passo.

P_1 `?- desc(X,e).` Resolução com 5.
P_2 `?- desc(X,Z2), pai(Z2,e).` Resolução com 5.
P_3 `?- desc(X,Z3), pai(Z3,Y3), pai(Z2,e).` Resolução com 5.
P_4 `?- desc(X,Z4), pai(Z4,Y4), pai(Z3,Y3), pai(Z2,e).` Resolução com 5.
⋮

A cláusula de consulta só aumenta a cada etapa, e é sempre possível fazer a resolução do primeiro literal com a quinta cláusula de programa.

Deste modo, é fundamental que tenhamos em mente a forma como o procedimento de resolução é realizado no Prolog. Uma regra essencial é que a chamada recursiva deve ocorrer o mais à direita possível na cláusula, porque os literais são avaliados da esquerda para a direita. Outro ponto importante é que o caso base deve vir, preferencialmente, antes das demais cláusulas.

Ex.: Vamos definir recursivamente os números naturais. Seja `nat(X)` o predicado que significa "`X` é um número natural".

1. `nat(0).`
2. `nat(Y) :- Y = X+1, nat(X).`

O primeiro fato diz que 0 é um número natural e a regra diz que Y é natural se X é natural e Y é o sucessor de X. A unificação que ocorre do lado direito da regra 2 pode ser feita diretamente do lado esquerdo, da seguinte forma.

1. `nat(0).`
2. `nat(X+1) :- nat(X).`

Uma consulta `nat(X)` dá infinitos resultados, e basicamente vemos o Prolog contando:

```
?- nat(X).
X = 0; X = 0+1; X = 0+1+1; X = 0+1+1+1; X = 0+1+1+1+1; …
```

Neste caso não tem nada errado com o programa, os resultados são infinitos porque de fato existem infinitos números naturais. Agora vamos consultar se um certo número é natural ou não.

```
?- nat(0).
true.
?- nat(1).
false.
```

Por que 1 não é um número natural? O Prolog enxerga tudo como símbolos, portanto, ao escrever 0 + 1 o que é visto pelo Prolog é uma estrutura que tem como nome o símbolo +, que ele vê simplesmente como um predicado, e como argumentos os números 0 e 1. Isto pode ser visto através da unificação:

```
?- 0 + 1 = 1.
false.
?- 0 + 1 = +(0, 1).
true.
```

Discutiremos mais sobre aritmética na próxima seção. Por enquanto, vamos continuar avaliando o aspecto recursivo do programa. Então, se queremos saber se 1 é um número natural devemos perguntar se 0+1 é natural, para saber se 2 é natural devemos perguntar se 0+1+1 é natural, e assim sucessivamente.

```
?- nat(0+1).
true.
?- nat(0+1+1+1+1+1+1).
true.
```

Tudo funcionando corretamente. O que acontece se invertermos a função recursiva? considere agora o seguinte programa:

1. `nat(0).`
2. `nat(Y) :- nat(X), Y = X+1.`

Ao fazer a consulta `?- nat(0)` ele responde `true`, como esperado, mas se tentarmos continuar buscando novas respostas um laço infinito terá início, pois sempre teremos `nat(X)` como primeiro elemento da consulta e sempre é possível unificar com a segunda regra, de modo que a consulta só cresce.
Perceba também que, apesar de ser aritmeticamente correto, não é possível definir os naturais usando subtração, da seguinte forma:

1. `nat(0).`
2. `nat(Y) :- X = Y-1, nat(X).`

`Y` é natural se `Y-1` é natural, tudo correto do ponto de vista matemático. Mas, como o Prolog não enxerga a operação como uma subtração, o caso base, `nat(0)`, nunca vai ocorrer.

É fundamental tomar cuidado com a ordem na qual escrevemos as cláusulas e os literais nos programas Prolog. Na dúvida, pode ser interessante fazer passo a passo o processo de resolução para entender o que está ocorrendo. Além da forma como escrevemos as cláusulas, também podemos controlar o fluxo da resolução através de um mecanismo chamado de corte

Corte

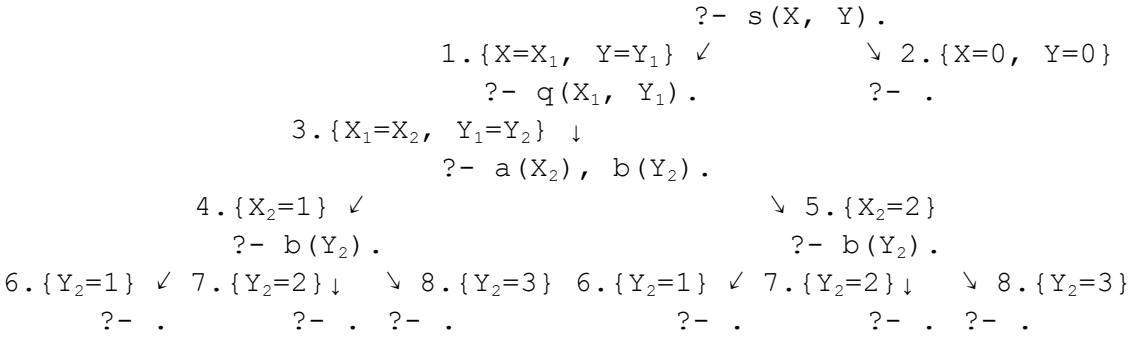
O operador de corte, denotado pela exclamação (`!`), é um operador que permite afetar a forma como o Prolog executa o *backtracking* na resolução. Como visto, ao falhar na unificação ou ao solicitar uma nova resposta o Prolog retorna na árvore de busca até a última unificação realizada e tenta realizar a unificação com outras cláusulas.

O operador de corte diz ao Prolog que o backtracking não deve retroceder além do ponto de corte.

Ex.: Considere o seguinte programa em Prolog:

```
1. s(X, Y) :- q(X, Y).
2. s(0, 0).
3. q(X, Y) :- a(X), b(Y).
4. a(1).
5. a(2).
6. b(1).
7. b(2).
8. b(3).
```

Ao fazer a consulta sobre quais os pares X e Y que satisfazem s, a seguinte árvore de busca é criada



Temos 7 cláusulas vazias, daí as respostas dadas pelo Prolog são:

```
?- s(X, Y).
```

X=Y, Y=1;
X=1, Y=2;
X=1, Y=3;
X=2, Y=1;
X=Y, Y=2;
X=2, Y=3;
X=Y, Y=0.

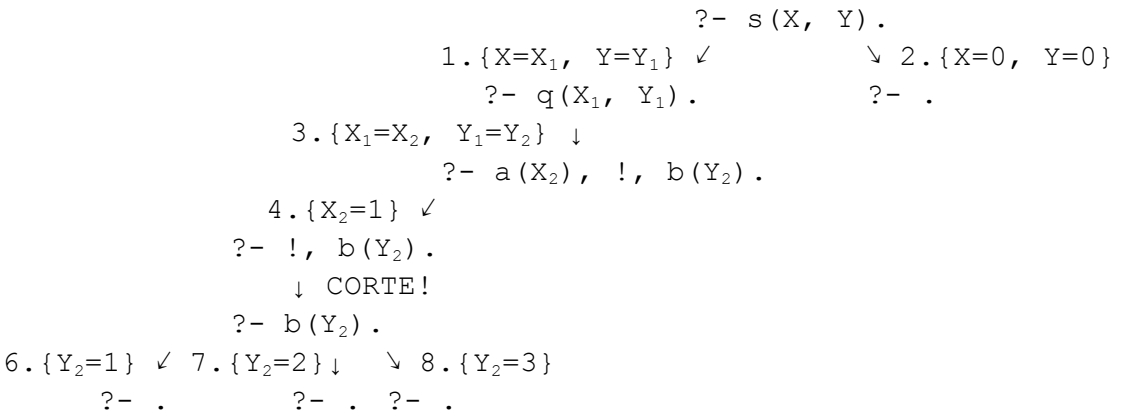
Agora vamos reescrever o programa incluindo um corte na cláusula que define q.

```
1. s(X, Y) :- q(X, Y).
2. s(0, 0).
3. q(X, Y) :- a(X), !, b(Y).
4. a(1).
5. a(2).
6. b(1).
7. b(2).
8. b(3).
```

Agora as respostas dada pelo Prolog são:

```
?- s(X, Y).
```
X=Y, Y=1;
X=1, Y=2;
X=1, Y=3;
X=Y, Y=0.

O que aconteceu? Vamos observar novamente a árvore de busca:



O corte ocorre no ramo esquerdo, portanto após unificar $a(X_2)$ com $a(1)$, são testadas as alternativas de unificação de $b(Y_2)$, mas não é permitido testar outra unificação para $a(X_2)$. Por isso os resultados que unificam X_2 com 2 não ocorrem. Depois disso, o *backtracking* retorna para $?- q(X_1, Y_1)$ e verifica que não há alternativas, daí retorna para $?- s(X, Y)$ e faz a unificação com $s(0, 0)$.

É importante lembrar que o corte só limita as opções a partir da unificação da cláusula que contém o corte. Por exemplo, considere uma cláusula:

$q :- p_1, \ldots, p_n, !, r_1, \ldots, r_m,$

quando o corte é alcançado, ele nos obriga a usar esta cláusula em particular para `q` e também nos obriga a usar as escolhas feitas para unificar os p_i. Estamos livres para fazer o *backtracking* entre as cláusulas r_i e também estamos livres para testar outras alternativas realizadas antes de alcançar a cláusula `q`.

Negação como Falha

Um aspecto interessante da programação lógica é a facilidade com que somos capazes de expressar regras genéricas. Se queremos dizer que um certo predicado `p(X)` é verdadeiro sempre que `q(X)` é verdade, basta escrevermos `p(X) :- q(X).`

No entanto, frequentemente as regras têm exceções, como podemos dizer que `p(X)` é verdade sempre que `q(X)` é verdade exceto quando `r(X)` é verdade? Você já deve ter se dado conta que no Prolog não tem o operador de negação lógica. Também não podemos afirmar fatos falsos, visto que toda cláusula de programa deve ter pelo menos um literal positivo; nem provar que um certo predicado é falso.

Do ponto de vista da programação lógica, tudo que podemos obter é uma falha em provar um predicado, o que não necessariamente quer dizer que o predicado seja falso. Se temos os fatos p(1) e p(2) e tentamos provar que existe um certo valor para o qual p é falso ($(\exists X)\ \neg p(X)$), só poderemos fazer isso se soubermos qual o domínio de X. No Prolog, apenas as constantes que aparecem no programa são testadas, o domínio não é definido em momento algum, então é impossível mostrar isso.

Lembre-se que quando Prolog responde `false`, ele está dizendo que não foi possível provar a consulta, ou seja, a cláusula vazia não foi atingida.

No Prolog há um predicado pré-definido chamado `fail` que, como o próprio nome sugere, sempre falha. Parece inútil, mas lembre que o Prolog faz o *backtracking* quando falha, então o `fail` pode ser visto como um modo forçado de fazer o *backtracking*. Usado em conjunto com o operador de corte, alguns efeitos interessantes podem ser obtidos.

Ex.: Considere o seguinte trecho de código:

```
p(X) :- r(X), !, fail.
p(X) :- q(X).
```

Como esse código funciona ao fazermos a consulta `?- p(c)`? Se `r(c)` não pode ser provado, a primeira regra não se aplica e a segunda regra é utilizada, de modo que `p(c)` é verdadeiro se `q(c)` é verdadeiro. Se `r(c)` é verdadeiro a primeira regra é aplicada e o corte é atingido. Isto nos proíbe de utilizar a segunda regra, e uma falha é forçada. Deste modo, `p(c)` é falso se `r(c)` é verdadeiro.

O comportamento obtido no exemplo anterior descreve a regra com exceções sugerida anteriormente, `p(X)` é verdade sempre que `q(X)` é verdade, desde que `r(X)` seja falso. Este exemplo depende bastante do aspecto procedural do Prolog, qualquer alteração na ordenação das cláusulas, ou a remoção do corte, mudam completamente o comportamento obtido.

Por isso, o conjunto corte-falha é encapsulado em um predicado pré-definido que define uma forma de negação chamada de negação como falha.

```
not(Objetivo) :- Objetivo, !, fail.
not(Objetivo).
```

O predicado `not(Objetivo)` é verdadeiro quando `Objetivo` falha. Para evitar confusões com a negação lógica, é preferível utilizar o predicado `\+` do que o predicado `not /1`. Podemos pensar no significado de `\+` como sendo “não satisfatível”, já que a barra invertida é normalmente utilizada como negação nos operadores do Prolog, como já vimos em `\=` e `\==`.

Ex.: Vamos refazer o código do exemplo anterior utilizando o predicado \+:

```
p(X) :- q(X), \+ r(X).
```

Temos o mesmo comportamento de antes, pois `\+ r(X)` só é verdade se não podemos provar `r(X)`.

Apesar da facilidade do operador `\+`, é importante notar que a ordem dos fatores ainda é importante, se escrevermos

```
p(X) :- \+ r(X), q(X).
```

o comportamento obtido é um pouco diferente, especialmente se a consulta é aberta, do tipo `?- p(X)`. Neste caso, se `r(X)` é definido para algum valor, `p(X)` é sempre falso.

Ex.: Considere o seguinte programa:

```
1. hamburguer(a).
2. hamburguer(b).
3. hamburguer(c).
4. burguer_queen(a).
5. mc_mickey(b).
6. bods(c).
7. gosto(X) :- hamburger(X), \+ bods(X).
```

Os três primeiros fatos dizem que `a`, `b` e `c` são hambúrgueres. Os três fatos seguintes especificam os tipos de cada hambúrguer. A linha 7 expressa o meu gosto, eu gosto de hambúrgueres desde que não seja do bods. Vamos fazer uma consulta para saber do que eu gosto:

```
?- gosto(X).
X = a;
X = b;
false
```

Então eu gosto de `a` e `b`, que são hambúrgueres, mas não gosto de `c`, porque é bods. Ao fazer a resolução, na obtenção da terceira resposta, `hamburguer(X)` é unificado com a terceira cláusula através da substituição `X = c`. Depois disso, `bods(c)` resolve com a sexta cláusula e, portanto, `\+ bods(c)` falha.

Se invertermos a ordem dos literais na sétima cláusula, `\+ bods(X)` sempre falha, porque sempre é possível resolver `bods(X)` com `bods(c)`.

De modo geral, podemos pensar em usar a negação como uma espécie de filtro para os resultados já obtidos através de outra resolução, ou seja, é fundamental que a variável já esteja instanciada quando utilizada na negação.

Aritmética

Como vimos em um exemplo anterior, o Prolog não trata os operadores aritméticos como matemática, ele simplesmente os reconhece como predicados. Mas seria um

desperdício não ser capaz de usar as capacidades aritméticas do computador, então o Prolog define um operador para utilizar esta facilidade.

O operador `is` tem dois argumentos, o primeiro argumento é um termo Prolog e o segundo é uma expressão aritmética. Ao invocar este operador a expressão aritmética é avaliada pelo computador e o resultado é unificado com o primeiro argumento. Se o primeiro argumento for uma variável, a variável é instanciada para o resultado da expressão, se o primeiro argumento é um número, funciona como um teste de igualdade.

Existem diversos operadores aritméticos pré-definidos em Prolog, vamos listar aqui apenas alguns deles, em todos eles E_1 e E_2 são expressões aritméticas.

Comparadores	
E_1 =:= E_2	Verdadeiro se o resultado de E_1 é igual ao de E_2.
E_1 =\= E_2	Verdadeiro se o resultado de E_1 é diferente do de E_2.
E_1 > E_2	Verdadeiro se o resultado de E_1 é maior que o de E_2.
E_1 < E_2	Verdadeiro se o resultado de E_1 é menor que o de E_2.
E_1 >= E_2	Verdadeiro se o resultado de E_1 é maior ou igual ao de E_2.
E_1 =< E_2	Verdadeiro se o resultado de E_1 é menor ou igual ao de E_2.
Funções	
E_1 + E_2	Soma E_1 a E_2.
E_1 - E_2	Subtrai E_2 de E_1.
E_1 * E_2	Multiplica E_1 e E_2.
E_1 / E_2	Divide E_1 por E_2.
E_1 // E_2	Divisão inteira de E_1 por E_2.
E_1 ** E_2	E_1 elevado a E_2.
E_1 ^ E_2	E_1 elevado a E_2.
mod(E_1, E_2)	Resto da divisão inteira de E_1 por E_2.
abs(E_1)	Valor absoluto de E_1.
sign(E_1)	1 se E_1 > 0, -1 se E_1 < 0, 0 se E_1 = 0.
max(E_1, E_2)	Máximo entre E_1 e E_2.
min(E_1, E_2)	Mínimo entre E_1 e E_2.
round(E_1)	Arredonda E_1 para o inteiro mais próximo.

`floor(E1)`	Arredonda `E1` para o inteiro anterior.
`ceil(E1)`	Arredonda `E1` para o próximo inteiro.
`sqrt(E1)`	Raiz quadrada de `E1`.
`sin(E1)`	Seno de `E1` (em radianos).
`cos(E1)`	Cosseno de `E1` (em radianos).
`tan(E1)`	Tangente de `E1` (em readianos).
`asin(E1)`	Ângulo (em radianos), cujo seno é `E1`.
`acos(E1)`	Ângulo (em radianos), cujo cosseno é `E1`.
`atan(E1)`	Ângulo (em radianos), cuja tangente é `E1`.
`log(E1)`	Logaritmo natural (base e) de `E1`.
`log10(E1)`	Logaritmo em base 10 de `E1`.
Constantes	
`e`	2.71828...
`pi`	3.14159...
`epsilon`	Menor ponto flutuante não nulo.
`inf`	Infinito positivo.
`nan`	Número indefinido (*Not a Number*).

As expressões aritméticas podem envolver variáveis do Prolog, mas é fundamental que as variáveis estejam instanciadas quando forem avaliadas. Não podemos definir novas funções aritméticas para serem utilizadas com o `is`, além das que são fornecidas pelo Prolog, mas podemos definir nossos predicados para realizar as funções que desejarmos.

Ex.: Vamos retomar o exemplo em que fizemos o Prolog contar os números naturais utilizando as operações aritméticas para obter os números e não a versão simbólica que tínhamos antes. Primeiramente vamos escrever a contagem de forma crescente.

1. `nat(0).`
2. `nat(Y) :- nat(X), Y is X+1.`

Ao consultar quem são os números naturais obtemos:

```
?- nat(X).
X = 0;
X = 1;
```

```
X = 2;
⋮
```

Ao fazer a consulta `?- nat(n)`, onde `n` é um número natural, ele responde `true`, como esperado. Se tentarmos continuar buscando novas respostas um laço infinito terá início, pois sempre teremos `nat(X)` como primeiro elemento da consulta e sempre é possível unificar com a segunda regra, de modo que a consulta só cresce. Não temos como inserir um ponto de corte porque como a contagem é crescente, não temos caso base.
Ao contrário do caso simbólico, é possível definir os naturais usando subtração, da seguinte forma:

```
1. nat(0).
2. nat(Y) :- X is Y-1, nat(X).
```

Não é possível listar os números naturais, como na definição anterior, porque ao tentar usar a segunda regra Y não está definido e não é possível subtrair 1.

```
?- nat(X).
X = 0;
ERROR: Arguments are not sufficiently instantiated
```

A contagem nesta segunda definição é regressiva, para verificar se 5 é natural, é verificado se 4 é natural, para verificar isso vemos se 3 é natural, e assim sucessivamente até encontrar o 0 e responder `true`. Se tentarmos obter novas respostas teremos um laço infinito novamente, porque estamos avaliando os números negativos e nunca mais chegaremos no caso base (0). Mas, agora podemos inserir um ponto de corte para não dar a possibilidade de buscar novas soluções.

```
1. nat(0) :- !.
2. nat(Y) :- X is Y-1, nat(X).
```

Essa definição é uma forma mais segura de verificar se um certo número é natural ou não, mas ainda não está completo. Se o número não for natural o caso base nunca vai ocorrer e teremos um laço infinito, pelo mesmo motivo de antes, então vamos inserir um teste para verificar se o número é maior ou igual a zero antes de fazer a chamada recursiva.

```
1. nat(0) :- !.
2. nat(Y) :- X is Y-1, X >= 0, nat(X).
```

Agora a definição está completa e temos uma função que nos permite verificar se um certo número é natural ou não. Ao contrário do exemplo simbólico, não podemos colocar a unificação do lado esquerdo da definição como `nat(is X+1)` ou `nat(is Y-1)`, o predicado `is` precisa de dois argumentos e, se tiver dentro do argumento de outro predicado não será avaliado como um predicado.

Ex.: Vamos definir agora uma função para calcular o fatorial de um número. Por definição, o fatorial de 0 é 1 e o fatorial de um natural *n* positivo qualquer é o produto de todos os números naturais de 1 até *n*. Recursivamente, podemos dizer que o fatorial de *n* é o fatorial

de *n* - 1 multiplicado por *n*. Aritmeticamente, temos *f*(*n*) = *n* × *f*(*n* - 1). No Prolog não temos funções, apenas relações, então vamos definir uma relação `f(X, Y)` que significa "`Y` é o fatorial de `X`". Além disso, vamos precisar utilizar as operações aritméticas duas vezes, uma para subtrair 1 de de *n*, e outra para multiplicar o fatorial de *n* - 1 por *n*.

```
1. f(0, 1) :- !.
2. f(X, Y) :- X1 is X-1, f(X1, Z), Y is Z*X.
```

O primeiro fato diz que o fatorial de `0` é `1`, e é o nosso caso base. O segundo diz que `Y` é o fatorial de `X` se `X1` é `X-1`, `Z` é o fatorial de `X1` e `Y` é `Z*X`.
Ao fazer a consulta `f(n, X)`, onde `n` é um número natural, o fatorial de `n` será unificado com `X`. Neste caso estamos confiando que o usuário sempre vai usar o predicado f com um número natural, caso contrário teremos um laço infinito.
Vamos agora fazer uma definição alternativa do fatorial usando uma variável auxiliar como acumulador:

```
1. fat(0, A, A) :- !.
2. fat(X, A, Y) :- X1 is X-1, A1 is X*A, fat(X1, A1, Y).
```

A variável Y só é unificada com A no caso base, antes disso A é multiplicado por todos os valores de naturais positivos até X. Como A é um acumulador de um produto, para utilizarmos esta relação em uma consulta devemos iniciar A sempre com 1, que é o elemento neutro do produto.
Em ambos os casos os resultados obtidos são idênticos:

```
?- f(20, X).
X = 2432902008176640000

?- fat(20, 1, X).
X = 2432902008176640000
```

A diferença está na forma como os resultados são obtidos, na segunda definição a recursão está no fim e no primeiro caso a recursão está no meio. Vamos fazer uma consulta para um número menor para ver a árvore de busca.

```
?- f(3, X).
↓ 2. {X1 = 3, Y1 = X}
?- X11 is 3-1, f(X11, Z1), X is Z1*3.
↓ {X11 = 2}
?- f(2, Z1), X is Z1*3.
↓ 2. {X2 = 2, Y2 = Z1}
?- X12 is 2-1, f(X12, Z2), Z1 is Z2*2, X is Z1*3.
↓ {X12 = 1}
?- f(1, Z2), Z1 is Z2*2, X is Z1*3.
↓ 2. {X3 = 1, Y3 = Z2}
?- X13 is 1-1, f(X13, Z3), Z2 is Z3*1, Z1 is Z2*2, X is Z1*3.
↓ {X13 = 0}
?- f(0, Z3), Z2 is Z3*1, Z1 is Z2*2, X is Z1*3.
```

```
↓ 1. {Z3 = 1}
?- Z2 is 1*1, Z1 is Z2*2, X is Z1*3.
↓ {Z2 = 1}
?- Z1 is 1*2, X is Z1*3.
↓ {Z1 = 2}
?- X is 2*3.
↓ {X = 6}
?- .

?- fat(3, 1, X).
↓ 2. {X1 = 3, A1 = 1, Y1 = X}
?- X11 is 3-1, A11 is 3*1, fat(X11, A11, X).
↓ {X11 = 2}
?- A11 is 3*1, fat(2, A11, X).
↓ {A11 = 3}
?- fat(2, 3, X).
↓ 2. {X2 = 2, A2 = 3, Y2 = X}
?- X12 is 2-1, A12 is 2*3, fat(X12, A12, X).
↓ {X12 = 1}
?- A12 is 2*3, fat(1, A12, X).
↓ {A12 = 6}
?- fat(1, 6, X).
↓ 2. {X3 = 1, A3 = 6, Y3 = X}
?- X13 is 1-1, A13 is 1*6, fat(X13, A13, X).
↓ {X13 = 0}
?- A13 is 1*6, fat(0, A13, X).
↓ {A13 = 6}
?- fat(0, 6, X).
↓ 1. {X = 6}
?- .
```

O número de inferências realizadas nos dois casos são idênticos, no entanto, a recursão à direita, utilizada na segunda definição, permite que o compilador libere a memória a cada nova recursão e não precisa manter diversos contextos na memória. Isto torna o procedimento mais eficiente. No meu computador, particularmente, calcular o fatorial de 25000 com o primeiro método leva cerca de 3 segundos enquanto o segundo método leva 0.2 segundos. Utilizando o servidor do SWISH a diferença é menor, mas ainda assim o primeiro método leva cerca do dobro do tempo. Esses casos são meramente ilustrativos, a diferença entre os métodos depende obviamente da máquina utilizada.
Como o segundo método precisa ser utilizado sempre com o acumulador iniciado em 1, é mais interessante criar uma relação auxiliar para encapsular ele (*wrapper*).

```
1. fatorial(X, Y) :- fat(X, 1, Y).
2. fat(0, A, A) :- !.
3. fat(X, A, Y) :- X1 is X-1, A1 is X*A, fat(X1, A1, Y).
```

Agora podemos calcular o fatorial utilizando apenas dois argumentos, como era na primeira definição, mas utilizando o método mais eficiente, que depende do acumulador. Nossa consulta será apenas `fatorial(n, Y)` para ter o fatorial de `n` na variável `Y`.

Listas

Uma lista em Prolog é apenas uma coleção finita de elementos, definidos entre colchetes e separados por vírgulas. Qualquer termo Prolog pode ser elemento de uma lista, inclusive outra lista.

No Prolog existe a lista vazia, `[]`, e todas as demais listas podem ser definidas como [H | T], onde os H é o primeiro elemento da lista, normalmente chamado de cabeça (*head*), e T é uma lista que contém os demais elementos, normalmente chamado de cauda (*tail*).

Essa definição recursiva das listas é utilizada na unificação e é fundamental para acessarmos os elementos das listas. Vamos começar verificando alguns exemplos de como se dá a unificação de listas.

Ex.: `[H | T] = [a, b, c].`
`H = a,`
`T = [b, c].`

Ex.: `[H | T] = [a].`
`H = a,`
`T = [].`

Ex.: `[H1, H2, H3 | T] = [a, b, c, d, e].`
`H1 = a,`
`H2 = b,`
`H3 = c,`
`T = [d, e].`
Este exemplo nos mostra que é possível unificar mais de um elemento simultaneamente.

Ex.: `[H1, H2, H3 | T] = [a, b].`
`false.`
Não podemos fazer a unificação porque tentamos obter três elementos mas a lista contém apenas dois elementos.

Ex.: `[H | T] = [].`
`false.`
A lista vazia é especial, e não possui cabeça ou cauda.

As listas são especialmente úteis para organizar informações. Existem diversos predicados pré-definidos para manipular listas em bibliotecas Prolog, que são nativas em algumas implementações, mas vamos criar nossa própria biblioteca de predicados nos próximos exemplos.

Ex.: Um primeiro comportamento útil é acessar os elementos da lista, um a um. Considere o seguinte código:

```
1. listar(X, Lista) :- Lista = [H | T], X = H.
2. listar(X, Lista) :- Lista = [H | T], listar(X, T).
```

Esse predicado tem o objetivo de fazer a consulta listar(X, lista) e obter como respostas os elementos de lista.

```
?- listar(X, [a, b, c]).
X = a;
X = b;
X = c;
false.
```

Como esse código funciona? A primeira cláusula dá o primeiro elemento da lista, a lista é dividida em cabeça e cauda e X é unificado com a cabeça. A segunda cláusula divide a lista em cabeça e cauda e pede para listar os elementos da cauda, removendo o primeiro elemento. Todas as unificações poderiam ser feitas diretamente, além disso o valor de T não é necessário na primeira cláusula e o de H não é necessário na segunda. O seguinte código é equivalente ao anterior:

```
1. listar(X, [X|_]).
2. listar(X, [_|T]) :- listar(X, T).
```

Vamos verificar a árvore de busca para a consulta anterior:

```
?- listar(X, [a, b, c]).
1. {X = a} ↓     ↘ 2. {X₁ = X, T₁ = [b, c]}
         ?- .     ?- listar(X, [b, c]).
           1. {X = b} ↓     ↘ 2. {X₂ = X, T₂ = [c]}
                    ?- .     ?- listar(X, [c]).
                      1. {X = c} ↓     ↘ 2. {X₁ = X, T₁ = []}
                               ?- .     ?. listar(X, []).
                                                 aberto
```

Como a cláusula vazia não pode ser unificada com a estrutura `[H | T]`, que denota uma lista não-vazia, não é necessário usar cortes.

Ex.: Outro comportamento interessante é verificar se um dado elemento pertence ou não a uma lista. O código do exemplo anterior também se presta a esse objetivo se fizermos uma consulta definida.

```
?- lista(a, [a, b, a, c, a]).
true;
true;
true;
false.
```

Para cada ocorrência do elemento na lista obtemos um resultado `true`. Em diversas implementações do Prolog o predicado `member` é definido exatamente como definimos `listar`.
Às vezes queremos saber apenas se um certo elemento ocorre ou não na lista, não estamos interessados em outros resultados. Para obter este comportamento basta acrescentar um corte ao identificar o primeiro resultado:

```
1. ocorre(X, [X|_]) :- !.
2. ocorre(X, [_|T]) :- listar(X, T).
```

Agora `ocorre(X, lista)` retorna `true` se `X` ocorre em `lista` e `false` caso não ocorra. O predicado `memberchk` é definido no Prolog como definimos `ocorre`.

Ex.: Outra utilidade interessante, relacionada à ocorrência de elementos na lista, é contar quantas vezes um certo valor ocorre na lista. Neste caso vamos precisar alterar um pouco o nosso código para incluir uma variável que vai conter o total de ocorrências:

```
1. conta(X, [], 0).
2. conta(X, [H|T], Y) :- X = H, conta(X, T, Z), Y is Z+1.
3. conta(X, [H|T], Y) :- X \= H, conta(X, T, Y).
```

A primeira cláusula diz que o número de ocorrências de `X` numa lista vazia é 0, esse é o nosso caso base porque, como nos casos anteriores, vamos remover um a um os elementos da lista. As duas cláusulas que seguem fazem a contagem de fato, Se `X` é o primeiro elemento da lista somamos um ao total, em ambos os casos é feita uma chamada recursiva da contagem sobre o resto da lista. Aproveitando as ferramentas que vimos anteriormente, podemos reescrever o código como:

```
1. conta(_, [], 0) :- !.
2. conta(X, [X|T], Y) :- !, conta(X, T, Z), Z is Y+1.
3. conta(X, [_|T], Y) :- conta(X, T, Y).
```